Testability - What is it?

- Not asking if the chip is functionally correct
  - You've done that already in RTL simulation
- Not looking for timing problems
  - You've checked that once in gate simulation
- But rather, has the chip been correctly manufactured?
- Was there....
  - Mask Contamination?
  - A defect in the silicon crystalline structure?
  - A mask alignment error?
  - A flaw in the oxide layer somewhere?
  - An improper doping profile?

Testability

- RTL/gate testing is easy
  - Unliminted access to all circuit nodes
  - Full-speed operation with no physical effects
  - 1000's of scenarios possbile with 10's of workstations for days

Testability

- Testing after manufacturing is tough
  - 100,000,000 nodes to test, using four to eight package pins
  - Pins are expensive, can't afford to have many single-use pins
  - Chips tested on special IC testing machines, charged by the second
  - Chips can't be run on tester at more than about 25Mhz
  - Tester time adds directly to cost, adds no performance

Testability

- How do you quickly test all the internal logic with just a few pins?
  - Don't try to find the physical problem, too complicated
  - Assume is that the logic is correctly designed
  - See if the logic functions, not if the function is logical
  - Are there other simplifying assumptions we can make?

Testability

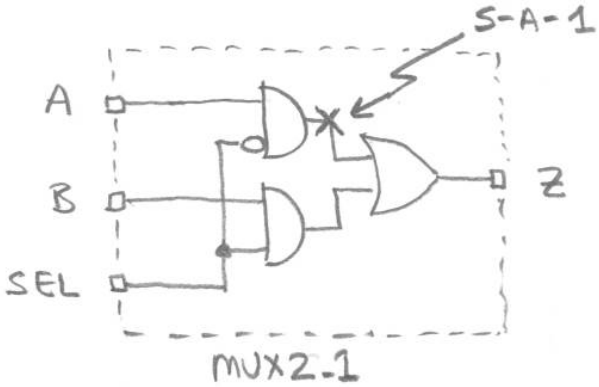
- One logical fault may cover many other physical faults

Possible Faults		
Physical Defects	Electrical Defects	Logical Faults
Opens in Metalization		
Silicon Defects	Shorts	
Photolithographic Defects	Opens	S-A-1
Mask Contamination	Transistor Stuck on	S-A-0
Defective Oxide	Vth Shift	
Pinholes in Oxide		

- So, all we need to be able to find are S-A-1, S-A-0 conditions

Testability - Finding Faults

- Two parts to finding the faults:
  - Excitation: Control the circuitry around the fault to "tickle" it
  - Propagation: Be able to then percolate the test result to the pins
- Assumptions:
  - Only one logical fault where you are testing
  - Faults are detected by incorrect circuit output
  - Faults are detected at cell pins and interconnect



To Excite: Force A → 0
Force SEL → 0

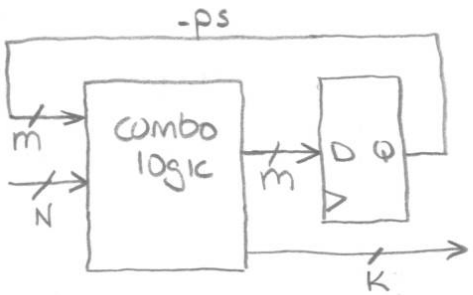
To Propagate: Force B → 0

Testability - Measures of Design Testability

- Controlability:
  - How easy it is to bring a node into a state using only I/O pins
- Observability:
  - How easy it is to observe a node's value at a I/O pin

To exhaustively test will take 2^N patterns.

For $N=20 \Rightarrow 1\times10^6$ patterns
1sec for 1μs/pattern



To exhaustively test will take $2^{(N+M)}$ patterns.

For M = 10 state bits, $\Rightarrow 1\times10^9$ patterns. ≈ 16 minutes

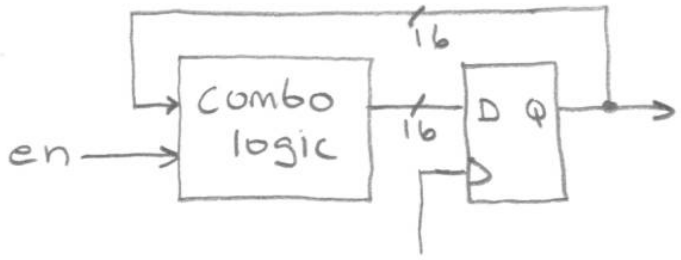
- Obviously, brute force will not work

Testability

- Two helpful facts and one bit of bad news...
  - 1. An exhaustive input pattern set contains considerable reduncancy. A single fault is often detectable by multiple input patterns. The rest of the patterns are not needed.
  - 2. If we don't need to detect all the faults, fewer patterns will be needed. If detecting the last .01% of faults takes 10x the patterns, skip it.
- So testing combinatorial logic may be easier than it looked

Testability - What About Sequential Circuits?

- To test a sequential circuit (e.g., state machine)....
  - 1. Bring sequential circuit to correct state
  - 2. Apply input to detect fault
- Getting to the right state may take 1000's for clock cycles
- For example, 16-bit counter.... 65,536 cycles
- Brute force fails again



16-bit counter, 65,536 states

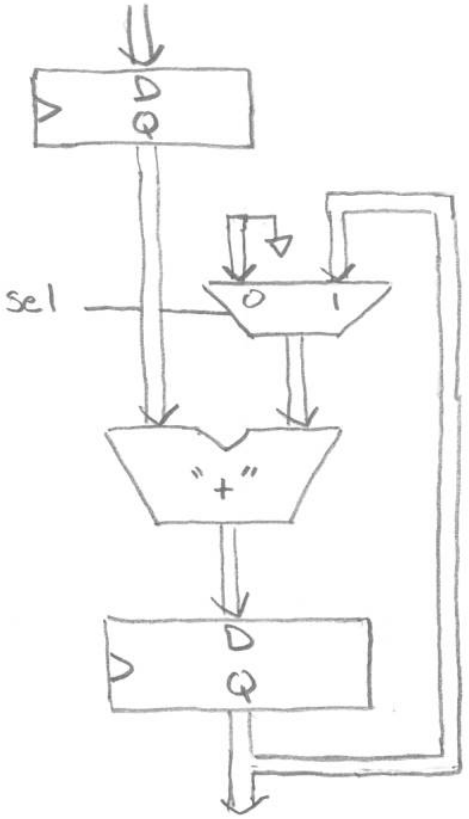
Testability - What About Sequential Circuits?

- Solution: Break up the sequential circuit into a combinatorial part and a state storage part
- Serially shift in a pattern to the combinatorial logic using FFs ...
  ... capture the combinatorial logic output in FFs ...
  ... then shift the combinatorial output serially to the IC pins.
- This is the idea behind scan-test methodology

Testability - What About Sequential Circuits?

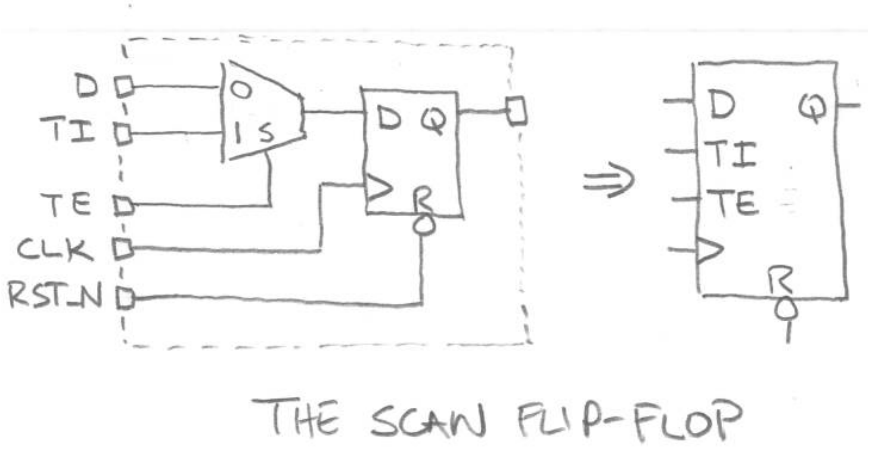
- Some sequential circuits such as data paths are easy to test
- A few patterns applied to the data path will check everything

Walking Ones	Walking Zeros
00000001	11111110
00000010	11111101
00000100	11111011
00001000	11110111
00010000	11101111
00100000	11011111
01000000	10111111
10000000	01111111

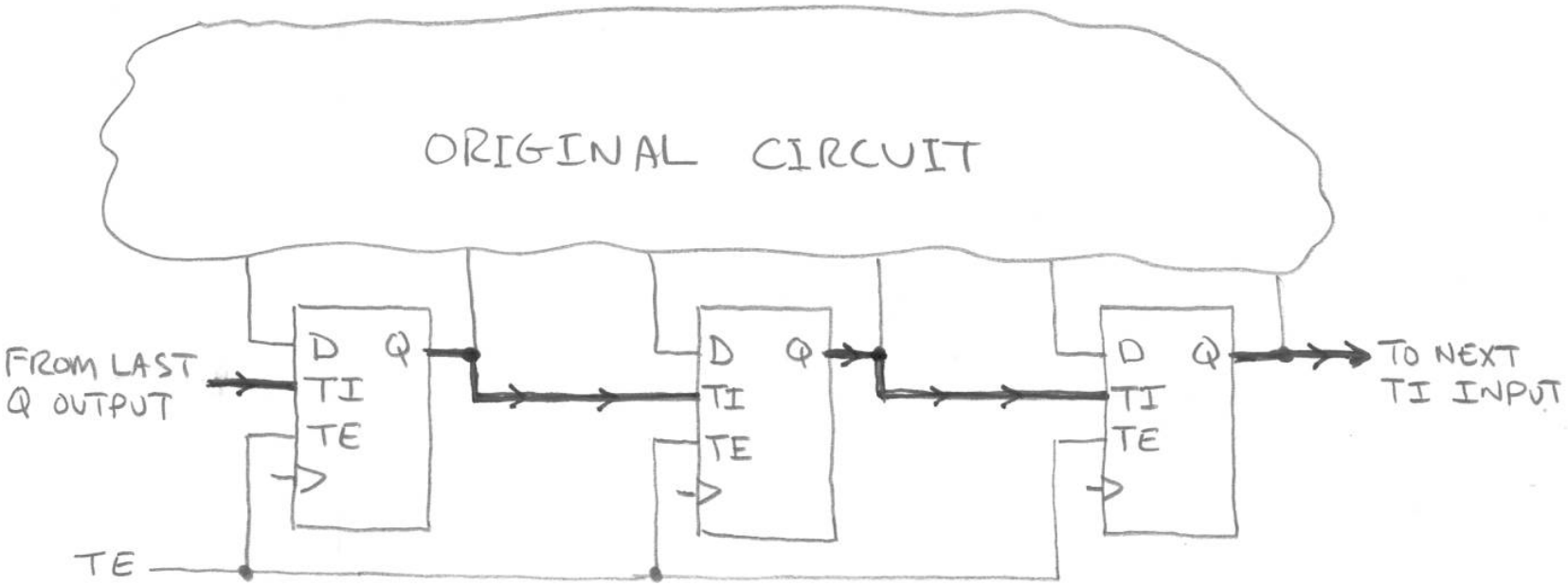


Testability - Scan Flip-Flop

- This structure will allow:
  - Shifting of new pattern for application to combo logic
  - Capturing of new pattern data from combo logic
  - Shifting out of pattern data to I/O pins



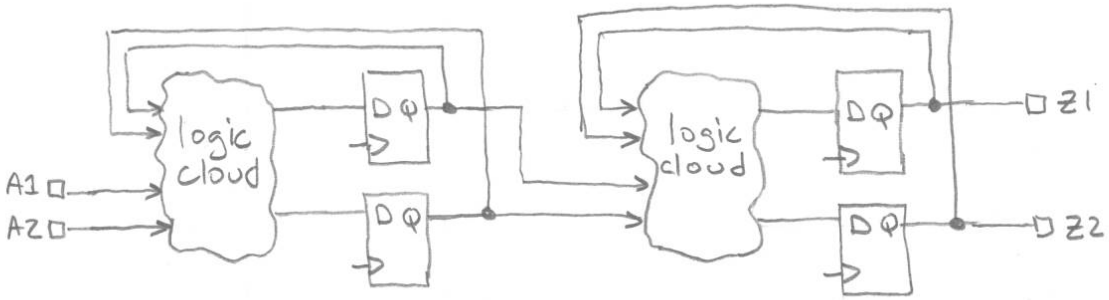
Testability - Scan Flip-Flop



- Scan FFs TI and Q pins are connected to form a long shift register

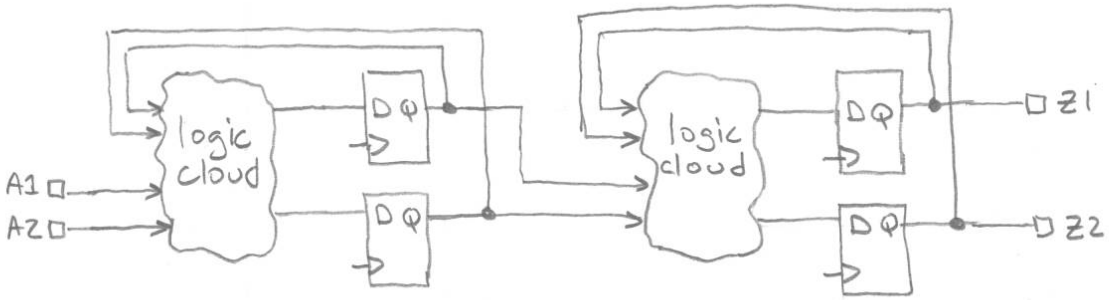
Testability - Scan-test

- Circuit prior to scan insertion

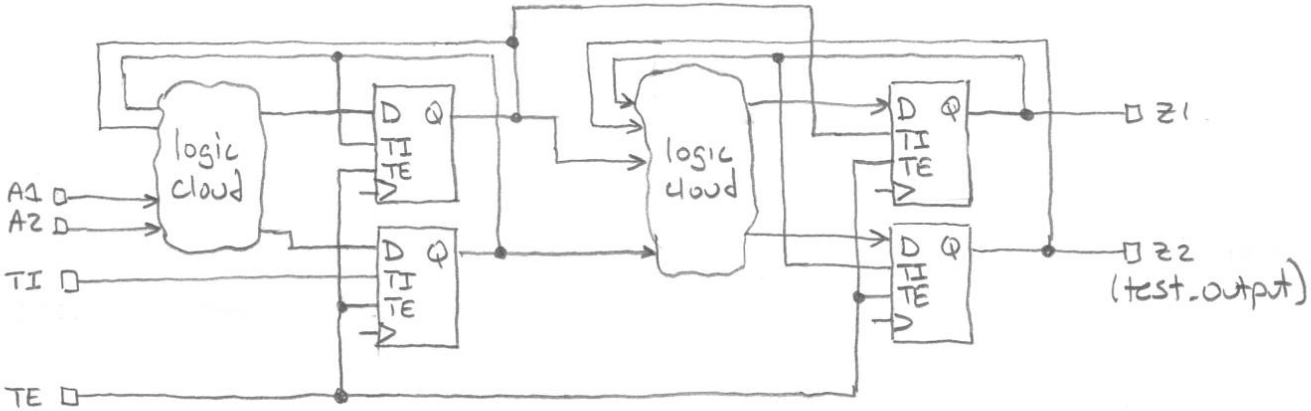


Testability - Scan-test

- Circuit prior to scan insertion

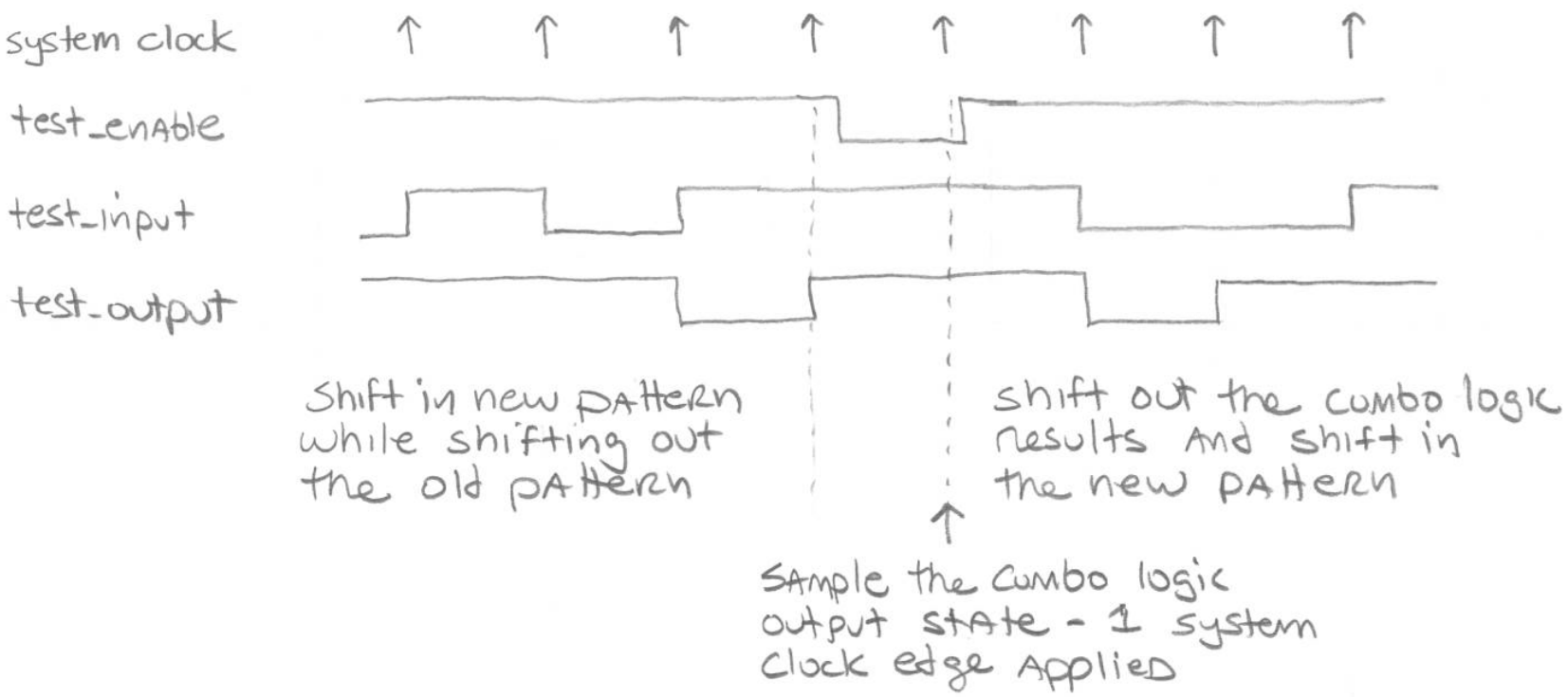


- Circuit after scan insertion



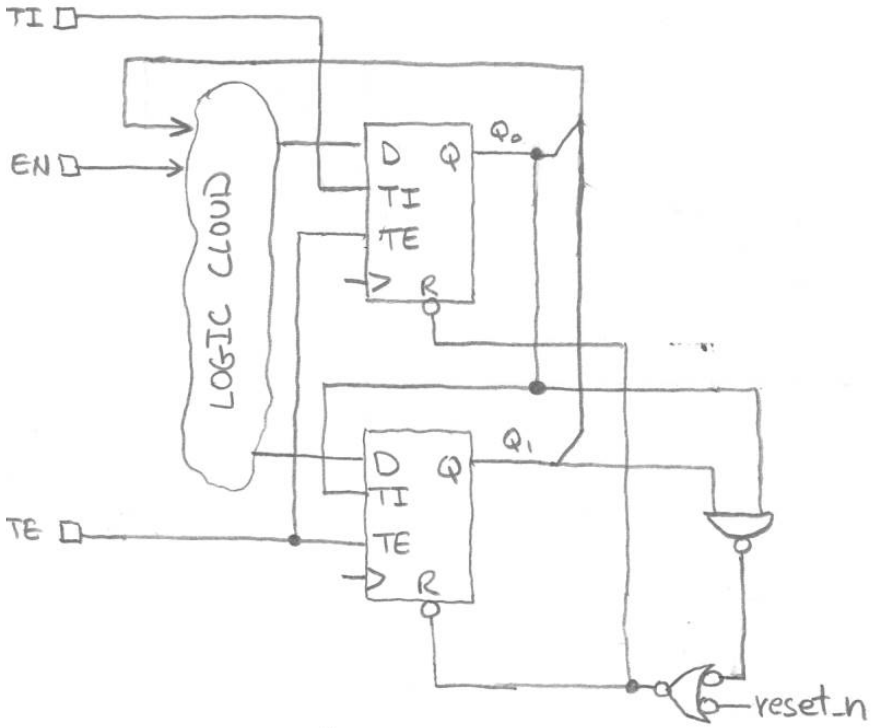
Testability - Scan-test

- TE asserted, patterns are shifted into the chip
- TE deasserted, system is clocked once, capturing combo data
- TE reasserted, captured pattern is shifted out, new pattern shifted in

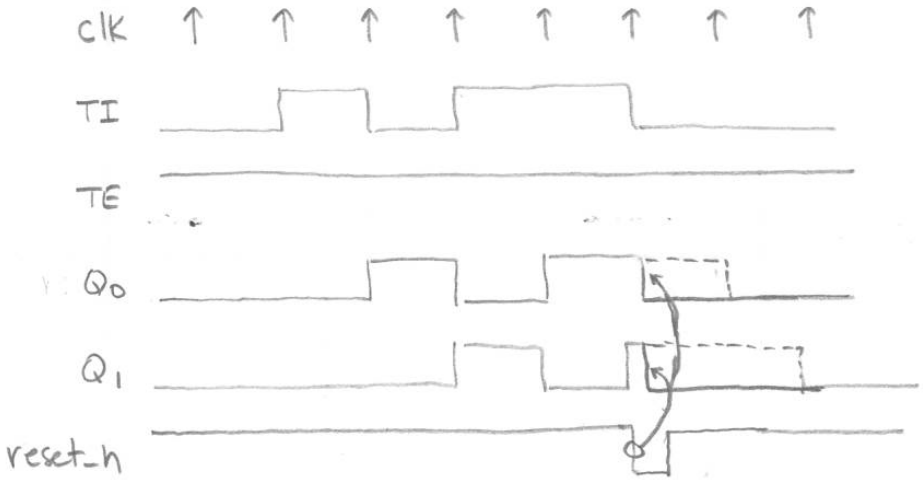


Testability - Design Implications

- Reset signals generated by internal logic



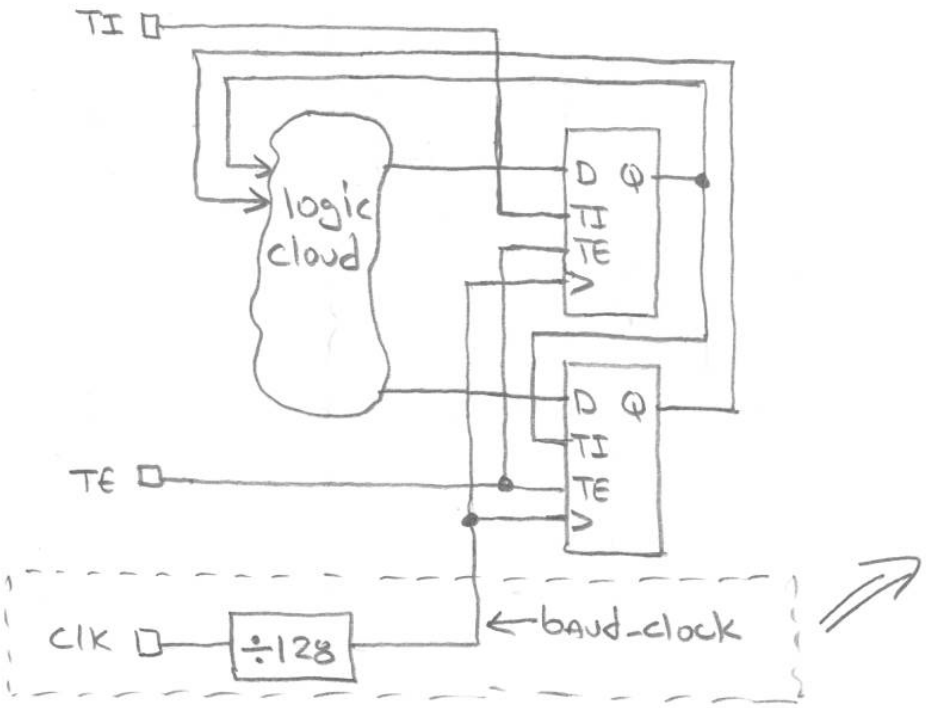
reset @ state "11" or reset_n



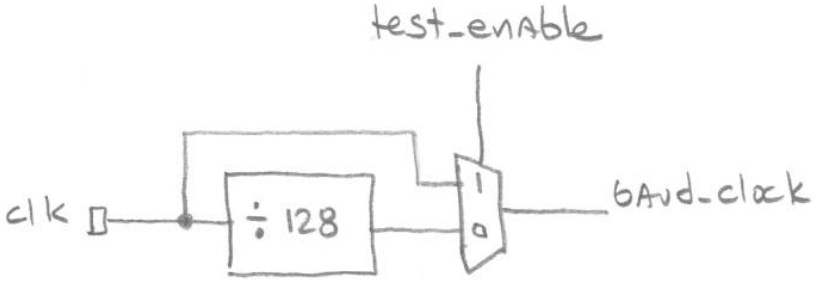
scan-in pattern can clobber itsself by combinatorial reset

Testability - Design Implications

- Clocks generated by dividers



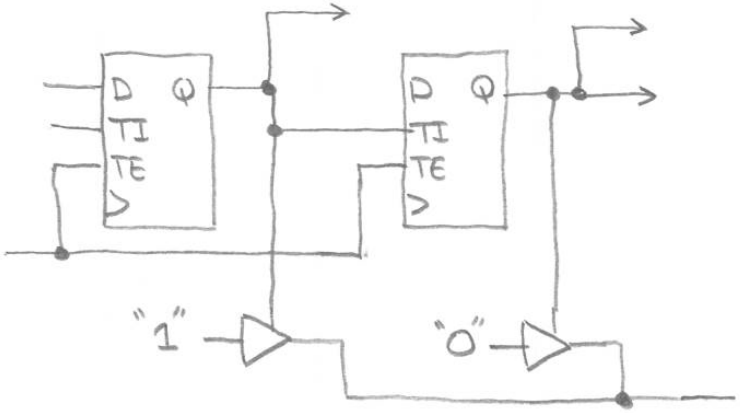
UART state machines often run off of a divided down system clock



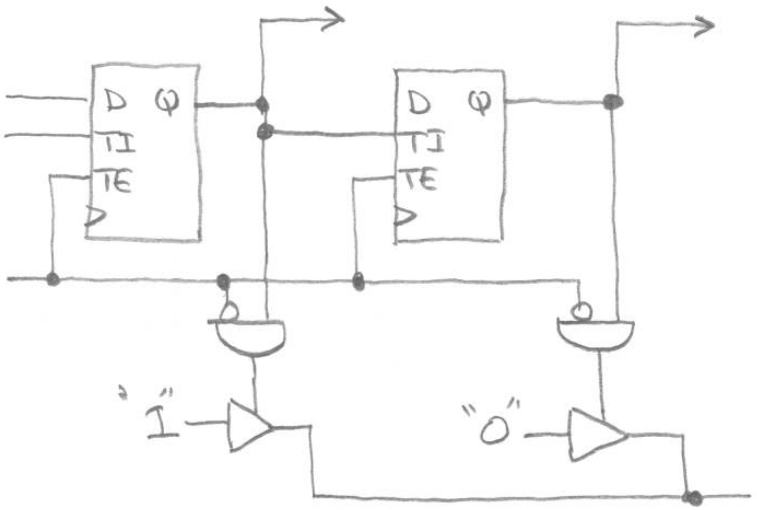
provide test circuitry a way to bypass clock dividers

Testability - Design Implications

- Tri-state buffers



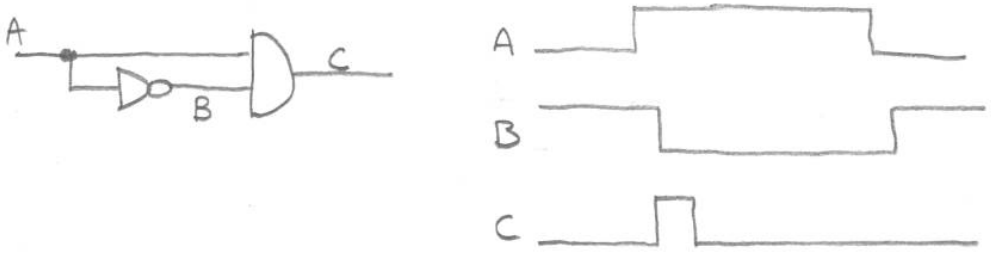
scan stéing can cause a floating line or even worst, driver contention



test tool can insert gating logic for tri-state buffers

Testability - Design Implications

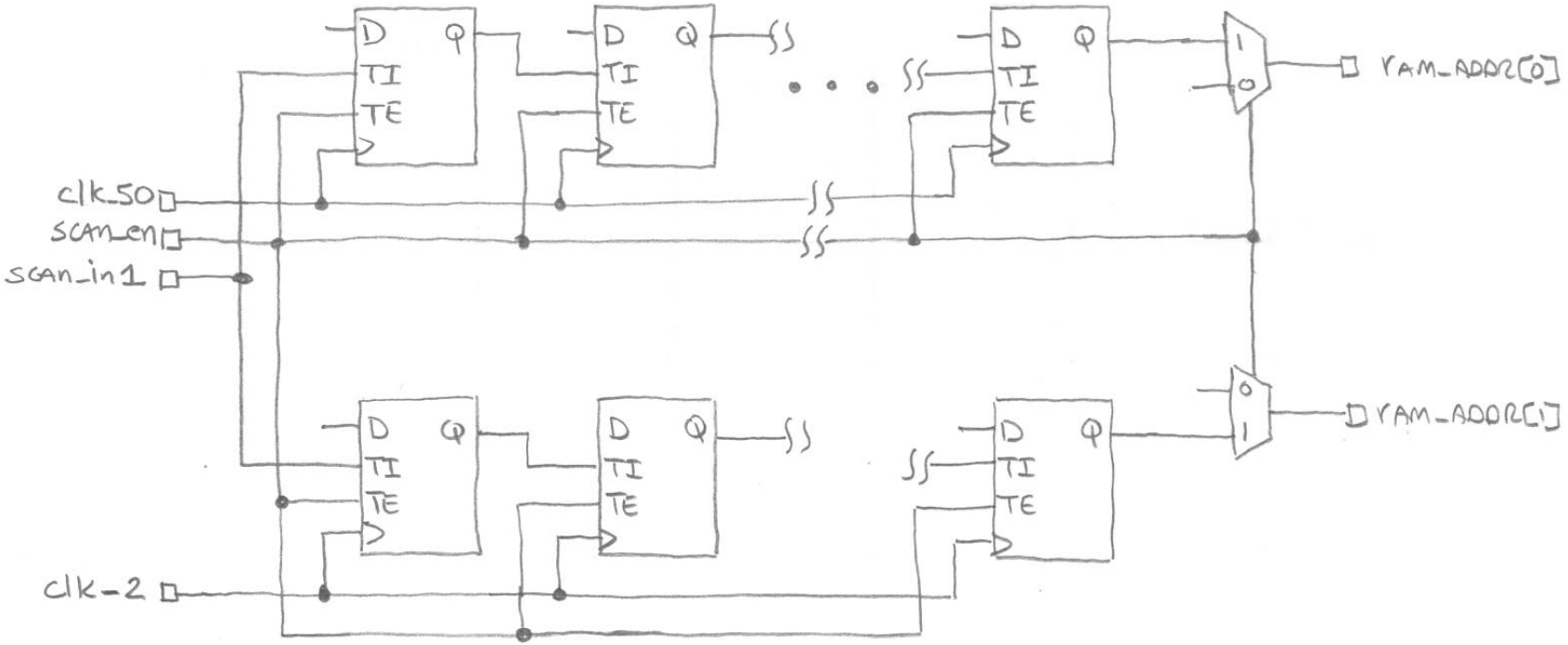
- Pulse Generators



Pulse Generators - Generally a bad idea
They can be accomidated in the test tools

Testability - Design Flow

Desired End Results of Scan Insertion



TAS with 2 scan chains, one scan input, two outputs

+

Test Vectors to Apply

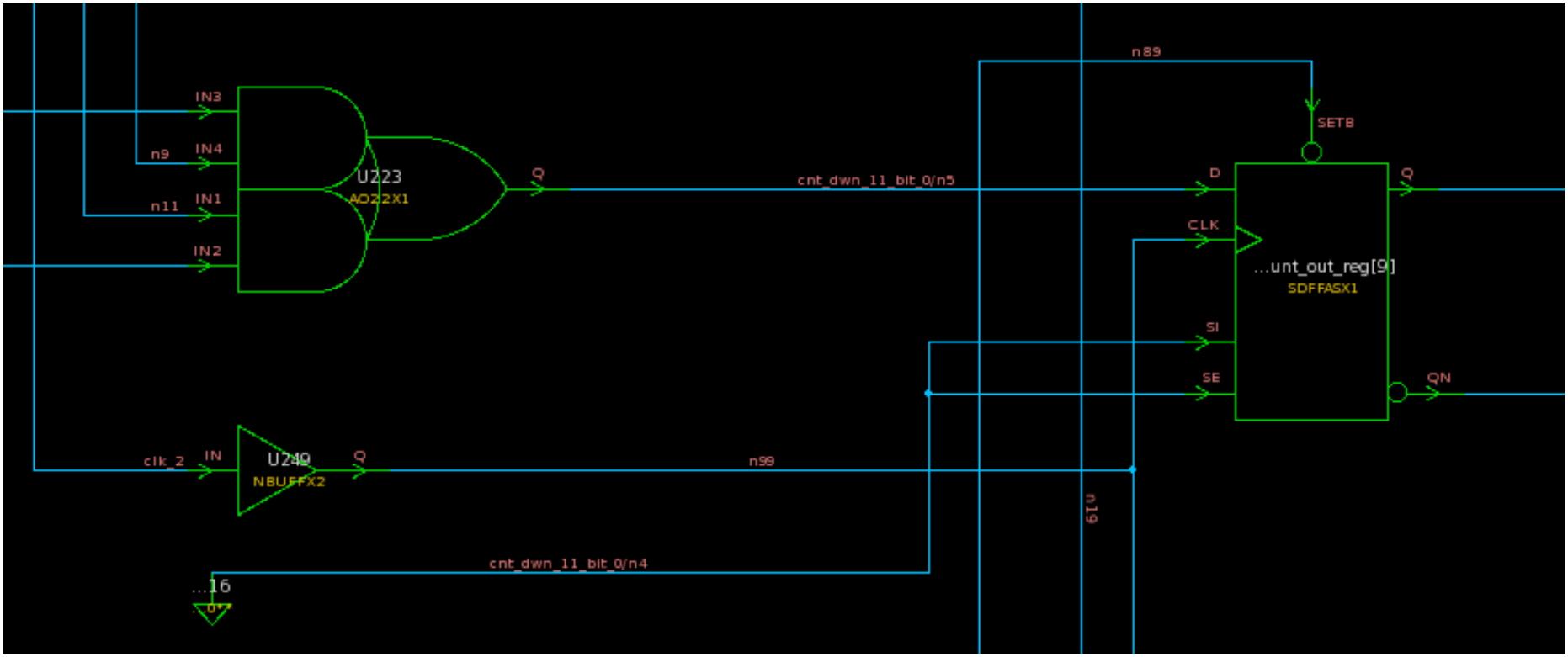
+

Instructions for ATE

Testability - Design Flow

- Usually, the initial RTL design is synthesized with scan flip-flops
- TI, TE pins are tied off to ground

```
compile -scan    #insert scan FFs but do not stitch
```



Testability - Design Flow

OLD SYNTHESIS FLOW

dc-syn

read rtl
set constraints

(synthesize) ←

write out sdf file
write out verilog netlist

implement_scan

ADDED FOR SCAN INSERTION

Synthesize using scan flip-flops

define the type of scan insertion
- muxed flip flop, 2 chains, one input

check to see if the RTL breaks scan insertion

define the I/O pins to use for scan
- scan_en, scan_in 1, ram_addr[0], [1], clks, reset_n

create the test protocol file (tas.stil) for ATE

stitch together the scan chain (insert_dft)

write out the test protocol file

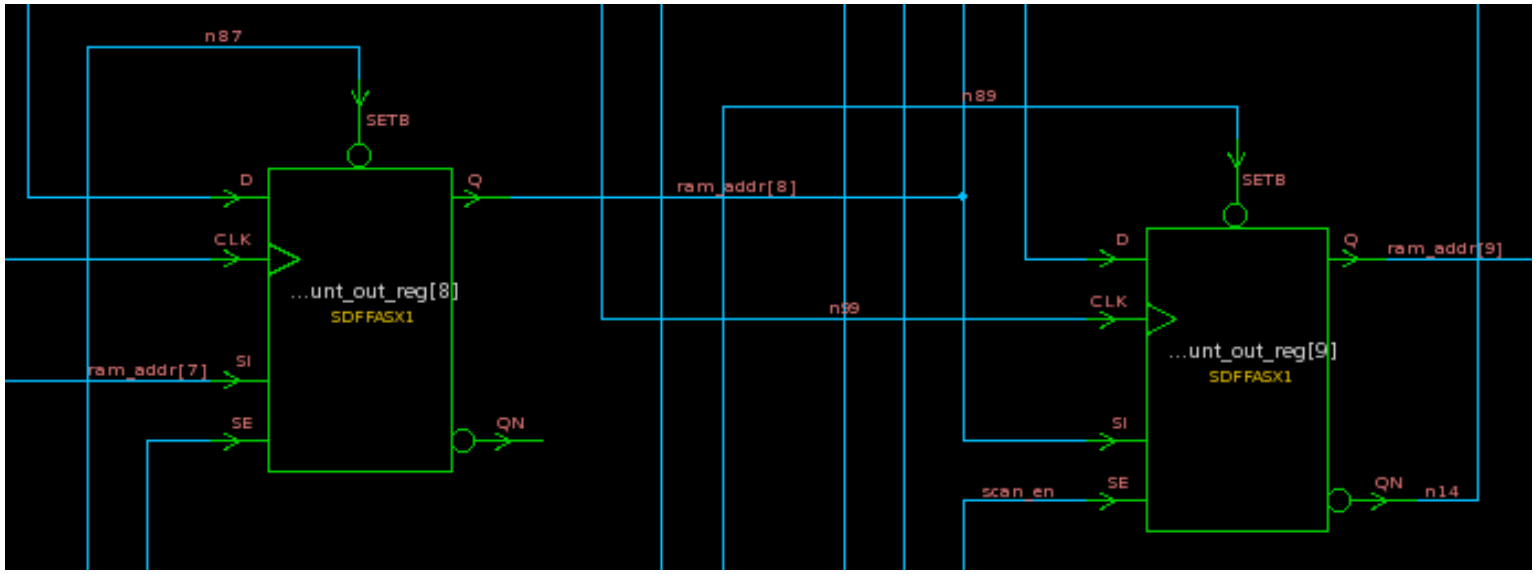
Final output of synthesis step

- verilog netlist with stitched scan chain (tas.scan.v)
- sdf file with prelayout delays (tas.scan.sdf)
- test protocol file for ATE use and fastscan (tas.stil)

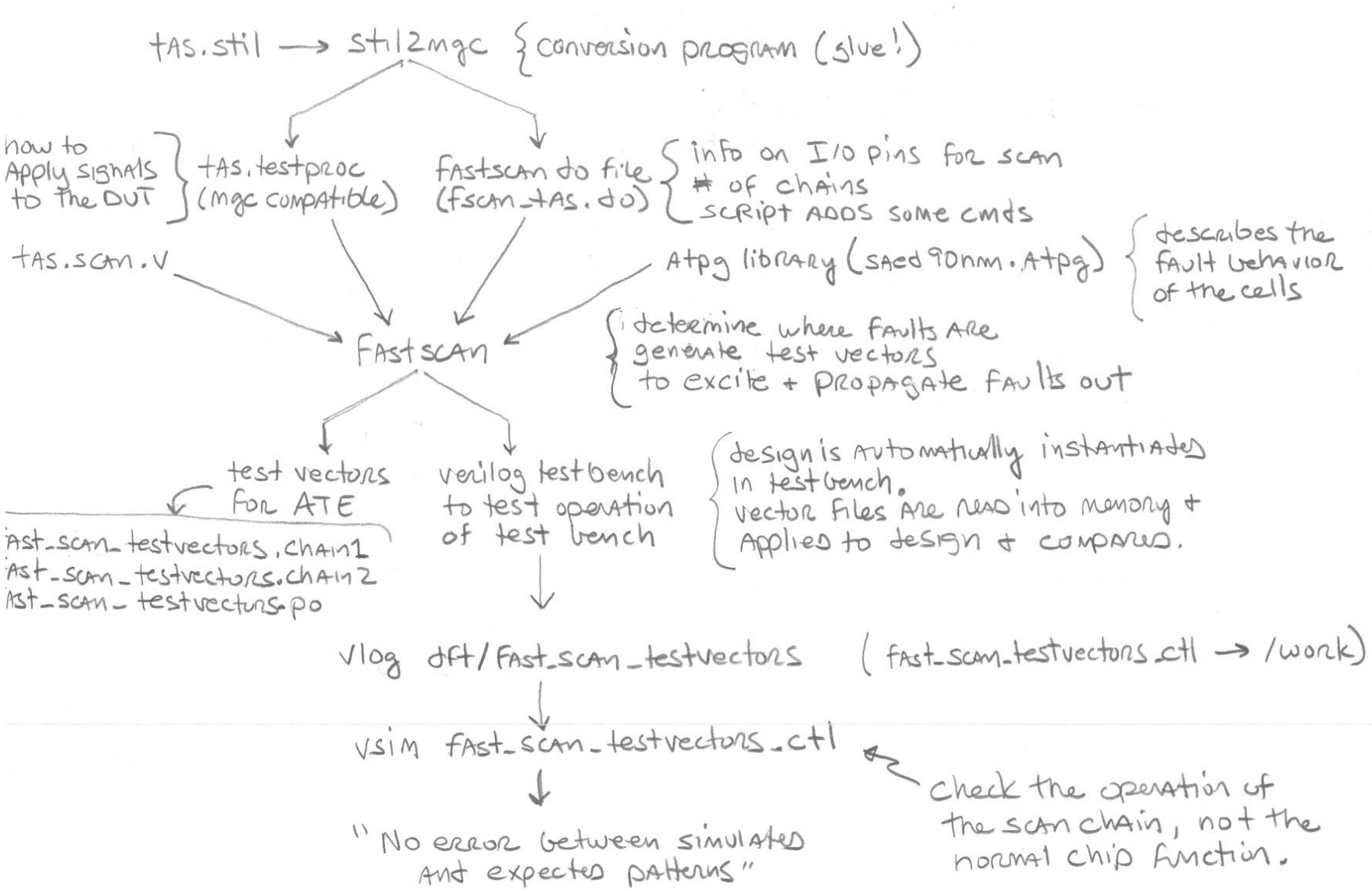
So far,
- dont know where the faults are
- dont have any test vectors

Testability - Design Flow

- Same FFs, but now the chain is stitched together



Testability - Design Flow

- We have a netlist, sdf file and test procedure file now
- We need to generate the test vectors
- "Fastscan" is optimized to generate test vectors for full scan designs
- It builds a list of all the faults in the design
- Then builds a set of vectors to detect the faults
- It also builds a verilog testbench to test the circuit and vectors!

Testability - Design Flow