NASA/CR—2007-214939

Filling the Assurance Gap on Complex Electronics

Richard A. Plastow
Science Applications International Corporation, Cleveland, Ohio

August 2007

NASA/CR—2007-214939

Filling the Assurance Gap on Complex Electronics

Richard A. Plastow
Science Applications International Corporation, Cleveland, Ohio

Prepared for the
Second International Conference on Space Safety
cosponsored by the ESA, NASA, and IAASS
Chicago, Illinois, March 14–16, 2007

Prepared under Contract NAS3–03140

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

August 2007

Acknowledgments

Thanks to the NASA Software Assurance Research Program which funded this research. Special thanks to Kalynnda Berens who developed the idea and initially assisted in the research. Finally, thanks to Phuoc Thai who assisted with the surveys and Anita Tenteris who proofread this paper.

Filling the Assurance Gap on Complex Electronics

Richard A. Plastow
Science Applications International Corporation
Brook Park, Ohio 44142

Abstract

Many of the methods used to develop software bare a close resemblance to Complex Electronics (CE) development. CE are now programmed to perform tasks that were previously handled by software, such as communication protocols. For example, the James Webb Space Telescope will use Field Programmable Gate Arrays (FPGAs), which can have over a million logic gates, to send telemetry. System-on-chip (SoC) devices, another type of complex electronics, can combine a microprocessor, input and output channels, and sometimes an FPGA for programmability. With this increased intricacy, the possibility of "software-like" bugs such as incorrect design, logic, and unexpected interactions within the logic is great.

Since CE devices are obscuring the hardware/software boundary, mature software methodologies have been proposed, with slight modifications, to develop these devices. By using standardized S/W Engineering methods such as checklists, missing requirements and "bugs" can be detected earlier in the development cycle, thus creating a development process for CE that can be easily maintained and configurable based on the device used.

Overview

As software and hardware merge into self adapting designs, how is correctness of their function assured? Complex Electronics is defined as neither hardware nor software but as "soft hardware". What resides in them is not really software but a logic structure for a hardware device that was created by software. Quality Assurance is struggling with how to adequately deal with these "software-like" aspects. Currently, software methods are being used to develop this "Soft hardware". Some problems and concerns with CE use are:

- Application Specific Integrated Circuits (ASICs) and FPGAs are used to avoid the rigors of the software assurance process which ultimately results in bypassing fundamental verifications.
- Complex Electronic devices are designed and programmed by electronic engineers (designers), often without quality assurance oversight or configuration management control of the designs. In addition, the development process may not be well defined or followed.
- ASICs, FPGAs, and SoC can contain embedded microprocessor cores with user-supplied software. They combine electronics and firmware onto one chip. The presence of the firmware (i.e., software) is not always obvious to assurance personnel.
- Hardware Designers now utilize high-level software languages (e.g., C, C++) to define complex electronic designs (in whole or in part).
- Hardware quality assurance personnel may not be fully cognizant of the functions, potential problems, and issues with these devices.
- Meaningful verification efforts require the person be knowledgeable about the complex electronic device and the tool suite used to create and implement the design.

The fast pace of technological change and the new uses of current technology are strong motivators to define acceptable assurance practices for complex electronics. Adaptive or reconfigurable computing, where computers, chips, or systems alter their functionality to adapt to changing applications, could be an assurance nightmare.

Software techniques, processes and checklists can help fill this assurance gap. Modern HDLs are derived from software languages with VHDL a derivative of ADA and Verilog based on C. This research proposes the uses of these software techniques in a hardware development setting. A large portion of the development process is software driven. The chip design is created in a Hardware Description Language (HDL). This design is then simulated, tested, and verified as software. A netlist is then created from this HDL and burned into the chip, creating the real hardware.

Complex Electronics

Is the Device Complex?

In the term complex electronics, complex is used to distinguish between simple devices, such as off-the-shelf Integrated Circuits (ICs) and logic gates, and complex user-creatable devices. A good rule of thumb to distinguish between simple and complex is, if the internal logic of the device has more than a few gates and connections, it is probably complex.

The Federal Aviation Administration (FAA) provides a definition in DO-254, "Design Assurance Guidance for Airborne Electronic Hardware" document: "*A hardware item is identified as simple only if a comprehensive combination of deterministic tests and analysis appropriate to the design assurance level can ensure correct functional performance under all foreseeable operating conditions with no anomalous behavior. When an item cannot be classified as simple, it* *should be classified as complex. An item constructed entirely from simple items may itself be complex."*

Firmware (which is essentially software stored on a read-only device) is not considered complex electronics. The most common definition of firmware is found in IEEE 610.12-1990: "The combination of hardware device and computer instructions and data that reside as read-only software on that device."

Types of Complex Electronics

There are two types of Complex Electronics, programmable or designable integrated circuits. "Programmable" logic devices are programmed by the designer and range from simple chips to complex devices capable of being programmed on-the-fly. Some types of programmable devices are:

- Complex Programmable Logic Device (CPLD)
- Field Programmable Gate Arrays
- FPGA microprocessors/systems
- Reconfigurable computing

"Designable" logic devices are integrated circuits that are designed but not burned into the chip by the design engineer. The design is submitted to the manufacturer for implementation in the device. The ASIC and SoC are examples of a designable device.

With complex electronics, there is no defined standard for developing the software that is used to create the hardware. The design engineer will design to some electronic standard and local best practices. This can create a wide difference between the verifications done when the hardware is created, even in the same organization.

Using Software Processes on Hardware

Hardware development follows a process that is very similar to software development. Many of the proven software techniques used in software development/verification can be used with little or no modification on hardware. In the next sections, each phase of the development cycle will be discussed and comparisons will be made between hardware and software.

Classify the Device

As with any software project, classifying the criticality of the function(s) the chip will be performing is necessary. Two important questions need to be asked. Does the chip perform safety or mission critical functions? What are the consequences if it fails? Table 1 lists the criteria for each CE criticality. The quality assurance engineer should determine if the device in question falls within the High Criticality, based on the criteria listed. If the device does not meet the high criteria, the software quality assurance engineer should review the Moderate category. If the device does not meet the moderate criteria then the device should be classified as Low criticality.

TABLE 1.—CE CRITICALITY CLASSIFICATION

Criticality	Criteria
High	• The complex electronics executes safety-critical functions • The complex electronics executes mission-critical functions and is a single point of failure • The design is expected to be highly complex • The design is expected to have significant risk due to one or more of these factors: o Unstable requirements o Technical concerns with the chosen technology, such as power consumption, design size for the chip, timing, packaging, or operating frequency o Highly innovative and untried design approach o Highly aggressive schedule o Inexperience of the design team
Moderate	• The complex electronics executes mission-critical functions but there is redundancy in the system • The design is expected to be moderately complex • The design is expected to have moderate risk due to one or more of these factors: o Some requirements undefined or unstable o Somewhat innovative and untried design approach o Aggressive schedule o Design team contains some inexperienced members
Low	The complex electronics is classified as *Low* if it does not fall into either of the above categories

Plan the Process

Once the criticality of the device is determined, the assurance process can be tailored based on that criticality. An assurance plan for the device listing pertinent information should be created. It should include the same type of information needed when software is being developed. This would include:

- Development system (tools) being used
- Configuration management system
- Training needed
- Problem tracking system
- Risk Management
- Device Criticality level

- Audits to be performed (Based on criticality)
- Reference and applicable documents

The plan should be reviewed by the complex electronics developer, the system engineer, quality assurance, and project management. Once approved, the Complex Electronics Assurance Plan will guide the assurance activities for the rest of the project life cycle. Like all project plans, the Complex Electronics Assurance Plan is a living document and should be reviewed and updated whenever any significant changes occur in the project.

Requirements

Like software, the requirements for complex electronics flow down from system and sub-system requirements; therefore, a requirements review should be done. Care must be taken to ensure the requirements meet the 3C's: *clear, concise, and confirmable*. Table 2 shows the software techniques that can be utilized to verify the complex electronics requirements. Each activity is based on the determined device criticality.

TABLE 2.—SOFTWARE TECHNIQUES FOR REQUIREMENTS

Techniques for Requirements Phase based on criticality level			
	Low	**Moderate**	**High**
Requirements evaluation	Check for main characteristics	All main characteristics plus assess for completeness	Complete evaluation against all criteria
Requirements review	Informal	Informal	Formal
Interface analysis	Informal Assessment	Focus on key interfaces and critical timing	Detailed analysis, all interfaces. Includes timing
Traceability analysis	Trace from higher-level requirements	Trace from higher-level requirements. Verify derived requirements correctly trace	Trace from higher to CE and from CE to higher level requirements Full assessment of derived requirements
Reverse Engineering of Requirements	Performed only if no requirements document available from project	Performed only if no requirements document available from project	Independent derivation of requirements used as a tool to assess requirements document.
Modeling	Not performed	Some diagrams are created to assess requirements	Complete modeling of the requirements

Table 3 lists the other Assurance Activities/Techniques that can be performed through out the development cycles based on the criticality of the complex electronic device. As an example, Fault Tree Analysis (FTA) allows the designer to note those critical areas and use additional care in the design of that area. The FTA aids in analysis of large and complex systems, and is particularly adept at representing and analyzing redundancy arrangements. In addition, common cause events are easily handled.

TABLE 3.—ADDITIONAL ASSURANCE ACTIVITIES

Continuing Assurance Activities based on criticality level			
	Low	**Moderate**	**High**
Risk analysis	Informal	Informal	Formal
Decision tables or trees	Not performed	Performed with most important conditions only	All inputs and conditions
Fault Tree Analysis (FTA)	Minimal	Minimal	Add design failures for CE to FTA, if not already included
Failure Modes and Effects Analysis (FMEA)	Minimal	Ensure that FMEA addresses CE failures	Add design failures for CE to FMEA, if not already included

Assurance techniques used for complex electronics are based on the needs of the project and should be updated as the device is developed and tested or changed.

Design Entry

The objective of assurance during design entry is to verify that the design correctly implements all the requirements designated.

The main design activities for complex electronics during the preliminary design phase are:

1. Create a high-level design description.
2. Identify major components, including third-party Intellectual Property (IP) modules or cores.
3. Feedback any derived requirements that result from the process to the requirements engineering process.
4. Resolve any omissions or errors in the requirements found during design entry.
5. Include reliability, such as Single Event Upset (SEU) protection, maintenance, and test features that are necessary to meet performance and quality requirements.
6. Ensure that testing performed will verify the requirements.
7. Identify constraints on other system elements as a result of this high-level design.

The assurance engineer should ask the following questions when evaluating the preliminary design:

- Is the CE design complete?
- Do all requirements trace to architectural blocks or functions?
- Are the requirements correctly implemented?
- Does the design contain functions that are not required?

- Does the design contain any internal inconsistencies or conflicts?
- Does the design conflict with any requirements or with another system element?
- Does the design clearly identify constraints on other system elements, and on the installation and operation processes?
- Is the quality of the design (documentation and Hardware Description Language (HDL) model) adequate for its usage?
- Will the design result in a testable system? Consider both physical access (e.g., test pins) and the ability to test modules within the device.
- Are special pins (e.g., mode pin on FPGA, Joint Test Action Group (JTAG) pins, no-connect pins) used correctly?
- If some requirements conflict (such as power consumption and performance speed), was a trade-off performed to determine the optimal levels for each requirement?
- Is the design flexible enough to accommodate anticipated requirements changes?
- Did the design follow the coding standard and design guidelines?

If Commercial Off The Shelf (COTS) or IP cores/modules are used within the complex electronics, their design needs to A be evaluated also. Areas to consider are:

- Functions within the IP module that will not be used.
- Interfaces (voltages, timing, signal characteristics, etc.)
- Verification performed by the vendor. What documentation exists for that verification? What additional testing needs to be performed?
- IP module upgrades. If the vendor indicates that there is a problem with the IP module, how will that problem (and fix) be evaluated for the complex electronics?

At this point in the project life cycle, the design is just code modules developed in the selected HDL. Although HDL is not “real code”, many software assurance techniques can be utilized effectively. For example, as a risk mitigation step, using coding standards make the code more readable and upgradeable. Using a standard naming convention allows another design engineers to continue/modify the design with greater assurance of success. Table 4 lists useful software techniques for the design stage of the development cycle based on device criticality. Preliminary design reviews provide alternate viewpoints of the current design and provide valuable feedback to the design engineer. Interface Analysis can be used to insure that all interfaces are correct and included in the design.

One example of a design assurance activity that is highly successful for software assurance is Change Impact Analysis (CIA). CIA should be performed for complex electronics to measure the impact of any planned HDL code change. This includes changes to fix problems encountered as well as enhancements.

TABLE 4.—DESIGN PHASE ASSURANCE ACTIVITIES

Assurance Techniques for the Design Phase			
	Low	**Moderate**	**High**
Simulation	Not performed	Not performed, or only performed on aspects that are in question	If model is executable, simulations should be designed for nominal and error conditions
Preliminary design evaluation	Evaluate design for compatibility with rest of system	Evaluate for compatibility, interfaces. Have expert evaluate important design areas.	Use expert to evaluate design
Preliminary design review (formal or informal)	Informal	Informal or Formal	Formal, with engineering expert
Interface analysis	Informal assessment	Focus on key interfaces and critical timing	Detailed analysis, all interfaces. Includes timing
Traceability analysis	Trace from requirements to design	Trace from complex electronics requirements to design blocks. Verify blocks correctly implement the requirements.	Trace from complex electronics requirements to design blocks and the reverse. Verify no functionality not covered by requirements

Design Testing

Testing is an activity that occurs during multiple phases of the development cycle. The initial design phase sees various components of the design developed and tested as individual pieces. Here it is important that the designer get the interfaces correct. Design reviews held using the Interface Control Document can prevent major problems later as the components are linked together in the detailed design phase. During the detailed design of the chip a test bench, a collection of simulated inputs and timing models, is used to test and verify the design. Test benches, often created by the designer, are not infallible. A review of the test plan, aka test bench, will help insure all requirements are tested. Simulation is used extensively during detailed design to verify the complex electronic design. Testing the actual chip during system verification can sometimes be an eye-opening experience.

Testing the programmed chip—independently and integrated onto a circuit board—are necessary steps in verifying the design. The assurance role during the testing phase is to ensure that the tests are adequate to verify the requirements. Beyond that, good assurance engineers help the project define tests that exercise the system, or system element, in various situations, including fault and failure conditions. Functional and Physical Configuration Audits should be performed on the device. Figure 1 shows the flow through the various tests performed.

Creation of a test plan is an important part of chip verification and should not be omitted. This plan should be used to insure all chip requirements have been tested. The plan should include more than “test for success” items. Test cases should include expected faults, all known error conditions, and missing inputs. During the design phase testing, faults should be injected into the simulations to insure they are handled gracefully.

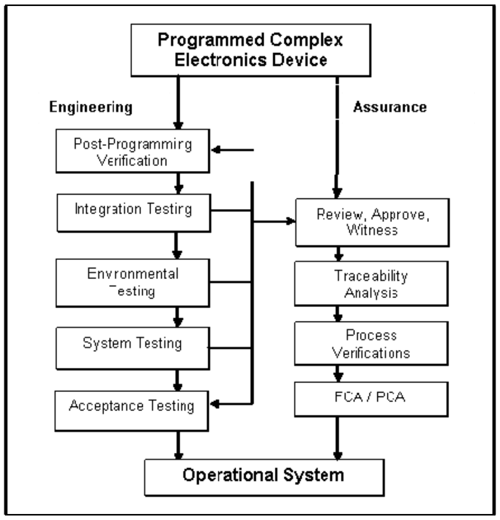

Figure 1.—CE testing.

Conclusions

Complex electronics development follows a design cycle very similar to software. Modern HDLs are derived from a software language with VHDL a derivative of ADA and Verilog based on C. With such a close tie to software, it is a logical step to use software methods. The software development community offers many tools that can streamline the development of complex electronics while enhancing the quality of the device.

Software techniques such as change impact analysis and fault tree analysis provide a way to better understand and verify the CE design. Requirements reviews can assure that requirements are complete, concise, and confirmable. Design reviews using software like coding standards will verify the HDL is correct, understandable and modifiable. Software-like methods can be used within all design phases to develop the final product. As software and hardware merge into self adapting designs, this interdisciplinary approach provides a way to insure design quality in the future.

Future Work

This work focused on the use of software quality assurance techniques and checklists with FPGAs. Future research will focus on verifying these techniques and checklists on other types of complex electronic devices such as SoCs and ASICs. In addition, more research is needed for reconfigurable computing to determine the most appropriate approach to device verification.

References

1. Federal Aviation Administration (FAA) Software Considerations in Airborne Systems and Equipment Certification document DO-178B
2. FAA Design Assurance Guidance for Airborne Electronic Hardware document DO-254
3. NASA Software Assurance Standard STD-8739.8
4. NASA Software Engineering Requirements NPR 7150.2
5. NASA Software Safety Standard NASA-STD-8719.13B

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01-08-2007	**2. REPORT TYPE** Final Contractor Report	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Filling the Assurance Gap on Complex Electronics	**5a. CONTRACT NUMBER** NAS3-03140
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Plastow, Richard, A.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER WBS 981155.03.03.01
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Science Applications International Corporation 21000 Brookpark Road Brook Park, Ohio 44142	**8. PERFORMING ORGANIZATION REPORT NUMBER** E-16173
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSORING/MONITORS ACRONYM(S)** NASA
	11. SPONSORING/MONITORING REPORT NUMBER NASA/CR-2007-214939

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Categories: 33, 38, and 61
Available electronically at http://gltrs.grc.nasa.gov
This publication is available from the NASA Center for AeroSpace Information, 301-621-0390

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Many of the methods used to develop software bare a close resemblance to Complex Electronics (CE) development. CE are now programmed to perform tasks that were previously handled by software, such as communication protocols. For example, the James Webb Space Telescope will use Field Programmable Gate Arrays (FPGAs), which can have over a million logic gates, to send telemetry. System-on-chip (SoC) devices, another type of complex electronics, can combine a microprocessor, input and output channels, and sometimes an FPGA for programmability. With this increased intricacy, the possibility of "software-like" bugs such as incorrect design, logic, and unexpected interactions within the logic is great. Since CE devices are obscuring the hardware/software boundary, mature software methodologies have been proposed, with slight modifications, to develop these devices. By using standardized S/W Engineering methods such as checklists, missing requirements and "bugs" can be detected earlier in the development cycle, thus creating a development process for CE that can be easily maintained and configurable based on the device used.

15. SUBJECT TERMS
Complex electronics; Firmware; Software; Assurance; Safety; Programmable; ASIC; FPGA; SoC; Process

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** STI Help Desk (email:help@sti.nasa.gov)
a. REPORT U	**b. ABSTRACT** U	**c. THIS PAGE** U	UU	11	**19b. TELEPHONE NUMBER** *(include area code)* 301-621-0390