2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
JAPAN

付加価値の高い高位機能検証へのシフト

Siemens EDAジャパン株式会社 龍田 純一

SIEMENS

accellera
SYSTEMS INITIATIVE

アジェンダ

I. はじめに
II. シミュレーション高速実行が可能な高位モデル
III. 高位検証にシフトするための検証技術
IV. 高位検証におけるコードカバレッジ
V. 高位検証における機能カバレッジ
VI. 高位検証におけるカバレッジ・クローズ
VII. 考察とまとめ

I.　はじめに

RTL設計の機能検証における調査 (Wilson Research Group 2022)

- 機能検証が占める割合　　= 60-70%(多くのプロジェクト)
- リスピンなく初回完動　　= 24%
- リスピンの第一の原因　　= 機能的、論理的な誤り

=> 課題：RTL機能検証の質をいかに効率よく上げるか

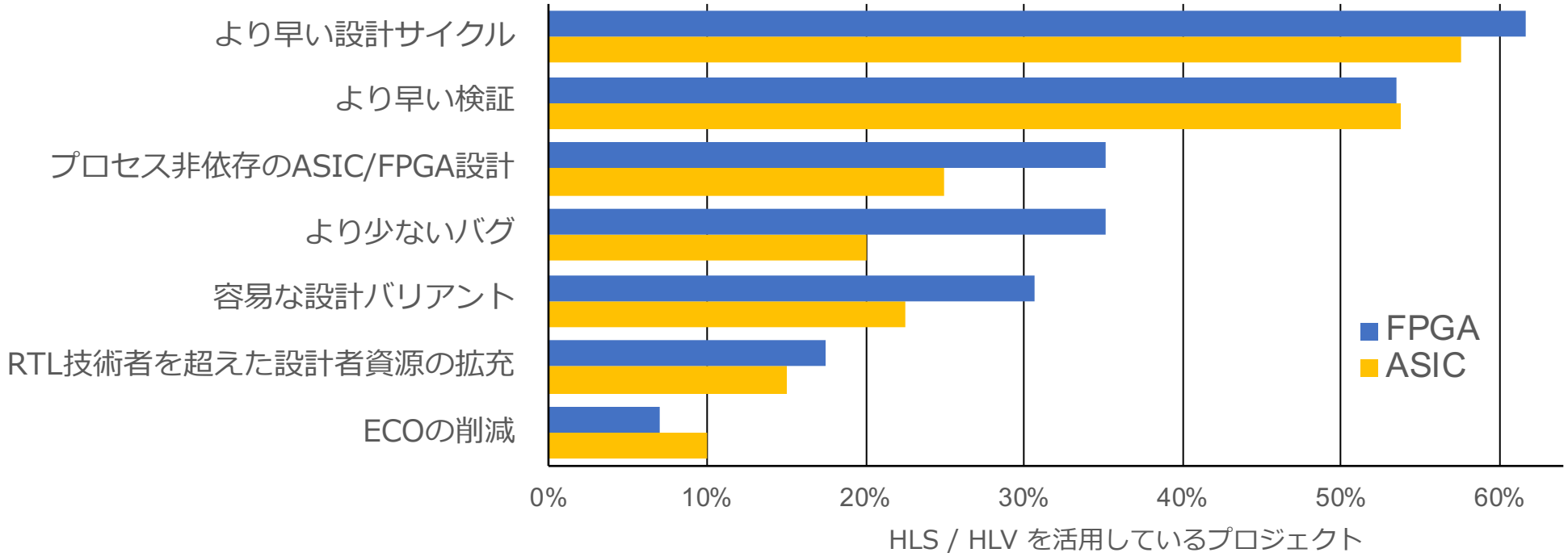
- 高位合成活用プロジェクト (ASIC/IC 30%, FPGA 40%)
- 課題(機能検証)に対する提案
  - 高位合成向け高位モデルを対象
  - 高位検証の検証技術に注目

II.　シミュレーション高速実行が可能な高位モデル

Wilson Research Group 2022
機能検証調査「高位合成を使用する利点・動機」

- より早い設計サイクル
- より早い検証
- プロセス非依存のASIC/FPGA設計
  - 共通の高位モデルからASIC/FPGA用RTLを生成
- より少ないバグ
- 容易な設計バリアント
  - 合成制約による多種多様なRTLを生成
- RTL技術を超えた設計者資源の拡大
- ECOの削減

Wilson Research Group 2022
機能検証調査「高位合成を使用する利点・動機」



- 高位合成モデル(C++, SystemC) = 数百倍高速
  - 等価なモデルにおける速度比較 vs RTL

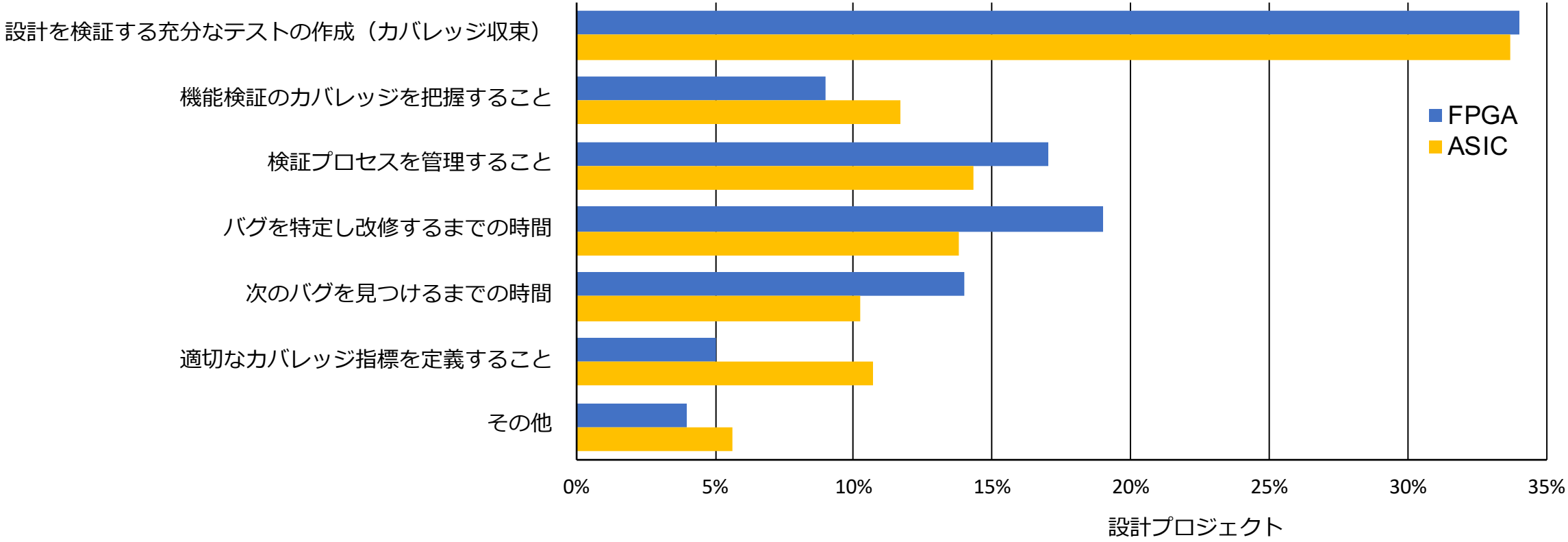
=> RTL機能検証を高位モデルへシフト

III.　高位検証にシフトするための検証技術

Wilson Research Group 2022
機能検証調査「機能検証の最大の課題」

- 設計を検証する充分なテストの作成（カバレッジ収束）
- 機能検証のカバレッジを把握すること
- 検証プロセスを管理すること
- バグを特定し改修するまでの時間
- 次のバグを見つけるまでの時間
- 適切なカバレッジ指標を定義すること

Wilson Research Group 2022
機能検証調査「機能検証の最大の課題」



高位検証へのシフトとして注目した課題

- 設計を検証する十分なテストの作成（カバレッジ収束）

高位検証にシフトするための検証技術

カバレッジ収束

1. 高位モデルにおけるビット精度が表現可能なデータタイプの活用
   → ACデータタイプ(オープンソース)が広く使用されている（スコープ外）
2. ハードウェア実装を考慮した高位モデルにおけるコードカバレッジの測定
   → Catapult Coverage(CCOV)コンセプト
3. 高位モデルに対する検証プランに基づいた機能カバレッジの策定と測定
   → SystemVerilog 機能カバレッジの手法にC++ライブラリを開発

IV.　高位検証におけるコードカバレッジ

IV. 高位検証におけるコードカバレッジ

- gcov = ソフトウェア向けカバレッジ・ツール
  - Linux標準装備
  - gcc/g++ へ追加オプションで利用可能
  - 高位モデル(C++,SystemC)でも利用可能

- ソフトウェア視点の活性化カバレッジを対象
  - ハードウェアのRTL実装を意識したカバレッジ測定はできない(=>課題)

ハードウェア実装を考慮したコードカバレッジとは

A) 関数のインスタンス単位のカバレッジ測定

```
// Testbench

data_in_a_t    a = -20 ;
data_in_b_t    b = 20 ;

top_level(a ,a_abs, b, b_abs);
```

SWカバレッジ=100%

```
 1  template <class T, class T2>
 2  void my_abs (T &a, T&b ){
 3      if (a<0) {
 4          b = -a ;
 5      } else {
 6          b = a ;
 7  }
 8
 9  void top_level (
10      data_in_a_t &a,
11      data_out_a_t &a_abs,
12      data_in_b_t &b,
13      data_out_b_t &b_abs,
14  ) {
15      my_abs(a, a_abs); //8-bit
16      my_abs(b, b_abs); //9-bit
17  }
```

HW実装を考慮したカバレッジ=50%

```
void top_level (
        data_in_a_t &a,
        data_out_a_t &a_abs,
        data_in_b_t &b,
        data_out_b_t &b_abs,
) {
    // 8-bit
    if (a<0) {
        b = -a ;
    } else {
        b = a ;
    }
    // 9-bit
    if (a<0) {
        b = -a ;
    } else {
        b = a ;
    }
}
```

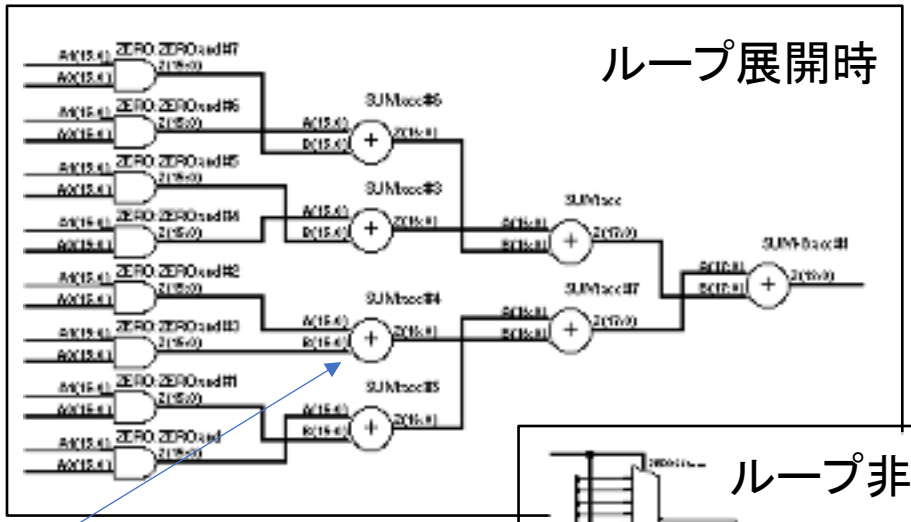
my_abs(a, a_abs);//8-bit

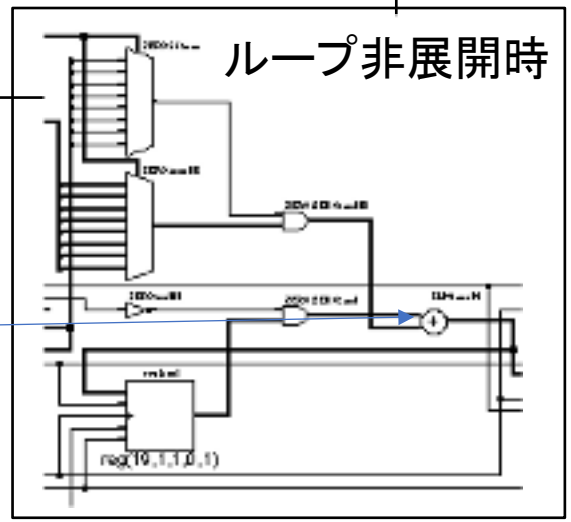
my_abs(b, b_abs);//9-bit

ハードウェア実装を考慮したコードカバレッジとは

B) ループ展開を考慮したカバレッジ測定(ループ制約：高位合成でよく使われる最適化)

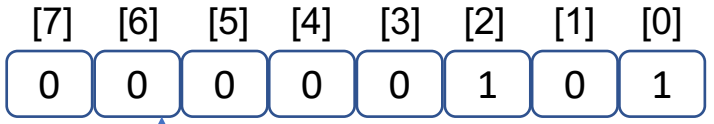
```
1  #include "unrolling.h"
2  void testcase (
3      ac_int<8,false> &set_in,
4      ac_int<16,false> a[8],
5      ac_int<19,false> &result
6  ) {
7      ac_int<8,false> set = set_in ;
8      ac_int<16,false> partial[8] ;
9      ZERO:for(int i=0;i<8;i++) {
10       partial[i] = a[i] ;
11       if (set[i]==0) {
12           partial[i] = 0 ;
13     }
14   }
15     ac_int<19,false> acc = 0 ;
16     SUM:for(int i=0;i<8;i++) {
17         acc += partial[i] ;
18     }
19   result = acc ;
20 }
```





ハードウェア実装を考慮したコードカバレッジとは

- ループ非展開設定時のカバレッジ




```
 1  #include "unrolling.h"
 2  void testcase (
 3     ac_int<8,false> &set_in,
 4     ac_int<16,false> a[8],
 5     ac_int<19,false> &result
 6  ) {
 7     ac_int<8,false> set = set_in ;
 8     ac_int<16,false> partial[8] ;
 9    ZERO:for(int i=0;i<8;i++) {
10        partial[i] = a[i] ;
11        if (set[i]==0) {
12            partial[i] = 0 ;
13        }
14    }
15     ac_int<19,false> acc = 0 ;
16    SUM:for(int i=0;i<8;i++) {
17         acc += partial[i] ;
18    }
19    result = acc ;
20  }
```

ループ文を含む高位モデル記述例

```
int main() {
  ac_int<8,false> set = 5;
  ac_int<16,false> a[8] =  {1,2,3,4,5,6,7,8};
  ac_int<19,false> result ;

  testcase(set, a, result);
}
```

テストベンチ記述例

※ ac_int<8,false> set;
set[i] ACデータタイプの[ ] ＝ビットアクセス

```
Design   |Hits %  |Coverage   |Bins  |Hits  |Misses|Weight|%Hit    |
Scope    |        |Type       |      |      |      |      |        |
---------+--------+-----------+------+------+------+------+--------+
testcase|100.00%|Statement  |    11|    11|     0|     1|100.00%|
         |        |Branches   |     6|     6|     0|     1|100.00%|
```

高位モデルのコードカバレッジ

ハードウェア実装を考慮したコードカバレッジとは

- ループ展開設定時のコードカバレッジ(ループ制約：高位合成でよく使われる最適化)

Coverage Summary by Structure:

Design Scope	Hits %	Coverage %
testcase_2	77.27%	71.42%
ZERO_1	50.00%	50.00%
ZERO_2	75.00%	75.00%
ZERO_3	50.00%	50.00%
ZERO_4	75.00%	75.00%
ZERO_5	75.00%	75.00%
ZERO_6	75.00%	75.00%
ZERO_7	75.00%	75.00%
ZERO_8	75.00%	75.00%
SUM_1	100.00%	100.00%
SUM_2	100.00%	100.00%
SUM_3	100.00%	100.00%
SUM_4	100.00%	100.00%
SUM_5	100.00%	100.00%
SUM_6	100.00%	100.00%
SUM_7	100.00%	100.00%
SUM_8	100.00%	100.00%

Coverage Summary by Type:

Total Coverage: (Filtering Active)					77.27%	71.42%
Coverage Type	Bins	Hits	Misses	Weight	% Hit	Coverage
Statements	28	26	2	1	92.85%	92.85%
Branches	16	8	8	1	50.00%	50.00%

※ZERO_1 – 8 の各if文条件が対象(真、偽)

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
0	0	0	0	0	1	0	1

```
int main() {
  ac_int<8,false> set = 5;
  ac_int<16,false> a[8] =  {1,2,3,4,5,6,7,8};
  ac_int<19,false> result ;

  testcase(set, a, result);
}
```

テストベンチ記述例

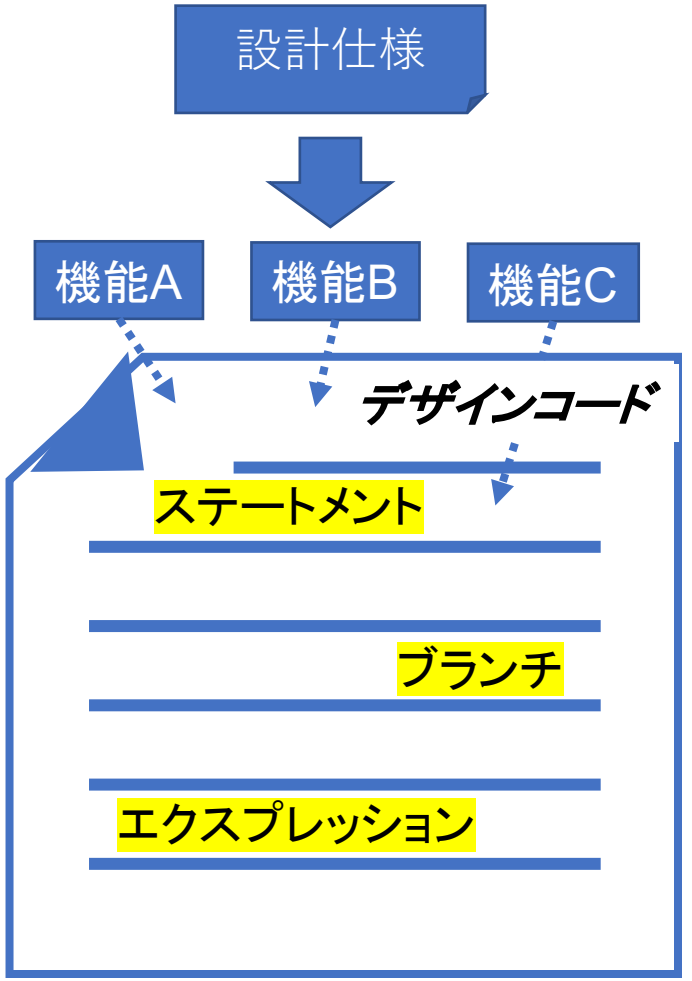
```
#include "unrolling.h"
void testcase (
    ac_int<8,false> &set_in,
    ac_int<16,false> a[8],
    ac_int<19,false> &result
) {
    ac_int<8,false> set = set_in ;
    ac_int<16,false> partial[8] ;
  ZERO:for(int i=0;i<8;i++) {
      partial[i] = a[i] ;
      if (set[i]==0) {
          partial[i] = 0 ;
      }
  }
    ac_int<19,false> acc = 0 ;
  SUM:for(int i=0;i<8;i++) {
      acc += partial[i] ;
  }
  result = acc ;
}
```

ループ文を含む高位モデル記述例

V.　高位検証における機能カバレッジ

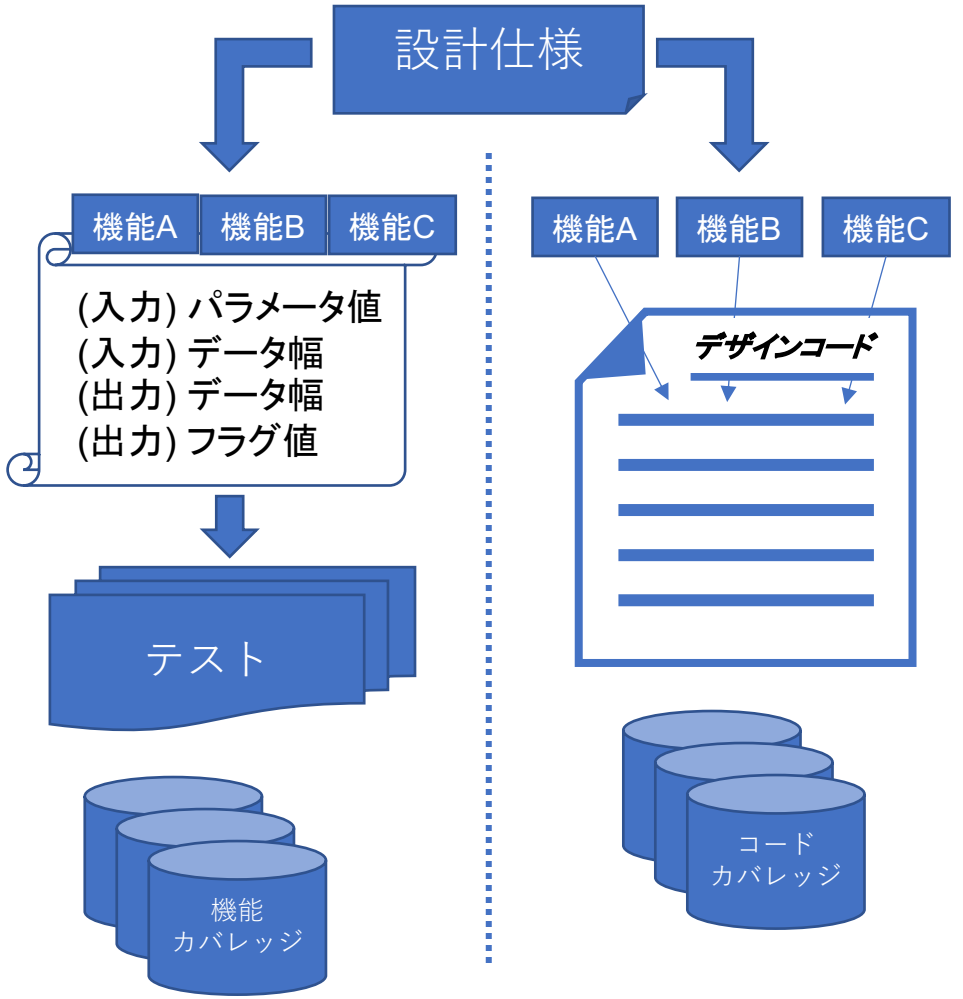
高位検証：コードカバレッジと機能カバレッジ

- コードカバレッジ
  - カバレッジ・メトリックスの視点
    - 実装されたコードを構造視点でどれだけテストされたかを測る指標
  - カバレッジ・メトリックスの作成
    - 決められたメトリックス(ステートメント、ブランチ、エクスプレッションなど)をツールが自動的に抽出
- IV. 高位検証におけるコードカバレッジ
  - ハードウェア実装を考慮したコードカバレッジ



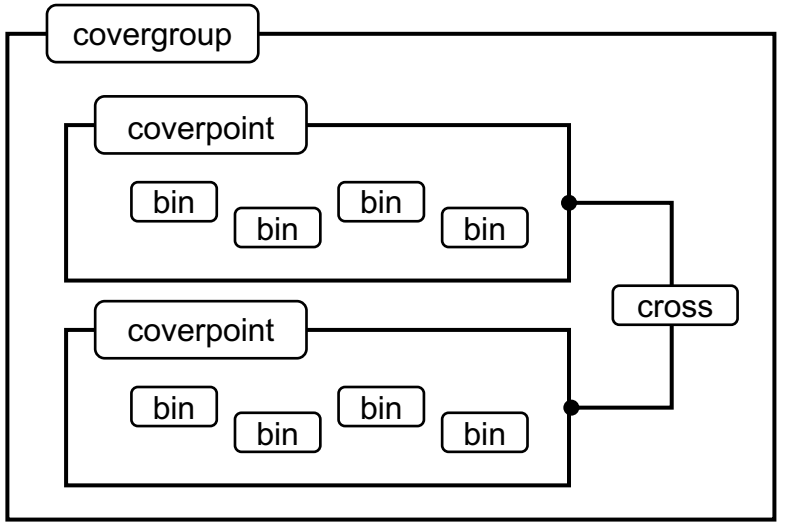
高位検証：コードカバレッジと機能カバレッジ

- 機能カバレッジ
  - カバレッジ・メトリックスの視点
    - 仕様書の定義に対しテストが機能的要件を満たすことを測る指標
  - カバレッジ・メトリックスの作成
    - 設計の要求書や仕様書から手作業でメトリックスを作成



高位検証：機能カバレッジ

- SystemVerilog仕様に倣いC++ライブラリを開発
  - IEEE Std. 1800-2017



高位モデルのテスト用機能カバレッジの構成

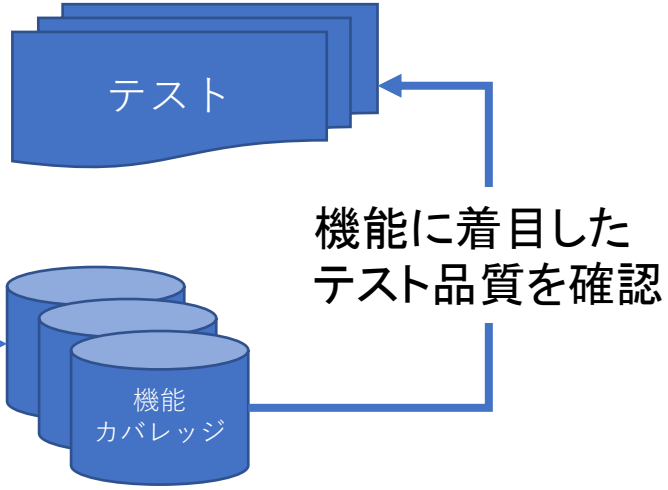
```
class MyCCoverGroup : public CCoverGroup {
public:
 ac_int<13, false> kType;
 ac_int<3, false> dataOut;
 MyCCoverGroup(std::string cg_inst_name):CCoverGroup(cg_inst_name)
 {
  CCoverPoint<ac_int<13, false> > *cp1 = AddCCoverPoint("cp_block_size", kType);
    cp1->AddPointBin("pb_zero", value<AC_VAL_MIN>(kType));
    cp1->AddRangeBin("rb_40_512_blksz_8", 40, 512);
    cp1->AddRangeBin("rb_2112_6144_blksz_64", 2112, 6144);

    CCoverPoint<ac_int<3,false> > *cp2 =AddCCoverPoint("cp_data_out",dataOut);
    cp2->AddFullRangeBin("rb_out",0,7);

    AddCCoverCross("cross_cp1_cp2",cp1, cp2);
  }
```

高位モデルで使用する機能カバレッジの記述例

機能A　機能B　機能C

(入力) パラメータ値
(入力) データ幅
(出力) データ幅
(出力) フラグ値

検証プラン毎に
機能カバレッジモデルを作成

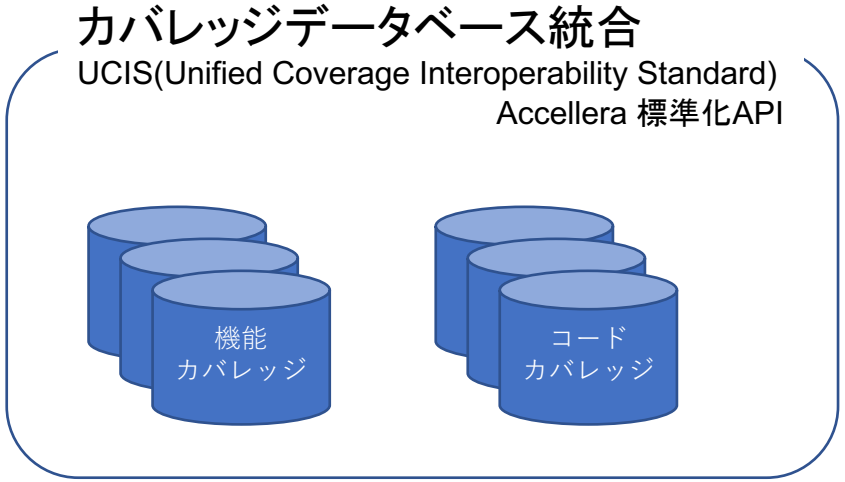


機能に着目した
テスト品質を確認

VI.　高位検証におけるカバレッジ・クローズ

コードカバレッジと機能カバレッジを集約

		コードカバレッジ	
		低	高
機能カバレッジ	低	• テスト不足	• コーナーケース未達 • DUT機能実装不足
	高	• 未到達コードの存在 • 設計仕様にない機能の存在	• 検証完了



- コードカバレッジ（高）/ 機能カバレッジ（**低**）
  - コーナーケース未達
  - DUT機能実装不足
- コードカバレッジ（**低**）/ 機能カバレッジ（高）
  - 設計仕様にない機能の存在
  - 未到達コードの存在（デッドコード）

カバレッジ結果を総合的に解析

VII. 考察とまとめ

VII.考察とまとめ

- プロジェクト課題：RTL機能検証の質をどうあげるか
- 高位合成活用プロジェクト(ASIC 30%, FPGA 40%)に注目
  - 「高位検証における早い検証」において機能検証の質を上げる
    1. 高位モデルにおけるビット精度が表現可能なデータタイプの活用 (スコープ外)
    2. ハードウェア実装を考慮した高位モデルにおける**コードカバレッジ**の測定
    3. 高位モデルに対する検証プランに基づいた**機能カバレッジ**の策定と測定

  => 高位検証へのシフトレフトを実現

- 高位検証によるカバレッジ品質向上＝高位合成の生産性を最大化
  - 比較：RTLシミュレーション時の検証プランの漏れや抜け(生産性低下原因)
    - シミュレーション時間もかかる

質疑応答