Scan Insertion With DRC & Coverage Analysis

# **The domain of the Project:**

# 

# **DFT(Design For Testability)**

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

# **Team Members:**

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

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

**DFT engineer—Struent semiconductors pvt ltd**

**Prof. Radhakumari**

**Executive Director & Founder**

**SURE Trust**

****Detailed Report :****

****Table :****

|  |  |  |
| --- | --- | --- |
| DATE | MINUTES OF MEETING | ATTENDEES |
| 26/05/25 | 1. Assigned a task to create document for a project report, which consists of the following three sections: Minutes of the meeting, Theoretical knowledge, Hands-on work. 2. Task: Revise the DFT module class notes. | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 28/05/25 | Go through these topics in the document:   1. Design Rule Overview 2. Scan Insertion Checking 3. How to troubleshoot Rules Violations 4. Clock Rules, Clock Terminology | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 03/06/25 | 1. Introduction to Clock Rules 2. Assigned the task to complete the C1,C2,C3,C6 Clock rules | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 07/06/25 | 1. Discussed the Clock rules of C1, C2 detaily | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 16/06/25 | 1. Reviwed the individual project reports. 2. Assigned task to implement the circuit and simualte. | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 28/07/25 | Modification of the design by adding Blackbox to the design. | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 03/07/25 | Correction of the errors and adding PLL and some other logics to introduce violations into the design. | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 06/07/25 | Fixing the DRC Violations in the design. | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |
| 09/07/25 | Fixing the E5 Violation and Scan Insertion in the design. | P. Pavan Kumar P. Vijay Kumar K. Tharun Krishna K. Chandu |

**26/05/25**

**Theoretical Knowledge :**

**Design for Testability (DFT)** in **VLSI** primarily focuses on incorporating features like **scan chains** to make ost-manufacturing testing of integrated circuits efficient. Scan chains connect flip-flops in the design into a shift register, improving **controllability** and **observability** of internal nodes. This allows engineers to apply and observe test patterns easily, making defect detection (such as stuck-at faults) faster and reducing test cost. By using scan chains, DFT ensures higher product quality and reliability in VLSI design.

**Scan Chain Process :**

**Step 1**: Replace regular DFFs with Scan DFFs.

**Step 2**: Connect Scan DFFs into a scan chain.

**Step 3**: Control the chain with the Scan Enable (SE) signal.

**Step 4**: Shift in test data using the Scan-In (SI) pin.

**Step 5**: Capture and shift out the circuit’s response.

To **Avoid “Reset interference**” during scan chain operation, extra circuitry such as **multiplexers**, **control logic**, or **scan-bypass circuits** can be used. These mechanisms isolate or disable the reset signal during scan mode (SE = 1), ensuring that the scan operation proceeds without interruption, while allowing normal reset behaviour during functional mode (SE = 0). This ensures that the reset does not affect the scan chain during testing but operates correctly in functional mode.

**28/05/25**

**Theoretical Knowledge :**

**Scan Insertion Checking:**  
Scan insertion checking is a key step in the DFT flow, ensuring the design is scannable and ready for ATPG. It involves performing design rule checks (DRC) to validate scan chain integrity, scan cell controllability and observability, and correct clock and scan enable setup. These checks cover scan cell connections, constant data inputs, and proper handling of gated clocks and resets. The checks are grouped into scannability rules (S-Rules) and trace rules (T-Rules), each targeting specific scan and data path aspects. Errors such as broken scan chains, unconstrained inputs, or incorrect scan enable logic must be corrected before ATPG. Warnings and notes may highlight areas for optimization. Passing scan insertion checks confirms the design is test-ready, enabling effective fault detection and diagnosis in silicon.

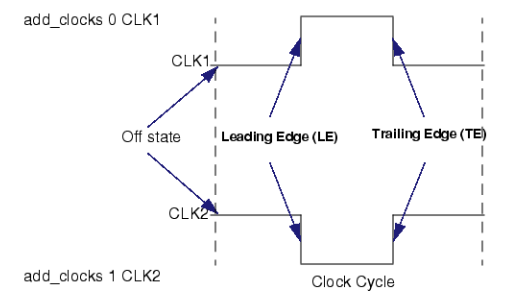
Troubleshooting Design Rule Violations:  
When design rule violations are detected during the DFT flow, it is essential to systematically analyze and resolve them. Tools like Tessent’s report\_drc\_rules and analyze\_drc\_violation help identify and categorize violations, including scan connectivity, clock control, data path issues, and other constraints. Visualization tools such as Tessent Visualizer provide graphical representations of problematic scan cells, clocks, and paths. To troubleshoot, designers trace the root causes, check signal definitions, correct logic or constraints, and ensure proper scan chain connectivity. After resolving issues in the RTL or design constraints, rerunning the DRC checks verifies the fixes. This iterative process ensures the design is fully testable and ready for ATPG and silicon validation.

Clock Rules (C-Rules):  
Clock rules check scan clocks to ensure correct definition and operation. They verify clock sources, connections, and control signals to avoid scan data issues. Violations can be flagged as error, warning, note, or ignored, helping prioritize fixes for robust test coverage.

* Error: Critical issue that must be fixed before continuing.
* Warning: Significant issue, but allows continuing with caution.
* Note: Informational message; not critical.
* Ignore: Skip the rule check altogether.

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:

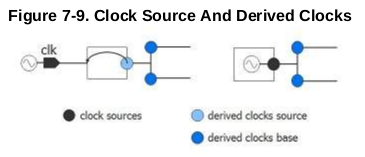
* Leading Edge: The transition of the clock from the off state to the on state is considered the leading edge of the clock.
* Falling Edge: The transition from the on state to the off state is considered the trailing edge of the clock.



Clock Cones and Effect Cones:  
A gate pin or output pin is considered to be in the clock cone of a clock signal when they are connected through combinational logic gates and transparent latches (TLAs) to the clock primary input.

**Clock Source and Derived Clocks :**

Clock rule checks in Scan Cell Data Rules (D rules) using DFT\_C5 (Memory BIST clock tracing) involve defining three types of clocks at primary ports or instance pins: Synchronous clocks, asynchronous clocks, Derived clock source and Derived clock branches.



Clock Source (Synchronous or Asynchronous):  
This is where the clock signal originates, either from an external clock generator or an internal clock generator in the design. It is usually a well-defined pin or instance pin that provides the clock pulse to the rest of the design.

Derived Clock Source:  
This is a clock that depends on another clock but is routed through a specific path. Sometimes, this path might be blocked, so the derived clock needs a reference to another clock to trace the path properly. For example, in designs with a PLL (Phase-Locked Loop) block, you can use the reference option to link the derived clock back to the original clock source.

Derived Clock Branch:  
This occurs when different branches of the clock tree, even if sourced from the same clock, can become unbalanced and unsynchronized. You define these branches using the add\_clocks -branch option. A derived clock branch must always be driven by a valid “clock source” or “derived clock source.” These branches can also be adjusted in later network operations.

**03/06/25**

**Theoretical Knowledge :**

**C-Rules :**

Clock Rule C1 – No Clock Activity When Clocks Are OFF  
**Default Handling**: Error

**report\_drc\_rules**: Supported

### **Objective**

Clock Rule C1 ensures that when all clocks in the design are turned OFF, there is no unintended clocking activity. This includes not only the clock signals but also set and reset inputs connected to scan and non-scan flip-flops.

The rule validates that these inputs remain stable and inactive, preventing any unexpected transitions or toggling that could compromise scan operations.

### **What It Verifies**

* When clocks are OFF, the clock, set, and reset inputs of all flip-flops must not be active or undefined.
* For non-scan flip-flops, if such a condition occurs, it is considered tied to an unknown state (TIEX), acknowledging its indeterminate nature.
* For scan flip-flops, the clock input must resolve to a stable binary value—either 0 or 1. If it is undefined (X) or toggles, it is treated as a rule violation.

### **Violation Scenario** **:**

A clock, set, or reset signal remains active or unknown (X) while clocks are expected to be OFF.

There is toggling or instability on these inputs during the clock-OFF state.

**Example :** If a clock signal is driven by a gating enable (clk = EN) and the enable signal becomes undefined (X), the resulting clock becomes X. This leads to a scenario where the flip-flop’s behavior is unpredictable, hence violating the C1 rule.

### **Consequences of Violation**

* Causes unpredictable data to be captured in scan flip-flops.
* Affects scan chain load/unload reliability.
* Can result in test coverage loss and incorrect diagnosis.
* May lead to functional corruption during scan operations.

### **How to Resolve**

* Ensure all clock, set, and reset signals are properly defined and stable during clock-OFF states.
* Fix any gating logic or tie-off points that might lead to undefined (X) values.
* Make sure asynchronous set/reset signals are included in the clock control logic where appropriate

## **Clock Rule C2 – Clock Reachability Check :**

## **Severity**: Warning (by default) report\_drc\_rules: Supported

### **Objective**

The C2 rule checks whether a defined clock signal actually reaches at least one memory element, such as a flip-flop or RAM. This ensures that the clock is structurally valid and contributes to scan or functional behavior during testing.

### **What It Verifies**

* Each clock input defined in the design must have a valid structural path to at least one of the following:
  + Clock port of a memory element (like a flip-flop)
  + SET or RESET ports of memory elements
  + Read/write enable pins of RAMs
* The check is done via backward tracing, starting from the memory elements and tracing back toward the clock primary inputs.

### **Violation Conditions**

A violation occurs if:

* A clock input pin cannot be structurally traced to any memory element in the design.
* The clock is declared but does not participate in actual clocking of any relevant element.

Such a situation typically arises when:

* A clock pin is incorrectly defined as a clock.
* A floating pin is marked as a clock but is unused or disconnected.

### **Consequences of Violation**

* A defined clock that does not reach any element cannot be used for scan or capture, reducing test coverage.
* These clocks may introduce confusion in DFT analysis, affecting pattern generation and validation.

### **Exception Handling**

To avoid false violations:

* If internal clocks are defined in the design (e.g., generated from a PLL), and an external reference clock is disconnected in the flat model, Tessent suppresses C2 violations for those floating pins.
* This exception prevents incorrect warnings when PLL-based clocks are not fully modeled.

### **Example Scenario**

* A signal named CK17 appears to be a clock due to its name.
* However, it does not connect to any memory element in the design.
* This leads to a C2 rule violation.

To fix the issue in such a case, the invalid clock definition should be removed to avoid misleading the tool.

## **Clock Rule C3 – Data Race Between Source and Sink on Same Clock :** **Severity**: Note (by default) report\_drc\_rules: Supported

## **Objective :**

Clock Rule C3 identifies potential data race conditions between two memory elements (flip-flops or RAMs) that are driven by the same clock or opposite edges of the same clock. It ensures that a sink flip-flop does not inadvertently capture a new value from the source flip-flop in the same cycle, which violates the expected scan behavior where old values should be captured.

What It Verifies :

This rule checks whether a sink element is capturing data from a source element during the same clock cycle, instead of the next. The expected behavior is that the sink captures the previous (old) value, not the new value that just arrived. The rule verifies clock and data path relationships to ensure the timing supports this assumption.

### **Violation Conditions :**

Both source and sink flip-flops are clocked by the same clock or opposite edges of the same clock (e.g., one with an inverter). Data from the source element propagates too quickly and is captured by the sink element in the same cycle.

This can happen when:

* Clock domains are not properly separated.
* The clock tree is shared or not buffered appropriately.
* Clock edges are closely spaced (leading/trailing)

### **Examples of Violation Scenarios**

| Source Type | Sink Type |
| --- | --- |
| Level-sensitive (LS) | Level-sensitive (LS) |
| Level-sensitive (LS) | Trailing edge triggered (TE) |
| Leading edge triggered (LE) | Level-sensitive (LS) |
| Leading edge triggered (LE) | Trailing edge triggered (TE) |

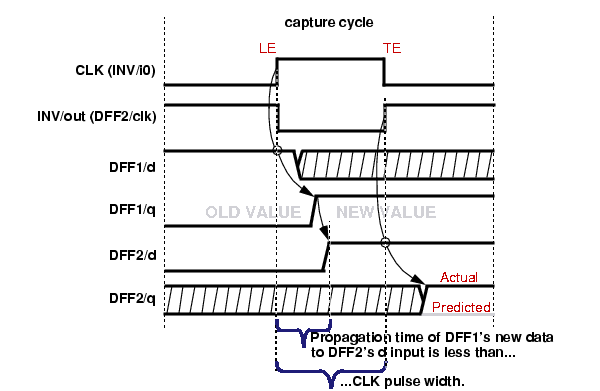
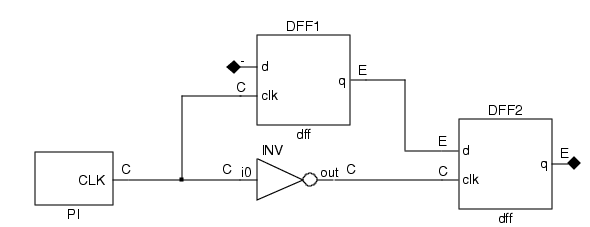
Such configurations can lead to races where the sink captures data that just passed through the source, violating the intended scan behavior.

### **Consequences of Violation**

* Causes race conditions that lead to unpredictable data capture.
* May result in simulation mismatches when comparing expected and actual scan data.
* Can reduce the reliability of scan patterns and potentially lead to test failures.

Example : Consider two flip-flops:

* **DFF1**: Triggered by the leading edge of a clock.
* **DFF2**: Triggered by the trailing edge of the same clock (via an inverter).

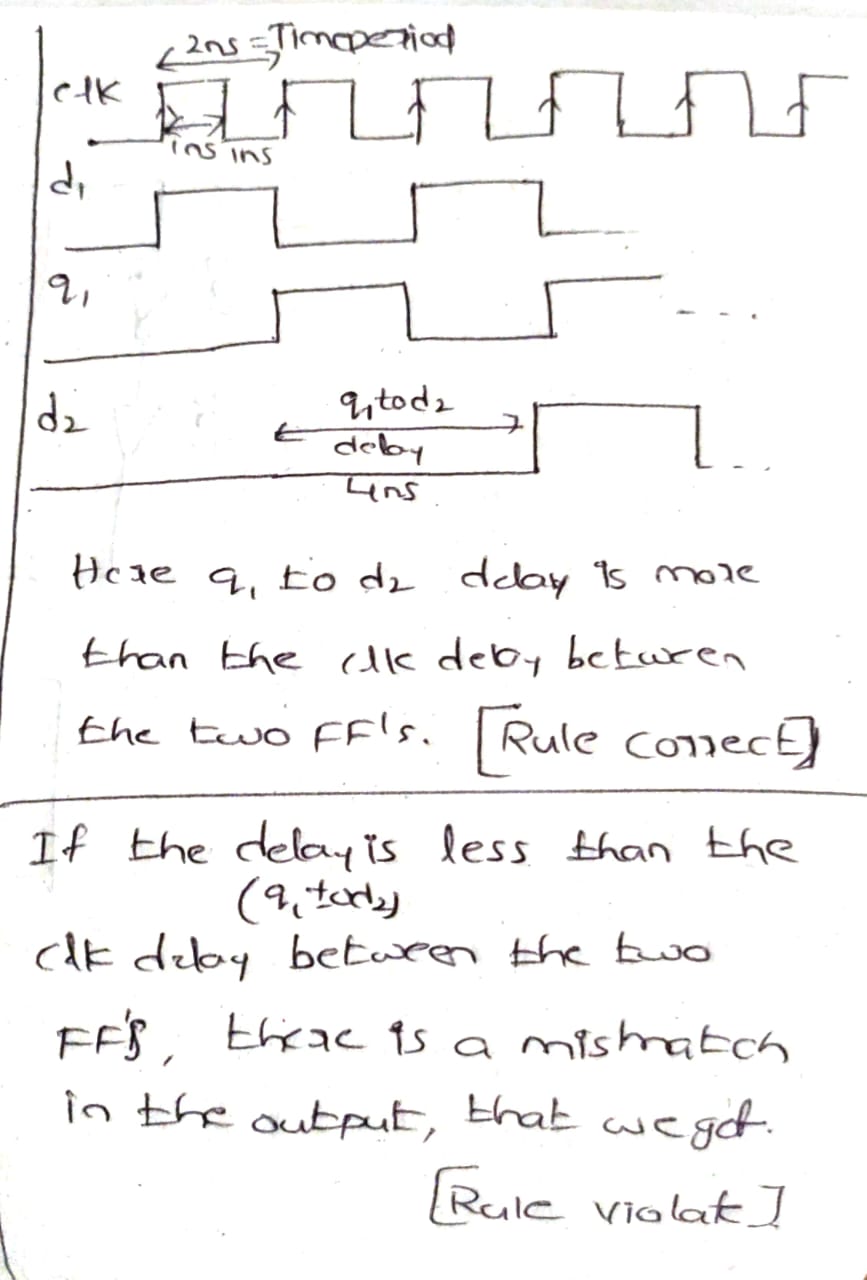
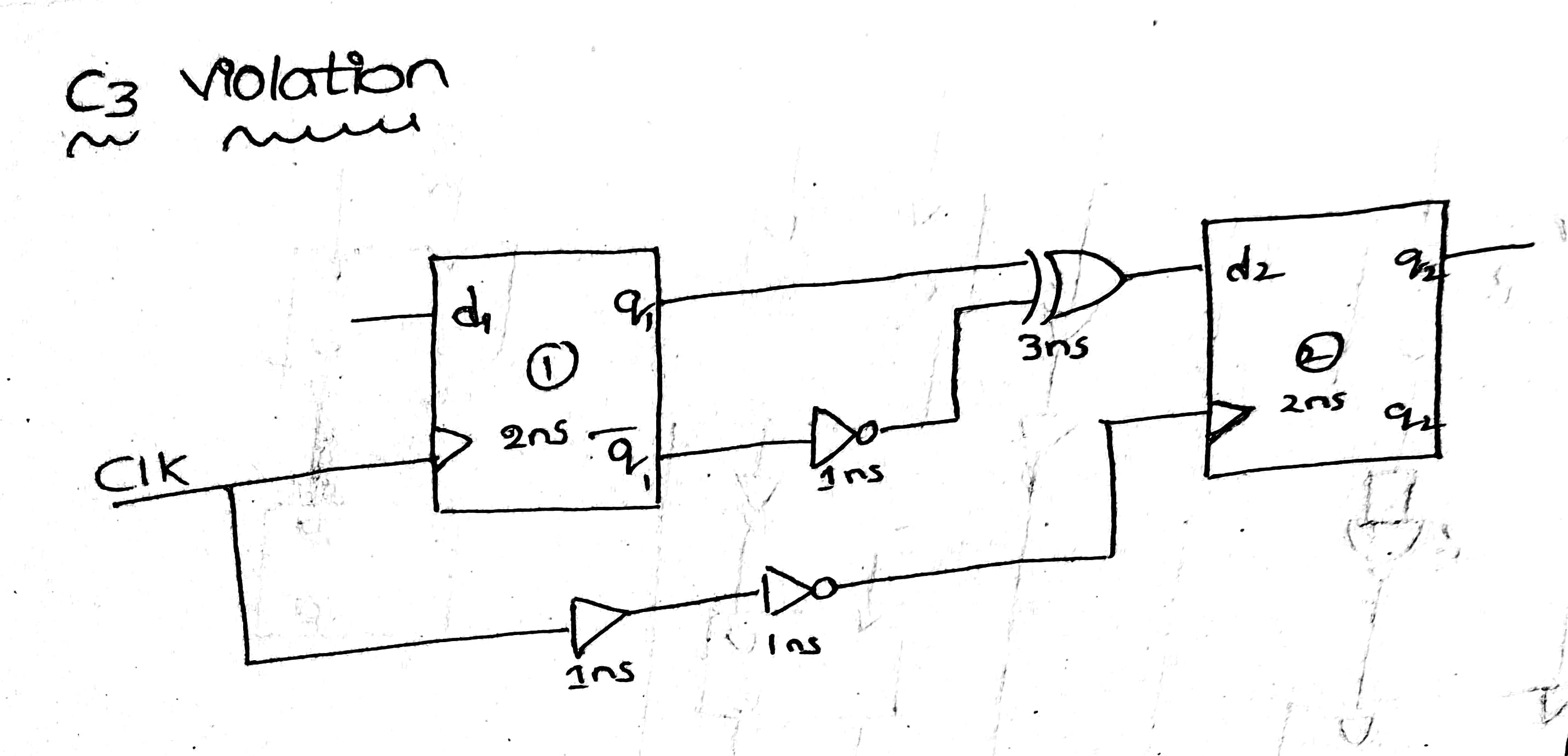


If the data path between DFF1 and DFF2 is fast, DFF2 might capture the new data immediately, instead of the old value. This is a classic C3 violation scenario.

### **Detection Logic**

The tool performs the following checks:

* Builds a clock cone: All sequential elements affected by a clock.
* Builds an effect cone: All logic driven by the data outputs of those sequential elements.
* Flags violations when a sink flip-flop is within the clock cone and its data input comes from the effect cone of the same clock.



## **Clock Rule C6 – Clock Must Not Affect Data It Is Capturing :**

Default Handling: Warning

report\_drc\_rules: Supported

**Objective**

A clock must not affect data that it is capturing. If it does, a race condition may result that produces inaccurate simulation results.

The clock’s job is to trigger the capture, not to control or modify the data it captures at the same time. This rule helps avoid race conditions and unpredictable scan behavior during testing.

### **What It Verifies**

* It checks whether a flip-flop's clock input and data input are both influenced by the same clock signal.
* If the clock also affects the data it is capturing, the captured value may become unpredictable.

### **Violation Conditions**

A C6 violation occurs when:

* A scan cell’s clock and data inputs are driven by logic that comes from the same clock.
* This creates a situation where the clock might change the data just as it's being captured.

### **Consequences of Violation**

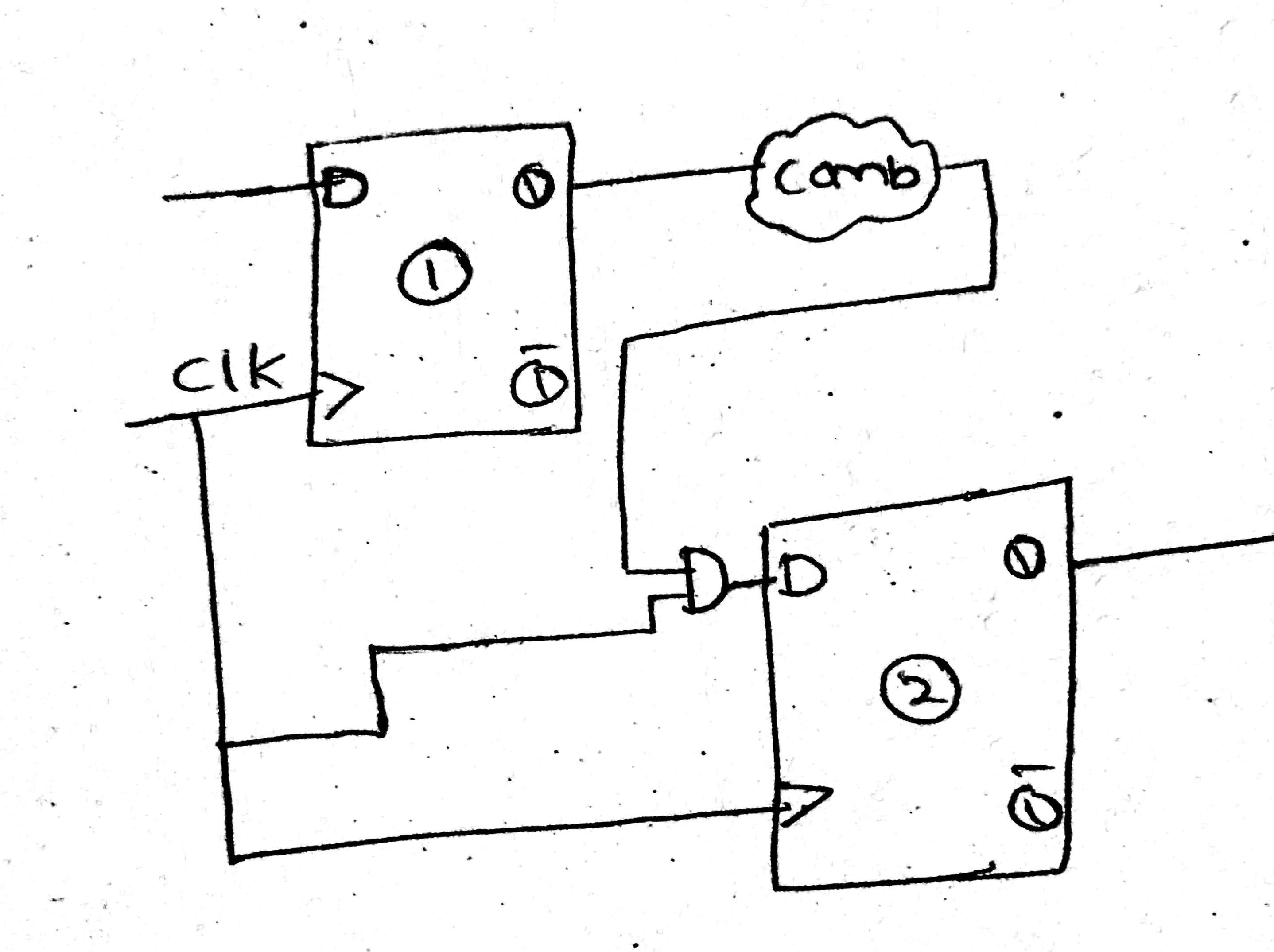
* Can lead to race conditions where data is not reliably captured.
* Makes simulation results unreliable or incorrect.
* May cause test patterns to fail or detect the wrong behavior.

### **Example**

* A flip-flop has both its clock input and data input connected to the same signal (CLK).
* When the clock goes high, it also changes the data input—leading to confusion about what value the flip-flop should capture.
* This is a classic C6 violation

## 

**C6 Violoation Circuit :**



**10/06/25**

**Theoretical Knowledge :**

**Design Rule Checks:**

Scan Cell Data Rule D5 – All Memory Elements Must Be Scannable

**Default Handling**: Warning

**report\_drc\_rules**: Supported

Objective

To ensure that all flip-flops and latches in the design are either included as part of the scan chain or are properly modeled for predictable behavior during scan testing.  
This rule helps prevent non-scan memory elements from causing incorrect values, X-states, or loss of test coverage during scan capture and simulation.

### **What It Verifies**

* After scan memory elements are identified, D5 checks if any remaining flip-flops or latches are not scannable.
* It simulates the circuit with a sequential depth of 0 or 1, treating it like a combinational circuit to detect memory elements left outside the scan path.
* Each non-scan memory element is then classified based on its behavior at the beginning of the first capture cycle.

### **Violation Conditions**

A D5 violation occurs when:

* A memory element is not identified as part of a scan cell.
* These unscanned memory elements are treated as unknown logic drivers, potentially introducing X values or invalid states.

Such memory elements are classified into one of the following behavioral models:

These classifications help the tool simulate how these elements might influence logic paths during test.

### **Consequences of Violation**

* Non-scan memory elements introduce uncontrollable or unobservable behavior.
* They may corrupt values during scan capture, lead to simulation mismatches, and reduce fault coverage.
* Although the rule is treated as a warning by default, it can impact the quality and reliability of scan patterns.

### **Example Scenario**

* A latch or flip-flop not included in the scan chain remains functional during scan operation.
* During pattern simulation, the tool identifies that its state is not predictable and models it as TIE-X, INIT-X, or similar.
* These unknown values can propagate and affect scan outputs or captured responses, potentially hiding real faults

|  |  |
| --- | --- |
| Model Type | Description |
| INIT-0 | At logic 0 at start of capture |
| INIT-1 | At logic 1 at start of capture |
| INIT-X | Unknown at start and may change |
| TIE-0 | Always 0 during capture |
| TIE-1 | Always 1 during capture |
| TIE-X | Always X during capture |
| TLA | Transparent latch when clock is off |

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

### **Rule ID**: E5

Category: Extra Rules (E-Rules)

Objective

To detect conditions where X states (unknown logic values) may become observable at primary outputs or scan observation points during scan test operations.  
E5 is especially important in compressed test environments (such as BIST or EDT), where uncontrolled X states reduce compression efficiency, increase pattern count, and affect test reliability.

### **What It Verifies**

* The rule checks whether certain gates in the design could output an X state under constrained simulation, and if that X is visible at an observation point.
* The design is simulated with:
  + Binary values applied to primary inputs (PIs) and scan cells
  + Constrained values applied to control pins
* Only gates that produce Xs and have a sensitizable path to outputs are considered.

### **Violation Conditions**

An E5 violation occurs if any of the following happens:

* WIRE gates: Inputs receive different values and the resolution logic causes an X.
* BUS gates:
  + Multiple tri-state drivers are ON simultaneously
  + All drivers are OFF simultaneously, producing a Z (interpreted as X)
* Tri-State Driver (TSD) / Switch (SW) gates: Not connected to a bus and output Z, which behaves as X.
* TIE-X gates: Sensitizable up to a visible observation point.
* Transparent latches (TLA): Clock not properly set ON or set/reset lines not disabled.
* ROM/RAM elements:
  + Read line is OFF and read\_off value is X
  + Uninitialized memory drives output
* Primary Inputs (PI): Z drives output and is treated as X unless NO\_Z constraint is applied.

### **Consequences of Violation**

* Unmasked X states can:
  + Corrupt scan test results
  + Lower fault coverage
  + Increase number of patterns in compressed ATPG
* May degrade pattern quality and test efficiency, especially in compression-based testing like EDT or TestKompress.

### **Example Scenario**

* A tri-state bus with multiple drivers accidentally enabled (or none enabled) causes a bus value of X.
* If that X propagates through logic and reaches an output, it causes an E5 violation.
* In compressed test environments, this X can’t be masked effectively, so more patterns are needed, reducing compression benefits

**E5 Violation Circuit :**

## 

## **T3– Scan Chain Cannot Be Traced from Output to Input**

Category: Scan Chain Trace Rules  
Severity: Error

### **Objective**

To ensure that each scan chain output (scan\_out) is connected back to a valid scan chain input (scan\_in) through a sensitizable scan path.  
This tracing is crucial for:

* Verifying structural correctness of scan chains
* Ensuring scan data can shift through every cell in both directions
* Allowing tools to analyze, reorder, or validate scan chains

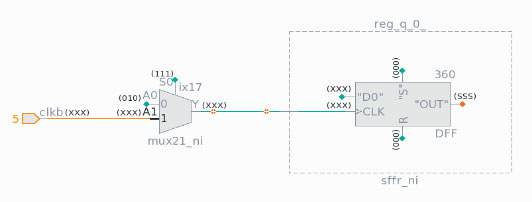
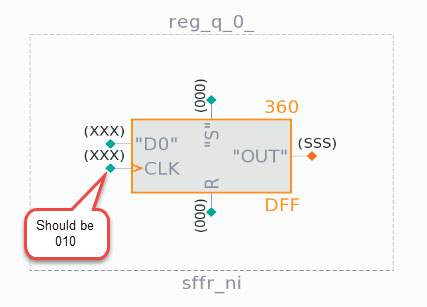
### **What It Verifies**

* The tool attempts to trace the entire scan shift path, starting from the scan-out pin backward through the chain to the scan-in pin.
* It checks for:
  + Continuous scan connectivity
  + Properly enabled scan multiplexers
  + Valid clocking and control paths
  + Scan\_enable and scan\_mode values that allow data to propagate

### **Violation Conditions**

A trace rule violation occurs when:

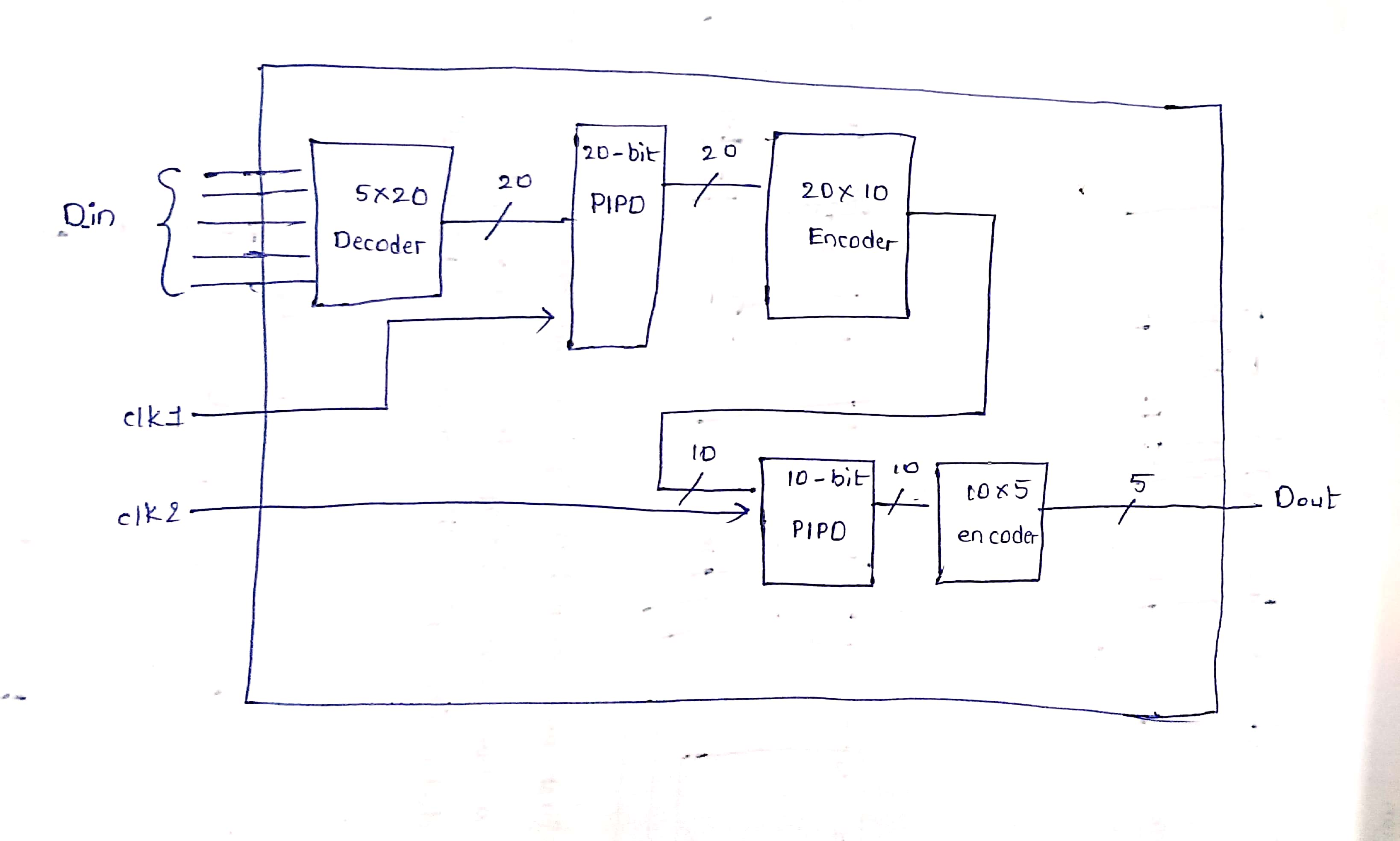
* The tool cannot complete a backward trace from scan\_out to scan\_in
* The path is logically blocked or disconnected due to:
  + Improper scan mux control
  + Missing scan cell link
  + Floating scan cell input/output
  + Broken net in the netlist
  + Unsensitizable logic path due to disabled enable signals



**16/06/25**

**Hands on work :**

Implementation of below circuit.



**GOLDEN DFT DESIGN :**

// Code your design here

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]);

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

**GOLDEN DFT TESTBENCH :**

module 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);

initial begin

$dumpfile("test.vcd");

$dumpvars(0,tb);

end

endmodule

**28/06/25**

**Theoritical Knowledge:**

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

Verify the design.

**03/07/25**

**Theoritical knowledge:**

* Creating the pll blackbox and some other logic.

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

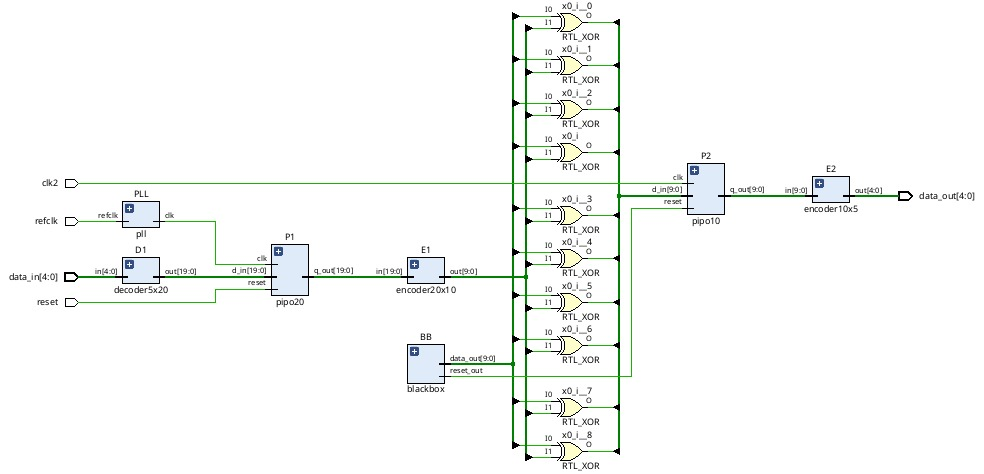
**Internally Generated clocks:**(1)---Clk1 from PLL to 20 pipo registers.

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

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

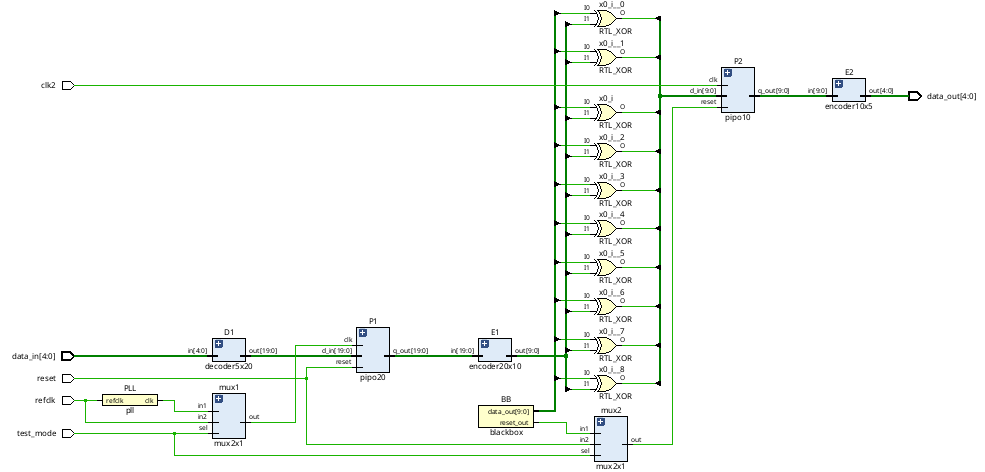
**Black boxes:**(2) PLL and empty Black box.

**Design Schematic :**

**06/07/25**

**Theoritical knowledge :**

Solving the DRC’s ocurred in the design like c6,c9 by adding some mux.

****

**09/07/25**

**Theoritical knowledge :**

To resolve an E5 violation caused by a black box, we leverage the concept of configuring test points. A common approach involves using a multiplexer with two inputs controlled by a testmode signal. When testmode is 0, the input is taken from the black box; when testmode is 1, the input should come from a test vector. However, directly supplying the test vector from the top level can degrade test coverage. Similarly, tying the input to a constant 0 or 1 may lead to loss of coverage, as it prevents detection of all stuck-at faults.

To address this, dynamic test points are used. These receive test vectors from a PIPO (parallel-in parallel-out) register, which in turn gets its input from preceding scan flip-flops. This setup ensures better fault coverage without compromising test access. Additionally, scan chains are incorporated into the design to support this configuration.

**UPDATED DFT DESIGN :**

`timescale 1ns / 1ps

module Updated\_DFT(REFCLK,CLK2,DATA\_IN,TESTMODE,SI1,SI2,SE,SO1,SO2,RESET,DATAOUT);

input REFCLK,CLK2;

input SI1,SI2,SE;

output SO1,SO2;

input [4:0]DATA\_IN;

input RESET,TESTMODE;

output [4:0]DATAOUT;

wire [19:0] deco1,pipo1;

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

wire [9:0] x,e5\_mux\_out;

wire m1,m2,RESET\_out;

wire SO1wire,SO2wire,SO11\_wire,SO22\_wire;

decoder5x20 D1(DATA\_IN,deco1);

pll PLL(REFCLK,clk1);

mux2x1 DRC\_C6(clk1,REFCLK,TESTMODE,m1);

pipo20 P1 (m1,RESET,SE,SI1,SI2,deco1,pipo1,SO1wire,SO2wire);

encoder20x10 E1(pipo1,enco1);

mux2x1 DRC\_C9(RESET\_out,RESET,TESTMODE,m2);

pipo10 P2 (CLK2,m2,SE,SO1wire,SO2wire,x,pipo2,SO1,SO2);

encoder10x5 E2(pipo2,DATAOUT);

blackbox BB(RESET\_out,blackbox\_out);

pipo10 e5fix(CLK2,RESET,SE,SO1wire,SO2wire,,,SO11\_wire,SO22\_wire);

mux2x1 DRC\_e5\_0(blackbox\_out[0],SO11\_wire,TESTMODE,e5\_mux\_out[0]);

mux2x1 DRC\_e5\_1(blackbox\_out[1],SO11\_wire,TESTMODE,e5\_mux\_out[1]);

mux2x1 DRC\_e5\_2(blackbox\_out[2],SO11\_wire,TESTMODE,e5\_mux\_out[2]);

mux2x1 DRC\_e5\_3(blackbox\_out[3],SO11\_wire,TESTMODE,e5\_mux\_out[3]);

mux2x1 DRC\_e5\_4(blackbox\_out[4],SO11\_wire,TESTMODE,e5\_mux\_out[4]);

mux2x1 DRC\_e5\_5(blackbox\_out[5],SO22\_wire,TESTMODE,e5\_mux\_out[5]);

mux2x1 DRC\_e5\_6(blackbox\_out[6],SO22\_wire,TESTMODE,e5\_mux\_out[6]);

mux2x1 DRC\_e5\_7(blackbox\_out[7],SO22\_wire,TESTMODE,e5\_mux\_out[7]);

mux2x1 DRC\_e5\_8(blackbox\_out[8],SO22\_wire,TESTMODE,e5\_mux\_out[8]);

mux2x1 DRC\_e5\_9(blackbox\_out[9],SO22\_wire,TESTMODE,e5\_mux\_out[9]);

xor (x[0],e5\_mux\_out[0],enco1[0]);

xor (x[1],e5\_mux\_out[1],enco1[1]);

xor (x[2],e5\_mux\_out[2],enco1[2]);

xor (x[3],e5\_mux\_out[3],enco1[3]);

xor (x[4],e5\_mux\_out[4],enco1[4]);

xor (x[5],e5\_mux\_out[5],enco1[5]);

xor (x[6],e5\_mux\_out[6],enco1[6]);

xor (x[7],e5\_mux\_out[7],enco1[7]);

xor (x[8],e5\_mux\_out[8],enco1[8]);

xor (x[9],e5\_mux\_out[9],enco1[9]);

endmodule

module pll(

input REFCLK,

output clk

);

//assign clk=REFCLK;

endmodule

module blackbox(

output RESET\_out,

output [9:0]DATAOUT

);

//assign RESET\_out = 1'b0;

//assign DATAOUT = 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

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 SE, // Scan enable

input wire SI1, // Scan in for chain 0

input wire SI2, // Scan in for chain 1

input wire [9:0] d\_in,

output wire [9:0] q\_out,

output wire SO1, // Scan out for chain 0

output wire SO2 // Scan out for chain 1

);

wire [5:0] scan\_chain0; // Chain for bits [0:9]

wire [5:0] scan\_chain1; // Chain for bits [10:19]

assign scan\_chain0[0] = SI1;

assign scan\_chain1[0] = SI2;

genvar i;

// Chain 0 (bits 0-9)

generate

for (i = 0; i < 5; i = i + 1) begin : sdff\_chain0

sdff sdff\_inst3 (

.clk(clk),

.RESET(RESET),

.d(d\_in[i]),

.si(scan\_chain0[i]),

.SE(SE),

.q(scan\_chain0[i+1])

);

assign q\_out[i] = scan\_chain0[i+1];

end

endgenerate

// Chain 1 (bits 10-19)

generate

for (i = 5; i < 10; i = i + 1) begin : sdff\_chain1

sdff sdff\_inst4 (

.clk(clk),

.RESET(RESET),

.d(d\_in[i]),

.si(scan\_chain1[i-5]),

.SE(SE),

.q(scan\_chain1[i-5+1])

);

assign q\_out[i] = scan\_chain1[i-5+1];

end

endgenerate

assign SO1 = scan\_chain0[5];

assign SO2 = scan\_chain1[5];

endmodule

module pipo20 (

input wire clk,

input wire RESET,

input wire SE, // Scan enable

input wire SI1, // Scan in for chain 0

input wire SI2, // Scan in for chain 1

input wire [19:0] d\_in,

output wire [19:0] q\_out,

output wire SO1, // Scan out for chain 0

output wire SO2 // Scan out for chain 1

);

wire [10:0] scan\_chain0; // Chain for bits [0:9]

wire [10:0] scan\_chain1; // Chain for bits [10:19]

assign scan\_chain0[0] = SI1;

assign scan\_chain1[0] = SI2;

genvar i;

// Chain 0 (bits 0-9)

generate

for (i = 0; i < 10; i = i + 1) begin : sdff\_chain0

sdff sdff\_inst1 (

.clk(clk),

.RESET(RESET),

.d(d\_in[i]),

.si(scan\_chain0[i]),

.SE(SE),

.q(scan\_chain0[i+1])

);

assign q\_out[i] = scan\_chain0[i+1];

end

endgenerate

// Chain 1 (bits 10-19)

generate

for (i = 10; i < 20; i = i + 1) begin : sdff\_chain1

sdff sdff\_inst2 (

.clk(clk),

.RESET(RESET),

.d(d\_in[i]),

.si(scan\_chain1[i-10]),

.SE(SE),

.q(scan\_chain1[i-10+1])

);

assign q\_out[i] = scan\_chain1[i-10+1];

end

endgenerate

assign SO1 = scan\_chain0[10];

assign SO2 = scan\_chain1[10];

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

module mux2x1(in1,in2,SEl,out);

input in1,in2,SEl;

output out;

wire SElb,w1,w2;

//out=in1SEl'&in2SEl

not(SElb,SEl);

and a1(w1,SElb,in1);

and a2(w2,SEl,in2);

or o1(out,w1,w2);

endmodule

module sdff(d,si,SE,clk,RESET,q);

input si,SE,d,RESET,clk;

output q;

wire m;

mux2x1 mux2(d,si,SE,m);

dff\_gate dff\_inst (clk,m,RESET,q);

endmodule

**UPDATED DFT TESTBENCH :**

module DFT\_TESTBENCH;

reg REFCLK,CLK2,RESET,TESTMODE;

reg SI1,SI2,SE;

reg [4:0] DATA\_IN;

wire [4:0] DATAOUT;

wire SO1,SO2;

UPDATED\_DFT dut(.REFCLK(REFCLK),.RESET(RESET),.TESTMODE(TESTMODE),.SI1(SI1),.SI2(SI2),.SE(SE),.SO1(SO1),.SO2(SO2),.CLK2(CLK2),.DATA\_IN(DATA\_IN),.DATAOUT(DATAOUT));

initial begin

$dumpfile("test.vcd");

$dumpvars(0,DFT\_TESTBENCH);

end

initial begin

REFCLK=1;

forever #5 REFCLK = ~REFCLK;

end

initial begin

CLK2=1;

forever #5 CLK2 = ~CLK2;

end

initial begin

RESET=1'b1; TESTMODE=1'b0; SE=1'b0; SI1=1'b1; SI2=1'b1; DATA\_IN=5'b01111;

//Shiftin -> capture->shiftout

//shiftin

#10 RESET=1'b0;

DATA\_IN = 5'b00001;

TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

//capture

// #10 SE=1'b0;

//shiftout

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

//shiftin

#10 DATA\_IN = 5'b01111; TESTMODE=1'b1; SE=1'b1; SI1=1'b1; SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b1; SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b1;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=0'b0;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b1;SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=0'b1;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=0'b1;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0;SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b1;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0;SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b1;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b1;SI2=1'b1;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0;SI2=1'b0;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b1;SI2=1'b1;

//capture

// #10 SE=1'b0;

//shiftout

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10 SE=1;

#10; DATA\_IN= 5'b00100;

#10; DATA\_IN = 5'b00011;

#10; DATA\_IN = 5'b00111;

#10; DATA\_IN = 5'b11111;

#40;

#300;

#10 TESTMODE=1'b1; SE=1'b1; SI1=1'b0; SI2=1'b0;

$finish;

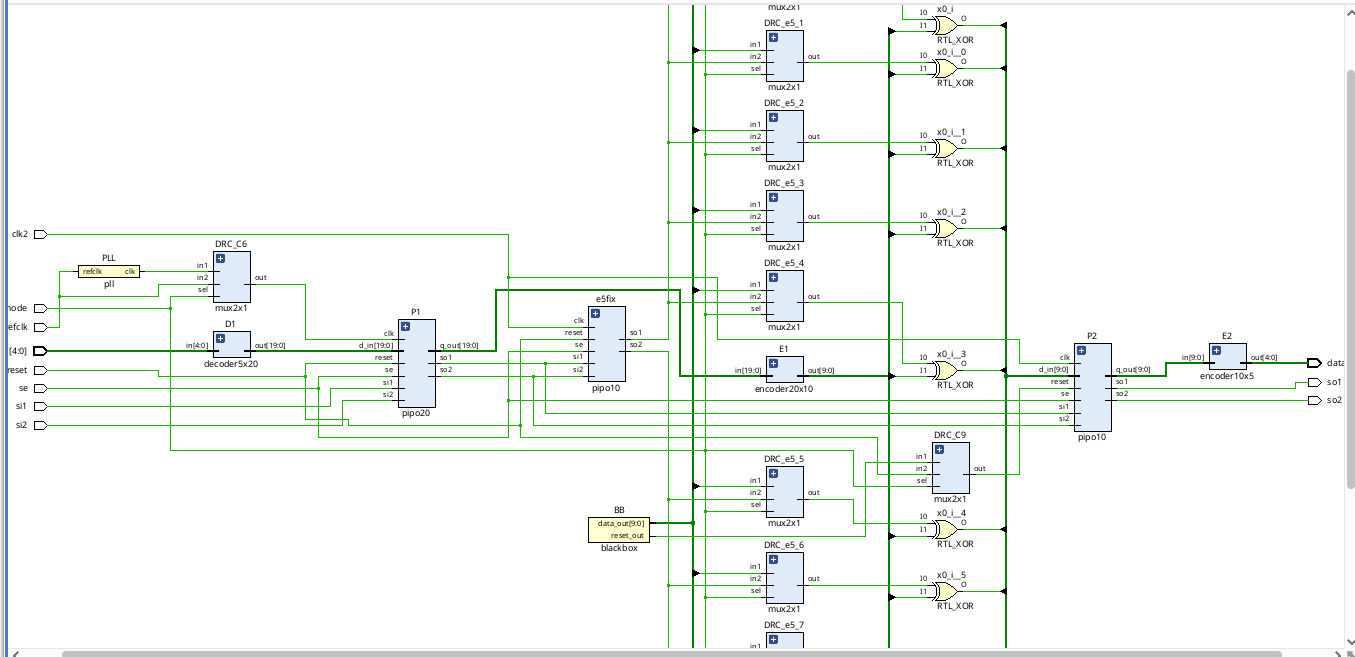
end

initial

$monitor("Time= %t DATA\_IN = %b and DATAOUT = %b and SE=%b SI1=%b and SI2=%b and SO1=%b and SO2=%b ", $time,DATA\_IN,DATAOUT,SE,SI1,SI2,SO1,SO2);

endmodule

**Design Schematic :**



**Executive Summary**

This project focuses on the **design, modification, and verification of a digital circuit** while ensuring compliance with industry-standard **Design Rule Checks (DRC)**. The work covers multiple aspects of physical and logical design such as clock terminology, clock rules implementation, black box insertion, scan chain insertion, error correction, and resolving violations. The project highlights the **practical challenges faced during VLSI design**, especially with clocking, PLL integration, and scan insertion. Finally, violations such as **E5 errors** were analyzed and corrected, ensuring that the final design met sign-off requirement

**Introduction**

### ****Background and Context of the Project****

With the continuous scaling of semiconductor technology, digital designs have become more complex, requiring strict adherence to **Design Rule Checks (DRC)** and robust clocking strategies. Clock management, design modifications, black box integration, and scan insertion are critical steps in achieving a **functional, manufacturable, and testable design**. This project was carried out to simulate an **industry-oriented design environment**, where each stage of the flow — from implementation to DRC fixing — mirrors real VLSI practices.

### ****Problem Statement or Goals of the Project****

The main challenges addressed in this project include:

* Understanding and applying **clock terminology and rules** to ensure proper timing.
* Implementing a **practical design** and integrating additional logic such as PLL and black boxes.
* Identifying and **resolving DRC and E5 violations** encountered during design flow.
* Performing **scan chain insertion** to enhance design-for-testability (DFT).
* Delivering a **final clean design** that is error-free and industry compliant.

### ****Scope and Limitations of the Project****

**Scope:**

* Covers **clock rule implementation**, design modification, and PLL insertion.
* Focuses on **error identification and correction** in physical design.
* Includes **DRC fixing and scan insertion** for improving manufacturability and testability.

### ****Innovation Component in the Project****

* **Integration of black boxes** to simulate **third-party IP insertion**, reflecting real SoC integration challenges.
* **PLL addition** for clock generation and synchronization, making the design closer to real-world silicon requirements.
* **Systematic fixing of E5 violations** and DRCs, showcasing a structured debugging approach.

**Project Objectives**

### ****Objectives and Goals of the Project****

* The objectives of this project were:
* To understand and apply **clock terminology and rules** for reliable circuit timing.
* To implement a **practical digital design** and modify it by adding black boxes and PLL logic.
* To identify and **correct design errors** that arise during synthesis and implementation.
* To analyze and resolve **Design Rule Check (DRC) violations** systematically.
* To specifically **fix E5 violations** encountered in the design flow.
* To perform **scan chain insertion** to enhance testability and design validation.
* To gain hands-on experience with **industry-relevant VLSI design methodologies**.

### ****Expected Outcomes and Deliverables****

* A **functional and verified design** free from DRC and E5 violations.
* Successful **integration of PLL and black box modules** into the design.
* A design enhanced with **scan chains** to support testability.
* A **documented methodology** highlighting the steps of error detection, correction, and verification.
* A **comprehensive project report** demonstrating both the technical execution and the industry relevance of the work.

**Methodology and Results**

## ****Methods / Technology Used****

* **Digital Design Flow:** The project followed a structured VLSI design methodology including design entry, synthesis, modification, verification, and DRC fixing.
* **Clock Domain Handling:** Implemented clock rules, managed skew, latency, and integrated PLL for clock synchronization.
* **Error Correction:** Identified and corrected **E5 violations** and DRC errors during place-and-route.
* **Design-for-Testability (DFT):** Inserted scan chains and verified scan shifting integrity using test patterns.
* **Iterative Debugging Approach:** Errors and violations were resolved in multiple iterations until a **clean design** was achieved.

### ****Tools / Software Used****

* **EDA Tools:**
  + Xilinx Vivado
* **Simulation Tools:** Xilinx Vivado for functional verification.
* **Version Control:** Daily report for tracking modifications and sharing final code/report.

### ****Project Architecture****

The **project architecture** followed a hierarchical digital design flow:

1. **Input RTL Design** → Base module developed.
2. **Clock Rule Integration** → Defined skew, latency, and duty cycle parameters.
3. **Design Modification** → Added black boxes and PLL module.
4. **Synthesis & Implementation** → Logical to physical conversion.
5. **Error Detection** → DRC and E5 violations identified.
6. **Violation Fixing** → Placement/routing fixes applied.
7. **Scan Chain Insertion** → DFT enhancement for testability.
8. **Verification & Final Clean Design → Functional + DRC verified.**

**Learning and Reflection**

## ****Learning & Reflection****

### ****New Learnings (Technology & Management)****

### ****Technical Learnings****

* + Gained a strong understanding of **clock terminology and clock rules**, including skew, latency, duty cycle, and jitter.
  + Learned how to **integrate PLLs** for clock synchronization and how to manage black box modules in design flow.
  + Acquired hands-on knowledge of **Design Rule Checks (DRC)**, identifying violations, and systematically fixing them.
  + Practiced **error correction of E5 violations** and understood their impact on physical design.
  + Implemented **scan chain insertion** and verified functionality through test patterns, reinforcing concepts of **Design for Testability (DFT)**.
  + Improved scripting skills in **TCL/Python** to automate design tasks and manage reports.

**Project/Management Learnings**

* + Developed the ability to **plan design iterations** effectively, handling errors step by step.
  + Gained experience in **collaborative documentation and version control** using GitHub.
  + Learned the importance of **time management** in completing multiple design cycles within deadlines.
  + Enhanced **problem-solving mindset**, as each violation required careful debugging and innovative fixes.

**Conclusion and Future Scope**

## ****Conclusion****

### ****Recap of Objectives and Achievements****

The main objectives of this project were to:

* Understand and apply **clock terminology and rules** for reliable design implementation.
* Modify the design by adding **PLL logic and black box modules**.
* Identify and resolve **DRC violations** and specifically address **E5 errors**.
* Enhance the design with **scan chain insertion** for improved testability.

**Achievements:**

* Successfully implemented clock rules and ensured proper synchronization using PLL.
* Modified the design to integrate black boxes without functional conflicts.
* Identified and corrected **all DRC violations**, ensuring a clean design ready for sign-off.
* Resolved **E5 violations**, demonstrating practical debugging skills.
* Performed **scan insertion** and verified scan integrity through pattern shifting.
* Documented the entire flow systematically, reflecting **real-world VLSI practices**