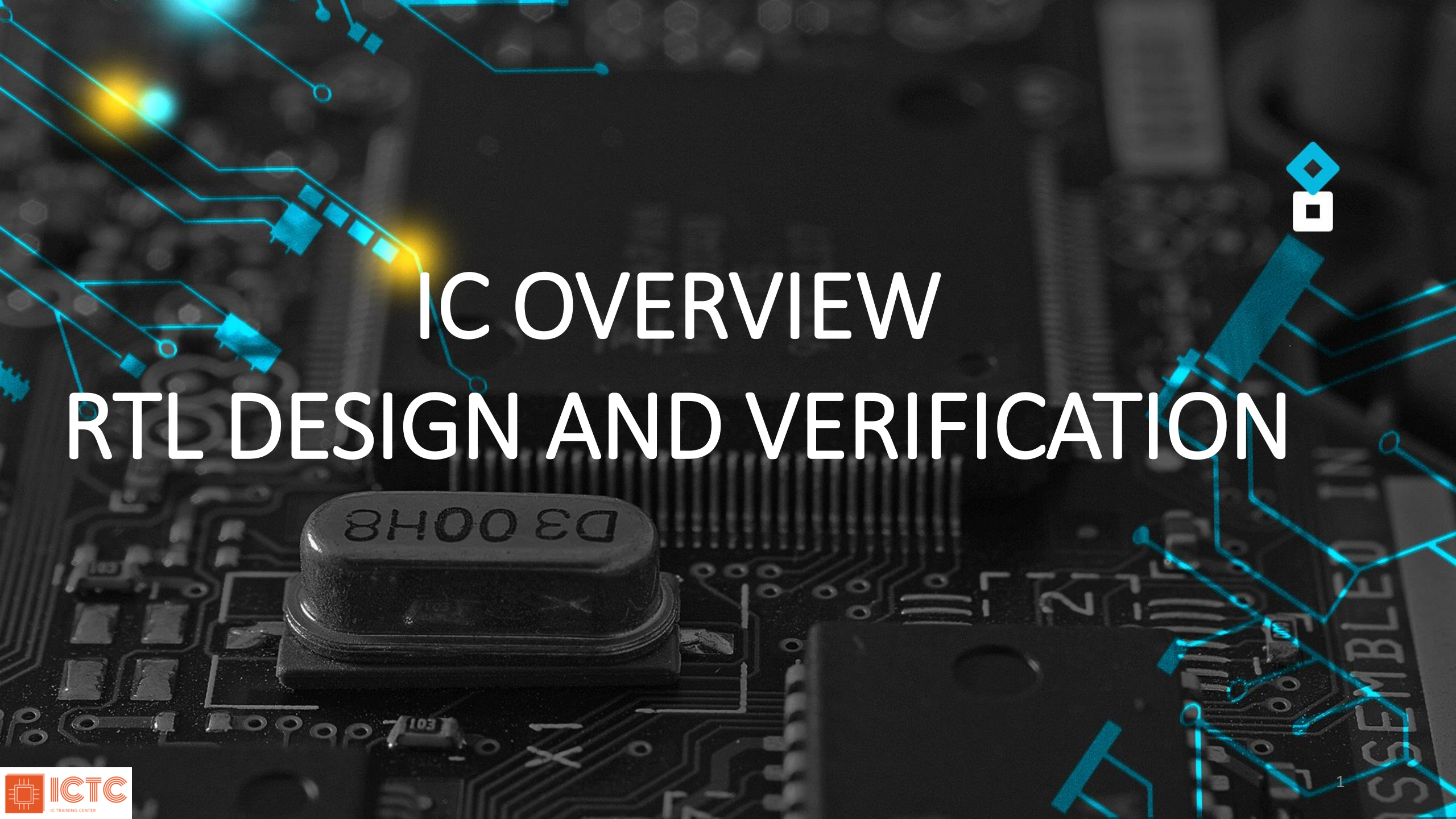


IC OVERVIEW
RTL DESIGN AND VERIFICATION

Session 15: FSM Verification and Code coverage

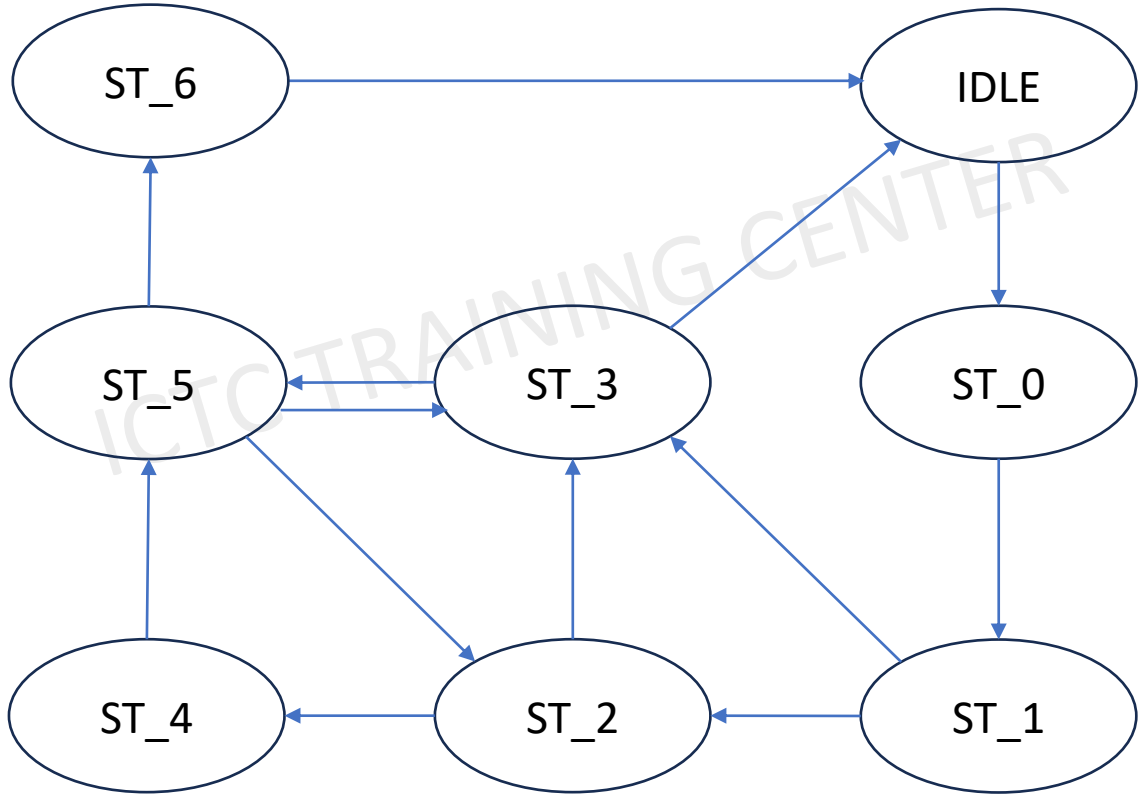
1. FSM verification

2. Code Coverage

FSM VERIFICATION

Complete state verification: Ensure that all states are reachable and testable. Each state should be entered and exited as intended.

Example: all the states in right hand side should be covered in the simulation



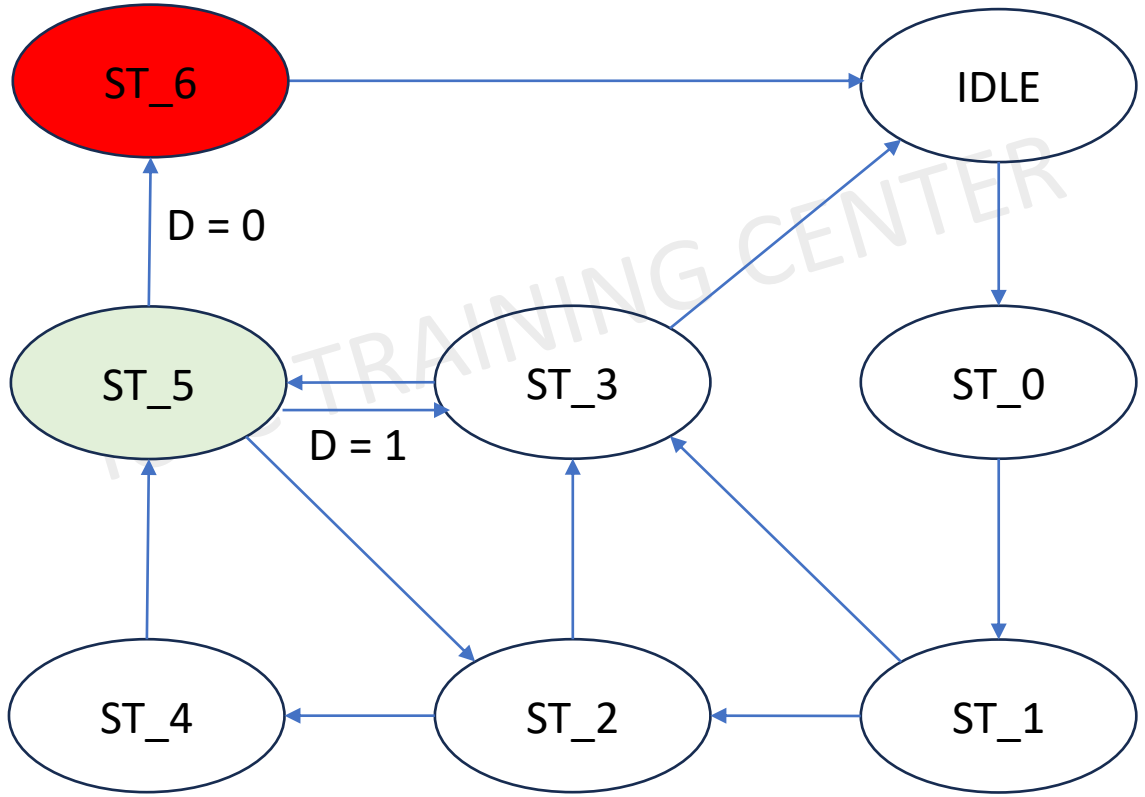
FSM VERIFICATION

Ensure that all states can be entered and exited.

Problem: unreachable states

States that cannot be reached from the initial state under any conditions.

Example: at ST_5, if D is always 1, ST_6 is an unreachable state.



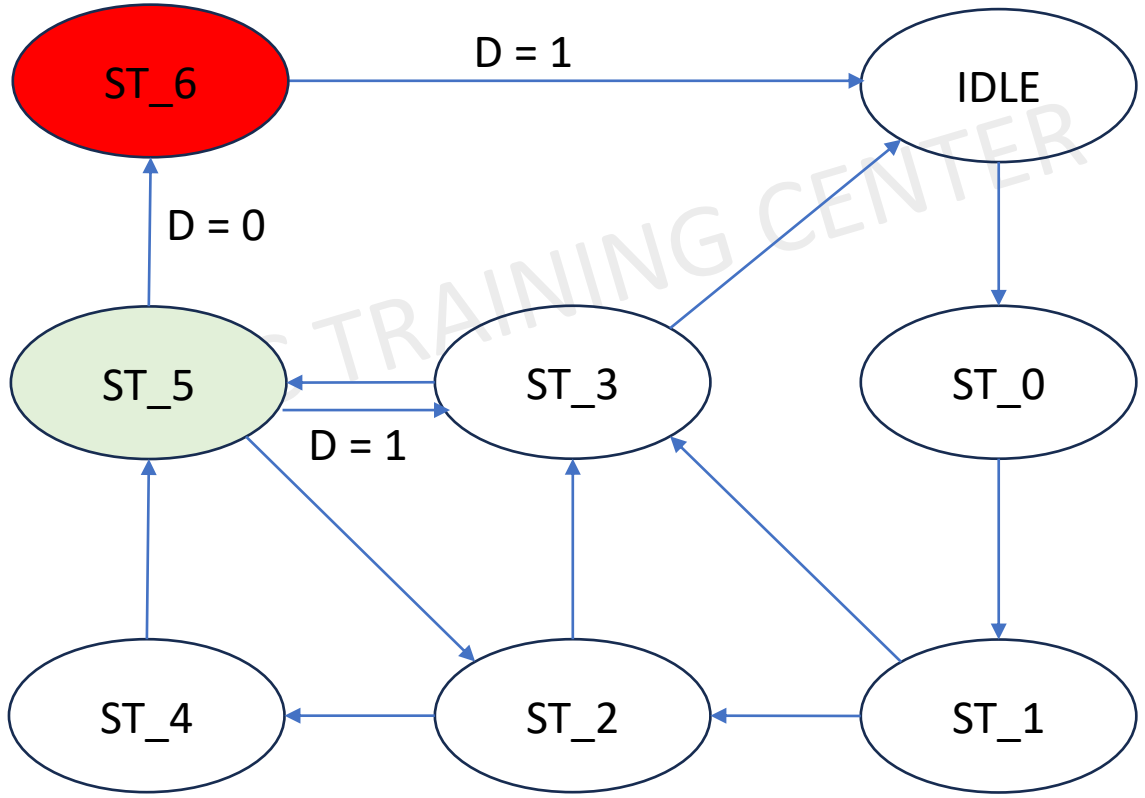
FSM VERIFICATION

Ensure that all states can be entered and exited.

Problem: Dead states

Once entered, can not transition to any other state.

Example: at ST_6, if D is always 0, the state machine is stucked. ST_6 is a dead state.



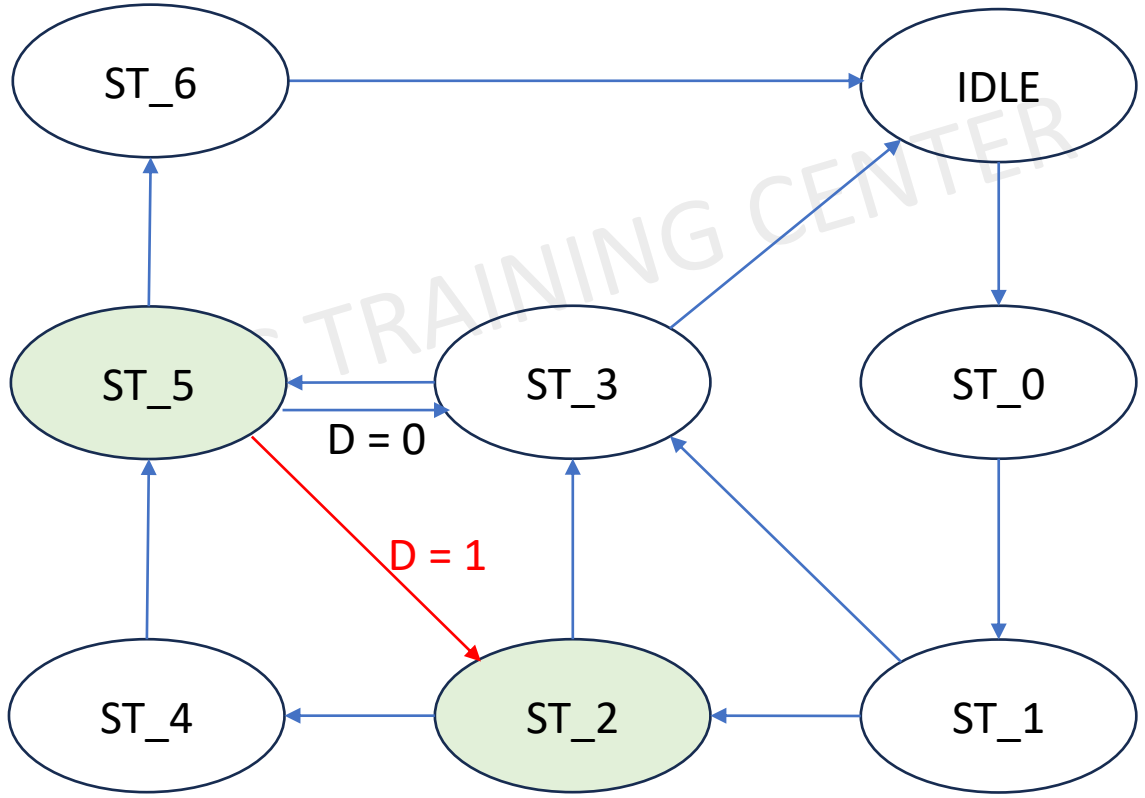
FSM VERIFICATION

Ensure that the conditions triggering state transitions are correct and do not overlap unintentionally!!!

Problem: Transition failure

Can not move do desire states as the specification

Example: as specification, ST_5 can be moved back to ST_2. But at ST_5, D is always 0, then state can not move back to ST_2. So, transition from ST_5 to ST_2 is not guaranteed.



FSM VERIFICATION

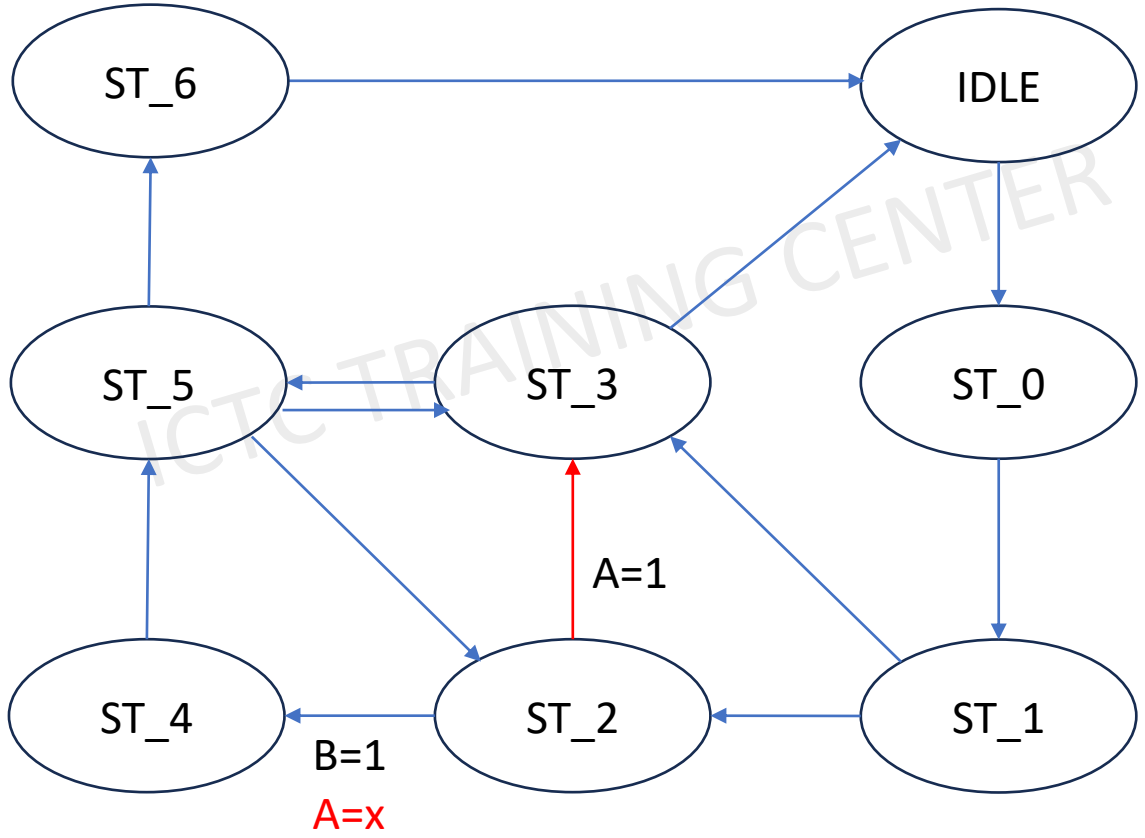
Ensure that all the conditions and combination of state transition are tested!!!

Problem: Wrong transition

The issue often occurs at states that has multiple conditions for transition due to priority conflict.

Example: as ST_2, when B=1, need to change to ST_4 regardless of A.

Failure case: when A=1 & B=1, state moves to ST_3, not ST_4 as the specification.



FSM VERIFICATION

State encoding must be uinique !!!

Problem: Wrong state encoding

State is uniquely encoded and that there is no ambiguity in state representation.

Example: state encoding

IDLE: 4'h00

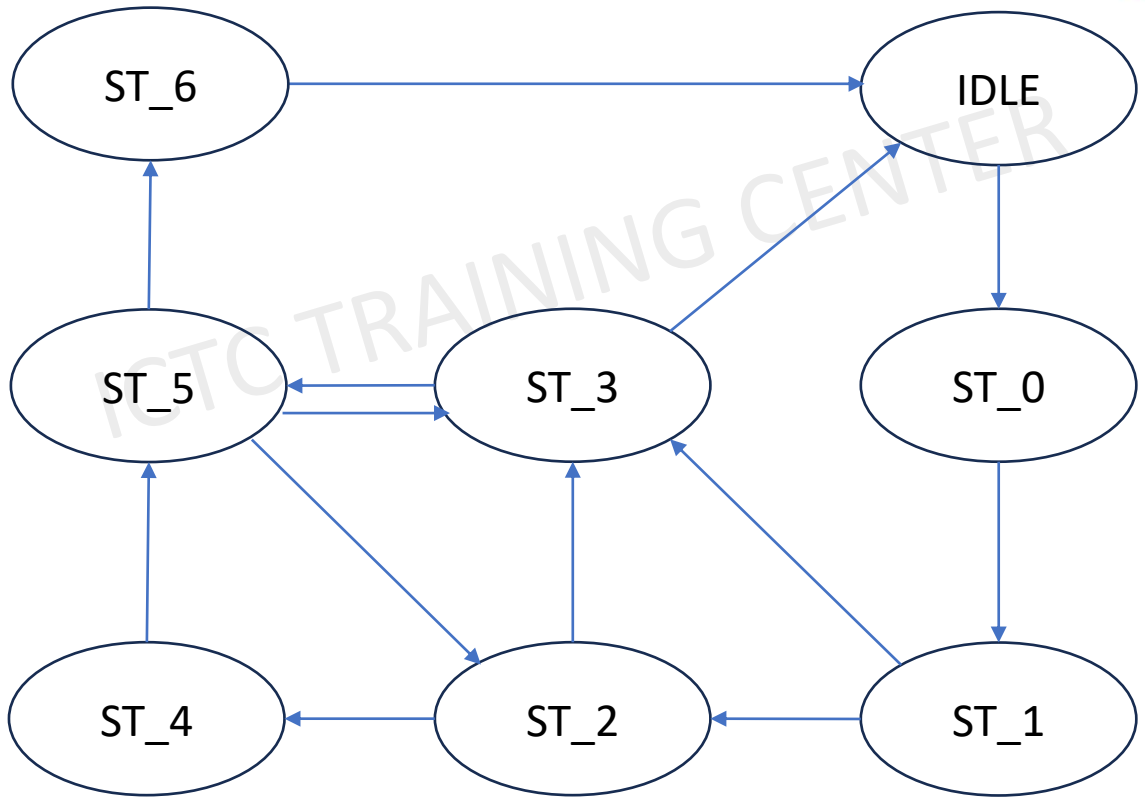
ST_0: 4'h01

ST_1: 4'h02

ST_2: 4'h03

ST_3: 4'h03 ← wrong state encoding

It can lead to moving to wrong state after reaching ST_1.



FSM VERIFICATION

Problem: Wrong operation after reset

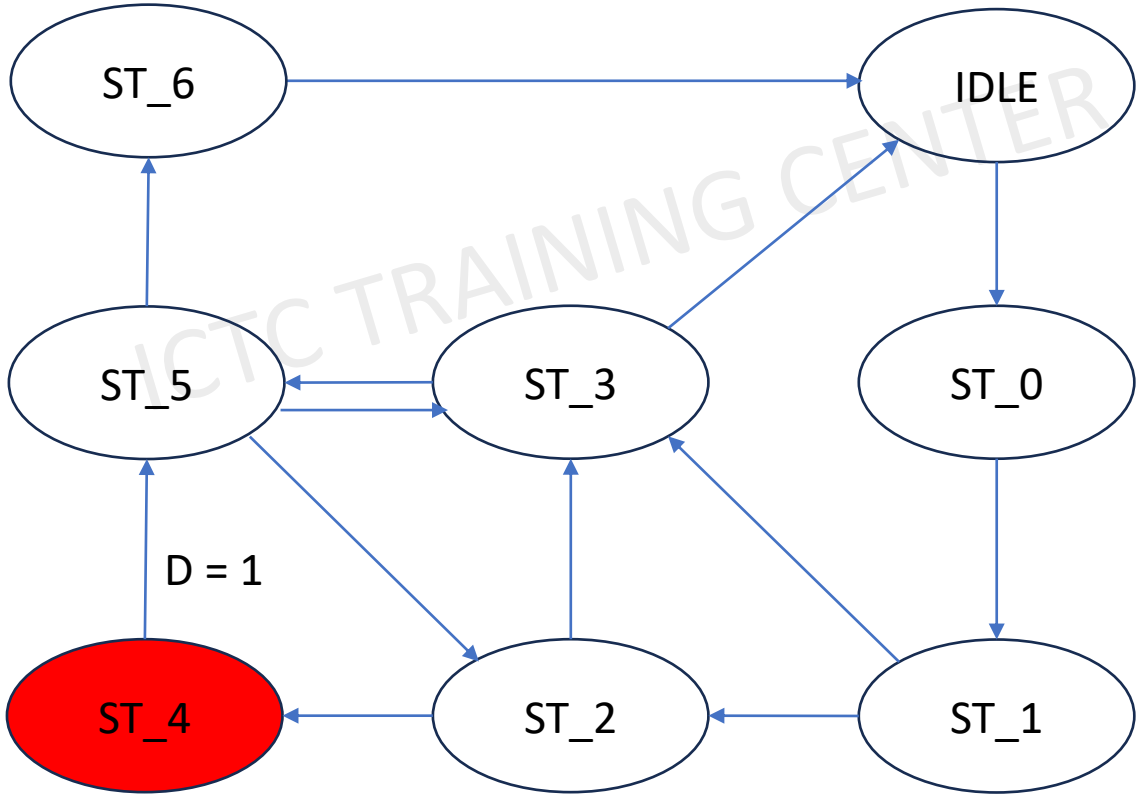
After reset occurs, states are not transit to initial state. This issue occurs at modules that have multiple reset, e.g. Power-ON (POR) reset and System reset.

Example: This FSM is inside module that has POR reset and system reset. This FSM is reset by POR reset only. When system reset occur at ST_4, D (initizlize by system reset) is reset to 0 so state is stucked at ST_4 and can not go back to IDLE state.

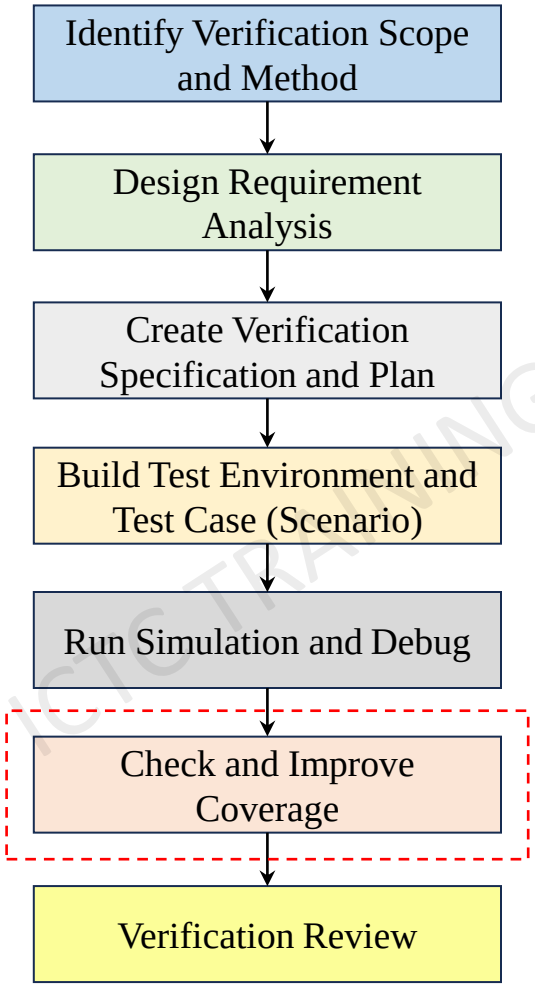
Power-ON reset: is a type of reset signal generated during the power-up of a device. It ensures that the device starts in a known state every time power is applied

System reset: is a broader reset mechanism that can be triggered by various conditions or sources during the normal operation of a system. It ensures the system can be brought back to a known state under specific conditions.

Ensure that the FSM correctly initializes to a known state upon reset !!!



COVERAGE RECAP



Coverage: helps DV engineers identify areas they haven't yet tested. Here are some examples:

- Some lines of code that haven't been tested
- An else-if condition that hasn't been tested
- A logical combination of values that hasn't been tested
- Some states of a state machine that haven't been tested

....

Coverage also helps RTL design engineers identify unreasonable lines of code and conditions that never occur. As a result, RTL engineers can optimize their designs to make them better.

Completing coverage is a MUST to demonstrate that the verification work is completed.

Coverage will help to detect the above issues !!!

CODE COVERAGE

Code coverage is a metric used to measure how much the design source in RTL or gates are tested. Code coverage measures how often a suite or tests exercises certain aspects of the source.

Missing code coverage is usually an indication of 2 things:

- Un-used code, or
- Holes in the verification

We can improve the coverage by:

- Remove the un-used code
- Add testcases to cover the verification holes.

We need to achieve 100% code coverage in the final project!!!

COMMON CODE COVERAGE TYPES

Statements coverage

Some common code coverage types are: statement, branch, expression, condition, toggle.

Statement coverage: sometimes called “line coverage” is the most basic form of code coverage. The EDA tools will count of how many times a given statement is executed during simulation.

```
Statement Coverage:
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Statements                       8         7         1    87.50%

================================Statement Details================================

Statement Coverage for instance /test_bench/u_fsm --

    Line         Item                      Count     Source
    ----         ----                      -----     ------
  File ../rtl/fsm_ctrl.v
    28              1                               next_state = ST_LOCK  ;
```

We need to achieve 100% code coverage in the final project!!!

COMMON CODE COVERAGE TYPES

Branch coverage

Branch coverage: relates to branching constructs such as “if” and “case” statements. It measurestrue branch and false branch execution.

```
module top;
integer i=10;
initial begin
    #3 i = 18;
    #3 i = 2;
    #1 $finish();
end
always @ (i) begin
    if (i == 16)
        $display("sweet");
    else if (i == 2)
        $display("terrible");
    else if (i == 10)
        $display("double digits at last");
    else if (i == 18)
        $display("can vote");
    else
        $display("just another birthday"); end endmodule
```

```
File: top.v
Branch Coverage:
    Enabled Coverage        Bins        Hits    Misses  Coverage
    ----------------        ------      ----    ------ ---------
    Branches                   5           3         2      60.0

==============================Branch Details==============================

Branch Coverage for file top.v --

------------------------------------IF Branch-----------------------------
    12                                          3     Count coming in to IF
    12              1                      ***0***         if (i == 16)
    14              1                           1          else if (i == 2)
    16              1                           1          else if (i == 10)
    18              1                           1          else if (i == 18)
    20              1                      ***0***         else
Branch totals: 3 hits of 5 branches = 60.0%
```

We need to achieve 100% code coverage in the final project!!!

COMMON CODE COVERAGE TYPES

Expression coverage

Expression coverage: analyzes the activity of expressions on the right-hand side of assignment statements, and counts when these expressions are executed.

FEC (Focused Expression Coverage): is a method that measures coverage for each input of an expression. If all inputs are fully covered, the expression has reached 100% coverage.

Example of expression:

```
assign y = a & (b | c);
```
← expression that will be analyzed by FEC

We need to achieve 100% code coverage in the final project!!!

COMMON CODE COVERAGE TYPES

Expression coverage

In FEC, an input is said to be fully covered when the following conditions apply:

- All other inputs are in a state that allow the covered input to control the output of the expression.
- The output can be seen in both 0 and 1 states while the target input is controlling it.

Example:

```
assign y = a & (b | c);
```

For terminal a to be covered, the value of expression (b | c) must be in a non-masking state (that is, 1 for the & operator)

- We need to achieve 100% code coverage in the final project!!!

COMMON CODE COVERAGE TYPES

Expression coverage

The tool will break the big expression into smaller basic expressions and evaluate one by one.

Example the expression:

assign z = a & (b | c);

```
================================Expression Details================================

Expression Coverage for instance /test_bench/u_dut --

  File ../rtl/top.v
----------------Focused Expression View-----------------
Line       8 Item    1  (a & (b | c))
Expression totals: 0 of 3 input terms covered = 0.00%

  Input Term   Covered  Reason for no coverage   Hint
 -----------  --------  ----------------------  -------------
           a         N  '_0' not hit            Hit '_0'
           b         N  '_0' not hit            Hit '_0'
           c         N  No hits                 Hit '_0' and '_1'

     Rows:       Hits  FEC Target            Non-masking condition(s)
 ---------  ---------  --------------------  ------------------------
  Row   1:     ***0***  a_0                   (b | c)
  Row   2:           1  a_1                   (b | c)
  Row   3:     ***0***  b_0                   (a && ~c)
  Row   4:           1  b_1                   (a && ~c)
  Row   5:     ***0***  c_0                   (a && ~b)
  Row   6:     ***0***  c_1                   (a && ~b)
```

Test vector (a,b,c) = (1,1,0)

```
12 ================================Expression Details================================
13
14 Expression Coverage for instance /test_bench/u_dut --
15
16   File ../rtl/top.v
17 ----------------Focused Expression View-----------------
18 Line       8 Item    1  (a & (b | c))
19 Expression totals: 1 of 3 input terms covered = 33.33%
20
21   Input Term   Covered  Reason for no coverage   Hint
22  -----------  --------  ----------------------  -------------
23            b         N  '_0' not hit            Hit '_0'
24            c         N  No hits                 Hit '_0' and '_1'
25
26       Rows:       Hits  FEC Target            Non-masking condition(s)
27  ---------  ---------  --------------------  ------------------------
28    Row   1:          1  a_0                   (b | c)
29    Row   2:          1  a_1                   (b | c)
30    Row   3:     ***0***  b_0                   (a && ~c)
31    Row   4:          1  b_1                   (a && ~c)
32    Row   5:     ***0***  c_0                   (a && ~b)
33    Row   6:     ***0***  c_1                   (a && ~b)
34
```

Test vector (a,b,c) = (1,1,0), (0,1,0)

We need to achieve 100% code coverage in the final project!!!

COMMON CODE COVERAGE TYPES

Condition coverage

Condition coverage: Similar to expression coverage but it analyzes the decision made in "if" and ternary statements (condition ? true_value : false value).

```
//current state logic
assign cnt_pre = cnt_clr ? 8'h0:
                 en1 | en2  ? cnt + 1'b1:
                 cnt;
-----------------------------------
always @* begin
   if( a & b)
      y = 1;
   else
     y = 0;
end
```

Target of condition coverage

We need to achieve 100% code coverage in the final project!!!

COMMON CODE COVERAGE TYPES

Toggle coverage

Toggle coverage: to measure how many times each bit of a signal has changed (or toggled) during simulation.

Data types which are target of toggle coverage calculation: wire, reg, bit, enum, real, shortreal and integer.

```
37 ================================================================================
38 === Instance: /test_bench
39 === Design Unit: work.test_bench
40 ================================================================================
41 Toggle Coverage:
42     Enabled Coverage              Bins      Hits    Misses  Coverage
43     ----------------              ----      ----    ------  --------
44     Toggles                          8         5         3    62.50%
45
46 ================================Toggle Details================================
47
48 Toggle Coverage for instance /test_bench --
49
50                                          Node      1H->0L      0L->1H  "Coverage"
51                                          ----------------------------------------
52                                             a           1           1      100.00
53                                             b           0           1       50.00
54                                             c           0           0        0.00
55                                             z           1           1      100.00
56
57 Total Node Count     =          4
58 Toggled Node Count   =          2
59 Untoggled Node Count =          2
60
61 Toggle Coverage      =      62.50% (5 of 8 bins)
```

Note: the Questa Sim does not monitor toggle coverage for ports of target instance. It only monitor toggles on signals.

But we can monitor the related port toggle on the top level.

The example on the left shows toggle report in testbench, which is top of dut.

Test vector (a,b,c) = (1,1,0), (0,1,0)

We need to achieve 100% code coverage in the final project!!!

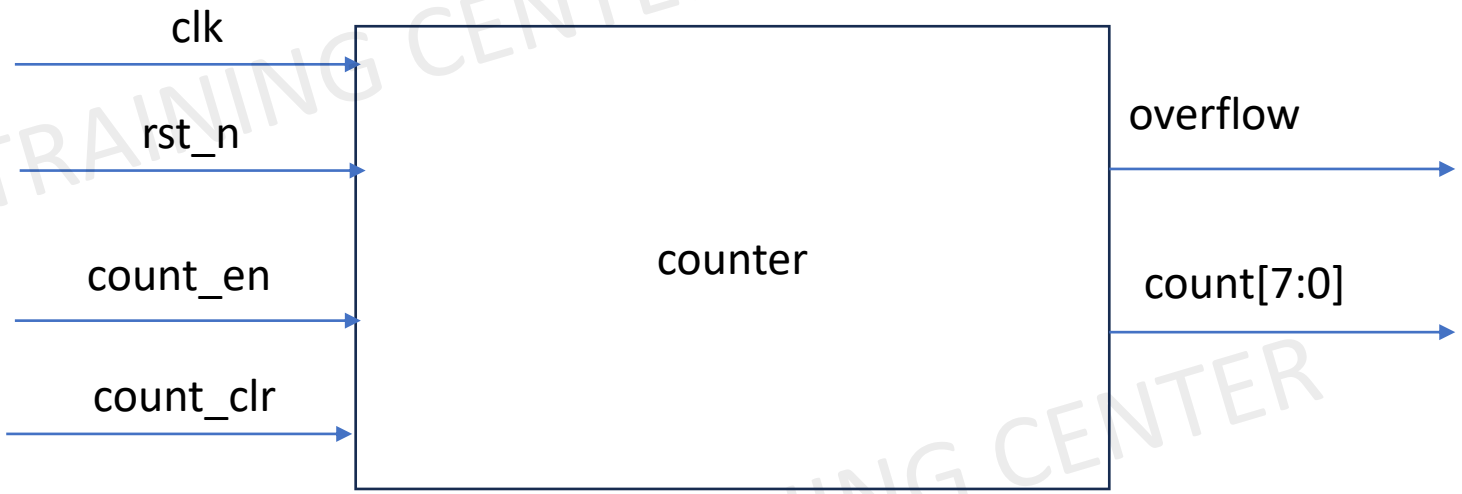
PRACTICE

1. Create 15_ss15 in your home directory
2. Copy share/teacher/15_ss15/cov_practice into your directory. The design contains only 1 expression as the previous examples: y = a & (b | c);
3. Start writing test case in home directory, write first vector (a,b,c) = (1,1,0).
4. % make help to see the new options
5. % make all_cov to build and run simulation in coverage mode. Simulation should be PASSED and test.ucdb is generated.
6. % make gen_cov to generate coverage report. The report is in **coverage** folder.
7. Read and analyze the summary and detail report
8. Improve the coverage by adding testcase one by one and see the differences in the coverage report (repeat step 3 until coverage of dut is 100%).
9. % make gen_html to generate coverage report in html format. Open the index.html in **covhtmlreport** folder report to see the coverage.
10. Apply this method for one another design to see other coverage metrics.

We need to achieve 100% code coverage in the final project!!!

HOMEWORK

Homework (standard): Let's make the coverage for the counter_hw (in 09_ss9) become 100%
Copy counter_hw folder to your 15_ss15 folder and check coverage there.



The advanced level points for session 15 is the presentation points about APB protocol.