Bug Hunting For Deep Sequential Complex S/M Using Formal

A PCIE 4.0 Case Study

Nitin Mhaske

Applications Engineer, Sr. Staff

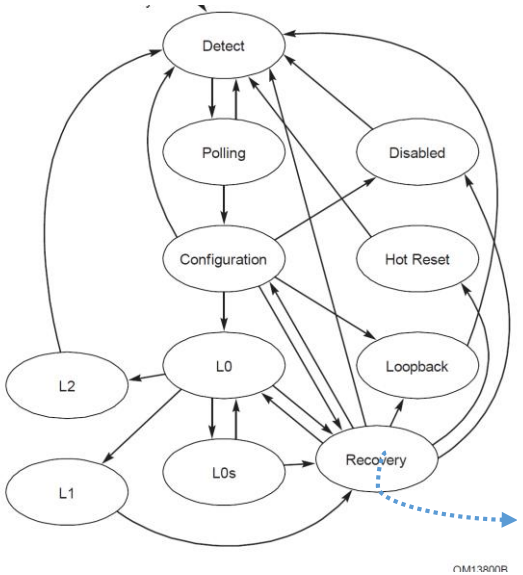
Synopsys Inc.

Iain Singleton

Applications Engineer, Sr

Synopsys Inc.

Challenges for Deep Sequential FSM Verification



PCIE/USB LTSSM
11 states; > 60 sub-states

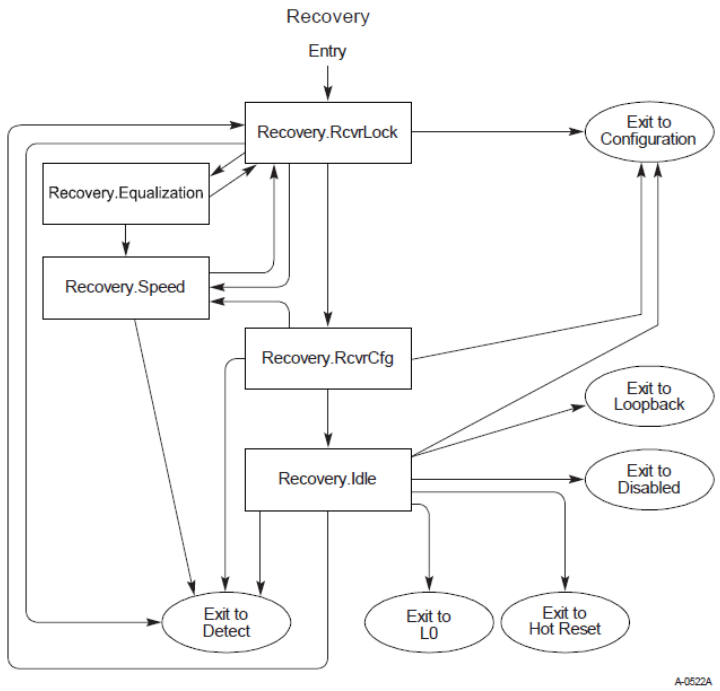


Figure 4-27: Recovery Substate Machine

- FSMs like Link Training and Status S/M (LTSSM) critical for link operations
- Many states that are sequentially deep w/ sub-states M/C
- Complex priority among simultaneously occurring transition conditions
- Recovery S/M with Complex Error handling, speed negotiation

Methodology For Bug Hunting

Planning/Scoping

- Selecting right design hierarchy: divide and conquer partitioning
- Define Scope (new, critical functionality) with challenging simulation coverage

Architecting Formal TB

- (Text Description) Plan intermediate micro-architecture and end-to-end assertions to develop
- Coverage properties to build for confidence, guiding formal

Execution Strategy

- Developing assumptions with constraint layering approach
- Developing assertions from intermediate to end-to-end assertions

Architecture & Execution For Hunting Bugs

Intermediate (Whitebox) properties

+ Good for bug hunting; smaller COI and easy to converge
- Needs design knowledge

```
assert st == RcvrCfg |-> s_eventually (st != RcvrCfg)

assert $rose(st == RcvrCfg) |-> $past(st == RcvrLock)

assert cfg_mode== Gen1|->  st != RcvrEqualization

assert st == Recovery.RcvrLock && equalize|-> ##1 st ==
RcvrEqualization
```

End-to-end checks (using boundary)

+ Map to spec level features; provides higher confidence
- Long temporal; harder to converge
- Harder to code and debug

```
assert curr_speed != cfg_target_speed && speed_supported &&
speed_negotiation_successful |-> ##[MIN:MAX] curr_speed ==
cfg_target_speed
```

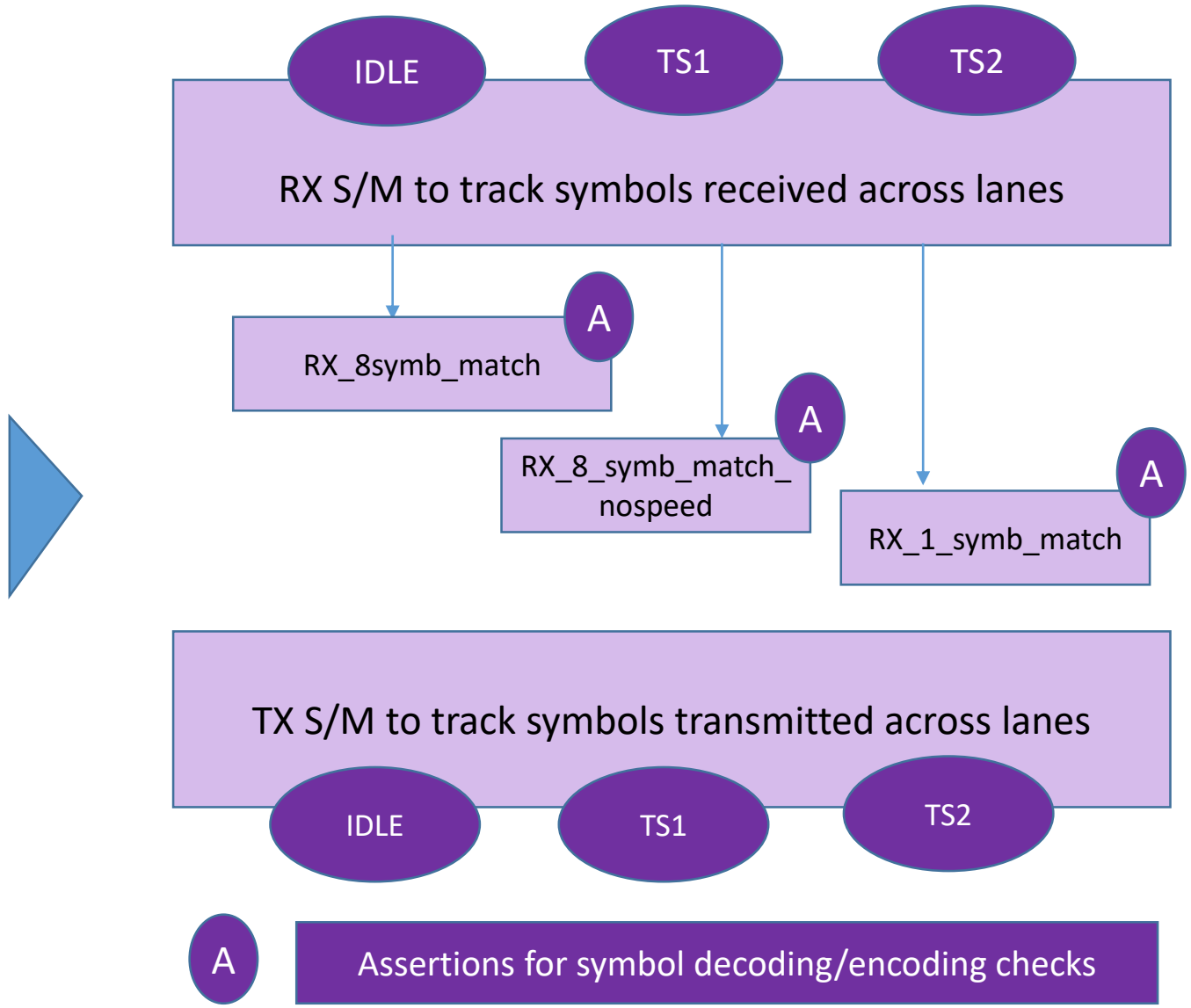
Safe Constraints and cover properties

+ Good for guiding deeper states
- Over-constraint (analysis required to mitigate risk)

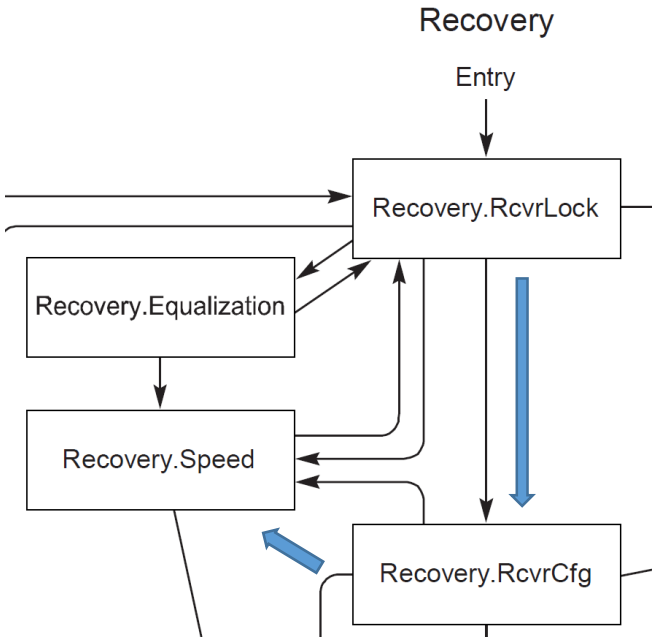
```
cover (past_speed == Gen2 && curr_speed == Gen3)
cover (past_speed == Gen3 && curr_speed == Gen4)
```

Technique 1: Separating data from control

- SPECIFICATION: From Recovery.RcvrCfg, Next state is Recovery.Idle if
  - Eight consecutive TS2 Ordered Sets are received with the same Link and Lane number that match what is being transmitted on those same Lanes with identical data rate w/ sub-conditions:
    - speed_change bit is 0b in the received eight consecutive TS2 Ordered Sets OR
    - current data rate is 2.5 GT/s and no higher, data rate identifiers are being transmitted in the TS2 Ordered Sets
  - 16 TS2 Ordered Sets are sent after receiving one TS2 Ordered Set



Technique 2: RTL style model for long input sequence



Avoid Long Temporal Assertions: speed_change_req ## [10:100] TS1_symb_recvd ## [10:100] (TS2_symb_rcvd && speed_change_bit == req_speed) |-> rtl_successful_speed_change == 1

RTL STYLE FORMAL TB:

```
assign recover_lock_to_recovery_config = sm == recovery_lock && TS_Symb_recvd;

always @(posedge clk…)
If(entry_to_recovery && speed_change_req)
  start_speed_neg <= 1'b1;
else if(exit_from_recovery)
  start_speed_neg <= 1'b0;

If(recover_lock_to_recovery_config) sm <= recovery_config
If(recovery_config && speed_change_bit == req_speed)
  sm <= recovery_speed
  expect_successful_speed_neg <= 1'b1;
```

Assertion: start_speed_neg && exp_succ_speed_neg |-> rtl_successful_speed_change == 1

Technique 3: Snipping For Deeper States

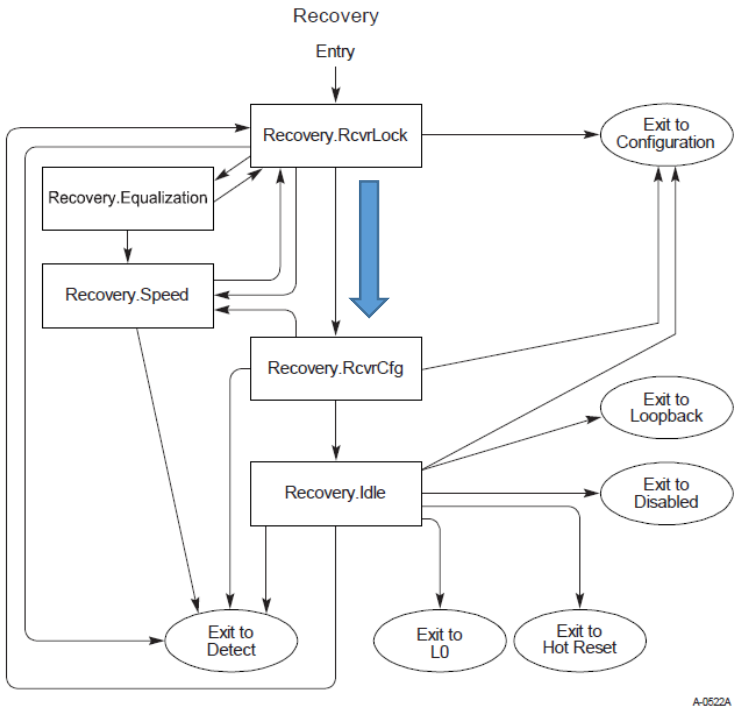


Figure 4-27: Recovery Substate Machine

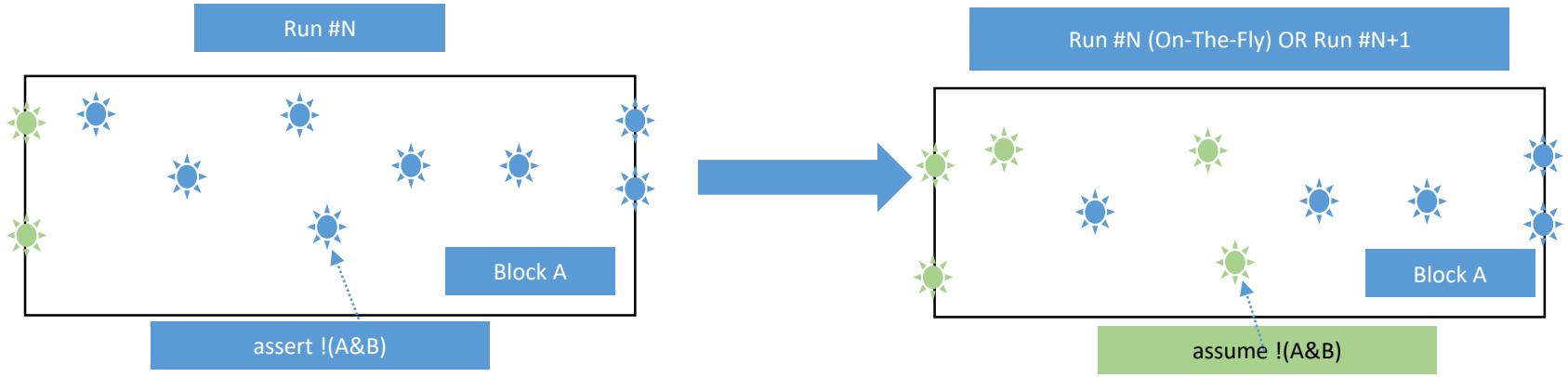
Cut-points to drive interesting scenarios on deep sequential signals with constraints

E.g. Deep state machine w/
- TS1_64_Symbol_Rcvd signal need to be asserted to go from RcvrLock to RcvrCgf State.
- Would need 64 consecutive TS1 symbols for signal to go high

Cut the driver for TS1_64_Symb_Rcvd
assume next_state == RcvrLock |-> ##[2:64] TS1_64_Symb_Rcvd

Technique 4: Assume Guarantee

Proven assertions are treated as assumptions to prove harder properties in common cone of influence

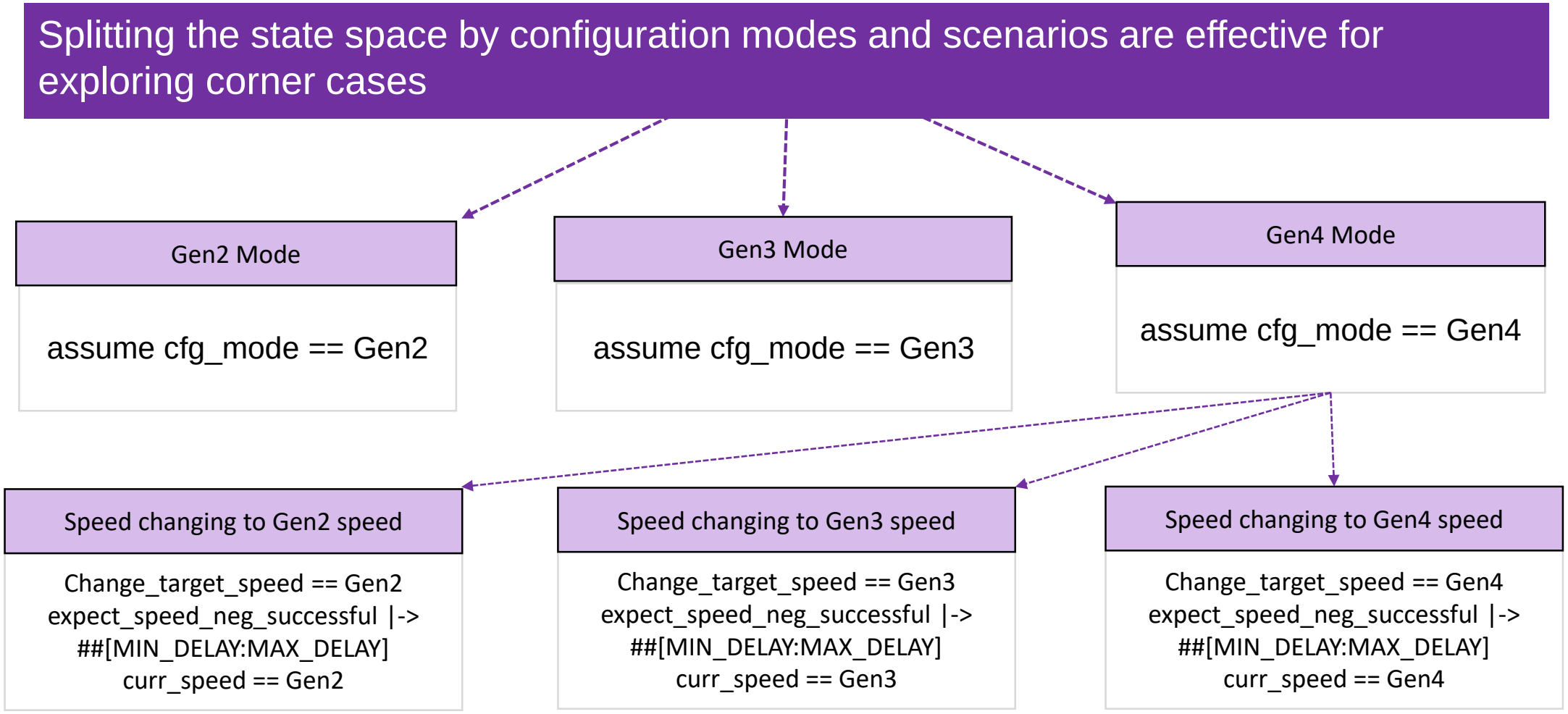



```
cfg_speed_supported == Gen1|->  state != Recovery.Equalization

State == Recovery.RcvrCfg  |-> $past(state == Recovery.RcvrLock)

State == Recovery.RcvrLock && equalize|-> Recovery.Equalization
```

```
assert Curr_speed != cfg_target_speed &&
cfg_speed_supported &&
speed_negotiation_successful
|-> ##[MIN:MAX] curr_speed == cfg_target_speed
```

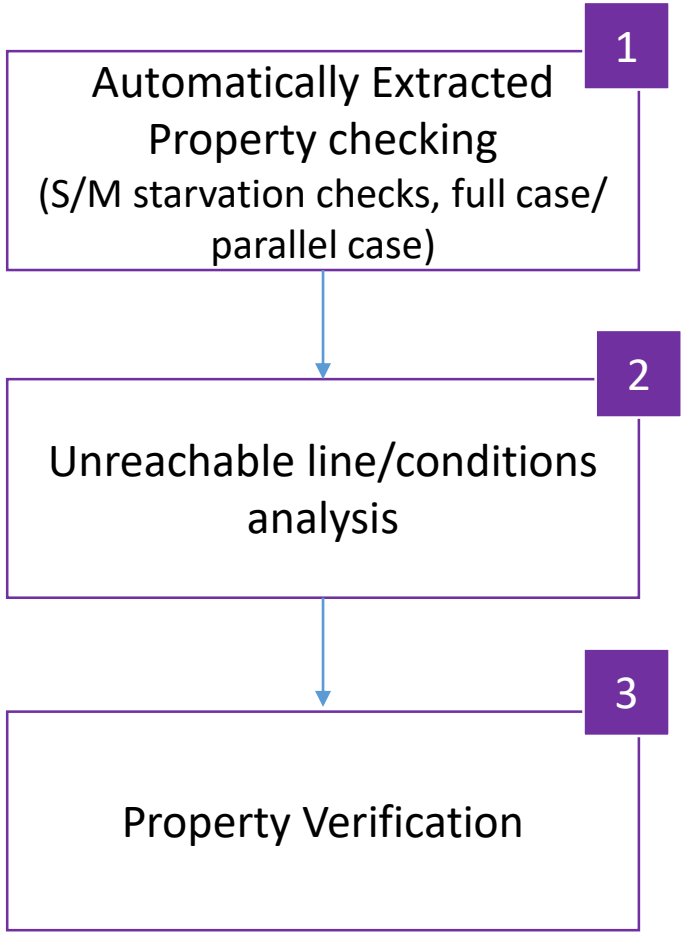
Internal (Whitebox) properties often act as effective invariants for harder end to end properties

Technique 5: Case or Scenario Splitting



Example Results

Design	PCIE 4.0 LTSSM
RTL Lines	~25K
FSM States	11 main states, total 65 states
Design Stage	PCIE 3.0 silicon proven, 4.0 new design; simulation coverage > 70%
# Bugs	10+ Corner case bugs founds
Formal Apps used	AEP, FCA, FPV
Time Required	3 Man Months



Example Bug

When multiple transition condition are true concurrently, priority among them may result into bugs.

Often harder to verify in simulation due to limited controllability and observability

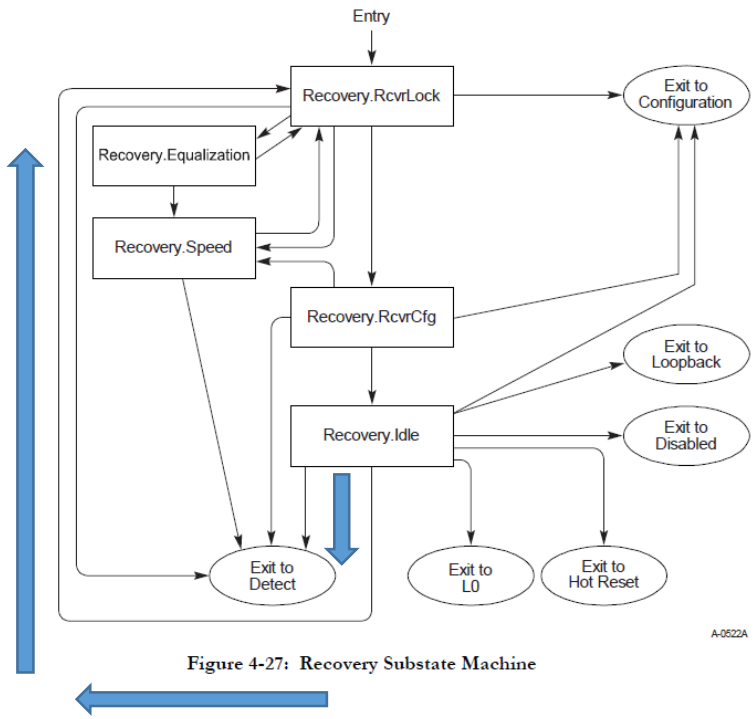


Figure 4-27: Recovery Substate Machine

case (Current_state)

…

Recovery.idle:

If(consecutive_8_symb_data_rcvd) next state == L0

….

else if(timeout_2ms_expired && no_activity) next_state = Detect

else if(!idle_to_rlock_transition_cnt_expired) next_state = Recovery.Lock

assert current_state == Recovery.Idle && *next_state == Detect |-> idle_to_rlock_trans_cnt == 8'hFF*

Summary

- Deep sequential FSMs pose challenges for verification and risk bug escapes
- Formal methodology w/ bug hunting intent rigorously applied from planning till execution stage
- Architecting formal TB with mix of intermediate design (whitebox) assertions and safe constraint key for finding bugs and end-to-end properties to converge
- Abstraction techniques such as snips, assume-guarantee, and cover properties guides formal engines for deeper state space exploration
- Corner case bugs are consistently found w/ reasonable efforts that are harder to find otherwise