The need for testing

- Real systems may have manufacturing defects. Short circuits, missing components, damaged components etc.
- Need to know if a system (board, IC, whole system) has a defect (and therefore doesn't work). Don't want to sell bad systems - don't want to reject good systems.
- $\therefore$ Need for testing is economic.

Two approaches

- Functional testing - does system work correctly?
- Structural testing - does system contain a fault?
- Functional testing can imply a long and difficult task!

Fault Models

- Fault Modelling
  - What Defects occur?
  - How can they be modelled?
  - What do models imply?

PCB Defects

- Breaks in connections
  - bad etching
  - stress
- Short Circuits
  - solder flow
- Bad solder joints

IC Defects

- Open Circuits
  - electromigration
  - current overstress
  - corrosion
- Short Circuits
- Incorrect Transistor Action
  - silicon or oxide defects
  - mask misalignment
  - impurities
  - gamma radiation
- "Latch-up"
  - transient currents
- Data Corruption
  - alpha particles
  - EMI

How do defects manifest themselves?

- Static failures (50%)
  - shorts, breaks etc
- Dynamic failures (49%)
  - out of spec components
  - timing failures
- Intermittent failures (1%)
  - environmental

Fault Model

- Physical Defect manifests itself as a logical fault
  - Applies only to digital circuits
  - (No analogue fault models.)
- Stuck Fault Model
  - Many physical defects can be modelled as a circuit node being:
  - stuck at 1 (s-a-1)
  - stuck at 0 (s-a-0)

Single-Stuck Fault Model

- Assumptions:
  - The fault directly affects only one node
  - The node is stuck at either 0 or 1
- These assumptions make test pattern generation easier
- Validity of Single-Stuck Fault Model?
  - Multiple faults do occur
  - Can multiple faults mask each other?
  - Number of faults might rise with complexity
  - Can all defects be modelled with stuck fault model?
  - The model appears to be valid most of the time
  - Almost all test pattern generation relies on this model
  - Multiple faults are found with test patterns for single faults

Fault-oriented Test Pattern Generation

- Prepare a fault list (e.g. all nodes stuck-at 0 & stuck-at 1)

```
repeat
    write a test
    check fault cover (one test may cover > 1 fault)
    (delete covered faults from list)
 until fault cover target is reached
```

1. Test pattern generation (writing a test) may be random or optimised. Ideally we want a minimum number of tests - cheaper to apply!
2. One test may cover more than one fault, often faults are indistinguishable.
3. If we simply want a pass/fail test, we can remove faults from further consideration, once we have found a test for a fault. If we want to diagnose a fault (for subsequent repair) we probably want to find all tests for a fault to deduce where the fault occurs.
4. Fault cover target may be less than 100%. The higher the cover, the greater the number of tests and hence the cost of applying the test.

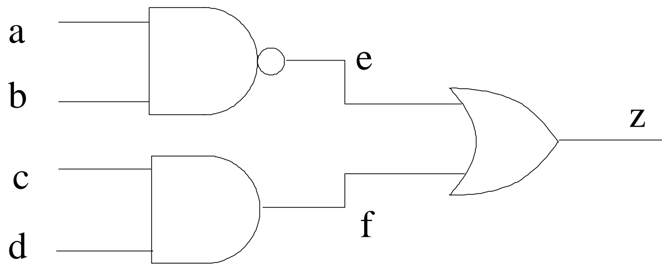
Testability

- How testable is a system?

a. Controllability - can we control all the nodes to establish if there is a fault?

b. Observability - can we observe and distinguish between a faulty node and a correct node?

Sensitive Path Algorithm



- 7 nodes, $\therefore$ 14 stuck faults:
  - a/0, a/1, b/0, b/1, c/0, c/1, d/0, d/1, e/0, e/1, f/0, f/1, z/0, z/1
  - "a stuck-at-0" etc.
- To test for a/0, need to set a to 1 (fault-free condition).
- Need to observe existence or otherwise of fault at z.
- If b is 0, e is 1 irrespective of a. $\therefore$ b must be 1.
- Similarly if f is 1, z is 1, irrespective of e, $\therefore$ f must be 0.
- Thus we are establishing a sensitive path from a to z.
- To force f to 0, either c or d or both must be 0.

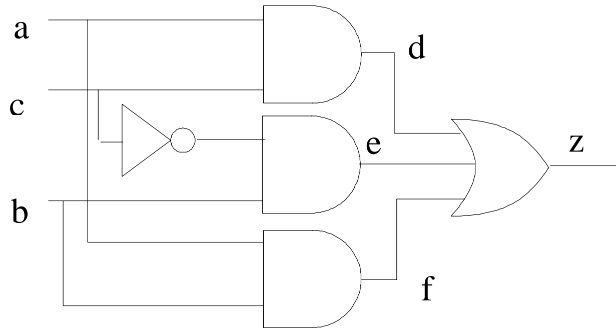
- If the fault a/0 exists, e is 1, z is 1. If the fault does not exist, e is 0, z is 0.
- We can conclude from this that a test for a/0 is a=1, b=1, c=0, d=1, for which the fault-free output is z=0. This can be expressed as 1101/0. Other tests are 1110/0 and 1100/0. Therefore, there is more than one test for the fault a/0.
- To test for e/1, requires that f=0 to make e visible at z. $\therefore$ c or d or both must be 0. To make e=0 requires that a=b=1. So a test for e/1 is 1101/0. This is the same test as for a/0! So one test can cover more than one fault.

Algorithm

1. Select a fault;
2. Set up inputs to force the node to a fixed value;
3. Set up inputs to transmit node value to an output;
4. Check for consistency;
5. Check for coverage of other faults;

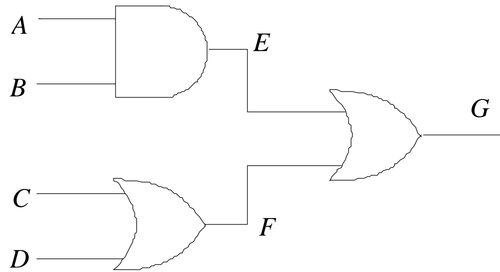
- The aim is to find the minimum number of tests that cover all the possible faults. 100% fault cover may not be possible.
- Fan-out and reconvergence can cause difficulties for this algorithm. Improved algorithms (D-algorithm, PODEM) use similar techniques but overcome drawbacks.

Undetectable Faults



- Consider the function $z = a.c + b.\overline{c}$
- To avoid hazards, the redundant term may be included:
  $z = a.c + b.\overline{c} + a.b$
- To test for f/0, f=1, $\therefore$ a=b=1
- To transmit to z, d=e=0.
- For e=0, b=0 and/or c=1.
- For d=0, a=0 and/or c=0.
- Thus there is an inconsistency!
- Untestable faults are due to redundancy.

Fault simulation



- One test pattern can be used to find more than one potential fault.
- For example, suppose we wish to detect if node *E* is stuck at 0 (*E*/0).
- *E*/0 cannot be distinguished from *G*/0 or *A*/0 or *B*/0.
- In all these cases, *G* will be 1 normally and 0 in the presence of one of these faults.
- Hence, the input pattern $A = 1$, $B = 1$, $C = 0$, $D = 0$ can be used to detect four possible faults.
- As there are 7 nodes in the circuit, there are 14 possible stuck-at faults.
- This pattern covers 4 faults and it can be shown that of the 16 possible input patterns, 6 are sufficient to detect all the possible stuck-at faults in the circuit.

Testing Sequential Circuits

- Testing combinational circuits is relatively easy, provided, there is no redundancy in the circuit. Number of test vectors << $2^{\text{(no. of inputs)}}$
- Testing sequential circuits is difficult because circuits have states. May require long sequences of inputs to reach states. Some faults may be untestable, because certain states cannot be reached.
- N.B. All sequential circuits MUST have a set or reset to initialise all flip-flops or testing is nearly impossible.