CS_313 High Integrity Systems/ CS_M13 Critical Systems

Course Notes

Chapter 6: Fault Tolerance

Anton Setzer
Dept. of Computer Science, Swansea University

http://www.cs.swan.ac.uk/∼csetzer/lectures/critsys/14/index.html

November 28, 2014

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

(a) Introduction

- Faults are essentially unavoidable.
- **Fault tolerance** aims at designing a system in such a way that faults do not result in system failure.
- All techniques are based on some degree of **redundancy**.
- Many reasons for introducing fault tolerance – it can be reliability, availability, dependability, safety, security.
  - E.g. main memory in computers has always some error checking mechanism in order to tolerate errors due to radioactive particles. Goal is in this case high degree of availability (or reliability).

Historic Remark

- Fault tolerance already present in early computers.
  - The EDVAC (designed 1949) had duplicated ALUs in order to detect errors in calculations.
- Von Neumann (1956) developed the theoretical basis for fault tolerance.

Material Moved to Additional Material

The material for this subsection has been moved to the additional material, which is available from the website.

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

Material Moved to Additional Material

The material for this subsection has been moved to the additional material, which is available from the website.

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

Material Moved to Additional Material

The material for this subsection has been moved to the additional material, which is available from the website.

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

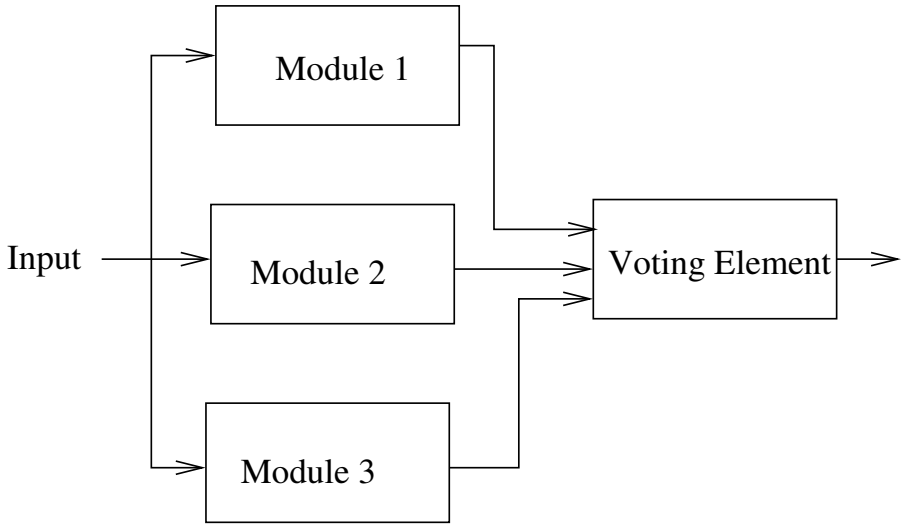
(e) Redundancy

- **Redundancy** is the use of additional elements within a system which would not be required if the system was free of faults.
- Already the early approaches towards fault-tolerant system used duplicated hardware modules.
  - For instance by using the **triple modular redundancy (TMR)** system.

Triple Modular Redundancy (TMR)

- In the TMR system, we have 3 identical hardware modules, and one voting module.
- The voting element will have as output the output of the majority of the modules.
- If one module fails, the majority of the three will be correct.
- If two modules fail, TMR might make a wrong decision.
- **Problem:** faults in the voting element are not tolerated. However, the voting element will usually be much simpler than the modules, and can therefore be designed with a higher degree of reliability.

Triple Modular Redundancy (TMR)



Forms of Redundancy

- **Hardware redundancy**.
  Use of redundant hardware. E.g. the TMR above.
- **Software redundancy**.
  Use of redundant software.
- **Information redundancy**.
  Use of redundant information.
  - E.g. parity bits and other error detecting/correcting codes.
  - E.g. extra data about persons (not only student number but as well name).

Forms of Redundancy

- **Temporal redundancy.**
  - Used in order to tolerate or detect **transient faults**.
  - E.g. repeating calculations and comparison of the results obtained.

Design Diversity

- Problem with TMR above is that
  - it doesn't cover design faults in the modules,
  - that if one module fails, it is likely that identical modules fail at the same time.
- For software faults, doubling the program doesn't help at all.
- Therefore, one aims at combining redundancy with some degree of diversity.
  - E.g. Use of hardware modules from different vendors.
  - E.g. Use of different architectures for different modules.
  - E.g. Writing of software by different groups or even companies, using different programming languages.

Limitations of Design Diversity

- Problem: The same typical software errors, especially (but not only) software specification errors, are made independently by different teams.
  - Has been demonstrated by studies.

6 (a) Introduction

6 (b) Types of Faults

6 (c) Fault Models

6 (d) Fault Coverage

6 (e) Redundancy

6 (f) Fault Detection Techniques

6 (g) Hardware Fault Tolerance

6 (h) Software Fault Tolerance

6 (i) Fault Tolerant Architectures

6 (j) Example: The Space Shuttle

(f) Fault Detection Techniques

- Many fault-tolerance techniques rely on detection of faults.
- In practice often difficult to detect faults – one can often only detect errors created by faults.
- Sometimes it is not even necessary to detect the exact faults – detecting that an error occurred as a consequence, suffices.
  - E.g. in case of transient faults, detection of faults is not necessary. One only needs to know that a fault has occurred, in order to deal with the consequences.
- We consider some software and hardware techniques for fault detection.

Functionality Checking

- **Functionality checking** is the use of software or hardware routines in order to check whether the hardware of a system is functioning correctly.

Examples of Functionality Checking

- Checking of RAM done by writing into memory and reading it back again, and checking whether the value stored is reproduced.
  - **Problem 1**: It is in general not feasible to check the complete memory space.
  - **Problem 2**: Stuck-at faults might be not detected, if the checking test vector checks for the value at which memory is stuck-at (so it always returns that value).
    - Can be overcome by checking with two test vectors, of which the second is the negation of the first one.

Examples of Functionality Checking

- **Problem 3**: Even if a memory location is non-existent, it might be that for a short time due to the capacitance of the bus the test vector is still reproduced.
  - Can be overcome by interleaving writes and reads to different locations.
- Other problems, which can be overcome by using sophisticated test routines.

Examples of Functionality Checking

- Checking of processors done by executing sequences of calculations and comparing them with known results (usually stored in ROM).
- Checking of connections in multiprocessor systems by checking that each processor can communicate with its neighbours.

Misc. Checking Methodologies

- **Consistency checking** uses knowledge about the nature of the information within the system, e.g. that data must be within a certain range (**range checking**).
- **Signal comparison** checks in systems with redundancies the signals at various points in the modules and compares them.
  - **Checking pairs** is a special case of signal comparison. Here one checks whether the outputs of identical modules with the same inputs are identical.
- **Information redundancy** uses error-detecting codes in order to detect errors in the data given.

Instruction Monitoring

- **Instruction monitoring** is checking for illegal instructions.
  - If the binary code is corrupted, one might (especially if the opcode is corrupted) obtain an illegal instruction.
  - Some processors immediately raise an exception, others might proceed.
    - Some processors use illegal instructions for internal testing purposes (this is often not documented), and therefore will not raise an exception.
    - For critical systems one should use processors which raise an exception in such cases.

Loopback Testing

- **Loopback testing** means that one sends a signal arriving at its destination back to its origin and checks whether the original signal and the signal sent back coincide.
  - Finds stuck-at failures in the signal lines.
  - If the outward and return path are too close, then the outward path might influence the return path and transmit its content even so the path is broken.

Watchdog Timers

- **Watchdog timers** are used in order to check whether the processor has crashed.
  - The watchdog timer starts with a certain value and decrements periodically.
  - If the watchdog timer has reached zero, the processor will be reset.
  - The processor regularly resets the watchdog timer, before it reaches 0.
  - If the processor crashes, it will in most cases (but not all) not reset the watchdog timer, and therefore the processor will be reset.

Watchdog Timers

- **Limitations:**
  - It takes a few milliseconds before a crash is detected.
    - During that period the results of the processor are wrong and this might cause hazards.
  - It might be that the processor crashes in such a way that the watchdog timer is reset periodically.

Bus Monitoring

- In **bus monitoring** one checks that the addresses send to the bus are in a certain range.
- Allows to detect, if the processor has crashed and is executing illegal instructions.

Power Supply Monitoring

- **Power supply monitoring** is not directly a fault detection mechanism but a fault prevention mechanism.
  - One checks whether the power is in a certain range, especially above a certain limit (against to high voltage usually overvoltage protection is sufficient).
  - If it is out of range, the processor is instructed to take measurements in order to protect its data (and in order not to produce incorrect output).
- In some cases one needs an **uninterruptible power source** using large-capacity batteries which are all the time reloaded in order to provide power in the event of a power supply failure.

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

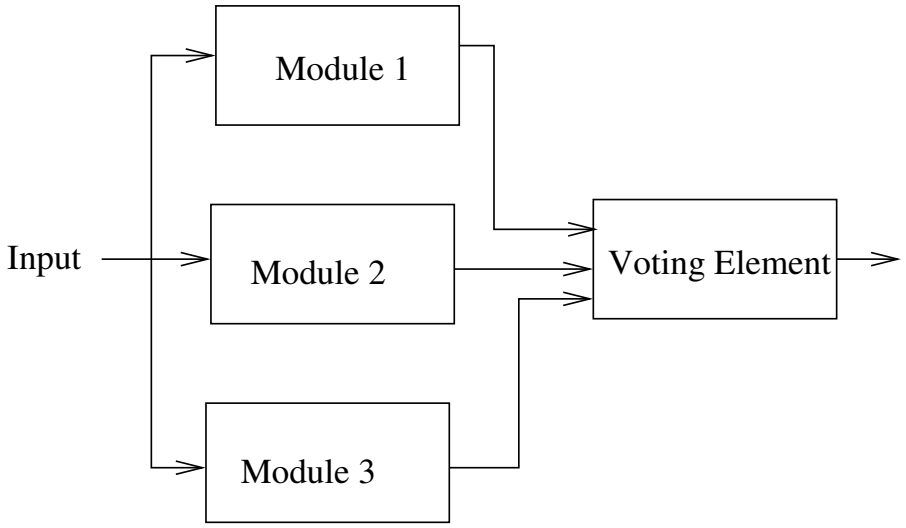
(g) Hardware Fault Tolerance

- There are 3 methods for obtaining hardware fault tolerance:
  - (i) **Static redundancy** uses **fault masking**, which means it hides faults, so that the system works still correctly even if a fault occurs.
  - (ii) **Dynamic redundancy** uses **fault detection**. If a fault occurs, the system reconfigures itself in order to nullify the effects of faults.
  - (iii) **Hybrid approaches** use fault masking in order to prevent errors from propagating through the system, and fault detection in order to reconfigure the systems so that faulty units are removed from the system.

(i) Static Redundancy

- Use of **voting mechanisms** in order to compare the output of modules and mask the effects of faults.
- We have seen already TMR (triple modular redundancy) systems (repeated on next slide)
- TMR prevents single-point failures in the modules.

Triple Modular Redundancy (TMR)

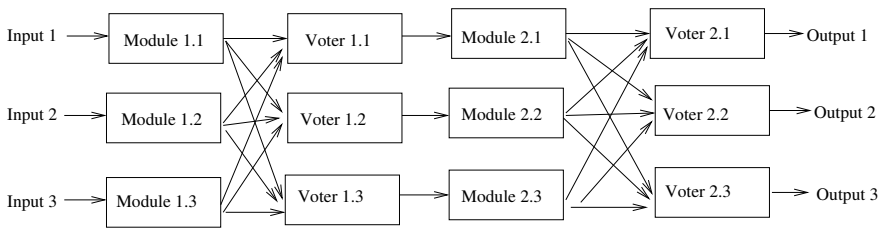


Modular Redundancy

- One problem is that sensors might fail.
  - Therefore one usually adds **static redundancy** to the **sensors** using doubled or tripled sensors and voting on them.
  - Problem is that sensor data are digitised analogue data,
    - which are therefore usually floating point values, which never coincide completely,
    - which are usually taken at slightly different times and physical locations, and are therefore never identical and usually not synchronised.
  - Voting therefore more complicated,
    - and it might be impractical to perform it directly by hardware.

Triplicated Voting

- One problem is the danger of a failure of the voting mechanism itself.
- In order to prevent single point failures in the voting elements, one can **triplicate the voting**, and pass the 3 outputs of the voting elements on to the next modules.
  - See next slide.



Limitations of TMR

- Helps only against random faults, not against systematic faults, which usually affect all modules simultaneously.
- Doesn't help against simultaneous failures of two or more modules.
- Necessary to add monitoring of discrepancies in the voting, in order to detect failures and be able to remove them during maintenance.

N-Modular Redundancy

- **N-Modular Redundancy** (**NMR**) uses N instead of 3 modules and voting among those.
  - The system will be able to tolerate the failure of $\frac{N-1}{2}$ modules without producing a system failure.
  - **Disadvantage:** Additional cost, size, weight and power consumption.
  - In practice the number of identical modules is **rarely greater than 4**

Implementation of the Voter

- Implementation of the voting element by **hardware**:
  - If one has 3 inputs i1, i2 i3, the best out of the three is obtained by the Boolean formula

$$(\mathrm{i1} \wedge \mathrm{i2}) \vee (\mathrm{i1} \wedge \mathrm{i3}) \vee (\mathrm{i2} \wedge \mathrm{i3})$$

  - This can be implemented by a very simple circuit.
  - Therefore it can be designed in a highly reliable way.
  - **Problems**
    - This arrangement doesn't provide the possibility for monitoring of discrepancies.
    - One often has a large amount of data to compare (not only one bit), therefore the voting elements can become still complicated.

Implementation of the Voter

- Implementation of the voting element by **software**.
  - **Advantages:**
    - More complex voting possible.
    - Monitoring of discrepancies easy.
  - **Disadvantages:**
    - Much longer response times than hardware voting (which reacts with almost no delay).
      Particular problem since safety critical systems are often real time systems (aerospace!!).
    - Higher complexity of the underlying computers and the software results in lower reliability.

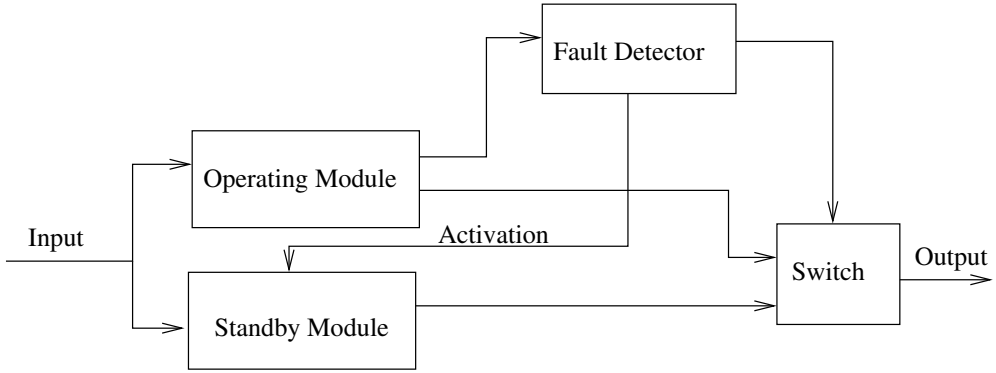
(ii) Dynamic Redundancy

- In dynamic redundancy one uses one unit, which is normally in use, and one or more **standby systems**, which are available, if the main unit fails.
  - Requires less units than static redundancy (where all units have at least to be tripled).
  - Necessary to have good fault detection mechanisms, as discussed in Subsect. (f).

Standby Spare Arrangement

- In a **standby spare arrangement**, one module is operated with some fault detection mechanism (as discussed in Subsect. (f)).
- Another module is on standby.
- Unless a fault is detected, the output of the first module is taken as output of the system.
- In case a fault is detected,
  - the standby module is activated,
  - the faulty module is deactivated,
  - and the output is taken from the standby module.

Standby Spare Arrangement



Standby Spare Arrangement

- The standby module can be
  - on **cold standby**, which means it is switched off.
  - **Disadvantage:**
    - In case of a fault the disruption is longer, since it takes a while before the standby module is activated.
    - Fault detection mechanism cannot make use of the data processed by the standby module.

Standby Spare Arrangement

- The standby module can be
  - on **hot standby**
  - Means that the standby module is effectively processing the input, but the output is dismissed.
  - **Disadvantage:**
    - More power consumption.
    - The standby unit is subject to the same operating stress as main module, therefore the likelihood of simultaneous failures of the main and the standby module is higher.

Multiple standby modules

- One can extend the above by having more than one standby module.

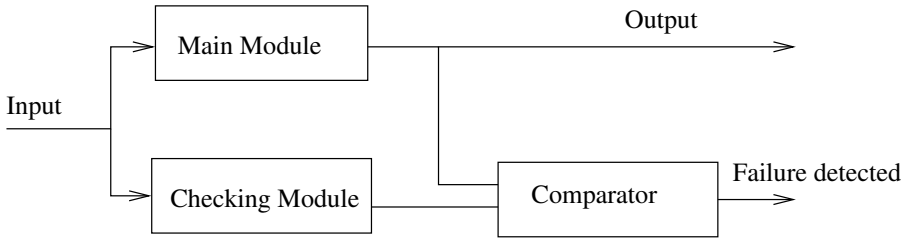
Self-Checking Pairs

- A **self-checking pairs** arrangement consists of
  - one main module,
  - one checking module,
    - both of which receive the same input,
  - a comparator, which checks, whether the output of the main module coincides with the output of the checking module.
- The output of the main module (**not** of the checking module) and the result of the comparison are passed onwards.

Self-Checking Pairs

- It is not part of the self-checking pairs arrangements to reconfigure itself or mask errors in case the output of the two modules doesn't coincide.
  - Therefore the arrangement itself does not provide fault tolerance, but only error detection.
  - However, the output of the comparison can be used for instance in dynamic fault-tolerant systems in order to carry out reconfiguration in a standby spare arrangement.

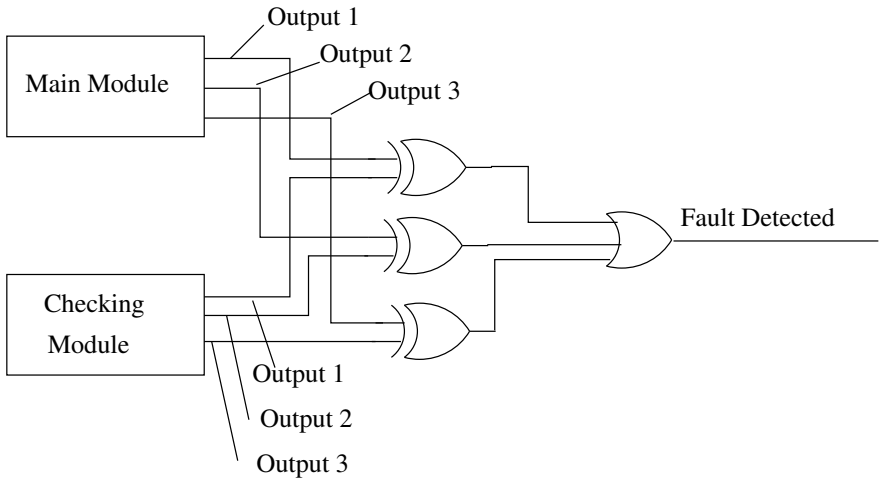
Self-Checking Pairs



Implementation of the Comparator

- The comparator can be implemented by **hardware**.
  - One can take the disjunction of the XOR of all the one-bit signal lines from both modules (see next slide).
    - Note that the XOR of two signals is 1 iff the two signals don't coincide.
    - The disjunction of the XORs is therefore true, iff there was at least one disagreement between the bits of the signal, i.e. if there was a fault.

Comparator (Hardware Implementation)



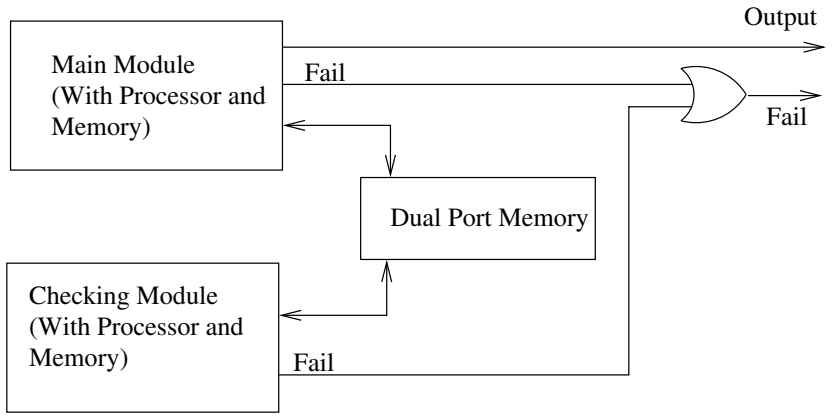
Comparator (Hardware Implementation)

- The advantages/disadvantages of the hardware implementation are similar to those of the TMR.
- In order to cover against a single-point failure in the comparator, one can duplicate the comparator and take the disjunction of the results of all the comparators.

Implementation of the Comparator

- The comparator can as well be implemented by **software**, in case the modules include a processor.
  - Then one can add a dual port memory, in which the output of both modules is written.
  - Then both processors compare after they have computed their results these results with the result obtained by the other processor.
    - Should be done by both processors in order to protect against a single-point failures during the comparison phase.
  - If there is a discrepancy, then a fail signal is output on a special line.
  - The disjunction of the two fail signals indicates that a failure has been detected.

Comparator (Software Implementation)



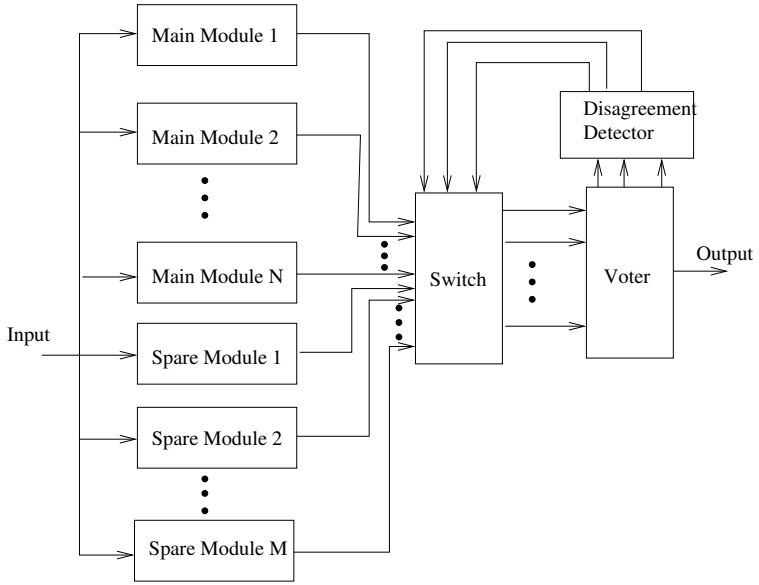
(iii) Hybrid Redundancy

- Use of a combination of voting, fault detection and module switching.

N-Modular Redundancy with Spares

- Use of N modules plus M spares connected to a voter.
  - Initially N modules take input in parallel, and their results are compared.
  - In case there is no disagreements, the result is passed on as output.
  - In case of a disagreement,
    - the output given by the majority of the active modules is passed on as output,
    - the faulty module is removed,
    - one of the spare modules is activated,
    - and afterwards the system continues using the N main modules (of which one is the spare one) and M-1 spares.

N-Modular Redundancy with Spares



N-Modular Redundancy with Spares

- System tolerates up to $\frac{N-1}{2}$ simultaneous faults in the main modules, and can compensate up to M faults by using the spare modules.
- **Analysis:**
- **Problem:**
  Doesn't tolerate single-point failures in the switch, the voter and the disagreement detector.
- **Advantage:**
  A good compromise between the advantages of
  - static redundancy
    - immediate fault masking
  - and dynamic redundancy
    - removal of faulty modules,
    - monitoring of faults in the system.

Module Synchronisation

- In all the above techniques one needs to compare the outputs of different modules.
- **Problem:** If the modules don't share the same clock their output will not be synchronised.
  - One solution is that all processors share the same clock.
    - Then the modules are said to be in **lock step**.
    - Problem: Single-point failures in the clock are not tolerated and not detected.
  - Otherwise one needs to construct the voting and fault detection mechanisms in such a way that problems with synchronisation are taken into account.
    - Easier, if this is done by software.

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

(h) Software Fault Tolerance

- In this subsection we will consider how to tolerate faults by using software.
- We will consider two techniques:
  - (i) **N-version programming.**
  - (ii) **Recovery blocks.**

(i) N-Version Programming

- N-version programming means that N different versions of the software are written.
  - All fulfil the same specification.
    - Note that this doesn't help if there is a problem in the specification.
  - Written by different teams or companies, and maybe using different tools, languages or techniques.
- The N versions are run on the same input data
  - on the same processor (interleaved)
  - or on separate processors (operating in parallel).
- The results are compared using a software implementation of one of the techniques introduced in the section on hardware redundancy.

N-Version Programming

- **Problem** of N-version programming:
  - Additional development costs.
  - Additional processing power needed for running several versions of the program and for the voting process.
- N-version programming only used for very critical applications, e.g.
  - Airbus 330/340,
  - Space shuttle.

(ii) Recovery Blocks

- **Recovery block technique** based on **acceptance tests**.
  - Acceptance tests are software versions of fault detection.
  - Acceptance test check the consistency of the output of one software module (a unit inside the program like a procedure, package or a class).

Examples of Acceptance Tests

- Check whether output is within certain boundaries.
- Check for run time errors (e.g. arithmetic over- or underflow),
- Check for excessive execution time.
- Check for arithmetic correctness by reversing the computation.
  - E.g., if a module computes the square root, the acceptance test checks whether the square of the output is equal to the input.

Recovery Blocks (Cont.)

- For critical modules, several implementations are developed, of which one is the main one.
- Then the following is executed:
  - Execute the main module.
  - Carry out acceptance test.
  - If this fails, switch to an alternative module.
  - Carry out acceptance test.
  - If this fails, switch to the second alternative module.
  - Etc.
  - If everything fails, raise an error.
- Recovery blocks can be considered as a software version of the standby spare arrangement.

Recovery Blocks

- Problem: if one module fails, one has to make sure that any side-effects carried out by it are reversed.
- Done by establishing a **recovery point** at the beginning of the module, and by introducing a mechanism in order to make sure that one can switch back to the recovery point in case of failure of the acceptance test.
- Storing the state of all variables when reaching the recovery point too expensive and time consuming in general.
- Instead one stores only those variables which are possibly changed by the module.

Recovery Blocks

- It was suggested to provide a special language construct as follows:

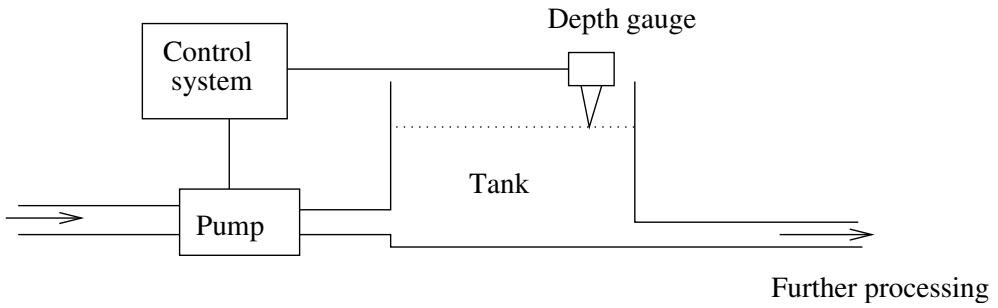
ensure <acceptance test>
by <main module>
else by <alternative module 1>
else by <alternative module 2>
. . .
else by <alternative module n>
else **error**

6 (a) Introduction
6 (b) Types of Faults
6 (c) Fault Models
6 (d) Fault Coverage
6 (e) Redundancy
6 (f) Fault Detection Techniques
6 (g) Hardware Fault Tolerance
6 (h) Software Fault Tolerance
6 (i) Fault Tolerant Architectures
6 (j) Example: The Space Shuttle

(i) Fault-Tolerant Architectures

- The use of **Fault-tolerant architectures** means that we use for the critical parts non-computer-based mechanisms as additional safe guards.
- Example on next slide:
  - Use of a computerised control method in order to control the pump which feeds toxic liquid into a tank.
  - Danger is that the tank overflows and the toxic liquid is spilled.
  - The computerised control system is safety critical, and needs to be implemented with highest standards.
    - Very expensive to implement this.

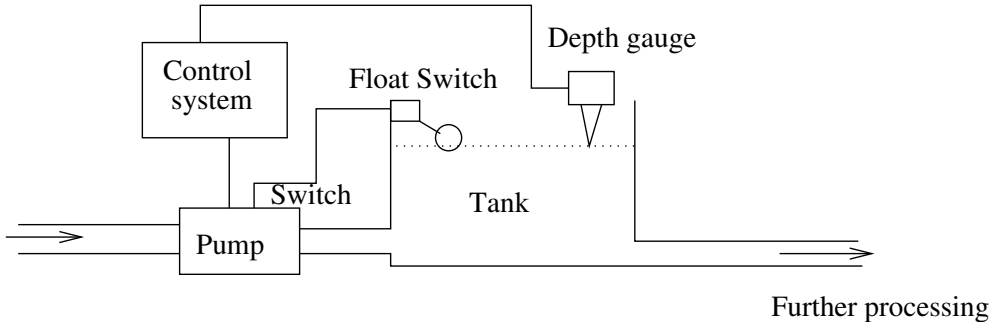
Fully Computerised Solution



Better Solution

- A better solution is to
  - keep the computerised control system,
  - but add an additional non-computerised float switch, which directly switches off the pump in case the tank is full.

Better Solution



Better Solution

- Result is that
  - there are two independent systems which control the tank,
  - the non-computerised float-switch can much more easily be designed to meet high reliability standards.
- Often one can, by using simple (usually non-computerised) safeguards, obtain a much higher degree of safety than using complex computerised solutions.
  - This reduces as well the cost.
- The example demonstrates that one can combine a complex computer system with a relatively simple safety control system.

Better Solution

- One can even achieve with very little extra cost an even higher degree of safety by adding two or more float switches, which independently switch of the pump.

6 (a) Introduction

6 (b) Types of Faults

6 (c) Fault Models

6 (d) Fault Coverage

6 (e) Redundancy

6 (f) Fault Detection Techniques

6 (g) Hardware Fault Tolerance

6 (h) Software Fault Tolerance

6 (i) Fault Tolerant Architectures

6 (j) Example: The Space Shuttle

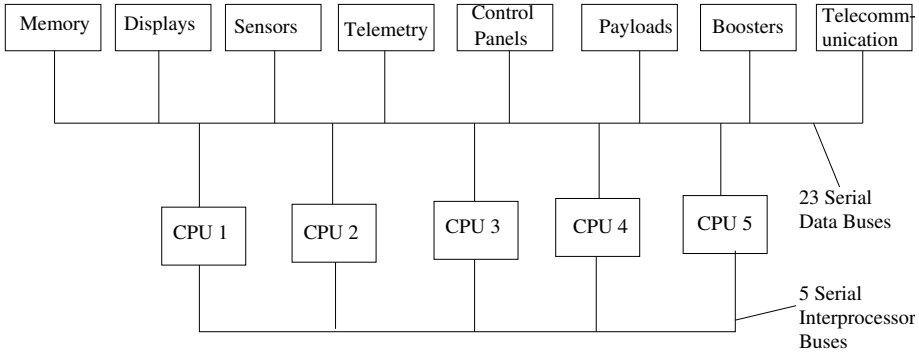
(j) Example: The Space Shuttle

- At certain stages of the flight many flight-critical functions of the space shuttle are totally dependent on its on-board computers.
- On this rely
  - the lives of the crew,
  - the vehicle, which costs several billion dollars,
  - the national prestige of U.S.
- Therefore the application has been developed up to highest standards w.r.t. reliability, integrity, availability and fault tolerance.
- A combination of redundancy, hardware and software voting, fault masking, fault tolerance and design diversity was used in order to achieve this.

Architecture of the Comp. System

- The computer system consists of 5 identical computers.
- Conventional processors are used.
- They are connected by using an array of serial buses.
  - 5 buses provide communication between the computers.
  - 23 buses link the computers to other subsystems.

Architecture of the Comp. System



Explanation of Terminology

- **Payload** seems to mean control of the load the space shuttle is carrying.
- **Boosters** = auxiliary rocket or engine for additional speed.
- **Telemetry** = Process of recording the reading of instruments and transmitting them by radio.

Architecture of the Comp. System

- Most fault detection is performed using software rather than hardware techniques.
- The configuration of the computers is controlled by software.
- During critical phases, 4 of the 5 computers are configured in a **4-modular redundancy (NMR)** arrangement.

Architecture of the Comput. System

- The 4 computers get similar input data and compute the same functions.
- Hardware voting is preformed by the actuators in order to provide fault masking.
- Additionally, each processor
  - has extensive self-test facilities.
    - If an error is detected, it is reported to the crew, which then can switch off the faulty unit.
  - compares its results with those produced by its neighbours.
    - If a processor detects a disagreement, it signals this, and voting is used in order to remove the offending computer.
  - has a watchdog timer, in order to detect crashes.

Architecture of the Comput. System

- The 5th computer normally performs non-critical functions (e.g. communications).
- It contains additionally a diverse implementation of the (critical) flight control software, produced by a different contractor, which can be used in an emergency (see below).

Architecture of the Comput. System

- If one processor is switched off, one obtains a triple modular redundancy arrangement (TMR).
- If a second processor is switched off, the system is switched into duplex mode, where the two computers compare their results in order to detect any further failure.
- In case of a third failure, the system reports the inconsistencies to the crew and uses fault detection techniques in order to identify the offending unit.
  - This provides therefore protection against failures of two units and fault detection and limited fault tolerance against the failure of a third unit.

Architecture of the Comput. System

- In an emergency, the fifth computer can take over critical functions
  - The 5th computer allows protection against systematic faults in the software.
- If one or two computers fail, the crew or the controllers on earth might decide to abort the mission.

Analysis

- The arrangement provides **excellent fault tolerance**, and **protection against systematic faults**.
- **Problems:**
  - **Heavy dependency on software** for voting, fault detection, configuration control.
    - Therefore heavy dependency on the **correct design of this software**.
  - The 5 computers are identical, therefore **no protection against systematic hardware faults** affecting all 5 computers simultaneously.
- **In total** the design is still very good.
  Due to the high costs this degree of redundancy can only be obtained for very critical applications.