Functional Verification: RTL2MμPATH (Overview)

Caroline

Stanford | ENGINEERING

 RTL2MµPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

Overview: Synthesizing formally verified microarchitectural execution paths (µPATHs) from SystemVerilog RTL

Design Under Verification (DUV): SystemVerilog RTL

Design Metadata:
- All instruction encodings
- µFSM signal names
- Instruction fetch register
- Commit signal

(designer-friendly)

→ **RTL2MµPATH**

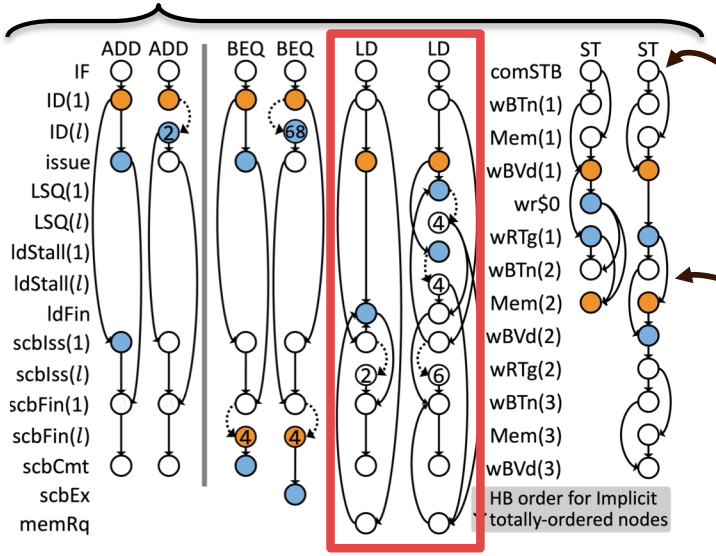
Can also uncover using HW event counters in a closed-box manner

CounterPoint • [Lindsay+, ASPLOS '26, **Best Paper Award**]

→ All reachable µPATHs per inst

→ **~Functional Contract**

Node: Specific **inst** (ST) inducing a specific **µarch event** (updating comSTB) in a **specific cycle**.

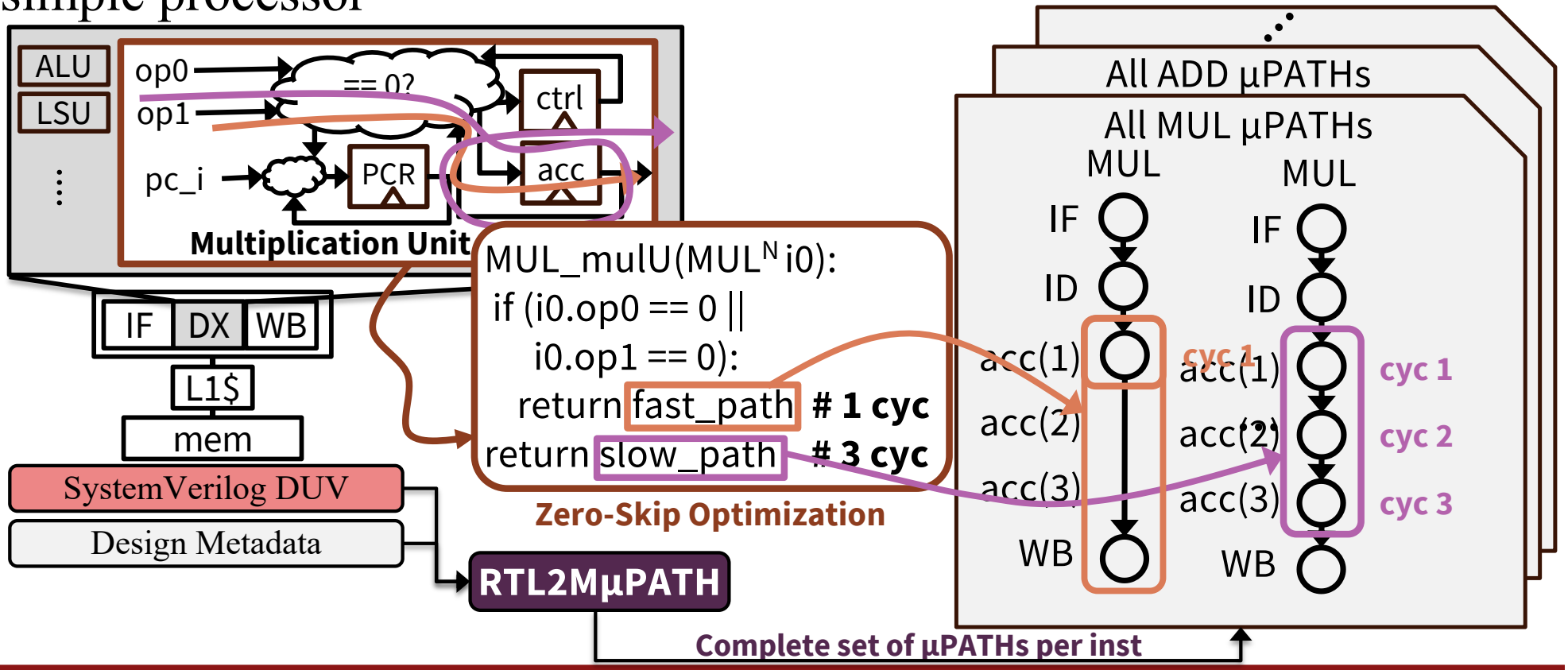


Edge: **One-cycle** happens-before relationship

Cycle-accurate (or **logical**) microarchitectural happens-before (µHB) graphs [Lustig+, MICRO'14]

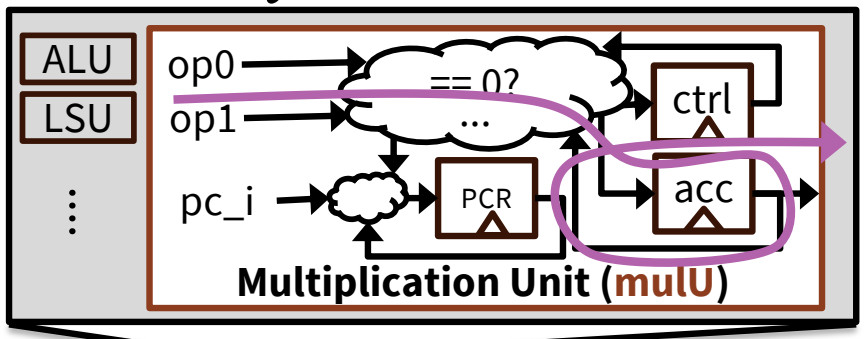
RTL2MμPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

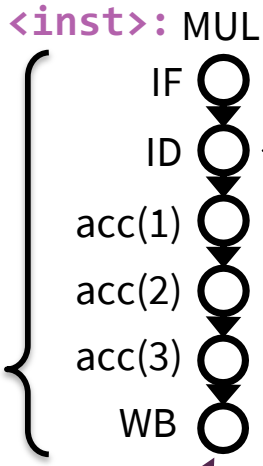
Example: Synthesizing formally verified MUL μPATHs from a simple processor



RTL2MμPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

Key Challenge: Encoding a μhb node—as a set of state updates induced by some instruction—as a function of DUV signals is hard





- **Encoding edges?** Easy with SystemVerilog Assertions (SVAs), ##1
- **Encoding nodes?** Hard with SVAs requires **detecting** & **attributing** state updates to a specific inst

RTL2MμPATH

Ask a model checker...

property template

```
P: Can <inst> exhibit
<μPATH> in any execution?
```

cover (P)

for example...

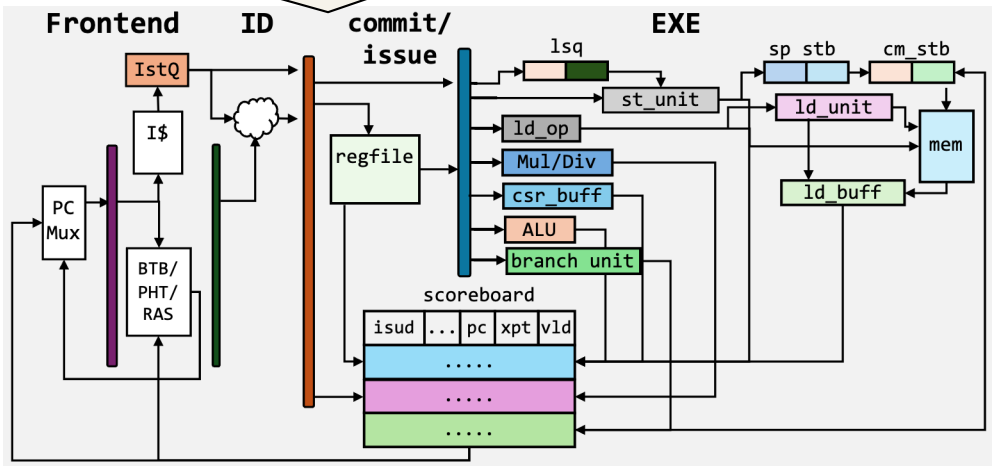
All reachable MUL μPATHs

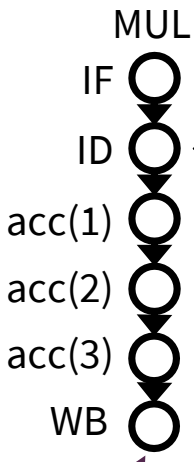
Discard unreachable μPATHs

RTL2MµPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

Key Challenge: Encoding a µhb node—as a set of state updates induced by some instruction—as a function of DUV signals is hard

Why? Many instructions (colors) **in-flight simultaneously**!





- **Encoding edges?** Easy with SystemVerilog Assertions (SVAs), ##1
- **Encoding nodes?** Hard with SVAs requires **detecting** & **attributing** state updates to a specific inst

SystemVerilog DUV

Design Metadata

RTL2MµPATH

Ask a model checker...

property template

P: Can <inst> exhibit <µPATH> in any execution?

cover (P)

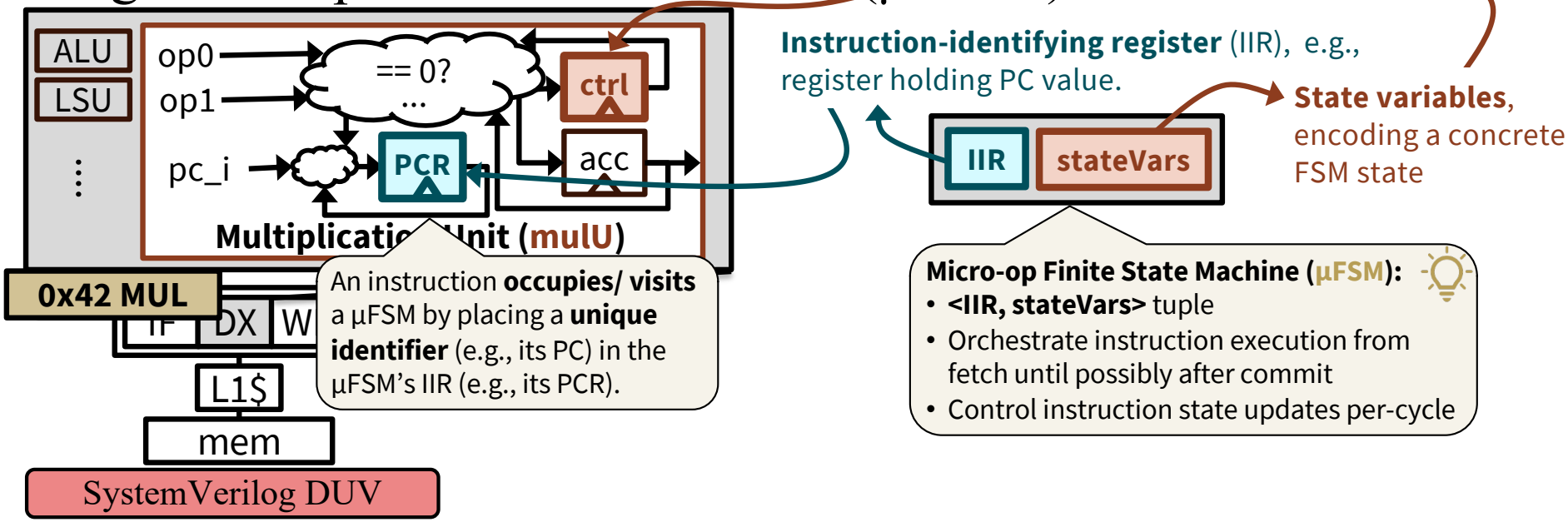
for example...

All reachable MUL µPATHs

Discard unreachable µPATHs

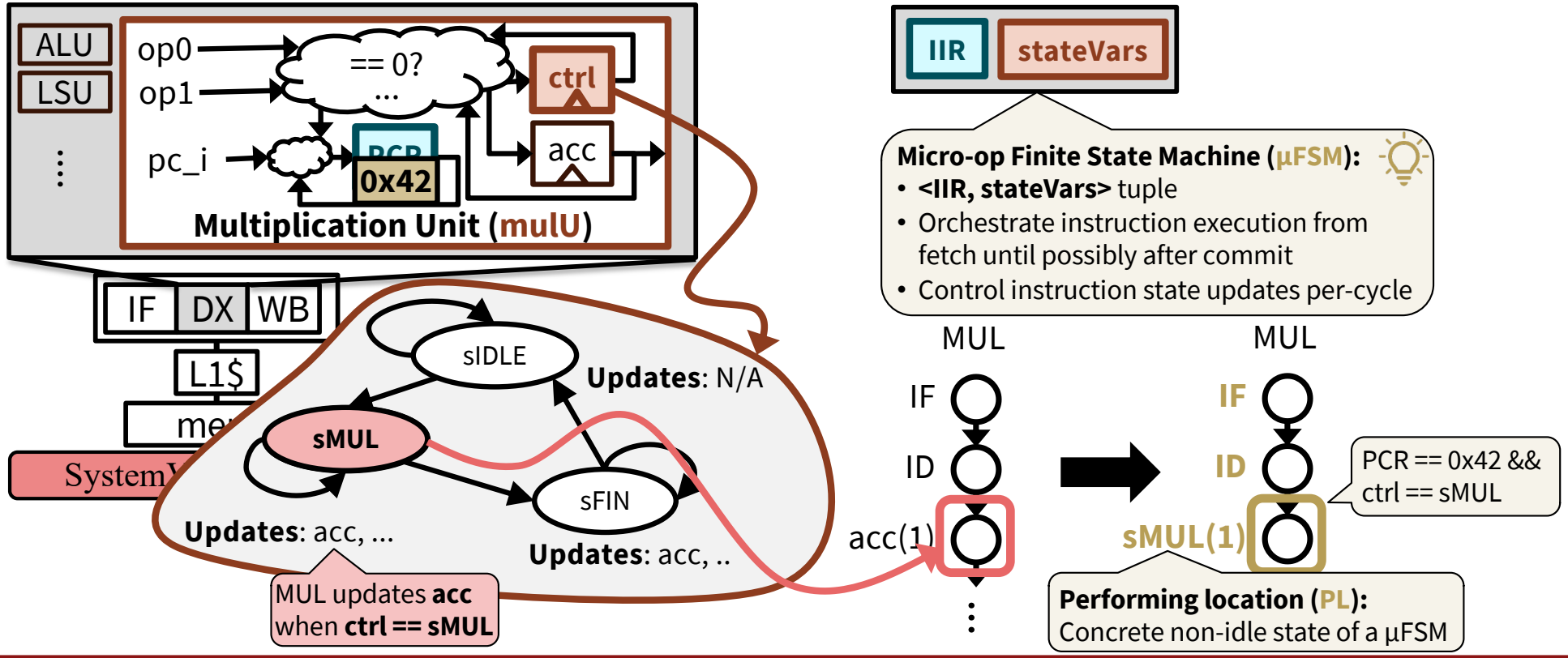
RTL2MμPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

Our Solution: Encoding μhb nodes as performing locations (PLs) using micro-op finite state machines (μFSMs)



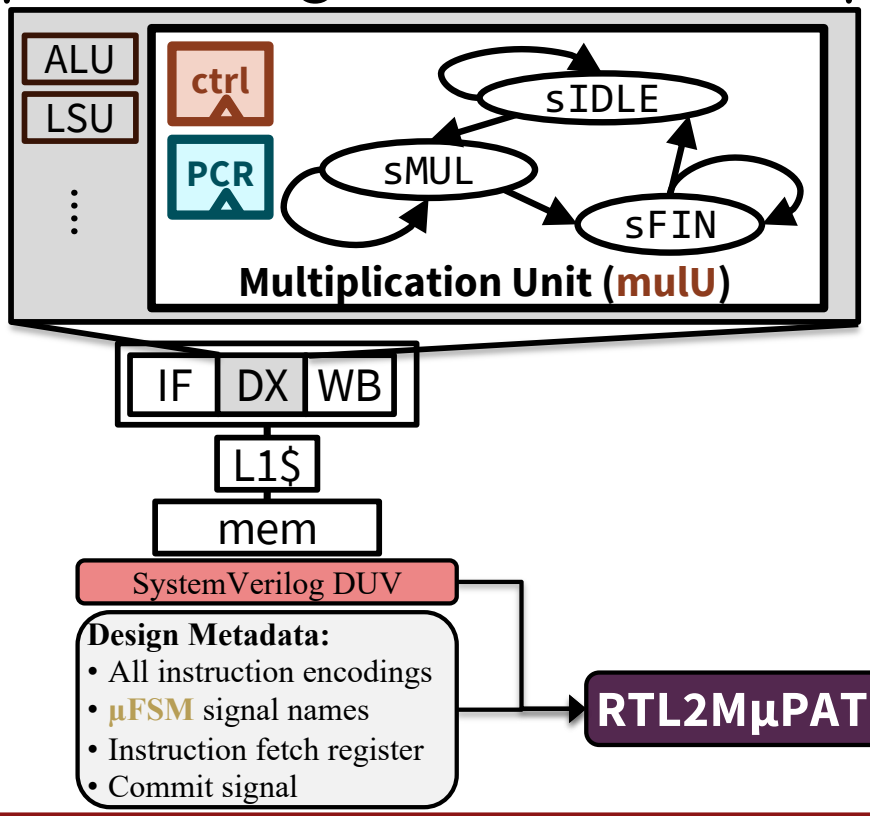
RTL2MμPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

Our Solution: Encoding μhb nodes as performing locations (PLs) using micro-op finite state machines (μFSMs)



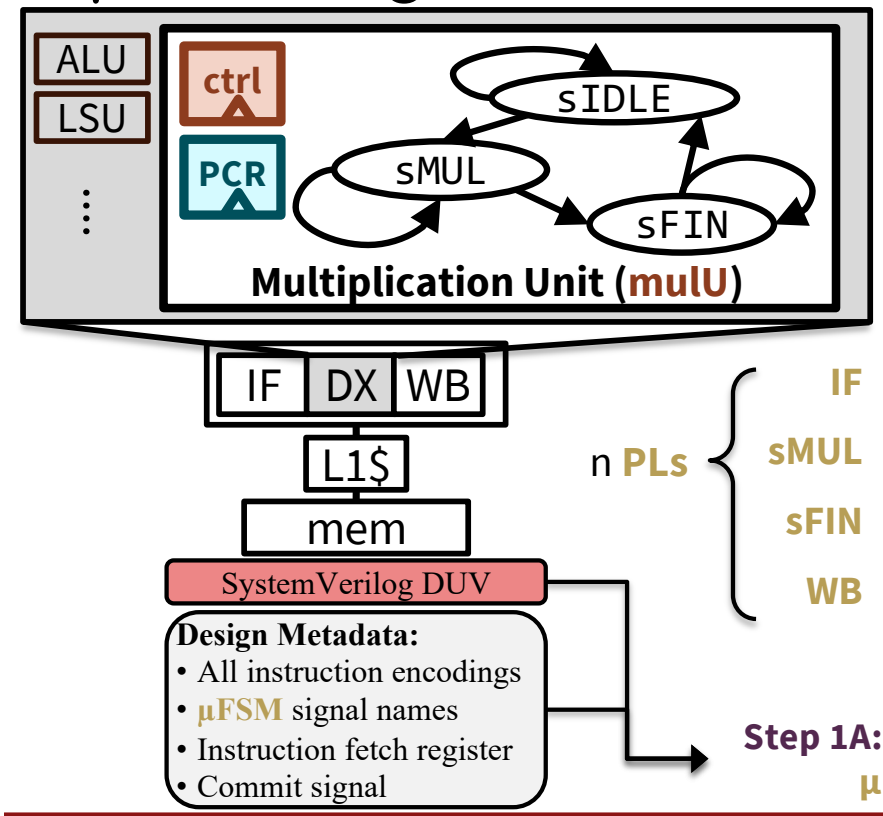
RTL2MμPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

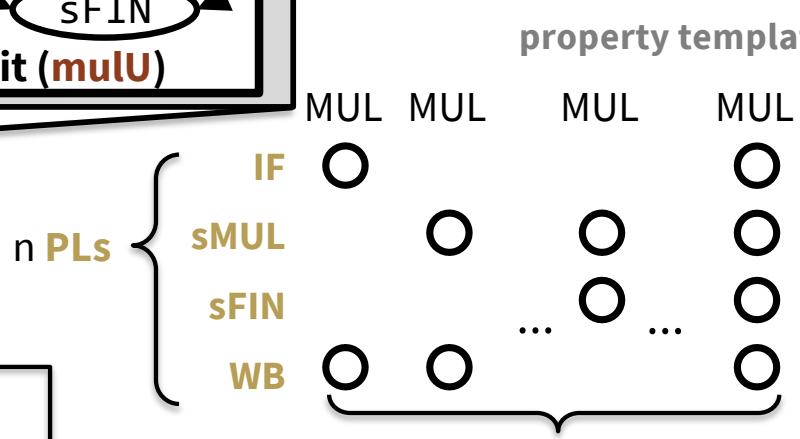
Our Procedure: Uncover node μPATH backbones → uncover μPATH edges → uncover full μPATHs



Step 1: Synthesizing **sets of nodes** that can form reachable μPATHs

Our Procedure: Uncover node μPATH backbones → uncover μPATH edges → uncover full μPATHs





Ask a model checker...

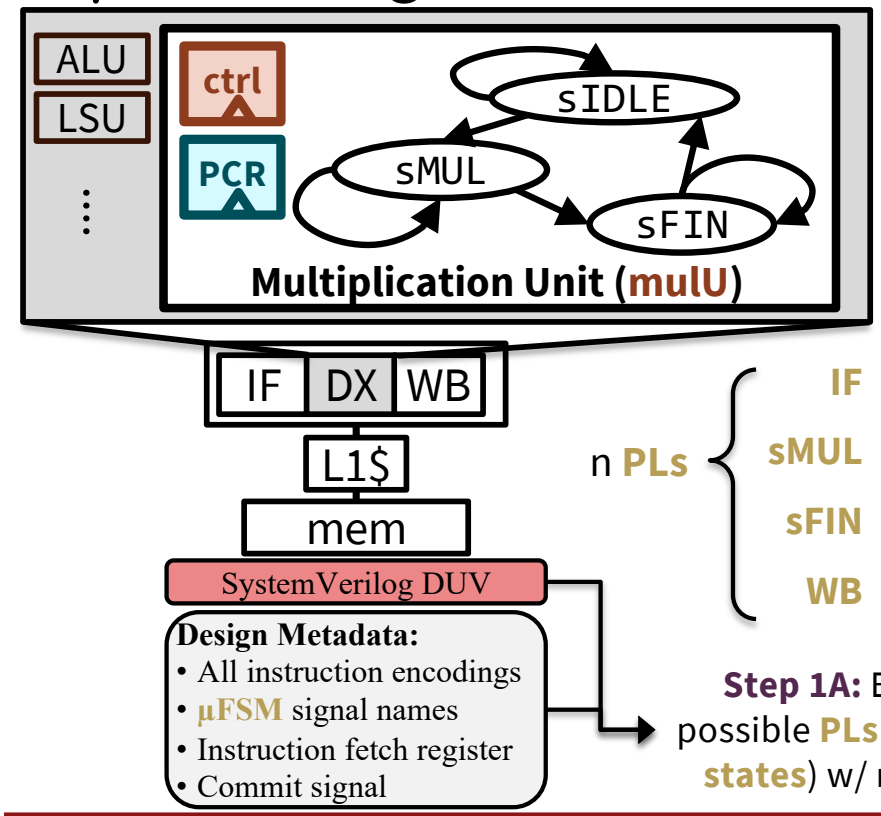
```
Can any inst visit <PL>?
Can <inst> visit <PL>?
Can <inst> visit <PL0> and not <PL1> in some μPATH?
Can <inst> visit <PL0> and <PL1> in some μPATH?
```

Step 1A: Enumerate all possible **PLs** (concrete **μFSM states**) w/ netlist analysis

Step 1B: SVA property-driven μPATH pruning

 RTL2MμPATH • [Hsiao+, MICRO'24] • [Hsiao+, **HES Top Pick *Shortlist*'26**]

Our Procedure: Uncover node μPATH backbones → uncover μPATH edges → uncover full μPATHs



Step 1: Synthesizing **sets of nodes** that can form reachable μPATHs

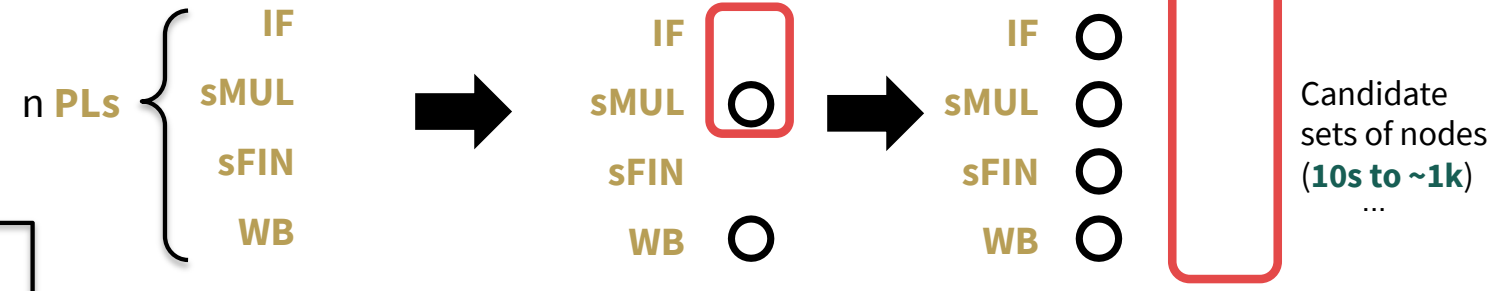
Step 2: Synthesizing **full μPATHs** by adding edges to reachable node sets

Ask a model checker...

property template

Details in the paper!

visit
set>
...ome μPATH?



Step 1A: Enumerate all possible **PLs** (concrete **μFSM states**) w/ netlist analysis

Step 1B: SVA property-driven μPATH pruning

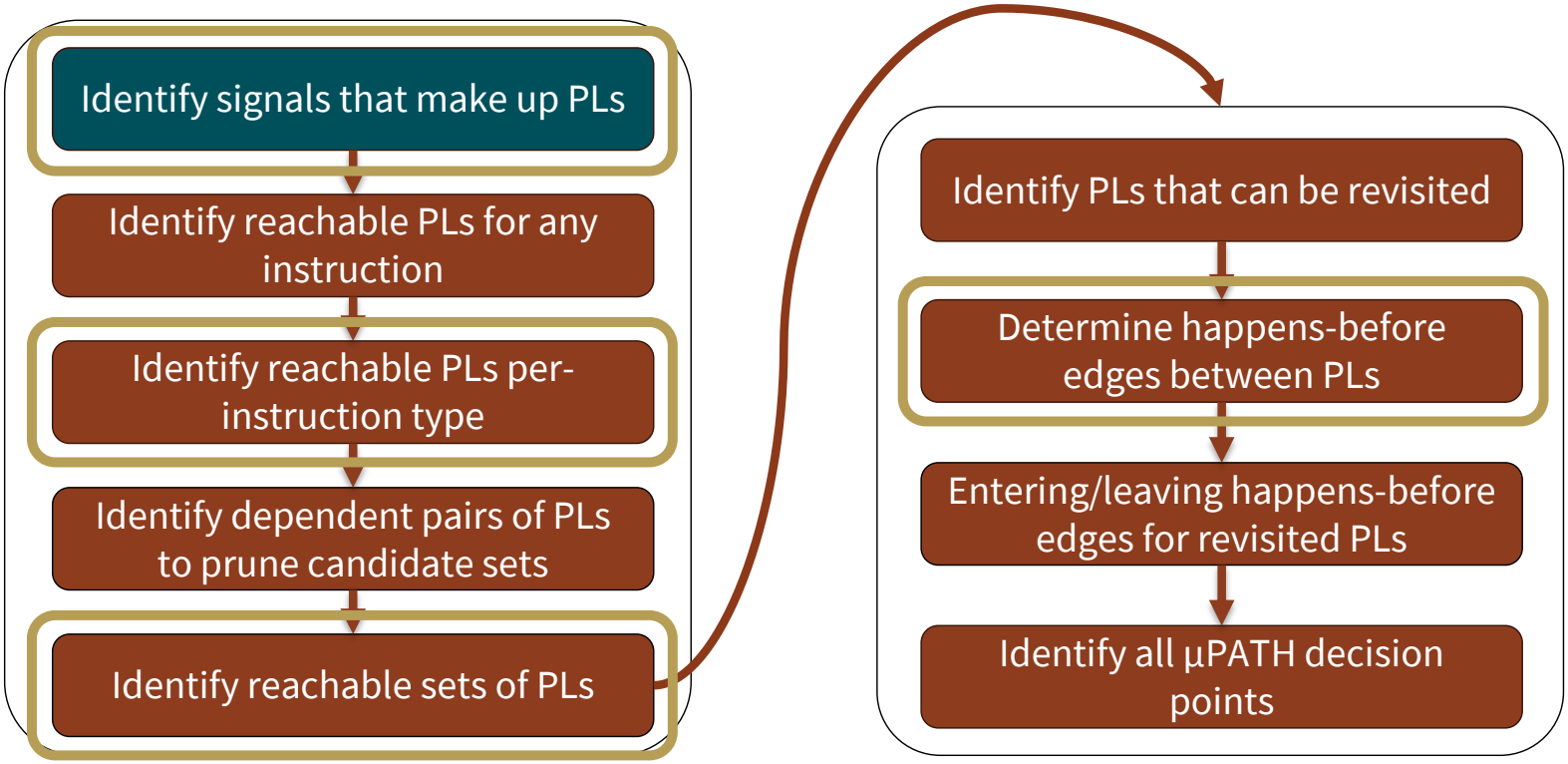
Step 1C: Embed node sets as SVA properties to deduce reachability

Functional Verification: RTL2MμPATH (Hands-On)

Yao

Stanford | ENGINEERING

RTL2MμPATH for μPATH Synthesis



Determine μPATH nodes

Determine μPATH edges

Roadmap: Apply RTL2MμPATH on CVA6

- **Step 1: [Input] μFSM annotation**
- Step 2: [Inside] Per-instruction PL reachability
- Step 3: [Inside] PL set (node set) discovery per instruction
- Step 4: [Inside] Happens-before edge discovery

All examples in this section use the CVA6 (Ariane) processor.
We will work under rtl2mupath/ folder:

```
$ cd ~/fava_isca2026/rtl2mupath/fv && . ./setup_scripts.sh && cd ../
```

[Hands-On] Finding and Annotating µFSMs

Step 1: Supply µFSM design metadata

- **PCR**: reg that holds the PC of the instruction occupying this µFSM
- **State vars**: regs that encode this µFSM's concrete state

```
$ vim fv/annotation_pcr_ufsms.txt
```

Input Metadata

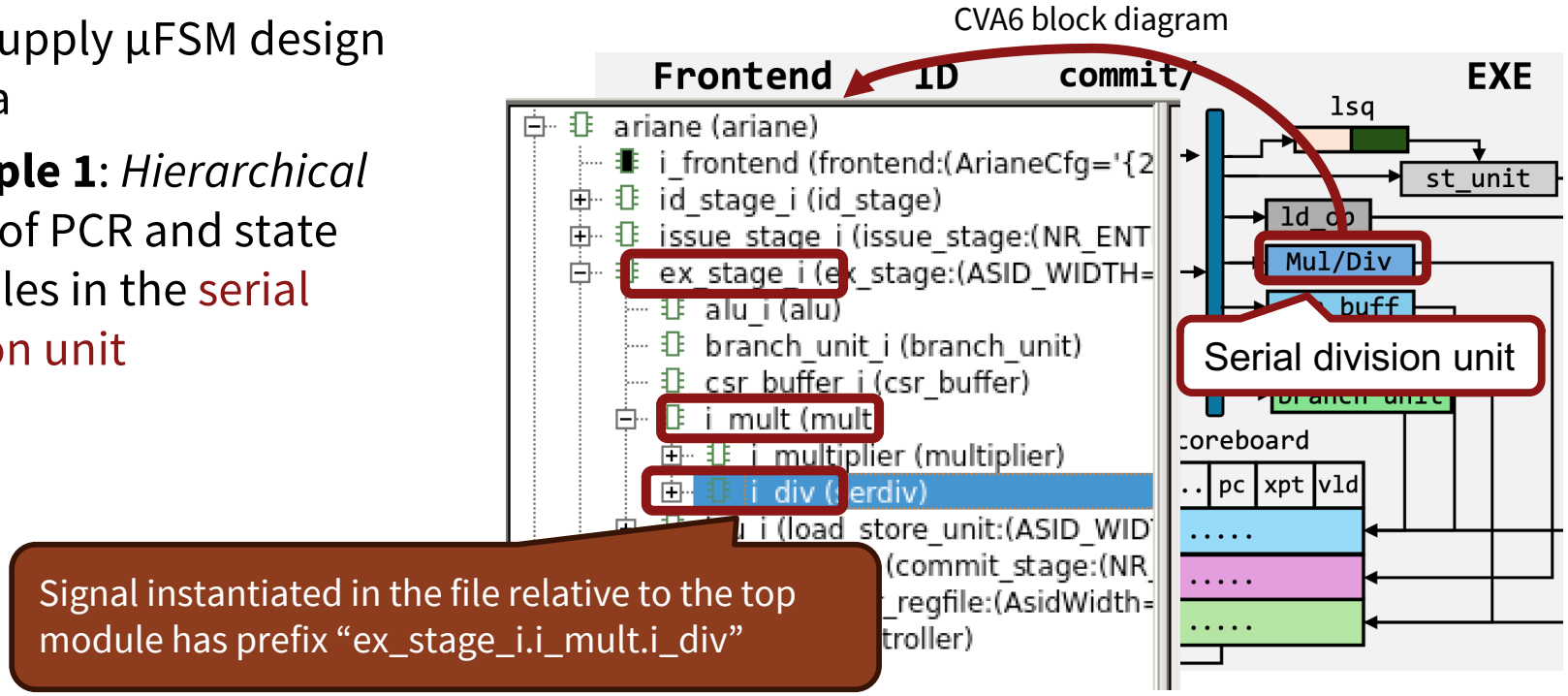
```
 1 // (name, pcr, state_var1|commit_cond, state_var2|commit_cond,...)
 2 // one line per field and each unique tuple is separated by empty lines
 3
 4 serdiv_unit_divide                        // User-defined µFSM name
 5 ex_stage_i.i_mult.i_div.pc_q              // PCR for serdiv unit
 6 ex_stage_i.i_mult.i_div.state_q,0         // State variable 1, idle state 0
 7
 8 id_stage // user-define
 9 ____________________________
10 id_stage_i.issue_q.valid,0
11
12 issue
...
```

fv/annotation_pcr_ufsms.txt

[Hands-On] Finding and Annotating µFSMs

Step 1: Supply µFSM design metadata

- **Example 1**: *Hierarchical name* of PCR and state variables in the serial division unit



Opening RTL hierarchy in Jasper

[Hands-On] Finding and Annotating µFSMs

Step 1: Supply µFSM design metadata

- **Example 1**: *Hierarchical name* of PCR and state variables in the serial division unit

```
$ vim cva6/core/serdiv.sv
```

Input Metadata

```
 1 // (name, pcr, state_var1|commit_cond,
state_var2|commit_cond,...)
 2 // one line per field and each unique
tuple is separated by empty lines
 3
 4 serdiv_unit_divide // user-defined
 5 ex_stage_i.i_mult.i_div.pc_q
 6 ex_stage_i.i_mult.i_div.state_q,0
```

fv/annotation_pcr_ufsms.txt

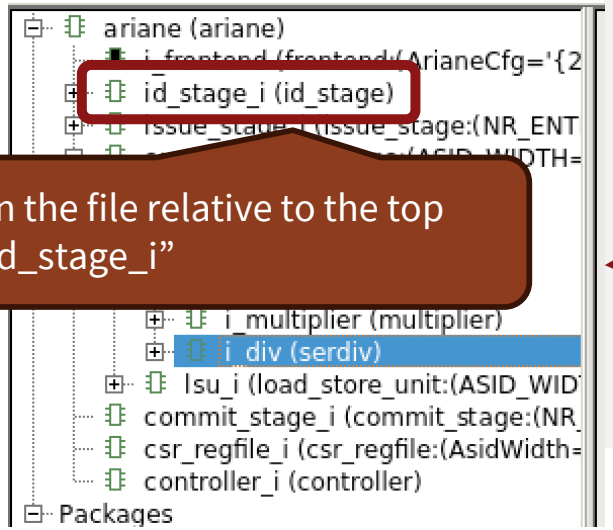
```
 18 module serdiv import ariane_pkg::*; #(
 19   parameter WIDTH       = 64
 20 ) (
...
 45   enum logic [1:0] {IDLE, DIVIDE, FINISH} state_d, state_q;
 54   logic [riscv::VLEN-1:0]    pc_q, pc_d;
...
162     unique case (state_q)
163       IDLE: begin
...
171           state_d   = DIVIDE;
...
174       DIVIDE: begin
175         if(~div_res_zero_q) begin
176           a_reg_en     = ab_comp;
177           b_reg_en     = 1'b1;
...
181         if(div_res_zero_q) begin
183           state_d   = FINISH;
...
230   assign op_a_d    = (a_reg_en) ? add_out :op_a_q;
231   assign op_b_d    = (b_reg_en)   ? b_mux :op_b_q;
232   assign res_d     = (load_en)   ? '0    :
233  (res_reg_en) ? {res_q[$high(res_q)-1:0],
```

cva6/core/serdiv.sv

[Hands-On] Finding and Annotating µFSMs

Step 1: Supply µFSM design metadata

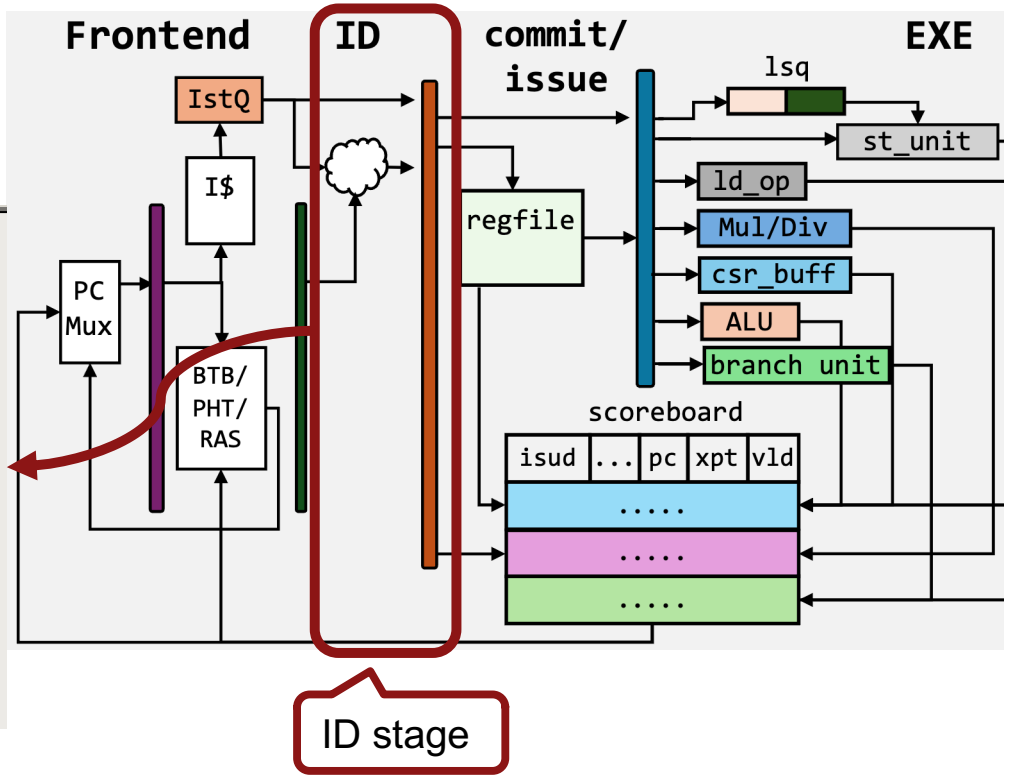
- **Example 2**: Find PCR for the decode pipeline stage



Signal instantiated in the file relative to the top module has prefix “id_stage_i”

Opening RTL hierarchy in Jasper

CVA6 block diagram



[Hands-On] Finding and Annotating µFSMs

Step 1: Supply µFSM design metadata

- **Example 2**: Find PCR for the decode pipeline stage

```
$ vim -o cva6/core/id_stage.sv
cva6/core/include/ariane_pkg.sv
fv/annotation_pcr_ufsms.txt
```

Input Metadata

```
 1 // (name, pcr, state_var1|commit_cond,
state_var2|commit_cond,...)
 2 // one line per field and each unique
tuple is separated by empty lines
 ...
 8 id_stage // user-define
 9 _____________________________
10 id_stage_i.issue_q.valid,0
```

fv/annotation_pcr_ufsms.txt

```
16 module id_stage (
17     input  logic         clk_i,
...
42     // ID/ISSUE register stage
43     struct packed {
44         logic                                valid;
45         ariane_pkg::scoreboard_entry_t sbe;
46         logic                                is_ctrl_flow;
47     } issue_n, issue_q;
...
120     always_ff @(posedge clk_i or negedge rst_ni) begin
...
124             issue_q <= issue_n;
125         end
126     end
```

cva6/core/id_stage.sv

```
590     // ----------------
591     // ID/EX/WB Stage
592     // ----------------
593     typedef struct packed {
594         logic [riscv::VLEN-1:0]   pc;
595         logic [TRANS_ID_BITS-1:0] trans_id;
...
614     } scoreboard_entry_t;
```

cva6/core/include/ariane_pkg.sv

[Hands-On] Finding and Annotating µFSMs

Step 1: Supply µFSM design metadata

- **Example 2**: Find PCR for the decode pipeline stage

```
$ vim -o cva6/core/id_stage.sv
cva6/core/include/ariane_pkg.sv
fv/annotation_pcr_ufsms.txt
```

Input Metadata

```
 1 // (name, pcr, state_var1|commit_cond,
state_var2|commit_cond,...)
 2 // one line per field and each unique
tuple is separated by empty lines
 ...
 8 id_stage // user-define
 9 id_stage_i.issue_q.sbe.pc
10 id_stage_i.issue_q.valid,0
```

fv/annotation_pcr_ufsms.txt

```
16 module id_stage (
17     input  logic         clk_i,
...
42     // ID/ISSUE register stage
43     struct packed {
44         logic                                valid;
45         ariane_pkg::scoreboard_entry_t sbe;
46         logic                                is_ctrl_flow;
47     } issue_n, issue_q;
...
120     always_ff @(posedge clk_i or negedge rst_ni) begin
...
124             issue_q <= issue_n;
125         end
126     end
```

cva6/core/id_stage.sv

```
590     // ----------------
591     // ID/EX/WB Stage
592     // ----------------
593     typedef struct packed {
594         logic [riscv::VLEN-1:0]   pc;
595         logic [TRANS_ID_BITS-1:0] trans_id;
...
614     } scoreboard_entry_t;
```

cva6/core/include/ariane_pkg.sv

Roadmap: Apply RTL2MμPATH on CVA6

- Step 1: [Input] μFSM annotation
- **Step 2: [Inside] Per-instruction PL reachability**
- Step 3: [Inside] PL set (node set) discovery per instruction
- Step 4: [Inside] Happens-before edge discovery

[Hands-On] Per-instruction PL reachability

Step 2a: Enumerate all possible PLs (conc
widths of the annotated µFSM signals

```
$ vim -o fv/annotation_pcr_ufsms.txt
fv/xDUVPLs/perf_loc.sv
```

Input Metadata

```
1 // (name, pcr, state_var1|commit_cond,
state_var2|commit_cond,...)
2 // one line per field and each unique
tuple is separated by empty lines
3
4 serdiv_unit_divide // user-de
5 ex_stage_i.i_mult.i_div.pc_q
6 ex_stage_i.i_mult.i_div.state_q,0
```

idle state

fv/annotation_pcr_ufsms.txt

Intermediate Output

```
// µFSM variable name, signal width
1 ex_stage_i.i_mult.i_div.state_q,2
2 id_stage_i.issue_q.valid,1
```

fv/xDUVPLs/sig_width.txt

```
18 module serdiv import ariane_pkg::*; #(
19   parameter WIDTH       = 64
20 ) (
...
45   enum logic [1:0] {IDLE, DIVIDE, FINISH}
state_d, state_q;
54   logic [riscv::VLEN-1:0]   pc_q, pc_d;
...
```

cva6/core/serdiv.sv

Intermediate Output

```
1 wire serdiv_unit_divide_s1 =
2   (ex_stage_i.i_mult.i_div.state_q == 2'd1) &&
3    1'b1;
4 wire serdiv_unit_divide_s2 =
5   (ex_stage_i.i_mult.i_div.state_q == 2'd2) &&
6    1'b1;
7 wire serdiv_unit_divide_s3 =
8   (ex_stage_i.i_mult.i_div.state_q == 2'd3) &&
9    1'b1;
...
```

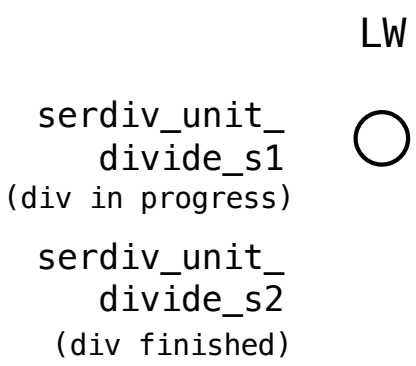
fv/xDUVPLs/perf_loc.sv

[Hands-On] Per-instruction PL reachability

Step 2b: **Uncover per-instruction PLs:** An inst visits a PL by placing its PC in the μFSM's PCR and assigning its concrete non-idle state to the μFSM's state variables

- **Example:** Fill in expression to encode a LW inst visits serdiv_unit_divide_s1

```
$ vim -o fv/synthlc/i_LW_out/xCoverAPerflocDiv/out/cover_individual.sv
```



Intermediate Output

```
 55 pc0_i0_assoc_1: assume property (@(posedge clk_i)
 56     id_stage_i.fetch_entry_i.address == pc0 |-> id_stage_i.instruction == i0);
...
 82 wire serdiv_unit_divide_s1 =
 83   ____________________________________  &&
 84   ____________________________________ &&
 85    1'b1;
...
318 i_LW_0: assume property (i0[14:12] == 3'b010);
319 i_LW_1: assume property (i0[11:7] != 5'd0);
320 i_LW_2: assume property (i0[6:0] == 7'b0000011);
...
321 C_0: cover property (@(posedge clk_i) serdiv_unit_divide_s1);
```

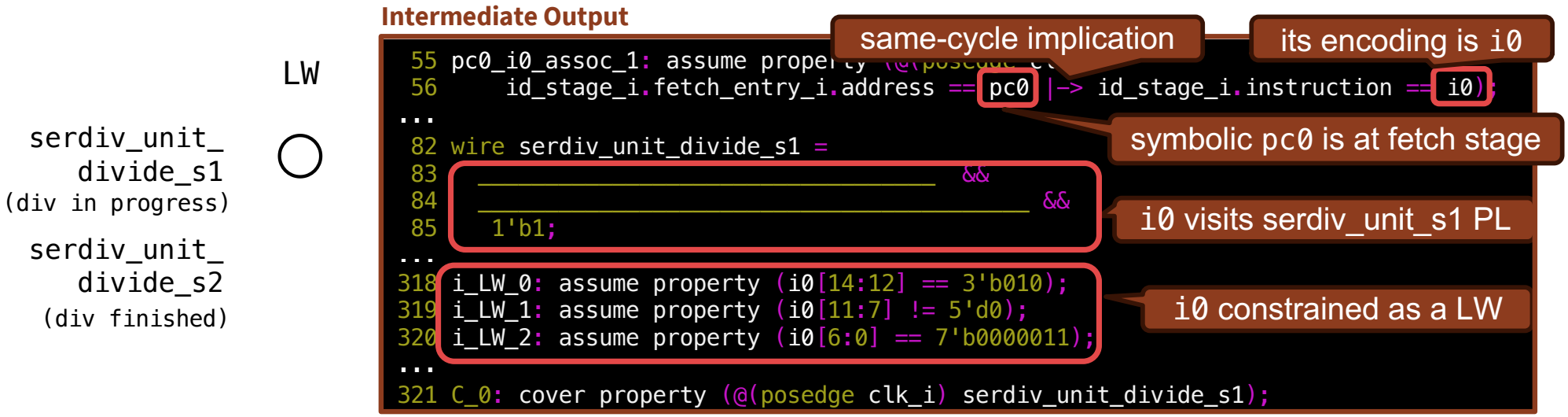
fv/synthlc/i_LW_out/xCoverAPerflocDiv/out/cover_individual.sv

[Hands-On] Per-instruction PL reachability

Step 2b: **Uncover per-instruction PLs:** An inst visits a PL by placing its PC in the μFSM's PCR and assigning its concrete non-idle state to the μFSM's state variables

- **Example:** Fill in expression to encode a LW inst visits serdiv_unit_divide_s1

```
$ vim -o fv/synthlc/i_LW_out/xCoverAPerflocDiv/out/cover_individual.sv
```

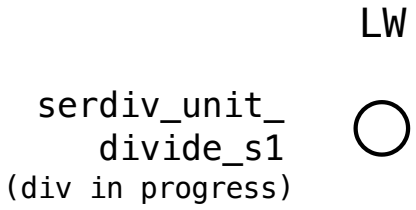


[Hands-On] Per-instruction PL reachability

Step 2b: **Uncover per-instruction PLs:** An inst visits a PL by placing its PC in the μFSM's PCR and assigning its concrete non-idle state to the μFSM's state variables

- **Example:** Fill in expression to encode a LW inst visits serdiv_unit_divide_s1

```
$ vim -o fv/synthlc/i_LW_out/xCoverAPerflocDiv/out/cover_individual.sv
```



Intermediate Output

```
 55 pc0_i0_assoc_1: assume property (@(posedge clk_i)
 56     id_stage_i.fetch_entry_i.address == pc0 |-> id_stage_i.instruction == i0);
...
 82 wire serdiv_unit_divide_s1 =
 83   (ex_stage_i.i_mult.i_div.pc_q == pc0) &&
 84   (ex_stage_i.i_mult.i_div.state_q == 2'd1) &&
 85    1'b1;
...
318 i_LW_0: assume property (i0[14:12] == 3'b010);
319 i_LW_1: assume property (i0[11:7] != 5'd0);
320 i_LW_2: assume property (i0[6:0] == 7'b0000011);
...
321 C_0: cover property (@(posedge clk_i) serdiv_unit_divide_s1);
```

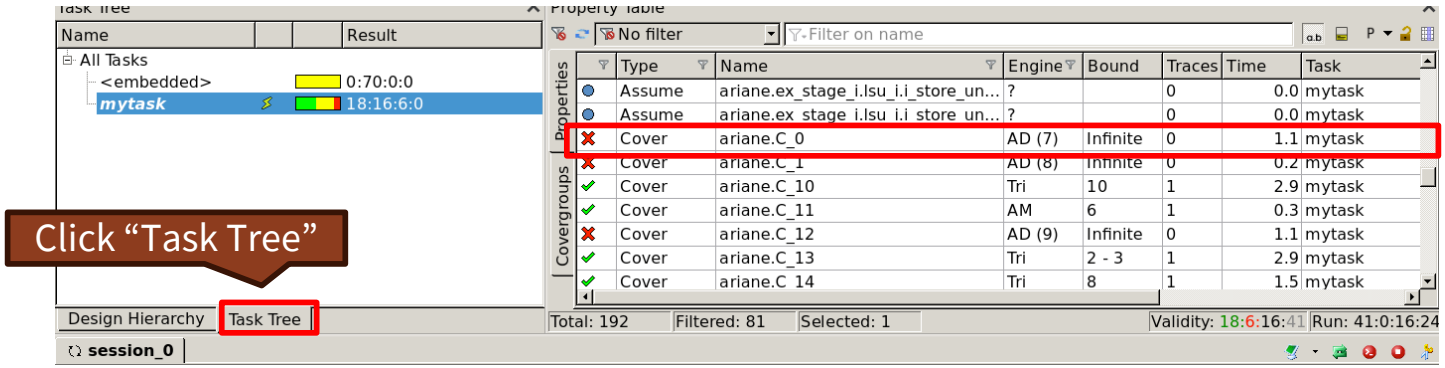
fv/synthlc/i_LW_out/xCoverAPerflocDiv/out/cover_individual.sv

[Hands-On] Per-instruction PL reachability

Step 2b: **Uncover per-instruction PLs:** An inst visits a PL by placing its PC in the µFSM's PCR and assigning its concrete non-idle state to the µFSM's state variables

- **Example:** Determine whether a LW inst visits `serdiv_unit_divide_s1`

```
$ cd fv && mkdir xEval
$ ./RUN_JG.sh -j xEval -s synthlc/i_LW_out/xCoverAPerflocDiv/out/cover_individual.sv
```




```
321 C_0: cover property (@(posedge clk_i) serdiv_unit_divide_s1);
```

fv/synthlc/i_LW_out/xCoverAPerflocDiv/out/cover_individual.sv

Roadmap: Apply RTL2MμPATH on CVA6

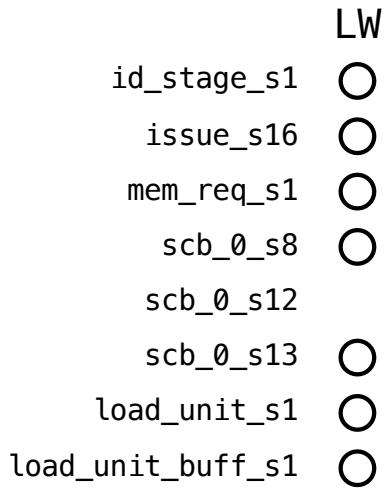
- Step 1: [Input] μFSM annotation
- Step 2: [Inside] Per-instruction PL reachability
- **Step 3: [Inside] PL set (node set) discovery per instruction**
- Step 4: [Inside] Happens-before edge discovery

[Hands-On] PL set (node set) discovery per instruction

Step 3: For each candidate PL set, check if an instruction can visit exactly the nodes in the set, but no others in some reachable execution.

Example: Fill in the property to determine if a LW inst visits the depicted PL set

```
$ vim synthlc/i_LW_out/xPerfLocSubsetDiv/covertest/coverset_0.sv
```



Intermediate Output

```
 55 pc0_i0_assoc_1: assume property (@(posedge clk_i)
 56     id_stage_i.fetch_entry_i.address == pc0 |-> id_stage_i.instruction == i0);
...
 90 wire id_stage_s1 =
 91   (id_stage_i.issue_q.sbe.pc == pc0) &&
 92   (id_stage_i.issue_q.valid == 1'd1) &&
...
322 reg id_stage_s1_hpn;
323 always @(posedge clk_i) begin
324     if (!rst_ni)
325         id_stage_s1_hpn <= 1'b0;
326     else if (id_stage_s1)
327         id_stage_s1_hpn <= 1'b1;
328 end
441 C_0_N: cover property (@(posedge clk_i) id_stage_s1_hpn & ____________________
_____________ _____________ _____________ _____________ _____________ & 1'b1 & !(id_stage_s1
| issue_s16 | scb_0_s13 | scb_0_s8 | load_unit_s1 | load_unit_buff_s1 | mem_req_s1 | 1'b0));
```

Symbolic instruction with pc0 with symbolic encoding being i0

Symbolic instruction i0 visiting id_stage_s1 PL

i0 visited id_stage_s1 PL in past

Ever a cycle where visited signals for the PLs in the set are true

and the i0 execution already ends

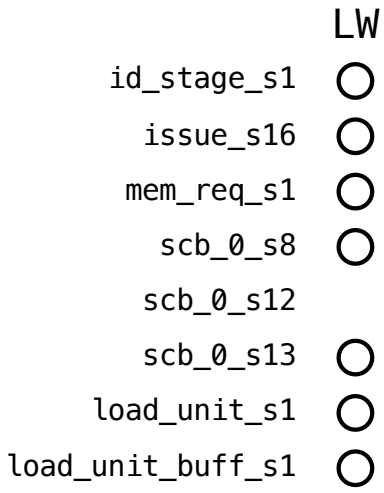
fv/synthlc/i_LW_out/xPerfLocSubsetDiv/c

[Hands-On] PL set (node set) discovery per instruction

Step 3: For each candidate PL set, check if an instruction can visit exactly the nodes in the set, but no others in some reachable execution.

Example: Fill in the property to determine if a LW inst visits the depicted PL set

```
$ vim synthlc/i_LW_out/xPerfLocSubsetDiv/covertest/coverset_0.sv
```



Intermediate Output

```
 55 pc0_i0_assoc_1: assume property (@(posedge clk_i)
 56     id_stage_i.fetch_entry_i.address == pc0 |-> id_stage_i.instruction == i0);
...
 90 wire id_stage_s1 =
 91   (id_stage_i.issue_q.sbe.pc == pc0) &&
 92   (id_stage_i.issue_q.valid == 1'd1) &&
...
322 reg id_stage_s1_hpn;
323 always @(posedge clk_i) begin
324     if (!rst_ni)
325         id_stage_s1_hpn <= 1'b0;
326     else if (id_stage_s1)
327         id_stage_s1_hpn <= 1'b1;
328 end
441 C_0_N: cover property (@(posedge clk_i) id_stage_s1_hpn & issue_s16_hpn & mem_req_s1_hpn &
scb_0_s13_hpn & scb_0_s8_hpn & load_unit_s1_hpn & load_unit_buff_s1_hpn & 1'b1 & !(id_stage_s1 |
issue_s16 | scb_0_s13 | scb_0_s8 | load_unit_s1 | load_unit_buff_s1 | mem_req_s1 | 1'b0));
```

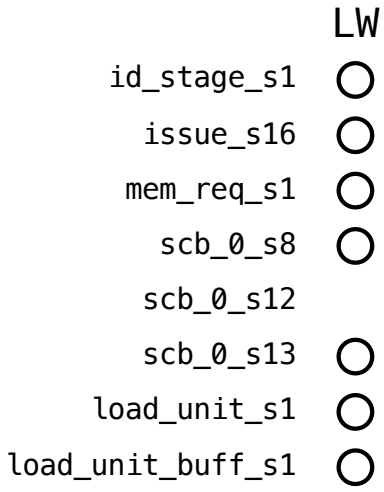
fv/synthlc/i_LW_out/xPerfLocSubsetDiv/covertest/coverset_0.sv

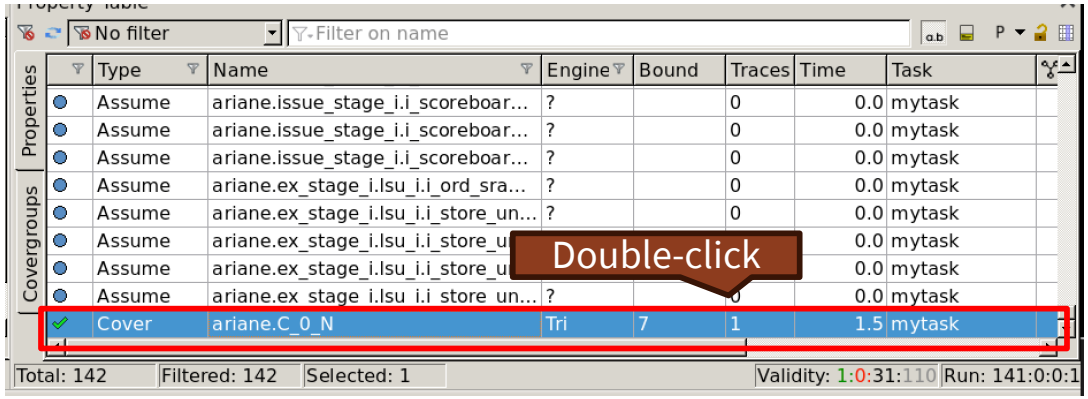
[Hands-On] PL set (node set) discovery per instruction

Step 3: For each candidate PL set, check if an instruction can visit exactly the nodes in the set, but no others in some reachable execution.

Example: Cover property to determine if a LW inst visits the depicted PL set

```
$ ./RUN_JG.sh -j xEval -s synthlc/i_LW_out/xPerfLocSubsetDiv/covertest/coverset_0.sv
```






```
441 C_0_N: cover property (@(posedge clk_i) id_stage_s1_hpn & issue_s16_hpn & scb_0_s13_hpn &
scb_0_s8_hpn & load_unit_s1_hpn & load_unit_buff_s1_hpn & mem_req_s1_hpn & 1'b1 & !(id_stage_s1
| issue_s16 | scb_0_s13 | scb_0_s8 | load_unit_s1 | load_unit_buff_s1 | mem_req_s1 | 1'b0));
```

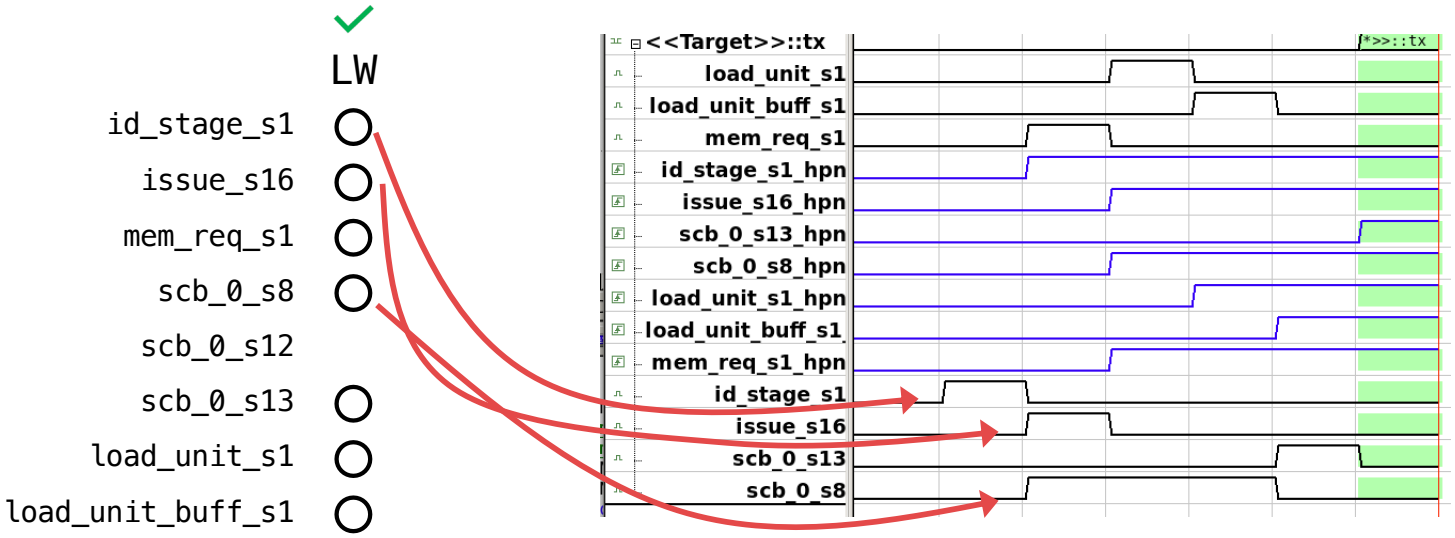
fv/synthlc/i_LW_out/xPerfLocSubsetDiv/covertest/coverset_0.sv

[Hands-On] PL set (node set) discovery per instruction

Step 3: For each candidate PL set, check if an instruction can visit exactly the nodes in the set, but no others in some reachable execution.

Example: Cover property to determine if a LW inst visits the depicted PL set

```
$ ./RUN_JG.sh -j xEval -s synthlc/i_LW_out/xPerfLocSubsetDiv/covertest/coverset_0.sv
```



Roadmap: Apply RTL2MμPATH on CVA6

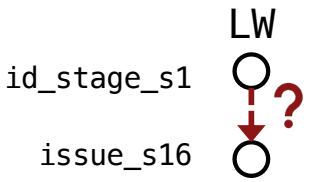
- Step 1: [Input] μFSM annotation
- Step 2: [Inside] IUV PL reachability
- Step 3: [Inside] PL set (node set) discovery per instruction
- **Step 4: [Inside] Happens-before edge discovery**

[Hands-On] Happens-before edge discovery

Step 4: Netlist analysis derives candidate happens-before edges among PLs. RTL2MμPATH encodes them as SVAs to check whether each is a true HB relation.

Example: A property that asserts a LW always visits id_stage_s1 before issue_s16

```
$ vim synthlc/i_LW_out/xHBPerfG_dfg_v3_div/out/HB_59.sv
```



Happens-before assertion template:

```
HB_<NUM>: assert property (@(posedge <CLK>)
     ((<PL1>) && !<PL1>_hpn) |-> !(<PL2>_hpn || (<PL2>)));
```

Instruction visiting PL1 for the 1st time

SVA syntax: Same-cycle implication

Instruction is not currently in PL2 and has never visited PL2 before

[Hands-On] Happens-before edge discovery

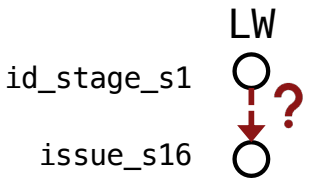
Step 4: Netlist analysis derives candidate happens-before edges among PLs. RTL2MμPATH encodes them as SVAs to check whether each is a true HB relation.

Example: A property that asserts a LW always visits id_stage_s1 before issue_s16

```
$ vim synthlc/i_LW_out/xHBPerfG_dfg_v3_div/out/HB_59.sv
```

Happens-before assertion template:

```
HB_<NUM>: assert property (@(posedge <CLK>)
     ((<PL1>) && !<PL1>_hpn) |-> !(<PL2>_hpn || (<PL2>)));
```



Intermediate Output

```
322 reg id_stage_s1_hpn;
...
326     else if (id_stage_s1)
327         id_stage_s1_hpn <= 1'b1;
...
334     else if (issue_s16)
335         issue_s16_hpn <= 1'b1;
...
339 HB_59: assert property (@(posedge clk_i) ((id_stage_s1) && !id_stage_s1_hpn) |->
!(issue_s16_hpn || (issue_s16)));
```

Instruction is visiting id_stage for the 1st time

Instruction is not visiting issue_s16 and has not visited issue_s16 before

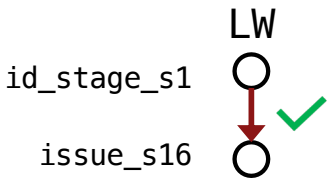
fv/synthlc/i_LW_out/xHBPerfG_dfg_v3_div/out/HB_59.sv

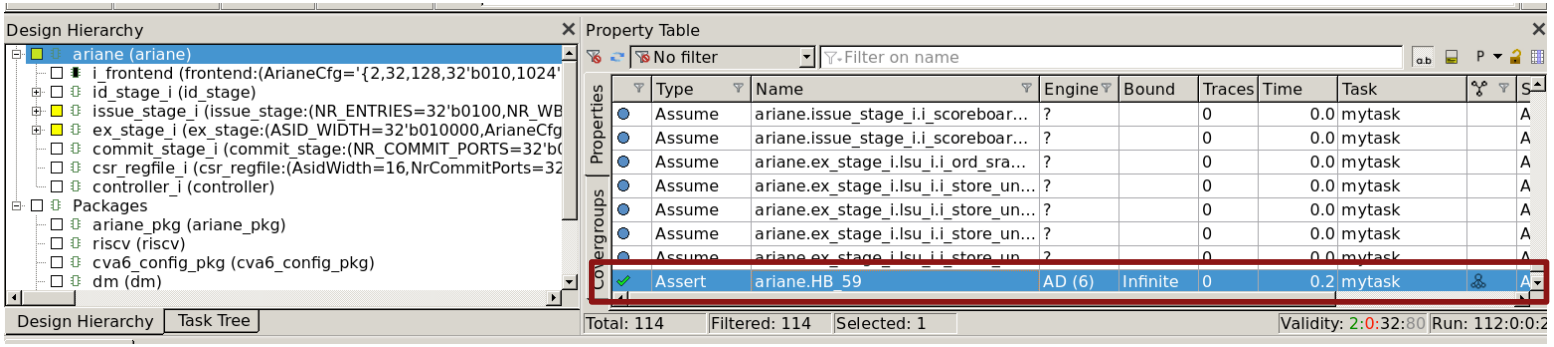
[Hands-On] Happens-before edge discovery

Step 4: Netlist analysis derives candidate happens-before edges among PLs. RTL2MμPATH encodes them as SVAs to check whether each is a true HB relation.

Example: A property that asserts a LW always visits id_stage_s1 before issue_s16

```
$ ./RUN_JG.sh -j xEval -s synthlc/i_LW_out/xHBPerfG_dfg_v3_div/out/HB_59.sv
```



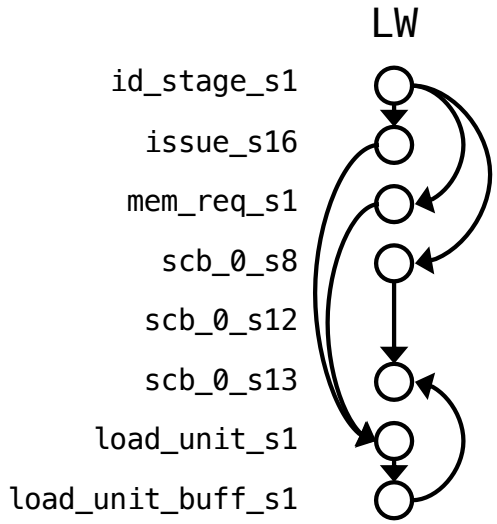



```
339 HB_59: assert property (@(posedge clk_i) ((id_stage_s1) && !id_stage_s1_hpn) |->
!(issue_s16_hpn || (issue_s16)));
```

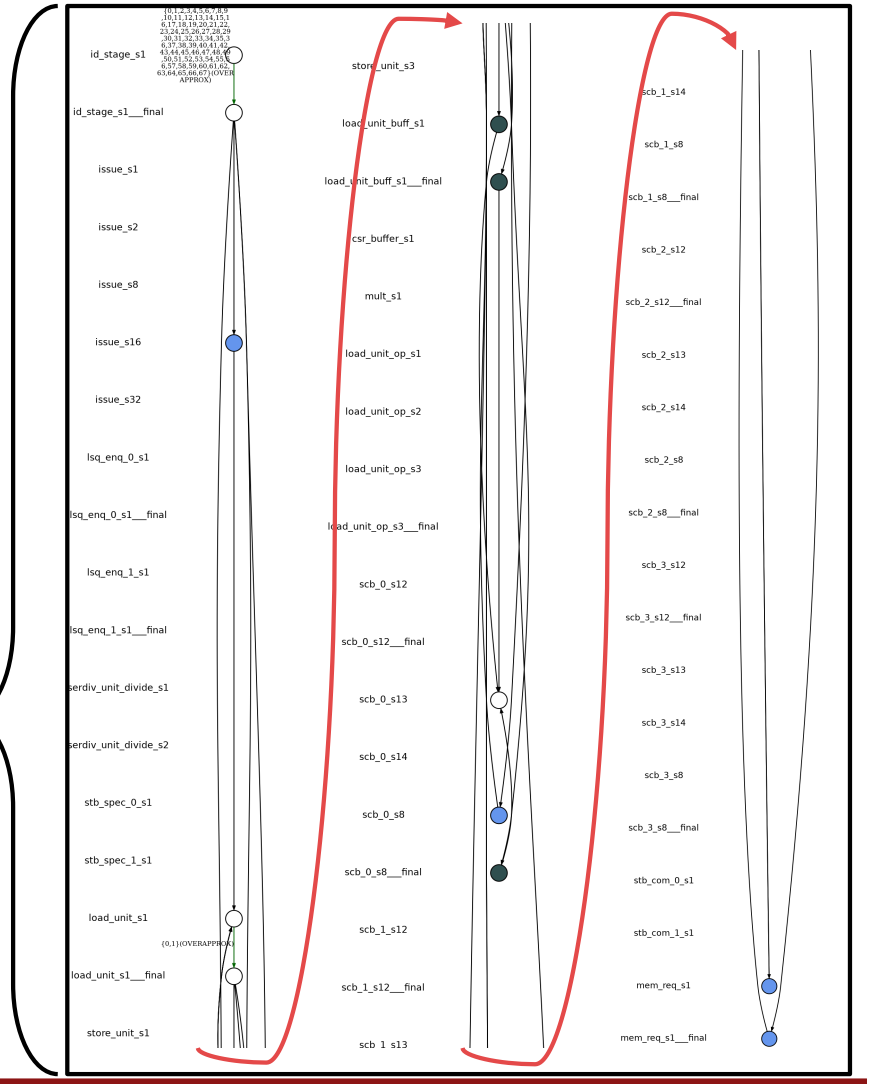
fv/synthlc/i_LW_out/xHBPerfG_dfg_v3_div/out/HB_59.sv

[Hands-On] µPATH visualization

```
$ cd ~/fava_isca2026/rtl2mupath
$ xdg-open
all_paths_html/IA/i_LW_III/xSummarize/nonisograph
/com_46_0_0.png
```



simplified core-level LW µpath w/ unused row labels omitted



Other outputs

Run RTL2MμPATH end-to-end:

- `./run_an_inst.sh <instr>.sv`

Per-instruction outputs are in directories:

- `fv/synthlc/i_LW_out/`
- `fv/synthlc/i_DIV_out/`
- `fv/synthlc/i_SW_out/`
- `...`

Per instruction directory, all μPATH outputs are depicted in .png files:

- `fv/synthlc/i_<instr>_out/xSummarize/nonisograph/*.png`