EDA122/DIT061 Fault-Tolerant Computer Systems
DAT270 Dependable Computer Systems

Welcome to Lecture 3

Hardware redundancy

Outline

- More on failure mode assumptions
- Hardware redundancy principles:
  - Voting redundancy
  - Standby redundancy
  - Active redundancy
- System example:
  - Hewlett Packards's NonStop Computers

Terminology

Fault - Cause of an error, e.g., an open circuit, a software bug, or an external disturbance.

↓

Error - Part of the system state which is liable to lead to failure, e.g., a wrong value in a program variable.

↓

Failure - Delivered service does not comply with the specification, e.g., a cruise control in a car locks at full speed.

Failure modes

- A failure mode describes the nature of a failure, i.e., the way in which a ***service provider*** (a system, subsystem, or module) can fail
- A ***service provider*** can have many failure modes
- Examples of failure modes:
  - ***Value failure*** – a service provider delivers an erroneous result
  - ***Timing failure*** – a service provider delivers a result too late, or too early
  - ***Silent failure*** – a service provider delivers no result
  - ***Signaled failure*** – a service provider sends a failure signal
- A service provider must have internal mechanisms for error detection to enforce silent or signaled failures

Note: A v*alue failure* is the same as a *content failure*. Both terms are used in the literature.

Failure model vs. Failure mode

- A ***failure model*** is a set of assumptions about likely *failure modes* for a service provider
- A ***failure mode*** describes the nature of a given class of failures

Two types of value failures

- **Detectable value failures**: the service user(s) can detect the failure
- **Non-detectable value failure**: the service user(s) cannot detect the failure

Example: Consider a service provider whose outputs are protected by a checksum

Detectable value failure: the output from the service provider has an invalid checksum => a service user can detect the failure by inspecting the checksum

Non-detectable value failure: the value of the output is wrong but the output has a valid checksum => failure cannot be detected by inspecting the checksum

Persistence of faults

- ***Permanent fault***
  - The fault is always *active*, i.e., it generates errors whenever the faulty component (for example a transistor) is used for storing or processing information.
  - Examples: (i) a bug in software, (ii) a permanently open circuit in a hardware component.
- ***Intermittent fault***
  - The fault switches between an *active state* and a *passive state*. It generates no errors when it is in the passive state.
  - Example: bad contact that works on and off.
- ***Transient fault***
  - A one time event that generates an error
  - Example: a bit-flip in a flip-flop or memory cell within an integrated circuit caused by a strike of a high energy neutron

Particle radiation-induced faults

- Ionizing particles, such as *alpha particles* and *high energy cosmic neutrons* can cause bit errors in binary information stored in integrated circuits.
- Such errors are known as *soft errors*, or *single event upsets* (SEU:s).
- Soft errors can occur in SRAM cells, DRAM cells, Flip-flops, etc.
- A strike by a single particle can cause a single bit upset or multiple bit upsets.
- Single bit upsets are more likely to occur than multiple bit upsets in current technologies, but multiple bit upstes are becoming increasingly frequent as technology scales.

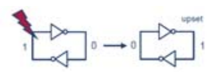
Particle radiation-induced faults

Soft errors vs. Hard errors

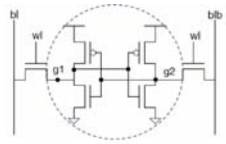
- The term *soft error* is used for events where a particle strike alters the binary information in a circuit without causing permanent damage to the circuit.
- A soft error can be recovered from by reloading the correct bit value(s) into the affected memory element.

- The term *hard error* is used for errors caused by permanent hardware faults.
- Strikes by ionizing particles can cause permanent damage to an integrated circuit, but such events are very rare on Earth
- Particle-induced hard errors is a concern for space applications, where circuits are exposed to protons and heavy-ions.

Soft Errors

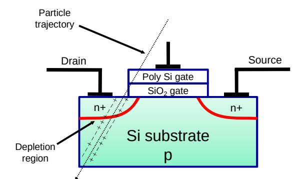


Bit-flips SRAM cell



6T bit cell

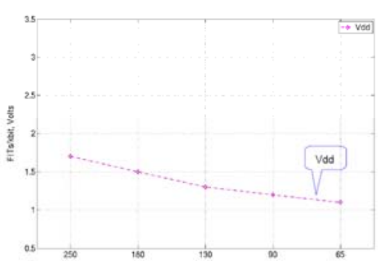
- Soft errors (or single event upsets) are particle induced upsets (bit-flips)
- Caused by highly energetic particles such as neutron, protons and muons



Particle strike in n-channel MOSFET transistor

SER trend for SRAM & Flip-Flops

(SER = Soft Error Rate)

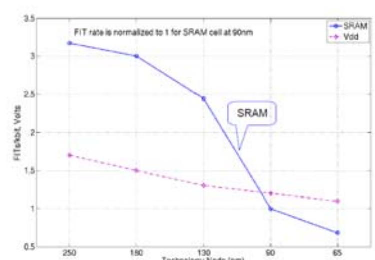


1 FIT = 10^{-9} faults per hour

Source: A. Dixit, R. Heald, and A. Wood, "Trends from Ten Years of Soft Error Experimentation, SELSE´09, Stanford, CA, USA.

SER trend for SRAM & Flip-Flops

(SER = Soft Error Rate

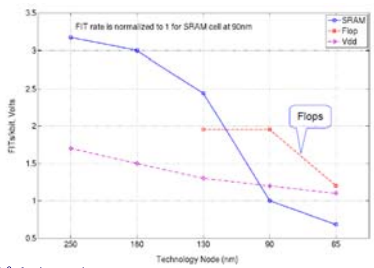


1 FIT = 10^{-9} faults per hour

Source: A. Dixit, R. Heald, and A. Wood, "Trends from Ten Years of Soft Error Experimentation, SELSE´09, Stanford, CA, USA.

SER trend for SRAM & Flip-Flops

(SER = Soft Error Rate)



1 FIT = 10^{-9} faults per hour

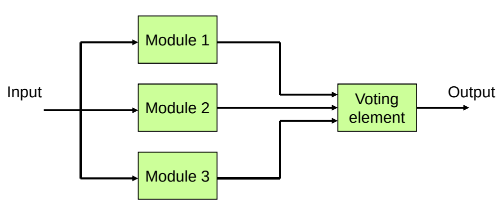
Source: A. Dixit, R. Heald, and A. Wood, "Trends from Ten Years of Soft Error Experimentation, SELSE´09, Stanford, CA, USA.

Hardware Redundancy Principles

- Voting redundancy
- Standby redundancy
- Active redundancy

Voting redundancy

Triple Modular Redundancy (TMR)



Voting Redundancy

- Three or more units are active and produce replicated outputs simultaneously
- Majority voting is used to mask errors in module outputs
- **Failure mode assumption for modules:** *non-detectable value failures* $\Rightarrow$ modules are not required to have internal error detection and failure signaling
- Can also cope with *detectable value failures*, *signaled failures* and *silent failures*.
- Requires *2f +1* units to tolerate *f* faulty units

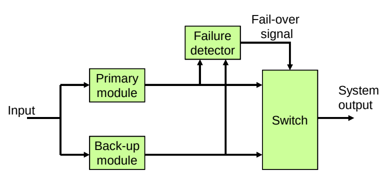
Hardware Redundancy Principles

- Voting redundancy
- Standby redundancy
- Active redundancy

Standby Redundancy

- One active (primary) module together with one or several back-up (spare) modules.
- Relies on failure detection and system reconfiguration
- Switching to a back-up module is called a *fail-over*
- **Failure modes assumptions**: *silent failures*, *signaled failures*, or *detectable value failures*
- Requires *f +1* units to tolerate *f* faulty units

Standby redundancy



Module 1 and 2 perform the same computations.

Module 1 delivers system output when the system is started.

The failure detector checks the output of the primary module and issues a fail-over command when it detects a failure in the output of the primary module.

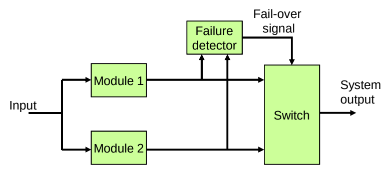
Failure mode assumptions for main modules: silent failures, signaled failures, or *detectable* value failures

Classification of Standby Systems

- Hot standby redundancy
- Warm standby redundancy
- Cold standby redundancy

Standby redundancy

Hot standby system



Module 1 and 2 perform the same computations.

Module 1 delivers system output when the system is started.

The failure detector issues a *fail-over* signal when it detects a failure in the output of Module 1.

Hot Standby Redundancy

- Characteristics
  - Back-up module updated simultaneously with primary module
  - Example: back-up module executes the same program as the primary module
- \+ Advantages
  - \+ Very short, or no outage time in conjunction with fail-over
  - \+ Back-up module does not need to load application state on fail-over
- \- Drawbacks
  - \- Back-up module cannot do other useful work
  - \- High failure rate
  - \- High power consumption

Warm Standby Redundancy

- Characteristics
  - Back-up module is powered-up
  - Primary module stores ”checkpoints” of the application state in a “save place” :
    - – Checkpoints are sent to the back-up module, **or**
    - – Checkpoints are stored in “crash-proof memory” (a.k.a. stable storage).
  - Back-up module loads the most recent “checkpoint” on fail-over.
- \+ Advantages
  - \+ Back-up module can perform other useful work during fault-free conditions.
- \- Drawbacks
  - \- Significant outage time during fail-over since the back-up module needs to load application state
  - \- High failure rate
  - \- High power consumption

Cold Standby Redundancy

- Characteristics
  - Back-up powered-down during fault-free operation
  - Application state saved in crash-proof memory (a.k.a. stable storage)
  - Common in space applications, especially deep space probes
- \+ Advantages
  - \+ Low failure rate
  - \+ Low power consumption
- \- Drawbacks
  - \- Long outage time at fail-over: back-up module needs to boot kernel/operating system and load application status

Hardware Redundancy Principles

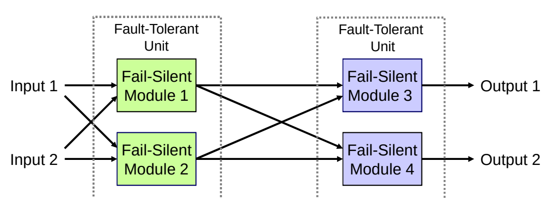
- Voting redundancy
- Standby redundancy
- Active redundancy

Active Redundancy

- Two or more modules are active and produce replicated results.
- **Failure mode assumptions**:
  - *silent failures:* a faulty module produces no result
  - *signaled failure*: a faulty modules sends a failure signal
  - *detectable value failures*: erroneous results can be detected by service user.
- Requires *f +1* units to tolerate *f* faulty units

Active Redundancy

Pairs of fail-silent modules



Fault Tolerant Units formed by pairs of *fail-silent* modules

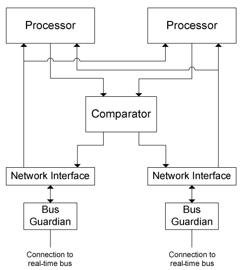
Fail-silent property: *a unit produces correct results or no results at all*

Error Detection Techniques

Two examples:

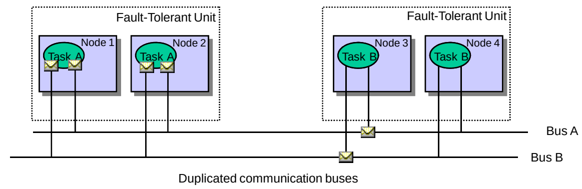
- Duplication and comparison
  - Two modules produce replicated results
  - Errors are detected by comparing the results
  - Ensures fail-silence
- End-to-end checksums
  - The service provider adds a checksum to its outputs
  - Checksums are checked by the service user
  - Ensures detectability of value failures

HW architecture for fail-silent node in a distributed system



- Processor failures are detected by ***duplication and comparison***
- The processors produce replicated messages that are compared by the comparator.
- The network interfaces receive messages from the comparator and send them to other nodes via two redundant real-time busses.
- The payload in the messages are protected by end-to-end checksums added by the processors.
- The *end-to-end* checksums ensures that faults in the comparator and network interfaces are detectable by the service users (other nodes).

Active redundancy in a distributed real-time system



Classification of Hardware Redundancy

- **Static Redundancy -** Does *not* require reconfiguration
  - Voting redundancy (requires *2f+1* units to tolerate *f* faulty units)
  - Active redundancy (requires *f+1* units to tolerate *f* faulty units)
- **Dynamic Redundancy -** Requires reconfiguration
  - Stand-by system (requires *f+1* units to tolerate *f* faulty units)
- **Hybrid Redundancy**
  - Combination of static and dynamic redundancy

HP's NonStop Computer Systems

- Highly available computers for on-line transaction processing (OLTP) systems
- Typical applications:
  - Automatic teller machines, Stock trading, Funds transfer, 911 emergency centers, Medical records, Travel and hotel reservations, etc
- Availability: 0,99999 – "five nines", or 5 min downtime per year
- Data integrity: 1 FIT = 10^{-9} undetected errors per hour (one undetected data error per billion hours)

Marketing information from HP
(from 2005)

- Telecommunications
  - 135 public telephone companies currently rely on NonStop technology.
  - More than half of all 911 calls in the United States and the majority of wireless calls worldwide depend on NonStop servers.
- Finance
  - Eighty percent of all ATM transactions worldwide and 66 percent of all point-of-sale transactions worldwide are handled by NonStop servers.
  - NonStop technology powers 75 percent of the world's 100 largest electronic funds transfer networks and 106 of the world's 120 stock and commodity exchanges.

NonStop System with self-checked processors

Self-checked processors
- Stop promptly if an error occurs
- Prevent error propagation

Process pairs
- Critical software is implemented as a process pair, with one primary and one backup process executing on different processors
- The primary process execute the program and sends state changes regularly to the backup process.
- Backup process takes over if the primary process fails by itself or as a result of a processor failure.

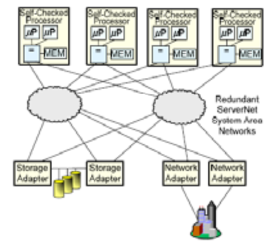
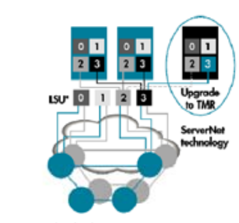


Figure 1: 4 Processor NonStop System with Duplicated and Compared Microprocessors

Evolution of self-checked processors

- Self-checking processors (mid 1970's to mid 1980's)
  - Custom designed processors
  - Self-checking logic circuits
  - Memories protected by error correcting codes
- Lock-stepped microprocessors (mid 1980's to late1990's)
  - Two microprocessors run synchronously using the same clock and their write operations to the main memory are compared
    - A mismatch between memory writes stops the processor
  - Main memory protected by error correcting codes
    - Non-correctable memory errors stops the processor
- Loosely synchronized processors (late 1990's – now)
  - Comparison of I/O-operation (e.g. disk read/write)
  - Dual or triple modular redundancy

NonStop Advanced Architecture
(released 2005)



- Four Intel Itanium processors per board
- Four logical processors
- The output from the LSU represents a logical processor
- Each logical processor consists of two processors (dual modular redundancy) or three processors (triple modular redundancy)
- TMR allows hardware faults to be masked without fail-over
- Fail-over is performed if a logical processor fails

Logical Processors

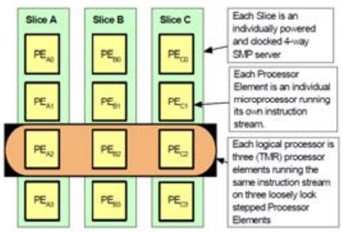


Figure 2: Four Logical NSK Processors built from TMR 4-way SMP servers.

NonStop Advanced Architecture

- Dual or Triple Modular Redundant Servers
- Built from standard 4-way SMP processor modules
- Logical processors are formed by two or three processing elements located in different processor modules
- The processors that comprise a logical processor are loosely synchronized by one or two Logical Synchronization Unit (LSU), which also perform voting.
- Critical processes (tasks) can be moved from one logical processor to another logical processor (fail-over)
- Redundant ServerNet SANs (System Area Networks) connect processor modules and I/O devices
- Logical disk volumes are implemented by a pair of mirrored disks

The Itanium Processor

Itanium 2 processor from Intel Corp.

For more info see: http://en.wikipedia.org/wiki/Intel_Itanium

Principles of Fault Tolerance (1)

Fault/Error Containment

- Each processor (slice) has its own memory
- Processors communicate via messages passed over the redundant System Area Network (SAN)

No single point of failure

- All system components (processors, I/O adapters, I/O devices, and the System Area Network) are replicated.

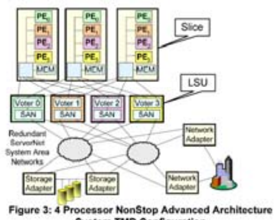


Figure 3: 4 Processor NonStop Advanced Architecture System TMR Configuration

Principles of Fault Tolerance (2)

Failure mode assumptions

- All system components are self-checking and *fail-fast*
  - Simple errors (e.g., memory access errors or I/O timeouts) can be corrected, or masked, by retry.
  - For other errors, components are *fail-fast*: the components stops when an error is detected (a.k.a. fail silent)

Error detection

- Voting or comparison in the LSU
- ServerNet messages protected by CRC-checksums
- Disk data protected by end-to-end checksum

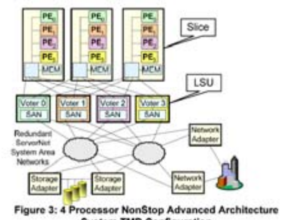


Figure 3: 4 Processor NonStop Advanced Architecture System TMR Configuration

Summary

- Failure mode assumptions
- Soft errors
- Voting redundancy
- Standby redundancy
  - Hot, Warm and Cold
- Active redundancy
  - Two or more active fail-silent modules
- System examples
  - HP NonStop System

Overview of Lecture 4

- Markov chain models
  - Hot standby system
  - Cold standby system
  - Coverage factor
  - Dormancy factor
- Preparations:
  - Storey: Section 7.2 Markov models (pages 183 - 186)
  - Lecture slides