2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

FVDebug: LLM-Driven Formal Verification Debugging Assistant

- **Presenter**: Yunsheng Bai
- **Other Authors**: Ghaith Bany Hamad, Chia-Tung (Mark) Ho, Syed Suhaib, Haoxing (Mark) Ren

Formal Verification Debugging

´ Δ’ À– ∫ “ Δƒ Æƒ¬÷À–

FV Engineer

Formal Testbench

Formal Tool, e.g. Jasper

Counterexample (CEX)

Design Engineer

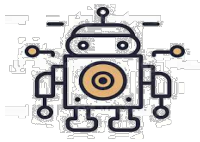
Design Implementation (RTL)

©◊» ØΔ‘Δº⁄¥ À’ À» ƒ—’ ÷‘¬À÷%œ ‘—» ¬’ ’ Δ‘÷À—%

Waveforms, etc.

Bug Here?

Can LLM Agents assist in FV debugging?

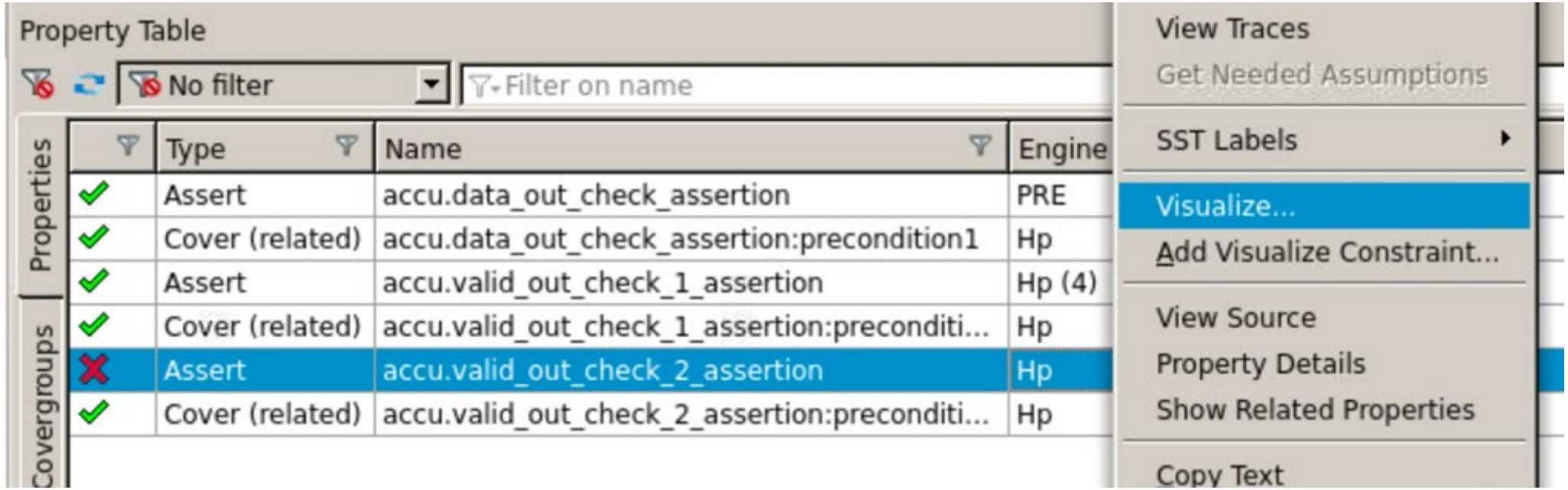


Example Input: Accumulator Design

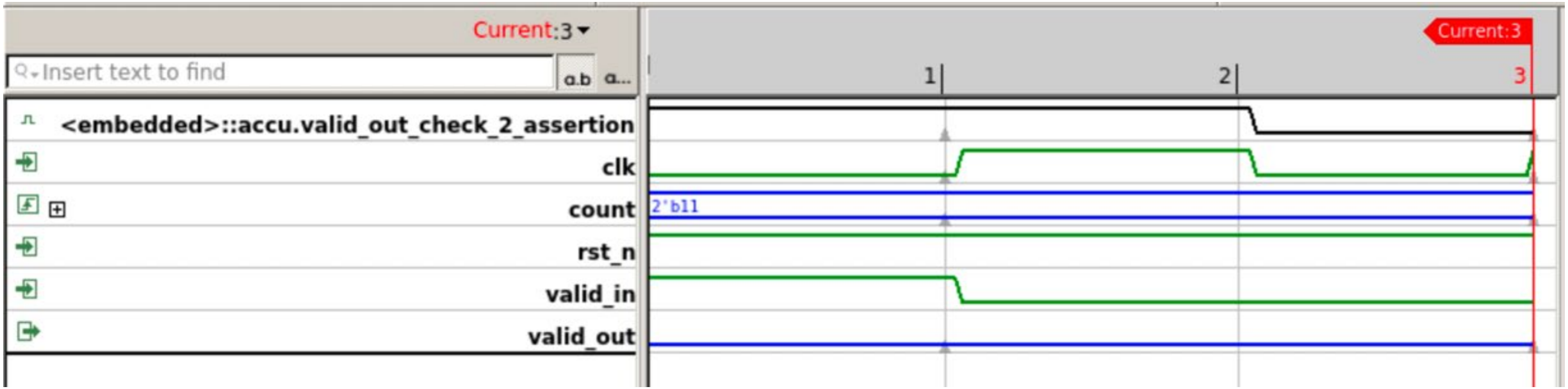
```
1   `timescale 1ns/1ns
2
3   module accu(
4       input               clk         ,
5       input               rst_n       ,
6       input       [7:0]   data_in     ,
7       input               valid_in     ,
8
9       output  reg         valid_out     ,
10      output  reg [9:0]   data_out
11  );
12
13     reg [1:0] count;
14     wire add_cnt;
15     wire ready_add;
16     wire end_cnt;
17     reg [9:0]   data_out_reg;
18
19     assign add_cnt = ready_add;
20     assign end_cnt = ready_add && (count == 'd3);
21
22     //count
```

```
96      property valid_out_check_2;
97          @(posedge clk)
98      disable iff(!rst_n)
99      (count == 3 && valid_in) |=> valid_out == 1;
100     endproperty
101
102     valid_out_check_2_assertion:assert property(valid_out_check_2)
103     else $error("vaild_out should be high when vaild_in high and count=3 ");
104
105
106 endmodule
```

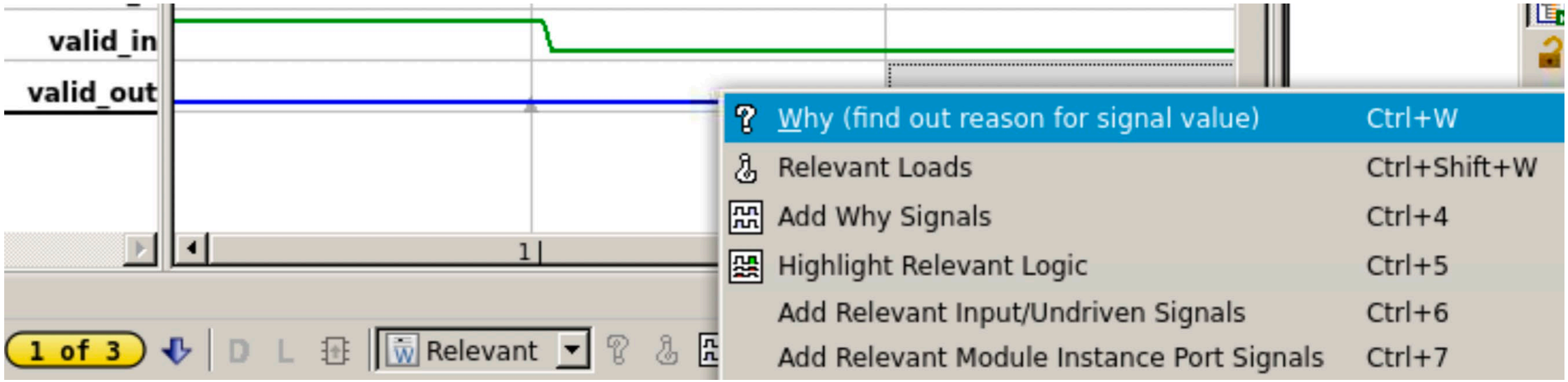
Example Workflow: Manual Debugging



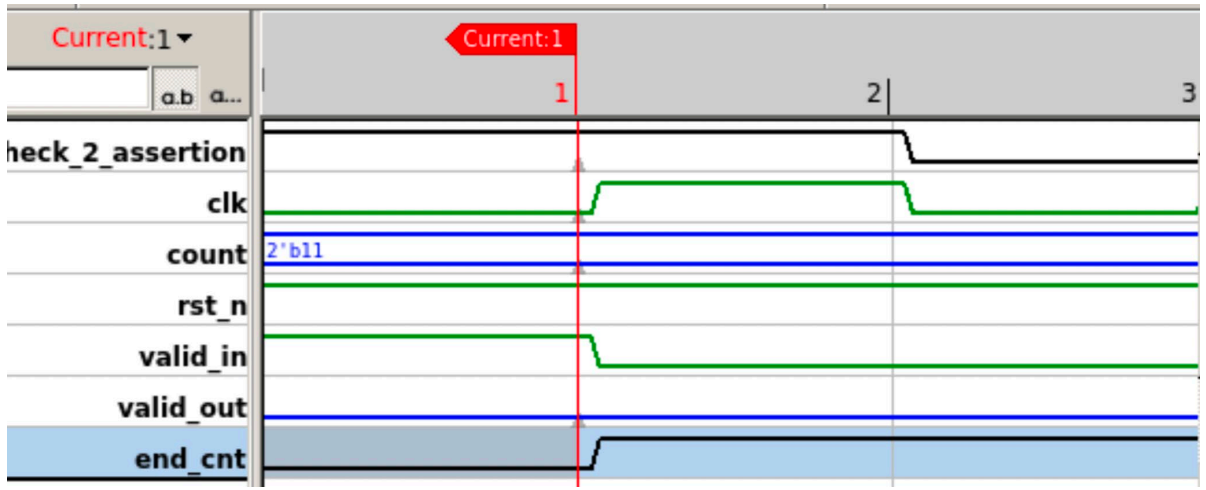
Example Input: Accumulator Design



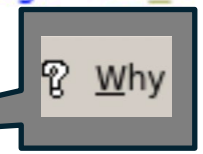
Example Workflow: Manual Debugging



Example Workflow: Manual Debugging



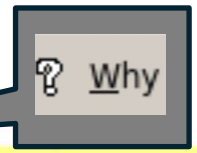

```
64      //valid_out
65      always @(posedge clk or negedge rst_n) begin
66          if(!rst_n) begin
67              valid_out <= 0;
68          end
69          else if(end_cnt) begin
                    1'b0
70              valid_out <= 1;
71          end
72          else begin
73              valid_out <= 0;
                             1'b0
74          end
75      end
76
```



≠Ω¨ –»ÀΔΔ‘

Example Workflow: Manual Debugging

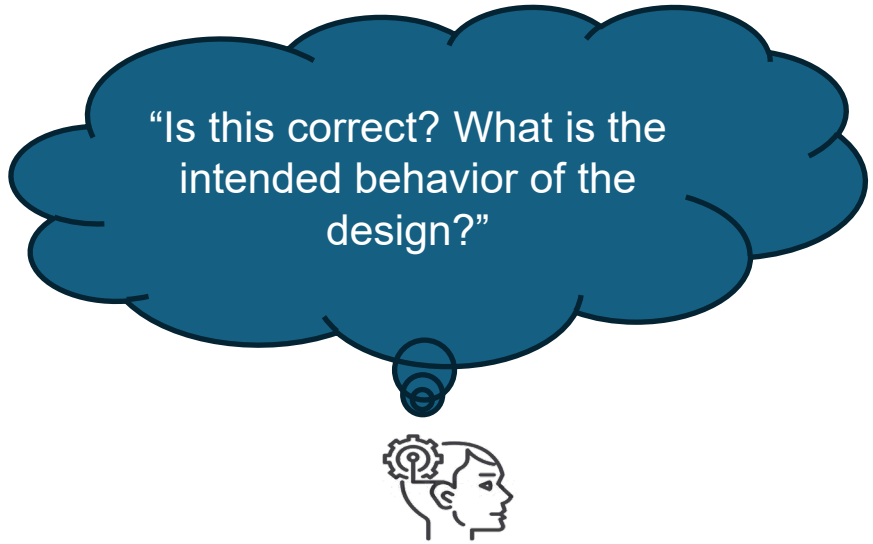
```
14      wire add_cnt;
15      wire ready_add;
16      wire end_cnt;
17      reg [9:0]   data_out_reg;
18
19      assign add_cnt = ready_add;
20      assign end_cnt = ready_add && (count == 'd3);
                         1'b0
21
```



FV Engineer

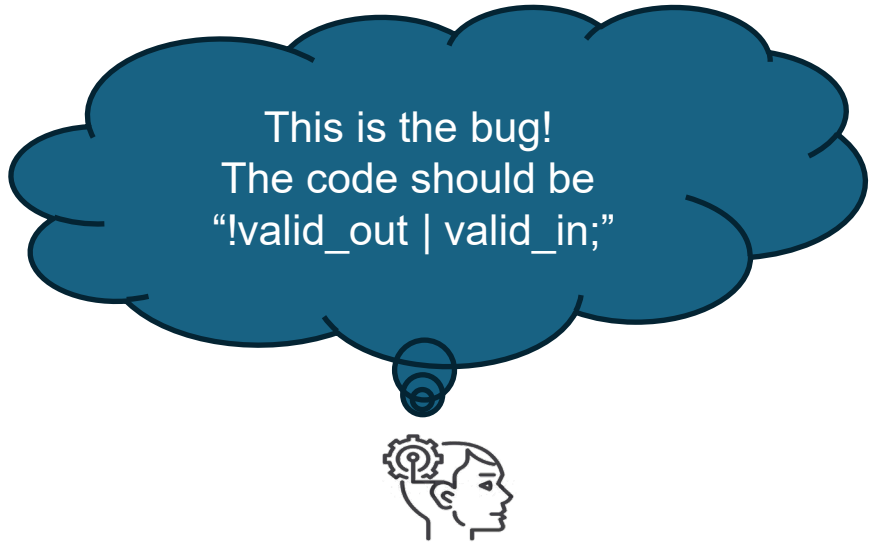
Example Workflow: Manual Debugging

```
56          else if (add_cnt) begin
57              data_out <= data_out + data_in;
58          end
59      end
60
61      //ready_add
62      assign ready_add = valid_out | !valid_in;
                           1'b0          1'b1
63
64      //valid_out
65      always @(posedge clk or negedge rst_n) begin
66          if(!rst_n) begin
67              valid_out <= 0;
68          end
69          else if(end_cnt) begin
70              valid_out <= 1;
```



Example Workflow: Manual Debugging

```
56          else if (add_cnt) begin
57              data_out <= data_out + data_in;
58          end
59      end
60
61      //ready_add
62      assign ready_add = valid_out | !valid_in;
                           1'b0          1'b1
63
64      //valid_out
65      always @(posedge clk or negedge rst_n) begin
66          if(!rst_n) begin
67              valid_out <= 0;
68          end
69          else if(end_cnt) begin
70              valid_out <= 1;
```



Example Output of FVDebug

Buggy Line
```
assign ready_add = valid_out | !valid_in;
```
Fixed Line
```
assign ready_add = !valid_out | valid_in;
```

Golden Answer

🎯 **Ranked Hypotheses**

Hypothesis 1: Ready_add Logic Issue Preventing valid_out Assertion

FVDebug Output

Confidence: 48.0%

[1] **Hypothesis**

The valid_out signal remains low because the ready_add condition (`ready_add = valid_out OR !valid_in`) does not become true when count reaches 3, causing end_cnt to never trigger a valid_out high; this indicates an RTL bug where the handshake logic for ready_add is flawed.

[2] **Evidence**

Supporting Evidence:

- The RTL code `'assign end_cnt = ready_add && (count == 'd3)'` implies that even if count reaches 3, end_cnt depends on ready_add being true. This supports the hypothesis if ready_add remains low, preventing valid_out from becoming high.

...

🌳 **Causal Chain Timeline**

...

3. **Cycle 1** 🔴

At cycle 1, with valid_in high, the node 'ready_add' evaluates to 1'b0 because it is computed as `valid_out OR !valid_in`. Since valid_in is high, !valid_in is false and valid_out is low, which blocks accumulation and consequently prevents the triggering of end_cnt and the high assertion of valid_out

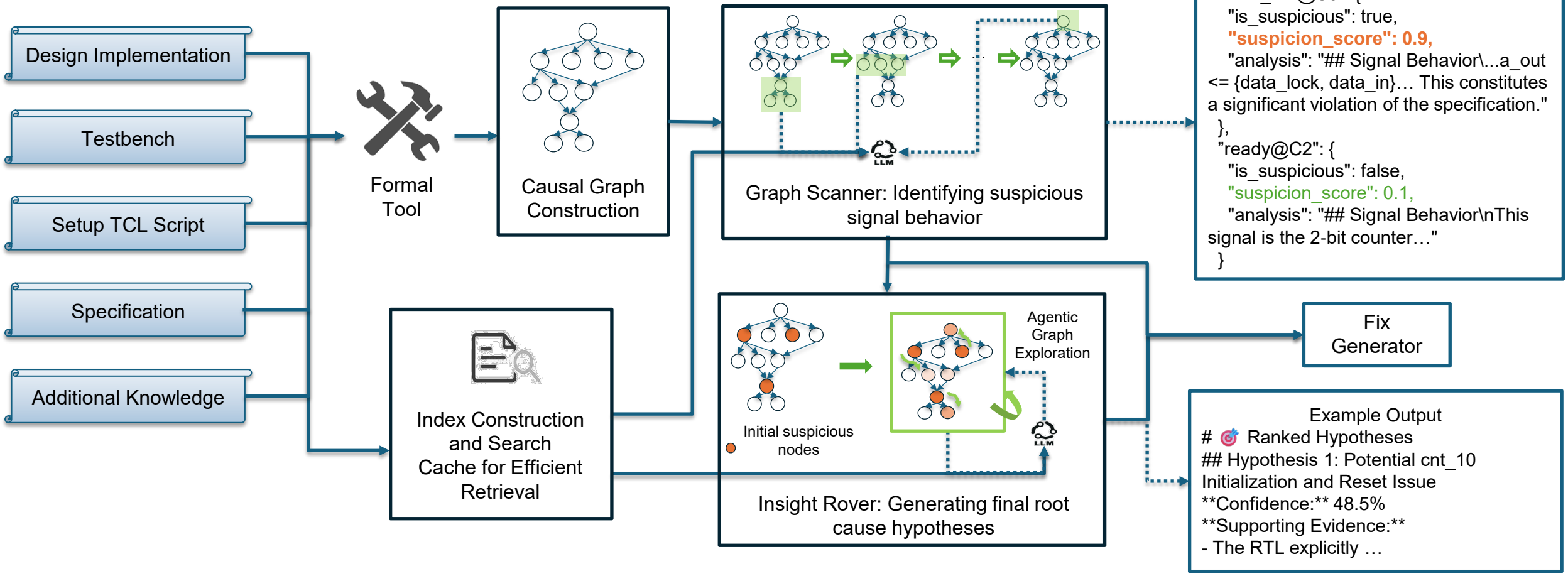
- Most important section: Ranked Hypotheses
  - Hypothesis statement
  - Supporting evidence
- Useful for understanding: Casual Chain timeline
  - Contains key events in the wave explaining the failure
- In addition: Suggested Fix

Prior Work

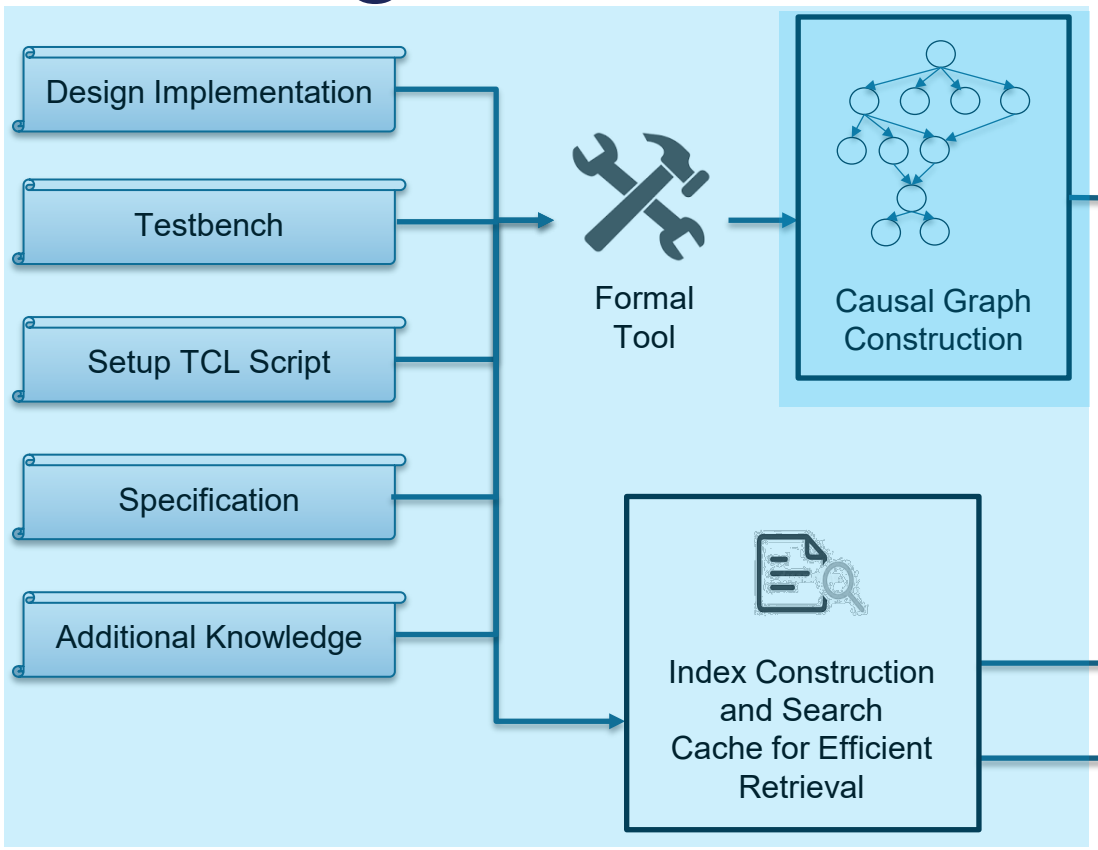
FVDebug: First end-to-end realistic debugger automating CEX to **Root Cause & Fix**

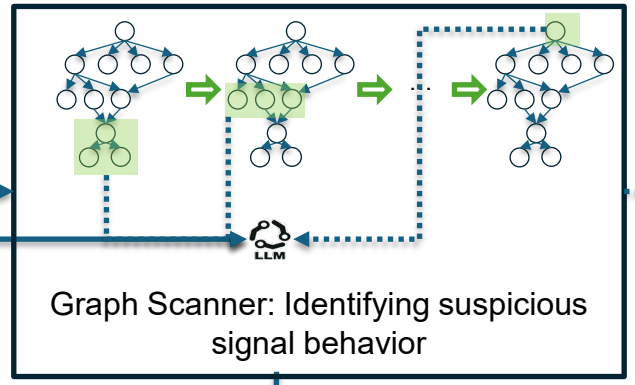
Category	Representative Papers / Tools	Key Gap FVDebug Addresses
Code-only Debugging (no waveforms)	● LLM-HDL (’25) ● VeriDebug (ICLAD ’25) ● HLSDebugger (ICCAD ’25)	No temporal reasoning; miss execution -only failures
Trace-aware LLM Debugging (waveforms in prompt)	● AssertSolver (DAC ’25) ● GenAI-Induction (SOCC ’24) ● SAARTHI (arXiv ’25)	Treat trace as flat text or images; limited multi -cycle causal reasoning
Industrial / Commercial Solutions	● Cadence Jasper, Synopsys Verdi, Cadence Indago ● Start -ups ChipAgents , ChipStack ● General copilots Claude Code, Cursor	Great viewers; automated root -cause reasoning still done by engineers No open-source implementations Customization needed

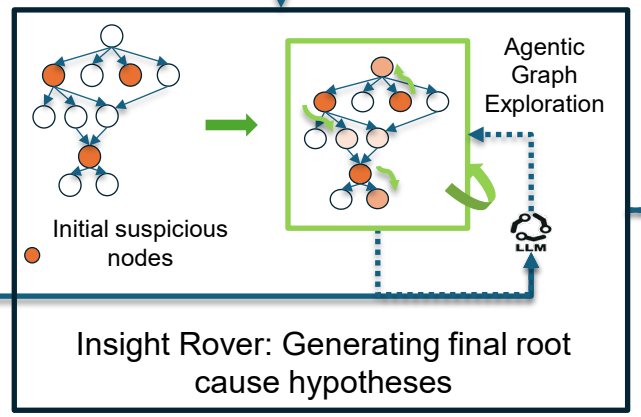
Method Overview



Stage 1: Causal Graph Synthesis







Example Output

```
"data_out@C3": {
  "is_suspicious": true,
  "suspicion_score": 0.9,
  "analysis": "## Signal Behavior\...a_out
<= {data_lock, data_in}… This constitutes
a significant violation of the specification."
 },
 "ready@C2": {
  "is_suspicious": false,
  "suspicion_score": 0.1,
  "analysis": "## Signal Behavior\nThis
signal is the 2-bit counter…"
 }
```

Fix Generator

Example Output

```
# 🎯 Ranked Hypotheses
## Hypothesis 1: Potential cnt_10
Initialization and Reset Issue
**Confidence:** 48.5%
**Supporting Evidence:**
- The RTL explicitly …
```

Causal Graph Construction

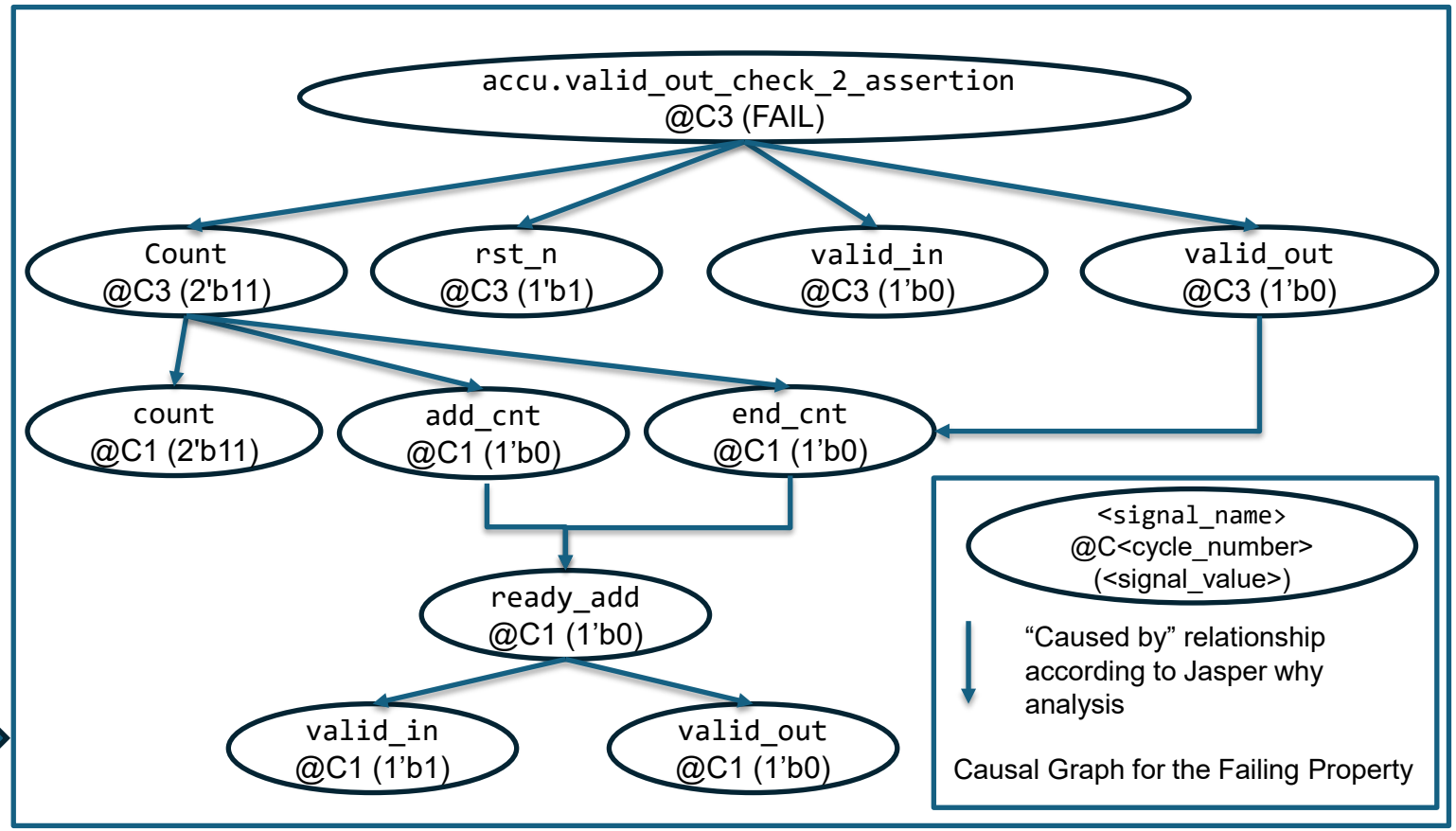
RTL

```
module accu(
    input               clk         ,
    input               rst_n       ,
    input       [7:0]   data_in     ,
    input               valid_in     ,
    output  reg         valid_out     ,
    output  reg [9:0]   data_out
);
```

Failing property with counter-example (CEX):

```
(count == 3 && valid_in) |=>
valid_out == 1;
```

Batch mode via TCL script recursively caling `visualize -why`, `visualize -get_value`, etc.



Causal Graph Construction

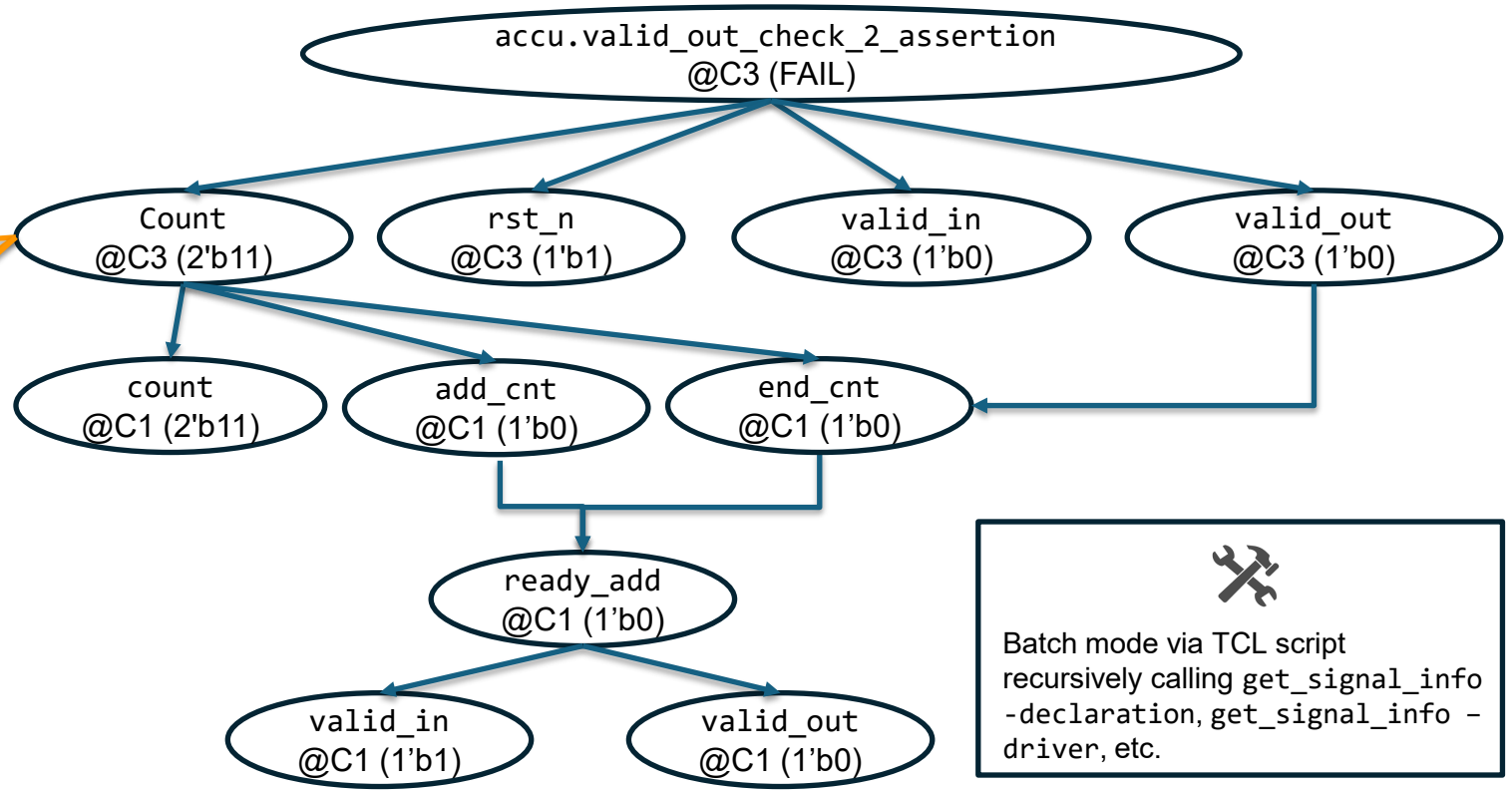
```
module accu(
    input               clk         ,
    input               rst_n       ,
    input       [7:0]   data_in     ,
    input               valid_in     ,
    output  reg         valid_out     ,
    output  reg [9:0]   data_out
);
   reg [1:0] count;
   wire add_cnt;
   wire ready_add;
   wire end_cnt;
   reg [9:0]   data_out_reg;

   assign add_cnt = ready_add;
   assign end_cnt = ready_add && (count == 'd3);

   //count
   always @(posedge clk or negedge rst_n) begin
       if(!rst_n) begin
          count <= 0;
       end
       else if(end_cnt) begin
          count <= 0;
```

Declaration Location

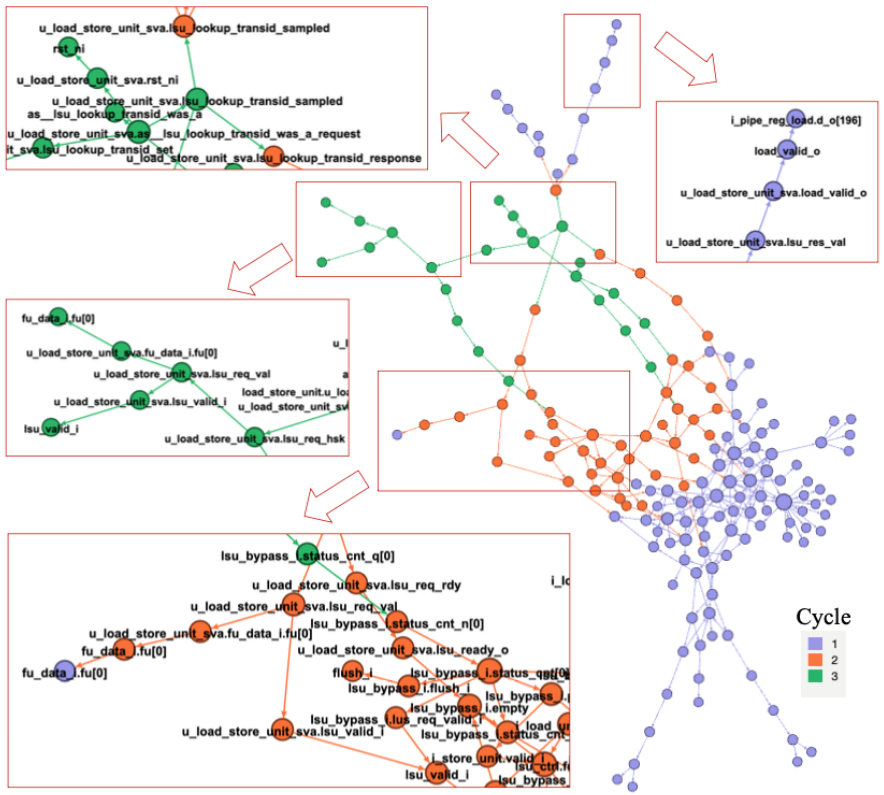
Driver Locations



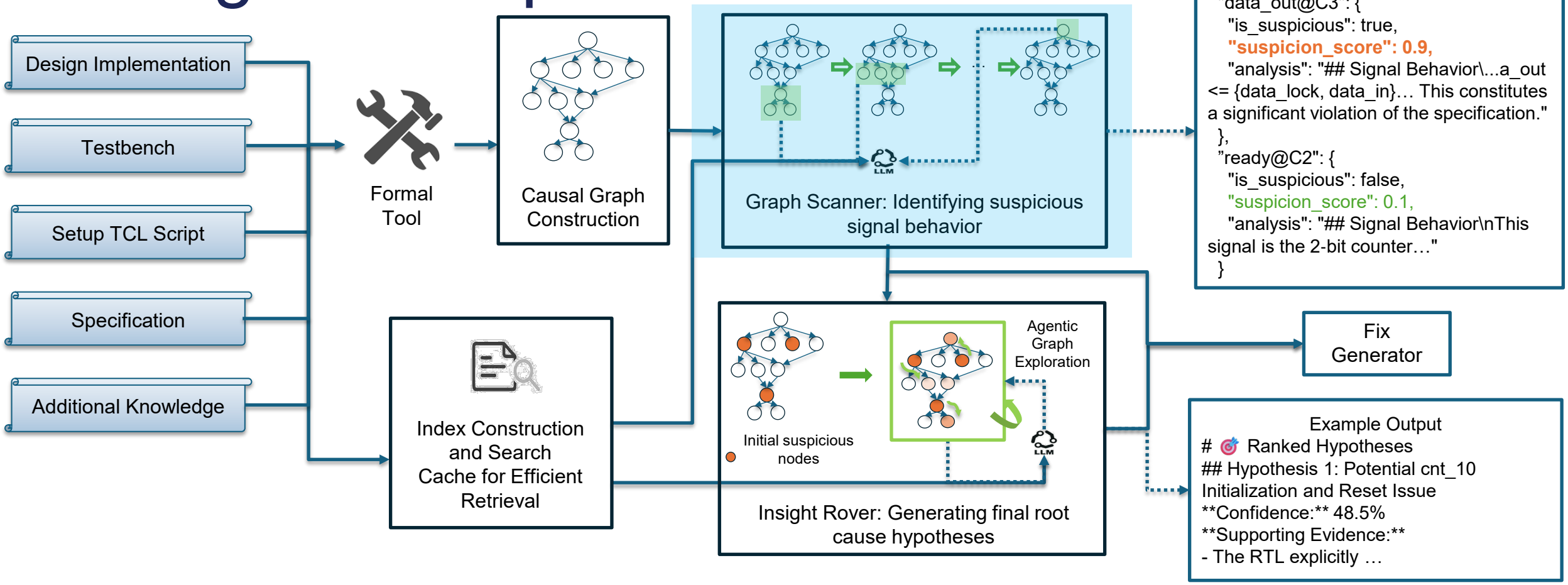
Batch mode via TCL script recursively calling `get_signal_info -declaration`, `get_signal_info -driver`, etc.

Causal Graph: A Realistic Example

```
as__lsu_lookup_transid_was_a_request (C:3, V:FAIL)
|-- lsu_lookup_transid_response (C:3, V:1'b1)
|   |-- lsu_res_hsk (C:3, V:1'b1)
|   |   `-- load_valid_o (C:3, V:1'b1)
|   |       `-- i_pipe_reg_load.d_i[196] (C:2, V:1'b1)
|   |           |-- i_load_unit.ex_i.valid (C:2, V:1'b1)
|   |           |   |-- i_mmu.misaligned_ex_q.valid (C:2, V:1'b1)
|   |           |   |   `-- i_mmu.misaligned_ex_n.valid (C:1, V:1'b1)
|   |           |   |       |-- i_mmu.lsu_req_i (C:1, V:1'b1)
|   |           |   |       |   |-- ld_translation_req (C:1, V:1'b1)
|   |           |   |       |   `-- lsu_ctrl.fu (C:1, V:LOAD)
|   |           |   |       `-- i_mmu.misaligned_ex_i.valid (C:1, V:1'b1)
|   |           |   |           |-- data_misaligned (C:1, V:1'b1)
|   |           |   |           |   |-- lsu_ctrl.operator (C:1, V:SD)
|   |           |   |           |   `-- lsu_ctrl.vaddr[2] (C:1, V:1'b1)
|   |           |   |           `-- lsu_ctrl.overflow (C:1, V:1'b0)
|   |           |   `-- i_mmu.pmp_data_allow (C:2, V:1'b1)
|   |           `-- i_load_unit.state_q[1:0] (C:2, V:2'b00)
|   `-- lsu_res_transid (C:3, V:3'b000)
|       `-- load_trans_id_o (C:3, V:3'b000)
|           `-- i_load_unit.load_data_q.trans_id (C:2, V:3'b000)
|-- lsu_lookup_transid_sampled (C:3, V:4'b0000)
|   |-- lsu_lookup_transid_response (C:2, V:1'b0)
|   |-- lsu_lookup_transid_sampled (C:2, V:4'b0000)
|   `-- lsu_lookup_transid_set (C:2, V:1'b0)
|-- lsu_lookup_transid_set (C:3, V:1'b0)
|   `-- lsu_req_hsk (C:3, V:1'b0)
|       |-- lsu_req_rdy (C:3, V:1'b0)
|       `-- lsu_req_val (C:3, V:1'b0)
|       |-- u_load_store_unit_sva.fu_data_i.fu[0] (C:3, V:1'b0)
|       |   `-- fu_data_i.fu[0] (C:3, V:1'b0)
|       `-- u_load_store_unit_sva.lsu_valid_i (C:3, V:1'b0)
|           `-- lsu_valid_i (C:3, V:1'b0)
`-- u_load_store_unit_sva.rst_ni (C:3, V:1'b1)
    `-- rst_ni (C:3, V:1'b1)
```



Stage 2: Graph Scanner



Stage 2: Graph Scanner

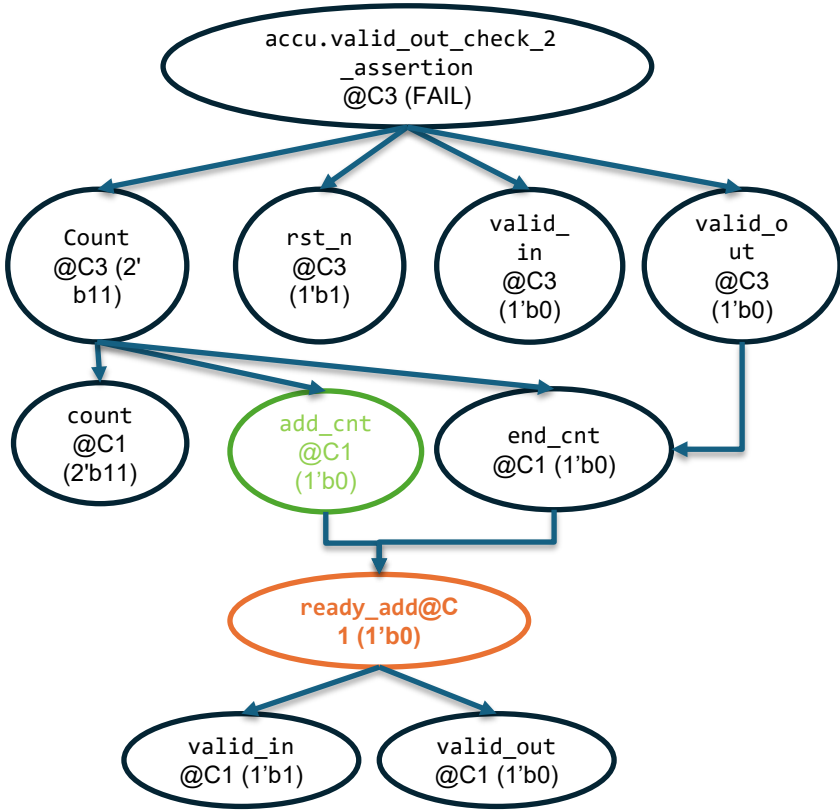
- **Breadth-First Traversal**
  - Dynamic node batching for efficient processing
  - Context-aware evaluation using parent node results
  - RTL + specification retrieval per node
- **For-and-Against Prompting**
  - LLM generates balanced arguments (Evidence vs. Normality)
  - Reduces hallucinations and bias
- **Outcome:** Ranked list of suspicious nodes with confidence scores

Example

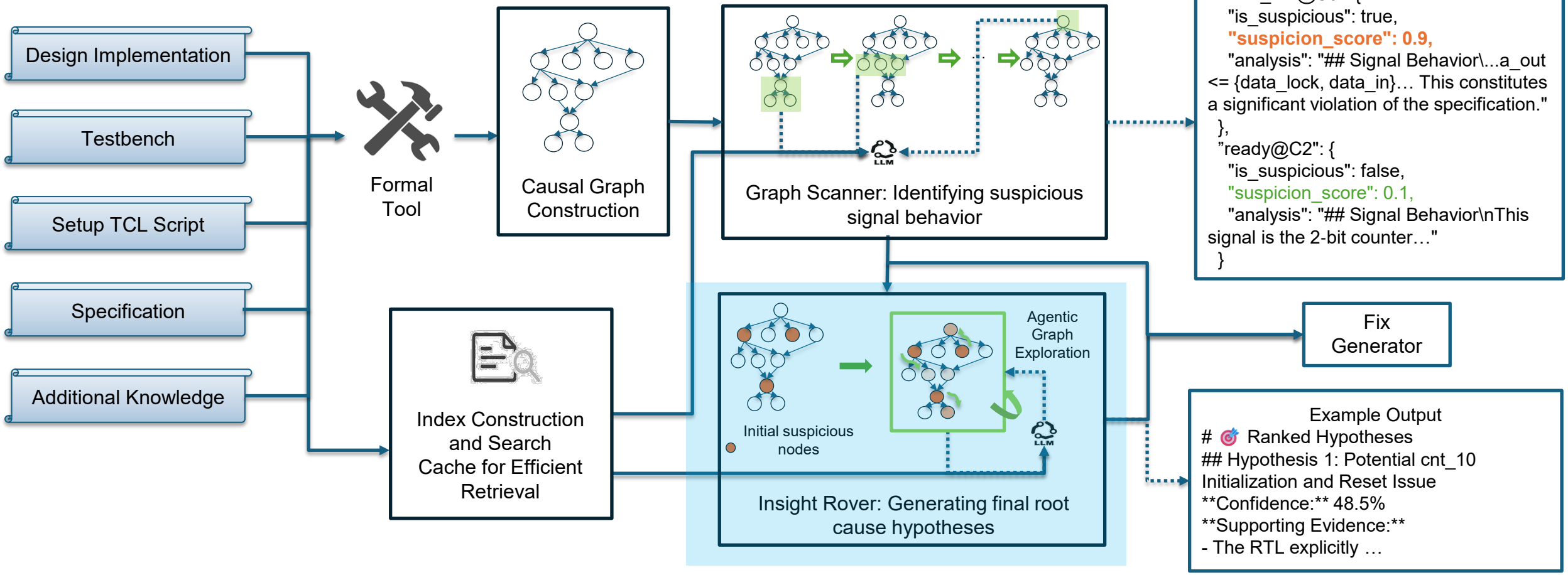
- [ready_add@C1] Conclusion: The computed value of ready_add at cycle 1 (0) may be acceptable during an initialization phase; however, **its derivation using !valid_in is a departure from the specified accumulation trigger** and remains moderately suspicious.
- [ready_add@C1] ⚠ SUSPICIOUS: Suspicion score 0.50

…

- [add_cnt@C1] NOT SUSPICIOUS (suspicion score: 0.10)



Stage 3: Insight Rover



Stage 3: Insight Rover

- **Hypothesis Generation**
  - Suspicious nodes as "seeds" for exploration
  - Instantiates multiple competing causal narratives
- **Beam-Style Agentic Search**
  - LLM-guided graph exploration from the seeds
  - Gathers evidence and builds a temporal event timeline
- **Iterative Refinement**
  - Dynamic confidence scoring for each hypothesis
  - Converges when one hypothesis reaches high confidence
- **Outcome:** Human-readable causal explanations

FVDebug Output

🎯 **Ranked Hypotheses**

Hypothesis 1: Ready_add Logic Issue Preventing valid_out Assertion

Confidence: 48.0%

[1] **Hypothesis**

The valid_out signal remains low because the ready_add condition (`ready_add = valid_out OR !valid_in`) does not become true when count reaches 3, causing end_cnt to never trigger a valid_out high; this indicates an RTL bug where the handshake logic for ready_add is flawed.

[2] **Evidence**

Supporting Evidence:

- The RTL code `'assign end_cnt = ready_add && (count == 'd3)'` implies that even if count reaches 3, end_cnt depends on ready_add being true. This supports the hypothesis if ready_add remains low, preventing valid_out from becoming high.

...

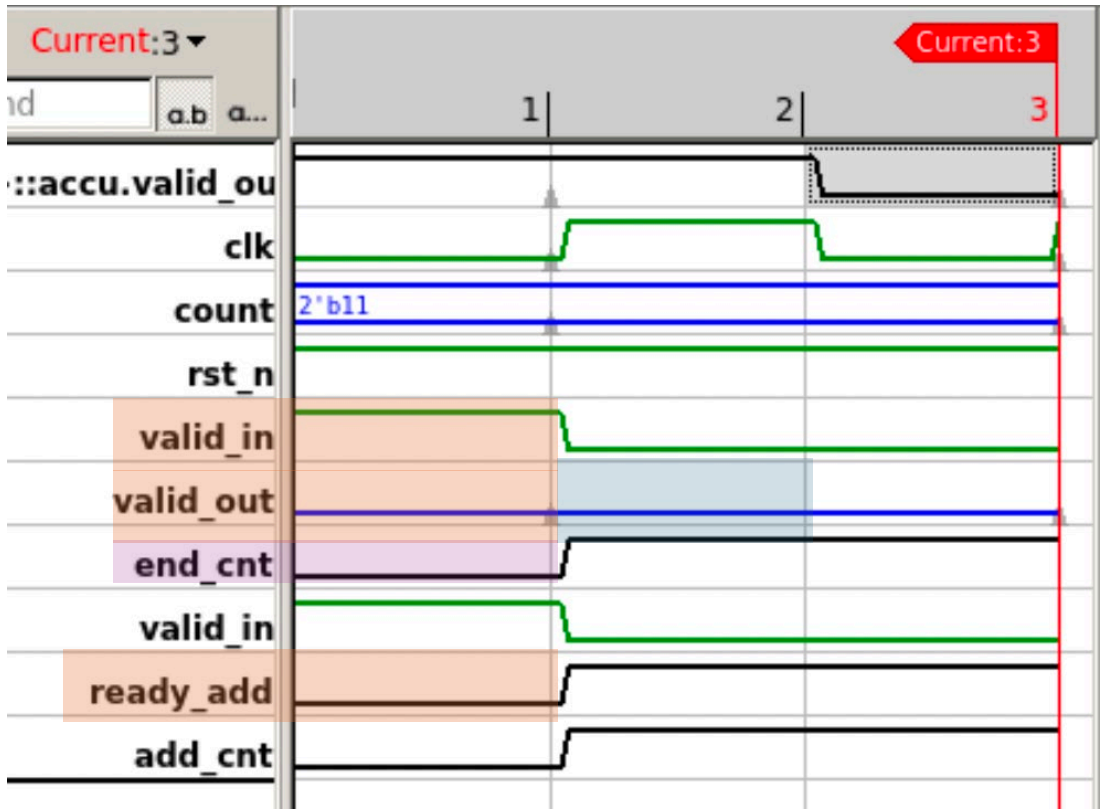
🌳 **Causal Chain Timeline**

...

3. **Cycle 1** 🔴

At cycle 1, with valid_in high, the node 'ready_add' evaluates to 1'b0 because it is computed as `valid_out OR !valid_in`. Since valid_in is high, !valid_in is false and valid_out is low, which blocks accumulation and consequently prevents the triggering of end_cnt and the high assertion of valid_out

Stage 3: Insight Rover Example Output



🌳 **Causal Chain Timeline**

Cycle 1 🔴
At cycle 1, with valid_in high, the node 'ready_add' evaluates to 1'b0 because it is computed as valid_out OR !valid_in.

Cycle 1
ready_add = 1'b0 (suspicious)

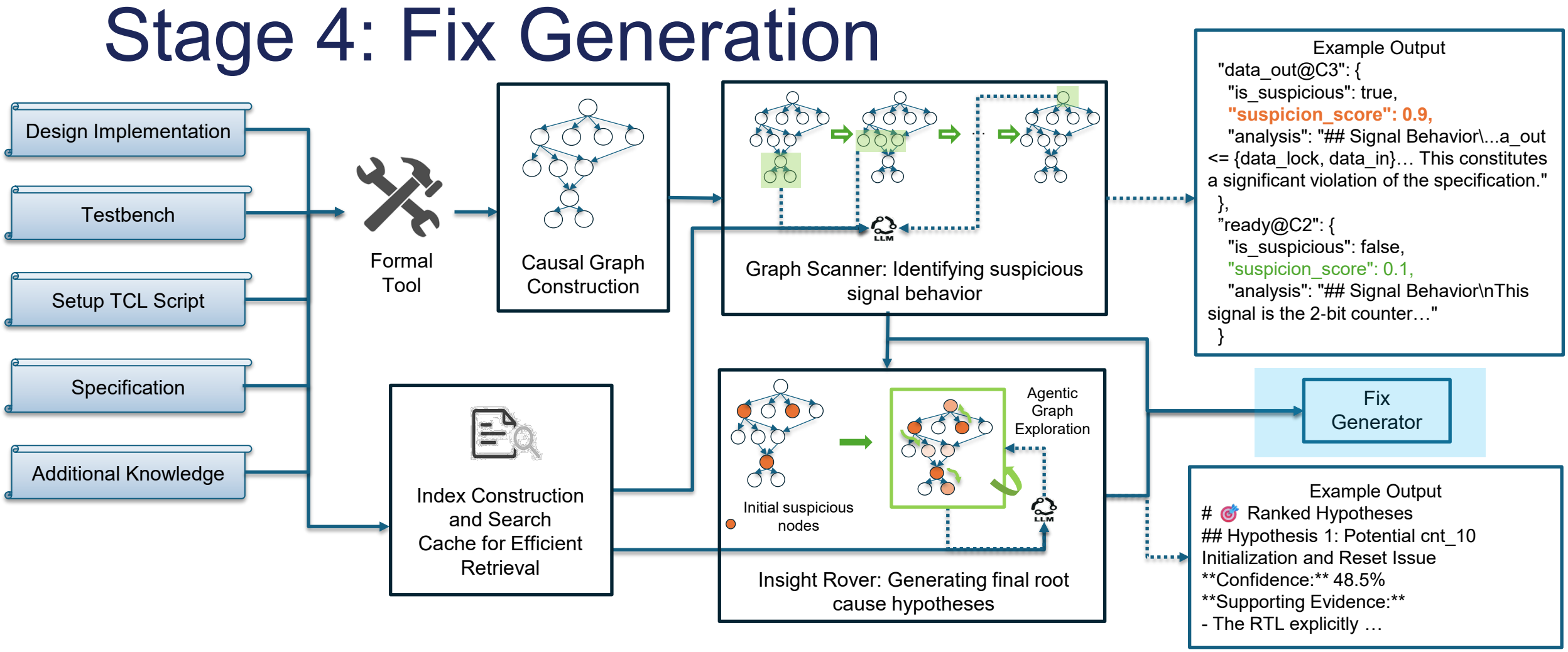
Cycle 1
end_cnt = ready_add && (count==3) stays 0 → valid_out never gets asserted at the next posedge.

Cycle 1 → Cycle 2
The assertion valid_out_check_2 triggers: (count == 3 && valid_in) is TRUE at cycle 1. The |=> operator requires valid_out == 1 at the next cycle.

Cycle 2 🔴 (Assertion Failure)
The assertion expects valid_out == 1 this cycle. But valid_out is still 0 — **assertion violated**.

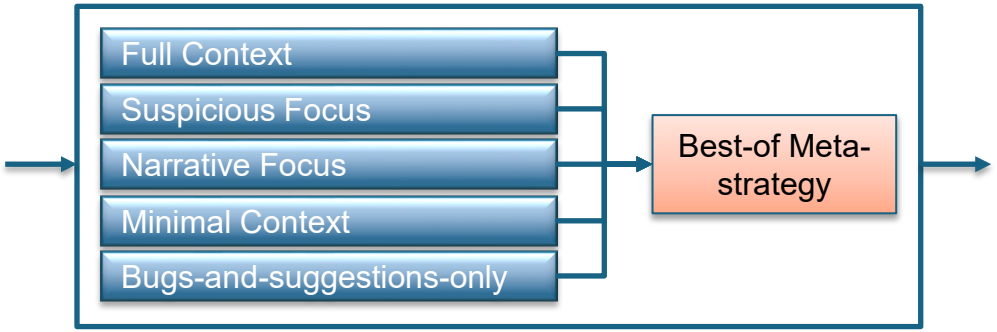
Time

Stage 4: Fix Generation



Stage 4: Fix Generation

- **Ensemble Fix Strategy**
  - Generates concrete RTL code fixes
  - Multiple prompting strategies → best-of meta-selection
  - Inspired by best-of-N sampling & self-consistency
- If fix fails re-verification → try next-ranked suggestion
- **Outcome:** RTL-level code fixes



Example

Buggy Code:

```verilog
assign ready_add = valid_out | !valid_in;
```

Fixed Code:

```verilog
assign ready_add = valid_in && !valid_out;
```

Location: `rtl_content_1.sv:63` in module `accu`

```
`timescale 1ns/1ns

module accu(
    input               clk         ,
    input               rst_n       ,
    input       [7:0]   data_in     ,
    input               valid_in     ,

    output  reg         valid_out     ,
    output  reg [9:0]   data_out
);

   reg [1:0] count;
   wire add_cnt;
   wire ready_add;
   wire end_cnt;

…

 //ready_add
  assign ready_add = valid_out |
!valid_in;

   //valid_out
   always @(posedge clk or negedge rst_n)
begin
```

Results on SVA-Eval-Human

Method	Root Cause Hypothesis Quality				Fix Generation	
	Quality@Best	NDCG@5	MRR	Kendall's τ	Pass@1	Pass@5
Unstructured Baselines						
Direct LLM	0.783	0.981	0.806	0.526	0.605	0.658
Flat Trace Analysis	0.474	0.948	0.804	0.511	0.632	0.816
Component Ablations						
FVDebug w/o For-Against	0.953	0.969	0.817	0.634	0.632	**0.868**
FVDebug w/o Rover	0.795	0.844	0.567	-0.176	0.643	0.821
FVDebug w/o Ensemble	**0.956**	**0.983**	**0.858**	**0.708**	0.684	0.763
FVDebug (Full)	**0.956**	**0.983**	**0.858**	**0.708**	**0.711**	**0.868**

- 95.6% hypothesis quality for root cause identification
- 71.1% Pass@1 fix rate (first attempt success)
- 86.8% Pass@5 fix rate (within top 5 suggestions)
- **Key Insights:**
  - Causal structure is essential
  - Narrative construction in Insight Rover drives performance
  - Ensemble strategies improve robustness

ò ÒΦ→!ΦÍ d’9Í Ď ű ¯Ò╒ï Ż Ê ÒΦΦ Ż? ÒΦpŽd’Φ

Method	Hypothesis Quality Metrics			
	Quality@Best	NDCG@5	MRR	Kendall's τ
Unstructured Baselines				
Direct LLM	0.575	0.801	0.500	0.333
Flat Trace Analysis	0.475	0.800	0.531	0.100
Component Ablations				
FVDebug w/o For-Against	0.644	0.791	0.531	0.306
FVDebug w/o Rover	0.300	0.742	0.276	0.294
FVDebug (Full)	**0.713**	**0.875**	**0.667**	**0.371**

Design	RTL Files	Lines of Code	Graph Nodes	Graph Edges	Unique Signals
CVA6 MMU	3	695	172	235	76
CVA6 LSU	7	1,918	170	239	122

- **Challenging Scenarios:**
  - *MMU Ghost Response:* Misaligned requests trigger duplicate exception responses.
  - *LSU Transaction ID Mismatch:* Simultaneous exceptions and cache responses cause untracked transaction IDs.
- Maintains strong performance despite:
  - Multi-module processor designs
  - Complex signal dependencies (100+ unique signals)
- Demonstrates scalability for multi-module processor debugging

Results on Industrial CEXs

- Evaluated on two proprietary, production-scale FV CEXs with over **500,000 lines of code**.
  - **Case 1: FV Testbench Error**
    - **Problem:** Incorrect internal token counter.
    - **Result:** "Excellent Match." FVDebug correctly identified the missing signal in the counter logic.
  - **Case 2: Missing Constraint**
    - **Problem:** FIFO overflow caused by a missing input constraint.
    - **Result:** "Strong Match." FVDebug identified the problematic input signal and recommended the correct solution.
- FVDebug pinpoints root causes in **minutes**, demonstrating its capability to accelerate debugging in production-scale designs.

Case	RTL Files	Lines of Code	Unique Signals	Graph Nodes	Graph Edges	Graph Depth	Jasper Calls	LLM Calls	Total Runtime
CEX-1 (Testbench Error)	265	560,202	123	189	227	15	163	25	4m 12s
CEX-2 (Missing Constraint)	257	554,983	41	80	86	11	67	21	6m 07s

ïÍ ФфÂÒU→!→ŻÒª ÍŻČ

- Extension to simulation-based dynamic verification and other formal engines
  - E.g., DV tracing via **Verdi**; other engines like **VC Formal**
- Scaling up FVDebug with massively parallel executions of graph scanning and insight roving
  - To further improve the speed and accuracy
- Customized model training for root cause analysis
  - Existing LLMs make mistakes in “thinking” and tracing may not provide enough information to correct the LLMs.

Thank you!

- Questions?
- Contact: yunshengb@nvidia.com