

CS5120 VLSI System Design, Spring 2025

# Code Coverage Analysis

黃稚存

Chih-Tsun Huang

[cthuang@cs.nthu.edu.tw](mailto:cthuang@cs.nthu.edu.tw)



國立清華大學  
NATIONAL TSING HUA UNIVERSITY

資訊工程學系  
Computer Science

Lecture 12

- ◎ 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容 (例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。



This lecture contains the training materials  
from Cadence,  
and is only for academic usage at NTHU.  
**DO NOT REDISTRIBUTE IT!!**



# Outline

- ① Introduction to Code Coverages
- ② Code Coverage Tool
- ③ Example case



# Introduction to Code Coverages



# Coverage Metrics

## ⦿ Code coverage

- ◆ Method of assessing how well the test cases have exercised the design
- ◆ Block & Branch / Expression / Toggle / FSM

## ⦿ Functional coverage

- ◆ Focuses on functional aspects of a design and provides a very good insight on how the verification goals set by a test plan are being met
- ◆ Assertion / CoverGroup



# Code Coverage vs. Functional Coverage

## ◎ Code coverage

- ◆ Reflect how thoroughly the DUT (Design Under Test) code has been exercised
- ◆ Enable detection of
  - Untested areas of DUT
  - Dead code or unused code
- ◆ Most of the work are done by coverage tools automatically

## ◎ Functional coverage

- ◆ Measure what features of DUT were exercised
- ◆ Lot of manual work
  - Planning
    - What are the features?
    - Where/how to measure?
  - Coding of assertions and covergroups



# Code Coverage: Block Coverage

Identifies the lines of code that are executed during a simulation run. It helps you determine what areas of the DUT design are not exercised by the testbenches and provide feedback for users to add more testcases.

A block is a statement or sequence of statements in Verilog/VHDL that executes with no branches or delays.

Either none or all of the statements in a block are executed.

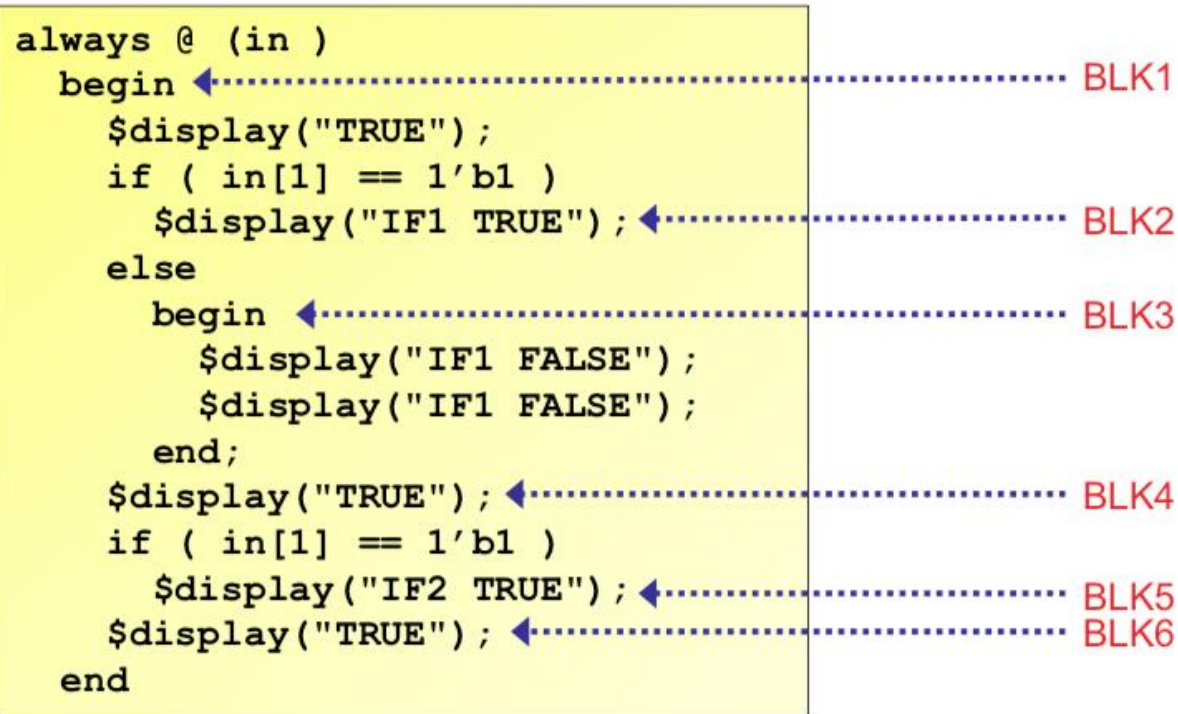
---

block coverage is an essential first step in the overall verification process.





# Code Coverage: Block Coverage



Any construct that breaks execution flow creates a new block, for example:

`begin, if, else, case, wait, #, @, for, forever, repeat, while, disable.`



# Code Coverage: Branch Coverage

- Branch Coverage: Yields more precise coverage details than block coverage by obtaining coverage results for various branches individually.
- With branch coverage, a piece of code is considered 100% covered when each branch of a conditional statement has been executed at least once.

```
assign out2 = (cond1) ? 1'b1 : 1'b0;
```

The above statement has the following branches:

- If cond1 evaluates to true, out2 is assigned 1'b1
- If cond1 evaluates to false, out2 is assigned 1'b0

Considered as a single block for block coverage. With branch coverage this statement is considered 100% covered when each branch of the statement has executed.

# Block Coverage Analysis Report

**BLOCKS**

Block Type	Source Start Line	Score
(no filter)	(no filter)	
code block	5	1
code block	6	1
code block	7	1
code block	8	1
code block	16	1
code block	17	1
code block	19	1
code block	23	1
true part of	24	1
false part of	28	1
code block	33	1
true part of	33	0
false part of	38	1
true part of	38	0
implicit else	38	1

**SOURCE OF: Block12**

```
code_coverage.sv
1 module top(output reg[1:0] out1, out2);
2   bit clk;
3   reg [1:0] v1, v2, v3;
4
5   initial begin
6     #10 v1 = 1; v3 = 0;
7     #10 v2 = 1;
8     #10 v1 = 0;
9     $finish();
10  end
11
12  assign out1 = v3
13    ?2'b11
14    :2'b00;
15
16  initial begin
17    forever begin
18      clk = 0;
19      #2 clk = ~clk;
20    end
21  end
22
23  always @(posedge clk)
24    if(v1==1 & v2==1) begin
25      $display("Both v1 and v2 are 1");
26      $display("%t v1=%0d v2=%0d v3=%0d out1=%b", $time(), v1, v2, v3, out1);
27    end
28    else begin
29      $display("v1 and v2 are NOT 1");
30      $display("%t v1=%0d v2=%0d v3=%0d out1=%b", $time(), v1, v2, v3, out1);
31    end
32
33    if(v1==2 & v2==1) begin
34      $display("Will never display this");
35      $display("Following statements are not executed either");
36      v1 = 0; v2 = 0; v3 = 0;
37    end
38    else if(v2==2) begin
39      $display("This statement and following statement are not executed either");
40      v2 = 0;
41    end
42  end
43
44 endmodule
```

Notice that conditional assign statement was not scored with block coverage analysis

Score is 0 for the true part of if statement line 33

Score is 0 for the true part of if statement line 38

This block of statements was never executed

This block of statements was never executed either



# Block Coverage Analysis Report

Blocks with Branches

Block Type	Source Start Line	Score
(no filter)	(no filter)	
code block	5	1
code block	6	1
code block	7	1
code block	8	1
ternary 1 true	13	0
ternary 1 false	14	1
code block	16	1
code block	17	1
code block	19	1
code block	23	1
true part of	24	1
false part of	28	1
code block	33	1
true part of	33	0
false part of	38	1
true part of	38	0
implicit else	38	1

code\_coverage.sv

```
1 module top(output reg[1:0] out1, out2);
2   bit clk;
3   reg [1:0] v1, v2, v3;
4
5   initial begin
6     #10 v1 = 1; v3 = 0;
7     #10 v2 = 1;
8     #10 v1 = 0;
9     $finish();
10  end
11
12  assign out1 = v3
13    ?2'b11
14    :2'b00;
15
16  initial begin
17    forever begin
18      clk = 0;
19      #2 clk = ~clk;
20    end
21  end
22
23  always @(posedge clk) begin
24    if(v1==1 & v2==1) begin
25      $display("Both v1 and v2 are 1");
26      $display("%t v1=%0d v2=%0d v3=%0d out1=%b", $time(), v1, v2, v3, out1);
27    end
28    else begin
29      $display("v1 and v2 are NOT 1");
30      $display("%t v1=%0d v2=%0d v3=%0d out1=%b", $time(), v1, v2, v3, out1);
31    end
32
33    if(v1==2 & v2==1) begin
34      $display("Will never display this");
35      $display("Following statements are not executed either");
36      v1 = 0; v2 = 0; v3 = 0;
37    end
38    else if(v2==2) begin
39      $display("This statement and following statement are not executed either");
40      v2 = 0;
41    end
42  end
43
44 endmodule
```

With "Branch Coverage" enable, conditional assign statement is now scored.



# Code Coverage: Expression Coverage



- Expression coverage measures how thoroughly the testbench exercises expressions in assignments and procedural control constructs (if/case conditions). It identifies each input condition that makes the expression true or false and whether that condition happened in simulation.

Before scoring expression coverage, make sure you have high block coverage in your regression.





# 3 Types of Expression Coverage

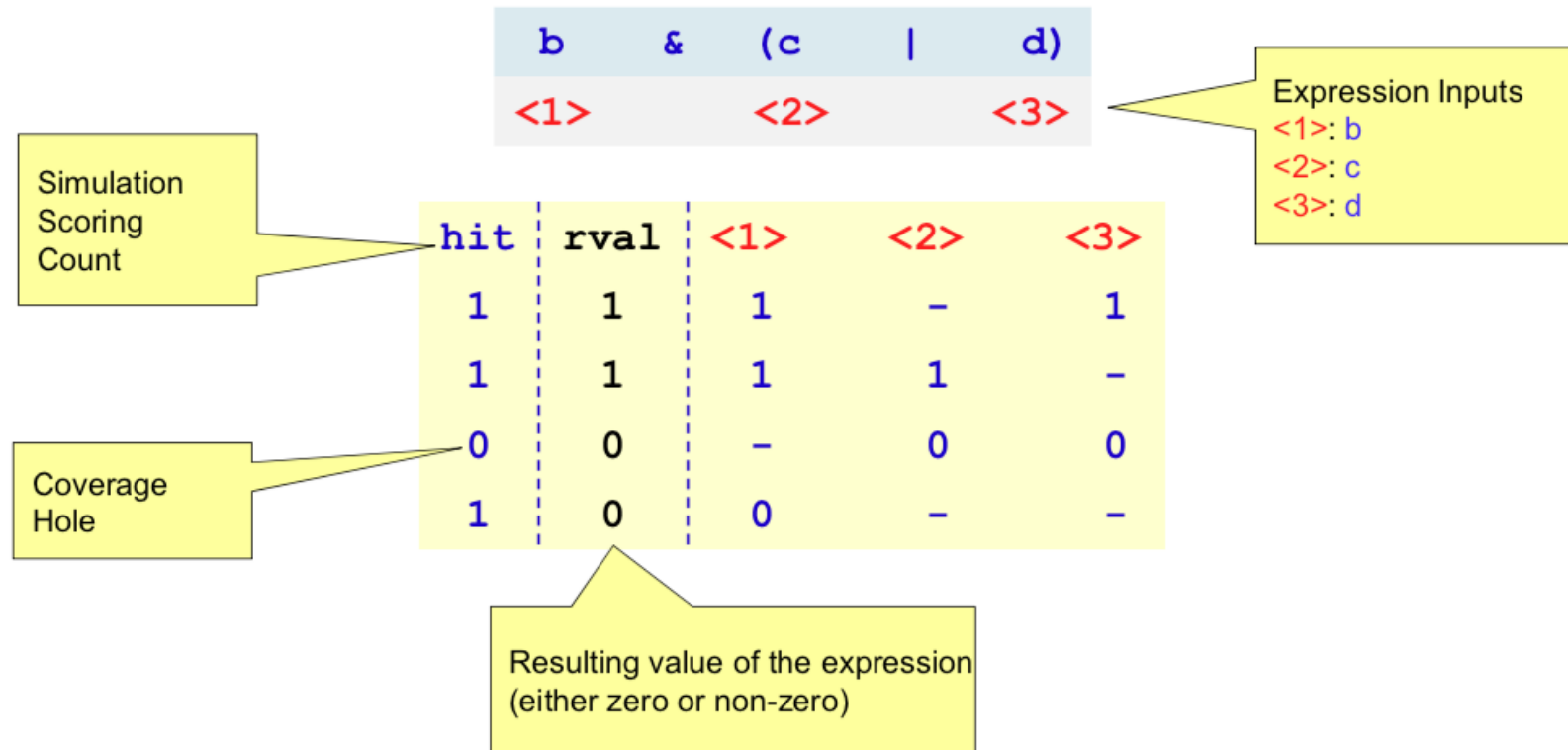
- **SOP – Sum of Products Scoring (Default Mode)**
  - Does each term take a 0 and non-0 value?
  - Vector inputs are scored as logical (single bit)
- **Control Scoring**
  - Does each term take a 0 and non-0 value \*AND\*
  - Does each term control the output result of the expression during simulation?
- **Vector Scoring**
  - Does each term take a 0 and non-0 value \*AND\*
  - Does each bit of each term control the result of the expression?

You can set different scoring modes for selected modules in your design.



# SOP Scoring

- Reduces expressions to a minimum set of expression inputs that make the expression both true and false, inherently first-level





# Control Scoring

- Checks if each input has controlled the output value of the expression at some time during the simulation
  - Known as “sensitized condition coverage” or “focused condition coverage”
  - Breaks an expression into a hierarchy of sub-expressions, more accurate

b	&	(c		d)
<1>		<--	2	-->
		<3>		<4>

Level 1:  
<1>: b  
<2>:(c | d)

Level 2:  
<3>: c  
<4>: d

hit	<1>	<2>
&	-----	
0	0	1
0	1	0
1	1	1

hit	<3>	<4>
	-----	
0	0	1
0	1	0
1	0	0





# Vector Scoring

- Vector scoring mode is an extension of control scoring mode.
- Each bit of a multi-bit signal is scored and reported separately and **lots of data to analyze**.

```
reg [3:0] a, b;  
Wire [3:0] val = (a && b);  
(a      &&      b)  
  
      <1>      <2>
```

hit	<1>	<2>
&&	-----	
0	0	1
0	1	0
1	1	1

Level 1:

<1>: a

<2>: b

Bit table for each  
vector operand

Hit	hit	T2[3]	[2]	[1]	[0]
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
1	1	1	0	0	0
0	0	0	0	0	0

# Expression Coverage Analysis Report

Source | Expression Hierarchy | Full Expression

code\_coverage.sv

```
22
23 always @(posedge clk) begin
24   if(v1==1 & v2==1) begin
25     $display("Both v1 and v2 are 1");
26     $display("%t v1=%0d v2=%0d v3=%0d out1=%b", $time(), v1, v2, v3, out1);
27   end
```

Source code window & selected expression

Source | Expression Hierarchy | Full Expression

Ex	Exp	Term	Index	Overall Average Grade	Expression Total	Expression Covered
-	(v1 == 1) & (v2 == 1)	n/a	1	33.33%	3	1 / 3 (33.33%)
-	(v1 == 1)	T1	2	50%	2	1 / 2 (50%)
-	v1	T3	n/a	n/a	0	0 / 0 (n/a)
-	1	T4	n/a	n/a	0	0 / 0 (n/a)
-	(v2 == 1)	T2	3	50%	2	1 / 2 (50%)
-	v2	T5	n/a	n/a	0	0 / 0 (n/a)
-	1	T6	n/a	n/a	0	0 / 0 (n/a)

Showing 7 items

& Coverage Table

Ex	rs	rs	T1	T2	Score
	0	-			0
	-	0			0
	1	1			1

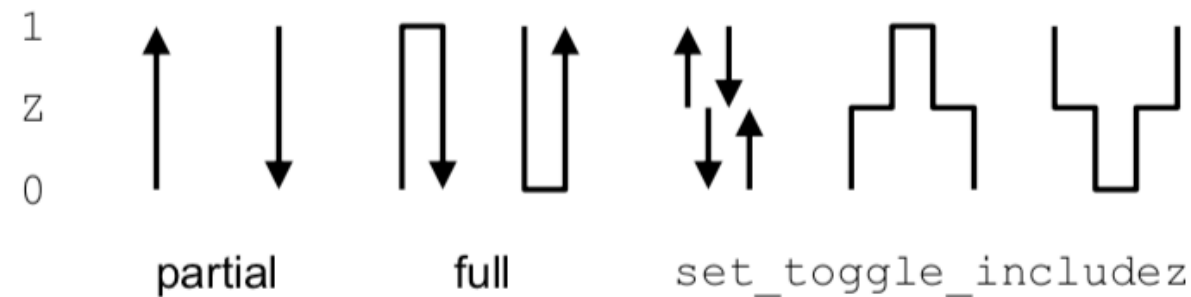
Detailed expression reports

(v1 == 1) was never false  
(v2 == 1) was never false  
→ Need additional test cases?



# Coverage Metrics: Toggle Coverage

- *What does toggle coverage do?*
  - Collect and report design signal toggle activity
- *What is a design signal toggle?*
  - Normally a binary transition (and return after a finite delay) of a DUT signal.
  - Signals may transition through (but not terminate at) an unknown state.
- *Why bother with toggle coverage?*
  - It's the only “code” coverage available for a gate-level netlist
  - Verify that design interconnect is connected and “wiggles”





# Code Coverage: FSM Coverage

- FSMs are critical control logic of DUT
- It is important that FSMs logic are exercised thoroughly by the testbench to ensure that there are no bugs
  - FSM coverage measures how well this logic is exercised

## FSM Coverage Types

- |                       |  |
|-----------------------|--|
| - State Coverage      | : reports what states were visited     |
| - Transition Coverage | : reports what transitions occurred    |
| - Arc Coverage        | : reports why each transition occurred |



# What Coverage Metrics Should I Enable?

- ⦿ Add in stages code coverage to identify holes in test environment
  - ◆ First-order: block or branch
  - ◆ Second-order: expression
- ⦿ Add in stages the simplest form of functional coverage in the form of FSM coverage
  - ◆ First-order: states and transitions
  - ◆ Second-order: arcs
- ⦿ NOTE: there is little value in scoring second-order coverages if the first-order ones show low coverage!



# Code Coverage Tool



# Generating Coverage Data

- Adding code coverage options with ncverilog

```
$ ncverilog gcd_t.v gcd.v \  
-coverage all \  
-covoverwrite \  
-covworkdir cov_work \  
-covscope scope \  
-covtest test
```

- Observing the coverage results by IMC (Integrated Metric Center) with GUI

```
$ imc &
```



# More about The Options

## 🕒 -coverage <coverage\_types>

- ◆ Enable coverage data generation
  - ▣ -coverage all
  - ▣ -coverage A
  - ▣ -coverage B:E

Coverage Type	Description
A or all	Enable all coverage types
B	Enable block coverage
E	Enable expression coverage
F	Enable FSM coverage
T	Enable toggle coverage
U	Enable functional coverage





# More about The Options

## ① -covworkdir <workdir>

- ◆ Basename for the work directory
- ◆ Default: cov\_work

## ① -covscope <scope>

- ◆ Specifies an alternate directory for storing coverage model files
- ◆ Default: scope

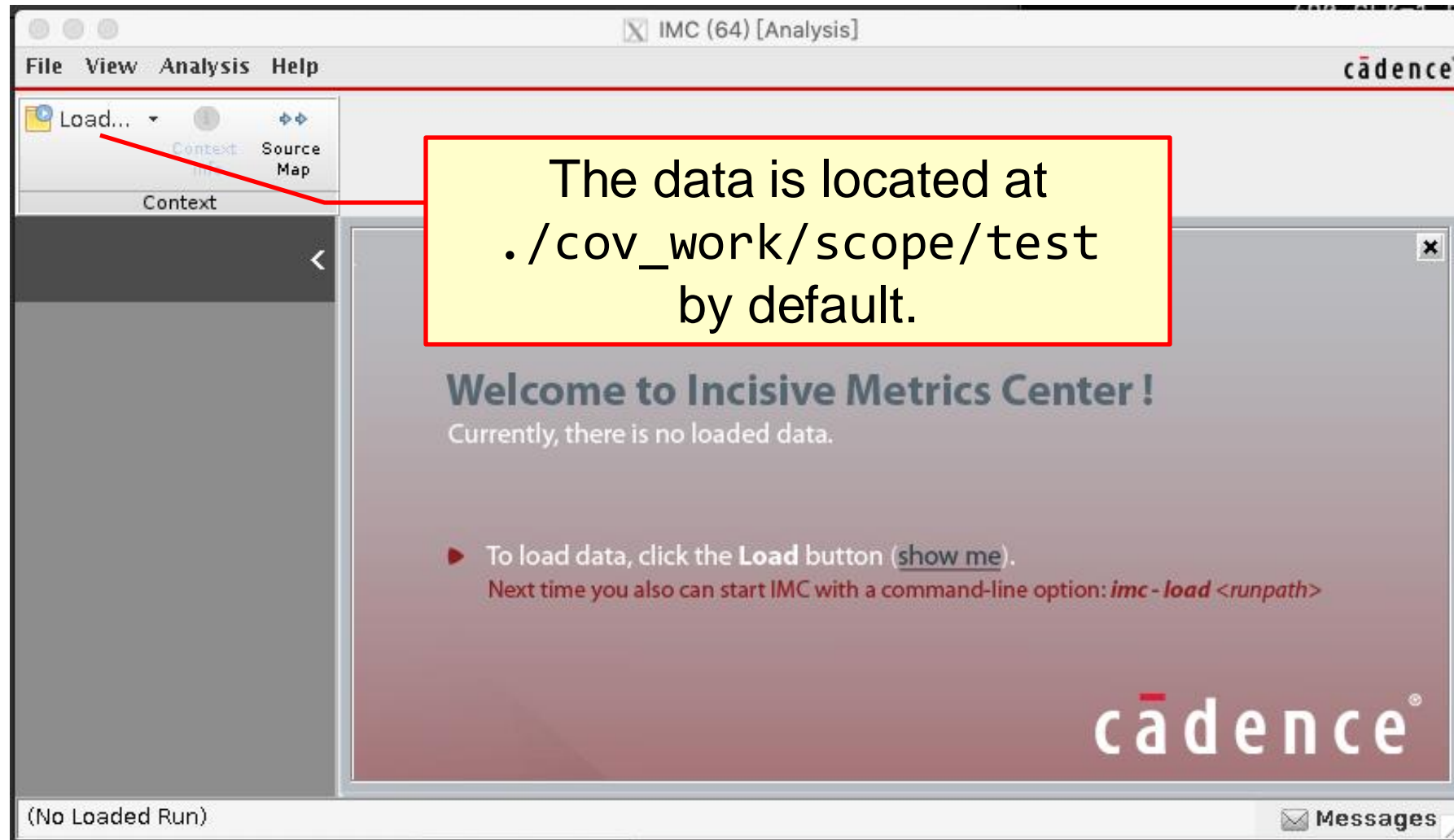
## ① -covtest <test>

- ◆ Test directory name
- ◆ Default: test

## ① -covoverwrite

- ◆ Enable overwriting of coverage files

# IMC: Load The Coverage Data





# IMC: Summary View

cadence®

**Activity Center" Navigation**

**Unified Instance/Type Verification Metrics Tree**

**Sorting / Filtering**

**Info Tab Sub-elements**

**Tools & Options**

**Details Tab**

Name	Overall Average Grade	Overall Covered
Verification Metrics	53.97%	1340 / 3249 (41.24%)
Types	53.38%	667 / 1610 (41.43%)
Instances	54.55%	673 / 1639 (41.06%)
uart_top	54.55%	673 / 1639 (41.06%)
wb_interface	67.44%	111 / 226 (49.12%)
regs	57.6%	497 / 1116 (44.53%)
dbg	29.84%	11 / 97 (11.34%)

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
wb_interface	67.44%	111 / 226 (...)	n/a
regs	57.6%	497 / 1116 ...	n/a
dbg	29.84%	11 / 97 (11....	n/a

Name	Overall Average Grade	Overall Covered
Overall	54.55%	673 / 1639 (41.06%)
Code	52.44%	643 / 1592 (40.39%)
Block	74.25%	306 / 437 (70.02%)
Statement	n/a	0 / 0 (n/a)
Expression	82.86%	34 / 42 (80.95%)
Toggle	29%	303 / 1113 (27.22%)
FSM	62.08%	30 / 47 (63.83%)
Functional	n/a	0 / 0 (n/a)
Assertion	n/a	0 / 0 (n/a)
CoverGroup	n/a	0 / 0 (n/a)



# IMC Block Analysis View

cadence®

Green Indicates covered ;  
Red indicates uncovered

Block Coverage Analysis

MC (64) [Analysis - Code: regs]  
View Analysis Help

Block Expression Toggle

Overall Local Grade: 62.96% | Block Local Grade: 85.26% | Expression Local Grade: 80% | Toggle Local Grade: 100%

Ln	Index	Block Type	Line	Score	Enclosing Entity
1		code block	406	1	uart_top regs
2		a case item of	408	1	uart_top regs
3		a case item of	409	1	uart_top regs
4		a case item of	410	0	uart_top regs
5		a case item of	411	1	uart_top regs
6		a case item of	412	0	uart_top regs
7		a case item of	413	0	uart_top regs
8		a case item of	414	0	uart_top regs
9		a case item of	415	1	uart_top regs
10		code block	422	1	uart_top regs
11		true part of	424	1	uart_top regs
12		false part of	426	1	uart_top regs
13		true part of	427	1	uart_top regs
14		false part of	429	1	uart_top regs
15		true part of	430	1	uart_top regs
16		implicit else	429	1	uart_top regs
17		code block	447	1	uart_top regs
18		true part of	449	1	uart_top regs
19		false part of	451	1	uart_top regs
20		code block	459	1	uart_top regs
21		true part of	461	1	uart_top regs
22		false part of	463	1	uart_top regs
23		true part of	464	1	uart_top regs
24		false part of	466	1	uart_top regs
25		true part of	467	0	uart_top regs
26		implicit else	466	1	uart_top regs
27		code block	476	1	uart_top regs
28		true part of	477	1	uart_top regs

Source

```
uart_regs.v | uart_top.v  
400 counter_t, rf_count, rf_data_out, rf_error_bit, rf_over  
401  
402  
403 // Asynchronous reading here because outputs are  
404 always @(dl or dlab or ier or iir or scratchpad or  
405 or lcr or lsr or msr or rf_data_out or wb_addr  
406 begin  
407 case (wb_addr_i)  
408 'UART_REG_RB : wb_dat_o = dlab ? dl["UART  
409 'UART_REG_IE : wb_dat_o = dlab ? dl["UART  
410 'UART_REG_II : wb_dat_o = {4'b1100,iir};  
411 'UART_REG_LC : wb_dat_o = lcr;  
412 'UART_REG_LS : wb_dat_o = lsr;
```

Attributes

Col #	Name	Value
	(no filter)	(no filter)
	Block Active	true
	Block Average Grade	100%
	Block Combined Covered	n/a
	Block Combined Covered Gr...	100%
	Block Combined Total	n/a
	Block Combined Uncovered	n/a
	Block Covered	1.0
	Block Covered Grade	100%



# IMC Expression Analysis View

cadence®

The screenshot displays the IMC Expression Analysis View in Cadence, showing various expressions, source code, and coverage tables. The interface includes a toolbar at the top with icons for actions like Show, Comment, Un, Clear, Approve, Clear orphan, Apply, Read, Save, and Reload. Below the toolbar, there are tabs for Expression, Statement, and Block. The main area is divided into several sections:

- Top Level Expressions:** A table listing expressions with columns for Index, Overall Average Grade, Overall Uncovered, Source Line, and Entity. The table shows 10 items, with the first item (Index 1) having a 100% grade and 0 uncovered.
- Source Code:** A window showing the source code for the selected expression. The code is in Verilog and includes a conditional statement: `if (wb_re_i && wb_addr_i == 'UART_REG_RB && !dlab) rf_pop <= #1; // advance read pointer`.
- Coverage Table with Output:** A table showing coverage data for the selected expression. It includes columns for Index, T1, T2, T4, Score, and Output. The table shows 4 items, with the first item (Index 1) having a score of 1 and output of 1.
- Coverage Tables:** A summary table showing the overall average grade and overall uncovered status for the selected expression. It shows a 100% grade and 0 uncovered.
- Attributes:** A table showing various attributes for the selected expression, including Code Average Grade (100%), Code Covered (4.0), Code Covered Grade (100%), and Code Excluded (0.0).

Callouts highlight specific features: "Various Expressions" points to the Top Level Expressions table; "Selected Expression" points to the first item in the table; "Source Code" points to the source code window; and "Selected Expression Table" points to the Coverage Table with Output.





# IMC Toggle Analysis View

cadence®

**Source Code**

Instance (default scope): `uart_top` `regs`

Overall Local Grade: 62.96% | Code Local Grade: 62.96% | Block Local Grade: 85.26% | Expression Local Grade: 80% | Toggle Local Grade: 23.62%

**Block Expression Toggle Statement**

**Variables**

Name	Range	Overall Average Grade	Overall Covered	Enclosing Entity
clk	(no filter)	100%	1 / 1 (100%)	uart_top.regs
wb_rst_i	(no filter)	100%	1 / 1 (100%)	uart_top.regs
wb_addr_i	[4:0]	40%	2 / 5 (40%)	uart_top.regs
wb_dat_i	[7:0]	62.5%	5 / 8 (62.5%)	uart_top.regs
wb_dat_o	[7:0]	62.5%	5 / 8 (62.5%)	uart_top.regs
wb_we_i	(no filter)	100%	1 / 1 (100%)	uart_top.regs
wb_re_i	(no filter)	100%	1 / 1 (100%)	uart_top.regs
stx_pad_o	(no filter)	0%	0 / 1 (0%)	uart_top.regs
srx_pad_i	(no filter)	100%	1 / 1 (100%)	uart_top.regs
ier	[3:0]	0%	0 / 4 (0%)	uart_top.regs
ier	[3:0]	0%	0 / 1 (0%)	uart_top.regs
ier	[3:0]	0%	0 / 1 (0%)	uart_top.regs
ier	[3:0]	0%	0 / 1 (0%)	uart_top.regs
ier	[3:0]	0%	0 / 4 (0%)	uart_top.regs

**Individual Vector bits**

Showing 112 items

**Signals**

Name	Score	Value
wb_dat_o[7]	1	1
wb_dat_o[6]	0	0
wb_dat_o[5]	0	0
wb_dat_o[4]	0	0
wb_dat_o[3]	1	1
wb_dat_o[2]	1	1
wb_dat_o[1]	1	1
wb_dat_o[0]	1	1

**Attributes**

Name	Value
Code Average Grade	62.5%
Code Covered	5.0
Code Covered Grade	62.5%
Code Excluded	0.0
Code Total	8.0
Code Total Weighted Coverage	5.0
Code Total Weights	8.0
Code Uncovered	3.0



# IMC FSM Analysis View

The screenshot displays the Cadence IMC FSM Analysis View for a component named 'uart\_receiver'. The interface includes a menu bar, a toolbar, and several panels.

**FSMs Panel:** Shows the overall analysis results for the 'uart\_receiver' component.

FSMs	Name	State Average Grade	Transition Average Grade	Arc Average Grade
(no filter)	rstate	100%	80%	n/a

**State Transitions / Arcs Panel:** A table listing the states and transitions of the FSM.

Name	Encoding	Score	Reset State
sr_idle	0000	2	true
sr_rec_start	0001	1	false
sr_rec_prepare	0110	1	false
sr_rec_bit	0010	8	false
sr_end_bit	0111	8	false
sr_rec_parity	0011	1	false
sr_ca_lc_parity	1000	1	false
sr_check_parity	0101	1	false
sr_wait1	1001	1	false
sr_rec_stop	0100	1	false
sr_push	1010	1	false

**Bubble diagram:** A visual representation of the FSM states and transitions. States are represented as bubbles, and transitions are shown as arrows. A callout points to this diagram with the text "FSM bubble diagram".

**Source viewer with syntax highlighting:** A panel showing the source code for the 'sr\_rec\_start' state. The code is syntax-highlighted. A callout points to this panel with the text "Source viewer with syntax highlighting".

```
301 rstate <= #1 sr_rec_start;
302 end
303 end
304 sr_rec_start : begin
305   rf_push <= #1 1'b0;
306   if (rcounter16_eq_7) // check the pulse
307     // (code continues)
```



# Example Case



# Example Case

The screenshot displays the IMC (64) [Analysis - Metrics] application window. The interface includes a menu bar (File, View, Analysis, Help), a toolbar with various analysis tools, and a main workspace divided into several panels.

**Verification Hierarchy Panel:**

Ex	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		Verification Metrics	86.78% 336 / 388 (86.6%)	
		Types	86.78% 168 / 194 (86.6%)	
		Instances	86.78% 168 / 194 (86.6%)	
		stimulus	86.78% 168 / 194 (86.6%)	
		gcd01	87.35% 106 / 124 (85.4...)	

Showing 5 items

**Info Tabs of: gcd01**

**Relative Elements Panel:**

Recursive

Ex	UNR	Name	Overall Average Grade	Overall Covered	Assertion St
		(no filter)	(no filter)	(no filter)	

Showing 0 items

**Details of: gcd01**

**Metrics Panel:**

Ex	UNR	Name	Overall Average Grade	Overall Covered
		Overall	87.35%	106 / 124 (85.48%)
		Code	87.19%	100 / 117 (85.47%)
		Block	96.43%	27 / 28 (96.43%)
		Expression	83.33%	10 / 12 (83.33%)
		Toggle	81.82%	63 / 77 (81.82%)
		FSM	87.5%	6 / 7 (85.71%)
		Functional	n/a	0 / 0 (n/a)
		Assertion	n/a	0 / 0 (n/a)
		CoverGroup	n/a	0 / 0 (n/a)

Showing 9 items

Loaded Run: /users/teacher/cthuang/workbench/cs5121-2012/gcd/cov\_work/scope/test

# Example Case: Block Coverage

Instance (default scope): stimulus gcd01

Overall Local Grade: 87.35% | Code Local Grade: 87.19% | Block Local Grade: 96.43% | Expression Local Grade: 83.33% | Toggle Local Grade: 81.82% | Edit...

Block Expression Toggle

Blocks

Ex	UNB	Index	Block Type	Source Line	Score
			(no filter)	(no filter)	(no filter)
		1	code block	37	✓ 1
		2	true part of	38	✓ 1
		3	false part of	41	✓ 1
		4	code block	47	✓ 1
		5	code block	51	✓ 1
		6	true part of	52	✓ 1
		7	false part of	54	✓ 1
		8	true part of	54	✓ 1
		9	implicit else	54	✓ 1
		10	code block	59	✓ 1
		11	true part of	60	✓ 1
		12	false part of	63	✓ 1
		13	true part of	63	✓ 1
		14	false part of	66	✓ 1
		15	code block	72	✓ 1
		16	true part of	73	✓ 1
		17	false part of	76	✓ 1
		18	code block	82	✓ 1
		19	a case item of	84	✓ 1
		20	true part of	86	✓ 1
		21	false part of	89	✓ 1
		22	a case item of	94	✓ 1
		23	true part of	97	✓ 1
		24	false part of	99	✓ 1
		25	true part of	99	✓ 1
		26	false part of	101	✗ 0
		27	a case item of	105	✓ 1
		28	a case item of	110	✓ 1

Showing 28 items

Source of: Block26

gcd.v x gcd\_t.v x

```
91     error_next = 0;
92     end
93     end
94     CALC: begin
95         DONE = 0;
96         error_next = ERROR;
97         if (found || ERROR) begin
98             state_next = FINISH;
99         end else if (!found && !ERROR) begin
100             state_next = CALC;
101         end else begin
102             state_next = IDLE;
103         end
104     end
105     FINISH: begin
106         error_next = 0;
107         state_next = IDLE;
108         DONE = 1'b1;
109     end
110     default: begin
111         DONE = 0;
112         state_next = IDLE;
```

Attributes of: Block26

Col #	Name	Value
		(no filter)
	Block Active	true
	Block Average Grade	0%
	Block Covered	0.0
	Block Excluded	0.0
	Block Total	1.0
	Block Total Weighted Coverage	0.0
	Block Total Weights	1.0
3	Block Type	false part of

# Example Case: Expression Coverage

Instance (default scope): stimulus gcd01

Overall Local Grade: 87.35% | Code Local Grade: 87.19% | Block Local Grade: 96.43% | Expression Local Grade: 83.33% | Toggle Local Grade: 81.82% Edit...

Block Expression Toggle

### Top Level Expressions

Ex	UNR	Index	Overall Average Grade (no filter)	Overall Uncovered (no filter)	Source Line (no filter)
		1	100%	0	34
		2	100%	0	88
		3	100%	0	97
		4	33.33%	2	99

Showing 4 items

### Coverage Tables

Ex	UNR	Index	Overall Average Grade (no filter)	Overall Uncovered (no filter)
		1	33.33%	2

Showing 1 items

### Info Tabs of: 4

Source Expression Hierarchy Full Expression

gcd.v gcd\_t.v

```
94 CALC: begin
95   DONE = 0;
96   error_next = ERROR;
97   if (found || ERROR) begin
98     state_next = FINISH;
99   end else if (!found && !ERROR) begin
100     state_next = CALC;
101   end else begin
102     state_next = IDLE;
103   end
104 end
```

### Coverage Table with Output

Ex	UNR	T1	T2	Score (no filter)	Output
		0	0	1	1
		-	1	0	0
		1	-	0	0

Showing 3 items

### Attributes of: 4

Col #	Name (no filter)	Value (no filter)
	Code Average Grade	33.3333%
	Code Covered	1.0
	Code Excluded	0.0
	Code Total	3.0
	Code Total Weighted Coverage	1.0

# Example Case: Toggle Coverage

Instance (default scope): stimulus gcd01

Overall Local Grade: 87.35% | Code Local Grade: 87.19% | Block Local Grade: 96.43% | Expression Local Grade: 83.33% | Toggle Local Grade: 81.82% | Edit..

Block Expression Toggle

### Variables

Ex	UNR	Name	Range	Overall Average G	Overall Coverage
		(no filter)		(no filter)	(no filter)
		A	[7:0]	62.5%	5 / ...
		B	[7:0]	75%	6 / ...
		START		100%	1 / ...
		Y	[7:0]	62.5%	5 / ...
		DONE		100%	1 / ...
		ERROR		100%	1 / ...
		found		100%	1 / ...
		err	Signal: ERROR	0%	0 / ...
		swap		100%	1 / ...
		reg_a	[7:0]	87.5%	7 / ...
		reg_b	[7:0]	75%	6 / ...
		data_a	[7:0]	87.5%	7 / ...
		data_b	[7:0]	87.5%	7 / ...
		diff	[7:0]	100%	8 / ...
		error_next		100%	1 / ...
		state	[1:0]	100%	2 / ...
		state_next	[1:0]	100%	2 / ...

Showing 19 items

### Source of: err

```
gcd.v | gcd.tv |
14 input wire CLK,
15 input wire RST_N,
16 input wire [7:0] A,
17 input wire [7:0] B,
18 input wire START,
19 output reg [7:0] Y,
20 output reg DONE,
21 output reg ERROR
22 };
23
24 wire found, err, swap;
25 reg [7:0] reg_a, reg_b, data_a, data_b;
26 reg [7:0] diff;
27 reg error_next;
28 reg [1:0] state, state_next;
29
30 parameter [1:0] IDLE = 2'b00;
31 parameter [1:0] CALC = 2'b01;
32 parameter [1:0] FINISH = 2'b10;
33
34 assign found = (reg_a == reg_b || A == B) ? 1'b1 : 0;
35 assign err = (reg_b > reg_a) ? 1'b1 : 0;
```

### Signals of: err

Ex	UNR	Name	Score
		(no filter)	(no filter)
		err	0

Showing 1 items

### Attributes of: err

Col #	Name	Value
	(no filter)	(no filter)
	Code Average Grade	0%
	Code Covered	0.0
	Code Excluded	0.0
	Code Total	1.0
	Code Total Weighted Coverage	0.0
	Code Total Weights	1.0
	Code Uncovered	1.0
	Enclosing Entity	stimulus.gcd01
	Exclusion Comment	

# Example Case: FSM Coverage

Instance (default scope) : stimulus gcd01

Overall Local Grade: 87.35% | FSM Local Grade: 87.5% | Transition Local Grade: 75% | State Local Grade: 100% | Arc | Edit...

FSMs

Ex	UNR	Name	State Average (no filter)	Transition Average (no filter)	Average (no filter)
		(no filter)			
		state	100%	75%	n/a

Bubble diagram of: state

Showing 1 items

Info tabs of: state

States

Ex	UNR	Name	Encoding	Is Reset S
		(no filter)	(no filter)	(no filter)
		IDLE	00	true
		CALC	01	false
		FINISH	10	false

Showing 3 items

Details of: CALC -> IDLE

Arc Conditions

Ex	Score
	(no filter)

Showing 0 items

Loaded Run: /users/teacher/cthuang/workbench/cs5121-2012/gcd/cov\_work/scope/test

Messages

# Example Case: FSM Coverage

Instance (default scope) : stimulus ▶ gcd01

Overall Local Grade: 87.35% | FSM Local Grade: 87.5% | Transition Local Grade: 75% | State Local Grade: 100% | Arc ▶ Edit...

**FSMs**

Ex	UNR	Name	State Average	Transition Average	Arc Average
		(no filter)	(no filter)	(no filter)	(no filter)
		state	100%	75%	n/a

Showing 1 items

**Bubble diagram of: state**

search >>

**Info tabs of: state**

**Transitions & Arc Sources**

Ex	UNR	From State Name	To State Name	S...	Is Reset Trans
		(no filter)	(no filter)	(no filter)	(no filter)
		IDLE	CALC	✓	false
		CALC	FINISH	✓	false
		CALC	IDLE	!	false
		FINISH	IDLE	✓	false

Showing 4 items

**Details of: CALC -> IDLE**

**Arc Conditions** **Attributes** **Source**

```
gcd.v
94 CALC: begin
95   DONE = 0;
96   error_next = ERROR;
97   if (found || ERROR) begin
98     state_next = FINISH;
99   end else if (!found && !ERROR) begin
100     state_next = CALC;
101   end else begin
```

Loaded Run: /users/teacher/cthuang/workbench/cs5121-2012/gcd/cov\_work/scope/test

Messages



# Summary

## ⦿ What code coverage **does**

- ◆ Evaluating the test suite – how much does it cover?
- ◆ Evaluating the design – redundant part or untestable part

## ⦿ What code coverage **does not**

- ◆ The test of 100% code coverage is not necessary a good test!
- ◆ The design being tested with 100% code coverage is not necessary a correct design!

## ⦿ To improve the code quality

- ◆ Achieve as higher coverage as you can