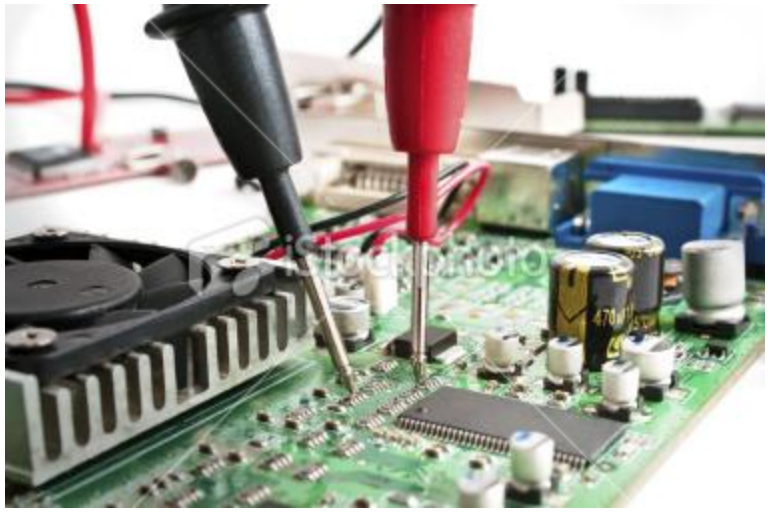
# Hardware testing and design for testability



# Introduction

## A Digital System requires testing before and after it is manufactured

- **Level 1:** behavioral modeling and test benches
  - Check for correct design and algorithms
- **Level 2:** logic level testing
  - Check for correct logic and whether the design meets specifications
- **Level 3:** circuit level testing
  - Check for correct implementation and whether the timing is correct
- **Level 4:** post-manufacture testing
  - Check for defects

# Introduction

- Digital systems should be designed so that they are easy to test

- Important to develop efficient testing methods
  - Design for testability (DFT)
  - Automatic test pattern generators (ATPG)
  - Built in Self Test (BST)

- Testing Combinational Logic
- Testing Sequential Logic
- Scan Testing
- Boundary Scan

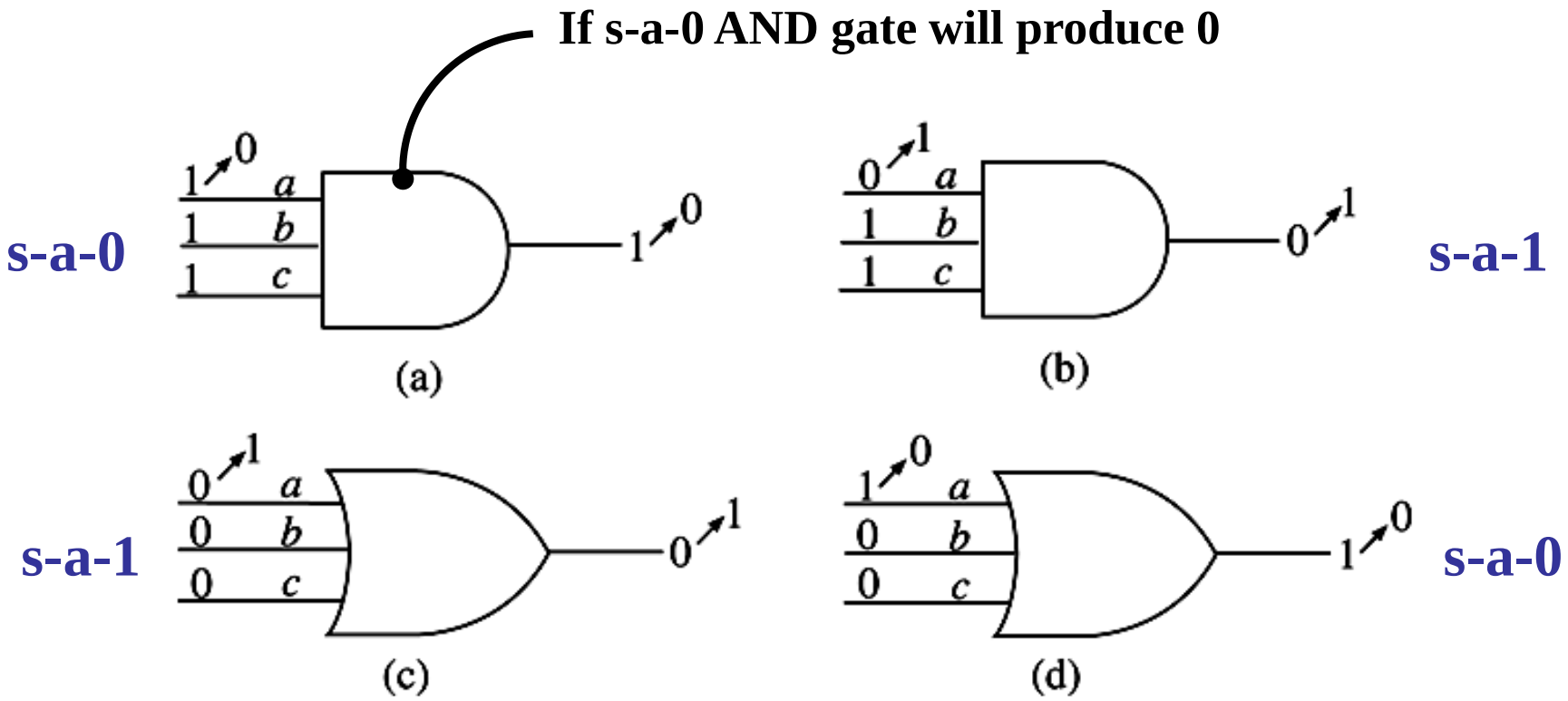
# Testing Combinational Logic

# Testing Combinational Logic

- **Stuck-at-1 (s-a-1) and Stuck-at-0 (s-a-0)**
  - if the input to a gate is ***shorted to ground***, it is **s-a-0**
  - if the input to a gate is ***shorted to positive power supply***, it is **s-a-1**
  - if the input to a gate is an ***open circuit***, it may act as **s-a-0** or **s-a-1**
- **For s-a-0, provide '1' as input**
  - To check if it will flip ‘1’ to ‘0’
- **For s-a-1, provide '0' as input**
  - To check if it will flip ‘0’ to ‘1’

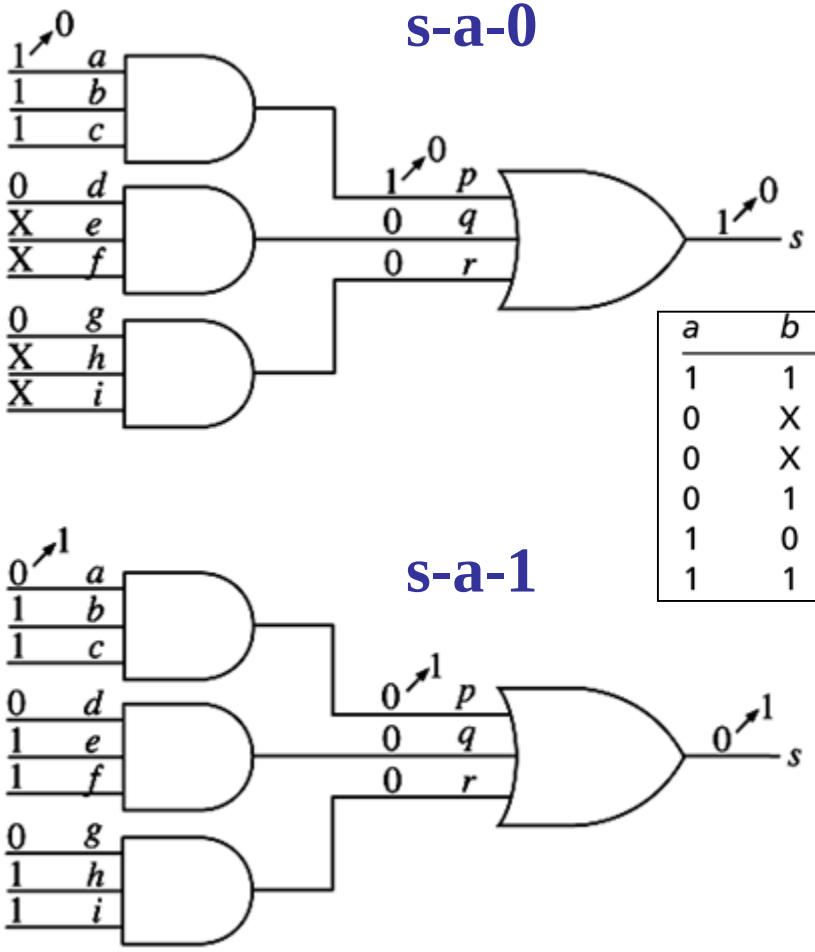
# Testing Combinational Logic: stuck-at faults

- Finding AND and OR Gate Stuck-at-faults



# Testing Combinational Logic: stuck-at faults

- Two-level circuit

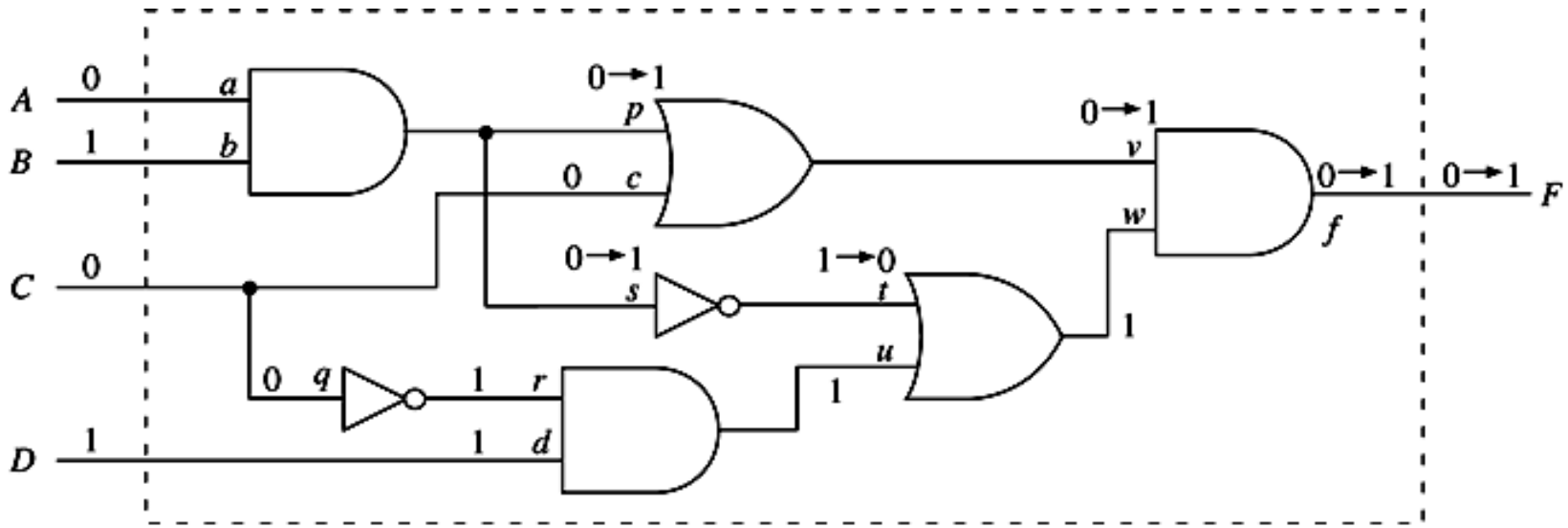



| a | b | c | d | e | f | g | h | i | Faults Tested |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | X | X | 0 | X | X | a0, b0, c0, p0 |
| 0 | X | X | 1 | 1 | 1 | 0 | X | X | d0, e0, f0, q0 |
| 0 | X | X | 0 | X | X | 1 | 1 | 1 | g0, h0, i0, r0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | a1, d1, g1, p1, q1, r1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | b1, e1, h1, p1, q1, r1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | c1, f1, i1, p1, q1, r1 |

**We have no clue about the internal faults, we can provide inputs and look at outputs: the fault will propagate by setting specific pins to 1 or 0**

# Testing Combinational Logic: stuck-at faults

- Multi-level circuit




| Test Vectors | | | | Normal Gate Inputs | | | | | | | | | | | | Faults Tested | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | a | b | p | c | q | r | d | s | t | u | v | w | F | | | | | | | | | |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | a1 | p1 | c1 | v1 | f1 | | | | |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | a0 | b0 | p0 | q1 | r0 | d0 | u0 | v0 | w0 | f0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | b1 | c0 | s1 | t0 | v0 | w0 | f0 | | | |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | a0 | b0 | d1 | s0 | t1 | u1 | w1 | f1 | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | a0 | b0 | q0 | r1 | s0 | t1 | u1 | w1 | f1 | |

# Testing Combinational Logic: bridging faults

- **Bridging fault occurs when two unconnected signal lines are shorted**

- Finding a minimum set of test vectors that will test all possible faults is very difficult
  - Use small set of testing vectors that can test most faults
  - Use algorithms and programs that will generate set of test vectors

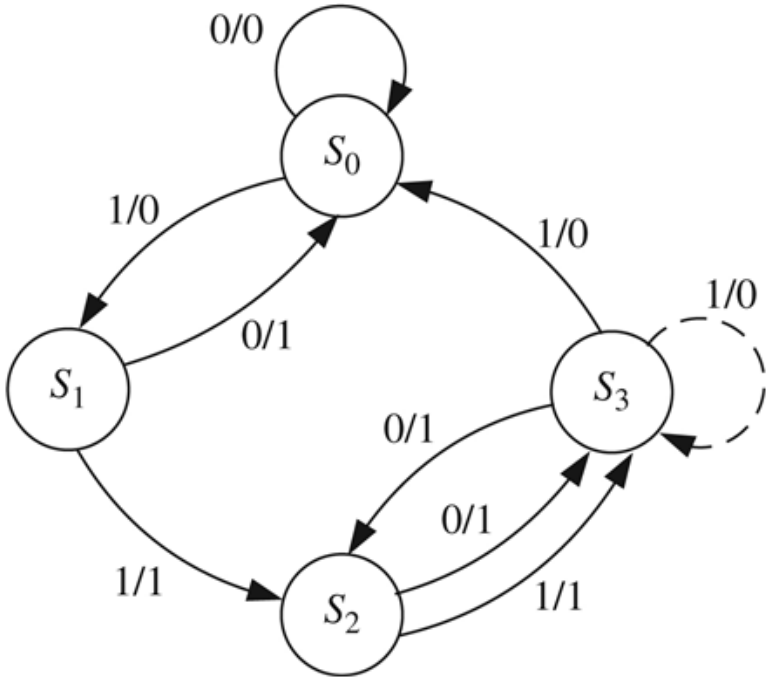
# Testing Sequential Logic

# Testing Sequential Logic

- **Testing Sequential Logic is more difficult than combinational logic**
  - Sequence of inputs and resulting outputs
  - No access to state of flipflops
  - Very large number of tests are required
  - Use a small set of test sequences that can be adequate

# Testing Sequential Logic

- **Look for all possible state transitions and outputs**



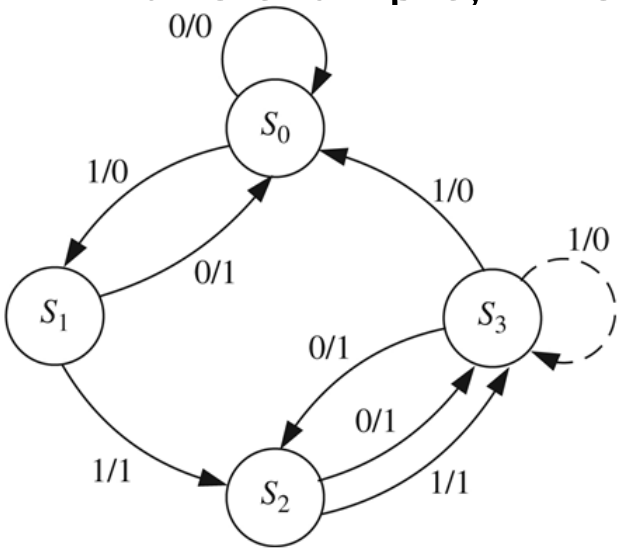

| Q1Q2 | State | Next State $X=0$ | Next State 1 | Output $X=0$ | Output 1 |
|---|---|---|---|---|---|
| 00 | $S_0$ | $S_0$ | $S_1$ | 0 | 0 |
| 10 | $S_1$ | $S_0$ | $S_2$ | 1 | 1 |
| 01 | $S_2$ | $S_3$ | $S_3$ | 1 | 1 |
| 11 | $S_3$ | $S_2$ | $S_0$ | 1 | 0 |

**X=010110011**

**Z=001111100**
**(both solid and dashed)**

# Testing Sequential Logic

- **Find the distinguishing sequence**
  - Two states are distinguishable if an input sequence produces different output sequences
  - In this example, **11** is the distinguishing sequence




| Q1Q2 | State | Next State X = 0 | Next State 1 | Output X = 0 | Output 1 |
|---|---|---|---|---|---|
| 00 | $S_0$ | $S_0$ | $S_1$ | 0 | 0 |
| 10 | $S_1$ | $S_0$ | $S_2$ | 1 | 1 |
| 01 | $S_2$ | $S_3$ | $S_3$ | 1 | 1 |
| 11 | $S_3$ | $S_2$ | $S_0$ | 1 | 0 |

- **How to find distinguishing sequence?**

1) Apply 1 to all states and see which states produce identical outputs

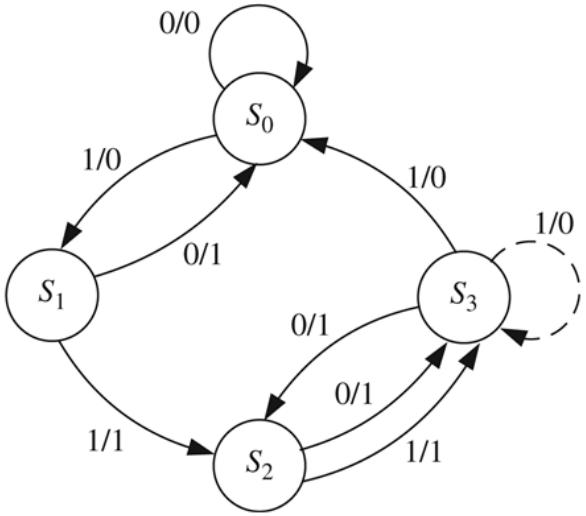
**1 ⟶ {S0, S3} ⟶ 0 and 1 ⟶ {S1,S2} ⟶ 1**

2) Apply 1 again to find dinstict states within each group

**11 ⟶ S0 ⟶ 01, 11 ⟶ S3 ⟶ 00, 11 ⟶ S1 ⟶ 11, 11 ⟶ S2 ⟶ 10**

# Testing Sequential Logic





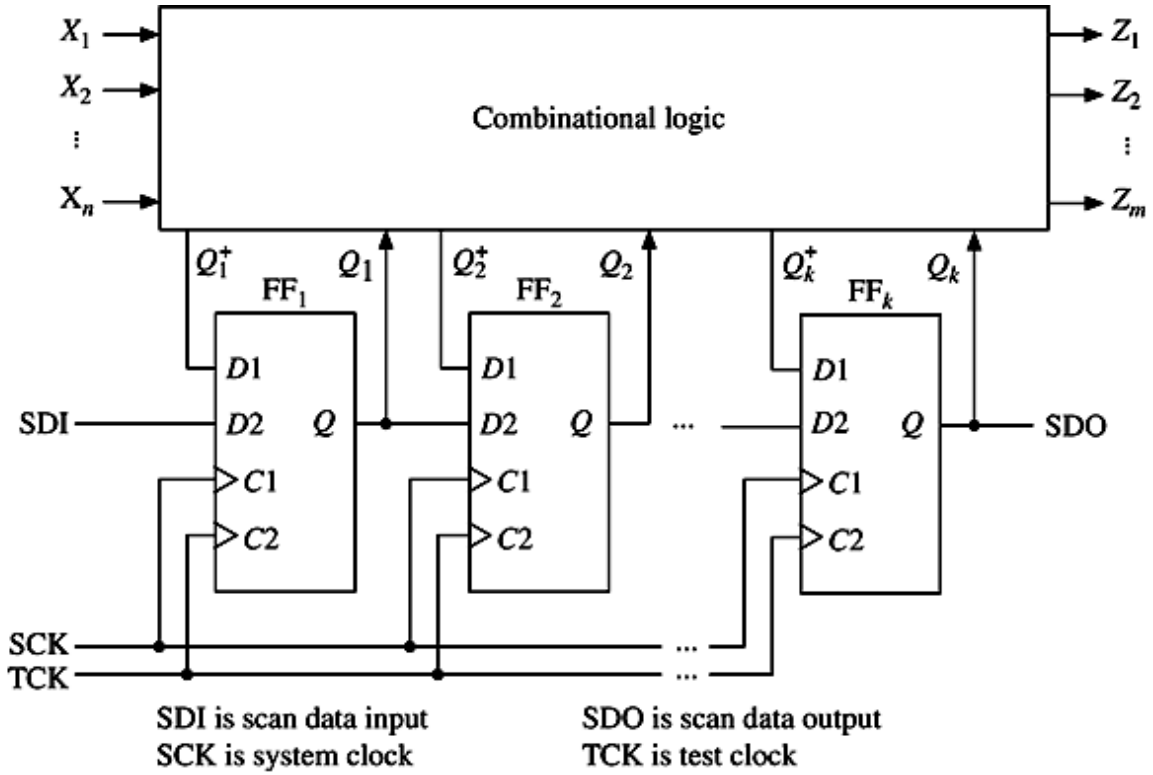

| $Q1Q2$ | State | Next State $X=0$ | Next State $1$ | Output $X=0$ | Output $1$ |
|---|---|---|---|---|---|
| 00 | $S_0$ | $S_0$ | $S_1$ | 0 | 0 |
| 10 | $S_1$ | $S_0$ | $S_2$ | 1 | 1 |
| 01 | $S_2$ | $S_3$ | $S_3$ | 1 | 1 |
| 11 | $S_3$ | $S_2$ | $S_0$ | 1 | 0 |

| Input | Output | Transition Tested |
|---|---|---|
| 0 1 1 | 0 0 1 | S0 to S2 |
| 1 1 1 | 0 1 1 | S0 to S3 |
| 1 0 1 1 | 0 1 0 1 | S1 to S0 |
| 1 1 1 1 | 0 1 1 0 | S1 to S2 |
| 1 1 0 1 1 | 0 1 1 0 0 | S2 to S3 |
| 1 1 1 1 1 | 0 1 1 0 0 | S2 to S3 |
| 1 1 0 0 1 1 | 0 1 1 1 1 0 | S3 to S2 |
| 1 1 0 1 1 1 | 0 1 1 0 0 1 | S3 to S0 |

# Scan Testing

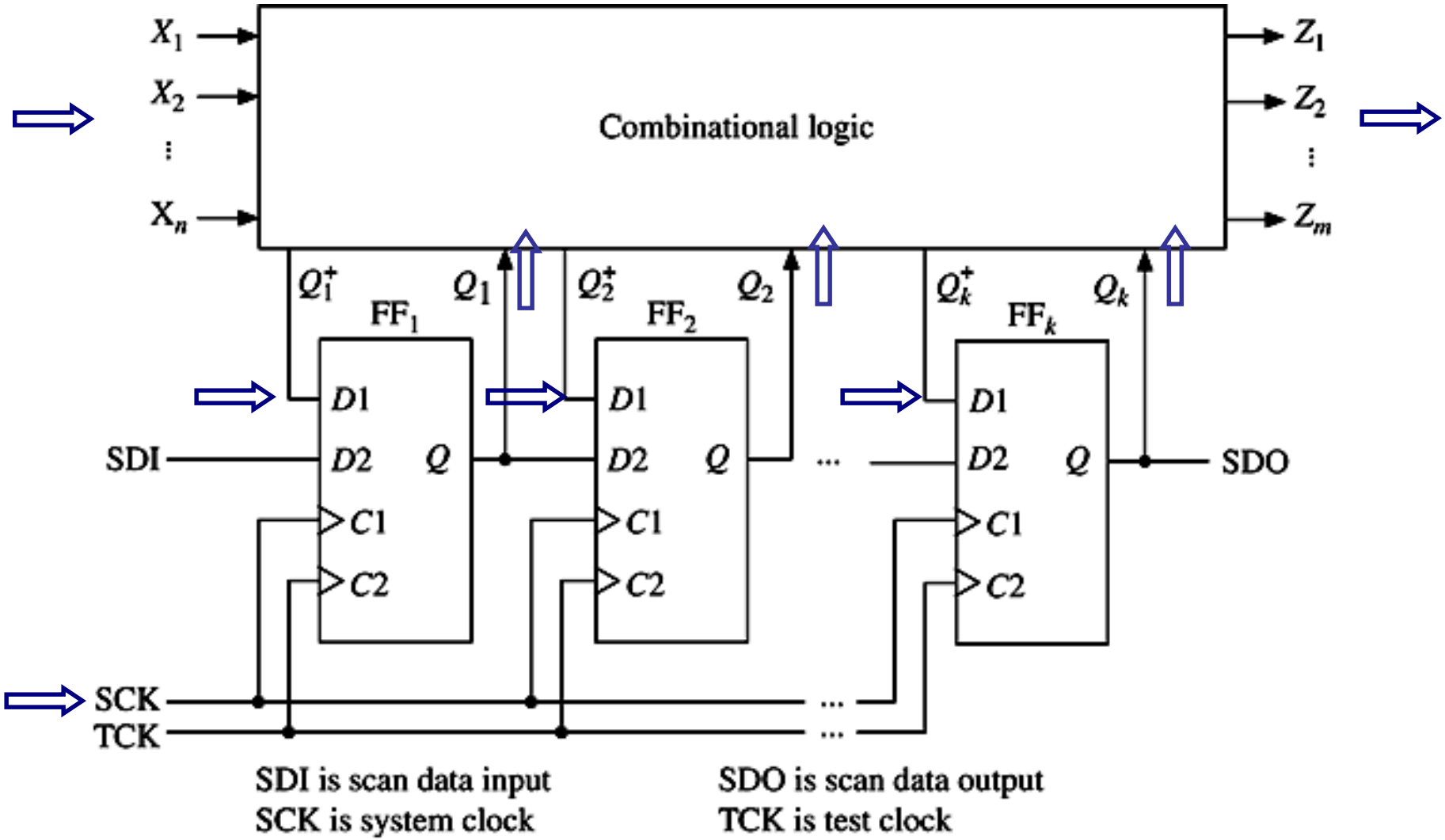
# Scan Testing

- **Instead of observing outputs, observe the state of flip-flops**
- How can we observe the state of all flip-flops without using up a large number of pins on the IC?
  - Create parallel to serial shift-register out of the flipflops and use a single serial output pin



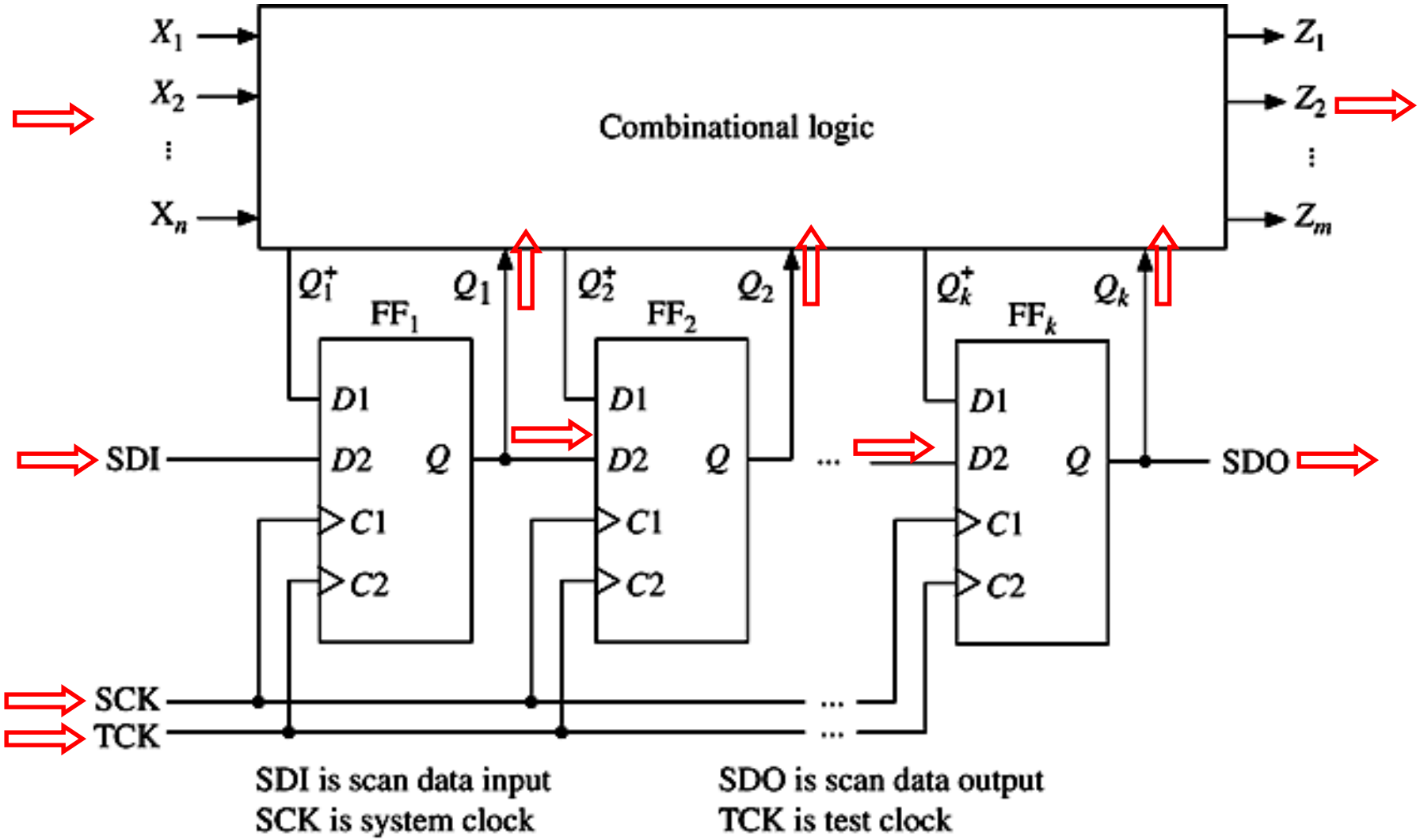
# Scan Testing

- Scan Path Test Circuit (Normal Operation)



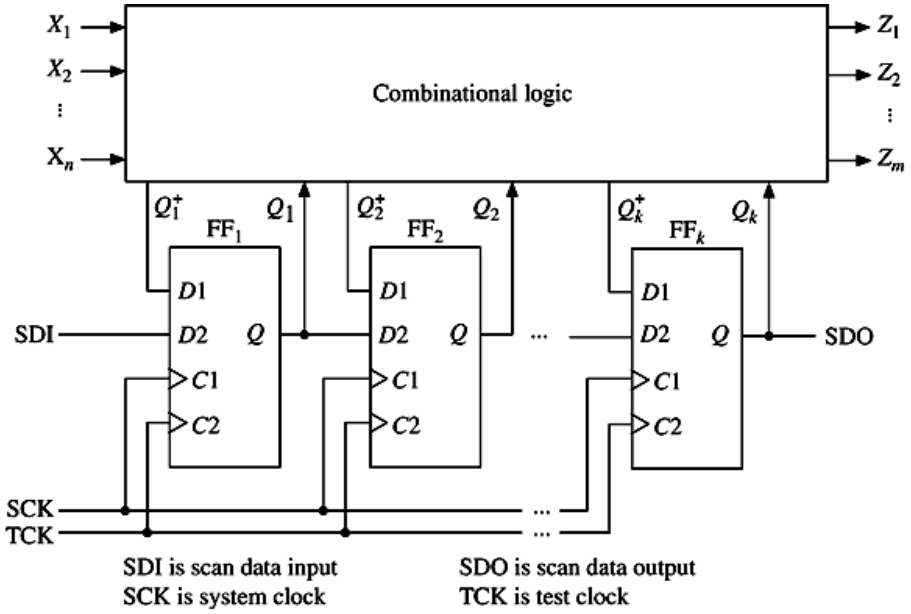
# Scan Testing

- Scan Path Test Circuit (Test)



# Scan Testing

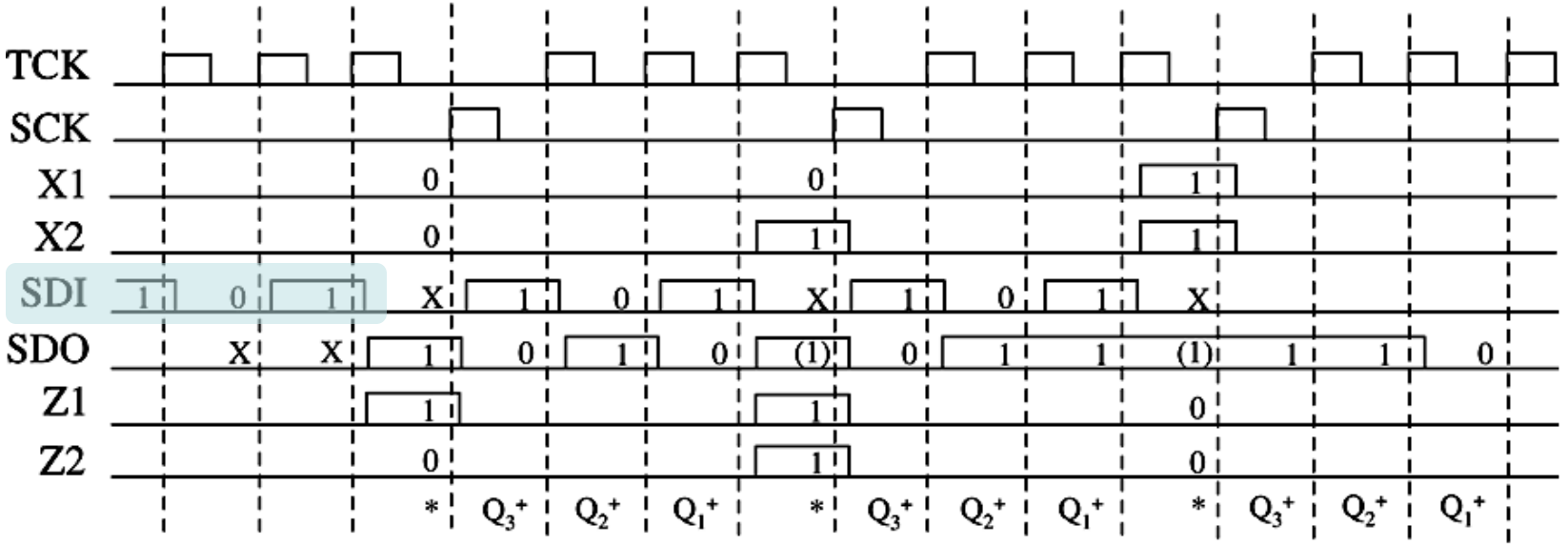
- Scan Path Test Circuit (Test)




- Scan in the test vector $Q_i$ values via **SDI** using test clock **TCK**
- Apply the corresponding test values to the $X_i$ inputs
- Verify output $Z_i$ values
- Apply one clock pulse **SCK** to store new values of $Q_i^+$ into the FFs
- Scan and verify Qi values by pulsing test clock TCK
- Repeat the above for each test vector

# Scan Testing: Example

| | $Q_1^+Q_2^+Q_3^+$ | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |

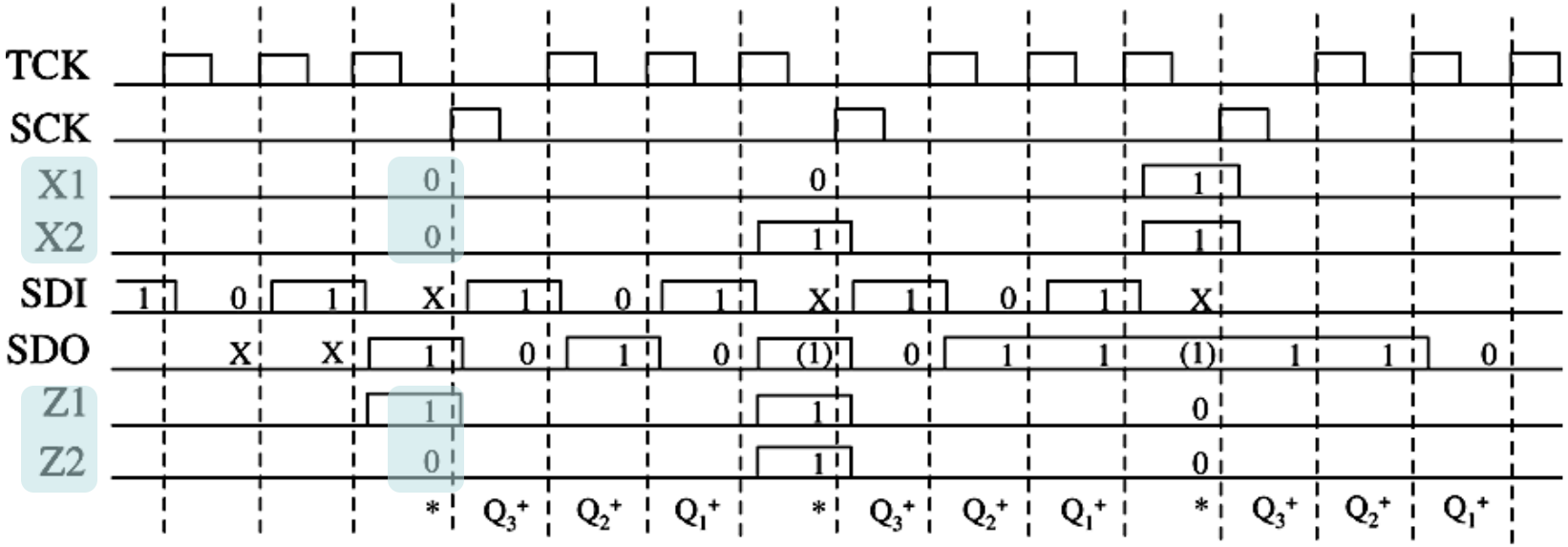



- Shift **101** using **TCK** via **SDI**: $Q_3$ (LSB) First, $Q_1$ (MSB) last

**Note that 101 is a unique pattern which does not rely on $X_1$ and $X_2$**

# Scan Testing: Example

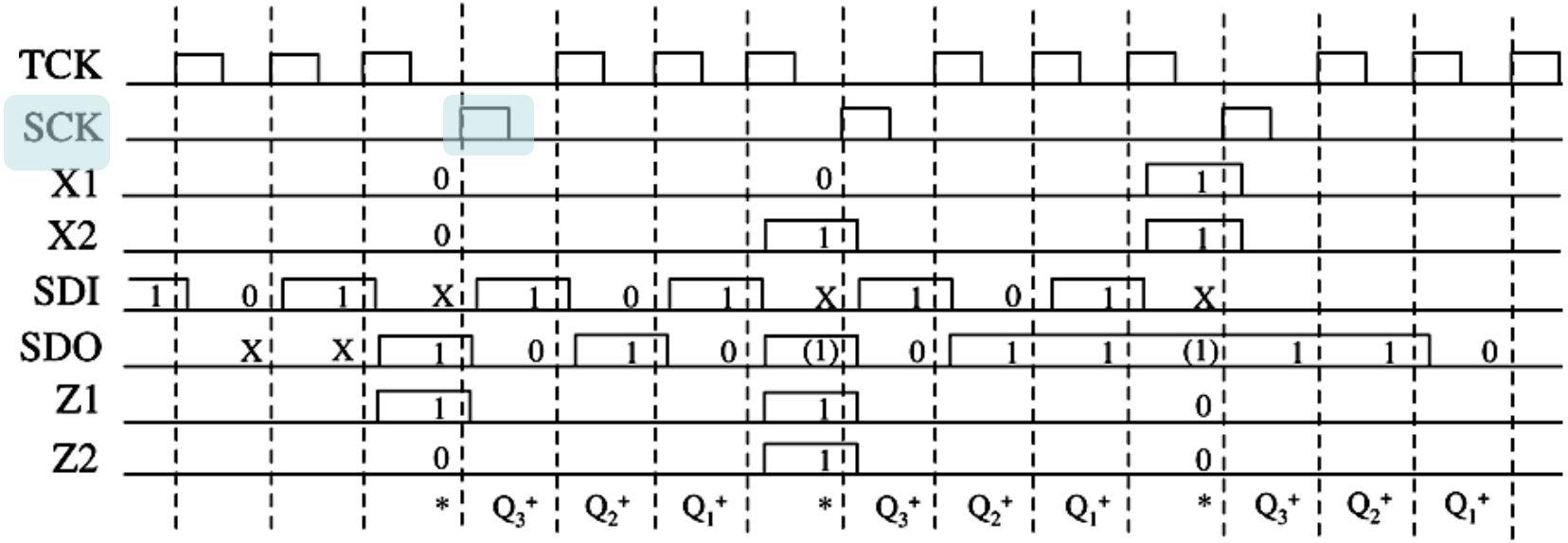
| | $Q_1^+Q_2^+Q_3^+$ | | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |




- Apply input $X_1X_2$=00, verify that $Z_1Z_2$=10

# Scan Testing: Example

| | | $Q_1^+Q_2^+Q_3^+$ | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |

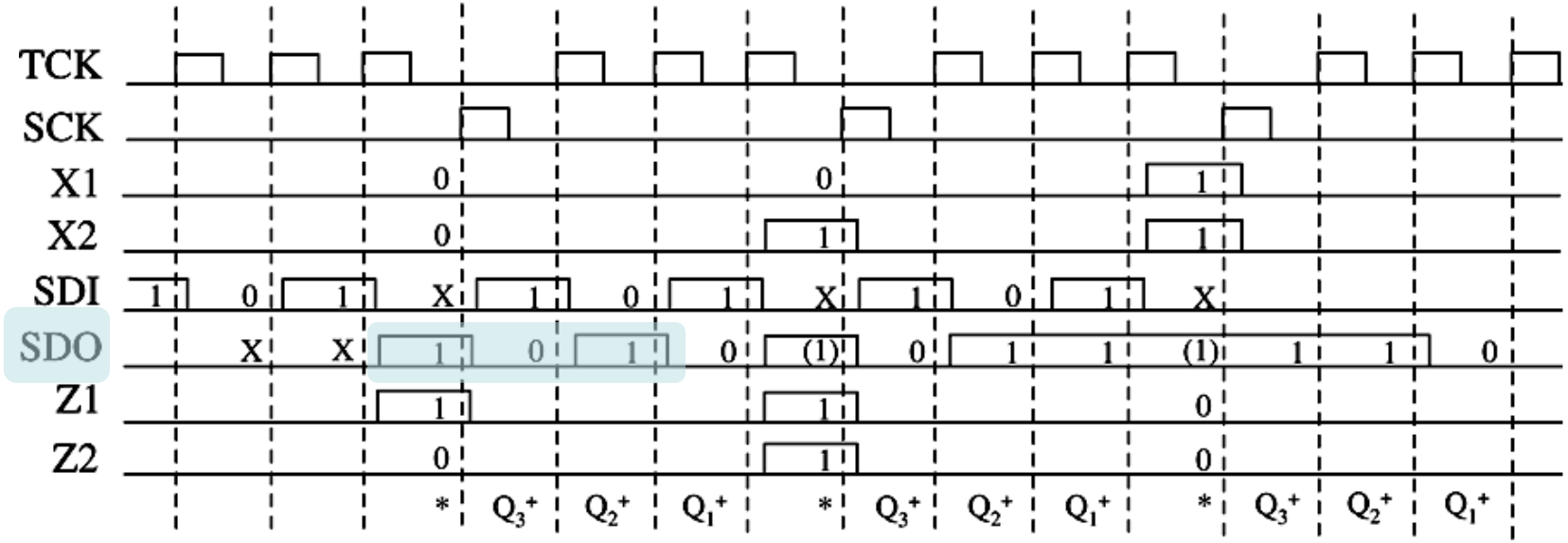



*Read output (output at other times not shown)

- Single pulse on **SCK** to advance circuit to state **010**

# Scan Testing: Example

| | $Q_1^+Q_2^+Q_3^+$ | | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |

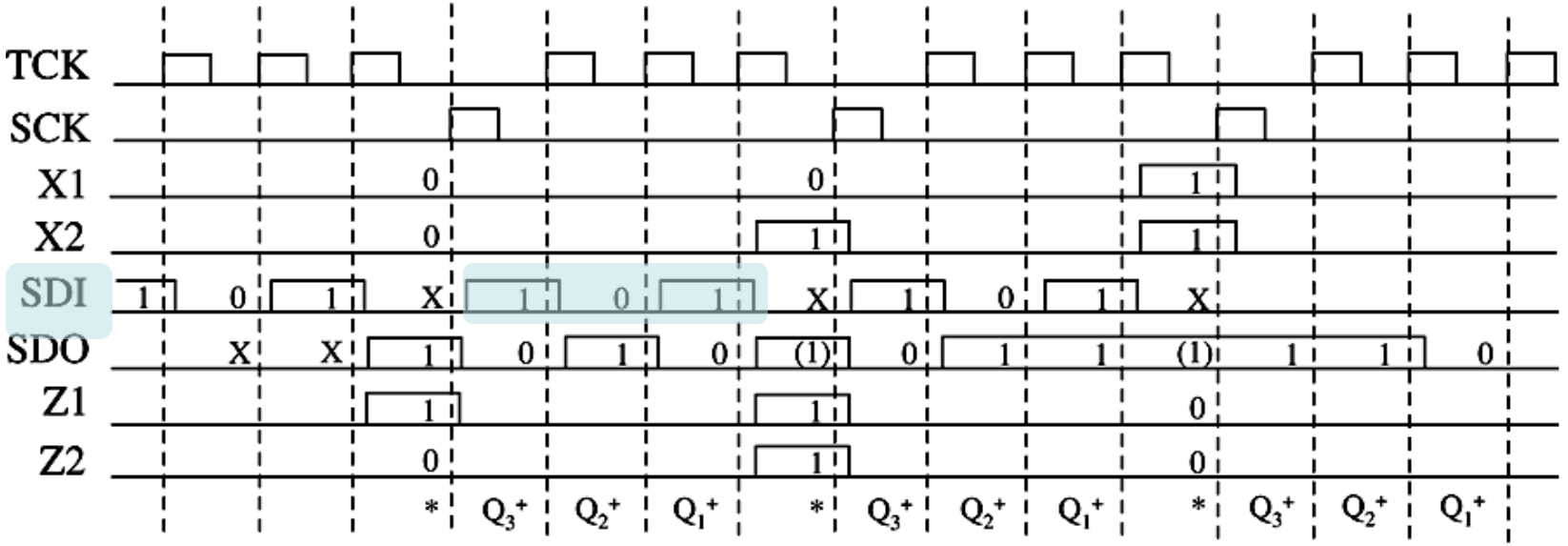



*Read output (output at other times not shown)

- Verify that **SDO** shows the previous state **101**

# Scan Testing: Example

| | $Q_1^+Q_2^+Q_3^+$ | | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |



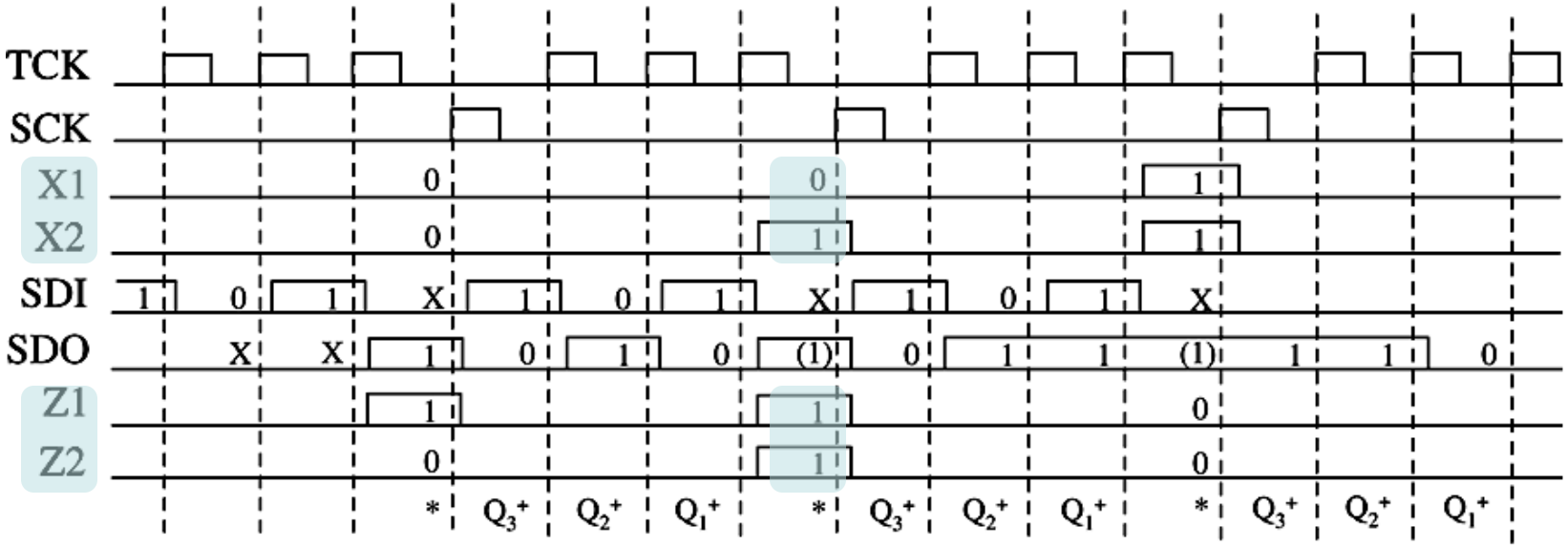

*Read output (output at other times not shown)

- Shift **101** using **TCK** via **SDI**: $Q_3$ (LSB) First, $Q_1$ (MSB) last

**Note that 101 is a unique pattern which does not rely on $X_1$ and $X_2$**

# Scan Testing: Example

| | $Q_1^+Q_2^+Q_3^+$ | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |

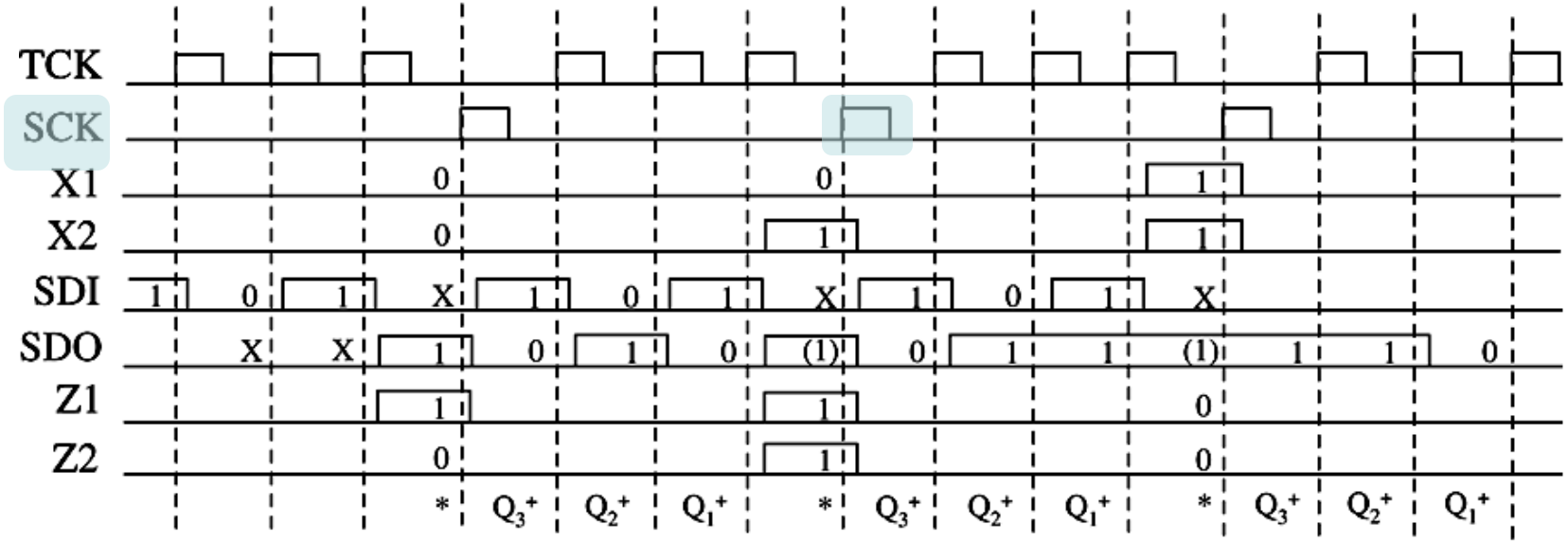



*Read output (output at other times not shown)

- Apply input $X_1X_2$=01, verify that $Z_1Z_2$=11

# Scan Testing: Example

| | $Q_1^+Q_2^+Q_3^+$ | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |

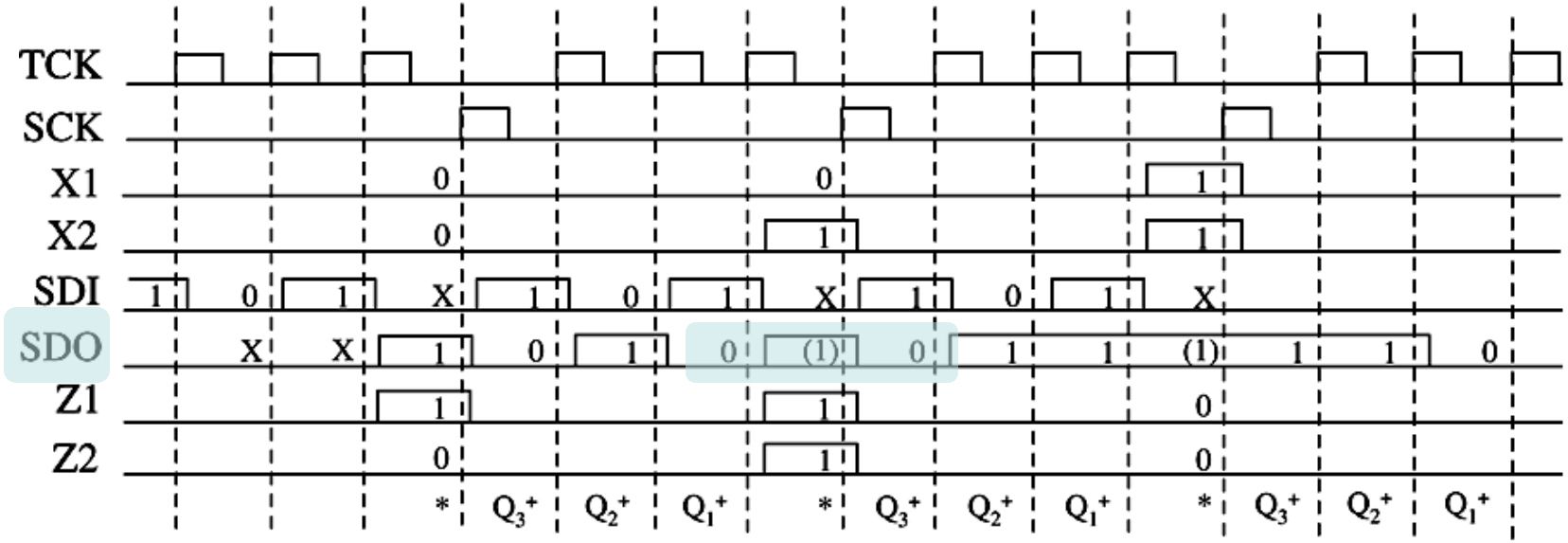



*Read output (output at other times not shown)

- Single pulse on **SCK** to advance circuit to state **110**

# Scan Testing: Example

| | $Q_1^+Q_2^+Q_3^+$ | | | | | $Z_1Z_2$ | | | |
|---|---|---|---|---|---|---|---|---|---|
| $Q_1Q_2Q_3$ | $X_1X_2$= | 00 | 01 | 11 | 10 | 00 | 01 | 11 | 10 |
| 101 | | 010 | 110 | 011 | 111 | 10 | 11 | 00 | 01 |



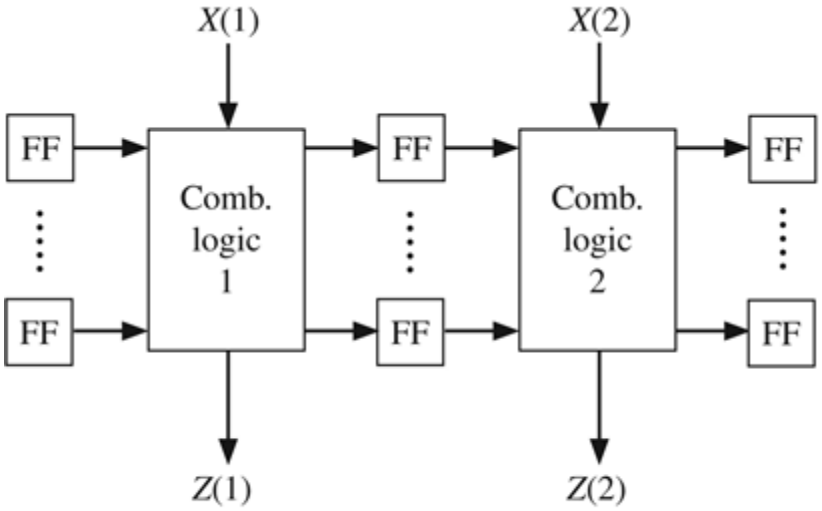

*Read output (output at other times not shown)

- Verify that **SDO** shows the previous state **010**

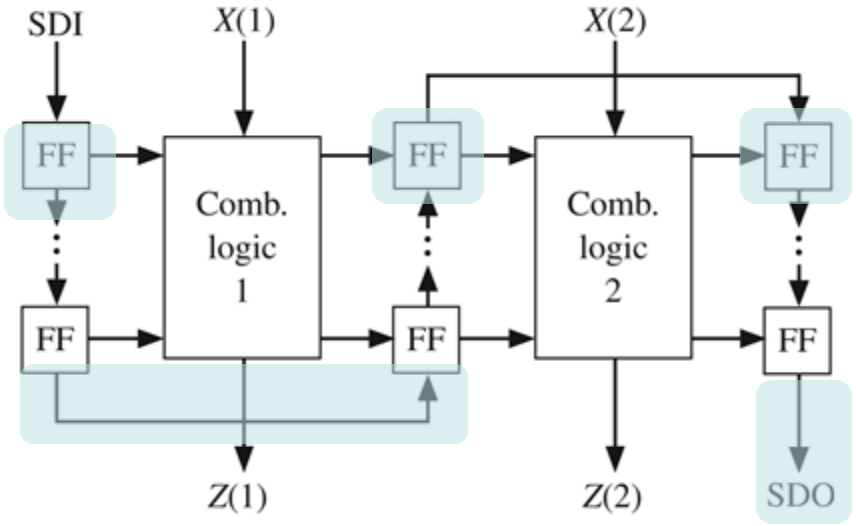
**Continue to the next state ...**

# Scan Testing for an IC

- Replace flip-flops with two-port flip-flops
- **Connect them together in a chain and create Scan chain (shift register)**




(a) Without scan chain

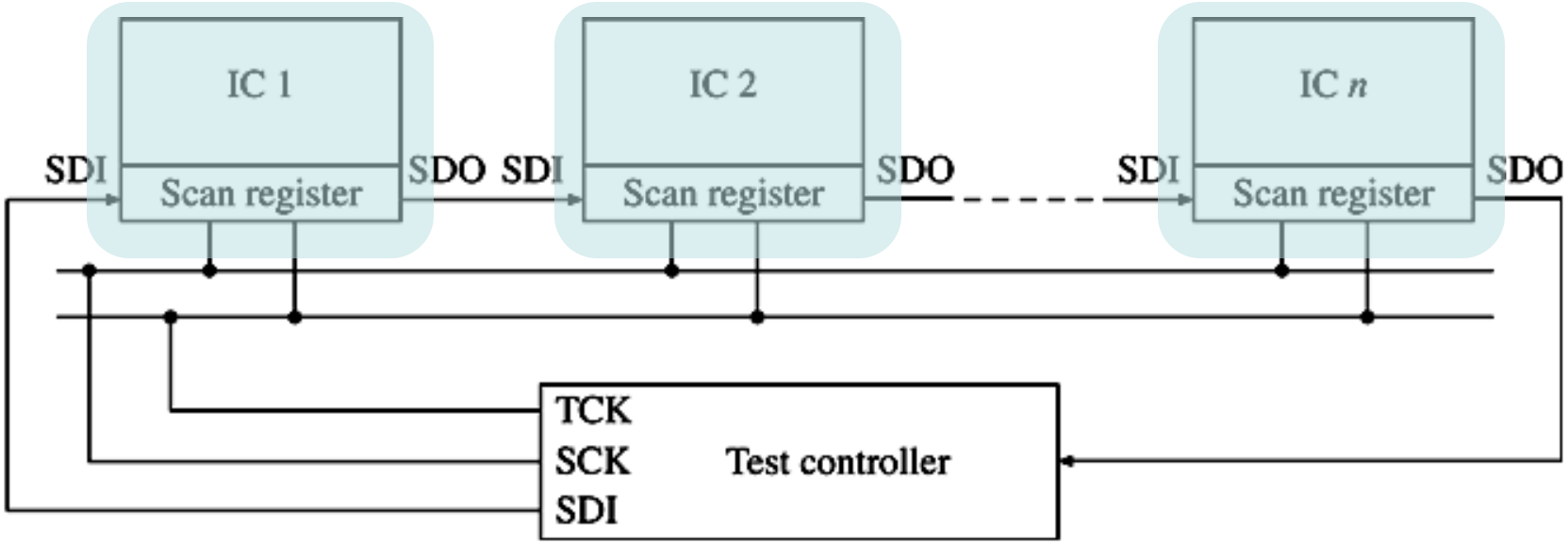



(b) With scan chain added

# Scan Testing for Multiple ICs on single PCB



- Chain scan registers together
- **The whole PCB can be scanned using single serial port**



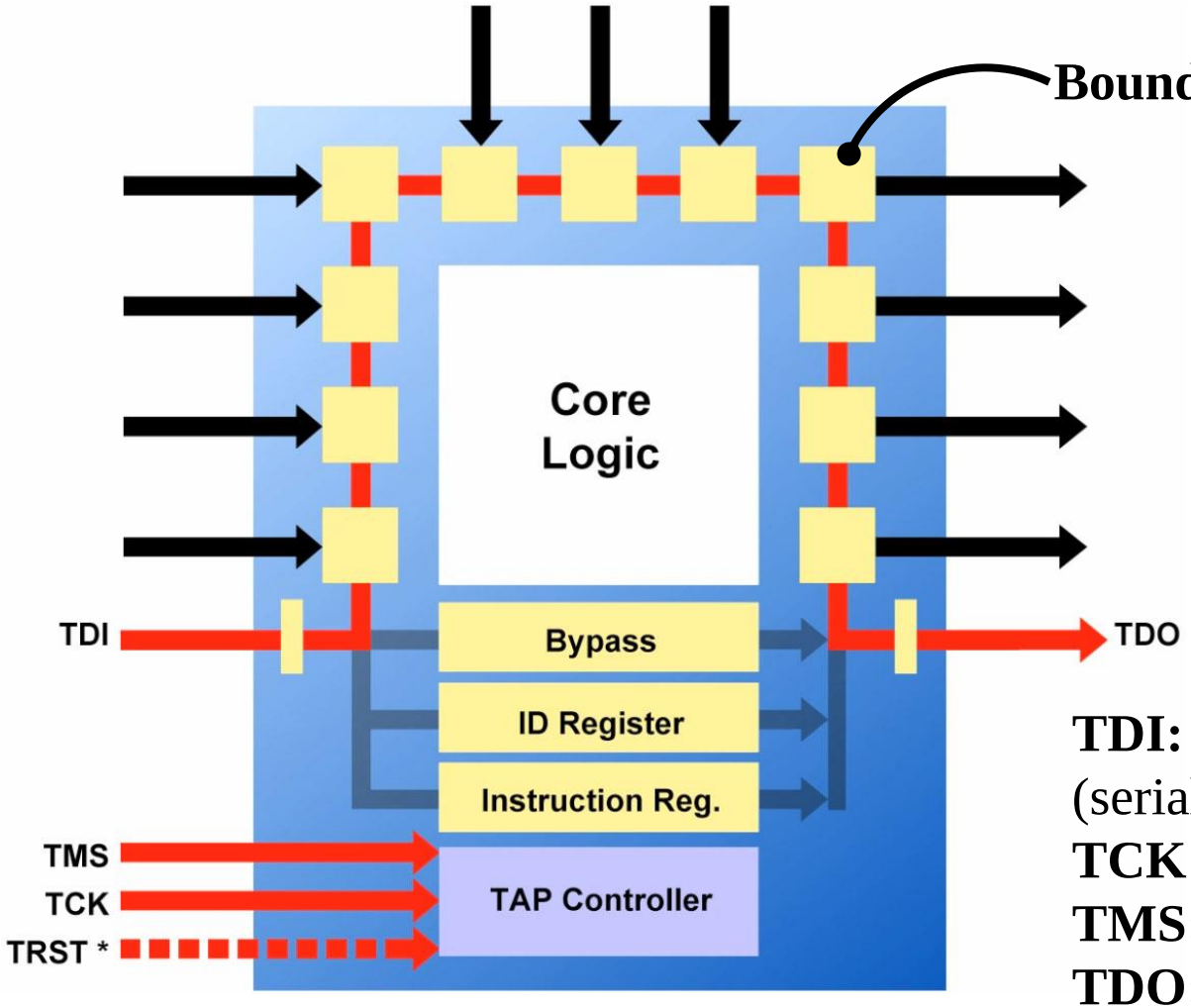
# Boundary Scan

# Boundary Scan

- **Boundary Scan was developed to test a complex PCB with many ICs**

- **JTAG (Joint Test Action Group**) developed the standard for boundary scan
  - Standard Test Access Port and Boundary-Scan Architecture
  - **Multiple ICs can be linked together on PCB and tested using few pins on the edge**

# Boundary Scan

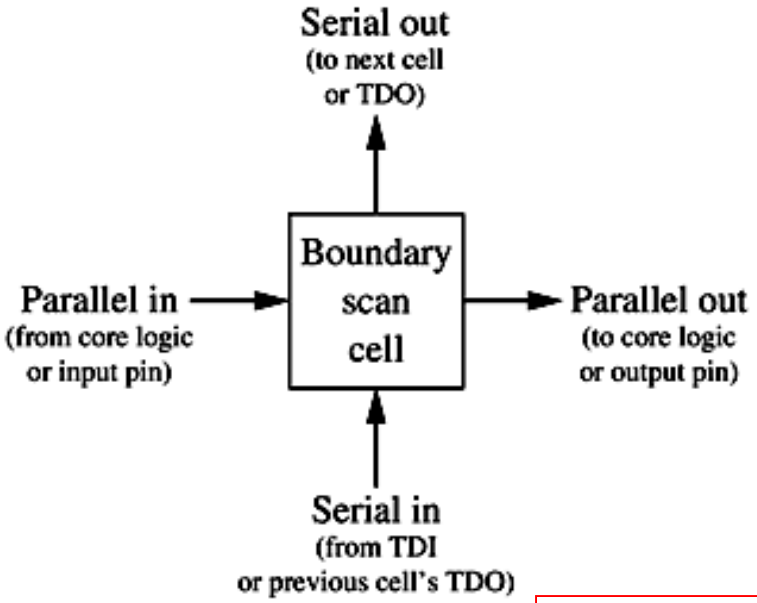


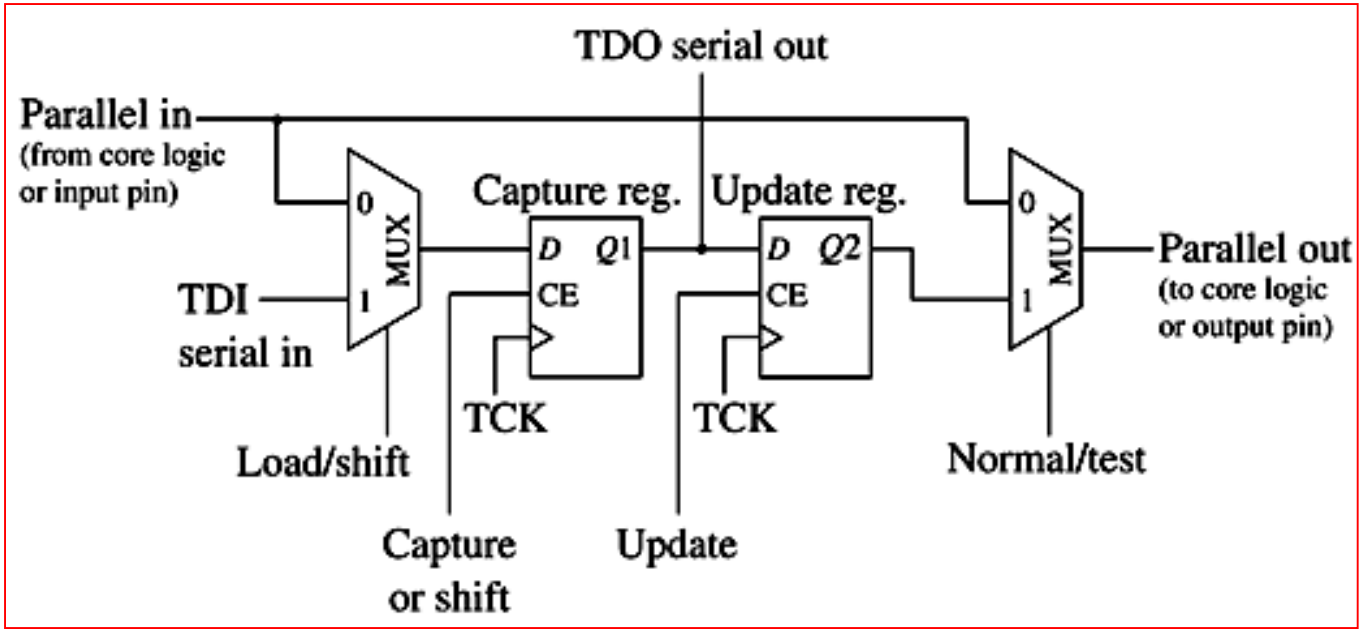

**One cell of boundary scan shift register (BSR) is placed between each I/O and the core logic**

**TDI:** Test data input
(serial input into the shift register)
**TCK:** Test Clock
**TMS:** Test Mode Select
**TDO:** Test Data Output
(serial output from the shift register)
**TRST:** Test Reset
(resets TAP controller and test logic)

# Boundary Scan Cell

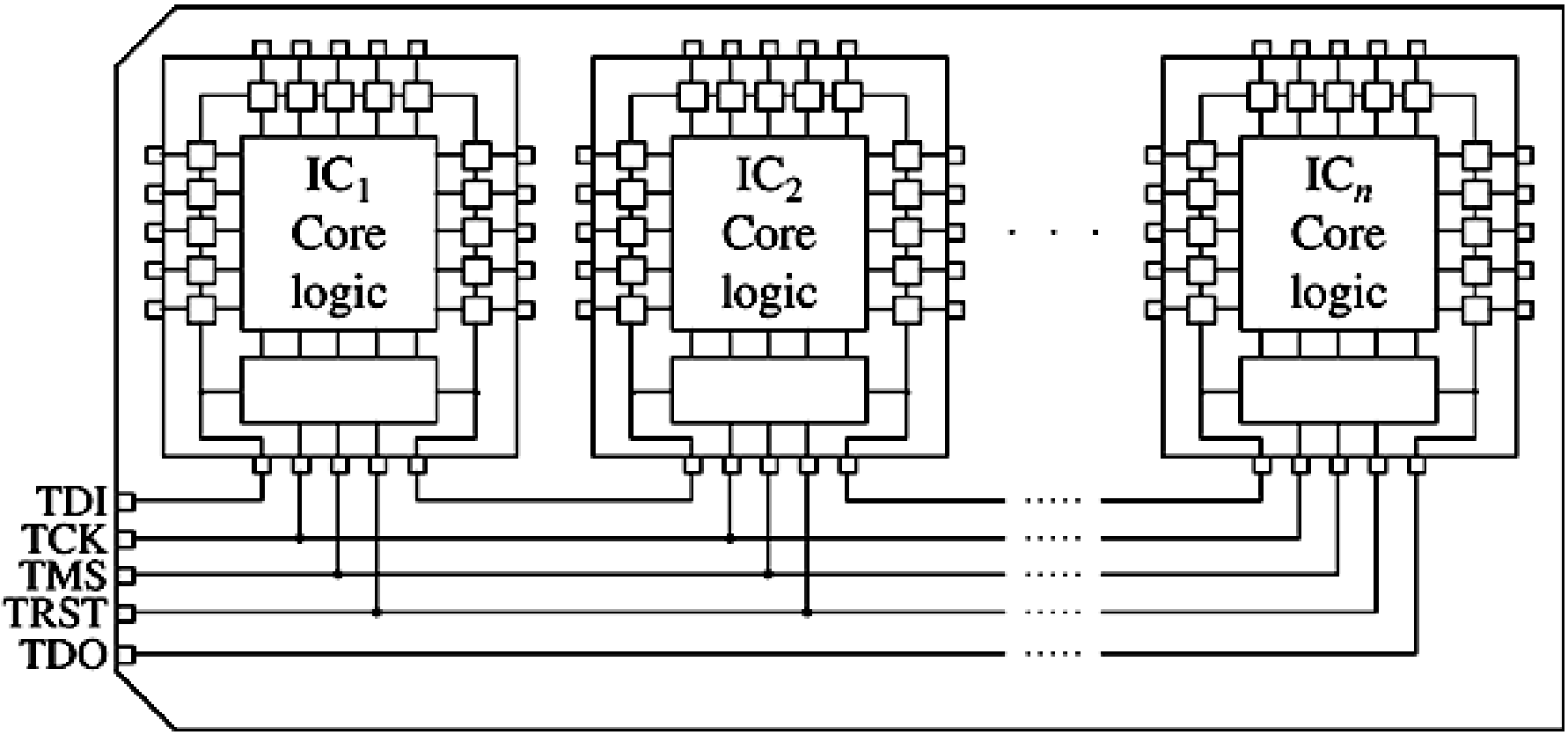




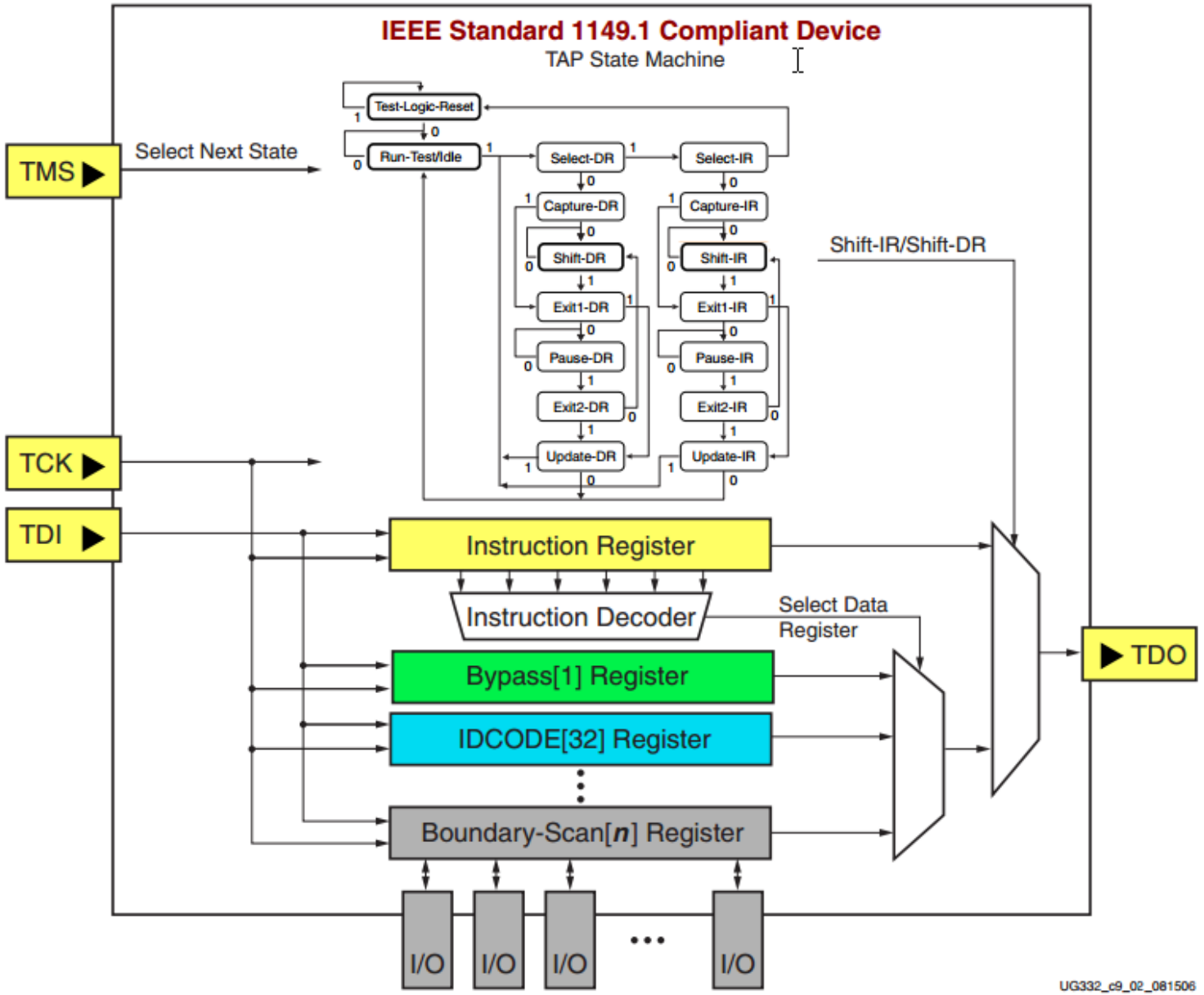
# PCB with Boundary Scan ICs





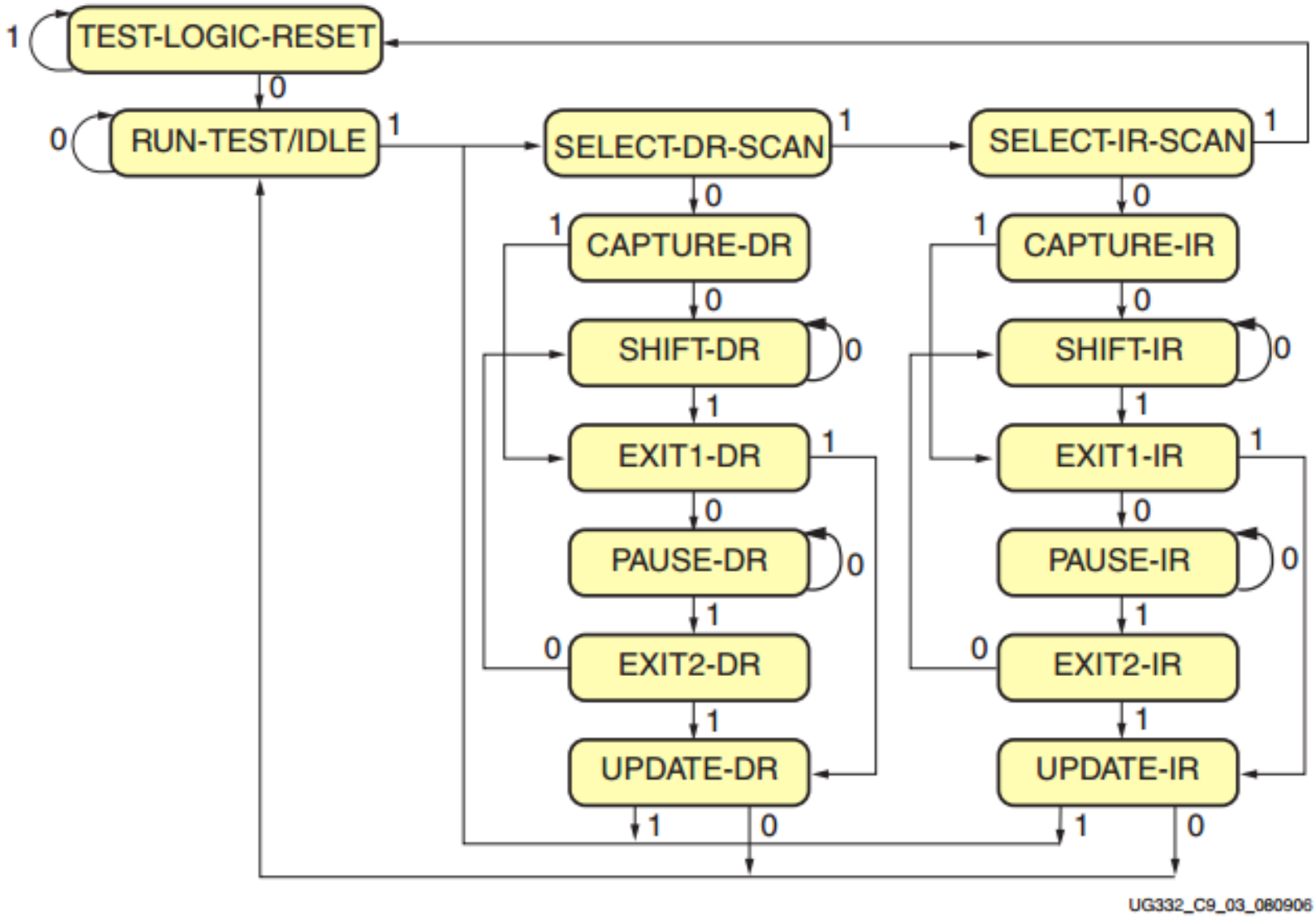
# Boundary Scan Architecture





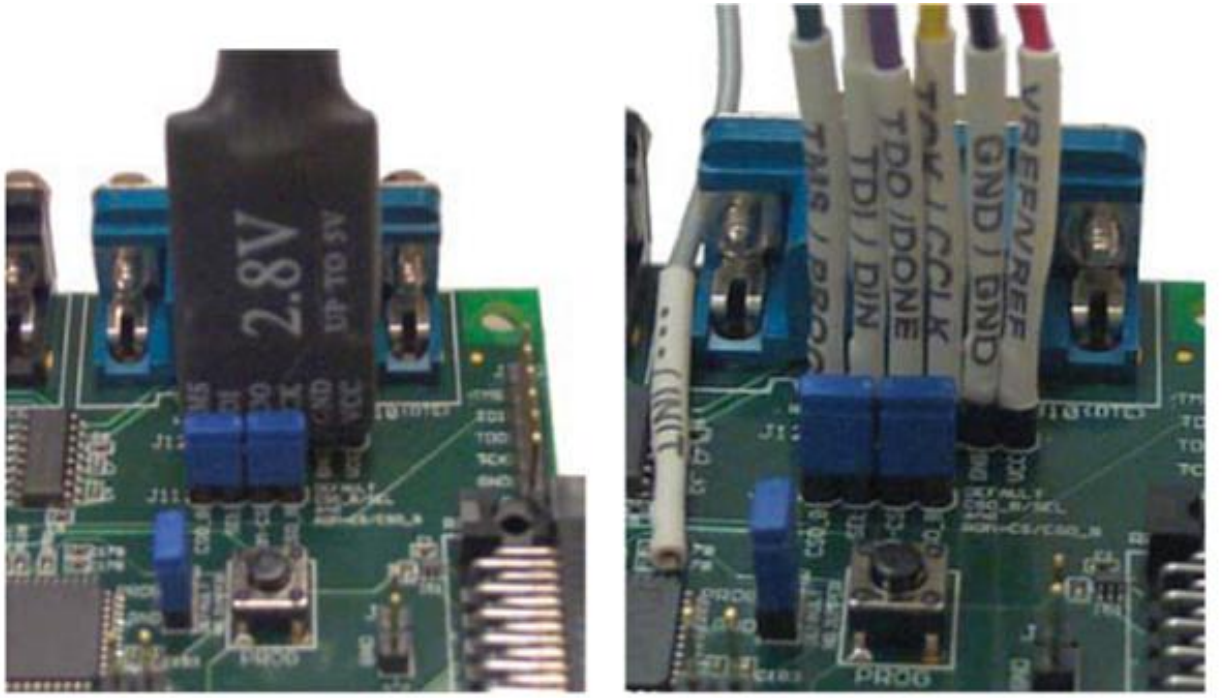
# TAP Controller State Machine





# Boundary Scan in Xilinx Spartan 3E PCB






a) JTAG3 Parallel Connector

b) Parallel Cable III or Parallel Cable IV with Flying Leads

UG230_c15_14_030206

*Figure 12-15:* **Attaching a JTAG Parallel Programming Cable to the Board**

*Table 12-2:* **Cable Connections to J12 Header**

| Cable and Labels | Connections | | | | | |
|---|---|---|---|---|---|---|
| J12 Header Label | SEL | SDI | SDO | SCK | GND | VCC |
| JTAG3 Cable Label | TMS | TDI | TDO | TCK | GND | VCC |
| Flying Leads Label | TMS/ PROG | TDI/ DIN | TDO/ DONE | TCK/ CCLK | GND/ GND | VREF/ VREF |

# Boundary Scan in Xilinx Spartan 3E PCB

## Does this look familiar?