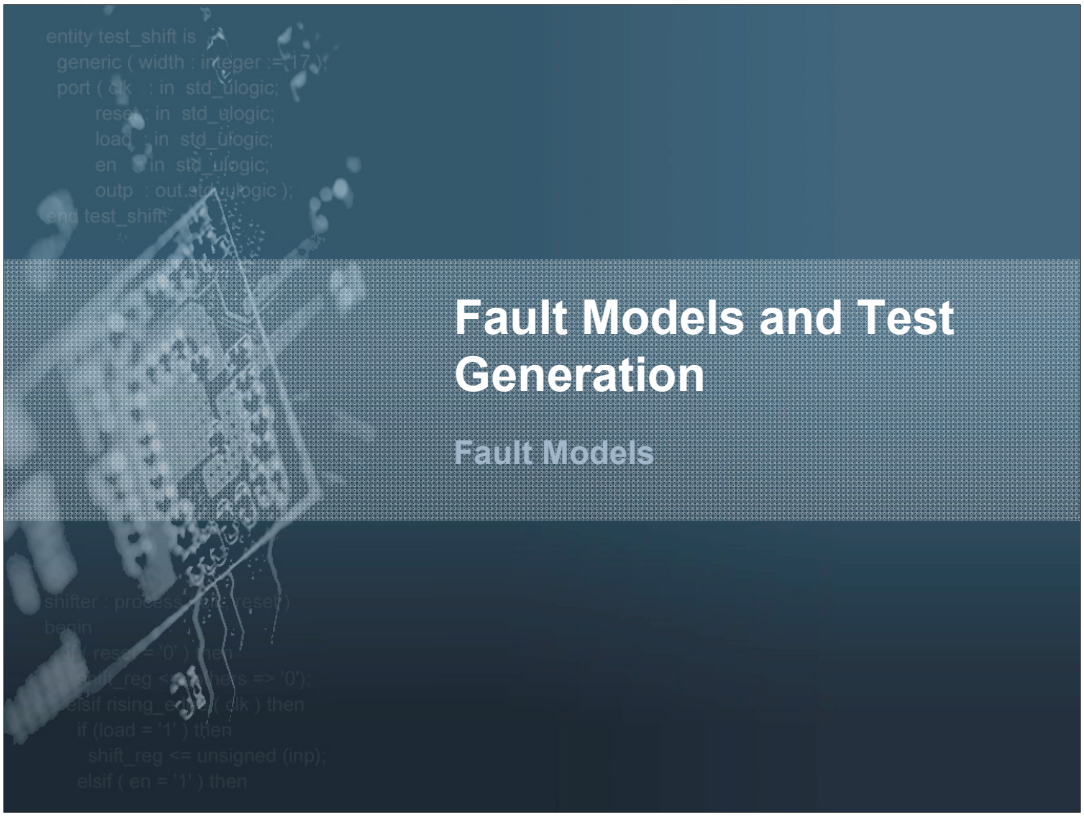

# Digitalni sistemi otporni na otkaz

## Predavanje III

## Lecture Content

- Fault models
  - Introduction
  - Fault models at different abstraction levels
  - Structural fault models
- Combinational logic and fault simulation
  - Introduction
  - Fault simulation
- Test generation for combinational circuits
  - Introduction
  - Simple test generation algorithm
- Test generation for sequential circuits and design for testability
  - Scan design
  - Scan architectures
  - Costs and benefits of scan




# Fault Models and Test Generation

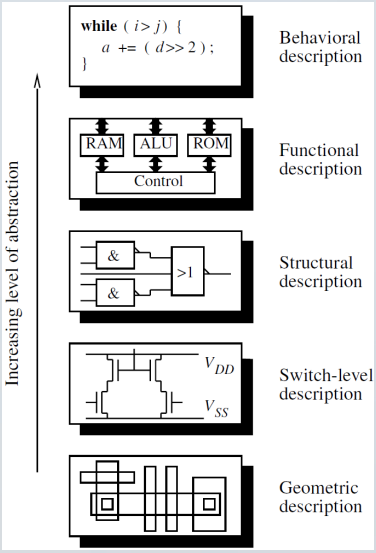
## Fault Models

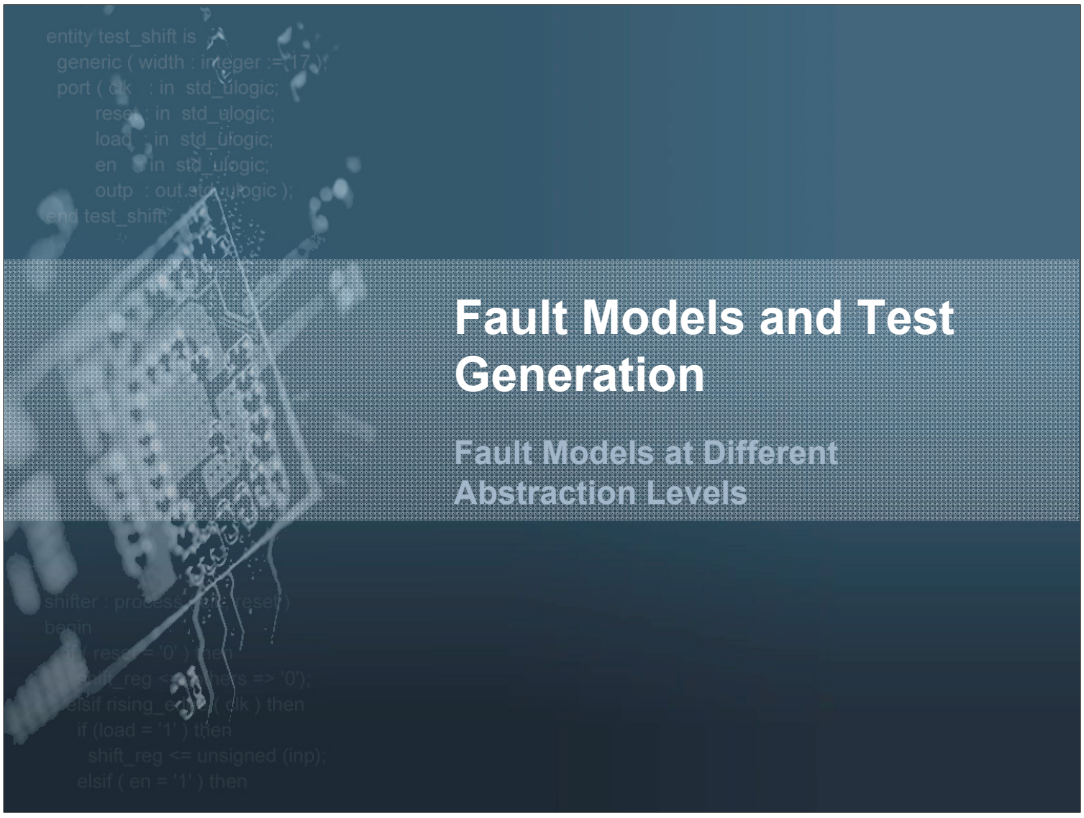
# Introduction

- In order to alleviate the test generation complexity, one needs to model the actual defects that may occur in a chip with fault models at higher levels of abstraction.
- This process of fault modeling considerably reduces the burden of testing because it obviates the need for deriving tests for each possible defect.
- This is made possible by the fact that many physical defects map to a single fault at the higher level.
- This, in general, also makes the fault model more independent of the technology.
- We begin this lecture with a description of the various levels of abstraction at which fault modeling is traditionally done.
- These levels are:
  - behavioral,
  - functional,
  - structural,
  - switch-level and
  - geometric.
- We will present various fault models at the different levels of the design hierarchy and discuss their advantages and disadvantages.

# Levels of Abstraction in Circuits

- Circuits can be described at various levels of abstraction in the design hierarchy (see Figure on the right).
- A **behavioral description** of a digital system is given using a hardware description language, such as VHDL or Verilog. It depicts the data and control flow.
- Increasingly, the trend among designers is to start the synthesis process from a behavioral description.
- A **functional description** is given at the register-transfer level (RTL). This description may contain registers, modules such as adders and multipliers, and interconnect structures such as multiplexers and busses.
- This description is sometimes the product of **behavioral synthesis** which transforms a behavioral description into an RTL circuit.
- A **structural description** is given at the logic level. It consists of logic gates, such as AND, OR, NOT, NAND, NOR, XOR, and interconnections among them. This is the most commonly used description level.
- A **switch-level description** establishes the transistor-level details of the circuit. In CMOS technology, each logic gate is described using an interconnection of a pMOS and an nMOS network. These networks themselves consist of an interconnection of several transistors.
- A **geometric description** is given at the layout level. From this description, one can determine line widths, inter-line and inter-component distances, and device geometries.






# Fault Models and Test Generation

## Fault Models at Different Abstraction Levels

# Fault Models at Different Abstraction Levels

- **Fault modeling** is the process of modeling defects at higher levels of abstraction in the design hierarchy.
- The advantage of using a fault model at the lowest level of abstraction is that it closely corresponds to the actual physical defects, and is thus more accurate.
- However, the sheer number of defects that one may have to deal with under a fault model at this level may be overwhelming. For example, a chip made of 50 million transistors could have more than 500 million possible defects.
- Therefore, to reduce the number of faults and, hence, the testing burden, one can go up in the design hierarchy, and develop fault models which are perhaps less accurate, but more practical.
- In fact, a good strategy may be to first derive tests for fault models at higher levels, and then determine what percentage of faults at the lower levels are covered by these tests.
- Fault models have been developed at each level of abstraction, i.e.,
  - behavioral,
  - functional,
  - structural,
  - switch-level, and
  - geometric.

# Behavioral Fault Models

- **Behavioral fault models** are defined at the highest level of abstraction.
- They are based on the behavioral specification of the system.
- For example, if a digital system is described using a hardware description language, such as VHDL or Verilog, one could inject various types of faults in this description.
- The collection of these faults will constitute a behavioral fault model.
- Precisely what types of faults are included in a behavioral fault model depends on the ease with which they allow detection of realistic faults at the lower levels of abstraction.

# Functional Fault Models

- **Functional fault models** are defined at the functional block level.
- They are geared towards making sure that the functions of the functional block are executed correctly.
- In addition, they should also make sure that unintended functions are not executed.
- For example, for a block consisting of random-access memory (RAM), one type of functional fault we may want to consider is when one or more cells are written into, other cells also get written into.
- This type of fault is called ***multiple writes***.
- Various other types of functional faults can also be defined for RAMs.
- For a microprocessor, a functional fault model can be defined at the instruction level or RTL.

# Structural Fault Models

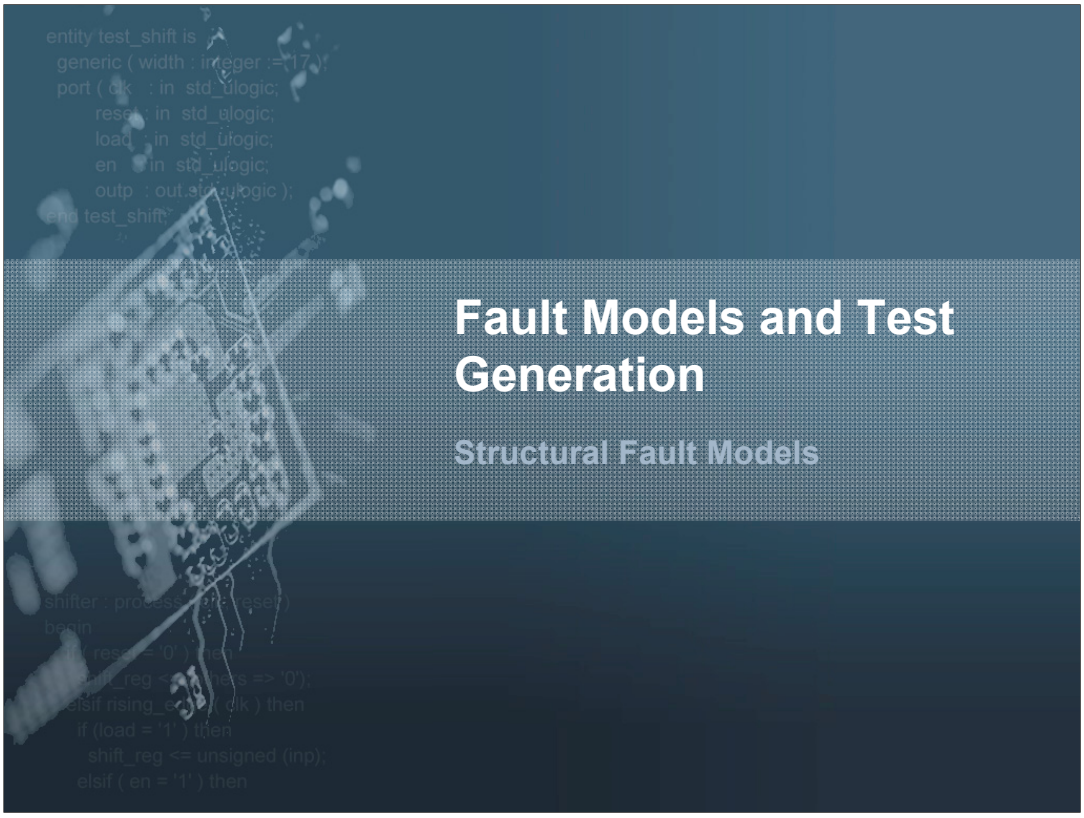
- **Structural fault models** assume that the structure of the circuit is known.
- Faults under these fault models affect the *interconnections* in this structure.
- The most well-known fault model under this category is the *single* ***stuck-at*** fault model.
- This is the most widely used fault model in the industry. Indeed, because of its longevity, it is sometimes termed the classical fault model.
- Its popularity depends on the fact that it can be applied to various semiconductor technologies, and that detection of all single stuck-at faults results in the detection of a majority of realistic physical defects (in many cases, up to 80–85% of the defects are detected).
- However, the current trend is towards augmenting this fault model with other fault models which allow detection of defects that the stuck-at fault model is unable to cover.

# Switch-Level Fault Models

- **Switch-level fault models** are defined at the transistor level.
- The most prominent fault models in this category are the *stuck-open* and *stuck-on* fault models.
- If a transistor is permanently non-conducting due to a fault, it is considered to be ***stuck-open***.
- Similarly, if a transistor is permanently conducting, it is considered to be ***stuck-on***.
- These fault models are specially suited for the CMOS technology.

# Geometric Fault Models

- **Geometric fault models** assume that the layout of the chip is known.
- For example, knowledge of line widths, inter-line and inter-component distances, and device geometries are used to develop these fault models.
- At this level, problems with the manufacturing process can be detected.
- The layout information, for example, can be used to identify which lines or components are most likely to be shorted due to a process problem.
- The ***bridging*** fault model thus developed leads to accurate detection of realistic defects.
- With shrinking geometries of very large scale integrated (VLSI) chips, this fault model will become increasingly important.




# Fault Models and Test Generation

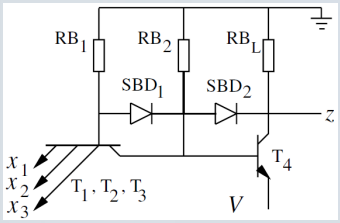
## Structural Fault Models

# Structural Fault Models I

- In structural testing, we need to make sure that the interconnections in the given structure are fault-free and are able to carry both logic 0 and 1 signals.
- The stuck-at fault (SAF) model is directly derived from these requirements.
- A line is said to be ***stuck-at 0*** (SA0) or ***stuck-at 1*** (SA1) if the line remains fixed at a low or high voltage level, respectively (assuming positive logic).
- An SAF does not necessarily imply that the line is shorted to ground or power line.
- It could be a model for many other cuts and shorts internal or external to a gate.
- For example, a cut on the stem of a fanout may result in an SA0 fault on all its fanout branches.
- However, a cut on just one fanout branch results in an SA0 fault on just that fanout branch.
- Therefore, SAFs on stems and fanout branches have to be considered separately.

# Structural Fault Models II

- We next show how the SAF model can detect many realistic defects. Consider the TTL NAND gate shown in figure on the right.
- Fault modeling for opens and shorts in such a gate was done and the results of this modeling are shown in table on the right.
- In the left column, various possible defects are given. In the right column, the SAF, which models the defects, is shown.
- Of these defects, *T*1, *T*2 and *T*3 base–emitter shorts do not directly map to the corresponding SAF, yet are detected when the SAF is. All other detectable defects directly map to the corresponding SAF.
- The defects which are labeled undetectable cannot be detected just by monitoring the logic value at output *z*.
- However, they may be detectable through other means. For example, the defect *RB*2 open, which significantly increases the delay of the gate, can be detected if we measure the delay of the circuit.
- From the table, it is clear that the SAF model can detect a large number of defects, yet the table also points to the need for augmenting this fault model with other fault models if we want a still higher coverage of defects.



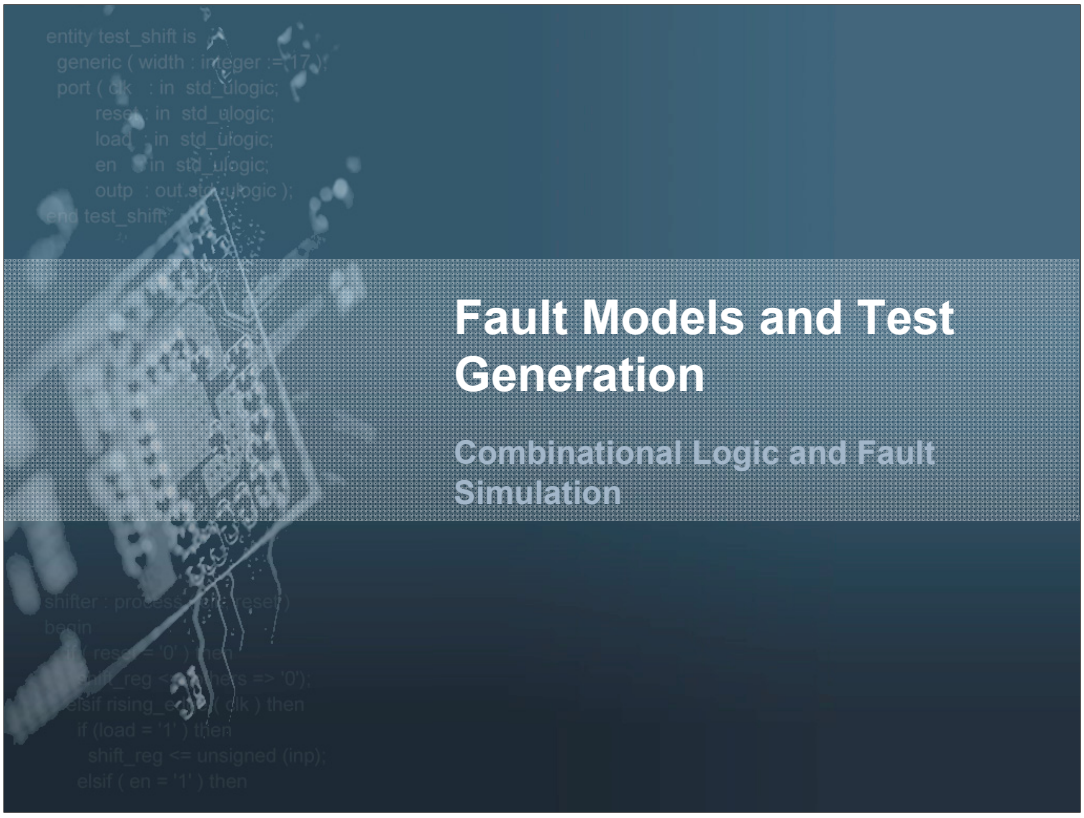

A three-input TTL NAND gate

| Physical defect | Defect detection |
|---|---|
| $RB_1$ open | $z$ SA0 |
| $RB_2$, $R_L$, $SBD_1$, $SBD_2$ open, $T_2$, $T_3$ base open | undetectable |
| $T_1$ emitter open, base open, base–emitter short | $x_1$ SA1 |
| $T_2$ emitter open, base–emitter short | $x_2$ SA1 |
| $T_3$ emitter open, base–emitter short | $x_3$ SA1 |
| $T_4$ emitter open, base open, collector open, base–emitter short | $z$ SA1 |
| $T_1$, $T_2$, $T_3$ collector open | $z$ SA0 |
| $T_1$, $T_2$, $T_3$ collector–emitter short | undetectable |
| $T_4$ collector–emitter short, collector-base short | $z$ SA0 |
| $T_1$, $T_2$, $T_3$ collector-base short | $z$ SA0 |
| $T_2$, $T_3$ base open | undetectable |

Defect detection in the TTL NAND gate

# Structural Fault Models III

- If the SAF is assumed to occur on only one line in the circuit, it is said to belong to the **single SAF model**.
- Otherwise, if SAFs are simultaneously present on more than one line in the circuit, the faults are said to belong to the **multiple SAF model**.
- If the circuit has $k$ lines, it can have $2k$ single SAFs, two for each line.
- However, the number of multiple SAFs is $3^k - 1$ because there are three possibilities for each line (SA0, SA1, fault-free), and the resultant $3^k$ cases include the case where all lines are fault-free.
- Clearly, even for relatively small values of $k$, testing for all multiple SAFs is impossible.




# Fault Models and Test Generation

## Combinational Logic and Fault Simulation

# Introduction I

- The objectives of fault simulation include
  - determination of the quality of given tests, and
  - generation of information required for *fault diagnosis* (i.e., location of faults in a chip).
- We will only describe fault simulation techniques for combinational circuits.
- While most practical circuits are sequential, they often incorporate the full-scan design-for-testability (DFT) feature (that will be discussed later).
- The use of full-scan enables test development and evaluation using only the combinational parts of a sequential circuit, obtained by removing all flip-flops and considering all inputs and outputs of each combinational logic block as primary inputs and outputs, respectively.
- If test vectors are applied using the full-scan DFT features and the test application scheme described later, the reported test quality is achieved.

# Introduction II

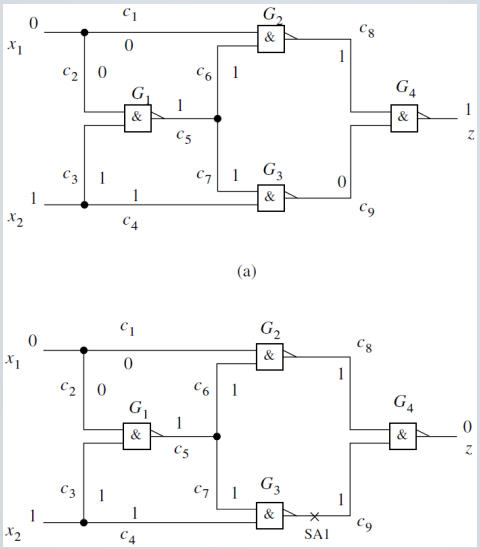
- Let $C$ be a combinational logic circuit with $n$ primary inputs, $x_1,x_2,...,x_n$, and $m$ primary outputs, $z_1,z_2,...,z_m$.
- We assume that the circuit is an interconnect of single-output gates where each gate implements an arbitrary logic function.
- The output of a gate is assumed to
  - directly drive the input of another gate,
  - drive the inputs of several gates via one or more *fanout systems*, or
  - be a primary output of the circuit.
- Let $c_1, c_2, \ldots, c_k$ be the internal lines of the circuit, where $c_j$ is either the output of a single-output gate, $G$, or a fanout of another line (in either case, the line must not be a primary output).
- The primary inputs, internal lines, and primary outputs of the circuit will be collectively referred to as **circuit lines**. Since most lines in any large circuit are internal lines, when it is clear from the context, we will use symbols such as $c_i$ to also refer to all circuit lines.
- When it is necessary to distinguish between all circuit lines and internal lines, we will use symbols such as $b$ and $b_i$ to denote all circuit lines and symbols such as $c$ and $c_i$ to denote internal lines.
- Let $L$ denote the total number of lines in the circuit including the primary inputs, internal lines, and primary outputs, i.e., $L = n + k + m$.

# Introduction III

- For a combinational circuit, a forward traversal – via gates and fanout systems – from the output of a gate cannot reach any of that gate's inputs.
- Let $F = \{f_1, f_2, \ldots, f_{Nf}\}$ be the **fault list**, i.e., the set of faults of interest. We will only consider fault lists that contain all possible single stuck-at faults (SAFs) in the circuit or some subset thereof.
- Let $C^{fi}$ denote a copy of circuit $C$ that has fault $f_i$.
- Let $P$ be an $n$-tuple $(p_1, p_2, \ldots, p_n)$, where $p_j \in \{0, 1\}$, $1 \leq j \leq n$. Each $p_j$ is called a **component** of the $n$-tuple.
- An $n$-tuple $P$ whose component values, $p_1, p_2, \ldots, p_n$, are to be applied, respectively, to the primary inputs $x_1, x_2, \ldots, x_n$ of a circuit is called an **input vector** or, simply, a **vector**. A vector is sometimes also called a **pattern**.
- A vector $P$ is said to **detect** or **test** fault $f_i$ in circuit $C$ if the response obtained on at least one of the outputs of $C$, say $z_j$, due to the application of $P$ at its inputs, is the complement of that obtained at the corresponding output of $C^{fi}$.
- Any vector that detects a fault $f_i$ in a circuit $C$ is said to be a **test vector** or a **test pattern** or, simply, a **test**, for $f_i$ in $C$. If no test vector exists for a fault, then the fault is said to be **untestable**.

# Introduction IV

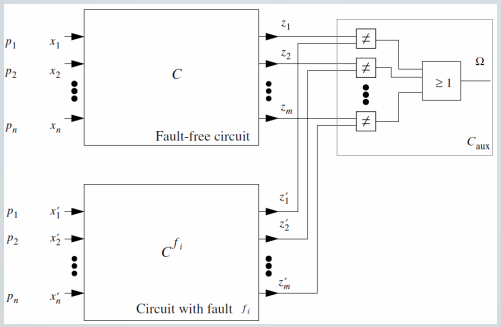
- Consider the circuit shown in Figure (a) where, for each line *b*, the value *v(b)* obtained due to the application of input vector *P* = *(0, 1)* is shown next to the line.
- In other words, vector *P* **implies** value *v(b)* at line *b*.
- Next, consider a faulty version of the above circuit that contains fault $f_{22}$, a stuck-at 1 (SA1) fault at line $c_9$, as shown in Figure (b).
- Also shown in this figure are the values $v^{f22}(b)$ implied by the same input vector, *P* = *(0, 1)*, at each line of the faulty circuit, $C^{f22}$.
- It can be seen that vector *P* is a test for this fault, since the output response to *P* for the faulty circuit is the complement of that for the fault-free circuit.




An example illustrating a test vector:
(a) the fault-free circuit, and
(b) (b) a faulty version of the circuit

# Introduction V

- Figure on the right illustrates the conditions under which a vector $P$ is a test. This figure depicts the fault-free circuit $C$ as well as the faulty circuit $C^{fi}$.
- The component values of vector $P$ are applied to the corresponding inputs of $C$ as well as $C^{fi}$.
- An **auxiliary circuit**, $C_{aux}$, that uses $m$ two-input XOR gates and an $m$-input OR gate, combines the values obtained at the outputs of $C$ with those obtained at the corresponding outputs of $C^{fi}$ into a single logic signal, Ω.
- Since the output of a two-input XOR gate is logic 1 if and only if its inputs have complementary logic values, the output of the $j^{th}$ XOR gate is logic 1 if and only if vector $P$ implies complementary values at the $j^{th}$ outputs of the good and faulty circuits, $z_j$ and $z'_j$ , respectively.
- Hence, a logic 1 appears at the output of the auxiliary circuit, , if and only if vector $P$ is a test for fault $f_i$ in circuit $C$.




Representation of the conditions under which a vector $P = (p_1, p_2, \ldots, p_n)$ is a test for a fault $f_i$ in a circuit $C$

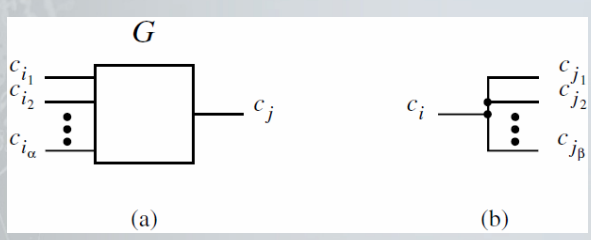
# Introduction VI

- One objective of fault simulation is to determine the faults in the fault list that are detected by the vectors in a given set.
- In some application scenarios, one of the outputs of a fault simulator is the set of faults detected by the given set of vectors.
- In other scenarios, only the number of faults detected may be reported.
- **Fault coverage** of a set of vectors is the number of faults in the fault list detected by its constituent vectors.
- Fault coverage is often reported as a percentage of the total number of faults in the fault list.
- A second objective of fault simulation is to determine, for each vector in the given set of vectors, the values implied at the outputs for each faulty version of the circuit.

# Model of a Combinational Circuit I

- Any gate-level description of a combinational logic circuit can be viewed as an interconnection of two types of **circuit elements** shown in Figure below, namely
  - a **single-output gate** and
  - a **fanout system**.

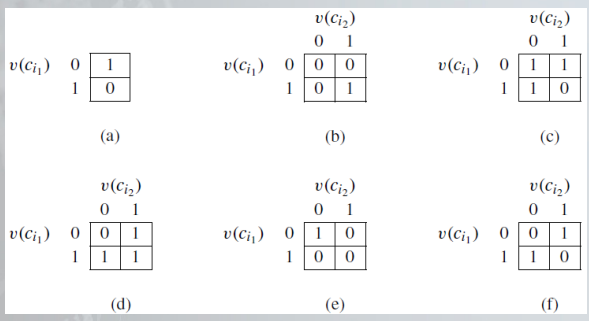
- We use a circuit model in which a gate may have an arbitrary number of inputs and may implement any arbitrary logic function of these inputs.




The basic circuit elements: (a) a single-output gate, (b) a fanout system

# Modeling the Behavior of Gates

- The behavior of a fault-free gate can be described using a two-valued truth table.
- Figure below shows the two-valued truth tables for two-input primitive gates and the XOR gate.
- The logic values associated with inputs $c_{i1}$ and $c_{i2}$ and output $c_j$ will be stored in variables $v(c_{i1})$, $v(c_{i2})$, and $v(c_j)$, respectively.
- Depending on the context, variables such as $v(c_{il})$ will represent variables associated with, or the values assigned to, lines $c_{il}$.




Two-valued truth tables describing the behavior of commonly used gates:
(a) NOT,
(b) two-input AND,
(c) two-input NAND,
(d) two-input OR,
(e) two-input NOR, and
(f) two-input XOR

# Modeling the Behavior of Fanout Systems

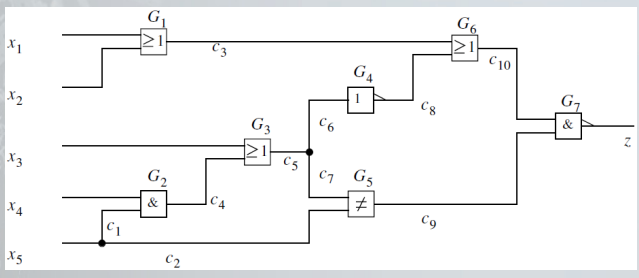
- Now consider the other circuit element, namely a fanout system with input $c_i$ and outputs $c_{j1}$, $c_{j2}$, . . . , $c_{j\beta}$.
- Input $c_i$ and each output $c_{jl}$ are called the **stem** and a **branch** of the fanout system, respectively.
- Figure below shows the two-valued truth-table for each fanout branch.

| $v(c_i)$ | $v(c_{j_1})$ | $\cdots$ | $v(c_{j_l})$ | $\cdots$ |
|---|---|---|---|---|
| 0 | 0 | $\cdots$ | 0 | $\cdots$ |
| 1 | 1 | $\cdots$ | 1 | $\cdots$ |

Two-valued truth tables describing the behavior of a fanout system

# Model of a Combinational Circuit II

- Any combinational logic circuit can be implemented as an interconnection of the two types of circuit elements:
  - gates and
  - fanout systems.
- Figure below shows an example circuit, $C_{16}$, which will be used in the following discussion to illustrate the definitions of some terms.
- Our circuit model allows only single-output gates. If the output of a gate drives inputs of multiple gates, then one or more fanout systems are used to explicitly model the connections.
- For example, since output $c_5$ of gate $G_3$ drives inputs of multiple gates, a fanout system is used to connect $c_5$ to the inputs of gates it drives.




An example circuit *C*16

# Description of Faulty Circuit Elements

- The single SAF model has been used in practice for decades with reasonable success. Thus, it provides a good target for fault simulation and test generation.
- A single SAF in a faulty circuit is associated with a specific circuit line, called the **fault site**. The fault site may be a gate input, a gate output, a fanout stem, or a fanout branch.
- Note that a circuit line may fall into more than one of the above categories.
- For example, in the circuit shown on Slide 21, *c*5 is the output of gate *G*1 as well as the stem of a fanout system; *c*9 is the output of gate *G*3 as well as an input of gate *G*4.
- If any such line is the site of a fault, the fault can be associated with either of the circuit elements.
- Two facts define the behavior of an SAF.
  - First, logic value *w* is associated with the line that is the site of an SA*w* fault, independent of the value implied at the line by the input vector.
  - Second, the SAF is assumed to not affect the lines that are not in the transitive fanout of the fault site.
  - This implies, for example, that an SA0 fault at line *c*6 in Figure on Slide 21 has no affect on the value at line *c*5. Hence, an input vector may imply a logic 1 at line *c*5 which in turn may imply a logic 1 at line *c*7. Similarly, an SA1 fault at *c*9 has no affect on the inputs of gate *G*3, namely, lines *c*4 and *c*7.

# Behavior of a Gate with an SAF at its Output

- The two-valued truth table of a two-input NAND gate whose output is SA0 is shown in Figure (a).
- The truth table of the gate with output SA1 fault is shown in Figure (b).
- Since the presence of an SAF at the output of a gate causes it to have a constant value, each truth table shown in Figure represents the behavior of any two-input gate with the corresponding SAF at its output, independent of the logic function implemented by the fault-free version of the gate.

| $v(c_{i_1})$ \ $v(c_{i_2})$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |

(a)

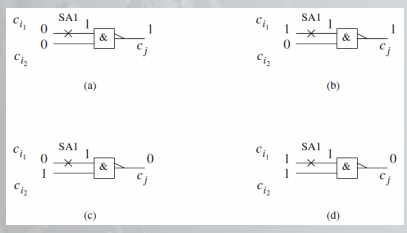
| $v(c_{i_1})$ \ $v(c_{i_2})$ | 0 | 1 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 1 |

(b)

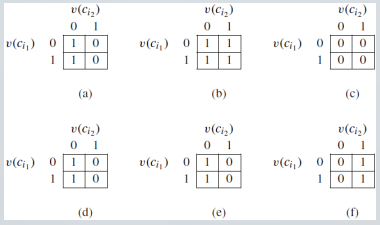
The two-valued truth tables of a two-input gate with an SAF at its output: (a) SA0, and (b) SA1

# Behavior of a Gate with an SAF at one of its Inputs

- Consider an SA1 fault at one of the inputs, say $c_{i1}$, of a two-input NAND gate with inputs $c_{i1}$ and $c_{i2}$ and output $c_j$.
- Figure on the left shows the gate with the SA1 fault under different input vectors. Due to the presence of the fault at $c_{i1}$, the value at the corresponding input is always interpreted as a logic 1 by the NAND gate, independent of the value applied to line $c_{i1}$.
- This is indicated by symbol 1 as shown in Figure (a) on the left, on the portion of line $c_{i1}$ between the mark indicating the fault and the NAND gate.
- This fact about the behavior of the fault can be used, in conjunction with the two-valued truth table of the fault-free version of a two-input NAND gate, to obtain the two-valued truth table of the faulty gate, shown in the Figure on the right.
- Two-valued truth tables of some other faulty gates are also shown in the Figure on the right.




A two-input NAND gate with SA1 fault at one of its inputs




Two-valued truth tables of two-input gates with SAFs at input *ci*1: (a) NAND with SA1, (b) NAND with SA0, (c) NOR with SA1, (d) NOR with SA0, (e) XOR with SA1, and (f) XOR with SA0

# Behavior of a Faulty Fanout System

- The above approach can be used to derive the truth tables of a fanout system shown in Figure on the Slide 26 with SAFs at its inputs and outputs.
- Figure on the right shows some of the cases.

| $v(c_i)$ | $v(c_{j_1})$ | $v(c_{j_2})$ | $\cdots$ | $v(c_{j_l})$ | $\cdots$ |
|---|---|---|---|---|---|
| 0 | 1 | 1 | $\cdots$ | 1 | $\cdots$ |
| 1 | 1 | 1 | $\cdots$ | 1 | $\cdots$ |

(a)

| $v(c_i)$ | $v(c_{j_1})$ | $v(c_{j_2})$ | $\cdots$ | $v(c_{j_l})$ | $\cdots$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | $\cdots$ | 0 | $\cdots$ |
| 1 | 0 | 0 | $\cdots$ | 0 | $\cdots$ |

(b)

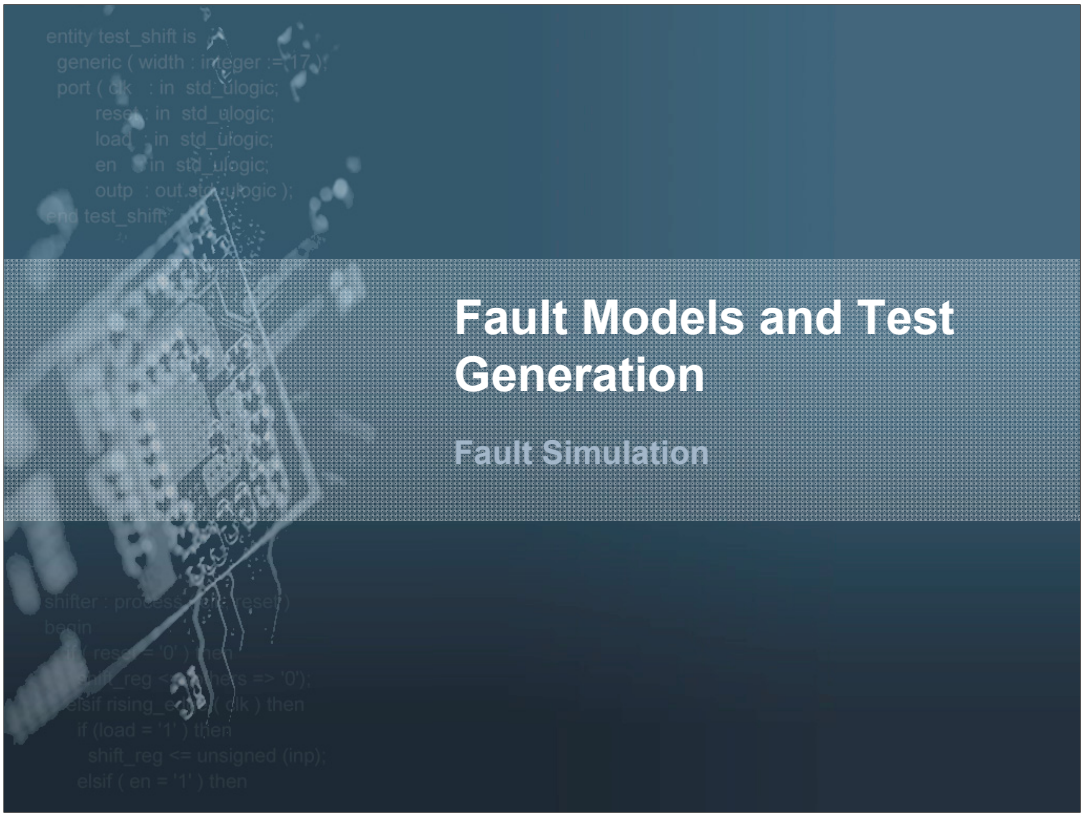
| $v(c_i)$ | $v(c_{j_1})$ | $v(c_{j_2})$ | $\cdots$ | $v(c_{j_l})$ | $\cdots$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | $\cdots$ | 1 | $\cdots$ |
| 1 | 1 | 1 | $\cdots$ | 1 | $\cdots$ |

(c)

| $v(c_i)$ | $v(c_{j_1})$ | $v(c_{j_2})$ | $\cdots$ | $v(c_{j_l})$ | $\cdots$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | $\cdots$ | 0 | $\cdots$ |
| 1 | 1 | 1 | $\cdots$ | 0 | $\cdots$ |

(d)

Two-valued truth tables of a fanout system with various single SAFs: (a) $c_i$ SA1, (b) $c_i$ SA0, (c) $c_{jl}$ SA1, and (d) $c_{jl}$ SA0




# Fault Models and Test Generation

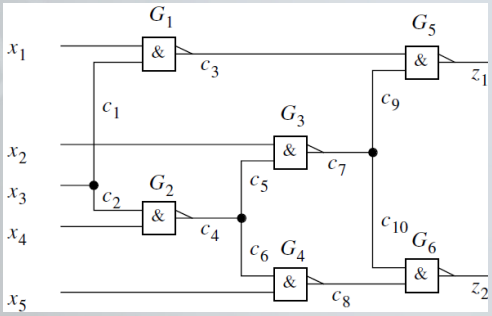
## Fault Simulation

# Logic Simulation

- Accurate simulation of logic values implied at circuit lines due to the application of a vector, as well as timing of each logic value transition and hazard caused, requires consideration of logic behavior as well as delays of each constituent logic element.
- In the case of synchronous sequential circuits, however, logic simulation is often performed primarily to determine the steady-state logic values implied at circuit lines by each vector being simulated. The temporal aspects of the circuit behavior, including times of occurrence of transitions and hazards and widths of hazards, are the purview of timing analysis.
- During test generation and fault simulation, it is even more common to focus exclusively on the steady-state logic behavior of the circuit under consideration.
- A logic or fault simulation algorithm can be used to simulate each combinational logic block of a synchronous sequential circuit. The value at each output of a combinational logic block computed above is what will be latched at the next appropriate clock event into the latch/flip-flop driven by the output in the sequential circuit, *provided the delay of the circuit is within the requirements posed by the clocking strategy employed*.
- Such simulation algorithms are sometimes called *cycle-based* simulators. They are also called *zero-delay* simulators, since they do not consider delay values.
- **Logic simulation** for a combinational logic circuit is defined as the determination of steady-state logic values implied at each circuit line by the vector applied to its primary inputs.

# A Simple Logic Simulation Algorithm I

- In its simplest form, a zero-delay logic simulation algorithm for a combinational logic block reads an input vector $P = (p_1, p_2, \ldots, p_n)$, assigns each component $p_i$ of the vector to the corresponding primary input $x_i$, and computes the new logic value implied at each line.
- These three steps are repeated for each vector for which simulation is desired.
- In the above simple algorithm, the value at an output of a circuit element should not be computed until the value implied by the current vector at each input of the circuit element has been computed.
- For example, in the circuit shown in Figure below, the value at line $c_7$, which is the output of gate $G_3$, should be computed only after the values implied by the current vector at inputs of $G_3$, namely lines $x_2$ and $c_5$, have been computed.



# A Simple Logic Simulation Algorithm II

**SimpleLogicSimulation( )**

1. Preprocessing - Compute the input level $\eta_{inp}(c_i)$ of each line $c_i$ in the circuit as well as to obtain $Q_{\eta inp}$, the ordered list of circuit lines.

2. While there exists a vector to be simulated, read a vector $P = (p_1, p_2, \ldots, p_n)$.
   a. For each primary input $x_i$, assign $v(x_i) = pi$.
   b. In the order in which lines appear in $Q_{\eta inp}$, for each line $c_i$ in $Q_{\eta inp}$, compute the value $v(c_i)$. Use appropriate equations if $c_i$ is the output of a gate or if it is a branch of a fanout system.

- The complexity of the above algorithm for simulation of one vector is $O(L)$, where $L$ is the number of lines in the circuit.

# A Simple Fault Simulation Algorithm

- A very straightforward fault simulation algorithm can be obtained by repeated use of any logic simulation algorithm. For each vector, simulation of the fault-free version of the circuit, $C$, can be followed by simulation of each of the faulty versions of the circuit, $C^{f_i}$, where $f_i$ is a fault in the fault list.
- Simulation of $C^{f_i}$ is identical to that of $C$ with the exception that the value at each output of the circuit element that is the site of fault $f_i$ is computed using functions that capture the behavior of the circuit element in the presence of fault $f_i$.
- The above approach is embodied in the following simple fault simulator.

**SimpleFaultSimulation( )**

For each vector $P$

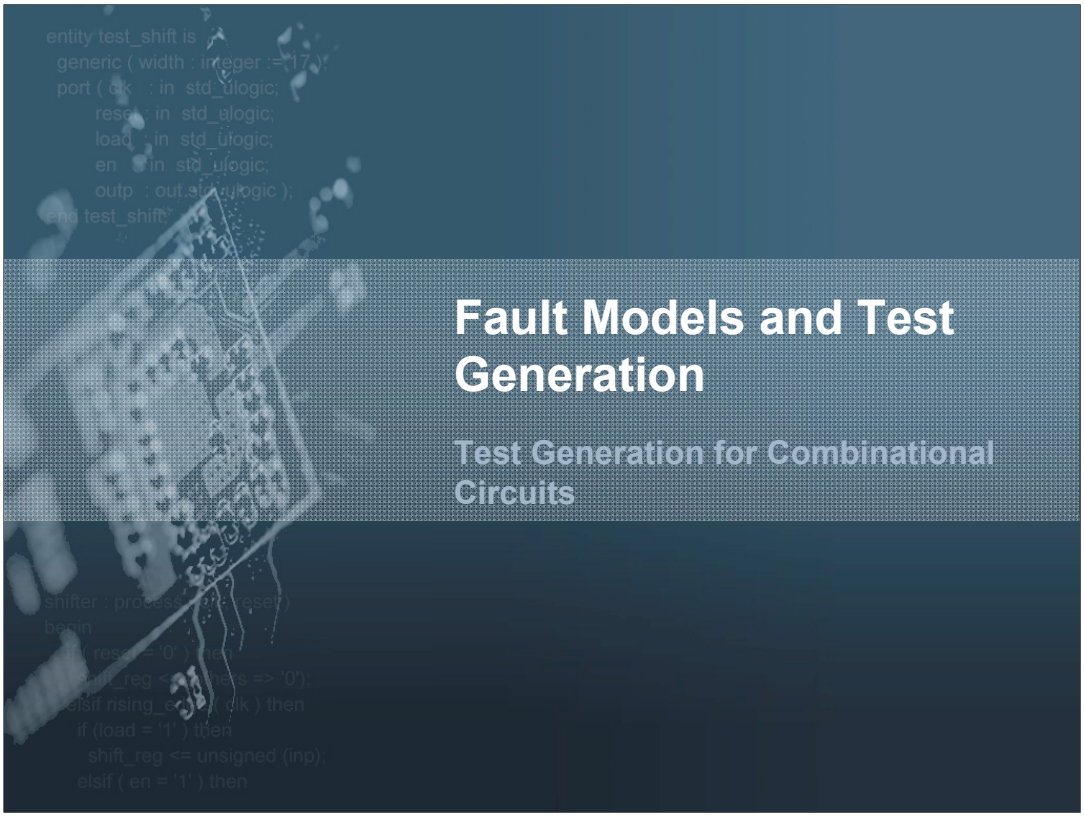
a. Simulate the fault-free version of circuit $C$ and compute $v(c_j)$ for all lines $c_j$.

b. For each fault $f_i \in F$

   i. Simulate $C^{f_i}$, the faulty version of the circuit with fault $f_i$, to compute value $v^{f_i}(c_j)$ for each line $c_j$.

   ii. If $v^{f_i}(z_l)$ is the complement of $v(z_l)$ for any output $z_l$, then mark fault $f_i$ as detected.

- Since the number of single SAFs is proportional to $L$, the number of lines in a circuit, the complexity of the above fault simulation approach is $O(L^2)$.




# Fault Models and Test Generation

## Test Generation for Combinational Circuits

# Introduction I

- *Let F* = {$f_1$, $f_2$, ... , $f_{Nf}$} denotes the fault list, i.e., the set of faults of interest. We will consider fault lists that contain all possible single stuck-at faults (SAFs) in the given circuit, or some subset thereof. $C^{fi}$ denotes a copy of circuit *C* that has fault $f_i$.
- During a given execution of a test generation procedure, typically a single fault is considered. This fault will be called the **target fault** and the corresponding faulty version of the circuit will sometimes be referred to as *C'*.
- Recall that a vector *P* is an *n*-tuple ($p_1$, $p_2$, ... , $p_n$), where $p_j$ € {0, 1}, 1 ≤ *j* ≤ *n*. Each $p_j$ is called a component of the vector. Component values of vector *P*, $p_1$, $p_2$, ... , $p_n$, are applied, respectively, to primary inputs $x_1$, $x_2$, … , $x_n$ of the circuit.
- The value implied by the vector at line *c* in the fault-free circuit *C* is denoted by *v(c)*. The value implied by the vector at line *c* in the faulty version with fault $f_i$, $C^{fi}$, is denoted by $v^{fi}(c)$. In particular, the value implied at line *c* in the faulty version with the target fault, *C*, is simply referred to as *v'(c)*.
- A vector *P* is said to **detect**, **test**, or **cover** fault $f_i$ in circuit *C* if the value implied at one or more of the outputs of *C*, due to the application of *P* at its inputs, is different from that obtained at the corresponding outputs of $C^{fi}$.
- Any vector that detects a fault $f_i$ in a circuit *C* is said to be a **test vector** or, simply, a **test**, for that fault in *C*. If no test vector exists for a fault, then the fault is said to be **untestable**.

# Introduction II

- In its simplest form, the objective of test generation is to generate a test vector for a given fault in a given circuit, or to declare it untestable.
- Typically, a set of tests must be generated within a given amount of computational effort. Hence, for some faults, test generation may be terminated before it is successful, i.e., before a test is generated or the fault proven untestable. Such a fault is sometimes called an **aborted** fault.
- A more practical version of test generation requires the generation of ***a set of test vectors whose constituent vectors collectively detect all, or a maximal fraction of, the testable faults in the given fault list***. In other words, for a given amount of computational effort, the number of aborted faults must be minimized.
- Another version of test generation requires the above objective to be satisfied using ***a test set of minimum size***, i.e., one containing a minimum number of test vectors.
- Yet another version requires the detection of all, or a maximal fraction of, the faults in the target fault list using ***a test sequence whose application to the circuit causes minimal heat dissipation***.
- In all the cases, the test generator also reports:
  - (a) the fault coverage obtained by the generated tests,
  - (b) a list containing each fault that was identified as being untestable, and
  - (c) a list containing each aborted fault.

# Test Generation Basics

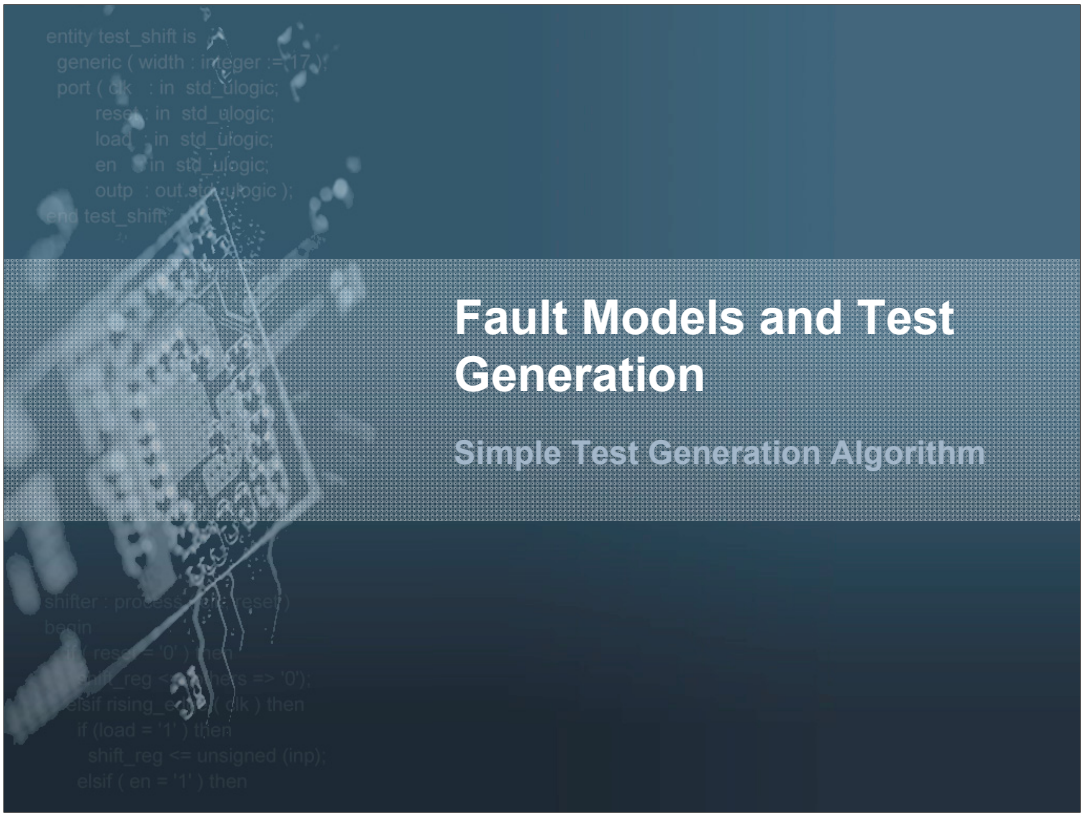
- Two key concepts, namely, ***fault effect excitation*** (FEE) and ***fault effect propagation*** (FEP), form the basis of all methods to search for test vectors.
- However, the search for a test for a fault can – in the worst case – have exponential run-time complexity.
- Hence, a large number of additional concepts have been developed to make test generation practical.
- **Fault effect excitation** is the process of creating a fault effect at one or more outputs of the faulty circuit element.
- **Fault effect propagation** is the process of assigning values to circuit lines such that a fault-effect propagates from an output of the faulty circuit element to a primary output of the circuit.
- All test generation algorithms can be divided into two groups:
  - Structural test generation algorithms
  - Non-structural test generation algorithms

# Structural Test Generation Algorithms

- Structural test generation algorithms analyze the ***structure*** of the given CUT to generate a test for a given target fault, or declare it untestable.
- A test generation *system* uses a test generation algorithm and other procedures, such as random vector generators, fault simulators, and test set compactors, to obtain a set of vectors that efficiently detect all target faults in a given fault list (or a large proportion thereof).
- Test generation for a given target fault is a search process that is said to be **successful** if either a test is generated for the fault or the fault is proven to be untestable.
- A test generation algorithm is said to be **complete** if it can guarantee that test generation will be successful for any given target fault in any given CUT, assuming sufficient memory is available and sufficient run-time is allowed.
- The **efficiency** of a test generation algorithm is loosely measured by the proportion of all target faults for which it is successful and the corresponding run-time.
- All structural test generation algorithms discussed ahead are **complete**.
- Most popular structural test generation algorithms are:
  - D-algorithm
  - PODEM

# Non-structural Test Generation Algorithms

- Structural algorithms for test generation directly and continually analyze the gate-level description of a circuit and implicitly enumerate all possible input combinations to find a test vector for a target fault.
- In contrast, an algebraic algorithm converts the test generation problem into an algebraic formula and applies algebraic techniques to simplify and then solve the formula to obtain a test.
- Most popular non-structural test generation algorithms are:
  - Test generation based on satisfability
  - Test generation using binary decision diagrams (BDDs)



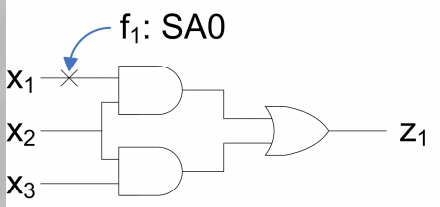

# Fault Models and Test Generation

## Simple Test Generation Algorithm

## Truth-table based Method for Finding Tests for Stuck-at Faults

- To find tests for some stuck-at fault $f_i$:
  - Calculate responses, $z_k$, of the fault-free circuit $C$, for all input assignments.
  - Calculate circuit responses, $z^{fi}_k$, of the faulty circuit $C^{fi}$, for all input assignments.
  - All input assignments for which $z_k \neq z^{fi}_k$, at least for one $k$, are tests for the fault $f_i$.

# Example




Test for $f_1$

| $x_1$ | $x_2$ | $x_3$ | $z_1$ | $z^{f1}_1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

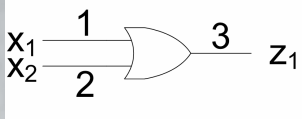
# Techniques to Obtain a Compact Set of Tests

- Two types of technique are used to obtain a compact test set for a given CUT and fault list.
- In the first type, first a test generation algorithm is used to generate a set of vectors that detects all the faults in the list, except for faults that were proven to be untestable or for which test generation failed.
- From here onwards, a test generation algorithm is not used but the tests in the generated test set are manipulated to reduce its size.
- Such techniques are called **static test compaction** techniques, since the test set is manipulated without the benefit of generating new tests.
- The other class of techniques, called **dynamic test compaction**, may use a test generator to generate alternative tests with properties that facilitate compaction.

## How to Find a Complete Minimal Test Set

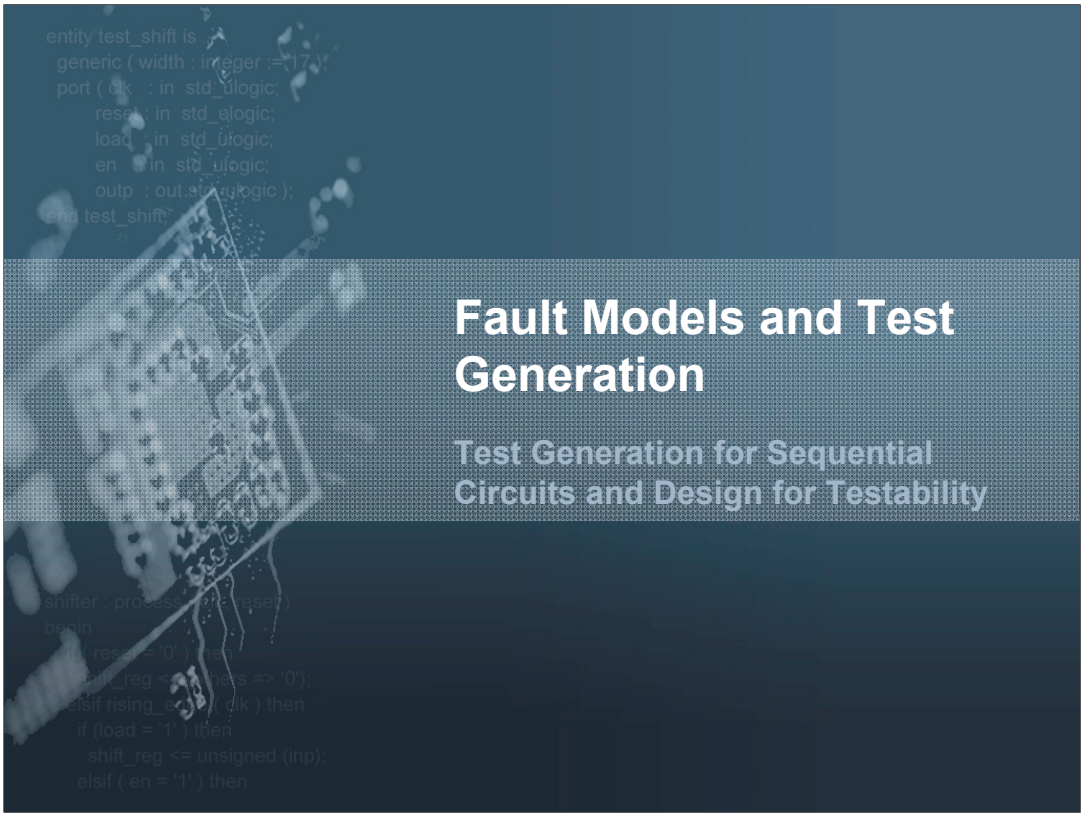
- Find all tests for all stuck-at faults in the circuit
- Make a table
  - One row for each fault (2·number of lines)
  - One column for each test ($2^n$, n = number of inputs)
- Put a star is a test detects a fault
- Select a minimal number of tests which detect all faults (i.e. choose a minimal subset of columns which covers all rows)

# Example




| | 00 | 01 | 10 | 11 |
|---|---|---|---|---|
| 1: SA0 | | | * | |
| 1: SA1 | * | | | |
| 2: SA0 | | * | | |
| 2: SA1 | * | | | |
| 3: SA0 | | * | * | * |
| 3: SA1 | * | | | |

- The complete minimal test set is {(00),(01),(10)}




# Fault Models and Test Generation

## Test Generation for Sequential Circuits and Design for Testability

# Test Generation for Sequential Circuits

- Sequential automatic test pattern generation (ATPG) is a difficult problem.
- The many challenges we face in this area include:
  - reduction in the time and memory required to generate the tests,
  - reduction in the number of cycles needed to apply the tests to the circuit, and
  - obtaining a high fault coverage.
- In spite of the excellent progress made in the last decade in this area, it is usually not feasible to satisfactorily test an entire chip using sequential ATPG.
- It needs to be aided in this process through ***testability insertion*** described next.

# Design for Testability (DFT) I

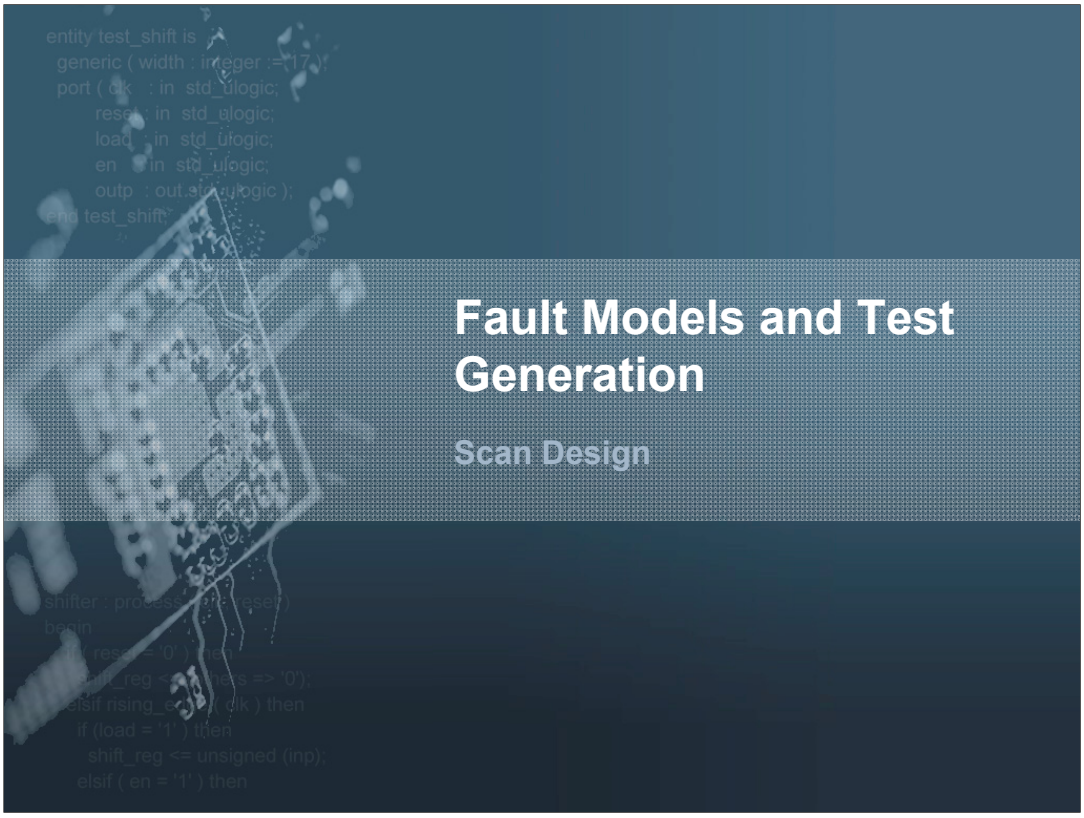
- The difficulty of testing a digital circuit can be quantified in terms of cost of *test development*, cost of *test application*, and costs associated with *test escapes*.
- **Test development** spans circuit modeling, test generation (automatic and/or manual), and fault simulation. Upon completion, test development provides test vectors to be applied to the circuit and the corresponding fault coverage.
- **Test application** includes the process of accessing appropriate circuit lines, pads, or pins, followed by application of test vectors and comparison of the captured responses with those expected.
- The cost associated with a high **test escape**, i.e., when many actual faults are not detected by the derived tests, is often reflected in terms of loss of customers.
- Even though this cost is often difficult to quantify, it influences the above two costs by imposing a suitably high fault coverage requirement to ensure that test escape is below an acceptable threshold.
- **Design for testability (DFT)** can loosely be defined as changes to a given circuit design that help decrease the overall difficulty of testing.
- The changes to the design typically involve addition or modification of circuitry such that one or more new modes of circuit operation are provided.

# Design for Testability (DFT) II

- Each new mode of operation is called a **test mode** in which the circuit is configured only for testing.
- During normal use, the circuit is configured in the **normal mode** and has identical input-output logic behavior as the original circuit design. However, the timing of the circuit may be affected by the presence of the DFT circuitry.
- The key objective of DFT is to reduce the difficulty of testing. ATPG techniques for sequential circuits are expensive and often fail to achieve high fault coverage.
- The main difficulty arises from the fact that the state inputs and state outputs cannot be directly controlled and observed, respectively.
- In such cases, a circuit may be modified using the *scan* design methodology, which creates one or more modes of operation which can be used to control and observe the values at some or all flipflops.
- This can help reduce the cost of test development. In fact, in many cases, use of some scan is the only way by which the desired fault coverage target may be achieved.
- However, in most cases, the use of scan can increase the test application time and hence the test application cost. In a slightly different scenario, DFT circuitry may be used to control and observe values at selected lines within the circuit using *test points* so as to reduce the number of test vectors required. In such a scenario, use of DFT may help reduce the test application cost.

# Design for Testability (DFT) III

- Next, consider a typical board manufacturing scenario where pre-tested chips are assembled onto a printed circuit board. Assuming that the chips were not damaged during assembly, assembled boards must be tested to ensure correctness of interconnects between chips.
- One possible way to test these interconnects is via application of tests and observation of responses at, respectively, the primary inputs and outputs of the board, commonly referred to as a board's **connector**.
- In this method, the circuit under test (CUT) is the aggregate of the circuitry in each chip on the board and all inter-chip interconnections. Clearly, in such a scenario, test development is prohibitively expensive.
- Test development may in fact be impossible due to the unavailability of the details of the circuit within each chip.
- Another alternative is to use physical *probes* to obtain direct access to the input and output pins of chips to test the interconnect.
  - The first difficulty in such an approach is that a value must be applied to a pin via a probe in such a manner that it does not destroy the driver of the pin.
  - Second, the cost of such a physical probing can be very high, especially since in packaging technologies, device pins are very close to each other.
  - Finally, in many new packaging technologies, such as *ball-grid array*, many of the 'pins' are located across the entire bottom surface of the chip and cannot even be probed.
- The ***boundary scan*** DFT technique provides a mechanism to control and observe the pins of a chip directly and safely. Once again, DFT not only reduces costs of test development and application, it makes high test quality achievable.

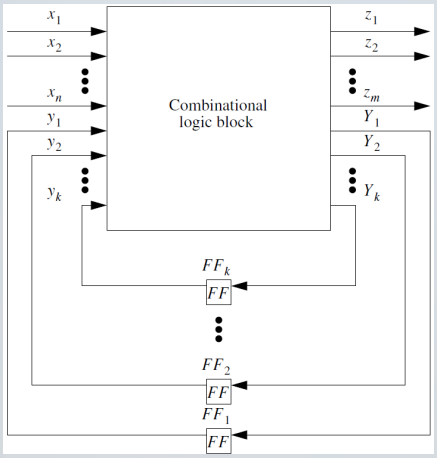



# Fault Models and Test Generation

## Scan Design

# Scan Design I

- Consider the sequential circuit shown in Figure on the right, comprised of a block of combinational circuit and a set of $k$ flip-flops.
- The primary inputs and outputs of the sequential circuit are $x_1, x_2, \ldots, x_n$, and $z_1, z_2, \ldots, z_m$, respectively.
- The present state variables, $y_1, y_2, \ldots, y_k$, constitute the **state inputs** of the combinational circuit.
- The next state variables, $Y_1, Y_2, \ldots, Y_k$, constitute the **state outputs** of the combinational circuit.
- $Y_l$ and $y_l$ are, respectively, the input and the output of flip-flop $FF_l$, $1 \le l \le k$.
- Test development for sequential circuits is difficult mainly due to the inability to control and observe the state inputs and outputs, respectively.
- In fact, for many sequential circuits, the cost of test development is so high that the fault coverage attained in practice is unacceptably low.



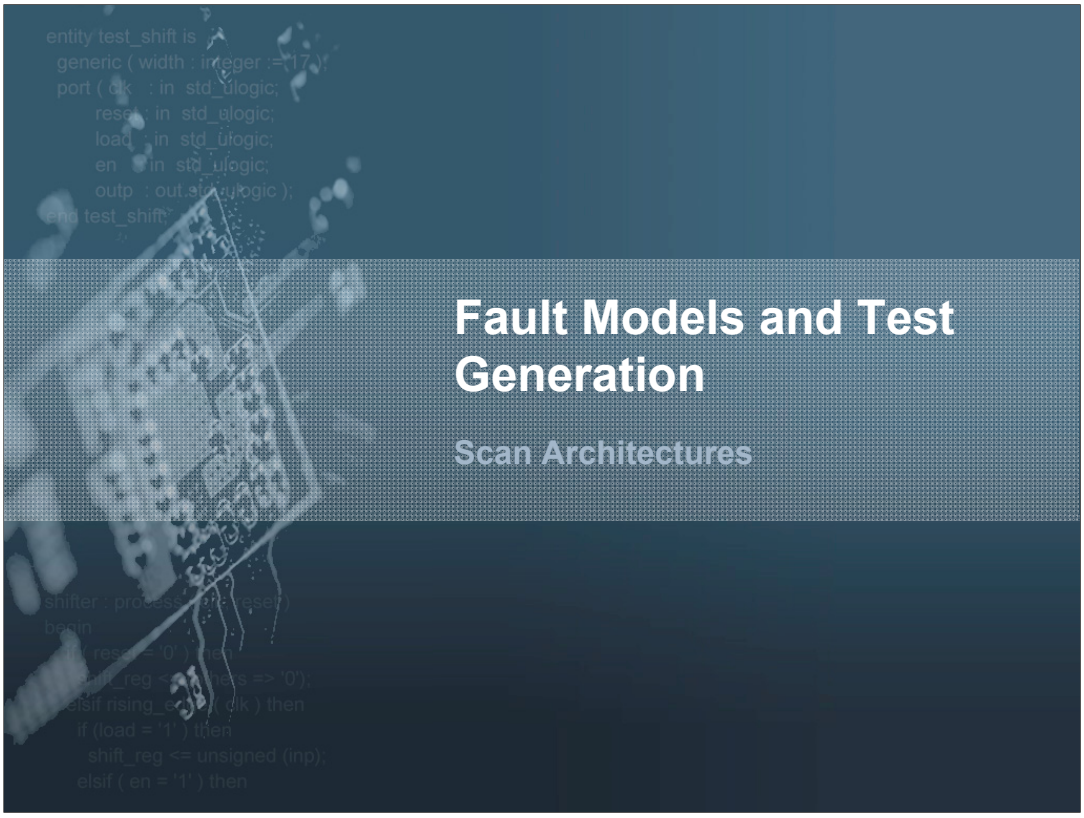

A general model of a sequential circuit

# Scan Design II

- Scan design is currently the most widely used structured DFT approach.
- It is implemented by connecting selected storage elements of a design into one or more shift registers, called scan chains, to provide them with external access.
- Scan design accomplishes this task by replacing all selected storage elements with scan cells, each having one additional scan input (SI) port and one shared/additional scan output (SO) port.
- By connecting the SO port of one scan cell to the SI port of the next scan cell, one or more scan chains are created.
- The scan-inserted design, called scan design, is now operated in three modes: ***normal mode***, ***shift mode***, and ***capture mode***.
- Circuit operations with associated clock cycles conducted in these three modes are referred to as normal operation, shift operation, and capture operation, respectively.
- In ***normal mode***, all test signals are turned off, and the scan design operates in the original functional configuration.
- In both ***shift*** and ***capture*** modes, a test mode signal TM is often used to turn on all test-related fixes in compliance with scan design rules.

# Scan Design III

- In general, combinational logic in a full-scan circuit has two types of inputs:
  - primary inputs (PIs) and
  - pseudo primary inputs (PPIs).
- Primary inputs refer to the external inputs to the circuit, whereas pseudo primary inputs refer to the scan cell outputs.
- Both PIs and PPIs can be set to any required logic values. The only difference is that PIs are set directly in parallel from the external inputs, whereas PPIs are set serially through scan chain inputs.
- Similarly, the combinational logic in a full-scan circuit has two types of outputs:
  - primary outputs (POs) and
  - pseudo primary outputs (PPOs).
- Primary outputs refer to the external outputs of the circuit, whereas pseudo primary outputs refer to the scan cell inputs.
- Both POs and PPOs can be observed. The only difference is that POs are observed directly in parallel from the external outputs, whereas PPOs are observed serially through scan chain outputs.




# Fault Models and Test Generation

## Scan Architectures

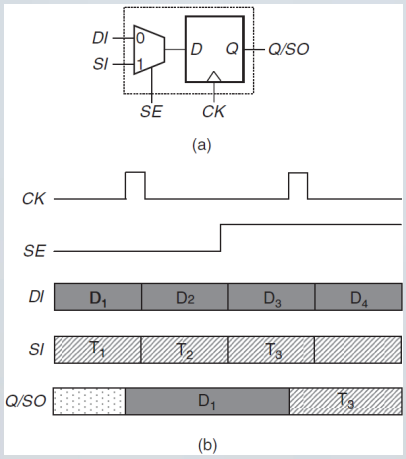
## Scan Architectures

- In this subsection, we first describe a few fundamental scan architectures.

- These fundamental scan architectures include:
  1. muxed-D scan design, in which storage elements are converted into muxed-D scan cells,
  2. clocked-scan design, in which storage elements are converted into clocked-scan cells, and
  3. LSSD scan design, in which storage elements are converted into level-sensitive scan design (LSSD) shift register latches (SRLs).

# Muxed-D Scan Design I

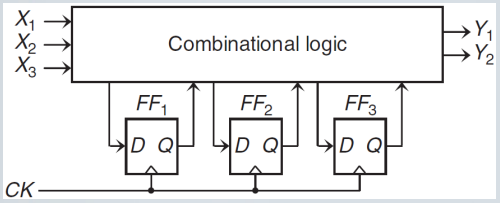
- An edge-triggered muxed-D scan cell design is shown in Figure (a) on the right.
- This scan cell is composed of a D flip-flop and a multiplexer.
- The multiplexer uses a scan enable (SE) input to select between the data input (DI ) and the scan input (SI ).
- In normal/capture mode, SE is set to 0. The value present at the data input DI is captured into the internal D flip-flop when a rising clock edge is applied.
- In shift mode, SE is set to 1. The scan input SI is now used to shift in new data to the D flip-flop, while the content of the D flip-flop is being shifted out.
- Sample operation waveforms are shown in Figure (b).



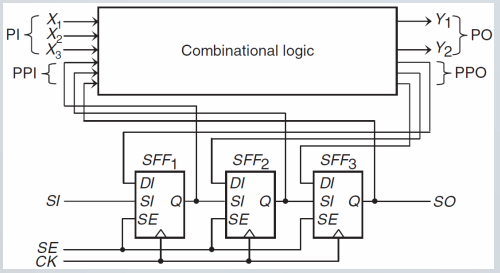

Edge-triggered muxed-D scan cell design and operation: (a) Muxed-D scan cell. (b) Sample waveforms.

# Muxed-D Scan Design II

- The three D flip-flops, $FF_1$, $FF_2$, and $FF_3$, shown in Figure above, are replaced with three muxed-D scan cells, $SFF_1$, $SFF_2$, and $SFF_3$, respectively, shown in Figure below.
- The data input DI of each scan cell is connected to the output of the combinational logic as in the original circuit.
- To form a scan chain, the scan inputs SI of $SFF_2$ and $SFF_3$ are connected to the outputs Q of the previous scan cells, $SFF_1$ and $SFF_2$, respectively.
- In addition, the scan input SI of the first scan cell $SFF_1$ is connected to the primary input SI, and the output Q of the last scan cell $SFF_3$ is connected to the primary output SO.
- Hence, in shift mode, SE is set to 1, and the scan cells operate as a single scan chain, which allows us to shift in any combination of logic values into the scan cells.
- In capture mode, SE is set to 0, and the scan cells are used to capture the test response from the combinational logic when a clock is applied.




Sequential circuit example




Muxed-D scan design

# Test Generation

- The use of scan allows the desired value to be shifted into each flip-flop, or **scanned in**, using the test mode and scan chains. Each state input of the combinational circuit may hence be viewed as being directly controllable.
- After the application of a test, the values at state outputs may be captured into the flip-flops by configuring them in their normal mode. The values thus captured may be shifted out, or **scanned out**, using the test mode and observed at the corresponding scan output pin, SO.
- The state outputs of the combinational circuit may hence be viewed as being directly observable.
- Hence, only the combinational logic block of the sequential circuit need be considered for test generation, as shown in Figure on Slide 55. Consequently, a test generator for combinational circuits may be used to generate a set of tests for the faults in the sequential circuit.
- Assume that test vectors $P_1$, $P_2$, … , $P_{Nv}$ are generated. Note that the combinational logic block of the sequential circuit has a total of $n + k$ inputs, since the circuit has $n$ primary inputs and $k$ state inputs.
- Hence, a vector $P_i$ has the form $(p_{1,i}, p_{2,i}, \ldots , p_{n,i}, p_{n+1,i}, p_{n+2,i}, \ldots , p_{n+k,i})$.
- Of these, values $p_{1,i}, p_{2,i}, \ldots , p_{n,i}$ are applied to primary inputs $x_1, x_2, \ldots , x_n$, respectively, and will collectively be referred to as the **primary input part** of test vector $P_i$.
- Values $p_{n+1,i}, p_{n+2,i}, \ldots , p_{n+k,i}$ are applied to state inputs $y_1, y_2, \ldots , y_k$, respectively, and will be referred to as the **state input part** of the vector.

# Test Application I

- The primary input part of each vector $P_i$ generated by the combinational test generator can be applied directly to the primary inputs of the sequential circuit.
- In contrast, the state input part of each vector can be applied to the respective state inputs only via the scan chain, as described next.

- Assume that the flip-flops are configured as a single scan chain.
- Also assume that, in the test mode, the SI input is connected to the SI input of $FF_1$, whose output drives the SI input of $FF_2$, and so on.
- Finally, the output of $FF_k$ is connected to the SO line.

- In such a circuit, the following steps are used to apply vector $P_i$:

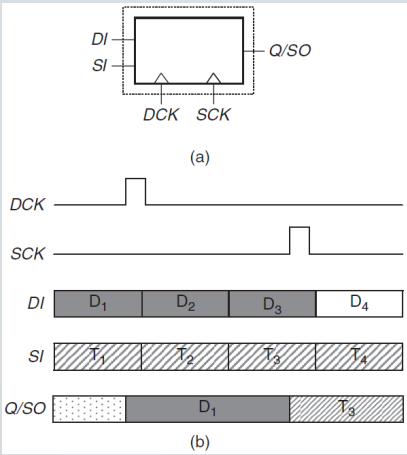
1. The circuit is set into test mode by setting $SE = 1$.
2. For the next $k$ clock cycles, the bits of the state input part of the vector are applied in the order $p_{n+k,i}$, $p_{n+k-1,i}$, ... , $p_{n+2,i}$, $p_{n+1,i}$, at the SI pin. Arbitrary values may be applied to the primary inputs during the above clock cycles. At the end of these cycles, values $p_{n+1,i}$, $p_{n+2,i}$, ... , $p_{n+k,i}$ are shifted into flip-flops $FF_1$, $FF_2$, ... , $FF_k$, respectively, and are hence applied to state inputs $y_1$, $y_2$, ... , $y_k$.
3. Values $p_{1,i}$, $p_{2,i}$, ... , $p_{n,i}$ are then applied to primary inputs $x_1$, $x_2$, ... , $x_n$, respectively. At the end of this step, all bits of vector $P_i$ are applied to the corresponding inputs (primary or state, as appropriate) of the combinational logic block.
4. The circuit is configured in its normal mode by setting $SE = 0$ and one clock pulse is applied. This causes the response at the state outputs of the combinational logic block, $Y_1$, $Y_2$, ... , $Y_k$, to be captured in flip-flops $FF_1$, $FF_2$, ... , $FF_k$, respectively. The response at primary outputs $z_1$, $z_2$, ... , $z_m$ are observed directly at this time.
5. The response captured in the scan flip-flops is scanned out and observed at the SO pin in Steps 1 and 2 when the above procedure is repeated to apply the next test vector, $P_{i+1}$.

# Test Application II

- $K$ clock cycles are required to scan the state input part of each test vector into the flip-flops.
- One clock cycle is then required to apply the vector.
- Finally, $k - 1$ clock cycles are required to scan out the response captured at the flip-flops for the vector.
- Since the response for vector $P_i$ is scanned out at the same time as when the state input part of the next vector, $P_{i+1}$, is scanned in, a total of $N_v(k + 1) + k - 1$ clock cycles are required to apply $N_v$ test vectors.

# Clocked Scan Design I

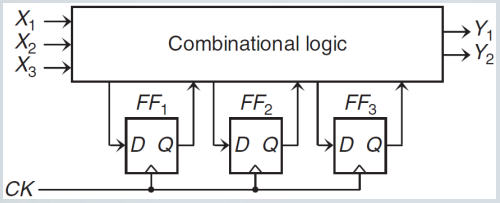
- An edge-triggered clocked-scan cell can also be used to replace a D flip-flop in a scan design.
- Similar to a muxed-D scan cell, a clocked-scan cell also has a data input DI and a scan input SI; however, in the clocked-scan cell, input selection is conducted with two independent clocks, data clock DCK and shift clock SCK, as shown in Figure (a).
- In normal/capture mode, the data clock DCK is used to capture the contents present at the data input DI into the clocked-scan cell.
- In shift mode, the shift clock SCK is used to shift in new data from the scan input SI into the clocked-scan cell, while the content of the clocked-scan cell is being shifted out.
- Sample operation waveforms are shown in Figure (b).



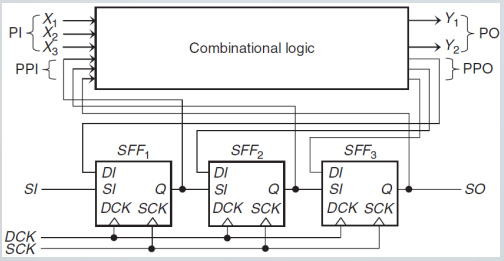

Clock-scan cell design and operation:
(a) Clocked-scan cell.
(b) Sample waveforms

# Clocked Scan Design II

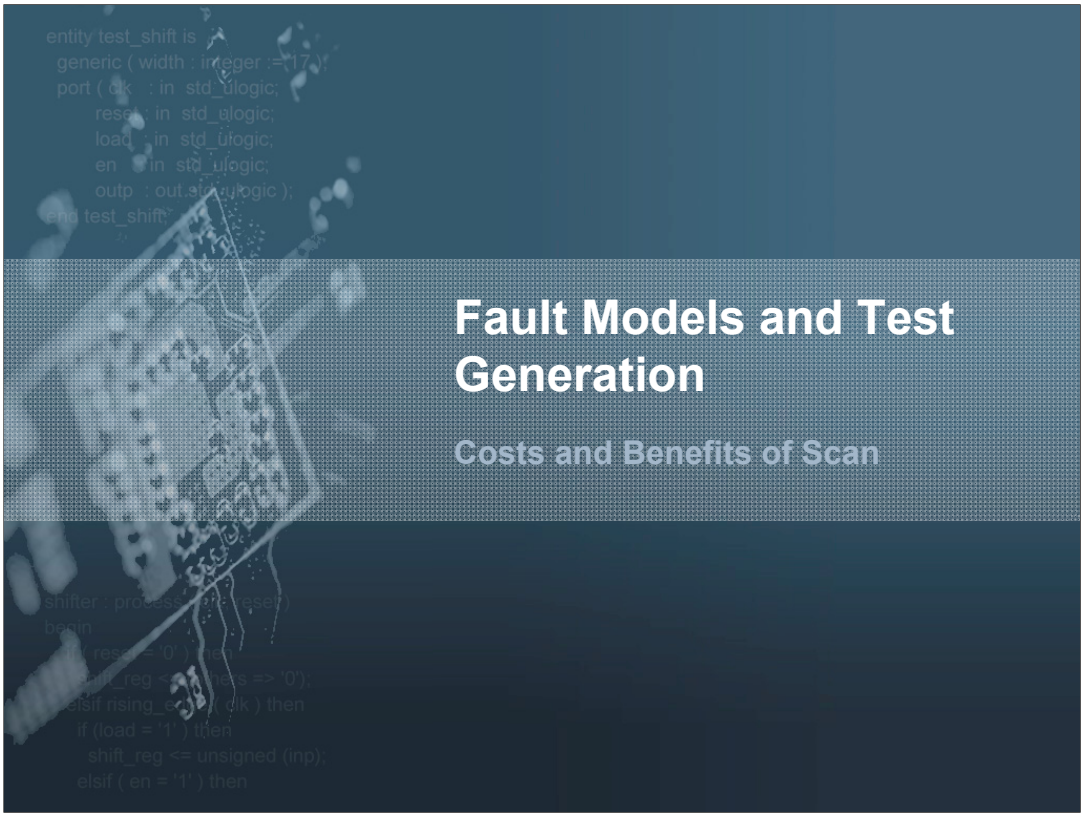
- The major advantage of the use of a clocked-scan cell is that it results in no performance degradation on the data input.
- A major disadvantage, however, is that it requires additional shift clock routing.
- Figure below shows a clocked-scan design of the sequential circuit given in Figure above.
- This clocked-scan design is tested with shift and capture operations, similar to a muxed-D scan design.
- The main difference is how these two operations are distinguished.
- In a muxed-D scan design, a scan enable signal SE is used, as shown in Slide 61.
- In the clocked scan shown in Figure below, these two operations are distinguished by properly applying the two independent clocks SCK and DCK during shift mode and capture mode, respectively.




Sequential circuit example




Clocked-scan design

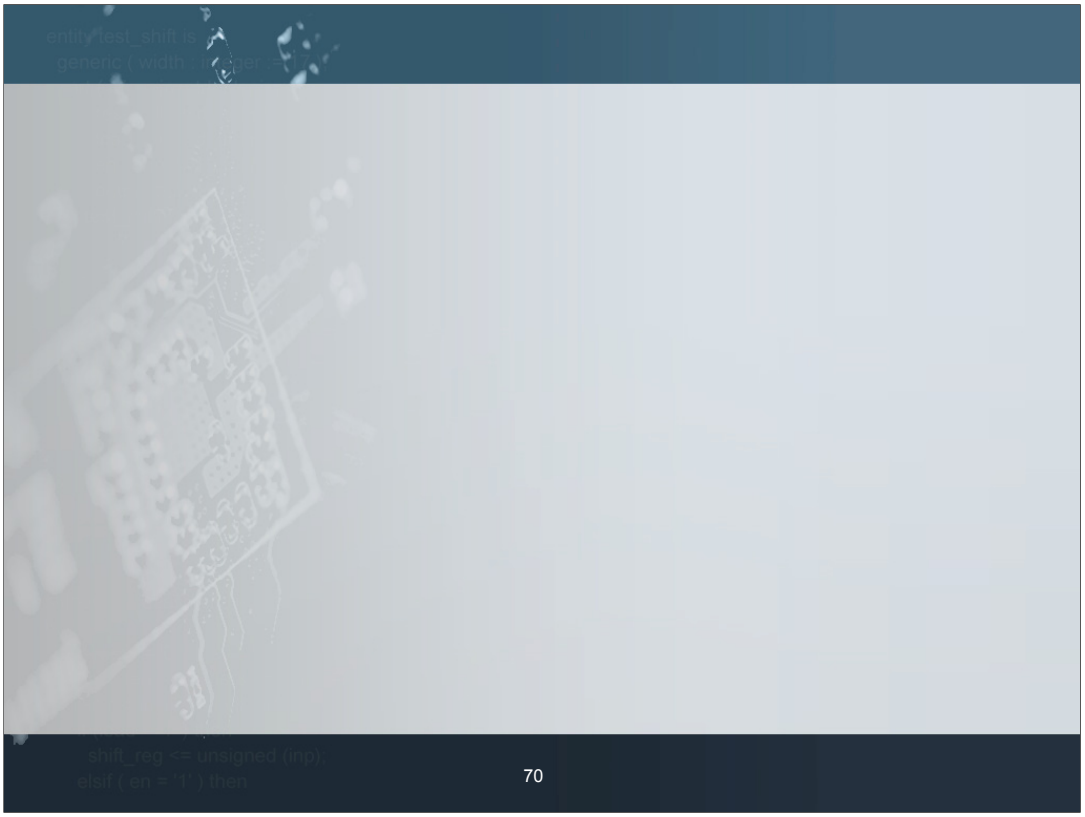

entity test_shift is
generic ( width : integer := 17 );
port ( clk : in std_ulogic;
reset : in std_ulogic;
load : in std_ulogic;
en : in std_ulogic;
outp : out std_ulogic );
end test_shift;
Fault Models and Test Generation
Costs and Benefits of Scan

# Costs and Benefits of Scan I

- Additional logic and routing are required to implement scan. Hence, the layout area of the scan version of a circuit is higher than that of its non-scan counterpart.
- This increase in area, typically called **area overhead**, increases the cost of fabrication of the circuit in two main ways:
  - First, higher layout area means fewer copies of a chip can be manufactured on a given semiconductor wafer.
  - Second, an increase in area is accompanied with a decrease in yield. Hence, a smaller percentage of manufactured chips work.
- If additional logic required for a scan circuit is incorporated in a straightforward manner into the circuit, then additional delay is introduced into the circuit. This can sometimes necessitate a reduction in the rate at which the circuit may be clocked.
- Consider as an example the multiplexed-input scan flip-flop. In this design, one multiplexer is added on many combinational paths in the circuit.
- The impact on clock rate of the circuit can be significant for high-speed circuits, where a combinational logic block contains only a few levels of logic gates.
- Any decrease in the speed of the circuit during normal operation is often referred to as **performance penalty**.
- The third main cost of scan is the need for extra pins required for signals such as *SI*, *SO*, mode control, and test clocks.
- Some of these signals, especially *SI* and *SO*, can be multiplexed with existing primary input and output pins. In the normal mode, these pins function as originally intended; in the test mode, they function as *SI* and *SO*.

# Costs and Benefits of Scan II

- The fourth cost of scan is that, for most circuits, test application time increases.
- The use of scan decreases the number of vectors required to test a circuit, but several clock cycles are required to apply each test vector via scan. In most circuits, this leads to an increase in the overall test application time. Some exceptions exist, however. This cost can be mitigated by using multiple scan chains and other techniques.
- One key benefit of scan is that for many sequential circuits, the use of some amount of scan is the only way to attain an acceptable fault coverage. This is one of the key reasons behind acceptance of scan.
- However, increased controllability and observability provided by scan are also useful for purposes other than test generation and application.
- Scan is used extensively to locate design errors and weaknesses during debugging of *first silicon*, i.e., the first batch of chips fabricated for a new design.
- Also, scan is used to locate failing components when an operational system fails. In such scenarios, scan is used to reduce the time required to repair a system, hence to increase system availability.


entity test_shift is
generic ( width : integer :=
shift_reg <= unsigned (inp);
elsif ( en = '1' ) then
70