Supercharge Your Verification Using Rapid Expression Coverage as the Basis of a MC/DC-Compliant Coverage Methodology

Gaurav Kumar Verma and Doug Warmke
Questa R&D, Mentor Graphics Corporation

Introduction

Code coverage is a popular measure of design quality and verification completeness. It has low setup cost and analysis is straightforward, which makes it a high ROI component of most modern verification methodologies. Expression coverage is one of the most complex and least understood types of code coverage. The main question verification engineers need to answer regarding expression coverage is:

"There are 2^N possible input vectors for my N-input expression. I saw a subset of this large number during simulation. How well is my expression tested?"

A variety of metrics are available to help answer this question.

Popular metrics

SOP Metric

Checks that each set of inputs that satisfies the expression (results in a '1') must be exercised at least once, but not necessarily independently. It does not check the set of inputs that results in the expression being evaluated to '0'.

UDP Metric

A UDP table describes the full range of behavior for a given expression. If the conditions described by a row are observed during simulation, that row is said to be hit. All rows in the UDP table must be hit for UDP coverage to reach 100%. Row minimization is attempted by use of wildcard matches.

MC/DC Metric

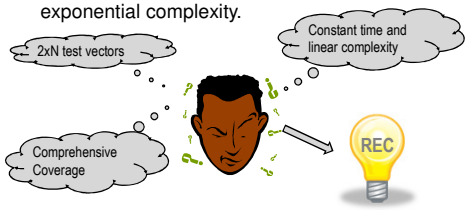
Checks that each condition in a decision has been shown to independently affect that decision's outcome. A condition is shown to independently affect a decision's outcome by varying just that condition while holding fixed all other possible conditions.

FEC Metric

An input is considered FEC-covered when other inputs are in their quiescent states, and the output has been seen in both '0' and '1' state. Checks that all inputs are FEC covered.

Shortcomings of existing metrics

- SOP does not provide comprehensive coverage.
- UDP is based on the truth table. In the worst case, it requires 2^N input vectors to cover an expression with N input terminals.
- FEC reduces verification effort by requiring 2xN input vectors to cover an expression. But since it is based on the truth table, it suffers from exponential complexity.



Modes of an expression

Inverting vs. non-inverting input terminal

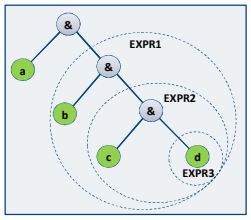
When setting the value of an input to '0' (or '1'), with all other inputs of the expression in their quiescent states, and the expression evaluates to '1' (or '0'), the input is operating in an inverting mode. Otherwise it is operating in a non-inverting mode.

Uni-modal vs. bi-modal expression

An expression whose inputs only ever operate in one mode, either inverting or non-inverting, is called a uni-modal expression. If there is at least one input capable of operating in both inverting and non-inverting modes, it is called a bi-modal expression.

Basic expressions

Every expression, however complex, can be broken down into N smaller expressions consisting of only one operator each, where N is the number of operators in the expression. We call these expressions 'basic expressions'.



Example:
f(a,b,c,d) = a & b & c & d
⇒ *a & (b & (c & d))*

Its basic expressions are:
a & EXPR1 (EXPR1 = b & (c & d))
b & EXPR2 (EXPR2 = c & d)
c & EXPR3 (EXPR3 = d)

Non-masking state

When working on any input in a basic expression, it is important that the input not be masked by the other input because of its value. For example, if we want to measure coverage of 'A' in 'A && B', the value of 'B' must be '1'. If 'B' is '0', the result of the expression gets fixed to '0' and the value of 'A' is no longer of any significance.

Operator	Expression	Non Masking State
NOT	NOT A	N/A
OR	A OR B	B = 0
NOR	A NOR B	B = 0
AND	A AND B	B = 1
NAND	A NAND B	B = 1
XOR	A XOR B	B = 0, B = 1
XNOR	A XNOR B	B = 0, B = 1
TERNARY	(COND)? A:B	COND = 1
	(COND)? B:A	COND = 0

REC collection

Break the expression into basic expressions. Once identified, only work on the basic expressions and ignore the actual expression.

At the time of coverage collection, make sure the other input of the basic expression is in a non-masking state.

An input will be REC-covered if it has taken both 0 and 1 value during simulation, in a state where the other input in its basic expression has a non-masking value.

An expression would be considered fully REC covered when all the inputs of the expression have been independently covered.

NOTE: REC collection can be woven into native code and thus has very minimal natural overhead.

Duplicate inputs

Relaxed: An input is considered covered if any of its occurrences is covered.

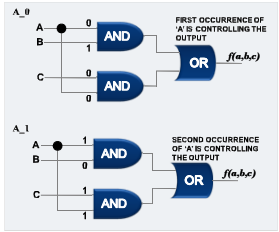
Strict: When coverage is collected for an input, all its occurrences in the expression must be simultaneously controlling the output of the expression.

Balanced: An input is considered covered if all of its occurrences are independently covered.

Relaxed Balanced: *An* input is considered covered when any of its occurrences is covered. Unlike relaxed approach, both the _0 and _1 hits must be from the same occurrence of the input.

Example: Relaxed collection

$$f(a,b,c) = (a \,\&\, b) \mid (a \,\&\, c)$$



Non-short-circuit operators

Since expressions are always evaluated left-to-right, short-circuiting enables us to assume that if an input is being evaluated during expression evaluation, the LHS of the input is already in a non-masking state. It is the responsibility of the implementation to ensure that LHS of the input is in a non-masking state when an operator doesn't short-circuit. An algorithm showing how to achieve this is presented in the proceedings paper.

Performance benchmarks

As expected, performance gain versus FEC increases as the size of the expression increases.

Expression	FEC Time	REC Time	Ratio
3-input AND	25.15	21.21	1.18x Faster
6-input XOR	41.53	33.95	1.22x Faster
7-input XOR	63.14	29.49	2.14x Faster
8-input XOR	113.8	27.14	4.19x Faster
9-input XOR	243.0	32.46	7.49x Faster
11-input XOR	1215	37.14	32.8x Faster