Reliability of Embedded Systems

EE8205: Embedded Computer Systems
http://www.ee.ryerson.ca/~courses/ee8205/

Dr. Gul N. Khan
http://www.ecb.torontomu.ca/~gnkhan
Electrical, Computer & Biomedical Engineering
Toronto Metropolitan University

Overview

- Fault, Error and Sources of Faults
- Reliability Concepts
- Fault-tolerant Techniques
- Hardware and Software Fault-tolerance
- Fault Recovery

Fault-tolerant articles at the course WebPage

High Performance Embedded Systems

Many Safety Critical Applications Demand:

- High Performance
  - High Speed I/O Mb $\Rightarrow$ Gb/Sec
  - Large Memory (128 MB $\Rightarrow$ 4 GB)
  - Redundant Hardware and Reliable Software
- Fault-tolerance
- Tight Performance, Reliability and Availability Deadlines

Fault Tolerant Embedded Systems

Embedded System Development

Stages	Error Sources	Error Detection
Specification & design	Algorithm Design Formal Specification	Consistency checks Simulation
Prototype	Algorithm design Wiring & assembly Timing Component Failure	Stimulus/response Testing
Manufacture	Wiring & assembly Component failure	System testing Diagnostics
Installation	Assembly Component failure	System testing Diagnostics
Field Operation	Component failure Operator errors Environmental factors	Diagnostics

Reliability

RELIABILITY: **Survival Probability**

When function is critical during the mission time.

AVAILABILITY:

The fraction of time a system meets its specification.

Good when continuous service is important
but it can be delayed or denied

FAILSAFE: **System fails to a known safe state**

DEPENDABILITY:

Generalization: system does the right thing at right time.

System Reliability: Preliminaries

The Reliability, $R_F(t)$ of a System is the probability that no fault of the class F occurs (i.e. system survives) during time *t*.

$$R_F(t) = \Pr[t_{init} \leq t < t_f \text{ for all } f \in F]$$

where t_{init} is time of introduction of the system to service

t_f is time of occurrence of the first failure f drawn from F

Failure Probability, $Q_F(t)$ is complementary to $R_F(t)$

$$R_F(t) + Q_F(t) = 1$$

We can take off the F subscript from $R_F(t)$ and $Q_F(t)$

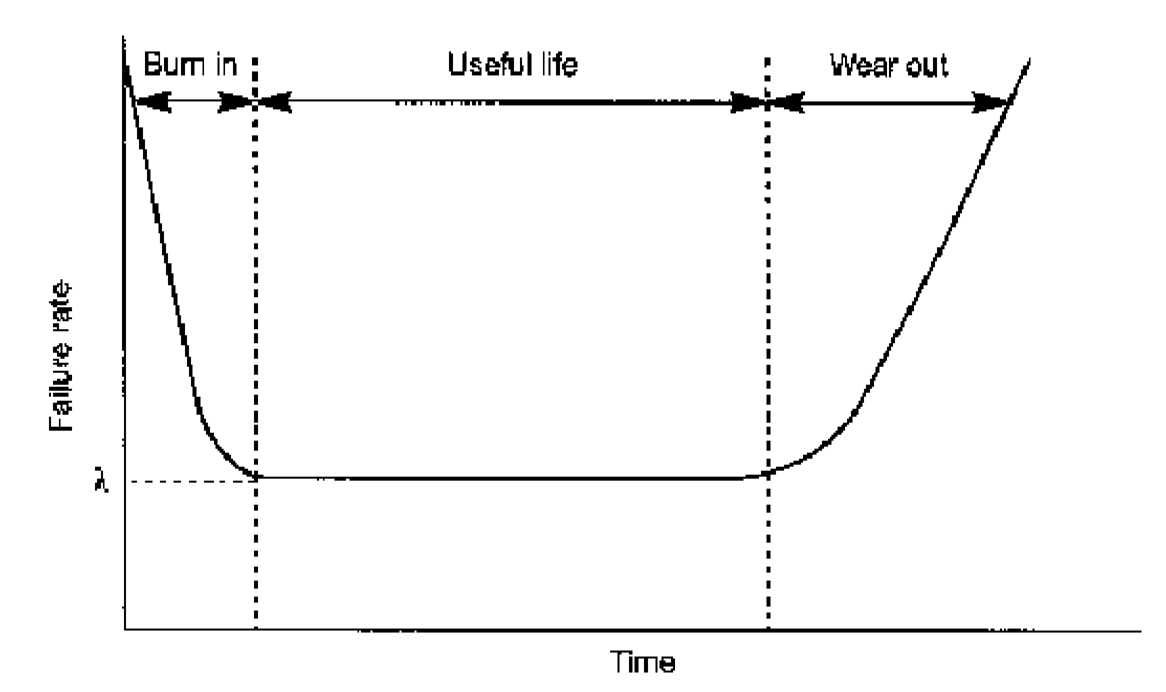
When the lifetime of a system is exponentially distributed, the reliability of the system is:

$$R(t) = e^{-\lambda t}$$

The parameter λ is called the failure rate

Component Reliability Model

It is not so straight forward.



During useful life, components exhibit a constant failure rate λ.

Reliability of a device can be modeled using an exponential distribution. $R(t) = e^{-\lambda t}$

λ is the failure rate

Component Failure Rate

Failure rates often expressed in failures / million operating hours

Automotive Embedded System Component	Failure Rate, λ
Military Microprocessor	0.022
Typical Automotive Microprocessor	0.12
Electric Motor Lead/Acid battery	16.9
Oil Pump	37.3
Automotive Wiring Harness (luxury)	775

MTTF: Mean Time To Failure

MTTF: Mean Time to Failure or Expected Life

MTTF: Mean Time To (first) Failure is defined as the expected value of t_f

$$MTTF = E[t_f] = \int_0^\infty R(t)dt = \frac{1}{\lambda}$$

where λ is the failure rate.

MTTF of a system is the expected time of the first failure in a sample of identical initially perfect systems.

MTTR: Mean Time To Repair is defined as the expected time for repair.

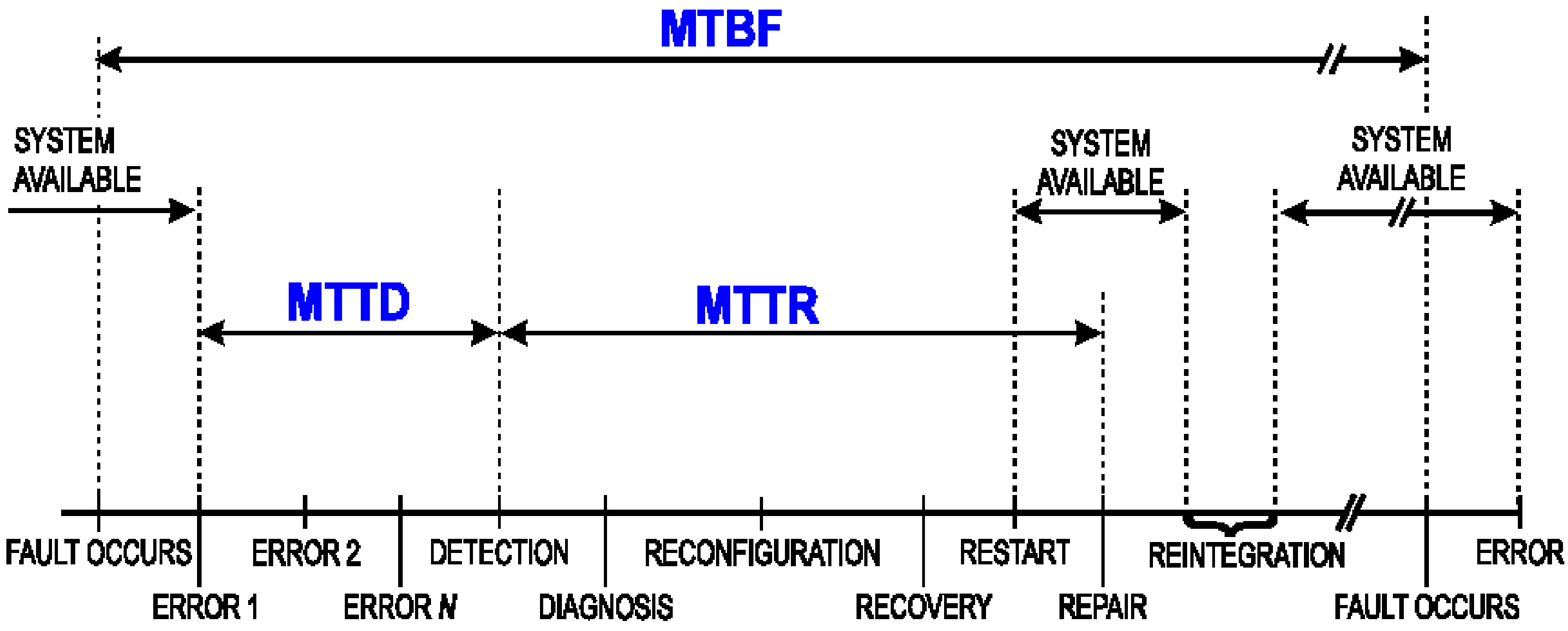
MTBF: Mean Time Between Failure

MTBF is approximated as MTTF for systems that do not permit repair.

MTBF, MTTR are applicable to repairable systems.

MTTF-MTTD-MTTR

Availability = MTBF/(MTBF + MTTR)



A Scenario for on-line detection and off-line repair. The measures – MTBF, MTTD, and MTTR are the average times to failure, to detection, and to repair.

Serial System Reliability

Serially Connected Components

Let $R_k(t)$ is the reliability of a single component k is given

$R_k(t) = e^{-\lambda_k t}$ where λ_k is constant failure rate

Assuming the failure rates of components are statistically independent. The overall system reliability $R_{ser}(t)$

$$R_{ser}(t) = R_1(t) \times R_2(t) \times R_3(t) \times \ldots. \times R_n(t)$$

$$R_{ser}(t) = \prod_{i=1}^{n} R_i(t)$$

where Π is a product operator

No redundancy: Overall system reliability depends on the proper working of each component

$$R_{ser}(t) = e^{-t\left(\sum_{i=1}^{n}\lambda_i\right)}$$

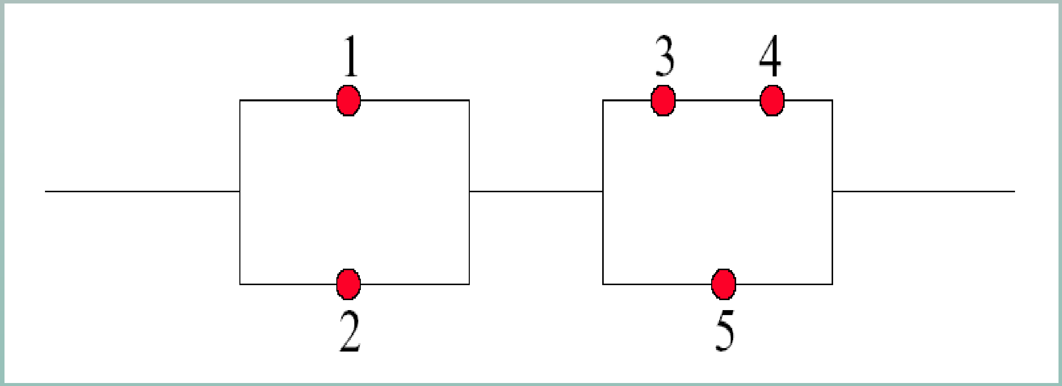
Serial Failure rate, $\lambda_{ser} = \sum_{i=1}^{n}\lambda_i$

System Reliability

Building a reliable serial system is extraordinarily difficult and expensive.

For example: if one is to build a serial system with 100 components each of which had a reliability of 0.999, the overall system reliability would be $(0.999)^{100} = 0.905$

Reliability of System of Components



Minimal Path Set:

Minimal set of components whose functioning ensures the functioning of the system

{1,3,4} {2,3,4} {1,5} {2,5}

Parallel System Reliability

Parallel Connected Components

$Q_k(t)$ is equal to $1 - R_k(t)$ where $R_k(t)$ is the reliability of a single component k

$Q_k(t) = 1 - e^{-\lambda_k t}$ where λ_k is a constant failure rate

Assuming the failure rates of components are statistically independent.

$$Q_{par}(t) = \prod_{i=1}^{n} Q_i(t)$$

Overall system reliability, $R_{par}(t) = 1 - \prod_{i=1}^{n} [1 - R_i(t)]$

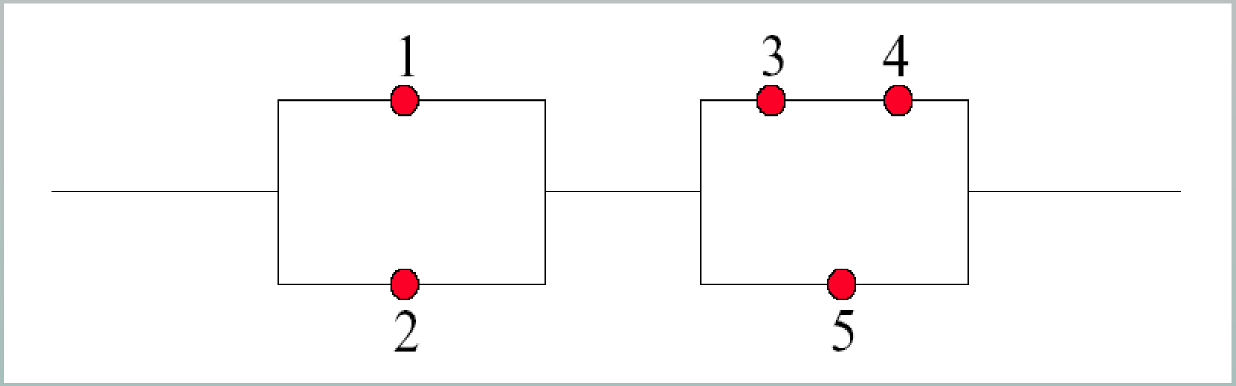
Consider 4 identical modules are connected in parallel

System will operate correctly provided at least one module is operational. If the reliability of each module is 0.95.

The overall system reliability = $1-[1-0.95]^4 = 0.99999375$

Parallel-Serial Reliability

Parallel and Serial Connected Components



Total reliability is the reliability of the first half, in serial with the second half.

Given R1=0.9, R2=0.9, R3=0.99, R4=0.99, R5=0.87

$$Rt = [1-(1-0.9)(1-0.9)][1-(1-0.87)(1-(0.99*0.99))]$$
$$= 0.987$$

Reliability Analysis of Serial-Parallel Systems

MTTF: Mean Time To Failure or Expected Life

MTTF is also used to specify the reliability of a system.

It is given by $E[X] = \int_0^\infty R(t)dt = 1/\lambda$

MTTF of Serial and Parallel Systems

$MTTF_{SER} = 1/\sum_{i=1}^{n}\lambda_i$

$MTTF_{PAR} = 1/\lambda \sum_{i=1}^{n} 1/i \approx \ln(n)/\lambda$

Reliability of TMR System

$R_{TMR}(t) = 3R^2(t) - 2R^3(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$

For smaller t, $R_{TMR}(t)$ is large but for large t it is small

Threshold time = $\ln 2/\lambda = 0.7/\lambda$

Faults and Their Sources

What is a Fault?

***Fault* is an erroneous state of software or hardware resulting from failures of its components**

Fault Sources

- Design errors
  - **Software or Hardware**
- Manufacturing Problems
  - **Damage, Fatigue and Deterioration**
- External disturbances
  - **Harsh environmental conditions, electromagnetic interference and ionization radiation**
- System Misuse

Fault Sources

Mechanical -- "wears out"
- Deterioration: wear, fatigue, corrosion
- Shock: fractures, overload, etc.

Electronic Hardware -- "bad fabrication; wears out"
- Latent manufacturing defects
- Operating environment: noise, heat, ESD, electro-migration
- Design defects (Pentium F-DIV bug)

Software -- "bad design"
- Design defects
- "Code rot" -- accumulated run-time faults

People
- Can take a whole lecture content...

Fault and Classifications

Failure: Component does not provide service

Fault: A defect within a system

Error: (manifestation of a fault)

A deviation from the required operation of the system or subsystem

Extent: Local (independent) or Distributed (related)

Value:

Determinate (stuck at high or low)

Indeterminate (varying values)

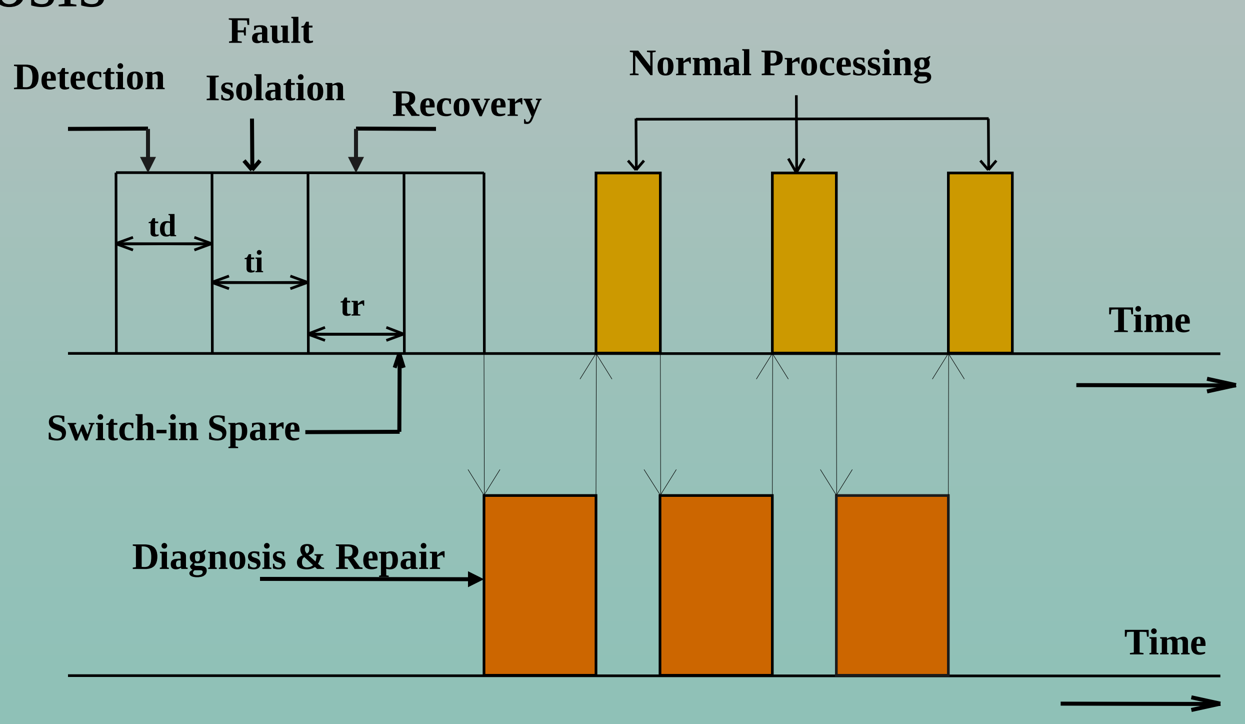
Duration:

- Transient- design errors, environment
- Intermittent- repair by replacement
- Permanent- repair by replacement

Fault-Tolerant Computing

Main aspects of FTC: Fault Tolerant Computing

- Fault detection
- Fault isolation and containment
- System recovery
- Fault Diagnosis
- Repair



Tolerating Faults

There is four-fold categorization to deal with the system faults and increase system reliability and/or availability.

Methods for Minimizing Faults

Fault Avoidance: How to prevent the fault occurrence.

by construction increase reliability by conservative design and use high reliability components.

Fault Tolerance: How to provide the service complying with the specification in spite of faults having occurred or occurring.

by redundancy

Fault Removal: How to minimize the presence of faults.

by verification

Fault Forecasting: How to estimate the presence, occurrence, and the consequences of faults. *by evaluation*

Fault-Tolerance **is the ability of a computer system to survive in the presence of faults.**

Fault-Tolerance Techniques

Hardware Fault Tolerance
Software Fault Tolerance

Hardware Fault-tolerance Techniques

- Fault Detection
- Redundancy (masking, dynamic)

  Use of extra components to mask the effect of a faulty component. (Static and Dynamic)
  Redundancy alone does not guarantee fault tolerance.
  It guarantee higher fault arrival rates (extra hardware).

  Redundancy Management is Important
  A fault tolerant computer can end up spending as much as 50% of its throughput in managing redundancy.

Hardware Fault-Tolerance

Fault Detection

Detection of a failure is a challenge

Many faults are latent that show up **(a lot)** later

Use watchdog timer ?

Fault detection gives warning when a fault occurs.

Duplication: Two identical copies of hardware run the same computation and compare each other results. When the results do not match a fault is declared.

Redundancy

Static and Dynamic Redundancy

Extra components mask the effect of a faulty component.

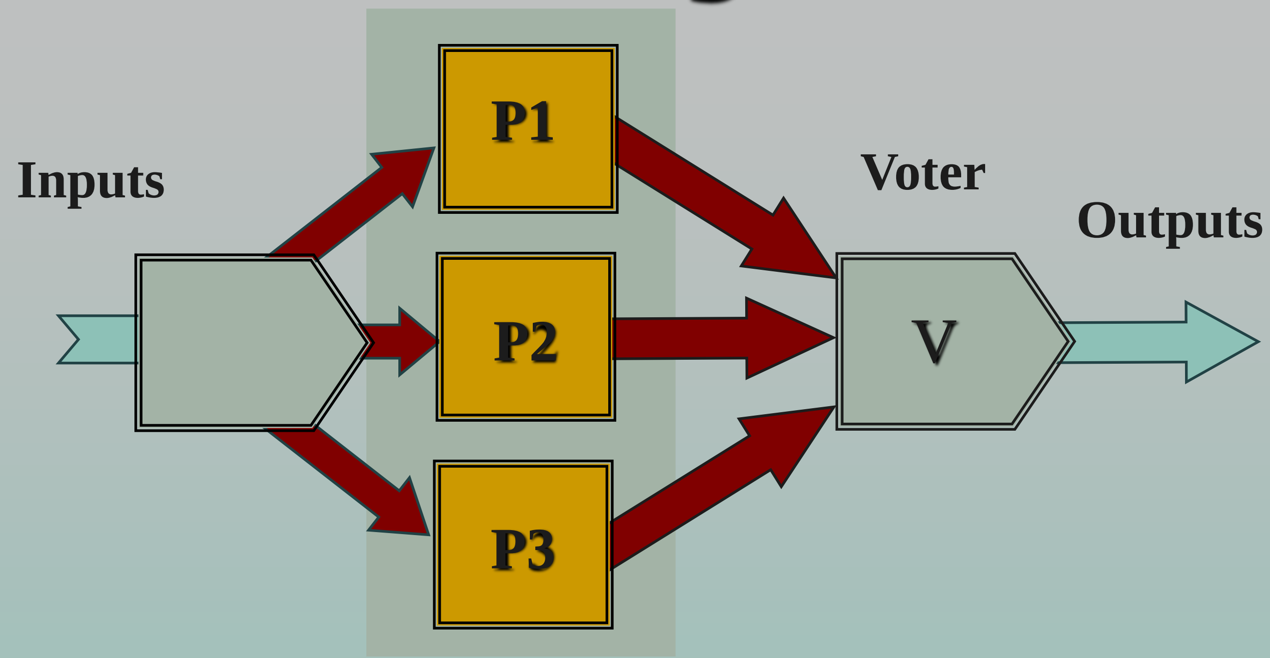
- **Masking Redundancy**

Static redundancy as once the redundant copies of an element are installed, their interconnection remains fixed e.g. TMR (Triple Modular Redundancy) where three identical copies of modules provide separate results to a voter that produces a majority vote.

- **Dynamic Redundancy**

System configuration is changed in response to a fault. Its success largely depends upon the fault detection ability.

TMR Configuration



- P1, P2 and P3 processors execute different versions of the code for the same application.
- Voter compares the results and forward the majority vote of results (two out of three).

TMR based hardware redundancy is transparent to the programmer

Software Fault-Tolerance

Hardware based fault-tolerance provides tolerance against physical i.e. hardware faults.

How to tolerate design/software faults?

It is virtually impossible to produce fully correct software.

We need something:

To prevent software bugs from causing system disasters.

To mask out software bugs.

Tolerating unanticipated design faults is much more difficult than tolerating anticipated physical faults.

Software Fault Tolerance is needed as:

Software bugs will occur no matter what we do.

No fully dependable way of eliminating these bugs.

These bugs have to be tolerated.

Some Software Failures

Software failure lead to partial/total system crashes

Cost of software has exceeded the cost of hardware.

Penalty costs for software failure are more significant.

Some Spectacular Software Failures

- **Space shuttle malfunction in 1982.**
- **Lethal doses of therapy radiation to Canadians in 1986.**
- **AT&T's telephone switching network failure in 1990.**
- **Loss of Ariane rocket and its payload in June 1996.**
- **Computer problems Airbus-330 Qantas flight from Singapore to Perth, October 2008.**
- **Airbus 330 AF flight 447, Rio de Janeiro to Paris May 2009**
- **iPhone 3G Glitches 2010 Dropped calls & choppy web surfing**

http://www5.in.tum.de/~huckle/bugse.html

Tolerating Software Failures

How to Tolerate Software Faults?

Software fault-tolerance uses *design redundancy* to mask residual design faults of software programs.

Software Fault Tolerance Strategy

Defensive Programming

If you can not be sure that what you are doing is correct.

Do it in many ways.

Some of them will turn to be right.

Review and test the software.

Verify the software.

Execute the specifications

Produce programs automatically

Full tools & technology were not available in the past

SW Fault-Tolerance Techniques

Software Fault Detection is a bigger challenge

Many software faults are of latent type that shows up later,
Can use a watchdog to figure out if the program is crashed

Fault-tolerant Software Design Techniques

- Recovery block scheme (RB)
  Dynamic redundancy-- Use an on-line acceptance test to determine which version to believe.
- N-version programming scheme (NVP)
  n-modular redundancy -- Write use multiple versions of the software and vote the results.

◆ **Hardware redundancy is needed to implement the above Software Fault-tolerance techniques.**

SW Fault-Tolerance Techniques

Fault-tolerant Software Design Techniques

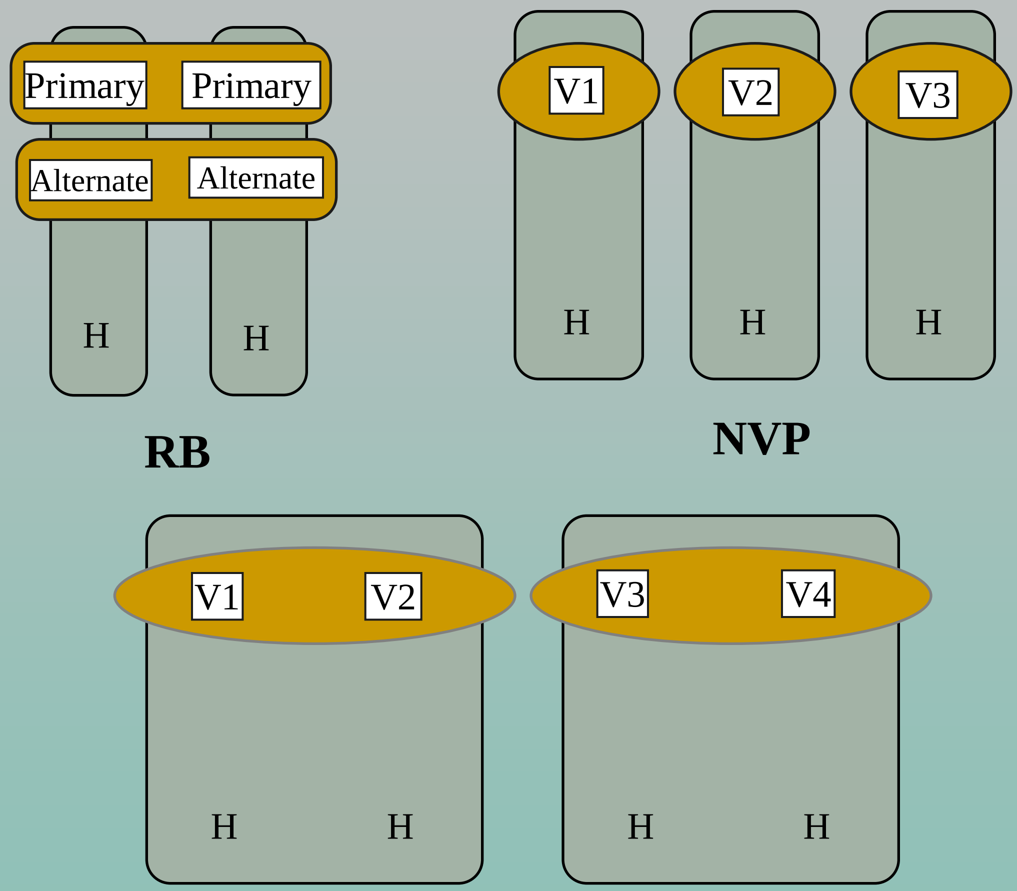
- Recovery block scheme (RB)
  Dynamic redundancy

- N-version programming scheme (NVP)
  n-modular redundancy

- **Hardware redundancy is needed to implement the above Software Fault-tolerance techniques.**

Software Fault-Tolerance

Fault-tolerant Software Design Techniques



NSCP: N-Self Checking Programming

NVP: N-Version Programming

N-independent program variants execute in parallel

Each program variant must be developed using different

Algorithms, Techniques, Programming Languages, Environments, Tools, etc.

For basic NVP, voting is done at the end

in *community-error-recovery* voting at intermediate points is done. Requires synchronization of programs at intermediate comparison points.

i.e. errors are detected and recovered at checkpoints which are inserted in all the versions of the software

N Self-Checking Version (NSCP) uses intermediate voting

Similar to *community-error-recovery* but acceptance test is by comparison checking.

RB: Recovery Block

RB Scheme comprises of three elements

- A primary module to execute critical software functions.
- Acceptance test for the output of primary module.
- Alternate modules perform the same functions as of primary.

A Simple Recovery Block Scheme

Calculating Square Root of x

Ensure AT $|y*y - x| = 0$

By P $y = sqrt(x)$

Else A_1

Else A_2

.

Else by A_{n-1}

Else Error

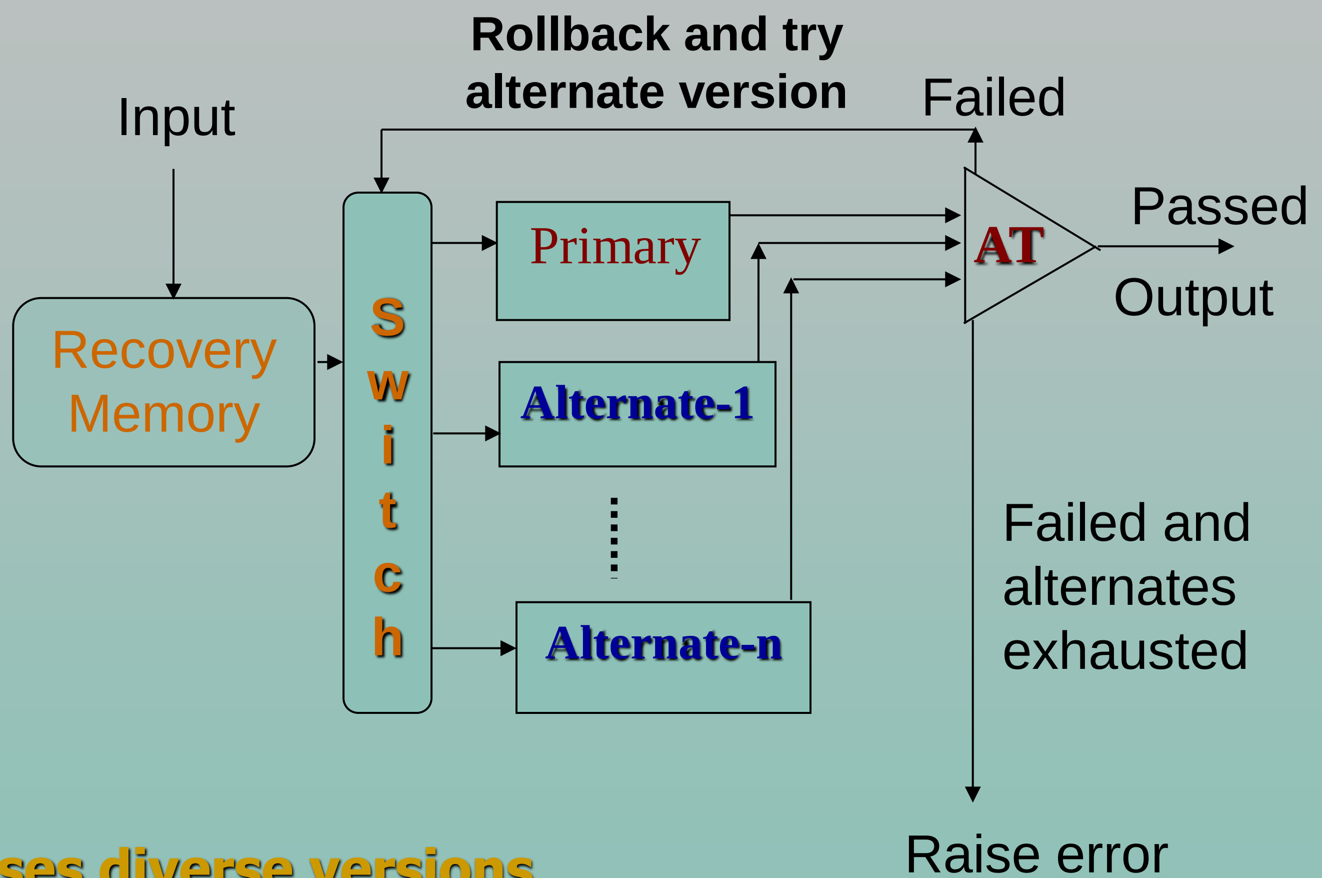
where AT = acceptance test condition

P is the primary module.

$A_{1 \Rightarrow n-1}$ are alternate modules.

RB: Recovery Block Scheme

An Architectural View of RB



RB uses diverse versions

Attempt to prevent residual software faults

Fault Recovery

Fault recovery technique's success depends on the detection of faults accurately and as early as possible.

Three classes of recovery procedures:

Full Recovery

It requires all the aspects of fault tolerant computing.

Degraded recovery: Also referred as *graceful degradation*

Similar to full recovery but no subsystem is switched-in.

Defective component is taken out of service.

Suited for multiprocessors.

Safe Shutdown

Often called fail-safe operations.

A limiting case of degraded recovery.

Fault Recovery

Fault recovery techniques restore enough of the system state that can restart a process execution without loss of acquired information.

Two Basic Approaches:

Forward Recovery

Produces correct results through continuation of normal processing.

Highly application dependent

Backward Recovery

Some redundant process and state information is recorded with the progress of computation.

Rollback the interrupted process to a point for which the correct information is available.

Backward Recovery Schemes

Retry

Operation is retried after fault detection.

Suits to transient faults.

In case of hard failures, reconfiguration is attempted.

Checkpointing

Some subset of system is saved at checkpoints and rollback is attempted.

JPL STAR and Tandem computer systems

Journaling

- Copy of the initial database is saved.
- All transactions that effect the data are kept on record during the process execution.
- When the process fails, recorded transactions are run on the backup data.

Concluding Remarks

- The common techniques for fault handling are fault avoidance, fault detection, masking redundancy, and dynamic redundancy.
- Any reliable embedded system must have its failure response carefully built into it, as some complementary set of actions and responses.
- System reliability can be modeled at a component and module level, assuming the failure rate is constant (exponential distribution).
- Reliability must be built into the embedded system project from the start.