Scan insertion with DRC and coverage analysis

# **The domain of the Project:**

# 

# **DFT(Design for testability)**

# **Team Mentors (and their designation):**

# **Team Members:**

1. Mr. K.Tharun Krishna B.Tech, 4th year pursuing ---- Team member
2. Mr. P.Vijay Kumar B.Tech, 4th year pursuing ---- Team member
3. Mr. P. Pavan kumar B.Tech, 4th year pursuing ---- Team member
4. Mr. K. Chandu B.Tech 4th year pursuing --- Team member

# **Period of the project**

# **May 2025 to July 2025**

**Declaration**

**The project titled “**Scan insertion with DRC and coverage analysis**” has been mentored by **Ryan Ebenezer**, organised by SURE Trust, from May 2025 to July 2025, for the benefit of the educated unemployed rural youth for gaining hands-on experience in working on industry relevant projects that would take them closer to the prospective employer. I declare that to the best of my knowledge the members of the team mentioned below, have worked on it successfully and enhanced their practical knowledge in the domain.**

Team Members:

1. **Mr. K.Tharun Krishna**
2. **Mr. P.Vijay Kumar**
3. **Mr. P. Pavan kumar**
4. **Mr. K. Chandu**

****Mr.Ryan Ebenezer****

**DFT engineer—Struent semiconductors pvt ltd**

**Prof. Radhakumari**

**Executive Director & Founder**

**SURE Trust**

**Table of contents**

1. **Detailed report**
2. **Executive summary**
3. **Introduction**
4. **Project Objectives**
5. **Methodology & Results**
6. **Social / Industry relevance of the project**
7. **Learning & Reflection**
8. **Future Scope & Conclusion**

**TABLE:-**

|  |  |  |
| --- | --- | --- |
| **DATE** | **MINUTES OF MEETING** | **ATTENDEES** |
| 26/05/25 | Assigned a task to create word document for project report that consist of three blocks  1)Minutes of meeting  2)Theoritical knowledge  3)Hands on work  Assigned another task to revise the dft concepts that was discussed earlier. | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |

|  |  |  |
| --- | --- | --- |
| 28/05/25 | Assigned a task to read Design rule checking  1)Design Rule Overview  2)Scan Insertion Checking  And clock rules  1)C Rules  2)The ATPG Analysis Option | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |
| 03/06/25 | Discussed about clock rules and clock terminology.Assigned to task to C1,C2,C3,C6  violations | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |

|  |  |  |
| --- | --- | --- |
| 07/06/25 | Clock Rule Explanation | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |
| 16/06/25 | Discussed about individual reports,assigned a task of implementing a block diagram | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |

|  |  |  |
| --- | --- | --- |
| 28/06/25 | Rectification of design and adding some Black Box | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |
| 03/07/25 | Correction of errors and adding PLL block and some other logic | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |

|  |  |  |
| --- | --- | --- |
| 06/07/25 | Solving the DRC’S formed in design | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |
| 09/07/25 | Fixing of an E5 violation | K. Tharun Krishna  P. Vijay Kumar  P. Pavan kumar  K. Chandu |

**26/05/25**

**Theoritical Knowledge:**

Design for Test (DFT) is a method used to make electronic circuits easier to test after they are built. One common problem DFT checks for is a stuck-at fault, where a signal gets stuck at 0 or 1 and doesn't change like it's supposed to.

DFT also includes special checks called DFT rule checks to make sure the design can be tested properly. Another issue DFT looks for is a transition delay fault, where a signal change (like from 0 to 1) happens too slowly or not at all.

To help with testing, designers use special flip-flops called Scan D Flip-Flops (SDFFs). These have a signal called Scan Enable (SE) that controls whether the flip-flops work normally or in test mode. The Scan In (SI) input of each flip-flop is connected to the output of the one before it, forming a chain called a scan chain.

A tool called Automatic Test Pattern Generation (ATPG) creates test patterns to find faults. It usually looks for one fault at a time. If the tool can't find a way to test for a fault, it marks it as untestable.

Sometimes, signals can be delayed due to something called a clock deficit, where the clock signal comes late. This can cause problems in real operation. To fix this, designers may add a small delay—like an extra gate or a latch (sometimes nicknamed a "cockup latch")—to give the signal more time.

**28/05/25**

**Theoritical Knowledge:**

**Scan insertion checking:**

Scan insertion checking is a critical step in the Design for Test (DFT) flow, ensuring the design is properly configured for Automatic Test Pattern Generation (ATPG). This process includes running design rule checks (DRC) to verify the integrity of scan chains, and the controllability and observability of scan cells. It also ensures correct setup of clock signals and scan enable logic.

The checks examine scan cell connections, constant data inputs, and proper handling of gated clocks and resets. These are categorized into Scannability Rules (S-Rules) and Trace Rules (T-Rules), each focusing on different aspects of scan and data path validation.

Common issues such as broken scan chains, unconstrained inputs, or incorrect scan enable configurations must be resolved before moving to ATPG. Warnings and informational notes may point to potential areas for improvement. Successfully passing scan insertion checks indicates that the design is test-ready, allowing for efficient fault detection and diagnosis during silicon testing.

**Troubleshooting Design Rule Violations:**

During the DFT (Design-for-Test) flow, identifying and addressing design rule violations is a critical step to ensure testability. Tools like **Tessent’s report\_drc\_rules** and **analyze\_drc\_violation** help detect and classify issues such as scan chain connectivity errors, clock control problems, data path violations, and constraint mismatches. To aid in debugging, **Tessent Visualizer** provides a graphical interface to visualize scan cells, clock domains, and signal paths affected by these violations.

Troubleshooting involves tracing the root cause of each violation by inspecting signal definitions, verifying design logic, reviewing constraints, and correcting scan chain configurations. After implementing fixes in the RTL or design constraints, designers rerun DRC checks to confirm resolution. This iterative process ensures that the design meets DFT requirements and is prepared for ATPG and silicon validation.

**Clock Rules (C-Rules):**

Clock rules are used to verify the correct definition and behavior of scan clocks in a design. These checks focus on validating clock sources, ensuring proper connections, and confirming the presence and correctness of clock control signals. The goal is to prevent issues that could compromise scan data integrity during test operations.

Violations detected by clock rules are categorized by severity. An **error** indicates a critical problem that must be resolved before the design can proceed further, as it could prevent correct test pattern generation or application. A **warning** highlights a significant issue that might affect test quality but does not necessarily stop progress, allowing the designer to continue with caution. A **note** is simply informational, pointing out non-critical conditions that may or may not need attention. Lastly, a rule can be set to **ignore**, meaning the check is skipped entirely—usually done when the rule does not apply to the specific design context.

Addressing clock rule violations effectively is essential for ensuring robust scan operation and achieving high fault coverage in the final silicon.

**Clock Terminology:**

The tool considers any signal to be a clock if it can change the state of a sequential

element, including system clocks, sets, and resets.

Two important terms arise out, that are:

• **Leading Edge**: The transition of the clock from the off state to the on state is considered the leading edge of the clock.

• **Falling Edge:** The transition from the on state to the off state is considered the trailing edge of the clock.

**Clock Cones and Effect Cones:**

In Design-for-Test (DFT) and clock analysis, **clock cones** and **effect cones** are used to understand how clock signals influence different parts of a design.

A **Clock Cone** refers to the **set of all sequential elements (e.g., flip-flops, latches)** that are **directly driven by a specific clock signal**. It defines the **region of the circuit** that is controlled by that particular clock. Analyzing clock cones helps ensure that scan clocks are properly defined, isolated, and do not interfere with each other.

An **Effect Cone**, on the other hand, represents the **set of logic elements or outputs** that are **affected by the operation of a clock cone**. This includes both the directly clocked elements and the **combinational logic downstream** that receives data from those elements. Effect cones help assess the **influence of a clock signal** throughout the design and identify regions impacted by clock control or enable logic.

Understanding and analyzing both cones is essential for:

* Ensuring proper clock domain separation
* Avoiding unintended interactions between clocks
* Enhancing testability and scan insertion accuracy
* Supporting robust ATPG pattern generation

**Clock Source and Derived Clocks :**

In Scan Cell Data Rules (D rules), **clock rule checks**—such as those performed using **DFT\_C5** (Memory BIST clock tracing)—require the definition of **three types of clocks** at either **primary ports** or **instance pins**: **synchronous clocks**, **asynchronous clocks**, and **derived clocks**, which include both **clock sources** and their **branches**.

**Clock Source (Synchronous or Asynchronous):**

* The origin point of a clock signal.
* Can come from an external clock generator or an internal clock block.
* Typically defined at a well-known pin or instance pin.
* Provides the main clock pulse to drive the design.
* **Derived Clock Source:**
* A clock that is based on another clock but passes through additional logic or routing.
* May require a reference back to the original clock to trace correctly, especially if the path is blocked.
* Common in designs with PLLs, where the -reference option links it to the original source clock.
* **Derived Clock Branch:**
* A branch of the clock tree that originates from a clock source or derived clock source.
* Can become unbalanced or unsynchronized, even if driven by the same source.
* Defined using the add\_clocks -branch option.

• Must be driven by a valid clock source or derived clock source.

• Can be modified later during clock network operations.

**03/06/25**

**Theoritical Knowledge:**

### ****C1 DRC Rule – Clock Domain Check****

* Category: **Clock**
* Contexts Supported:
  + dft -scan
  + dft -test\_points
  + patterns -scan
  + patterns -scan\_diagnosis
* Default Handling: **Error**
* Supported by: report\_drc\_rules command

### ****Rule Description:****

* Ensures **scan and non-scan cells do not capture data** when all clocks are in the **off state**.
* When clocks are off (defined via add\_clocks), all clock-related inputs (including **set/reset pins**) must also be off.
* For **scan D flip-flops (DFFs)**:
  + Clock input must be a **stable binary value (0 or 1)**.
  + If clock input is X, tool checks if it can be resolved to 0 or 1.

### ****Violation Handling:****

* Non-scan cell violations → Converted to **TIEX**.
* Scan cell clock input not off → **C1 violation**.
* Violations on **set/reset or latch enable** pins:
  + **Not reported as errors**
  + Excluded from violation list
  + Viewable via: report\_drc\_rules C1 -excluded

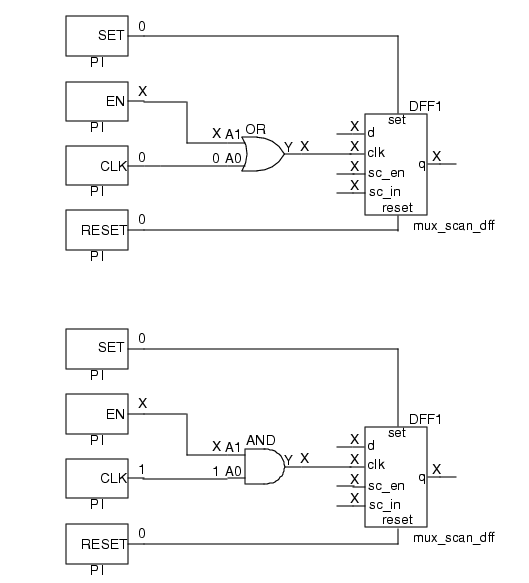
### ****Impact:****

* **High number** of excluded cells can **increase ATPG runtime**.
* Common cause: **Set/reset pins not declared as clocks**.

### ****Fix Recommendation:****

* Use add\_clocks to define **set/reset pins as clocks** to avoid violations and ATPG performance issues.

## ****Figure C1 Violation Example:****

**The clock input value to the scan cell is X because the EN signal value is X. If EN is left at X,the signal arriving at the clk input of the scan cell cannot be held off.**

## ****C2 Violation : “Clock with No Destination”****

The **C2 rule** checks if **every defined clock** actually **reaches** a **memory element** (like a flip-flop, latch, or RAM).

If a clock **does not reach any memory element’s clock/set/reset pin**, it triggers a **C2 violation**.

**Why It Happens (Common Causes)**

| Cause | Explanation |
| --- | --- |
| **Clock is wrongly defined** | You defined a pin as a clock (add\_clocks) but it doesn’t behave like clock. |
| **No path to FF or latch** | The net connected to the "clock" does not fan out to any clock pin. |
| **Floating or unused net** | Pin defined as clock is disconnected or trimmed during synthesis. |
| **Design has internal PLL clocks** | External clock ports (e.g. ref clock) may appear unused in flat netlist. |

**How to Fix a C2 Violation:**

1.Identify the bad clock pin

**2.Check if the pin is actually a clock**

* If **not really a clock**, remove it

3.If it is a clock but no FF uses it

Fix the netlist connection.

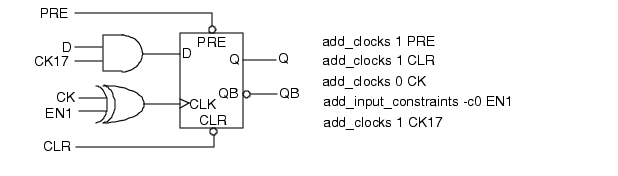
* Make sure the clock signal **reaches a flip-flop or latch**.

4.**If it’s a reference clock in a PLL-based design (not connected in netlist):**

* You may **ignore** the warning — the tool suppresses false violations in that case.

5.Re-run DRC.

**Figure C2 Rule Example Circuit**

If you run rules checking on this design, you get a C2 rules violation because while the CK17signal appears to be a clock (due to its name), it cannot be reached from the clock port or SET/RESET ports of the flip-flop. To fix this problem, add the command:

delete\_clocks CK17

Then, you must re-run checks.

**C3 Violation (Clock Capture Conflicts)**

A **C3 violation** happens when a **sink register (DFF2)** captures data from a **source register (DFF1)** **in the same clock cycle**. This can cause the **sink to latch the source's new data**, not its old one, breaking the assumption that only **old data** is captured.

### ****When It Happens:****

* **Both source and sink** are triggered by the **same clock**.
* **Sink's data input** is in the **effect cone** of the clock.
* **Sink's clock input** is in the **clock cone** of the same clock.
* **Specific cases:**
  + TE (Trailing Edge) sink gets data from LE (Leading Edge) source.
  + Latch with data in effect cone and clock in clock cone.
  + RAM with write/read inputs in the clock cone and data/address in the effect cone.

### ****How the Tool Detects It:****

* Traces **clock cone** (what the clock affects) and **effect cone** (what affects the clock).
* Uses **partial ATPG analysis** by default.
* Can use **full ATPG analysis** for deeper checks:  
  set\_drc\_handling -Atpg\_analysis

**Debugging Steps:**

1.Run:  
report\_drc\_rules C3  
(Lists all C3 violations.)

2.For a specific case:  
report\_drc\_rules C3-1  
Example Output:  
Clock /CLK failed rule C3 on input 3 of /DFF2. Source: input 3 of /DFF1.

3.Use **Tessent Visualizer** to see the violation graphically.

**Screening Out False C3 Violations**:****

**1.Enable full ATPG justification**

* Screens out impossible scenarios (mutually exclusive control signals).
* **This checks if the conflicting values can actually occur together.**

**2.Check mutual exclusivity:**

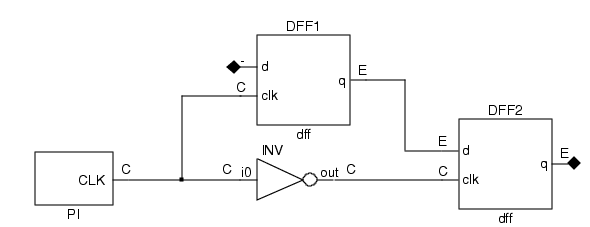
* The tool analyzes if **clock/write/read** and their data inputs can be active **at the same time**.
* If not, the tool **ignores** the violation.

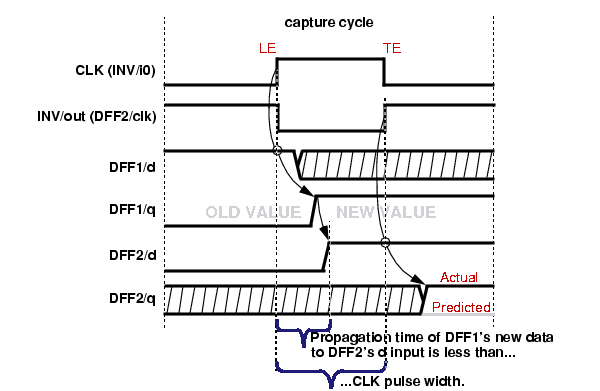
**3.Best Practice:  
 Always use full ATPG analysis before DRC to filter out non-issues and keep reports meaningful.**

## ****Why C3 Matters****

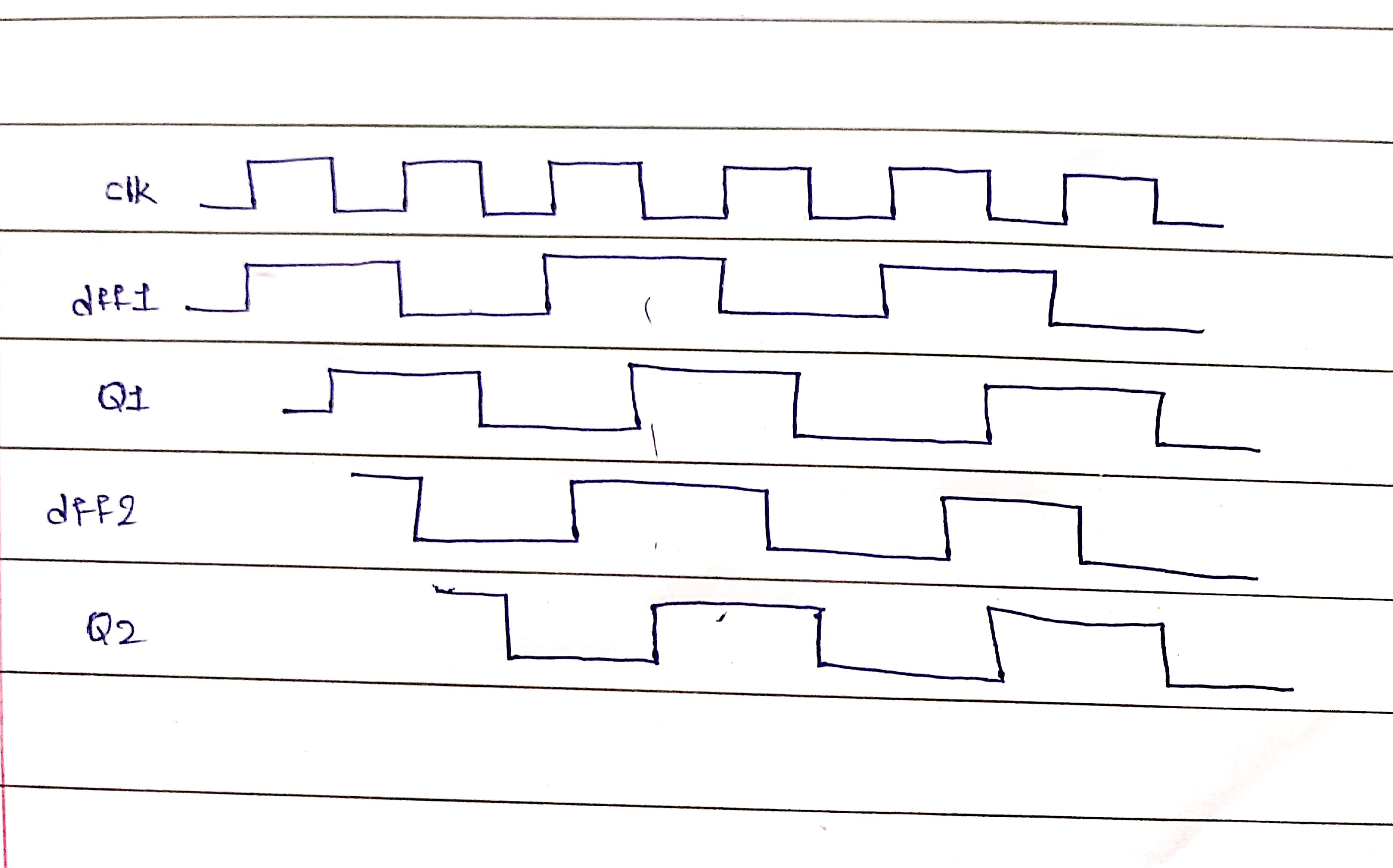
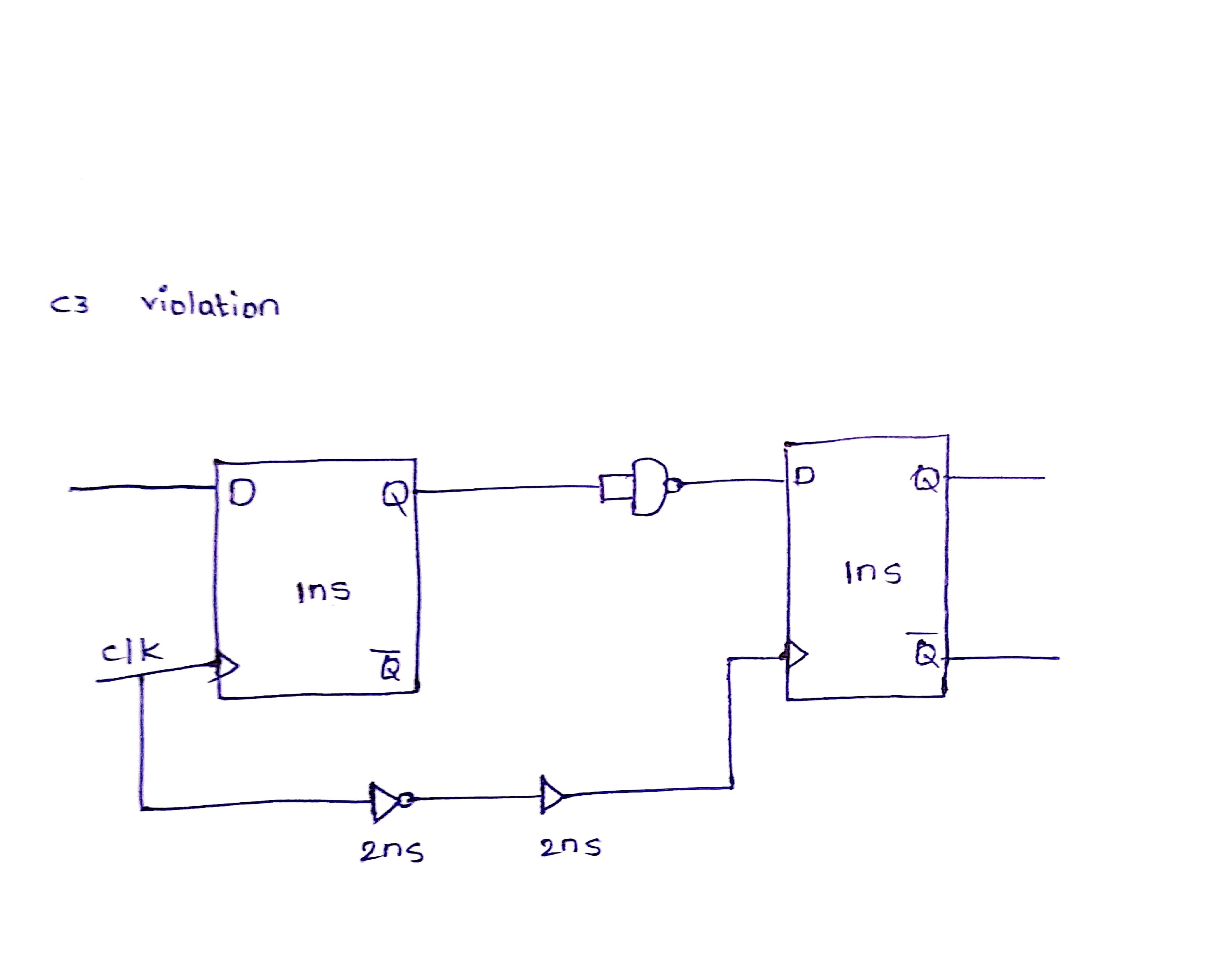
* If unresolved, it leads to:
  + **Simulation mismatches**
  + **Tester failures**
  + **Debugging headaches**

****Figure : C3 Violation Example****

****Figure : Example Timing That Allows Sink to Capture Source’s New Data****



**The tool performs this check by determining the forward cones of influence for a clock pin (itsclock cones). For an introduction to clock and effect cones, refer to the clock cones and effect cones discussion under “Clock Signals.” The bounds for the clock cones are scan cells andcircuitry set to a fixed value when the constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The clock cone stops at read ports ofRAMs that have the read\_off attribute set to hold, and then the effect cone propagates from its outputs.**



****C6 Violation : “Clock Shouldn’t Affect Its Own Data”****

**A C6 violation happens when:**

* The clock signal (CLK) not only triggers a memory element (like a DFF)
* but also influences the data (D) input of that same memory element.

**This creates a race condition — the clock is capturing data that it potentially changed itself — leading to inaccurate simulation results.**

**How the Tool Detects C6 Violations:**

- Builds a clock cone (everything influenced by the clock).

- If both CLK and D inputs of a scan cell are in the cone, it’s a C6 violation.

- Example message: Clock P failed rule C6 on input I of N (G). (C6-1)

P=clock pin name

I=input number

N=gate/instance name

G=gate ID

**Debugging a C6 Violation:**

1. Run: set\_gate\_report drc C6-1

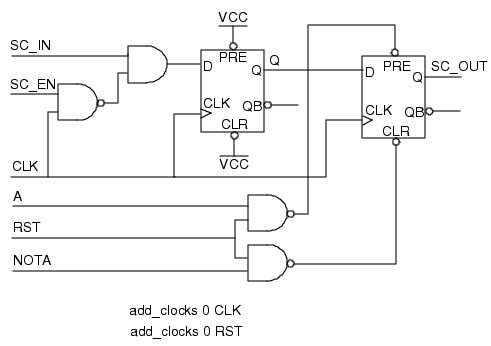
2. Report gate info: report\_gates

3. Trace back from input to identify clock influence on data.

**Risks if Unresolved:**

* Failing ATPG patterns
* Flaky debug process
* Scan simulation mismatches

**Figure :C6 Rule Example Circuit**



If you ran rules checking on this design, given the setup commands shown, you would get a C6rules violation. The default handling for this rule violation is warning. In this design, the CLK signal goes to both the CLK and D inputs of the first flip-flop. Thus, data can be captured in this flip-flop that may be affected by the capturing clock. To prevent the CLK signal from influencing the data, add the command:

add\_input\_constraints -c0 SC\_EN

Constraining the SC\_EN signal to 0 ensures that changes in the clock do not change the data.

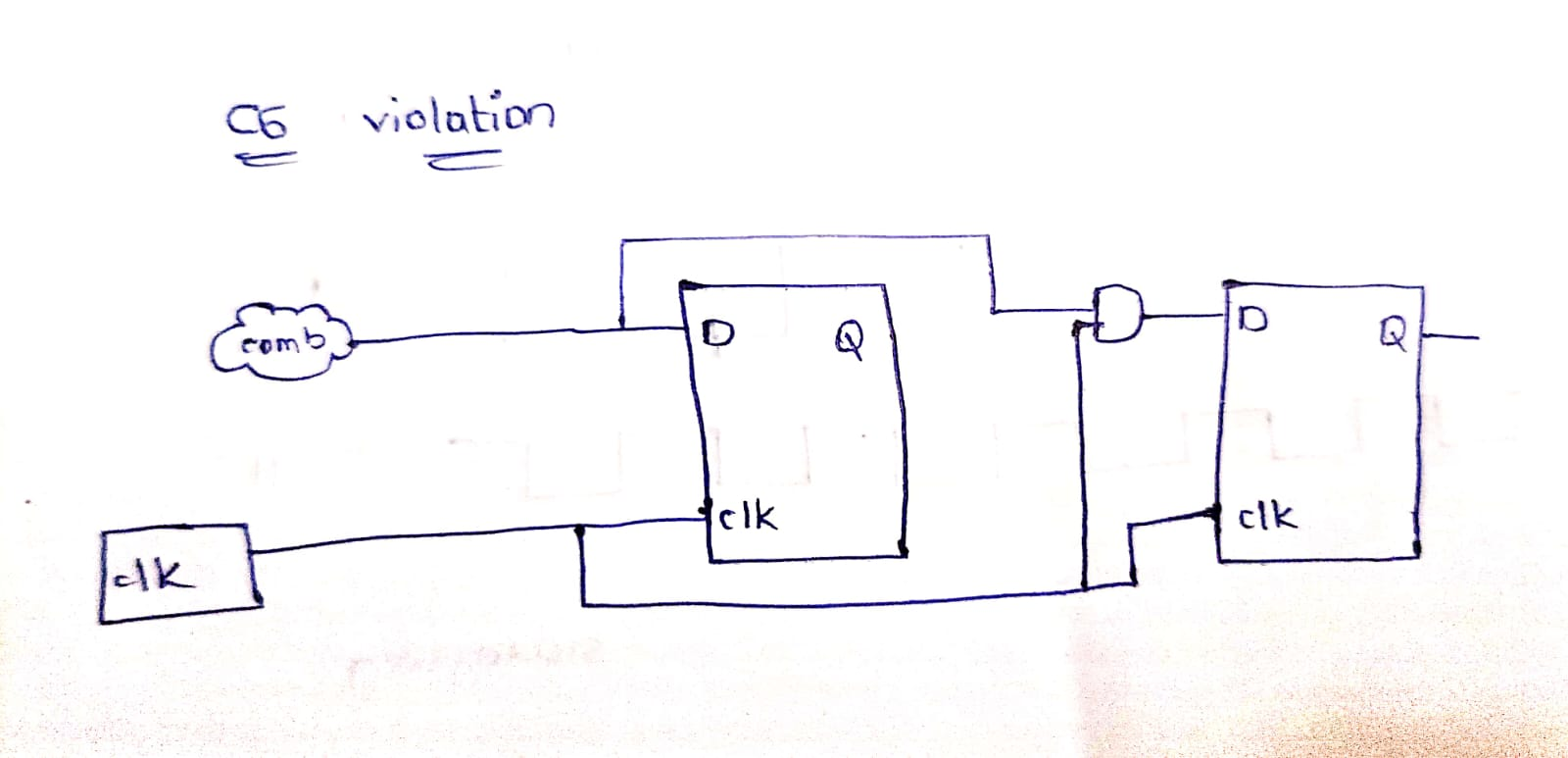
For ATPG, you can handle C6 rule violations by issuing“set\_clock\_off\_simulation on”command prior to creating patterns. See the set\_clock\_off\_simulation command description for additional information.

In some designs, different flip-flops may capture the off state and on state of the clock, causing the simulation to fail. In this case, you can use the following command to mask the signal

creating the C6 violation:

set\_simulation\_options -C6\_mask\_races ON

For more information, see the set\_simulation\_options command.

****

**10/06/25**

**Theoritical Knowledge:**

**D5 Rule:**

**Category:** Data

**Supported Contexts:**

dft -scan

dft -test\_points

patterns -scan

patterns -scan\_diagnosis

**Default Handling:** Warning

**report\_drc\_rules:** Supported

**Rule Description:**

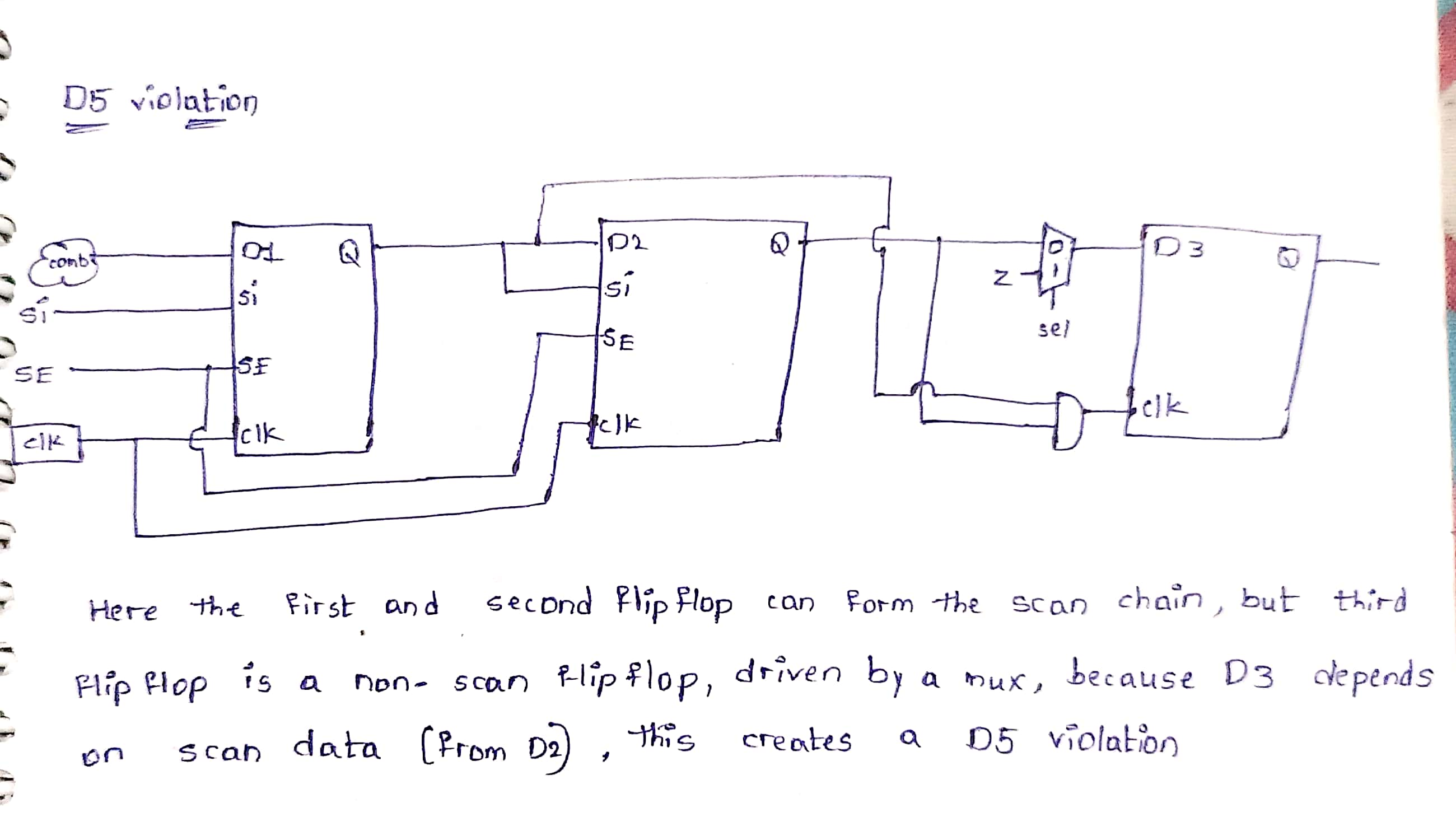
All memory elements (latches and flip-flops) **must be scannable**.  
This check is done **after** identifying all scan memory elements.A **D5 violation** occurs when a memory element is **not part of a scan cell**.

**Simulation Behavior:**

Circuit is simulated as **combinational** (sequential depth = 0 or 1).

* Based on simulation values before capture, each non-scan element is **modeled as**:

| Type | Meaning |
| --- | --- |
| INIT-0 | Value = 0 at start of first capture cycle |
| INIT-1 | Value = 1 at start of first capture cycle |
| INIT-X | Value is unknown (X) at start of first capture cycle |
| TIE-0 | Always 0 during capture |
| TIE-1 | Always 1 during capture |
| TIE-X | Always unknown (X) during capture |
| TLA | Transparent when its clock is off |

****

## ****Extra Rule E5 – Observable X-State Propagation****

### ****Objective:****

To identify scenarios where **X states** (unknown logic values) may **propagate to observable points** (e.g., primary outputs or scan observe points) during **scan testing**, particularly in **compressed test environments** (e.g., BIST, EDT).

### ****What It Verifies:****

The tool performs a constrained simulation to check if any **gate produces an X** that can propagate to:

* Primary outputs
* Scan observation points

**Simulation Conditions:**

* Binary values at **Primary Inputs (PIs)** and **Scan Cells**
* Constrained values on **control pins**

Only those gates that:

* Produce an X **AND**
* Have a **sensitizable path** to observation points  
  are flagged.

### ****Violation Conditions (Triggers for E5 Violation):****

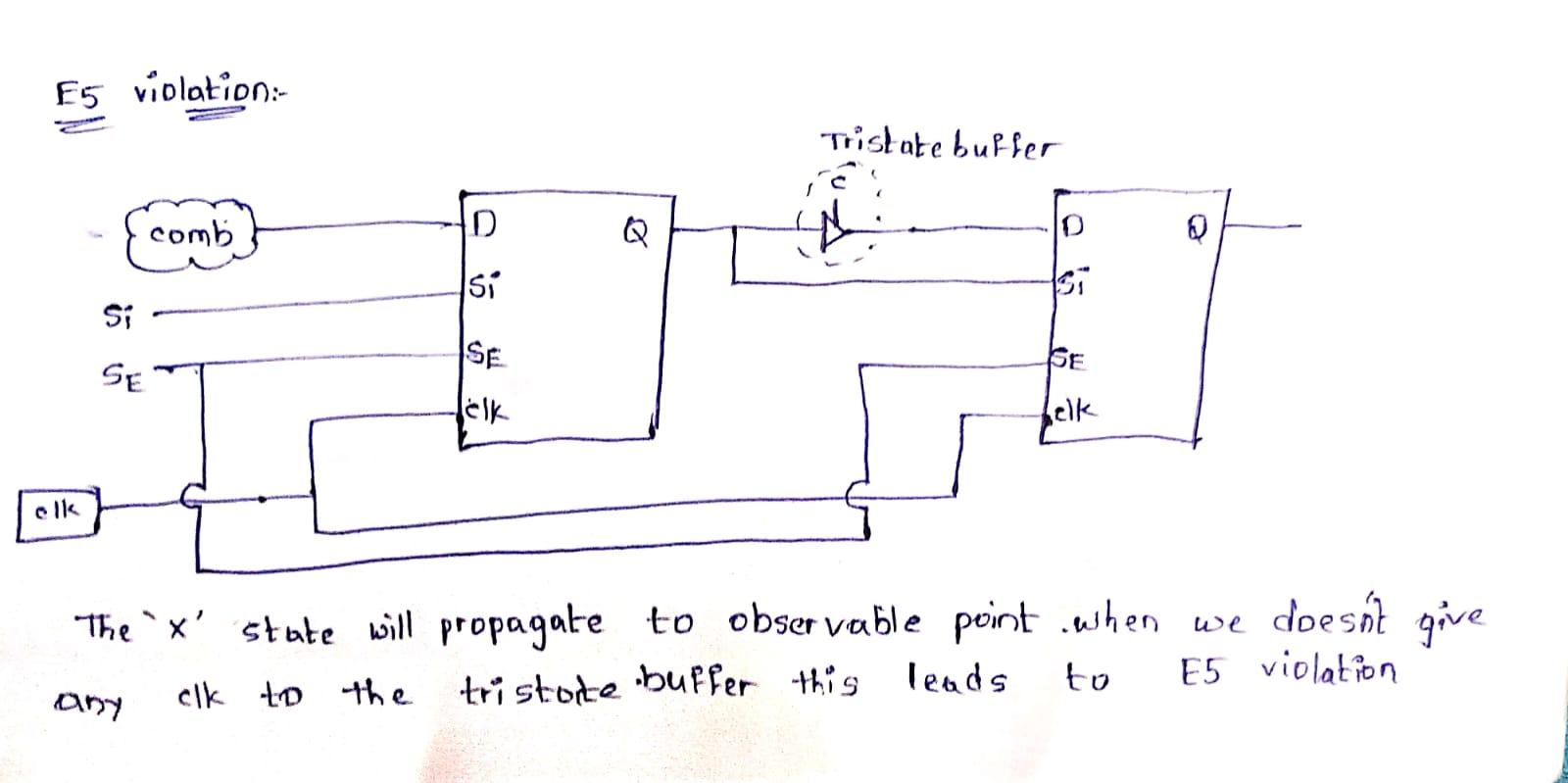
| Component | Violation Description |
| --- | --- |
| **WIRE gates** | Input mismatch leads to logic resolution producing an X |
| **BUS gates** | - Multiple tri-state drivers ON at once - All drivers OFF (produces Z → interpreted as X) |
| **Tri-State Drivers/Switches** | Output Z while not connected to a bus (Z treated as X) |
| **TIE-X gates** | Sensitizable X reaches an output |
| **Transparent Latches (TLA)** | Improper clocking or uncontrolled set/reset signals |
| **ROM/RAM** | - Read line OFF + read\_off = X - Uninitialized memory outputs |
| **Primary Inputs** | PI drives Z (treated as X unless NO\_Z constraint is active) |

### ****Consequences of an E5 Violation:****

* X states that propagate to outputs:
  + **Corrupt scan test results**
  + **Reduce fault coverage**
  + **Increase pattern count** in compressed ATPG (Automatic Test Pattern Generation)
  + **Degrade test compression** (e.g., with TestKompress or EDT)

### ****Example Scenario:****

* A **tri-state bus** has multiple drivers enabled or none enabled → output is X.
* If this X travels through the logic and reaches an output, it triggers an E5 violation.
* In **compressed testing**, this unmasked X:
  + Cannot be easily ignored or suppressed
  + Forces ATPG to generate **extra patterns**, reducing **compression efficiency**

****

**Test Point Rule T3 – Ineffective Test Point**

### The T3 trace violation occurs when the DFT tool fails to trace a complete

sensitizable path from a scan chain output (scan\_out) back to a scan chain input (scan\_in) during scan shift path analysis.This means that during scan shifting,some logic or gate is blocking the propagation of data, breaking the shift path required for proper scan operation.

When the tool encounters the error, it reports a message similar to the following:

// Error: Scan chain chain1 blocked at gate /datao/reg\_q\_0\_ (360) after

tracing 0 cells. (T3-1)

// Error: Rules checking unsuccessful, cannot exit SETUP mode.

Correct this error condition by accessing the simulated values of all time

periods of the shift procedure.

Scan chain S blocked at gate N (G) after tracing C cells. (T3-1)

This means:

•**S:** The scan chain being traced

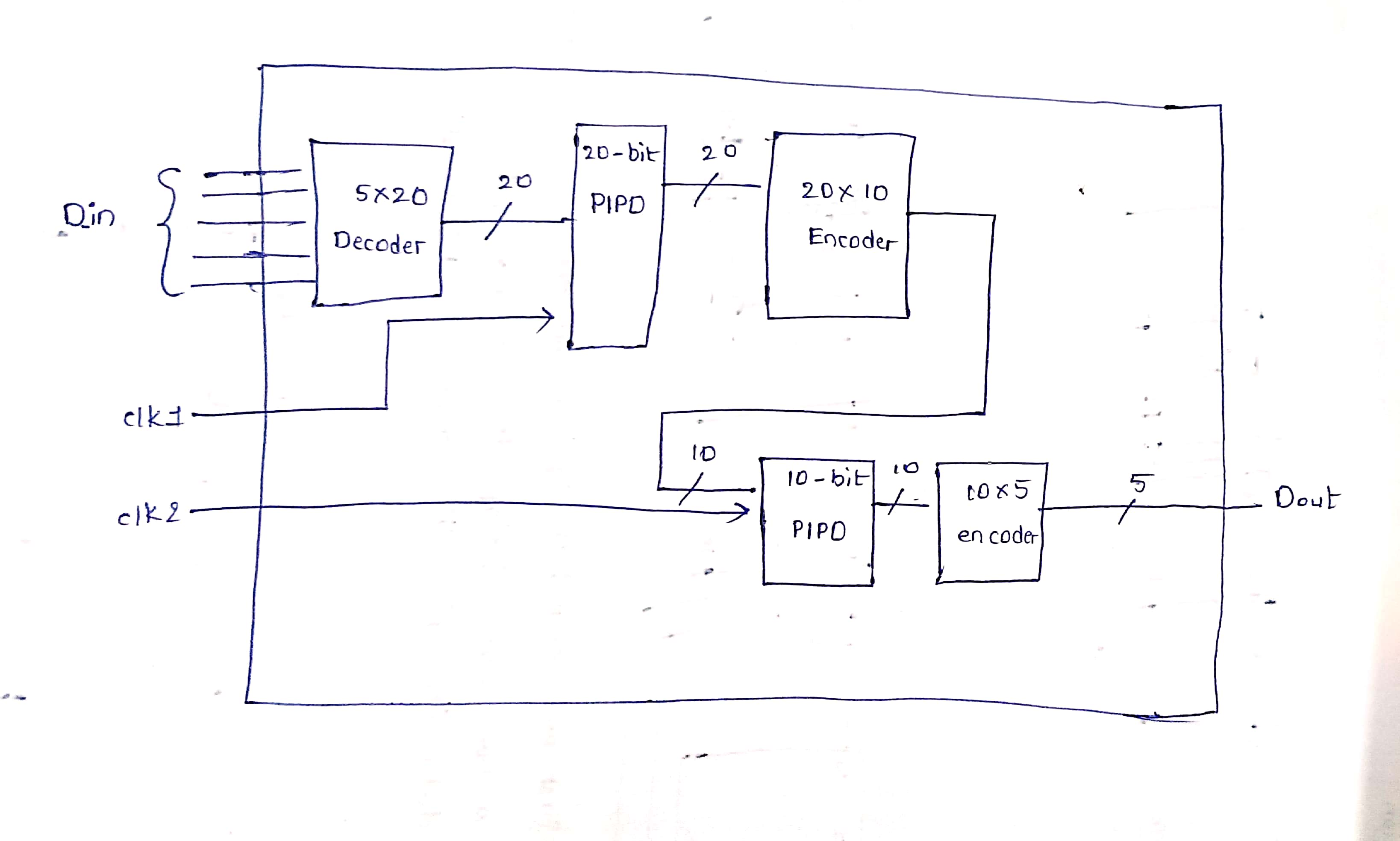
•**N:** Name of the gate where tracing failed

**•G:** Gate type

**•C:** Number of scan cells successfully traced before blockage

**16/06/25**

**Hands On Work:**

****

**Design:**

`timescale 1ns / 1ps

module golden\_design(refclk,clk2,data\_in,reset,data\_out);

input refclk,clk2;

input [4:0]data\_in;

input reset;

output [4:0]data\_out;

wire [19:0] deco1,pipo1;

wire [9:0] enco1,blackbox\_out,pipo2;

wire [9:0] x;

decoder5x20 D1(data\_in,deco1);

pll PLL(refclk,clk1);

pipo20 P1(clk1,reset,deco1,pipo1);

encoder20x10 E1(pipo1,enco1);

pipo10 P2(clk2,reset\_out,x,pipo2);

encoder10x5 E2(pipo2,data\_out);

blackbox BB(

.reset\_out(reset\_out),

.data\_out(blackbox\_out)

);

xor (x[0],blackbox\_out[0],enco1[0]);

xor (x[1],blackbox\_out[1],enco1[1]);

xor (x[2],blackbox\_out[2],enco1[2]);

xor (x[3],blackbox\_out[3],enco1[3]);

xor (x[4],blackbox\_out[4],enco1[4]);

xor (x[5],blackbox\_out[5],enco1[5]);

xor (x[6],blackbox\_out[6],enco1[6]);

xor (x[7],blackbox\_out[7],enco1[7]);

xor (x[8],blackbox\_out[8],enco1[8]);

xor (x[9],blackbox\_out[9],enco1[9]);

endmodule

module pll(

input refclk,

output clk

);

//assign clk=refclk;

endmodule

module blackbox(

output reset\_out,

output [9:0]data\_out

);

//assign reset\_out = 1'b0;

//assign data\_out = 10'd0;

endmodule

module decoder5x20 (

input [4:0] in,

output [19:0] out

);

wire A, B, C, D, E;

wire nA, nB, nC, nD, nE;

assign A = in[4];

assign B = in[3];

assign C = in[2];

assign D = in[1];

assign E = in[0];

not g0(nA, A);

not g1(nB, B);

not g2(nC, C);

not g3(nD, D);

not g4(nE, E);

// out[0] = ~A & ~B & ~C & ~D & ~E

and g5(out[0], nA, nB, nC, nD, nE);

// out[1] = ~A & ~B & ~C & ~D & E

and g6(out[1], nA, nB, nC, nD, E);

// out[2] = ~A & ~B & ~C & D & ~E

and g7(out[2], nA, nB, nC, D, nE);

// out[3] = ~A & ~B & ~C & D & E

and g8(out[3], nA, nB, nC, D, E);

// out[4] = ~A & ~B & C & ~D & ~E

and g9(out[4], nA, nB, C, nD, nE);

// out[5] = ~A & ~B & C & ~D & E

and g10(out[5], nA, nB, C, nD, E);

// out[6] = ~A & ~B & C & D & ~E

and g11(out[6], nA, nB, C, D, nE);

// out[7] = ~A & ~B & C & D & E

and g12(out[7], nA, nB, C, D, E);

// out[8] = ~A & B & ~C & ~D & ~E

and g13(out[8], nA, B, nC, nD, nE);

// out[9] = ~A & B & ~C & ~D & E

and g14(out[9], nA, B, nC, nD, E);

// out[10] = ~A & B & ~C & D & ~E

and g15(out[10], nA, B, nC, D, nE);

// out[11] = ~A & B & ~C & D & E

and g16(out[11], nA, B, nC, D, E);

// out[12] = ~A & B & C & ~D & ~E

and g17(out[12], nA, B, C, nD, nE);

// out[13] = ~A & B & C & ~D & E

and g18(out[13], nA, B, C, nD, E);

// out[14] = ~A & B & C & D & ~E

and g19(out[14], nA, B, C, D, nE);

// out[15] = ~A & B & C & D & E

and g20(out[15], nA, B, C, D, E);

// out[16] = A & ~B & ~C & ~D & ~E

and g21(out[16], A, nB, nC, nD, nE);

// out[17] = A & ~B & ~C & ~D & E

and g22(out[17], A, nB, nC, nD, E);

// out[18] = A & ~B & ~C & D & ~E

and g23(out[18], A, nB, nC, D, nE);

// out[19] = A & ~B & ~C & D & E

and g24(out[19], A, nB, nC, D, E);

endmodule

// D latch gate-level

module d\_latch (

input wire d,

input wire en,

output wire q

);

wire dbar, s, r, qbar;

not u1(dbar, d);

and u2(s, d, en);

and u3(r, dbar, en);

nor u4(q, r, qbar);

nor u5(qbar, s, q);

endmodule

// Master-Slave D flip-flop

module dff\_gate (

input wire clk,

input wire d,

input wire reset,

output wire q

);

wire nclk,nreset;

wire qm,d\_mux;

not u1(nclk, clk);

not u2(nreset, reset);

and(d\_mux,d,nreset);

d\_latch master (

.d(d\_mux),

.en(nclk),

.q(qm)

);

d\_latch slave (

.d(qm),

.en(clk),

.q(q)

);

endmodule

// 20-bit PIPO register

module pipo20 (

input wire clk,

input wire reset,

input wire [19:0] d\_in,

output wire [19:0] q\_out

);

genvar i;

generate

for (i = 0; i < 20; i = i + 1) begin : dff\_array

dff\_gate dff\_inst (

.clk(clk),

.reset(reset),

.d(d\_in[i]),

.q(q\_out[i])

);

end

endgenerate

endmodule

module encoder20x10 (in,out);

input [19:0] in;

output [9:0] out;

assign out[9:5] = 5'b0;

or(out[0], in[1] , in[3] , in[5] , in[7] , in[9] , in[11] , in[13] , in[15] , in[17] , in[19]);

or(out[1] , in[2] , in[3] , in[6] , in[7] , in[10] , in[11] , in[14] , in[15] , in[18] , in[19]);

or(out[2] , in[4] , in[5] , in[6] , in[7] , in[12] , in[13] , in[14] , in[15] , in[19]);

or(out[3] , in[8] , in[9] , in[10] , in[11] , in[12] , in[13] , in[14] , in[15]);

or(out[4] , in[16] , in[17] , in[18] , in[19]);

endmodule

module pipo10 (

input wire clk,

input wire reset,

input wire [9:0] d\_in,

output wire [9:0] q\_out

);

genvar i;

generate

for (i = 0; i < 10; i = i + 1) begin : dff\_array

dff\_gate dff\_inst (

.clk(clk),

.reset(reset),

.d(d\_in[i]),

.q(q\_out[i])

);

end

endgenerate

endmodule

module encoder10x5(in, out);

input [9:0] in;

output [4:0] out;

wire P0, P1, P2, P3, P4, P5, P6, P7, P8, P9;

// Intermediate wires for inverted inputs to simplify logic

wire not\_in\_0, not\_in\_1, not\_in\_2, not\_in\_3, not\_in\_4, not\_in\_5, not\_in\_6, not\_in\_7, not\_in\_8, not\_in\_9;

not (not\_in\_0, in[0]);

not (not\_in\_1, in[1]);

not (not\_in\_2, in[2]);

not (not\_in\_3, in[3]);

not (not\_in\_4, in[4]);

not (not\_in\_5, in[5]);

not (not\_in\_6, in[6]);

not (not\_in\_7, in[7]);

not (not\_in\_8, in[8]);

not (not\_in\_9, in[9]);

// Priority terms (P\_N)

assign P9 = in[9];

and (P8, in[8], not\_in\_9);

and (P7, in[7], not\_in\_8, not\_in\_9);

and (P6, in[6], not\_in\_7, not\_in\_8, not\_in\_9);

and (P5, in[5], not\_in\_6, not\_in\_7, not\_in\_8, not\_in\_9);

and (P4, in[4], not\_in\_5, not\_in\_6, not\_in\_7, not\_in\_8, not\_in\_9);

and (P3, in[3], not\_in\_4, not\_in\_5, not\_in\_6, not\_in\_7, not\_in\_8, not\_in\_9);

and (P2, in[2], not\_in\_3, not\_in\_4, not\_in\_5, not\_in\_6, not\_in\_7, not\_in\_8, not\_in\_9);

and (P1, in[1], not\_in\_2, not\_in\_3, not\_in\_4, not\_in\_5, not\_in\_6, not\_in\_7, not\_in\_8, not\_in\_9);

and (P0, in[0], not\_in\_1, not\_in\_2, not\_in\_3, not\_in\_4, not\_in\_5, not\_in\_6, not\_in\_7, not\_in\_8, not\_in\_9);

// Output bit 0 (LSB)

or (out[0], P1, P3, P5, P7, P9);

// Output bit 1

or (out[1], P2, P3, P6, P7);

// Output bit 2

or (out[2], P4, P5, P6, P7);

// So out[3] is 1 for P8 or P9

or (out[3], P8, P9);

assign out[4] = 1'b0;

endmodule

**Test Bench:-**

`timescale 1ns / 1ps

module golden\_design\_tb;

reg refclk,clk2,reset;

reg [4:0] data\_in;

wire [4:0] data\_out;

golden\_design dut(.refclk(refclk),.reset(reset),.clk2(clk2),.data\_in(data\_in),.data\_out(data\_out));

initial begin

refclk=1;

forever #5 refclk = ~refclk;

end

initial begin

clk2=1;

forever #5 clk2 = ~clk2;

end

initial begin

reset=1'b1;

#10 reset=1'b0;

data\_in = 5'b00001;

#10; data\_in= 5'b00100;

#10; data\_in = 5'b00011;

#10; data\_in = 5'b00111;

#10; data\_in = 5'b11111;

#40;

$finish;

end

initial

$monitor("data\_in = %b and data\_out = %b ", data\_in,data\_out);

endmodule

**28/06/25**

****Theoritical Knowledge:****

**Adding some black box to the previous design and creating inbuilt E5 violation in the design.**

**03/07/25**

****Theoritical knowledge:****

****Clocks:** (2)---Refclk from top level to Pll , Clk2 from top level to 10 pipo registers.**

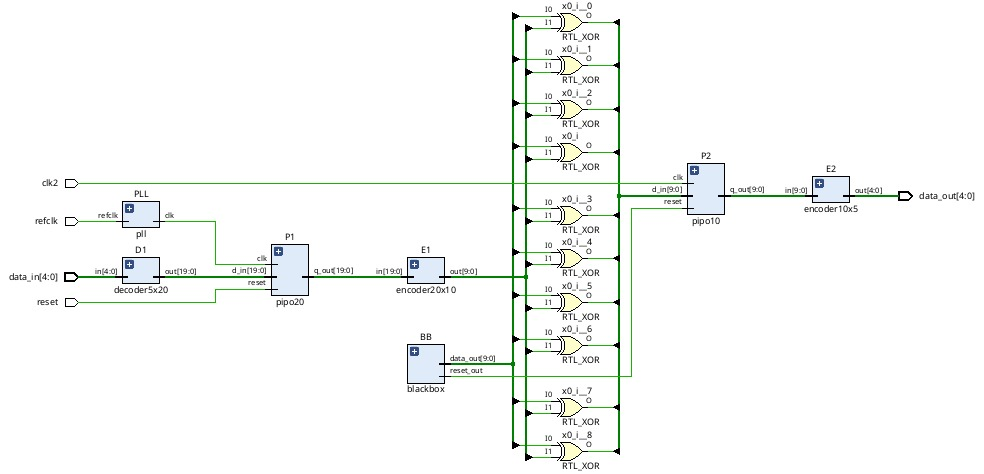
****Internally Generated clocks:**(1)---Clk1 from PLL to 20 pipo registers.**

****Resets:**(1) Reset from top level**

****Internally Generated resets:** (1)--- reset from Black box to 10 pipo registers/**

****Black boxes:**(2) PLL and empty Black box**

**Hands On Work:**

****

**06/07/25**

****Theoritical knowledge:****

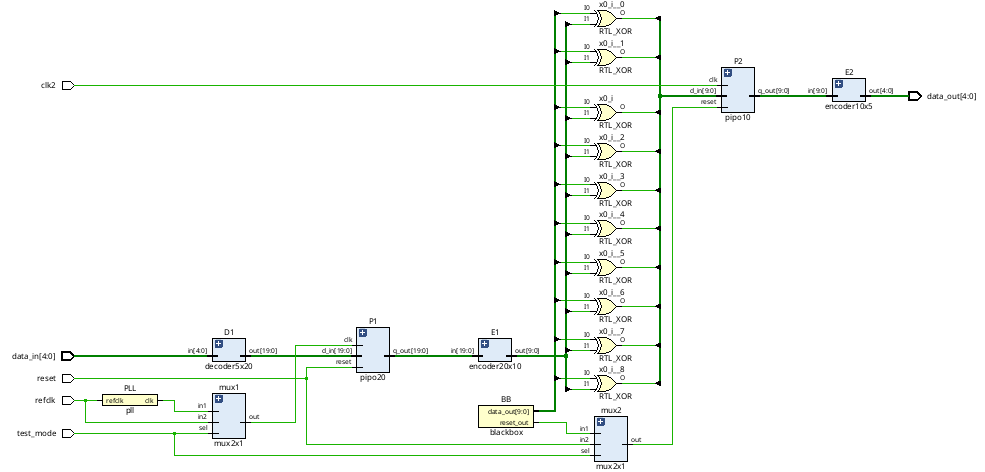
**During the verification of the design, DRC violations such as C6 and C9 were identified.**

To address these problems, **multiplexers (MUXes)** were added at key points in the design. These MUXes help to:

* **Switch signals correctly** between normal operation and test mode.
* **Avoid signal conflicts** that happen when signals come from different clock sources.
* **Ensure proper signal flow** and improve the overall reliability of the circuit.

By adding the MUXes, the design now passes the DRC checks, and the circuit works correctly in both functional and test environments.

**Hands On Work:**



****09/07/25****

****Theoritical knowledge:****

**To avoid the E5 violation in the design, a multiplexer (MUX) is used to select between two input sources, based on the** test\_mode **signal:**

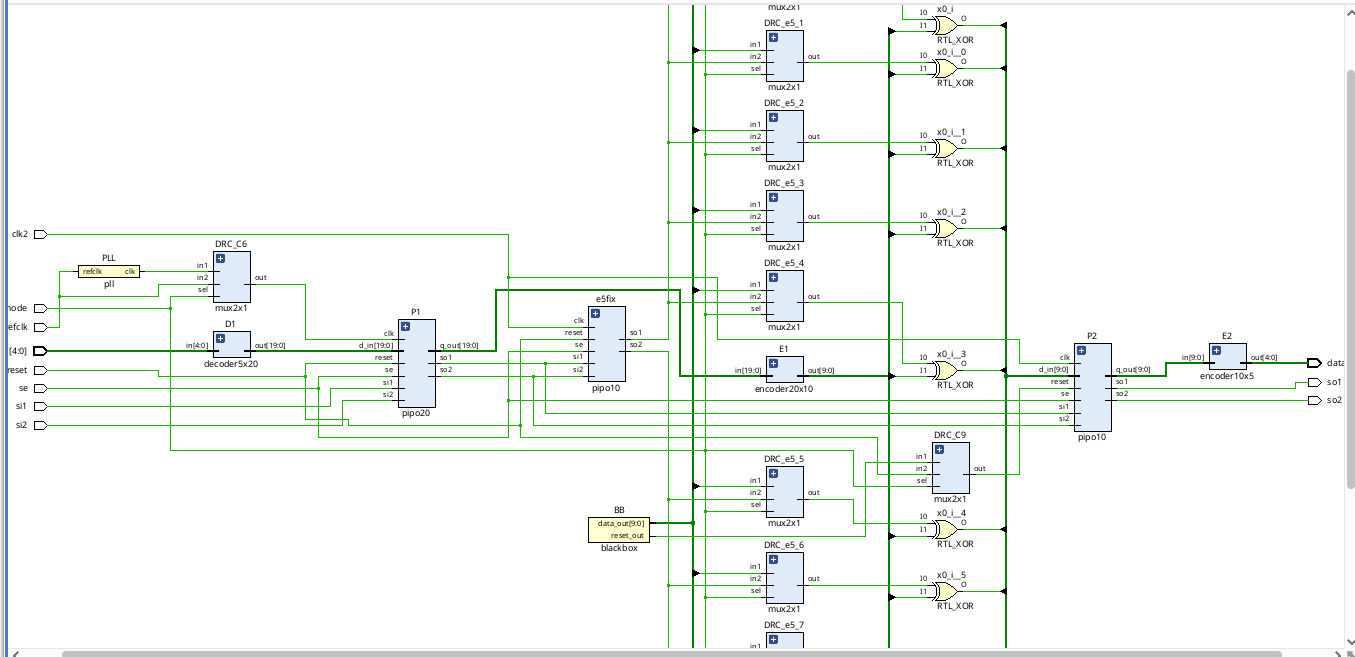
* When test\_mode = 0, the input is taken from the **blackbox output** (normal functional path).
* When test\_mode = 1, the input should ideally come from a **test vector**.

However, directly supplying the test vector from the **top-level port** can reduce **fault coverage**, especially for **stuck-at faults**. Additionally, **hard-wiring the input** to a constant 0 or 1 also limits the coverage, as it does not allow for full testing of all possible fault conditions.

To overcome these limitations, we use **Dynamic Test Points (DTPs)**. These DTPs receive their test input from a **PIPO (Parallel-In Parallel-Out) register**, which in turn gets its data from **previous scan flip-flops** in the scan chain. This allows dynamic and controllable test vector injection during scan operations, ensuring:

* Higher test coverage,
* Improved detection of stuck-at faults,
* Better support for Automatic Test Pattern Generation (ATPG).

**Hands On Work:**

****

****EDA LINK:**** [**https://www.edaplayground.com/x/smn8**](https://www.edaplayground.com/x/smn8)

**Executive Summary**

The project presented here deals with the complete cycle of a digital circuit design, emphasizing not just functionality but also compliance with **Design Rule Checks (DRC)**. The work spans across essential aspects of VLSI such as **clock design**, **integration of additional modules (black boxes and PLLs)**, **design-for-testability using scan chains**, and above all, **violation analysis and correction**. A particular focus was placed on understanding and resolving **E5 violations**, a common but critical issue in physical design. The project reflects the workflow followed in real semiconductor industries, ensuring that the design is both **practical and sign-off ready**.

**Introduction**

## ****Project Context****

The continuous miniaturization of semiconductor devices has made digital design highly sophisticated. Every stage, from the way clocks are managed to how designs are tested, requires precision. **Clocking strategies, rule adherence, error detection, and correction** are no longer optional but mandatory. In this project, a complete design cycle was recreated in a controlled environment, bringing together **theoretical knowledge and industrial practices**.

**Project Objectives**

## Motivation and Objectives

The motivation was to simulate the conditions of an actual VLSI development process, where issues such as clock synchronization, **IP (black box) integration**, and **design rule compliance** frequently arise. The objectives were:

* Develop and modify a digital design with additional modules.
* Apply clock rules for stable timing.
* Insert scan chains for enhancing testability.
* Detect and correct violations, with emphasis on **E5 errors**.
* **Deliver a final design that passes verification and DRC checks.**

**Methodology and Results**

# **Methodology**

Instead of a straightforward step-by-step list, the methodology was carried out as **an iterative cycle of design, modification, verification, and correction**.

1. **Base Design** – A simple RTL module served as the starting point.
2. **Clock Handling** – Rules related to skew, latency, and duty cycle were applied. PLL modules were integrated to refine clock distribution.
3. **Design Expansion** – Black boxes were inserted to represent third-party IPs, ensuring the project simulated a real SoC environment.
4. **Synthesis & Implementation** – The RTL was converted into a physical netlist and placed/routed.
5. **Violation Analysis** – Errors were checked; DRC and specifically **E5 violations** were flagged.
6. **Correction Cycle** – Debugging was carried out through multiple iterations until violations were fully cleared.
7. **DFT Enhancement** – Scan chains were introduced and tested with shifting patterns to validate the structure.
8. **Verification** – The final design was validated both functionally and for rule compliance.

# Tools and Environment

* **Xilinx Vivado**: Used for design entry, synthesis, implementation, and simulation.
* **Version Control**: All iterations and updates were tracked in a daily log and managed through GitHub.
* **Scripting**: TCL/Python scripts were used wherever automation or report handling was required.

**Project GitHub Link:**

**Learning and Reflection**

# ****Key Learnings****

### Technical

* Clearer understanding of **clocking concepts** (skew, jitter, latency).
* Practical knowledge of **PLL integration** and **black box handling**.
* Experience in violation detection and systematic correction.
* Implementation of **scan insertion** as a testability measure.
* Improved use of automation scripts for handling repetitive tasks.

### Project/Management

* Learned to plan iterations effectively and deal with errors progressively.
* Improved coordination and documentation practices through GitHub.
* Strengthened problem-solving mindset while facing rule violations under tight design constraints.

**Conclusion and Future Scope**

# **Conclusion**

This project was a hands-on experience that bridged classroom knowledge with **industry-relevant practices**. The major achievements include:

* A fully **verified and sign-off ready design**.
* Successful **PLL integration and black box insertion**.
* Complete **elimination of DRC and E5 violations**.
* Functional **scan chain implementation and validation**.

By simulating the **real challenges of VLSI physical design**, the project not only met its stated objectives but also built a strong foundation for future explorations.

# **Future Scope**

* Extending the design to include **multi-clock domains**.
* Experimenting with **low-power design strategies**.
* Automating larger portions of **violation fixing** through advanced scripting.
* Expanding the methodology to larger SoC-level designs with multiple IPs.