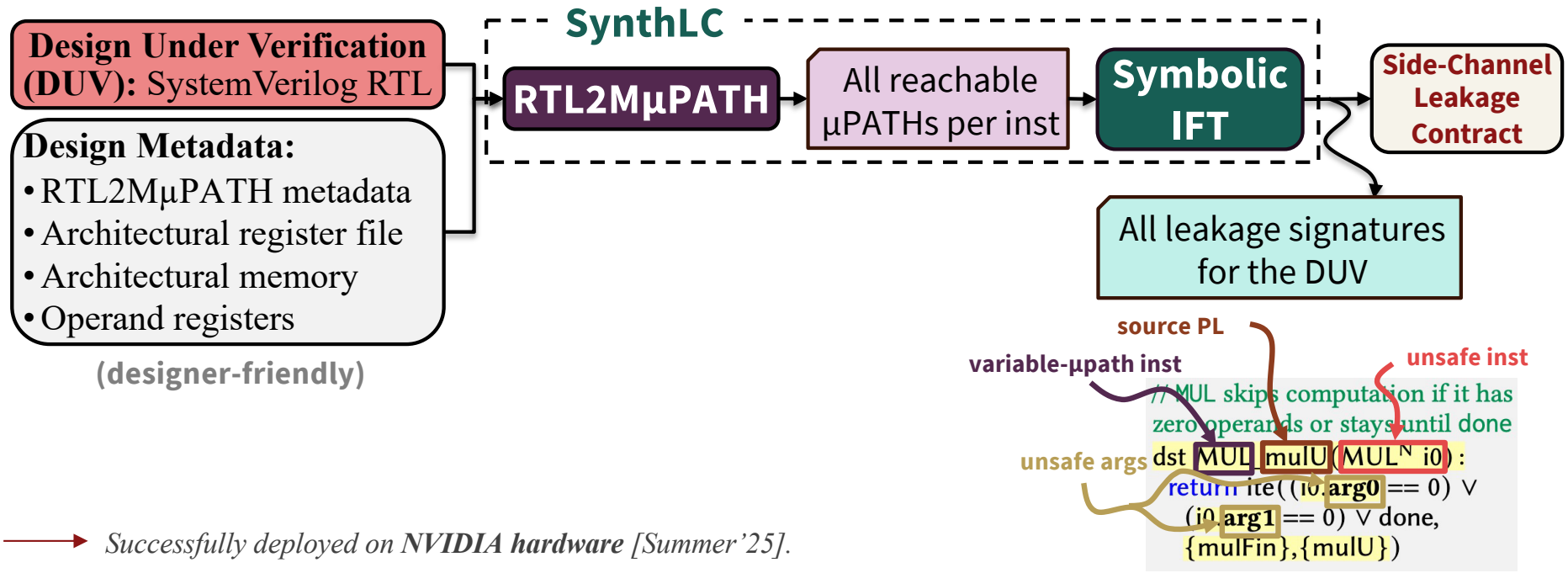
Side-Channel Security Verification: SynthLC (Overview)

Caroline

Stanford | ENGINEERING

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

Overview: Synthesizing formally verified side-channel leakage signatures, and thus leakage contracts, from SystemVerilog RTL



Each **localizes** and **characterizes** a **hardware side-channel**.

→ *Successfully deployed on **NVIDIA hardware** [Summer'25].*

https://github.com/yaohsiaopid/SynthLC
https://github.com/samanthaarcher0/SynthLC

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

Key Insight: µPATH variability (>1 µPATH) is a strong indicator of a hardware side-channel in the DUV

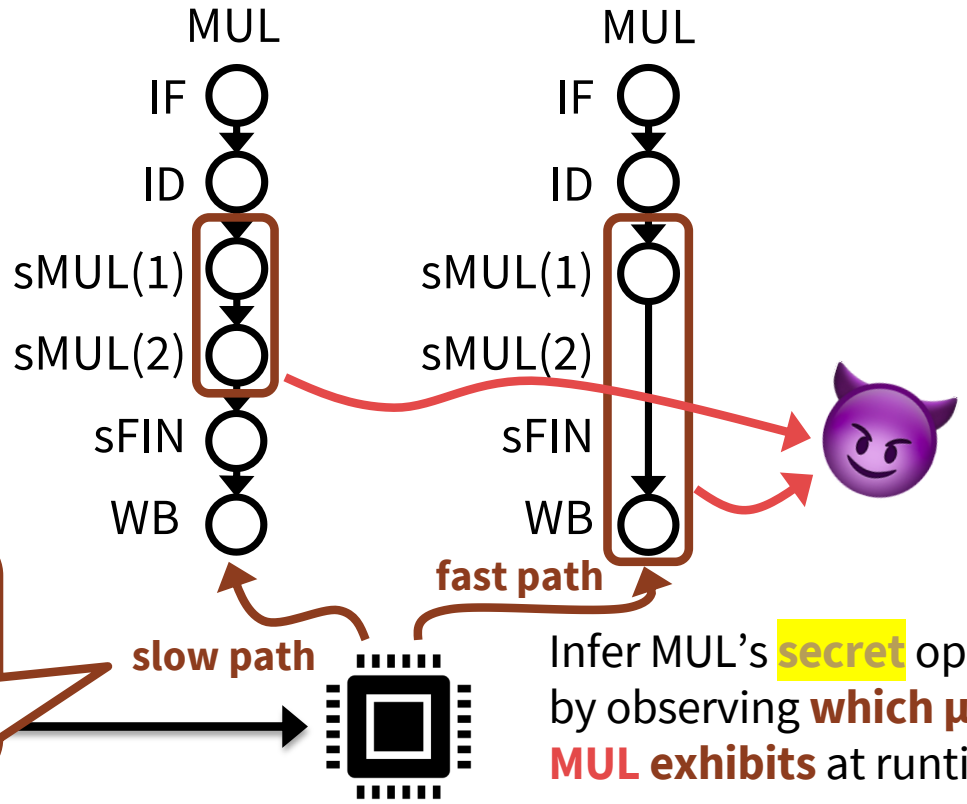
A typical victim program pattern in a **classic side-channel attack**:

```
unsafe_instruction secret
```

Example on some processor:

```
MUL secret op1
```

```
MUL_mulU(MUL^N i0):
 if (i0.op0 == 0 || i0.op1 == 0):
   return fast_path  # 1 cycle
 return slow_path    # 3 cycle
```



Infer MUL's secret operand by observing **which µPATH MUL exhibits** at runtime!

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

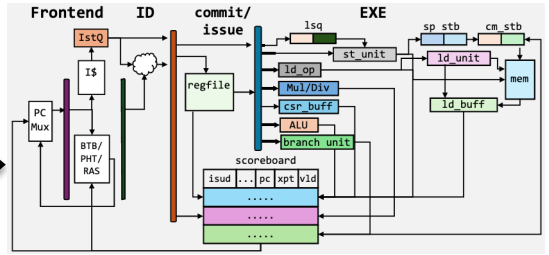
Key Insight: µPATH variability (>1 µPATH) is a strong indicator of a hardware side-channel

A **more subtle** victim program pattern in a **side-channel attack**:

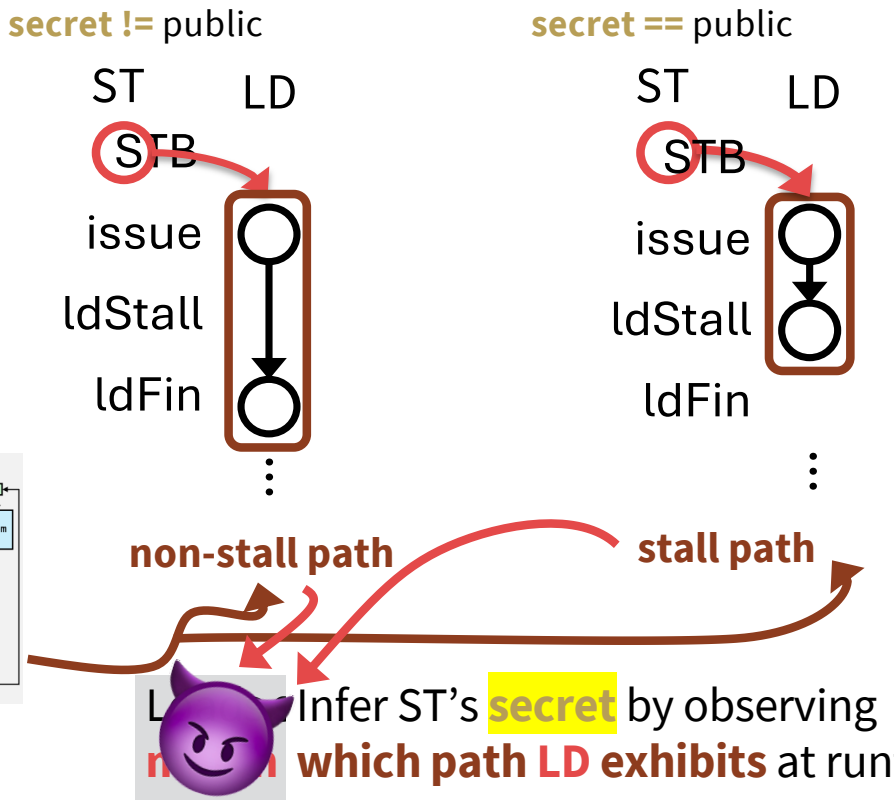
```
unsafe_instruction secret
other_instruction public
```

Example on RISC-V CVA6 Core:

```
ST [secret]
LD [public]
```



RISC-V CVA6 [Zaruba+, VLSI19]



Infer ST's **secret** by observing **which path LD exhibits** at runtime!

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

Our Procedure: Synthesize and compare all μPATHs for all inst → identify μPATH variability per inst → attribute it to unsafe operands

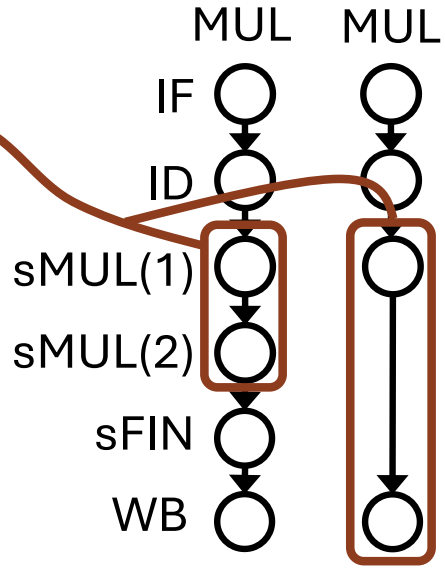
✓:(MUL, op0), (MUL, op1)
✗:(SUB, op0), ...

Ask a model checker...

Does μPATH decision <src>→<dst0> vs. <src>→<dst1> of inst <P> depend on operand <unsafe-op> of inst <T>?

…when inst <T> is <the same as, younger than, older than> inst <P>.

property templates



Side-Channel Leakage Contract

unsafe inst-op

arm intel RISC-V

br, cond
ld, addr
st, addr
div, op0
div, op1

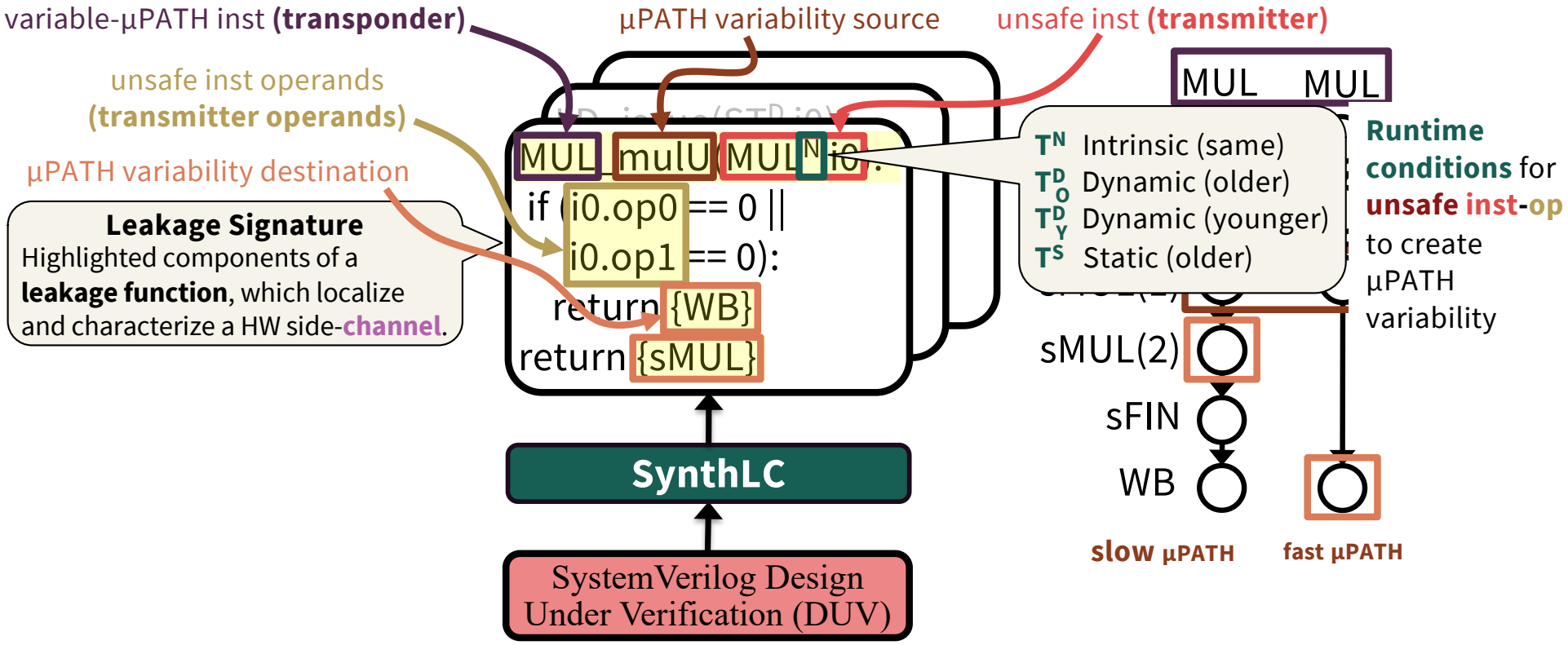
Automatically attribute path variability to **≥1 unsafe inst-op** under various hardware **conditions**

SystemVerilog DUV
Design Metadata
→ **RTL2MμPATH** → All reachable μPATHs per inst → **Symbolic IFT**

CellIFT [Solt, Sec'22]
Jasper SPV

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

SynthLC outputs a complete set of leakage signatures for the DUV, which localize and characterize its hardware side-channels



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

Leakage signatures are a unifying formalism for hardware side-channels and the leakage contracts that characterize them

transponders — μPATH variability source — transmitters — transmitter operands

Defenses	Side-Channel Leakage Contract: Leakage Contract Components	μPATHs	Leakage Signature Components: P	src	T^N	T^D	T^S	args
CT [e.g., Cauligi+, SecDev'7], **SpecShield** [Barber+, PACT19], **ConTExt** [Schwarz+, NDSS20] **Serberus** • [Mosier+, SP'24, **HES Top Pick'26**]	unsafe inst-op (arm, intel, RISC-V)	-	-	-	✓	✓	✓	✓
MI6 [Bourgeat+, MICRO19]	Contention-based dynamic channels	-	✓	✓	✓	✓	-	-
	Static channels	-	✓	✓	-	-	✓	-
OISA [Yu+, NDSS19]	Input-dependent arithmetic units	-	-	✓	✓	-	-	✓
STT [Yu+, MICRO19] **SDO** [Yu+, ISCA20] **SPT** [Choudhary+, MICRO21] **Protean** • [Mosier+, HPCA'26]	Explicit channels	-	✓	✓	✓	-	-	✓
	Implicit channels	-	✓	✓	-	✓	✓	✓
	Implicit branches	-	✓	-	-	✓	✓	✓
	Prediction-based channels	-	✓	✓	-	-	✓	✓
	Resolution-based channels	-	✓	✓	-	✓	-	✓
SDO [Yu+, ISCA20]	Data-oblivious variants	✓	-	-	✓	-	-	✓
Dolma [Loughlin+, ISCA21]	Variable-time micro-ops	-	-	-	✓	-	-	✓
	Contention-based dynamic channels	-	✓	✓	✓	✓	-	✓
	Inducive micro-ops	-	✓	-	-	✓	-	✓
	Resolvent micro-ops	-	-	-	-	✓	-	✓
	Prediction resolution points	-	✓	✓	-	✓	-	✓
	Persistent state modifying micro-ops	-	-	-	-	-	✓	✓

Side-Channel Security Verification: SynthLC (Hands-On)

Samantha

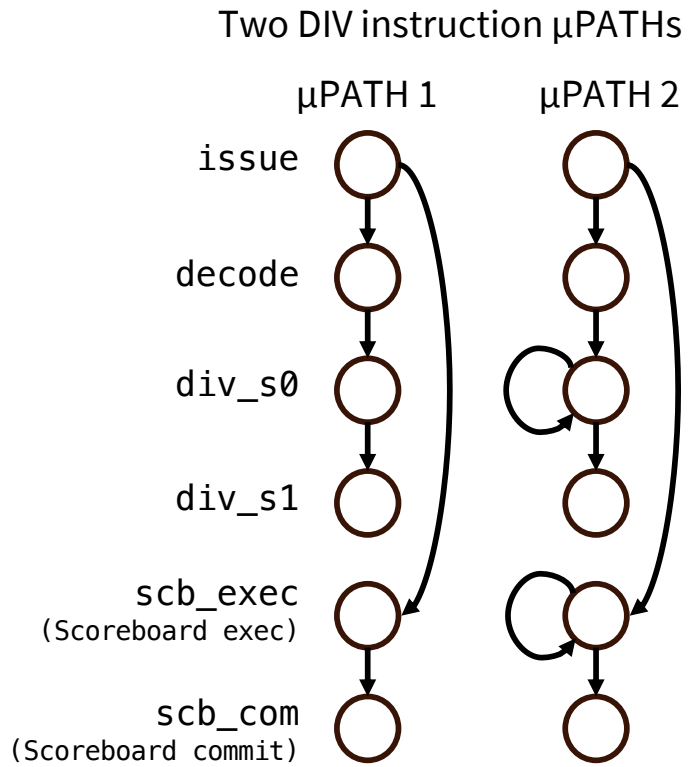
Stanford | ENGINEERING

SynthLC: RTL2MμPATH for leakage contract synthesis

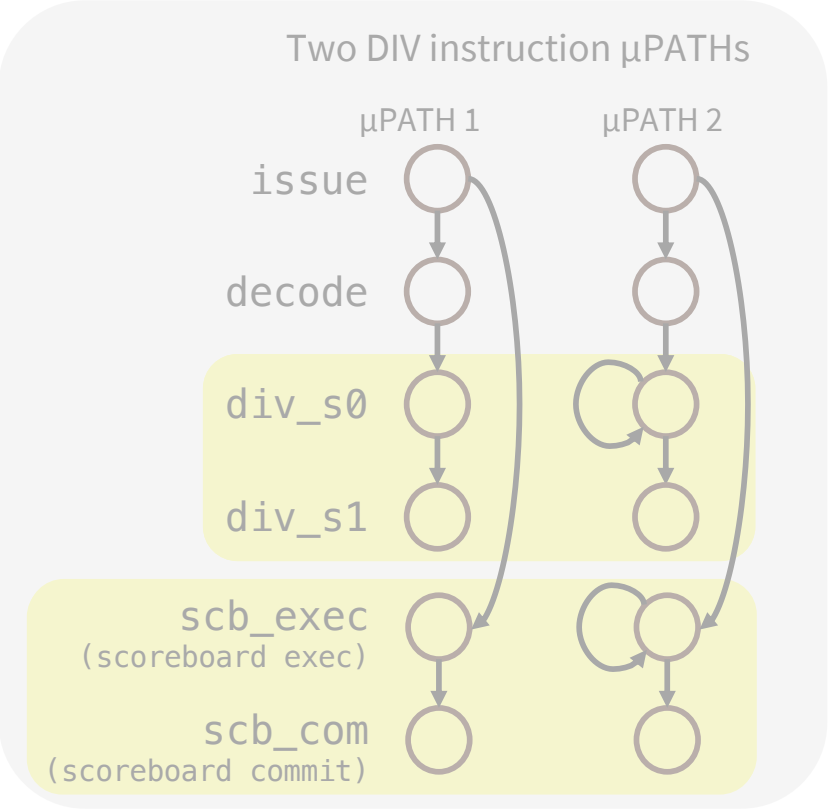
Recap: RTL2MμPATH discovers all μPATHs per instruction

SynthLC: Attribute μPATH variability to instructions' data operands using:

- information flow tracking
- property generation from templates
- model checking



Optimizing RTL2MμPATH for leakage contract synthesis



μPATH decision: `(source PL, {sets of follower PLs})`

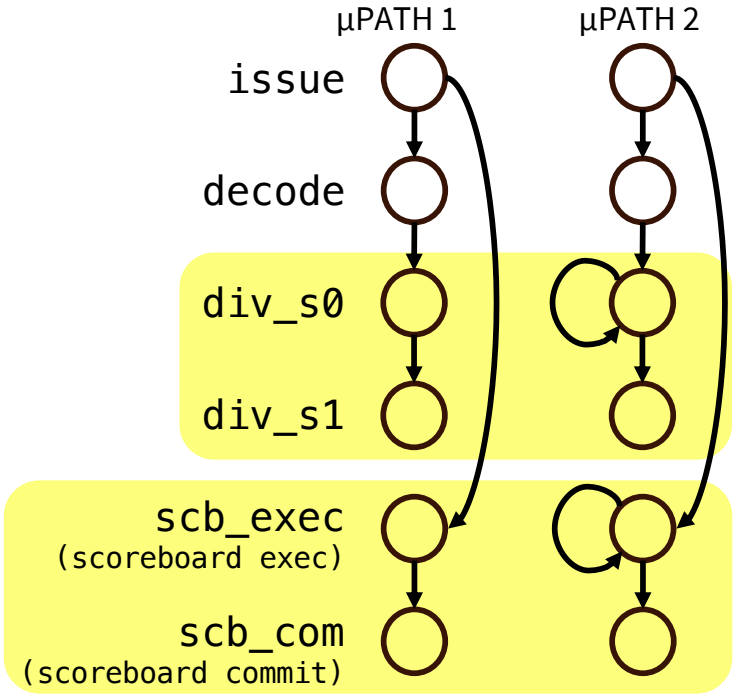
Decision 1: `(div_s0: {{div_s1}, {div_s0}})`

Decision 2: `(scb_exec: {{scb_com}, {scb_exec}})`

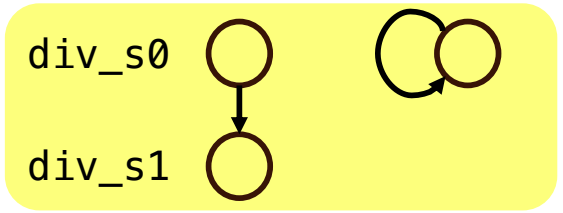
Goal: Optimize RTL2MμPATH to construct **μPATH decisions** without full μPATHS

Optimizing RTL2MμPATH for leakage contract synthesis

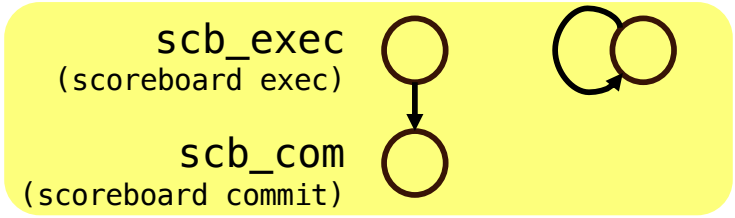
Two DIV instruction μPATHs



μPATH decision: (source PL, {sets of follower PLs})



Decision 1: (div_s0: {{div_s1}, {div_s0}})



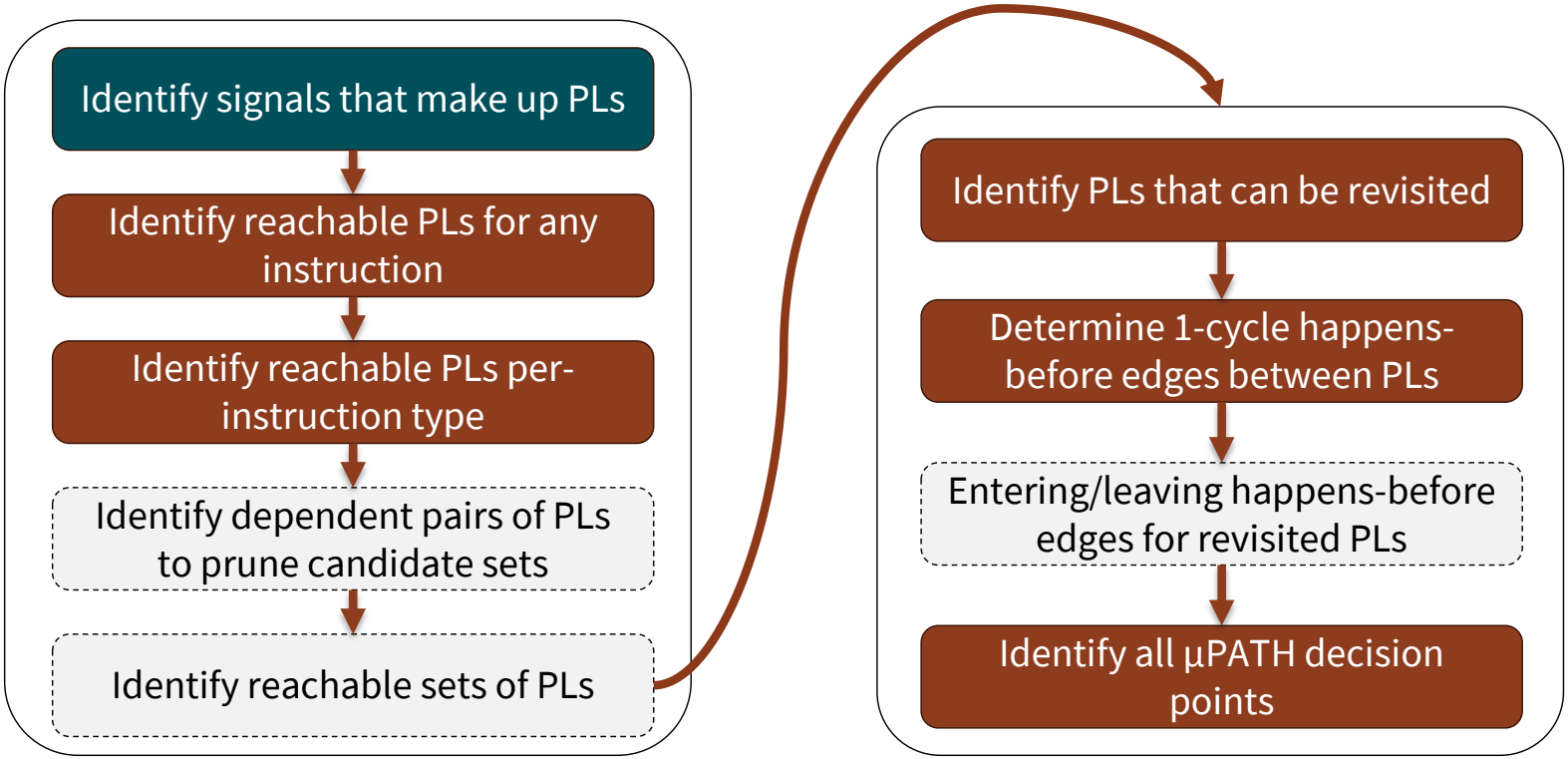
Decision 2: (scb_exec: {{scb_com}, {scb_exec}})

Goal: Optimize RTL2MμPATH to construct **μPATH decisions** without full μPATHS

Roadmap: Apply SynthLC on CVA6

- **Optimized RTL2MμPATH**
  - Step 1: [inside] Per-instruction PL reachability
  - Step 2: [inside] Obtain PL followers
  - Step 3: [inside] Construct decisions
- SynthLC: Leakage contract synthesis using symbolic IFT
  - Step 4: [inside] Determine if decisions are tainted under the intrinsic transmitter assumption
  - Step 5: [inside] Determine if decisions are tainted under the dynamic transmitter assumption

Optimized RTL2MμPATH for Decision Construction



Reduces the number of properties significantly

Per-design Steps: Identical to RTL2MμPATH

Setup:
1. PL annotation (input)

Two global steps:
1. Get all PLs reachable by any instruction
2. Generate DFG between all PL signals via static analysis

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

```
$ cd ~/fava_isca2026/synthlc_optimized/fv; ./setup_scripts.sh
```

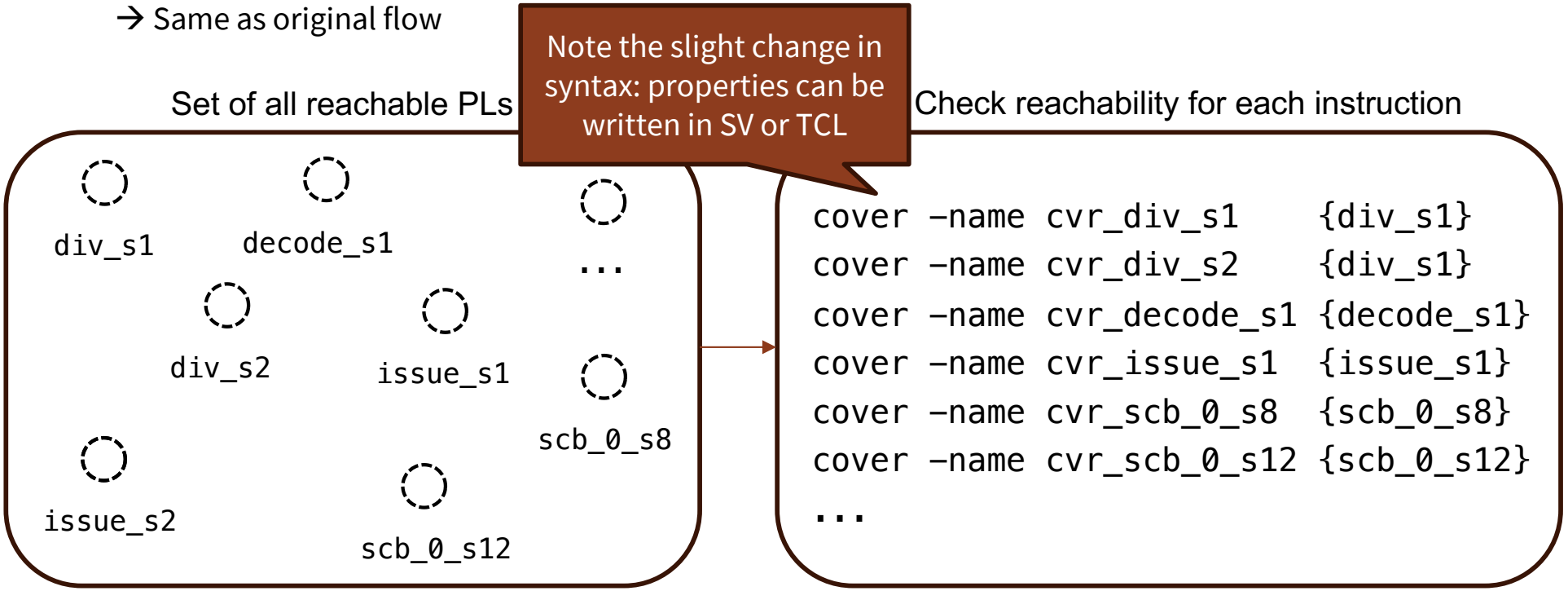
Roadmap: Apply SynthLC on CVA6

- **Optimized RTL2MμPATH**
  - **Step 1: [inside] Per-instruction PL reachability**
  - Step 2: [inside] IUV PL followers
  - Step 3: [inside] Construct decisions
- SynthLC: Leakage contract synthesis using symbolic IFT
  - Step 4: [inside] Determine if decisions are tainted under the intrinsic transmitter assumption
  - Step 5: [inside] Determine if decisions are tainted under the dynamic transmitter assumption

Step 1: Per-instruction PL Reachability

Which PLs can each instruction type visit?

→ Same as original flow



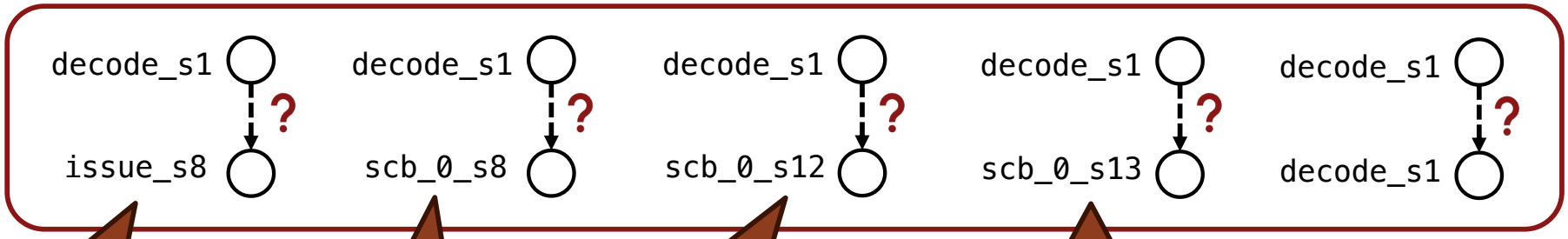
Roadmap: Apply SynthLC on CVA6

- **Optimized RTL2MμPATH**
  - Step 1: [inside] Per-instruction PL reachability
  - **Step 2: [inside] Obtain one-cycle PL followers**
  - Step 3: [inside] Construct decisions
- SynthLC: Leakage contract synthesis using symbolic IFT
  - Step 4: [inside] Determine if decisions are tainted under the intrinsic transmitter assumption
  - Step 5: [inside] Determine if decisions are tainted under the dynamic transmitter assumption

Step 2: One-cycle PL followers

Which PLs can be visited exactly one cycle after another?

- Candidate HB edges are determined by static analysis of a circuit's dataflow graph (DFG)



issued to the the mult/div unit

scoreboard: in-flight

scoreboard: results written back

scoreboard: committing

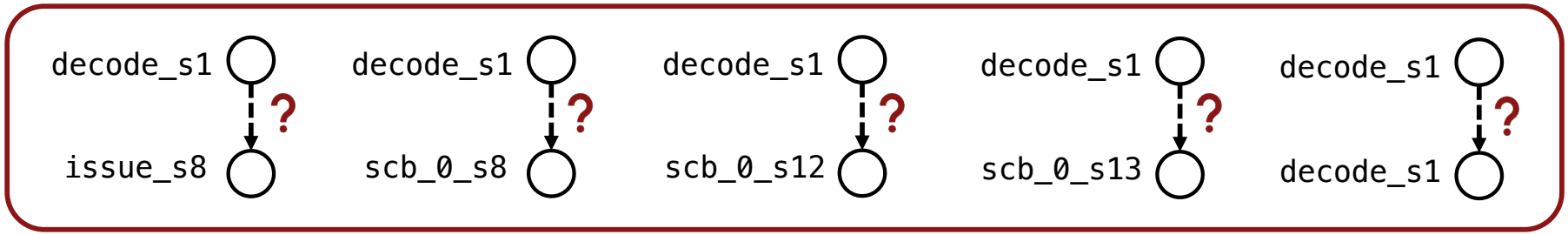
```
cover -name cvr_<PL_1>_HB_1_cyc_<PL_2>  {<PL_1> ##1 <PL_2>}
```

SVA syntax: One cycle later

Step 2: One-cycle PL followers

Which PLs can be visited exactly one cycle after another?

- Candidate HB edges are determined by static analysis of a circuit's dataflow graph (DFG)



Property template:

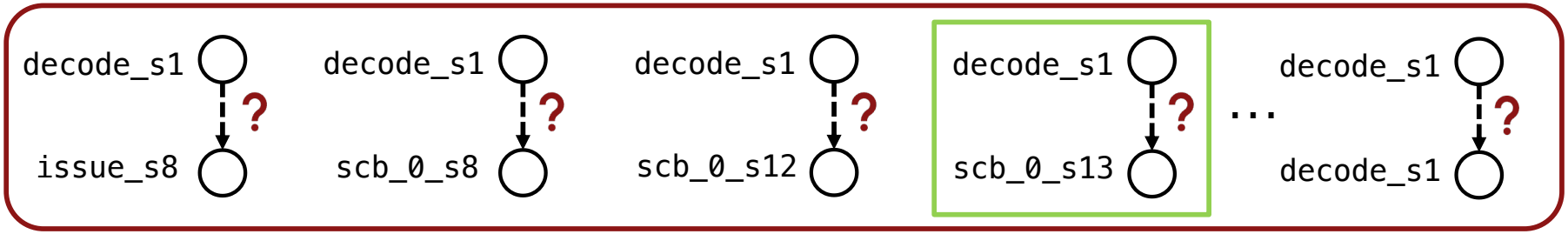
```
cover -name cvr_<PL_1>_HB_1_cyc_<PL_2>  {<PL_1> ##1 <PL_2>}
```

SVA syntax:
One cycle later

[Hands-On] Generate Properties for One-Cycle PL Followers

Step 2a: Open TCL file and fill in missing cover properties for follower reachability:

```
$ vim synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.tcl
```



Intermediate Output

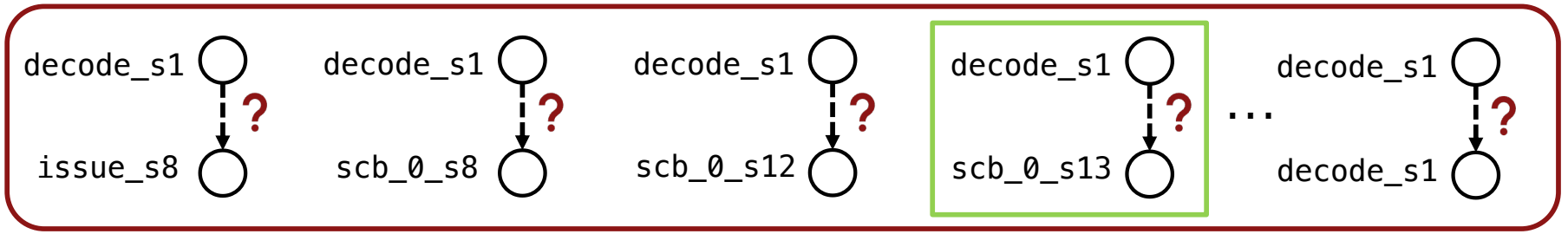
```
 7 cover -name cvr_decode_s1_HB_1_cyc_issue_s8  { decode_s1 ##1 issue_s8 }
 8 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s8  { decode_s1 ##1 scb_0_s8 }
 9 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s12 { decode_s1 ##1 scb_0_s12 }
10 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s13 { ________________________ }
 ...
61 cover -name cvr_decode_s1_HB_1_cyc_decode_s1 { decode_s1 ##1 decode_s1 }
```

i_DIV_out/xCoverAPerfLoc/rtl2mupath_candidate_HB.tcl

[Hands-On] Generate Properties for One-Cycle PL Followers

Step 2a: Open TCL file and fill in missing cover properties for follower reachability:

```
$ vim synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.tcl
```



Intermediate Output

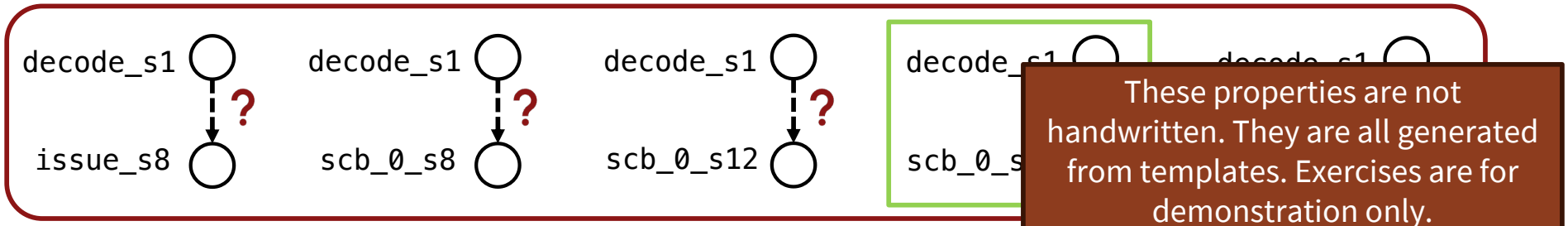
```
 7 cover -name cvr_decode_s1_HB_1_cyc_issue_s8  { decode_s1 ##1 issue_s8 }
 8 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s8  { decode_s1 ##1 scb_0_s8 }
 9 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s12 { decode_s1 ##1 scb_0_s12 }
10 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s13 { decode_s1 ##1 scb_0_s13 }
 ...
61 cover -name cvr_decode_s1_HB_1_cyc_decode_s1 { decode_s1 ##1 decode_s1 }
```

i_DIV_out/xCoverAPerfLoc/rtl2mupath_candidate_HB.tcl

[Hands-On] Generate Properties for One-Cycle PL Followers

Step 2a: Open TCL file and fill in missing cover properties for follower reachability:

```
$ vim synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.tcl
```



Intermediate Output

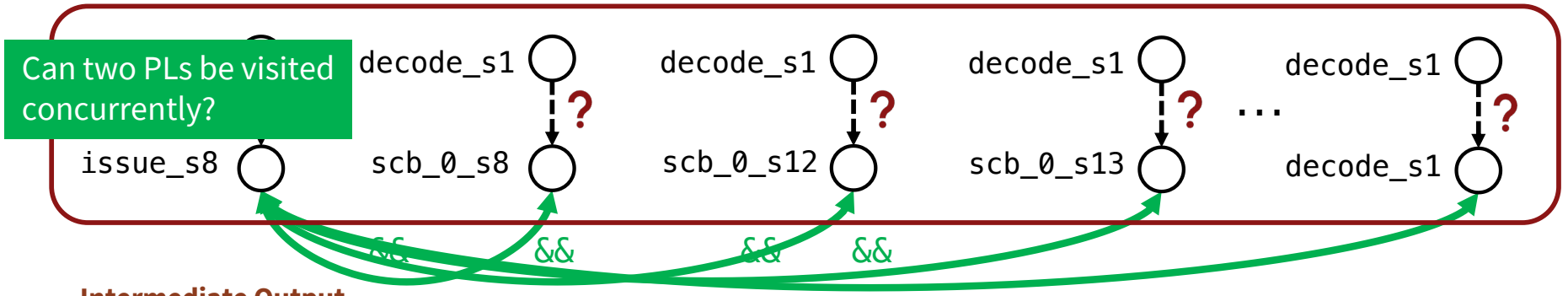
```
 7 cover -name cvr_decode_s1_HB_1_cyc_issue_s8  { decode_s1 ##1 issue_s8 }
 8 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s8  { decode_s1 ##1 scb_0_s8 }
 9 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s12 { decode_s1 ##1 scb_0_s12 }
10 cover -name cvr_decode_s1_HB_1_cyc_scb_0_s13 { decode_s1 ##1 scb_0_s13 }
 ...
61 cover -name cvr_decode_s1_HB_1_cyc_decode_s1 { decode_s1 ##1 decode_s1 }
```

i_DIV_out/xCoverAPerfLoc/rtl2mupath_candidate_HB.tcl

[Hands-On] Generate Properties for One-Cycle PL Followers

Step 2b: Check if each pair of PLs can be visited concurrently (for later filtering)

```
$ vim synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.tcl
```



Intermediate Output

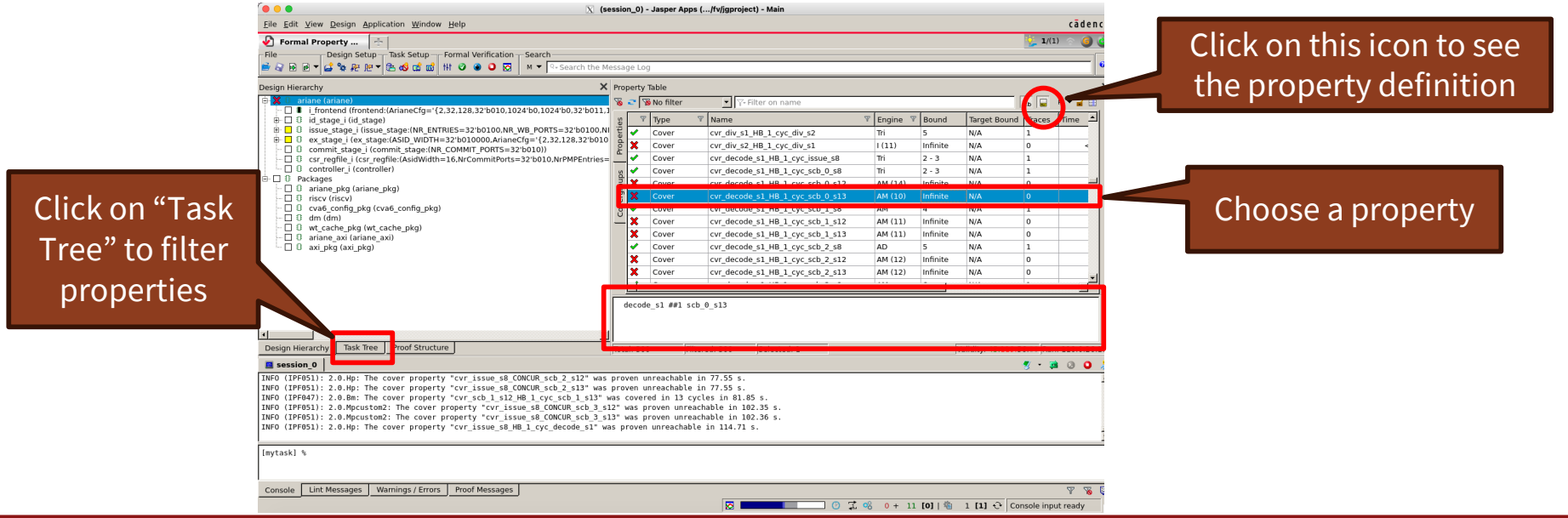
```
104 cover -name cvr_decode_s1_CONCUR_issue_s8 { decode_s1 && issue_s8 }
 ...
117 cover -name cvr_issue_s8_CONCUR_scb_0_s8 { issue_s8 && scb_0_s8 }
118 cover -name cvr_issue_s8_CONCUR_scb_0_s12 { issue_s8 && scb_0_s12 }
119 cover -name cvr_issue_s8_CONCUR_scb_0_s13 { issue_s8 && scb_0_s13 }
```

i_DIV_out/xCoverAPerfLoc/rtl2mupath_candidate_HB.tcl

[Hands-On] Check Properties for One-Cycle PL Followers

Step 2c: Get PLs that can be visited one cycle after (and concurrently as) another PL

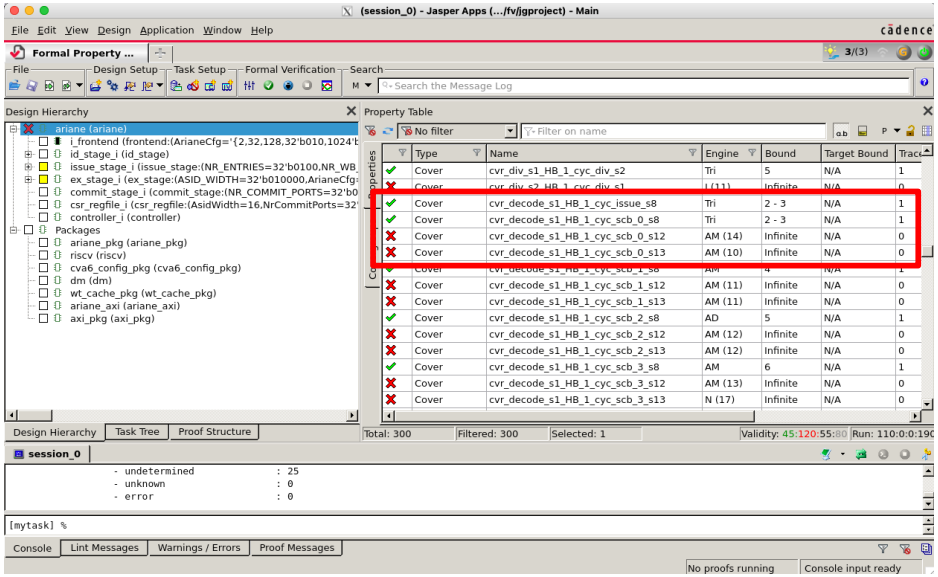
```
$ mkdir xEval; ./RUN_JG.sh -j xEval -s \
  synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.sv -t \
  synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.tcl -g 1
```



[Hands-On] Check Properties for One-Cycle PL Followers

Step 2c: Get PLs that can be visited one cycle after (and concurrently as) another PL

```
$ ./RUN_JG.sh -j xEval -s \
  synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.sv -t \
  synthlc/i_DIV_out/xCoverCandidateHBEdges/rtl2mupath_candidate_HB.tcl -g 1
```




```
cvr_decode_s1_HB_1_cyc_issue_s8
cvr_decode_s1_HB_1_cyc_scb_0_s8
cvr_decode_s1_HB_1_cyc_scb_0_s12
cvr_decode_s1_HB_1_cyc_scb_0_s13
```

✓ ✓ ✗ ✗

```
cvr_decode_s1_CONCUR_issue_s8
cvr_issue_s8_CONCUR_scb_0_s8
cvr_issue_s8_CONCUR_scb_0_s12
cvr_issue_s8_CONCUR_scb_0_s13
```

✗ ✓ ✗ ✗

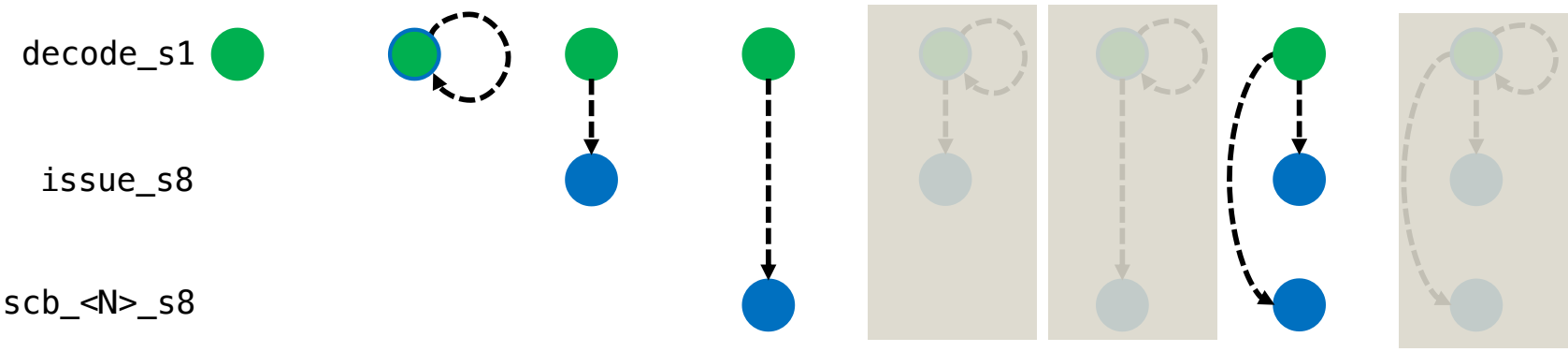
Roadmap: Apply SynthLC on CVA6

- **Optimized RTL2MμPATH**
  - Step 1: [inside] Per-instruction PL reachability
  - Step 2: [inside] Obtain one-cycle PL followers
  - **Step 3: [inside] Construct decisions**
- SynthLC: Leakage contract synthesis using symbolic IFT
  - Step 4: [inside] Determine if decisions are tainted under the intrinsic transmitter assumption
  - Step 5: [inside] Determine if decisions are tainted under the dynamic transmitter assumption

Step 3: Construct Decisions

Which sets of PLs are visited after another PL?

- In step 2, we found that the `decode_s1` PL can be followed by:
  1. `decode_s1` (stay in decode for another cycle)
  2. `issue_s8` (issued to the mult/div unit)
  3. `scb_<N>_s8` (in scoreboard & in-flight)
- Now we need to find all subsets of these 3 PLs that can be visited exactly one cycle after `decode_s1`



Step 3: Construct Decisions

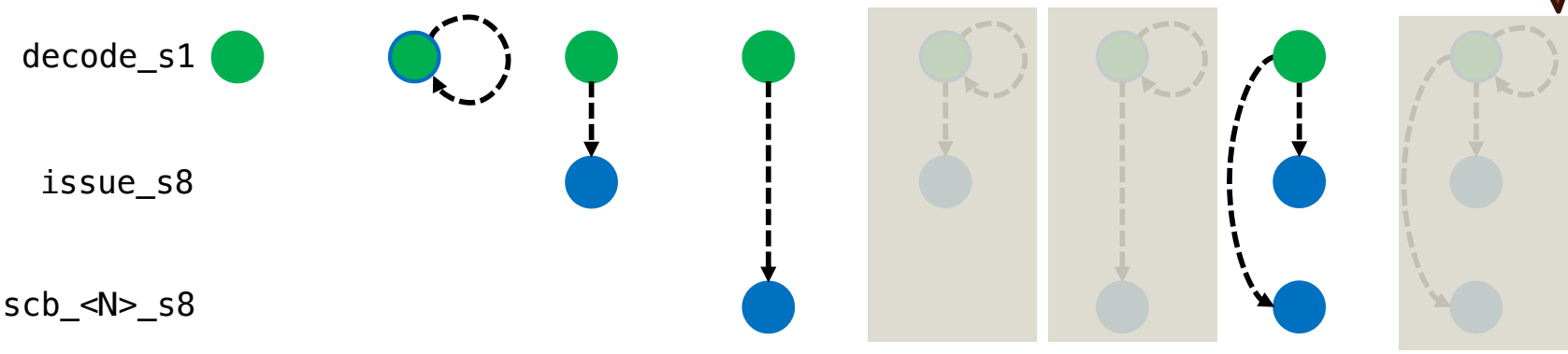
Which sets of PLs are visited after another PL?

- In step 2, we found that the `decode_s1` PL can be followed by:
  1. `decode_s1` (stay in decode for another cycle)
  2. `issue_s8` (issued to the mul
  3. `scb_<N>_s8` (in scoreboard &
- Now we need to find all subsets of these 3 PLs that can be visited exactly one cycle after `decode_s1`

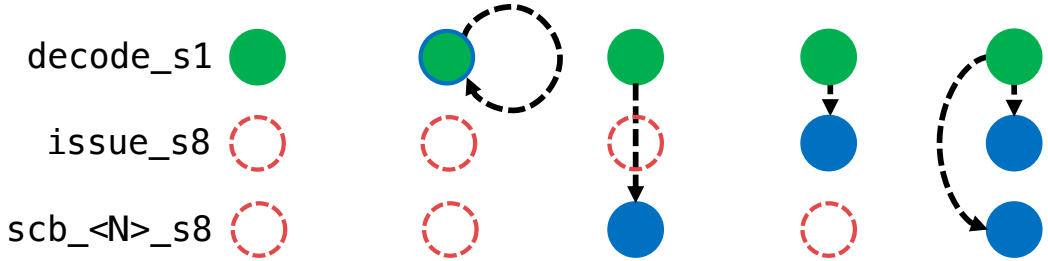
Decision source

Decision destination sets

Ruled out via concurrent PL filtering in step 2



Step 3: Generate Properties for Decision Construction



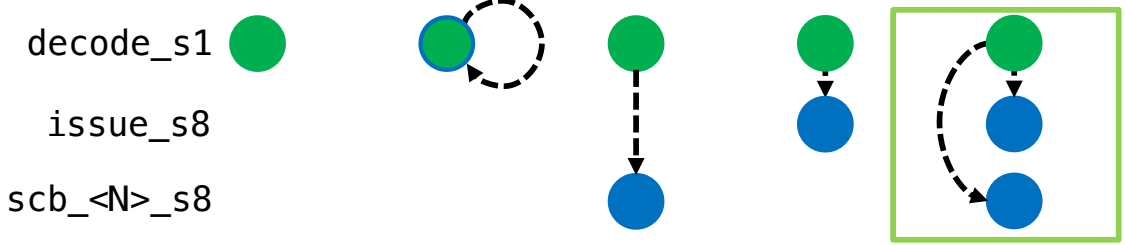
Property template:

```
cover -name cvr_src_<DECISION_SRC>_dest_set__<DEST_SET>
    { <DECISION_SRC> ##1
      ((<CONJUNCTION_OF_DEST_SET> & 1'b1 ) &
       !(<DISJUNCTION_OF_OTHER_FOLLOWERS> | 1'b0 )
      )
    }
```

[Hands-On] Generate Properties for Decision Construction

Step 3a: Open TCL file and fill in missing source & destination set in property:

```
$ vim synthlc/i_DIV_out/xFollowerSetsOnly/rtl2mupath_followers.tcl
```



Intermediate Output

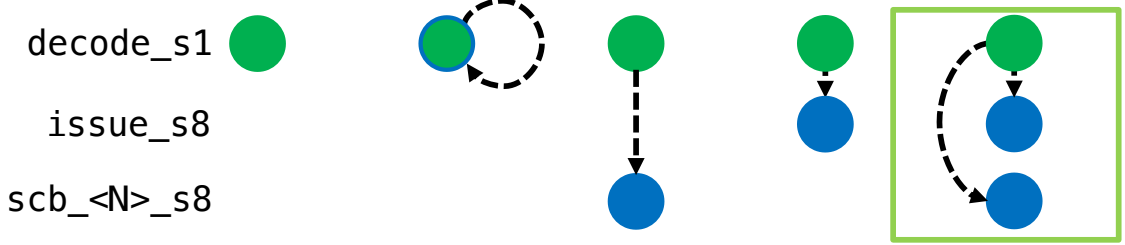
```
 8 cover -name cvr_src_decode_s1_dest_set_ {decode_s1 ##1 (( 1'b1 ) & ! (issue_s8 | scb_0_s8 | scb_1_s8 |
scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
 9 cover -name cvr_src_decode_s1_dest_set__decode_s1 {decode_s1 ##1 ((decode_s1 & 1'b1 ) & ! (issue_s8 |
scb_0_s8 | scb_1_s8 | scb_2_s8 | scb_3_s8 | 1'b0 )) }
...
13 cover -name cvr_src_decode_s1_dest_set__scb_0_s8 {decode_s1 ##1 ((scb_0_s8 & 1'b1 ) & ! (issue_s8 |
scb_1_s8 | scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
14 cover -name cvr_src_decode_s1_dest_set__issue_s8 {decode_s1 ##1 ((issue_s8 & 1'b1 ) & ! (scb_0_s8 |
scb_1_s8 | scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
...
18 cover -name cvr_src_decode_s1_dest_set__issue_s8_scb_0_s8 {_________ ##1 ((________ & ________ & 1'b1
) & ! (scb_1_s8 | scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
```

i_DIV_out/xFollowerSetsOnly/rtl2mupath_followers.tcl

[Hands-On] Generate Properties for Decision Construction

Step 3a: Open TCL file and fill in missing source & destination set in property:

```
$ vim synthlc/i_DIV_out/xFollowerSetsOnly/rtl2mupath_followers.tcl
```



Intermediate Output

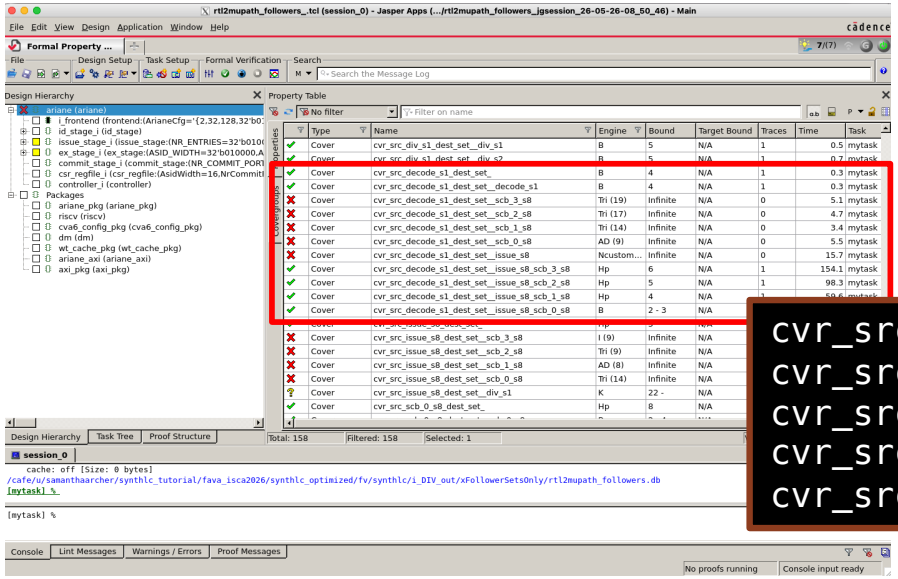
```
 8 cover -name cvr_src_decode_s1_dest_set_ {decode_s1 ##1 (( 1'b1 ) & ! (issue_s8 | scb_0_s8 | scb_1_s8 |
scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
 9 cover -name cvr_src_decode_s1_dest_set__decode_s1 {decode_s1 ##1 ((decode_s1 & 1'b1 ) & ! (issue_s8 |
scb_0_s8 | scb_1_s8 | scb_2_s8 | scb_3_s8 | 1'b0 )) }
...
13 cover -name cvr_src_decode_s1_dest_set__scb_0_s8 {decode_s1 ##1 ((scb_0_s8 & 1'b1 ) & ! (issue_s8 |
scb_1_s8 | scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
14 cover -name cvr_src_decode_s1_dest_set__issue_s8 {decode_s1 ##1 ((issue_s8 & 1'b1 ) & ! (scb_0_s8 |
scb_1_s8 | scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
...
18 cover -name cvr_src_decode_s1_dest_set__issue_s8_scb_0_s8 {decode_s1 ##1 ((issue_s8 & scb_0_s8 & 1'b1
) & ! (scb_1_s8 | scb_2_s8 | scb_3_s8 | decode_s1 | 1'b0 )) }
```

i_DIV_out/xFollowerSetsOnly/rtl2mupath_followers.tcl

[Hands-On] Check Properties for Decision Construction

Step 3b: Determine the sets of PLs that can be visited one cycle after the source PL

```
$ ./RUN_JG.sh -j xEval -s \
  synthlc/i_DIV_out/xFollowerSetsOnly/rtl2mupath_followers.sv -t \
  synthlc/i_DIV_out/xFollowerSetsOnly/rtl2mupath_followers.tcl -g 1
```




```
cvr_src_decode_s1_dest_set_
cvr_src_decode_s1_dest_set__decode_s1
cvr_src_decode_s1_dest_set__scb_0_s8
cvr_src_decode_s1_dest_set__issue_s8
cvr_src_decode_s1_dest_set__issue_s8_scb_0_s8
```

[Hands-On] Check Properties for DIV Decisions

Step 3c: Inspect the resulting decisions for a DIV instruction

```
$ vim synthlc/i_DIV_out/xFollowerSetsOnly/decisions.txt
```

Output

```
1 div_s1, [[], ['div_s1'], ['div_s2']]
```

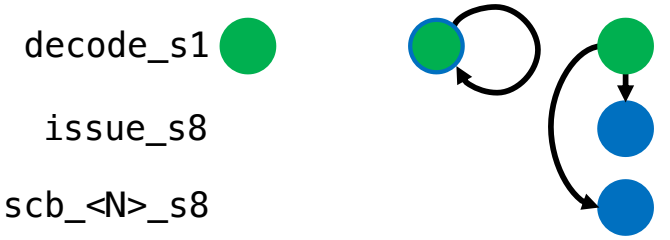
i_DIV_out/xFollowerSetsOnly/decisions.txt

Describes μPATHs where DIV takes variable cycles in serial division unit

Output

```
2 decode_s1, [[], ['decode_s1'], ['issue_s8', 'scb_3_s8'],
   ['issue_s8', 'scb_2_s8'], ['issue_s8', 'scb_1_s8'], ['issue_s8', 'scb_0_s8']]
```

i_DIV_out/xFollowerSetsOnly/decisions.txt



Describes μPATHs where DIV remains in decode for more than one cycle before being issued and varies which scoreboard entry is occupied

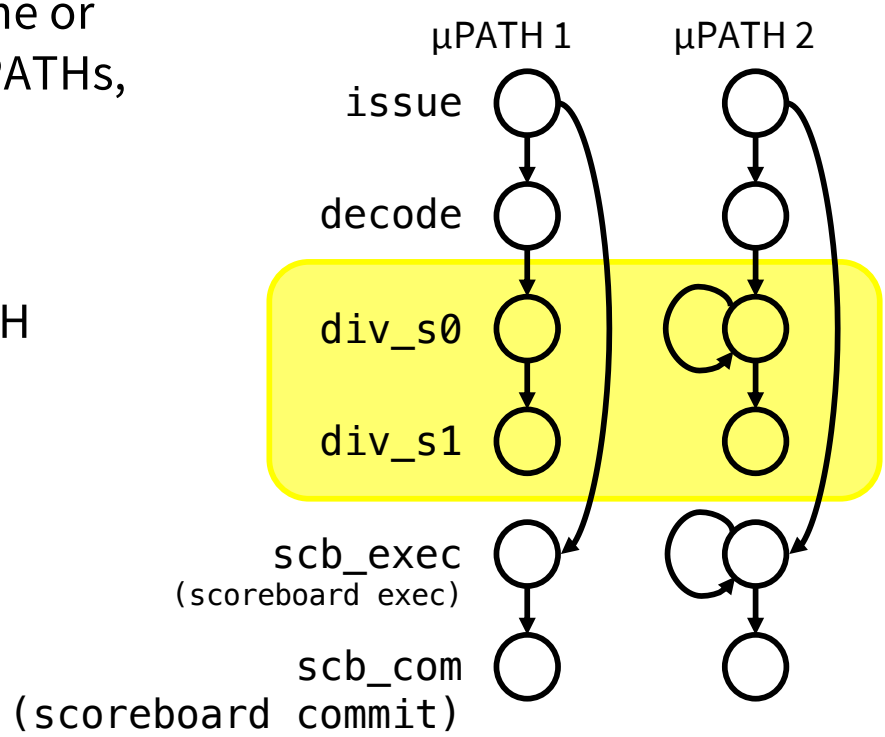
Roadmap: Apply SynthLC on CVA6

- Optimized RTL2MμPATH
  - Step 1: [inside] Per-instruction PL reachability
  - Step 2: [inside] Obtain one-cycle PL followers
  - Step 3: [inside] Construct decisions
- **SynthLC: Leakage contract synthesis using symbolic IFT**
  - **Step 4: [inside] Determine if decisions are tainted under the intrinsic transmitter assumption**
  - Step 5: [inside] Determine if decisions are tainted under the dynamic transmitter assumption

SynthLC: Attribute μPATH Variability to Instruction Operands

If an instruction's operand causes the same or another instruction to exhibit different μPATHs, this indicates the operand can leak via a hardware side-channel

How do we prove operands "cause" μPATH variability? **Symbolic information flow**



Symbolic Information Flow Background

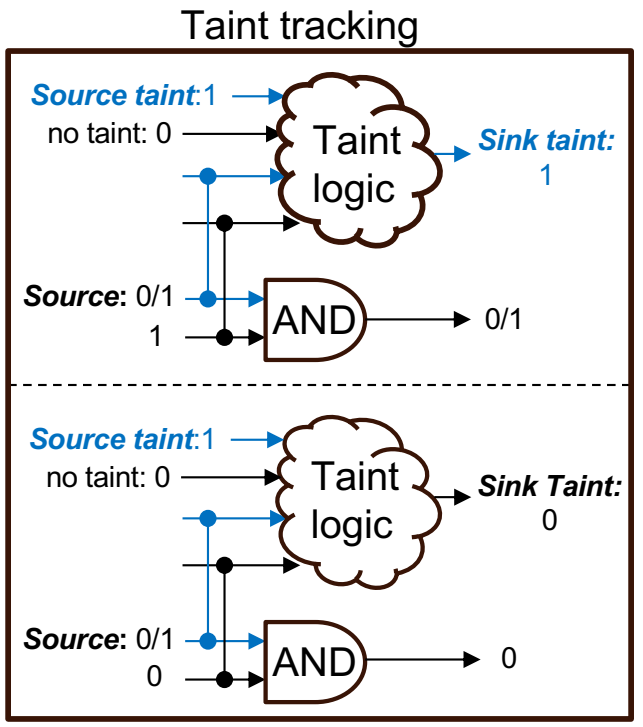
Two methods for symbolic information flow:

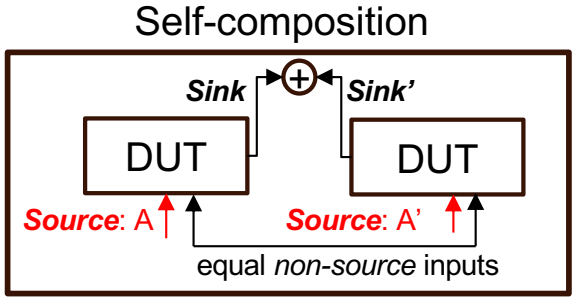
Taint tracking:

- Augment the design one bit of *taint* per state bit in the design, connected via *taint tracking* logic
- Taint is injected at the *source taint signal* and checked at the *sink taint signal*

Self-composition:

- Instantiate two copies of the same design with all *non-source* inputs constrained to be equal
- Determine whether a *sink signal* in each design copy can have different values





Figures from [Ceesay-Seitz et al. ICCAD '25]

Taint tracking: Instrument the Design with IFT Logic

- For this tutorial, CVA6 has already been instrumented with taint logic using CellIFT [Solt+, USENIX Sec'22] (paper, repo)
- CellIFT adds taint logic using a custom compilation pass in the open-source synthesis tool Yosys [Wolf+, 2013]
- Instrumented design files can be found in directories:
  - `fava_isca2026/synthlc_optimized/fv/src_ift`
  - `fava_isca2026/synthlc_optimized/fv/src_ift_2flag`

Step 4: IFT Properties for Decisions Under Intrinsic Transmitter Assumption

Assume that taint originates at DIV instruction's own register operands:

Intermediate output

```
11178 wire read_op_i = (issue_stage_i.i_issue_read_operands.pc_o == pc0);
...
11197 I0_T0: assume property (@(posedge clk_i)
11198     read_op_i |-> issue_stage_i.i_issue_read_operands.operand_a_q_t0 == {64{1'b1}});
11199 I0_T1: assume property (@(posedge clk_i)
11200     !read_op_i |-> issue_stage_i.i_issue_read_operands.operand_a_q_t0 == {64{1'b0}});
11201 I0_T2: assume property (@(posedge clk_i)
11202     issue_stage_i.i_issue_read_operands.operand_b_q_t0 == {64{1'b0}});
```

Flag when pc0 reads its operands

Taint is introduced at operand rs1 read_op_i is high

synthlc/i_DIV_out/xIftIntrinsic/ift_intr_rtl2mupath_taint_rs1_top.sv

For each destination set in each decision, one property is generated from the template:

```
cover -name taint_rs1_src_<DECISION_SRC>_dest_<i>
   {<DECISION_SRC> ##1
      (<DECISION_DEST_SET_EXACT> && (|{<DECISION_DEST_SET_TAINTED>,1'b0})
   }
```

Generated index of destination set

Exact implies only the dest set PLs and no other followers

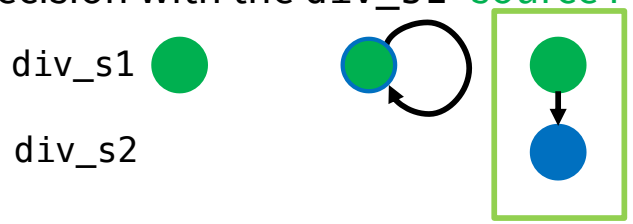
SVA syntax: Disjunction over a set of wires

[Hands-On] Generate IFT Properties for DIV Decisions

Step 4a: Open TCL file and fill in the source, destination set, and taint signal in an IFT property:

```
$ vim synthlc/i_DIV_out/xIftIntrinsic/ift_intr_rtl2mupath_taint_rs1.tcl
```

- Recall the decision with the `div_s1` source PL and 3 destination sets:



Describes μPATHs where DIV takes variable cycles in serial division unit

Intermediate Output

```
1 cover -name {taint_rs1_src_div_s1_dest_0} {@(posedge clk_i) div_s1 ##1 ( !div_s2 &&
!div_s1 && 1'b1 && (|{div_s2_t0, div_s1_t0, 1'b0}))}
2 cover -name {taint_rs1_src_div_s1_dest_1} {@(posedge clk_i) div_s1 ##1 ( !div_s2 &&
div_s1 && 1'b1 && (|{div_s1_t0, 1'b0}))}
3 cover -name {taint_rs1_src_div_s1_dest_2} {@(posedge clk_i) ______ ##1 (
__________________ && 1'b1 && (|{_________, 1'b0}))}
```

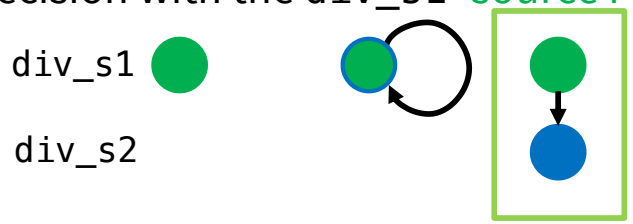
synthlc/i_DIV_out/xIftIntrinsic/ift_intr_rtl2mupath_taint_rs1.tcl

[Hands-On] Generate IFT Properties for DIV Decisions

Step 4a: Open TCL file and fill in the source, destination set, and taint signal in an IFT property:

```
$ vim synthlc/i_DIV_out/xIftIntrinsic/ift_intr_rtl2mupath_taint_rs1.tcl
```

- Recall the decision with the `div_s1` source PL and 3 destination sets:



Describes μPATHs where DIV takes variable cycles in serial division unit

Intermediate Output

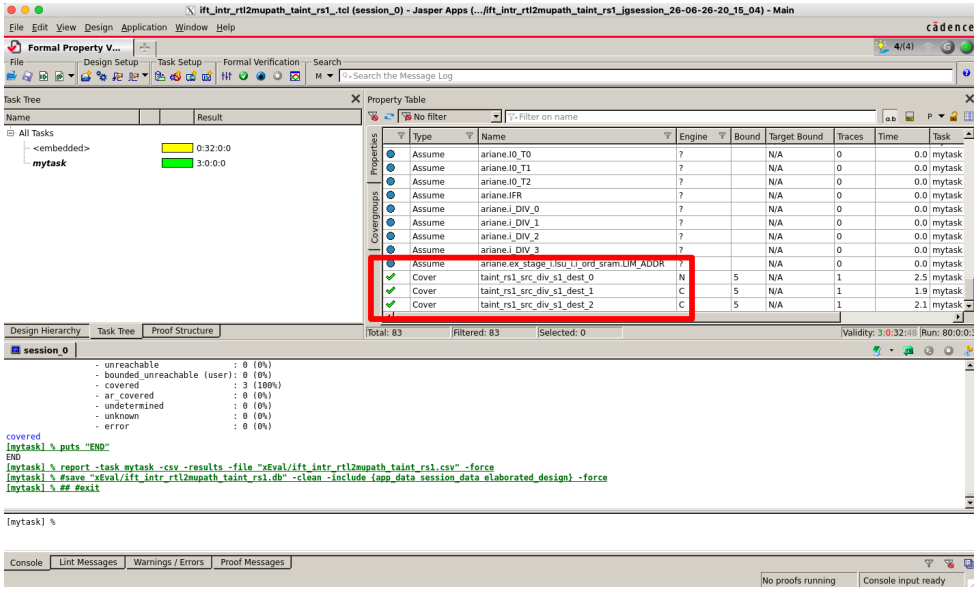
```
1 cover -name {taint_rs1_src_div_s1_dest_0} {@(posedge clk_i) div_s1 ##1 ( !div_s2 &&
!div_s1 && 1'b1 && (|{div_s2_t0, div_s1_t0, 1'b0}))}
2 cover -name {taint_rs1_src_div_s1_dest_1} {@(posedge clk_i) div_s1 ##1 ( !div_s2 &&
div_s1 && 1'b1 && (|{div_s1_t0, 1'b0}))}
3 cover -name {taint_rs1_src_div_s1_dest_2} {@(posedge clk_i) div_s1 ##1 ( !div_s1 &&
div_s2 && 1'b1 && (|{div_s2_t0, 1'b0}))}
```

synthlc/i_DIV_out/xIftIntrinsic/ift_intr_rtl2mupath_taint_rs1.tcl

[Hands-On] Check Intrinsic IFT Properties for DIV Decisions

Step 4b: Determine if the decision destinations can become tainted

```
$ ./setup_scripts_ift.sh; ./RUN_JG_ift.sh -j xEval -s \
 synthlc/i_DIV_out/xIftIntrinsic/ift_intr_rtl2mupath_taint_rs1.sv -t \
 synthlc/i_DIV_out/xIftIntrinsic/ift_intr_rtl2mupath_taint_rs1.tcl -g 1
```



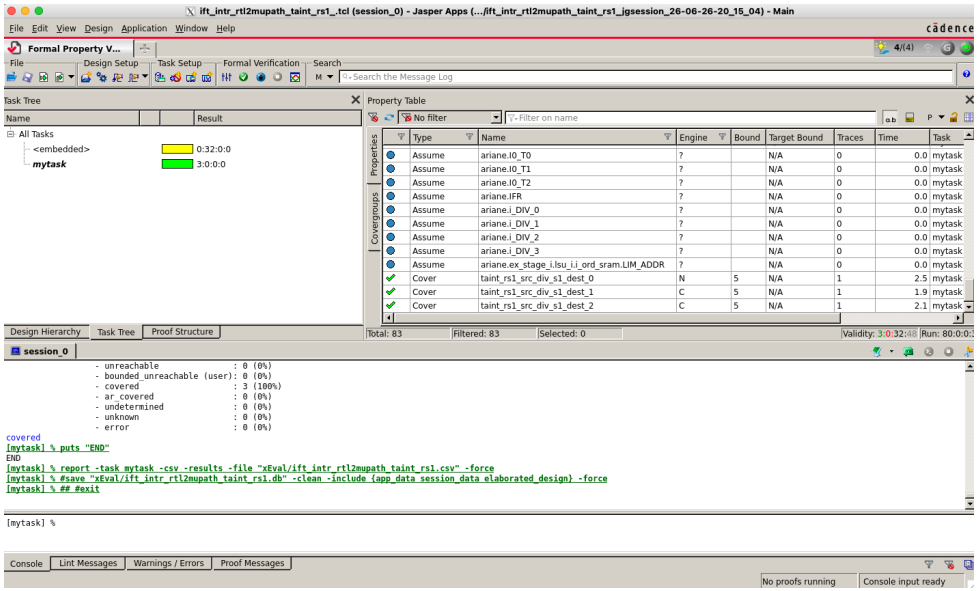
Decision depends intrinsically on the DIV's first operand value.

DIV instruction is classified as an intrinsic transmitter.

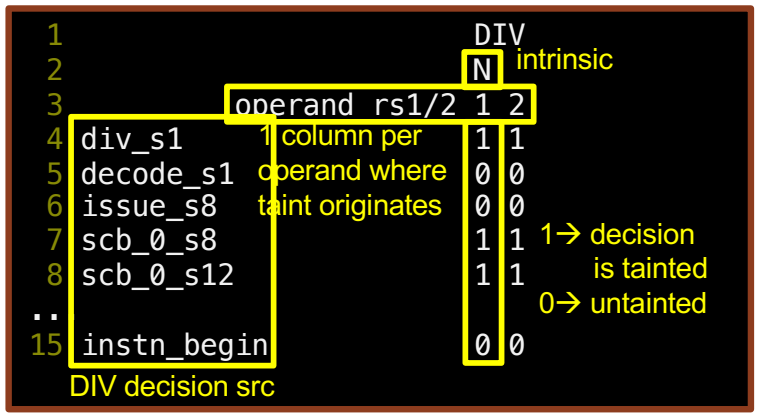
[Hands-On] Check Intrinsic IFT Properties for DIV Decisions

Step 4c: Examine the summary of intrinsic transmitter results

```
$ vim synthlc/i_DIV_out/xIftDynamic/leakage_signature.txt
```



Output



`synthlc/i_DIV_out/xIftDynamic/leakage_signature.txt`

Roadmap: Apply SynthLC on CVA6

- Optimized RTL2MμPATH
  - Step 1: [inside] Per-instruction PL reachability
  - Step 2: [inside] Obtain one-cycle PL followers
  - Step 3: [inside] Construct decisions
- **SynthLC: Leakage contract synthesis using symbolic IFT**
  - Step 4: [inside] Determine if decisions are tainted under the intrinsic transmitter assumption
  - **Step 5: [inside] Determine if decisions are tainted under the dynamic transmitter assumption**

Step 5: Dynamic Transmitter/Transponder Interactions

- Assume that a load (LW) is the transponder (decisions have been pre-generated in `synthlc/i_LW_out/xFollowerSetsOnly/decisions.txt`)
- Below we have a decision for a LW instruction where issue is the source and there are three destination sets

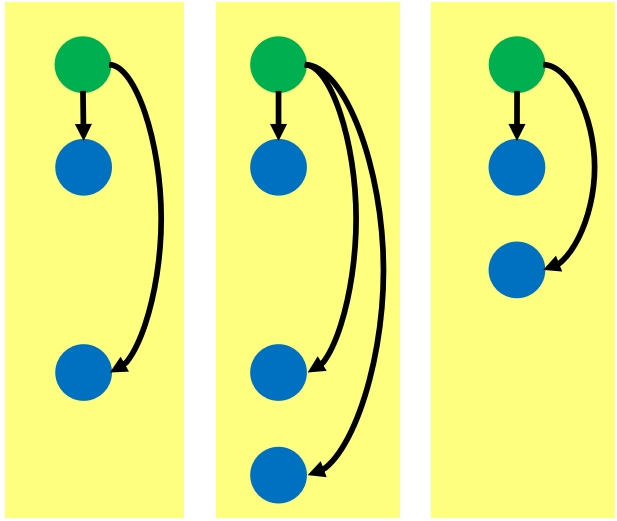
`issue_s16`
(`issue to LSU`)

`scb_N_s8`
(`scoreboard in flight`)

`load_unit_s1`
(`load data ready`)

`lsq_enq_N_s1`
(`enqueued in LSQ`)

`load_unit_op_s3`
(`load stall due to page offset conflict`)



Step 5: IFT Properties for Decisions Under Dynamic Transmitter Assumption

- A transmitter is classified as *dynamic* if its operands cause μPATH variability for a different in-flight instruction

Intermediate Output

```
11141 pc1_i0_assoc_1: assume property (@(posedge clk_i)
11142     id_stage_i.\fetch_entry_i.address == pc1 |-> id_stage_i.instruction == i1);
...
11182 wire read_op_i = (issue_stage_i.i_issue_read_operands.pc_o == pc1);
...
11197 I0_T0: assume property (@(posedge clk_i)
11198     read_op_i |-> issue_stage_i.i_issue_read_operands.operand_a_q_t0 == {64{1'b1}});
11199 I0_T1: assume property (@(posedge clk_i)
11200     !read_op_i |-> issue_stage_i.i_issue_read_operands.operand_a_q_t0 == {64{1'b0}});
11201 I0_T2: assume property (@(posedge clk_i)
11202     issue_stage_i.i_issue_read_operands.operand_b_q_t0 == {64{1'b0}});
...
11244 group_4_i1: assume property (
11245 ((i1[14:12] == 3'b010) && (i1[6:0] == 7'b0100011) && 1'b1) ||
11246 1'b0);
```

synthlc/i_LW_out/xIftDynamic/ift_dyn_rtl2mupath_taint_rs1_group4_top.sv

pc1 is associated with instruction i1

Flag when pc1 reads its operands

Assume taint is introduced at operand rs1 only if read_op_i is 1

Assume instruction i1 is a store

Step 5: IFT Properties for Decisions Under Dynamic Transmitter Assumption

Dynamic transmitter property requires that both instructions are in flight at the same time in the precondition.

Property template:

Generated index of destination set

```
cover –name taint_rs1_src_<DECISION_SRC>_dest_<i>
   {(<DECISION_SRC> && i1_in_some_pl) ##1
      (<DECISION_DEST_SET_EXACT> && (|{<DECISION_DEST_SET_TAINTED>,1'b0})
   }
```

where **`i1_in_some_pl`** is 1 when instruction `i1` is inflight (occupying any PL)

[Hands-On] IFT Properties for Decisions Under Dynamic Transmitter Assumption

Step 5a: Open TCL file and fill in an IFT property for dynamic transmitter/transponder pair

```
cover -name taint_rs1_src_<DECISION_SRC>_dest_<i>
    {(<DECISION_SRC> && i1_in_some_pl) ##1
       (<DECISION_DEST_SET_EXACT> && (|{<DECISION_DEST_SET_TAINTED>,1'b0})
    }
```

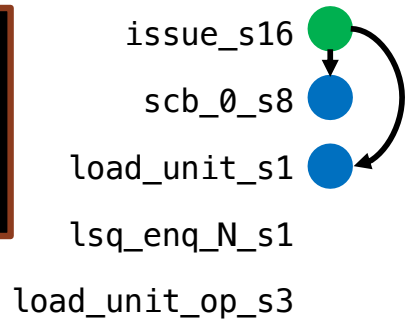
Generated index of destination set

```
$ vim synthlc/i_LW_out/xIftDynamic/ift_dyn_rtl2mupath_taint_rs1_group4.tcl
```

Intermediate Output

```
23 cover -name {taint_rs1_src_issue_s16_dest_4} {@(posedge clk_i) (_________ &&
____________) ##1 ( ________ && ____________ && !scb_1_s12 && !load_unit_op_s3 &&
!scb_0_s13 && !scb_2_s12 && !scb_3_s13 && !scb_2_s13 && !scb_2_s8 && !issue_s16 &&
!scb_1_s13 && !decode_s1 && !lsq_enq_0_s1 && !lsq_enq_1_s1 && !scb_0_s12 &&
!load_unit_buff_s1 && !scb_3_s12 && !load_unit_op_s1 && !scb_1_s8 && !mem_req_s1 &&
!scb_3_s8 && !load_unit_op_s2 && 1'b1 && (|{__________, _____________, 1'b0}))}
```

synthlc/i_LW_out/xIftDynamic/ift_dyn_rtl2mupath_taint_rs1_group4.tcl



[Hands-On] IFT Properties for Decisions Under Dynamic Transmitter Assumption

Step 5a: Open TCL file and fill in an IFT property for dynamic transmitter/transponder pair

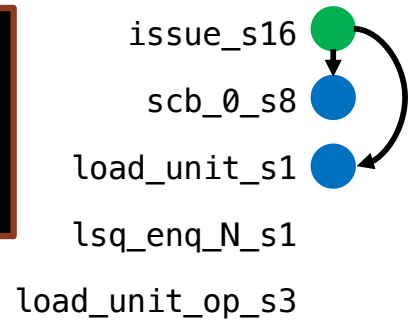
```
cover -name taint_rs1_src_<DECISION_SRC>_dest_<i>
   {(<DECISION_SRC> && i1_in_some_pl) ##1
      (<DECISION_DEST_SET_EXACT> && (|{<DECISION_DEST_SET_TAINTED>,1'b0})
   }
```

```
$ vim synthlc/i_LW_out/xIftDynamic/ift_dyn_rtl2mupath_taint_rs1_group4.tcl
```

Intermediate Output

```
23 cover -name {taint_rs1_src_issue_s16_dest_4} {@(posedge clk_i) (issue_s16 &&
i1_in_some_pl) ##1 ( scb_0_s8 && load_unit_s1 && !scb_1_s12 && !load_unit_op_s3 &&
!scb_0_s13 && !scb_2_s12 && !scb_3_s13 && !scb_2_s13 && !scb_2_s8 && !issue_s16 &&
!scb_1_s13 && !decode_s1 && !lsq_enq_0_s1 && !lsq_enq_1_s1 && !scb_0_s12 &&
!load_unit_buff_s1 && !scb_3_s12 && !load_unit_op_s1 && !scb_1_s8 && !mem_req_s1 &&
!scb_3_s8 && !load_unit_op_s2 && 1'b1 && (|{scb_0_s8_t0, load_unit_s1_t0, 1'b0}))}
```

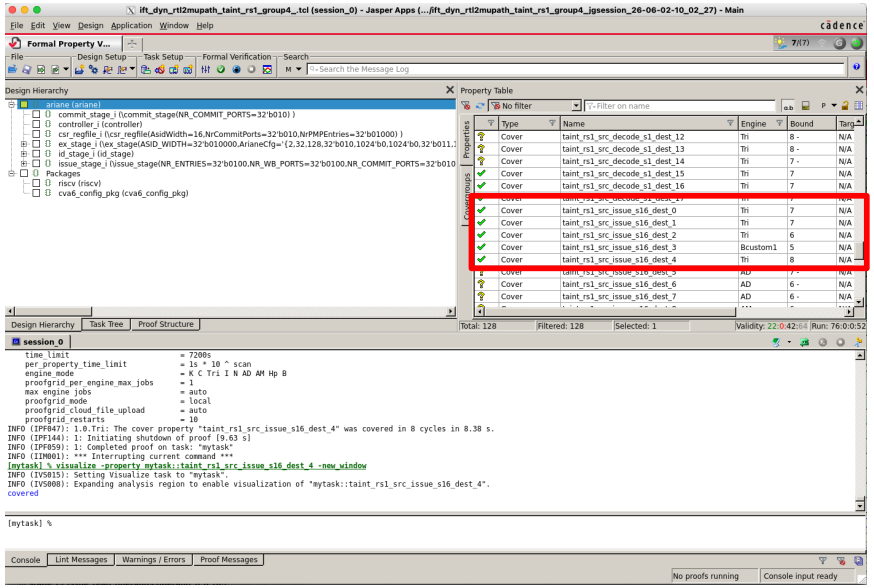
synthlc/i_LW_out/xIftDynamic/ift_dyn_rtl2mupath_taint_rs1_group4.tcl



[Hands-On] Check Dynamic IFT Properties for LW Decisions

Step 5b: Determine if the decision destination can be tainted via dynamic store:

```
$ ./RUN_JG_ift.sh -j xEval -s \
 synthlc/i_LW_out/xIftDynamic/ift_dyn_rtl2mupath_taint_rs1_group4.sv -t \
 synthlc/i_LW_out/xIftDynamic/ift_dyn_rtl2mupath_taint_rs1_group4.tcl -g 1
```



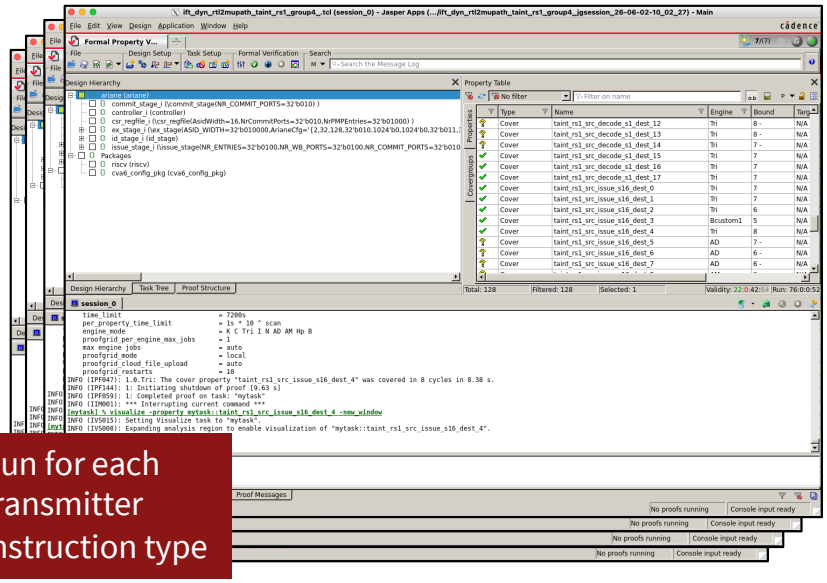
Decision depends on a dynamic store instruction's address operand value.

Store is classified as dynamic transmitter that can cause μPATH variability for a load transponder.

[Hands-On] Check Dynamic IFT Properties for LW Decisions

Step 5c: Examine the summary of dynamic transmitter results (i.e. complete *leakage signature*):

```
$ vim synthlc/i_LW_out/xIftDynamic/leakage_signature.txt
```



Run for each transmitter instruction type

Output

Dynamic transmitter

```
                    DIV     ADD     LW      SW      BEQ
                    N   D   N   D   N   D   N   D   N   D
operand rs1/2       1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
decode_s1           0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 1
issue_s16           0 0 1 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 1
```

LD decision src

1 column per operand where taint originates

synthlc/i_LW_out/xIftDynamic/leakage_signature.txt

Results for Other Transponders

- To run all SynthLC steps for another transponder instruction (`instr`) under intrinsic and dynamic transmitter assumptions:

```
$ cd synthlc; run_an_instn_optimized.sh <instr>.sv
```

- Decisions for a transponder instruction (`instr`) can be found in the decisions output files:

```
$ synthlc/i_<instr>_out/xFollowerSetsOnly/decisions.txt
```

- Leakage signatures for each transponder instruction (`instr`) under intrinsic and dynamic transmitter assumptions are in the leakage signature files:

```
$ synthlc/i_<instr>_out/xIftDynamic/leakage_signature.txt
```

Applications of Leakage Signatures for Security Analysis

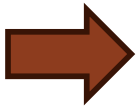
- Follow-on work: *Helium: Quantifying Microarchitectural Side-Channel Leakage with Probabilistic Guarantees*
  - **Will be presented at the Security Session (3C) on Monday at 2:50pm!**

- This work extends leakage signatures in order to quantify hardware side-channel leakage of program secrets

Side-channel models derived from SynthLC's leakage signatures

```
// Bit-Serial optimization: Division instruction exhibits 65 μobs depending
on the difference in the two operands' number of leading zeros.
μobs bit_serial(DIV i):
  if (i.op1==0 V (i.op1<i.op2)): return μobs_1
  else:
    for d in range(64):
      if ((i.op1>>d)≥i.op2)∧((i.op1>>(d+1))<i.op2):
        return μobs_{d+2}
```


Symbolic and simulation-based program analysis of security-critical programs



Novel probabilistic privacy guarantees