King Fahd University of Petroleum & Minerals
College of Computer Sciences and Engineering
Department of Computer Engineering

CSE 641 Reliability and Fault Tolerance of Computer Systems (3-0-3)

Professor Mostafa Abd-El-Barr

mostafa@ccse.kfupm.edu.sa
www.kfupm.ccse.edu.sa/~mostafa

Term 022 (February – June 2003)

Module # 1

Introduction to the Design and Analysis of Fault Tolerant Systems

Outline

I. Fundamental Concepts

a. Mathematical Basis
b. Basic Terminology

II. Fault Tolerant Systems

a. Measures
b. Attributes
c. Lifecycle

III. Redundancy Techniques

a. Hardware
b. Information
c. Software
d. Time

I. Fundamental Concepts

(A) Mathematical Basis

- The number of ways in which unordered subset j things can be taken from N things is given by

$$\binom{N}{j} = \frac{N!}{j!*(N-j)!}$$

- The joint event of A and B occurring, P(AB), is given by

$$P(AB) = P(A) * P(B \mid A) = P(B) * P(A \mid B)$$

Where P(B|A) is the conditional probability that event B occurs given that event A has occurred & P(A|B) is the conditional probability that event A occurs given that event B has occurred.

$$P(B \mid A) = \frac{P(B) * P(A \mid B)}{P(A)}$$

$$P(A \mid B) = \frac{P(A) * P(B \mid A)}{P(B)}$$

- If events A and B are independent Then P(A|B) = P(A), P(B|A) = P(B), and P(AB) = P(A)*P(B).

- A generalized form – Conditional Probabilities (Baye's Rule)

Given a probability space (Ω, A, P) and $\{B_n\} \subseteq A$ a disjunct partition of Ω, i.e. $\Omega = \bigcup_{n=1}^{\infty} B_n$ with $B_i \cap B_j = \phi \ \ \forall i \neq j$ then

$$P(B_i \mid A) = \frac{P(A \mid B_i) P(B_i)}{\sum_{n=1}^{\infty} P(A \mid B_n) P(B_n)}$$

- The probability of k successes in n independent trials of an experiment that has a probability p of success on each trial is given by (binomial Distribution)

$$b(k,n,p)=\binom{n}{k}\bullet p^k \bullet (1-p)^{n-k}$$

- A random variable, x, can take on values $x_1, x_2, ..., x_n$.
- The probability of a x taking a value x_i is given by $P_x(x_i)$.
- If the r.v. is discrete then $\sum_i P_x(x_i)=1$
- The mean value (also called the average value, or expected value) of a r.v. x is defined as $\bar{x}=E[x]=\sum_i x_i P_x(x_i)$
- If the variable is continuous then $\bar{x}=E[x]=\int_{-\infty}^{\infty} x p_x(x)dx$.

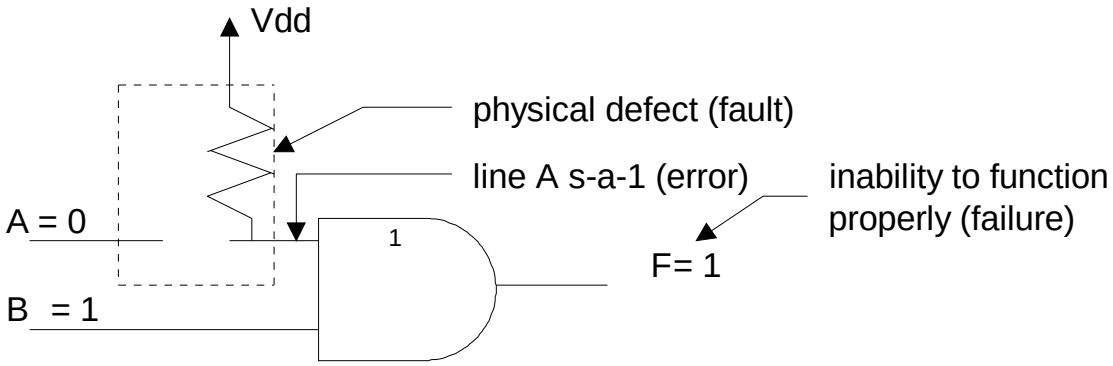
(B) Basic Terminology

A *fault* is a physical defect in some part(s) of a system.

An *error* is the manifestation of a fault.

A *failure* is an active encounter with a fault.

Example



II. Fault Tolerant Systems

a. Definition

A fault tolerant (FT) system is one that can continue its specified task in the presence of hardware and/or software faults.

b. Fault Tolerant System Measures

1. **Dependability:** A measure of reliance on the delivered service

2. **Reliability:** A measure service continuity

3. **Availability:** A measure of service delivery at an instant

4. **Safety:** A measure of non-occurrence of catastrophic failures

5. **Security:** A measure of tolerance to deliberate faults

6. **Perform-ability:** A measure of the level of service at time *t*

7. **Testability:** A measure of the ease of testing

8. **Maintainability:** A measure of the ease of repair

c. Some Definitions

1. Reliability *R(t)*

The conditional probability that service has been delivered properly during a time interval [0, t], given that it was functioning at time 0.

2. Availability *A(t)*

The probability that service has been delivered properly at instant *t*

3. Mean time to failure *(MTTF)*

The expected time elapsed before the system fails (expected system lifetime)

4. Mean time to repair *(MTTR)*

The expected time elapsed before the system is repaired after it fails

5. Reliability Improvement Factor *(RIF)*

Ratio between MTTF with FT to MTTF without FT

d. Fault Tolerant System Attributes

1. Fault Detection
2. Fault Localization
3. Reconfiguration
4. Recovery & Restart
5. Repair

e. Fault Tolerance & Lifecycle

- Specifications & Design Phase
    1. Sources: Ambiguous Specifications, faulty algorithms
    2. Detection: Audits, Model Checking, Simulation

- Prototyping Phase
    1. Sources: Assembly faults, defective components, timing
    2. Detection: Structured Testing

- Production & Installation Phase
    1. Sources: Wiring faults, defective components
    2. Detection: System Testing, Self-Diagnosis, BIST

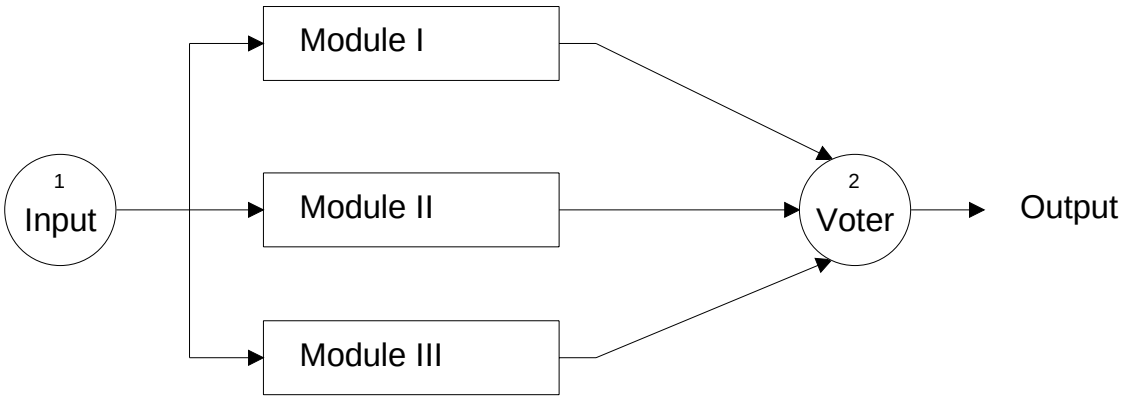
- Utilization/Operational Phase
    1. Sources: Components aging, misuse, external faults
    2. Detection: Diagnosis, testing, Self-checking

III. Redundancy Techniques

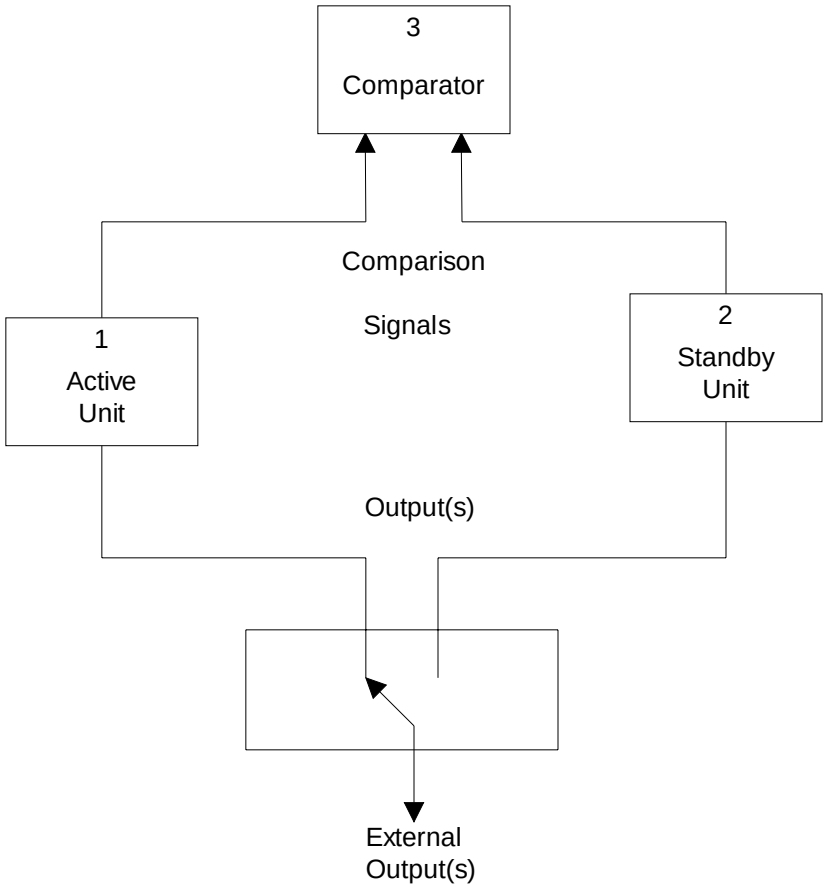
(A) Hardware Redundancy (HWR)

1. Passive HWR: *N*-Modular Redundancy (NMR)

Example: *TMR*

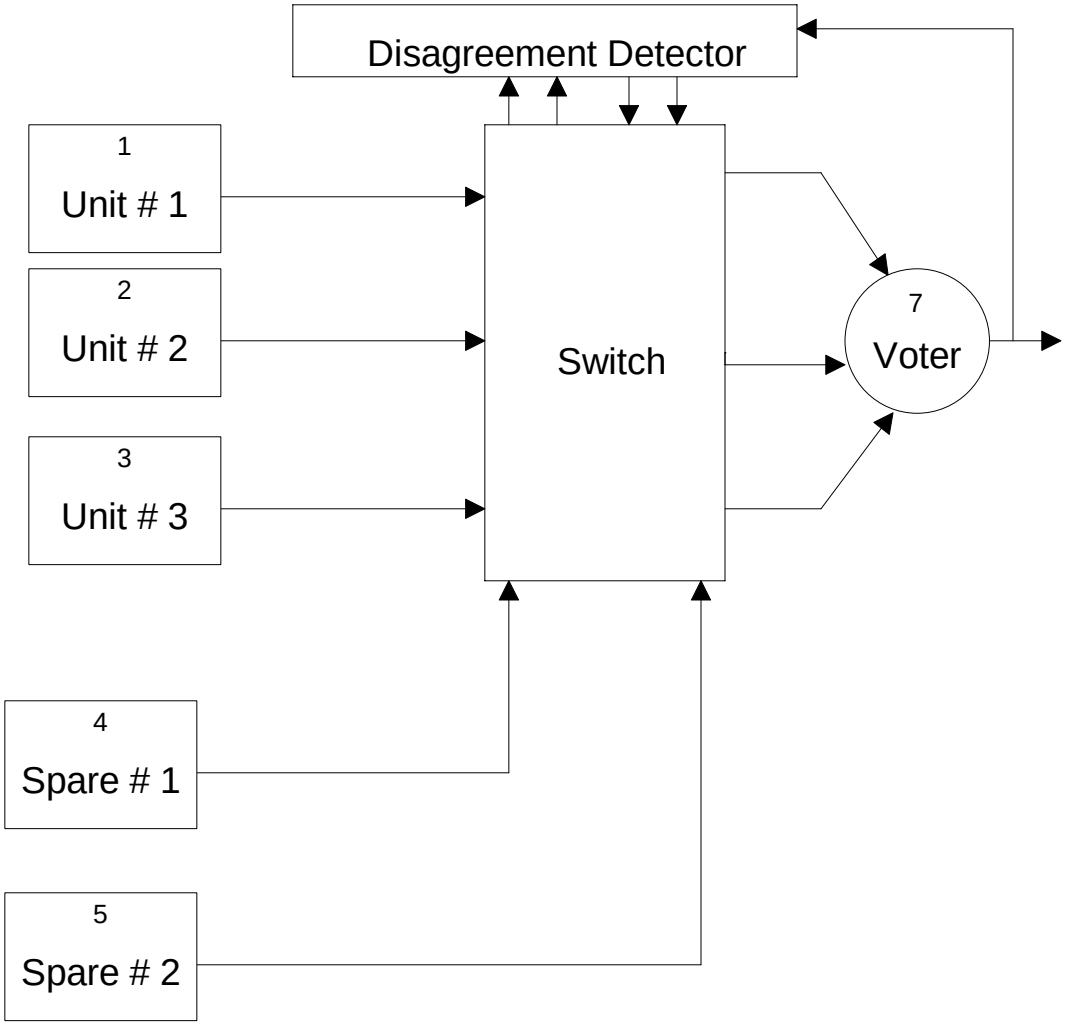


2. Active HWR: Duplication with Comparison



3. Hybrid HWR: NMR with spares (*N, k*) system

Combines advantages of passive, i.e. use of *NMR* and active redundancy, i.e. use of set of spares. When one of the *NMR* modules fails, it is replaced by a spare, so that the basic *NMR* configuration can continue. For *N* basic modules and *K* spare modules, i.e. (*N*, *K*) systems, the system can tolerate the failure of up to $K + \left\lfloor \frac{N}{2} \right\rfloor$ modules.



(B) Information Redundancy

1. Error Detecting Codes

- Parity Checking

a. Add a parity bit such that it makes the total number of "1" per word even (or odd)
b. Inexpensive in particular if *P{single}>> P{multiple}*
c. Easy to fail

- Checksum

a. Given a data (or information) word (D1, D2, D3, D4), add a bit vector (more than one bit, called *check bits*) *C = f* (D1, D2, D3, D4)
b. Store (or send) data with check bits (D1, D2, D3, D4, *C*)
c. Upon retrieving (or receiving), compute *f*
d. Compare *f* with stored *C,* if match, then no error; otherwise error has occurred.
e. Suppose that |C|= *b* bits, i.e. $C \varepsilon[0, 2^b - 1]$
f. Suppose that *f* maps data (D1, D2, D3, D4) uniformly in the range $[0, 2^b - 1]$, *i.e. f* is a hashing function.
g. The probability *p* that a random fault is not detected is $\frac{1}{2^b}$

Example
B= 2 C ε[0,3], p= ¼

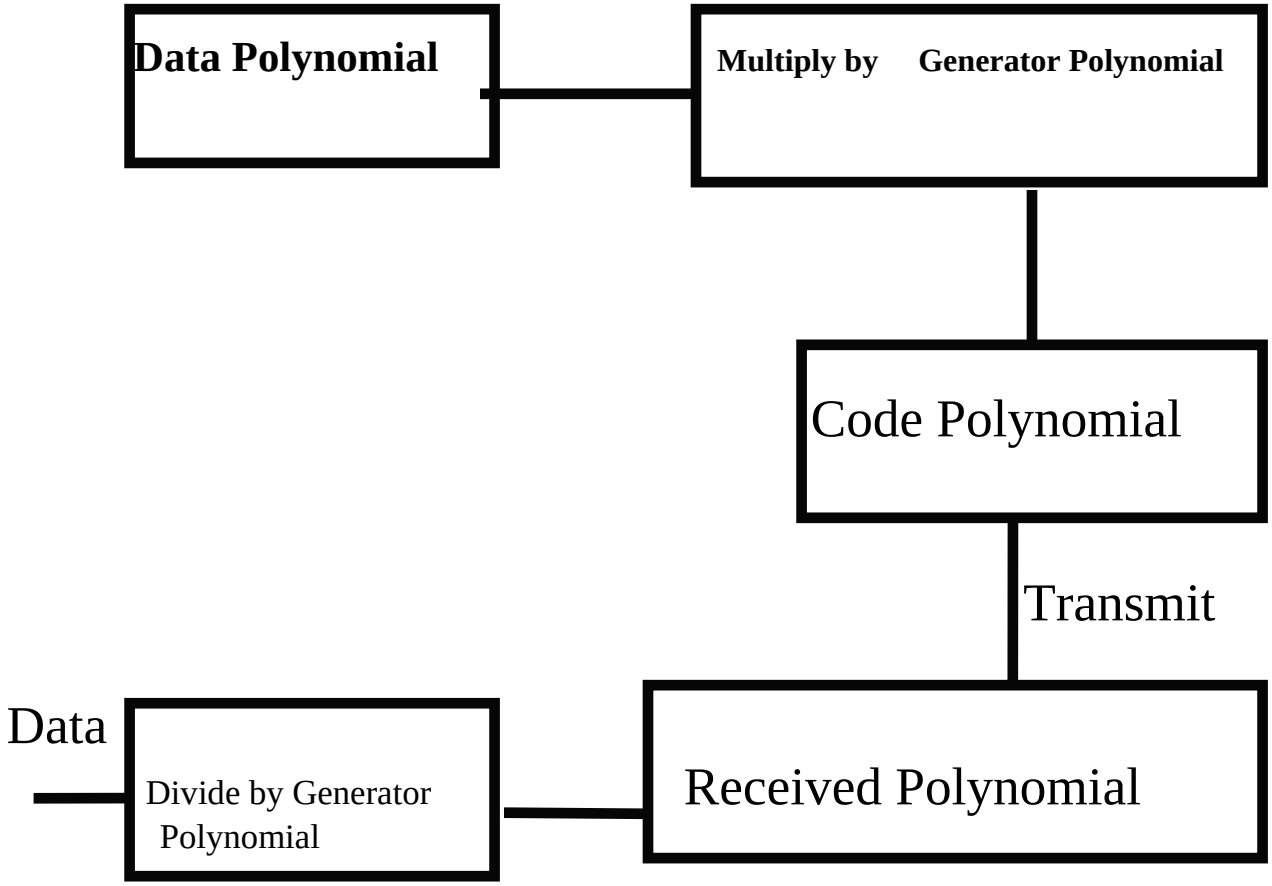
- *m-out-of-n*

a. *n* data bits out of which there is exactly *m 1s*
b. Any one bit change from *0* to *1* (or *1* to *0*) will change the number of *1s* to become *m+1* (or *m-1*); thus error detected.
c. Multiple unidirectional errors can be detected
d. The simplest (but the most expensive) is the *i-out-of-2i*

- *Cyclic Codes*

- Suppose k = number of information bits
  n = number of bits in the complete CW
  G(x) = generator polynomial of degree n-k.

- An (n, k) cyclic code will detect all single errors and all multiple adjacent errors affecting (n-k) or fewer bits (burst length).
- **General Scheme:**



Example:

k = 4, Data is 0101, i.e. D(x) = x^2+1**, n= 7,**
G(x) = x^3+x+1**,** then D(x) *G(x) = x^5+x^2+x+1**,** i.e. the data to be sent is 0100111.

$$D(X)G(X)=X^5+X^2+X+1 = 0100111$$

- At the receiving end, if the CW received is R(x), then $R(x)=D(x)*G(x)+S(x)$ where S(x) is the remainder which should normally be zero if no error has occurred (syndrome).

- Other Codes

a. Arithmetic Codes (See Handouts)
b. Berger Codes (See Handouts)

2. Error Correcting Codes:

(A) General Scheme

a. n bit words, *m* of which are data and *k* of which are for check
b. Word space is [0, $2^n - 1$] and code space is [0, $2^m - 1$]
c. Upon write (or send)
Write *m* data bits, compute *k* check bits, and append *k* check bits
d. Upon read (or receive)
Read *n* word bits, compute syndrome *s*
e. If *s* =0, then no errors; otherwise correct the word using *s*

(B) Examples

Hamming Error-Correcting Codes ($2^m - 1, 2^m - 1$ –m)

a. Overlapping parity Principles

$$2^k \geq m + k + 1$$

b. Example (7, 3) Hamming code
n = 7, *m* = 4, and *k* = 3

p1 =D1ΘD2 ΘD3
p2 = D2 ΘD3 ΘD4
p3 = D1 Θ D2 Θ D4

Data bit in error	Group affected P1 p2 p3
D1	P1 p3
D2	P1 p2 p3
D3	P1 p2
D4	P2 p3
p1	P1
p2	P2
p3	p3

c. Basic Hamming code can detect and correct single-bit error.

d. Modified Hamming code (with additional parity check bit that checks parity over the entire Hamming codeword) can be used to indicate double-bit error.

(C) Error Detecting/Correcting Conditions

$CD_{\min} \geq d$ to detect (d -1) errors.
$CD_{\min} \geq 2t + 1$ to detect and correct t or fewer errors
$CD_{\min} \geq t + d + 1$ to detect d and correct t ($t \leq d$)

(C) Software Redundancy

a. Consistency Check
b. N Version Programming
c. N Self-Checking Programming
d. Recovery Block

(D) Time Redundancy

a. Good for detecting transient errors
b. Alternating Logic (Self Dual functions)
c. Re-computing with shifted operands