Digitalni sistemi otporni na otkaz

Predavanje IV

Lecture Content

- Introduction
- Redundancy allocation
- Passive redundancy
  - Triple modular redundancy
  - Voting techniques
  - Voter logic
  - N-modular redundancy

Hardware Fault Tolerance Techniques

Introduction

Introduction I

- Hardware fault tolerance is achieved through the hardware redundancy, by providing two or more physical instances of a hardware component.
- For example, a system can include redundant processors, memories, buses or power supplies.
- Hardware redundancy is often the only available method for improving the dependability of a system, when other techniques, such as better components, design simplification, manufacturing quality control, software debugging, have been exhausted or shown to be more costly than redundancy.
- Originally, hardware redundancy techniques were used to cope with the low reliability of individual hardware elements.
- Designers of early computing systems replicated components at gate and flip-flop levels and used comparison or voting to detect or correct faults.
- As reliability of hardware components improved, the redundancy was shifted at the level of larger components, such as whole memories or arithmetic units, thus reducing the relative complexity of the voter or comparator with respect to that of redundant units.

Basic Forms of Hardware Redundancy

- There are three basic forms of hardware redundancy:
  - passive,
  - active and
  - hybrid.

- ***Passive redundancy*** achieves fault tolerance by masking the faults that occur without requiring any action on the part of the system or an operator.

- ***Active redundancy*** requires a fault to be detected before it can be tolerated.
- After the detection of the fault, the actions of location, containment and recovery are performed to remove the faulty component from the system.

- ***Hybrid redundancy*** combine passive and active approaches.
- Fault masking is used to prevent generation of erroneous results.
- Fault detection, location and recovery are used to replace the faulty component with a spare component.

Introduction II

- Hardware redundancy brings a number of penalties:
  - increase in weight, size, power consumption,
  - time to design, fabricate, and test.
- A number of choices need to be examined to determine a best way to incorporate redundancy into the system.
- For example, weight increase can be reduced by applying redundancy to the lower-level components.
- Cost increase can be minimized if the expected improvement in dependability reduces the cost of preventive maintenance for the system.
- In this lecture, we examine a number of different redundancy configurations and calculate the effect of redundancy on system dependability.
- We also discuss the problem of common-mode failures which are caused by faults occurring in a part of the system common to all redundant components.

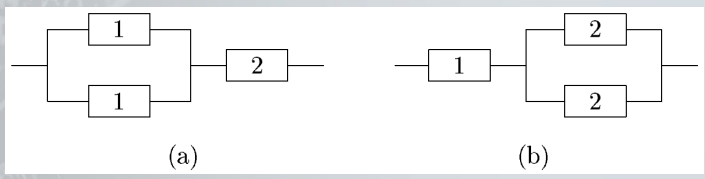

entity test_shift is
generic (width : integer := 17);
port (clk : in std_ulogic;
reset : in std_ulogic;
load : in std_ulogic;
en : in std_ulogic;
outp : out std_ulogic);
end test_shift;
Hardware Fault Tolerance Techniques
Redundancy Allocation

Redundancy Allocation I

- A number of possibilities have to be examined to determine at which level redundancy needs to be provided and which components need to be made redundant.
- To understand the importance of these decisions, consider a serial system consisting of two components with reliabilities $R1$ and $R2$. If the system reliability $R = R_1 R_2$ does not satisfy the design requirements, the designer may decide to make some of the components redundant.
- The possible choices of redundant configurations are shown in Figure below. Assuming the component failures are mutually independent, the corresponding reliabilities of these systems are

$$R_a = (2R_1 - R_1^2)R_2$$

$$R_b = (2R_2 - R_2^2)R_1$$



Redundancy allocation

Redundancy Allocation II

- Taking the difference of R_b and R_a, we get

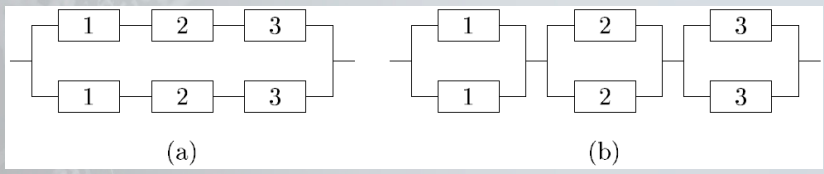
$$R_a - R_b = R_1 R_2 (R_2 - R_1)$$

- It follows from this expression that the higher reliability is achieved if we duplicate the component that is least reliable.

- If $R_1 < R_2$, then configuration shown on Figure (a) on the previous slide is preferable, and vice versa.

Level of Redundancy I

- Another important parameter to examine is the *level of redundancy*.
- Consider the system consisting of three serial components.
- In high-level redundancy, the entire system in duplicated, as shown in Figure (a) below.
- In low-level redundancy, the duplication takes place at component level, as shown in Figure (b).
- If each of the block of the diagram is a subsystem, the redundancy can be placed at even lower levels.



High-level and low-level redundancy

Level of Redundancy II

- Let us compare the reliabilities of the systems in Figures (a) and (b) shown on Slide 10.
- Assuming that the component failures are mutually independent, we have

$$R_a = 1 - (1 - R_1 R_2 R_3)^2$$
$$R_b = (1 - (1 - R_1)^2)(1 - (1 - R_2)^2)(1 - (1 - R_3)^2)$$

- The system in Figure (a) is a parallel combination of two serial sub-systems.
- The system in Figure (b) is a serial combination of three parallel sub-systems.
- As we can see, the reliabilities R_a and R_b differ, although the systems have the same number of components.

Level of Redundancy III

- If $R_1 = R_2 = R_3 = R$, then the difference is

$$R_b - R_a = 6R^3(1 - R)^2$$

- Consequently, $R_b > R_a$, i.e. low-level redundancy yields higher reliability than high-level redundancy.
- However, this dependency only holds if the components failures are truly independent in both configurations.
- In reality, common-mode failures are more likely to occur with low-level rather than with high-level redundancy, since in high-level redundancy the redundant units are normally isolated physically and therefore are less prone to common sources of stress.


entity test_shift is
generic (width : integer := 17);
port (clk : in std_ulogic;
reset : in std_ulogic;
load : in std_ulogic;
en : in std_ulogic;
outp : out std_ulogic);
end test_shift;
Hardware Fault Tolerance Techniques
Passive Redundancy

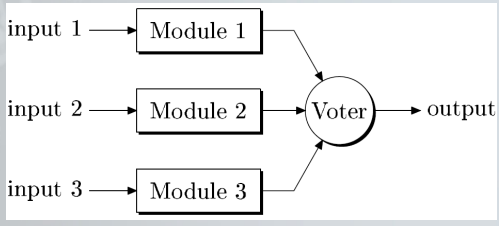
Introduction

- Passive redundancy approach masks faults rather than detect them.
- Masking insures that only correct values are passed to the system output in spite of the presence of a fault.
- In this section we first study the concept of triple modular redundancy, and then extend it to a more general case of N-modular redundancy.


entity test_shift is
generic (width : integer := 17);
port (clk : in std_ulogic;
reset : in std_ulogic;
load : in std_ulogic;
en : in std_ulogic;
outp : out std_ulogic);
end test_shift;
Hardware Fault Tolerance Techniques
Triple Modular Redundancy

Triple Modular Redundancy I

- The most common form of passive hardware redundancy is ***triple modular redundancy*** (TMR).
- The basic configuration is shown in Figure below.
- The components are triplicated to perform the same computation in parallel.
- Majority voting is used to determine the correct result.
- If one of the modules fails, the majority voter will mask the fault by recognizing as correct the result of the remaining two fault-free modules.
- Depending on the application, the triplicated modules can be processors, memories, disk drives, buses, network connections, power supplies, etc.



Triple modular redundancy

Triple Modular Redundancy II

- A TMR system can mask only one module fault. A failure in either of the remaining modules would cause the voter to produce an erroneous result.
- Dependability of a TMR system can be improved by removing failed modules from the system.
- TMR is usually used in applications where a substantial increase in reliability is required for a short period.
- For example, TMR is used in the logic section of launch vehicle digital computer (LVDC) of Saturn 5. Saturn 5 is a rocket carrying Apollo spacecrafts to the orbit.
- The functions of LVDC include the monitoring, testing and diagnosis of rocket systems to detect possible failures or unsafe conditions.
- As a result of using TMR, the reliability of the logic section for a 250-hr mission is approximately twenty times larger than the reliability of an equivalent simplex system.
- However, as we see in the next section, for longer duration missions, a TMR system is less reliable than a simplex system.

Reliability Evaluation I

- The fact that a TMR system which can mask one module fault does not immediately imply that the reliability of a TMR system is higher than the reliability of a simplex system.
- To estimate the influences of TMR on reliability, we need to take the reliability of modules as well as the duration of the mission into account.
- A TMR system operates correctly as long as two modules operate correctly. Assuming that the voter is perfect and that the component failures are mutually independent, the reliability of a TMR systems is given by

$$R_{TMR} = R_1R_2R_3 + (1-R_1)R_2R_3 + R_1(1-R_2)R_3 + R_1R_2(1-R_3)$$

- The term $R_1R_2R_3$ gives the probability that the first module functions correctly *and* the second module functions correctly *and* the third module functions correctly.
- The term $(1-R_1)R_2R_3$ stands for the probability that the first module has failed *and* the second module functions correctly *and* the third module functions correctly, etc.
- The overall probability is an *or* of the probabilities of the terms since the events are mutually exclusive. If $R_1 = R_2 = R_3 = R$, the above equation reduces to

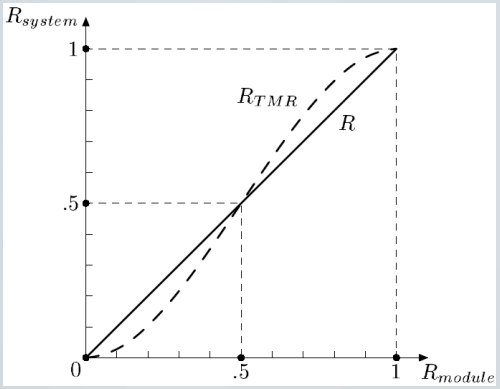
$$R_{TMR} = 3R^2 - 2R^3$$

Reliability Evaluation II

- Figure on the right compares the reliability of a TMR system R_{TMR} to the reliability of a simplex system consisting of a single module with reliability R.
- The reliabilities of the modules composing the TMR system are assumed to be equal R.
- As can be seen, there is a point at which $R_{TMR} = R$.
- This point can be found by solving the equation

$$3R^2 - 2R^3 = R.$$

- The three solutions are 0.5, 1 and 0, implying that the reliability of a TMR system is equal to the reliability of a simplex system when:
  - the reliability of the module is $R = 0.5$,
  - when the module is perfect ($R = 1$), or
  - when the module is failed ($R = 0$).



TMR reliability compared to simplex system reliability

Reliability Evaluation III

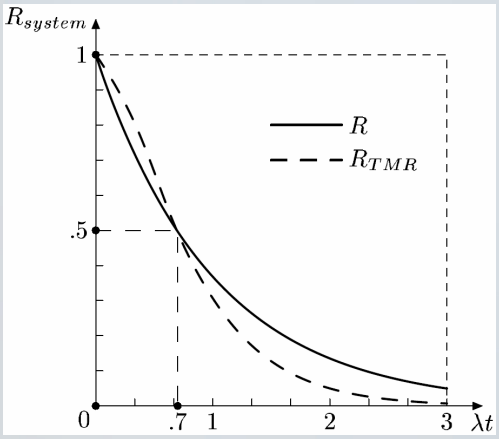
- Previous analysis further illustrates a difference between fault tolerance and reliability.
- A system can be fault-tolerant and still have a low overall reliability.
- For example, a TMR system build out of poor-quality modules with $R = 0.2$ will have a low reliability of $R_{TMR} = 0.136$.
- Vice versa, a system which cannot tolerate any faults can have a high reliability, e.g. when its components are highly reliable.
- However, such a system will fail as soon as the first fault occurs.

Reliability Evaluation IV

- Next, let us consider how the reliability of a TMR system changes as a function of time.
- For a constant failure rate λ, the reliability of the system varies exponentially as a function of time

  $R(t) = e^{-\lambda t}$.
- Substituting this expression in last equation on Slide 18, we get

  $$R_{TMR}(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$
- Figure on the right shows how the reliabilities of simplex and TMR systems change as functions of time.
- The value of λ*t*, rather than *t* is shown on the *x*-axis, to make the comparison independent of the failure rate.
- Recall that 1/ λ=*MTTF*, so that the point λ*t* =1 corresponds to the time when the system is expected to experience the first failure.
- One can see that the reliability of the TMR system is higher than the reliability of the simplex system in the period between 0 and approximately 0.7λ*t*.
- That is why TMR is suitable for applications whose mission time is shorter than 0.7 of MTTF.



TMR reliability as a function of λ*t*


entity test_shift is
generic (width : integer := 17);
port (clk : in std_ulogic;
reset : in std_ulogic;
load : in std_ulogic;
en : in std_ulogic;
outp : out std_ulogic);
end test_shift;
Hardware Fault Tolerance Techniques
Voting Techniques

Voting Techniques

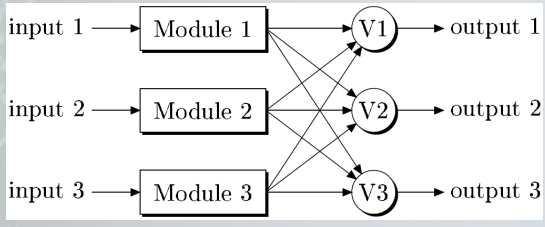
- In the previous section we evaluated the reliability of a TMR system assuming that the voter is perfect. Clearly, such an assumption is not realistic. A more precise estimation of the reliability of a TMR system takes the reliability of the voter into account:

$$R_{TMR} = (3R^2 - 2R^3)R_v$$

- The voter is in series with the redundant modules, since if it fails, the whole system fails.
- The reliability of the voter must be very high in order to keep the overall reliability of the TMR system higher than the reliability of a corresponding simplex system.
- Fortunately, the voter is typically a very simple device compared to the redundant components and therefore its failure probability is much smaller.
- Still, in some systems the presence of a single point of failure is not acceptable by qualitative requirement specifications.
- We call ***single point of failure*** any component within a system whose failure leads to the failure of the system. In such cases, more complicated voting schemes are used.

Triplicated Voters Fault Tolerant Architecture

- One possibility is to *decentralize* voting by having three voters instead of one, as shown in Figure below.
- Decentralized voting avoids the single point of failure, but requires establishing consensus among three voters.
- Another possibility is the so called ***master-slave approach*** that replaces a failed voter with a standby voter.



TMR system with three voters

Voter-Related Problems

- Voting heavily relies on an accurate timing.
- If values arrive at a voter at different times, incorrect voting result may be generated.
- Therefore, a reliable time service should be provided throughout a TMR or NMR system.
- This can be done either by using additional interval timers, or by implementing asynchronous protocols that rely on the progress of computation to provide an estimate of time.
- Multiple-processor systems should either provide a fault-tolerant global clock service that maintains a consistent source of time throughout the system, or to resolve time conflicts on an ad-hoc basis.
- Another problem with voting is that the values that arrive at a voter may not completely agree, even in a fault-free case.
- For example, analog to digital converters may produce values which slightly disagree.
- A common approach to overcome this problem is to accept as correct the *median* value which lies between the remaining two.
- Another approach is to ignore several least significant bits of information and to perform voting only on the remaining bits.

Hardware Fault Tolerance Techniques

Voter Logic

Voter Logic I

- Voting can be implemented in either hardware or software.
- Hardware voters are usually quick enough to meet any response deadline.
- If voting is done by software voters that must reach a consensus, adequate time may not be available.

Voter Logic II

- It is useful to discuss the structure of a majority logic voter. This allows the designer to appreciate the complexity of a voter and to judge when majority voter techniques are appropriate.
- The structure of a voter is easy to realize in terms of logic gates and also through the use of other digital logic-design techniques.
- The basic logic function for a TMR voter is based on the Truth Table given in Table shown below.

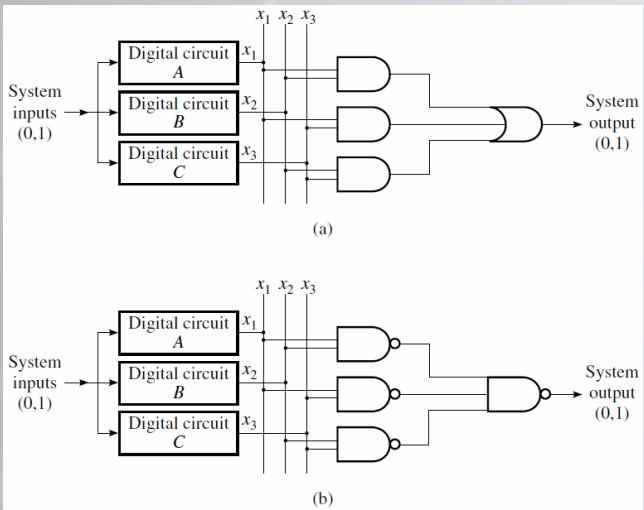
Inputs			Outputs	
x_1	x_2	x_3	$f_v(x_1x_2x_3)$	
0	0	0	0	Two
0	0	1	0	or
0	1	0	0	three
1	0	0	0	zeroes
1	1	0	1	Two
1	0	1	1	or
0	1	1	1	three
1	1	1	1	ones

- The result of the logic simplification yields a voter logic function, as follows:

$$f_v(x_1, x_2, x_3) = x_1x_2 + x_1x_3 + x_2x_3$$

Voter Logic III

- Such a circuit is easy to realize with basic logic gates as shown in Figure (a), where three AND gates plus one OR gate is used, and in Figure (b), where four NAND gates are used.



Two circuit realizations of a TMR voter.
(a) A voter constructed from AND/ OR gates
(b) a voter constructed from NAND gates

Voting and Error Detection I

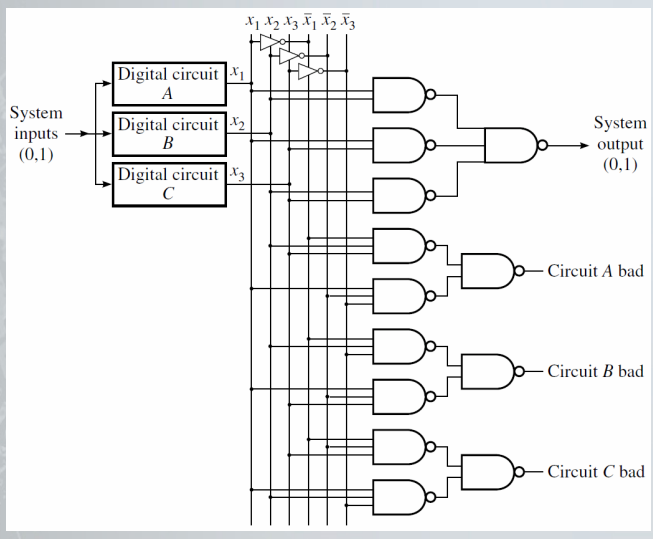
- There are many reasons why it is important to know which circuit has failed when *N*-modular redundancy is employed, such as the following:
  1. If a panel with light-emitting diodes (LEDs) indicates circuit failures, the operator has a warning about which circuits are operative and can initiate replacement or repair of the failed circuit.
     This eliminates much of the need for off-line testing.
  2. The operator can take the failure information into account in making a decision.
  3. The operator can automatically lock out a failed circuit.
  4. If spare circuits are available, they can be powered up and switched in to replace a failed component.
- If one compares the voter inputs the first time that a circuit disagrees with the majority, a failed warning can be initiated along with any automatic action.

Voting and Error Detection II

- We can illustrate this by deriving the logic circuits that would be obtained for a TMR system.
- If we let $f_v(x_1, x_2, x_3)$ represent the voter output as before and $f_{e1}(x_1, x_2, x_3)$, $f_{e2}(x_1, x_2, x_3)$, and $f_{e3}(x_1, x_2, x_3)$ represent the signals that indicate errors in circuits one, two, and three, respectively, then the truth table shown in Table below holds.
- A simple logic realization of these 4 outputs using NAND gates is shown on the next Slide.

Inputs			Outputs			
x_1	x_2	x_3	f_v	f_{e1}	f_{e2}	f_{e3}
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	1	0
0	1	1	1	1	0	0
1	0	0	0	1	0	0
1	0	1	1	0	1	0
1	1	0	1	0	0	1
1	1	1	1	0	0	0

Voting and Error Detection III



Circuit that realizes the four switching functions given in Table shown on the Slide 31 for a TMR majority voter and error detector.

Voting and Error Detection IV

- The reader should realize that circuit shown on the Slide 33, with 13 NAND gates and 3 inverters, is only for a single bit output.
- For a 32-bit computer word, the circuit will have 96 inverters and 416 NAND gates.
- The integrated circuit failure rate, λ, is *roughly* proportional to the square root of the number of gates,

$$\lambda \approx \sqrt{g}$$

- and for our example,

$$\lambda \approx \sqrt{512} \approx 22.6$$

- If we assume that the circuit on which we are voting should have 10 times the failure rate of the voter, the circuit would have 51,076 or about 50,000 gates.
- The implication of this computation is clear:

One should not employ voters to improve the reliability of small circuits because the voter reliability may wipe out most of the intended improvement.

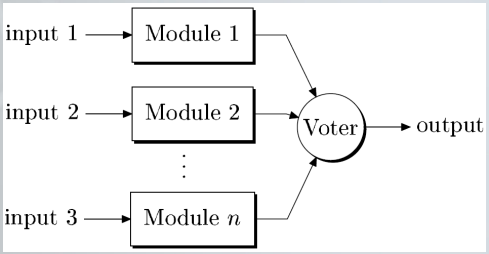

entity test_shift is
generic (width : integer := 17);
port (clk : in std_ulogic;
reset : in std_ulogic;
load : in std_ulogic;
en : in std_ulogic;
outp : out std_ulogic);
end test_shift;
Hardware Fault Tolerance Techniques
N-modular Redundancy

N-modular Redundancy I

- The preceding section introduced TMR as a majority voting scheme for improving the reliability of digital systems and components.
- Of course, this is the most common implementation of majority logic because of the increased cost of replicating systems.
- However, with the reduction in cost of digital systems from integrated circuit advances, it is practical to discuss *N*-version voting or, as it is now more popularly called, *N*-modular redundancy.
- In general, *N* is an odd integer; however, if we have additional information on which systems are malfunctioning and also the ability to lock out malfunctioning systems, it is feasible to let *N* be an even integer.
- If we contemplate using *N*-modular redundancy for a digital system composed of the three subsystems *A*, *B*, and *C*, the question arises:
- Do we use *N*-modular redundancy on three systems ($A_1B_1C_1$, $A_2B_2C_2$, and $A_3B_3C_3$) with one voter, or do we apply voting on a lower level, with one voter comparing $A_1A_2A_3$, a second comparing $B_1B_2B_3$, and a third comparing $C_1C_2C_3$?
- We will expect that voting on a component level is superior and that the reliability of the voter must be considered. This section explores such models.

N-modular Redundancy II

- Architecture of the *N*-modular redundancy (NMR) system is shown in the Figure below.
- It consists from *N* identical modules and one voting module that makes the final output calculation.
- A NMR system can mask $\lfloor N/2 \rfloor$ module faults.



N-modular redundancy - Basic operating principle

System Voting I

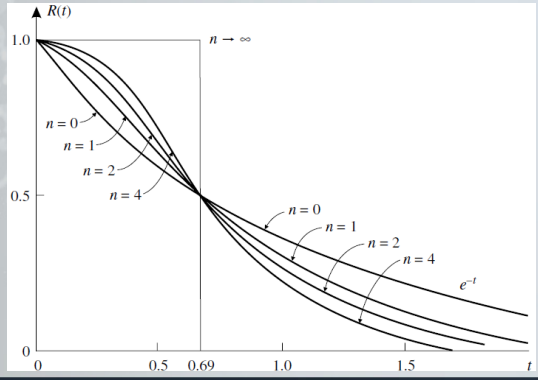
- In this case voting is applied on the system level, meaning that the whole system that should have high reliability is replicated *N* times, and only one voter is used.
- If one considers a system of $2n + 1$ voters (note that this is an odd number), parallel digital elements, and a single perfect voter, the reliability expression is given by

$$R = \sum_{i=n+1}^{2n+1} \binom{2n+1}{i} R^i (1-R)^{2n+1-i}$$

- The preceding expression is plotted in Figure shown on the next slide, for the case of one, three, five, and nine elements, assuming $R = e^{-\lambda t}$.

System Voting II

- Note that as $n \rightarrow \infty$, the MTTF of the system $\rightarrow 0.69/\lambda$. In the limiting behavior as $n \rightarrow \infty$, the reliability function approaches the three straight lines shown in Figure below.
- Further study of this figure reveals another important principle - *N*-modular redundancy is only superior to a single system in the high-reliability region.
- To be more specific, *N*-modular redundancy is superior to a single element for $\lambda t < 0.69$; thus, in system design, one must carefully evaluate the values of reliability obtained over the range $0 < t <$ maximum mission time for various values of n and λ.



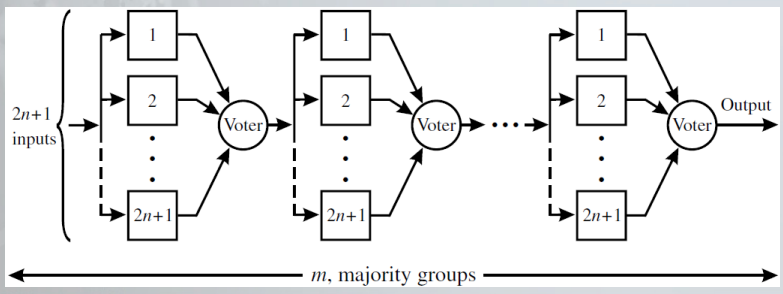
Reliability of a majority voter containing $2n + 1$ circuits

Subsystem Level Voting I

- Assume that a digital system is composed of *m* series subsystems, each having a constant-failure rate λ, and that voting is to be applied at the subsystem level. The majority voting circuit is shown in Figure below.
- Since this configuration is composed of just the *m*-independent series groups of the same configuration as previously considered, the reliability is simply given by Equation shown on Slide 37 raised to the *m*th power

$$R = \left[\sum_{i=n+1}^{2n+1} \binom{2n+1}{i} R_{ss}^{i} \left(1 - R_{ss}\right)^{2n+1-i} \right]^{m}$$

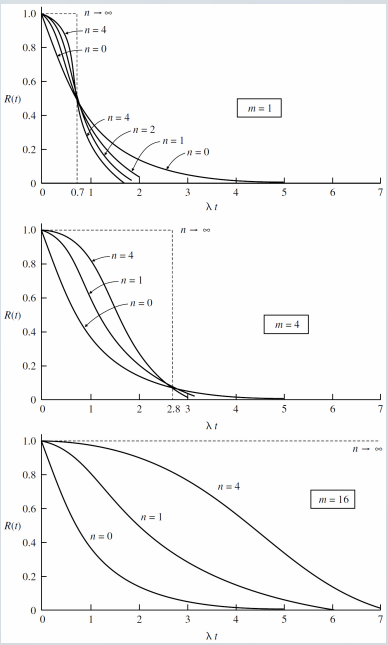
- where R_{ss} is the subsystem reliability.

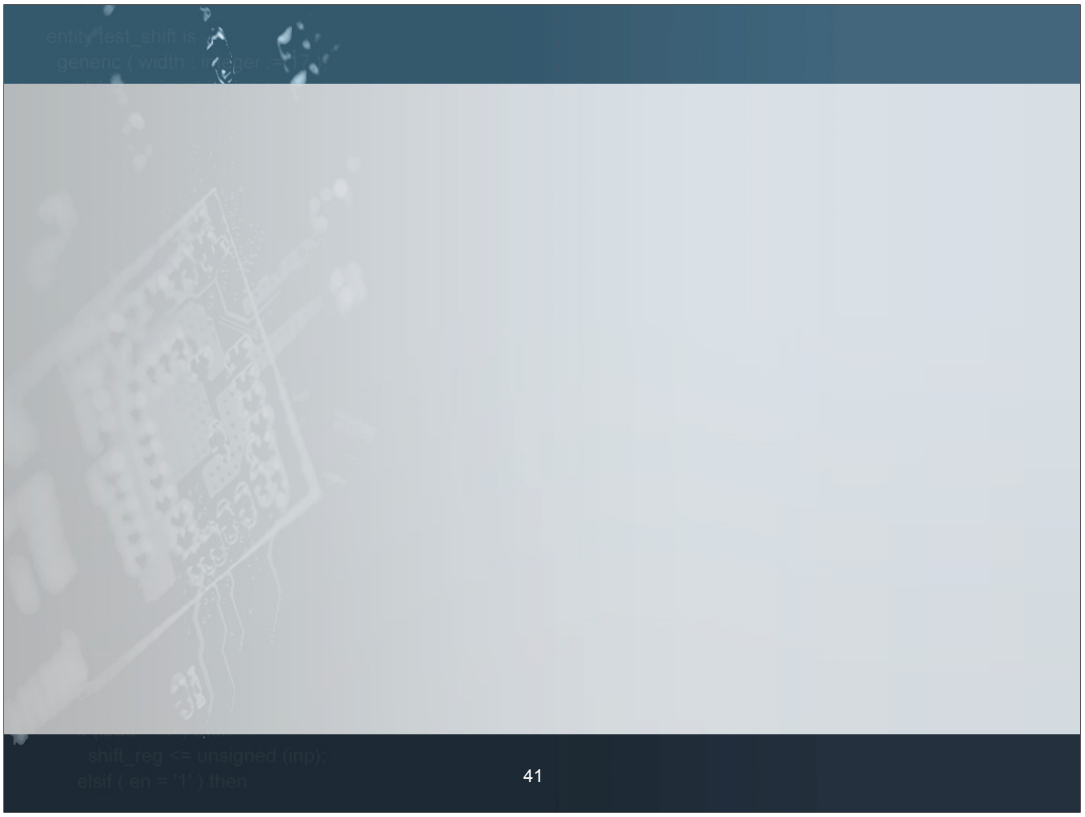


Subsystem voting architecture

Subsystem Level Voting II

- The subsystem reliability R_{ss} is, of course, not equal to a fixed value of R; it instead decays in time.
- In fact, if we assume that all subsystems are identical and have constant-hazard and failure rates, and if the system failure rate if λ, the subsystem failure rate would be λ/m, and $R_{ss} = e^{-\lambda t/m}$.
- Substitution of the time-dependent expression ($R_{ss} = e^{-\lambda t/m}$) into equation on Slide 39 yields the time-dependent expression for $R(t)$.
- Numerical computations of the system reliability functions for several values of m and n are shown in Figure on the right.
- It has been shown that as $n \rightarrow \infty$, the MTTF $\approx 0.7m/\lambda$. This is a direct consequence of the limiting behavior of Equation from Slide 37, as was discussed previously.
- To use Equation from Slide 39 in design, one chooses values of n and m that yield a value of R, which meets the design goals.
- If there is a choice of values (n, m) that yield the desired reliability, one would choose the pair that represents the lowest cost system.




entity test_shift is
generic (width : integer :=
shift_reg <= unsigned (inp);
elsif (en = '1') then
41