



ISSUES IN THE STANDARDISATION OF HARDWARE FOR HIGH INTEGRITY SYSTEMS FOR APPLICATION TO AVIONIC SYSTEMS *

0 INTRODUCTION

The use of programmable digital hardware in high integrity systems has been increasing over the past ten years. Reasons for this include ease of use, adaptability, physical stability, potential for built-in integrity, etc. Areas where digital technology has been applied include fly-by-wire aircraft control systems, autonomous robotics and command and control systems.

The development of software for safety critical systems has been addressed at many levels and is the subject of standardisation activities sponsored by the UK MoD with Def Stan 00-55, the FAA in the US with RTCA 178, and the US DoD with DOD STD-2167A. These activities address software solely. The concept of the system and the use of a system wide approach to high integrity is given little or no concern. This is especially true for hardware.

The use of hardware in high integrity systems is usually decided at the requirements capture stage in the system life cycle. It will always be a system issue rather than isolated to individual components of the hardware. The techniques available for hardware are seen in this document in the context of the system as a whole and are not considered in isolation. It is important that all the individuals involved in the project realisation have a coherent view of the fault tolerance philosophy adopted. Establishing this philosophy at the outset of a project can be viewed as a risk reduction process.

* **This report is published by the Avionic Systems Standardisation Committee (ASSC) to advance the role of standardisation in avionics. The use of this is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom is the sole responsibility of the user.** Copies of this paper are obtainable from the ASSC Agency as an Avionic Systems Standardisation Committee publication.

Table of contents		Page no.
0	Introduction	1
1	Scope	3
2	Definitions	3
3	Referenced Documents and Acknowledgements	3
4	Avionic Systems	4
5	The Need for Fault Tolerance	4
6	System Issues	5
	6.1 Specification	5
7	Fault Tolerance Considerations	6
	7.1 Error Detection	6
	7.2 Error Confinement	8
	7.2.1 Error Isolation	8
	7.2.2 Error Assessment	8
	7.3 Error Recovery	8
	7.3.1 Forward Error Recovery	9
	7.3.2 Backward Error Recovery	9
	7.4 Fault Treatment and Continued Service	9
8	Conclusions	10

1 SCOPE

This document is concerned with the use of fault tolerance to achieve high integrity. The application domain under consideration is avionic systems. It was originally intended that this document should cover hardware only. However, in the development of the document it has become apparent that the application of fault tolerance techniques is fundamentally a system issue. For this reason this document treats hardware as part of the system. The system will include the software but software is not explicitly considered in this document as this is adequately covered elsewhere. Operating systems, where hardware and software are closely related, are considered to be within the scope of this document.

The use of fault tolerant techniques discussed in this document in areas other than that which concerns the ASSC is not precluded.

Tools to support the development of high integrity technology are similar to those which support the development of hardware in other applications and will not be considered further in this document.

This document is intended to be used by all individuals involved in avionic projects where fault tolerance is an issue. Such individuals could include project managers, both MoD and contractors, system designers and implementors and those individuals involved in the quality aspects of a project.

2 DEFINITIONS

There have been a number of attempts to establish an agreed terminology in this area, e.g. reference 3.1.4 below. In cases where these definitions are not fully applicable, the alternative will be given for use in this document only.

3 REFERENCED DOCUMENTS AND ACKNOWLEDGEMENTS

The following documents and publications are referred to in this document:

Def Stan 00-55	The Procurement of Safety Critical Software in Defence Equipment. Part 1, Requirements and Part 2, Guidance
RTCA/DO-178B	Software Considerations in Airborne Systems and Equipment Certification
DoD-STD-2167A	Defence System Software Development
IEEE 610.12	Glossary of Software Engineering Terminology (update to ANSI/IEEE Std 729-1983), Nov, 1989.
Institution of Electrical Engineers	Guidelines for the documentation of computer software for real time and interactive systems (2nd edition) 1990

The following book has been used as a reference:

"Fault Tolerance - principles and practice," T. Anderson and PA Lee.

Significant contributions have been made by members of the Avionic Systems Standardisation Committee, Computing Working Group.

4 AVIONIC SYSTEMS

The scope of this document is that of avionic systems. The use of the document is not precluded outside this application domain but the specific nature of avionic systems places certain requirements on the discussion. Avionic systems are intrinsically real-time. Real time systems are defined as those systems where the failure to meet a deadline will impair the operation of the system. The level of impairment will depend on the deadline not met.

The overall architecture for most avionic systems is, though need not necessarily be, functionally distributed. Sensors and effectors are connected (using point to point or by data buses) to computers (e.g. line replaceable units, line replaceable modules, smart sensors, smart actuators, etc.) which communicate by means of one or more data buses.

5 THE NEED FOR FAULT TOLERANCE

The requirements for fault tolerance in military avionics systems arise from two main areas of concern. These are safety concerns and economic concerns. Fault tolerance is used to reduce the likelihood of the occurrence of accidents which result in injury to people or damage to property. Additionally, it is used for economic reasons. The cost of avionic systems is now around 40% of the cost of an aircraft. This figure is rising and one of the ways of mitigating this increase is through the use of fault tolerance. Fault tolerance can allow degraded performance of a system until scheduled maintenance can be carried out.

The adoption of fault tolerance techniques will usually necessitate extra hardware and therefore complexity. This extra complexity will reduce the reliability of the system as a whole thereby, paradoxically, countering the intent of introducing the extra hardware.

Fault tolerant systems are more expensive in early stages of the life cycle. It is possible that the extra expense incurred in these stages will not be recouped. A trade off has to be made between the benefits the fault tolerance brings and the expense of implementing the capability. The serious consequences of failure in avionics systems and the high costs of maintenance will usually make the implementation of fault tolerance techniques worthwhile thereby reducing life cycle costs overall.

6 SYSTEM ISSUES

The incorporation of fault tolerance into a system has implications at all levels of the system life cycle. It is fundamentally a system issue. For this reason a top down design methodology is required to ensure coherency. The partitioning of the required fault tolerance functionality amongst hardware and software and the interfaces between these partitions and the application functionality are required at an early stage and must be rigorously defined and agreed.

Many of the terms used in this area are not established or agreed. Def Stan 00-55 does not contain a definition of reliability. In order to provide a common terminology for these discussions, a glossary of frequently used terms may be found in the IEE Guidelines for the documentation of computer software for real time and interactive systems (2nd edition) 1990.

6.1 Specification

As a consequence of the definition of reliability a fundamental requirement is that a system be specified in terms of its functionality. This specification will be generated from the requirements as identified by the contractor, the customer, the user if they are not the same and possibly the regulator. An exact specification is required to judge system behaviour definitively. An exact specification is complete, consistent and applicable.

A complete specification is one that describes the behaviour of a system for all possible inputs and all possible states of the system. This form of specification is impractical and uneconomic to produce. However a specification can be made more complete by stating that the system's behaviour is indeterminate under circumstances not defined in the specification. The performance of the system cannot be proved to be incorrect if the circumstances and result of any unexpected behaviour are not defined in the specification.

The English language has traditionally been used for specification writing. This approach is now considered by many to be inappropriate for the specification of both hardware and software. English is open to interpretation in a number of different ways. The use of more formal methods for systems specification is recommended to avoid the discrepancies arising from different interpretation. Such methods include Yourdon, VDM and the Z specification language and tools to support the use of such methods are commonly available, if not in widespread use.

A further technique which may be applied to refine specifications is the modelling of the specified system. In this manner the feasibility of implementation may be optimised before committing to full scale development. Modelling can take into account such things as the definition of system boundaries and functional partitioning. It also allows some consideration of the interaction of the system with its environment and vice-versa. The use of hardware description languages (e.g. VHDL and ELLA) for this purpose may be advantageous since it is then possible to relate a model of a specification directly to a

hardware design. Automatic tools for this purpose are available. Other tools support the modelling of systems include Statemate and SES Workbench.

7 FAULT TOLERANCE CONSIDERATIONS

A fault in a system is an entity that causes an error. An error is the tangible state arising from a fault. In most systems the likelihood of fault arising in a system can be mitigated by the use of fault avoidance. Fault avoidance is the use of techniques, components and practices which have been successfully and widely used and are therefore known to be reliable under certain circumstances. One example of such a component is the microprocessor used in personal computers; another could be the operating system. It may be that the dependability requirements can be met through the use of fault avoidance only.

Once it has been established that fault avoidance has been employed where possible, faults will have to be tolerated in order to improve dependability. There are four main aspects in the provision of fault tolerance in systems. These are: Error Detection, Error Confinement, Error Recovery and Fault Correction/System Continuance. Each of these will be dealt with separately in the following sections.

7.1 Error Detection

Detection of an erroneous condition is the first step in all fault tolerance strategies. The effective detection and categorisation of errors will facilitate effective measures to tolerate the fault which produces the error. Since it is generally impossible (uneconomic) to develop a complete specification it will be impossible to provide a scheme and develop a system to detect all errors as required by a specification.

The basic mechanism for detection of errors is the check. Checks for errors are made against the system specification. In order to avoid errors resulting from identical errors in both the system under examination and the checking system, the implementation of each should be independent. However, the use of checks is implementation dependent since the check must have knowledge of that which is checked. A checking system which checks for all possible errors is clearly untenable. It is possible, however, to check for acceptability. Given that in most cases, systems are functioning correctly (i.e. they have been professionally specified, designed, implemented, debugged and commissioned) a system's behaviour is more likely to be within the specified limits and so acceptability testing is satisfactory.

For these reasons the aim of checking is to detect the majority (not all) of erroneous situations and thereby improve reliability to that which is required by the specification.

Checks which are possible include:-

- a. **Replication Checks:** These checks are powerful and complete. They are simple to implement but expensive for anything other than very simple systems. The technique uses exact replicas of system being checked to provide the necessary

redundancy. This technique is exemplified by Triple Modular Redundancy systems, where the outputs from three identical systems are compared. Replication checks cannot detect specification or system design errors. Such errors can be avoided by the use of multiple design teams to implement functionally similar but differently implemented systems. This approach is very expensive and may not produce the desired result since different teams are composed of engineers who use similar techniques and tools to specify and design hardware.

- b. **Timing Checks:** Where a system includes temporal aspects, then the operation of the system can be checked to see if it is within the specified limits. The lack of an error generated by timing checks does not indicate correct system operation, however. The use of a watchdog timer is a well known example of such a technique.
- c. **Reversal Checks:** This technique is used in simple systems where it is possible to calculate the inputs to a system from the outputs which are observed. These calculated inputs are compared with the actual inputs to decide correct operation. This approach is difficult to implement in anything other than very simple systems.
- d. **Coding Checks:** This type of checking is based on redundancy in the representation of an object. The check is formed by some fixed relationship between the data and the check. A well known example is the use of parity to provide a check on a sequence of bits. Other examples include Cyclic Redundancy Checks, Hamming codes, Reed Solomon and Convolutional codes. Generally the more redundancy in a message the more likely an error can be detected, although algorithmic techniques make this relationship non-linear. This approach provides an efficient and economical way of providing redundancy.
- e. **Reasonable Checks:** These checks are used on abstract data structures and check for values which are within a particular range. This implies a knowledge of the values which may exist in the system and therefore the implementation of the system. This type of checking is usually found in software although the use of such techniques is not precluded in hardware.
- f. **Structural Checks:** These are checks on the integrity of the structure of data. These checks are similar to reasonable checking but usually apply to complex data structures, e.g. data linked by pointers i.e. lists, queues and records.
- g. **Diagnostic Checks:** These are checks of the components in the system rather than a check on the behaviour of the system. This involves correlating the outputs of a system with an expected set of outputs for a given set of inputs. Diagnostic checks are often used to try to refine the location of errors in a system which are visible from some other aspect of system behaviour.

In any particular system the use of checking will be dictated by the requirement embodied in the specification. From a hardware point of view check types e) and f) are more applicable in software rather than hardware. Check d) may be implemented in either hardware or software but the more specialised algorithms such as Reed Solomon coding are best implemented in software.

The use of the various checks mentioned will depend on the implementation of the system. The use of software or hardware to implement these features is within the remit of the system specifier/designer any will depend upon the performance and reliability requirements.

7.2 Error Confinement

This has two aspects which should occur following the detection of an error. These are error isolation and error assessment.

7.2.1 Error Isolation

The confinement or isolation of a fault is the action which must be taken first to restrict the propagation of faults throughout the rest of the system. Once again, the approach to implementation of fault confinement is defined at the specification design phase of a project. In order to facilitate fault isolation the potential areas where faults can occur are assessed and packaged as a unit with a minimum of rigidly defined interfaces. This approach is applicable to both hardware and software. It is the system analyst/designer who will be responsible for implementing the modularity in a system to allow fault isolation. For modular avionics a suitable atomic unit would be a module.

7.2.2 Error Assessment

Error assessment is central in a fault tolerance scheme. The errors are assessed and this will then determine the actions, if any, which are required to recover from the error. The first action in error assessment is an estimate of the damage which has occurred. This is followed by an estimate of the potential for propagation of the fault which has caused the error.

Static damage assessment is where the assumption is made that all of the functionality of the unit under consideration is damaged. The faulty unit is then de-commissioned and the damage cannot spread as a result. The alternative approach for damage assessment is dynamic damage assessment. This approach uses additional software or hardware to refine the assessment of an error inside a unit under assessment. This approach is more difficult to implement since the potential for error propagation must also be considered. However, if the use of the maximum functionality of a degraded system unit is a requirement then this approach should be used.

7.3 Error Recovery

The first two phases of the fault tolerance mechanism are passive in that no action is taken to change the behaviour of a system. Error recovery, fault treatment and continued service

are active since a system will change as a result of the process. The aim of these phases of the fault tolerance mechanism is to regain some or all of the system functionality.

In error recovery the aim is to restore the system to an error free state. Return to full functionality or state can only be possible where the faults are anticipated by the specification and the fault assessment phase is effective enough to identify faults which have occurred. Unanticipated faults may be recovered from but the mechanism, since it is not specified, cannot be reliable.

Two options are available. These are forward and backward error recovery.

7.3.1 Forward Error Recovery

This approach institutes a system state which had not previously been entered. The intention here is to avoid a repeat of the circumstances which led to the error. A transition is made must be one which cannot occur in normal operation. This is exemplified by the shut-down mechanisms in areas such as nuclear power stations. For continued service as would be required in real time avionics systems this mechanism is appropriate only in situations where degraded service is acceptable.

7.3.2 Backward Error Recovery

This approach institutes a system state which had previously been entered. It requires some memory of the system state information and an on-going update of this information. Backward error recovery can be implemented quite easily since it is a reversion to a previous known state. All backward error recovery mechanisms use recovery points. A particular state in the system sequence is deemed the recovery point and all subsequent states use the stored recovery data to facilitate the restoration of a recovery point. A number of recovery points may be established in a system thereby reducing the amount of data storage required. An example of this is the grandfather, father, son method of computer backup.

The use of backward error recovery is not always successful. If the system, as a result of an error, is placed in a known state and provided with a set of inputs which produce a fault, the fault will always result. This produces a functionality which may be resilient but which is not that which is useful.

7.4 Fault Treatment and Continued Service

This is the final phase of the fault tolerance scheme. In this phase an attempt is made to correct the fault or the results of the fault and to provide a continued service, possibly degraded. In modular avionics systems this is known as reconfiguration.

Fault treatment techniques can be implemented wholly in hardware. For example, some wafer scale devices are designed to take into account the intrinsic physical defects of silicon wafers. This is fairly specialised and such capabilities are usually implemented using software. In real-time avionics systems it is important to bear in mind that fault treatment will channel processing capability away from an already degraded service. The implementation in software is also very difficult since software contains only systematic

faults. The concept of self modifying software is an advanced one and generally regarded as being very dangerous, particularly in a real-time system.

Where this type of functionality can be implemented it will be implementation specific and again it will be formulated early in the system analysis/design phases of the life cycle.

8 CONCLUSIONS

A number of conclusions can be drawn from the above discussions.

The use of fault tolerance generally and fault tolerant hardware specifically is a system issue. As such it is essential that it is considered at the earliest stage of system analysis and specification. In this way a coherent approach to the implementation of fault tolerance can be adopted.

The specification of hardware systems in English and the use of formal mathematical methods is a subject which is still widely debated, especially for safety critical applications. The development of modular avionics systems has, for mainly economic reasons, become multi-national activities based on standards development. English is the working language but it is not the first language for many participants and the use of formal methods or formalised languages will help eliminate linguistic anomalies which may arise. Additionally other advantages may be available.

The specification is pivotal in systems design especially for high integrity systems and some form of automated capability should be used in all specifications. The use of hardware description languages for high integrity components or components to be used in high integrity systems is also one which should be recommended. This allows system modelling and thereby reduces the unanticipated failures and systematic design failures that are difficult to accommodate in fault tolerance schemes.

Within high integrity systems much of the hardware components will be implementation specific as a result of choices made at system design time. Standardisation at the component level is not, therefore, appropriate. A better method to allow standardisation for high integrity hardware is at the interfaces between the hardware components and the general approach to fault tolerance. An example of this is the IEEE P1149.1 standard, "Test Access Port and Boundary Scan Architecture" which standardises on the use of a serial port for clocking test vectors in and out of a device and the facilities a component should provide, but does not standardise on the format of the data or use to which it can be put. This allows the standard to be independent of the implementation, resulting in a degree of technological independence.

R P Gray
Electronic Systems Division
ERA Technology Ltd.