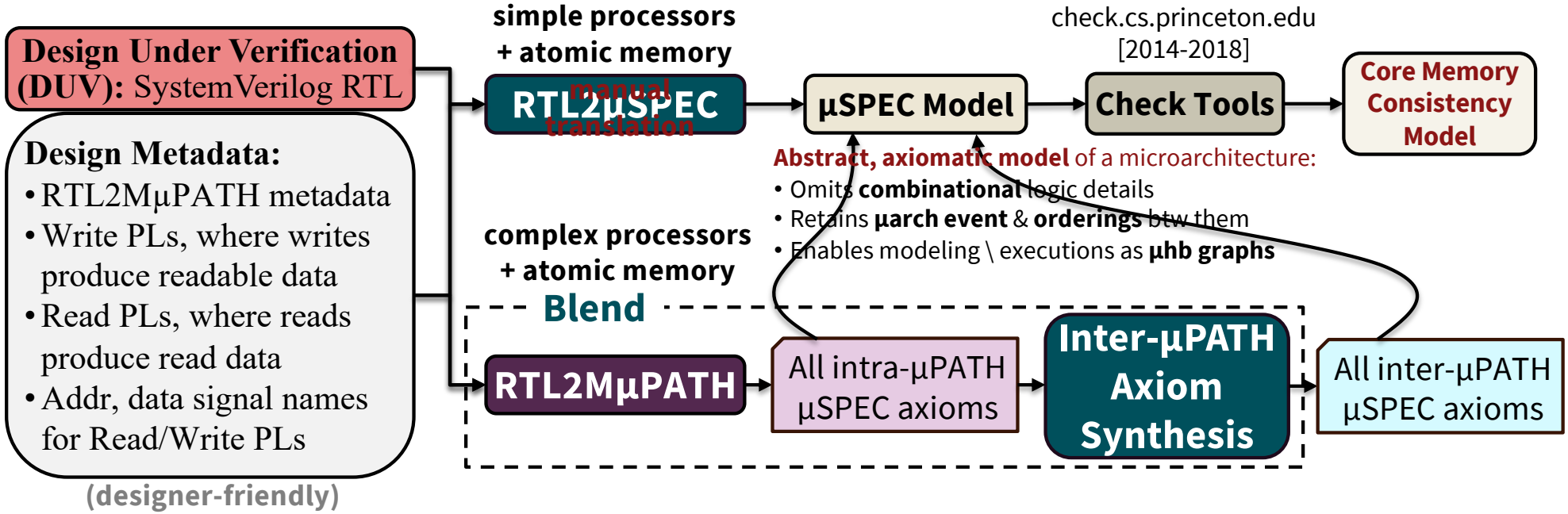
Memory Consistency Model Verification: RTL2μSPEC (Overview)

Caroline

Stanford | ENGINEERING

 RTL2μSPEC • [Hsiao+, MICRO'21] **Blend** • [Hsiao, Seneviratne+, In Prep.

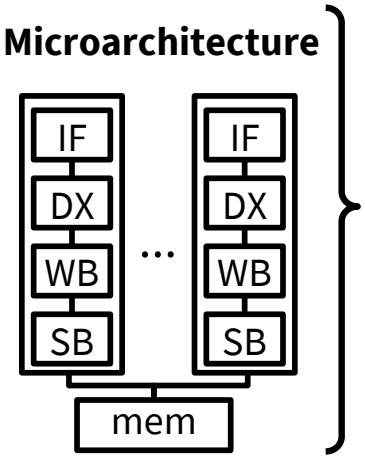
Overview: Synthesizing formally verified memory consistency models from SystemVerilog RTL - Cores w/ Atomic Memory



→ *Early prototype evaluated on **Apple hardware** [Summer'21].*

https://github.com/yaohsiaopid/rtl2uspec

Prior Work: The Check Tools verify a microarchitecture's memory consistency model implementation w.r.t. a μSPEC model representation of it



μSPEC Model

```
// μPATH axioms
Axiom "St_exe_path":
forall microops i, IsAnyWrite i ⇒ AddEdges [((i, IF), (i, DX));
((i, DX), (i, WB));
((i, WB), (i, SB)); ((i, SB), (i, mem));].
…
// inter-μPATH axioms
Axiom "DX_in_order":
forall microops i, j, ProgramOrder i j ⇒
AddEdge ((i, DX), (j, DX)).
Axiom BeforeAllWrites:
forall microops i, j, IsAnyRead i /\ IsAnyWrite j /\ SamePA i j /\
DataFromInitStateAtPA i ⇒ AddEdge ((i, DX), (j, (0, mem))).
…
```

Encodes the space of **all μarch executions** of all programs as **μHB graphs**.

[Lustig+, ASPLOS'16]

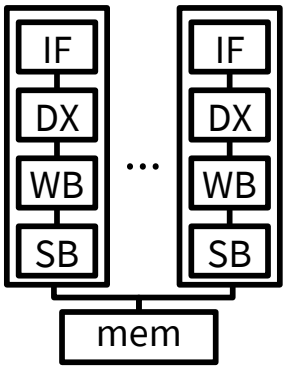
Prior Work: The Check Tools verify a microarchitecture's memory consistency model implementation w.r.t. a μSPEC model representation of it

Litmus Test Program

Initially: [x1]=0, [x2]=0, x3=1, x5=1	
Thread 0	Thread 1
(i0) sw x3, 0(x1)	(i3) sw x5, 0(x2)
(i1) fence rw, rw	(i4) fence rw, rw
(i2) lw x4, 0(x2)	(i5) lw x6, 0(x1)
Forbidden by IBM370: x4=0, x6=0	

Memory consistency models define legal return values for shared memory loads.

Microarchitecture



Encodes the space of **all μarch executions** of all programs as **μHB graphs**.

μSPEC Model

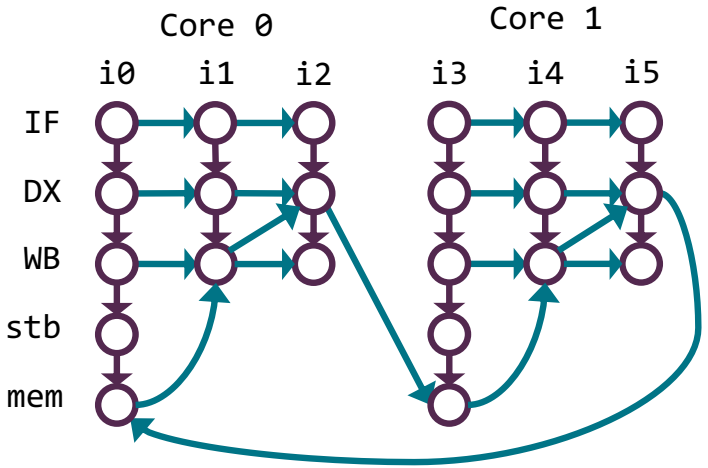
Check Tools

Use an SMT solver to prove a **forbidden litmus test*** induces only **cyclic μHB graphs**.

~50% intra-μPATH μSPEC axioms

~50% inter-μPATH μSPEC axioms

Microarchitectural Happens-Before (μHB) Graph [Lustig+, MICRO'14]



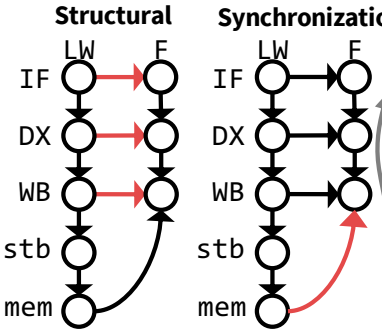
* or **all forbidden litmus tests** [Manerkar+, MICRO'18]

RTL2μSPEC • [Hsiao+, MICRO'21] **Blend** • [Hsiao, Seneviratne+, In Prep.]

Our Procedure: Synthesize intra-μPATH axioms for relevant committed instructions → synthesize struct/sync and dataflow inter-μPATH axioms

Structural/Temporal/Synchronization Happens-Before

Arise between instructions when they induce HW events involving accesses to the same/different HW resource and require serialization.

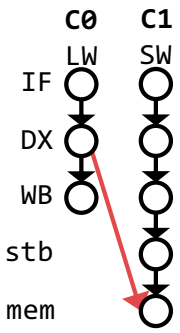


Does (<i0>, <PL0>) → (<i1>, <PL1>), given <preconditions>?

Structural example: Does (i0, DX) → (i1, DX), given (i0, IF) → (i1, IF)?

Dataflow Happens-Before

Arise when a write instruction visits a Write PL and produces data for some address that is later read by a read instruction that visits a Read PL.



reads-from: Does (<w0>, <WPL0>) → (<r0>, <RPL0>), given <preconditions>?

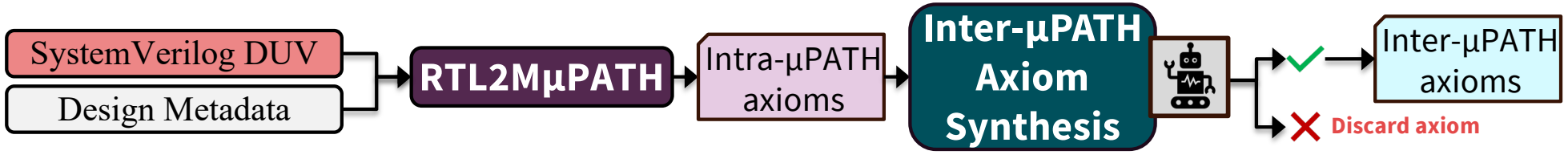
coh-order: Does (<w0>, <WPL0>) → (<w1>, <WPL1>), given <preconditions>?

reads-before: Does (<r0>, <RPL0>) → (<w0>, <WPL0>), given <preconditions>?

property templates

Requires some μhb nodes to have associated value, addr signals

Ask a model checker…



Memory Consistency Model Verification: RTL2μSPEC (Hands-On)

Yao

Stanford | ENGINEERING

Roadmap: Apply RTL2μSPEC on CVA6

- **Step 1: [Inside] Structural Dependencies**
- Step 2: [Input] Annotate Write PLs and Read PLs
- Step 3: [Inside] Data Dependencies

In this section, we will work under rtl2mupath/fv folder:

```
$ cd ~/fava_isca2026/rtl2mupath/fv;
```

[Hands-On] Structural Dependencies

Step 1: A pair of instructions are involved in a structural dependency if they update the same state element(s), since these updates must be serialized.

Example: For all store/load pairs in program order, do they visit the `mem_req` PL (i.e., accessing memory hierarchy) in program order?

```
$ cd InterHBI/xStructural && ./run.sh && vim -o gen/SW_LW/out/HB_6.sv \
  ../i0_pl.sv ../i1_pl.sv
```

```
234 wire i0_mem_req_s1 =
235     (ex_stage_i.lsu_i.i_ord_sram.pc_i == pc0) &&
236     (ex_stage_i.lsu_i.i_ord_sram.req_i == 1'b1) &&
237     1'b1;
```

fv/InterHBI/i0_pl.sv

```
236 wire i1_mem_req_s1 =
237     (ex_stage_i.lsu_i.i_ord_sram.pc_i == pc1) &&
238     (ex_stage_i.lsu_i.i_ord_sram.req_i == 1'b1) &&
239     1'b1;
```

fv/InterHBI/i1_pl.sv

```
113 I0_PO_I1: assume property (@(posedge clk_i)
!i0_inst_begin_hpn -> !i1_instn_begin);
```

fv/InterHBI/xStructural/gen/SW_LW/out/HB_6.sv

```
126 wire e0 = i0_mem_req_s1 ;
127 wire e1 = i1_mem_req_s1 ;
...
144 reg e0_hb_e1;
145 always @(posedge clk_i) begin
146     if (!rst_ni)
147         e0_hb_e1 <= 1'b0;
148     else begin
149         if (e0 && !e0_hpn)
150           e0_hb_e1 <= !(e1_hpn || e1);
151     end
152 end
...
168 HB_6: assert property (@(posedge clk_i)
169   (i1_cmt_hpn & i0_cmt_hpn & e0_hpn) |-> e0_hb_e1);
```

fv/InterHBI/xStructural/gen/SW_LW/out/HB_6.sv

[Hands-On] Structural Dependencies

Step 1: A pair of instructions are involved in a structural dependency if they update the same state element(s), since these updates must be serialized.

Example: For all store/load pairs in program order, do they visit the `mem_req` PL (i.e., accessing memory hierarchy) in program order?

```
$ cd InterHBI/xStructural && ./run.sh && vim -o gen/SW_LW/out/HB_6.sv \
  ../i0_pl.sv ../i1_pl.sv
```

```
234 wire i0_mem_req_s1 =
235     (ex_stage_i.lsu_i.i_ord_sram.pc_i == pc0) &&
236     (ex_stage_i.lsu_i.i_ord_sram.req_i == 1'b1) &&
237     1'b1;
```

fv/InterHBI/i0_pl.sv

```
236 wire i1_mem_req_s1 =
237     (ex_stage_i.lsu_i.i_ord_sram.pc_i == pc1) &&
238     (ex_stage_i.lsu_i.i_ord_sram.req_i == 1'b1) &&
239     1'b1;
```

fv/InterHBI/i1_pl.sv

```
113 I0_PO_I1: assume property (@(posedge clk_i)
!i0_inst_begin_hpn -> !i1_instn_begin);
```

Assume i0 and i1 are in program order

fv/InterHBI/xStructural/gen/SW_LW/out/HB_6.sv

```
126 wire e0 = i0_mem_req_s1 ;
127 wire e1 = i1_mem_req_s1 ;
...
144 reg e0_hb_e1;
145 always @(posedge clk_i) begin
146     if (!rst_ni)
147         e0_hb_e1 <= 1'b0;
148     else begin
149         if (e0 && !e0_hpn)
150
151
152 end

HB_6: assert property (@(posedge clk_i)
  (i1_cmt_hpn & i0_cmt_hpn & e0_hpn) |-> e0_hb_e1);
```

When e0 happens (1st time), e0_hb_e1 records whether e1 has already occurred

Assert than when both instructions commit and e0 has occurred, then e0 must have happened before e1

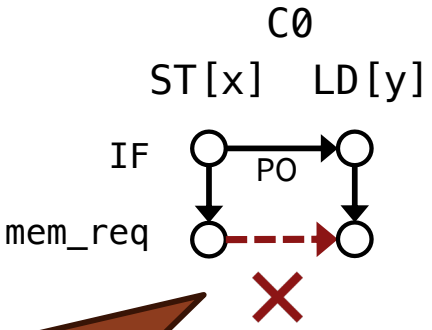
fv/InterHBI/xStructural/gen/SW_LW/out/HB_6.sv

[Hands-On] Structural Dependencies

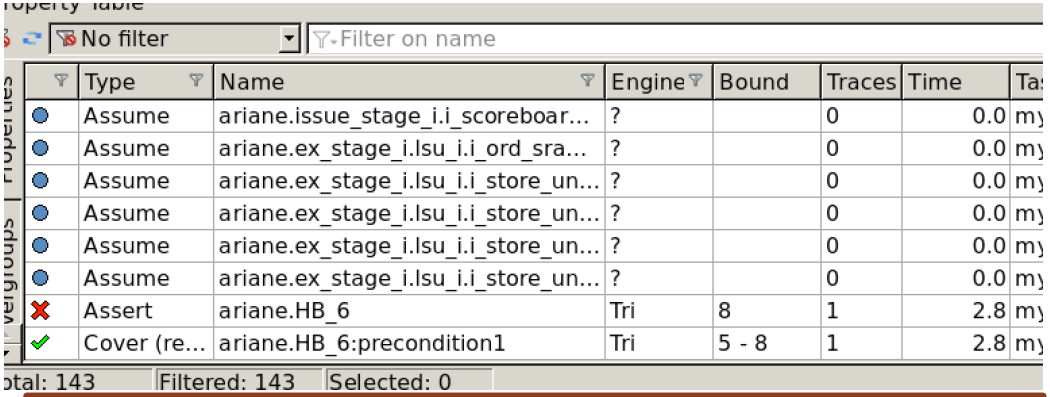
Step 1: A pair of instructions are involved in a structural dependency if they update the same state element(s), since these updates must be serialized.

Example: For all store/load pairs in program order, do they visit the `mem_req` PL (i.e., accessing memory hierarchy) in program order?

```
$ cd ../../
$ ./RUN_JG.sh -j xEval -s InterHBI/xStructural/gen/SW_LW/out/HB_6.sv
```



Because loads can access memory with different-address stores in the store-buffer!




```
168 HB_6: assert property (@(posedge clk_i)
169   (i1_cmt_hpn & i0_cmt_hpn & e0_hpn) |-> e0_hb_e1);
```

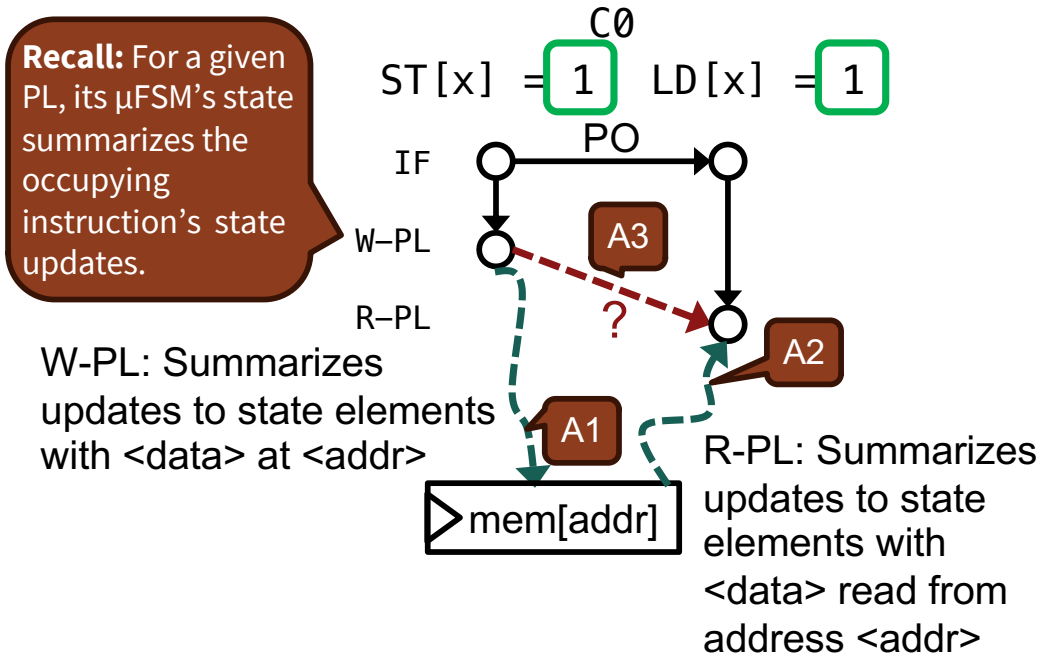
fv/InterHBI/xStructural/gen/SW_LW/out/HB_6.sv

Roadmap: Apply RTL2μSPEC on CVA6

- Step 1: [Inside] Structural Dependencies
- **Step 2: [Input] Annotate Write PLs and Read PLs**
- Step 3: [Inside] Data Dependencies

[Hands-On] Annotate Write PLs and Read PLs

Step 2: Data dependencies arise when a write instruction visits a Write PL (W-PL) and produces data for some address that is later read by a read instruction that visits a Read PL (R-PL).




```
A1: For all writes w with addr <addr> and data value <v>, does w always introduce <v> to <W–PL.data> when <W–PL.addr> = <addr>?
```

Decomposed property W-PL template

```
A2: For all reads r with addr <addr> that return data value <v>, does r's <v> always equal <R–PL.data> when r's <addr> equals <R–PL.addr>?
```

Decomposed property R-PL template

```
A3: For write w in program order before read r, where <W–PL.addr> = <R–PL.addr>, does w always visit <W–PL> before r visits <R–PL>?
```

Decomposed property W-PL HB R-PL template

[Hands-On] Annotate Write PLs and Read PLs

Step 2: Data dependencies arise when a write instruction visits a Write PL (W-PL) and produces data for some address that is later read by a read instruction that visits a Read PL (R-PL).
Example: CVA6 W-PLs and R-PLs and related correctness property

```
$ cd InterHBI/Assoc && vim config.py
```

Input

```
  1 issue_assoc_addr = ("issue_s16",
"issue_stage_i.i_issue_read_operands.fu_data_o.operand_a +
issue_stage_i.i_issue_read_operands.fu_data_o.imm")
  2 w_pls = [("mem_req_s1",
"ex_stage_i.lsu_i.dcache_req_ports_o_st.virtual_address",
"ex_stage_i.lsu_i.dcache_req_ports_o_st.data_wdata")]
  3 r_pls = [("mem_req_s1",
"ex_stage_i.lsu_i.dcache_req_ports_o_ld.virtual_address",
"ex_stage_i.lsu_i.dcache_req_ports_i_ld.data_rdata")]
```

config.py

Expression capturing the effective address of a memory instruction when the instruction is issued

Each W-PL is a tuple of:
(W-PL name, address signal, write data signal)

Each R-PL is a tuple of:
(R-PL name, address signal, read data signal)

Roadmap: Apply RTL2μSPEC on CVA6

- Step 1: [Inside] Structural Dependencies
- Step 2: [Input] Annotate Write PLs and Read PLs
- **Step 3: [Inside] Data Dependencies**

[Hands-On] Data Dependencies Proof A1

Step 3a: Decomposed property A1 on W-PL

```
A1: For all writes w with addr <addr> and data value <v>, does w always introduce <v> to <W-PL.data> when <W-PL.addr> = <addr>?
```

Decomposed property W-PL template

Example: Instantiated on CVA6

```
$ vim out/data_sw.sv
```

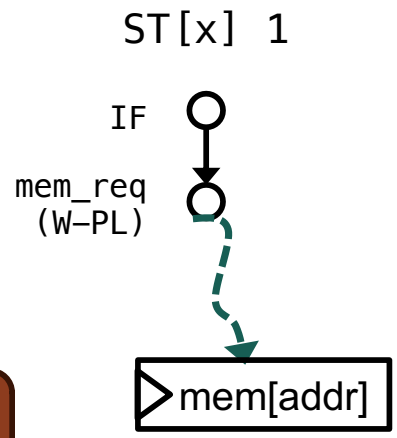
Intermediate Output

```
322 wire [64-1:0] data0;
323 data0_const: assume property (@(posedge clk_i) CONST(data0));
324 // register-read stage
325 i0_data: assume property (@(posedge clk_i) issue_s16 |->
326  (issue_stage_i.i_issue_read_operands.fu_data_o.operand_b == data0));
...
332 DATA: assert property (@(posedge clk_i) mem_req_s1 |->
     (ex_stage_i.lsu_i.dcache_req_ports_o_st.data_wdata == data0));
```

out/data_sw.sv

When store `i0` is issued, assume its data operand is `data0`

Assert that when `i0` visits W-PL `mem_req_s1`, `W-PL.data` holds the same value as specified by instruction operand



[Hands-On] Data Dependencies Proof A1

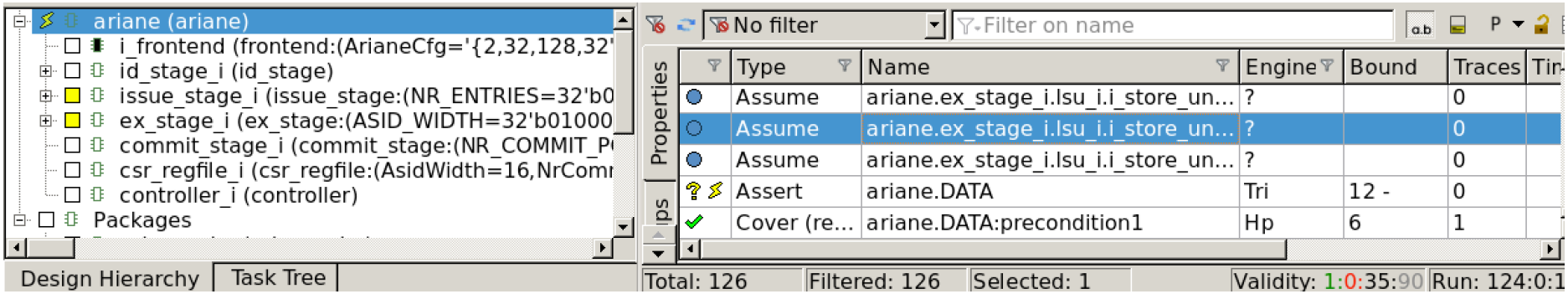
Step 3a: Decomposed property A1 on W-PL

```
A1: For all writes w with addr <addr> and data value <v>, does w always introduce <v> to <W-PL.data> when <W-PL.addr> = <addr>?
```

Decomposed property W-PL template

Example: Instantiated on CVA6

```
$ cd ../../ && ./RUN_JG.sh -j xEval -s InterHBI/Assoc/out/data_sw.sv
```



[Hands-On] Data Dependencies Proof A3

Step 3b: Decomposed property A3

```
A3: For write w in program order before read r,
where <W-PL.addr> = <R-PL.addr>, does w always
visit <W-PL> before r visits <R-PL>?
```

Decomposed property W-PL HB R-PL template

Example: Instantiated on CVA6 based on the annotation

```
$ cd InterHBI/xDataDep && ./run.sh && vim same_addr_dep.sv
```

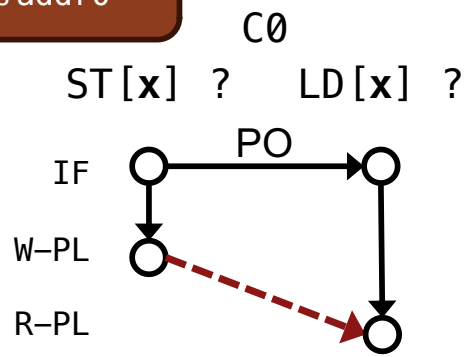
Intermediate Output

```
125 wire [64-1:0] addr0;
126 addr0_const: assume property (@(posedge clk_i) CONST(addr0));
127 // register-read stage
128 i0_addr: assume property (@(posedge clk_i) i0_mem_req_s1 |->
129   ex_stage_i.lsu_i.dcache_req_ports_o_st.virtual_address == addr0);
130 i1_addr: assume property (@(posedge clk_i) i1_mem_req_s1 |->
131   ex_stage_i.lsu_i.dcache_req_ports_o_ld.virtual_address == addr0);
...
139 wire e0 = i0_mem_req_s1 ; // Write-PL
140 wire e1 = i1_mem_req_s1 ; // Read-PL
```

same_addr_dep.sv

When i0 visits W-PL, W-PL.addr matches symbolic address addr0

When i1 visits R-PL, R-PL.addr matches symbolic address addr0



[Hands-On] Data Dependencies Proof A3

Step 3b: Decomposed property A3

```
A3: For write w in program order before read r,
where <W-PL.addr> = <R-PL.addr>, does w always
visit <W-PL> before r visits <R-PL>?
```

Decomposed property W-PL HB R-PL template

Example: Instantiated on CVA6 based on the annotation

```
$ cd InterHBI/xDataDep && ./run.sh && vim same_addr_dep.sv
```

Intermediate Output

```
125 wire [64-1:0] addr0;
126 addr0_const: assume property (@(posedge clk_i) CONST(addr0));
127 // register-read stage
128 i0_addr: assume property (@(posedge clk_i) i0_mem_req_s1 |->
129   ex_stage_i.lsu_i.dcache_req_ports_o_st.virtual_address == addr0);
130 i1_addr: assume property (@(posedge clk_i) i1_mem_req_s1 |->
131   ex_stage_i.lsu_i.dcache_req_ports_o_ld.virtual_address == addr0);
...
139 wire e0 = i0_mem_req_s1 ; // Write-PL
140 wire e1 = i1_mem_req_s1 ; // Read-PL
```

same_addr_dep.sv

e0: i0 visits mem_req_s1 (W-PL)
e1: i1 visits mem_req_s1 (R-PL)

```
157 reg e0_hb_e1;
158 always @(posedge clk_i) begin
159     if (!rst_ni)
160         e0_hb_e1 <= 1'b0;
161     else begin
162         if (e0 && !e0_hpn)
163           e0_hb_e1 <= !(e1_hpn || e1);  //
1 iff no re-ordering
164     end
165 end
...
181 HB_0: assert property (@(posedge clk_i)
182   (i1_cmt_hpn & i0_cmt_hpn & e0_hpn) |->
    e0_hb_e1);
```

e0_hb_e1: record if in the past e0 happens-before e1

if i1 and i0 both commit and e0 occurred, then e0_hb_e1 should be 1

[Hands-On] Data Dependencies Proof A3

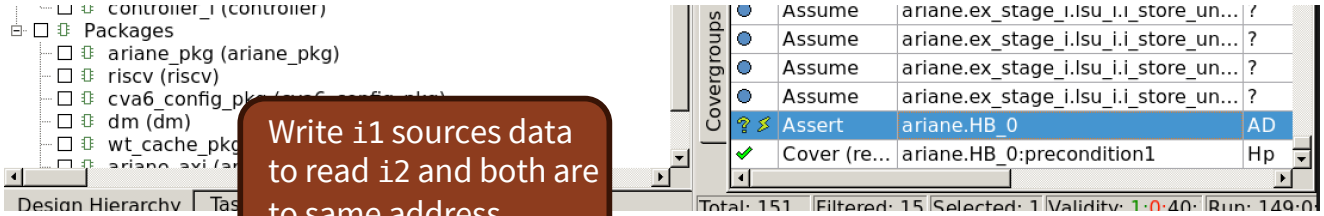
Step 3b: Decomposed property A3

```
A3: For write w in program order before read r,
where <W-PL.addr> = <R-PL.addr>, does w always
visit <W-PL> before r visits <R-PL>?
```

Decomposed property W-PL HB R-PL template

Example: Evaluated against CVA6

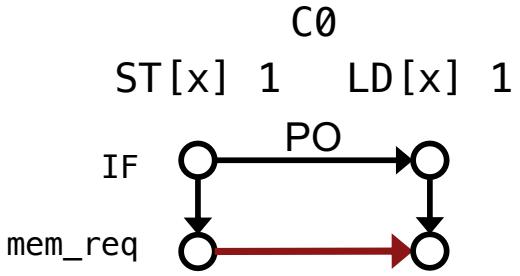
```
$ cd ../../
$ ./RUN_JG.sh -j xEval -s InterHBI/xDataDep/same_addr_dep.sv
```



Output

```
195 Axiom "data_1_atomic_mem":
196 forall microops "i1", forall microops "i2",
197 (IsAnyWrite i1 /\ IsAnyRead i2 /\ SameData i1 i2 /\
SamePhysicalAddress i1 i2) =>
198 AddEdge ((i1, (0, mem_req)), (i2, (0, mem_req)), "rf", "green").
```

fv/artifact/cva6.uarch



Then there is a happens-before edge between i1 at W-PL and i2 at R-PL

Other outputs

Run full RTL2µSPEC to generate µSPEC model:

- `./run_interhbi.sh`

µSPEC model output:

- `fv/artifact/cva6.uarch`

Evaluate µSPEC model for adherence to the IBM370 MCM:

- `fv/eval_uspec.sh`