Static Checking for Correctness of Functional Coverage Models

Wael Mahmoud

Mentor®
A Siemens Business

Agenda

- Introduction
- Functional coverage closure problems
- Static enhancements of functional coverage models
  - Part A: Enhancements of input/output functional coverage
  - Part B: Enhancements of design-centric functional coverage
- Results and conclusion

accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Introduction

- Today’s designs are getting more bigger and more complex (SoC and ASIC)
- Achieving fully verified SoC is an arduous task.
- Recent industry studies, shows that the average total project time spent in verification was 57%.
- Number of projects that spent more than 80% of time in verification has been increased from the past.

© Accellera Systems Initiative

Motivation

- The intent of verifying "SoC" is to ensure that the design is an accurate representation of the specification.
- Functional coverage provides visibility into the verification process.
- Writing a complete, correct, and concise functional coverage models, that conform design functionality to specs.
- Accelerate functional coverage closure.
- Assist verification teams with techniques to write concise functional coverage models.

Agenda

- Introduction
- Functional coverage closure problems
- Static enhancements of functional coverage models
  - Part A: Enhancements of input/output functional coverage
  - Part B: Enhancements of design-centric functional coverage
- Results and conclusion

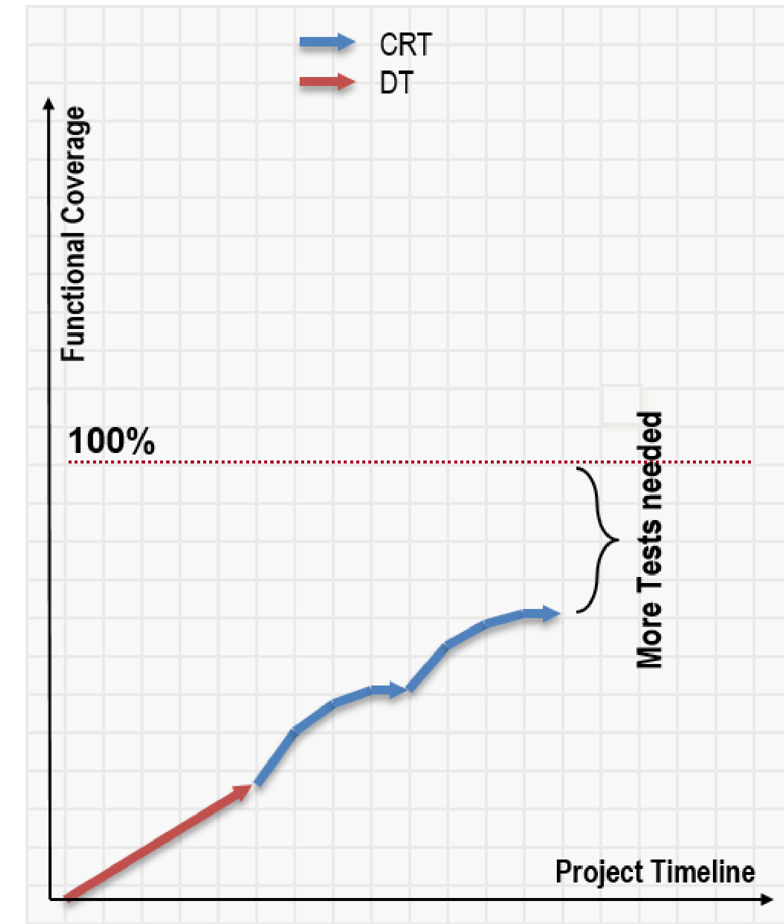
© Accellera Systems Initiative

Functional coverage closure problems

- Functional coverage closure can't be achieved due to many problems, like:
  1. Problems with input stimuli, like: incomplete, insufficient, and/or redundant stimuli
  2. Incorrect implementation of functional coverage model.
  3. Non-optimized forms of functional coverage.

1- Incomplete/redundant input stimuli

- Write more directed tests to cover specific corner case scenarios.
- Run test cases multiple times with different random seeds, and hope more interesting scenarios are covered.
- Alternatively, try out other methodologies (e.g. intelligent test-bench automation "iTBA" tools) when applicable.



© Accellera Systems Initiative

2- Incorrect implementation of functional coverage model

- Functional coverage model is contradicting with test-bench’s or design’s constraints.
- The proposed methodology will shows that there are no possible solutions.

```
rand logic unsigned [0:3] a;

constraint C {
        a inside {[10:15]};
}

cp_a: coverpoint a {
        bins b1[] = {[0:9]};
}
```

3- Non-optimized forms of functional coverage

Functional coverage model is not written in an optimized form (i.e. it is not considering unreachable bins).

Input functional coverage

```
rand bit [3:0] A;

constraint A_constr {
                A < 8;
}
…
covergroup cov;
                A_cp: coverpoint A;
endgroup
```

Coverage of ***A_cp*** is 50 %

Design-centric functional coverage

```
always @(posedge fsm_clk or negedge
fsm_reset_n)
if(!fsm_reset_n)
   int_state <= idle;
else
   int_state <= nxt_state;

always @(*)
begin
  nxt_state = int_state;
  case (int_state)
   idle:
     if(in_hs)
      nxt_state = send_bypass;
     else
      nxt_state = idle;
   send_bypass:
    if(out_hs)
     if(enable)
      nxt_state = load_bypass;
     else
      nxt_state = idle;
   load_bypass:
     if(in_hs)
      nxt_state = send_bypass;
   wait_idle:
    if(out_hs)
     nxt_state = idle;
  endcase
end
```

```
covergroup sm_cvg @(posedge pins.clk);
  coverpoint int_state;
endgroup
```

Coverage of ***wait_idle*** is 0 %

accellera SYSTEMS INITIATIVE
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Introduction
- Functional coverage closure problems
- Static enhancements of functional coverage models
  - Part A: Enhancements of input/output functional coverage
  - Part B: Enhancements of design-centric functional coverage
- Results and conclusion

© Accellera Systems Initiative

Static enhancements of functional coverage models

This paper proposes a complete framework to enhance functional coverage models of both "*input/output*" and "*design-centric*"

"Part A"

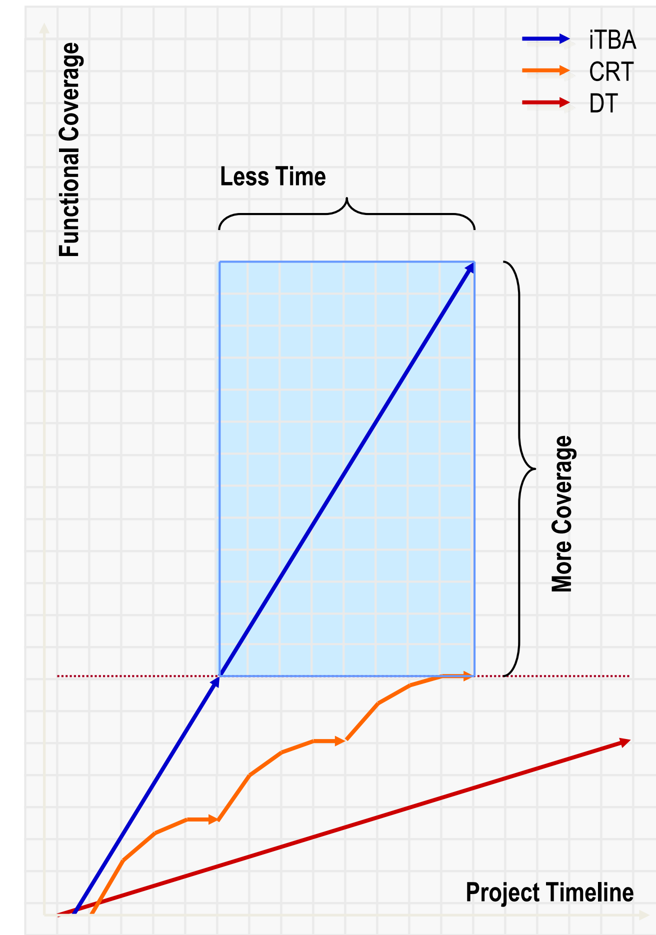
Intelligent test-bench automation (iTBA) tool, which internally use constraint solver technologies, is used to enhance "input/output" functional coverage model

"Part B"

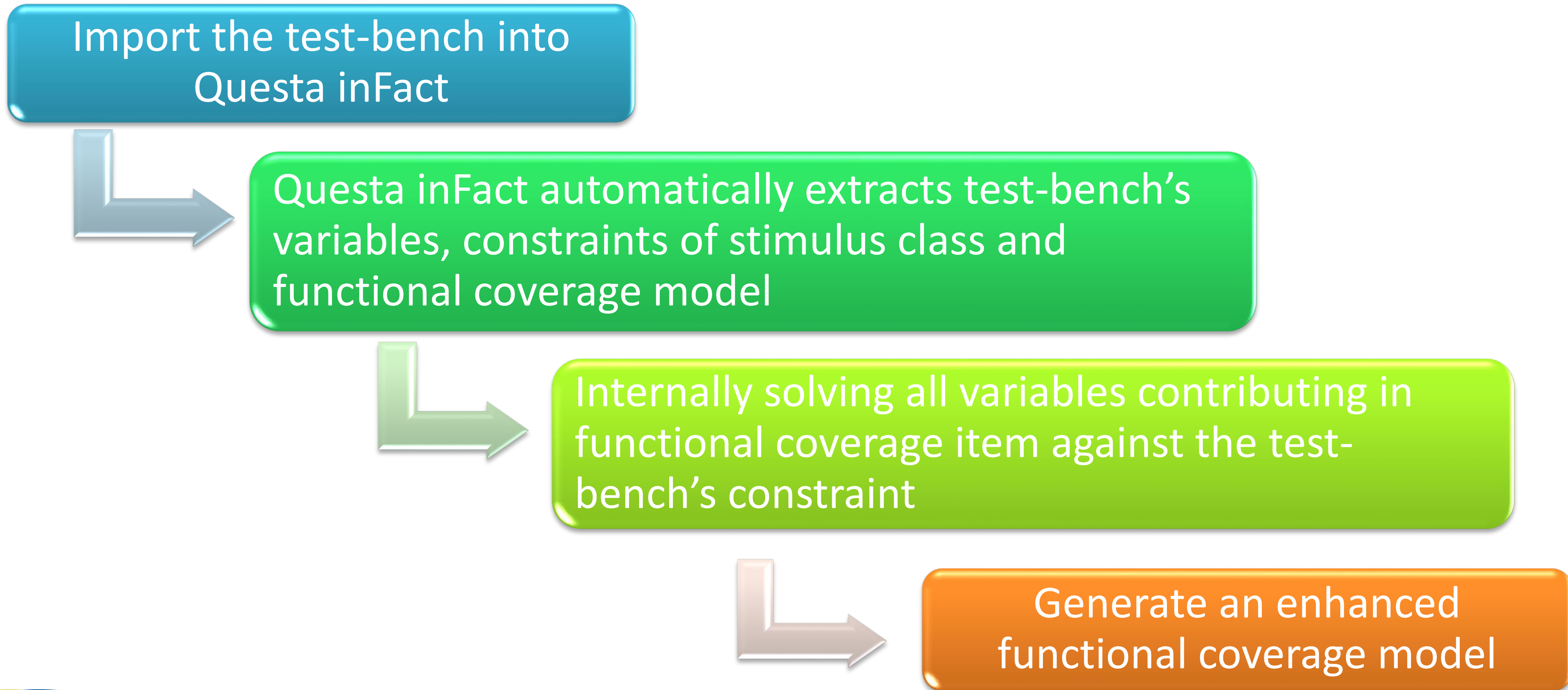
Formal-based coverage analysis tool, which internally use formal-based analysis, is used to enhance "design-centric" functional coverage model

Intelligent test-bench automation (iTBA) tools

- iTBA tools achieves input coverage 10-100x faster than random stimulus.
- More than 100x productive than directed test
  - It provides an efficient description of stimulus scenarios
  - It reduces time spent in writing testbenches
- Mote than 10X efficient than constrained random tests
  - No redundant tests
  - It helps to find tough corner case bugs easier and earlier
- This paper is using iTBA tool to enhance input/output functional coverage models.



Part A: Enhancements of input/output functional coverage (1/3)



© Accellera Systems Initiative

Part A: Enhancements of input/output functional coverage (2/3)

Original F.C.

```
rand bit [3:0] A;
 constraint A_constr {
                A < 8;
}
covergroup cov;
                A_cp: coverpoint A;
endgroup
```

Enhanced F.C.

```
A_cp : coverpoint A {
option.weight = 8;
bins cfg_item_inst_A[] = {[64'd0:64'd7]};
}
```

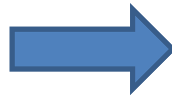
© Accellera Systems Initiative

accellera SYSTEMS INITIATIVE
2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Part A: Enhancements of input/output functional coverage (3/3)

Original F.C.

```
rand logic unsigned [0:3] A, B;
constraint add_constr {
    A + B >= 0;
    A + B <= 10;
  }
…
covergroup cov;
A_cp: coverpoint A;
B_cp: coverpoint B;
cr1: cross A_cp, B_cp;
endgroup
```



Enhanced F.C.

```
A_cp : coverpoint A {
        option.weight = 0;
        bins A_bins[] = {[64'd0:64'd10]};
    }
B_cp : coverpoint B {
        option.weight = 0;
        bins B_bins[] = {[64'd0:64'd10]};
    }
cr1 : cross A_cp, B_cp {
        option.weight = 66;
        ignore_bins unreachable_bins = ((binsof(A_cp) intersect {64'd1} &&
binsof(B_cp) intersect {64'd10}) || (binsof(A_cp) intersect {64'd2} && binsof(B_cp)
intersect {64'd9, 64'd10}) || (binsof(A_cp) intersect {64'd3} && binsof(B_cp) intersect
{64'd8, 64'd9, 64'd10}) || (binsof(A_cp) intersect {64'd4} && binsof(B_cp) intersect
{64'd7, 64'd8, 64'd9, 64'd10}) || (binsof(A_cp) intersect {64'd5} && binsof(B_cp)
intersect {64'd6, 64'd7, 64'd8, 64'd9, 64'd10}) || (binsof(A_cp) intersect {64'd6} &&
binsof(B_cp) intersect {64'd5, 64'd6, 64'd7, 64'd8, 64'd9, 64'd10}) || (binsof(A_cp)
intersect {64'd7} && binsof(B_cp) intersect {64'd4, 64'd5, 64'd6, 64'd7, 64'd8, 64'd9,
64'd10}) || (binsof(A_cp) intersect {64'd8} && binsof(B_cp) intersect {64'd3, 64'd4,
64'd5, 64'd6, 64'd7, 64'd8, 64'd9, 64'd10}) || (binsof(A_cp) intersect {64'd9} &&
binsof(B_cp) intersect {64'd2, 64'd3, 64'd4, 64'd5, 64'd6, 64'd7, 64'd8, 64'd9, 64'd10}) ||
(binsof(A_cp) intersect {64'd10} && binsof(B_cp) intersect {64'd1, 64'd2, 64'd3, 64'd4,
64'd5, 64'd6, 64'd7, 64'd8, 64'd9}) || (binsof(A_cp) intersect {64'd10} && binsof(B_cp)
intersect {64'd10}));        }
```

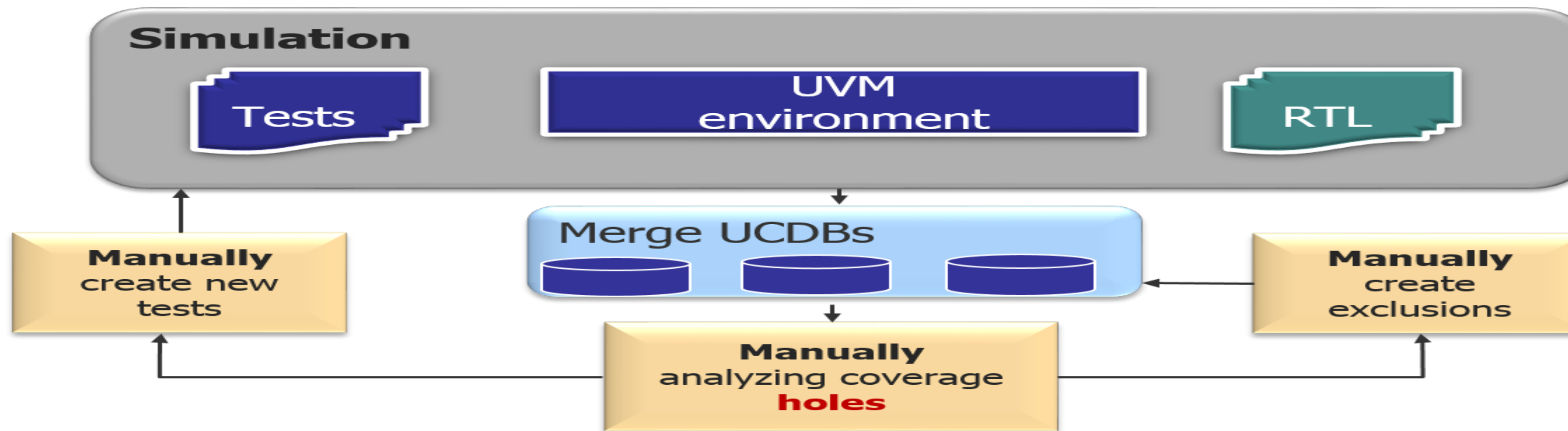

© Accellera Systems Initiative

Agenda

- Introduction
- Functional coverage closure problems
- Static enhancements of functional coverage models
  - Part A: Enhancements of input/output functional coverage
  - Part B: Enhancements of design-centric functional coverage
- Results and conclusion

© Accellera Systems Initiative

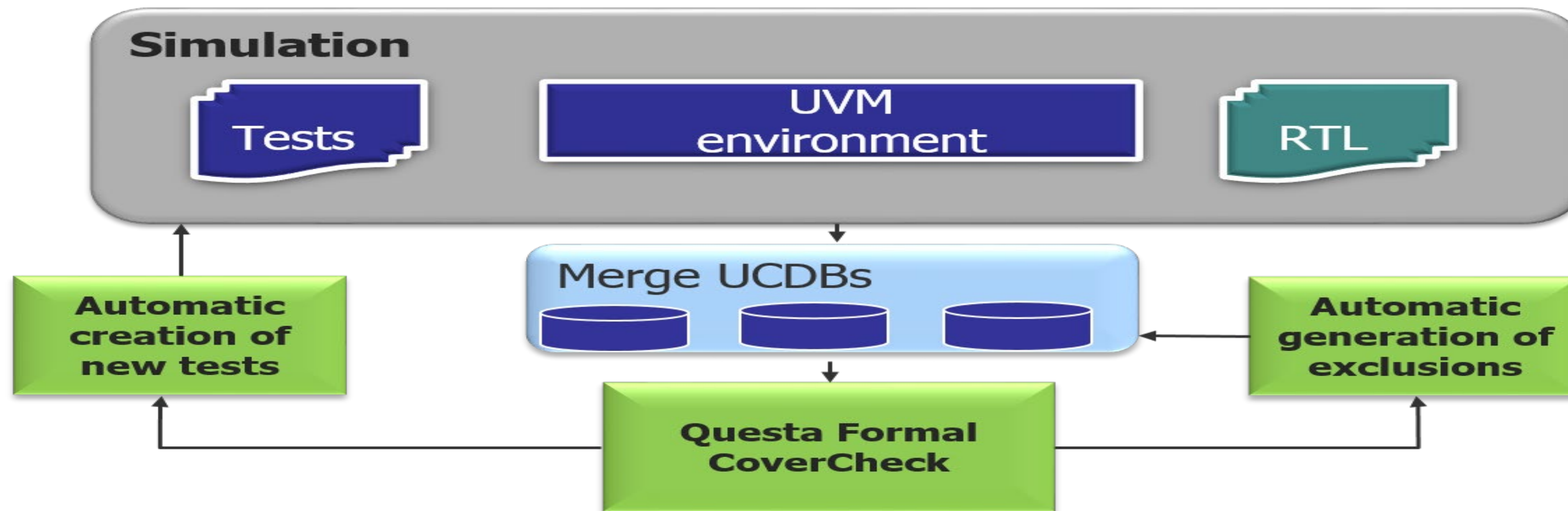
Manual coverage closure (design-centric)



Manual Coverage Closure challenges

- ❑ Coverage verification is to verify that coverage goal is achieved in simulation
- ❑ Testing all possible scenarios and states are generally so hard
- ❑ Coverage holes indicate:
  - ➢ Some blocks, states and transactions in the design are unreachable
  - ➢ Some coverage items are reachable with complex test scenarios
- ❑ Huge effort and time are consumed to determine unreachable code and to create complex tests

Coverage closure using formal-based analysis (design-centric)



Formal-based analysis tool for automatic Coverage Closure

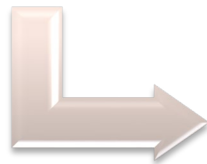
- ❑ Save time that would been spent for manually analyzing coverage holes
- ❑ CoverCheck provides an automatic solutions for the Coverage Closure challenges
  - ✓ **Automatically exclude coverage items for unreachable code**
  - ✓ **Automatically generate Witness waveforms for reachable code**
- ❑ Customers can easily improve the code and the tests for better coverage metrics

© Accellera Systems Initiative

Part B: Enhancements of design-centric functional coverage

Run Questa CoverCheck on DUT and pass the UCDB generated from a simulation run

Questa CoverCheck automatically analyzes DUT for formal/static reachability using formal-based analysis



Exclusions file is generated with unreachable functional coverage bins, which is applied to simulation UCDB to exclude unreachable functional coverage

Agenda

- Introduction
- Functional coverage closure problems
- Static enhancements of functional coverage models
  - Part A: Enhancements of input/output functional coverage
  - Part B: Enhancements of design-centric functional coverage
- Results and conclusion

© Accellera Systems Initiative

Results

Input/Output F.C.

Interleaver Design

Coverage item name	Type	Coverage results without new approach	Coverage results with new approach
up_cvg::upcov_data	Cover-point	0.7%	100%
up_cvg::upcov_sync	Cover-point	40%	100%
up_cvg::up_delay	Cover-point	95%	100%

Ethernet Design

Coverage item name	Type	Coverage results without new approach	Coverage results with new approach
ethmac_rxtx_seq_cg::tx_size	Cover-point	85.9%	92%
ethmac_rxtx_seq_cg::rx_size	Cover-point	84.4%	84.6%
ethmac_rxtx_seq_cg::rx_tx_size	Cross	2.9%	3.1%

Design-Centric F.C.

Interleaver Design

Coverage item name	Type	Coverage results without new approach	Coverage results with new approach
sm_cvg::int_state	Cover-point	92.3%	96%
sm_cvg::in_hsXint_state	Cross	46.1%	92.3%
sm_cvg::out_hsXint_state	Cross	46.1%	100%

Ethernet Design

Coverage item name	Type	Coverage results without new approach	Coverage results with new approach
HASH0_1_Cvg::BYTE2	Cover-point	0.7%	100%
HASH0_1_Cvg::BYTE3	Cover-point	0.7%	100%
HASH0_1_Cvg::BYTE4	Cover-point	0.7%	100%
HASH0_1_Cvg::BYTE5	Cover-point	0.7%	100%
EthRw_Cvg::wrXaddrXdata	Cross	25%	25%

Functional coverage development become easier

Testbench constraints

- Automatically exclude unreachable coverage bins, and provide concise forms of F.C., which leverage coverage results

Design conditions

- Automatically exclude unreachable bins, which leads to improve DUT for better coverage metrics

Detect conflicts

- Constraints and original functional coverage conflict can be easily detected

Minimize manual mistakes

- Manual writing of exclusion bins is a common source of mistakes

© Accellera Systems Initiative

Conclusion

- Writing complete, correct, and concise functional coverage models to verify the correctness of SoC is a challenging task.
- The proposed methodology uses constraint solvers and formal-based analysis to enhance functional coverage models.
- The proposed methodology is helpful in writing correct and concise functional coverage models.
- The proposed methodology helps verification engineer to start writing functional coverage models, or re-calibrate existing coverage metrics.
- Proposed methodology saves effort and time to determine unreachable code or coverage bins.

© Accellera Systems Initiative

Thank You!

Any questions?