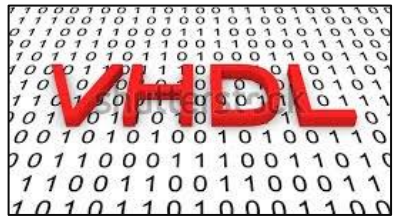
Code Coverage in Verification

EEL 4712

HDL vs. SystemVerilog

Limitations of HDLs (VHDL, Verilog)

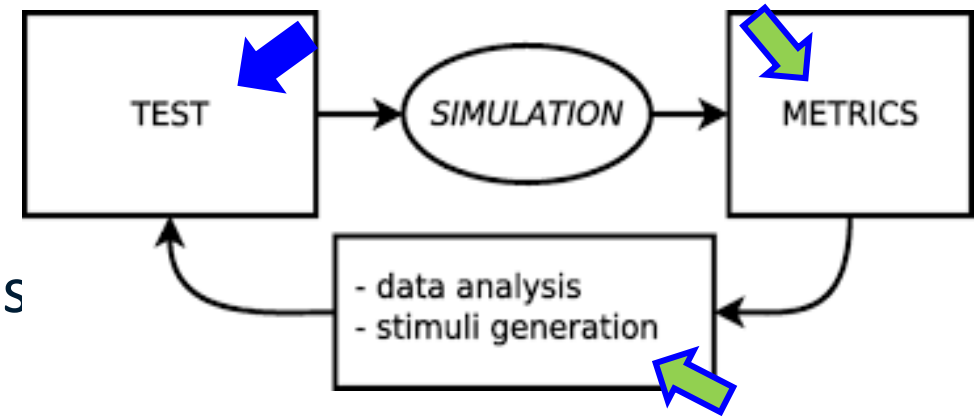
- Limited support for *abstraction* and *reusability*
  - *Low-level* abstraction (e.g., gate-level and register-transfer): adequate for digital designs and systems of modest complexity
- Limited support for *design verification*
  - VHDL has ASSERT statement



Key desirable features of SystemVerilog

- Enhanced *abstraction*
  - For design and *reuse* (e.g., classes, objects, encapsulation)
- Robust *verification capabilities*
  - Assertation, constrained random stimuli, functional coverage
- *Industry adoption*

Design Verification: Testing and Coverage



Directed Testing, Constrained-Random Testing

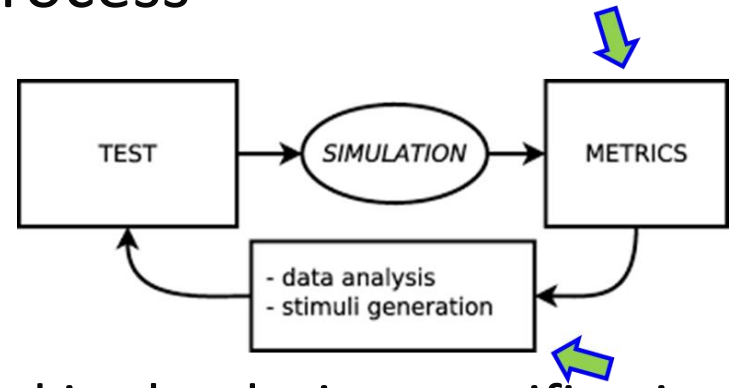
- **Directed tests:** Explicitly define inputs to test specific scenarios to verify certain design features
- **Random, constrained-random tests**:
  - **Random:** generate a large number of tests with random inputs
  - **Constrained-random:** random, but within user-defined constraints to uncover corner cases and bugs that might be missed by directed or completely random tests.

Coverage:

- a *simulation metric* used to measure the *progress* and *completeness* (quality) of the verification process
  - To provide feedback to improve test

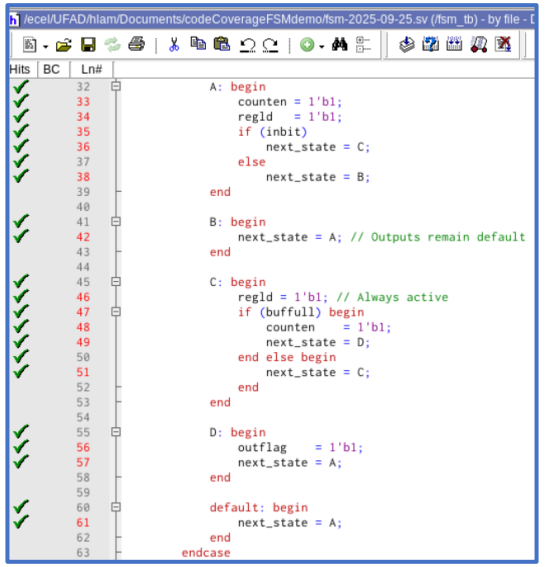
Types of Coverage

- Coverage: a *simulation metric* used to measure the *progress* and *completeness* (quality) of the verification process
- Major types of coverage methods:
  - Functional coverage
  - Code coverage



- **Functional** coverage:
  - Measures how much of the functionality defined in the design specification have been exercised during the simulation
  - Specified by the designer in the testbench
- **Code** coverage:
  - Measures how much of the **design code *have been exercised*** during simulation
    - Helps identify **untested parts of the design**, reduces the risk of hidden bugs
  - Collected automatically by the simulation tool (e.g., Questa)

Code Coverage Types



- **Code coverage** is collected automatically by the simulation tool (e.g., Questa)
- In the Questa transcript window, type in the following command:

  vopt +cover=sbecxf ...

  Each **flag** specifies which code coverage type has been enabled.

s (**s**tatement) tracks which statements in your design code are executed

b (**b**ranch c) if/case statements has been taken both true and false path

e (**e**xpression) logical expressions as a whole evaluated to both true and false

c (**c**onditional) each individual boolean sub-expression inside a decision expression

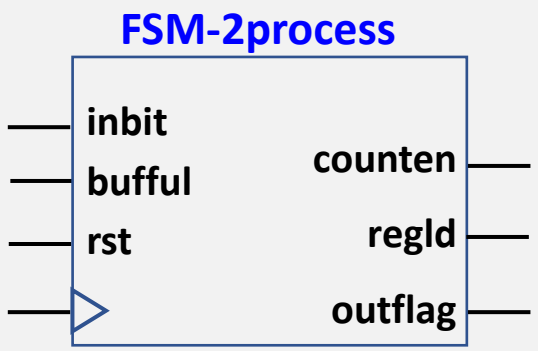
x (e**x**tended **toggle**) tracks signal bits toggling (0→1 and 1→0)

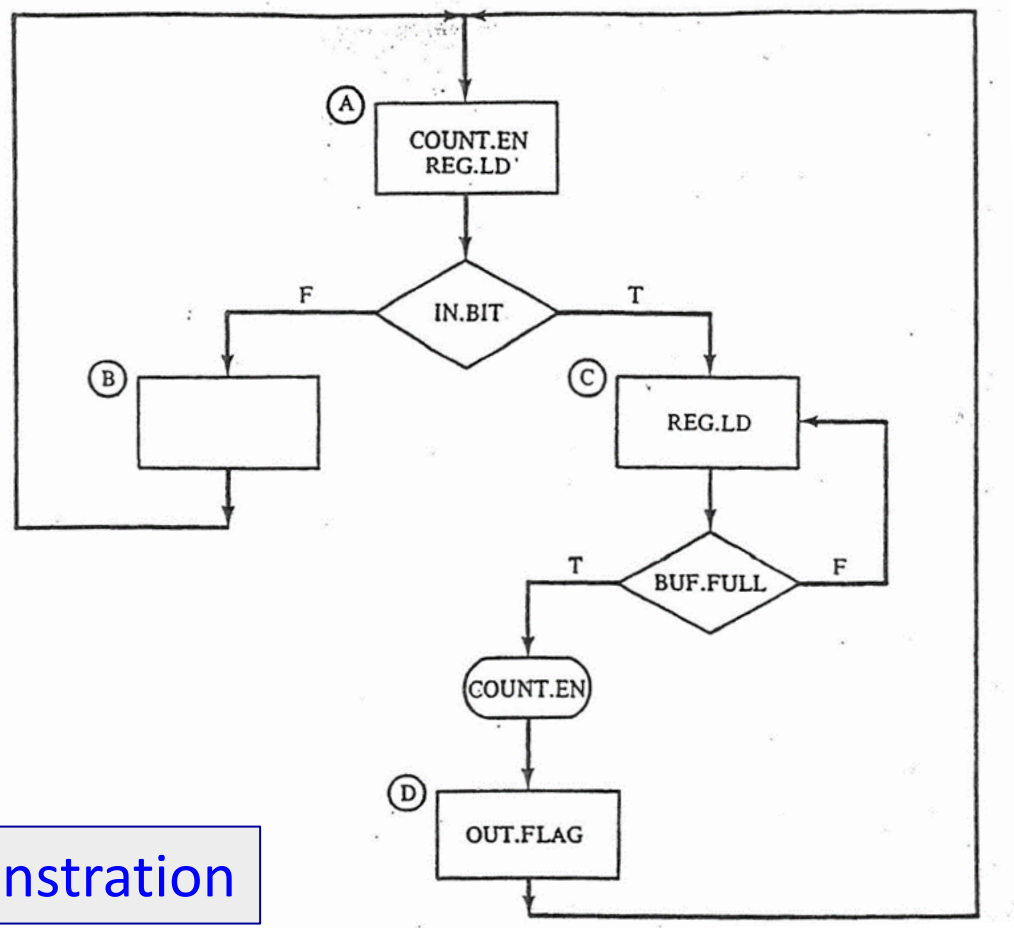
f (**f**sm coverage) tracks state and transition coverage for finite state machines

Example of Code coverage

Design file: fsm.sv (link to fsm.sv code)

```
module FSM_2process (
    input  logic clk,
    I nput  logic rst,
    input  logic inbit,
    input  logic buffull,
    output logic counten,
    output logic regld,
    output logic outflag );
    typedef enum logic [1:0] {    // Enumerated type
        A, B, C, D   // 4  values representing the 4 states
    } state_type;      // Name of the new signal type
```





Testbench files:

- link to fsm_tb.sv
- link to fsm_tb_modified.sv

Demonstration

Summary of Commands

vopt +cover=bcesxf fsm_tb -o fsm_tb_opt

// where fsm_tb is the **module** name of your testbench file (not file name)

// Optimize tb design, instrument it for coverage; collect the coverage info specified by flags

vsim -coverage fsm_tb_opt

// starts simulation, with code coverage enabled

add wave *

run -all

To view results and generate a summary report

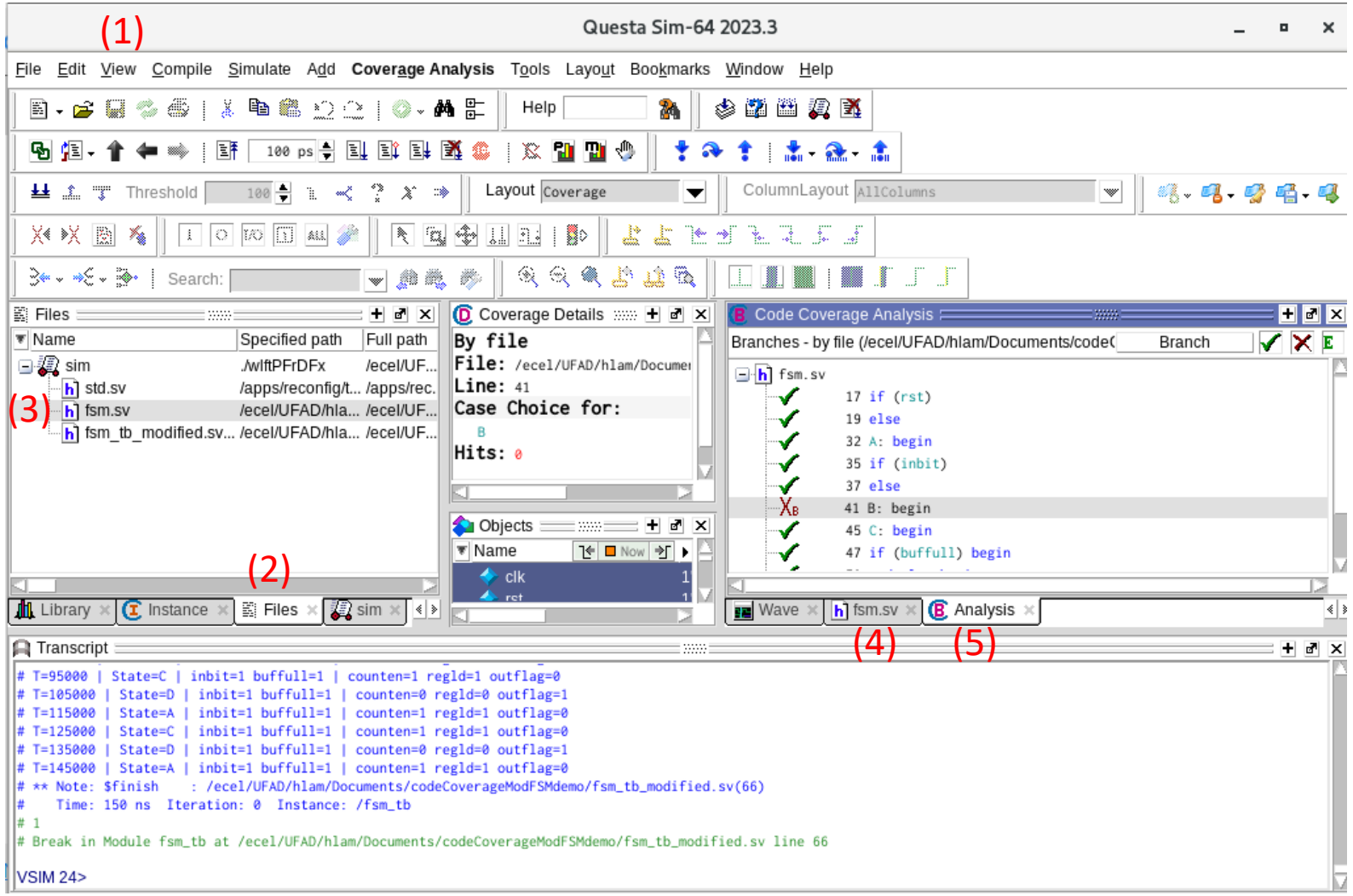
- View → Coverage → Code Coverage Analysis
- Command: coverage report –summary

```
QuestaSim> vopt +cover=sbxf fsm_tb -o fsm_tb_opt
# QuestaSim-64 vopt 2023.3 Compiler 2023.07 Jul 17 2023
# Start time: 15:02:49 on Nov 13,2025
# vopt -reportprogress 300 "+cover=sbxf" fsm_tb -o fsm_tb_opt
#
# Top level modules:
#       fsm_tb
#
# Analyzing design...
# -- Loading module fsm_tb
# -- Loading module fsm
# Optimizing 2 design-units (inlining 1/2 module instances):
# -- Inlining module fsm(fast)
# -- Optimizing module fsm_tb(fast)
# ** Note: (vopt-143) Recognized 1 FSM in module "fsm(fast)".
# Optimized design name is fsm_tb_opt
# End time: 15:02:50 on Nov 13,2025, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
QuestaSim> vsim -coverage fsm_tb_opt
# vsim -coverage fsm_tb_opt
# Start time: 15:02:59 on Nov 13,2025
# Loading sv_std.std
# Loading work.fsm_tb(fast)
VSIM 31> add wave *
VSIM 32> run -all
# T=0 | State=A | inbit=0 buffull=0 | counten=1 regld=1 outflag=0
# T=10000 | State=A | inbit=1 buffull=0 | counten=1 regld=1 outflag=0
# T=15000 | State=C | inbit=1 buffull=0 | counten=0 regld=1 outflag=0
# T=70000 | State=C | inbit=1 buffull=1 | counten=1 regld=1 outflag=0
# T=75000 | State=D | inbit=1 buffull=1 | counten=0 regld=0 outflag=1
```

Transcript

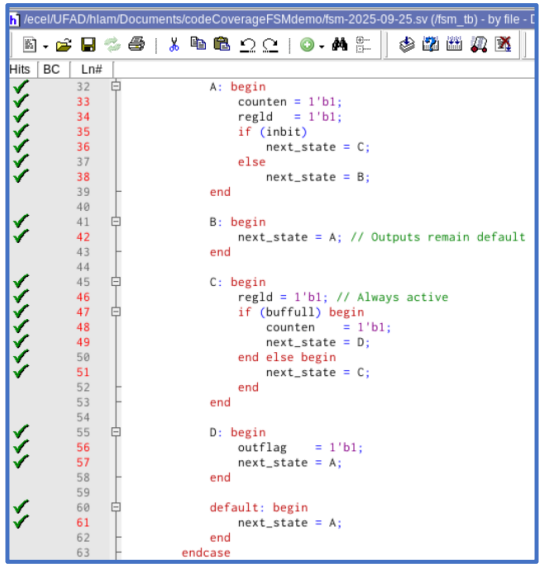
```
#      Enabled Coverage              Bins      Hits    Misses    Weight  Coverage
#      ----------------              ----      ----    ------    ------  --------
#      Branches                        11        10         1         1    90.90%
#      FSM States                       4         3         1         1    75.00%
#      FSM Transitions                  6         3         3         1    50.00%
#      Statements                      40        39         1         1    97.50%
#      Toggles                         48        23        25         1    47.91%
# Total coverage (filtered view): 72.26%
```

After the “run -all” Command



- Click on (1) **View -> Files** to get the Files window (2)
- In the Files window, double-click on fsm.sv (3) to open up the fsm.sv window (4)
- Assuming you have already clicked on (1) **View→Coverage->Code Coverage Analysis** to get the coverage Analysis window (5), you will now see the information in the following slides.

Code Coverage Types



- **Code coverage** is collected automatically by the simulation tool (e.g., Questa)
- In the Questa transcript window, type in the following command:

  vopt +cover=sbecxf ...

  Each **flag** specifies which code coverage type has been enabled.

s (**s**tatement) tracks which statements in your design code are executed

b (**b**ranch c) if/case statements has been taken both true and false path

e (**e**xpression) logical expressions as a whole evaluated to both true and false

c (**c**onditional) each individual boolean sub-expression inside a decision expression

x (e**x**tended **toggle**) tracks signal bits toggling (0→1 and 1→0)

f (**f**sm coverage) tracks state and transition coverage for finite state machines

Statement Coverage (s flag)

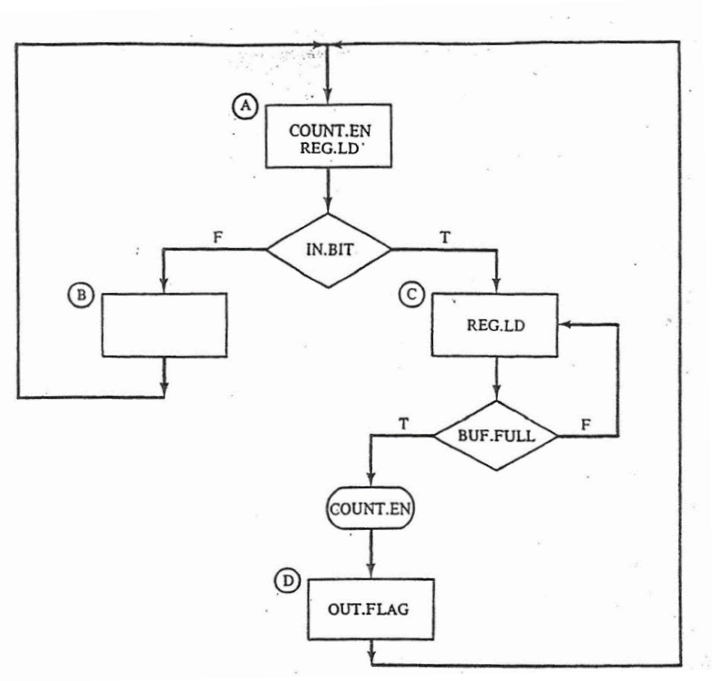
s (**s**tatement) coverage: checks if each executable statement in ***design file*** is executed at least once during simulation.

- Highlighting untested code paths

Code coverage summary table*

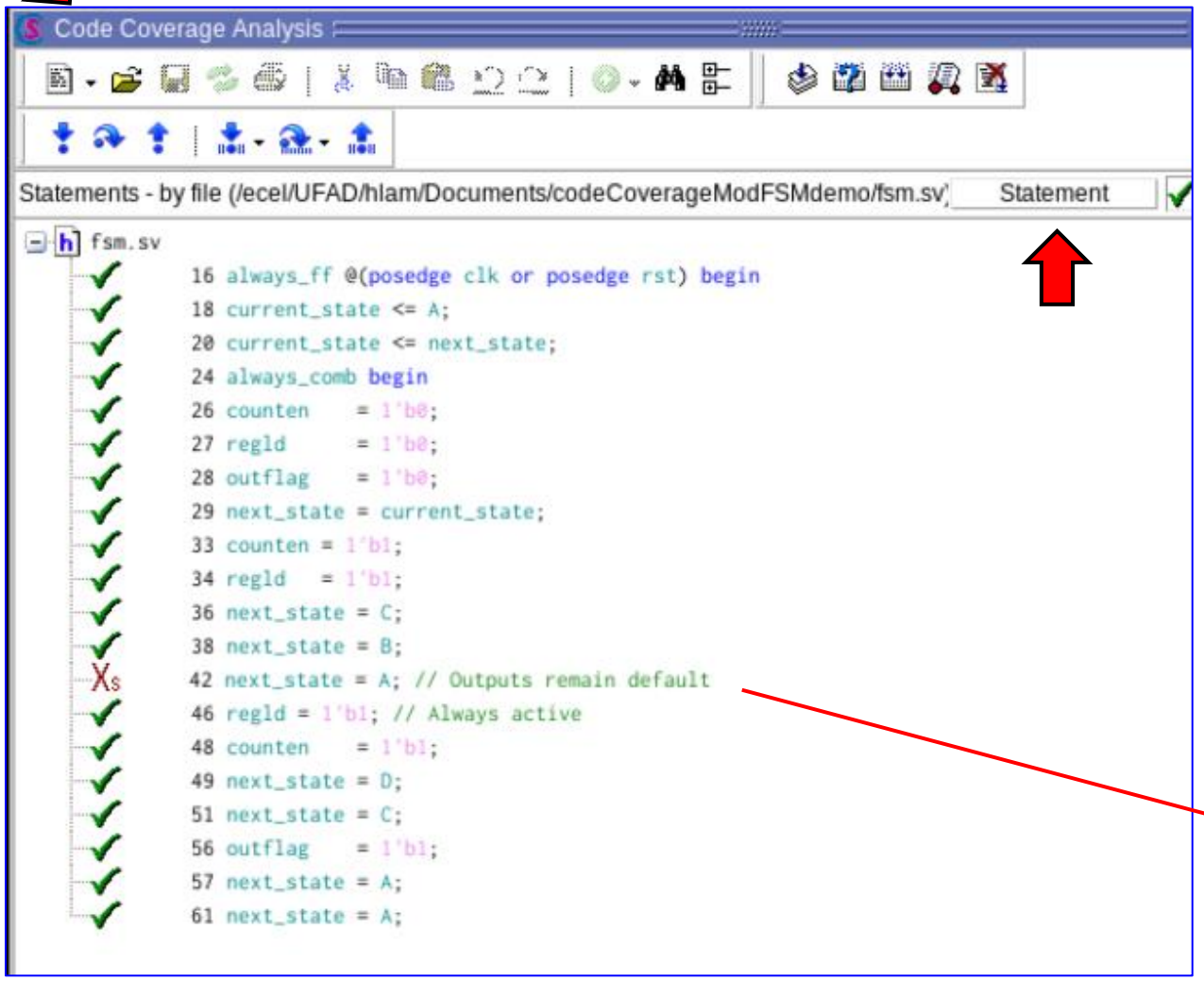
Transcript

```
#      Enabled Coverage              Bins      Hits    Misses     Weight  Coverage
#      ----------------              ----      ----    ------     ------  --------
#      Branches                        11        10         1          1    90.90%
#      FSM States                       4         3         1          1    75.00%
#      FSM Transitions                  6         3         3          1    50.00%
#      Statements                      40        39         1          1    97.50%
#      Toggles                         48        23        25          1    47.91%
# Total coverage (filtered view): 72.26%
```

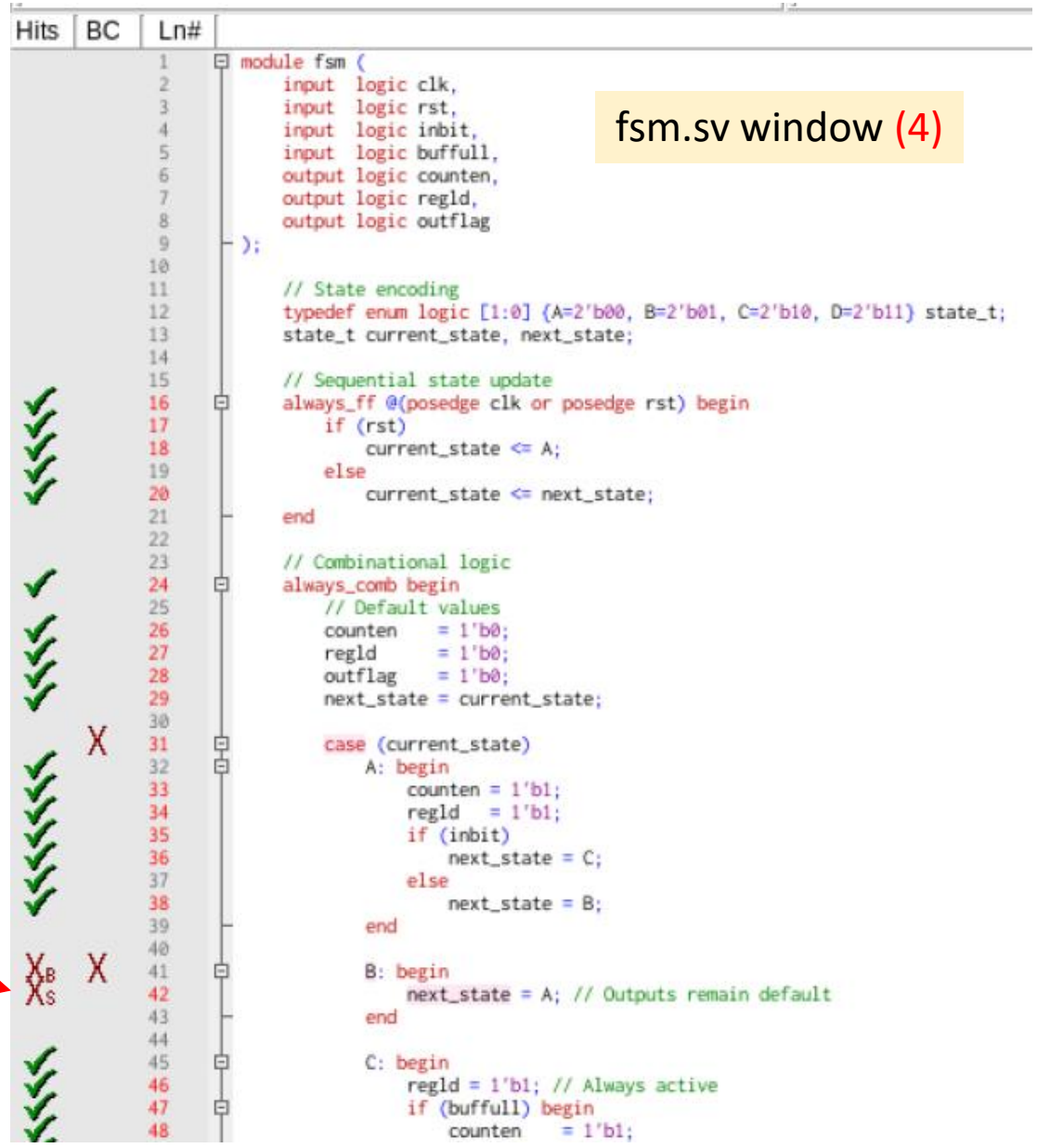


* Obtained by typing the following command in the transcript window after the run -all command: **coverage report –summary**

Statement Coverage (s flag)



fsm.sv window (4)



Xs: Statement not executed (Hits column)

Questa Icon Description

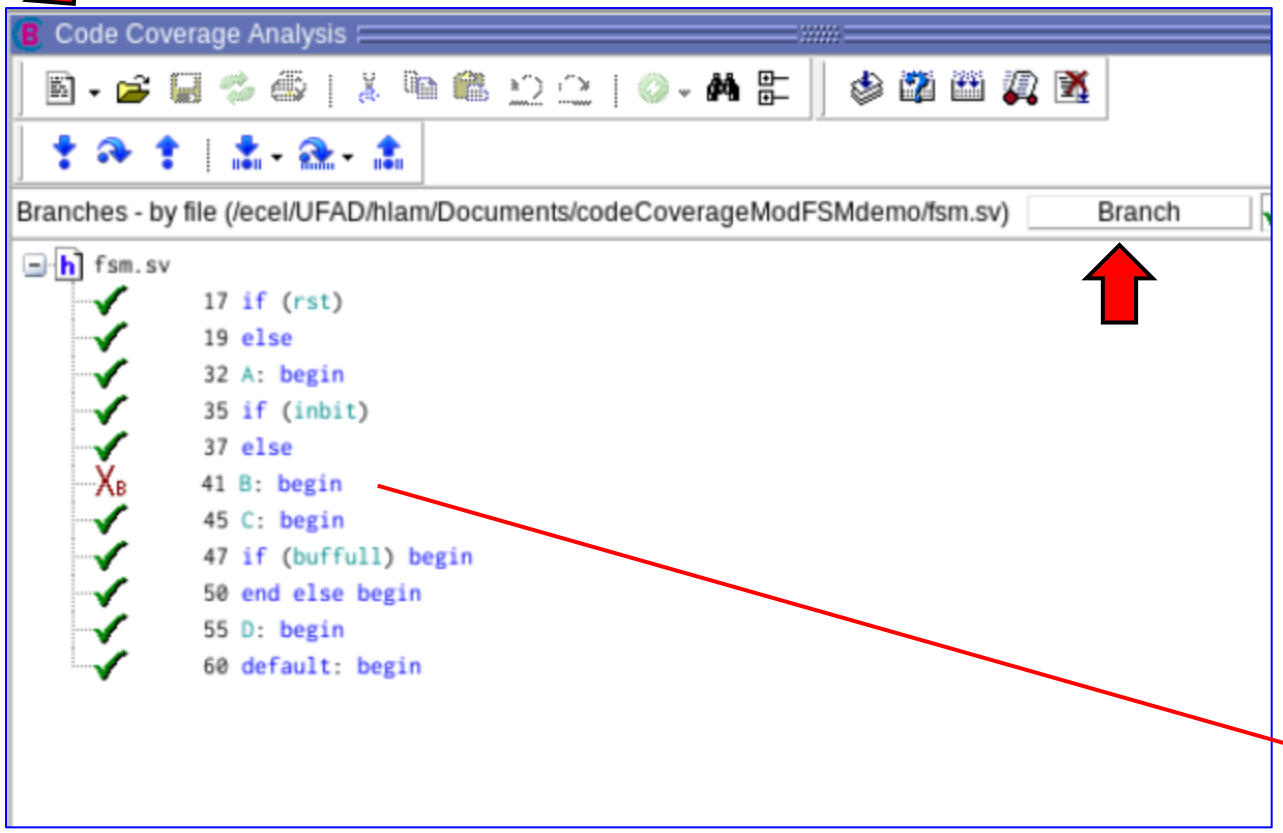
Icon	Description/Indication
✔	All statements, branches, conditions, or expressions on a particular line have been executed
X	Multiple kinds of coverage on the line were not executed
X_T	True branch not executed (BC column)
X_F	False branch not executed (BC column)
X_C	Condition not executed (Hits column)
X_E	Expression not executed (Hits column)

Icon	Description/Indication
X_B	Branch not executed (Hits column)
X_S	Statement not executed (Hits column)
E	Indicates a line of code to which active coverage exclusions have been applied. Every item on the line is excluded; none are hit.
E	Some excluded items are hit
E	Some items are excluded, and all items not excluded are hit
E	Some items are excluded, and some items not excluded have missing coverage
E	Auto exclusions have been applied to this line. Hover the cursor over the EA and a tool tip balloon appears with the reason for exclusion,

Reference: Questa® SIM Tutorial

http://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/swdocs/questasim/questa_sim_tut_2024_2.pdf

Branch Coverage (B flag)



fsm.sv window (4)

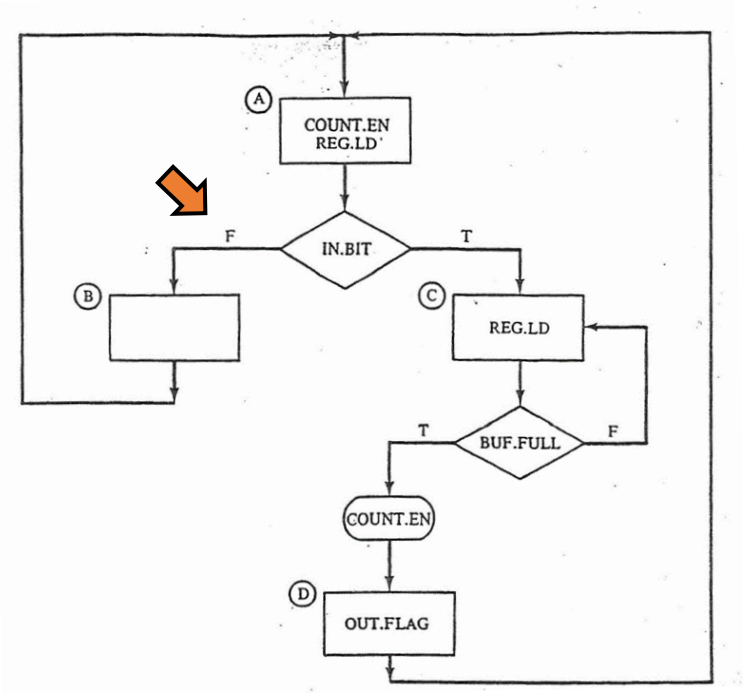
```
Hits BC Ln#
     1  module fsm (
     2      input  logic clk,
     3      input  logic rst,
     4      input  logic inbit,
     5      input  logic buffull,
     6      output logic counten,
     7      output logic regld,
     8      output logic outflag
     9  );
    10
    11      // State encoding
    12      typedef enum logic [1:0] {A=2'b00, B=2'b01, C=2'b10, D=2'b11} state_t;
    13      state_t current_state, next_state;
    14
    15      // Sequential state update
✓   16      always_ff @(posedge clk or posedge rst) begin
✓   17          if (rst)
✓   18              current_state <= A;
✓   19          else
✓   20              current_state <= next_state;
    21      end
    22
    23      // Combinational logic
✓   24      always_comb begin
    25          // Default values
✓   26          counten    = 1'b0;
✓   27          regld      = 1'b0;
✓   28          outflag    = 1'b0;
✓   29          next_state = current_state;
    30
X   31          case (current_state)
✓   32              A: begin
✓   33                  counten = 1'b1;
✓   34                  regld   = 1'b1;
✓   35                  if (inbit)
✓   36                      next_state = C;
✓   37                  else
✓   38                      next_state = B;
    39              end
    40
XB X 41             B: begin
XS  42                  next_state = A; // Outputs remain default
    43              end
    44
✓   45              C: begin
✓   46                  regld = 1'b1; // Always active
✓   47                  if (buffull) begin
✓   48                      counten    = 1'b1;
```

X_B: Branch not executed (Hits column)

Branch Coverage (b flag)

b (**b**ranch c) if/case statements has been taken both true and false path

- After rst became '0', inbit was never set to '0' in this testbench.



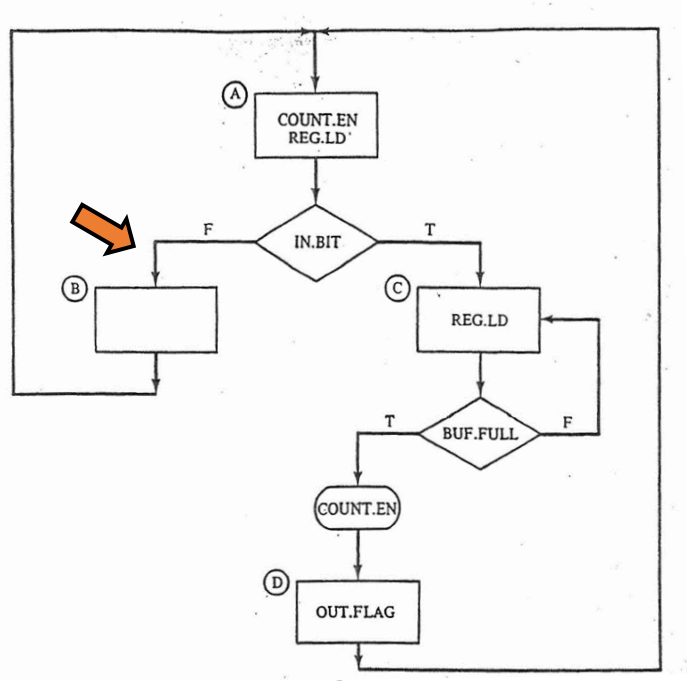
fsm.sv window (4)

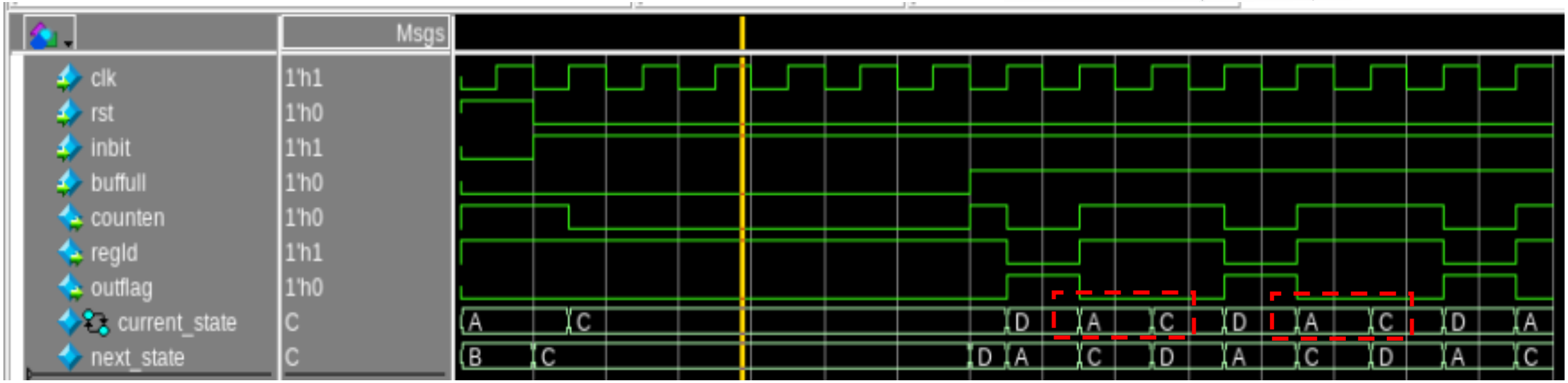
```
Hits | BC | Ln#
     26    );
     27
✓    28    // Clock generation
     29    always #5 clk = ~clk;
     30
     31    // Stimulus
     32    initial begin
✓    33      // Init
✓    34      clk = 0; rst = 1;
✓    35      inbit = 0; buffull = 0;
     36      #10 rst = 0;
     37
     38      // --- Test path: A -> B -> A ---
     39     //inbit = 0;  For demo, commented out branch to State B
     40      //#20;
     41
✓    42      // --- Test path: A -> C (inbit=1) ---
✓    43      inbit = 1;
     44      #20;
     45
✓    46      // --- Stay in C while buffull=0 ---
✓    47      buffull = 0;
     48      #40;
     49
✓    50      // --- Transition C -> D when buffull=1 ---
✓    51      buffull = 1;
     52      #20;
     53
✓    54      // --- D -> A ---
     55      #20;
     56
     57      // --- Another cycle: A -> B again ---
     58      //inbit = 0;  For demo, commented out branch to State B
     59      //#20;
     60
✓    61      // --- Another cycle: A -> C -> D ---
✓    62      inbit = 1;
```

Branch Coverage (b flag)

b (**b**ranch c) if/case statements has been taken both true and false path

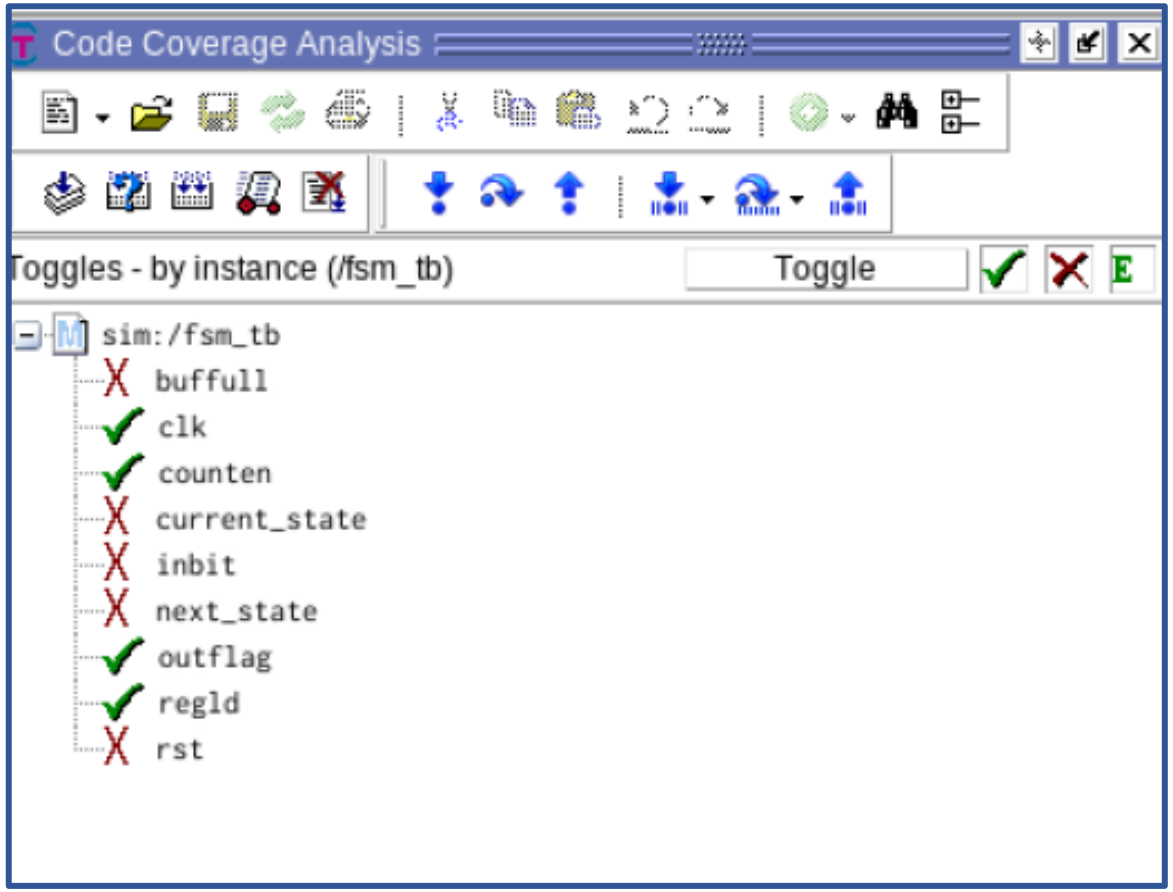
- After rst became ‘1’, inbit was never set to ‘0’ in this testbench.





Toggle Coverage (x flag)

x (e**x**tended **toggle**) tracks signal bits toggling (0→1 and 1→0)



Testbench files:

- link to fsm_tb.sv
- link to fsm_tb_modified.sv

Toggle Coverage (x flag)

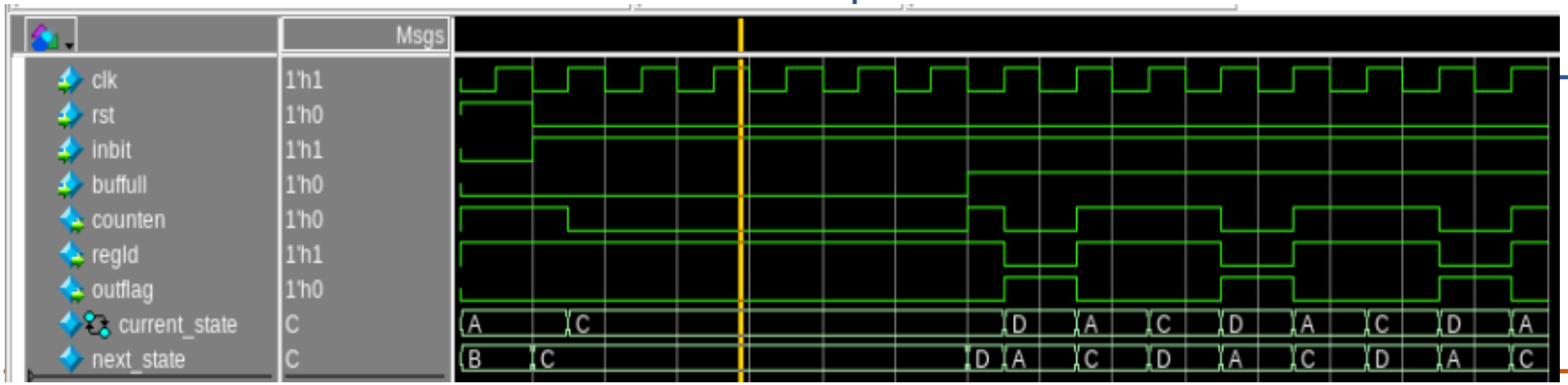
Transcript

```
#      Enabled Coverage              Bins      Hits    Misses     Weight  Coverage
#      ----------------              ----      ----    ------     ------  --------
#      Branches                        11        10         1          1    90.90%
#      FSM States                       4         3         1          1    75.00%
#      FSM Transitions                  6         3         3          1    50.00%
#      Statements                      40        39         1          1    97.50%
#      Toggles                         48        23        25          1    47.91%
# Total coverage (filtered view): 72.26%
```

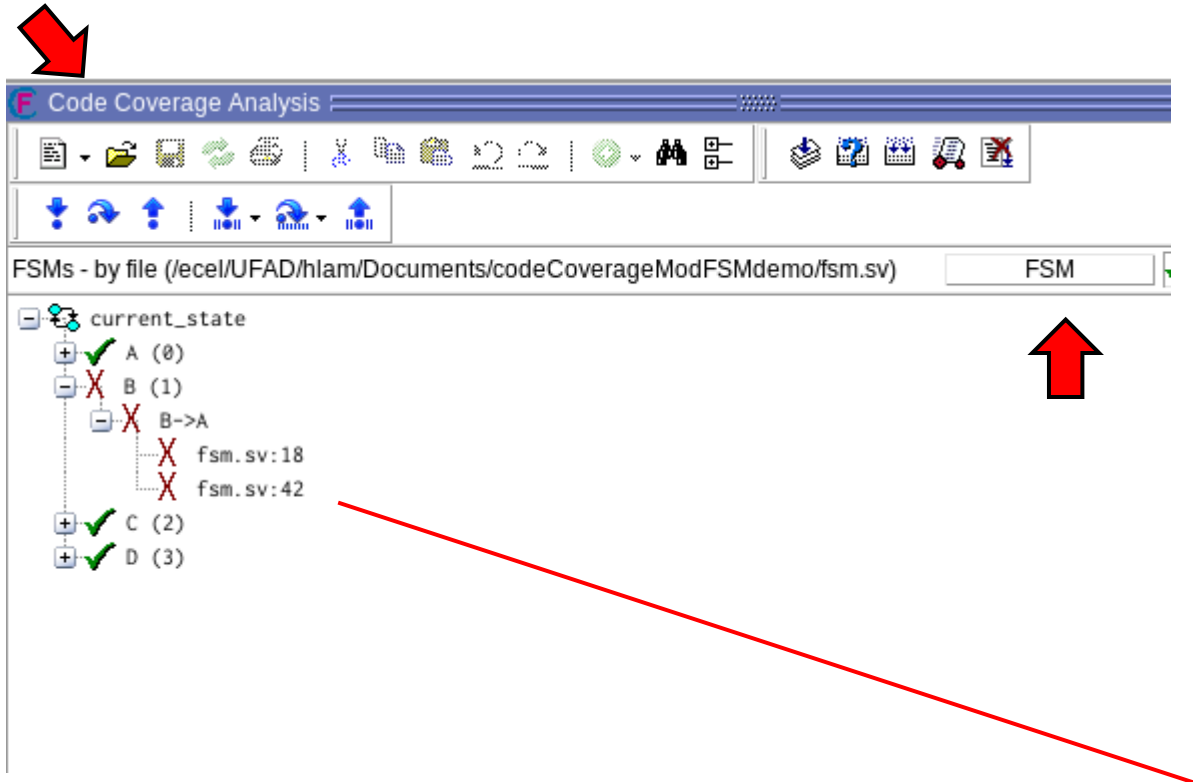
Code Coverage Analysis

Toggles - by instance (/fsm_tb) Toggle E

- sim:/fsm_tb
  - buffull
  - clk
  - counten
  - current_state
  - inbit
  - next_state
  - outflag
  - regld
  - rst



FSM Coverage (f flag)



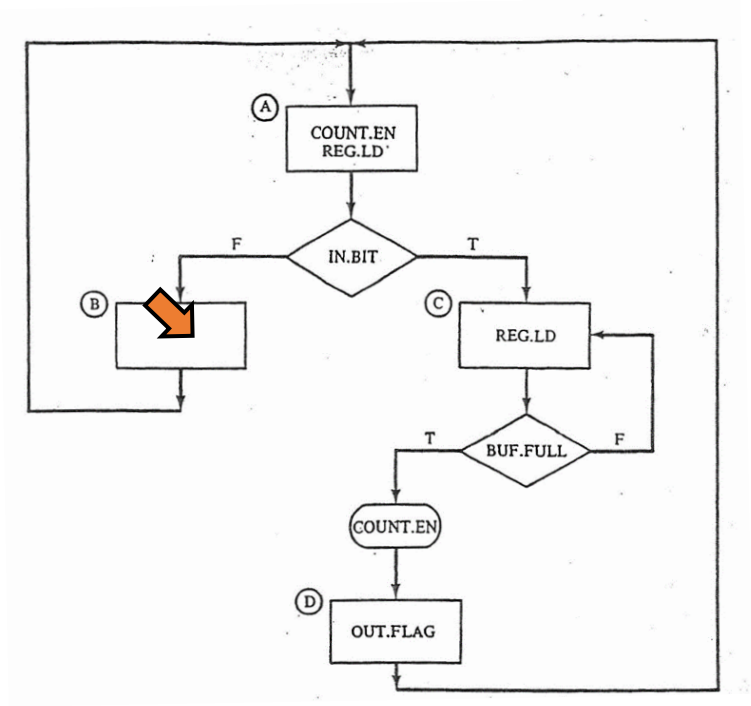
fsm.sv window (4)

```
Hits BC Ln#
                1 module fsm (
                2     input  logic clk,
                3     input  logic rst,
                4     input  logic inbit,
                5     input  logic buffull,
                6     output logic counten,
                7     output logic regld,
                8     output logic outflag
                9 );
               10
               11     // State encoding
               12     typedef enum logic [1:0] {A=2'b00, B=2'b01, C=2'b10, D=2'b11} state_t;
               13     state_t current_state, next_state;
               14
               15     // Sequential state update
✓              16     always_ff @(posedge clk or posedge rst) begin
✓              17         if (rst)
✓              18             current_state <= A;
✓              19         else
✓              20             current_state <= next_state;
               21     end
               22
               23     // Combinational logic
✓              24     always_comb begin
               25         // Default values
✓              26         counten    = 1'b0;
✓              27         regld      = 1'b0;
✓              28         outflag    = 1'b0;
✓              29         next_state = current_state;
               30
  X            31         case (current_state)
✓              32             A: begin
✓              33                 counten = 1'b1;
✓              34                 regld   = 1'b1;
✓              35                 if (inbit)
✓              36                     next_state = C;
✓              37                 else
✓              38                     next_state = B;
               39             end
               40
XB X           41             B: begin
Xs             42                 next_state = A; // Outputs remain default
               43             end
               44
✓              45             C: begin
✓              46                 regld = 1'b1; // Always active
✓              47                 if (buffull) begin
✓              48                     counten    = 1'b1;
```

Xs: Statement not executed (Hits column)

FSM Coverage (f flag)

f (**f**sm coverage) tracks state and transition coverage for finite state machines



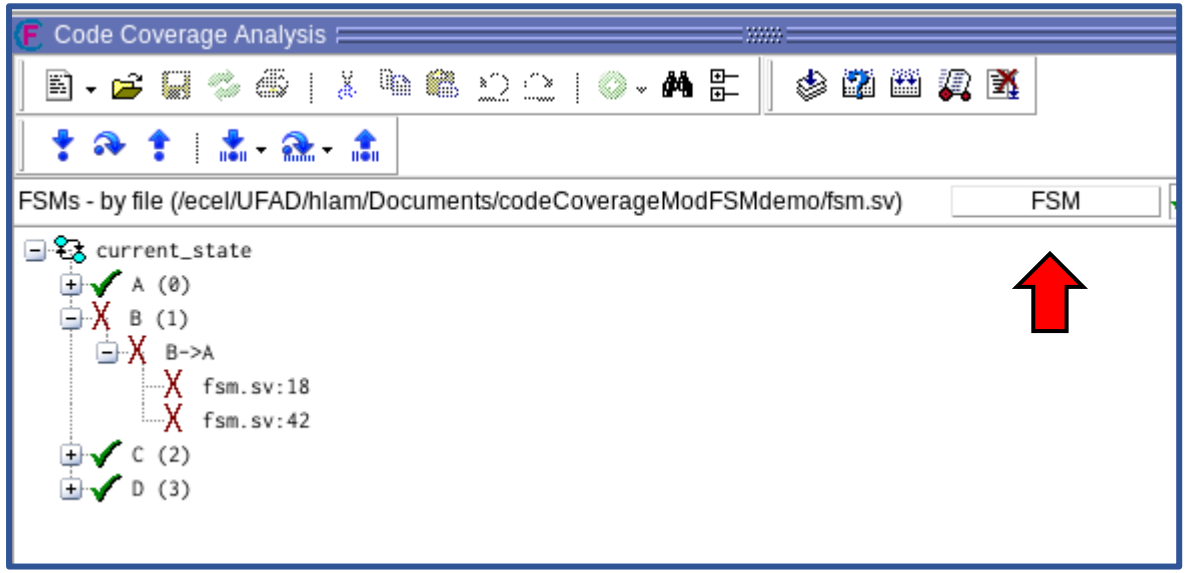
fsm.sv window (4)

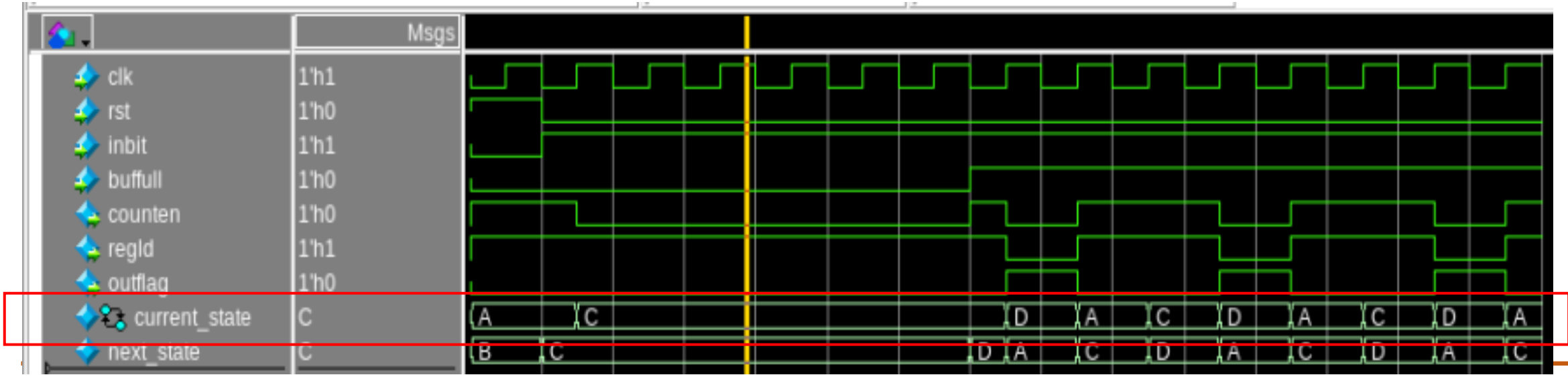
```
Hits | BC | Ln#
     |    |  1  module fsm (
     |    |  2      input  logic clk,
     |    |  3      input  logic rst,
     |    |  4      input  logic inbit,
     |    |  5      input  logic buffull,
     |    |  6      output logic counten,
     |    |  7      output logic regld,
     |    |  8      output logic outflag
     |    |  9  );
     |    | 10
     |    | 11      // State encoding
     |    | 12      typedef enum logic [1:0] {A=2'b00, B=2'b01, C=2'b10, D=2'b11} state_t;
     |    | 13      state_t current_state, next_state;
     |    | 14
     |    | 15      // Sequential state update
  ✓  |    | 16      always_ff @(posedge clk or posedge rst) begin
  ✓  |    | 17          if (rst)
  ✓  |    | 18              current_state <= A;
  ✓  |    | 19          else
  ✓  |    | 20              current_state <= next_state;
     |    | 21      end
     |    | 22
     |    | 23      // Combinational logic
  ✓  |    | 24      always_comb begin
     |    | 25          // Default values
  ✓  |    | 26          counten    = 1'b0;
  ✓  |    | 27          regld      = 1'b0;
  ✓  |    | 28          outflag    = 1'b0;
  ✓  |    | 29          next_state = current_state;
     |    | 30
     |  X | 31          case (current_state)
  ✓  |    | 32              A: begin
  ✓  |    | 33                  counten = 1'b1;
  ✓  |    | 34                  regld   = 1'b1;
  ✓  |    | 35                  if (inbit)
  ✓  |    | 36                      next_state = C;
  ✓  |    | 37                  else
  ✓  |    | 38                      next_state = B;
     |    | 39              end
     |    | 40
  X  |  X | 41              B: begin
  Xs |    | 42                  next_state = A; // Outputs remain default
     |    | 43              end
     |    | 44
  ✓  |    | 45              C: begin
  ✓  |    | 46                  regld = 1'b1; // Always active
  ✓  |    | 47                  if (buffull) begin
  ✓  |    | 48                      counten    = 1'b1;
```

FSM Coverage (f flag)

Transcript

```
#      Enabled Coverage              Bins      Hits    Misses     Weight  Coverage
#      ----------------              ----      ----    ------     ------  --------
#      Branches                        11        10         1          1    90.90%
#      FSM States                       4         3         1          1    75.00%
#      FSM Transitions                  6         3         3          1    50.00%
#      Statements                      40        39         1          1    97.50%
#      Toggles                         48        23        25          1    47.91%
# Total coverage (filtered view): 72.26%
```





Questa Icon Description

Icon	Description/Indication
✔	All statements, branches, conditions, or expressions on a particular line have been executed
X	Multiple kinds of coverage on the line were not executed
X_T	True branch not executed (BC column)
X_F	False branch not executed (BC column)
X_C	Condition not executed (Hits column)
X_E	Expression not executed (Hits column)

Icon	Description/Indication
X_B	Branch not executed (Hits column)
X_S	Statement not executed (Hits column)
E	Indicates a line of code to which active coverage exclusions have been applied. Every item on the line is excluded; none are hit.
E	Some excluded items are hit
E	Some items are excluded, and all items not excluded are hit
E	Some items are excluded, and some items not excluded have missing coverage
E	Auto exclusions have been applied to this line. Hover the cursor over the EA and a tool tip balloon appears with the reason for exclusion,

Reference: Questa® SIM Tutorial

http://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/swdocs/questasim/questa_sim_tut_2024_2.pdf