Innovation & Entrepreneurship Hub for Educated Rural Youth (SURE Trust – IERY)

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

Period of the project

May 2025 to July 2025

Innovation & Entrepreneurship Hub for Educated Rural Youth (SURE Trust – IERY)

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

Mr.Ryan Ebenezer
DFT engineer—Struent semiconductors pvt ltd

Prof. Radhakumari
Executive Director & Founder
SURE Trust

Table of contents

Detailed Report:

DATE	MINUTES OF MEETING	ATTENDEES
26/05/25	Assigned a task to create word document for project report,Revise the Concepts discussed earlier	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar
28/05/25	Design Rule Overview Scan insertion Checking How to troubleshoot Rules violations Clock rules , Clock terminology	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar
03/06/25	1. Introduction to clock Rules 2. Assigned to Complete the rules – C1, C2, C3, C6	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar
07/06/25	Explaination of clock rules C1, C2	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar
16/06/25	Implementation of the Design	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar
28/06/25	Modification of the Design and Adding Black Box	K.Chandu P. Vijay Kumar K. Tharun Krishna

		P. Pavan kumar
03/07/25	Addition of PLL to the Design	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar
06/07/25	Correction Of DRC in the Design	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar
09/07/25	Fixing of E5 violation and Scan insertion	K.Chandu P. Vijay Kumar K. Tharun Krishna P. Pavan kumar

DFT PROJECT REPORT

26/05/2025(Monday)

Minutes of the Meeting

Agenda: 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

Task given : Revise the Concepts discussed earlier

Theory : Design for Test (DFT) helps make sure digital circuits can be tested effectively after manufacturing. One important concept in DFT is controllability and observability—how easily a test can control or observe the values inside a chip.

To improve testability, designers insert scan chains, which are just connected flip-flops that allow internal signals to be controlled and read out during testing. These flip-flops switch between normal and test modes using a control signal like Scan Enable (SE).

During testing, the design enters a scan mode where data is shifted into the scan chains (shiftoperation), and then clocked normally to capture the effect of the logic (capture operation). This makes it easier to find faults deep inside the chip.

ATPG (Automatic Test Pattern Generation) tools create the exact set of input patterns needed to detect possible faults. These patterns are then used in manufacturing to check if each chip is working properly.

28/05/2025(Wednesday)

Minutes of the meeting :

Task given : Design Rule Overview

1. Scan insertion Checking

2. How to troubleshoot Rules violations

3. Clock rules , Clock terminology

Theory :

Scan insertion checking :

• When design has clock signals that are controlled by other logic, a test procedure file is used to describe how those clocks can be activated during testing.

• If some clock logic can’t be controlled directly, the tool can help by adding special test logic to make testing possible.

• If we already have scan circuitry in your design or use a test procedure file that defines which parts can be scanned, the tool will do deeper checking. After adding scan logic and writing the test procedure file, you should re-enter setup mode to give this info to the tool. Then, go back to analysis mode to perform detailed checks before creating scan test patterns

Clock Rules :

The application checks the scan clocks to ensure their proper definition and operation. You may select the handling of any clock rule to be error, warning, note, or ignore.

Clock Terminology :

Clock signals :

The transition of the clock from the off state to the on state is considered the leading edge of thecclock, while the transition from the on state to the off state is considered the trailing edge of the clock.

Clock cones and Effect cones :

• A Clock Cone is the part of the circuit controlled by one clock signal.

It includes all flip-flops and logic that receive that clock.

• An Effect Cone is the logic and outputs affected by a particular signal or flipflop.

It shows how a change in one part of the circuit can affect outputs.

03/06/2025

Minutes Of Meeting :

Agenda : Introduction to Clock Terminology

Task given : Assigned to complete the Clock rules – C1 , C2 , C3, C6and update to the document

C1 :

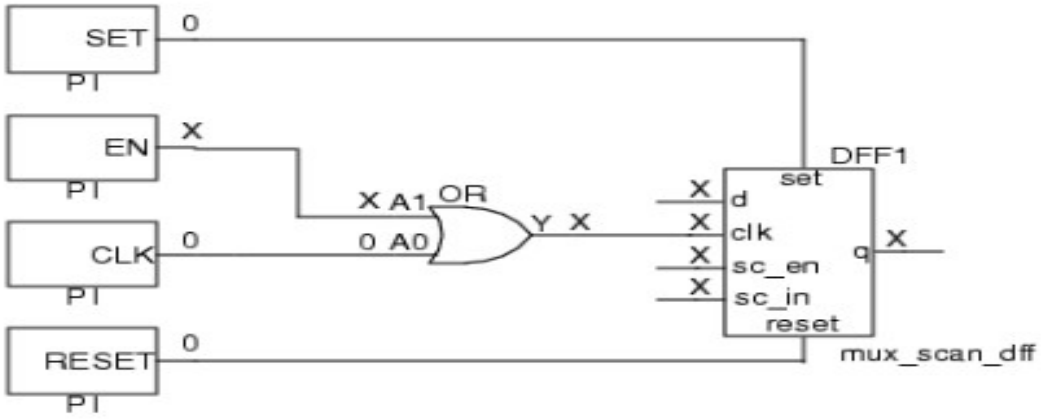
Rule : A scan or non-scan cell must not capture data when all specified clocks are set to their off states .

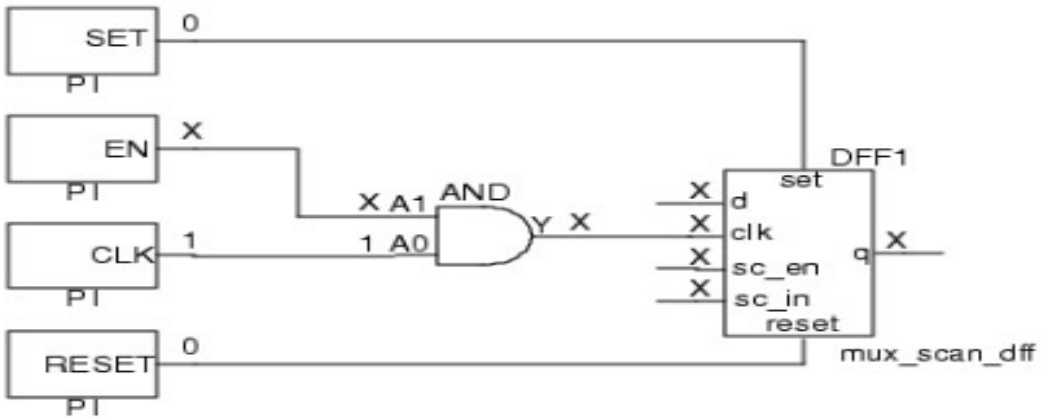
• When all clocks are at their off state as defined with the add_clocks command, all clock inputs (including sets and resets) of scan and non-scan cells must be at their off state. For non-scan cell violations, the tool converts these to TIEX.

Violations :

- **incorrect Clock Off-State Definition:** If the off-state of a clock is not correctly defined, cells may inadvertently capture data
- **Uninitialized Primary Inputs:** Primary inputs not initialized to known values can cause indeterminate states, leading to violations

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.

Possible Resolution : If your debugging effort shows an offending clock's off state is incorrect, use the add_clocks command to redefine the clock with the correct off state. If the off state is correct,the problem typically is due to an indeterminate value (X) in related logic that causes the clock'svalue at a scan cell to be X. This is the case in the example shown in the figure above. You can fix these by using the add_input_constraints command to constrain a primary input pin to a value that removes the X: add_input_constraints -c0 EN This allows the tool to consider the CLK signal to be the only clock signal to the clk input of the scan cell.

C2:

Rule : Each clock must have a structural path to the clock port of at least one memory element.

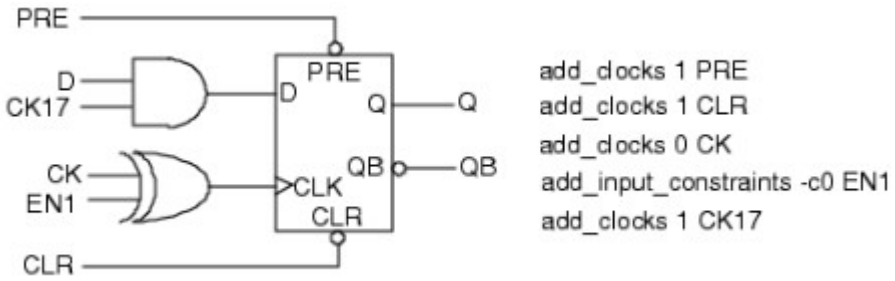
• This rule ensures that every defined clock has a valid path to at least one memory element's clock port, such as flip-flops or latches. Without this, the clock cannot control any memory elements, leading to potential test coverage issues.

Violation :

• **Missing Clock Definitions:** A clock signal is used in the design but isn't defined as a clock in the tool's clock domain.

• **Incorrect Clock Definitions:** A signal is defined as a clock but doesn't have a valid path to any memory element's clock port

• **Clock MUXing Issues:** A clock signal is routed through a multiplexer, and the select line isn't constant, making the clock path unreliable.



If you run rules checking on this design, you get a C2 rules violation because while the CK17 signal 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:

When a sequential element (or RAM) source and sink are clocked by the same clock and the sink captures data from the source, a potential exists for the captured data to pass through both the source and sink in the same clock cycle. That is, the sink might capture the source's new data (captured in the same cycle) instead of the source's old data (captured in the previous cycle).

source: stores and outputs some data.

sink: takes the data from the source and captures it on the same clock edge.

In a clean design:

• On **clock cycle N**, the **source** captures new data.

• On **clock cycle N+1**, the sink captures the output **of the source** — which is now stable and

valid from the previous cycle.

This creates a **one-cycle delay** between source and sink — a common pipeline behavior.

If there's **very little delay** between the source and sink :

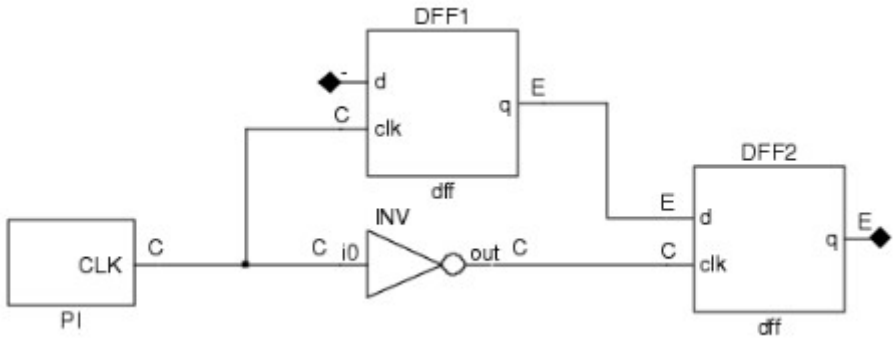
• The **source** captures **new input data.**

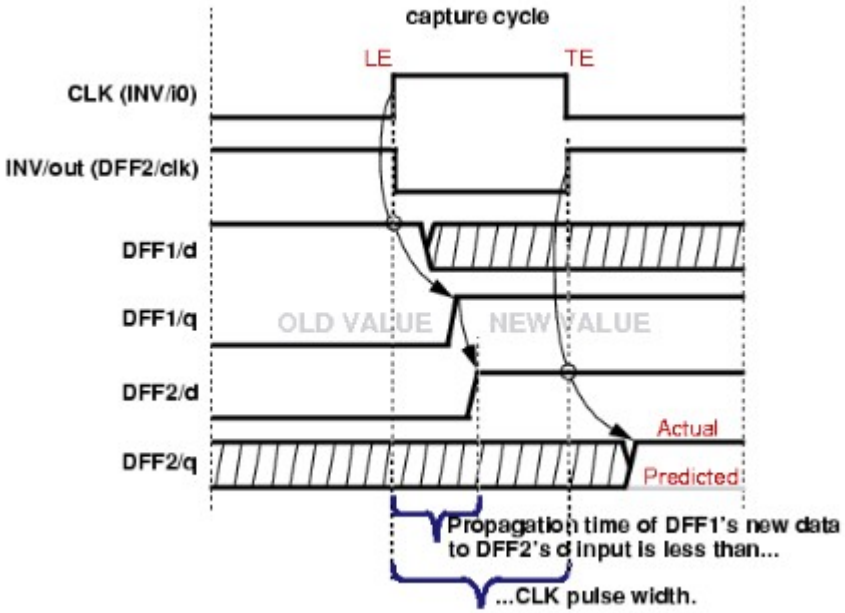
• The **sink** tries to capture the **source's output** — but:

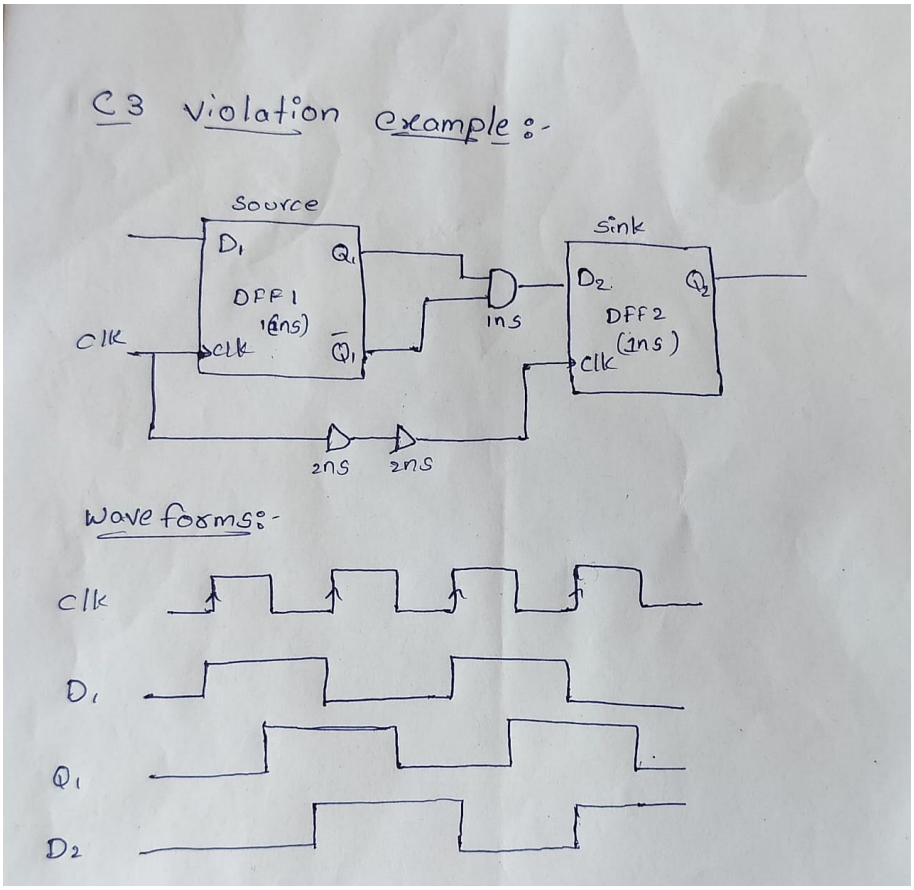
• If the source's new data propagates through the combinational logic too quickly,

• The **sink could capture the new data**, not the old one

Example:




capture cycle
LE
TE
CLK (INV/i0)
INV/out (DFF2/clk)
DFF1/d
DFF1/q
OLD VALUE
NEW VALUE
DFF2/d
DFF2/q
Actual
Predicted
Propagation time of DFF1's new data to DFF2's d input is less than...
...CLK pulse width.


C3 violation example :-
Source
sink
DFF1
(1ns)
DFF2
(1ns)
1ns
2ns
2ns
clk
Wave forms :-
clk
D1
Q1
D2

The rule violation may occur for a clock if one of the following is true:

• The clock input of a DFF state element is in the clock cone, its input data is in the effect cone, the element is trailing edge triggered, and the effect data comes from an element that is leading edge triggered.

• The clock input of a LA state element is in the clock cone and its input data is in the effect cone.

• The write input of a RAM is in the clock cone and a data input or address input of the associated write port is in the effect cone.

•The read input of a RAM is in the clock cone and an address input of the associated read port is in the effect cone

Debug with Tessent Visualizer:

To view the location of a C3 DRC violation using Tessent Visualizer, use the following command

steps:

1. set_system_mode analysis

2. Choose Open > DRC browser from the Tessent Visualizer menu and double-click the desired C3 DRC violation in the table (or right-click and select “Visualize DRC” from the popup menu). The gates between the source cell and the failing cell, including the clock cone data for the failing clock, are displayed in the Flat Schematic.

C6:

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

C6 rule is primarily about **avoiding clock-gated race conditions** during scan shift operations in **scan chains.**

What C6 verifies :

In DFT:

• During **scan shift,** scan DFFs are connected in a chain and data is shifted using the **scan clock (typically the same system clock).**

• **Each scan DFF’s SI (scan in)** must receive its data from a **stable source** before the clock edge.

The **C6 check ensures** that:

• The **clock** that captures scan data does **not also control or influence the logic** (i.e.,**combinational path**) generating that scan data in the **same cycle**.

• This avoids a **race condition:** where a clock signal races with the data it is capturing,

potentially causing incorrect data capture.

Violation :

A C6 violation happens when:

• A **clock signal** or **clock-gated enable** controls both:

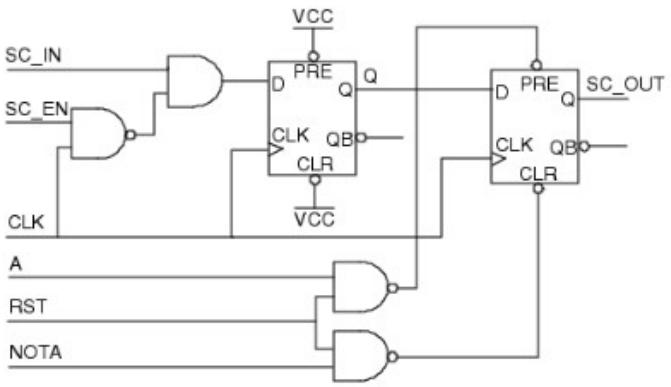
• The **generation** of scan data (via logic gates)

• And the **capture** of that data by a flip-flop

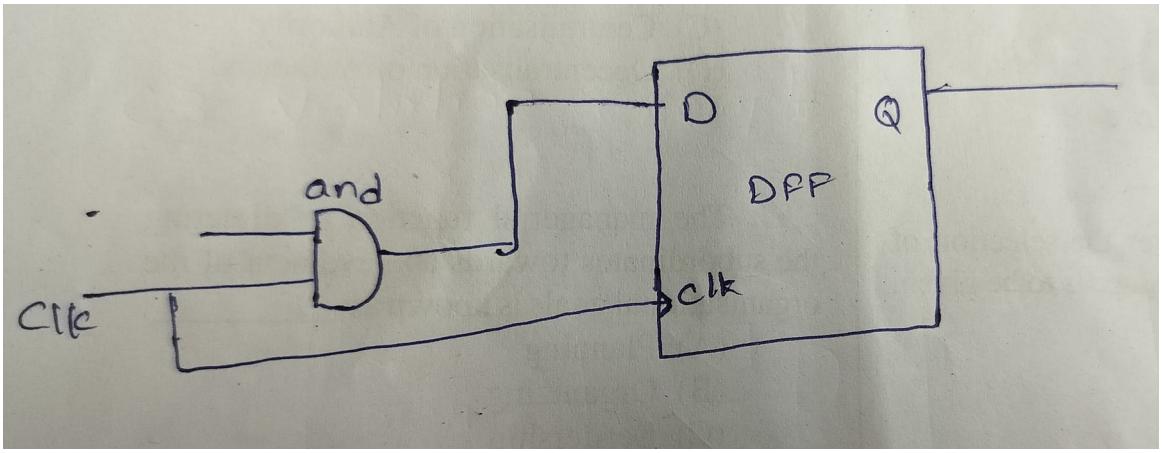
This means that a **change in clock** might cause both the data and the capturing edge to shift at the

same time - **race condition.**

Example:



If you ran rules checking on this design, given the setup commands shown, you would get a C6 rules 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.



10/06/2025

D5:

Rule : All memory elements (latches and flip-flops) must be scannable. The application performs this check after identifying all scan memory elements. The rule violation occurs for all memory elements not identified as part of a scan cell.

• Any latch or flip-flop not part of the scan chain and not having a valid reason for exclusion violates this rule.

What D5 verifies :

The DFT tool checks whether **all sequential elements** (typically **flip-flops** and **latches**) in the RTL

are:

• Included in **scan chains** (i.e., connected via scan_in / scan_out ports)

• Or explicitly **exempted** with valid justification (like clock gating cells, test-only logic, etc.)

Violation :

Any latch or flip-flop not part of the scan chain and not having a valid reason for exclusion violates this rule.

Occurs when:

• One or more flip-flops **do not have scan enable (SE)** or **scan input/output connections.**

• Memory elements **not connected to scan chains** are **not declared exempt.**

• This results in **reduced test coverage,** as these FFs cannot be directly controlled or observed during scan testing.

When the tool identifies a non-scan memory element, the tool classifies each element into one of the following types:

• INIT-0 — If it is at 0 at the beginning of the first capture cycle.

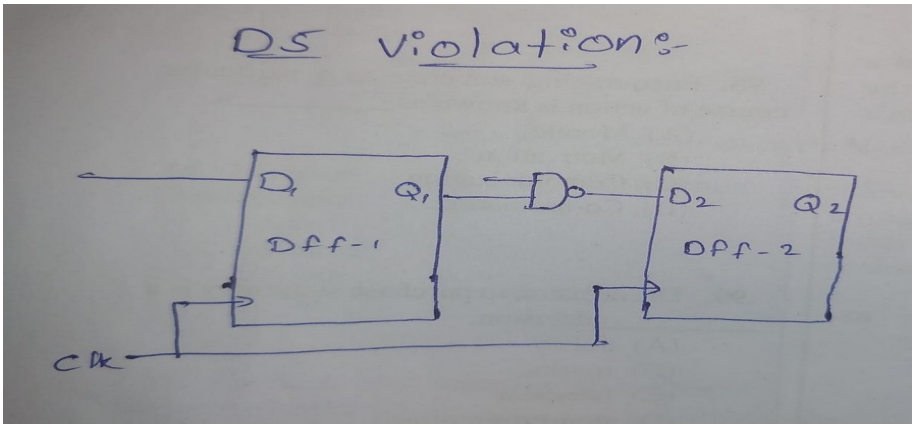
• INIT-1— If it is at 1 at the beginning of the first capture cycle.

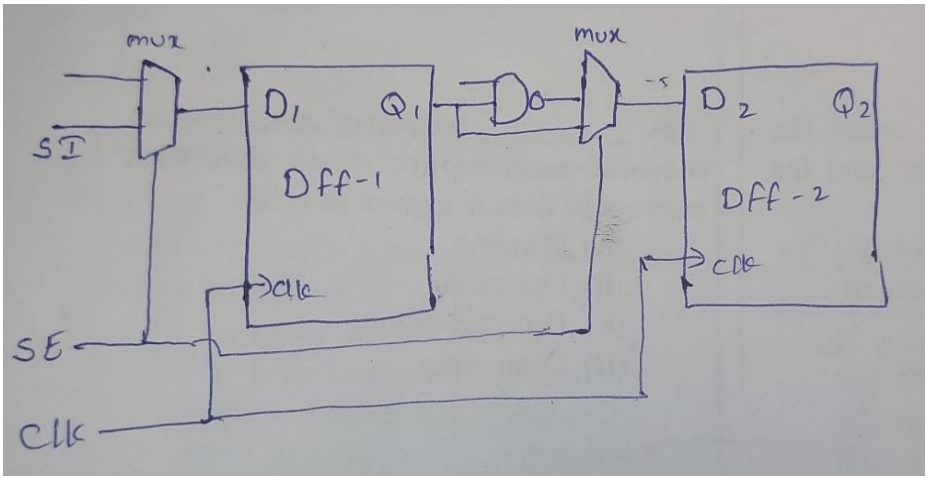
• INIT-X — If its state is unknown at the beginning of the first capture cycle and it may go to any state during capture.

• TIE-0 — If it is always at 0 during capture.

• TIE-1 — If it is always at 1 during capture.

• TIE-X — If it is always at an unknown state during capture





E5:

Rule : When the application places constrained states on constrained pins and binary states on PIs and scan cells, X states must not propagate to an observable point.

What E5 verifies :

During ATPG (e.g., stuck-at fault or transition testing), the tool sets:

• **Constrained values** (0/1) on **constrained pins** (e.g., clocks, resets)

• **Binary values** on **Primary Inputs (PIs)** and **scan cells**

The rule **checks that X (unknown) values** inside the design **do not propagate:**

• To **Primary Outputs (POs)**

• To **Scan Output (SO)** of a scan chain

• To any **observable point** used in fault simulation

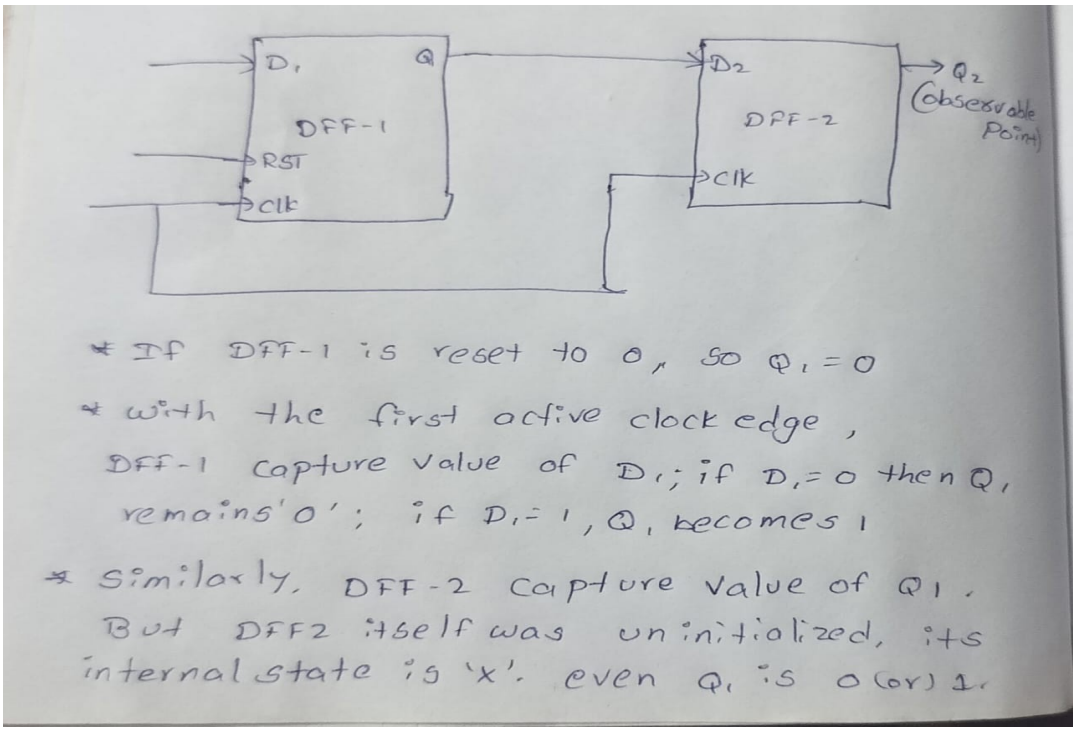
Violation: X-States Reaching Observables

Violation happens when :

• An internal X (unknown) value **from an uninitialized register**, **tri-state bus, or undriven** net leaks to an output.

• X propagates through **combinational logic** to a scan output or functional output.

• The ATPG tool **can't determine** the logic value at that point — reducing fault coverage.

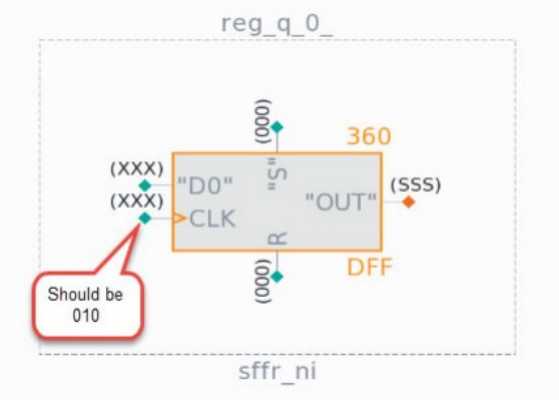


T3:

Rule : A failure in Tracing a scanchain from its output back to its input pin arises when . This typically tool encounters a blockage in the scan chain path, preventing complete tracing

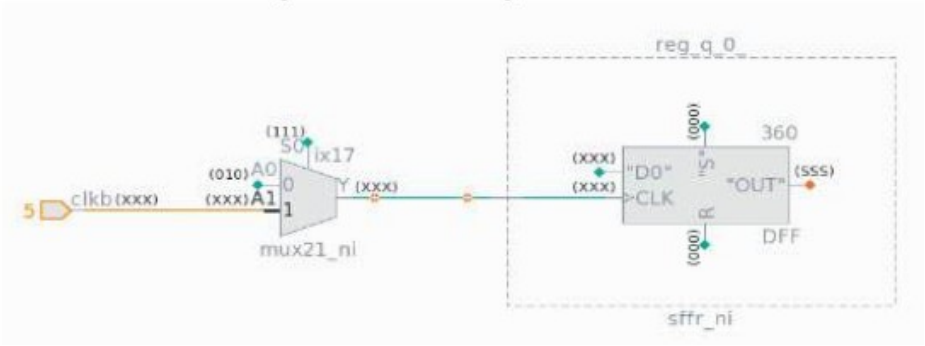
"The shift procedure must create a sensitizable path from the scan chain output back to the scan chain input."

Violation:



The clock “CLK” should be 010,not XXX.

Figure 7-51. Tracing T3 Violation

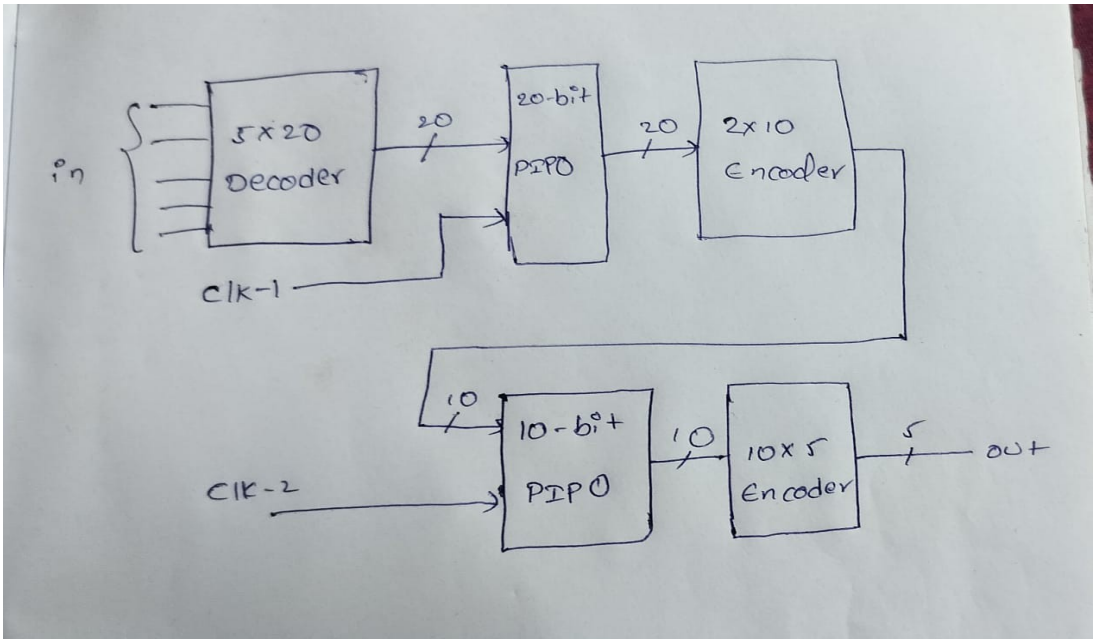


16/06/25

Agenda :

- **review of the project**
- **design and implementing phase**

Theoritical Knowledge :



Hands On Design:

design:

code:

```
module
GOLDEN_DESIGN(refclk,clk2,data_in,test_mode,si1,si2,se,so1,so2,reset,data_out;
  input refclk,clk2;
  input si1,si2,se;
  output so1,so2;
  input [4:0]data_in;
  input reset,test_mode;
  output  [4:0]data_out;


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,test_mode,m1);
 pipo20 P1 (m1,reset,se,si1,si2,deco1,pipo1,so1wire,so2wire);
 encoder20x10 E1(pipo1,enco1);
 mux2x1 DRC_C9(reset_out,reset,test_mode,m2);
 pipo10 P2 (clk2,m2,se,so1wire,so2wire,x,pipo2,so1,so2);
 encoder10x5 E2(pipo2,data_out);
  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,test_mode,e5_mux_out[0]);
 mux2x1 DRC_e5_1(blackbox_out[1],so11_wire,test_mode,e5_mux_out[1]);
 mux2x1 DRC_e5_2(blackbox_out[2],so11_wire,test_mode,e5_mux_out[2]);
 mux2x1 DRC_e5_3(blackbox_out[3],so11_wire,test_mode,e5_mux_out[3]);
 mux2x1 DRC_e5_4(blackbox_out[4],so11_wire,test_mode,e5_mux_out[4]);
 mux2x1 DRC_e5_5(blackbox_out[5],so22_wire,test_mode,e5_mux_out[5]);
 mux2x1 DRC_e5_6(blackbox_out[6],so22_wire,test_mode,e5_mux_out[6]);
 mux2x1 DRC_e5_7(blackbox_out[7],so22_wire,test_mode,e5_mux_out[7]);
 mux2x1 DRC_e5_8(blackbox_out[8],so22_wire,test_mode,e5_mux_out[8]);
 mux2x1 DRC_e5_9(blackbox_out[9],so22_wire,test_mode,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]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

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
```

Testbench:

```
module scan_integrity_tb;
  reg refclk, clk2, reset, test_mode, se;
  reg si1, si2;
  reg [4:0] data_in;
  wire [4:0] data_out;
  wire so1, so2;

  // Instantiate DFT1 module
  GOLDEN_DESIGN uut (
    .refclk(refclk),
    .clk2(clk2),
    .data_in(data_in),
    .test_mode(test_mode),
    .si1(si1),
    .si2(si2),
    .se(se),
    .so1(so1),
    .so2(so2),
    .reset(reset),
    .data_out(data_out)
  );

  // Generate clocks
  initial begin
    refclk = 0;
    forever #5 refclk = ~refclk; // 10ns period
  end
```

```
initial begin
  clk2 = 0;
  forever #6 clk2 = ~clk2; // 12ns period
end

// Task to shift scan data
task scan_shift(input bit scan_val);
  integer i;
  begin
    for (i = 0; i < 10; i = i + 1) begin
      si1 = scan_val;
      si2 = scan_val;
      #10; // one clock cycle
    end
  end
endtask

// Test procedure
initial begin
  $dumpfile("scan_integrity_tb.vcd");
  $dumpvars(0, scan_integrity_tb);

  // Initial values
  se = 1;          // scan enable active
  test_mode = 1;    // test mode active
  reset = 1;        // apply reset
  si1 = 0;
  si2 = 0;
  data_in = 5'b00000;
  #20;
```

```
    reset = 0;


    // Scan shift: all 0s
    $display("Shifting all 0s through scan chain");
    scan_shift(1'b0);
    #20;
    if (so1 !== 1'b0 || so2 !== 1'b0)
      $display("ERROR: Expected SO1=0, SO2=0 after shifting 0s. Got SO1=%b, SO2=%b", so1, so2);
    else
      $display("PASS: Scan out SO1=%b, SO2=%b after all 0s", so1, so2);


    // Scan shift: all 1s
    $display("Shifting all 1s through scan chain");
    scan_shift(1'b1);
    #20;
    if (so1 !== 1'b1 || so2 !== 1'b1)
      $display("ERROR: Expected SO1=1, SO2=1 after shifting 1s. Got SO1=%b, SO2=%b", so1, so2);
    else
      $display("PASS: Scan out SO1=%b, SO2=%b after all 1s", so1, so2);


    $finish;
  end
endmodule
```

28/06/25

Theoritical Knowledge :

- Add some black box to the design
- Creating inbuilt E5 violation in 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.

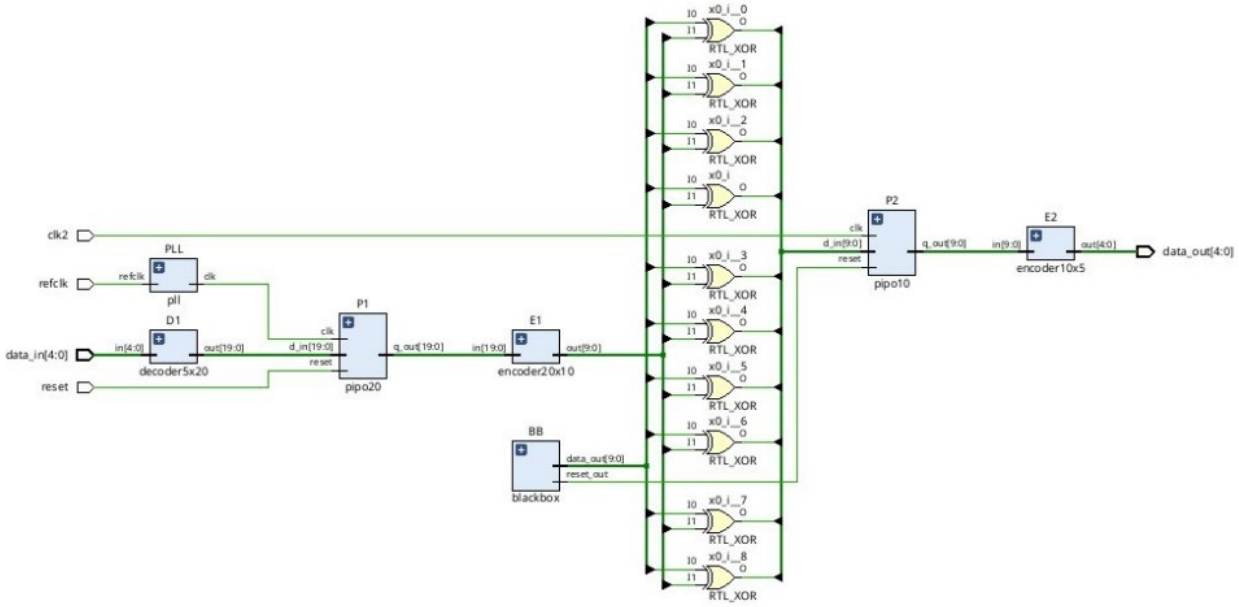
Internally Generated clocks:(1)---Clk1 from PLL to 20 pipo registers.

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.

Schematic :

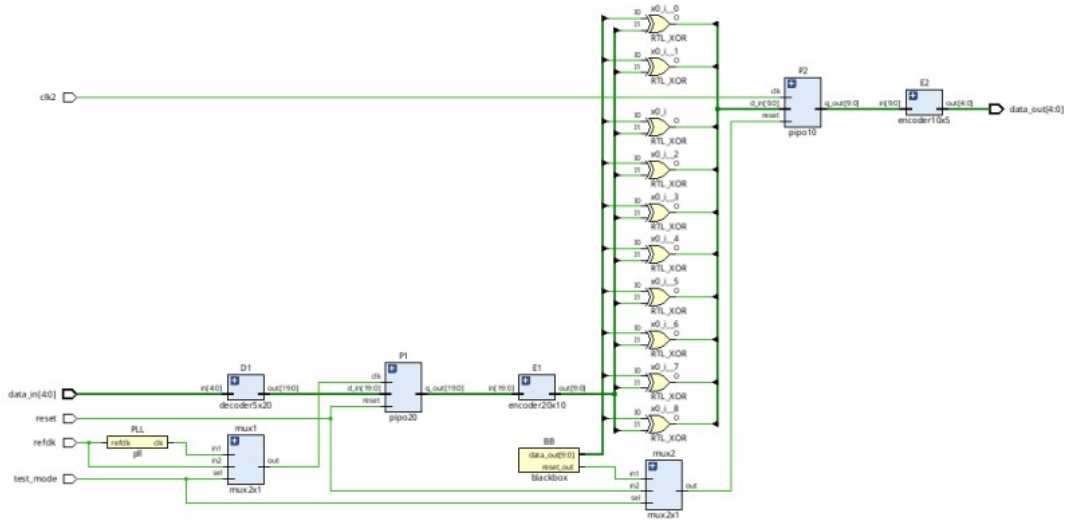


06/07/25

Theoritical Knowledge :

- **DRC** violations C6 and C9 are identified .
- To fix these , MUX were added

Schematic :



09/07/25

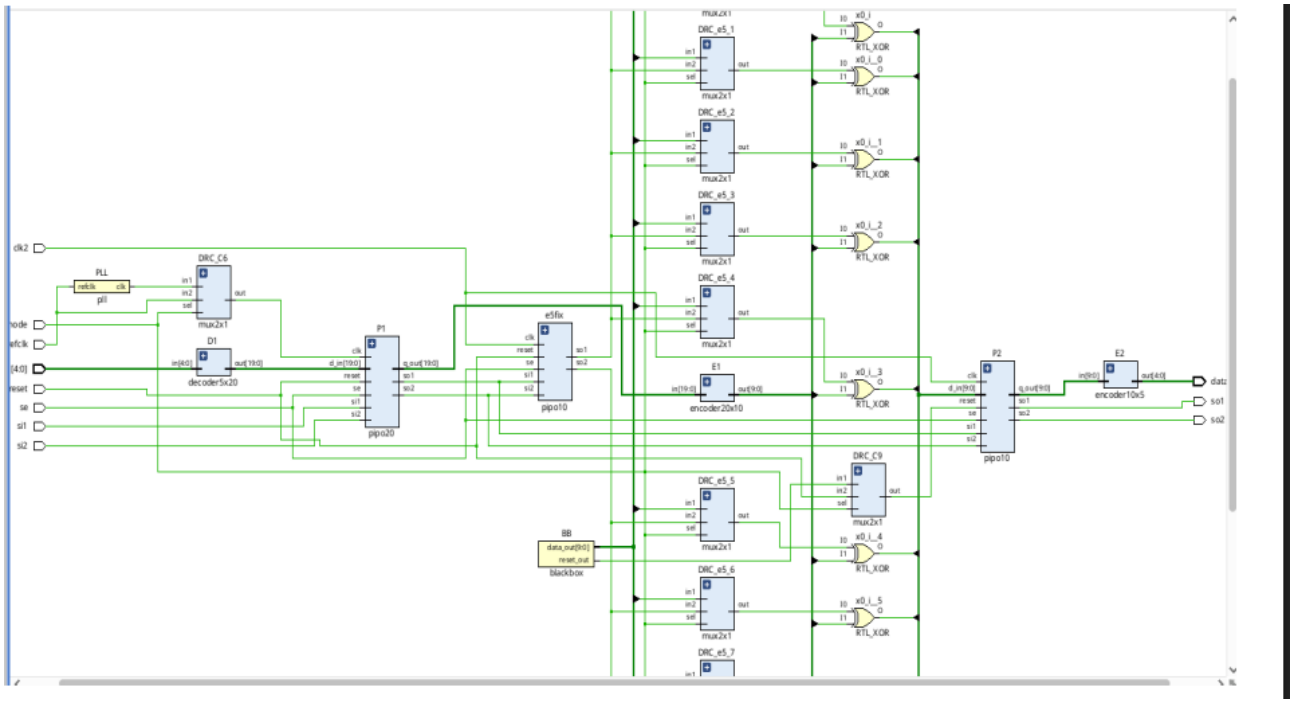
Theoritical Knowledge :

To avoid an E5 violation, we use a multiplexer to select between two input sources based on a control signal called testmode. When testmode is 0, the input is taken from the functional logic (e.g., a blackbox). When testmode is 1, the input should ideally come from a test vector.

However, sourcing this test vector directly from the top level or tying it to constant values (0 or 1) can negatively impact fault coverage. This is because we cannot effectively control the input to test all possible stuck-at faults, reducing the effectiveness of ATPG.

To overcome this limitation, we implement Dynamic Test Points. These test points derive their values from a PIPO (Parallel-In Parallel-Out) register, which in turn gets its data from the preceding scan flip-flops. This way, the test vector becomes controllable via the scan chain, improving observability and controllability, and thereby enhancing fault coverage without causing an E5 violation

Schematic :


clk2
PLL
pll
DRC_C6
mux2x1
D1
decoder5x20
P1
pipo20
e5fix
pipo10
BB
blackbox
DRC_e5_1
DRC_e5_2
DRC_e5_3
DRC_e5_4
E1
encoder20x10
DRC_e5_5
DRC_e5_6
DRC_e5_7
DRC_C9
RTL_XOR
P2
pipo10
E2
encoder10x5
se
si1
si2
so1
so2

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 involved several key aspects of both physical and logical design, including:

- Understanding and applying **clocking concepts** and **clock rules**.
- **PLL integration** and handling clock domain interactions.
- **Black-box insertion** for hierarchical design.
- **Scan chain insertion** to support testability.
- **Error detection and correction**, along with resolving DRC violations.

During the process, practical challenges commonly faced in VLSI design—such as clocking issues, scan chain connectivity, and rule violations—were encountered and systematically resolved. In particular, violations such as **E5 errors** were analyzed and corrected.

The final outcome was a design that successfully met **sign-off requirements**, ensuring functional correctness and adherence to industry standards.

Introduction

Background and Context
With continuous scaling in semiconductor technology, digital designs have become increasingly complex, demanding strict compliance with **Design Rule Checks (DRC)** and robust **clocking strategies**. Effective clock management, design modification, black-box integration, and scan insertion are essential for achieving a **functional, manufacturable, and testable design**. This project was developed to simulate an **industry-oriented design environment**, where each stage—from implementation to DRC fixing—closely mirrors real VLSI practices.

Problem Statement / Goals
The project primarily focused on addressing the following challenges:

- Applying **clock terminology and rules** to achieve proper timing closure.
- Implementing a practical design and integrating additional logic such as **PLL** and **black boxes**.
- Identifying and resolving **DRC and E5 violations** during the design flow.
- Performing **scan chain insertion** to improve Design-for-Testability (DFT).
- Delivering a **final error-free design** that is compliant with industry standards.

Scope and Limitations
Scope of the project includes:

- **Clock rule implementation**, design modifications, and PLL integration.
- **Error identification and correction** in the physical design flow.
- **DRC resolution** and **scan insertion** to improve manufacturability and testability.

Innovation Components

- Integration of **black boxes** to simulate third-party IP insertion, reflecting real **SoC integration challenges**.
- Addition of **PLL** for clock generation and synchronization, making the design more realistic for silicon implementation.
- Systematic debugging and fixing of **E5 violations and DRCs**, highlighting a structured problem-solving approach.
- Inclusion of **scan insertion**, bridging functional design with testability and reinforcing DFT awareness.

Project Objectives

Objectives and Goals
The goals of this project were:

1. To learn and apply **clock concepts and rules** for achieving reliable timing in digital circuits.
2. To design and enhance a digital system by integrating **PLL logic** and **black box modules**.
3. To detect and correct **errors** occurring during synthesis and implementation stages.
4. To study and systematically resolve **Design Rule Check (DRC) issues**.
5. To specifically address and eliminate **E5 violations** within the design flow.
6. To carry out **scan chain insertion** to strengthen design-for-testability (DFT) and validation.
7. To build **practical experience** with workflows aligned to real-world VLSI design practices.

Expected Outcomes and Deliverables

- A **verified design** free of DRC and E5 violations.
- Successful **PLL and black box integration** into the digital design.
- A **scan-enabled design** supporting improved testability.
- A **step-by-step methodology document** covering error detection, correction, and verification.
- A **detailed project report** reflecting both technical execution and its relevance to industry practice.

Methodology and Results

Methods / Technologies Applied

- **Digital Design Methodology:** Adopted a structured VLSI flow covering design entry, synthesis, modifications, verification, and DRC resolution.
- **Clock Domain Management:** Applied clocking rules, addressed skew and latency, and integrated a PLL for clock synchronization.
- **Error Resolution:** Detected and corrected E5 rule violations and DRC issues during place-and-route.
- **Design-for-Test (DFT):** Incorporated scan chains and validated scan shifting functionality using test patterns.
- **Iterative Debug Process:** Resolved errors and violations through multiple refinement cycles until a clean design was achieved.

Tools / Software Utilized

- **EDA Platform:** Xilinx Vivado for synthesis, implementation, and debugging.
- **Simulation Environment:** Functional verification performed using Xilinx Vivado simulator.
- **Version Control:** Daily progress documentation maintained for tracking changes and delivering final code/report.

Project Architecture

The design was executed following a hierarchical digital design methodology:

1. **RTL Design Entry** – Developed the base module.
2. **Clock Integration** – Defined skew, latency, and duty cycle parameters.
3. **Design Enhancement** – Introduced black boxes and PLL module.
4. **Synthesis & Implementation** – Transformed RTL to physical design.
5. **Error Identification** – DRC and E5 rule violations captured.
6. **Violation Correction** – Applied placement and routing fixes.
7. **Scan Chain Insertion** – Enhanced DFT through scan integration.
8. **Verification & Sign-off** – Achieved functional correctness and DRC-clean design.

Project GitHub Link:

Learning & Reflection

New Learnings (Technology & Management)

Technical Learnings

- Built a solid understanding of clock concepts and rules such as skew, latency, duty cycle, and jitter.
- Gained knowledge on integrating PLLs for clock synchronization and handling black box modules within the design flow.
- Acquired practical experience with Design Rule Checks (DRC), identifying violations, and applying systematic fixes.
- Practiced resolving E5 violations and analyzed their impact on physical design.
- Implemented scan chain insertion and validated functionality using test patterns, strengthening Design for Testability (DFT) skills.
- Enhanced scripting proficiency in TCL/Python for automating design processes and managing reports.

Project/Management Learnings

- Learned to plan and execute design iterations efficiently, addressing errors step by step.
- Developed experience in collaborative documentation and version control using GitHub.
- Improved time management by completing multiple design cycles within set deadlines.
- Cultivated a problem-solving approach, as each violation demanded detailed debugging and creative solutions

Conclusion and Future Scope

Conclusion

Recap of Objectives and Achievements

Objectives

- Gain a clear understanding of clock terminology and rules for robust design implementation.
- Incorporate PLL logic and black box modules into the design flow.
- Detect and fix DRC violations, with a focus on addressing E5 errors.
- Improve testability by adding scan chain insertion.

Achievements

- Successfully applied clock rules and achieved proper synchronization through PLL integration.
- Modified the design to include black box modules without causing functional issues.
- Identified and resolved all DRC violations, delivering a clean design ready for sign-off.
- Debugged and corrected E5 violations, demonstrating hands-on problem-solving skills.
- Implemented scan chain insertion and validated scan integrity using test patterns.
-
- Maintained systematic documentation of the entire process, aligning with industry-standard VLSI design practices.