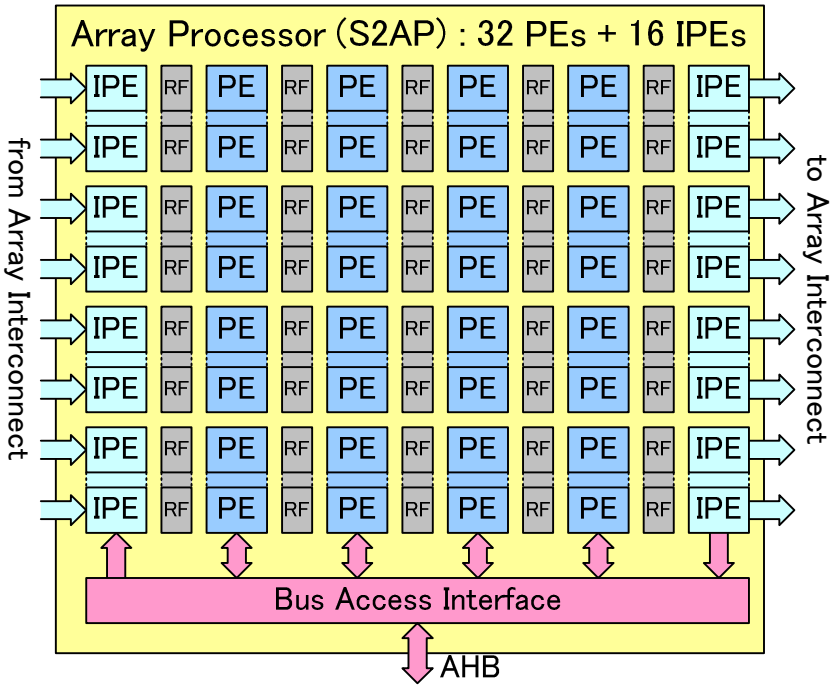
SystemVerilogで構築した
アレイプロセッサ検証環境とその効果

ソナック株式会社
LSI事業部
清水圭典

発表内容

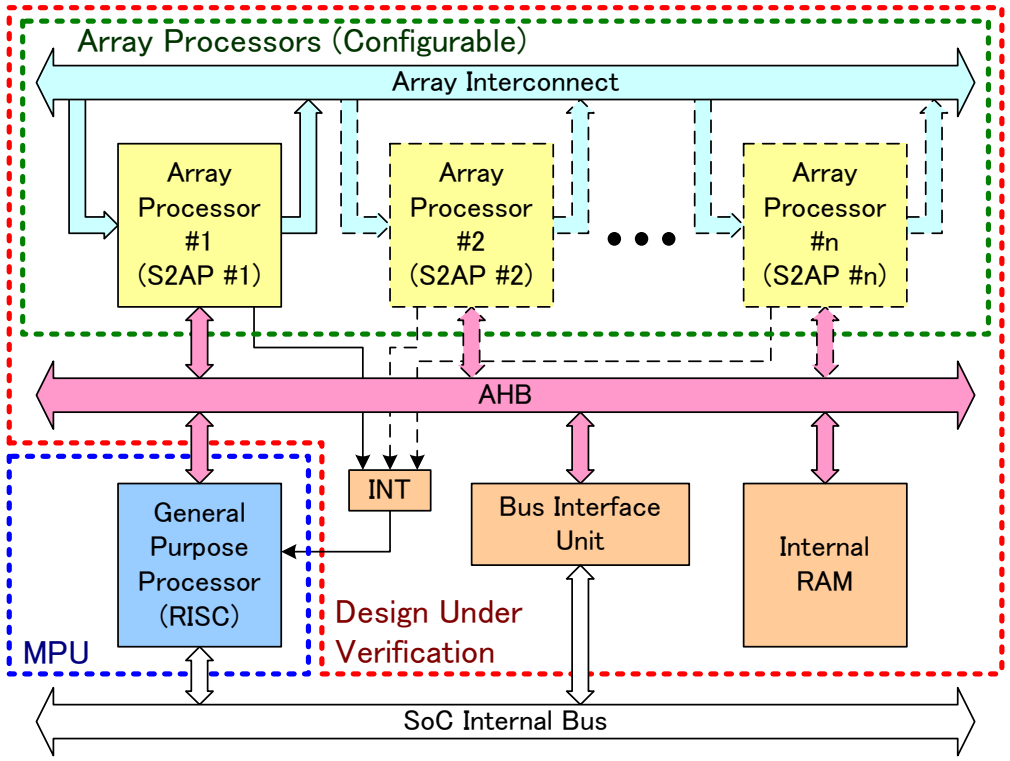
- アレイプロセッサの概要
- 従来の検証手法の問題点
- 検証アーキテクチャの特徴
- 成果と今後の課題

アレイプロセッサの概要 (1)



- 16ビット演算エレメント(PE)を32個搭載
  - 2個のPEを一組として32ビット演算も可能
- 各PEはRISCと同レベルの命令セットを持つ
  - ロード/ストアも可能
- 接続エレメント(IPE)を用意し複数のアレイプロセッサを直接接続
- ストリームデータ処理が可能

アレイプロセッサの概要 (2)



- アレイプロセッサ数は要求性能に応じて変更可能
- MPUがアレイプロセッサを制御
- アレイプロセッサ、内部メモリおよびMPUはAHBで接続
- アレイプロセッサ間のデータはAHBを介さずアレイ・インターコネクトを使用

発表内容

- アレイプロセッサの概要
- 従来の検証手法の問題点
- 検証アーキテクチャの特徴
- 成果と今後の課題

ダイレクト検証の問題点

- 膨大な(1万以上の)テストケースが必要となる
  - RTL開発効率に追いついておらず、LSI開発のボトルネックになっている
- いくらテストケースを用意しても、コーナーケースのバグがとれない
  - テストケースがなければ、その機能は検証されない
  - 想定外のケースが必ず残る
    - 複雑なコーナーケースをすべて想定することは不可能
    - 想定できたとしても、テストケースとして実現するのは困難な場合が多い
- 冗長なテストケースが多数存在する

制約付きランダム検証の導入

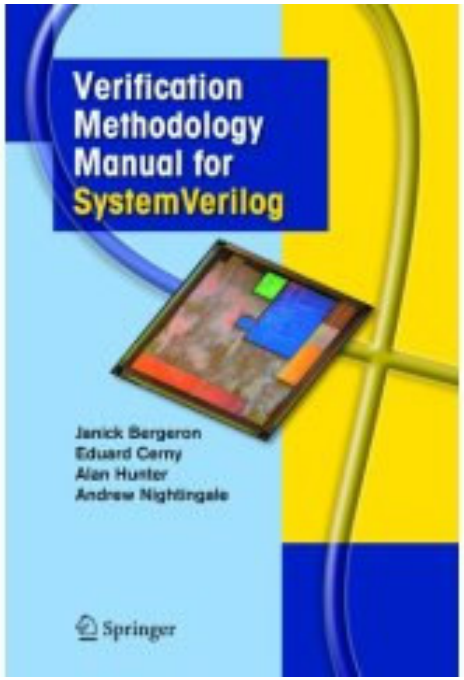
- 複数のシナリオからテストベンチがスティミュラスを自動生成
  - 現実には起こりうるが、想定外のコーナーケースにヒットし易い
- テストケースが単純
  - 主にランダム・パラメタの設定のみ
  - テストケースではなく、テストベンチがデータ・チェックを行う
- ランダム・シードを変えることで検証カバレッジを上げる
  - ダイレクト検証
    - 多数(1万以上)のテストケース × 1回の実行
  - 制約付きランダム検証
    - 最小(数十)のテストケース × 数百シードの実行

発表内容

- アレイプロセッサの概要
- 従来の検証手法の問題点
- 検証アーキテクチャの特徴
- 成果と今後の課題

ベースとした検証メソドロジ

- Verification Methodology Manual for SystemVerilog (VMM)
- VMMを採用した理由
  - アレイプロセッサのコーナーケース・バグを体系的に発見するためのメソドロジが必要であった
  - 将来のプロダクトのためにテストベンチを最大限再利用したかった
  - SystemVerilogなのでVerilogデザインとの親和性に優れている
  - 仕様がオープンなので、将来、業界標準になりうる



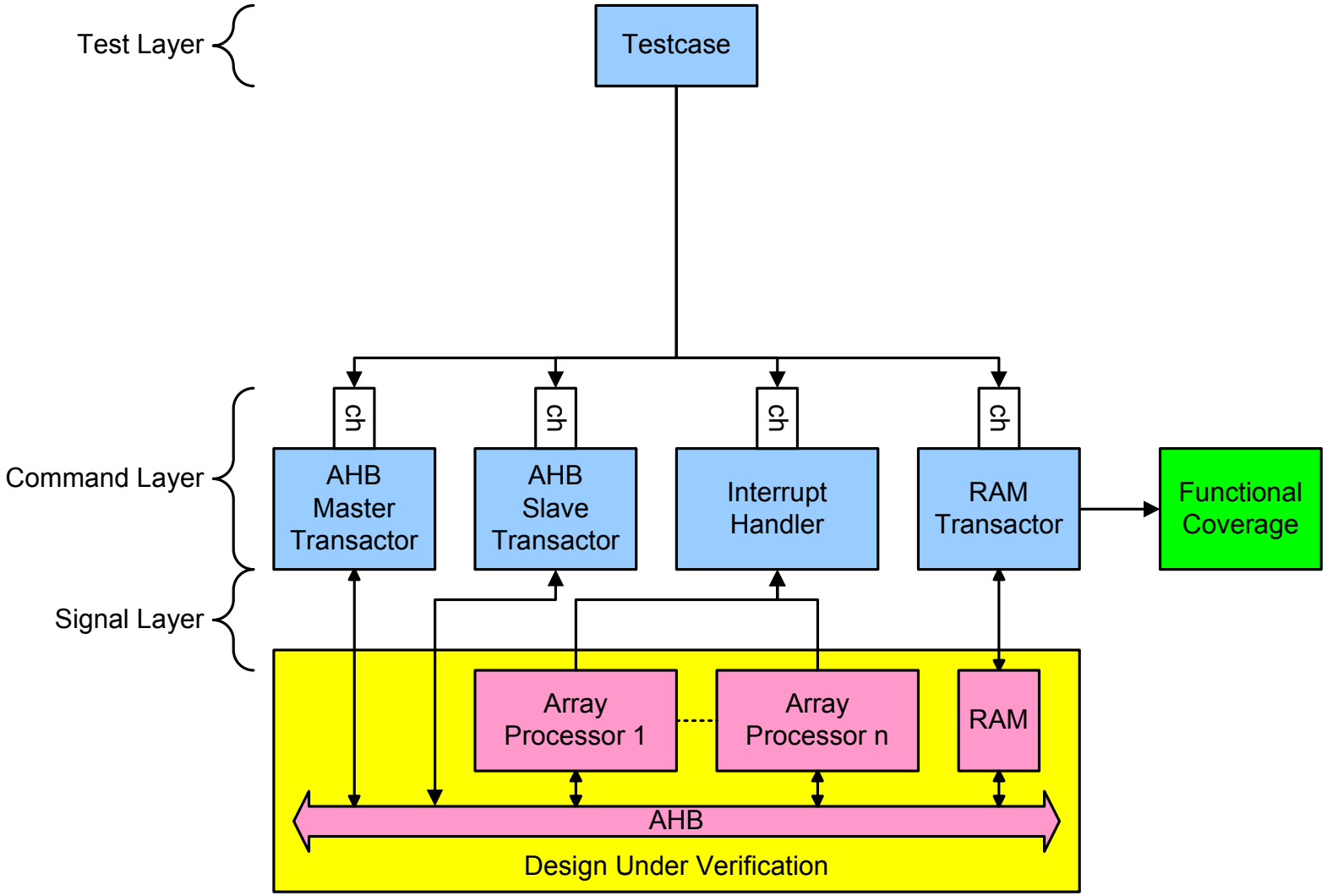
検証アーキテクチャの特徴

- 階層的テストベンチ
  - インクリメンタルなテストベンチ
- 制約付きランダム検証
- 機能カバレッジ・ドリブン検証
- アサーション
- 検証環境の再利用
- VMMに完全準拠

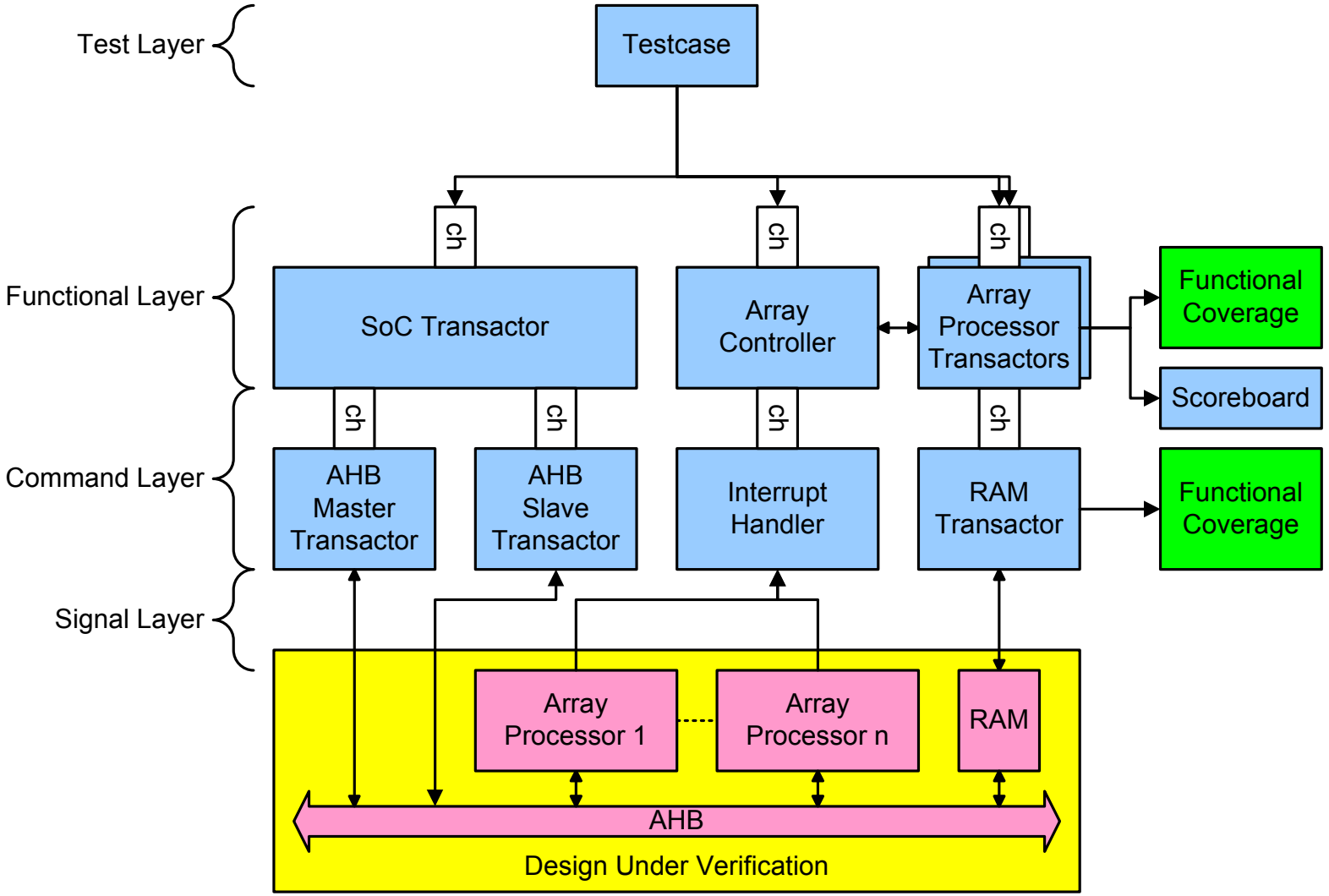
階層的テストベンチ

- インクリメンタルなテストベンチ
  - テスト階層　　… シナリオの選択、ランダム・パラメタの設定
  - シナリオ階層　　… トランザクションの生成
  - 機能階層　　… トランザクションからコマンドへの変換
  - コマンド階層　　… コマンドから信号への変換
  - 信号階層　　… 検証対象とテストベンチの接続
- 検証環境の完成を待たずに検証作業が開始可能
  - 制約を制限することで、機能の一部実装段階からランダム検証が開始可能
- ダイレクト検証もサポート

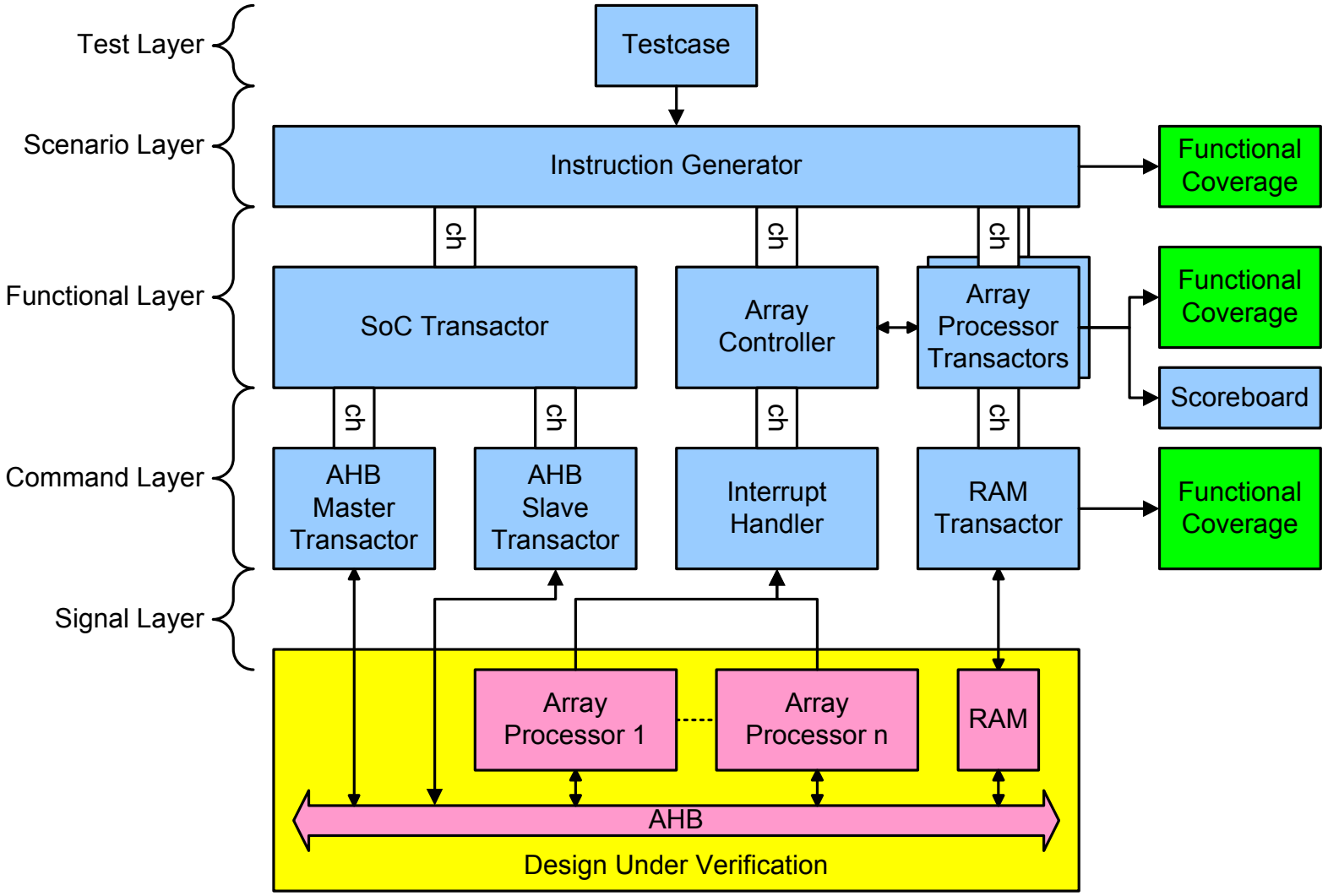
インクリメンタルなテストベンチ（1）



インクリメンタルなテストベンチ (2)



インクリメンタルなテストベンチ（3）

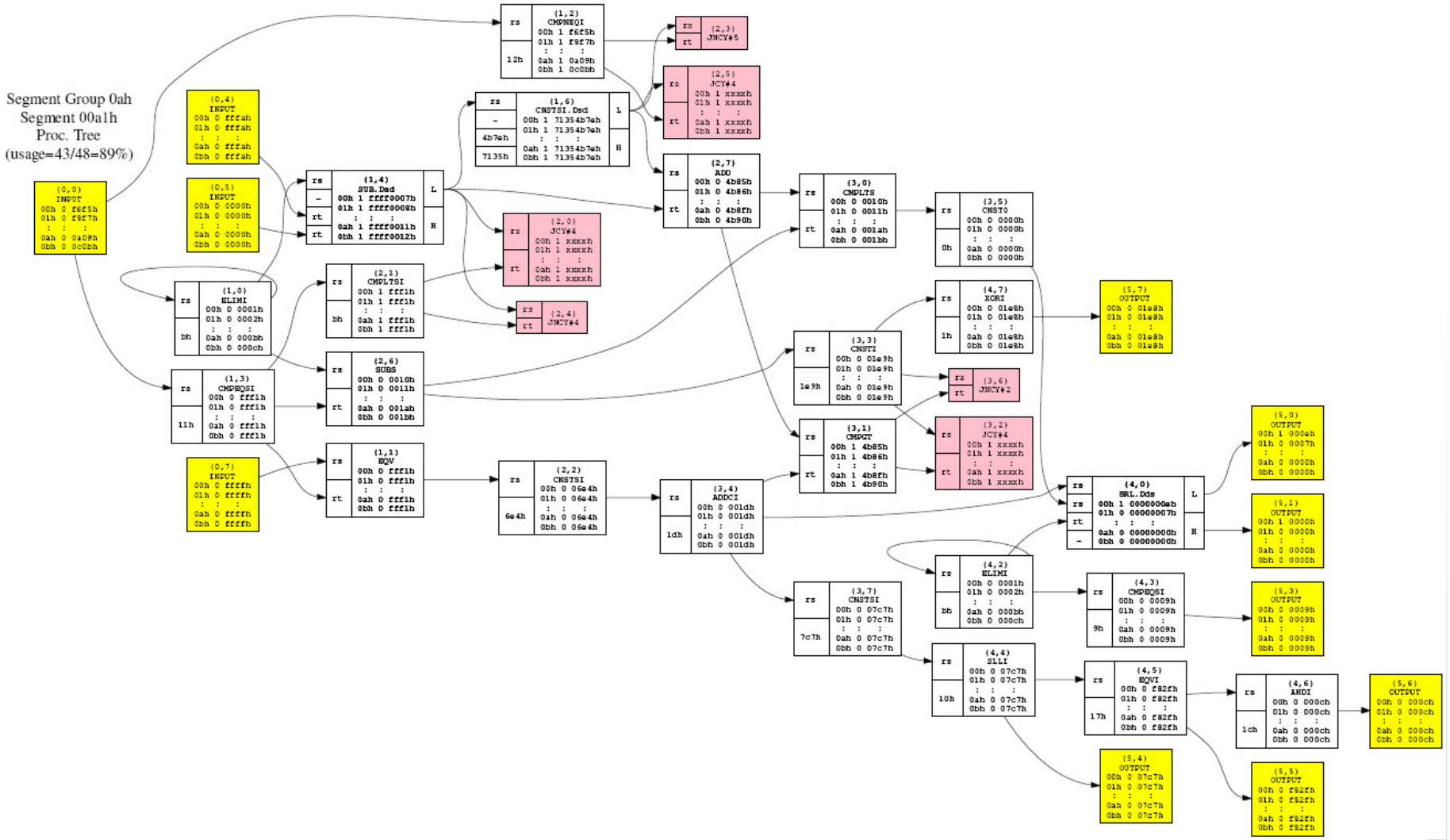


制約付きランダム検証

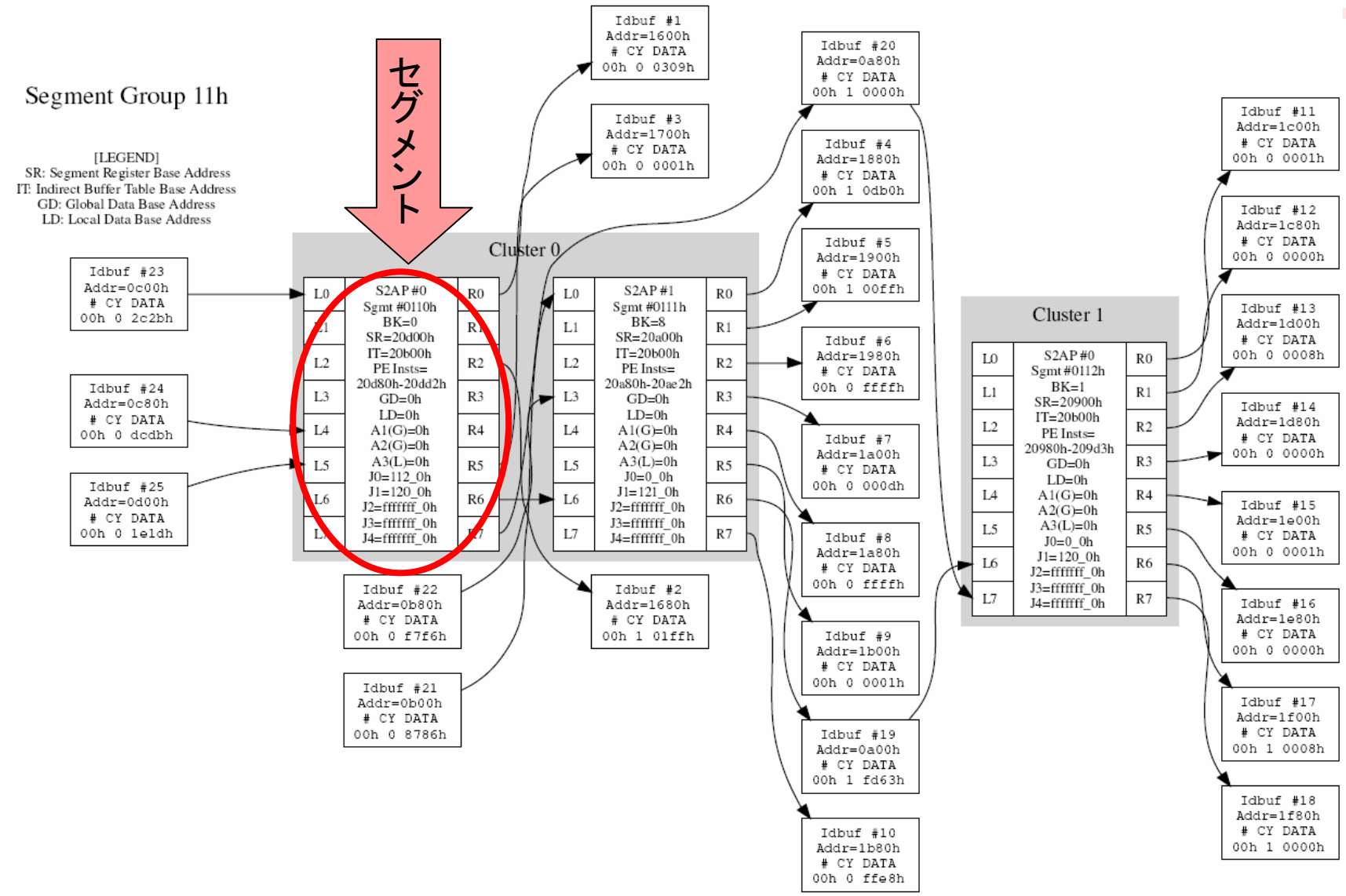
- SystemVerilogの**constraint**構文によって有効な入力パターンを自動生成する

```
constraint src0_con {
  solve op, D before src0;
  if ( D == 1 ) src0 % 2 == 0;
  if ( op == NU || op == INPUT ) src0 == 0;
  if ( op == OUTPUT ) src0 == y;
  if ( op == ELIM )
    src0 == y + 8; // self-loop
  else if ( closed_loop_en == 0 )
    src0 < 8;
  else if ( ! ( op == CNST || op == CNSTS ) )
    src0 != y + 8; // no self-loop
}
constraint D_con { // double instruction
  if ( y % 2 ) D == 0;
  D dist { 0 := 100 - D_pct, 1 := D_pct };
}
```

自動生成されたセグメント（アレイプロセッサ命令ツリー）の例



自動生成されたセグメントグループの例



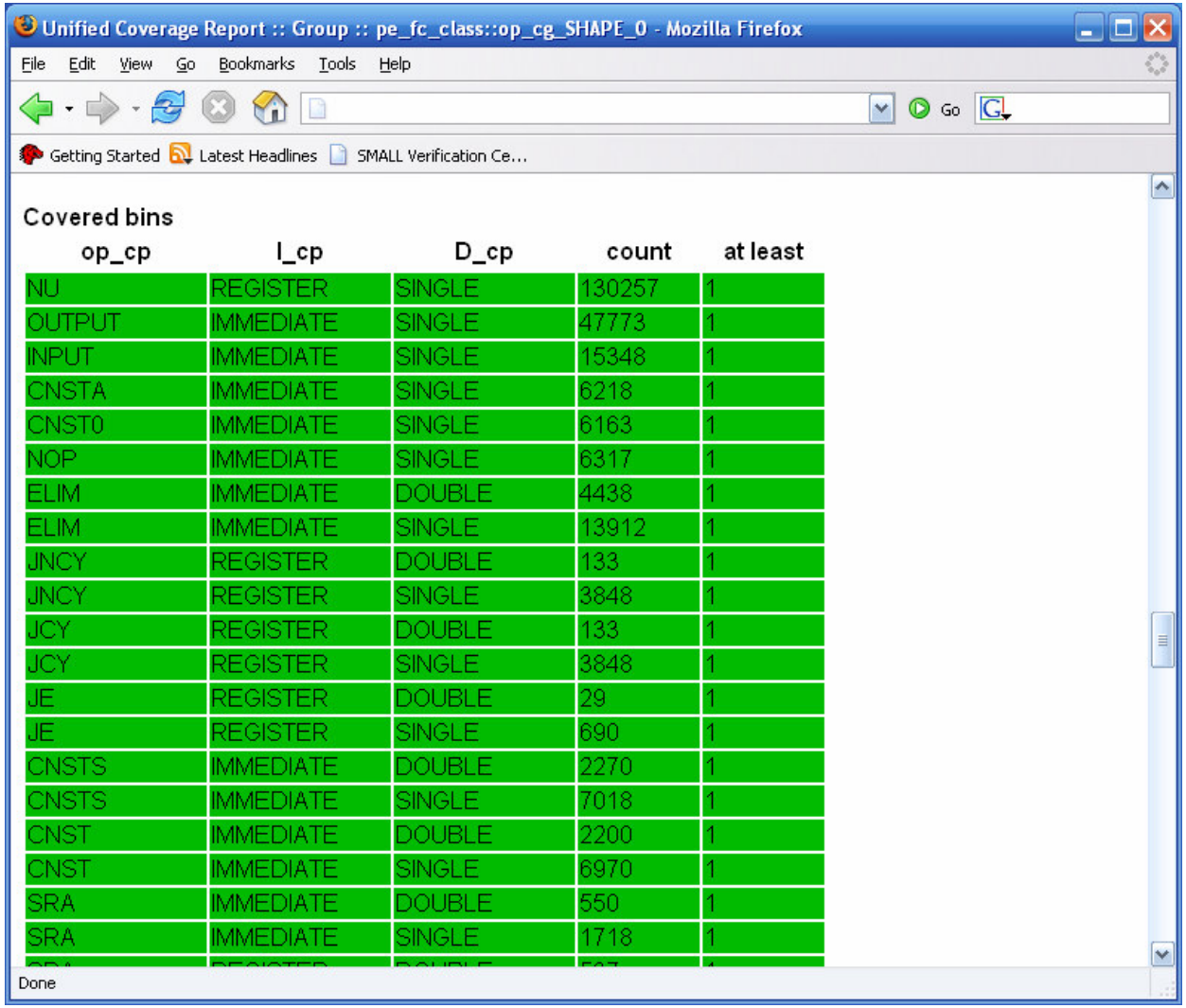
機能カバレッジ・ドリブン検証

- 機能カバレッジとランダム検証は両輪の関係
- カバレッジをフィードバックすることによって、シミュレーション効率を上げる
  - 同じ機能を何度も検証しない
  - カバーされていない機能を重点的にカバーするように制約を変更する
- ユーザが後からカバレッジ・ポイントを容易に追加できるようにvmm_xactor_callbacksを利用

SystemVerilogによる機能カバレッジ記述例

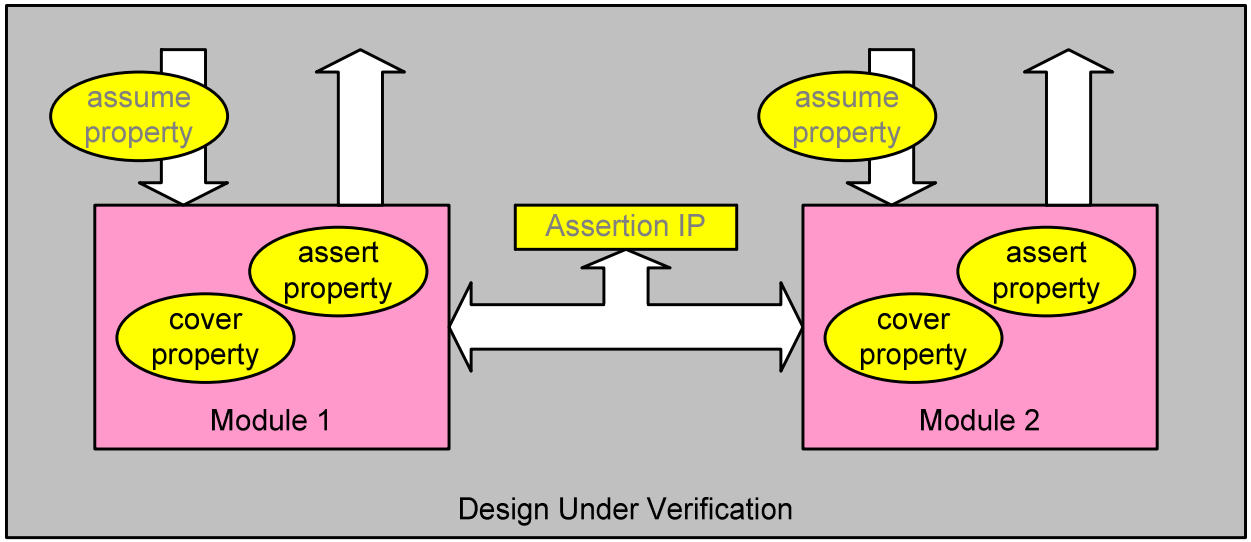
```
covergroup op_cg; // op-code covergroup
  op_cp: coverpoint op {
    bins ADD = { pe_inst_class::ADD };
    bins SUB = { pe_inst_class::SUB };
    …
  }
  …
  op_xc: cross op_cp, I_cp, D_cp { // cross coverage among
    …                              // op-code, immediate, double
    illegal_bins illegal_imm =
      binsof( op_cp ) intersect { pe_inst_class::NOT } &&
      binsof(  I_cp ) intersect { 1 };
  }
}
covergroup ldst_cg; // load/store operation covergroup
  sgl_ld_cnt_cp: coverpoint sgl_ld_cnt { // single load count
    bins NO_LOAD             = { 0 };
    bins SINGLE_LOAD_1_TO_7 = { [1:7] };
    bins SINGLE_LOAD_8       = { 8 };
  }
}
```

機能カバレッジ出力例


Unified Coverage Report :: Group :: pe_fc_class::op_cg_SHAPE_0 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
Getting Started
Latest Headlines
SMALL Verification Ce...
Covered bins
op_cp | l_cp | D_cp | count | at least
NU | REGISTER | SINGLE | 130257 | 1
OUTPUT | IMMEDIATE | SINGLE | 47773 | 1
INPUT | IMMEDIATE | SINGLE | 15348 | 1
CNSTA | IMMEDIATE | SINGLE | 6218 | 1
CNST0 | IMMEDIATE | SINGLE | 6163 | 1
NOP | IMMEDIATE | SINGLE | 6317 | 1
ELIM | IMMEDIATE | DOUBLE | 4438 | 1
ELIM | IMMEDIATE | SINGLE | 13912 | 1
JNCY | REGISTER | DOUBLE | 133 | 1
JNCY | REGISTER | SINGLE | 3848 | 1
JCY | REGISTER | DOUBLE | 133 | 1
JCY | REGISTER | SINGLE | 3848 | 1
JE | REGISTER | DOUBLE | 29 | 1
JE | REGISTER | SINGLE | 690 | 1
CNSTS | IMMEDIATE | DOUBLE | 2270 | 1
CNSTS | IMMEDIATE | SINGLE | 7018 | 1
CNST | IMMEDIATE | DOUBLE | 2200 | 1
CNST | IMMEDIATE | SINGLE | 6970 | 1
SRA | IMMEDIATE | DOUBLE | 550 | 1
SRA | IMMEDIATE | SINGLE | 1718 | 1
Done

アサーション

- バグの発生箇所をピンポイントで見つけるために各種アサーションを使用
  - **assert property**
  - **assume property**
  - アサーションIP
- 設計に依存したコーナーケースがどれだけカバーされたかを把握
  - **cover property**



カバレッジプロパティの記述例

```
// property declarations

property fifo_full_rose_p;
  @ ( posedge clk ) ( rst_n == 1 && clr == 0 ) && $rose( full );
endproperty

property fifo_empty_rose_p;
  @ ( posedge clk ) ( rst_n == 1 && clr == 0 ) && $rose( empty );
endproperty

// cover statements

fifo_full_rose_cp: cover property ( fifo_full_rose_p );

fifo_empty_rose_cp: cover property ( fifo_empty_rose_p );
```

検証環境の再利用

- テストベンチの再利用
  - テストファイルからテストベンチを再構成

```
env = new( 2 ); // # of array processors
```

- トランザクタの再利用
  - コールバックによってトランザクタの動作を変更

VMM

```
scen_cbs = new; // scenario callbacks
env.scen_gen.append_callbacks( scen_cbs );
```

- 検証モジュールの入れ替えが容易
  - チャネル接続により検証モジュール間が疎結合
  - プロジェクト間での再利用

VMM

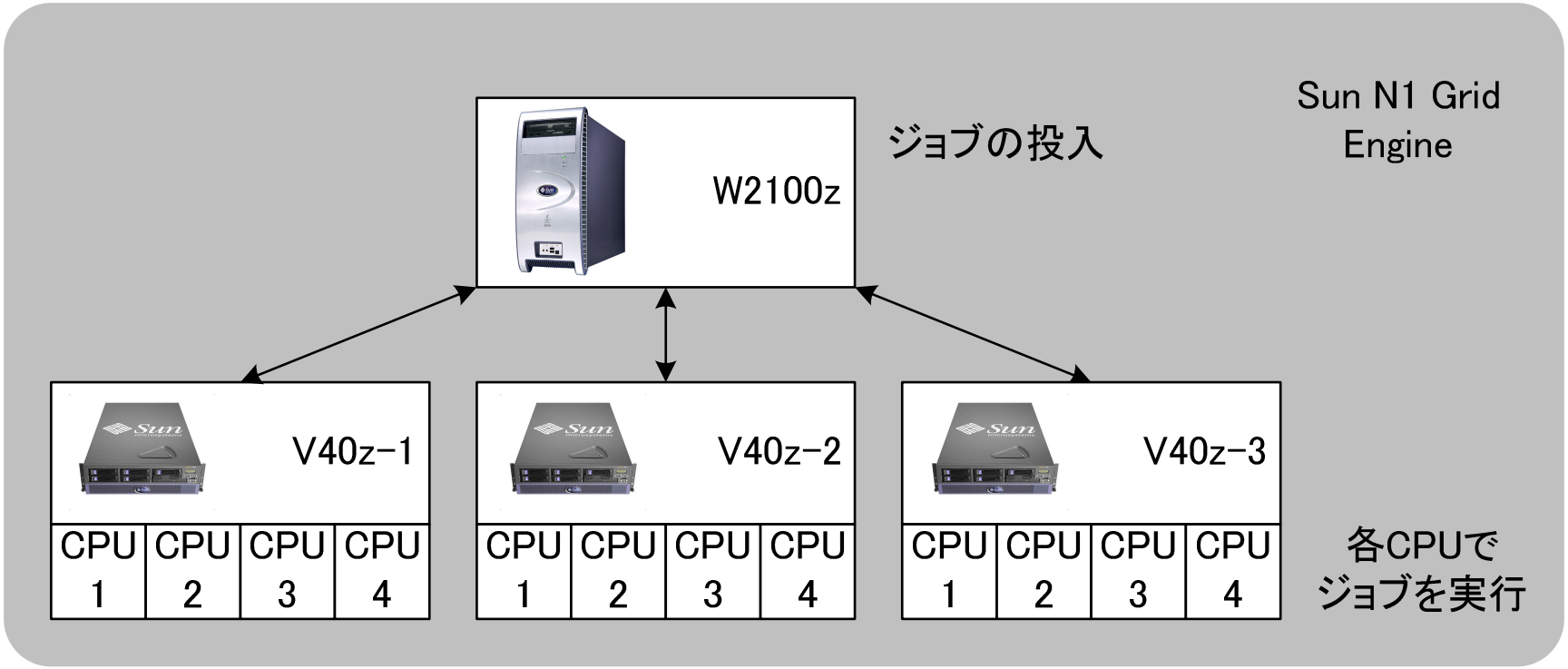
発表内容

- アレイプロセッサの概要
- 従来の検証手法の問題点
- 検証アーキテクチャの特徴
- 成果と今後の課題

シミュレーションリソース

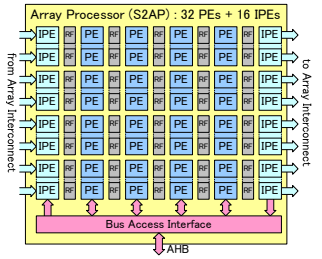
- サン・マイクロシステムズ(株)様より
  以下のリソースを貸与していただきました
  - Sun Fire V40z 3台
    - 2.6GHz CPU×4 （トータル12 CPU）
    - 各16GB Memory
    - x64 Solaris 10
  - Sun Java Workstation W2100z 1台
    - トータル2 CPU
    - Gridジョブ投入用
  - Sun N1 Grid Engine
- 使用ツール
  - VCSシミュレータ 16本

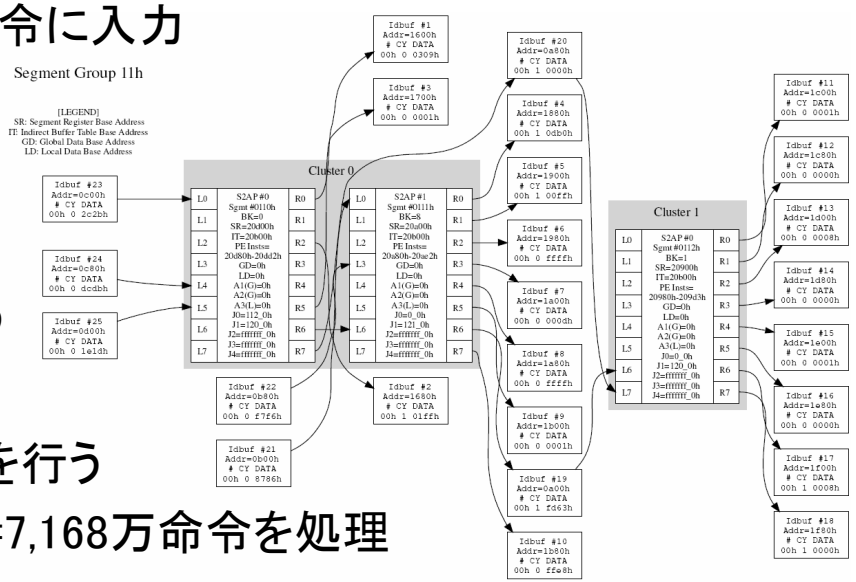
シミュレーション環境


Sun N1 Grid Engine
ジョブの投入
W2100z
V40z-1
V40z-2
V40z-3
CPU 1
CPU 2
CPU 3
CPU 4
CPU 1
CPU 2
CPU 3
CPU 4
CPU 1
CPU 2
CPU 3
CPU 4
各CPUで
ジョブを実行

リグレッション用テストケースの概要

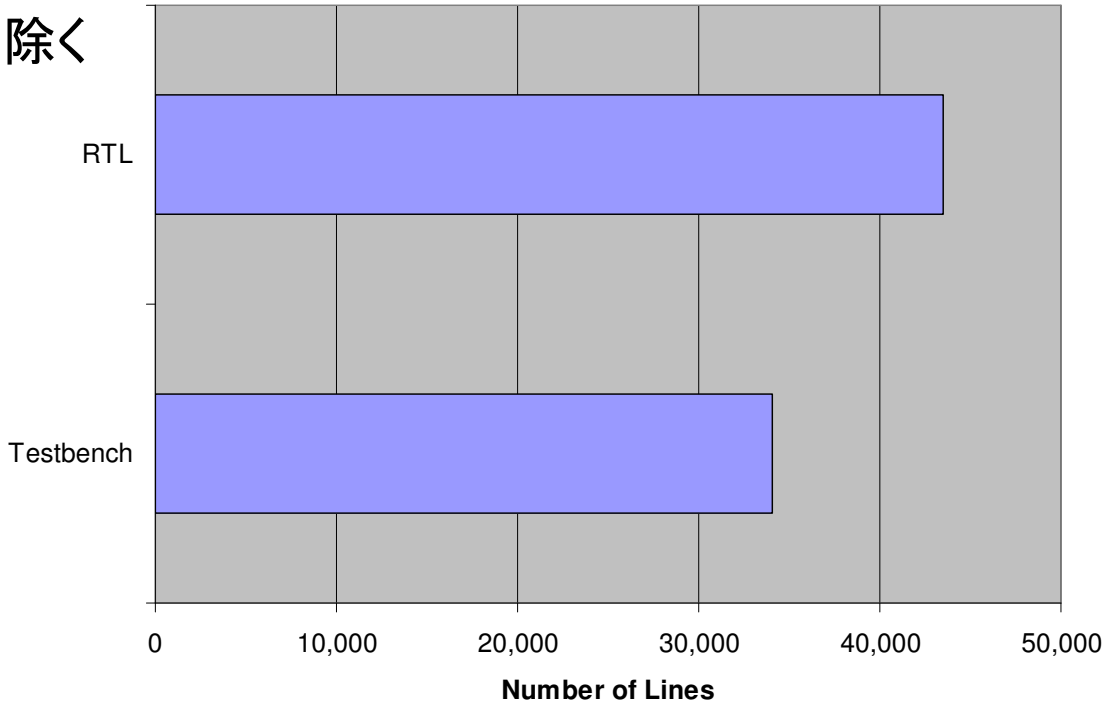
- すべての命令を制約付きランダムに生成
- （アレイ）セグメント
  - 最大32個の命令を含む
  - 最大32個のストリームデータを各命令に入力
- セグメントグループ
  - 1～7個のセグメントで構成
- テスト
  - 100セグメントグループの処理を行う
- リグレッション
  - 異なる100シードのシミュレーションを行う
  - 最大で合計32×32×7×100×100=7,168万命令を処理



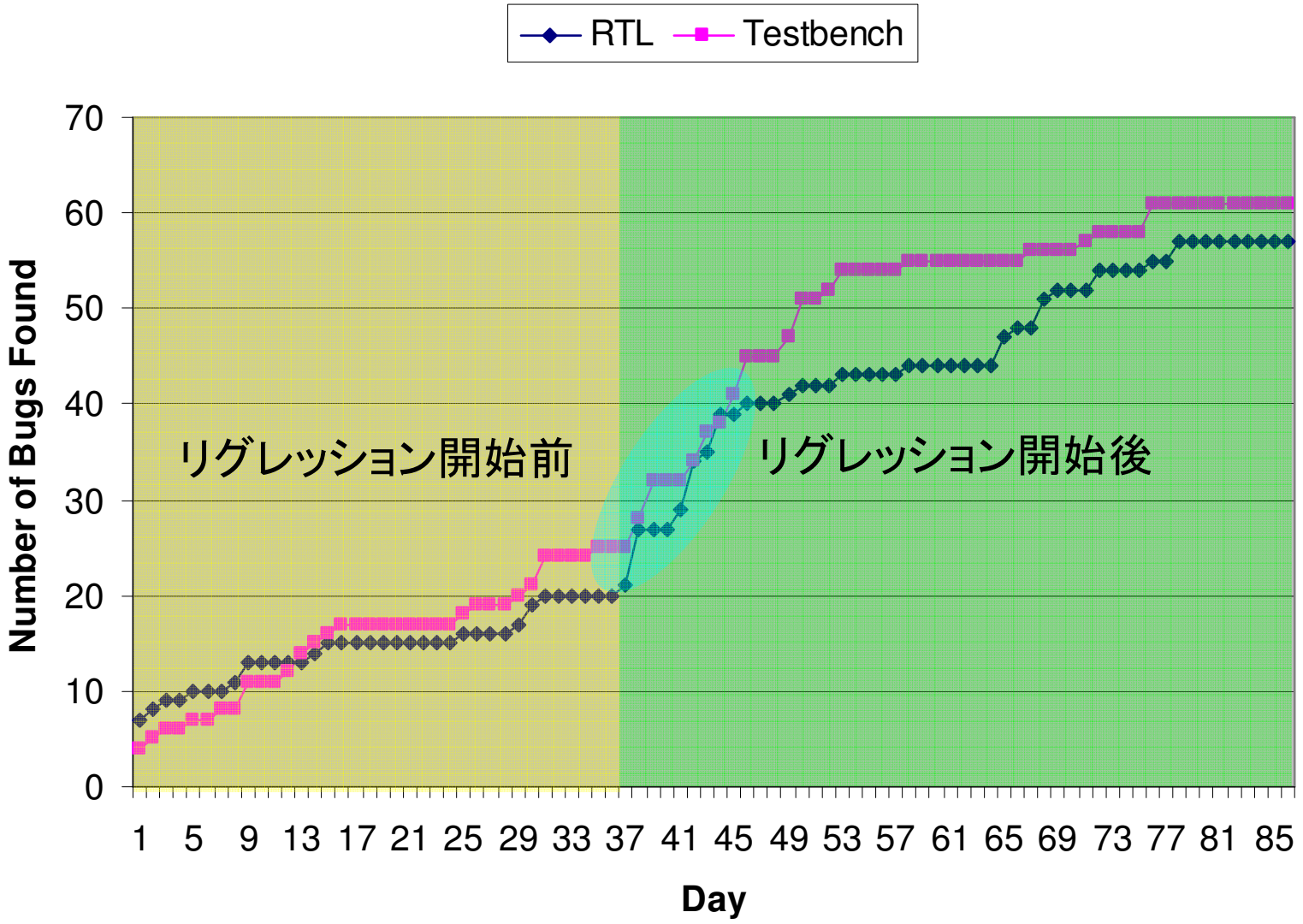


コード量比較（1月24日現在）

- Verilog RTL合計ライン数＝43,513
  - コメントを含む
- SystemVerilog検証環境合計ライン数＝34,074
  - コメントを含む
  - テストケース記述は除く

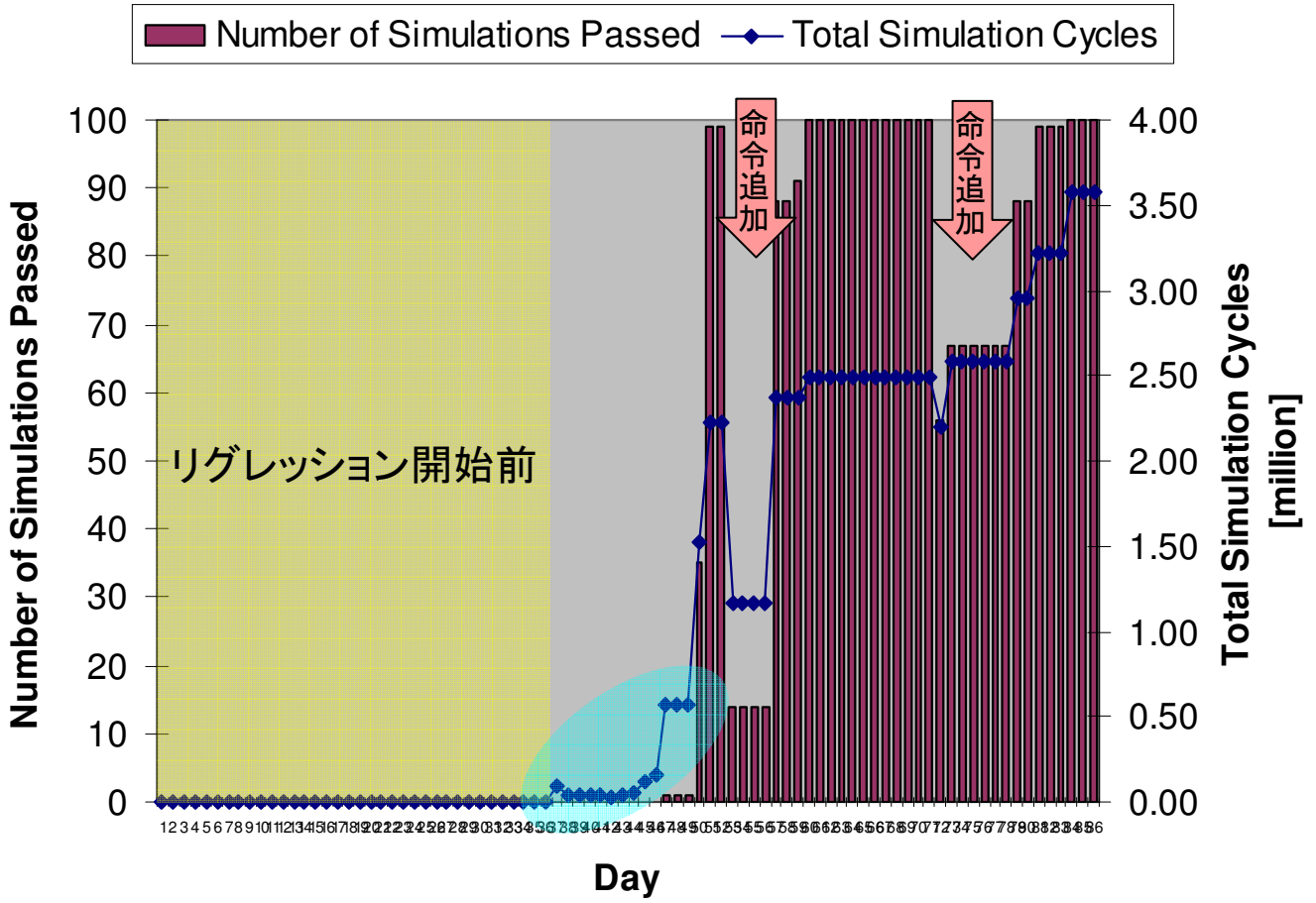


累積バグ数の推移(1月24日現在)



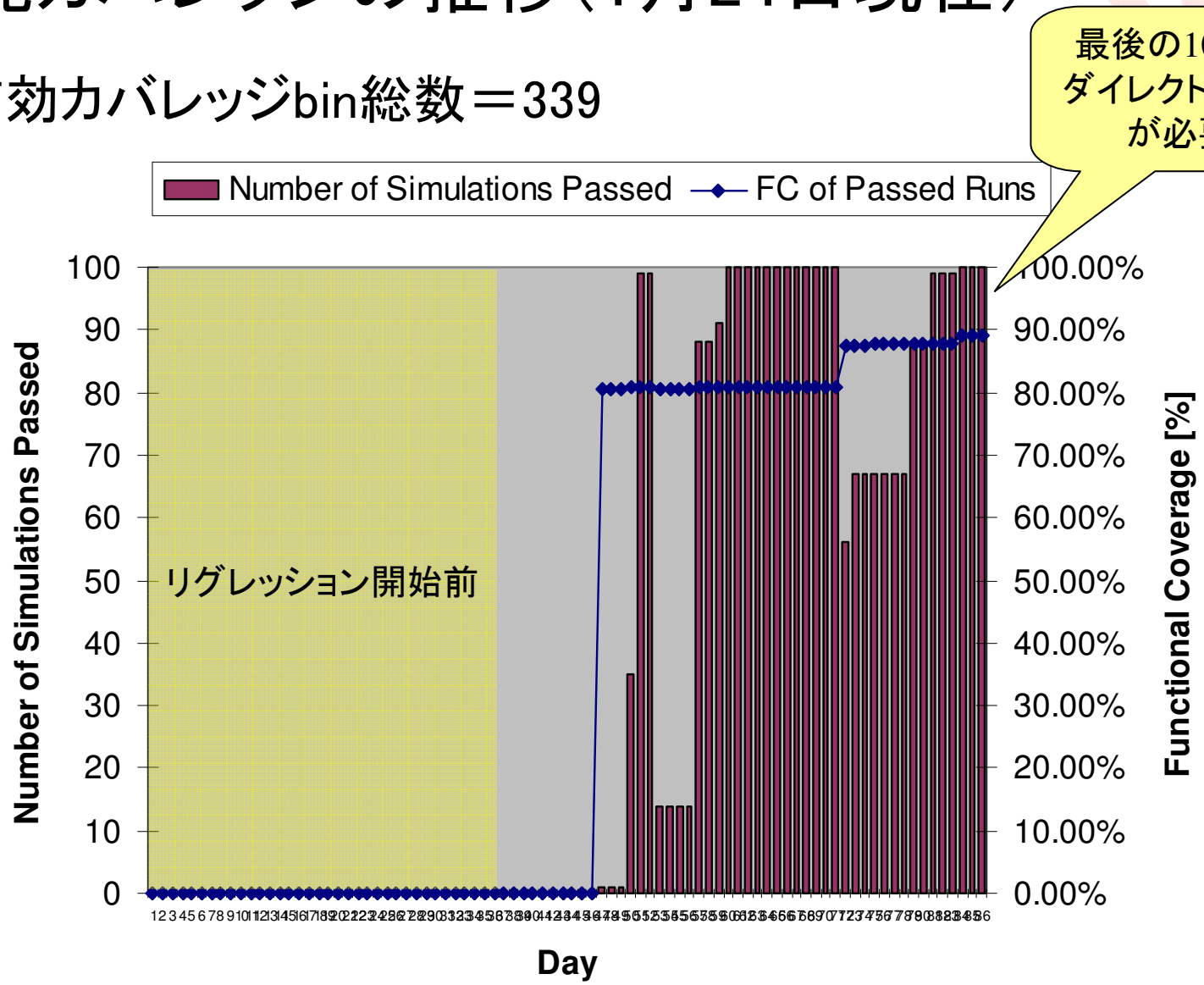
シミュレーションサイクル数の推移（1月24日現在）

- リグレッション開始後10日程度でサイクル数が急速に増加



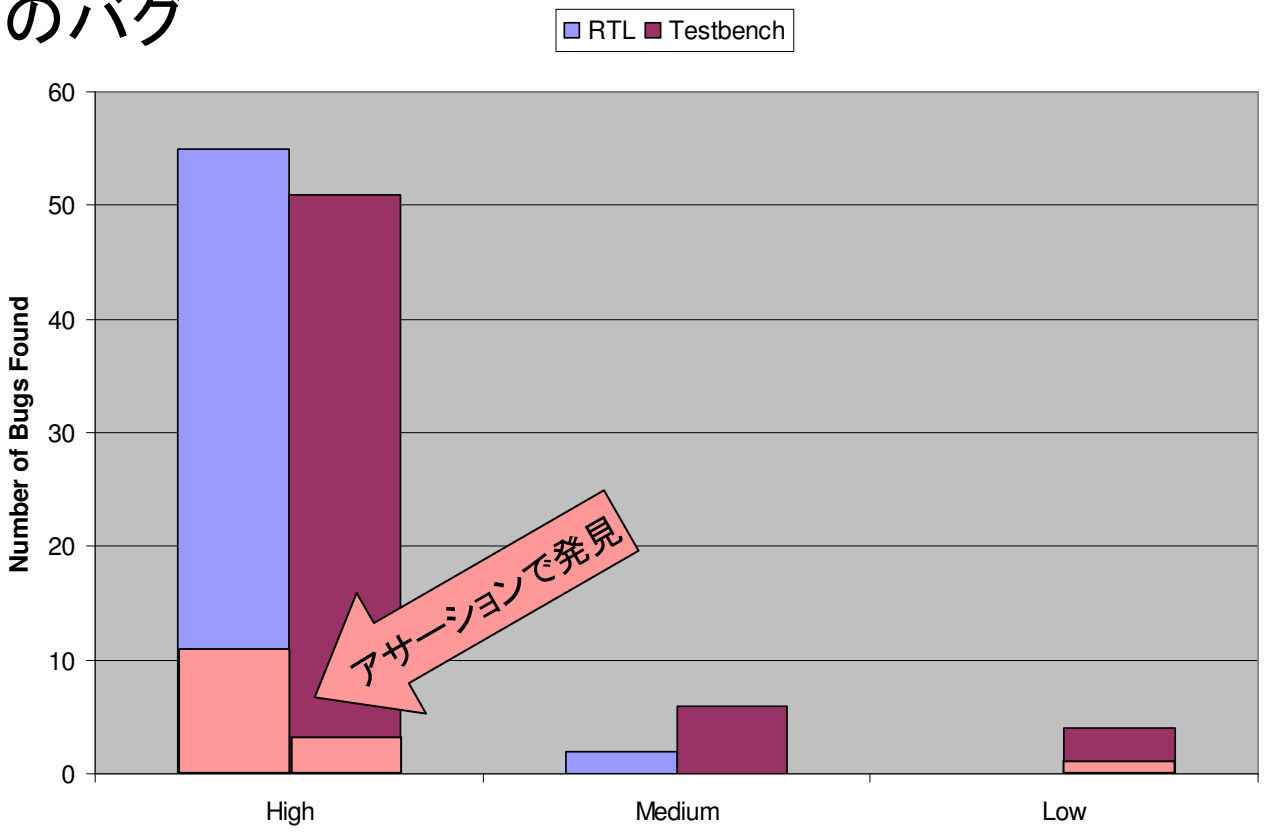
機能カバレッジの推移(1月24日現在)

- 有効カバレッジbin総数＝339



重要度別バグの分類(1月24日現在)

- High – 修正しなければシミュレーションが継続できないバグ
- Medium – シミュレーションは継続できるが重要なバグ
- Low – その他のバグ

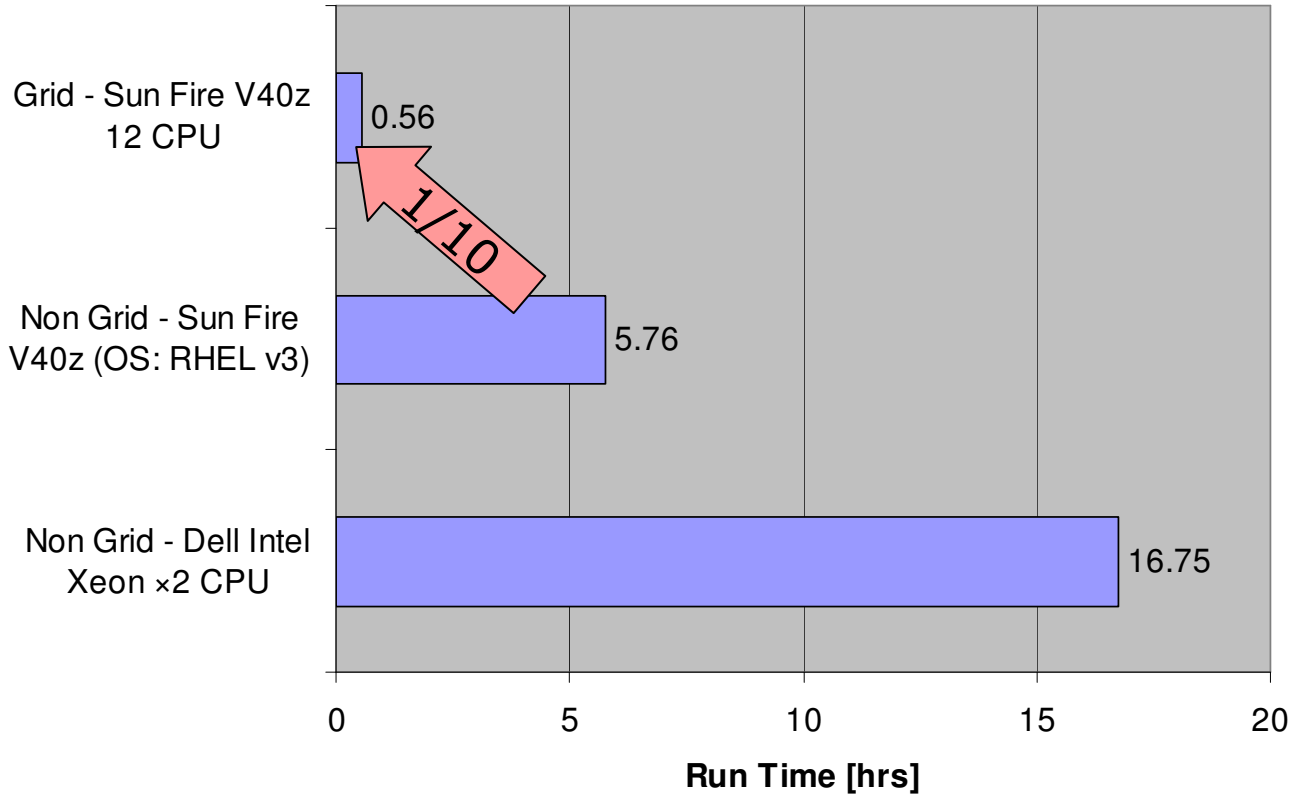


検出されたコーナーケースバグの例

- コーナーケースバグ1
  - “2つ以上のセグメントを連続で動かす際、S2APがジャンプをISDモジュールに発行し同期待ちをしている間、同期実行終了のACKを受ける前に、Rフラグつきのセグメントのフェッチが完了したとき、S2APがStop→Decode→Runへ状態遷移してしまう”
- コーナーケースバグ2
  - “あるクラスタからジャンプ例外を伴ってセグメント切替を行う場合、その例外をクリアするのと同時に、次クラスタの実行要求リクエストがS2APから到達した場合、ISDモジュールが実行開始のACKを返さない”

Grid環境を用いたシミュレーションの効果

- リグレッション時間の比較
  - Grid Engine (12 CPU)を用いることによりシミュレーション時間を約1/10に短縮



成果

- SystemVerilogを使用することにより、再利用可能な制約付きランダム検証環境を効率的に構築できた
- 下位の階層から順次実装することによって、検証環境の完成を待たずに検証作業を開始することができた
- 制約付きランダム検証により、短期間で十分な機能カバレッジを達成することができた
- Grid Engineによるサーバーファームを用いることで、制約付きランダム検証の効率を大幅にあげることができた

今後の課題

- 機能カバレッジ結果から制約条件へのフィードバック手法の確立
- アサーションIPの開発と活用

検証サービスのご案内

- 検証アーキテクチャの構築
  - 再利用を念頭に置いたアーキテクチャ
  - 制約付きランダム検証環境の構築にはノウハウが必要
    - 検証環境の品質が検証対象の品質を左右する
- 検証ライブラリを用いた検証環境、検証IPの開発
  - コントロール・レジスタ、AMBAトランザクタ等の汎用クラス・ライブラリ
- 検証ユーティリティの開発と提供
- 検証マネジメントに関するコンサルティング

お問合せ先

- ソナック株式会社LSI事業部
  - 〒222-0033 神奈川県横浜市港北区新横浜2-13-6 第一KSビル4階
  - 電話: (045) 470-0313
  - FAX: (045) 470-0326
  - E-mail: inq_lsi@sonac.co.jp
  - URL: http://lsi.sonac.co.jp