Practical Scan for ASIC Designers

Anand J. Bariya

Testing Integrated Circuits

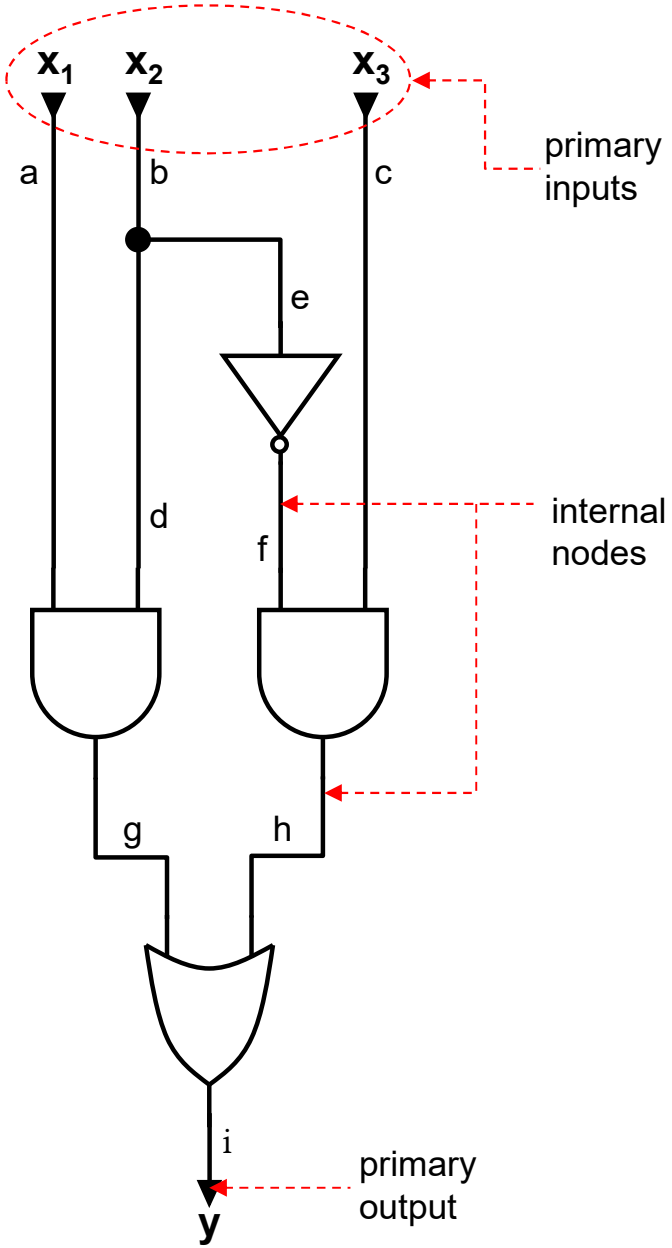
- After manufacture, chips are tested to check that they are free of manufacturing defects
  - testing is done on Automated Test Equipment (ATE), a.k.a. “testers”
  - signals are sent through primary inputs, and primary outputs checked for expected values
    - cannot probe internal nodes
    - there can be millions of internal nodes: defect on any one can cause a chip to fail

- An IC is tested multiple times before being shipped to customer:
  1. while still on the wafer, before die are cut ("die sort"): probe card
     - detect & discard devices that have defects due to wafer processing
  2. after packaging ("final test"): load board
     - detect & discard devices that have defects introduced during packaging (a.k.a. "assembly")
  3. after "burn in":
     - detect & discard "marginal" devices that fail after going through a stress treatment
       - would otherwise fail early in use ("infant mortality")

Detecting Defects

- An IC is considered defective if any node in the circuit has a defect (“fault”)
  - checking for a defect on every node of the IC ensures the IC is exhaustively tested
    - difficult to guarantee exhaustive testing with direct functional vectors; also test time can get very large
- “fault model”: indicates how a fault affects a node
  - Cannot determine how to detect a faulty node without a fault model
- “stuck-at” fault model is most common:
  - Assume fault at a node causes it to be stuck to VDD (“stuck-at-1”) or to GND (“stuck-at-0”)
  - Stuck-at fault can be detected by simply trying to toggle the node (from 0 to 1, and 1 to 0) and observing if the toggling happens
  - Many defects that do not strictly result in nodes being stuck at 0 or 1 also end up being detected by tests based on the “stuck-at” fault model

Detection of Stuck-At Faults in Combinational Logic

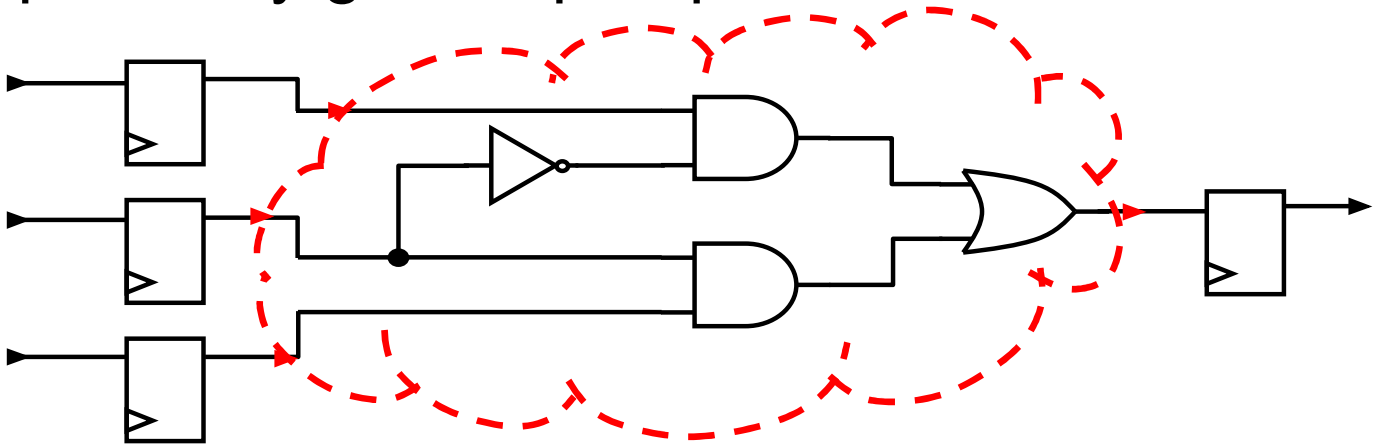


Controllability and Observability

- Controllability: ability to apply a desired logic value to a node by applying logic values to one or more primary inputs (PIs)
- Observability: ability to deduce the logic value at a node by checking the logic values at one or more primary outputs (POs)
- Necessary condition to test combinational logic: all inputs should be controllable and outputs observable
  - Condition is obviously satisfied if all inputs and outputs are primary

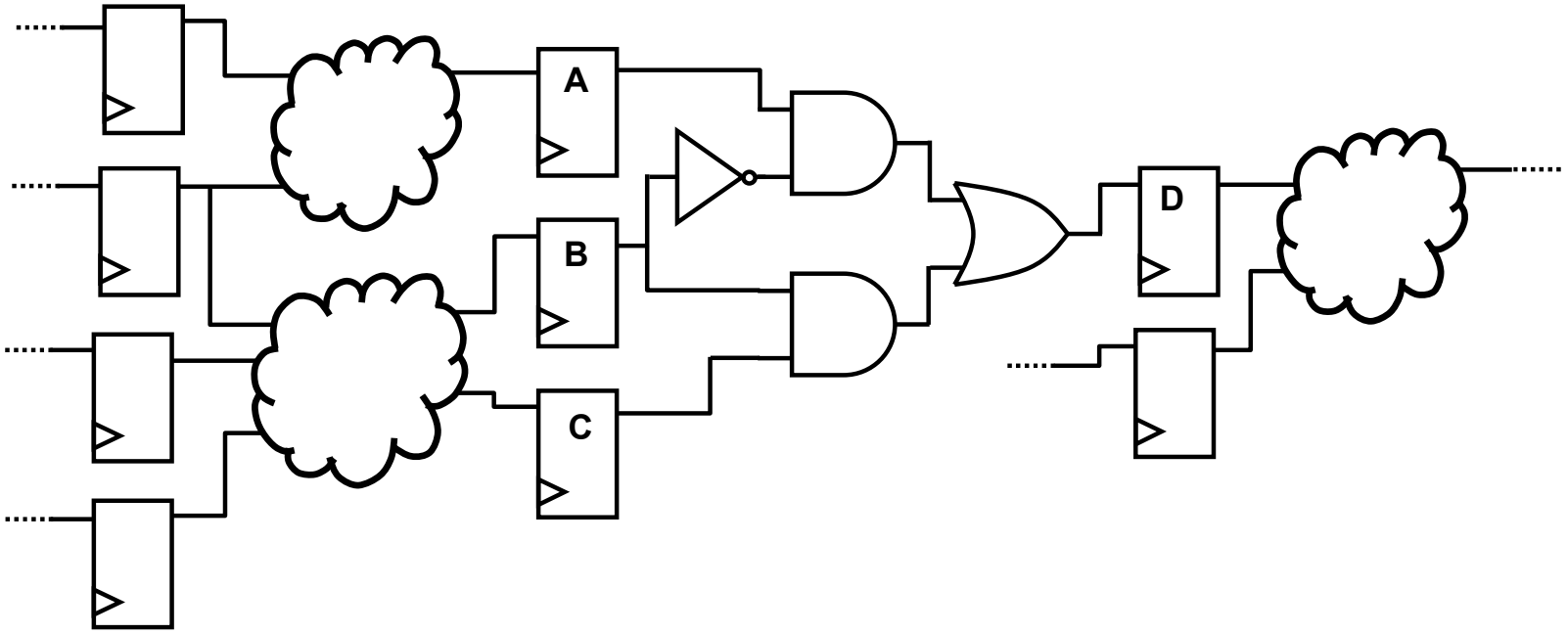
Detecting Stuck-At Faults in Sequential Circuits

- Inputs to combinational logic may come from flip-flops, and outputs may go to flip-flops:



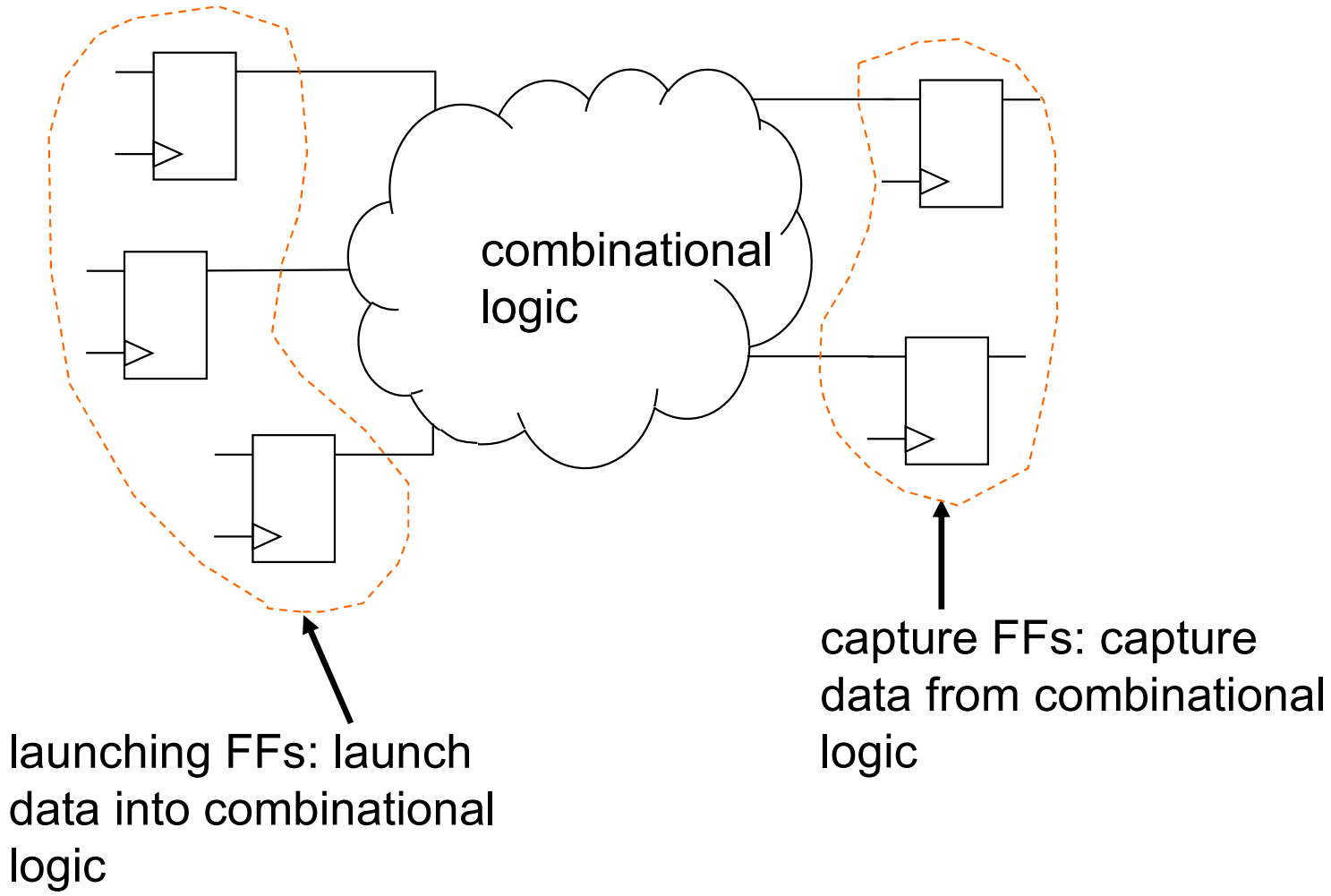
- If FFs at inputs are controllable, and FFs at outputs are observable, combinational logic in sequential circuit can be tested
- Circuit above satisfies this condition because:
  - FFs A, B & C: directly connected to PIs, therefore controllable
  - FF D: directly connected to PO, therefore observable

Detecting Faults in Deeply Pipelined Sequential Logic



- Flip-flops A, B & C: difficult to control
  - inputs come from other FFs thru combo logic
  - many pipeline stages between primary inputs & FFs
- Flip-flop D: difficult to observe
  - lot of combo logic and pipeline stages between FF and primary outputs

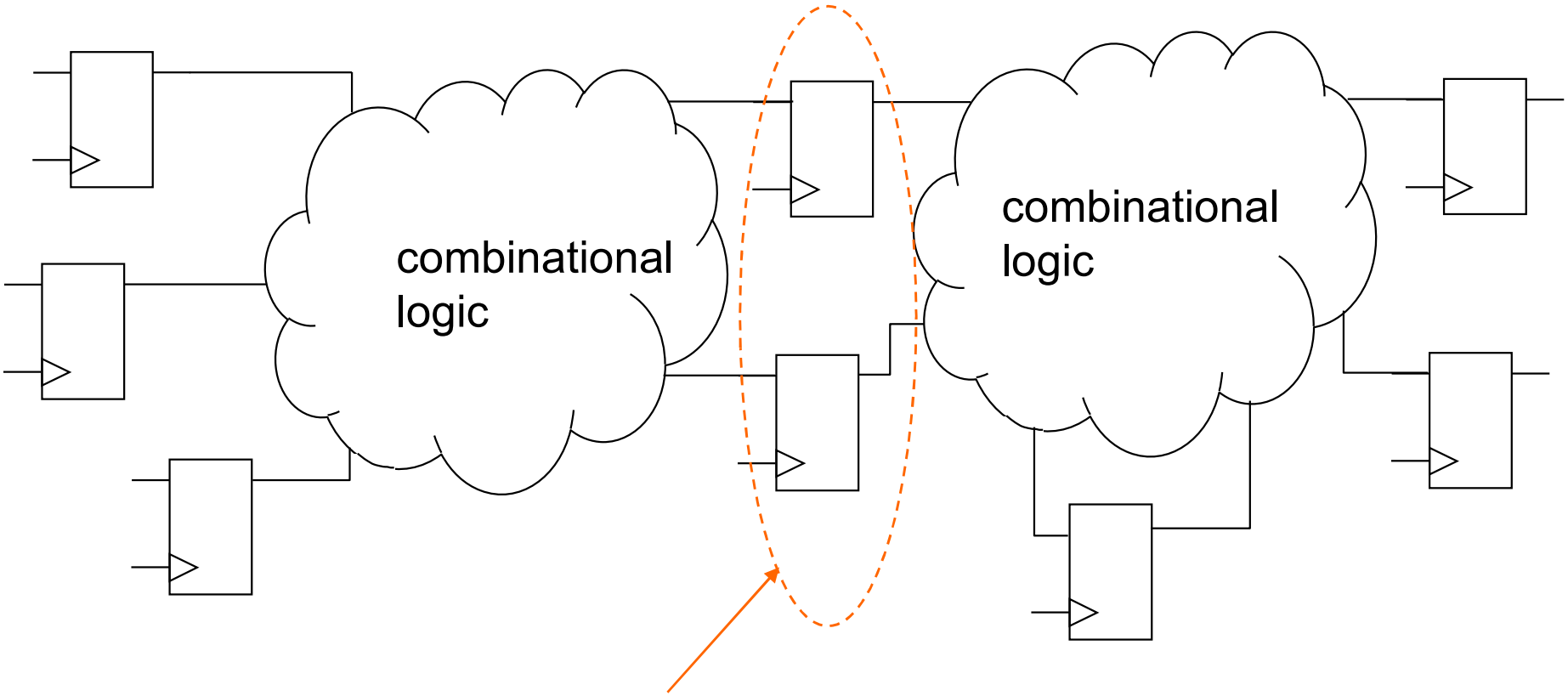
Sequential Circuit as seen from the Point of View of Test:



FFs provide inputs to, and capture outputs from, combinational logic

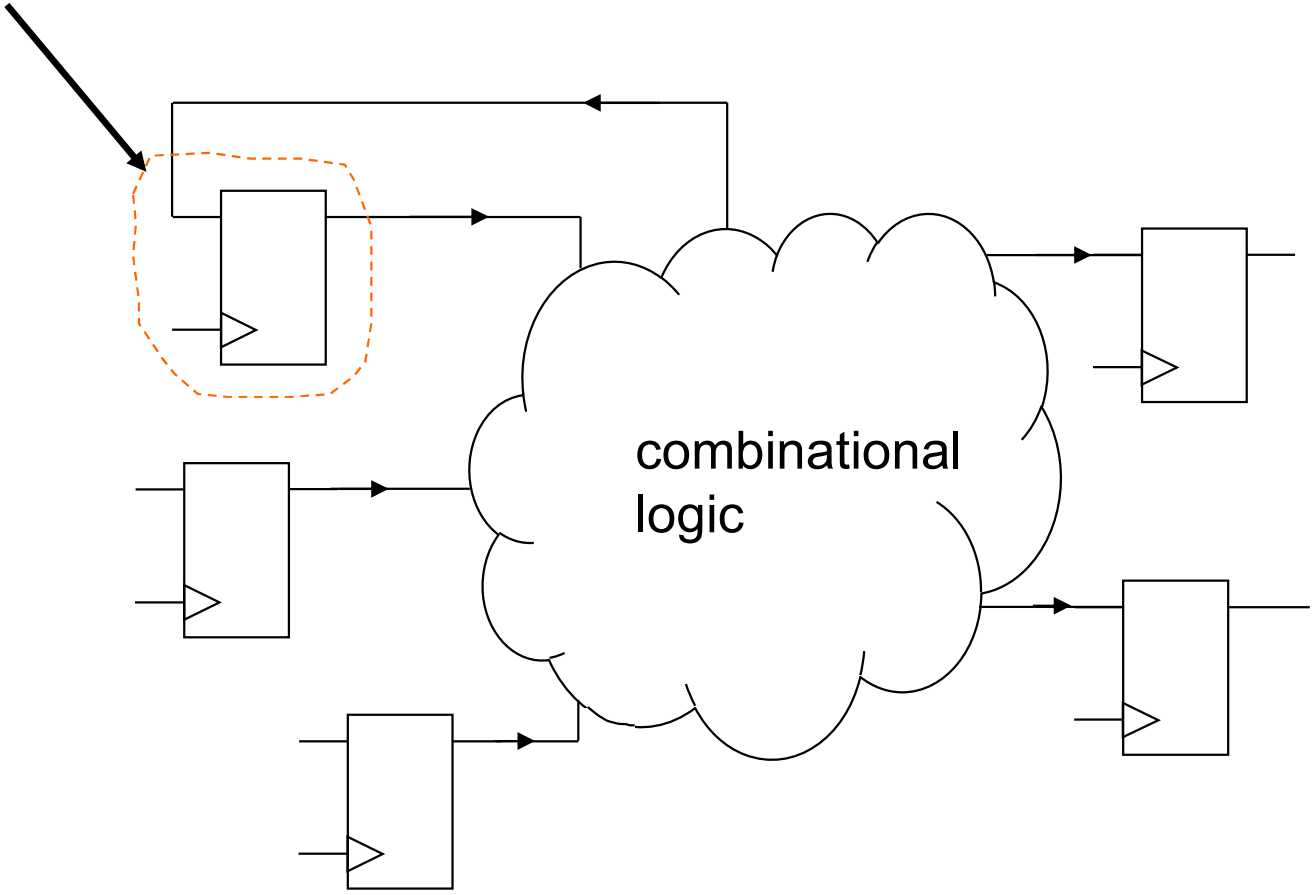
Testing Sequential Circuits

- Typical flip-flop will both launch data into combo logic, and capture data from combo logic:



Sequential Circuit Tested with Scan

- A flip-flop may launch into, and capture from, the same “cloud” of combinational logic



Design For Test (DFT) & Scan

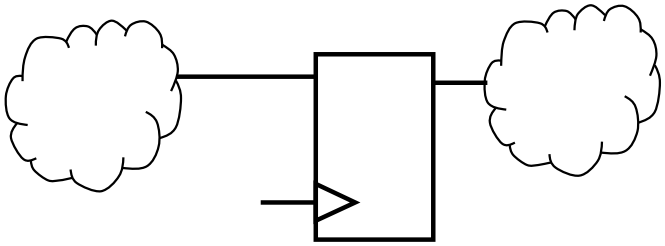
- DFT: the inclusion of circuits or circuit elements that play no role in the functionality of a chip, but exist only to enable testing of the chip
- Scan is one among many DFT techniques
  - Used for comprehensive testing of sequential logic (logic consisting of flip-flops and combinational gates)
    - "Comprehensive" $\Rightarrow$ large fraction of internal nodes (ideally all) are tested

Basic Idea of Scan

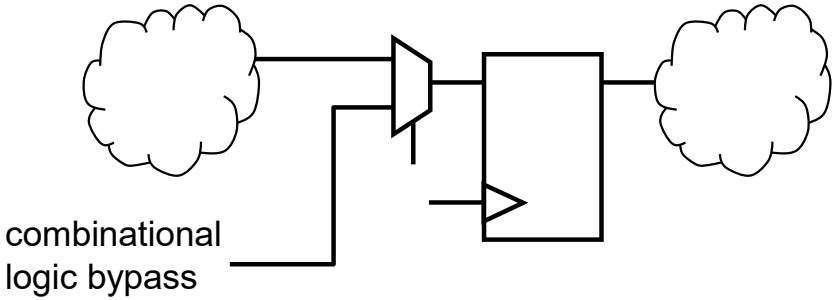
1) Load all flip-flops with known values
2) Launch these values into combinational logic
3) Check if values in capturing FFs are expected values
   - if not, die is defective

Scan Chains

- To load FFs with known values for test, bypass combinational logic



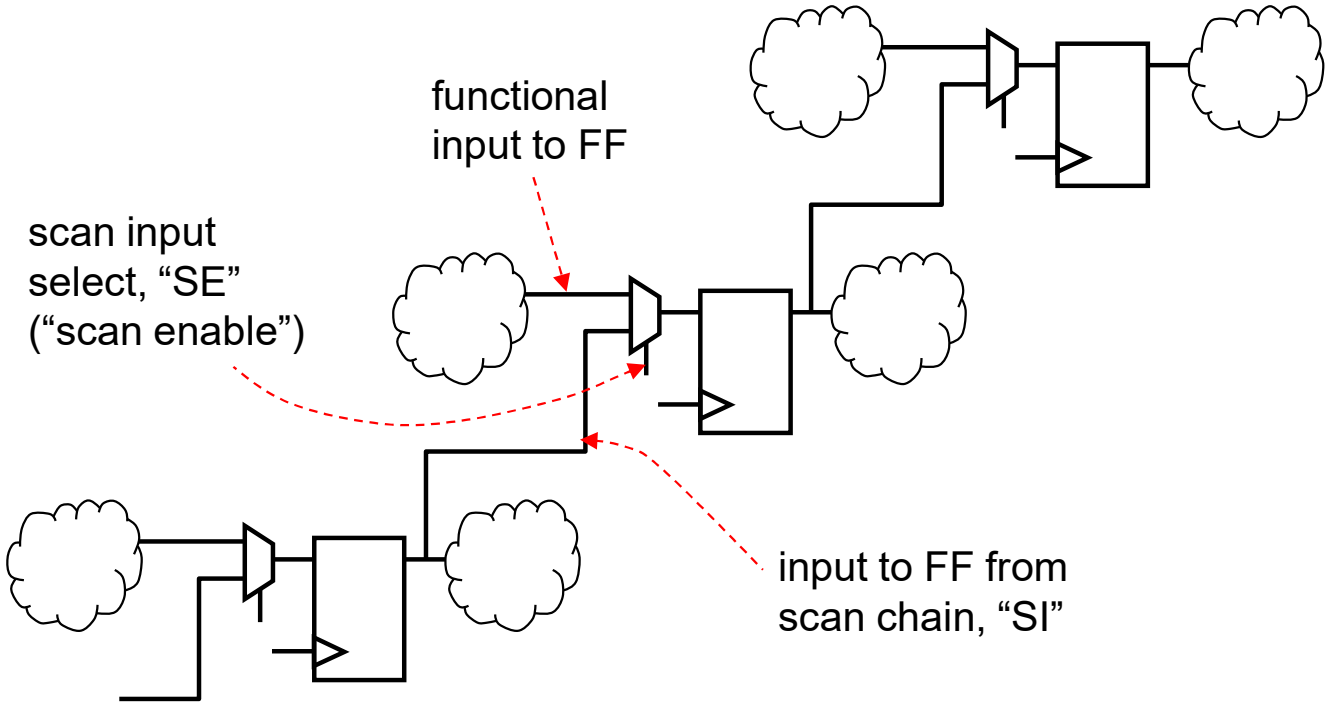
Original Logic



Mux insertion for combinational logic bypass

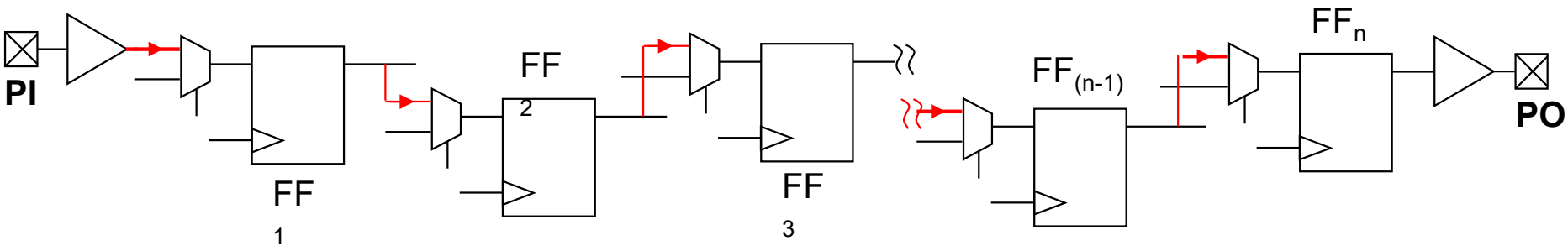
Scan Chains

- String FFs together through the combinational logic bypass to form “scan chains”



Scan Chains

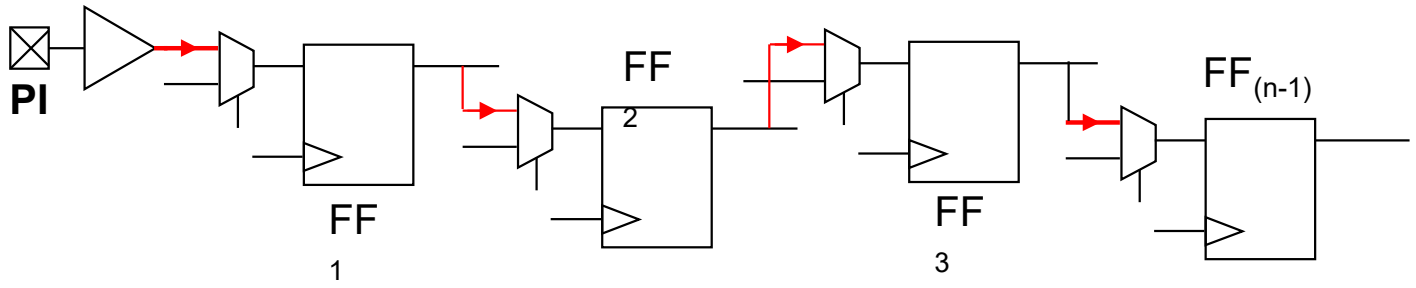
- First FF in scan chain connected to primary input (PI)
- Last FF in scan chain connected to primary output (PO)



Scan chain of length “n”

Scan Shift

- For scan chain of length “n”, all FFs can be set to desired values over “n” clock cycles
  - values fed to scan chain from PI
  - SE = 1 throughout

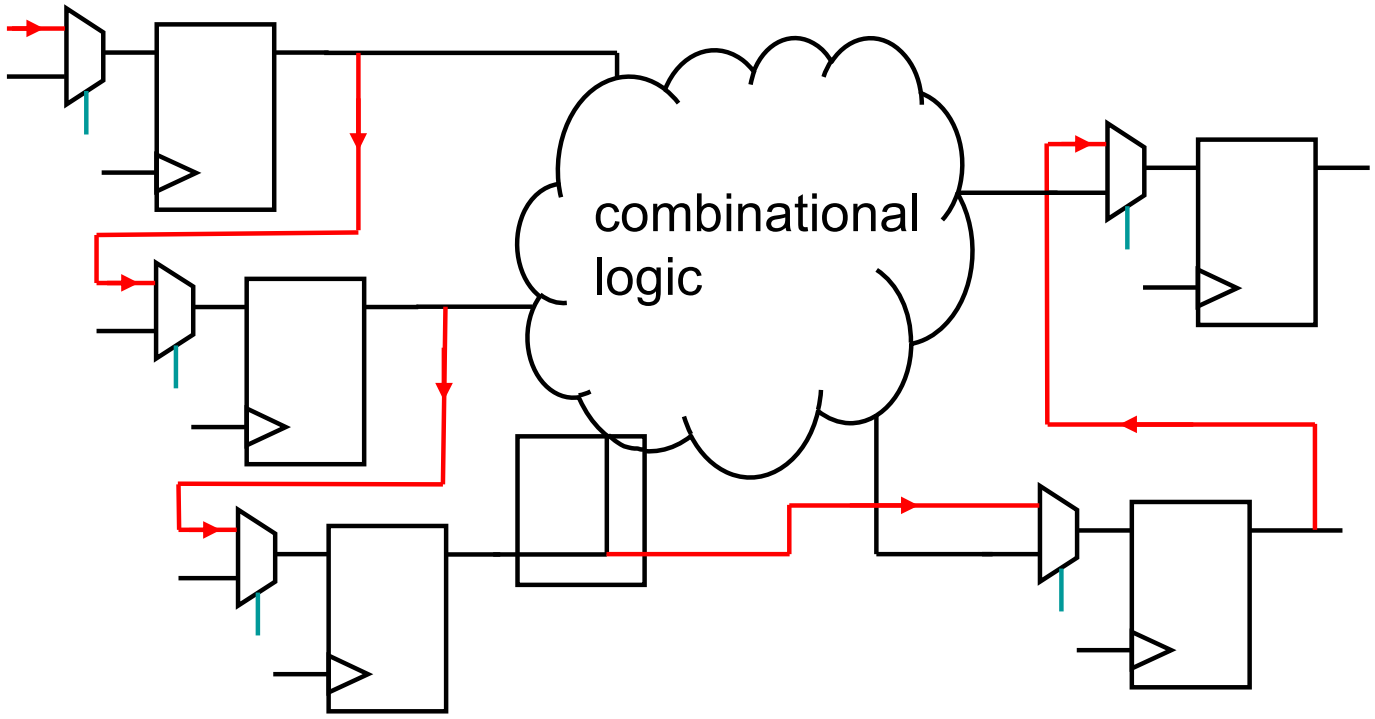


Scan Chains

- To load FFs with known values for test, bypass combinational logic and pipeline stages
- String flip-flops into “scan chains”, and shift values in sequentially from a primary input
  - first flip-flop in scan chain connected to primary input
  - last flip-flop in scan chain connected to primary output
  - for a scan chain with “n” flip-flops, all flip-flops will get loaded in “n” clock cycles

Scan Chains

- Add two-input mux just before FF D-input to enable construction of a scan chain:



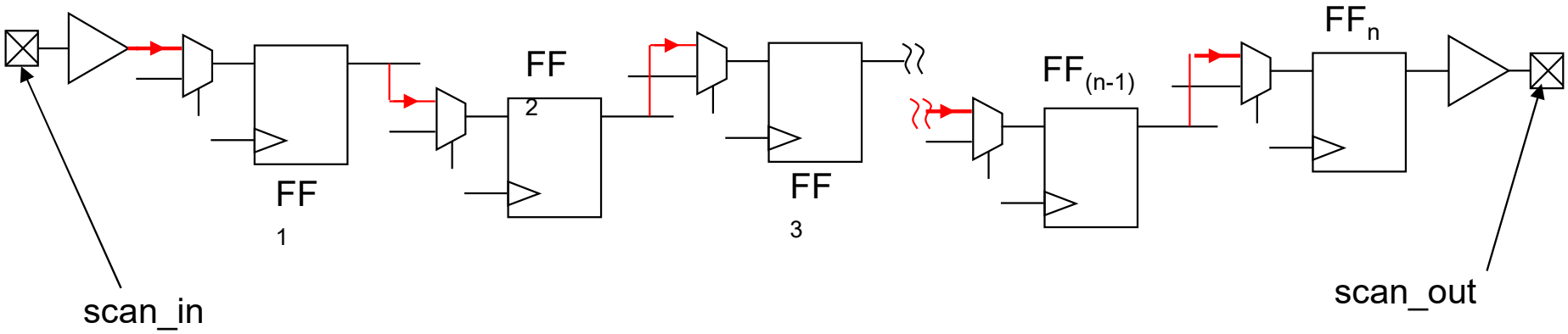
Mux selects of all scan muxes connected together: signal called “scan enable” or, more correctly, “shift enable” (SE)

Basics of Scan

- Scan testing involves the following steps:
  a) shift in: load FFs with pre-determined values
     - takes “n” clock cycles for a chain of length “n”
     - SE = 1
  b) capture: drive shifted in values through combinational logic
     - takes a single clock cycle
     - SE = 0
  c) shift out: observe values captured in FFs
     - takes “n” clock cycles for a chain of length “n”
     - SE = 1
- Steps (a) and (c) can be combined $\Rightarrow$ scan involves two modes:
  i. shift (steps “a” & “c”)
  ii. capture (step “b”)

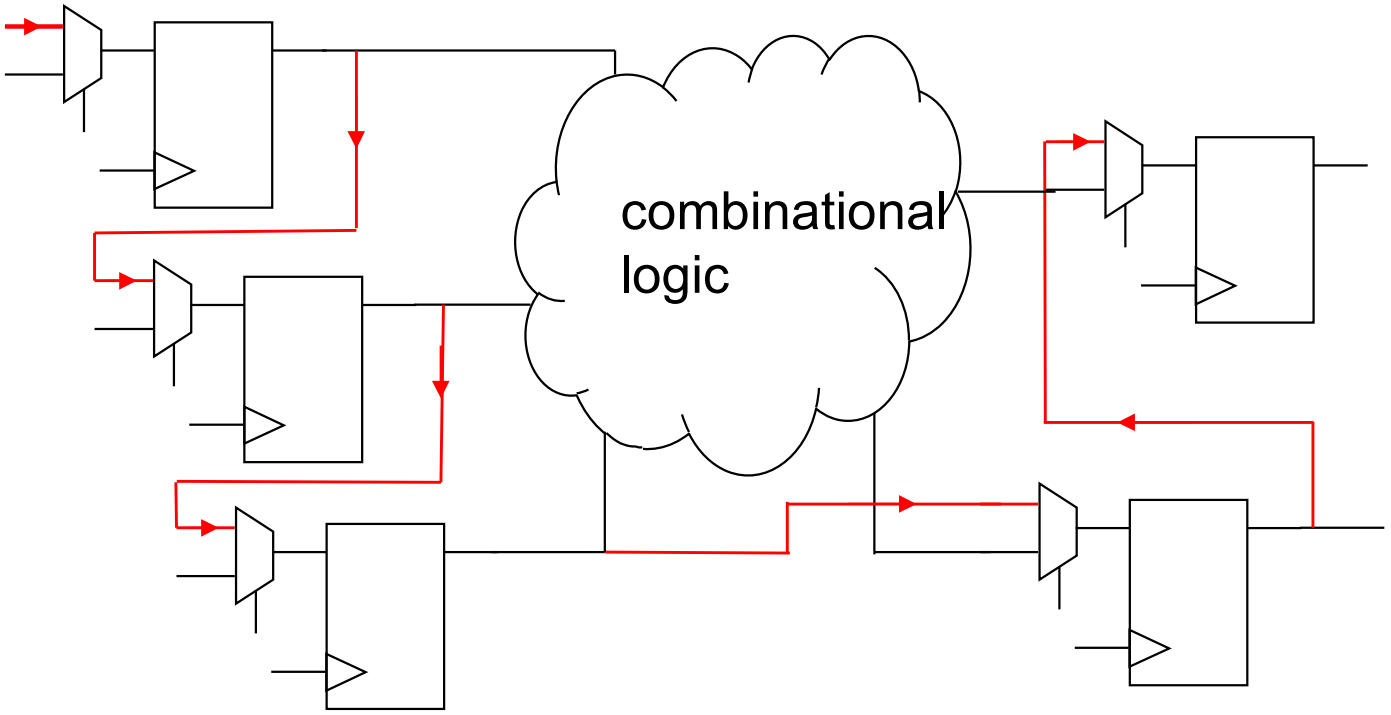
Scan Chains

- Order of flip-flops in scan chain can be completely arbitrary
  - Adjacent FFs in scan chain may drive data into same or different combinational logic
- Data enters first FF in a scan chain from a primary input and leaves last FF in scan chain from a primary output



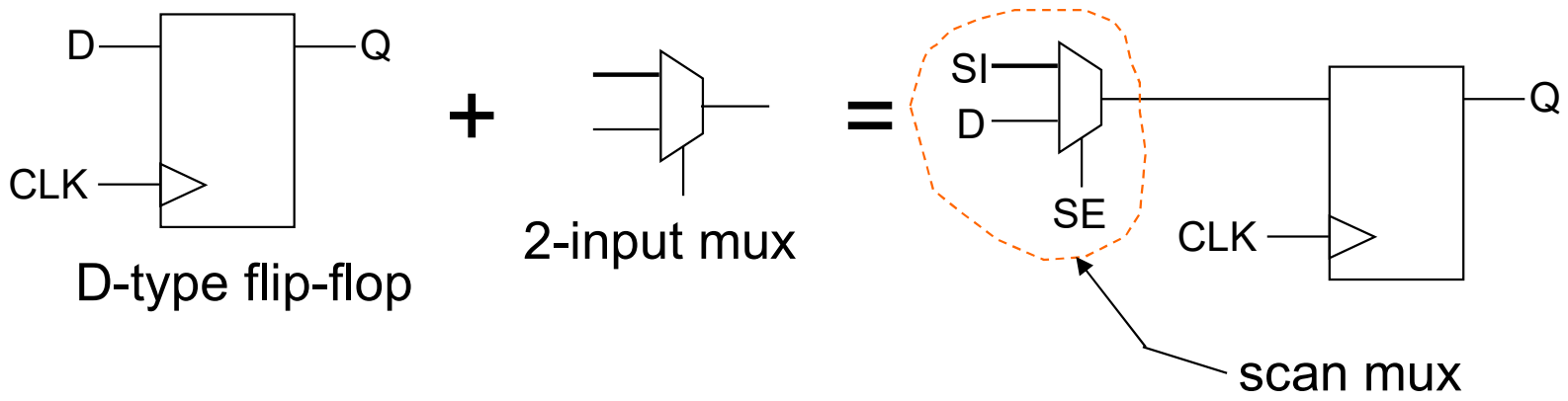
Capture Mode

- In capture mode, flip-flops drive data through, and capture data from, combinational logic



Making a Design Scannable

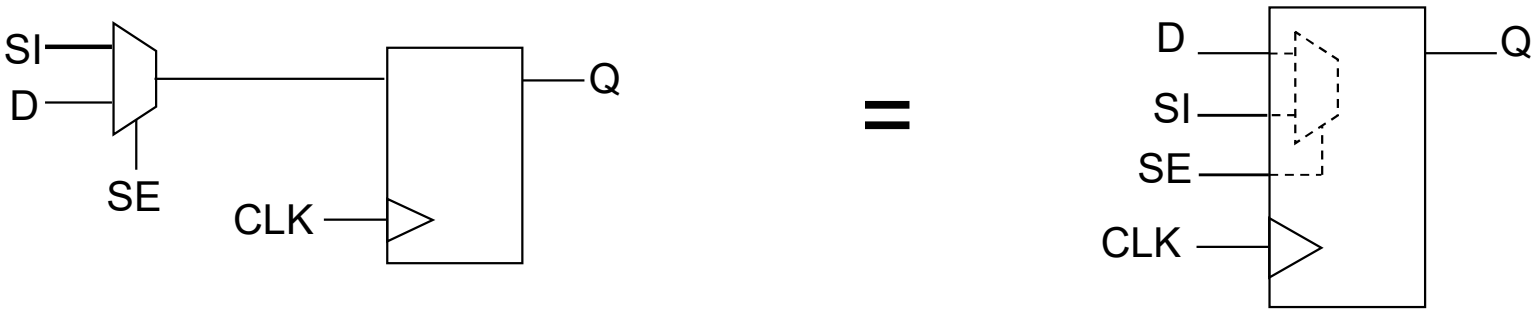
- A design is made “scannable” by the addition of a two-input mux in front of every flip-flop



- SE = “shift enable” signal
  - 1 in shift mode, 0 otherwise
- SI = scan input: input from previous FF in scan chain
  - or from “scan_in” port for first FF in scan chain
- D = functional input (comes from combinational logic)
- Note SE = 0 in both functional mode and scan capture mode

Integrated Scan Flip-flops

- Most standard cell libraries include D flip-flops with built-in scan mux:



Value of SE in different modes

Functional mode	Scan Mode	
	Shift	Capture
0	1	0

Cost of Scan

- Both a delay penalty and an area penalty is incurred as a result of scan
  - Scan FF has higher setup time and area

Example: A flip-flop from a 40nm standard cell library

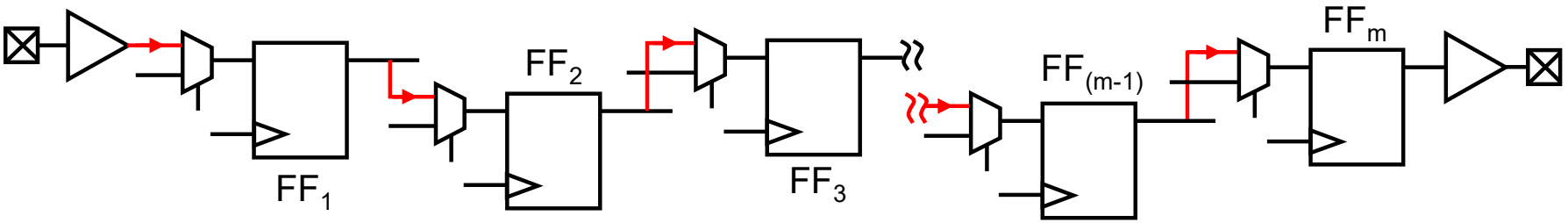
FF type	Timing (ps)		Area (μm^2)	area increase
	Setup	Hold		
Plain FF	22	6	5.3	
Scan FF	65	-30	6.5	23%

- Scan FF typically has better hold characteristics due to delay of built-in mux

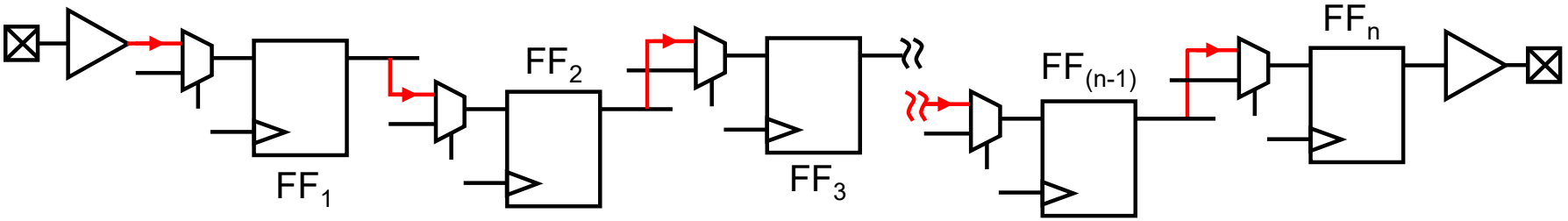
Multiple Scan Chains

- More than one scan chain can be created:
  - each scan chain has its own scan_in and scan_out ports
  - SE is common for all scan chains
  - more scan chains $\Rightarrow$ shorter scan chains $\Rightarrow$ fewer clock cycles needed to scan in values into all FFs
  - number of scan chains usually limited by number of primary ports available for scan in & scan out, or sometimes limitation of tester

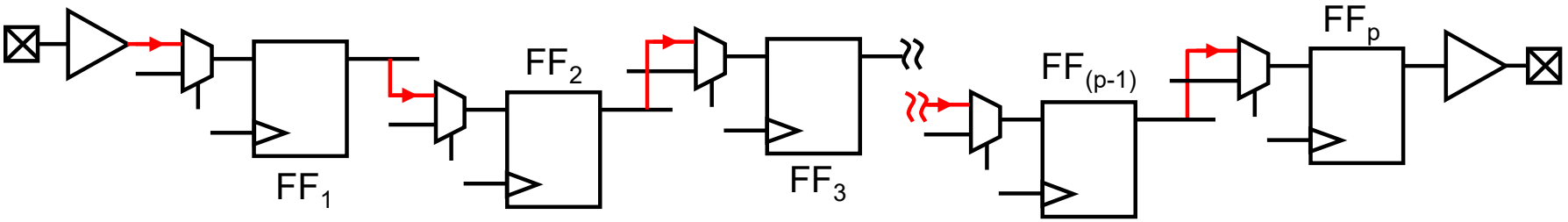
Multiple Scan Chains



Chain 1: length = m



Chain 2: length = n



Chain 3: length = p

- No. of clock periods for shift = max chain length = max(m, n, p)
- With single chain, no. of clock periods for shift = length of single scan chain = m+n+p

Automatic Test Pattern Generation

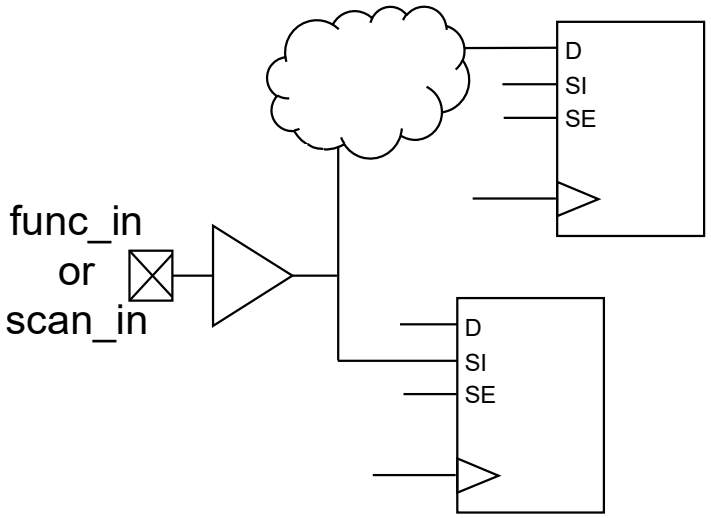
- ATPG determines:
  a) what values to load into every flip-flop in scan chain during shift
  b) what is the expected value in flip-flops after capture
  c) multiple shift-capture cycles to get coverage on all nodes

Scan Chain Balancing: Optimizing Multiple Scan Chains

- N_{shift} = no. of clock periods needed for full shift operation = max chain length
- To minimize N_{shift}, minimize max chain length
- Max chain length is minimum if all scan chains are of equal length
- Equalizing scan chain length is called “scan chain balancing”

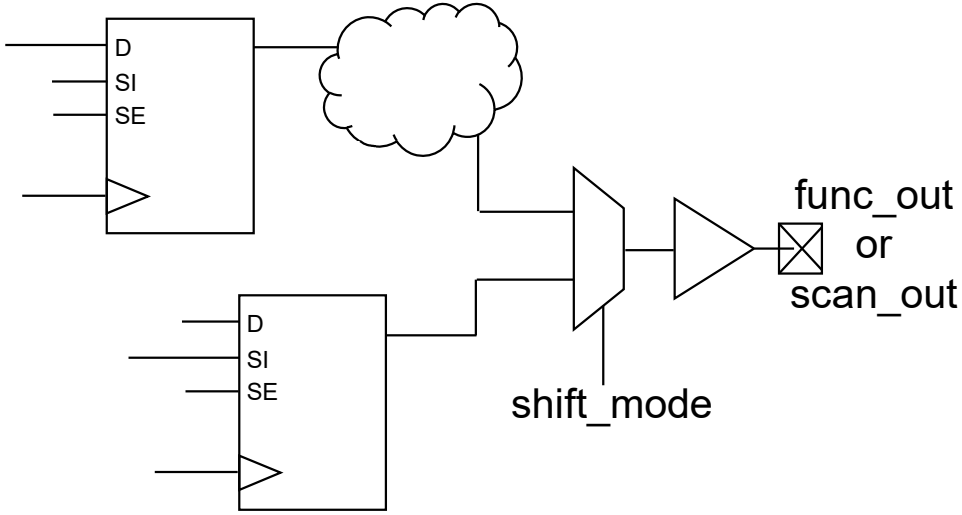
Port Sharing

- scan_in & scan_out ports need not be dedicated
  - can share functional ports



Input sharing

- Delay through receiver increases because of increased load

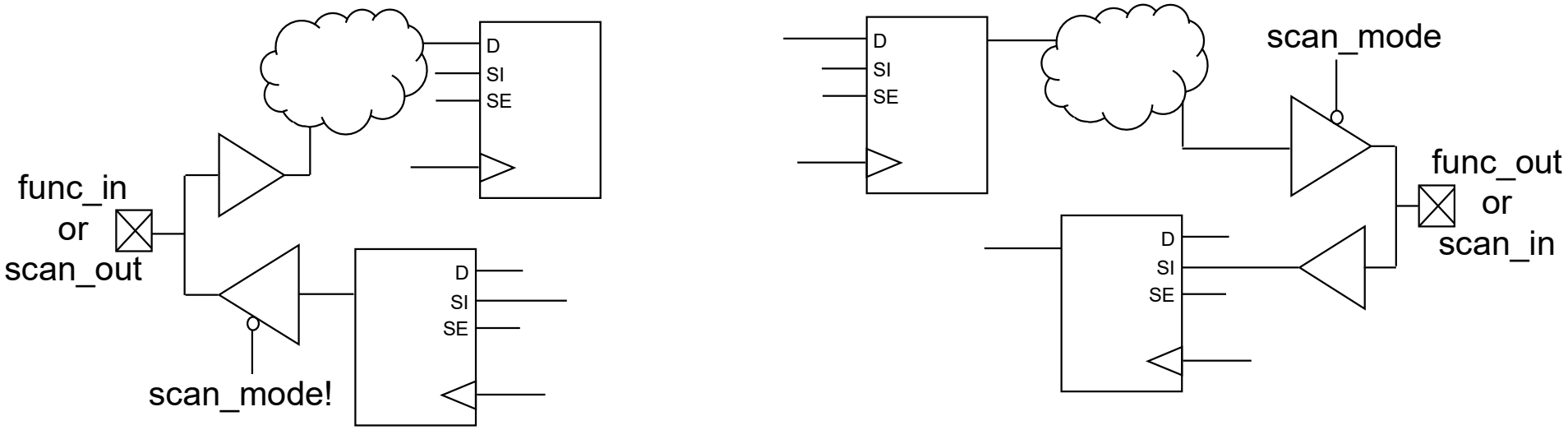


Output sharing

- Delay through mux adds to output delay

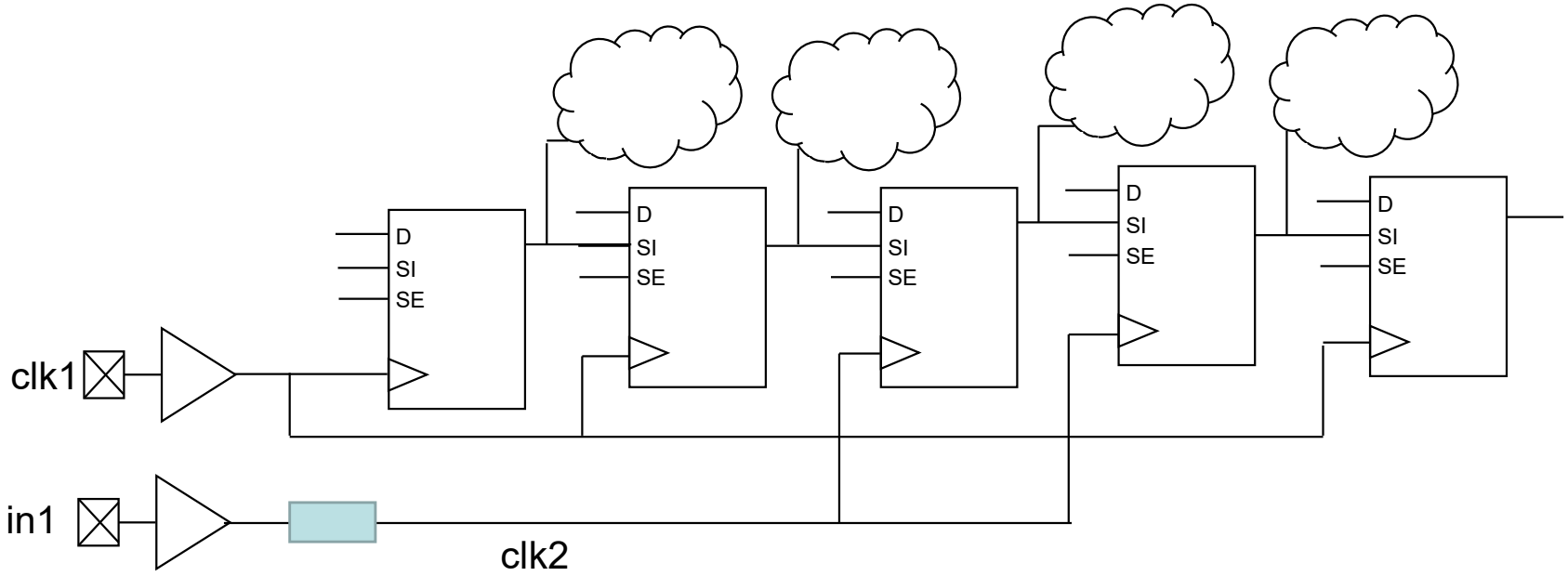
Port Sharing

- Could use bidirectional I/Os for inputs & outputs to reduce impact of port sharing on timing



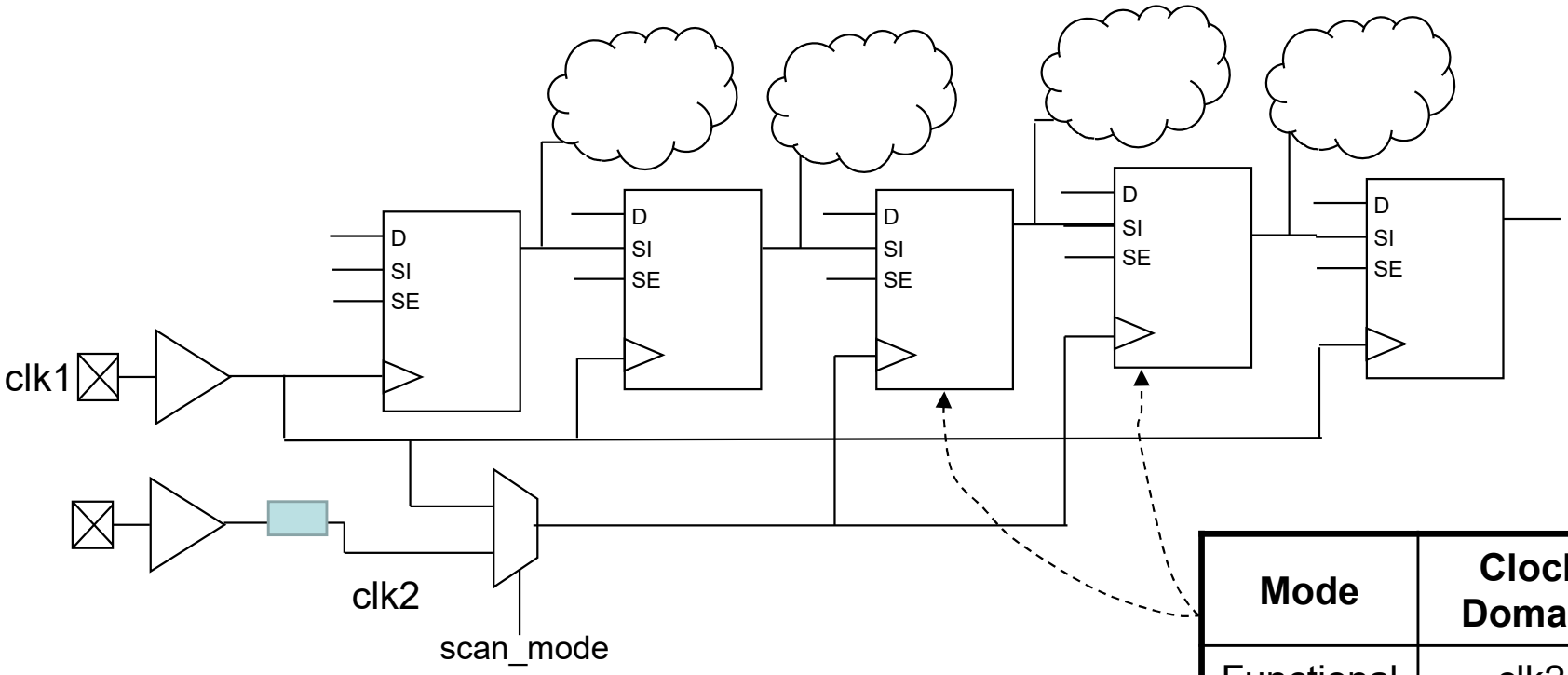
Scan Clocking

- FFs in the same scan chain can have different clock sources
  - happens if the FFs are in different clock domains in functional mode



Scan Clocking

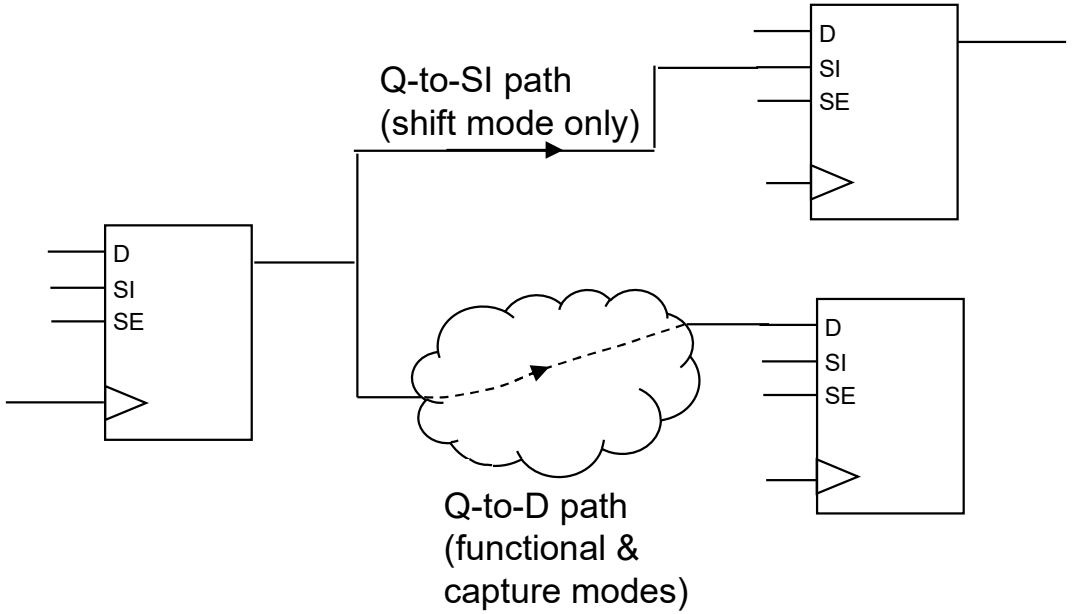
- FFs can be made to have same clock in scan mode
  - requires insertion of muxes in clock paths



Mode	Clock Domain
Functional	clk2
Scan	clk1

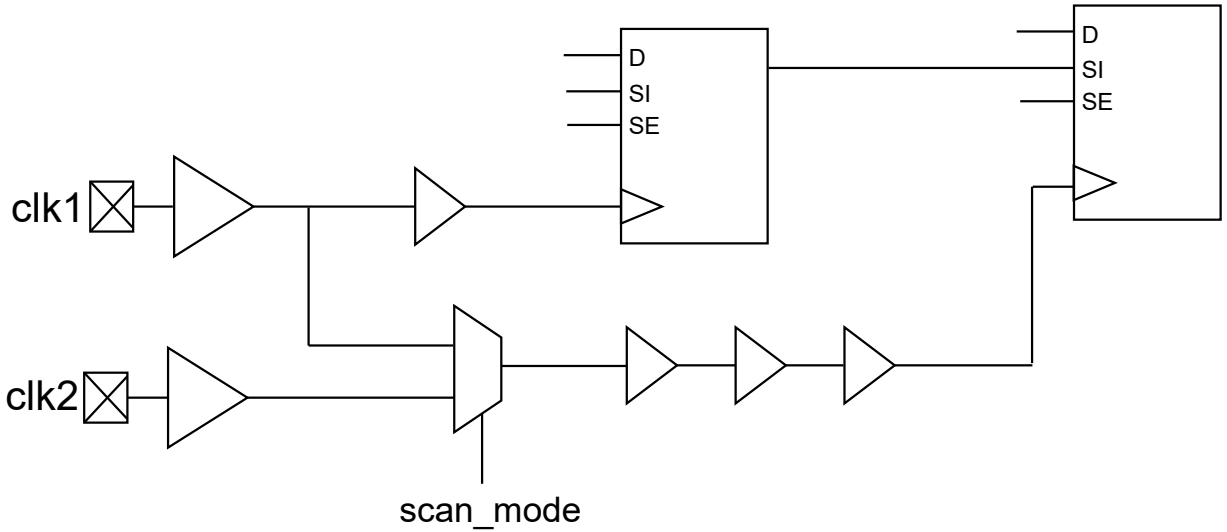
Shift Mode Timing

- Shift can occur at slow clock speed ⇒ no setup violations
- Hold violations can occur during shift
  - no combinational logic on “Q-to-SI” paths ⇒ data goes fast from Q to SI



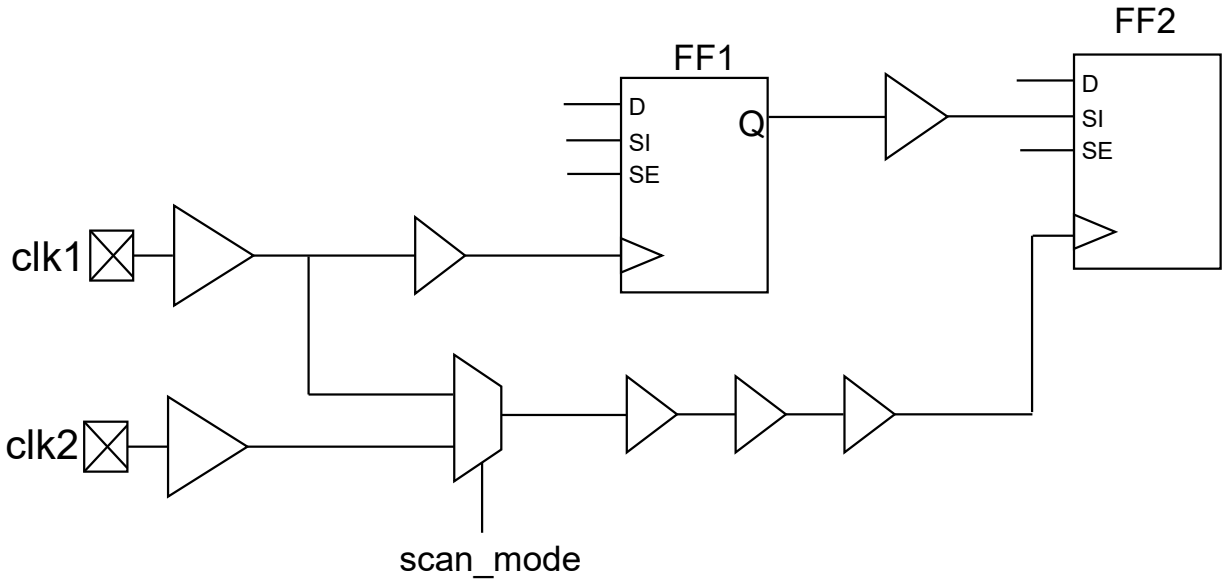
Shift-Mode Hold Violations Due to Clock Skew

- clock skew between adjacent FFs in scan chain can lead to hold violations
  - skew can be particularly high if adjacent FFs are on different clock trees



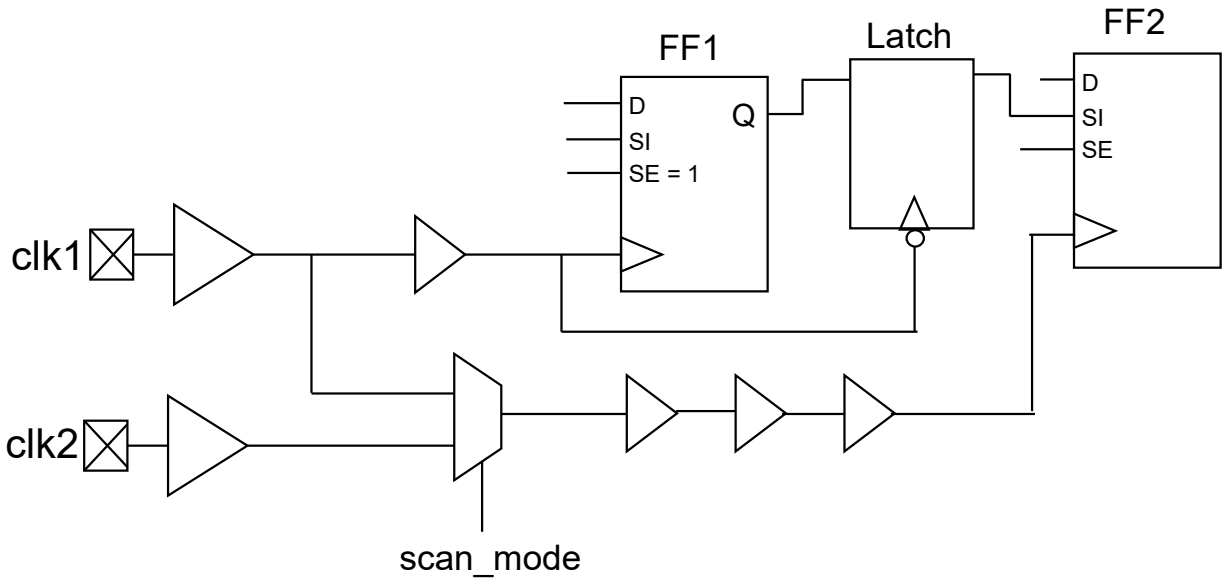
Delay Insertion for Hold Avoidance

- During scan stitching, blindly add large delay to Q-to-SI path when adjacent FFs are on different clock tree
  - kill the possibility of hold violation
  - setup is not a concern due to low frequency in shift mode

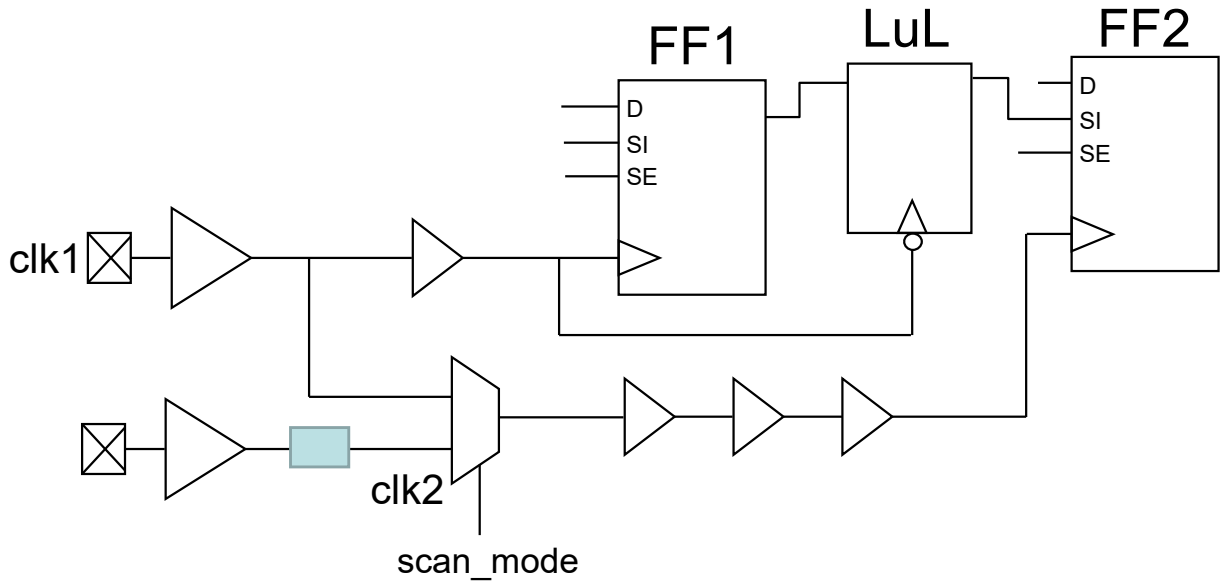


Lock-Up Latches as Delay Elements

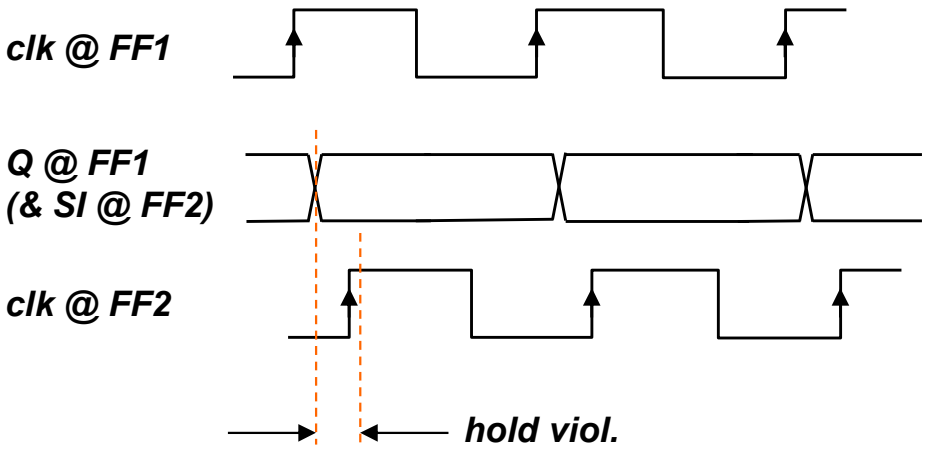
- A level-sensitive latch is the most effective delay element
  - will add delay of half clock cycle (for 50% duty-cycle clock)

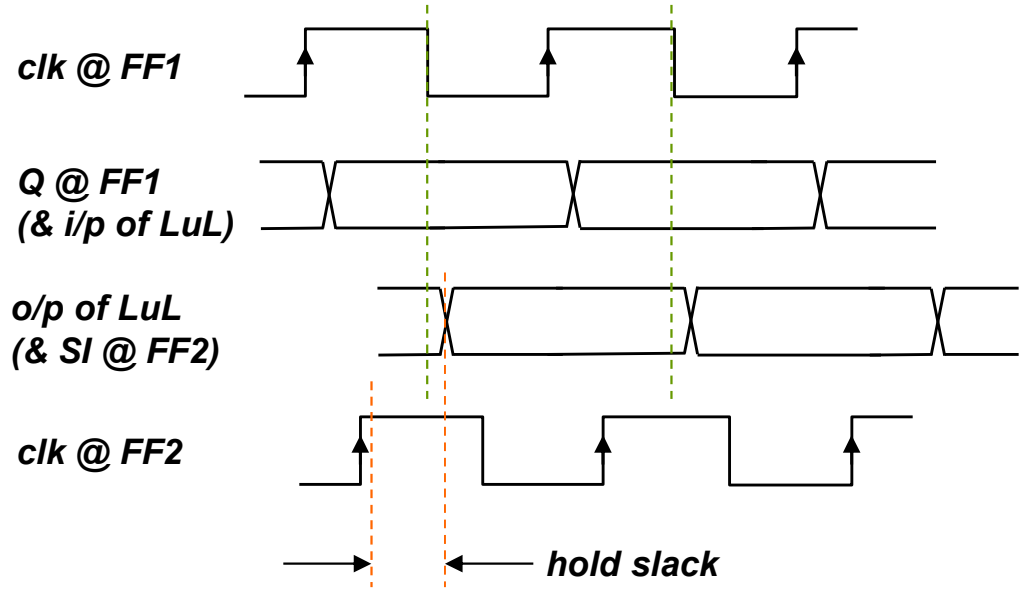


How a Lock-up Latch Works



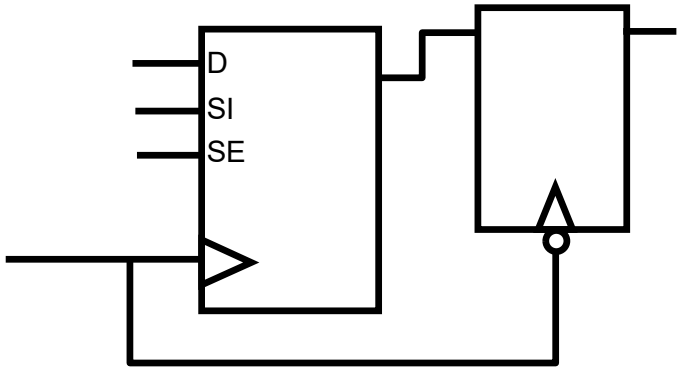
Without LuL ← assuming 0 wire delay → ***With LuL***



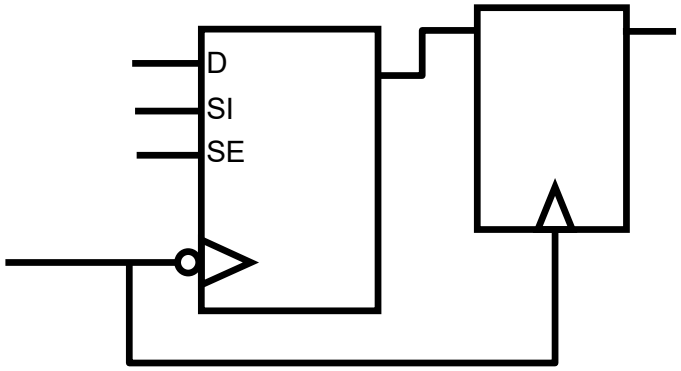


Type & Connectivity of LuL

- "Clock" of LuL must be connected to clock of upstream FF, not downstream FF
- Must ensure right "clock polarity" of LuL:



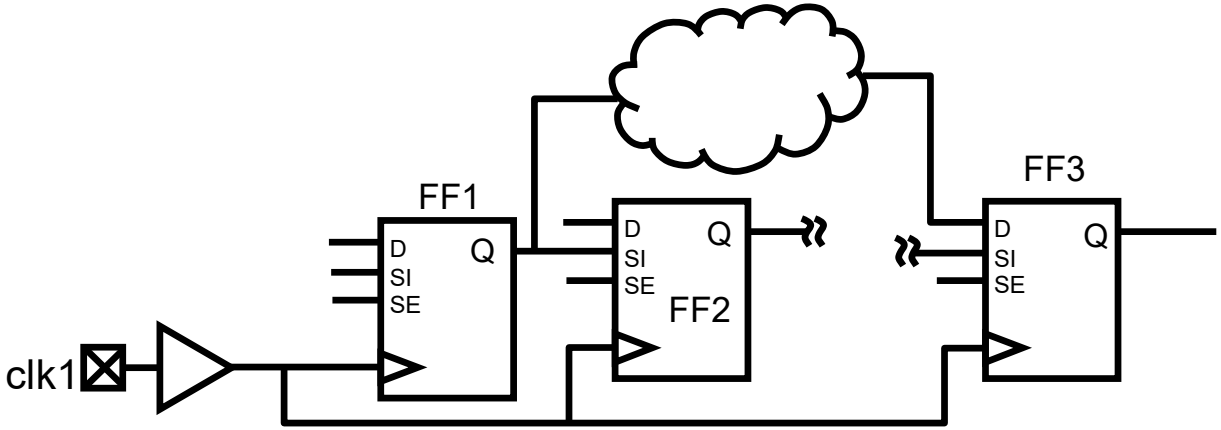
+ve edge triggered upstream FF
⇒ use "active low" LuL



-ve edge triggered upstream FF
⇒ use "active high" LuL

Timing in Capture Mode

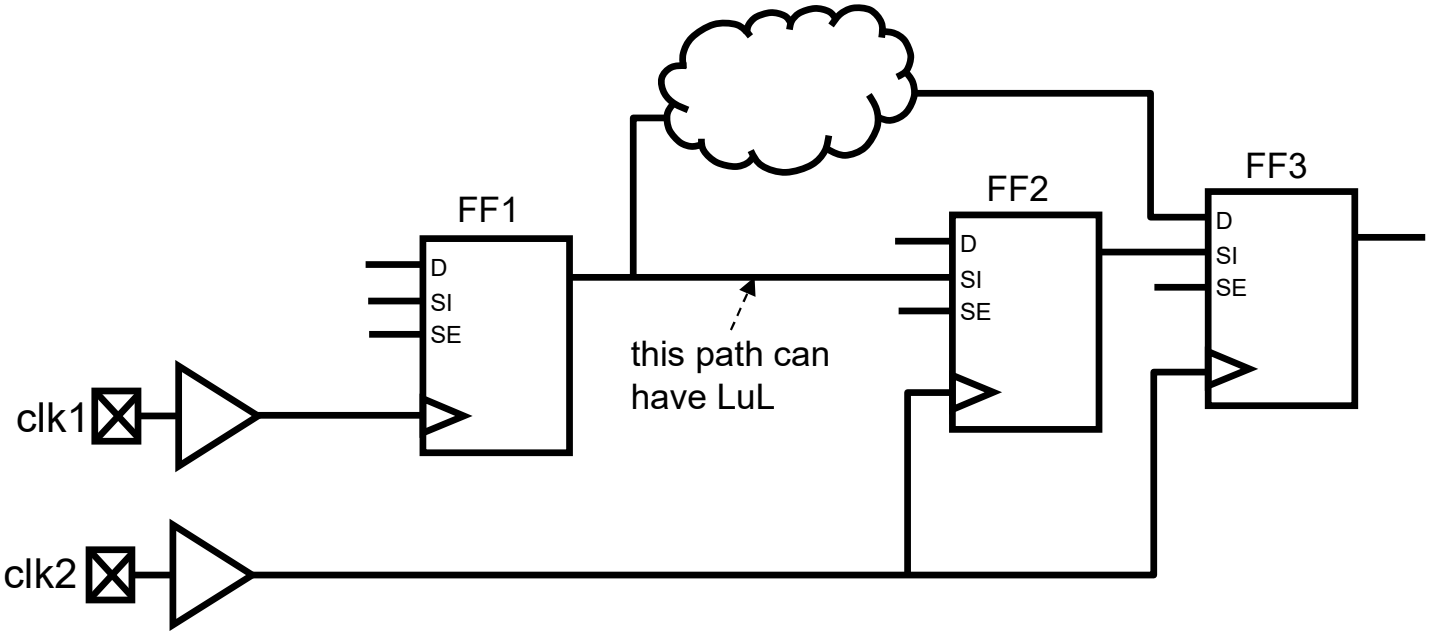
- Most functional-mode timing paths will also exist in scan capture mode
  - paths to/from bypassed macros exist in functional mode but not in scan capture mode
  - conversely, paths through scan bypass logic exist in scan capture mode but not in functional mode



- no false paths in scan capture mode
  - FPs in functional mode are not FPs in scan capture mode
- no multicycle paths in scan capture mode
- because clock freq. in scan mode typically much lower than in functional mode, both of the above conditions are usually okay

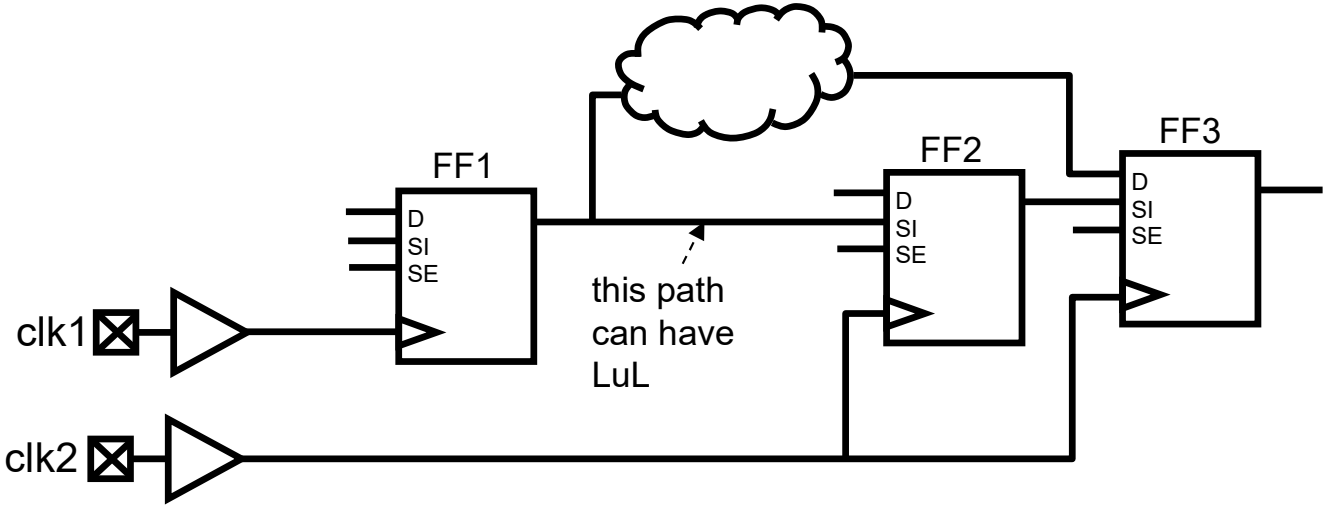
Cross Clock-Domain Paths in Capture Mode

- In functional mode, paths between asynchronous clock domains are false. Same paths are synchronous, and not false paths, in scan capture mode
  - all clocks have same frequency in scan mode: no asynchronous paths

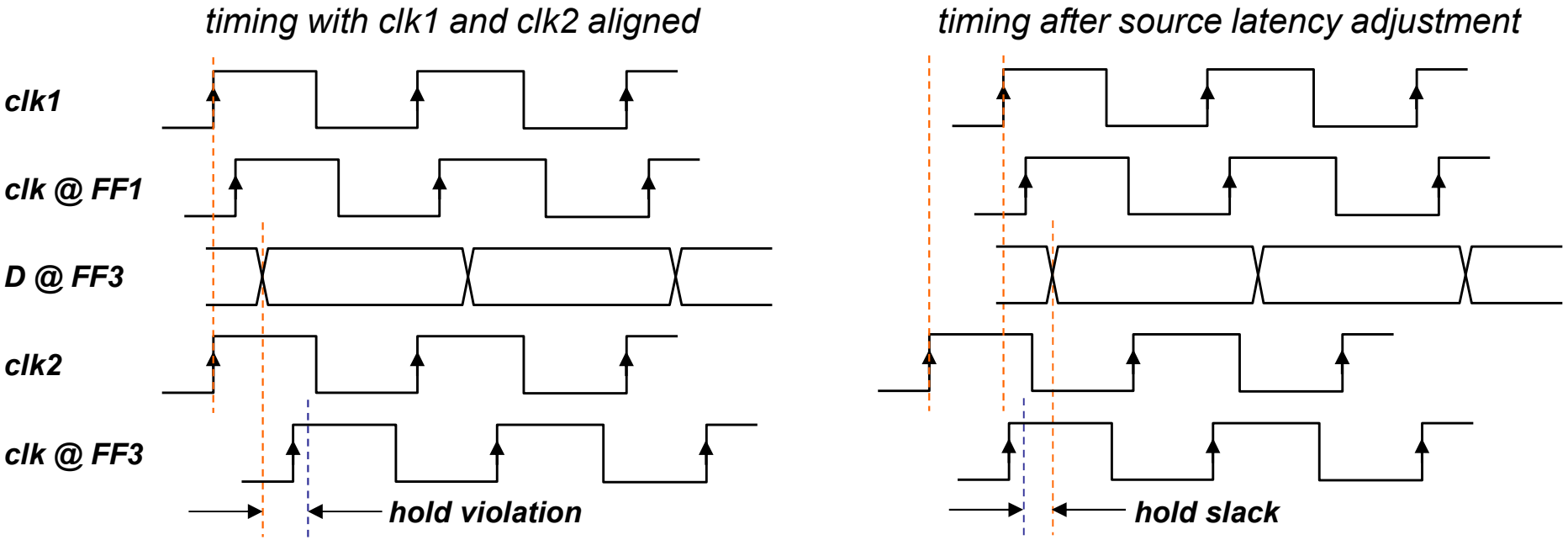


- clk1, clk2 asynchronous in functional mode $\Rightarrow$ false path between FF1 & FF3
- clk1, clk2 synchronous in scan mode
  - must meet setup and hold timing between FF1 and FF3

Cross Clock-Domain Paths in Capture Mode

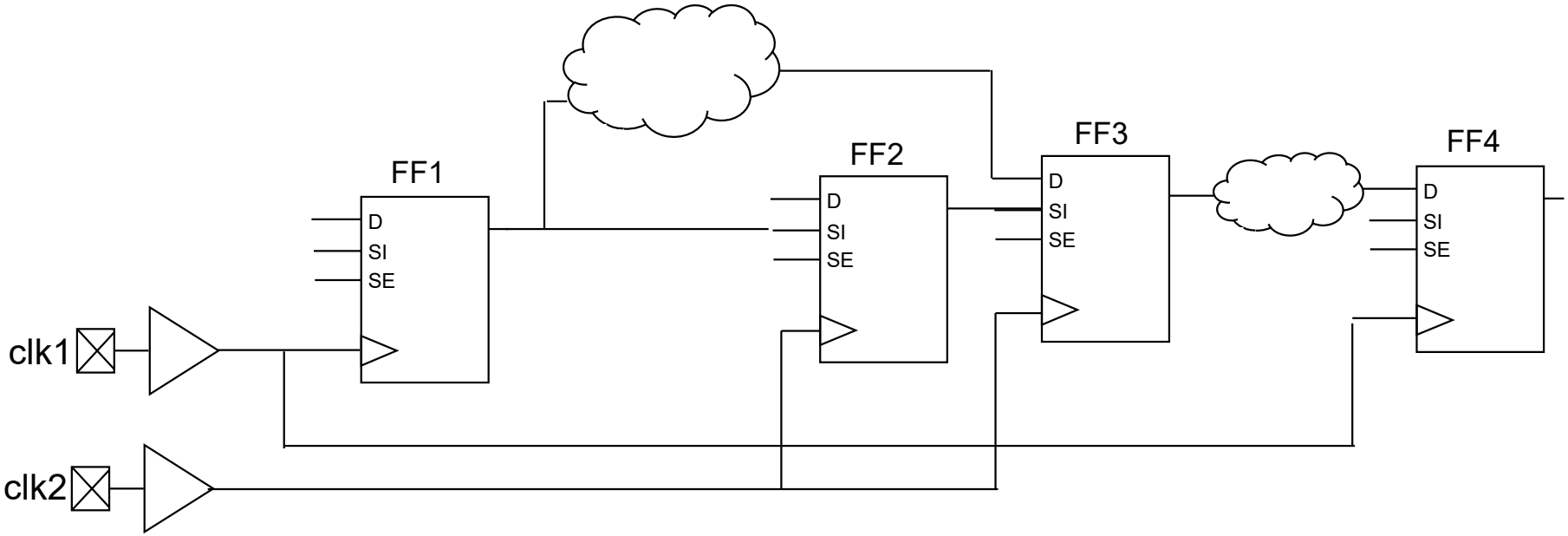


clk1, clk2 separate clocks in scan mode ⇒ can fix hold violations by "source latency adjustment" on tester:



Cross Clock-Domain Paths in Capture Mode

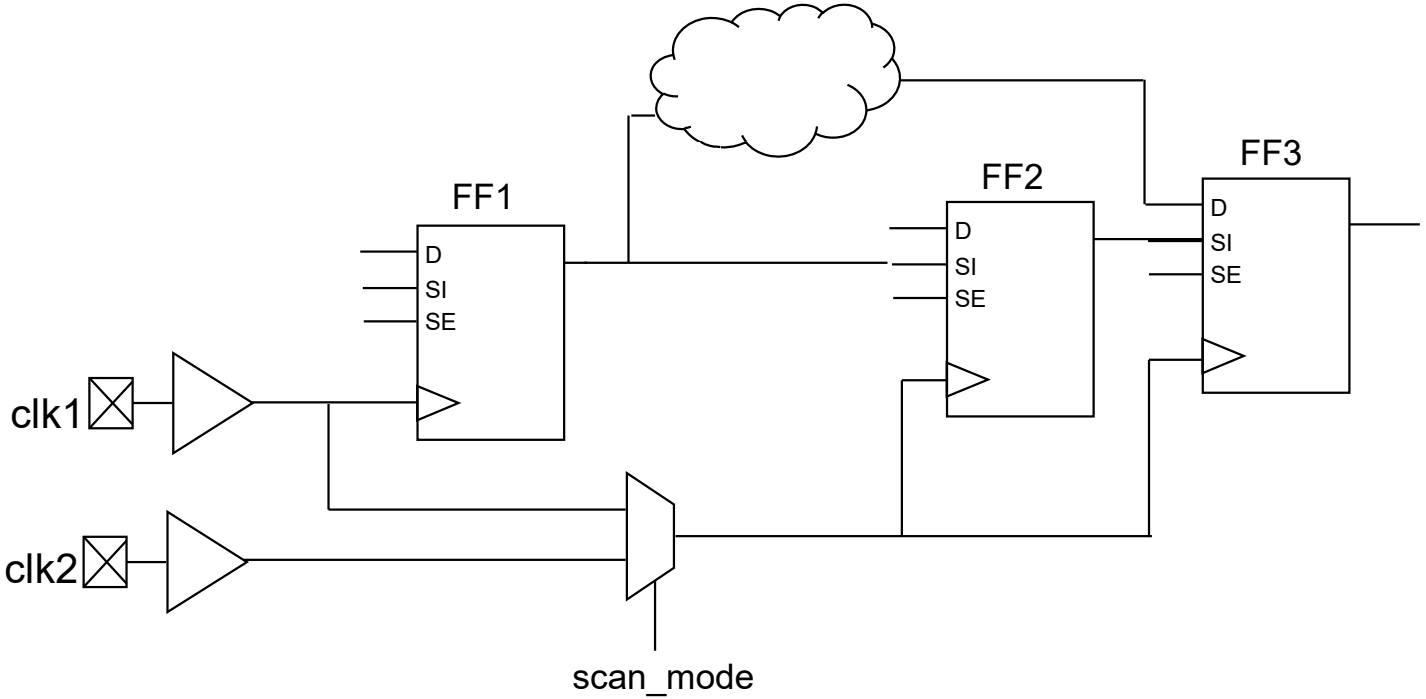
- Adjusting clock waveforms on tester will not work if there are violations in both directions (clk1-to-clk2 *and* clk2-to-clk1)



- Hold viols. in FF1-to-FF3 path *and* FF3-to-FF4 path $\Rightarrow$ fixing one by adjusting clk1 & clk2 waveforms will worsen the other

Cross Clock-Domain Paths in Capture Mode

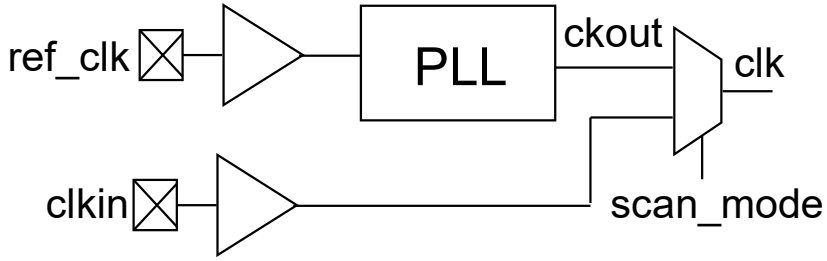
- clk2 bypassed by clk1 in scan mode $\Rightarrow$ would need to add delay into data path between FF1 and FF3 to fix hold violation



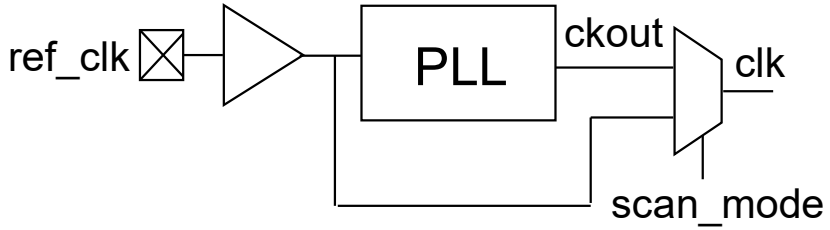
  - can affect timing in functional mode
  - when possible, keep FFs in different clock domains on separate, independent clocks in scan mode

Clock Controllability

- Clocks to all FFs in scan chains must be controllable during scan
  - Cannot use internally generated clock sources (e.g., PLLs) since tester cannot control their waveform: **internal clock sources must be bypassed in scan mode**



clk = scan_mode?clkin:ckout

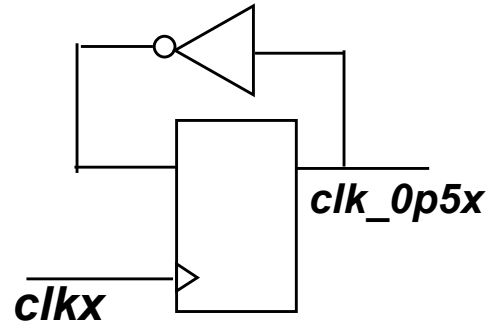


clk = scan_mode?ref_clk:ckout

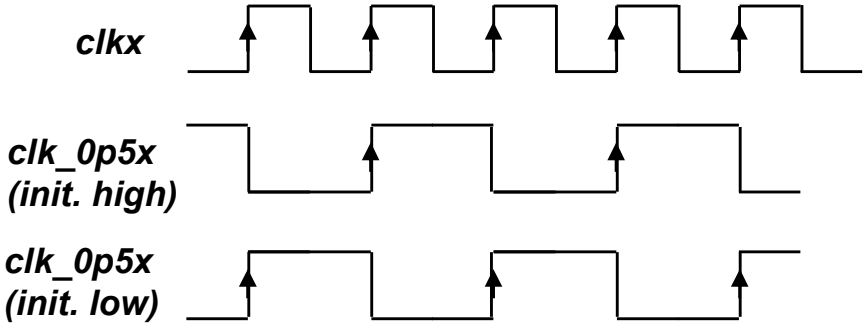
Examples of PLL bypassing in scan mode

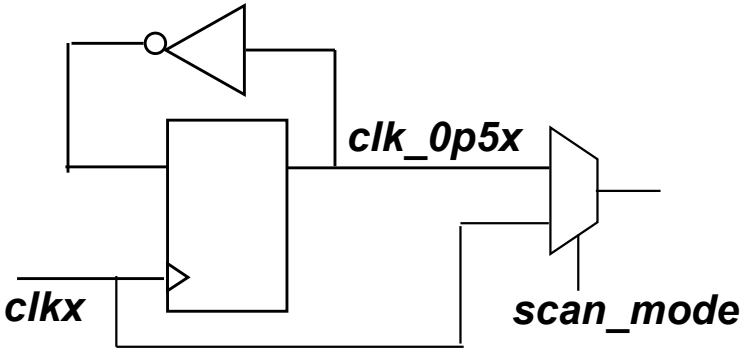
Clock Controllability: Clock Dividers

- Clock dividers should be bypassed
  - Phase of divided clock depends on initial state of divider output $\Rightarrow$ cannot have direct controllability of divided clock



Divide-by-2 clock divider





Scan-Mode Control Signals

- shift_enable: asserted only during shift operation (deasserted in capture)
  - connected to the “SE” pins of scannable FFs
- scan_mode:
  - kept asserted during entire scan operation (shift + capture)
    - deasserted in normal functional mode
  - used for “mux select” of scan-mode bypass muxes (e.g. for clocks and resets)
    - may not be needed if there are no scan-mode bypass muxes

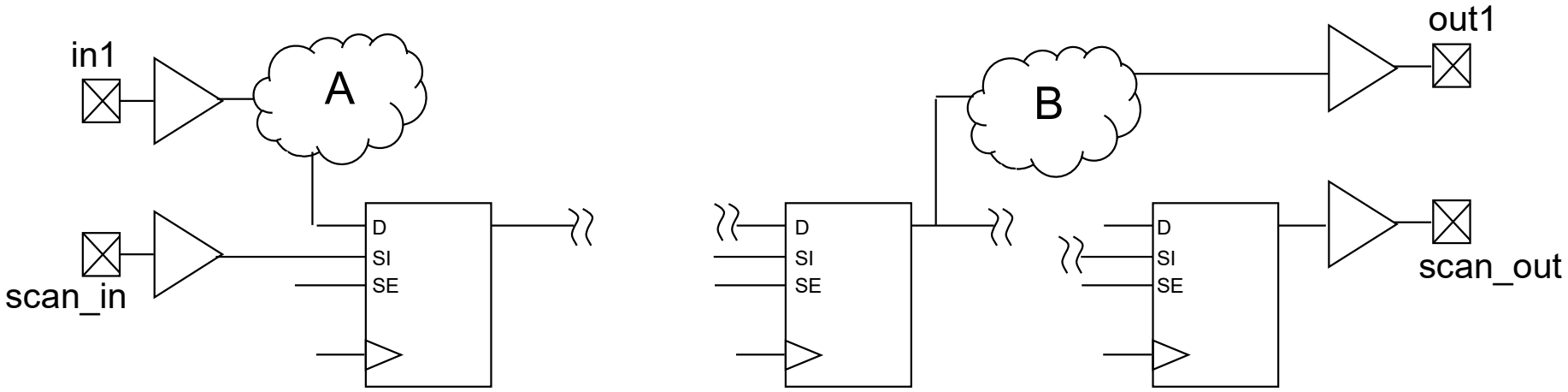
Scan-Mode Control Signals

Signal	Mode		
	Functional	Scan	
		Shift	Capture
shift_enable	0	1	0
scan_mode	0	1	1

ATPG

- With scan chains stitched up, an ATPG tool generates scan patterns that primarily contain:
  - the sequence of 1's and 0's shifted into scan chains
  - the expected sequence of 1's and 0's shifted out of scan chains
  - clock input waveforms
  - logic levels of scan-mode control signals (scan_mode, shift_enable)
- Input to the tool is gate-level netlist
  - Tool identifies and understands combinational logic between FFs
  - Tool determines what values to drive through the logic to catch faults on as many internal nodes as possible
    - Values to be driven through combinational logic are loaded onto launching flip-flops in shift cycle

Coverage of Combinational Logic near I/Os



- scan_in, scan_out: “scan” input and output
- in1, out1: “non-scan” input and output

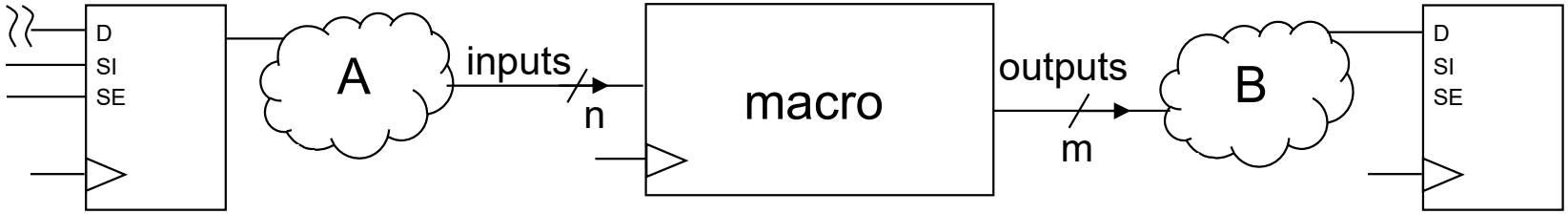
During capture:

- data is driven in from in1 to get coverage in comb. logic “A”
- data is observed at out1 to get coverage in comb. logic “B”

$\Rightarrow$ Patterns generated by ATPG tool will include input values on “non-scan” inputs, and expected values on “non-scan” outputs

Problem of Embedded Sequential Macros

- Large embedded circuits blocks, sequential in nature (e.g. memories), have “shadow combinational logic” which can escape scan coverage



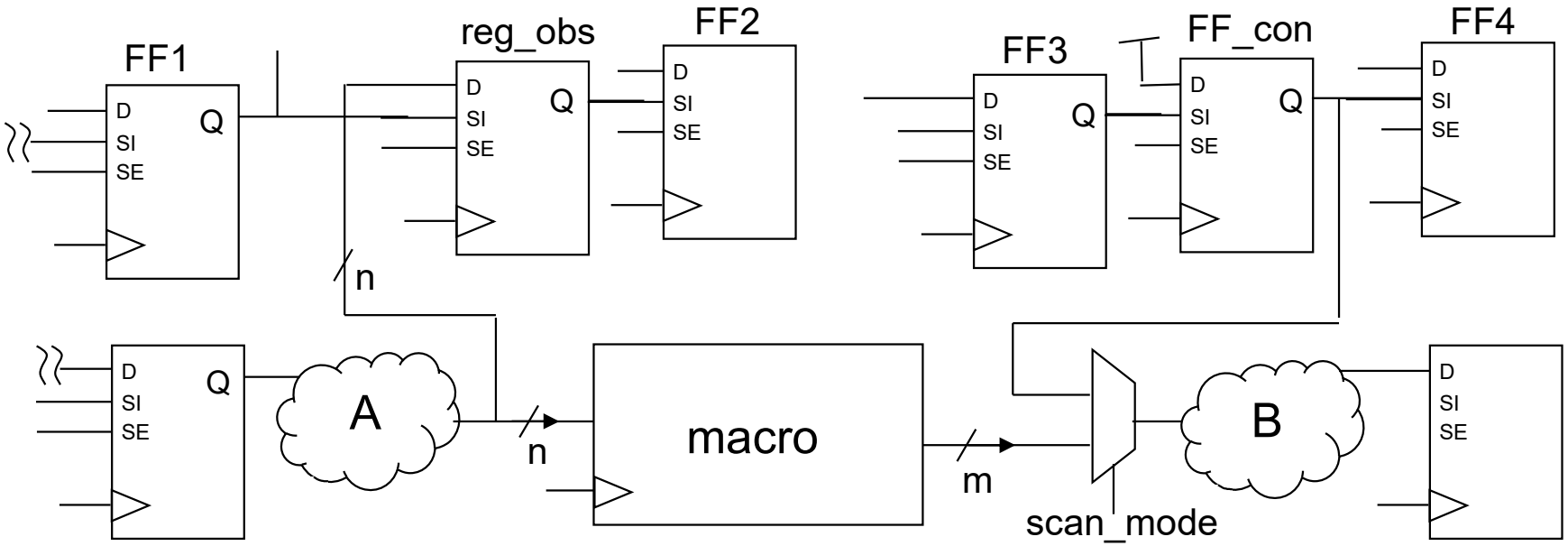
- A, B: shadow combinational logic of the macro
  - outputs of A not captured in a scannable FF
  - inputs of B are not controllable during scan

    $\Rightarrow$ no scan coverage of A & B

Getting Coverage on Shadow Logic

- Two approaches:
  1. Add capture/control FFs
  2. Macro bypass

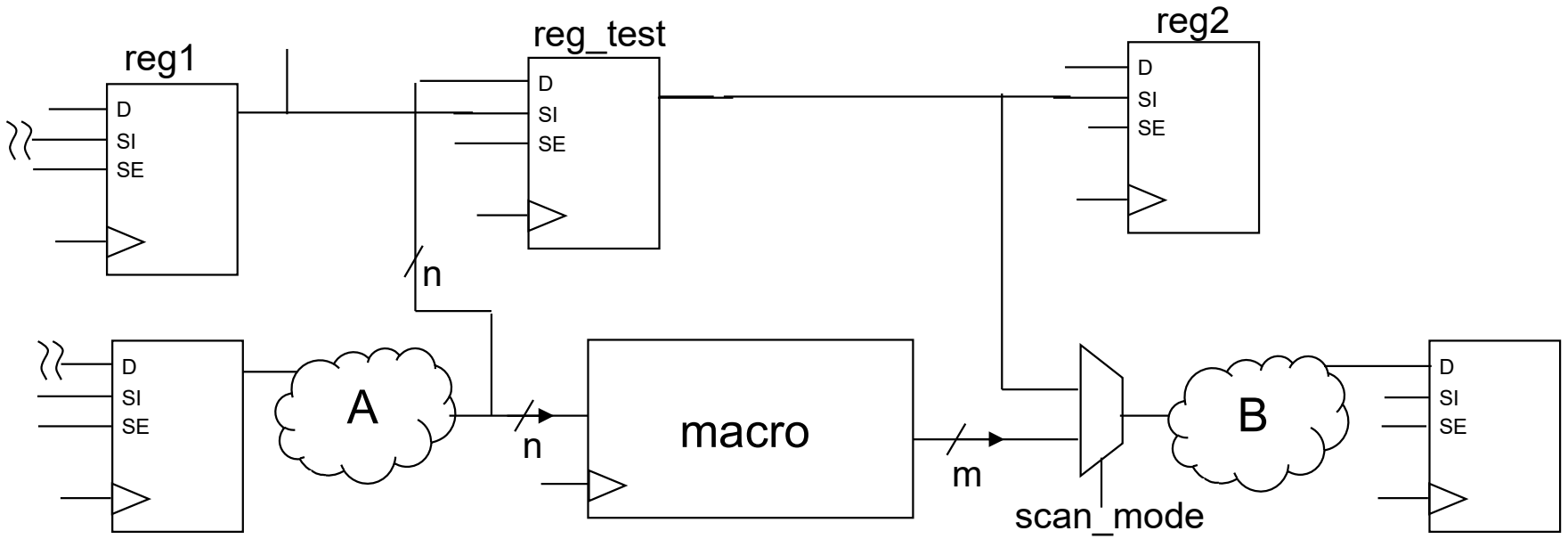
Adding Observe/Control FFs



- FF1, FF2, FF3, FF4: existing functional FFs
- reg_obs: capture FFs (inserted only for test)
  - Capture outputs of “A” (at “D” pins)
  - Q’s go only to SI pins (of next FFs in scan chain) and not to any D pins: no functional role
- FF_con: control FFs (inserted only for test)
  - Drive data through “B” in scan mode
  - No functional inputs (“D” pins can be tied high or low)
  - Q’s go only to SI pins (of next FFs in scan chain) and not to any D pins: no functional role

Reducing Count of Inserted FFs

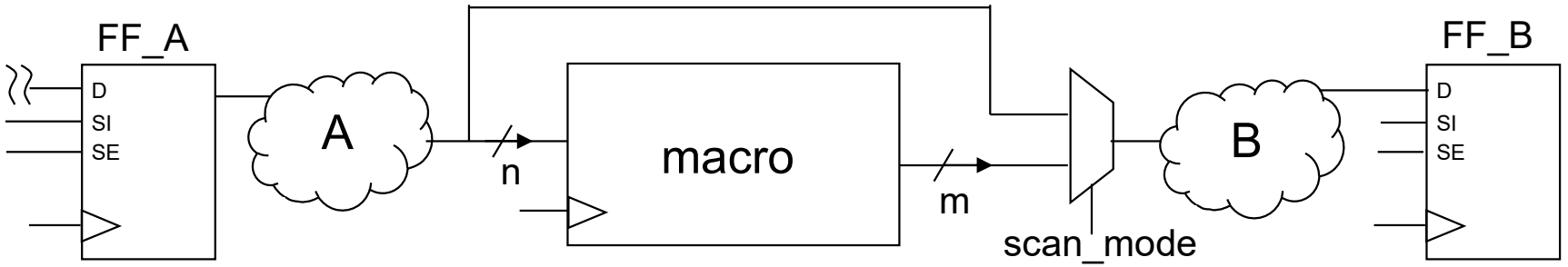
- Can use same FF for capture and control



- reg_test: capture & control FFs (inserted only for test)
  - capture outputs of “A” in capture mode
  - drive data through “B” is scan mode
- If n ≠ m, will need additional capture-only or control-only flip-flops

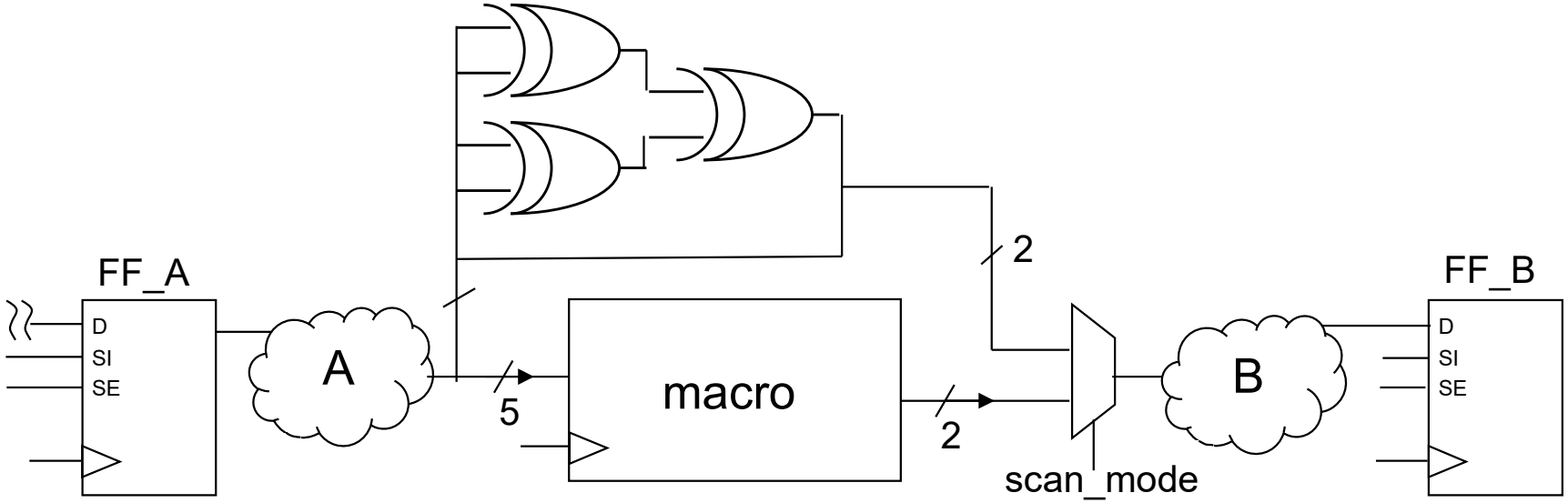
Macro Bypass

- Can eliminate common capture/control FF and simply bypass the macro in scan mode:



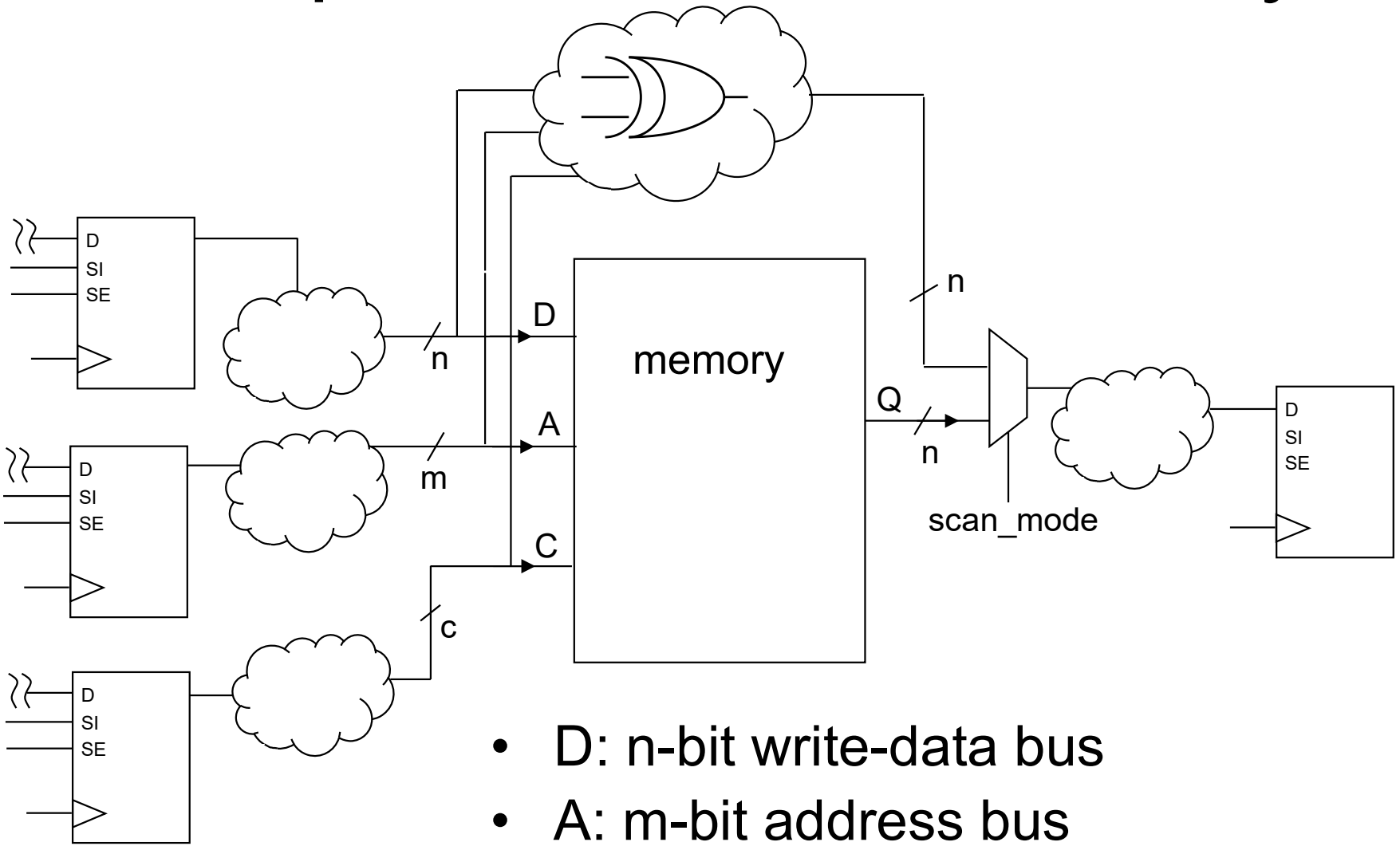
- In scan mode, data shifted into FF_A flows through A&B, and is captured in FF_B
  - macro bypassed
  - coverage obtained on A & B
  - If n < m, reuse inputs so that all outputs of macro are bypassed
  - if n > m, XOR inputs to reduce their count to “m”, and then bypass all macro outputs

XOR'ing extra inputs



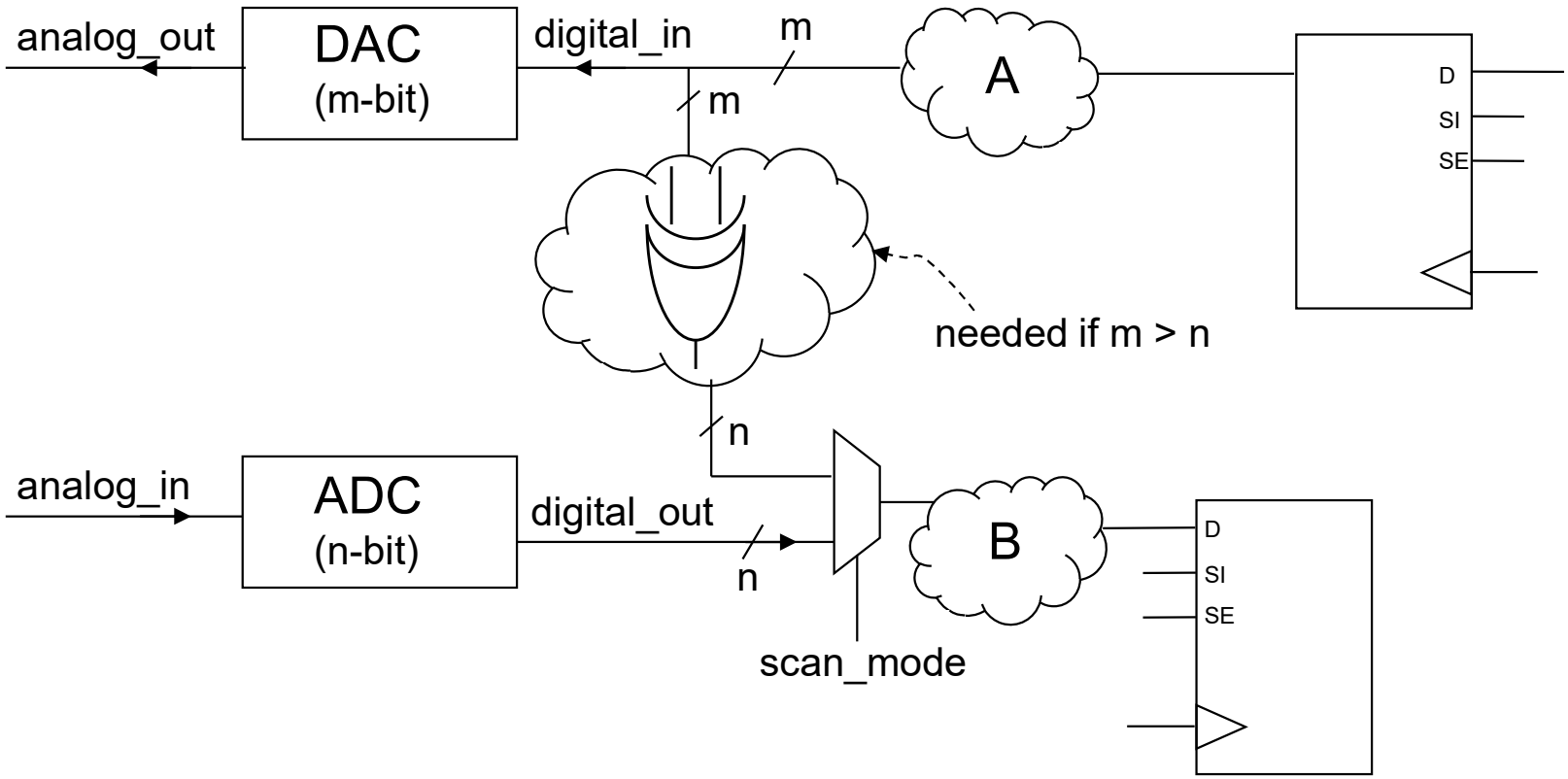
- XOR gate preferred because output toggles any time an input toggles
  - makes it easier for ATPG tool to generate patterns
  - in 2-input NAND gate (for example), output does not toggle (remains "1") when one input toggles, if the other input is "0"

Example: Embedded Memory



- D: n-bit write-data bus
- A: m-bit address bus
- C: c-bit control bus
- Q: n-bit read-data bus

Example: DAC & ADC



Scan Chain Balancing

- No. of shift cycles per pattern = length of longest scan chain
- Making all chains of equal length (or at least as equal as possible) reduces maximum scan chain length: this is called “scan chain balancing”

Tester Time & Cost

- Number of clock cycles required to fully populate a scan chain = number of flip-flops in the chain ("length" of scan chain)
- Test cost $\propto$ time on tester
  - longer the length of the scan chain, the more it costs to test the chip with scan
- Test time for scan dominated by shift operation
  - capture takes one clock cycle, while shift can take hundreds or thousands of cycles. Ignoring capture cycle,

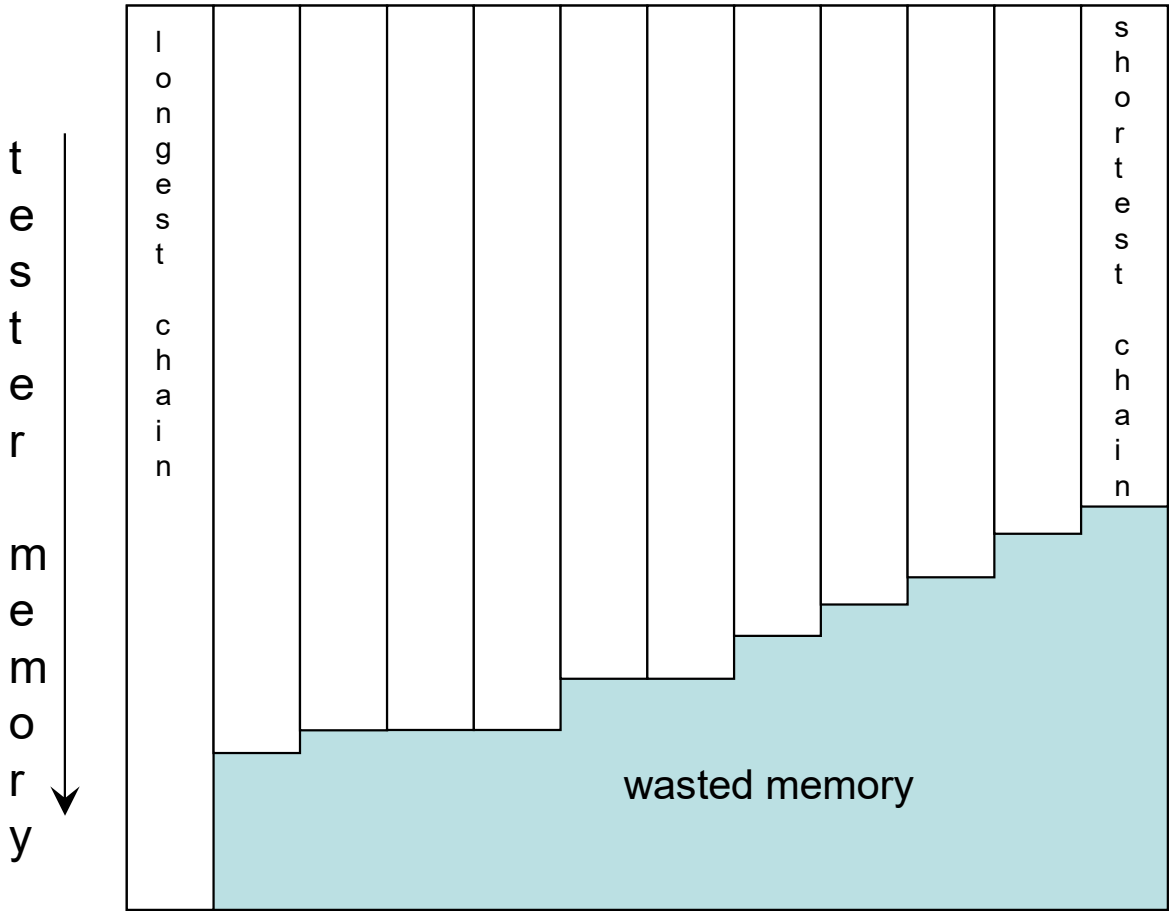
    test time =

Tester Memory

- A tester has a fixed amount of memory in which the “test program” (test patterns in a tester-specific format) must be stored
- Required tester memory $\propto$ (pattern count) X (max. length of scan chain)
- If required memory exceeds tester memory, patterns have to be loaded and executed in chunks $\Rightarrow$ large increase in tester time (and therefore test cost)

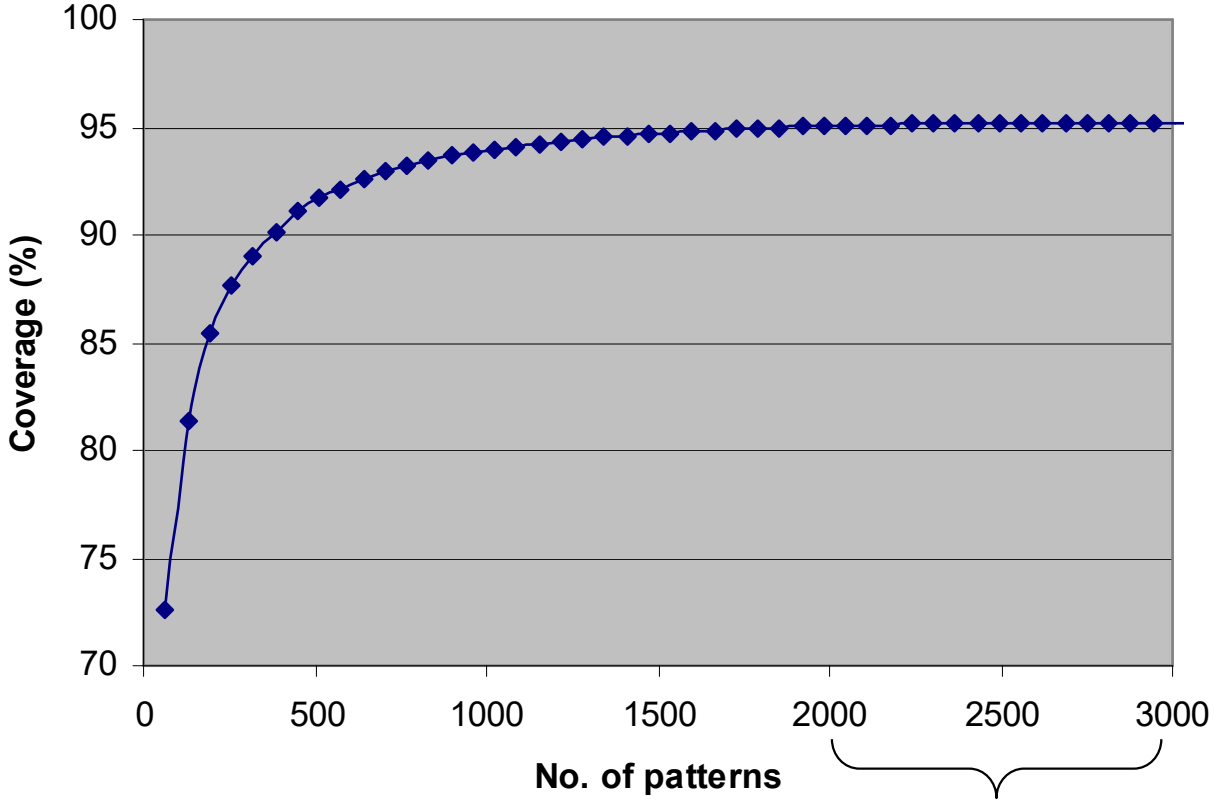
Scan Chain Balancing and Tester Memory

- Max scan chain length is minimized if scan chains are balanced
  - scan chain balancing leads to optimal use of tester memory



Coverage vs. Pattern Count

- Beyond certain count, additional patterns may make only a small incremental contribution to coverage
  - Can eliminate these patterns to reduce tester time and memory
  - ATPG software will generate such “fault grading” data



last 1000 patterns contribute an additional coverage of only 0.15%

“At-Speed” Scan

- Scan clocks are run at low frequency (compared to usual functional clocks) in conventional scan
  - less power consumption
  - fewer setup violations
  - “Stuck at” faults, which are manifested at any speed, can be detected with slow scan clocks
- Timing-related faults are not detected at conventional scan clock frequencies
  - Manifest only at high clock frequencies, going undetected at slow clock frequencies
    - transition faults (slow-to-rise, slow-to-fall)
    - path delay faults

“At Speed” Scan

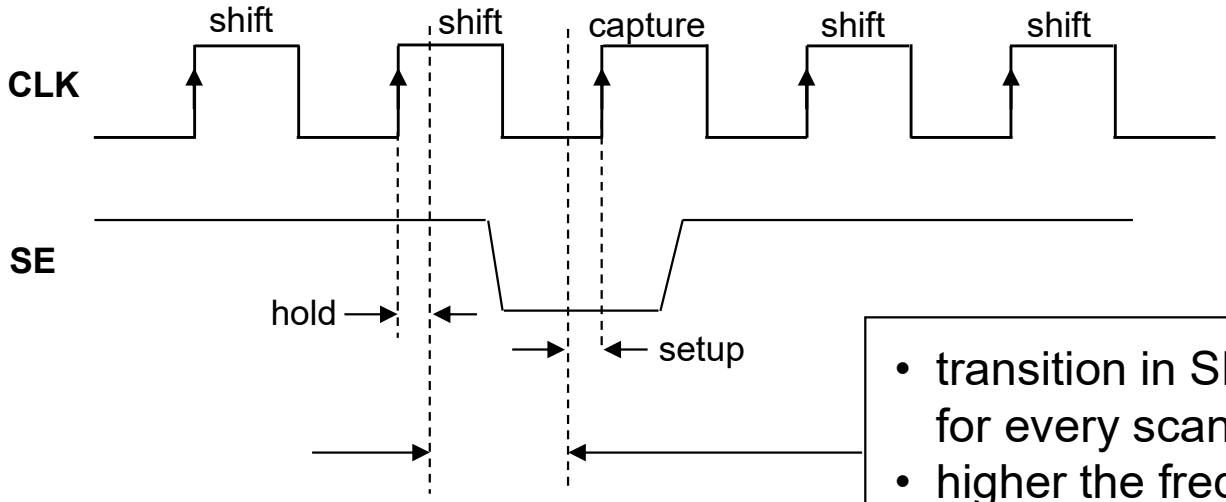
- Circuit activity can be very high during shift, leading to excessive power consumption and IR-drop at high clock frequency

$$P = CV^2f\alpha$$

- Biggest problem is meeting setup timing of the SE signal when it toggles between shift & capture cycles

SE Timing in At-Speed Test

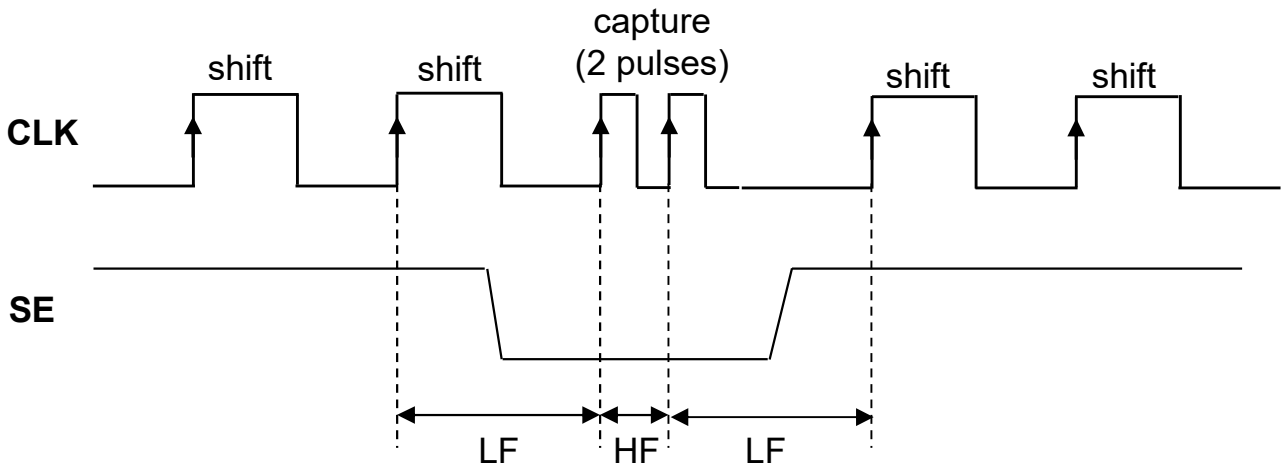
- SE signal goes to all scannable FFs
  - distributed in a similar way to the clock (i.e., as a “tree”)
- delay from SE source to FF (“insertion delay”) can be large (esp. to FFs distant from source)
- SE must meet setup & hold requirements at every FF
  - possible, but difficult when clock frequency is high



- transition in SE must fall within this window for every scannable FF in the netlist
- higher the frequency, narrower the window
  $\Rightarrow$ more difficult to hit everywhere
  - clock skew shifts window location for different FFs

Doing At-Speed Test

- One way to do at-speed testing is by using two capture pulses:
  - Capture pulses at high-speed
  - Shift cycles at low speed



- No setup requirements for SE at HF
- May need on-chip circuitry to generate pair of high-speed capture pulses when SE is low
- ATPG tool should be able to generate the right vectors
  - can target only minimum slack paths