

Code Coverage in Verification

EEL 4712

HDL vs. SystemVerilog

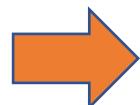
Limitations of HDLs (VHDL, Verilog)



- 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*
 - Assertion, constrained random stimuli, functional coverage
- *Industry adoption*

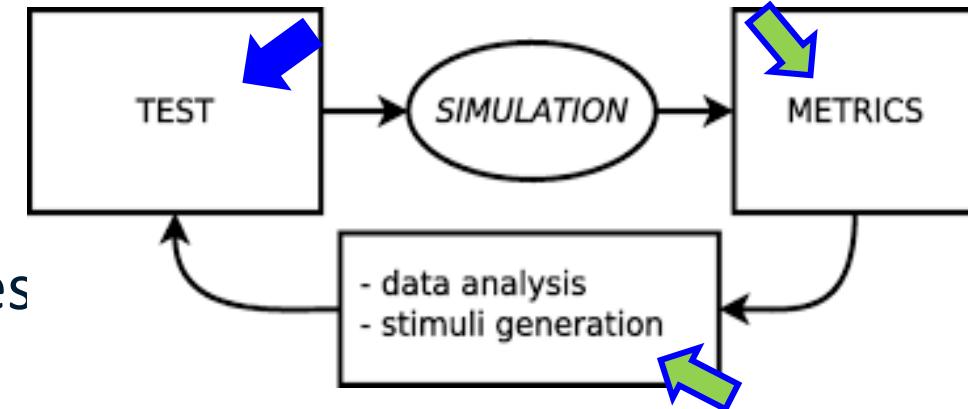
Design Verification: Testing and Coverage

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.

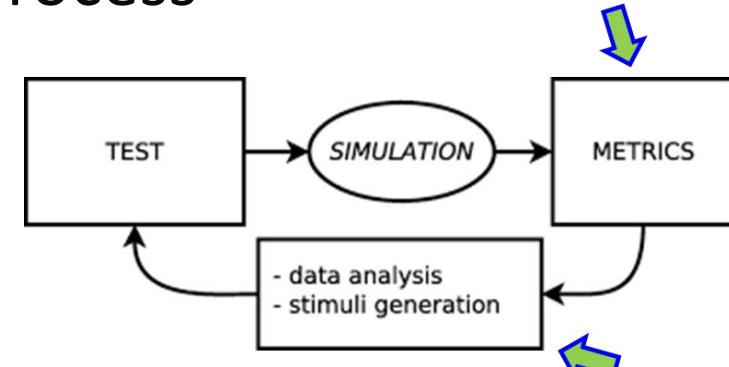
Coverage: ←

- 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.

Hits	BC	Ln#	
✓		32	
✓		33	A: begin
✓		34	counten = 1'b1;
✓		35	regld = 1'b1;
✓		36	if (inbit)
✓		37	next_state = C;
✓		38	else
✓		39	next_state = B;
✓		40	end
✓		41	B: begin
✓		42	next_state = A; // Outputs remain default
✓		43	end
✓		44	C: begin
✓		45	regld = 1'b1; // Always active
✓		46	if (buffull) begin
✓		47	counten = 1'b1;
✓		48	end else begin
✓		49	next_state = D;
✓		50	end
✓		51	end
✓		52	D: begin
✓		53	outflag = 1'b1;
✓		54	next_state = A;
✓		55	end
✓		56	default: begin
✓		57	next_state = A;
✓		58	end
✓		59	endcase
✓		60	
✓		61	
✓		62	
✓		63	

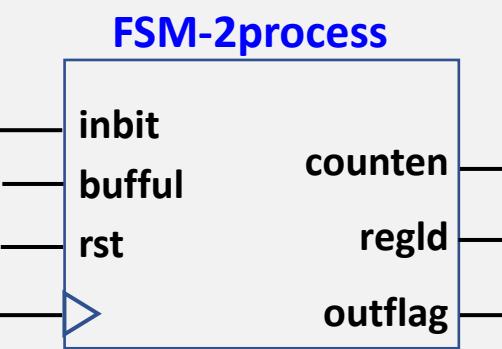
- s (statement)** tracks which statements in your design code are executed
- b (branch c)** if/case statements has been taken both true and false path
- e (expression)** logical expressions as a whole evaluated to both true and false
- c (conditional)** each individual boolean sub-expression inside a decision expression
- x (extended toggle)** tracks signal bits toggling ($0 \rightarrow 1$ and $1 \rightarrow 0$)
- f (fsm 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,
    input logic rst,
    input logic inbit,
    input logic bufful,
    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
```

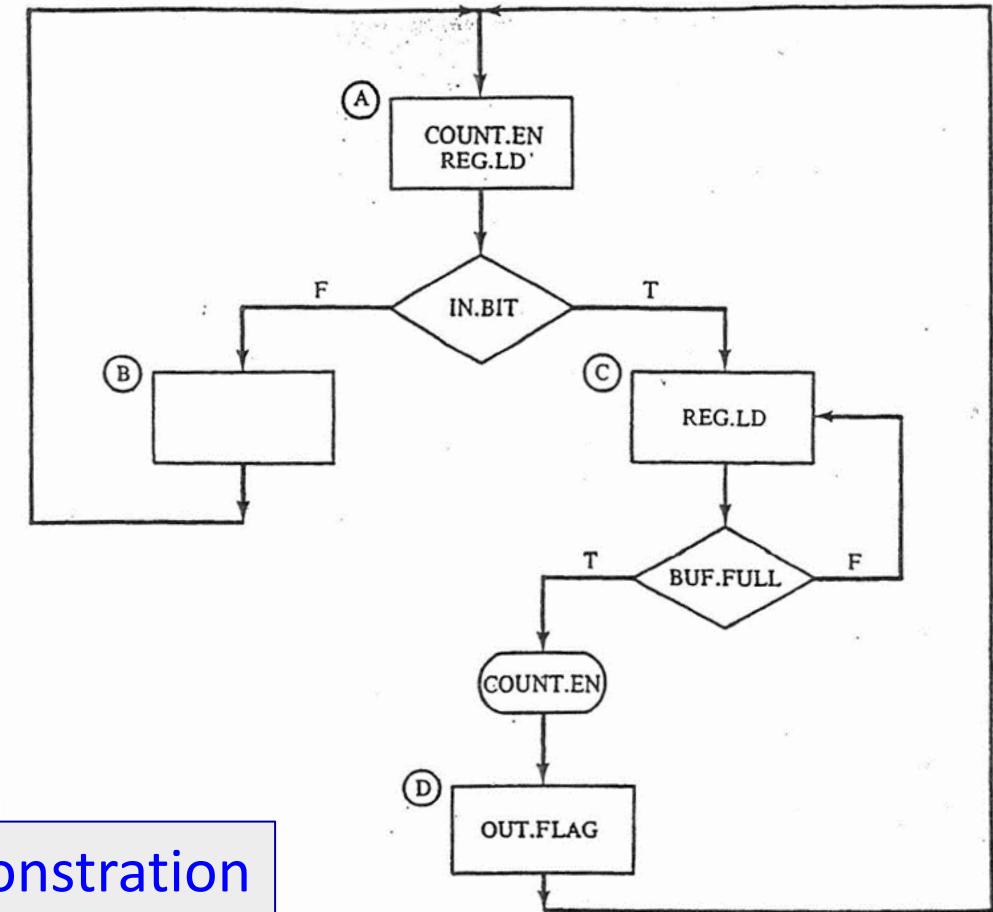
FSM-2process



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

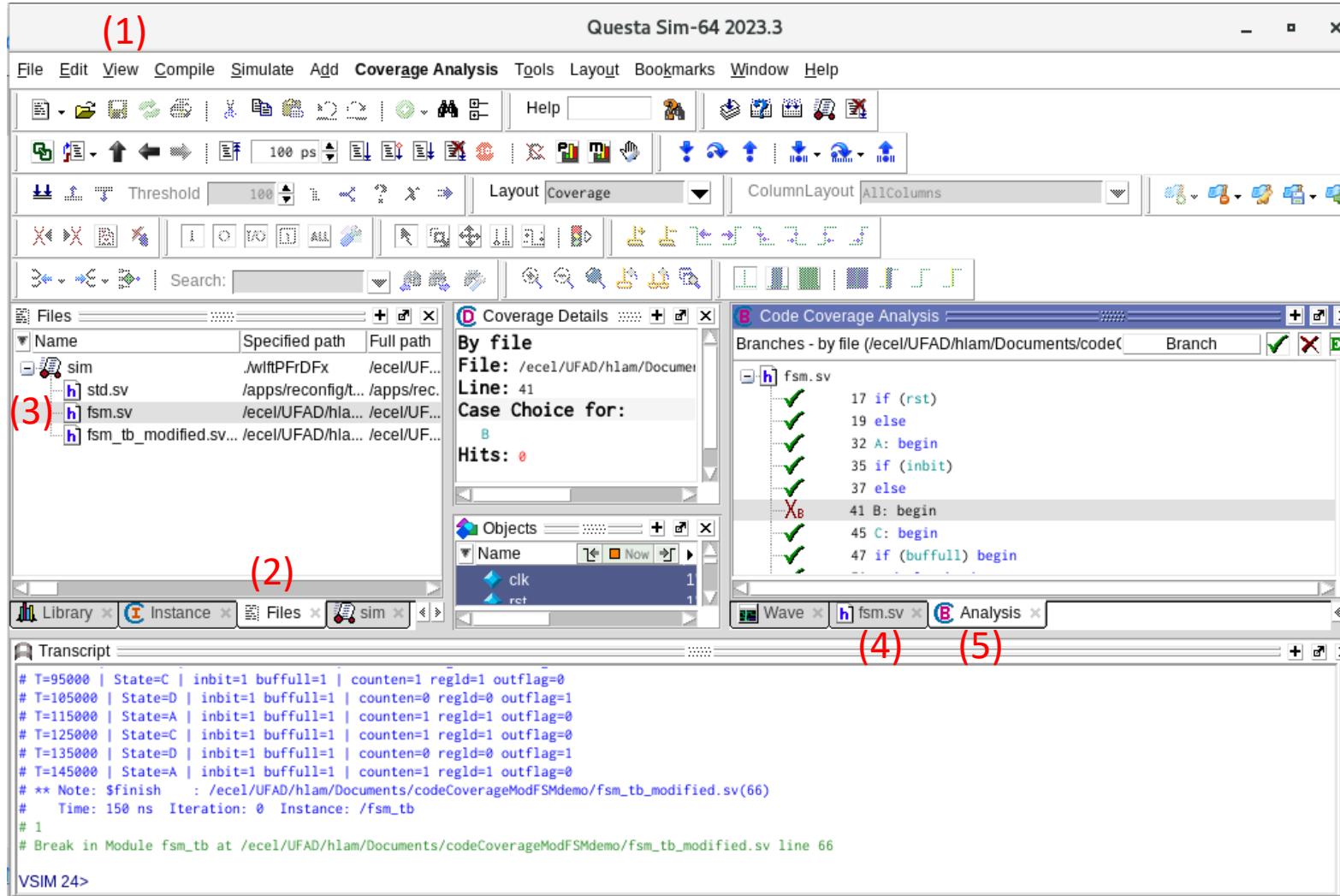
To view results and generate a summary report

- 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 | counter=1 regld=1 outflag=0
# T=10000 | State=A | inbit=1 buffull=0 | counter=1 regld=1 outflag=0
# T=15000 | State=C | inbit=1 buffull=0 | counter=0 regld=1 outflag=0
# T=70000 | State=C | inbit=1 buffull=1 | counter=1 regld=1 outflag=0
# T=75000 | State=D | inbit=1 buffull=1 | counter=0 regld=0 outflag=1
```

#	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.

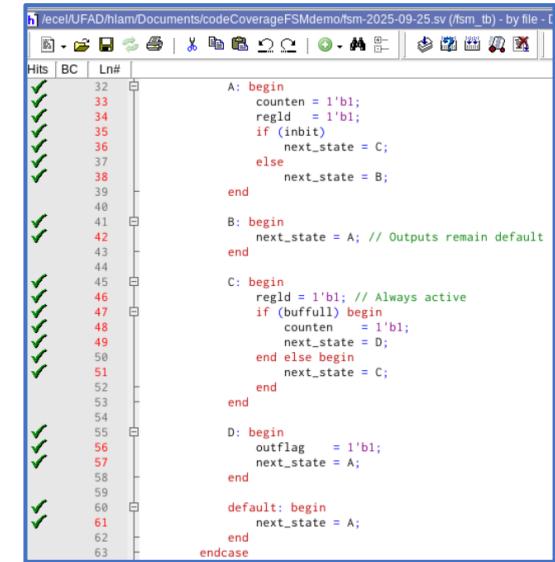
Code Coverage Types

- **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 (statement)** tracks which statements in your design code are executed
- b (branch c)** if/case statements has been taken both true and false path
- e (expression)** logical expressions as a whole evaluated to both true and false
- c (conditional)** each individual boolean sub-expression inside a decision expression
- x (extended toggle)** tracks signal bits toggling ($0 \rightarrow 1$ and $1 \rightarrow 0$)
- f (fsm coverage)** tracks state and transition coverage for finite state machines



The screenshot shows a Questa transcript window displaying a code coverage report for a file named fsm-2025-09-25.sv. The report lists various statements (labeled A through D) and their execution status. The left column shows the line number (Ln#), the middle column shows the branch condition (BC), and the right column shows the statement content. Green checkmarks indicate successful hits for each line. The code itself defines a state machine with transitions based on counter values and regld conditions.

Hits	BC	Ln#	Statement
✓		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	B: begin
✓		41	next_state = A; // Outputs remain default
✓		42	end
✓		43	C: begin
✓		44	regld = 1'b1; // Always active
✓		45	if (buffull) begin
✓		46	counten = 1'b1;
✓		47	end else begin
✓		48	next_state = D;
✓		49	end
✓		50	end
✓		51	D: begin
✓		52	outflag = 1'b1;
✓		53	next_state = A;
✓		54	end
✓		55	default: begin
✓		56	next_state = A;
✓		57	end
✓		58	endcase
✓		59	
✓		60	
✓		61	
✓		62	
✓		63	

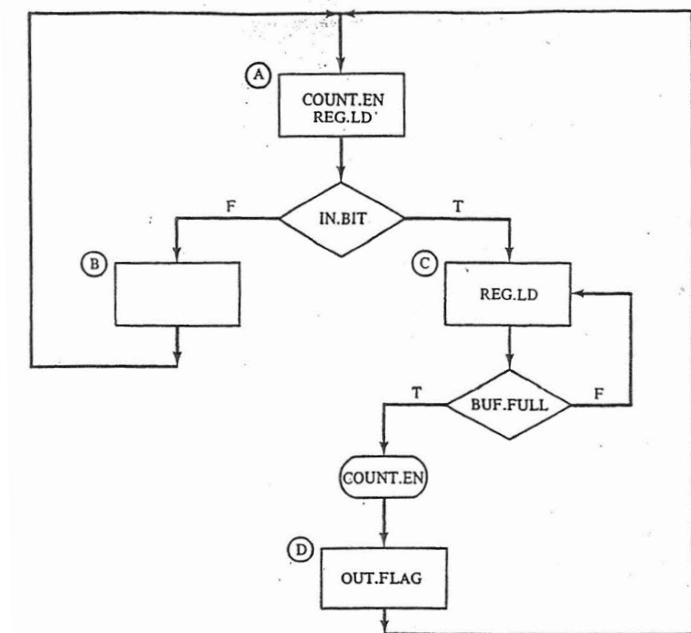
Statement Coverage (s flag)

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

- Highlighting untested code paths

Code coverage summary table*

#	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	48	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)

Code Coverage Analysis

Statements - by file (/ecel/UFAD/hlam/Documents/codeCoverageModFSMdemo/fsm.sv) Statement ✓

fsm.sv

```
16 always_ff @(posedge clk or posedge rst) begin
17   current_state <= A;
18   current_state <= next_state;
19
20   always_comb begin
21     counten    = 1'b0;
22     regld     = 1'b0;
23     outflag   = 1'b0;
24     next_state = current_state;
25     counten = 1'b1;
26     regld   = 1'b1;
27     next_state = C;
28     next_state = B;
29
30   next_state = A; // Outputs remain default
31   regld = 1'b1; // Always active
32   counten = 1'b1;
33   next_state = D;
34   next_state = C;
35   outflag   = 1'b1;
36   next_state = A;
37   next_state = A;
```

Hits BC Ln#

```
module fsm (
  input logic clk,
  input logic rst,
  input logic inbit,
  input logic bufffull,
  output logic counten,
  output logic regld,
  output logic outflag
);

// State encoding
typedef enum logic [1:0] (A=2'b00, B=2'b01, C=2'b10, D=2'b11) state_t;
state_t current_state, next_state;

// Sequential state update
always_ff @(posedge clk or posedge rst) begin
  if (rst)
    current_state <= A;
  else
    current_state <= next_state;
end

// Combinational logic
always_comb begin
  // Default values
  counten    = 1'b0;
  regld     = 1'b0;
  outflag   = 1'b0;
  next_state = current_state;

  case (current_state)
    A: begin
      counten = 1'b1;
      regld   = 1'b1;
      if (inbit)
        next_state = C;
      else
        next_state = B;
    end
    B: begin
      next_state = A; // Outputs remain default
    end
    C: begin
      regld = 1'b1; // Always active
      if (bufffull) begin
        counten = 1'b1;
      end
    end
  endcase
end
endmodule
```

fsm.sv window (4)

Xs: Statement not executed (Hits column)

Questa Icon Description

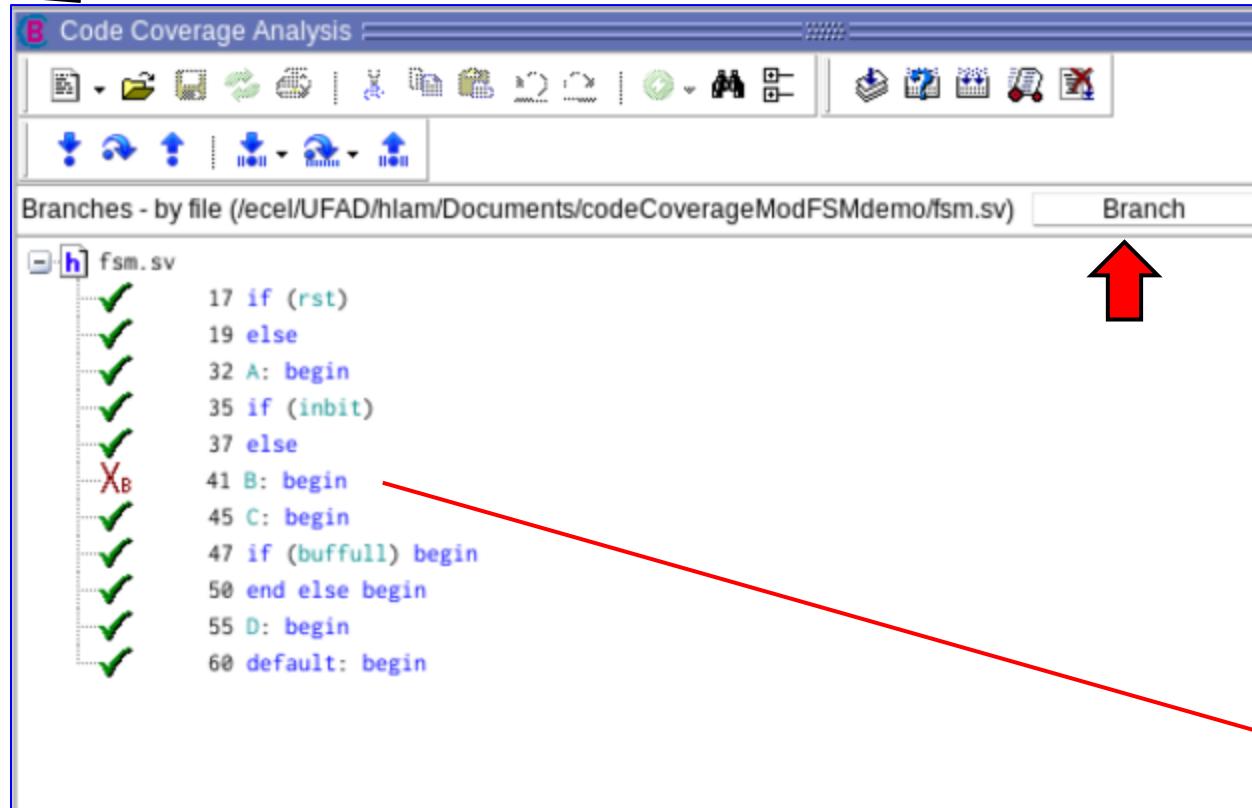
Icon	Description/Indication
✓	All statements, branches, conditions, or expressions on a particular line have been executed
X	Multiple kinds of coverage on the line were not executed
X _T	True branch not executed (BC column)
X _F	False branch not executed (BC column)
X _C	Condition not executed (Hits 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 _H	Some excluded items are hit
E _A	Some items are excluded, and all items not excluded are hit
E _X	Some items are excluded, and some items not excluded have missing coverage
E _A	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)



→ X_B X_S

fsm.sv window (4)

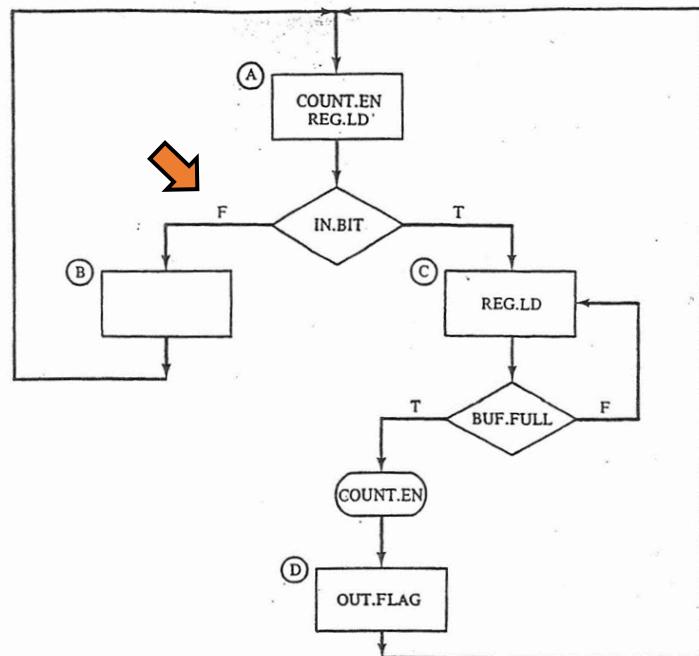
Hits	BC	Ln#
		1 module fsm (
		2 input logic clk,
		3 input logic rst,
		4 input logic inbit,
		5 input logic buffull,
		6 output logic counter,
		7 output logic regld,
		8 output logic outflag
		9);
		10
		11 // State encoding
		12 typedef enum logic [1:0] (A=2'b00, B=2'b01, C=2'b10, D=2'b11) state_t;
		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 counter = 1'b0;
		27 regld = 1'b0;
		28 outflag = 1'b0;
		29 next_state = current_state;
		30
		31 case (current_state)
		32 A: begin
		33 counter = 1'b1;
		34 regld = 1'b1;
		35 if (inbit)
		36 next_state = C;
		37 else
		38 next_state = B;
		39 end
		40
		41 B: begin
		42 next_state = A; // Outputs remain default
		43 end
		44
		45 C: begin
		46 regld = 1'b1; // Always active
		47 if (buffull) begin
		48 counter = 1'b1;

X_B: Branch not executed (Hits column)

Branch Coverage (b flag)

b (branch 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.



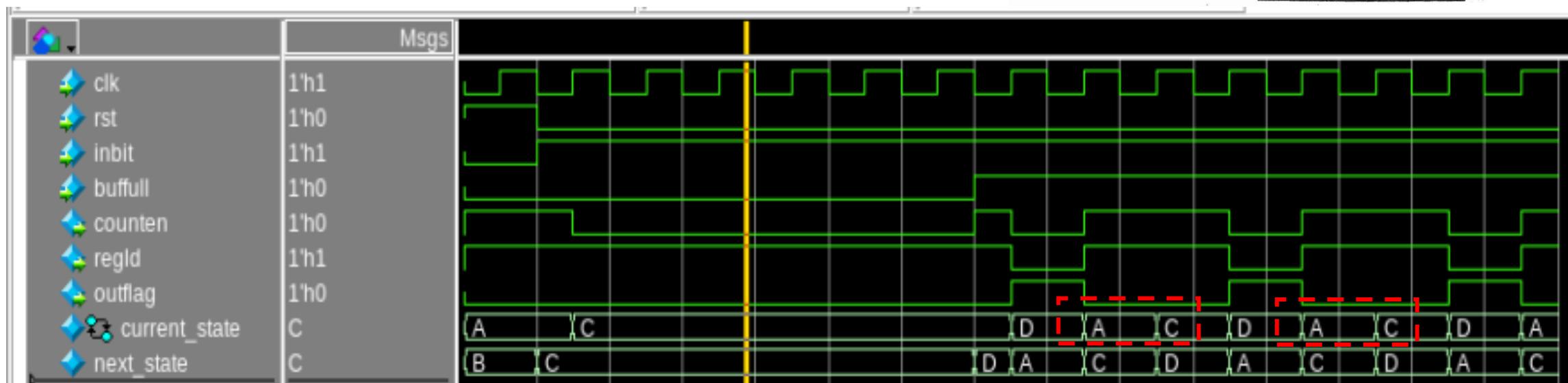
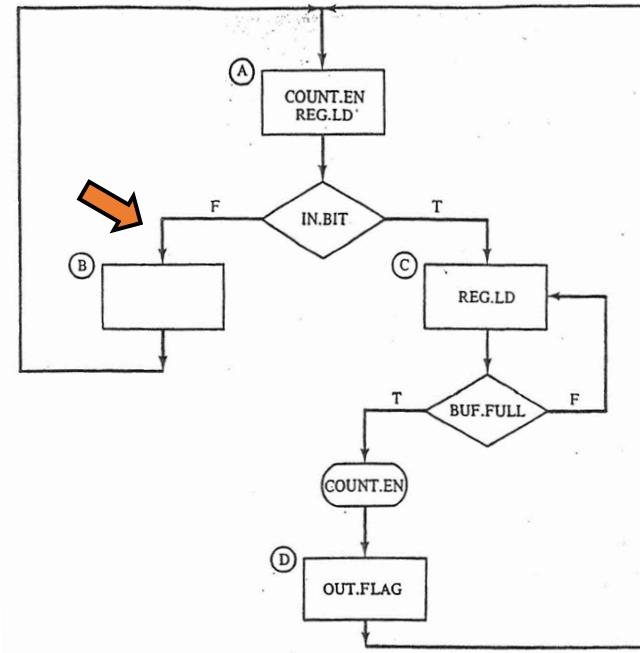
Hits	BC	Ln#	
		26);
✓		27	// Clock generation
		28	always #5 clk = ~clk;
		29	
		30	// Stimulus
✓		31	initial begin
✓		32	// Init
✓		33	clk = 0; rst = 1;
✓		34	inbit = 0; buffull = 0;
✓		35	#10 rst = 0;
		36	
		37	
		38	// --- Test path: A -> B -> A ---
		39	//inbit = 0; For demo, commented out branch 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;
✓		63	buffull = 1;

fsm.sv window (4)

Branch Coverage (b flag)

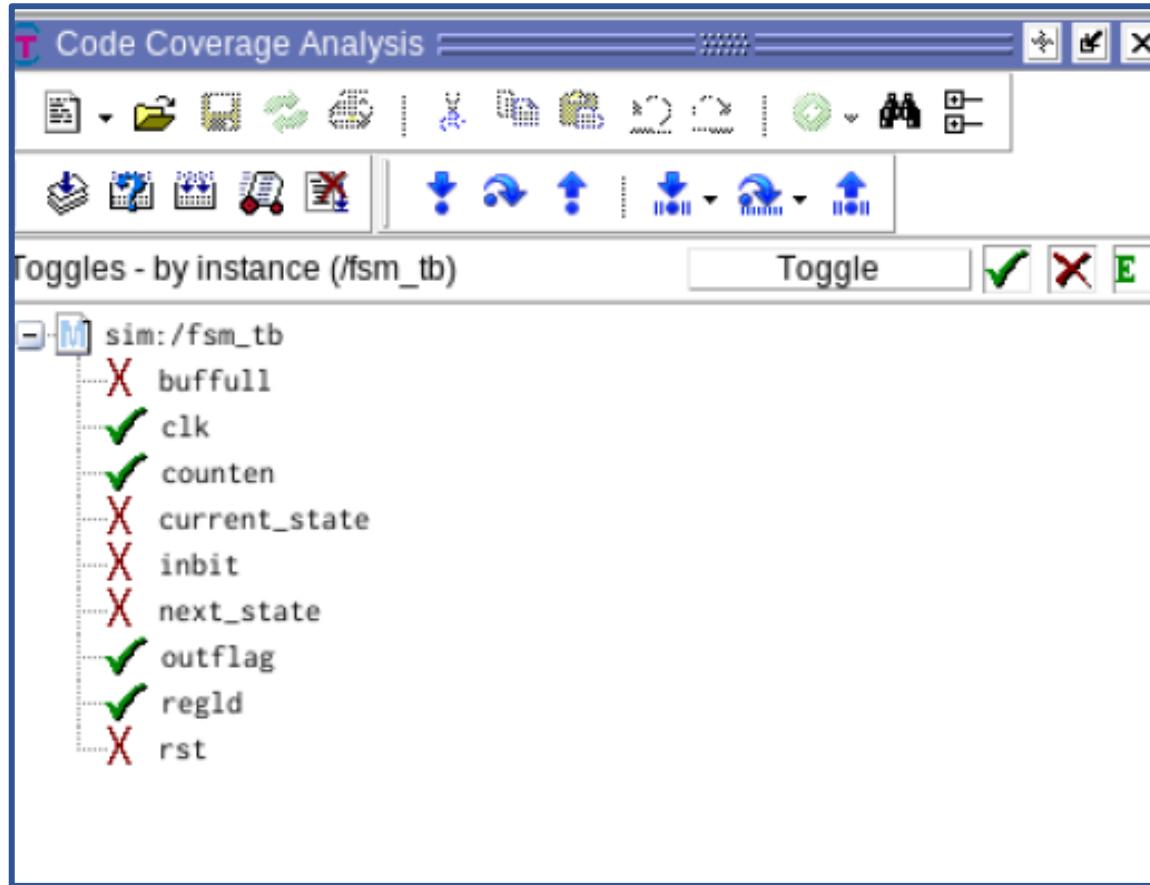
b (branch c) if/case statements has been taken both true and false path

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



Toggle Coverage (x flag)

x (extended toggle) tracks signal bits toggling ($0 \rightarrow 1$ and $1 \rightarrow 0$)



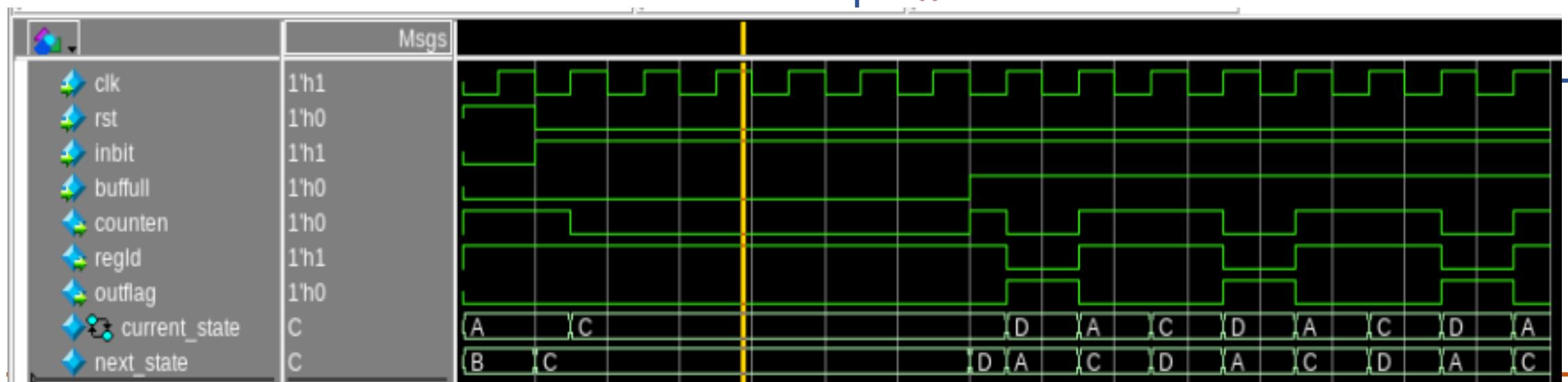
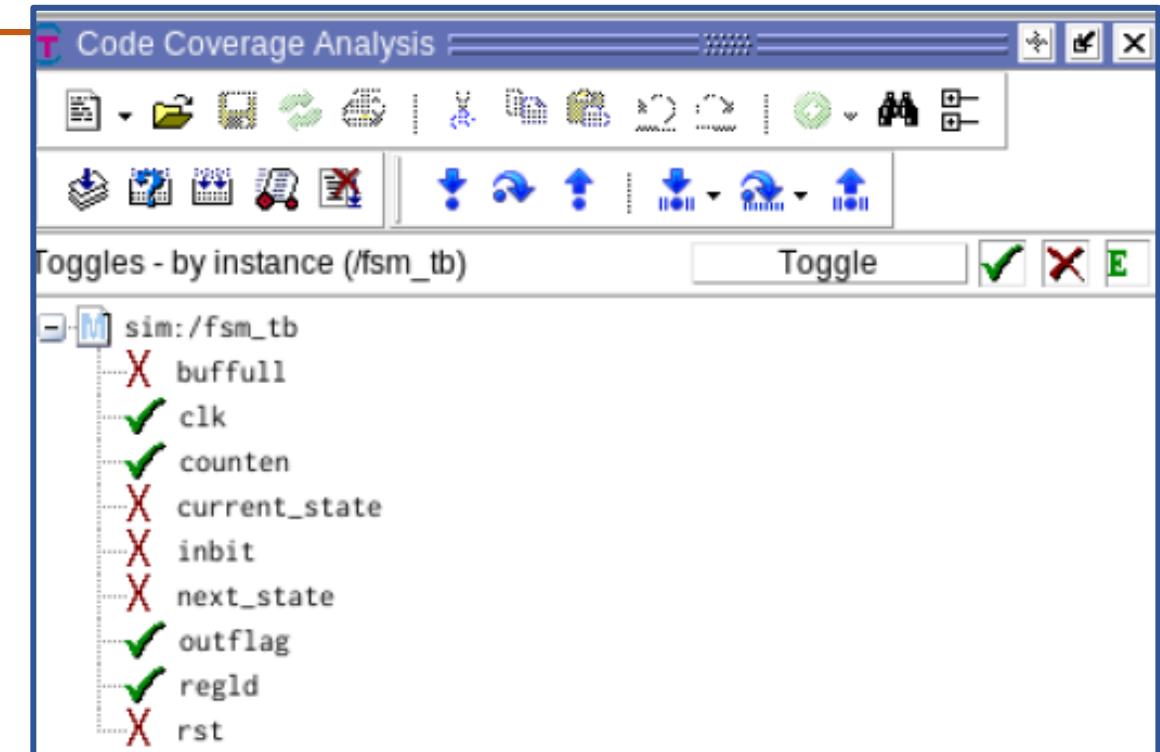
Testbench files:

- [link to fsm_tb.sv](#)
- [link to fsm_tb_modified.sv](#)

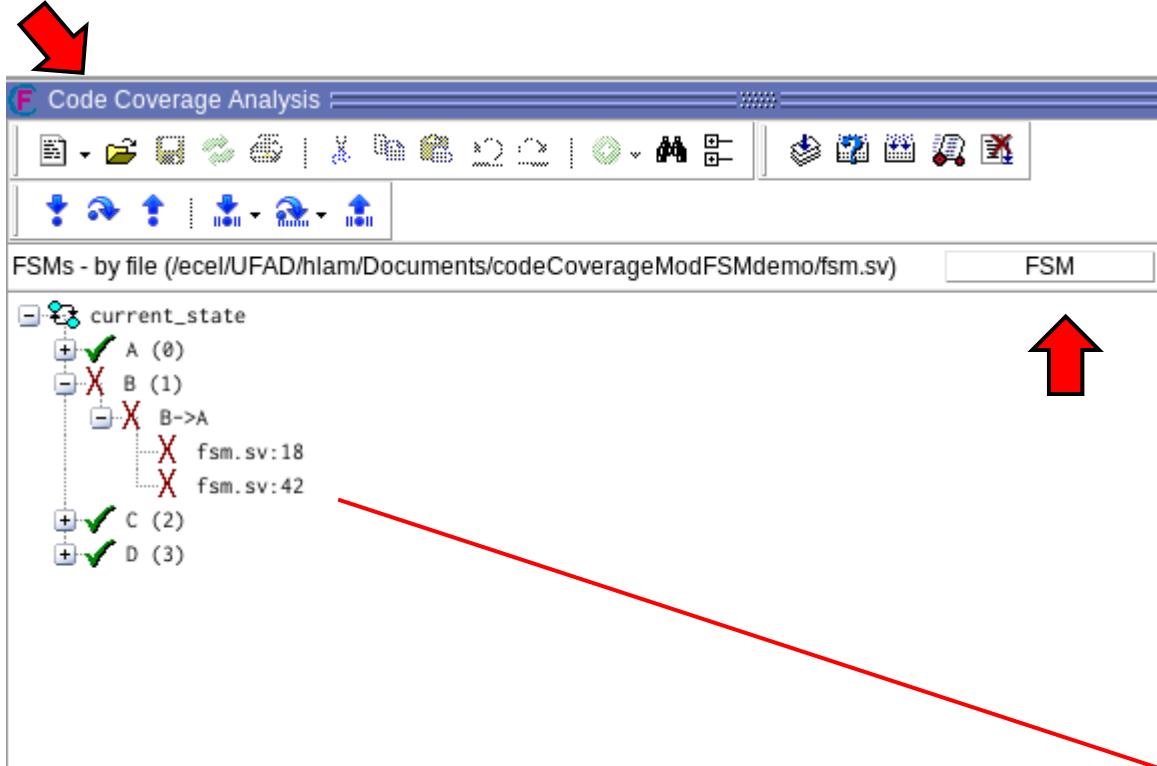


Toggle Coverage (x flag)

#	Enabled Coverage	Bins	Hits	Misses	Weight	Coverage
<hr/>						
#	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%						



FSM Coverage (f flag)



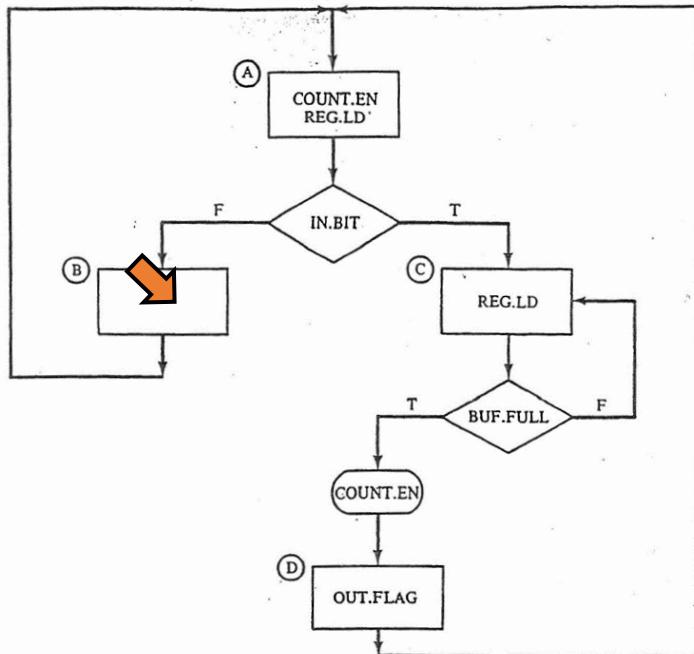
Hits	BC	Ln#	
✓		1	module fsm (
✓		2	input logic clk,
✓		3	input logic rst,
✓		4	input logic inbit,
✓		5	input logic buffull,
✓		6	output logic counter,
✓		7	output logic regld,
✓		8	output logic outflag
		9);
		10	
		11	// State encoding
		12	typedef enum logic [1:0] (A=2'b00, B=2'b01, C=2'b10, D=2'b11) 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	counteren = 1'b0;
		27	regld = 1'b0;
		28	outflag = 1'b0;
		29	next_state = current_state;
		30	
		31	case (current_state)
		32	A: begin
		33	counteren = 1'b1;
		34	regld = 1'b1;
		35	if (inbit)
		36	next_state = C;
		37	else
		38	next_state = B;
		39	end
		40	
		41	B: begin
		42	next_state = A; // Outputs remain default
		43	end
		44	
		45	C: begin
		46	regld = 1'b1; // Always active
		47	if (buffull) begin
		48	counteren = 1'b1;

fsm.sv window (4)

Xs: Statement not executed (Hits column)

FSM Coverage (f flag)

f (fsm coverage) tracks state and transition coverage for finite state machines



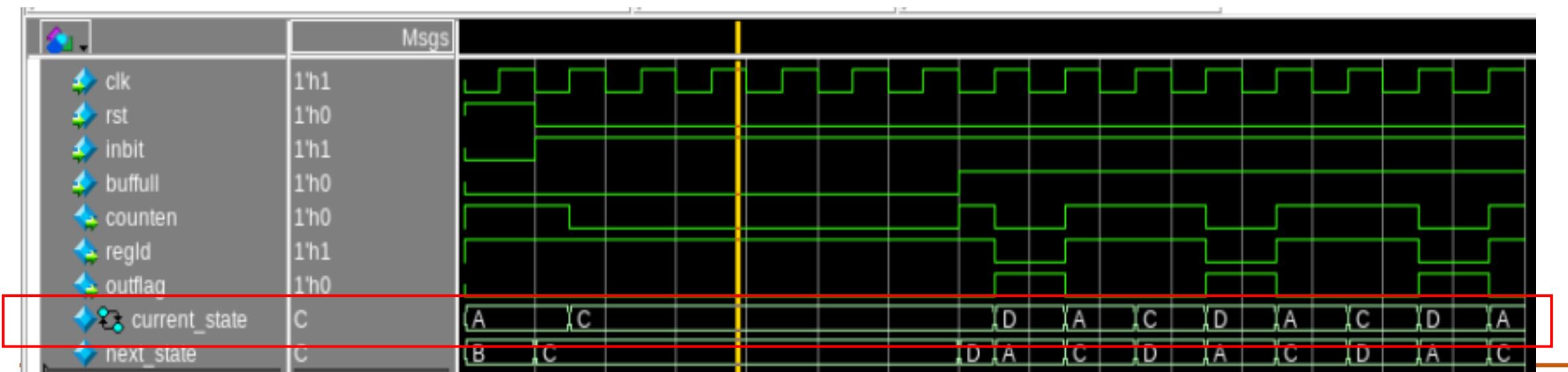
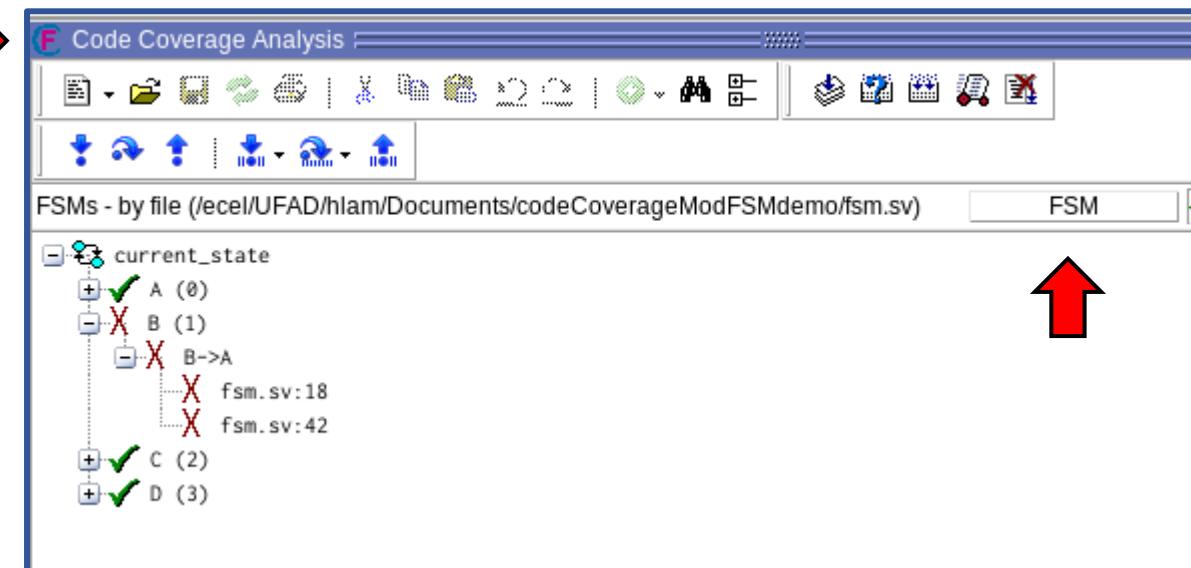
Hits	BC	Ln#
✓		1 module fsm (
✓		2 input logic clk,
✓		3 input logic rst,
✓		4 input logic inbit,
✓		5 input logic buffull,
✓		6 output logic counter,
✓		7 output logic regld,
✓		8 output logic outflag
✓		9);
✓		10
✓		11 // State encoding
✓		12 typedef enum logic [1:0] (A=2'b00, B=2'b01, C=2'b10, D=2'b11) 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 counteren = 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 counteren = 1'b1;
✓		34 regld = 1'b1;
✓		35 if (inbit)
✓		36 next_state = C;
✓		37 else
✓		38 next_state = B;
✓		39 end
X _B		40 B: begin
X _S		41 next_state = A; // Outputs remain default
X _S		42 end
X _S		43
✓		44 C: begin
✓		45 regld = 1'b1; // Always active
✓		46 if (buffull) begin
✓		47 counteren = 1'b1;
✓		48 end

fsm.sv window (4)

FSM Coverage (f flag)



Transcript :						
#	Enabled Coverage	Bins	Hits	Misses	Weight	Coverage
<hr/>						
#	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 _H	Some excluded items are hit
E _A	Some items are excluded, and all items not excluded are hit
E _X	Some items are excluded, and some items not excluded have missing coverage
E _A	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