2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

Efficient Coverage Optimization with Formal-Guided Testcase Generation in UVM Verification

Yu–Shien Shen[1], Yean–Ru Chen[1], Yu–Tung Chen[1], En–Hsiang Lin[2]

1 Department of Electrical Engineering, National Cheng Kung University, Taiwan (R.O.C.)

2 Realtek Semiconductor Corp., Taiwan (R.O.C.)

accellera SYSTEMS INITIATIVE

Outline

- Introduction
- Related Work
- Case Study
- Experiment Results
- Conclusions

Outline

- Introduction
- Related Work
- Case Study
- Experiment Results
- Conclusions

Introduction

- Design Complexity
  - Verification is a critical bottleneck in modern IC development, consuming up to 70% of design time.
  - Challenges in modern verification
    - Hard-to-detect bugs in large-scale designs
    - High verification cost and computational demand
    - Time-to-market pressure for competitive product releases

Introduction

- Verification Method
  - Simulation Verification
    - Advantages : Excellent scalability / Constrained Random Verification
    - Disadvantages : Biased test scenarios / Missing corner cases
  - Formal Verification
    - Advantages : Precisely verifies correctness / Exhaustive state exploration
    - Disadvantages : Highly susceptible to state explosion / False alarms may lead to unnecessary debugging efforts

Introduction

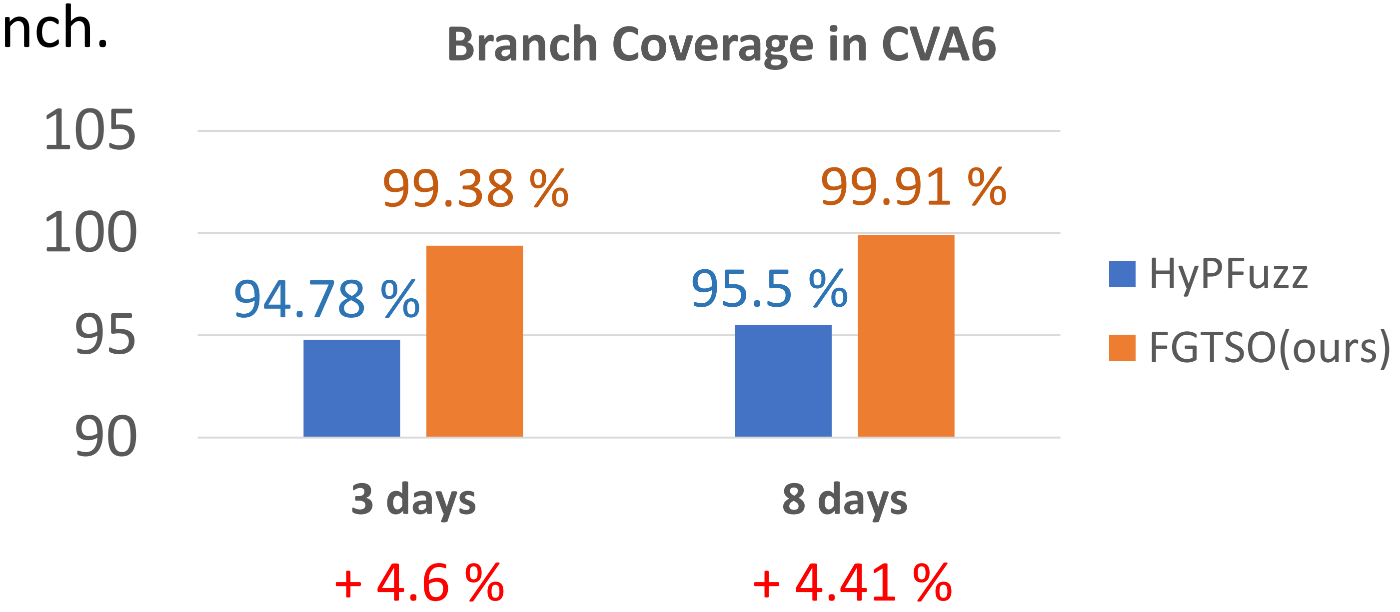
- Our Proposal
  - We aim to develop a verification framework that maximizes simulation coverage.
  - Formal-Guided Verification:
    - Integrating simulation’s scalability with formal’s precision and guidance
    - Mitigating formal false alarms to ensure consistent execution results
    - Mitigating coverage holes related to black-box module by formal tools (due to excessive storage and computational complexity)

Introduction – Contribution

- Framework (Formal-Guided Simulation)
  - Propose Formal-Guided Test Sequence Optimization (FGTSO)
  - Refining formal assumptions to align formal verification with simulation, minimizing false alarms.
  - Mitigating the limitations of formal verification in handling black-box modules caused by high storage and computational complexity.

Introduction – Contribution

- Achievements on RISC-V processor CVA6
  - Within ten days, we achieved 100% coverage across all metrics — including line, toggle, condition, and branch.



Outline

- Introduction
- Related Work
- Case Study
- Experiment Results
- Conclusions

Related Work

Related Work	RISC-V Torture [1]	RISC-V DV [2]	TheHuzz [3]	HyPFuzz [4]
Advantages	Basic functionality check	Constrained random verification	Fuzzing-based + Golden reference model	Fuzzing-based + Formal verification
Disadvantages	Limited test scope	Rely on randomness, missing corner cases	No systematic guidance, incomplete coverage	Prone to false alarms / Lack handle black-box issues

[1]https://github.com/ucb-bar/riscv-torture

[2]https://github.com/google/riscv-dv

[3] Rahul Kande, Addison Crump, Garrett Persyn, Patrick Jauernig, Ahmad-Reza Sadeghi, Aakash Tyagi, and Jeyavijayan Rajendran. {TheHuzz}:Instruction fuzzing of processors using {Golden-Reference} models forfinding {Software-Exploitable} vulnerabilities

[4] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen,Aakash Tyagi, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran.{HyPFuzz}:{Formal-Assisted} processor fuzzing

Related Work

Feature	HyPFuzz [4]	FGTSO (Ours)
Test Generation	Fuzzing-based mutation engine + Formal verification	Formal-guided test case generation
False Alarms	No discussion	✓ Align formal and simulation
Black-box Handling	No discussion	✓ Construct abstraction model
Simulation Coverage Efficiency	Good, but random-based	✓ Better, more effective test cases
Automation	✓ Easy to automation	Need few manual intervention

[4] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen,Aakash Tyagi, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran.{HyPFuzz}:{Formal-Assisted} processor fuzzing

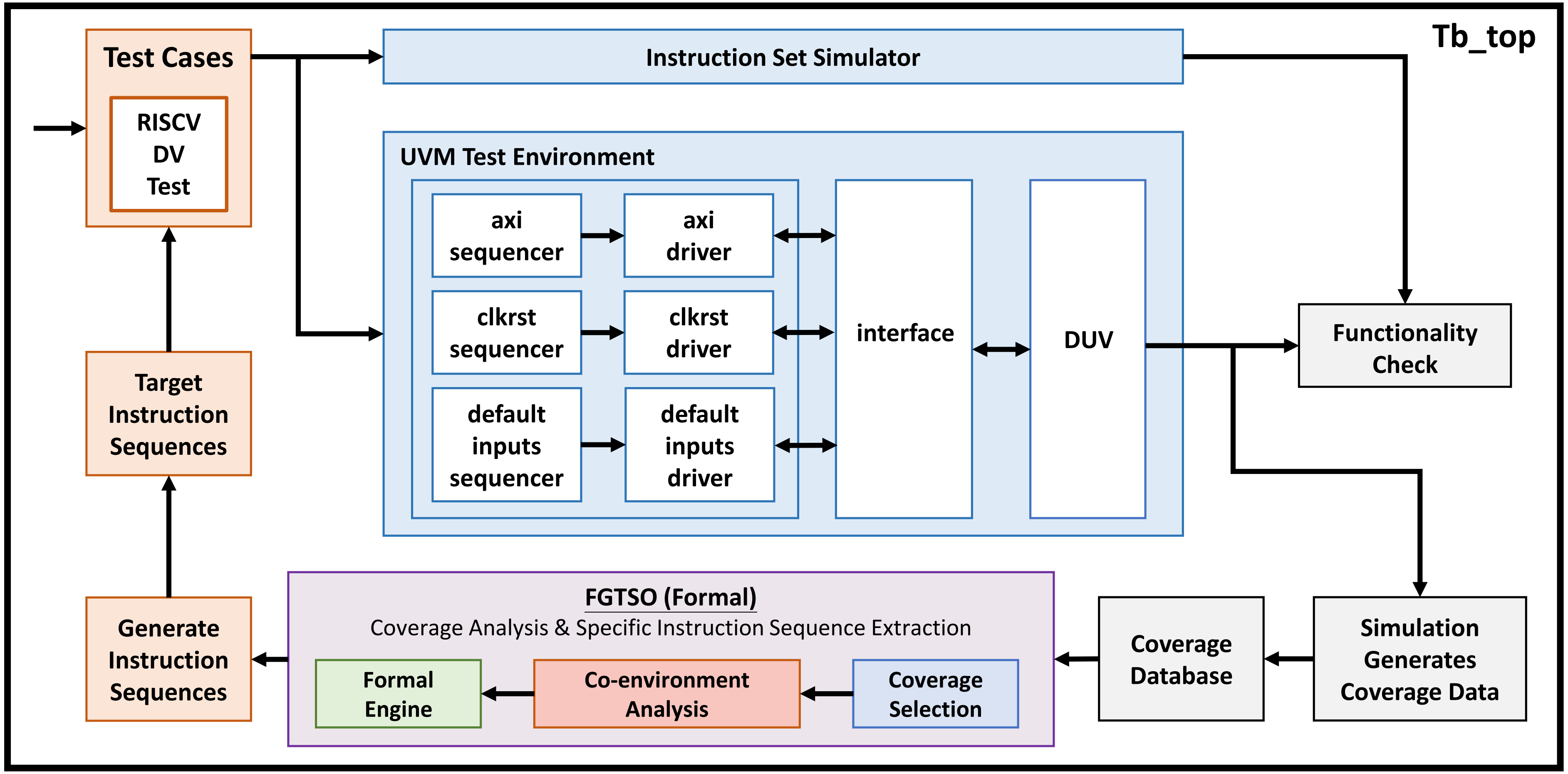
Outline

- Introduction
- Related Work
- Case Study
- Experiment Results
- Conclusions

Case Study – SVA Types

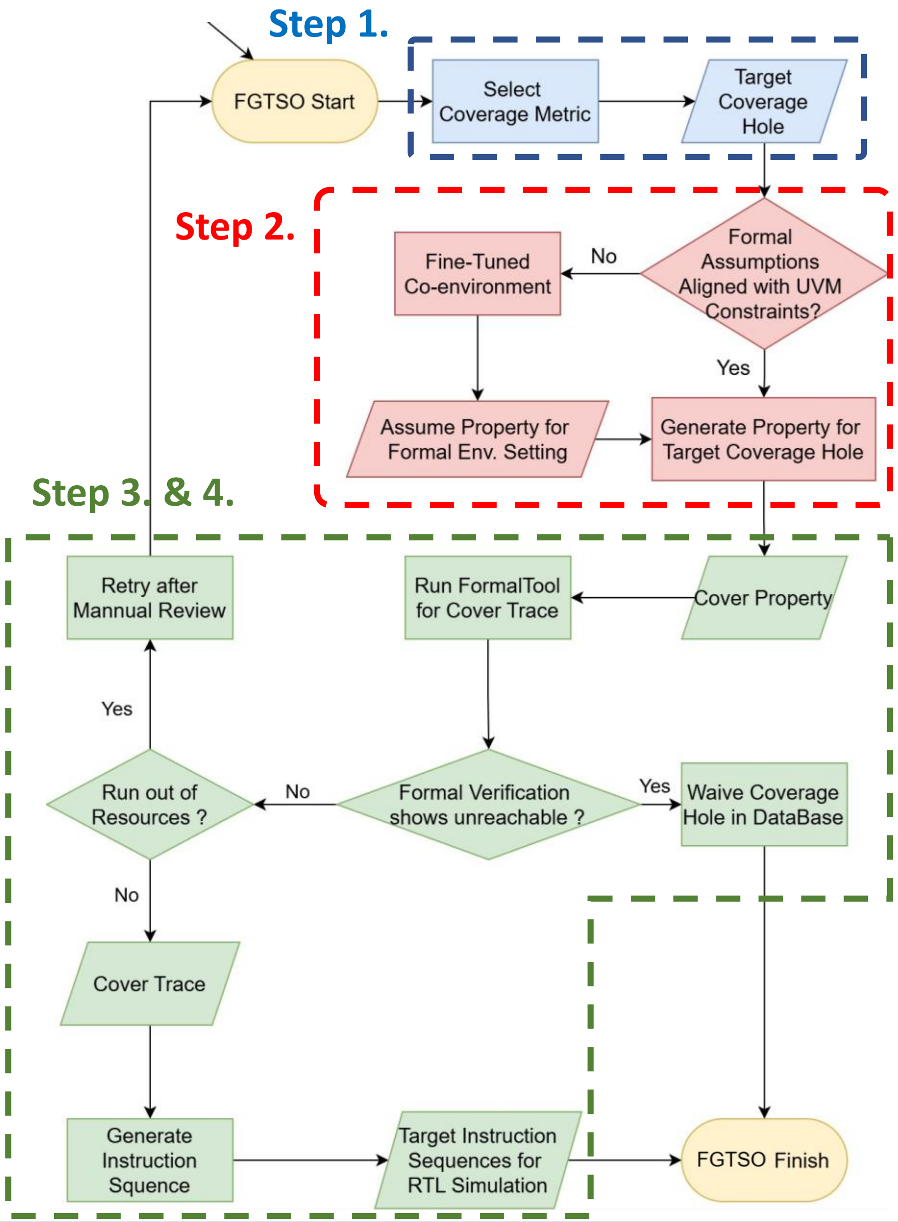
- Assume
  - Constraints on inputs : Define the operational environment
- Cover
  - Observability check : Identify reachable states
- Assert
  - Correctness check : Guarantee a property must always hold

Case Study – Proposed Framework

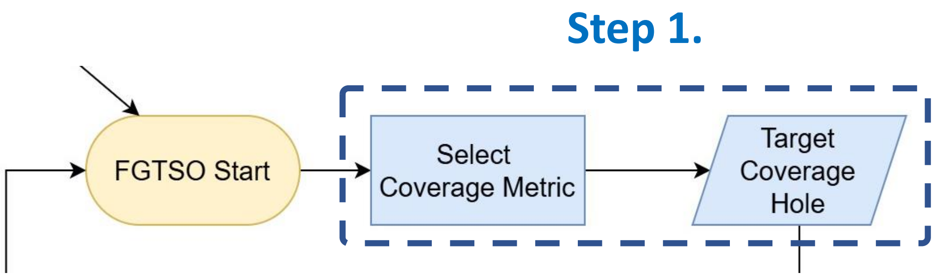


Case Study – FGTSO Workflow

1. Simulation Coverage Hole Selection
   - Prioritize coverage holes for targeted verification
2. Environment Consistency Check
   - Align formal with simulation to reduce false alarm
   - Mitigate bbox issues in formal
3. Formal Cover Property Generation
   - Define uncovered design behaviors to guide verification
4. Targeted Instruction Sequences Extraction
   - Extract specific test sequences to address identified holes



Case Study – Step 1.

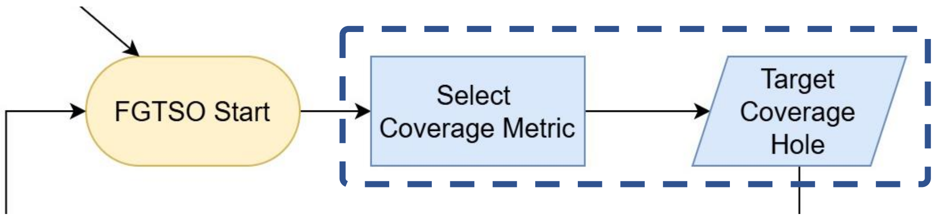


- Select an impactful simulation coverage hole
  - MaxUncovd Strategy [2]

SCORE	BRANCH	NAME
86.05	86.05	wt_dcache_ctrl
87.50	87.50	fpnew_opgroup_fmt_slice
88.19	88.19	control_mvp
88.24	88.24	controller
90.00	90.00	scoreboard
90.00	90.00	bht
91.67	91.67	fifo_v3

[2] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen,Aakash Tyagi, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran.{HyPFuzz}:{Formal-Assisted} processor fuzzing.

Step 1.



Case Study – Step 1.

- Select an impactful simulation coverage hole
  - MaxUncovd Strategy [2]

SCORE	BRANCH	NAME
86.05	86.05	wt_dcache_ctrl
87.50	87.50	fpnew_opgroup_fmt_slice
88.19	88.19	control_mvp
88.24	88.24	controller
90.00	90.00	scoreboard
90.00	90.00	bht
91.67	91.67	fifo_v3

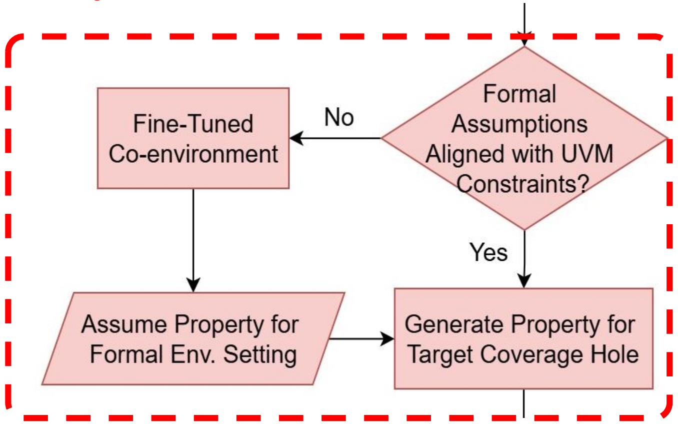
```
      MISS_REQ: begin
        miss_req_o = 1'b1;

        if (req_port_i.kill_req) begin
          req_port_o.data_rvalid = 1'b1;
          if (miss_ack_i) begin
            state_d = KILL_MISS;
          end else begin
            state_d = KILL_MISS_ACK;
          end
        end else if (miss_replay_i) begin
          state_d = REPLAY_REQ;
        end else if (miss_ack_i) begin
          state_d = MISS_WAIT;
        end
```


[2] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen,Aakash Tyagi, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran.{HyPFuzz}:{Formal-Assisted} processor fuzzing.

Case Study – Step 2.

Step 2.

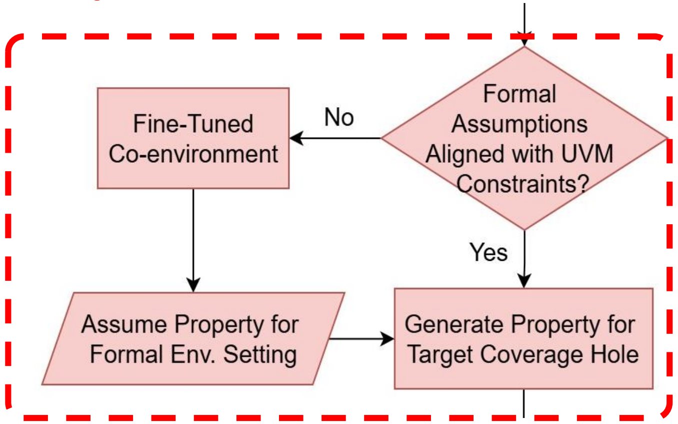


- Mitigation False Alarm in Formal
  - Align Formal and Simulation Environment

1. Primary inputs treated as free nets
   - Allow signals to take arbitrary values
2. Large storage or complex computation units treated as black boxes
   - Outputs unconstrained by internal logic are regarded as arbitrary values

Case Study – Step 2.

Step 2.



1. Primary inputs treated as free nets
   - Allow signals to take arbitrary values

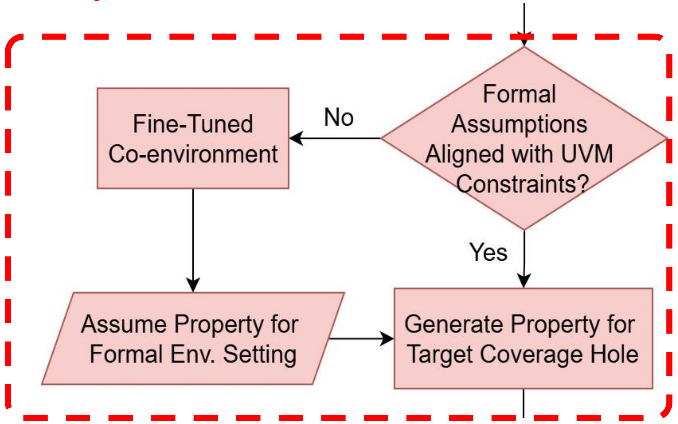
- Case (Automatic / Manual)
  - Insufficiently generated instruction pattern for memory operation coverage holes
  - Enforce AXI **formal assumption** to address memory operation coverage holes

```
r_valid: assume property (
  (req.ar_valid && req.ar_ready) |=> ##[0:$] (resp.r_valid)
);
```

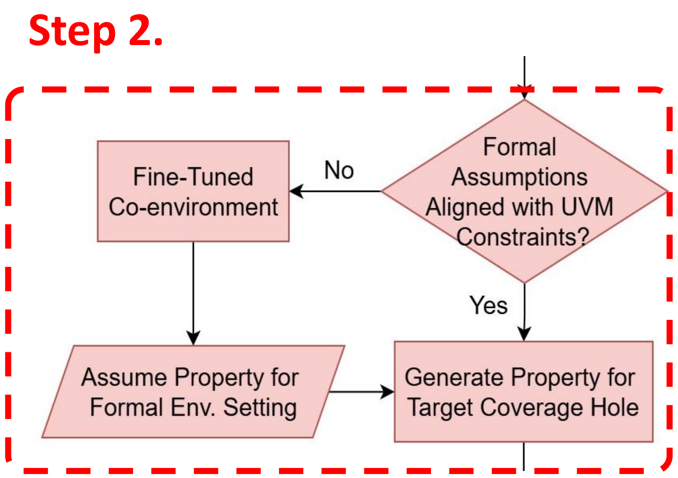
Case Study – Step 2.

2. Large storage or complex computation units treated as black boxes

Step 2.

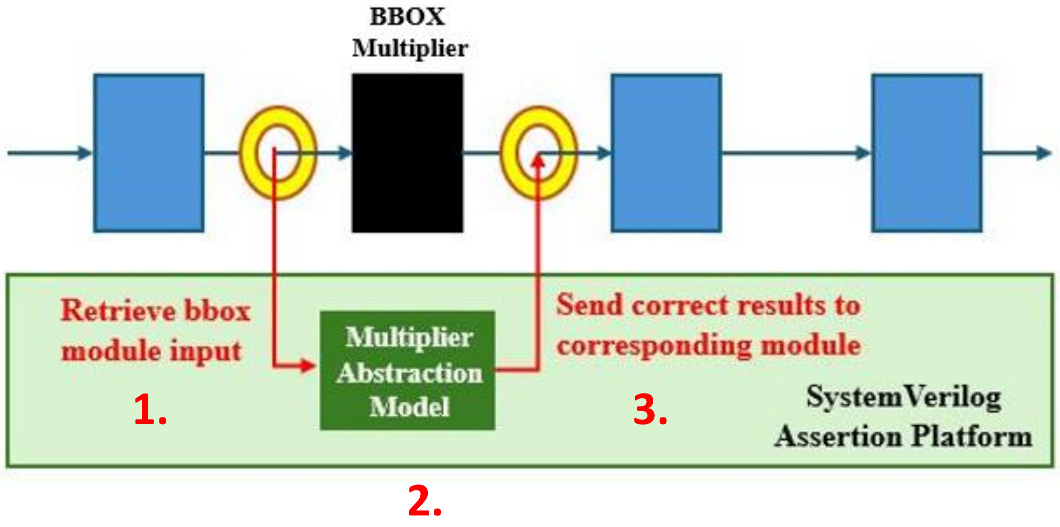


Case Study – Step 2.



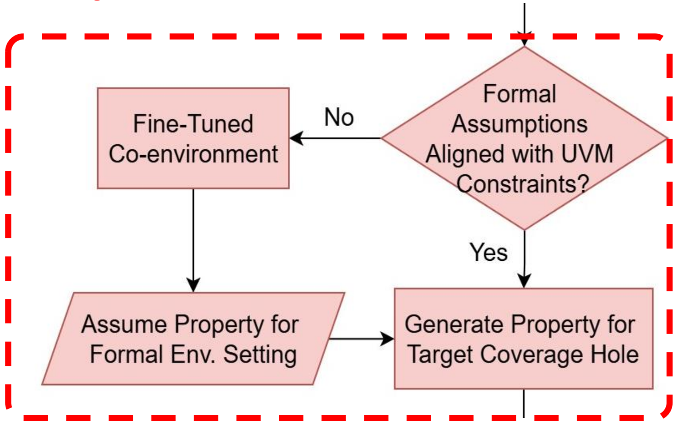
2. Complex computation units treated as black boxes
   - Outputs unconstrained by internal logic are regarded as arbitrary values

- Implementation (Manual)
  1. Capture the original multiplier input
  2. Compute the correct result using SVA
  3. Output the result with control signals



Case Study – Step 2.

Step 2.



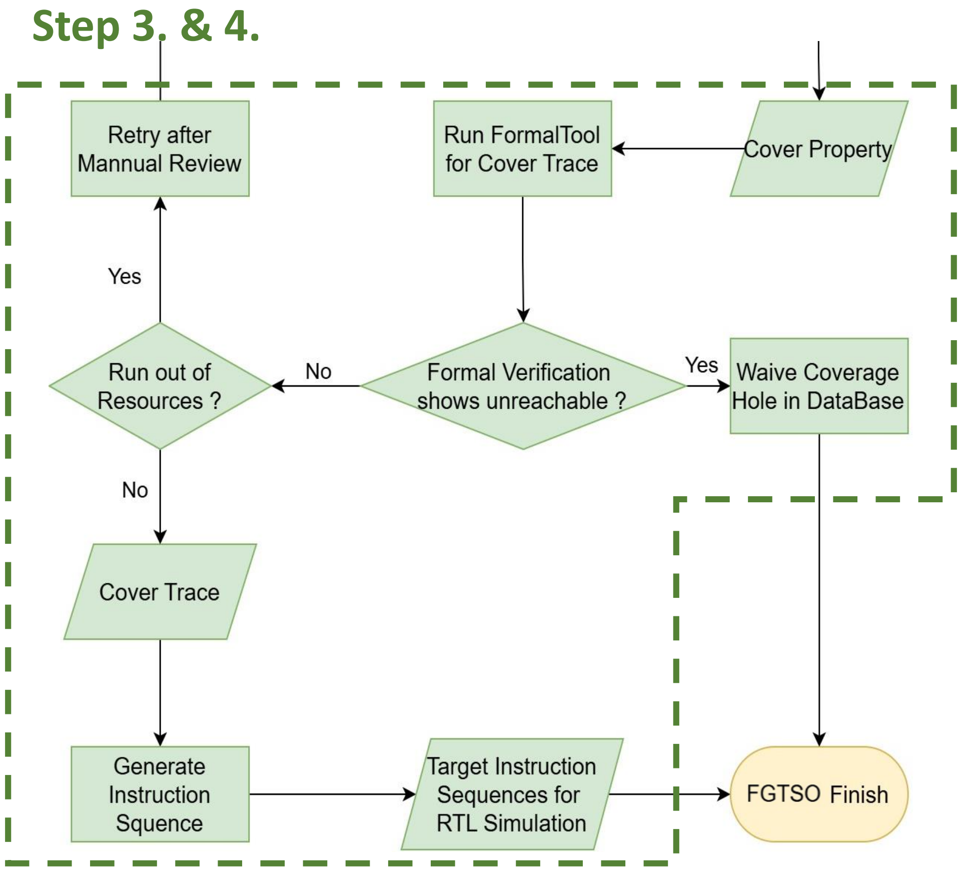
- Mitigation False Alarm in Formal
  - Align Formal and Simulation Environment

- Misalign Detection
  1. Generate target instruction sequences using formal tool
  2. Execute the target instruction sequences simultaneously in both environments
  3. If results mismatch, misalignment is detected

Methodology – Step 3. & 4.

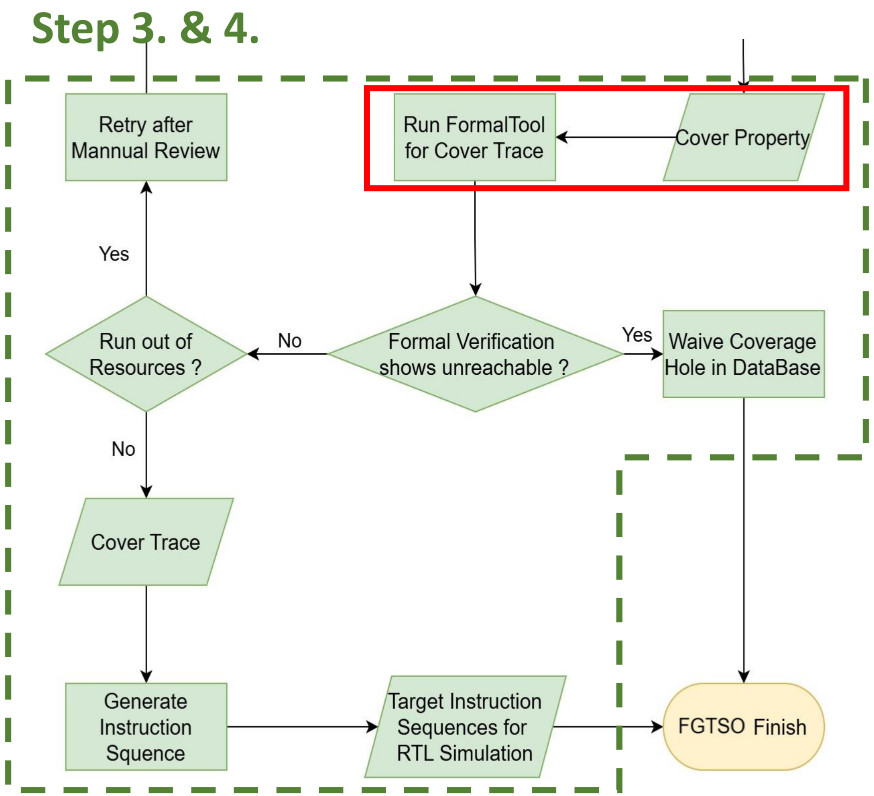
- Generate instruction sequences for target coverage hole

- Formal-Guided Waveform Extraction
  1. Automatically integrate coverage hole expressions into formal cover templates
  2. Acquire cover traces from formal cover property
  3. Extract target instruction sequence from cover traces



Methodology – Step 3. & 4.

- Formal-Guided Waveform Extraction
  1. Automatically integrate coverage hole expressions into formal cover templates
  2. Acquire cover traces from formal cover property
  3. Extract target instruction sequence from cover traces

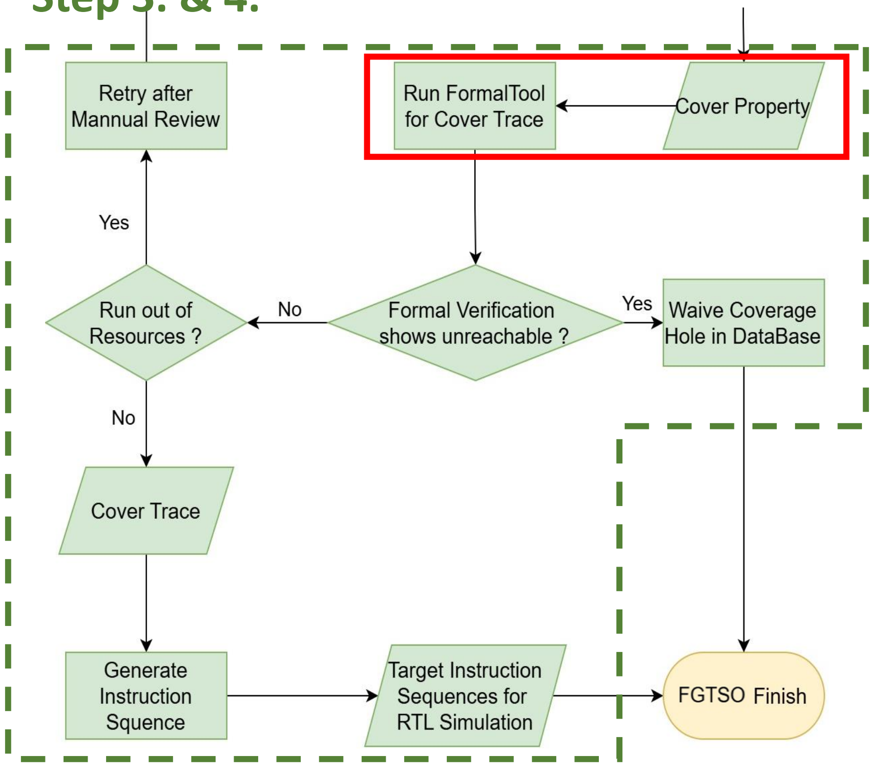


Case Study – Step 3. & 4.

```
    1. MISS_REQ: begin
         miss_req_o = 1'b1;

    2.   if (req_port_i.kill_req) begin
           req_port_o.data_rvalid = 1'b1;
           if (miss_ack_i) begin
             state_d = KILL_MISS;
           end else begin
             state_d = KILL_MISS_ACK;
           end
         end else if (miss_replay_i) begin
           state_d = REPLAY_REQ;   3.
         end else if (miss_ack_i) begin
           state_d = MISS_WAIT;
         end
```

Step 3. & 4.

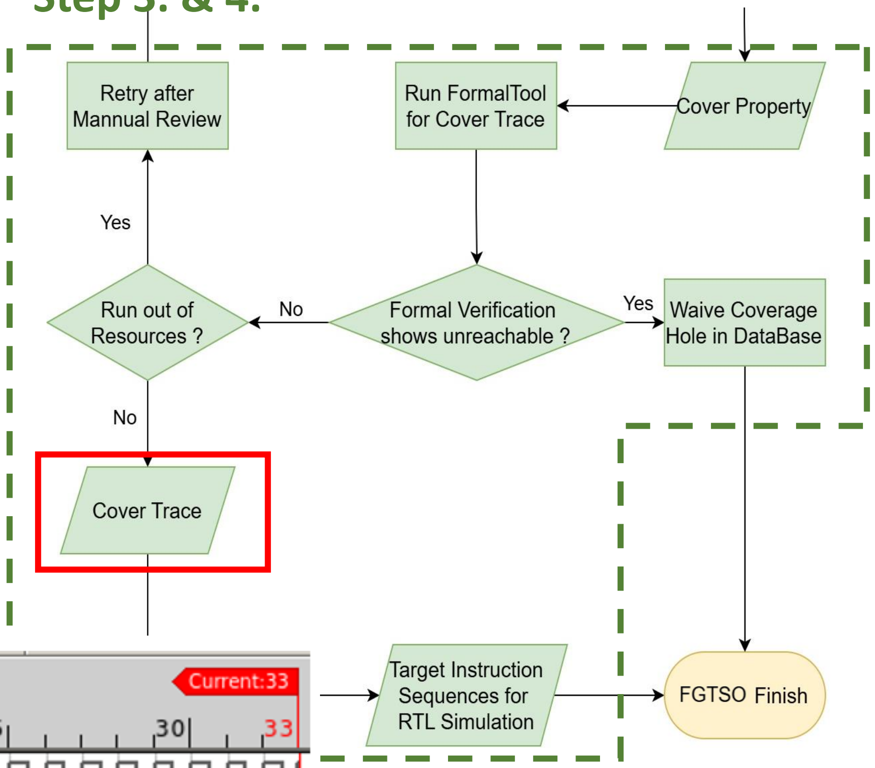


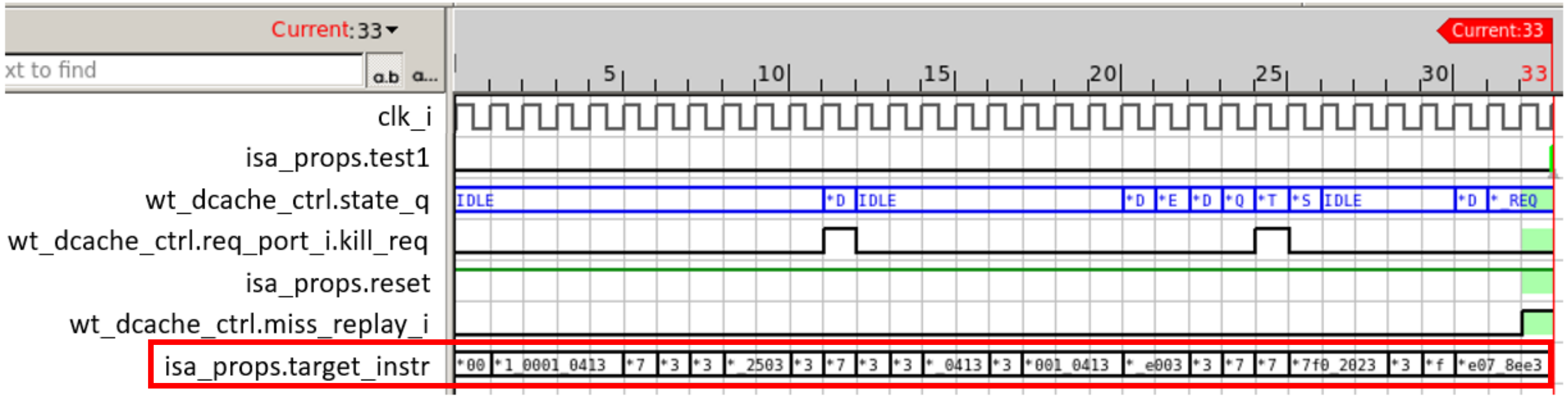

```
test1: cover property
(
    gen_cache_wt.i_cache_subsystem.i_wt_dcache.gen_rd_ports[1].genblk1.i_wt_dcache_ctrl.state_q == 'h2        1.
    &&
    ~gen_cache_wt.i_cache_subsystem.i_wt_dcache.gen_rd_ports[1].genblk1.i_wt_dcache_ctrl.req_port_i.kill_req  2.
    &&
    gen_cache_wt.i_cache_subsystem.i_wt_dcache.gen_rd_ports[1].genblk1.i_wt_dcache_ctrl.miss_replay_i         3.
);
```

Case Study – Step 3. & 4.

- Formal-Guided Waveform Extraction
  1. Automatically integrate coverage hole expressions into formal cover templates
  2. Acquire cover traces from formal cover property
  3. Extract target instruction sequence from cover traces

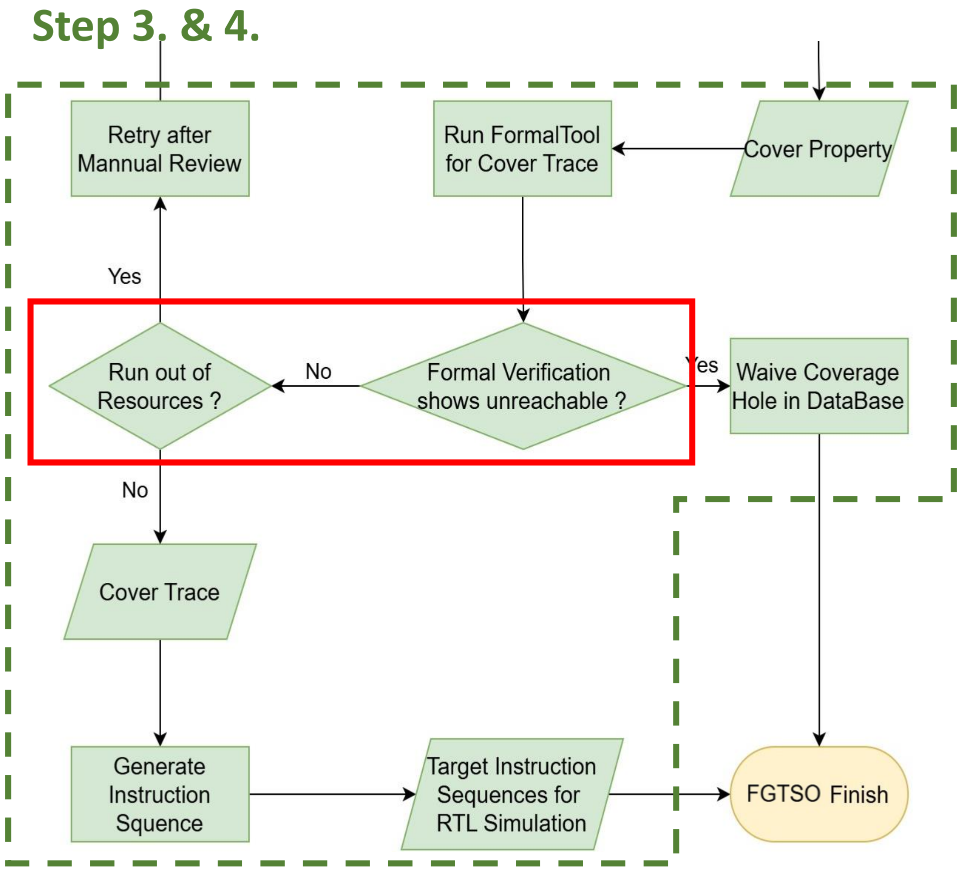
Step 3. & 4.





Case Study – Step 3. & 4.

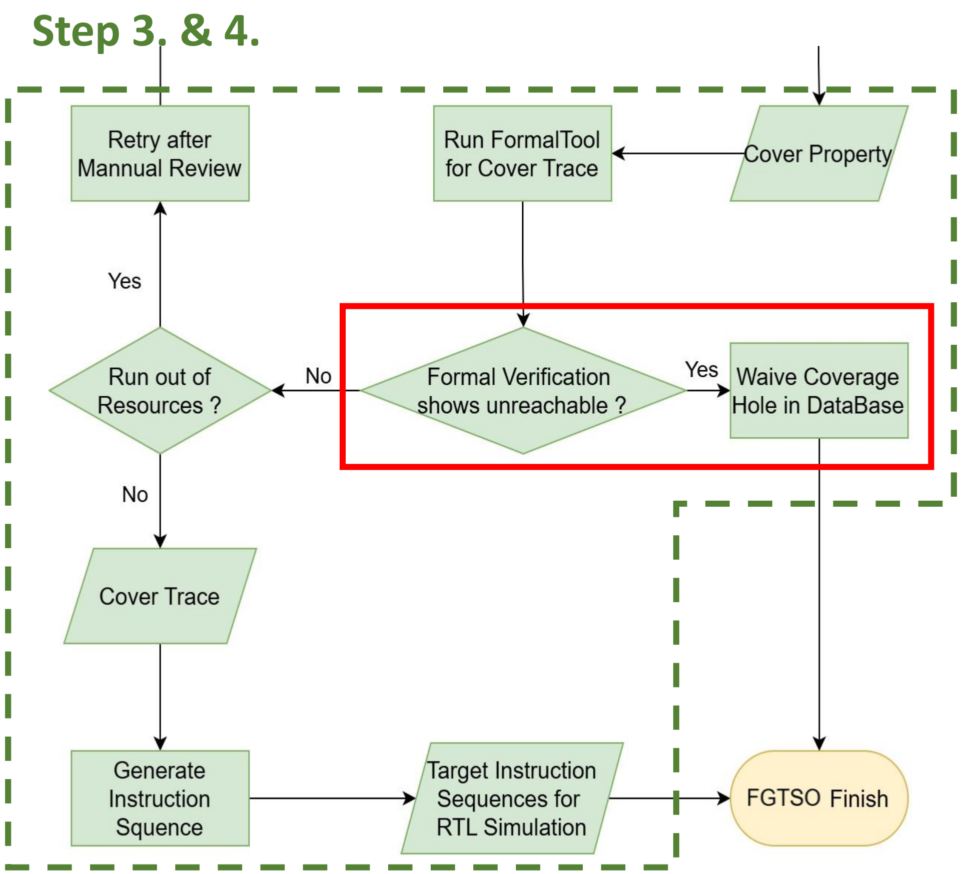
- Generate instruction sequences for target coverage hole
- Limitation of Formal Verification Tool
  - Unreachable
  - Run out of resource



Case Study – Step 3. & 4.

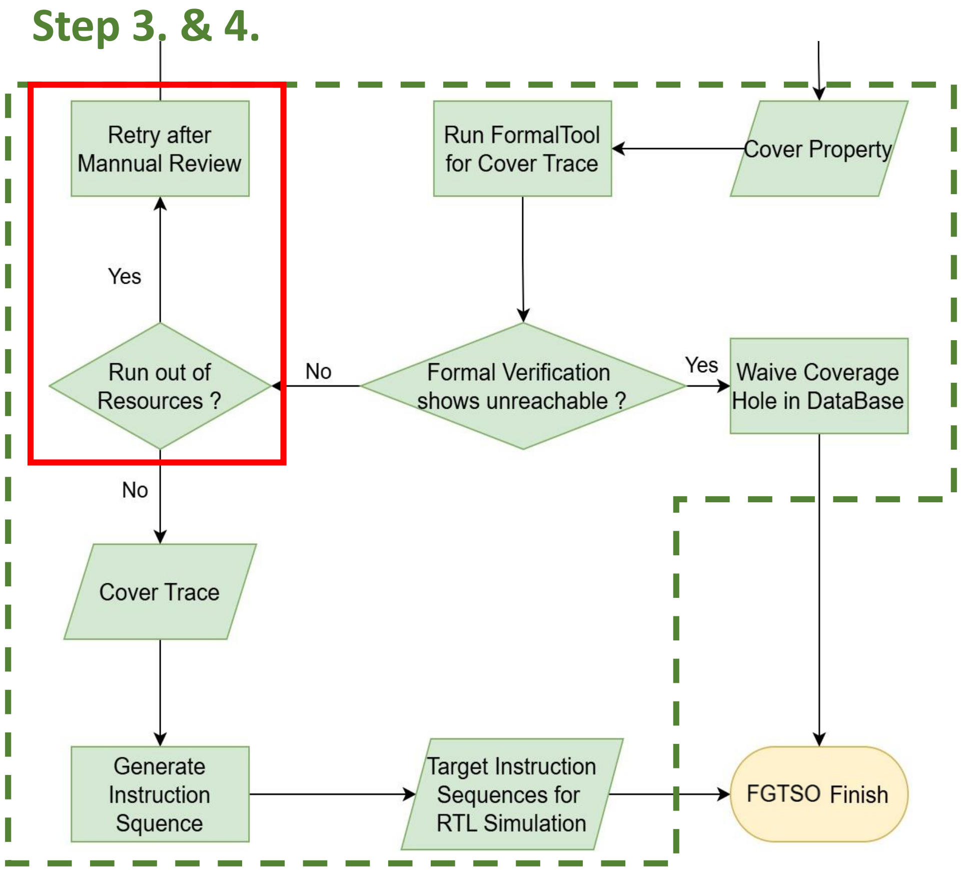
- Generate instruction sequences for target coverage hole

- Formal tool shows unreachable
    1. Check whether moderate constraints exist in both environment.
    2. If constraints are moderate, waive them to ensure progress, pending designer review for validation and refinements.



Case Study – Step 3. & 4.

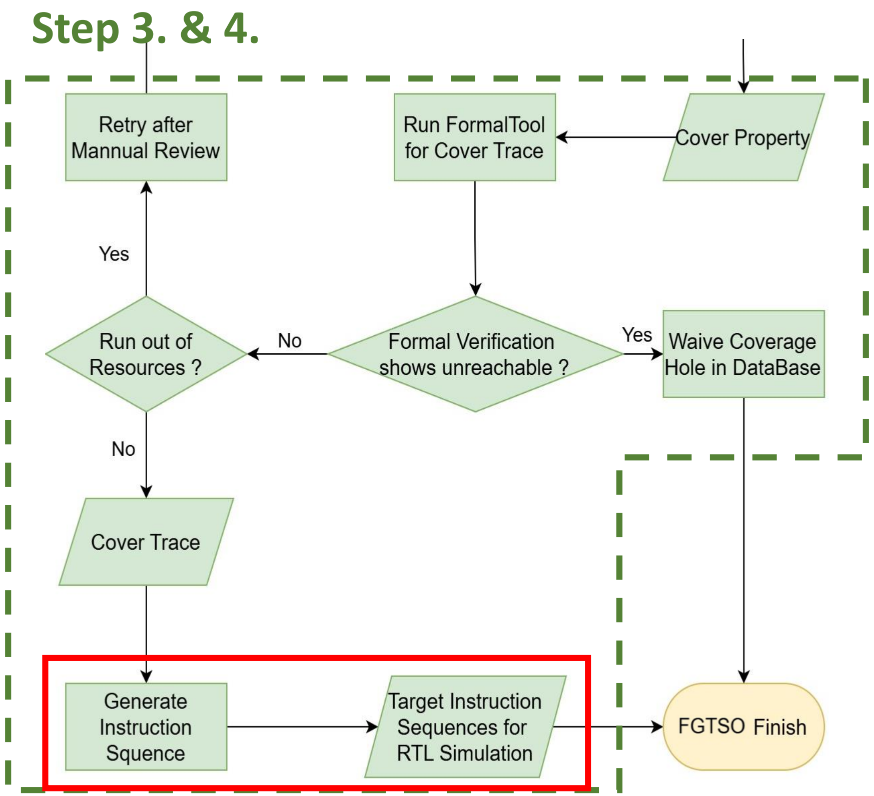
- Generate instruction sequences for target coverage hole
- Formal tool shows run out of resource
  1. Dividing complex property into sub-properties
  2. Collect respective results from sub-properties, and integrate them into target instruction sequences



Case Study – Step 3. & 4.

- Formal-Guided Waveform Extraction
  1. Automatically integrate coverage hole expressions into formal cover templates
  2. Acquire cover traces from formal cover property
  3. Extract target instruction sequence from cover traces

```
main:
        addiw x10, x10, -27
        ld x24, 0(x2)
        slli x8, x8, 8
        addi x8, x2, 0
        csrrw x16, 0x7c1, x2
        lh x15, 4(x2)
        lw x9, 0(x8)
        lhu x3, 1192(x24)
        csrrsi x8, 0x7c1, 3
        lb x28, 1536(x9)
        beq x1, x2, next13632
next13632:
        slli x8, x8, 8
        lbu x0, -2047(x15)
```



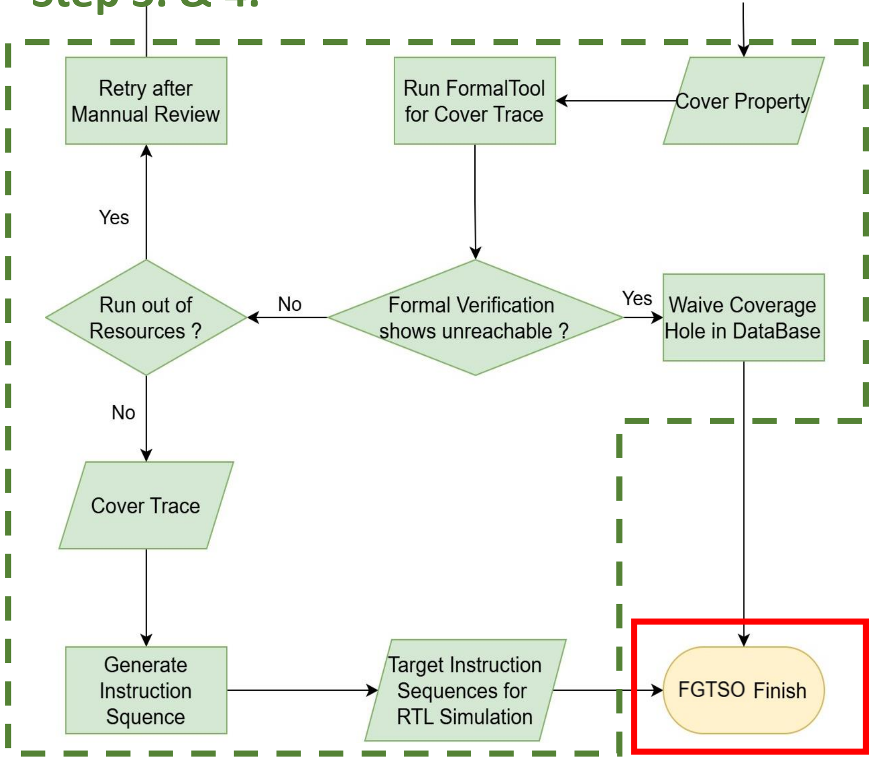
Case Study – Step 3. & 4.

- Formal-Guided Waveform Extraction
  1. Automatically integrate coverage hole expressions into formal cover templates
  2. Acquire cover traces from formal cover property
  3. Extract target instruction sequence from cover traces

```
    MISS_REQ: begin
      miss_req_o = 1'b1;

      if (req_port_i.kill_req) begin
        req_port_o.data_rvalid = 1'b1;
        if (miss_ack_i) begin
          state_d = KILL_MISS;
        end else begin
          state_d = KILL_MISS_ACK;
        end
      end else if (miss_replay_i) begin
        state_d = REPLAY_REQ;
      end else if (miss_ack_i) begin
        state_d = MISS_WAIT;
      end
```

Step 3. & 4.



Outline

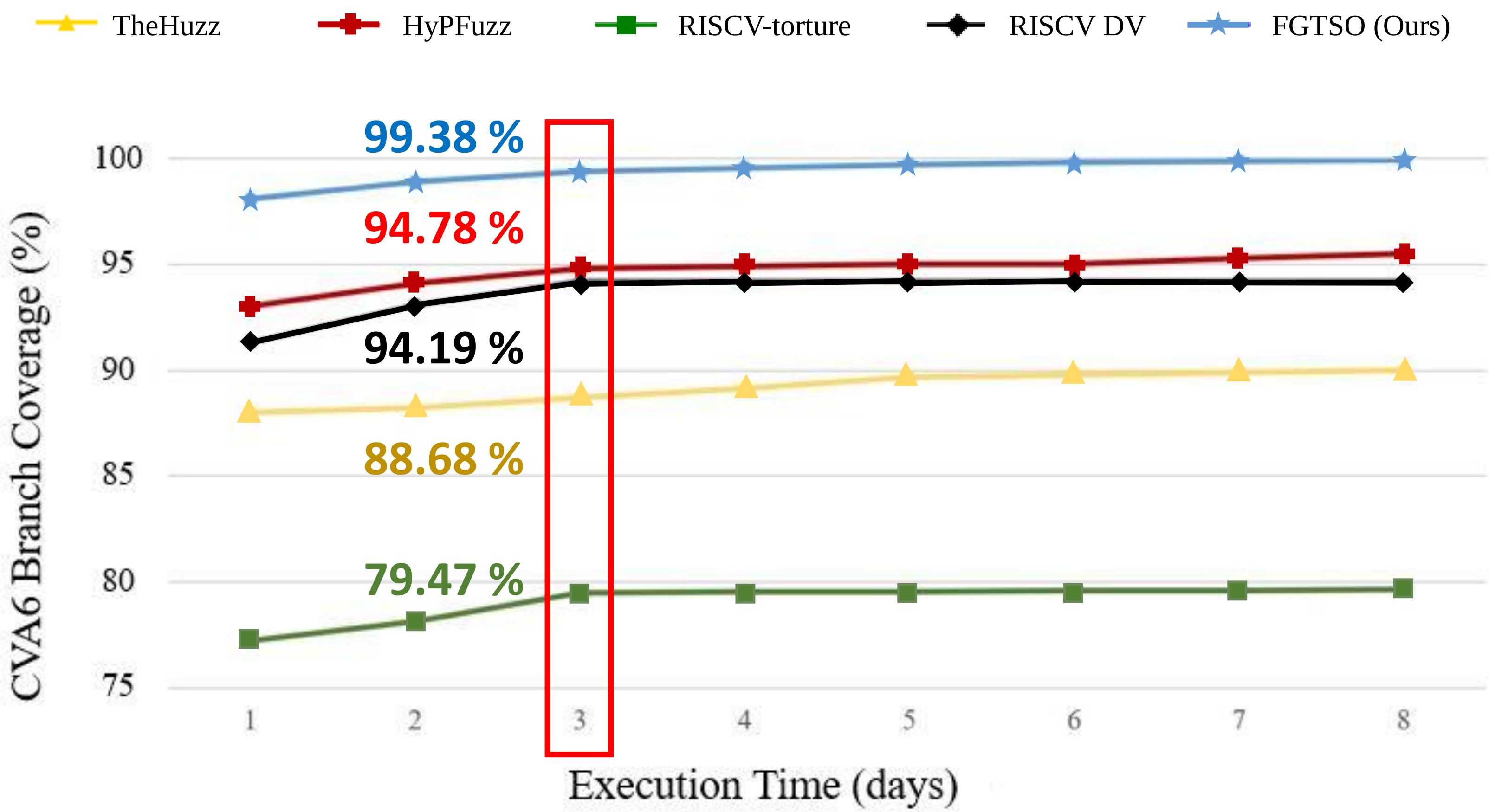
- Introduction
- Related Work
- Case Study
- Experiment Results
- Conclusions

[2] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen,Aakash Tyagi, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran.{HyPFuzz}:{Formal-Assisted} processor fuzzing.

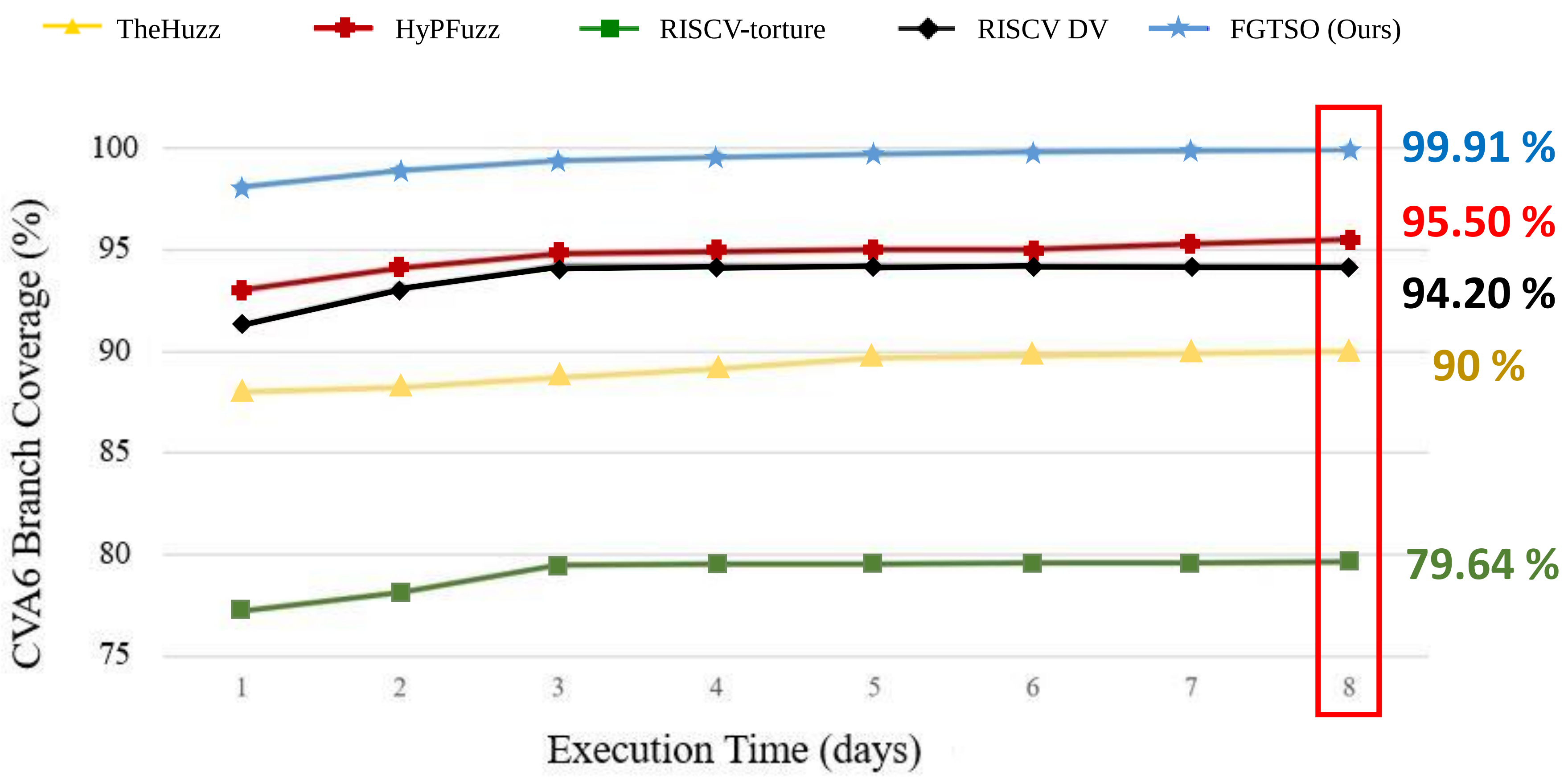
Experiment Setup

- Design Under Verification
  - CVA6 RISC-V core
- Tools Used
  - Cadence Jasper for formal verification, Synopsys VCS for simulation
- Execution Time
  - Experiments conducted over 3-day and 8-day periods
- Comparison
  - RISC-V Torture、RISCV-DV : re-run by their open source
  - TheHuzz、HyPFuzz : gather from Chen et al.[2]

Experiment Results



Experiment Results



Experiment Results

- Within ten days, our FGTSO-based framework reaches 100% across all metrics including line, toggle, condition, and branch coverage

Name	Score	Line	Toggle	Condition	Branch
uvmt_cva6_tb	100.00%	100.00%	100.00%	100.00%	100.00%
cva6_dut_wrap	100.00%	100.00%	100.00%	100.00%	100.00%
cva6_tb_wrapper_i	100.00%	100.00%	100.00%	100.00%	100.00%
i_cva6	100.00%	100.00%	100.00%	100.00%	100.00%
commit_sta...	100.00%	100.00%	100.00%	100.00%	100.00%
controller_i	100.00%	100.00%	100.00%	100.00%	100.00%
csr_regfile_i	100.00%	100.00%	100.00%	100.00%	100.00%
ex_stage_i	100.00%	100.00%	100.00%	100.00%	100.00%
gen_cache_...	100.00%	100.00%	100.00%	100.00%	100.00%
i_frontend	100.00%	100.00%	100.00%	100.00%	100.00%
id_stage_i	100.00%	100.00%	100.00%	100.00%	100.00%
issue_stage_i	100.00%	100.00%	100.00%	100.00%	100.00%

Outline

- Introduction
- Related Work
- Case Study
- Experiment Results
- Conclusions

[2] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen,Aakash Tyagi, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran.{HyPFuzz}:{Formal-Assisted} processor fuzzing.

Conclusions

- Formal-Guided Simulation Framework
  - Successfully combines formal precision with simulation scalability
  - FGTSO deals with false alarm and bbox issues in formal for simulation
- Superior Performance in Branch Coverage
  - Outperforms related work in simulation coverage and efficiency
  - 4.41 % improvement compared to HyPFuzz [2]
- Comprehensive Verification
  - Achieves 100% simulation coverage across all metrics within ten days

Thank you for your attention

Q & A

Author Contact Information: chenyr@mail.ncku.edu.tw