CS5120 VLSI System Design, Spring 2025

Code Coverage Analysis

黃稚存

Chih-Tsun Huang

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

聲明

- 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

- Introduction to Code Coverages
- Code Coverage Tool
- 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

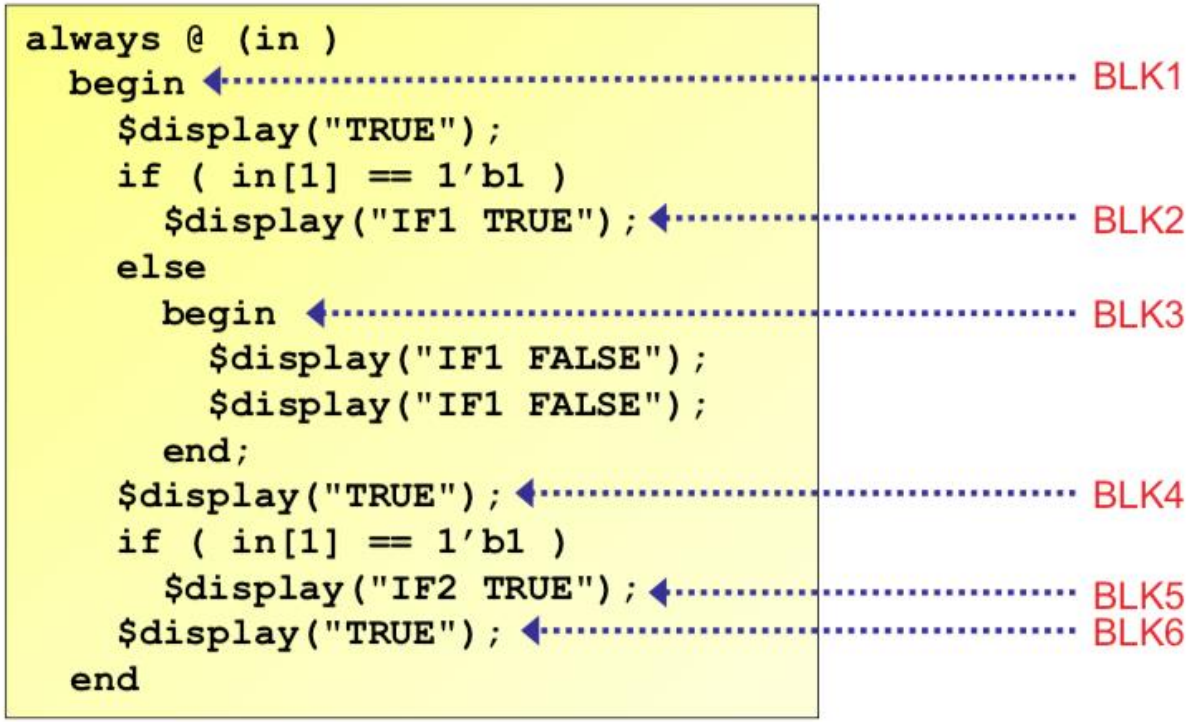
cadence

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

cadence

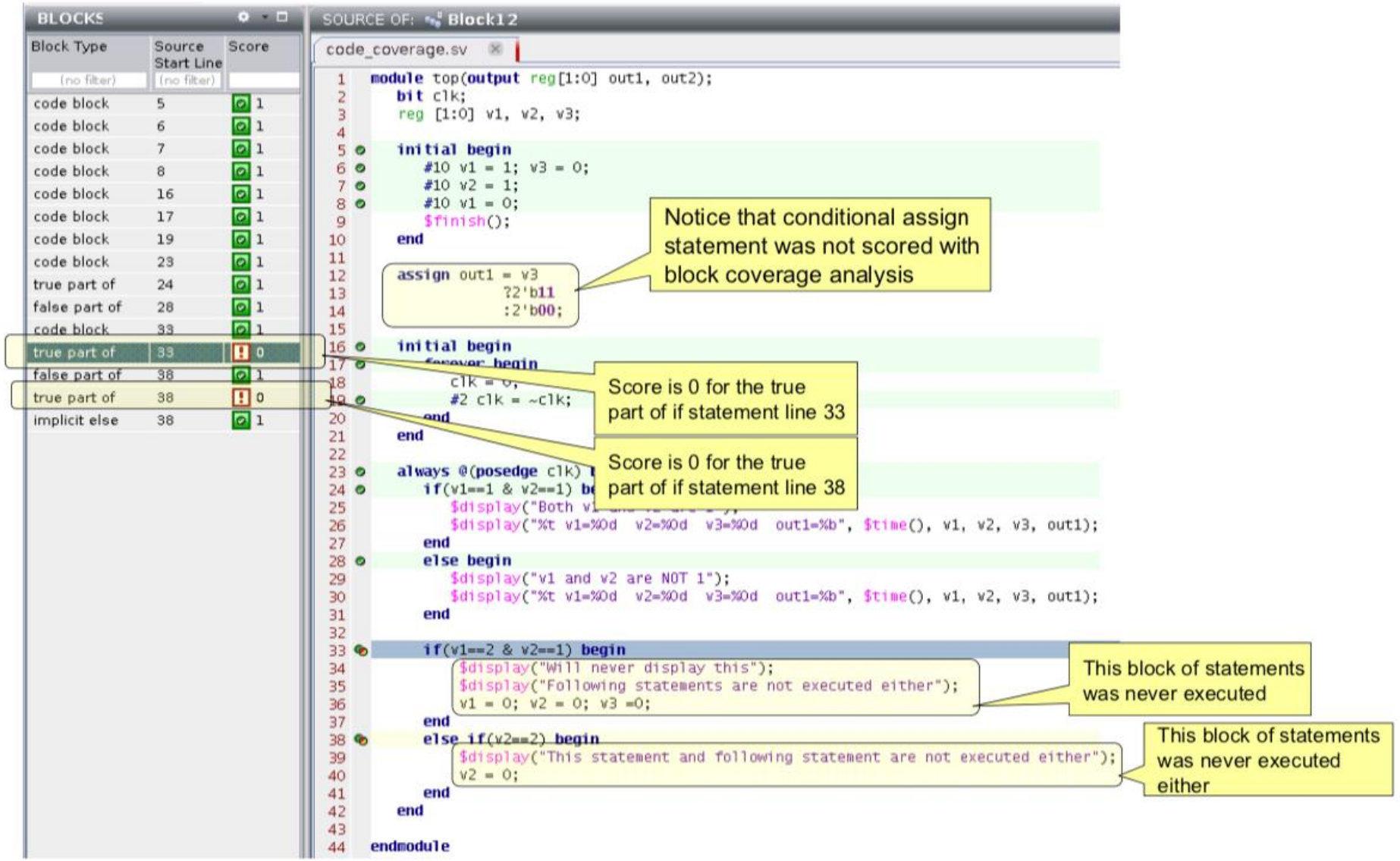
- Branch Coverage: Yields more precise coverage details than block coverage by obtaining coverage results for various branches individually.
- With branch coverage, a piece of code is considered 100% covered when each branch of a conditional statement has been executed at least once.

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

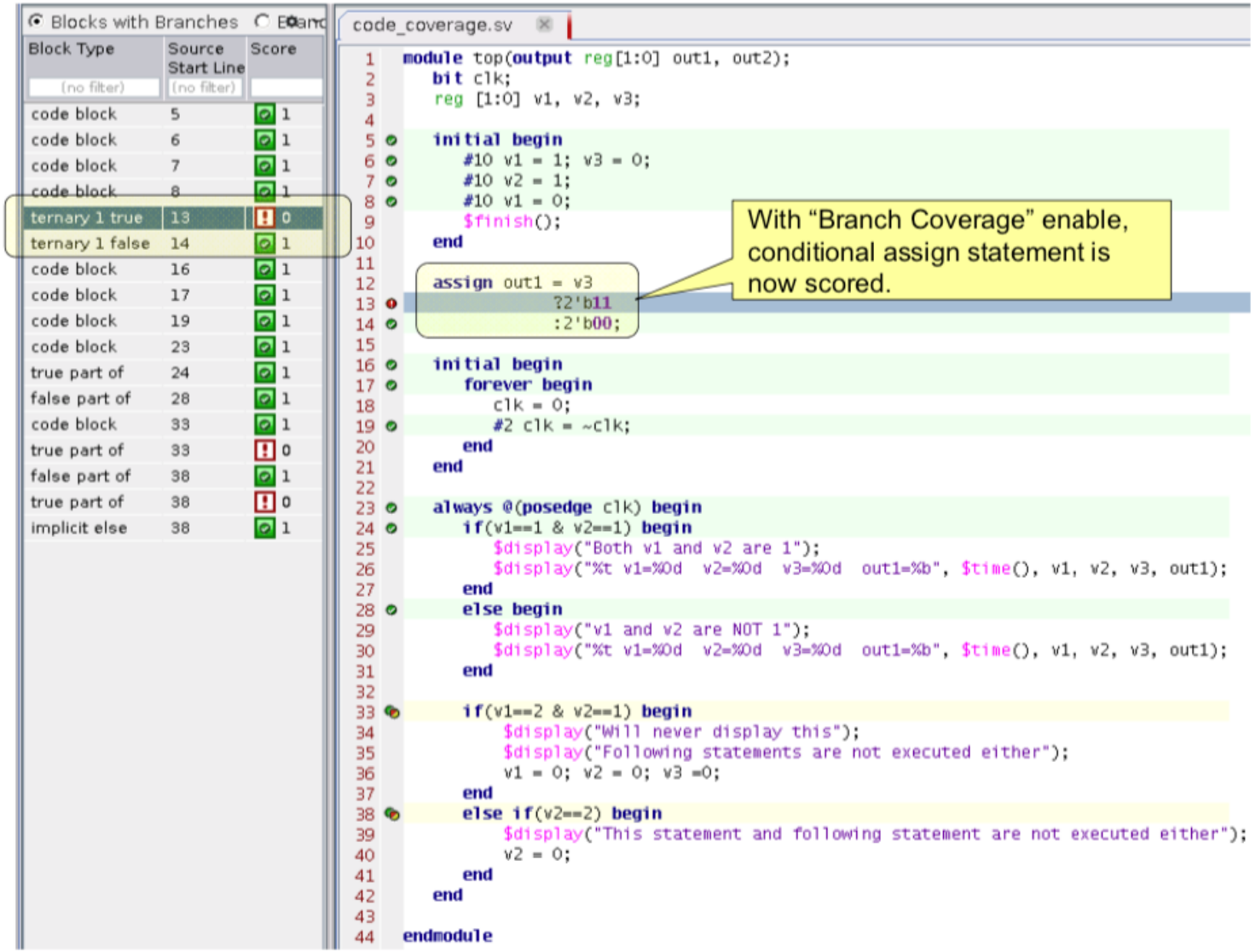
The above statement has the following branches:
- If cond1 evaluates to true, out2 is assigned 1'b1
- If cond1 evaluates to false, out2 is assigned 1'b0

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



Block Coverage Analysis Report




```
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
```

Code Coverage: Expression Coverage

cadence

- 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

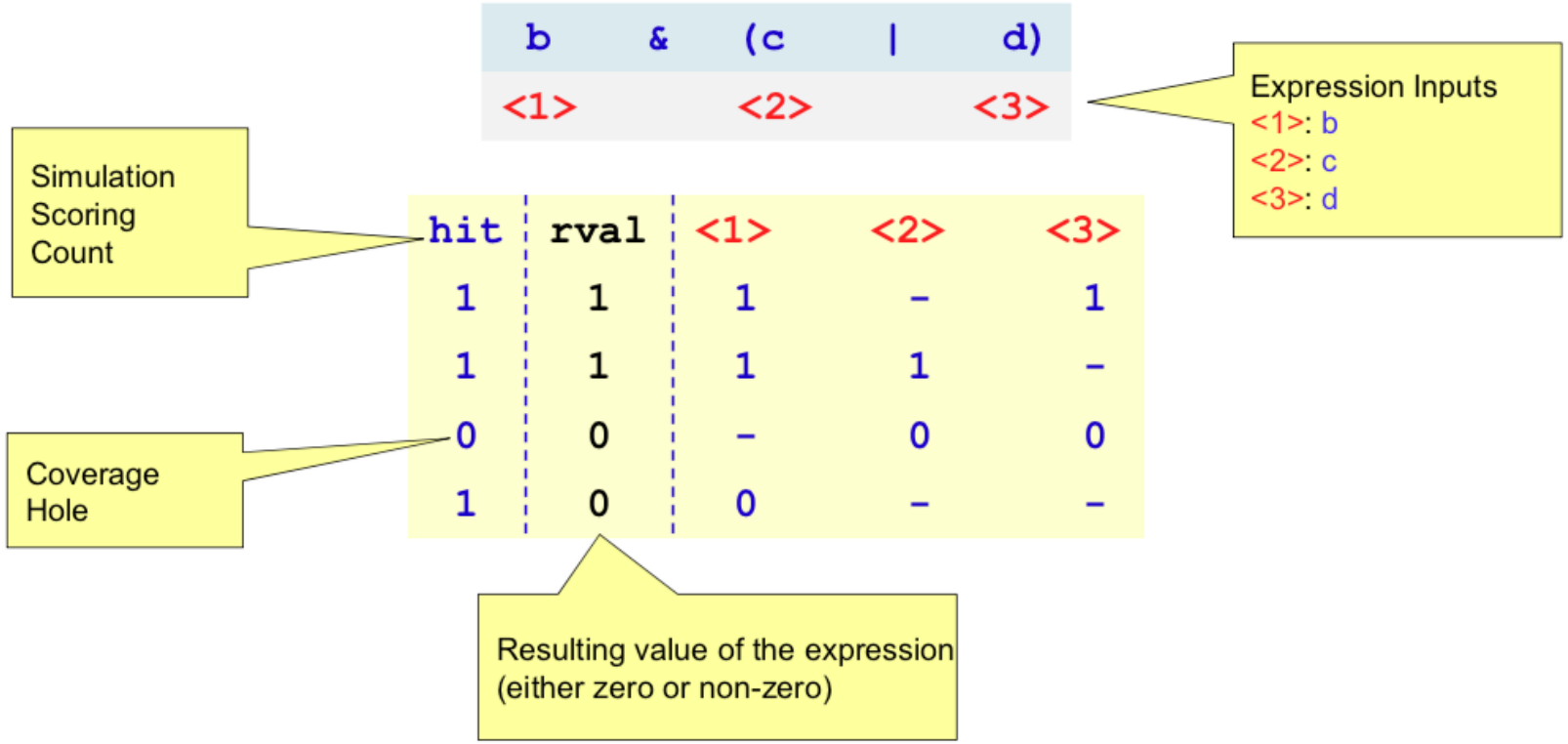
cadence

- **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

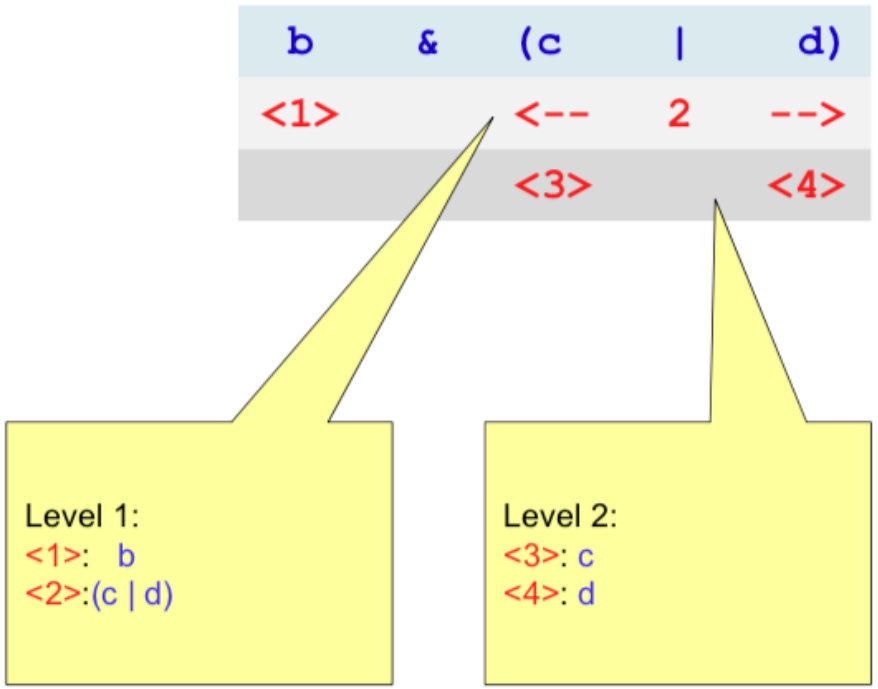
- Reduces expressions to a minimum set of expression inputs that make the expression both true and false, inherently first-level



Control Scoring

cadence

- 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



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

cadence

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

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

Detailed expression reports

& Coverage Table

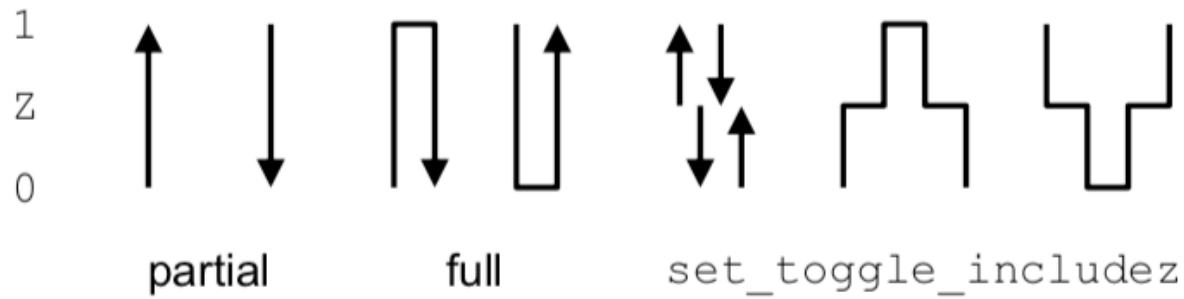
T1	T2	Score
0	-	0
-	0	0
1	1	1

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

Coverage Metrics: Toggle Coverage

cadence

- *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

cadence

- 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

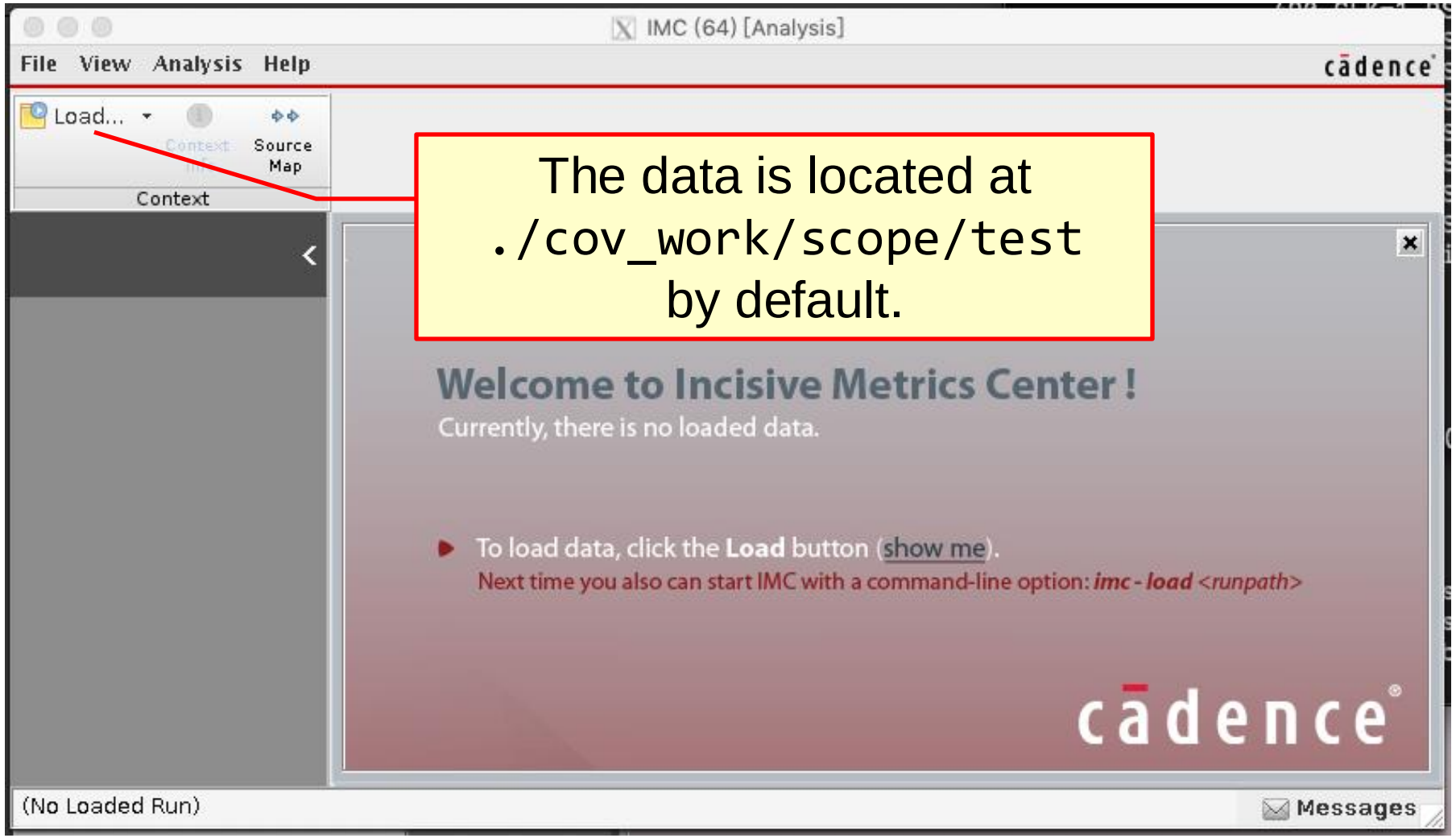
- -coverage <coverage_types>
  - Enable coverage data generation
    - -coverage all
    - -coverage A
    - -coverage B:E

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

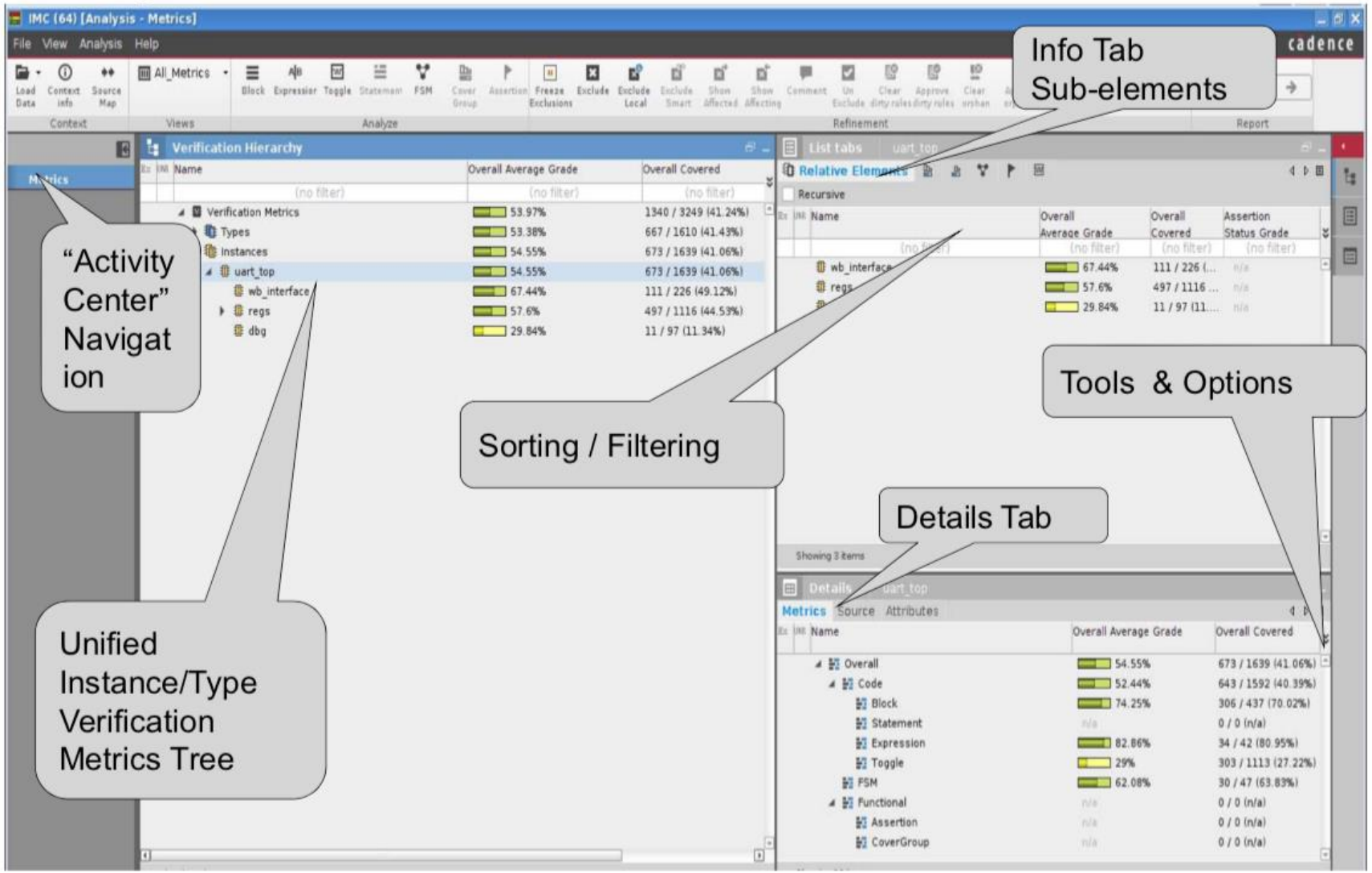
More about The Options

- -covworkdir <workdir>
  - Basename for the work directory
  - Default: cov_work
- -covscope <scope>
  - Specifies an alternate directory for storing coverage model files
  - Default: scope
- -covtest <test>
  - Test directory name
  - Default: test
- -covoverwrite
  - Enable overwriting of coverage files

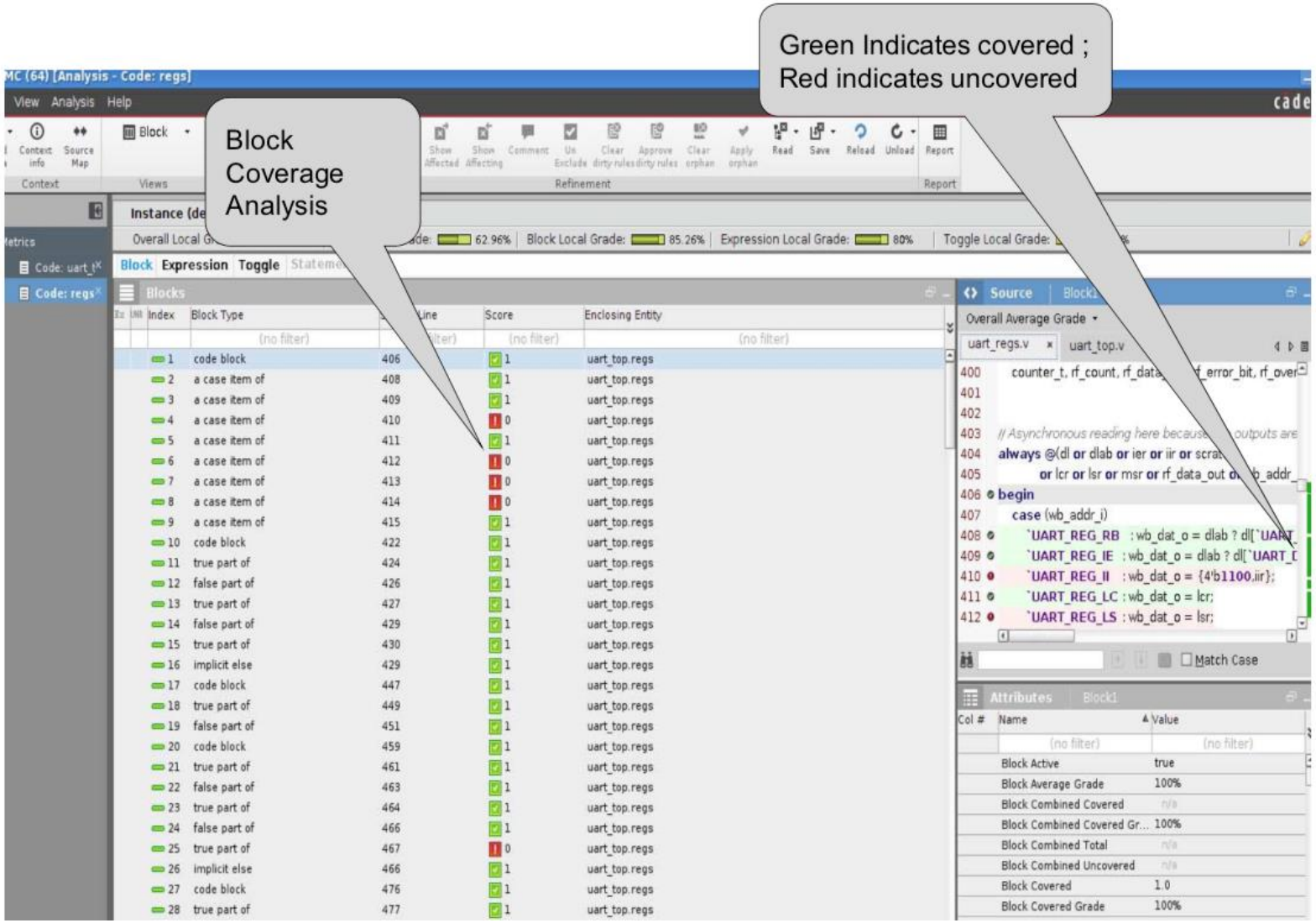
IMC: Load The Coverage Data



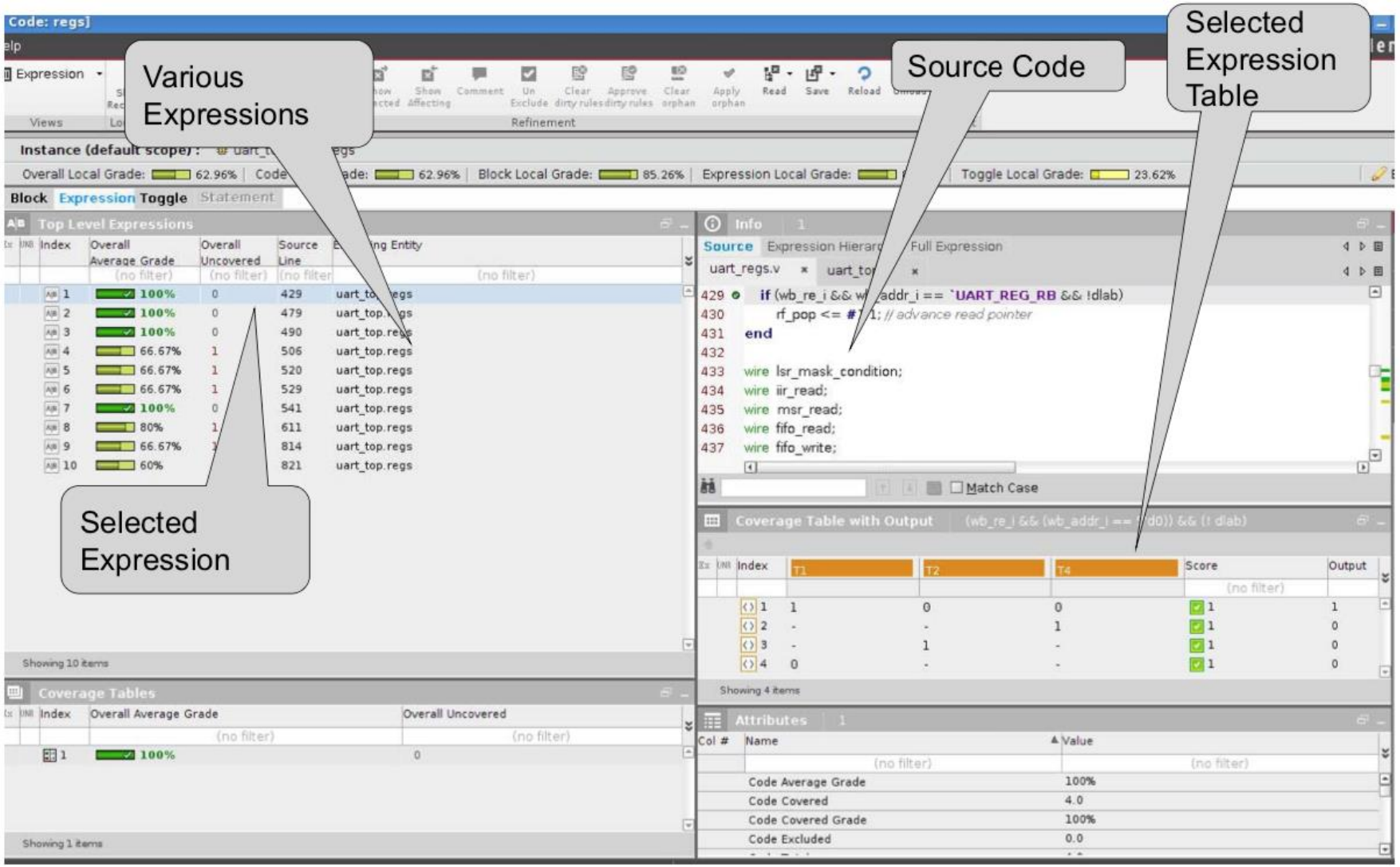
IMC: Summary View



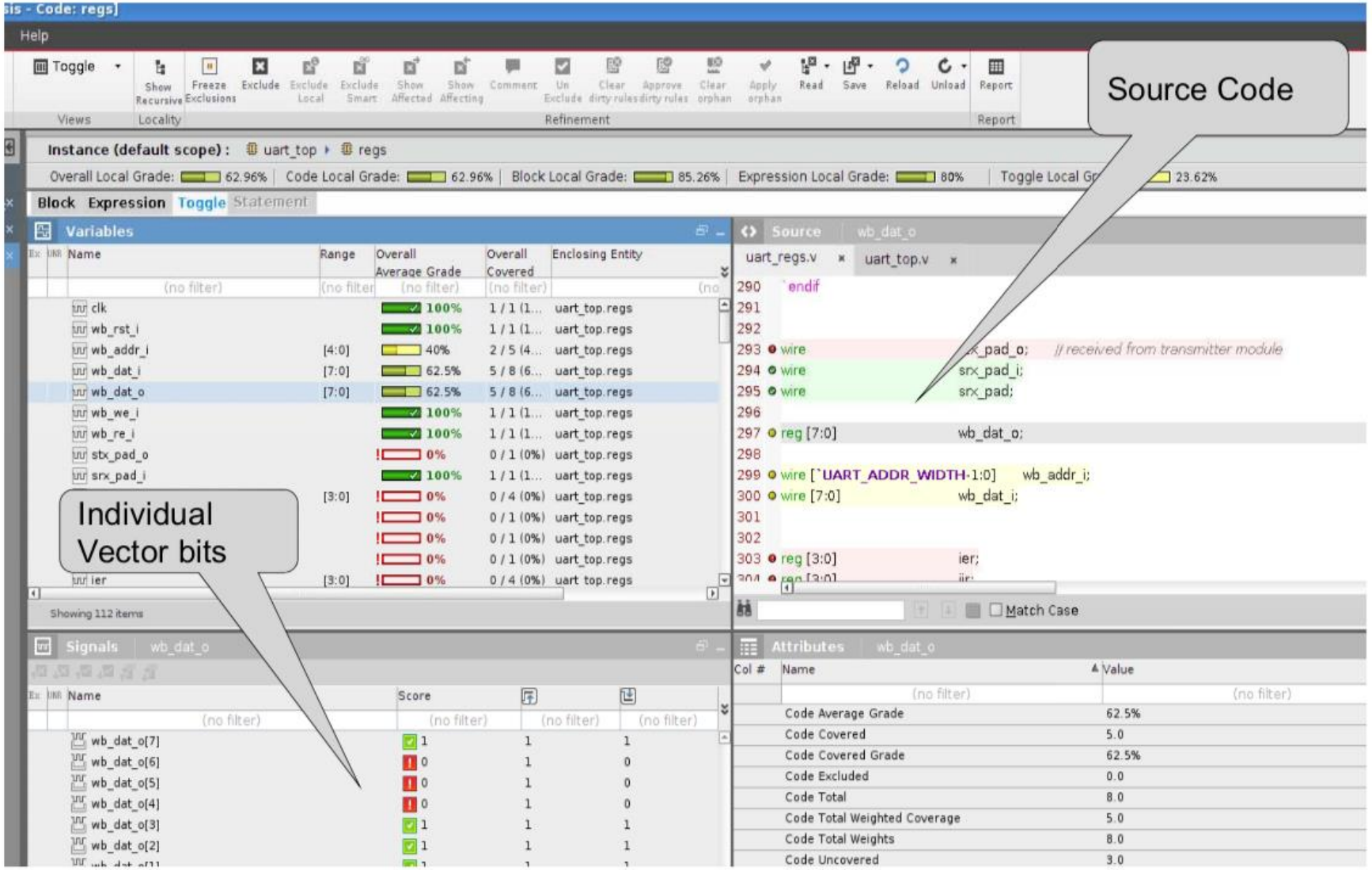
IMC Block Analysis View



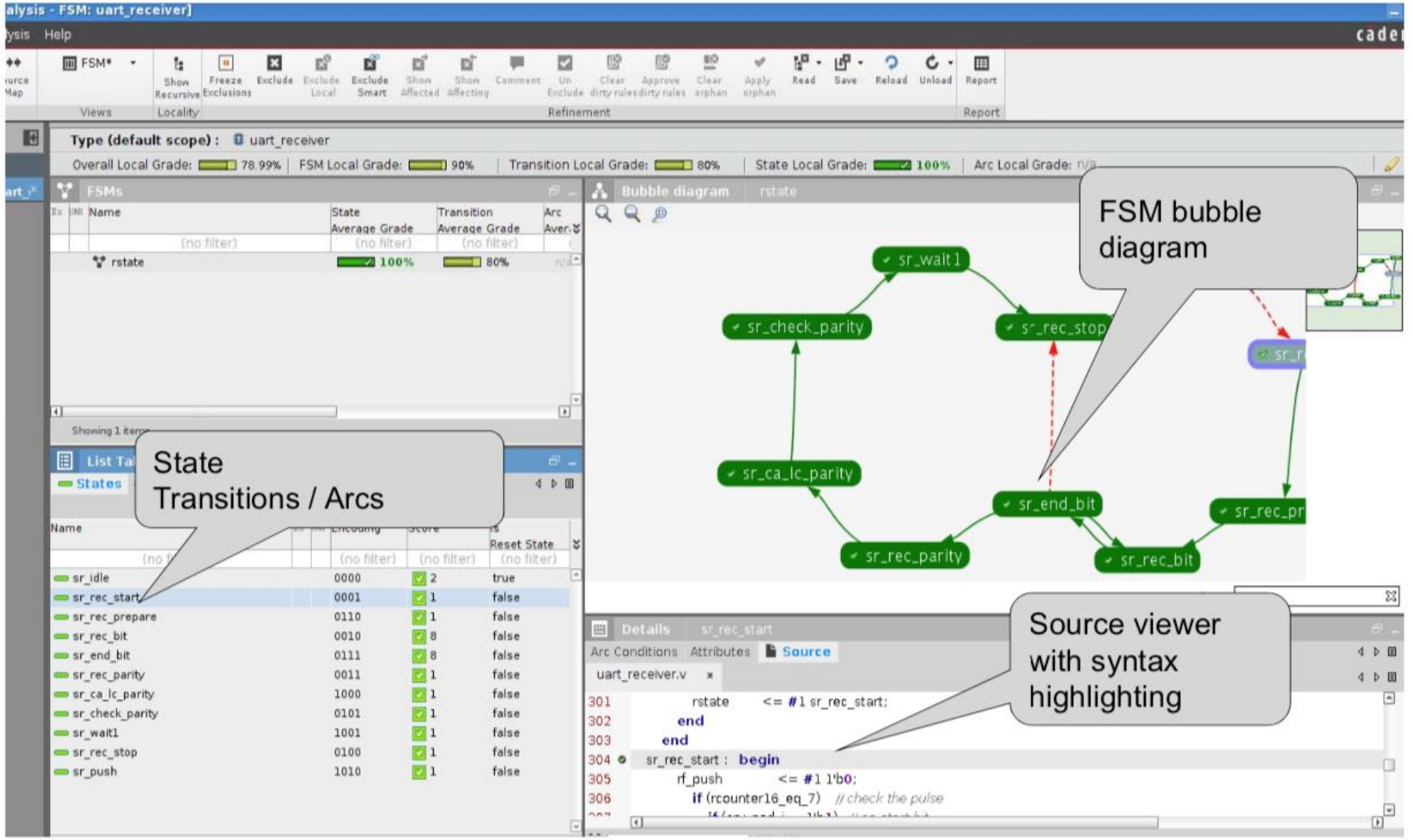
IMC Expression Analysis View



IMC Toggle Analysis View

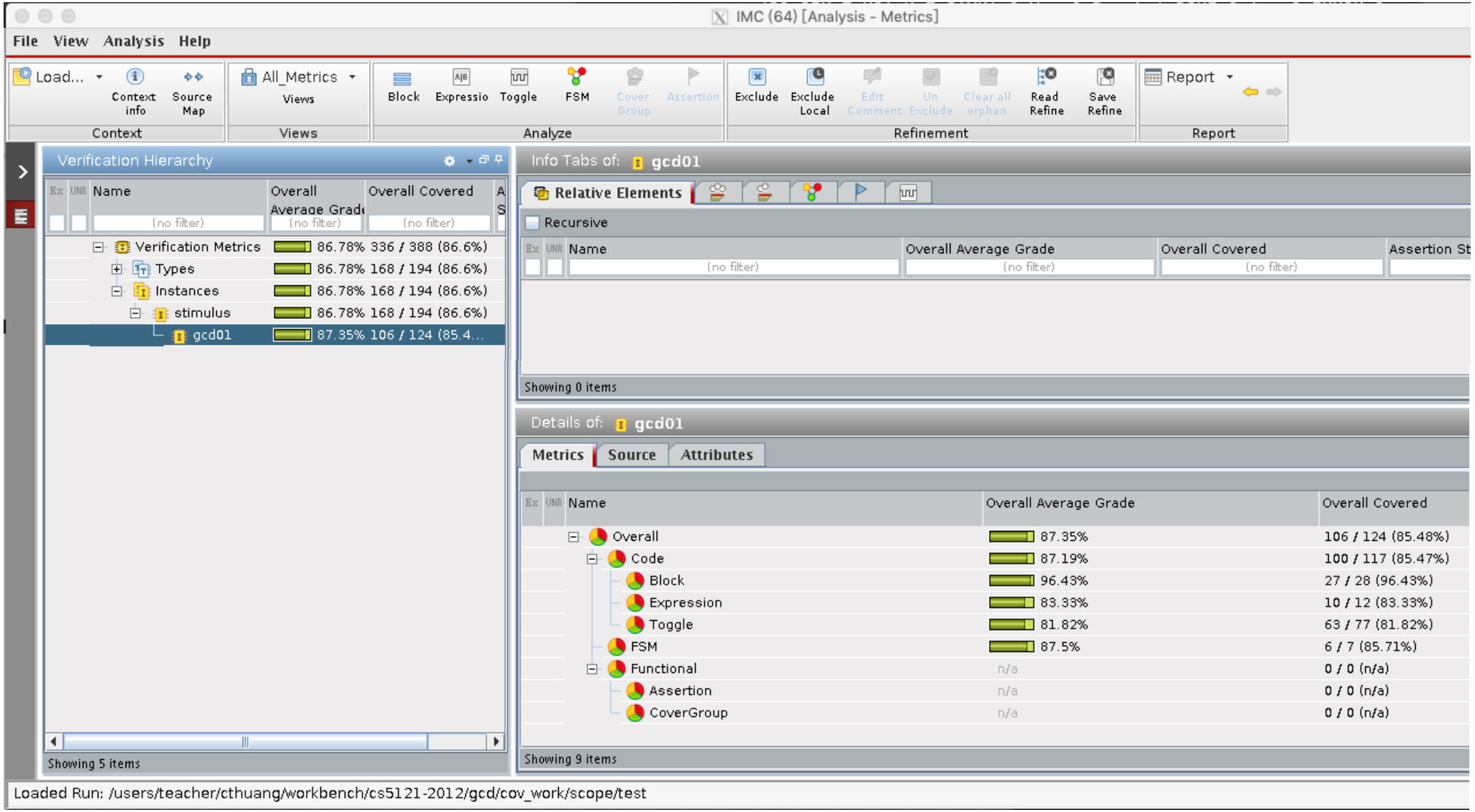


IMC FSM Analysis View

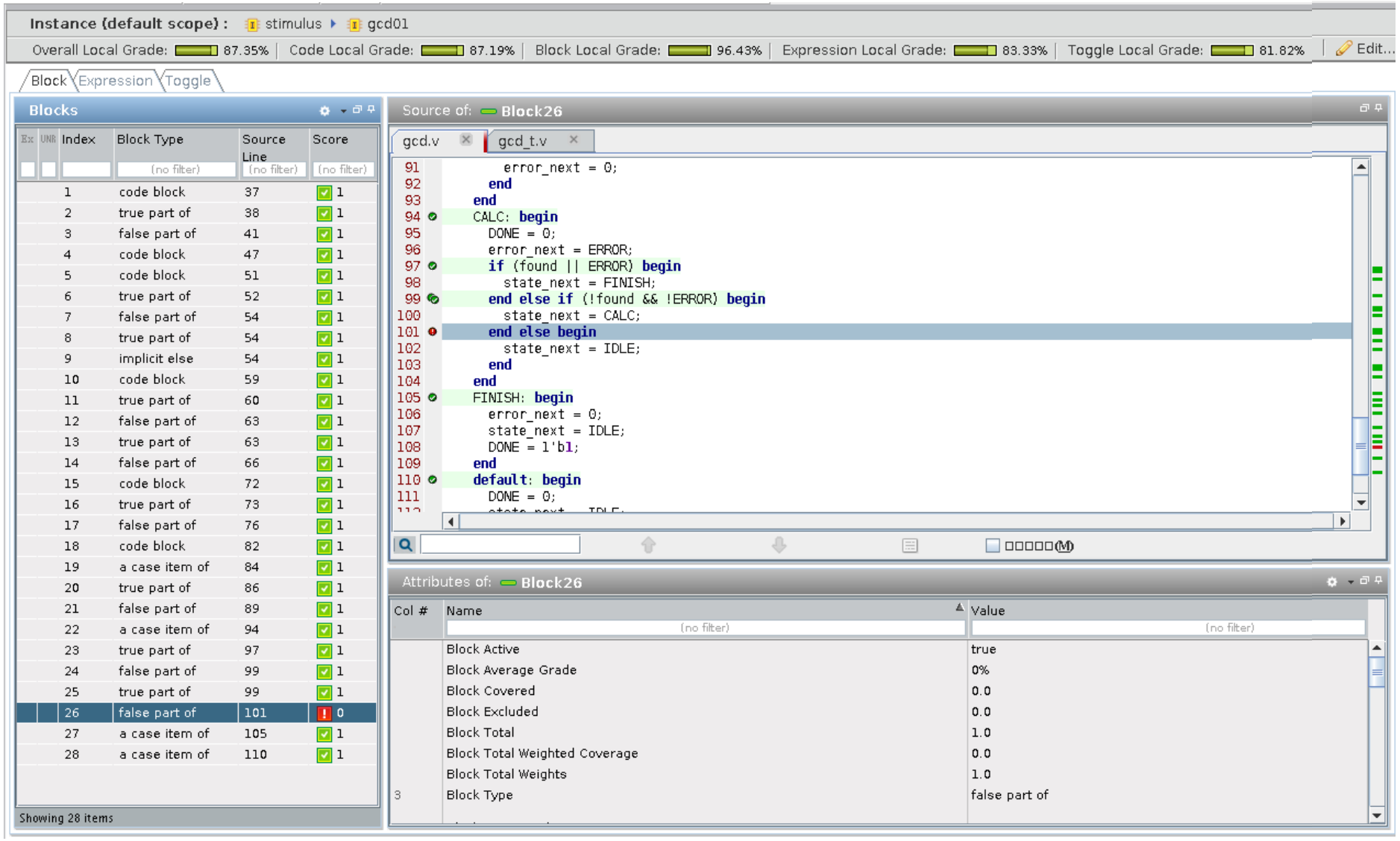


Example Case

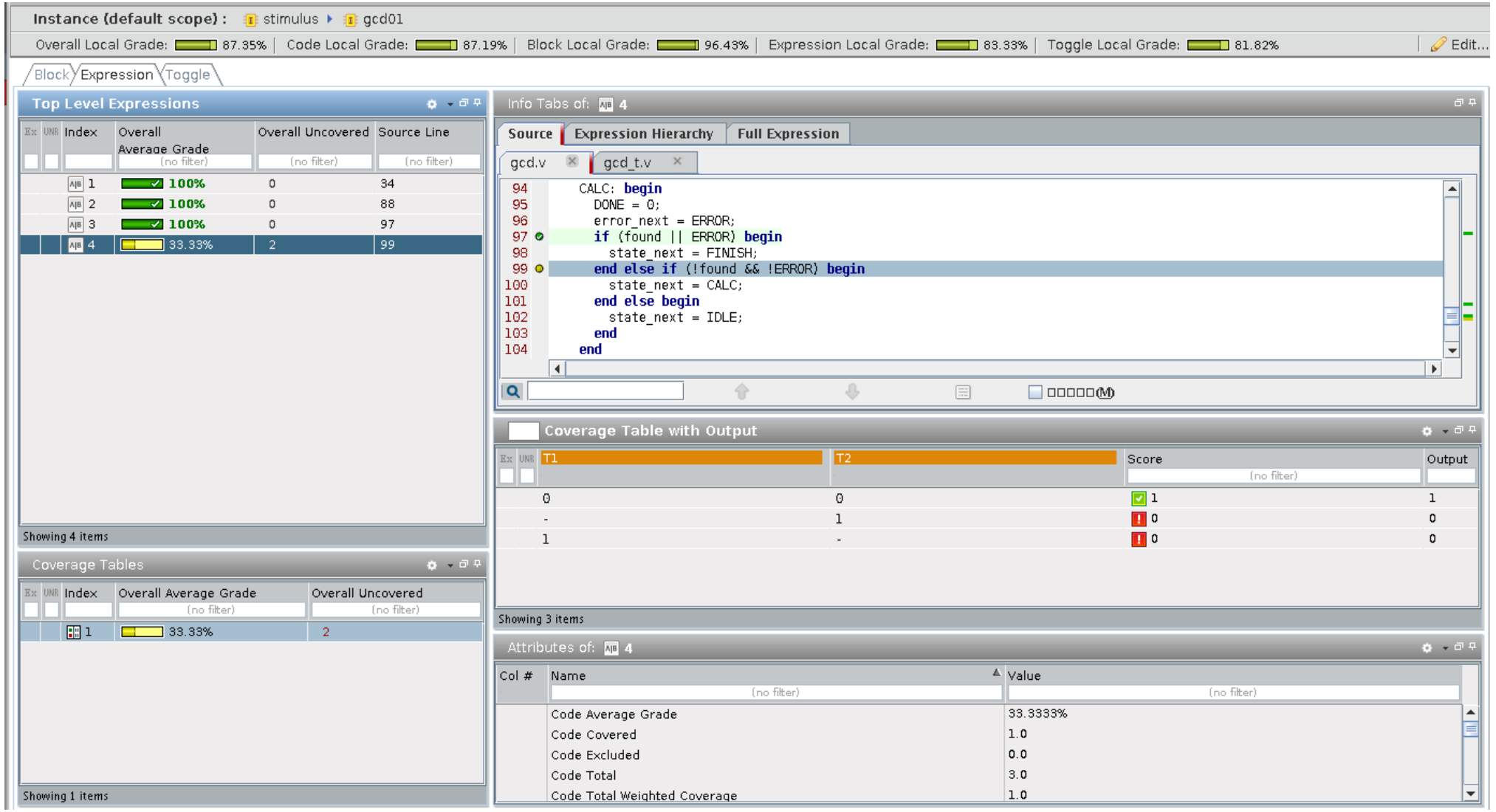
Example Case



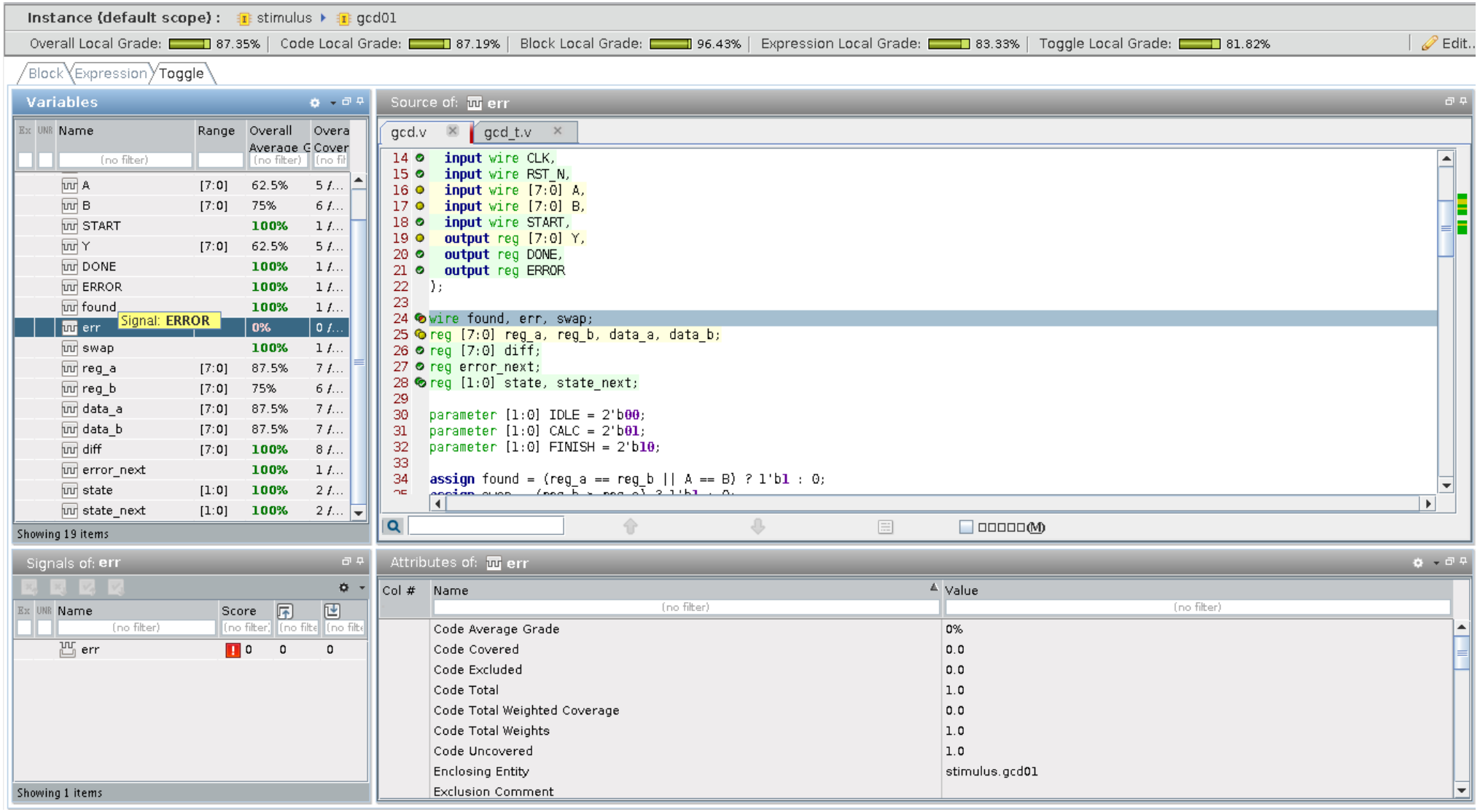
Example Case: Block Coverage



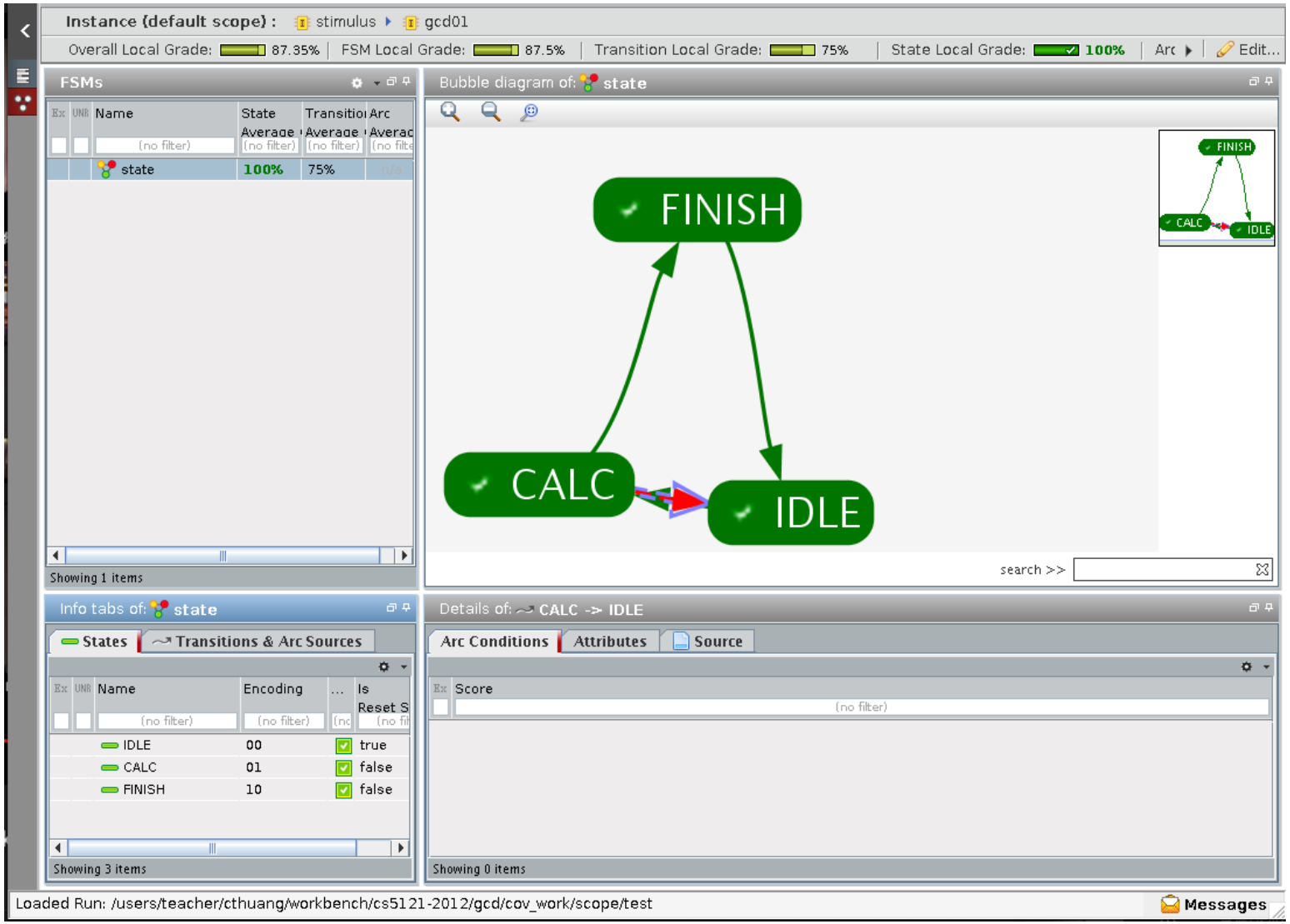
Example Case: Expression Coverage



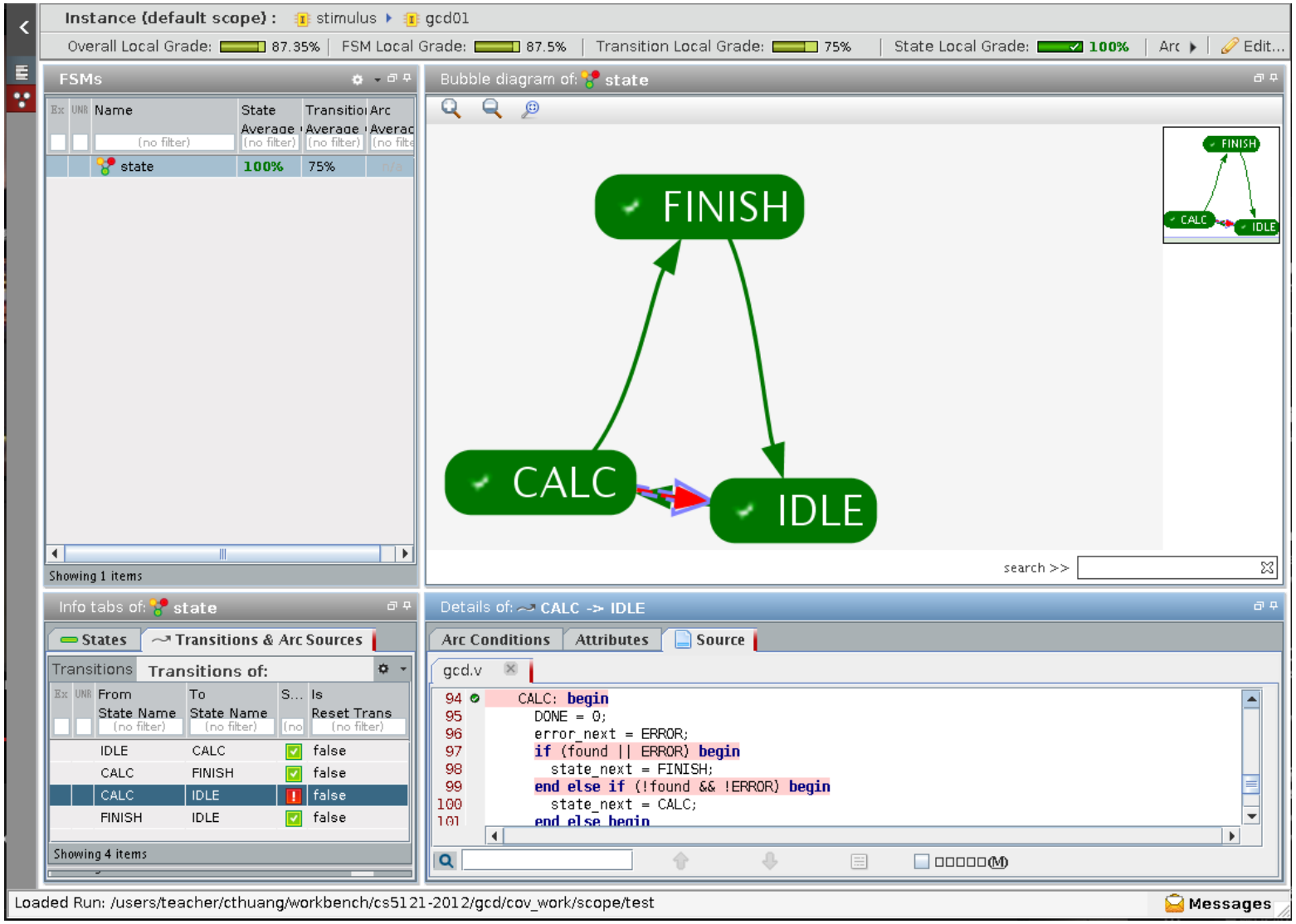
Example Case: Toggle Coverage



Example Case: FSM Coverage



Example Case: FSM Coverage



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