

New Developments in Early RTL Formal Analysis (早期RTLフォーマル解析の新開発手法)

Chris Browy: Avery Design Systems VP Marketing and Sales
横川秀美: ChipStart Japan Sales Director

「EDSフェア2011、出展社セミナー1月27日(木) 12:30~13:15 会場E204」



背景・動機

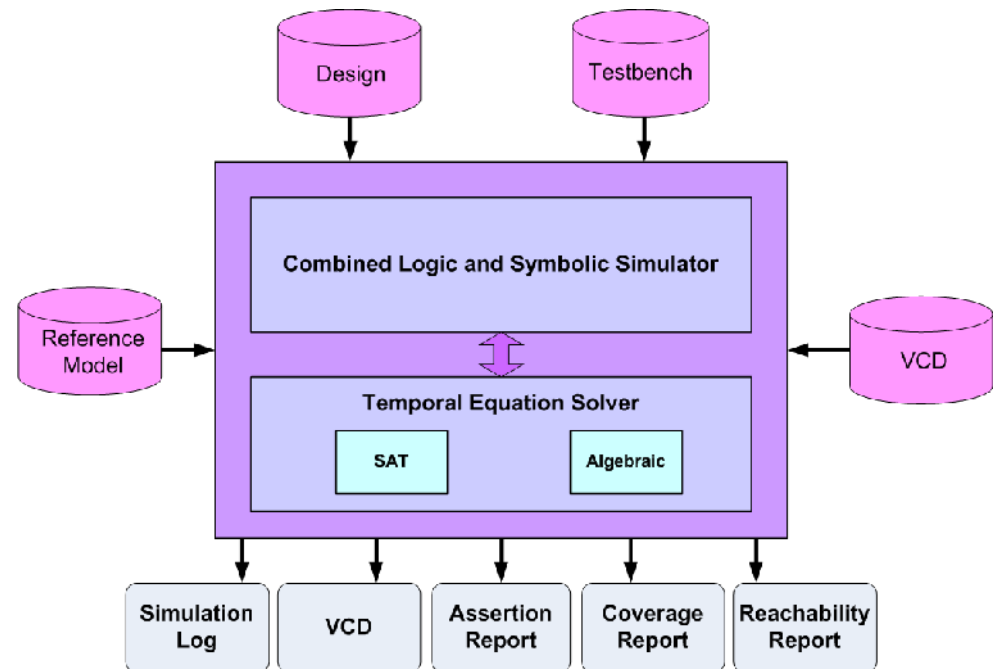
- SoC設計は次のような影響によって、益々チャレンジが増大しています
 - Process technology
 - Design complexity
 - Convergence of HW/SW
 - Time to market
- 設計手法は、様々なチャレンジをしながら、より積極的且つトップダウン・アプローチで常に進化しています
 - Increase productivity and eliminate bottlenecks
 - Improve overall quality

手法の進化

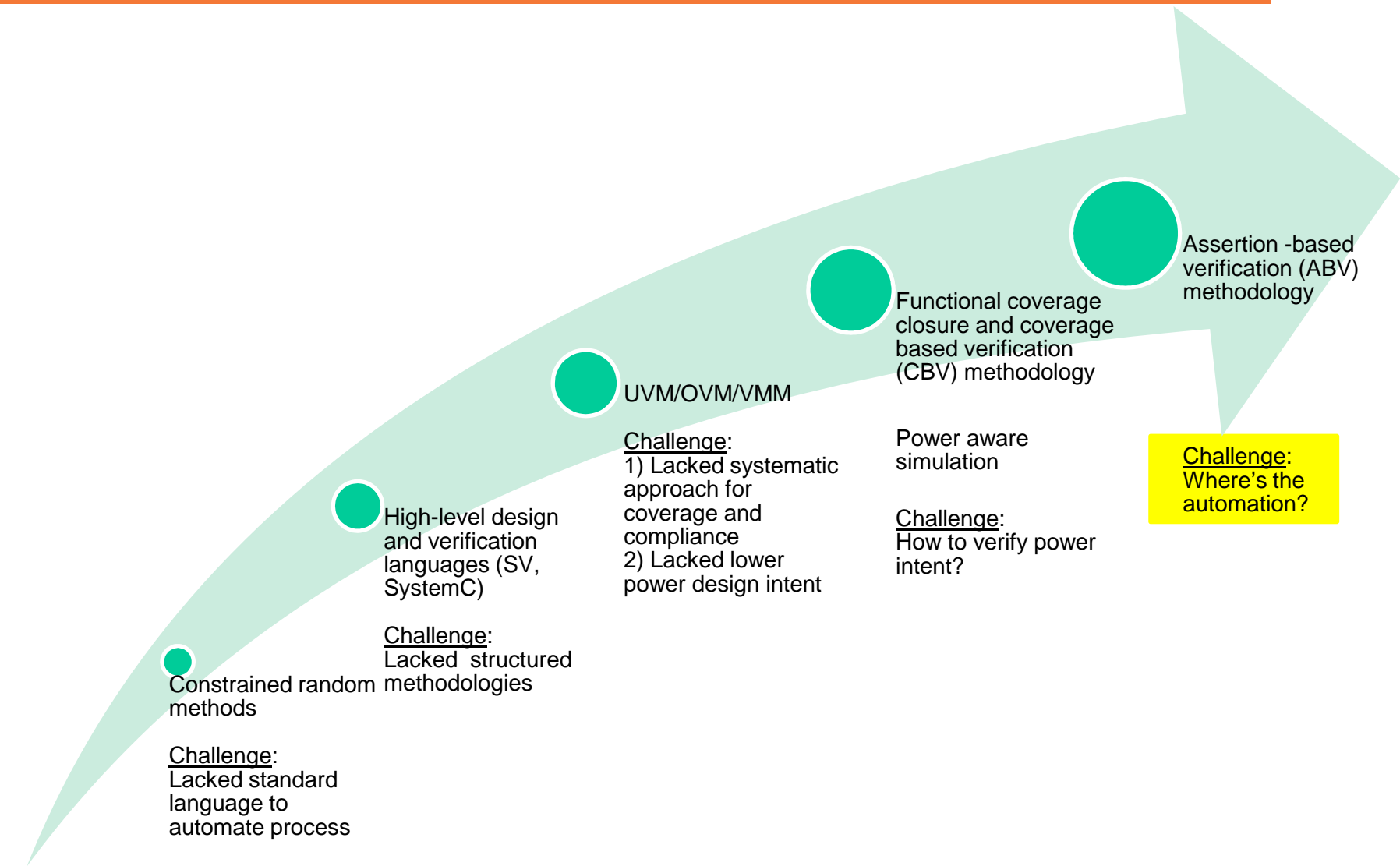
- 2つの設計手法について議論します
 - Functional verification (機能検証)
 - DFT
- フォーマル解析の新たな技術が、どのようにこれらの手法を進歩させているか検討します

Insight

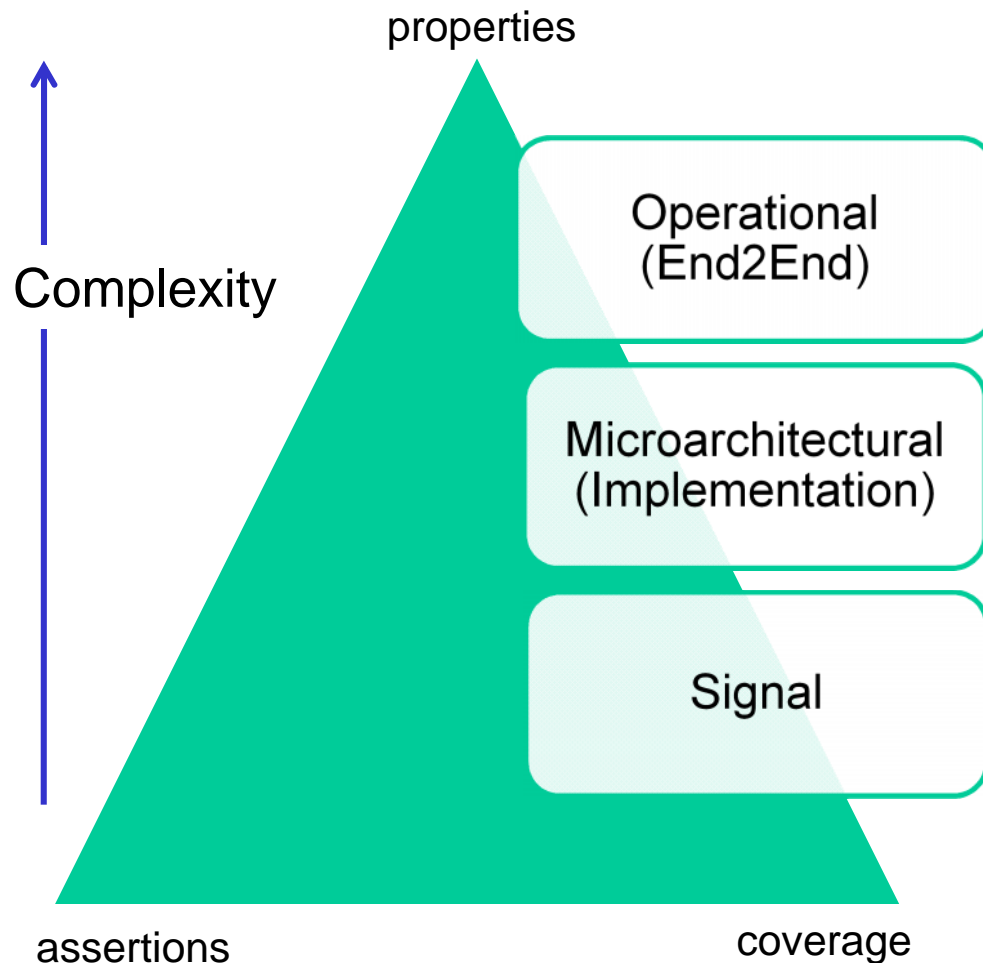
- 1st ビヘイビア・レベル
フォーマル解析ツール
- シミュレーション中心
- 早期RTLフォーマル・アプリケーションをターゲット
 - Assertion synthesis
 - X-Verification including low power analysis
 - Path delay fault estimation and coverage closure
 - Reachability analysis



機能検証の進化



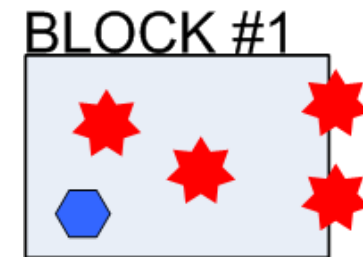
ABV と CBVのチャレンジ(1)



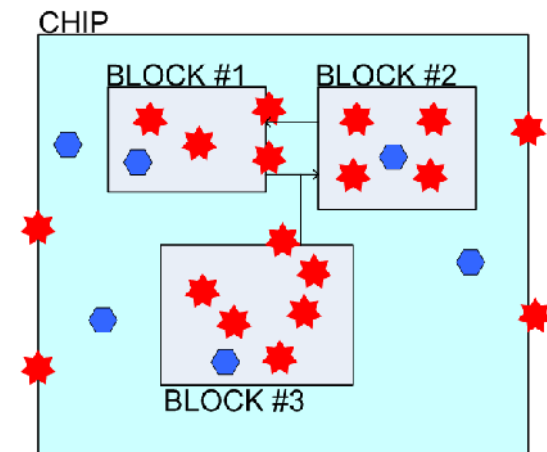
- オペレーションのプロパティは、エンドツーエンドのシーケンスをモデル化
- マイクロ・アーキテクチャのプロパティは、インプレメンテーションでのハードウェアのインターアクションを検証
- 信号レベル・プロパティは、設計の不変部分を表記

ABV と CBVのチャレンジ(2)

- すべてのオペレーションとマイクロ・アーキテクチャ要求を持つ検証プランの方針作成
- インプレメンテーションを理解する為の適切な設計資料の不足
- アサーションとカバレッジ・モデルに対して十分なプロパティ記述
- SVA構築の複雑さ
- シミュレーションとフォーマル解析ツールでのプロパティのデバッグ



Formal analysis



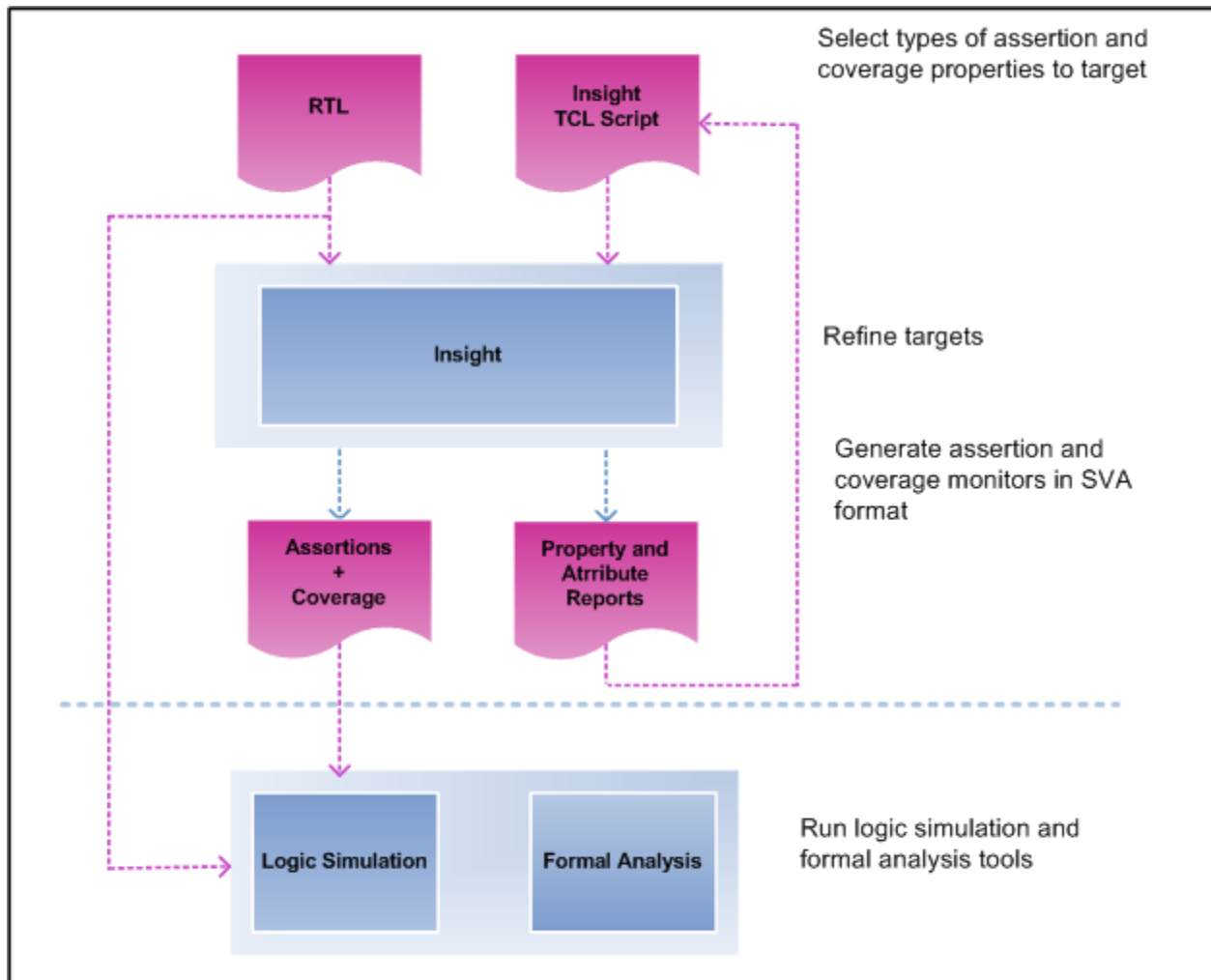
Formal analysis +
Logic simulation



Insight はABV と CBVを可能にする

- アサーションとカバレッジ・プロパティ合成
 - アサーションとカバレッジ・プロパティを自動生成
 - テストベンチ不要
 - 機能認識に対してフォーマル・シンボリック解析を活用
- ABV とCBVの自動化と手法を進化
- マニュアル操作のアサーションとカバレッジの補強
- チップレベル・シミュレーションとフォーマル解析の拡張

アサーション合成フロー



マイクロ・アーキテクチャ機能認識

- フォーマル・シンボリック解析は、各種機能を認識
 - FSM, arbiter, queue, memory, counter, internal bus, clock domain crossings, key control functions, and power controller
- ソースコード・テンプレート・マッチング手法よりも強力
- TCLスクリプトが、高レベルのクエリとフィルター操作を提供

```
set fsms [insight_get_objects -type fsm]
insight_asyn_script_gen $fsms -file fsm_properties.tcl

fsm_properties.tcl:
insight_asyn_property_gen tb.f1.fifo_state -fsm_state -fsm_transition
insight_asyn_property_gen tb.r2.state_1 -fsm_state -fsm_transition
insight_asyn_property_gen tb.r1.state_0 -fsm_state -fsm_transition
```

マイクロ・アーキテクチャ アサーションとカバレッジ(1)

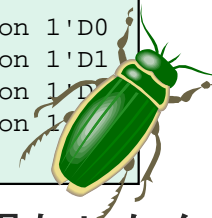
■ FSMアサーションとカバレッジ

- FSMビヘイビアとのインタラクションをターゲット
- FSMデッドロックとライブロックのアサーション

```
property avy_prop_tb_fsm_state_37;  
  @(posedge tb.clk) (tb.fsm.state == 3'D2) |->  
    (tb.fsm.state == 3'D2) [*0:7] ##1 (tb.fsm.state != 3'D2);  
endproperty  
avy_property_tb_fsm_state_36: assert property(avy_prop_tb_fsm_state_37)  
  else $warning("AVERY_ASSERT: FSM timeout detected");
```

- FSMステートとステート・トランジション カバレッジモニター
- 現在の無効ステートと次のステートのFSMステート割り付けをレポート

```
FSM: tran1.fsm  
Transition from State2 to State3 has constant condition 1'D0  
Transition from State2 to State2 has constant condition 1'D1  
Transition from State3 to State0 has constant condition 1'D1  
Transition from State3 to State3 has constant condition 1'D1
```



- 2つ又はそれより多くのFMS間のクロス信号に参与したタイムアウト・アサーションの要求/許可

マイクロ・アーキテクチャ アサーションとカバレッジ(2)

- キュー・アサーションとカバレッジ
 - FIFO、スタックのサポート
 - キュー・リーク(アサーション)
 - Outputs dropped and never used
 - オーバーフローとアンダーフロー(アサーション)
 - # empty/full と # almost empty/full (カバレッジ)
 - # of passthroughs, # cycles between queue ops (カバレッジ)

- 例: FIFO 出カリーク アサーション

```
set fifos [insight_get_objects -type fifo]
set fifo_list [split $fifos " "]
foreach fifo $fifo_list {
    insight_asyn_property_gen $fifo -reg_leak }
```

マイクロ・アーキテクチャ アサーションとカバレッジ(3)

- アービター・アサーションとカバレッジ
 - タイムアウト期間内で、どの要求に対しての許可かチェック
 - アービターの公正さのチェック、マスタが同じ数の許可が得られることを確認
 - マスタ(req→grant)のレイテンシのチェック
 - 一度に一つのみ許可をアクティブさせているかチェック
 - 要求が無ければ、許可をアサートしていないかチェック
- 要求/許可のレイテンシ・カバー・プロパティの例

```
assign req_local4[2:1] = {tb.r1.req0, tb.r2.req1};
assign gnt_local4[2:1] = {tb.arbiter.gnt0, tb.arbiter.gnt1};
genvar latency4, m4;
generate
  for (latency4=1; latency4 <=8; latency4++) begin
    for (m4=1; m4<=2; m4++) begin
      c_gnt_o: cover property( @(posedge tb.clk) $rose(gnt_local4[m4]) |->
        ($past(req_local4[m4], latency4) == 1'b0));
    end
  end
endgenerate
```

マイクロ・アーキテクチャ アサーションとカバレッジ(4)

- Memory/Reg アクセス アサーション
 - X on memory address, write, or read data
 - memory read/write address bounds
 - multiple assignment in same clock cycle or different clock edges
- 条件(if/case) アサーションとカバー・モニター
 - X on if/case condition
 - parallel case with more than one case active
 - priority case with only one case active (can be made parallel case)
 - case without default, which has none of the branches active
 - full case to check whether default is indeed not needed
- Tri-Stateでのバス・コンテンション・アサーション
 - one driver active
 - multiple drivers active but same data value

マイクロ・アーキテクチャ アサーションとカバレッジ(5)

- Power aware アサーションとカバレッジ
 - リセット後、アイソレーション・レジスタでの“X”は無い
 - リテンションsave / restore 信号の分割を除外
 - パワー信号の正しいシーケンス
 - パワーステート・テーブルからパワーステート・コンビネーション

- UPF が必要

マイクロ・アーキテクチャ アトリビュート・クエリ

- クエリ機能が詳細設計アトリビュートをレポート
 - FSM states, next state transition variables
 - Cross FSM dependency analysis

```
insight_query_object -type fsm -info 2
```

```
===== FSM Analysis Report =====  
Variable: cpu8080.state  
Type: State register (FSM)  
S1, Reset, Signal wait, -> S1 (reset)  
    -> S2 (reset)  
S2-> S1 (reset)  
    -> S3 (reset)  
S3, Signal wait, -> S1 (reset)  
    -> S4 (reset, waitr)  
    -> S3 (reset, waitr)  
S4, Signal wait, -> S4 (reset, opcode)  
    -> S1 (reset, opcode, sign, parity,  
        carry, zero)  
    -> S14 (reset, opcode)  
    -> S18 (reset, opcode, sign, parity,  
        carry, zero)  
    -> S5 (reset, opcode)  
    -> S6 (reset, opcode)  
    -> S7 (reset, opcode)  
    -> S34 (reset, opcode)  
S5, Signal wait, -> S1 (reset, intr, ei)  
    -> S5 (reset, intr, ei)  
=====
```

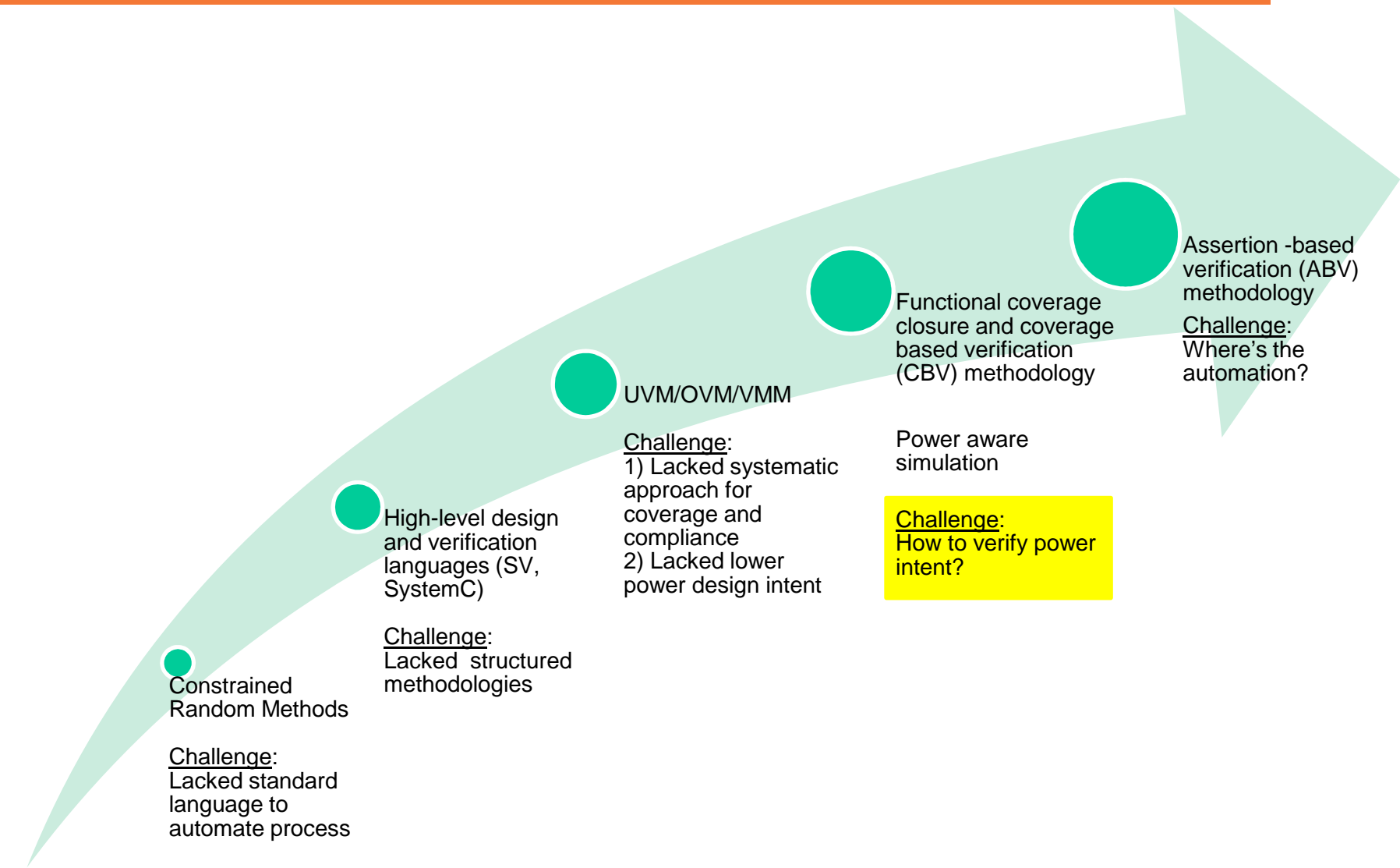
```
insight_fsm_dependency_analysis  
[insight_get_objects -type fsm]
```

```
===== FSM Dependency Analysis Report =====  
Variable fsms_testbench.dut.fsm1 is affected by  
the following variables at time 675, cycle 6:  
  
Module inputs:  
    fsms_testbench.a (seq_depth= 1)  
    fsms_testbench.b (seq_depth= 1)  
  
Other registers:  
    fsms_testbench.dut.fsm2 (seq_depth= 1)  
    fsms_testbench.dut.fsm3 (seq_depth= 2)  
    fsms_testbench.dut.temp (seq_depth= 3)  
===== End of FSM Dependency Analysis Report =====
```

アサーション合成事例

	Design A	Design B
FSM	48 recognized 104 state cover properties 364 state transition cover properties 103 deadlock assert properties	112 recognized 188 state cover properties 4953 state transition cover properties 187 deadlock assert properties
Counter	85 recognized 79 wrap cover properties	179 recognized 133 wrap cover properties
Memory	62 memories 310 assert properties	137 memories 573 assert properties
FIFO	21 recognized 105 assert properties 147 cover properties	11 recognized 55 assert properties 77 cover properties
Arbiter	0 recognized	2 recognized 6 assert properties 32 cover properties
Lines of RTL	39,000	38,000
# of flops	17,899	50,000
Size (gates)	750 K	3.0 M
Runtime	1 min	22 min

機能検証の進化



ローパワー検証のチャレンジ

- パワーシーケンス・プロトコルの検証とその設計が完全に機能し、決定論的で且つデータに影響を与えない
 - Show no X's propagate from powered down blocks to powered blocks during all power state transitions
 - No built-in retention and isolation assertion violations
 - Adheres to built-in and user properties of power sequencing rules
- 論理シミュレーションで継承されているX-Handling問題の意味は、幾つかのバグが、RTLとゲートレベル・シミュレーションとのミスマッチまで、又はハードウェア・プロトタイプまで発見できないことです。
 - X-optimism: 0/1 values are propagated instead of a real unknown
 - X-pessimism: Xs are propagated even though real value is 0/1

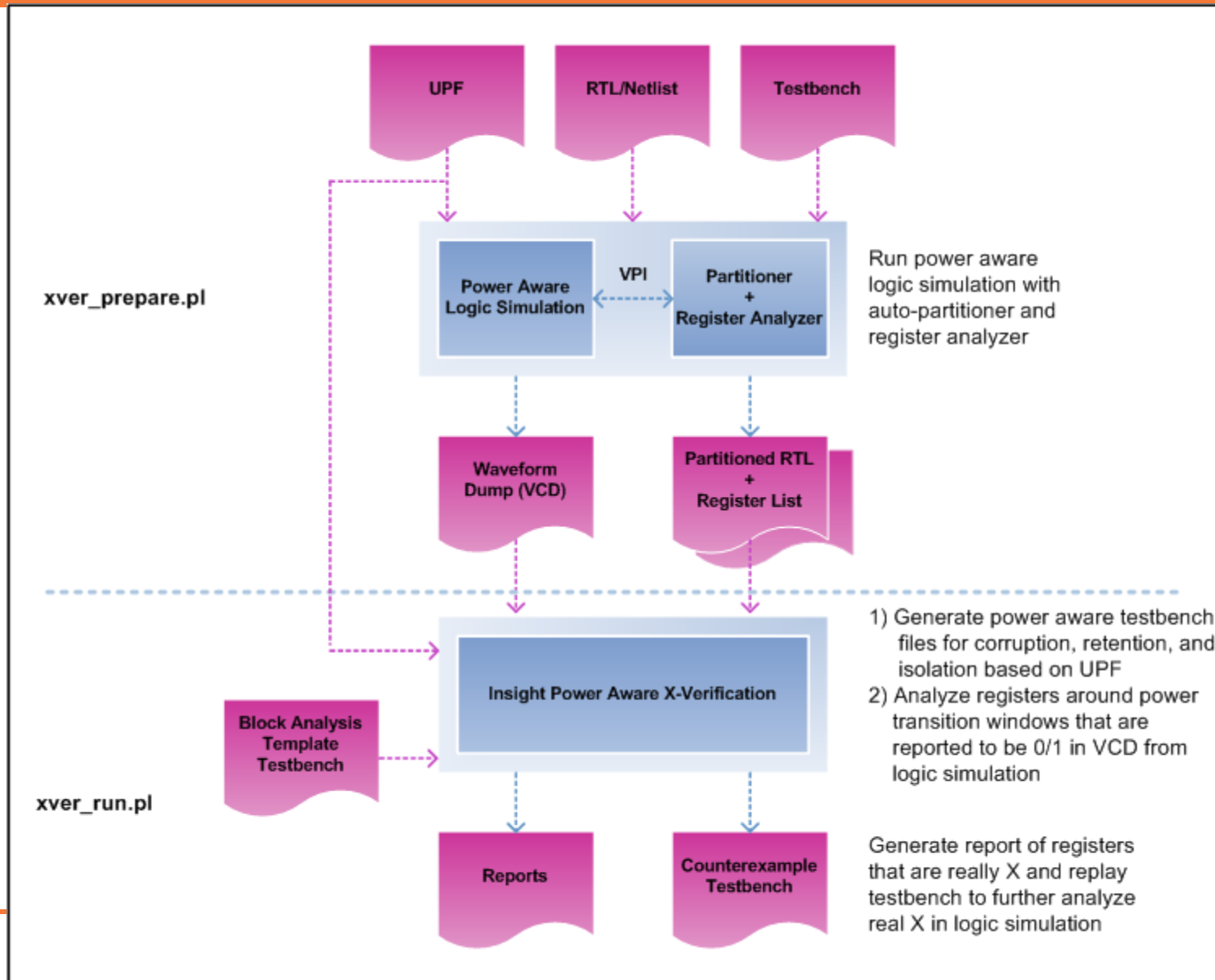
```
X-optimism
a = 1'bx;
if (a) out = b;
else out = c;
result: out = c;
```

```
X-pessimism
a = 1'bx; b = 1'b1; c = 1'b1;
out = ( a & b ) | ( ~a & c );
result: out = 1'bx;
```

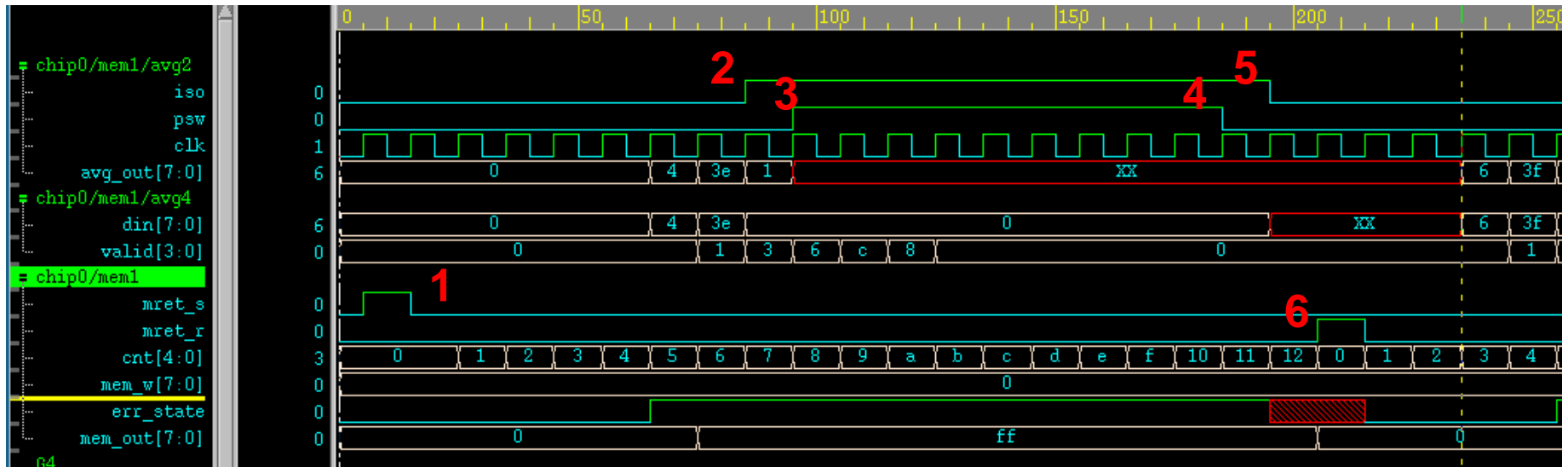
Insight はローパワー検証を可能にする

- パワーステートの変化時に、X-伝搬に対して解析する為に、X-検証は、正確なハードウェア・セマンティックを使用
 - Unknown (X) represents both 0 and 1 (non-determinism)
 - Could be entirely missed by logic simulation
 - Provides counterexamples to debug each X condition
 - Represents possible retention, reset, and isolation bugs
- パワー設計目的に対してはUPF 2.0
- チップレベルRTLを通してブロック上で実行
- テストベンチ又はパワー・トランジション・サイクルの波形が必要
- パワー・コントロール信号シーケンス違反を検出する為に、アサーションとカバレッジ合成を連結して使用

ローパワーに対しての“X Optimism検証フロー”

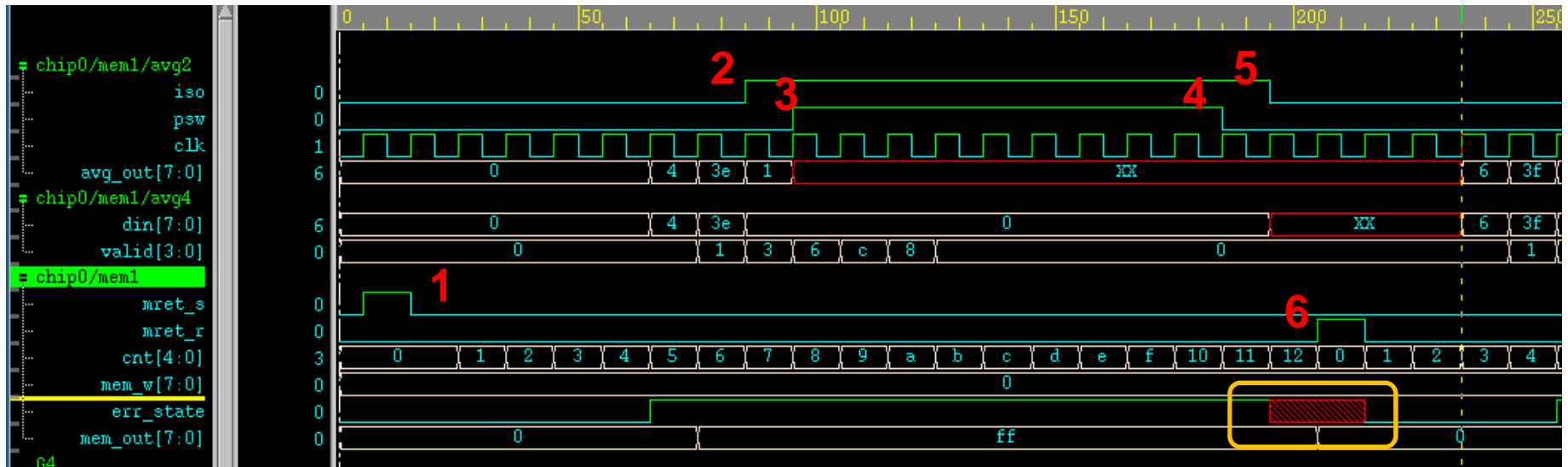


典型的なパワー・トランジション・シーケンス



- Events for low power simulation
 - 1: retention (save)
 - 2: isolation (enable)
 - 3: power off
 - 4: power on
 - 5: isolation (disable)
 - 6: retention restore

リテンション・バグ



- Retention (restore) is asserted 2 cycles after m1_avg2_pd is powered on
- The value of chip0.mem1.cnt exceed threshold, causing chip0.mem1.err_state becomes X
 - X in chip0.mem1.err_state is propagated out to chip0.mem1.mem_out

X-検証結果

- リテンション・バグに対してのローパワーX-検証レポート

205 [INFO] PERFORM X-VERIFICATION

Symbolic X values propagate to ins_xver_tb_mem_block.DUV.mem_out (level= 2) due to the following variables and values:

ins_xver_tb_mem_block.DUV.err_state_i195 at mem_block.v: 86: value1= 1'b1, value2= 1'b0

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.apsw (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.aiso (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.mret_s (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.mret_r (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.cnt (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.avg4.valid (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.avg4.d4 (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.avg4.d3 (level= 0)

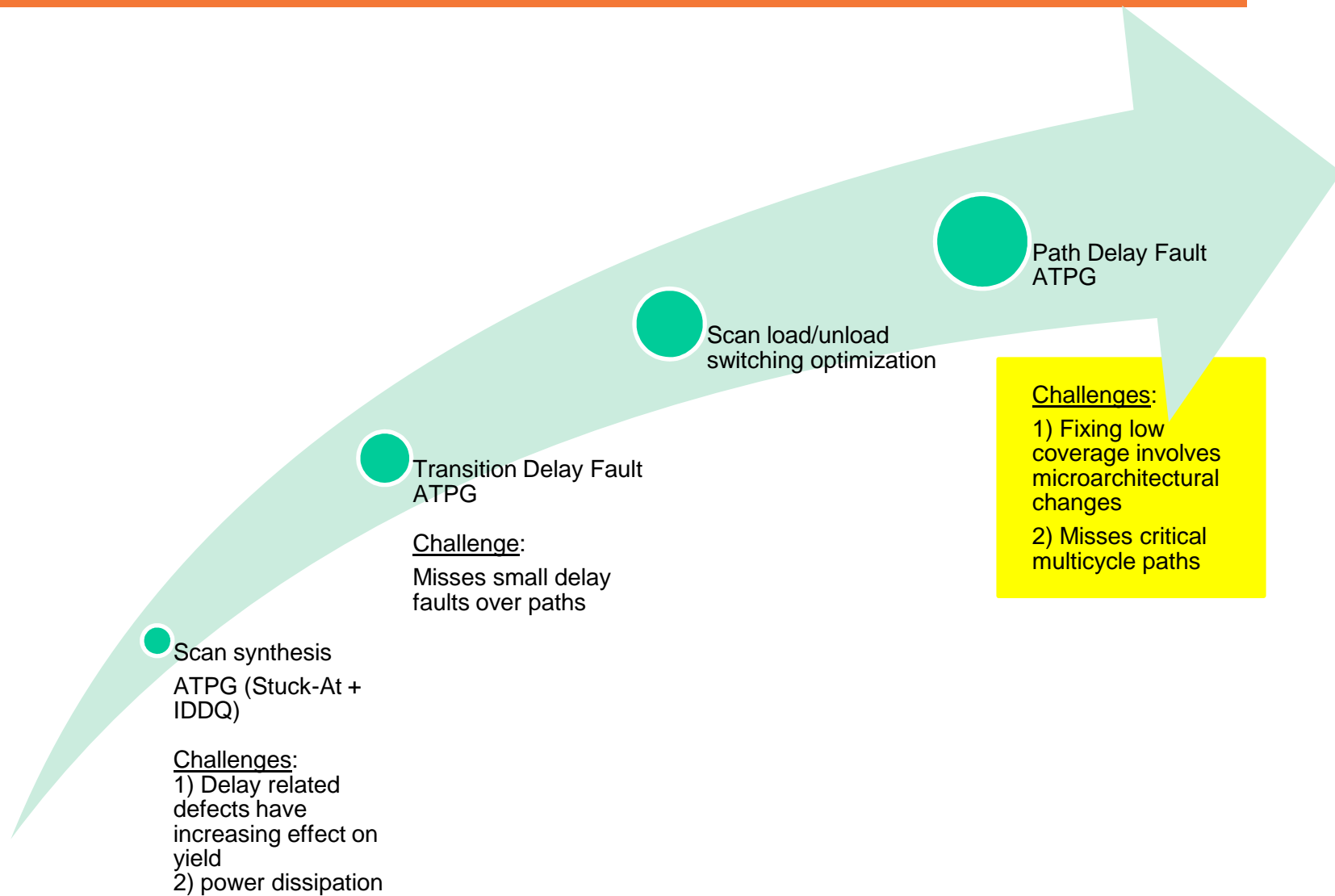
Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.avg4.d2 (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.avg4.d1 (level= 0)

Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.avg4.avg_out (level= 0)

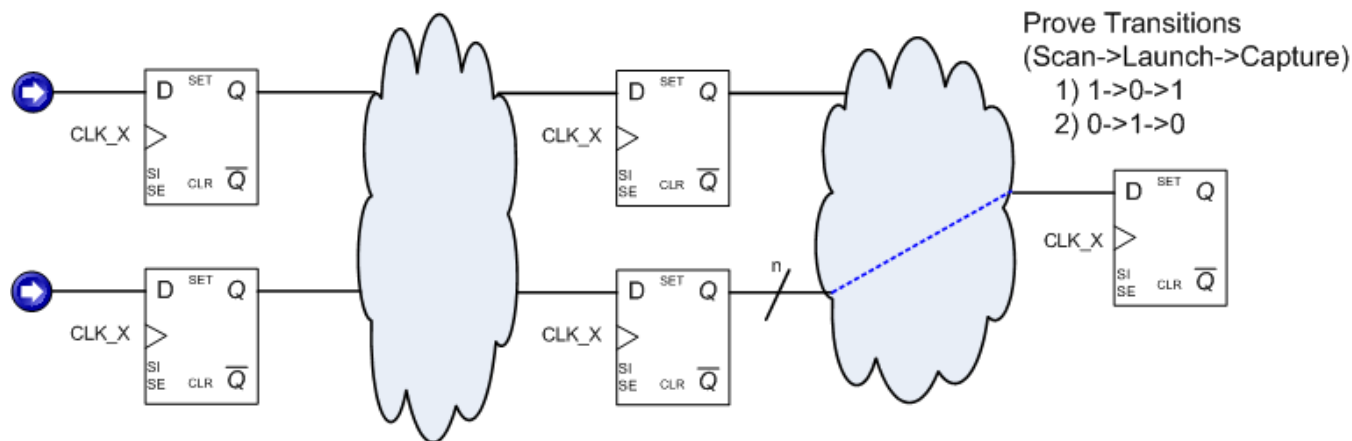
Symbolic X values do not propagate to ins_xver_tb_mem_block.DUV.avg2.d1 (level= 0)

Scan-Based DFTの進化



At-Speed (高速) DFTのチャレンジ

- トランジションとパス遅延フォルト・カバレッジは、Stuck-at カバレッジより通常10%低い
- カバレッジの改善は、マイクロ・アーキテクチャ変更でテストモードのコントロール性と観察度を追加することに関与
- 修正を確認する為に、高コストの設計繰り返しを引き起こす
 - RTL change, re-simulate, synthesize, STA, and ATPG
- 大きな労力を要する

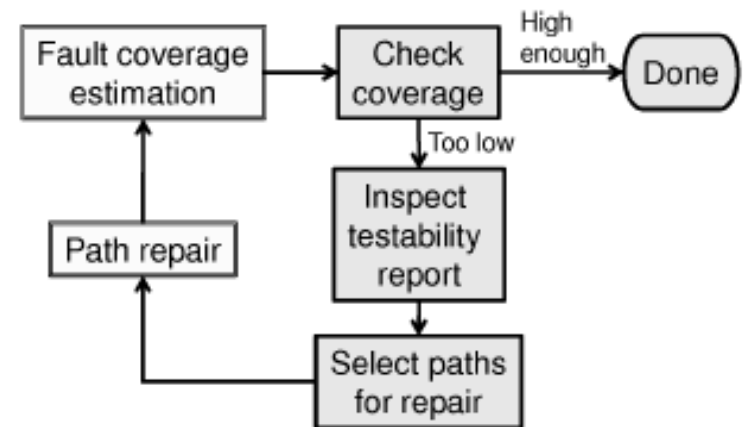


Insight はAt-Speed DFTを可能にする

- より少ない時間でAt-Speedパス遅延カバレッジを改善
- RTL設計フェーズでのテストビリティ課題を見つける
- 1stパスATPGカバレッジ結果の改善
- 最大限のカバレッジ改善を得る為に、修正の発見とランク付けする為にテスト修正解析を使用

Insight At-Speed解析フロー

- スキャン設計目的を確認
 - Scan registers, I/O mode, X-generators, memory bypass modes, multicycle paths
- スキャン・データ安定性解析を実行
 - 実行中及びサイクルを収集中に、安定性を保持できないワードレベルのレジスタ、ファンアウトのサイズによってランク付け



At-Speed 解析フロー

- 初期のカバレッジ見積もりに対してテスト容易性解析を実行
 - Select small % of critical paths to analyze
 - Critical paths automatically generated by estimating complexity of logic path
 - Causes of untestability identified in report
 - Supports distributed processing to cut run time
- 修正解析実行
 - Generates repair report with ranking for candidates
- ユーザがRTL修正変更する
 - Select N repairs from repair analysis
- 新たなカバレッジを再確認する為に、テスト性解析の再実行

Insight At-Speed 解析

- 完全テスト容易性レポートの例

```
=== Begin of At-Speed Path Report ===
top.a => top.a_1 (AIG nodes = 234) : 010 (UP), 101 (UP)
top.c => top.e (AIG nodes = 154) : 010 (UP), 101 (UP)
top.a_1 => top.d (AIG nodes = 112) : 010 (T), 101 (UP) UC
top.b => top.d (AIG nodes = 80) : 010 (UP), 101 (UP) UC
Number of paths: 4, testable: 1, coverage: 25.00%
=== End of At-Speed Path Report ===
```

Ranked by critical
path complexity

Insight At-Speed 修正

- レポート例

Shows non-launch
register control
problem on
pdata_valid or wrptr

```
=== Begin Repair for Trace "dut.up.a[3] => dut.up.tr.tp.c[1]: 01" ===  
(Each diagnosis is the register that needs to have its launch value changed.  
The value is also returned.)  
Try changing only 1 register(s)  
Diagnosis 1: dft_testbench.dut.pldata_valid,  
    Suggested values to solve problem:  
    Variable dft_testbench.dut.pldata_valid, at time 400, value= 1'b0  
Diagnosis 2: dft_testbench.dut.up.tr.wrptr,  
    Suggested values to solve problem:  
    Variable dft_testbench.dut.up.tr.wrptr, at time 400, value= 2'b10  
=== End of Repair ===
```

Either fix will make
path delay fault
testable

RTL 修正実施

```
reg stable_p1, stable_p2, transition, launch, capture;
reg test_mode_reg;
wire capture_dd;
assign capture_dd = ~stable_p2 & launch;
wire out = capture;

always @(posedge clock or reset)
    if (reset) begin
        stable_p1 <= 0;
        stable_p2 <= 0;
        transition <= 0;
        launch <= 0;
        capture <= 0;
    end
    else begin
        stable_p1 <= in_stable;
        stable_p2 <= test_mode_reg ? stable_p1 : transition;
        transition <= in_transition;
        launch <= transition;
        capture <= capture_dd;
    end
end
```

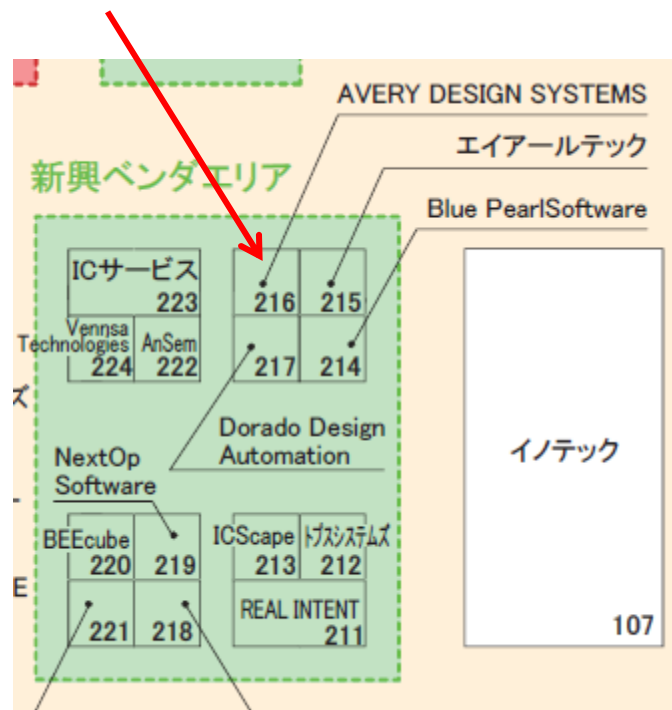
User adds test mode
logic for stable_p2

まとめ

- 早期RTLフォーマル手法は、検証とDFT手法を劇的に進歩させることが可能
 - アサーションとカバレッジ合成が、検証収束を自動化
 - X-検証は、リセットとローパワー・シーケンスを通して決定論的オペレーションを証明
 - At-speed テスト容易性解析は、テスト性カバレッジ課題を明らかにする
- この手法の展開は簡単
- 積極的な設計戦略
- 高コストの繰り返し除去
- 論理合成前に、RTL設計の品質を改善
- 全体スケジュールの正確性を改善

ご静聴ありがとうございました。

- より詳しい情報を希望され方は、
 - Avery Design Systemsブース:216



- 本社コンタクト先:
 - E-mail: cbrowy@avery-design.com
 - URL:<http://www.avery-design.com/>
- 日本営業担当
 - 横川秀美: yokokawa@chip-start.com