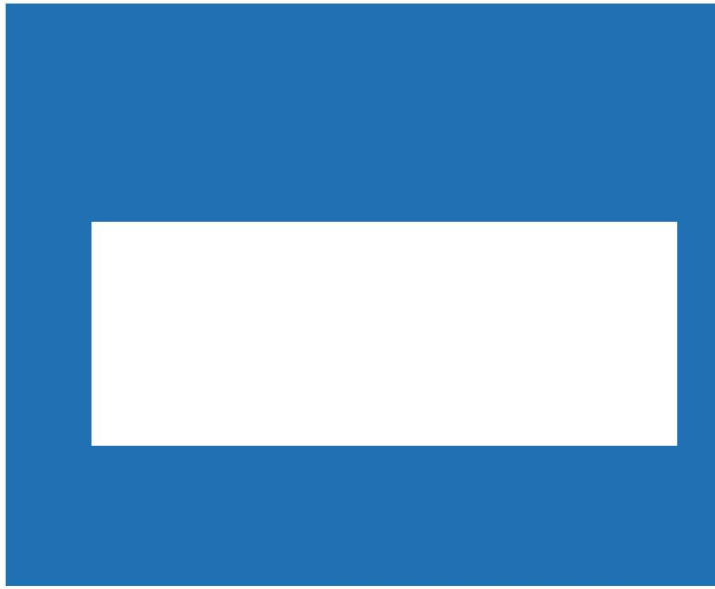
SAND2020-10997C

Ensuring completeness of formal verification with Gap Free Verification

Ratish Punnoose, Sandia National Laboratories

October 2020



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

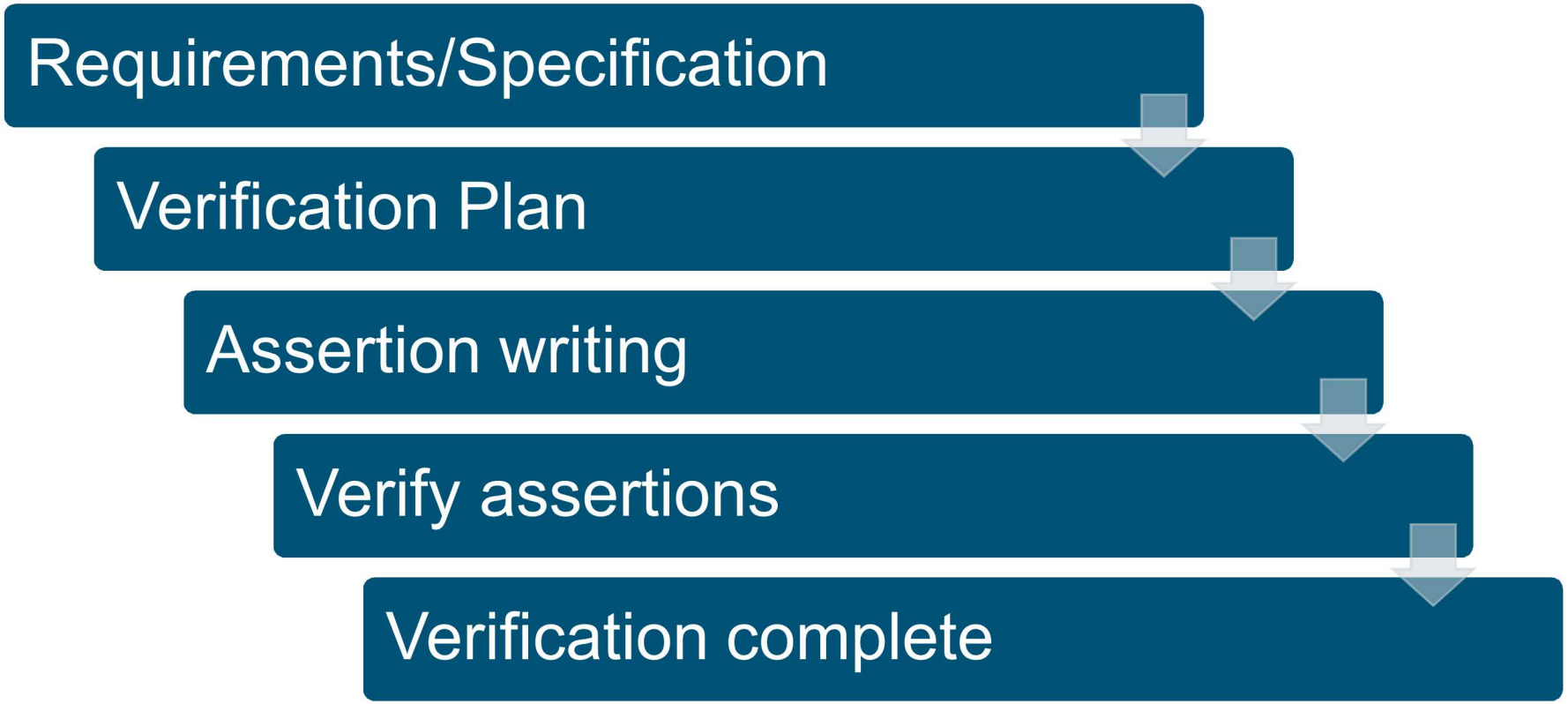
About Sandia National Labs

- Sandia is the engineering arm of the U.S. nuclear weapons enterprise.
- The nation's nuclear weapons must *always* work when commanded and authorized, and must *never* detonate otherwise.
- Digital systems at Sandia include embedded systems with custom ASICs, processors, and cryptographic capability
- In-house ASIC Fabrication capability

Problem

- How much formal verification is enough?
  - Are we done yet?

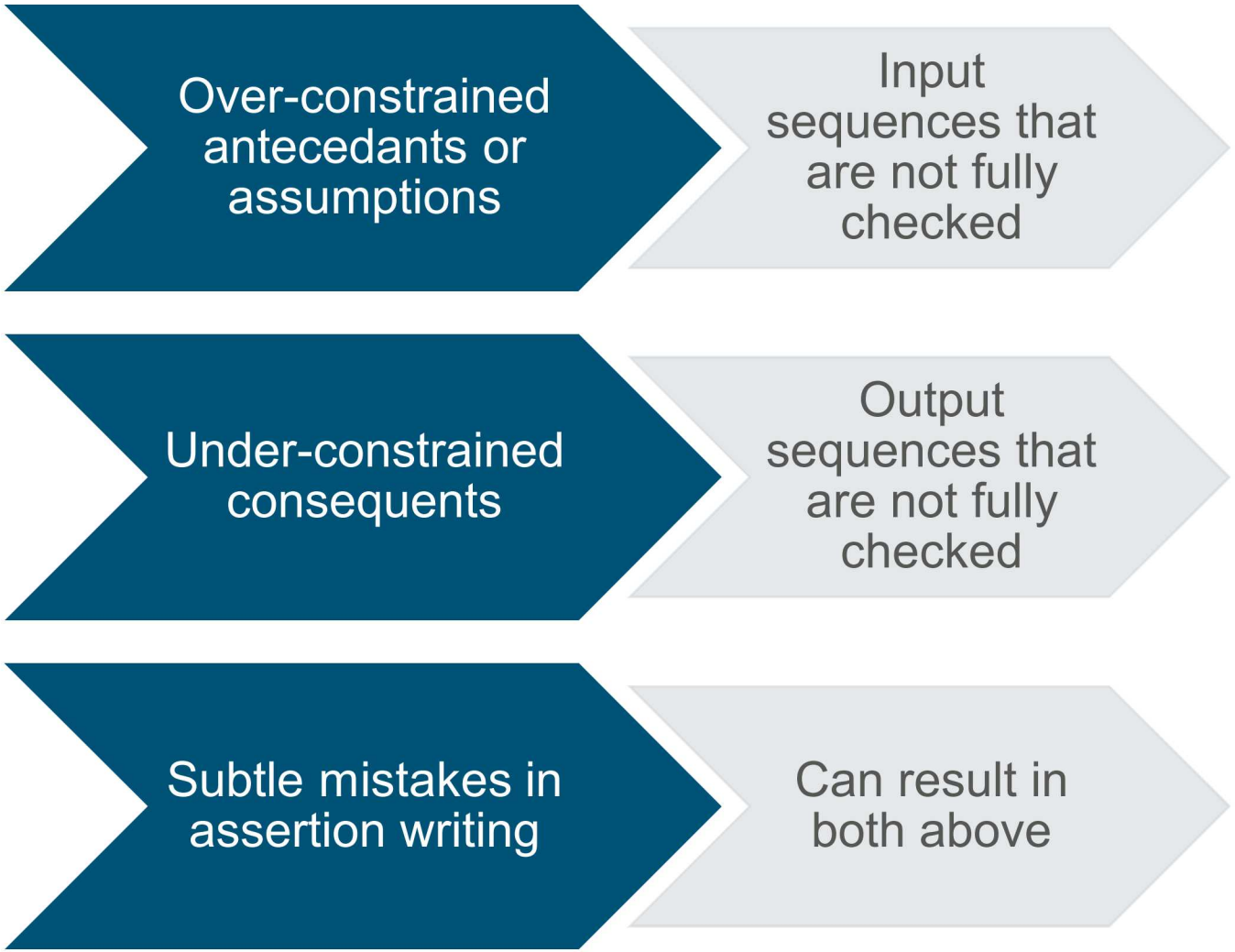
A typical approach



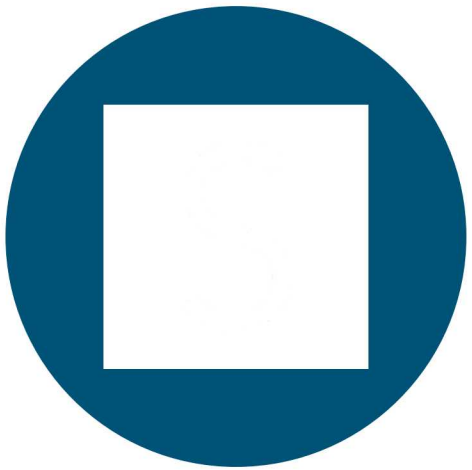
Process Pitfalls

- Requirements are usually written to describe function (matches simulation, UVM, testing)
- “Shall not” requirements are sometimes missed (strengths of formal verification)

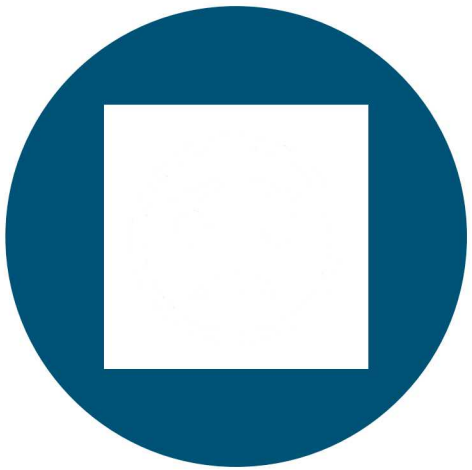
Verification pitfalls



Outcome of verification deficiencies



RE-DESIGN AND RE-VERIFICATION COST



LACK OF CONFIDENCE IN FORMAL VERIFICATION

A solution: GapFree Verification (GFV)

- Usage experience
  - Systems with well-defined requirements/specifications
    - Arm Peripheral Bus (APB) controller
    - APB General Purpose Input Output (GPIO) slave
  - Systems without full requirements
    - Processor Arithmetic Accelerator
    - Interrupt controller

A solution: GapFree Verification

What is it

How does it work conceptually

A mental model of what happens behind the scenes helps use the technique more effectively.

How do you use it in the tool

onespin osmosis

onespin meeting on solutions, innovation, & strategy

The OneSpin Gap Free Verification method

What is it, How does it work

What is it

onespin osmosis
onespin meeting on solutions, innovation, & strategy

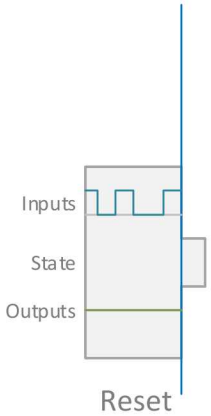
An **automated** and **formal** check on the properties to check for completeness

Completeness:

All possible input sequences are examined
All outputs are verified.
At all time

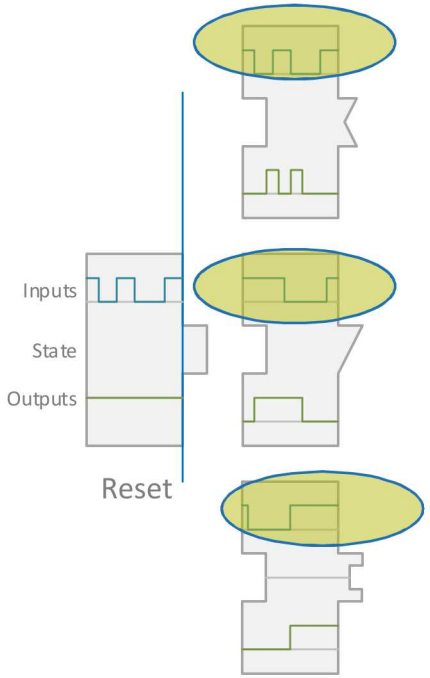
How does it work 1/5

- Start with a reset property that specifies the reset state of the system
  - Outputs
  - Externally visible state of the design
    - Eg: Upon release of reset, a design is expected to be in “idle”, when an operation starts it is “busy”, when operation completes it is “idle” again.



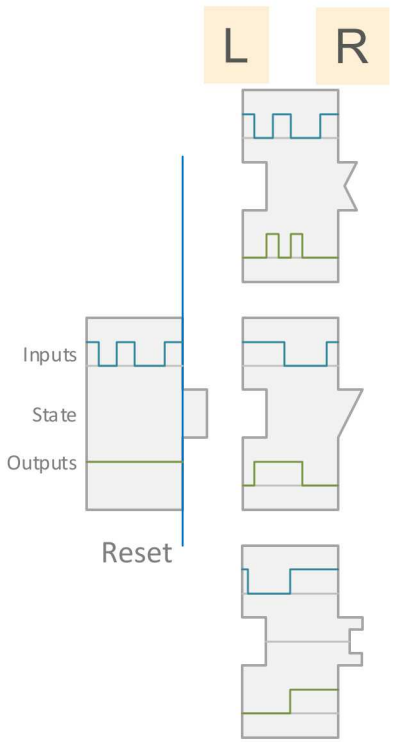
How does it work 2/5

- From the reset state, check that all possible input sequences are covered by the property set
  - i.e., check that antecedants don’t exclude any input sequence
  - Note: the properties have an antecedent and consequent (if-then OR assume-prove) separated by an implication operator



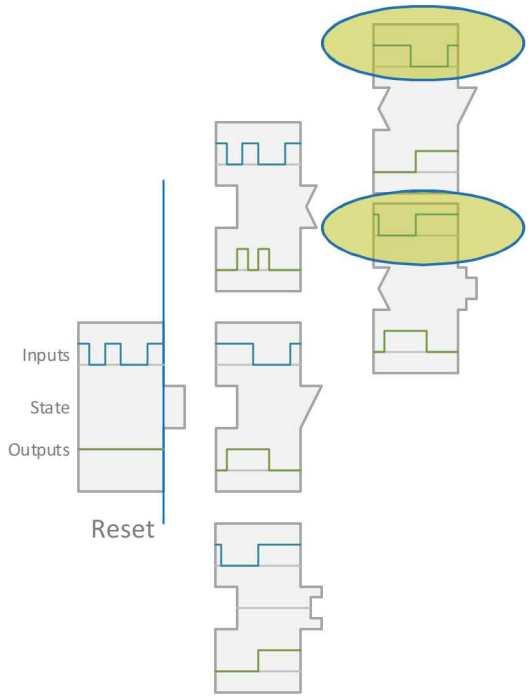
How does it work 3/5

- Each property has a designated start and end cycle
  - Left-hook: Point at which it takes over from the preceding property.
  - Right-hook: Point at which the succeeding property takes over
  - The designated start and end are for transfer of responsibility



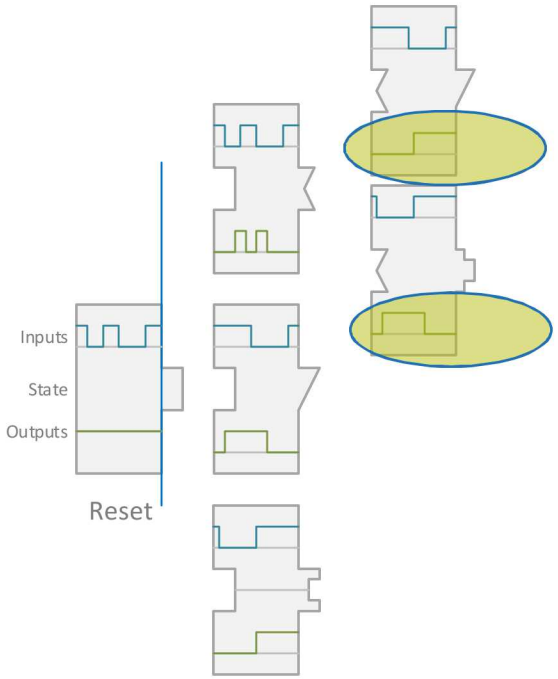
How does it work 4/5

- At the right-hook of each property, the design will be in a specific state
- From the right-hook, check all possible input sequences that follow are covered by the property set.



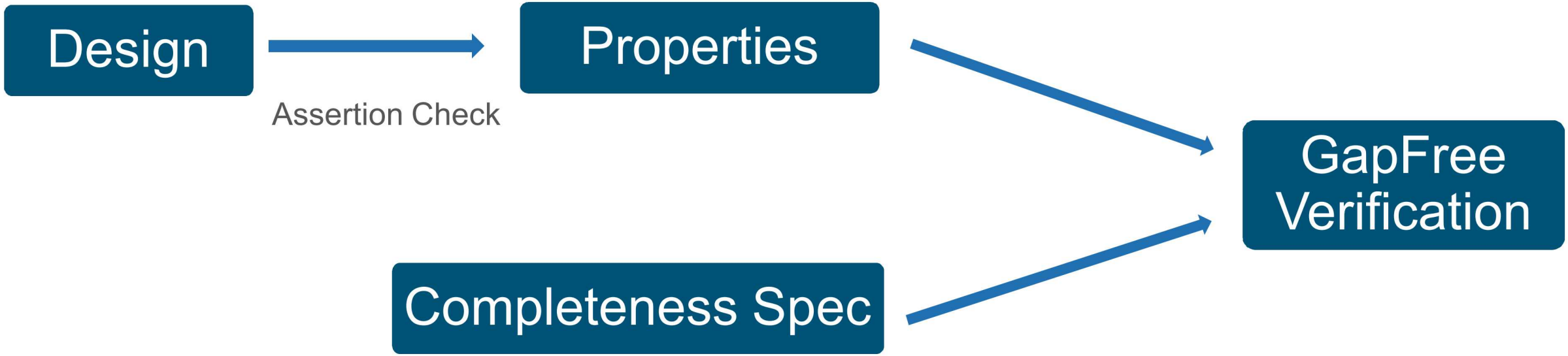
How does it work 5/5

- In the consequents for all properties, check that all outputs are specified uniquely.



Using GapFree verification in OneSpin

Using GapFree Verification



- Note: The properties have to hold on the design. The GapFree Check is a check on the properties.

Completeness file describes the following

Inputs

Outputs

Resets

Property graph

Exceptions

```
completeness cpu_mul;

disable iff: !Rst_n;

determination_assumptions:
        determined(Clk);
        determined(MultOp);
        determined(Rst_n);
        determined(Start);
        ...;

determination_requirements:
        determined(almostDone), end_offset=-1;
        phi_det: determined(prodHi);

reset_property: cpu_mul_sva.ops.reset_a;

property_graph:
   possible_ops := cpu_mul_sva.ops.mult_a,
                   cpu_mul_sva.ops.madd_a,
                   ...
                   cpu_mul_sva.ops.noop_a;
   cpu_mul_sva.ops.reset_a, possible_ops -> possible_ops;

end completeness;

local_completeness sva/cpu_mul_sva/ops/mult_a;
  allow_undetermined :
      during [t+1, sva/cpu_mul_sva/ops/t_almostDone]: allow_undetermined(phi_det);
end local_completeness;
```

Properties

- For GapFree Verification, Properties have to be written using TIDAL (Timing Diagram Assertion Library)
  - OneSpin library of System Verilog sequences, properties, macros
  - More constrained than general SVA

```
property ldaccum_p;
  logic [2*W-1:0] tmp_rsoutsq;
  t ##0 ready_for_op().triggered and
  t ##0 set_freeze(tmp_rsoutsq, { {W{1'b0}} , ALU.rs}) and
  t ##0 ( !(((ALU.OpCode == SQADDOP) || (ALU.OpCode == SQOP)) && (ALU.Start)) && ALU.ldaccum)
  implies
  t ##0 ALU.almostDone and
  t ##1 (ALU.rsoutsq == tmp_rsoutsq) and
  t ##1 right_hook;
endproperty // ldaccum_p
```

TIDAL conveniences

- Time points:
  - TIDAL library provides convenient constructs for describing time points and time-intervals

- Hooks:
  - TIDAL macro to demarcate left and right-hooks

GFV needs TIDAL properties written in a certain way

- Reset: use specific pattern for reset sequence

- Implication operator:
  - A single “implies” implication operator to divide the property into an antecedent and consequent
    - Operators “|->”, “|=>” not allowed

- Local variables captured using the set_freeze macro
  - SVA local variable assignments with complex flow rules not allowed

- Substitute liveness operators (eg. eventually, [*0:$]) with:
  - a TIDAL time interval + an assumption that the awaited event will happen within n cycles

GFV checks

Find ambiguous outputs

Reset

Status: hold Validity: up_to_date

Reset | Successor | Determination | Case_split | Contradiction | Minimality

	Status
1 assumption	H
2 determination	H
3 case_split	H

Determination

Status: hold Validity: up_to_date

Reset | Successor | Determination | Case_split | Contradiction | Minimality

	2	3	4	5	6	7
1 gpio_sva.ops.reset_a	H	H	H	H	H	H
2 gpio_sva.ops.addr_err_a	H	H	H	H	H	H
3 gpio_sva.ops.no_read_write_a	H	H	H	H	H	H
4 gpio_sva.ops.read_addr_0_a	H	H	H	H	H	H
5 gpio_sva.ops.read_addr_1_a	H	H	H	H	H	H
6 gpio_sva.ops.write_addr_0_a	H	H	H	H	H	H
7 gpio_sva.ops.write_addr_1_a	H	H	H	H	H	H

Extra Checks

Contradiction

Status: hold Validity: up_to_date

Reset | Successor | Determination | Case_split | Contradiction | Minimality

	Status
1 gpio_sva.ops.reset_a	H
2 gpio_sva.ops.addr_err_a	H
3 gpio_sva.ops.no_read_write_a	H
4 gpio_sva.ops.read_addr_0_a	H
5 gpio_sva.ops.read_addr_1_a	H
6 gpio_sva.ops.write_addr_0_a	H
7 gpio_sva.ops.write_addr_1_a	H

Find overconstrained or missing properties

Case-Split

Status: hold Validity: up_to_date

Reset | Successor | Determination | Case_split | Contradiction | Minimality

	Status
1 gpio_sva.ops.reset_a	H
2 gpio_sva.ops.addr_err_a	H
3 gpio_sva.ops.no_read_write_a	H
4 gpio_sva.ops.read_addr_0_a	H
5 gpio_sva.ops.read_addr_1_a	H
6 gpio_sva.ops.write_addr_0_a	H
7 gpio_sva.ops.write_addr_1_a	H

Successor

Status: hold Validity: up_to_date

Reset | Successor | Determination | Case_split | Contradiction | Minimality

	2	3	4	5	6	7
1 gpio_sva.ops.reset_a	H	H	H	H	H	H
2 gpio_sva.ops.addr_err_a	H	H	H	H	H	H
3 gpio_sva.ops.no_read_write_a	H	H	H	H	H	H
4 gpio_sva.ops.read_addr_0_a	H	H	H	H	H	H
5 gpio_sva.ops.read_addr_1_a	H	H	H	H	H	H
6 gpio_sva.ops.write_addr_0_a	H	H	H	H	H	H
7 gpio_sva.ops.write_addr_1_a	H	H	H	H	H	H

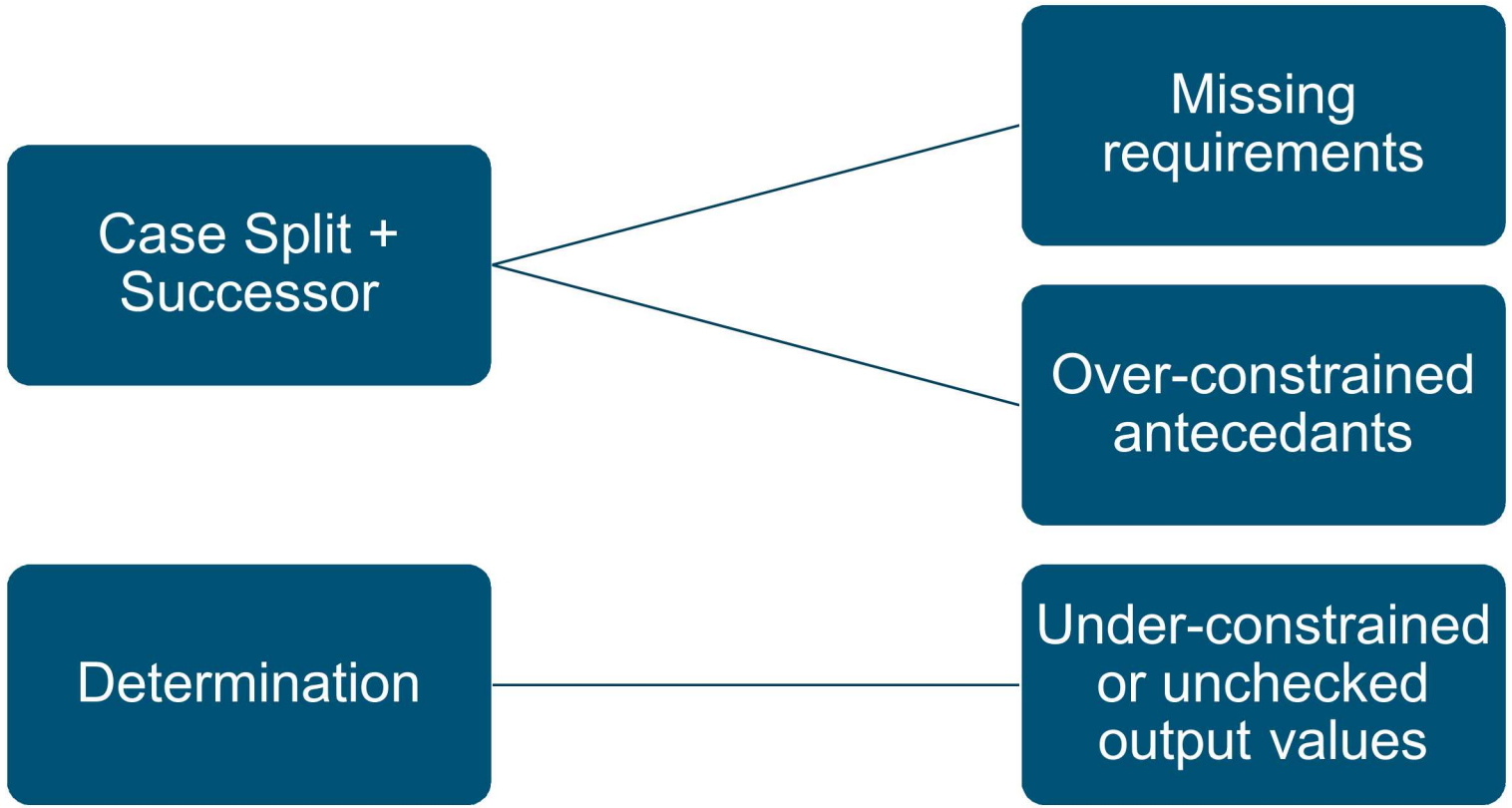
Minimality

Status: hold Validity: up_to_date

Reset | Successor | Determination | Case_split | Contradiction | Minimality

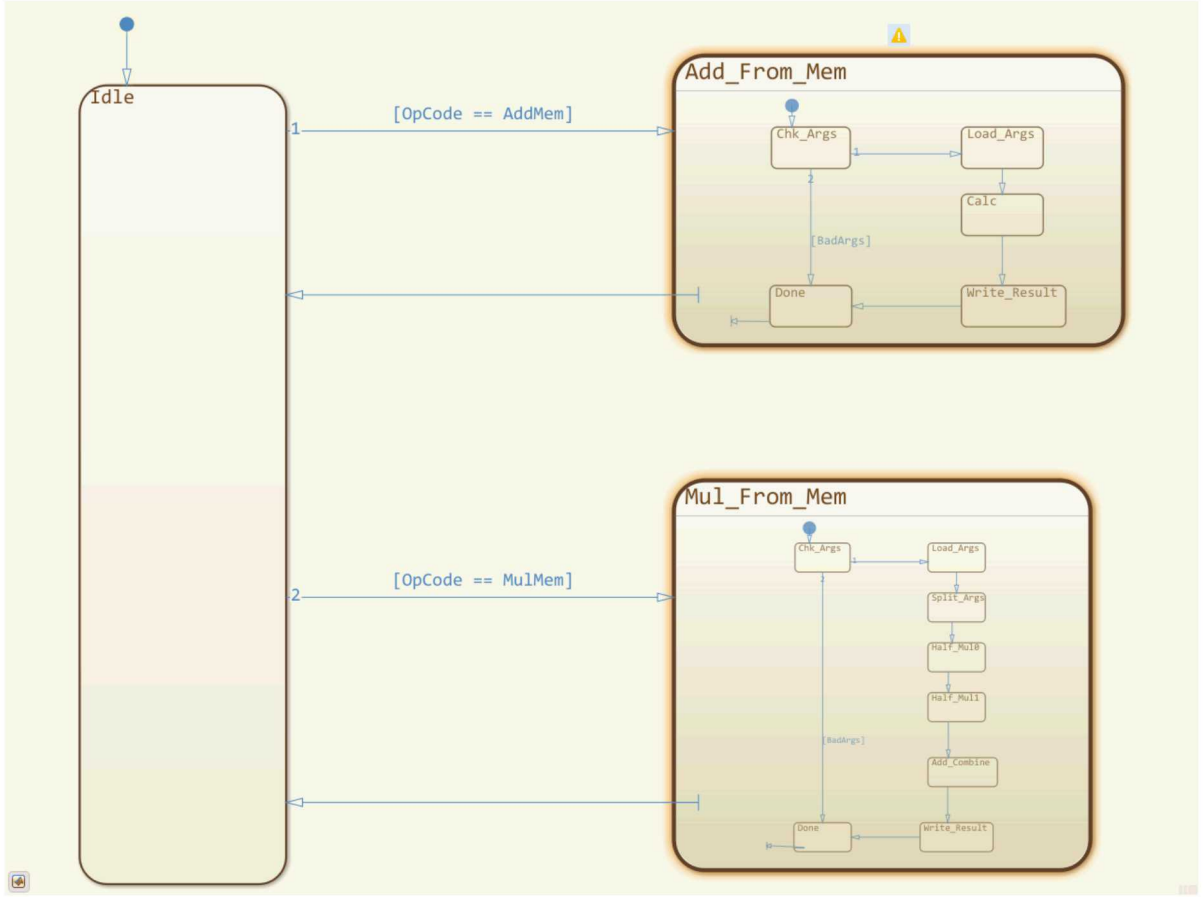
	2	3	4	5	6	7
1 gpio_sva.ops.reset_a	H	H	H	H	H	H
2 gpio_sva.ops.addr_err_a	H	H	H	H	H	H
3 gpio_sva.ops.no_read_write_a	H	H	H	H	H	H
4 gpio_sva.ops.read_addr_0_a	H	H	H	H	H	H
5 gpio_sva.ops.read_addr_1_a	H	H	H	H	H	H
6 gpio_sva.ops.write_addr_0_a	H	H	H	H	H	H
7 gpio_sva.ops.write_addr_1_a	H	H	H	H	H	H

How does GapFree Verification solve the pitfalls identified earlier



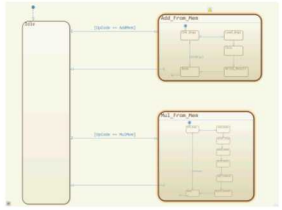
Other benefits of GapFree

- A complete property set is an “abstract” model of the system



- All design behavior is exposed (to the level of the properties)
  - Hidden behavior detected in antecedent check

- Can back-out undocumented specification from the “abstract” model



Experiences with GapFree

- Missing assertion check of outputs in GPI slave module, AMBA master
- Missing assertion check of certain input cases.
  - SVA assertions ignored two back-to-back out-of-band operations.
- Seeming bug in ALU math operation (output modified by toggling inputs in the middle of computation)
  - Designers intended it would not be used this way
  - Backed out specification from the property set.

Summary

- The OneSpin GapFree Verification method provides an automated and formal check for completeness

- Ideal for high-consequence systems where the complete verification is desired (as opposed to “targeted formal”)
  - Identifies missing assertions and checks on outputs.
    - What if missing assertions would have failed ? Undiscovered bug.
  - Can identify over-constrained antecedants and unchecked outputs

- Unique among Formal Property Verification (FPV) tool capabilities