Reliability and Redundancy

ENGN8537

Embedded Systems

Overview

- Reliability, Failure and Faults
- Failure modes and fault prevention
- Dynamic redundancy

Ref: Burns and Wellings *Real Time Systems and Programming Languages*

Reliability, Failure and Faults

Sources of Faults

- Inadequate Specification
- Design Errors
- Processor Failure
- Interference on communications channels, interface faults

Reliability, Failure and Faults

Sources of Faults

- Inadequate Specification Not covered
- Design Errors
- Processor Failure Not covered
- Interference on communications channels, interface faults Not covered

Reliability, Failure and Faults

A system's reliability is a measure of how well it conforms to some specification of its behaviour. A deviation from specification is called a failure.

Failures result from unexpected internal behaviour that manifests itself in the system's external behaviour.

If a system has a problem which doesn't manifest itself on an external interface, it cannot be called a failure.

Reliability, Failure and Faults

The internal problems that result in failure are called errors and their mechanical or algorithmic cause is termed the fault.

Systems are composed of components which are themselves systems, hence

➔ Failure ➔ Fault ➔ Error ➔ Failure ➔ Fault ➔

Failure Modes and Fault Prevention

Faults can be classified in the time domain

Transient Faults

Faults that come and go e.g. communications interference

Intermittent Faults

Transient faults that occur repeatedly e.g. overheating

Persistent Faults

Stay in the system until they're rectified by external means

Failure Modes and Fault Prevention

The failures caused by the faults can also be classified

Time Domain Failures

Result or action happens at the wrong time

Value Domain Failures

Result or action isn't correct or is inappropriate for the situation

Fail Uncontrolled

Arbitrary performance of the system as a whole

Fail Never

Ideal case!

Achieving Reliability

The core of a reliable system is to produce well-defined failure modes.

Fault Tolerance seeks to build systems that can provide correct function even in the face of system faults. That is, faults are allowed but don't cause errors (therefore don't become failures).

Fault Prevention attempts to eliminate any possible sources of fault before the system becomes operational

Achieving Reliability

Fault prevention has two stages, Fault Avoidance and Fault Removal. The successful completion of these stages depends on a thorough understanding of the system itself.

- Understanding constraints on materials
- Knowing boundary constraints on input and output systems
- Constraints of the processor system and software
- Dynamic application specifications (requested real-time behaviour)

Understanding all the critical Real Time constraints on the system!

Achieving Reliability

Fault avoidance seeks to prevent faults entering the design and construction

- Use high reliability components
- Use thoroughly tested methods and techniques for assembly and interconnect
- Packaging the hardware to physically prevent external interference, either on communication channels or otherwise
- Rigorous (preferably formal) specification of requirements and compliance procedures
- Use of proven design methodologies
- Use of standardised software and hardware engineering environments to manage complexity

Achieving Reliability

Fault Removal seeks to find and manage faults that make it through construction before the system becomes operational. Fault Removal covers aspects of the system design process such as

- Design reviews
- Testing procedures
- Program verification
- System testing
- Hardware Inspection
- Code Inspection

Achieving Reliability

Fault Removal procedures should be followed throughout the design phase but selected portions will often be applied to production systems as well. The testing of production boards may cover every board off the line or some representative sample.

Often, high reliability systems will have special test tags built on to the board itself. The tag has particular patterns of copper, solder, componentry etc. which are designed to 'stress test' the production procedure. The tags are snapped off the boards and filed for future reference if that board develops a fault.

Achieving Reliability

System testing can never be exhaustive and remove all potential faults. Testing can only show the presence of faults, not their absence.

It's often hard to test under realistic conditions, much of the effort in high reliability designs may be expended in providing high fidelity test and simulation environments.

Achieving Reliability

Recall: A system's reliability is a measure of how well it conforms to some specification of its behaviour, a failure is a deviation from this specification.

This is important as failures are defined with respect to the specification, not with respect to actual correct behaviour! The corollary is that errors that have been introduced in the specification may not be detected until the system becomes operational.

No amount of fault avoidance and removal can guarantee operation that has not been correctly specified.

Achieving Reliability

Faults may develop even in a high reliability system. In that case, they must not propagate to failures, or at least uncontrolled failures. That is, if a failure is unavoidable given the nature of the fault, it should at least fail in a known way. This leads to the concept of fault tolerant systems.

This is particularly important for

- Systems that are (temporarily) inaccessible
- Unmanned vehicles that operate autonomously by default
- Systems in remote and/or dangerous environments

Failure Modes and Fault Prevention

Full fault tolerance states that the system can continue (reasonably) correct operation in the face of all ‘foreseeable’ fault conditions.

The presence of the fault condition may, however, change some system parameters such as operational lifetime.

Failure Modes and Fault Prevention

Graceful degradation, also known as ‘fail-soft’, allows for the system to continue operation in the face of all ‘foreseeable’ fault conditions, accepting a reduction in functionality or performance.

Failure Modes and Fault Prevention

Fail safe is the term for a failure mode in which the system remains controlled and ‘safe’ during and after a failure, as it shuts itself down.

This can also be viewed as graceful degradation to zero functionality.

Failure Modes and Fault Prevention

The level of fault tolerance required will depend on the application and must be specified during the system design.

Most high reliability systems will request full fault tolerance, but settle for graceful degradation towards functionality that is not significantly reduced.

Failure Modes and Fault Prevention

The method for actually implementing different levels of fault tolerance can vary. The most common method is by some means of hardware redundancy.

These extra resources may be used for

- The early detection of faults that may cause failure
- For the handling of exceptional situations and error recovery
- As functional replacements for complete (sub)systems to facilitate hot swap procedures to remove faulty (sub)systems from operation.

Failure Modes and Fault Prevention

Common fault detection and recovery hardware includes

- Watchdog timers to detect when a hardware device hasn't "checked in" for some amount of time, assume it has failed and restart it
- Limit switches to detect when an actuator is being commanded to move past its maximum physical extent and shut down that actuator
- Additional physical sensors to provide a "sanity check" for each other
- Transient Recording Systems (emergency system dump) to at least provide a set of post-failure data to assist in analysis
- Overload backup systems
- etc.

Failure Modes and Fault Prevention

The unit of failure is often full subsystems, with fault tolerance then acting to isolate one failed system from interfering with the overall system function.

Triple-Modular Redundancy (TMR) or N-Module Redundancy (NMR) provides multiple copies of critical subsystems which are either

- Statically combined by a masking/voting/comparing system
- Dynamically switched in or out depending on their operation (assumes a failure can be unequivocally detected!)

Failure Modes and Fault Prevention

TMR or NMR are only useful if only one (or a small subset) of the systems fail at once. This then requires that

- The source of the fault is localised to one instance of the module, or
- The modules differ in their internal implementation sufficiently that the same fault in two modules lead to different failures or, for preference, at least once instance of no failure at all.

Failure Modes and Fault Prevention

For some high risk systems, this approach is applied in the form of redundant subsystems with

- The same specification
- Different computer systems (CPUs, busses, memory systems, drives etc.)
- Different Operating Systems
- Different software development languages and environments
- Even restricting the communication between different development teams

Failure Modes and Fault Prevention

TMR Example: Boeing 777

Contains three identical primary flight computers, each consisting of:

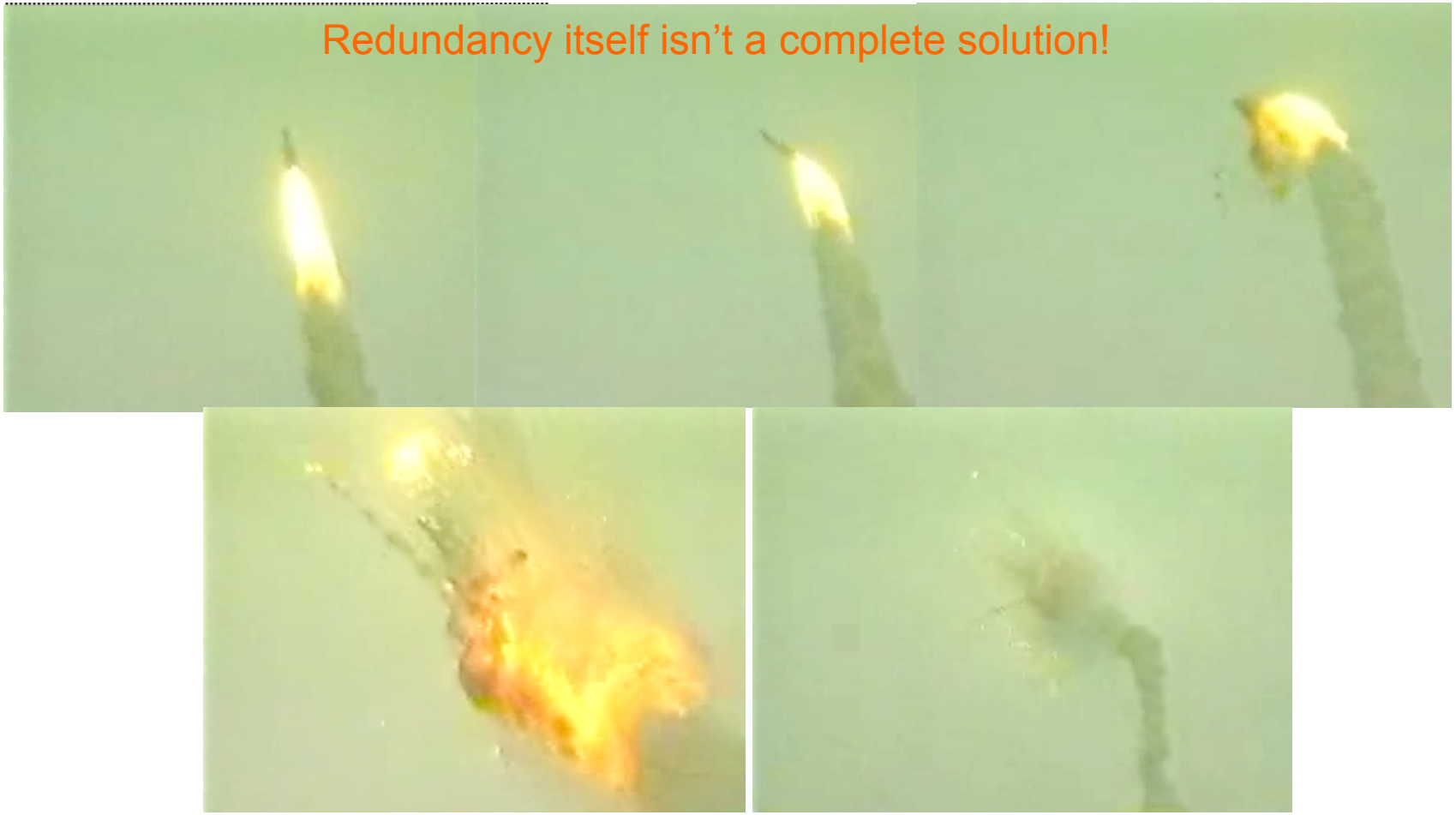
- Three processors: AMD 29050, Motorola 68040, Intel 80486 called ‘Lanes’. Two level redundancy!
- Independent power sources and inertia measurements
- Code built by three different ADA compilers
- The same ADA software source code (called ‘the specification’), but with different ‘monitor’ functions written to supervise correct operation.

Targeted failure probability < 10^{-10} /h: Over 1 million flight years per fault! (e.g. UK Seizewell B Nuclear Reactor (emerg.): 10^{-3}/h)

No single fault onboard a 777 should occur without failure identification

No single fault onboard a 777 should result in the failure of more than one flight computer

Failure Modes and Fault Prevention



Failure Modes and Fault Prevention

TMR Example: Ariane 5

The Ariane 5 rocket is a heavy-lifting platform designed to move satellites in to low-earth or geostationary orbit. In 1996, after 39 seconds of flight, the maiden launch of this platform ended in a self-destruct of the system at a cost of over $300 Million.

- The Ariane 5 shared its inertial reference system with the proven Ariane 4
- Ariane 5 followed a different launch trajectory to Ariane 4, but this trajectory had not been programmed in to the system simulators
- Shortly after launch, the inertial reference system crashed as a result of an input velocity that was too high – it was within the Ariane 5 flight envelope, but wasn't physically achievable on the Ariane 4 for which it had been developed
- This specification mistake was common across all three redundant systems, each failed in turn, the rocket become unstable, was ripped apart due to aerodynamic forces before exploding

Failure Modes and Fault Prevention

The Six Language Project

Joint project between UCLA and the Honeywell Commercial Flight Systems Division (1992) to determine which programming language provided the most reliable system outcomes.

Failure Modes and Fault Prevention

The Six Language Project

The specifications (about a flight controller) were original system specification documents provided by Honeywell enhanced by additional cross-check points and included some forced diversity elements. Development teams were isolated and communication strictly defined and controlled by a central team. Specified tests were performed by the coordinating team before a version was accepted for integration. The N-Version paradigm was applied to all stages of the development cycle

Failure Modes and Fault Prevention

The Six Language Project

Language	Source (l.o.c)	Test Runs	Errors	Failure Rate
Ada	2256	5127400	0	0
'C'	1531	5127400	568	1.108×10^{-4}
Modula-2	1562	5127400	0	0
Pascal	2331	5127400	0	0
Prolog	2228	5127400	680	1.326×10^{-4}
T (like LISP)	1568	5127400	680	1.326×10^{-4}
Average	1913	5127400	321	0.627×10^{-4}

Failure Modes and Fault Prevention

The Six Language Project

Failure Category	Average Single Version Failure	Average Three-Version Failure	Average Five-Version Failure
No Errors	0.99993733	0.9998409	0.9997807
Single Error	6.27×10^{-5}	13.05×10^{-5}	19.15×10^{-5}
Two Distinct Errors		0.20×10^{-5}	0.23×10^{-5}
Two Coincident Errors		2.65×10^{-5}	2.21×10^{-5}
Three Errors			0.34×10^{-5}

Failure Modes and Fault Prevention

The Six Language Project

The resulting three- and five-version systems displayed lower failure rates than the ‘gold master’ reference implementation from Honeywell. Coincident failures involving more than two versions were never observed, a total of 93 faults were detected

Failure Modes and Fault Prevention

TMR or NMR imply that multiple independent modules can have their results usefully compared. The truth of this statement depends somewhat on the actual form of the result.

Integer arithmetic has exactly defined results that can be compared between units in a straight-forward manner. How about other types of result?

Failure Modes and Fault Prevention

Real valued results (floating point) will usually be different by some small amount, comparisons must include some concept of a tolerance rather than a blunt comparison.

If the process isn't fully continuous (thresholds, bifurcations, quantizations) then comparisons need to re-model the whole process in order to evaluate similarities. In this process, you loose the system's independence and therefore the value of TMR/NMR.

Failure Modes and Fault Prevention

Some issues with NMR:

Specification

N-Version modules are built to the same specification so won't help if that specification is incorrect

Diversity Assumption

Diversity can be enforced in some areas, while coincident errors are observed in others. Rigorous identification of adequate domains for N-version programming is a current source of research

Project Costs

The development costs will increase by N times, plus the cost of coordination.

Dynamic Redundancy

Dynamic Redundancy is the process of identifying fault conditions at run-time and taking controlled action to rectify the fault before a system failure occurs. It consists of four phases:

- Error Detection
- Damage Confinement and assessment
- Error Recovery
- Fault Treatment

Dynamic Redundancy

Error Detection is the first and most critical step of dynamic redundancy. In order to act correctly to rectify a fault before it causes a failure, the nature of the error must be understood.

Error states might be reported by the environment

- Hardware: CPU, controllers, communications systems
- Run-time Environment

Dynamic Redundancy

Error states may also stem from the application process itself: Bugs! May fix by:

- Replication: Employ N-Version programming to detect error states
- Timing: Watchdog Timers and overrun detectors
- Reversal: Apply inverse function and compare, i.e. a ?= $f^{-1}(f(a))$
- Coding: Apply error check functions such as CRCs
- Reasonableness: Apply coded ‘sanity checks’ or ‘assertions’ to results
- Structural: Check structural integrity of data structures, e.g. File Systems, Lists, Databases etc.
- Continuity: Assume a maximum allowable delta between successive result values

Dynamic Redundancy

Confinement and Diagnosis is required once an error has been identified. The confinement condition is required to prevent the error propagating and affecting external systems

- 'Firewall'
- Atomic Action
- Modular Decomposition

Dynamic Redundancy

Assessment of the error is required by analyzing the location of the error and the possible paths of propagation both towards the error state and from it. The root cause of the error may not be able to be directly measured so the reverse paths from the error location must be analyzed in order to try and determine where the root fault might lie.

The forward paths are analyzed as well in order to determine what effects the error may have had before the containment could be enacted.

Dynamic Redundancy

The next step is error recovery, Forwards or Backwards:

Backwards Error Recovery

- Set checkpoints and save the state of the system with each passing checkpoint.
- If an error is detected, fall the entire system back to the last known good checkpoint.

Applicable even if the fault cannot be discovered.

Not applicable if any portion of the system is non-reversible, resettable or controllable (such as if a bad value gets sent across a communications channel before the system is checkpointed)

Dynamic Redundancy

Forwards Error Recovery

The error is detected and the result fixed before moving on. Usually the preferred method for embedded systems. The implementation of forward error recovery is highly application specific.

In a real-time system, there may not be time for a new result to be generated; a value error becomes a timing error.

Dynamic Redundancy

Once the error has been identified, contained, analyzed and recovered, the risk of failure has passed for that error.

The fault that generated that error may remain. The final stage of Dynamic Redundancy is to attempt to treat the fault that caused the initial error.

Hardware faults are usually easier to deal with than software faults. Redundant hardware modules may be switched in to place, sensors may be marked as faulty and not trusted etc.

Dynamic Redundancy

Granularity in Dynamic Redundancy is usually better than normal N-Module Redundancy, however the number of redundant modules available is still limited.

Many Dynamic Redundancy systems will assume transient fault conditions and retry the same hardware at a later time, perhaps reporting a module as untrustworthy if it suffers multiple faults over some period of time.

Conclusion

- Faults cause errors that lead to failures
- Errors are only failures if
  - They are externally visible
  - They violate the external interface specification
- Faults in a subsystem should be *avoided* in the design process then *removed* in the review process.
- Some will survive and may require that the system then have some degree of fault tolerance
- This is usually by means of redundancy across hardware and software.
- The redundancy might be static (N-module) or dynamic (fix the error and get on with it).