

# New Developments in Early RTL Formal Analysis

Chris Browy, Avery Design Systems: VP Marketing and Sales  
Hidemi Yokokawa, Chip-Start: Japan Sales Director

EDSFair 2011 Seminar, Thursday, 27 Jan, 12:30-1:15 PM



# Introduction

---

- SoC design faces continuous and ever increasing challenges imposed by
  - Process technology
  - Design complexity
  - Convergence of HW/SW
  - Time to market
- Design methodologies keep evolving to take on these challenges in more proactive, top-down approaches
  - Increase productivity and eliminate bottlenecks
  - Improve overall quality

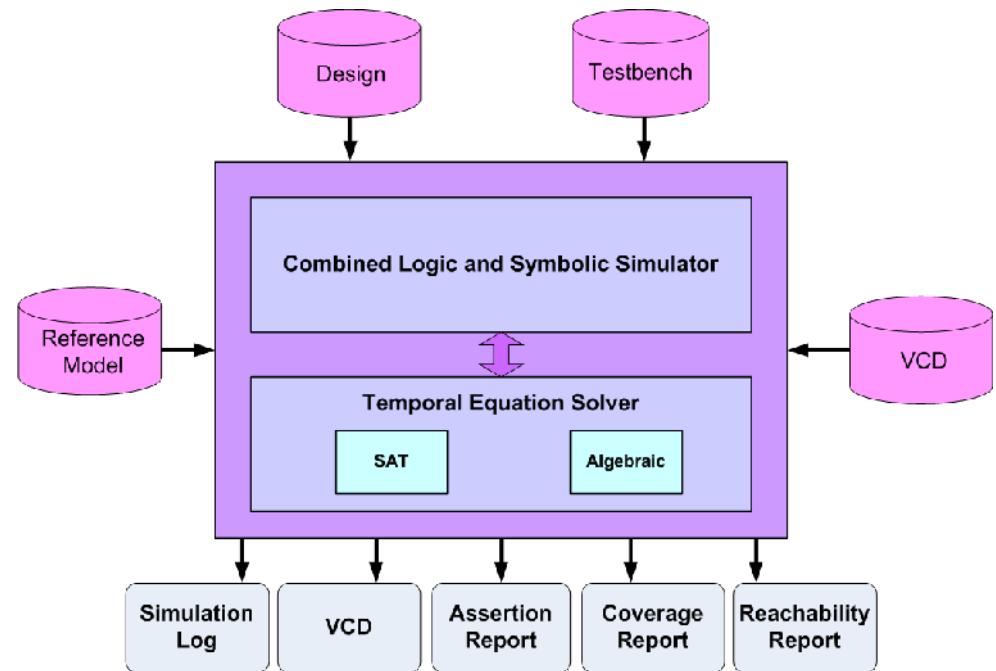
# Evolution of Methodologies

---

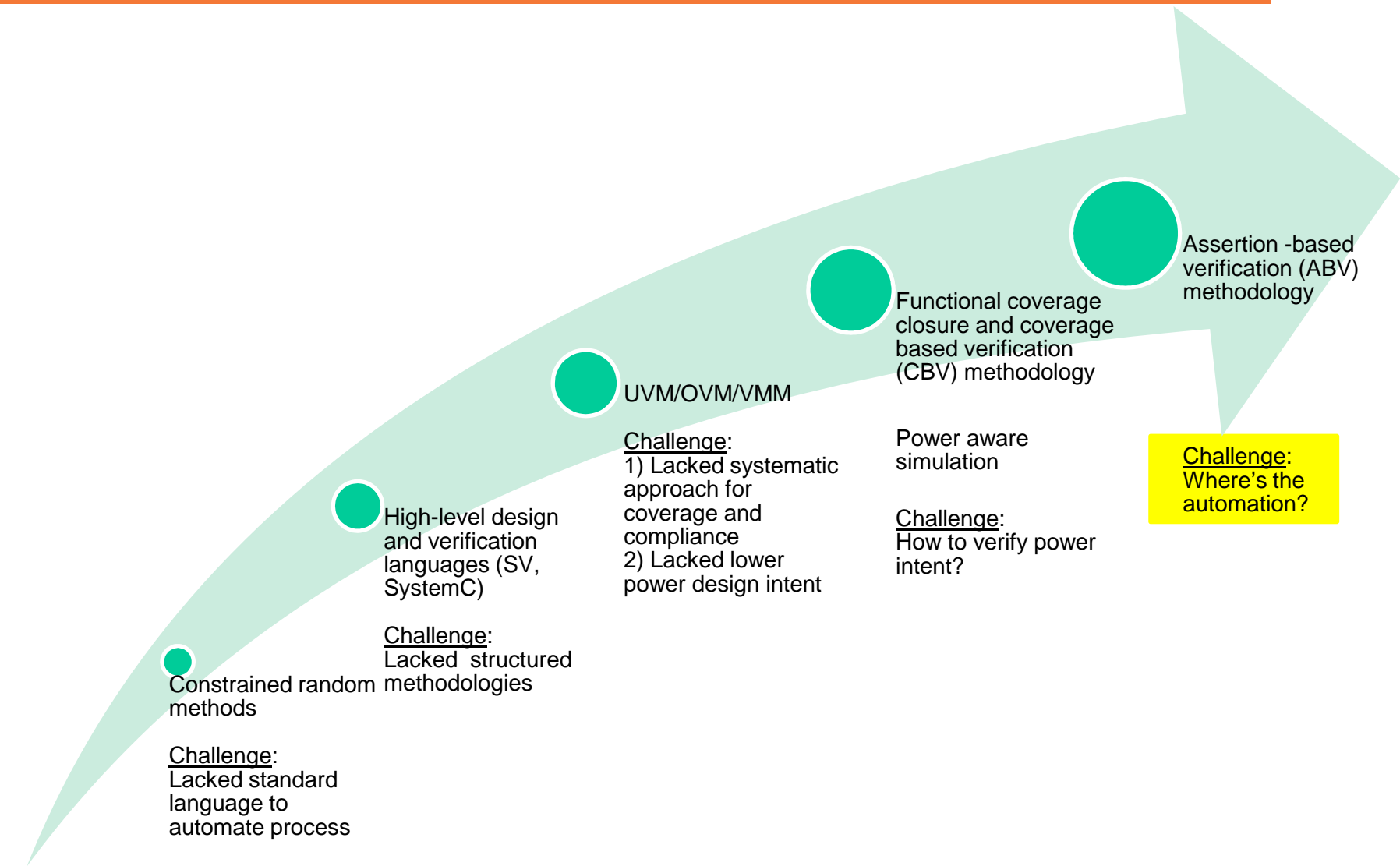
- Let's examine 2 design methodologies
  - Functional verification
  - DFT
- Consider how new techniques in formal analysis advance these methodologies

# Insight

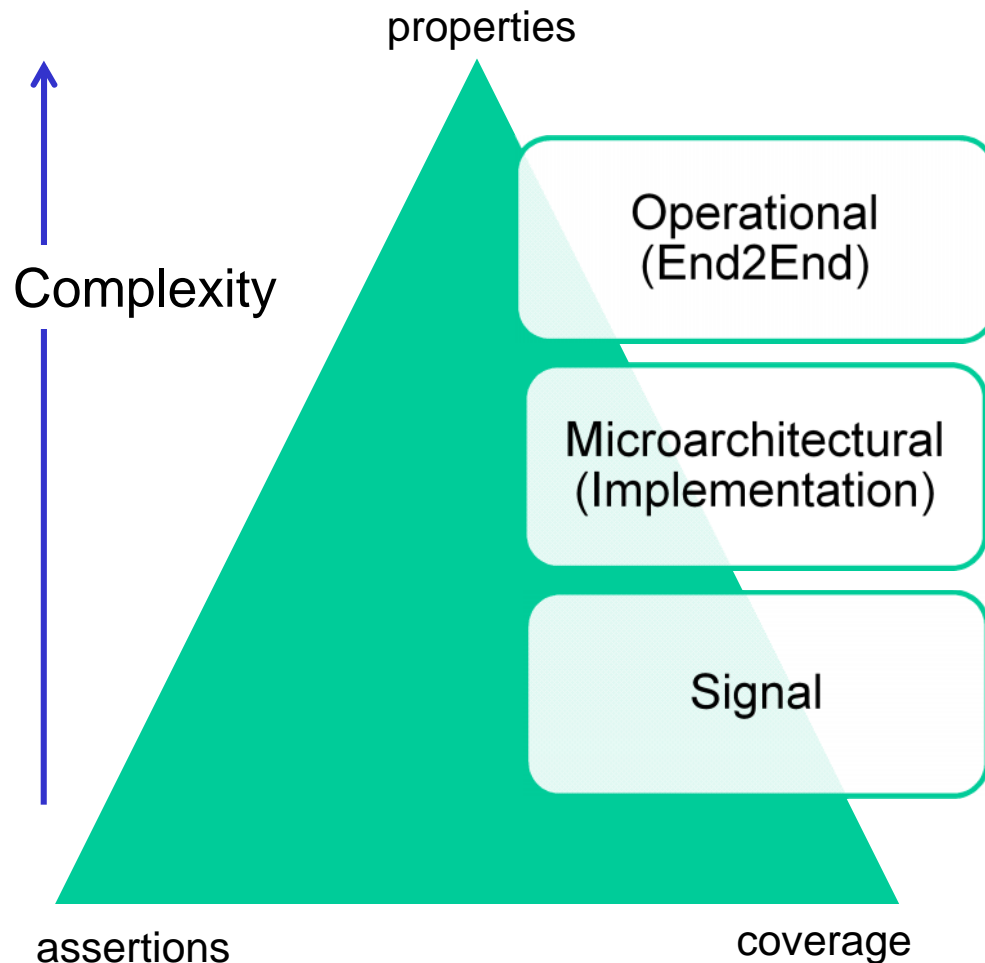
- 1<sup>st</sup> behavioral-level formal analysis tool
- Simulation-centric
- Targets early RTL formal applications
  - Assertion synthesis
  - X-Verification including low power analysis
  - Path delay fault estimation and coverage closure
  - Reachability analysis



# Evolution of Functional Verification



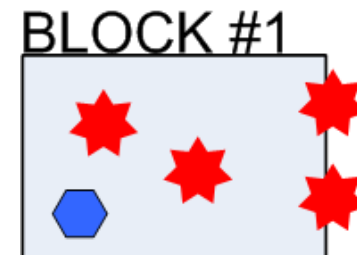
# Challenges of ABV and CBV



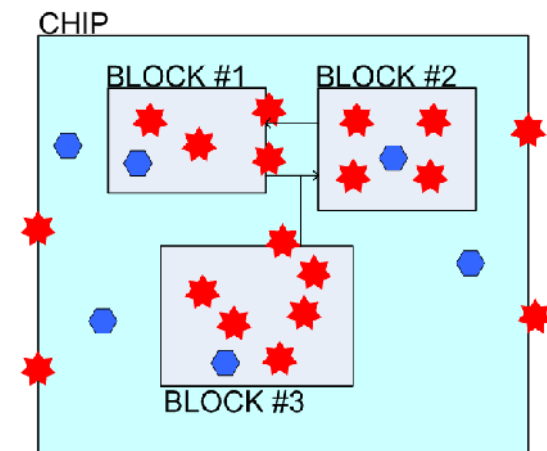
- Operational properties model End2End sequences
- Microarchitectural properties verify hardware interactions of implementation
- Signal level properties representing design invariants

# Challenges of ABV and CBV

- Formulating the verification plan with all operational and microarchitectural requirements
- Lack of good design documentation to understand implementation
- Writing enough properties for assertions and coverage models
- Complexity of SVA constructs
- Debugging properties in simulation and formal analysis tools



Formal analysis



Formal analysis +  
Logic simulation

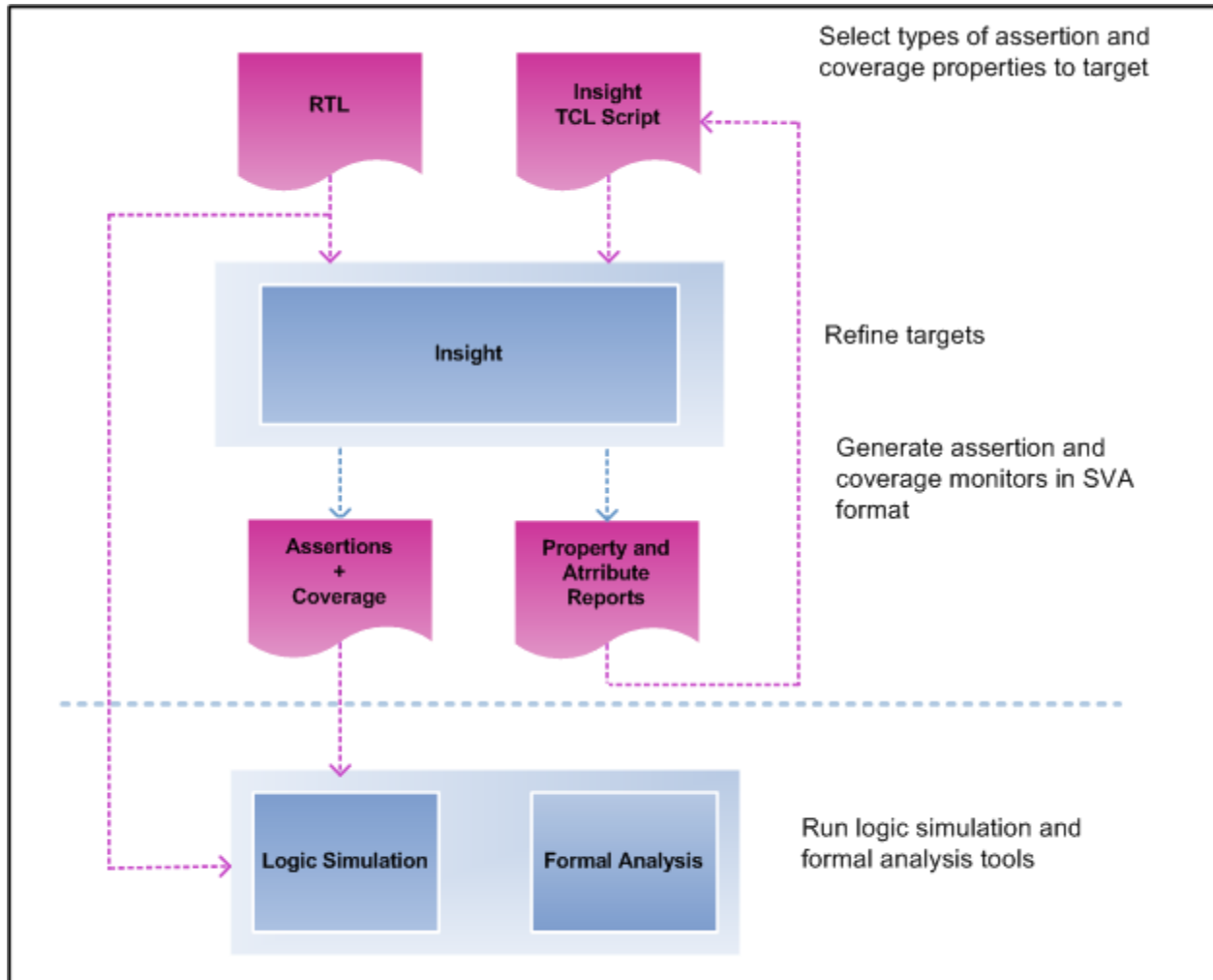


# Insight Enables ABV and CBV

---

- Assertion and coverage property synthesis
  - Generates assertions and coverage properties automatically
  - No testbench required
  - Utilizes formal symbolic analysis methods for feature recognition
- Advances automation and methodology of ABV and CBV
- Augments manual operational assertions and coverage
- Enhances chip-level simulation and formal analysis

# Assertion Synthesis Flow



# Microarchitecture Feature Recognition

- Formal symbolic analysis recognizes various features
  - FSM, arbiter, queue, memory, counter, internal bus, clock domain crossings, key control functions, and power controller
- More robust than source code template matching methods
- TCL scripts provide high-level query and filter operations

```
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
```

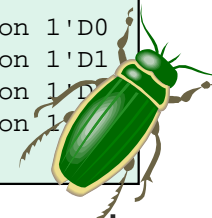
# Microarchitectural Assertion/Coverage

- FSM assertions and coverage
  - Targets interacting FSM behaviors
  - FSM deadlock and livelock assertions

```
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 state and state transition coverage monitors
- Reports invalid current state/next state FSM state assignments

```
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
```



- Request/grant timeout assertion involving cross signaling between 2 or more FSMs

# Microarchitectural Assertion/Coverage

- Queue assertions and coverage
  - Supports FIFOs, Stacks
  - Queue leaks (assertions)
    - Outputs dropped and never used
  - Overflow and underflow (assertions)
  - # empty/full and # almost empty/full (coverage)
  - # of passthroughs, # cycles between queue ops (coverage)
  
- Example: FIFO output leak assertion

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

# Microarchitectural Assertion/Coverage

- Arbiter assertions and coverage
  - Check grant for every request within timeout period
  - Check fairness of arbiter, make sure masters are getting equal number of grants
  - Check the latency of master req->grant
  - Check only one grant should be activated at a time
  - Check no grant should be asserted if there is no request
- Example of request/grant latency cover property

```
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
```

# Microarchitectural Assertion/Coverage

---

- Memory/Reg access assertions
  - X on memory address, write, or read data
  - memory read/write address bounds
  - multiple assignment in same clock cycle or different clock edges
- Conditions (if/case) assertions and cover monitors
  - 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 bus contention assertions
  - one driver active
  - multiple drivers active but same data value

# Microarchitectural Assertion/Coverage

---

- Power aware assertions and coverage
  - No X on isolation registers after its reset has been sequenced
  - Exclusion of separate retention save / restore signals
  - Proper sequence order of power signals
  - Power state combinations from power state table
- UPF required

# Microarchitecture Attribute Query

- Query feature reports detailed design attributes
  - 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)  
=====
```

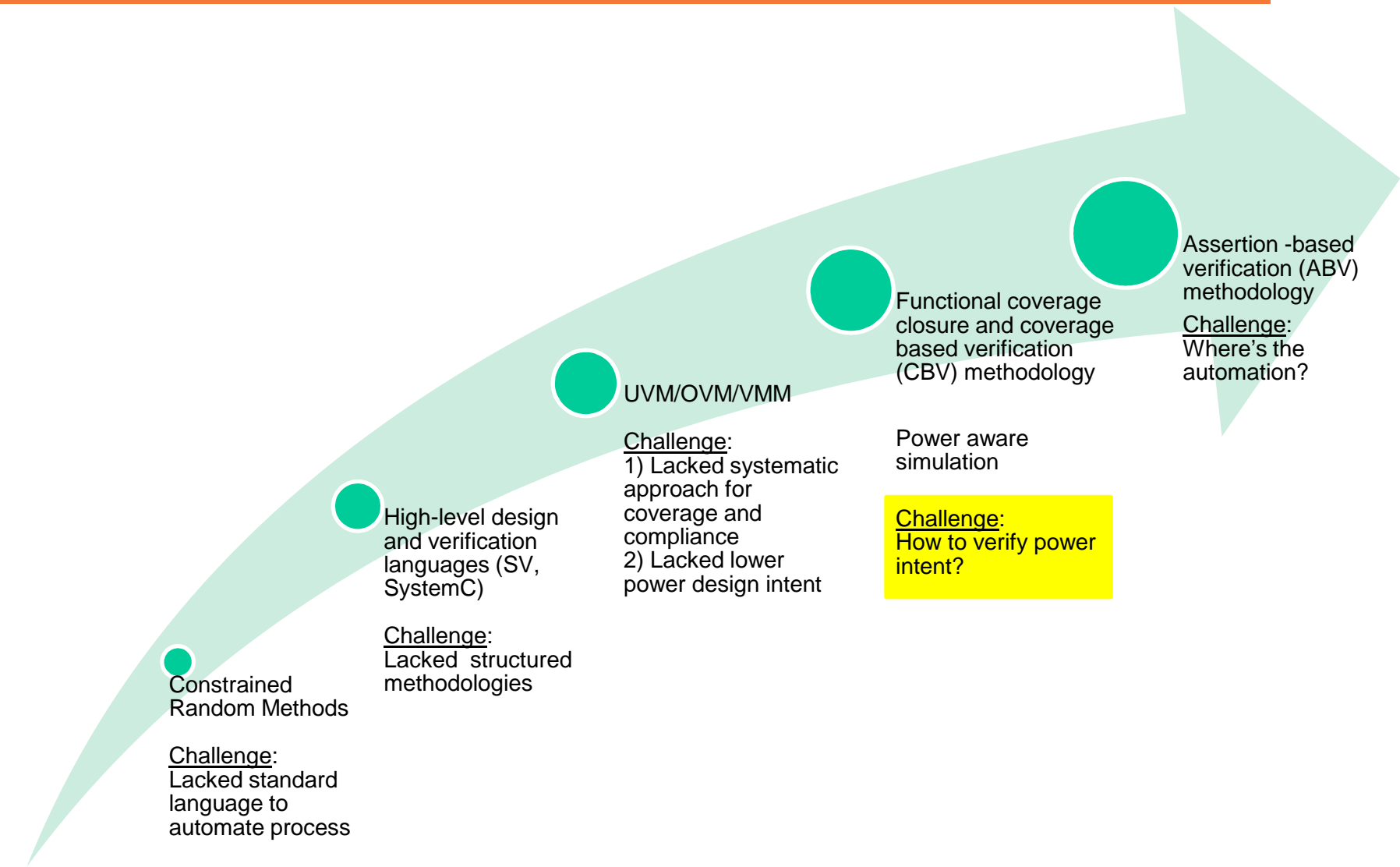
```
insight_fsm_dependency_analysis  
    [insight_get_objects -type fsm]
```

```
===== 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 =====
```

# Assertion Synthesis Case Studies

	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

# Evolution of Functional Verification



# Challenges of Low Power Verification

- Verify power sequence protocols and that design is fully operational, deterministic, and retains non-destructive data
  - 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
- Inherent X-handling problems in logic simulation mean some bugs may not be found until RTL vs Gate-Level simulation mismatch, or until hardware prototyping
  - 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;
```

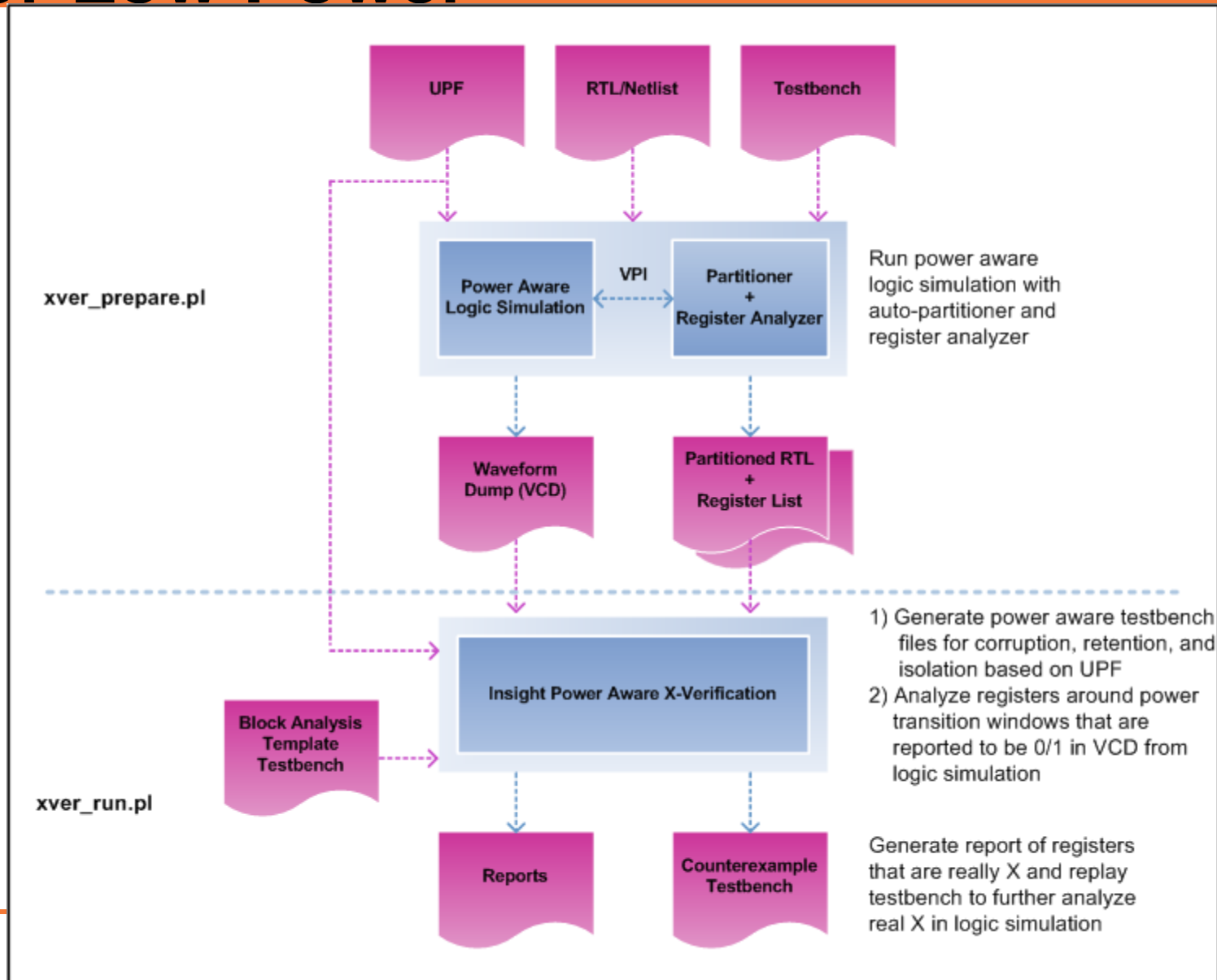
```
X-pessimism
a = 1'bx; b = 1'b1; c = 1'b1;
out = ( a & b ) | ( ~a & c );
result: out = 1'bx;
```

# Insight Enables Low Power Verification

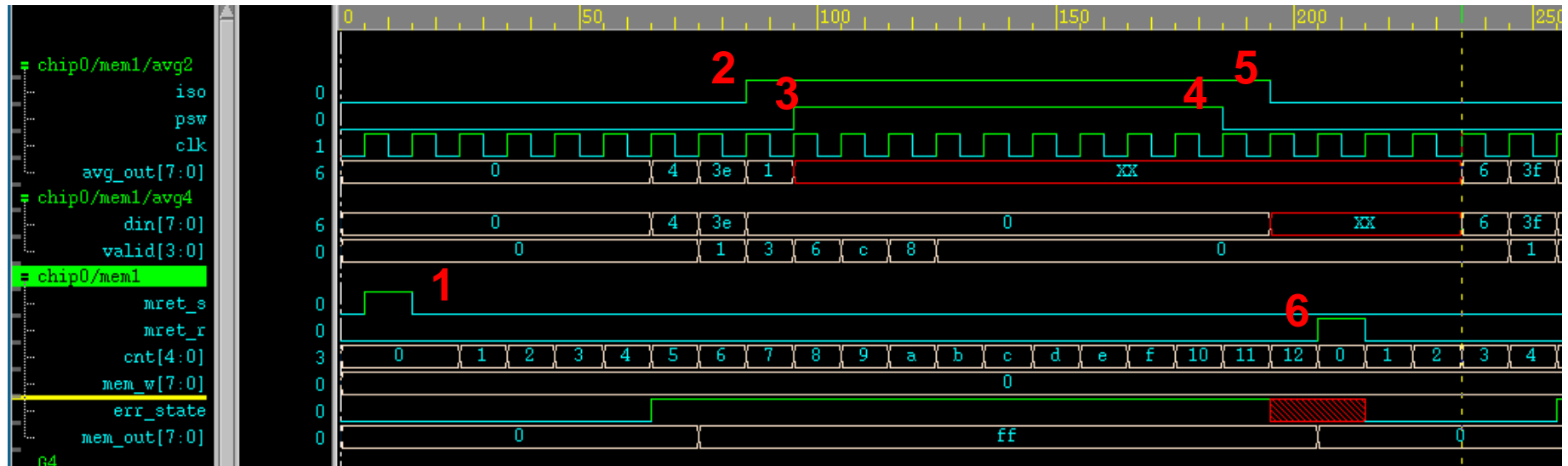
---

- X-verification uses accurate hardware semantics to analyze for X propagations during power state changes
  - 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
- UPF 2.0 for power design intent
- Run on block through chip-level RTL
- Requires testbench or waveform of power transition cycles
- Use in conjunction with assertion and coverage synthesis to detect power control signal sequence violations

# X Optimism Verification Flow for Low Power

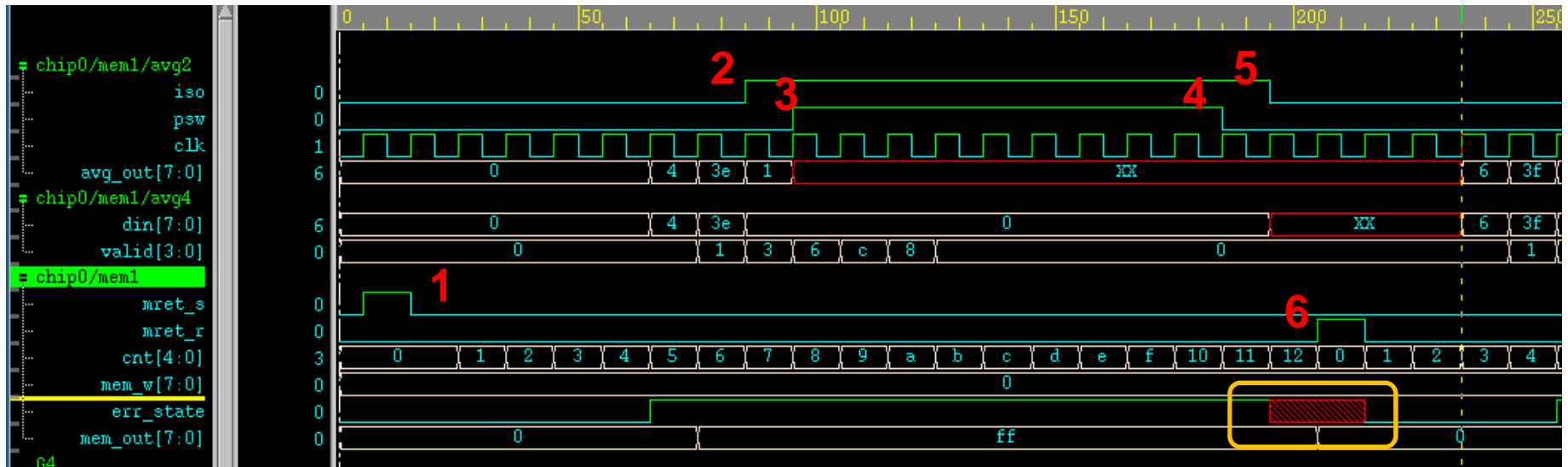


# Typical Power Transition Sequence



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

# Retention Bug



- 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-verification Results

- Low power X verification report for retention bug

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)

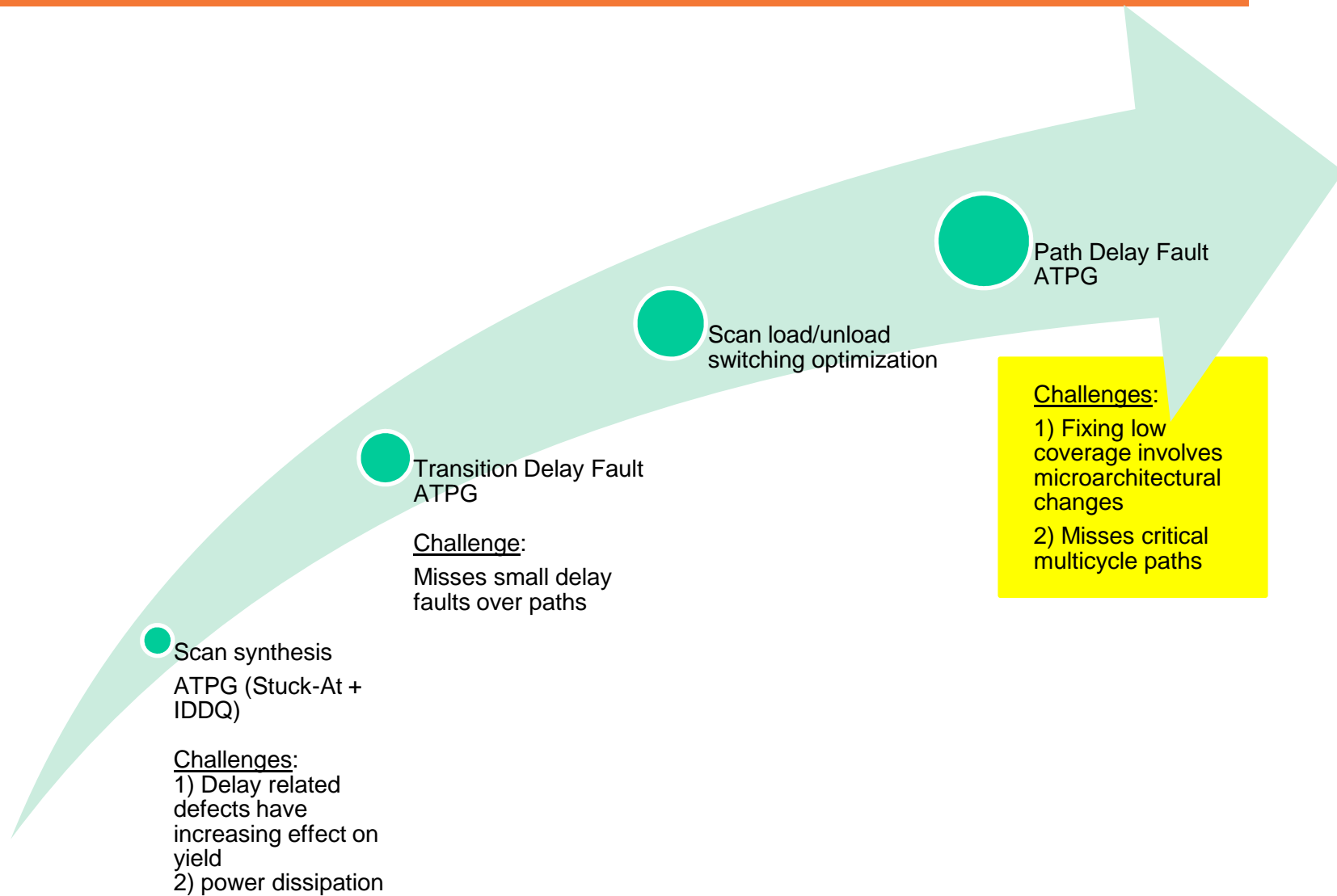
Symbolic X values do not propagate to `ins_xver_tb_mem_block.DUV.avg4.d2` (level= 0)

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)

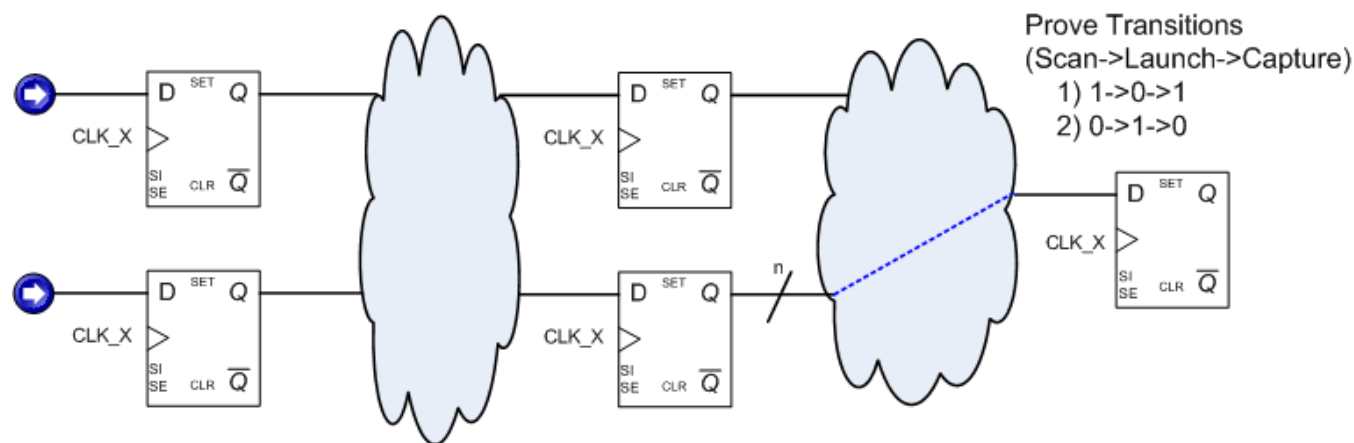
Symbolic X values do not propagate to `ins_xver_tb_mem_block.DUV.avg2.d1` (level= 0)

# Evolution of Scan-Based DFT



# Challenges of At-Speed DFT

- Transition and path delay fault coverage typically >10% lower than stuck-at coverage
- Improving coverage involves microarchitectural changes to add test mode controllability and observability
- Involves costly design iteration steps to confirm fixes
  - RTL change, re-simulate, synthesize, STA, and ATPG
- Labor intensive



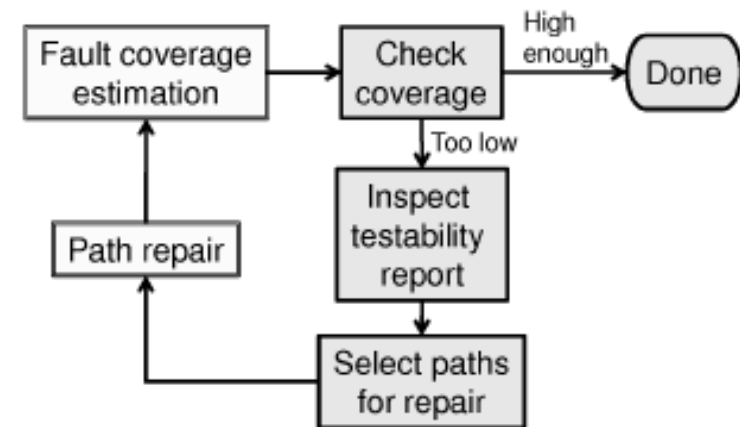
# Insight Enables At-Speed DFT

---

- Improve at-speed path delay coverage in less time
- Find testability issues earlier at RTL design phase
- Improves 1<sup>st</sup> pass ATPG coverage results
- Use test repair analysis to find and rank repairs for greatest coverage improvement

# Insight At-Speed Analysis Flow

- Identify scan design intent
  - Scan registers, I/O mode, X-generators, memory bypass modes, multicycle paths
- Perform scan register data stability analysis
  - Word level registers that can't be held stable during launch and capture cycles, ranked by size of fanout



# At-Speed Analysis Flow

---

- Run testability analysis for initial coverage estimate
  - 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
- Run repair analysis
  - Generates repair report with ranking for candidates
- User implements repairs in RTL
  - Select N repairs from repair analysis
- Rerun testability analysis to confirm new coverage

# Insight At-Speed Analysis

- Example Full Testability Report

```
=== 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 Repair

- Example Report

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

# Implement RTL Repair

```
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

# Summary

---

- Early RTL formal methods can significantly advance verification and DFT methodologies
  - Assertion and coverage synthesis automates verification closure
  - X-Verification proves deterministic operation through reset and low power sequences
  - At-speed testability analysis uncovers testability coverage issues
- Low effort to deploy
- Proactive design strategy
- Eliminate costly iterations
- Improve quality of the RTL design prior to logic synthesis
- Improve overall schedule reliability