A CASE-STUDY APPLICATION OF RTCA DO-254: DESIGN ASSURANCE GUIDANCE FOR AIRBORNE ELECTRONIC HARDWARE

Paul S. Miner, Victor A. Carreño, Mahyar Malekpour, and Wilfredo Torres

NASA Langley Research Center, Hampton, VA

{p.s.miner, v.a.carreno, m.r.malekpour, w.torres-pomales}@larc.nasa.gov

A CASE-STUDY APPLICATION OF RTCA DO-254: DESIGN ASSURANCE GUIDANCE FOR AIRBORNE ELECTRONIC HARDWARE

Paul S. Miner, Victor A. Carreño, Mahyar Malekpour, and Wilfredo Torres

NASA Langley Research Center, Hampton, VA

{p.s.miner, v.a.carreno, m.r.malekpour, w.torres-pomales}@larc.nasa.gov

Abstract

In a joint project with the FAA, NASA Langley is developing a hardware design in accordance with RTCA DO-254: *Design Assurance Guidance for Airborne Electronic Hardware*. The purpose of the case study is to gain understanding of the new guidance document and generate an example suitable for use in training.

For the case study, we have selected a core subsystem of the Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER). SPIDER is a new fault-tolerant architecture under development at NASA Langley Research Center.

Introduction

RTCA, Inc. has recently approved DO-254: *Design Assurance Guidance for Airborne Electronic Hardware* [1]. This document is intended to provide a basis for the certification of complex electronic hardware devices used in aircraft. As with all new guidance documents, there will be a period of uncertainty while developers and the FAA learn how best to apply the document. This uncertainty will likely be compounded by the flexibility DO-254 offers for developing a design assurance strategy.

To alleviate some of this uncertainty the FAA initiated this case study to gain some experience with this new guidance document. This case study has limited scope; there are insufficient resources to explore all aspects of DO-254. The focus of the case study is on logical aspects of design, and is targeted to early stages of the design process.

There are two goals for the case study. The primary purpose is to help identify problems in applying the new document. A secondary purpose is to provide material suitable for use in training. The second objective places some constraints on the project scope. Specifically, the hardware device should be non-proprietary, non-commercial, and of limited complexity.

NASA Langley Research Center also has some objectives for this case study. One goal is to demonstrate the application of formal methods on a representative example. Another priority for NASA is to develop a hardware platform to support in-house research targeted toward demonstrating systematic recovery from multiple correlated transient failures.

With these objectives and restrictions in mind, the chosen device for the case study is a core subsystem of a new fault-tolerant architecture under development at NASA Langley Research Center. Several factors motivated the choice of a fault-tolerant system for this exercise. Hardware realizations of fault-tolerant protocols are generally compact designs; this allows for comprehensive treatment within the time constraints of a training exercise. Also, the behavior of fault-tolerant devices is inherently complex; such a device is clearly within the scope of DO-254. Furthermore, there is a considerable amount of research literature addressing the formal analysis of fault-tolerant protocols; a fault-tolerant system is a good candidate for a formal methods demonstration. Finally, any device expected to recover from transient failures will necessarily need to deal with a bounded set of permanent failures, as well.

The architecture being explored for this case study is the Scalable Processor-Independent Design for Electromagnetic Resilience (SPIDER). In this architecture, the primary basis for fault-tolerance is a communication subsystem called the Reliable Optical Bus (ROBUS). The concept for the SPIDER architecture builds on previous fault-

tolerant computing research at NASA Langley Research Center. The main concept was inspired by a fault-tolerant system designed as part of the Fly-by-Light/Power-by-Wire (FBL/PBW) program [2] [3].

Project Overview

The project consists of three sequential phases beginning in August 1999 and ending in December 2001. The first phase, which began in August 1999 and ended in January 2000, consisted of planning the hardware development activities. The initial *Plan for Hardware Aspects of Certification* was presented to the FAA in January and was subsequently revised. The second phase consists of the development of a detailed design in accordance with the submitted plan. The detailed design will include a limited laboratory prototype intended to illustrate certain characteristics of the architectural concept. The final phase will culminate in a more complete prototype implementation; this prototype will include sufficient features to make a fair assessment of the proposed design.

Planning Phase

Any organization applying DO-254 for the first time must decide how to map its existing development processes to those of DO-254. Since our research group did not have any defined hardware development processes, we were free to define all aspects of our hardware development environment. This served as both a blessing and a curse. We were able to select from a variety of options. However, some of the choices we made were necessarily uninformed. It is likely that we shall need to update our processes as the project progresses.

During the planning phase of the project, we identified the target design artifacts and determined the various components of our development environment. In addition to identifying the design and verification methods, we needed to define the supporting processes that serve to control the development and maintenance of the developed product. These supporting processes include:

- Certification Liaison
- Process Assurance
- Configuration Management

The certification liaison activities consist of regular interaction with a small team from the FAA. The initial *Plan for Hardware Aspects of Certification* was presented to the FAA in January. They agreed with the proposed plan, in principle, but deferred judgement on some of the proposed verification activities until more detailed descriptions are developed.

The process assurance activities are primarily focused on monitoring the development activities to ensure that they are proceeding in accordance with the approved plans. For this project, the emphasis is on ensuring the internal consistency of the set of controlled data we accumulate throughout the development effort.

An integral part of developing hardware in accordance with DO-254 is the management of design and verification data throughout the life of the product. Hence, there is a need for an effective recording and configuration management system.

The purpose of configuration management is to ensure consistent replication and controlled modification of artifacts produced during the design process. There are several interrelated aspects to configuration management. The primary requirement is a consistent repository of data. This repository contains identification of the design environment, a collection of design artifacts sufficient for consistent replication, and verification data that provides sufficient evidence that the design meets its requirements. Information in this repository needs to be maintained such that controlled changes preserve a consistent set of data.

Configuration Management Tools

Configuration management for this project is supported by two tools, the Concurrent Versions System (CVS) [4] and GNATS [5]. Both tools are freely available on the Internet and are distributed under the GNU public license. Both of these tools provide access for users from both UNIX and Windows systems. This is necessary for the SPIDER project because the development involves data generated on both UNIX and Windows platforms. Additionally, both of these tools have auxiliary programs that provide a web-based interface.

CVS is a revision control system. As such, it maintains a history of changes to the controlled project files. Specifically, it records:

- who makes a change
- when it is made
- why it is made (if the user provides meaningful log messages)
- what other changes are made at the same time

The change history is traceable. CVS does not implement any change control policies. However, it can be used in a manner where changes are restricted. CVS is based on a *copy-modify-merge* philosophy for version control. Specifically, each developer maintains a copy of their part of the development tree, implements changes on the local copy, then merges the changes into the central repository. Specific policies can be implemented to restrict the authority to commit changes to the central repository.

GNATS is a problem reporting system. Problem reports are submitted via e-mail and are automatically logged into a database and forwarded to a responsible party. The problem reports can later be updated to reflect actions taken to resolve the problems. The problem report database is always accessible for review. As the design is modified in response to problem reports, the CVS change logs can readily reflect the problem report number from GNATS.

The GNATS system can also be used to log other reports generated during the course of a design. For this project, we are using GNATS to record all of our process-control actions.

The hardware design life cycle data is being accumulated following the process data-flow model depicted in Figure 1. The emphasis is on managing all data that will serve as a basis for certification. The data flow depicted in Figure 1 corresponds to controlled data. The central idea is that data cannot be controlled unless all the data on which it depends is also controlled.

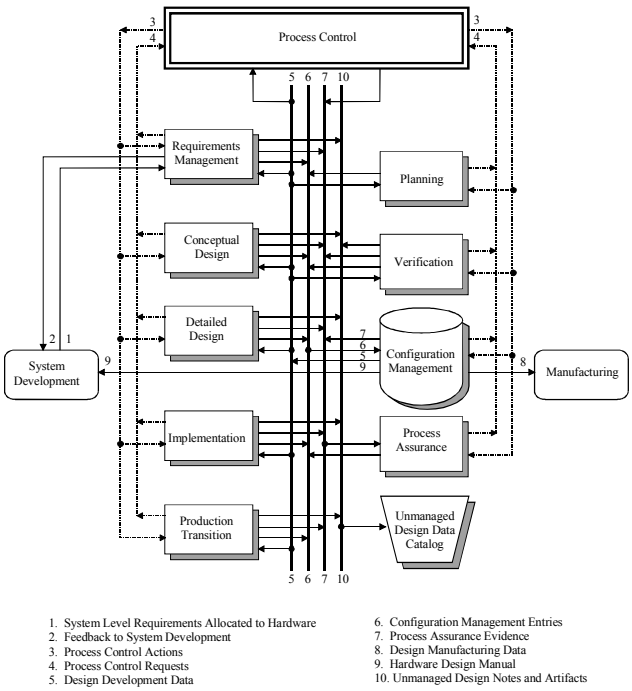

Figure 1: Documented Hardware Design Life Cycle

Design Assurance Strategy

For this project, we are developing the ROBUS to support any level A aircraft function. Since the design is being developed to the most stringent requirements, it is not necessary to justify the selected design assurance level. If we desired to develop part of the hardware system to a lower design assurance level, we would need to justify our decision using a Functional Failure Path Analysis (FFPA). A high-level description of performing an FFPA is presented in Appendix B of DO-254 [1]. A worked example illustrating an FFPA is presented in [6].

Our primary strategy is to focus on ensuring correctness at the conceptual design stage and then preserving the design integrity as we proceed through detailed design and implementation. Since the certification basis depends upon the conceptual design, the conceptual design data will be maintained under configuration management.

This is not the only strategy allowed by DO-254. In fact, DO-254 does not require conceptual design data to be controlled, if it is not used to

support certification arguments. When a system is being developed to assurance level A or B, DO-254 requires that the design assurance strategy be developed using the guidance of Appendix B [1].

Selection of Design Assurance Method

Subsection 3 of Appendix B in DO-254 suggests a number of methods that may be appropriate for a level A design assurance strategy [1]. The methods enumerated include architectural mitigation, service history, and three advanced analysis techniques:

- Safety-Specific Analysis
- Elemental Analysis
- Formal Methods

Architectural mitigation strategies are employed to constrain potentially adverse effects of errors in the design. The ROBUS may ultimately be used as part of an architectural mitigation scheme, but there is no architectural solution to mask design errors within the ROBUS. Hence, we cannot base the design assurance of ROBUS on architectural mitigation techniques. Also, since ROBUS is a new design, we cannot appeal to service history as part of our design assurance strategy. Therefore, we need to consider the suggested advanced analysis techniques.

Since we have expertise in the application of formal methods, that will be the core of our design assurance strategy. We will focus on formal proof at the conceptual design level to ensure that the SPIDER family of fault-tolerant systems is correct. We will then use conventional design and verification techniques to ensure that our detailed design and implementation are correct realizations of our conceptual design.

Fault Assumptions

There are at least two approaches to reasoning about faults and failures in a digital system. One is to postulate possible component failures and then assess the resulting impact on the system. Alternatively, one may assume that all faults have potentially devastating consequences and then design the system relative to this worst case assumption. Our approach is closer to the latter, but we allow some variation into the potential impact of faults. We will adopt the fault-classification strategy used in the development of the Multiprocessor Architecture for Fault-Tolerance (MAFT) [7]. Faulty nodes are globally classified based on the locally observable characteristics to other nodes within the system. The system is partitioned into Fault Containment Regions (FCRs) that ensure independence of random physical failures. The failure status of an FCR is then one of four mutually exclusive possibilities. An FCR may be

- Good
- Benign Faulty: All good nodes that observe its behavior know that it is bad
- Symmetric Faulty: All good nodes observe consistent error manifestations, but do not know that it is bad
- Asymmetric Faulty: No assumption is made about the behavior. The behavior of an asymmetrically faulty unit has different manifestations to at least two distinct good nodes

Several formal verifications have been performed using this fault classification. We will be able to adapt some of these proofs to the conceptual design of the SPIDER. When the development reaches the stage of a detailed implementation, we will use this same classification when conducting the Failure Modes and Effects Analysis (FMEA).

Reliability Analysis

The system level reliability analysis uses the Semi-Markov Unreliability Range Evaluator (SURE) tool developed at NASA Langley Research Center [8]. Some of the reliability models have been generated using the Abstract Semi-Markov Specification Interface to the SURE Tool (ASSIST) [9]. The developers of these tools and techniques are available for consultation on this project. In addition, they are available for expert review of the generated models.

Additional Considerations

Several of the verification activities employ formal methods. As part of the conceptual design activities, the algorithms for providing the fault-tolerant services are being formally specified and

verified using PVS [10] (http://pvs.csl.sri.com/). The models, algorithms, and proofs will be reviewed by Langley personnel that have expertise in both formal methods and fault-tolerance. In addition, some of the detailed design artifacts may be subjected to formal analysis.

The focus of the verification activities during the detailed design and implementation stages will be to preserve the integrity of the verification performed at the conceptual design level. We intend to explore elemental analysis and safety-specific analysis as we proceed, but we do not yet have sufficient understanding of these techniques.

Tool Assessment and Qualification

Section 11.4 of DO-254 details the requirements for tool assessment and qualification. If the output of a tool is independently checked in some manner, it is not necessary to qualify the tool. Given the limited resources of this project, it is not feasible to undertake a tool qualification exercise. Therefore, in this design effort, the output of each tool will be independently checked.

Design Concept

The system concept for this case study is the SPIDER family of fault-tolerant architectures. One of the design goals is that the SPIDER will support various fault-tolerant configurations. This will enable experimentation with different schemes for automatic recovery from multiple correlated transient faults.

The SPIDER architecture is intended to support a collection of *N* simplex general purpose processing elements communicating over a Reliable Optical Bus (ROBUS). One logical view of the SPIDER architecture is depicted in Figure 2.

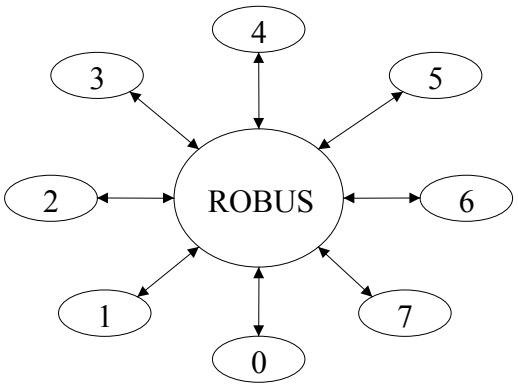

Figure 2: SPIDER Logical View

The ROBUS behaves as a time-division multiple access (TDMA) broadcast bus. For the ROBUS to provide unhindered access to all good nodes, it must be protected against any one node monopolizing its capacity. Furthermore, the communication model must support several fundamental services. The essential goal is to ensure reliable communication between all pairs of fault-free processing elements in the system. To ensure this flexibility, the ROBUS design shall guarantee that all *good* processing elements observe an identical sequence of messages. This will enable the development of several fault-tolerance strategies combining the simplex nodes. For example, Figure 3 illustrates a possible SPIDER configuration with three processors in a Triple Modular Redundant (TMR) configuration, four processors in a dual-dual configuration and a single simplex processor.

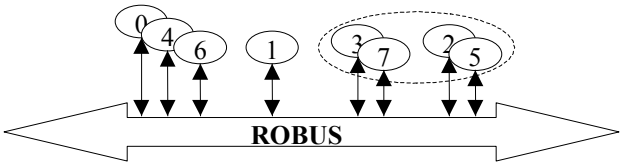

Figure 3: Sample SPIDER Configuration

Key Design Requirements

The primary requirement for the ROBUS is that it shall ensure that all fault-free attached processing elements observe identical message sequences, even if there are a bounded number of

physical component failures within the ROBUS. This implies that the ROBUS will be a realization of a special purpose fault-tolerant device. If the ROBUS can be shown to meet this requirement, then we have assurance that the failure modes of the attached processing elements are limited to symmetric or benign manifestations only. It will be impossible for an attached node to exhibit asymmetric behavior.

Schedule Agreement

The first implementation of the SPIDER will be based upon an assumption of a static, predetermined communication schedule. This is similar to the approach taken for ARINC 659 [11] and the Time-Triggered Architecture [12]. The communication protocol for the first implementation of the SPIDER will be based upon the protocol developed by Malekpour for the FBL/PBW testbench [3]. This is a statically scheduled TDMA protocol.

All analysis will be based upon the weaker assumption that all fault-free nodes agree on the communication schedule. This will allow future exploration of dynamic scheduling algorithms for later instances of the SPIDER architecture.

Interactive Consistency

In a redundant computer system, it is necessary to ensure that all single-source data items are consistently replicated among the redundant computational elements. Otherwise, a single faulty source may overwhelm the redundancy in the system. There are several published algorithms for ensuring interactive consistency; the first fully general solution is by Pease, Shoshtak, and Lamport [13]. Interactive consistency requirements are:

Agreement---All non-faulty receivers agree on the single-source data value received

Validity---If the originator of the data is non-faulty, then all non-faulty receivers receive the transmitted value

Protocols achieving interactive consistency are frequently referred to as Byzantine Agreement protocols, following the presentation of the problem in [14]. Byzantine agreement protocols depend on the assumption that redundant elements fail independently. Specifically, it is required that the nodes participating in the protocol are sufficiently physically and electrically isolated to ensure that a fault in one node cannot cause a fault in another node. These isolation regions of the design are termed Fault-Containment Regions (FCRs). An FCR may exhibit erroneous behavior. Additional logic is required to address potential error propagation. This is only possible if a sufficient number of FCRs are fault-free. There are several examples of formally verified interactive consistency algorithms available. The internal topology of the ROBUS is sufficiently similar to the Draper FTP architecture [15] that we were able to adapt its interactive consistency protocol. In addition, we were also able to adapt the PVS verification presented by Lincoln and Rushby [16].

Clock Synchronization

Both interactive consistency and TDMA scheduling require that the redundant nodes be synchronized within a known skew. The general requirements for clock synchronization are:

Precision---There is a small constant *d* such that for any two *good* clocks at real time *t:*

$$|C_1(t) - C_2(t)| < d$$

Accuracy---All good clocks maintain an accurate measure of the passage of time

As in the case of interactive consistency, clock synchronization protocols assume that the redundant clocks are in separate FCRs and that a sufficient number of FCRs are fault-free. There are several clock synchronization protocols discussed in the research literature. Ramanathan et al provide a survey of different approaches [17]. We have adapted a synchronization scheme proposed by Davies and Wakerly [18] for use in the ROBUS. There are established techniques for formal verification of clock synchronization algorithms [19] [20]. We have modified the approach presented in [20] for the verification of the SPIDER synchronization protocol.

Diagnosis

The ROBUS shall support distributed diagnosis in the presence of a bounded number of FCR failures. The goals of a diagnosis algorithm are to ensure the following properties:

Correctness---Every FCR diagnosed as faulty by a good FCR is indeed faulty

Completeness---Every faulty FCR is eventually diagnosed as faulty

There exist fault scenarios where it is impossible to identify which FCR is faulty, so in these cases diagnosis is necessarily incomplete. However, it is essential to always ensure the correctness property. The ROBUS will be designed against a modified completeness property that is consistent with the fault assumptions of the clock synchronization and Byzantine agreement protocols. We will adapt the algorithms and verification presented in [21].

Concluding Remarks

We are currently involved in the conceptual design phase of a case study exercising the new RTCA document DO-254: *Design Assurance Guidance for Airborne Electronic Hardware.* For the case study, we have chosen to design a central subsystem of a new fault-tolerant architecture. For this design, we have chosen to emphasize early life-cycle development and verification activities. It is our belief that if we get the conceptual design right, then it will be easier to assure correctness of the detailed design and implementation.

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

Acknowledgements

We are grateful to Leanna Rierson and Pete Saraceni of the FAA for partially funding this effort under Interagency Agreement DTFA03-96-X-90001.

References

[1] RTCA, 2000, *DO-254: Design Assurance Guidance for Airborne Electronic Hardware,* RTCA, Inc., Washington, DC.

[2] Palumbo, D.L., 1996, *Fault-tolerant processing system,* United States Patent 5,533,188.

[3] Malekpour, M., To Appear, *Fly-by-Light/Power-by-Wire Fault-Tolerant Fiber-Optic Backplane,* NASA Contractor Report, NASA Langley Research Center, Hampton, VA.

[4] Free Software Foundation, 1998, CVS - Concurrent Versions System, *http://www.gnu.org/software/cvs/cvs.html*.

[5] Osier, J. M., and B. Kehoe, 1996, Keeping Track: Managing Messages with GNATS, *http://sourceware.cygnus.com/gnats/gnats_toc.html*.

[6] Beland, S.C., and B. BonJour, 2000, Functional Failure Path Analysis of Airborne Electronic Hardware, in *Proceedings of the 19th Digital Avionics Systems Conference,* Philadelphia, PA.

[7] Kieckhafer, R. M., C. J. Walter, A. M. Finn, and P. M. Thambidurai, 1988, The MAFT Architecture for Distributed Fault Tolerance, *IEEE Transactions on Computers, 37* (4), pp. 398-405.

[8] Butler, R. W., and A. L. White, 1988, SURE Reliability Analysis: Program and Mathematics, *NASA Technical Paper, 2764*.

[9] Johnson, S.C., and D. P. Boerschlein, 1995, *ASSIST User Manual,* NASA Technical Memorandum 4592, NASA Langley Research Center, Hampton, VA.

[10] Owre, S., J. Rushby, N. Shankar, and F. von Henke, 1995, Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS, *IEEE Transactions on Software Engineering, 21* (2), pp. 107-125.

[11] ARINC, 1993, *ARINC Specification 659: Backplane Data Bus,* Aeronautical Radio, Inc., Annapolis, MD.

[12] Kopetz, H., 1997, *Real-Time Systems: Design Principles for Distributed Embedded Applications,* Kluwer Academic Publishers, Boston.

[13] Pease, M., R. Shostak, and L. Lamport, 1980, Reaching Agreement in the Presence of Faults, *Journal of the ACM, 27* (2), pp. 228-234.

[14] Lamport, L., R. Shostak, and M. Pease, 1982, The Byzantine Generals Problem, *ACM Transactions on Programming Languages, 4* (3), pp. 382-401.

[15] Smith, T.B., 1984, Fault Tolerant Processor Concepts and Operation, in *Fourteenth International Conference on Fault-Tolerant Computing*, IEEE Computer Society Press, pp. 158-163.

[16] Lincoln, P., and J. Rushby, 1994, Formal Verification of an Interactive Consistency Algorithm for the Draper FTP Architecture Under a Hybrid Fault Model, *Proceedings of the Ninth Annual Conference on Computer Assurance,* pp. 107-120.

[17] Ramanathan, P., K. G. Shin, and R. W. Butler, 1990, Fault-Tolerant Clock Synchronization in Distributed Systems, *IEEE Computer, 23* (10), pp. 33-42.

[18] Davies, D., and J. F. Wakerly, 1978, Synchronization and Matching in Redundant Systems, *IEEE Transactions on Computers, C-27* (6), pp. 531-539.

[19] Miner, P.S., 1993, *Verification of Fault-Tolerant Clock Synchronization Systems,* NASA Technical Paper 3349, Hampton, VA.

[20] Schwier, D., and F. von Henke, 1998, Mechanical Verification of Clock Synchronization Algorithms, *Proceedings 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems,* pp. 262-271.

[21] Walter, C. J., P. Lincoln, and N. Suri, 1997, Formally Verified On-Line Diagnosis, *IEEE Transactions on Software Engineering, 23* (11), pp. 684-721.