life.augmented

Design for Test

FDP University Program 2023

Akhil Garg

Agenda

#1 Introduction to Test & DFT

#2 Yield, defect & faults

#3 Fault models & Test algo.

#4 ATPG

#5 Memory test

Analog test

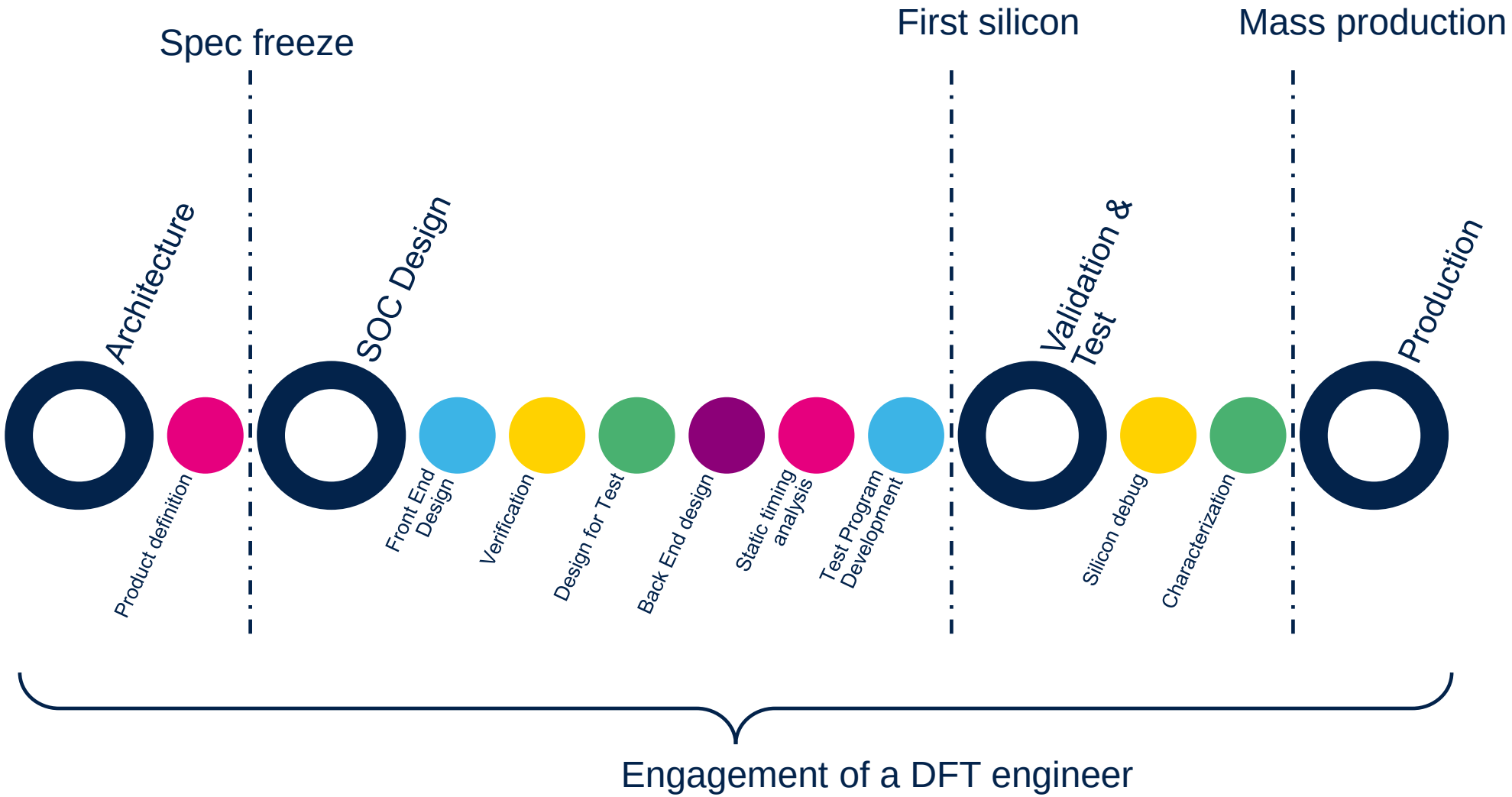
Boundary scan

Topic

life.augmented

Introduction to DFT

ASIC production cycle



life.augmented

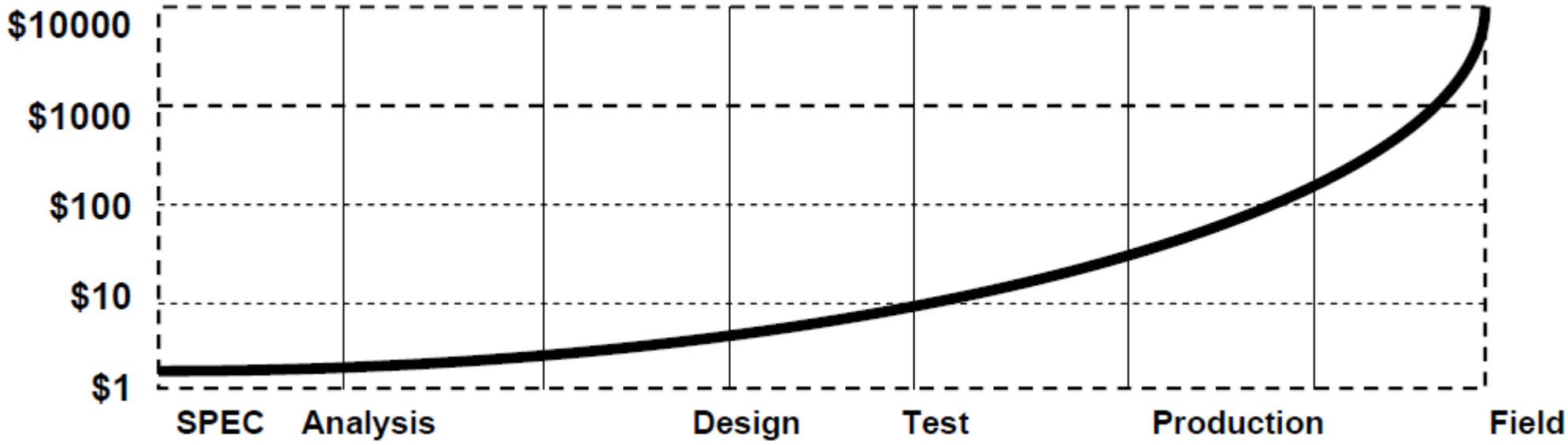
Design for Test

- **Design for Test** (DFT) consists of IC **design** techniques that add **testability** features to a hardware product **design**.

- It eases development and application of manufacturing **tests** to the designed hardware.

Cost of a defect

- Cost to fix a defect / problem grows exponentially through a design cycle.
- Cost of bad part in critical device (for example, a pacemaker or airplane) is immeasurable.



life.augmented

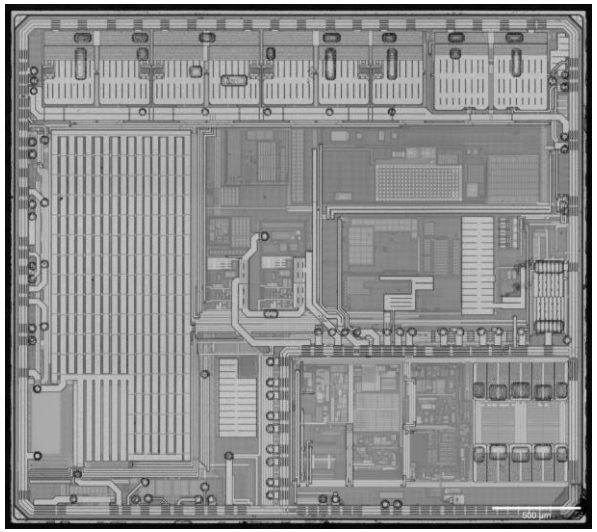
Verification vs Test

- Verification
  - Checks the correctness of design
  - Verifies if the implemented design is as per the defined specifications.
  - Performed using simulation / formal methods.
  - Performed on the design prior to manufacturing
  - Ensures quality of the design.
- Test
  - Checks correctness of the manufactured Hardware.
  - Checks if the part is manufactured as per design.
  - Types of test
    - Digital
    - Analog
    - Memory
    - IOs
  - Performed on each manufactured part
  - Ensures quality of the product.

Bugs vs Defects

Physical defect

Defect:
Defect is an on-chip **flaw** caused during fabrication or packaging of an ASIC resulting in a logical **malfunction**.



- Typical Examples

✓ Short circuit
✓ Open circuit
✓ Unconnected Via
✓ Oxide pin holes transistor always on

Some examples

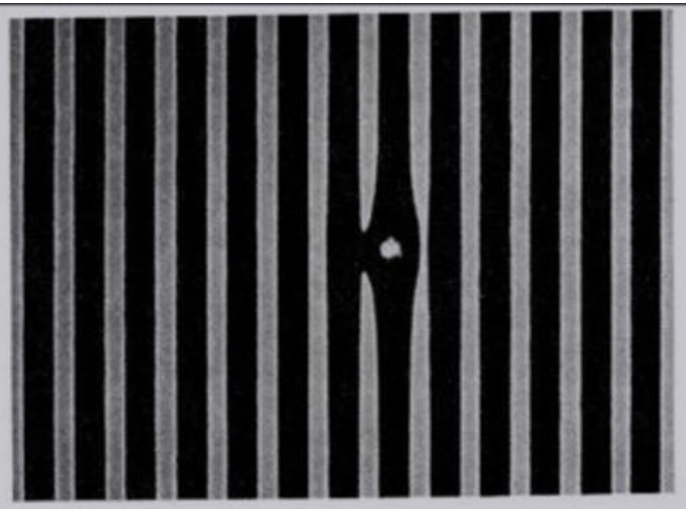


Figure 1: Example of a break in a metal line.

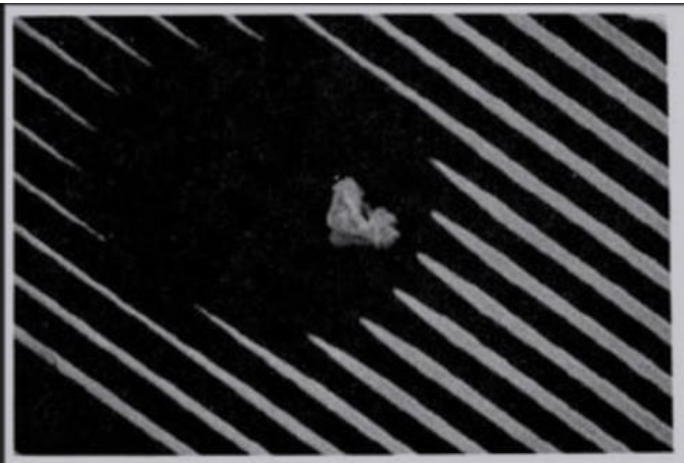


Figure 2: Example of a break of 7 metal lines caused by the interaction between a contaminating particle and the photoresist.

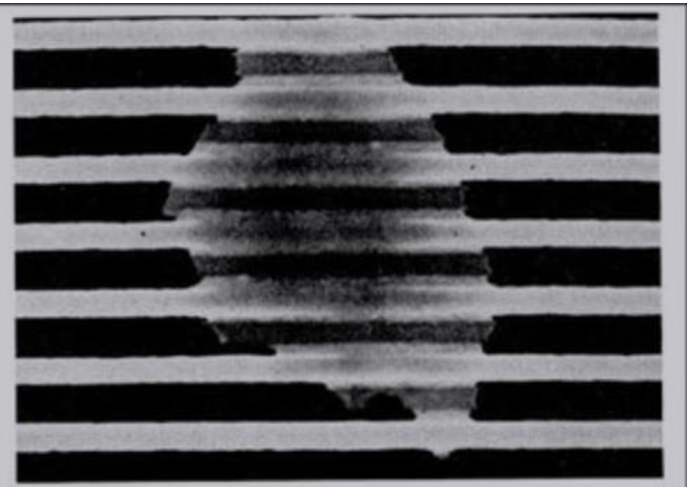


Figure 3: Example of a short of 7 metal lines caused by unexposed photo resist.

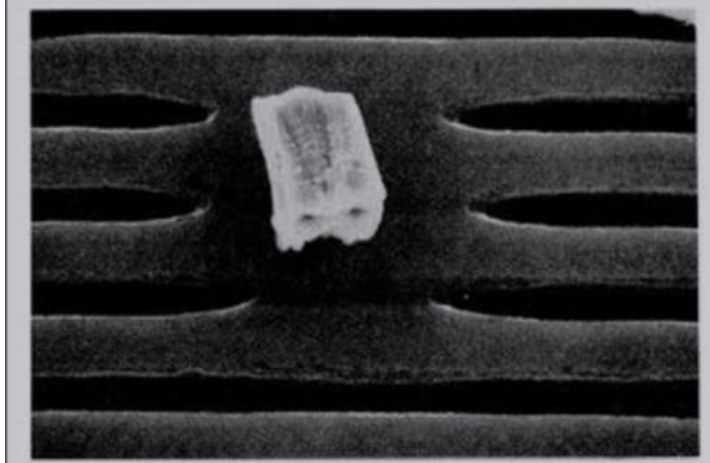


Figure 4: Short of 4 metal lines caused by a solid state particle deposited on the metal layer before metal lithography.

Figure 5: Shorts and breaks of metal lines caused by a scratch in the photoresist.

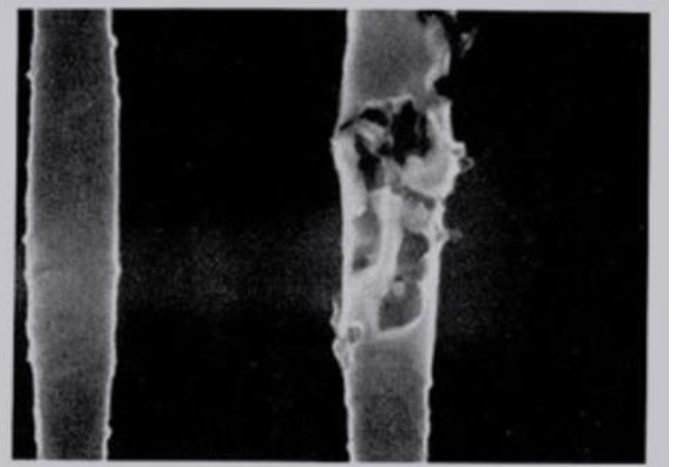
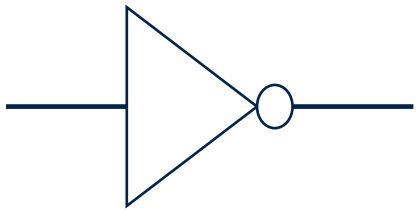


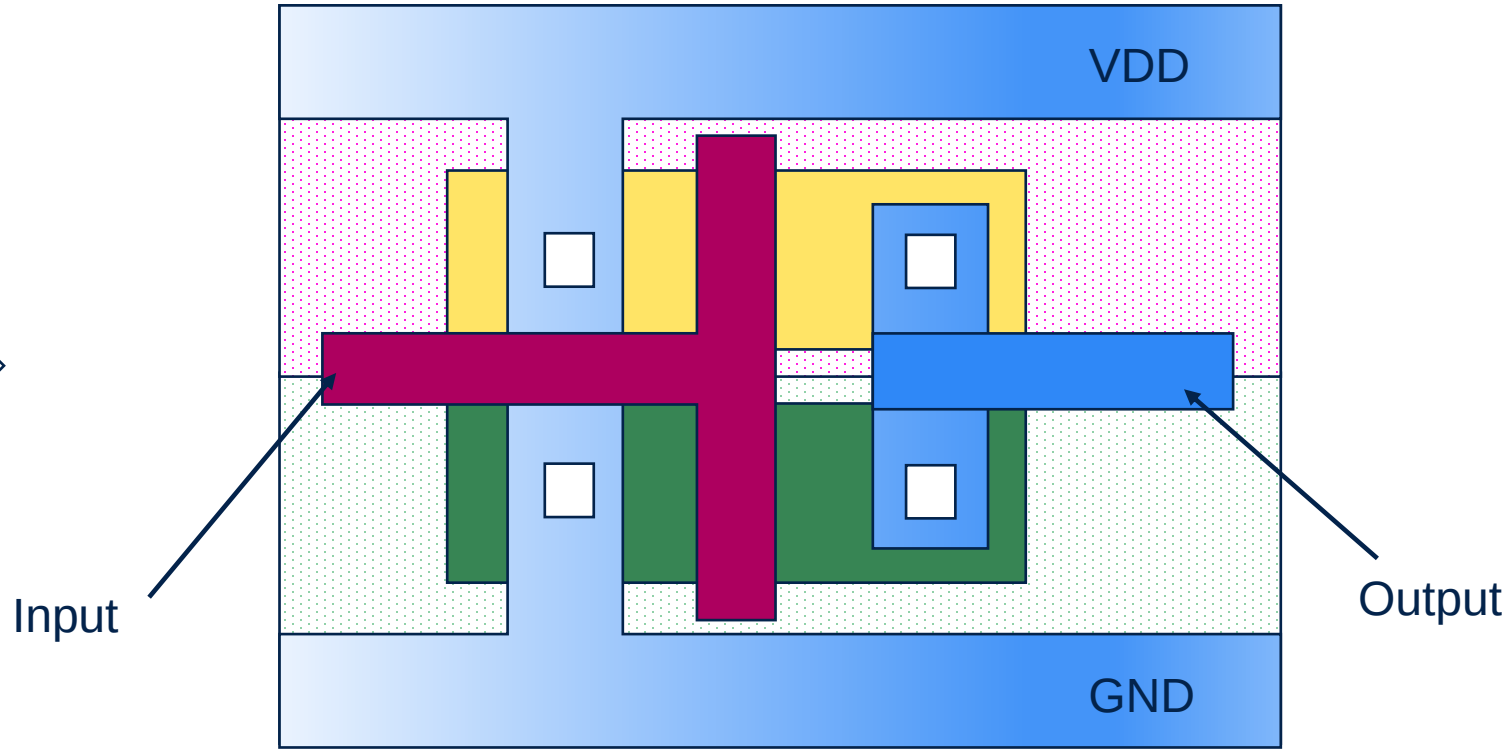
Figure 6: Example of metal line corrosion that eventually may result in breaks.

life.augmented

Physical defect in CMOS

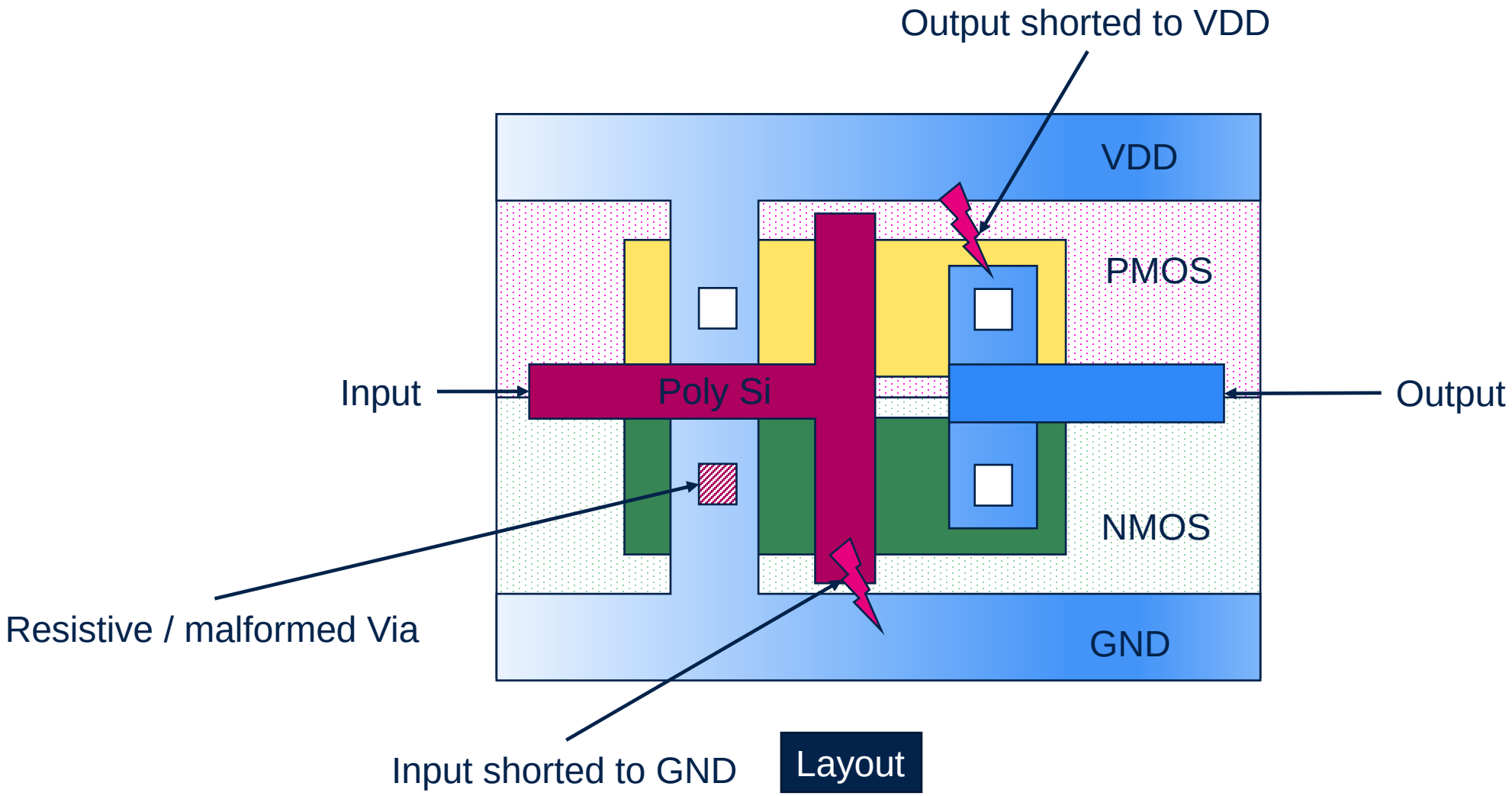


CMOS inverter

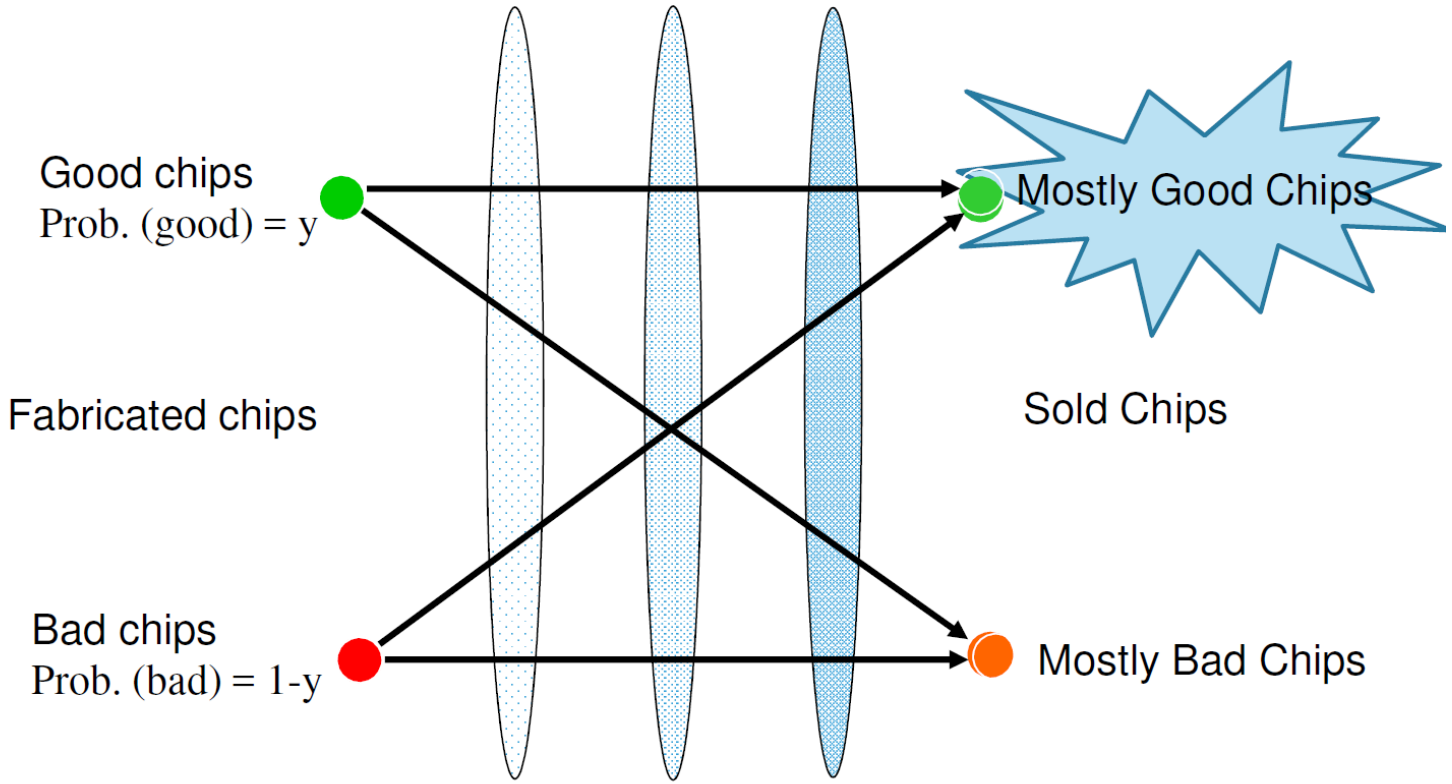


Layout

Physical defect in CMOS (cntd.)



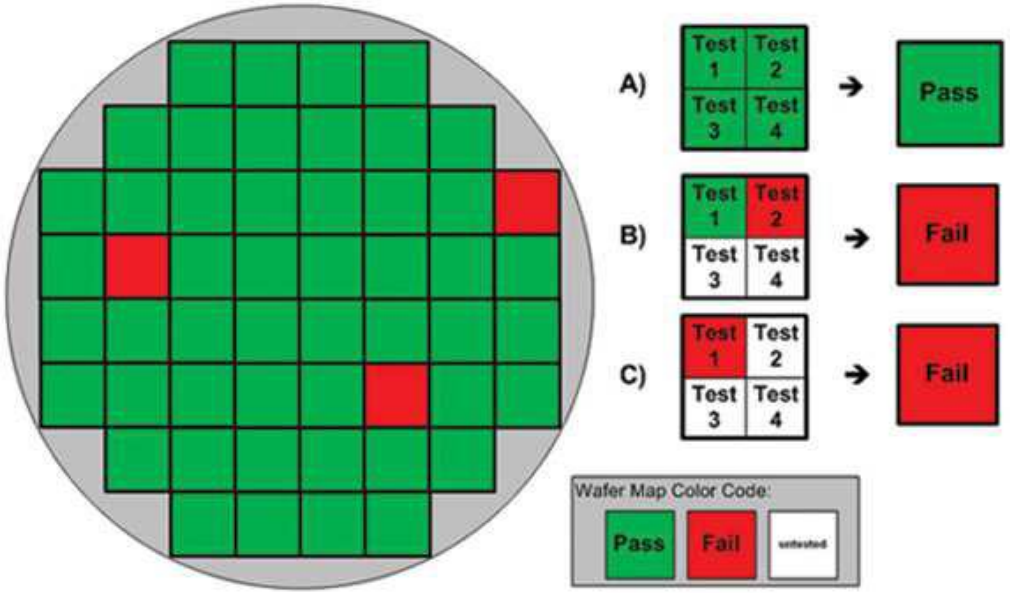
Testing as a filter process



Coverage of Tests = % of Faults our Tests are able to Detect; not 100%

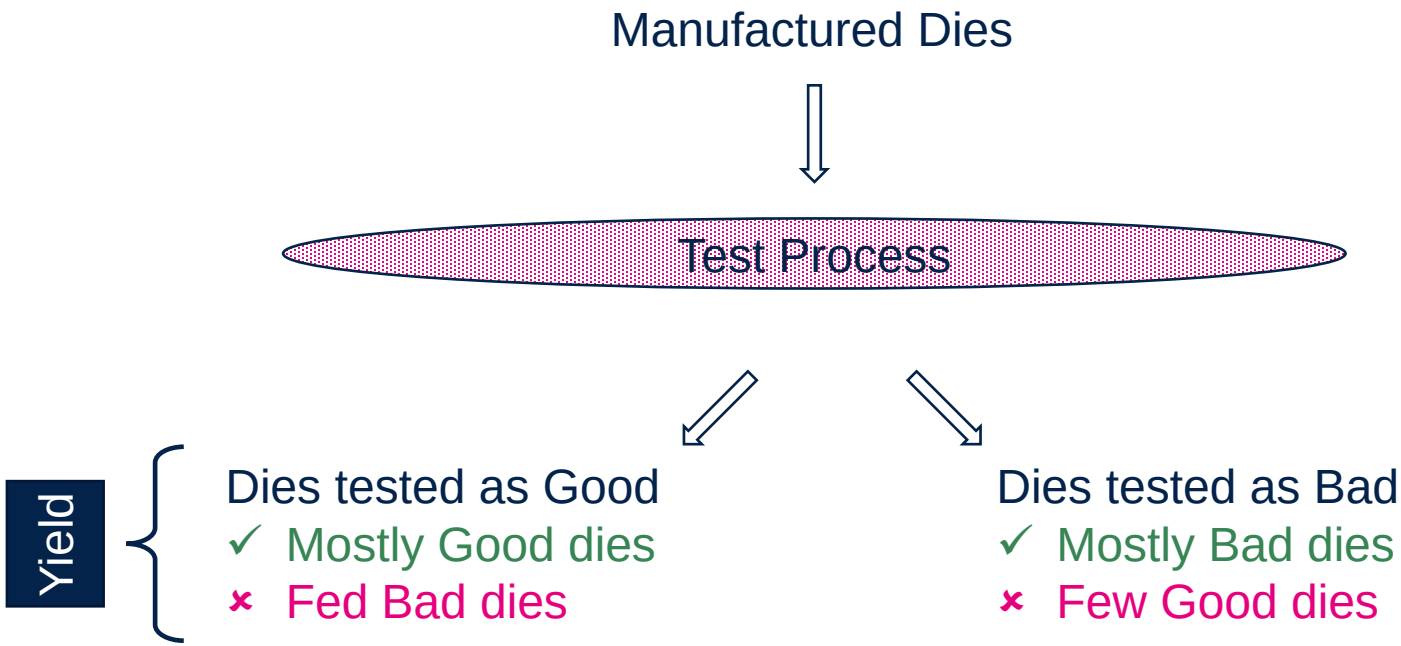
Yield

- A chip with no manufacturing defect is called a good chip.
- Fraction (or percentage) of good chips produced in a manufacturing process is called the yield.
- Yield is denoted by symbol Y.



$$Yield = \frac{No.of\ Good\ Dies}{Total\ manufactured\ Dies}$$

Defect level



- ✓ Number of defective parts sold as good ones to customer is called as defect level (DL)
- ✓ Defective parts per million (DPM)
- ✓ Good dies tested as bad are called as yield loss

What is a fault?

- Fault is the electrical manifestation of a physical defect or defects.
- A Fault model abstracts the defects at the logic gate level.
- Different fault models represent different types of fault

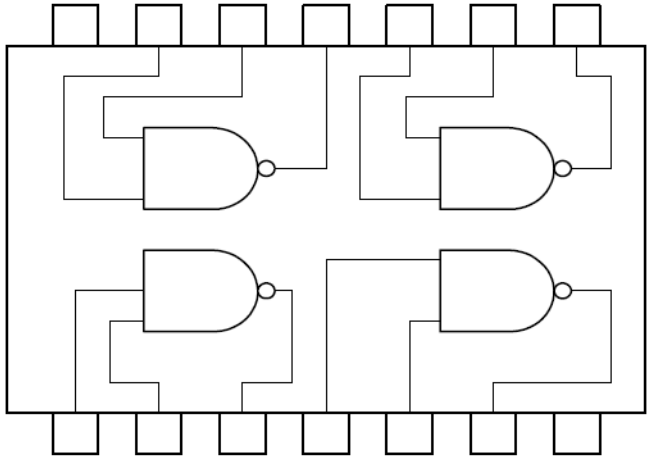
Test Type	Fault Model	Defects
Functional	Functional model	Open/Shorts in circuit
Structural Tests	Single stuck-at, multiple stuck-at, Bridging, Stuck-open	CMOS Transistor stuck-open, short, resistive bridging, Leaky transistors.
	Transition, Path delay	Partially conducting wires, resistive vias and contacts.

Functional test vs structural test

Functional test	Structural test
Verifies intended functionality	Verifies the structure and detects defects
Higher Abstraction Level	Lower Abstraction Level
Functional vectors are too many to verify the design completely.	Structural test vectors reduce the test cost while giving more Fault coverage
Custom test program generation and requires detailed knowledge of the design	Software tools can generate the test patterns. No need to understand specific functionality
No additional circuitry is required.	Additional circuit specifically for Test is required.
Hard to diagnose the defect	Easy to diagnose the defect with help of software tools.

Product testing in early days

7400
Quad
NAND



Each individual gate was easy to access.

Is it the case today with millions of gates on the design ?

What is testability?

- An attribute of a design.
- The ability to create a Test Program to determine the quality of a manufactured design.

- The combination of a design's Controllability and Observability :
  - **Controllability**
    - Be able to force a known value from the primary input
  - **Observability**
    - Be able to observe its value from the primary outputs

Controllability

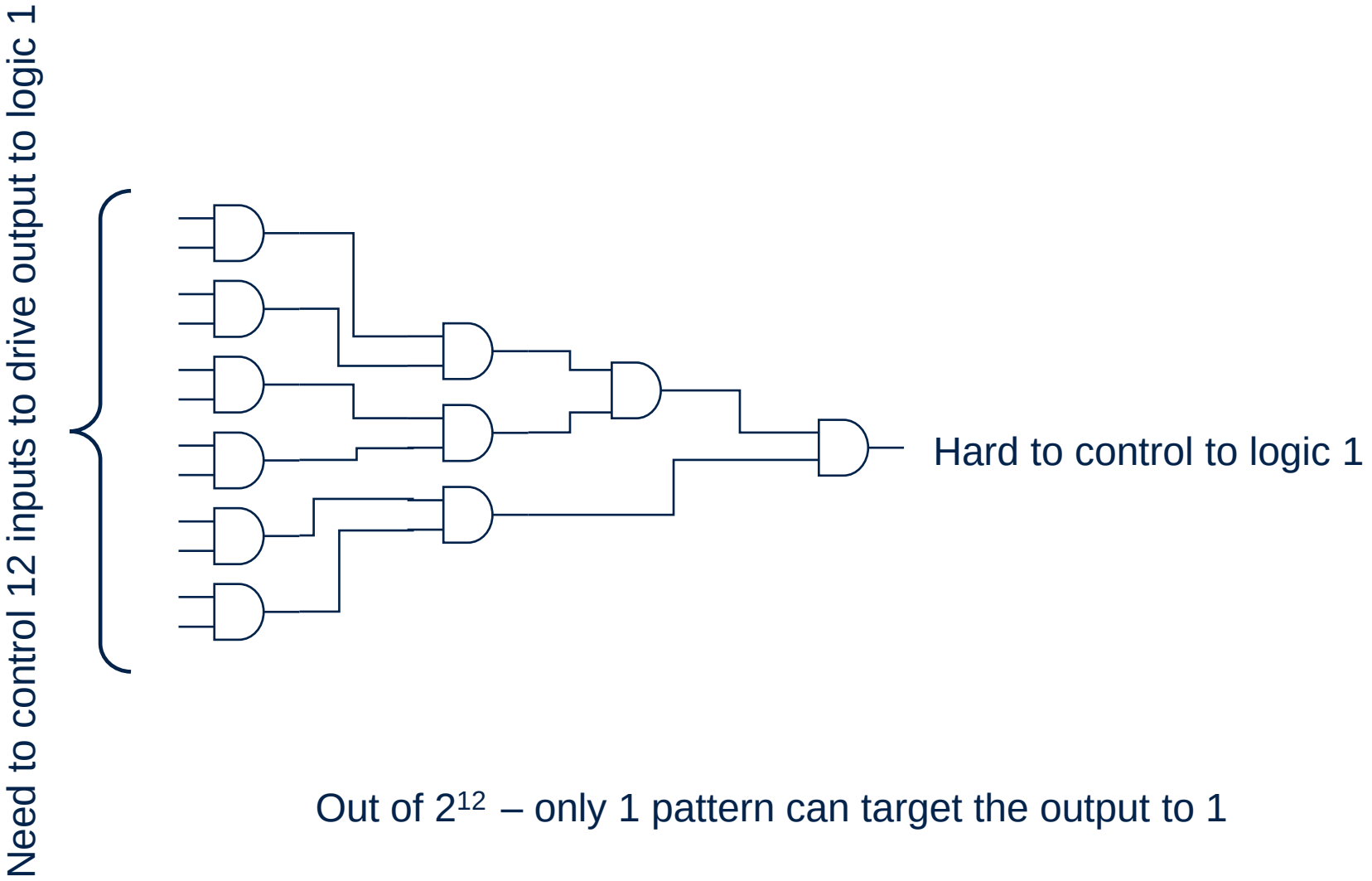
Controllability:

The ease with which Test equipment can establish a `0` or `1` value on an internal node, by applying values to the primary input ports.

Thumb rule:

The more **controllable** a logic network, the more **testable**. It's easier for ATPG tools to activate faults and detect them.

Example of hard to control logic



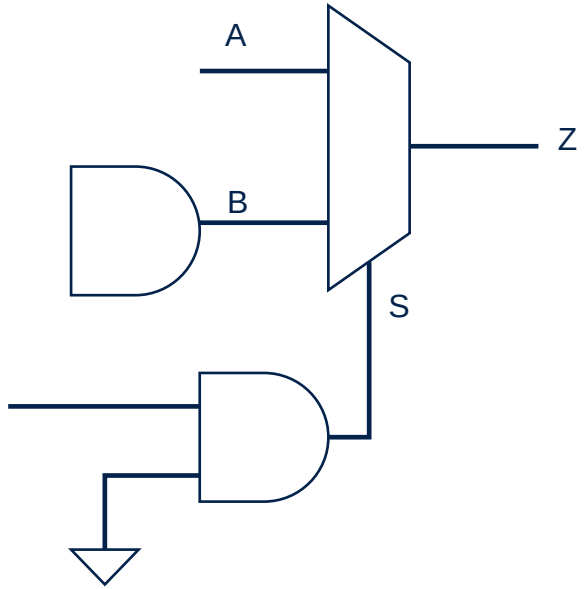
Out of 2^{12} – only 1 pattern can target the output to 1

Observability

Observability:
The ease with which ATE equipment can propagate a fault effect (**1/0** or **0/1**) to a primary output, by applying values to primary inputs.

Rule of Thumb:
The more **observable** a logic network, the more **testable**. It's easier for ATPG tools to propagate the effects of faults and detect them.

Example of unobservable logic

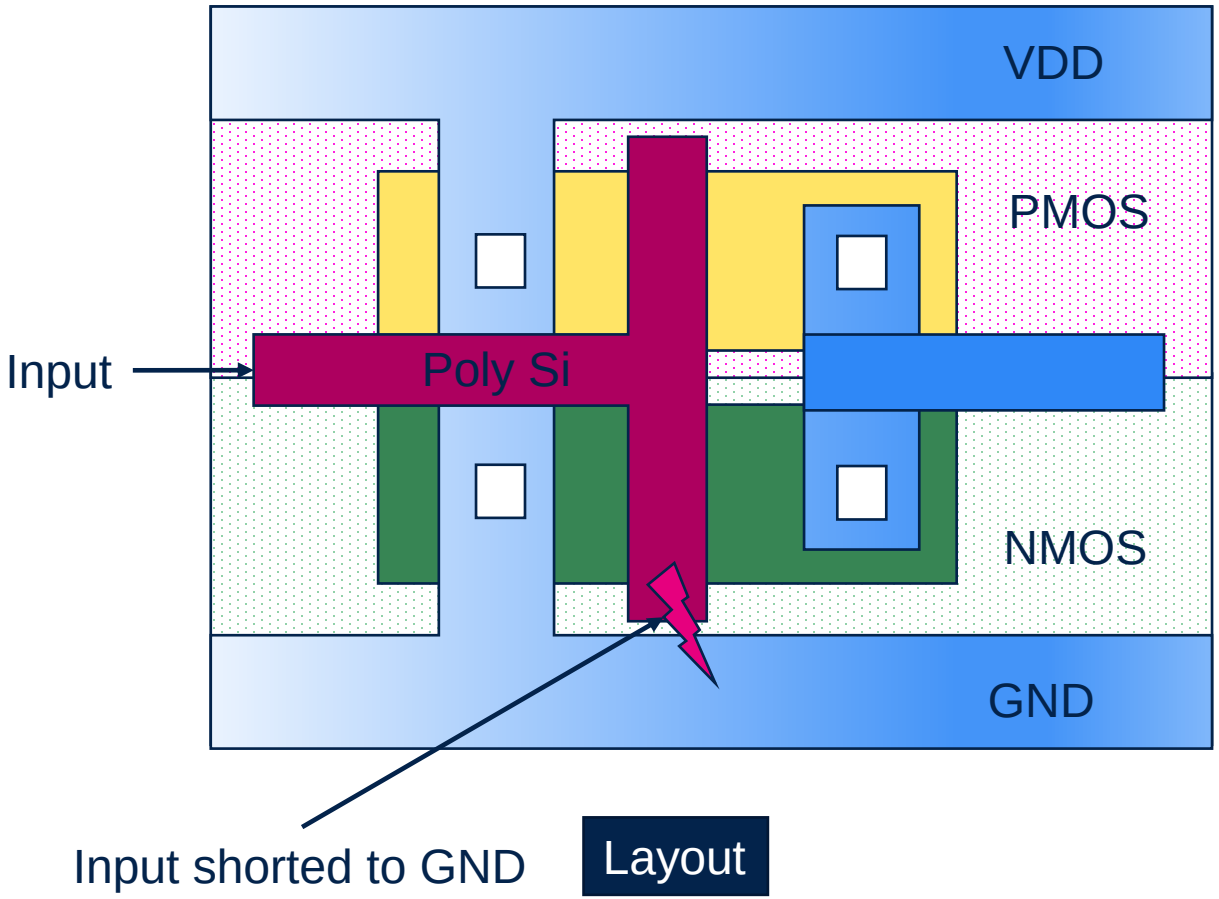


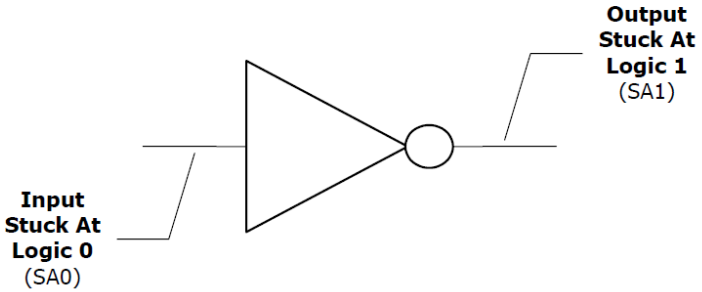
Internal node B cannot be observed at the output.

Design for testability

- Extra design to make internal nodes more observable and controllable.
- Designing For Testability simplifies the task of the ATPG tool.
- Decreases Pattern Count, hence Lower Test Cost
- Reduces Test Time due to reduced pattern count
- Less Field Returns and Increased Customer Confidence

Stuck-at fault model

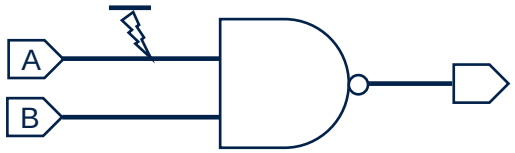




Stuck-at fault (SAF):
A logical model representing the *effects* of an underlying physical defect.

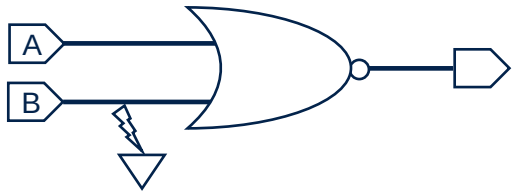
life.augmented

Examples of stuck-at fault model



Input stuck at 1 independent of value applied at A.

Output stuck at 1 independent of value applied at A and B.



Benefits of stuck-at fault model

- Simple logical model is independent of technology details.
- It reduces the complexity of fault-detection algorithms.
- Applicable to any physical defect manifesting as a signal that is stuck at a fixed logic level.
- One stuck-at fault can model more than one kind of defect.

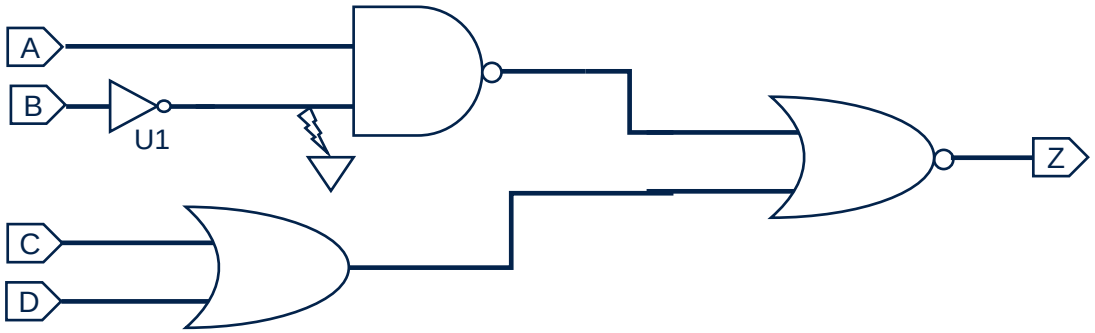
Rules for testing

No Internal Probing!

Rules of the game:
Access to a chip under test is **only** through its primary I/O ports.

Algorithm for detecting a SA fault

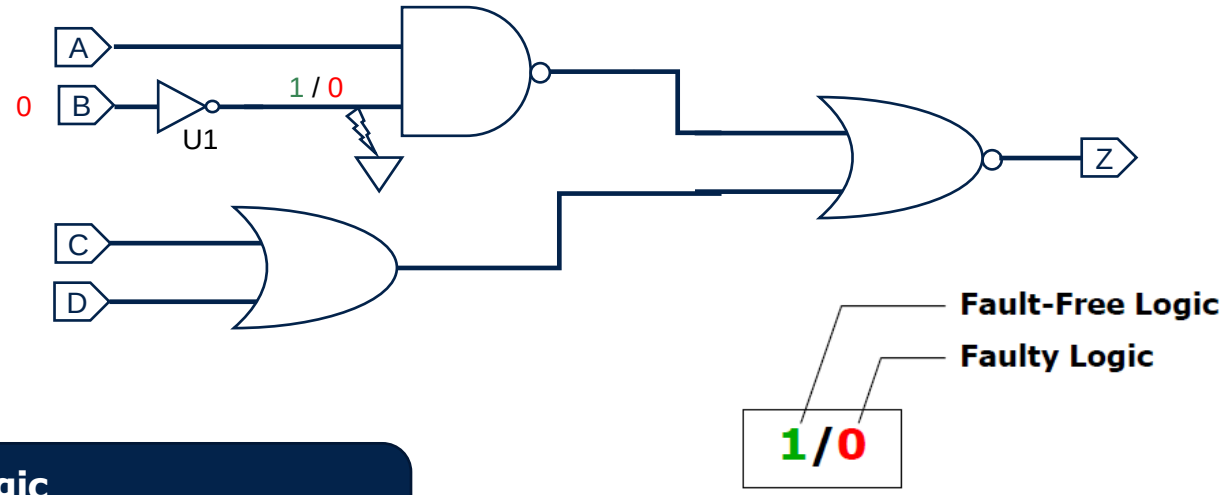
If this SA0 fault is present then U1/Z stays at logic 0.



If not present, then U1/Z is driven to its normal value.

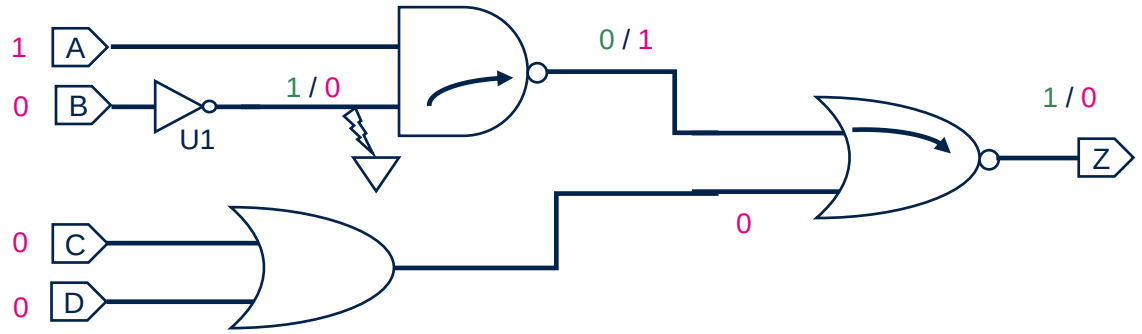
We can exploit this “either/or behavior” to detect the fault.

D Algorithm - Activate the fault



D Algorithm: Faulty logic
1. Target a specific stuck-at fault.
2. Drive fault site to **opposite** value....

D Algorithm - Propagate the fault



D Algorithm: Faulty logic
1. Target a specific stuck-at fault.
2. Drive fault site to **opposite** value....

Faulty Free Value / Faulty Value

```
Vector {SignalGroup = 1000 H}
```

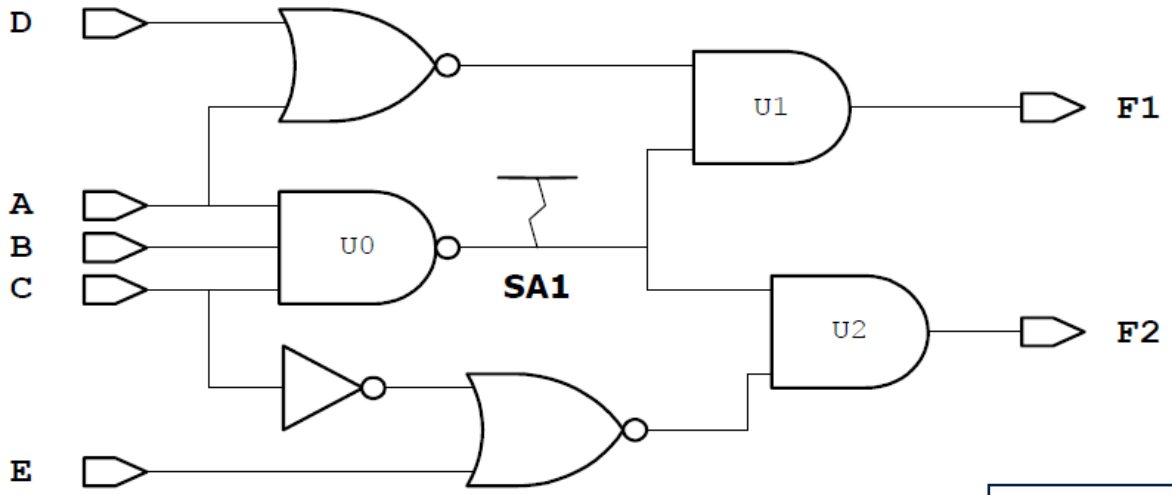
Inputs Outputs

Summary of the D algorithm

D Algorithm:

1. Target a specific stuck-at fault.
2. Drive fault site to **opposite** value.
3. Propagate error to primary output.
4. Record pattern; go to next fault.

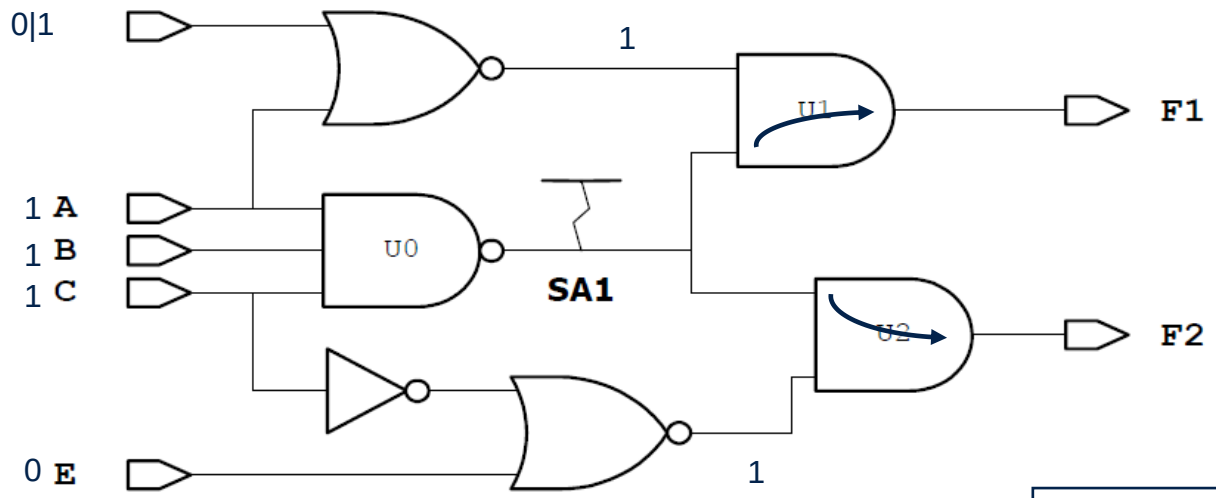
Exercise: Detect a SA1 fault




```
Vector {ALL=________;}
```

What To Do:
Apply **D** algorithm, devising test pattern to detect the **SA1** fault. Record pattern.

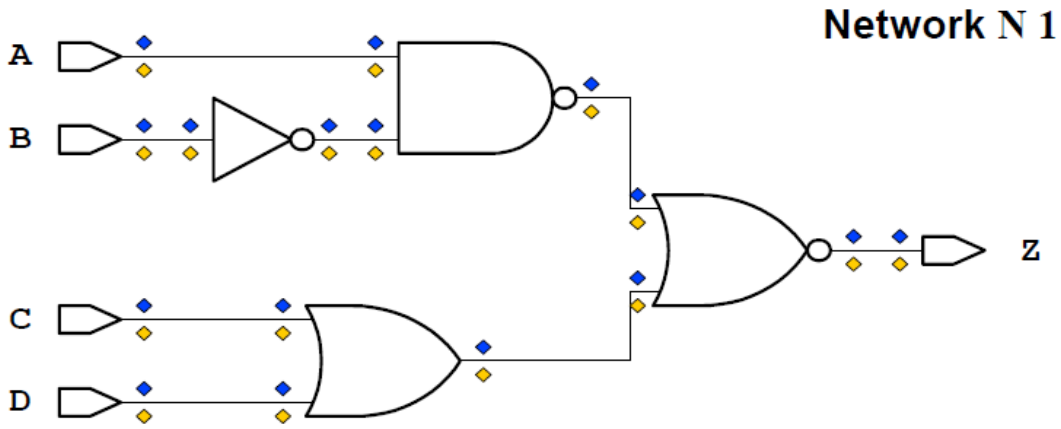
Solution




```
Vector {ALL=11100_X0;}
```

What to do:
Apply **D** algorithm, devising test pattern to detect the **SA1** fault. Record pattern.

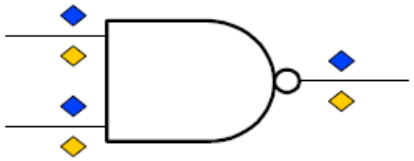
Total potential faults



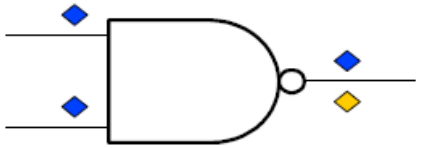
Total Fault Count

$$N_f = 2\,(N_{pins} + N_{ports}) = 2\,(11 + 5) = 32$$

Fault collapsing



Before Collapsing



After Collapsing

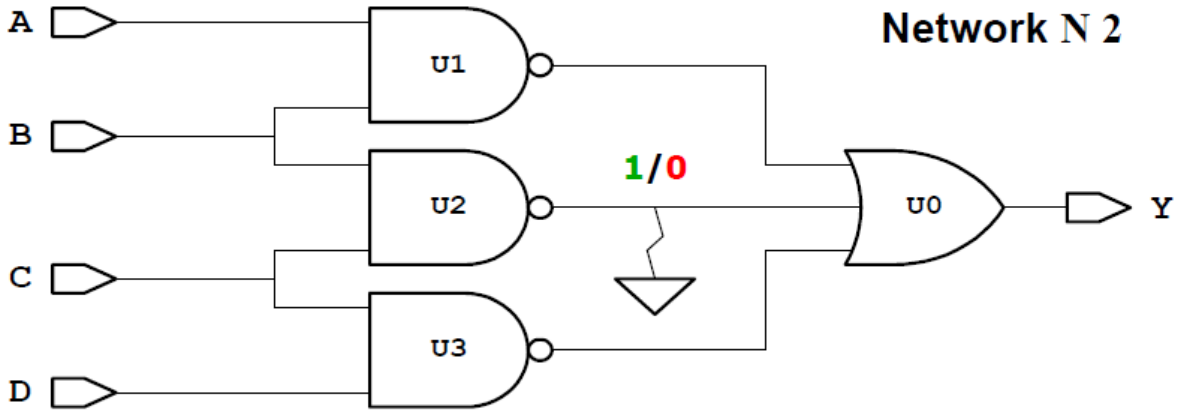
- The gate behaves the same for any input SA0 or the output SA1.
- The three faults { `A` SA0, `B` SA0, `Z` SA1 } are thus all equivalent.
- Only one of them, `Z` SA1, need be included in the fault universe.

Equivalent faults

Equivalent Faults:
Some of these faults are functionally **equivalent** to one another.
A set of faults is equivalent if **no test pattern** exists to tell them apart.

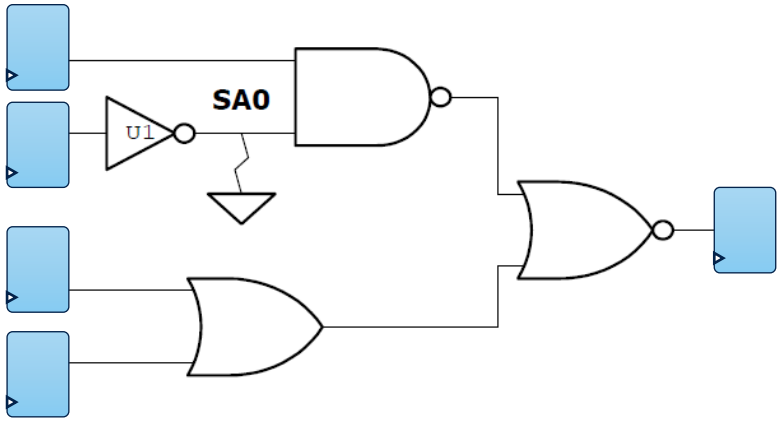
Fault Collapsing:
By testing for only **one** fault per equivalence set, we can greatly reduce (or collapse) the fault universe. This considerably speeds up fault simulation.

An undetectable fault



Some circuits have inherently undetectable faults.
One reason, shown here, re-convergent fanout.
No pattern can be devised to detect fault U2/Z SA0.

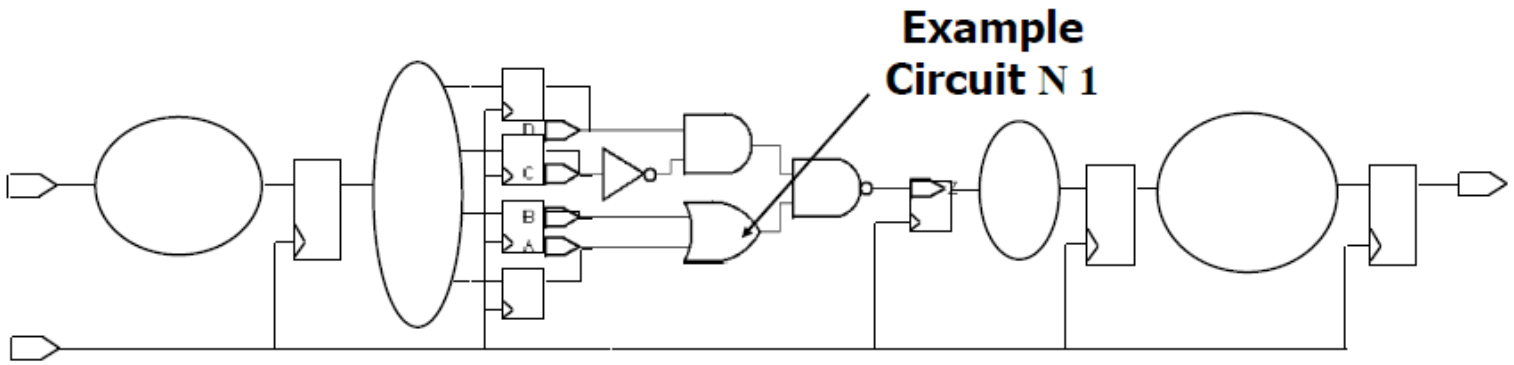
Testing sequential logic



This modification of circuit N 1 has two register stages to deal with.
We need to clock in the stimulus, and then clock out the response.

Testing sequential logic

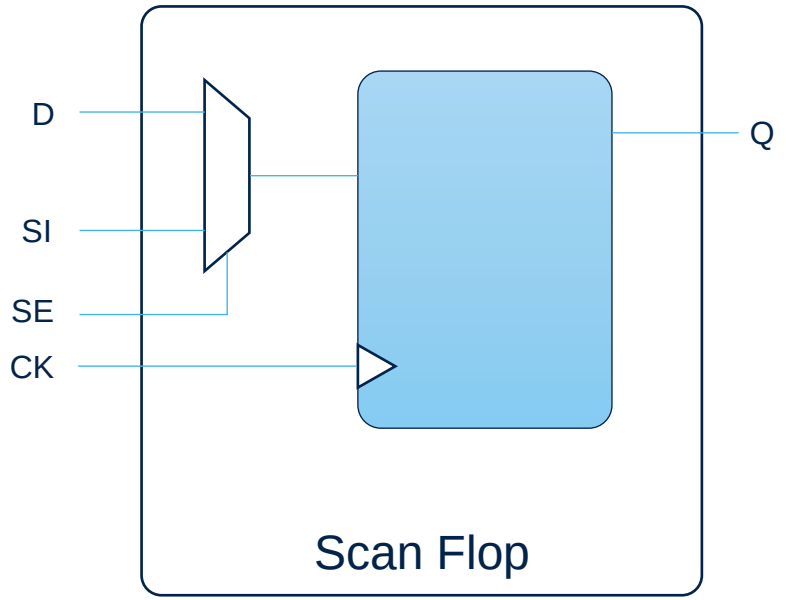
- To detect a stuck-at fault in synchronous sequential logic, we can still use the familiar D algorithm, but it'll take….
  - • One or more clock cycles to activate the fault.
  - • One or more clock cycles to propagate the fault effect.
- In general, we'll need a sequence of patterns to detect a fault!



For large sequential logic blocks with complex circuits, Sequential ATPG is just not practical.

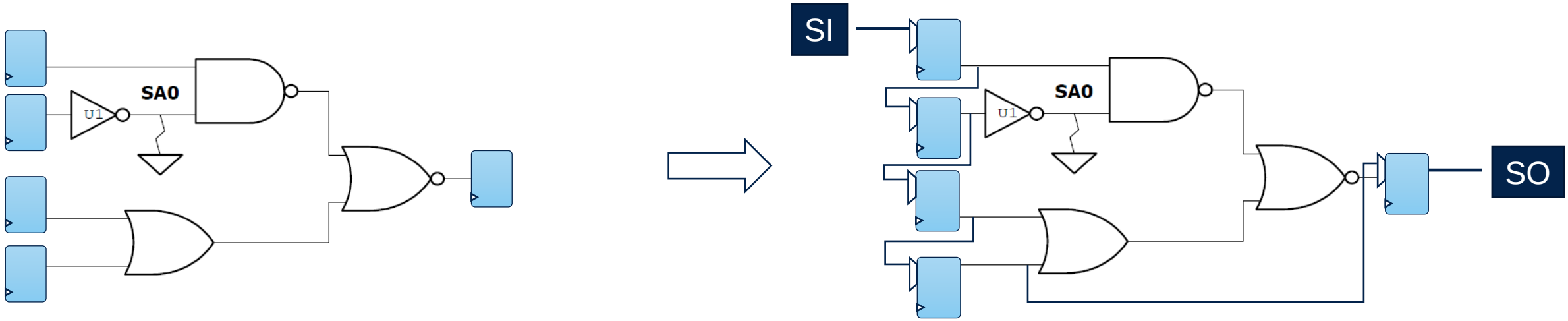
Scan Flip-Flop

- Convert Flip Flop to provide additional control input via MUX.
- Scan Flop = Scan Mux + Flop



Scan overview – Basic concept

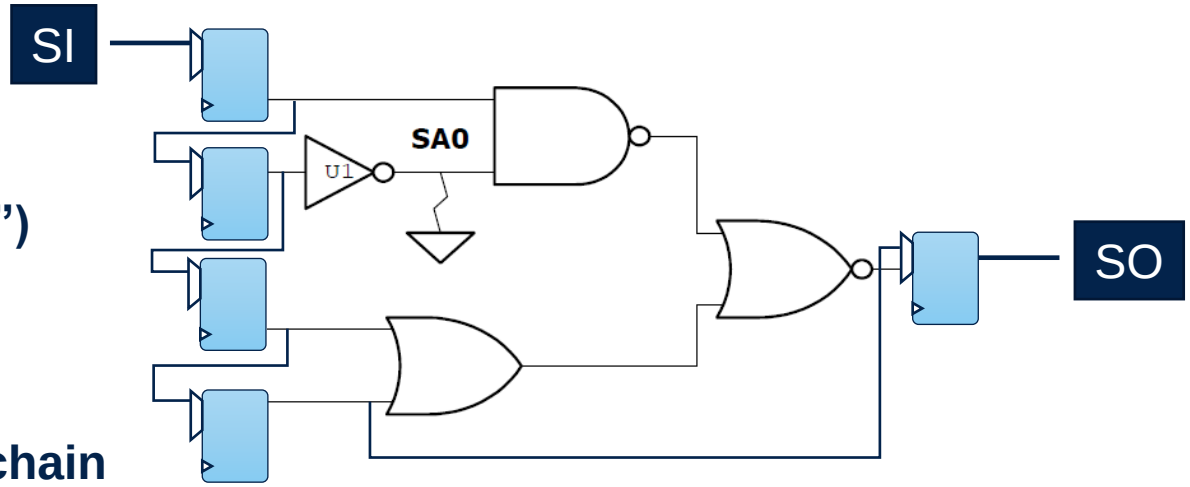
- Transform all flip-flops in sequential logic into a shift register
- Arbitrary stimulus may be applied to the logic by means of **loading**
- data into each of the flip-flops in the scan chain
- The response of the logic may be “**captured**” back into the flip-flops
- and observed by means of unloading the flip-flops in the scan chain



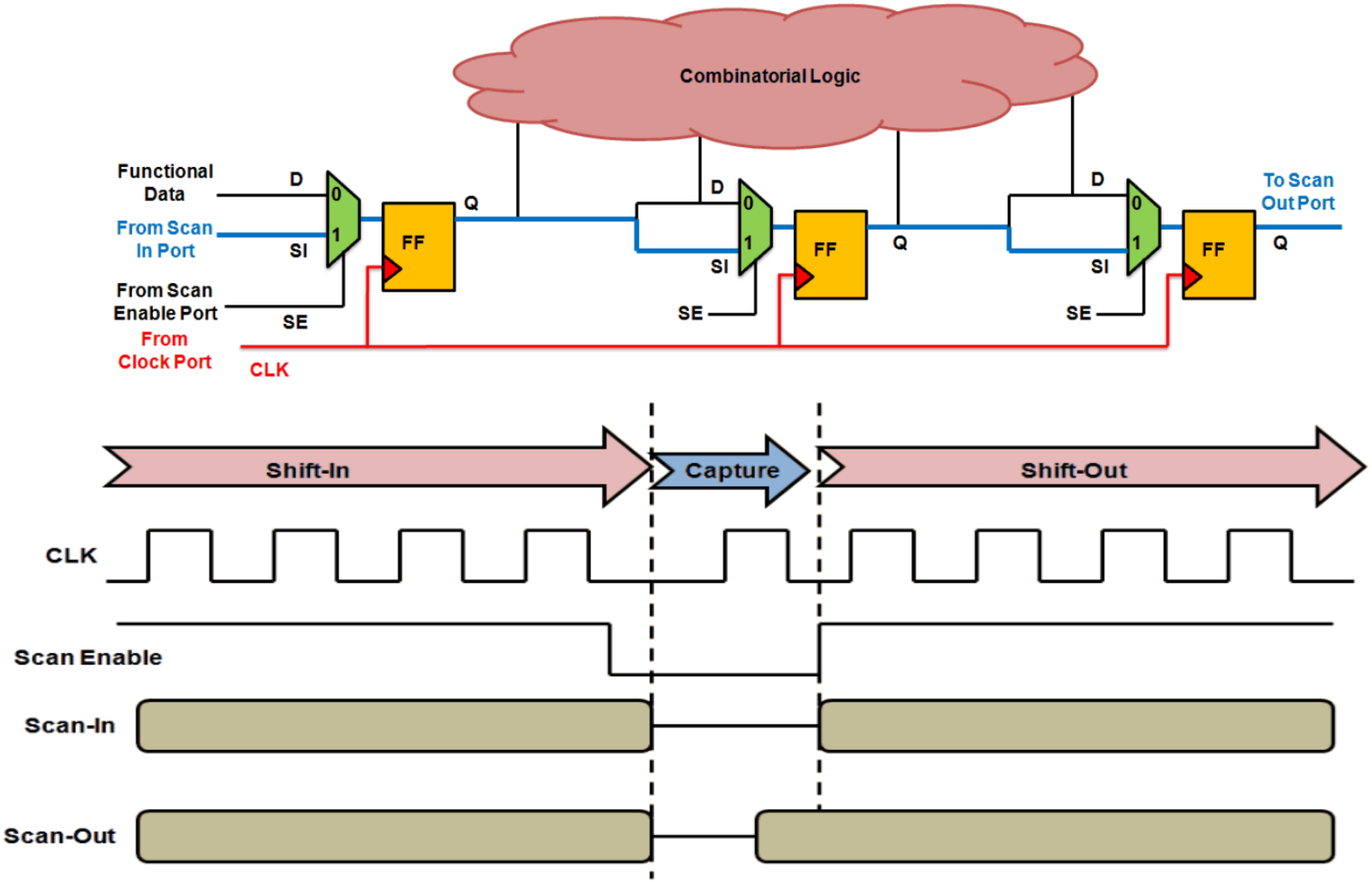
life.augmented

Scan overview – Scan test sequence

- **Step 1 – Shift some patterns through the scan chain**
  - Flip-flops are now completely tested
- **Step 2 – Scan in a test pattern**
  - Scan enable pin = 1
- **Step 3 – Apply test pattern to device input pins (“PI’s”)**
  - The internal logic now evaluates
- **Step 4 – Test results at device output pins (“PO’s”)**
- **Step 5 – Apply scan clock to capture results in scan chain**
  - Scan enable = 0
- **Step 6 – Scan out results and check against expected values**
  - Scan enable = 1



Scan overview – Scan chain



Scan Test - Advantages

- The test problem is simplified to that of testing combinational logic

- The process of generating patterns is fully automated
  - ✓ Well supported by CAD vendors
    - Synopsys TestMax
    - Mentor - Tessent
    - Cadence Modus

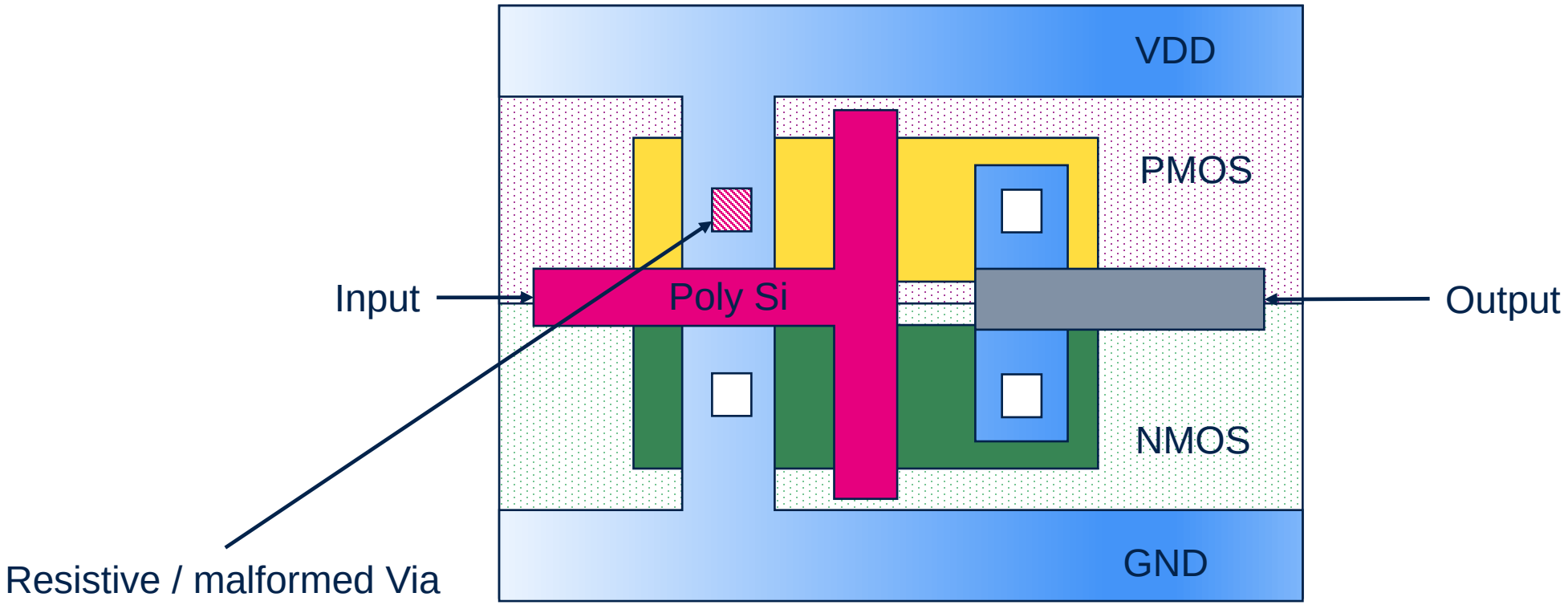
- High fault coverage
  - Nearly 100% for gate level stuck at fault model
  - Typical target for automotive segment is > 99%

- Same approach at all levels of hierarchy
  - module => IP => chip

Scan test - Cost

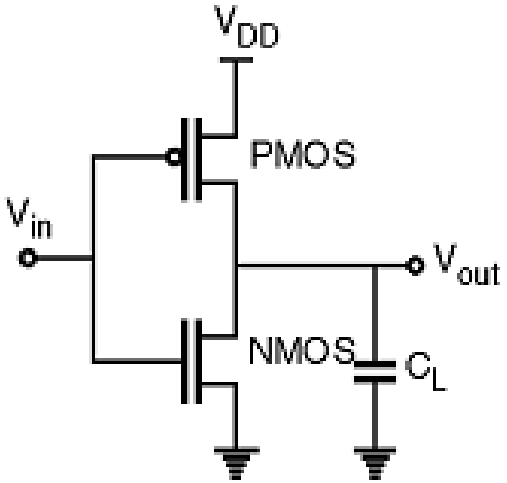
- Overhead
  - Pin requirement
    - ✓ Scan input
    - ✓ Scan output
    - ✓ Scan enable
  - Area increase
    - ✓ 1 mux per flip-flop (typically 2-5% area overhead)
    - ✓ Routing of scan chain
  - Performance degradation
    - ✓ Added delay through the mux
- Long test application time
  - # cycles = # test patterns x scan chain length

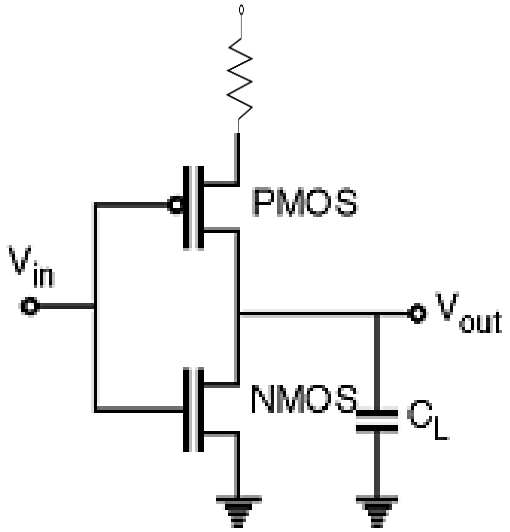
Transition fault

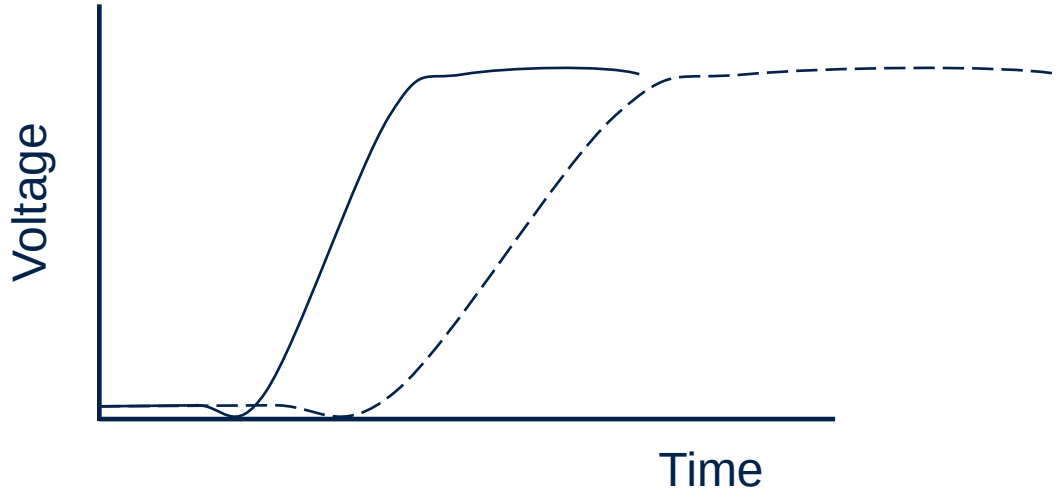


How will the circuit with a resistive via behave?

Transition fault





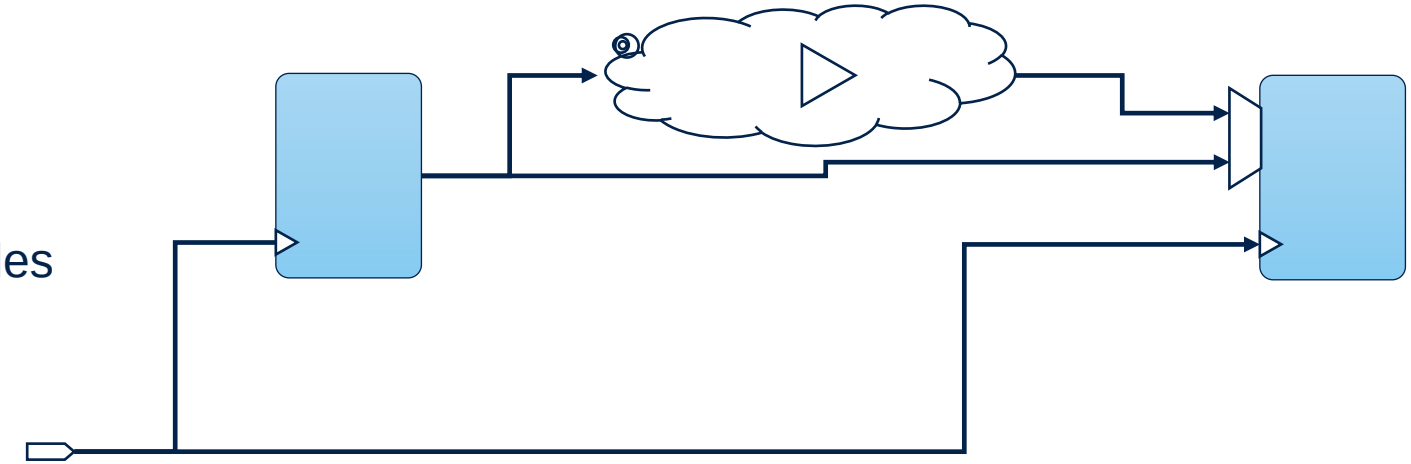


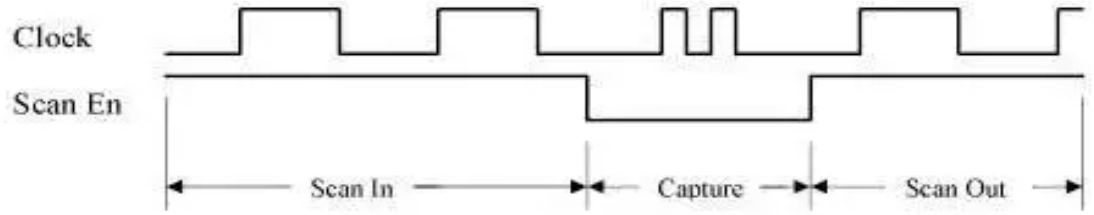
How will the circuit with resistive vias behave?

life.augmented

Transition fault model

- Models gross delay on gate terminals
  - Terminals are slow to rise or fall
  - Partially conducting transistors interconnections
- Possible faults
  - Slow-to-Rise
  - Slow-to-Fall
- Requires at-speed application of 2 clock cycles
  - Launch
  - Capture





Fault coverage

$$\textbf{Fault Coverage} = \frac{\text{number of detected faults}}{\text{total number of faults}}$$

Since not all faults are detectable, in general FC $<$ 100%.

Fault classification

- **Testable faults**
  - **Detected (DT)**
    - Bad ckt output opposite of good ckt output at PO.
    - Either by Simulation (DS) or by Implication (DI)
  - **Possibly Detected (PT) (O/P may be X)**
    - Atpg_untestable , Possible detected (AP)
    - Not_analyzed, Possibly detected (NP)
- **Undetectable faults (UD) (Not part of “Test Coverage”)**
  - Undetectable Unused (UU)
  - Undetectable Tied (UT)
  - Undetectable Blocked (UB)
  - Undetectable Redundant (UR)
- **Not Detected (ND)**
  - Not controllable (NC) ,
  - Not observable (NO) nodes

Coverage metrics

Test coverage

Percentage of all testable faults that are detected by the pattern set.

TEST COVERAGE = Detected faults (DT) / Total testable faults X 100

Fault coverage

Percentage of all the faults detected by the pattern set

FAULT COVERAGE = Detected faults (DT) / Total number of faults X100

Our technology starts with You

Find out more at www.st.com

life.augmented