# *Veri*Sure

# *VHDL*Cover

**version 5**

# 機能詳細

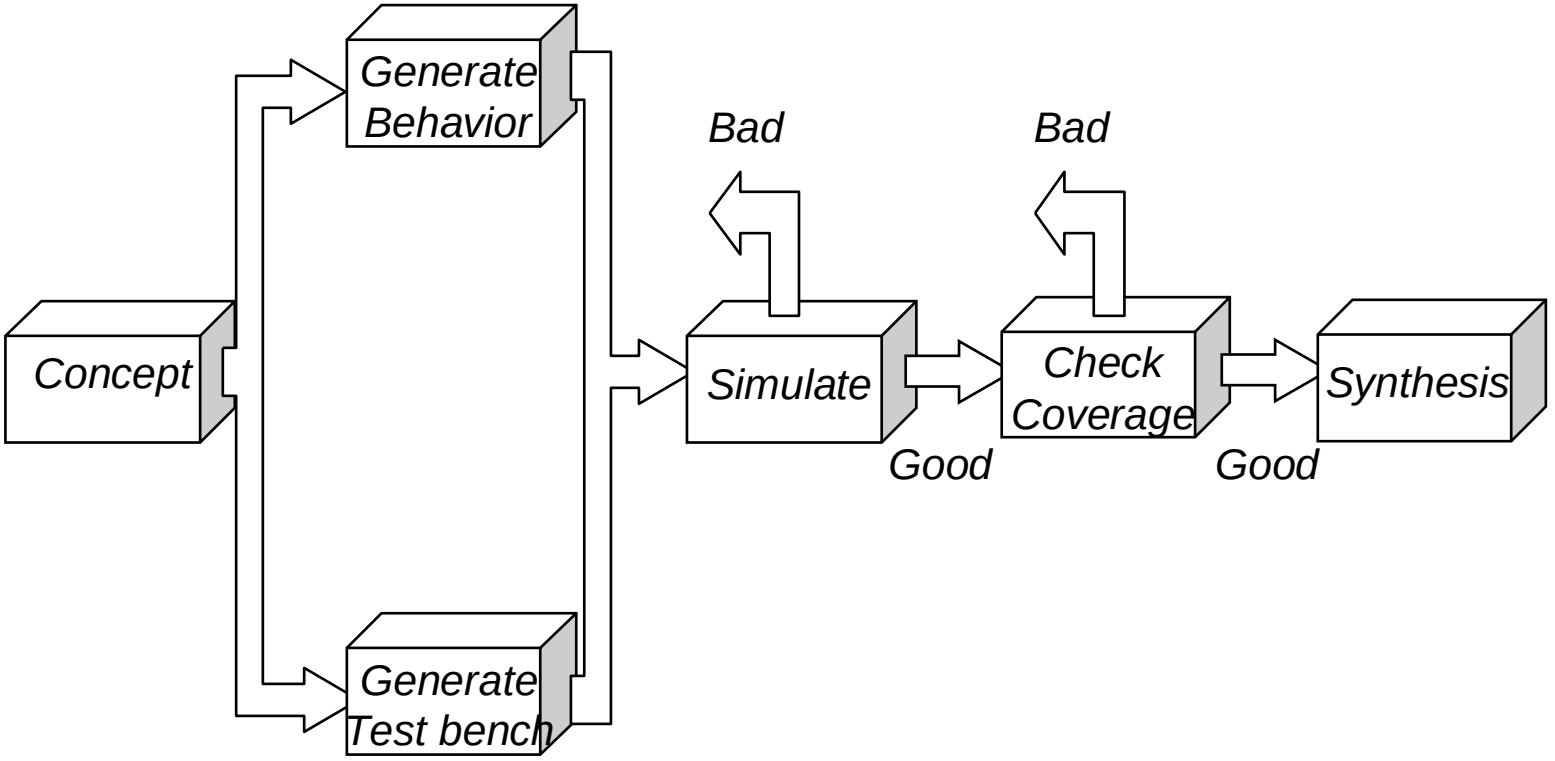
transEDA

# Section 1

# Code Coverageの概要

# VeriSureをデザインフローに組合せる



# Coverageの各項目

- ステートメントカバレッジ
  - Statement coverage
- ブランチカバレッジ
  - Branch coverage
- コンディションカバレッジ
  - Condition coverage
- パスカバレッジ
  - Path coverage
- トグルカバレッジ
  - Toggle coverage
- バリアブルカバレッジ
  - Variable coverage

transEDA

# Statement coverage

- **Continuous assignments と procedural statements**

```
assign ack = &data[3:0] && !rst;

always @ (posedge clk)
     q = d;
```

- **Statement文がカウントされてない部分（未テスト）を示す**
- **Checks modules, tasks and functions**

transEDA

# Branch coverage

- ネステッドbranchesのテスト確認

```
if (a)
    if (b)
        q = 1;
```

- **Check default assignments are tested**

```
q = 0;
if (a)
    q = 1;
```

# Condition Coverage

- **subcondition combinationsの確認**

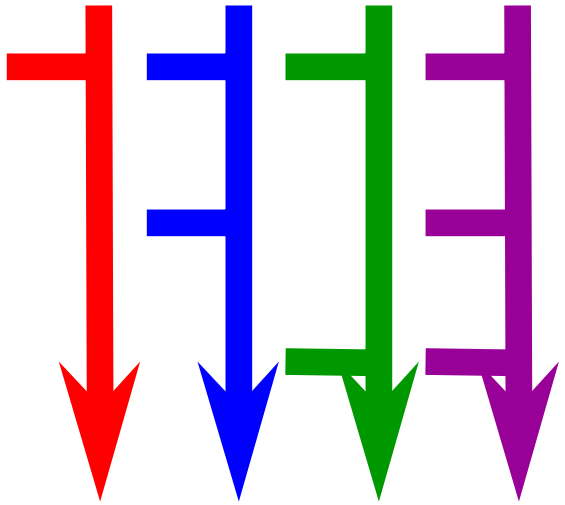
```
if (a && b || c)
    ...


assign d = a && b || c;
```

# Path coverage

- **Checks sequential code**

| Statement Count | Code |
|---|---|
| 8 | state = `CHECKS; |
| | if (done) |
| 1 | state = `ENDS; |
| | if (lsb) |
| 4 | state = `ADDS; |



# Toggle coverage

- variables上のトランジションを確認


- Tests input/outputs, wires and regs

# Variable coverage

- ユーザーセットのvariablesグループの値をモニター可能
- hierarchical namesをサポート
- 使用例:
  - Check state transitions in a controller
  - Check inputs to RAM, ROM or an ALU
  - Check for bus contention

# Section 2

# Using VeriSure

# VeriSure structure

- Graphical User Interface (GUI)
- Command Line Interface (CLI)

- カバレッジを確認する3ステップ:
  - Instrumentation
  - Simulation
  - Results

- 通常の使用方法:
  - Instrumentation は、コマンドライン
  - Simulation は、コマンドライン
  - Results を、GUIで確認

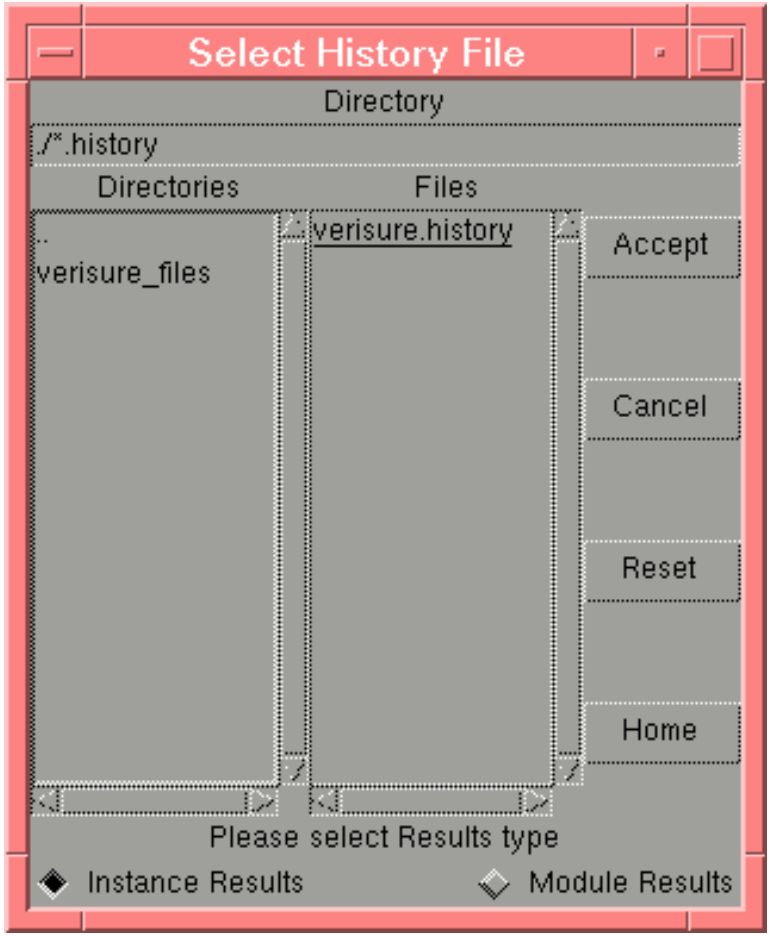
 

# coverage 結果の確認

- results windowの内容
  - Level 1: 全体的なサマリー
  - Level 2: モジュール毎の結果
  - Level 3: ソースコードに対する詳細と注釈
  - Level 4: 問題点の詳細記述

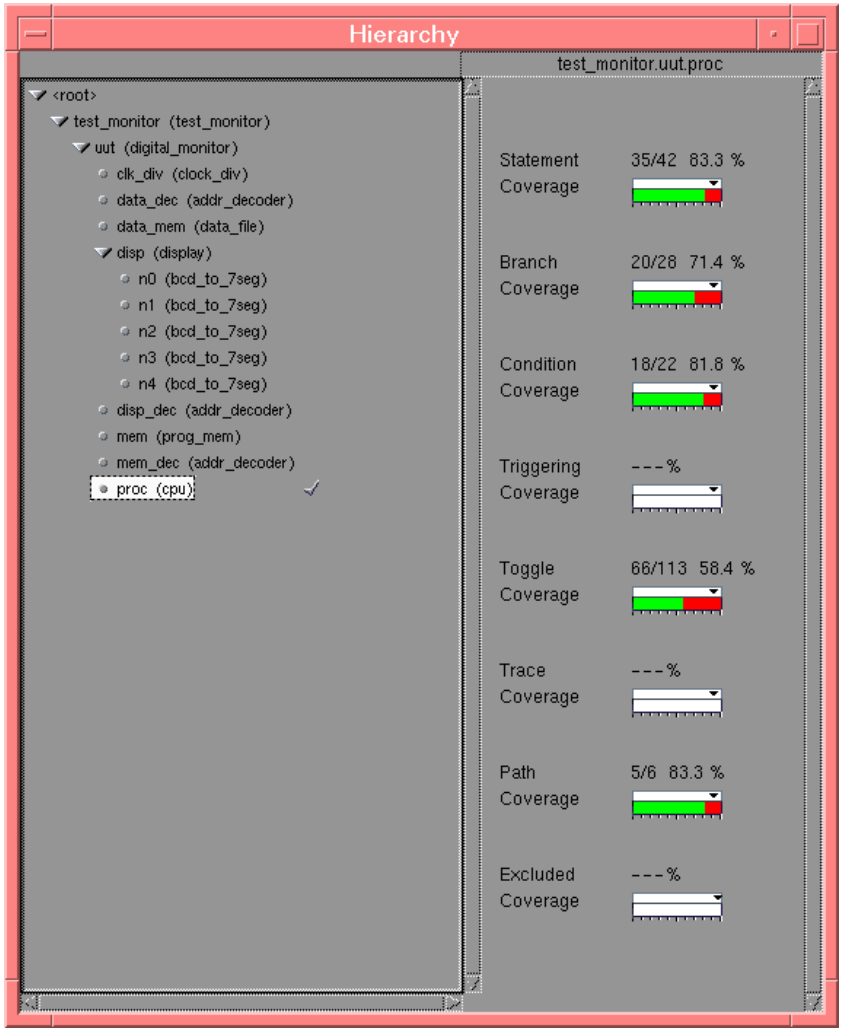
# history file(s)の選択



# Results Browser

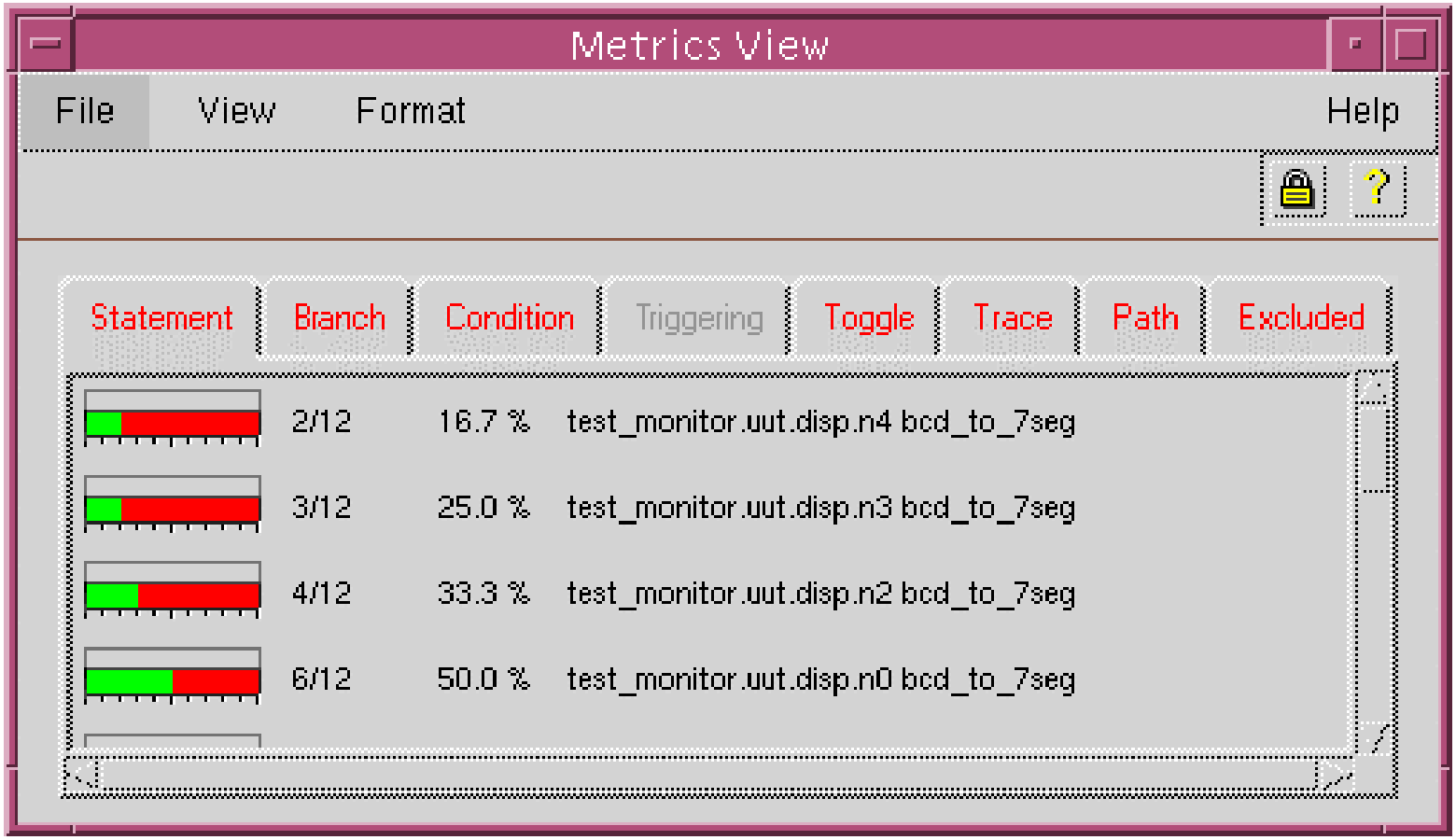
階層構造
の表示




各カバレッジ値
の表示

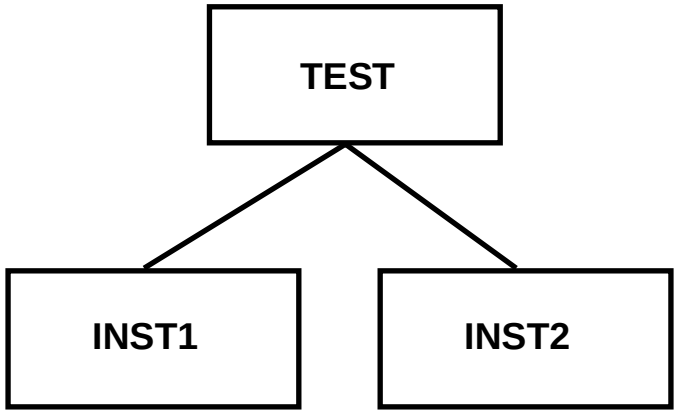
# 各モジュールにおけるカバレッジ値




ワーストケースから並べるように
ソーティング可能

# モジュール毎の結果と見方




| TEST.INST1 | TEST.INST2 | Module Data |
|---|---|---|
| 5 | 2 | 7 |
| 0 | 4 | 4 |
| 8 | 1 | 9 |
| 0 | 0 | 0 |

```
module logic_op(result, op, a, b);
output [31:0]result;
input  [31:0]a;
input  [31:0]b;
input   [1:0]op;
reg    [31:0]result;
always @ (op or a or b)
begin
   case (op)
      2'b00:
         result = a & b;
      2'b01:
         result = a | b;
      2'b10:
         result = ~a;
      2'b11:
         result = ~b;
   endcase
end
endmodule
```

# Section 3

# Code coverage
# 結果の内容確認

# 基本的なカバレッジの見方

- **Statement coverage**
  - 全てのステートメント（命令文）が実行/活性化されたか
- **Branch coverage**
  - 隠れた分岐を含んで全ての分岐を実行されたか
- 以上の項目は、カバレッジの測定基準として**100%**でないと意味がない

# Condition coverage

- ‘if’ ブランチによる各コンディション状態をレポート
- アサイメント上にある、各ブーリアン表現（and,or等）をレポートする
- 問題点を明記:
  - 変数のもつ値をコントロール
  - 論理的なオペレーションのコントロール
  - コンディションの分岐の冗長性

# Variable Coverage

- 一つ以上の変数の値をトレースする
- モニターしている変数値の組合せを表示
  （実際にはマトリックス化して）
- 階層的な変数名をサポート

# Variable coverageの応用

- ステートマシン遷移をモニター
- ステートマシンの相互作用をモニター
- RAM/ROM アドレスまたはデータ値をモニター
- CPU バスのモニター

# Toggle coverage

- モジュール中の全ての変数をモニター
  考慮されていないI/Oポートやメモリを助変数化する
  （立ち上がりエッジ数、立ち下がりエッジ数として）

- 立ち上がり/立ち下がりエッジの数をカウント
  エッジの定義はパラメータ化される　例、01,0x,x1

- Toggleは少なくとも、
  1つの立ち上がり/立ち下がりエッジを持つ
  トランジションカバレッジを得るための指標となる

transEDA

# Toggle coverage の応用

- データパス設計のチェック
  例） ALUの入力/出力,RAM/ROMデータや
  アドレス、バスデータ

- **Fault simulationに対する、対策・検討**

# Coverage 結果をどの様に使用するか？

- カバレッジレポート中の未テストのコードを検出する
- 潜在的な問題（コーナーバグ等）を解消する
- 必要であればテストやソースコードを修正