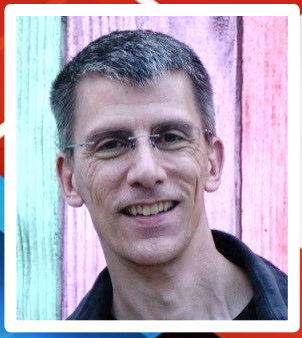
DOULOS KnowHow WEBINARS

Delivering KnowHow www.doulos.com

What Can Formal Do For Me?



Presenter: Doug Smith
Engineer / Instructor

Webinar partner:

What Can Formal Do For Me?

- What is formal?
- Where can formal be used?
- Applications for formal
- Wrap-up

DOULOS

What is Formal?

> "Formal verification uses mathematical formal methods to prove or disprove the correctness of a system's design with respect to formal specifications expressed as properties…."
>
> (*Using Formal Methods to Verify Complex Designs*, IBM Haifa Research Lab)

Formal …

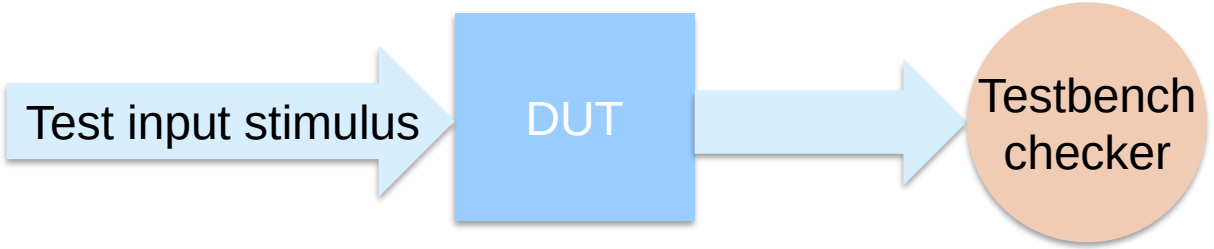
- Is mathematical and algorithmic
- Proves the correctness of a design
- Guarantees the implementation meets the requirements
- Requires no testbench or stimulus

Simulation vs Formal

Simulation

Tests the design

Testbench generates all stimulus and performs checking

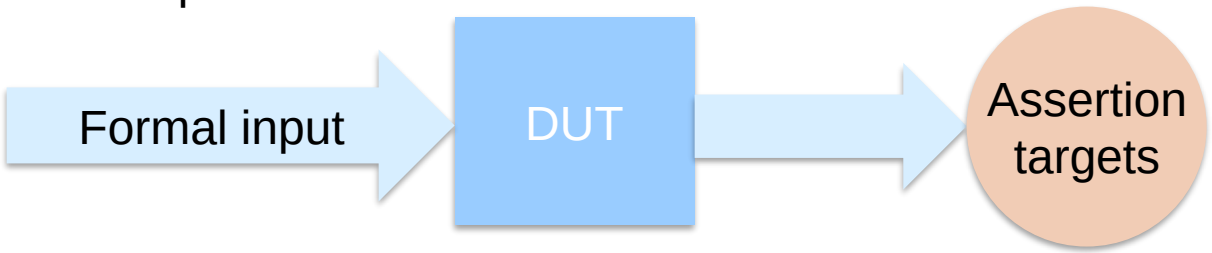


Formal

Proves the design meets the requirements

Requirements become formal target

Formal generates all input

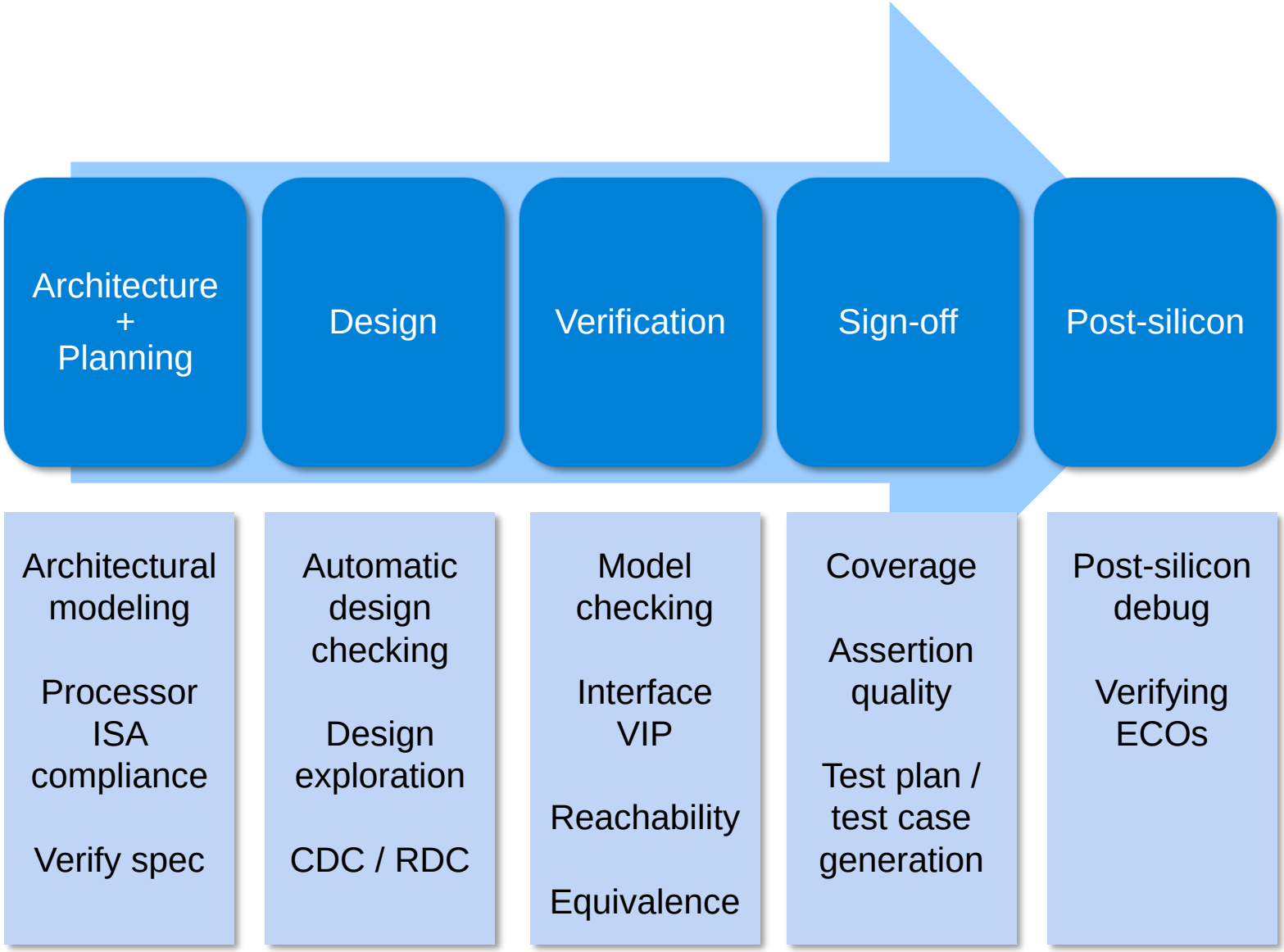


What Can Formal Do For Me?

- What is formal?
- Where can formal be used?
- Applications for formal
- Wrap-up

DOULOS

Formal Throughout the Design Cycle



What Can Formal Do For Me?

- What is formal?
- Where can formal be used?
- ➡ Applications for formal
- Wrap-up

DOULOS

Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

DOULOS

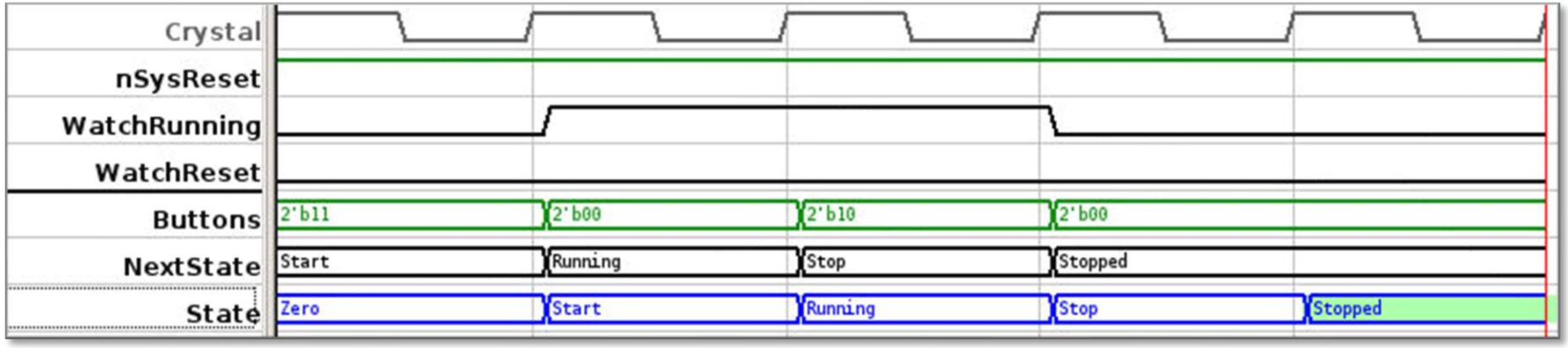
Design Exploration

```
unique case (State)
        Zero: if (Buttons[1]) NextState = Start;
       Start: begin
                 WatchRunning = 1;
                 if (!Buttons) NextState = Running;
              end
     Running: begin
                 WatchRunning = 1;
                 if (Buttons[1]) NextState = Stop;
              end
        Stop: if (!Buttons  ) NextState = Stopped;
     Stopped: if (Buttons[1]) NextState = Start;
              else if (Buttons[2]) NextState = Reset;
       Reset: begin
                 WatchReset = 1;
                 if (!Buttons) NextState = Zero;
              end
   endcase
```

```
cover property ( State == Stopped );
```

Formal Generated Trace

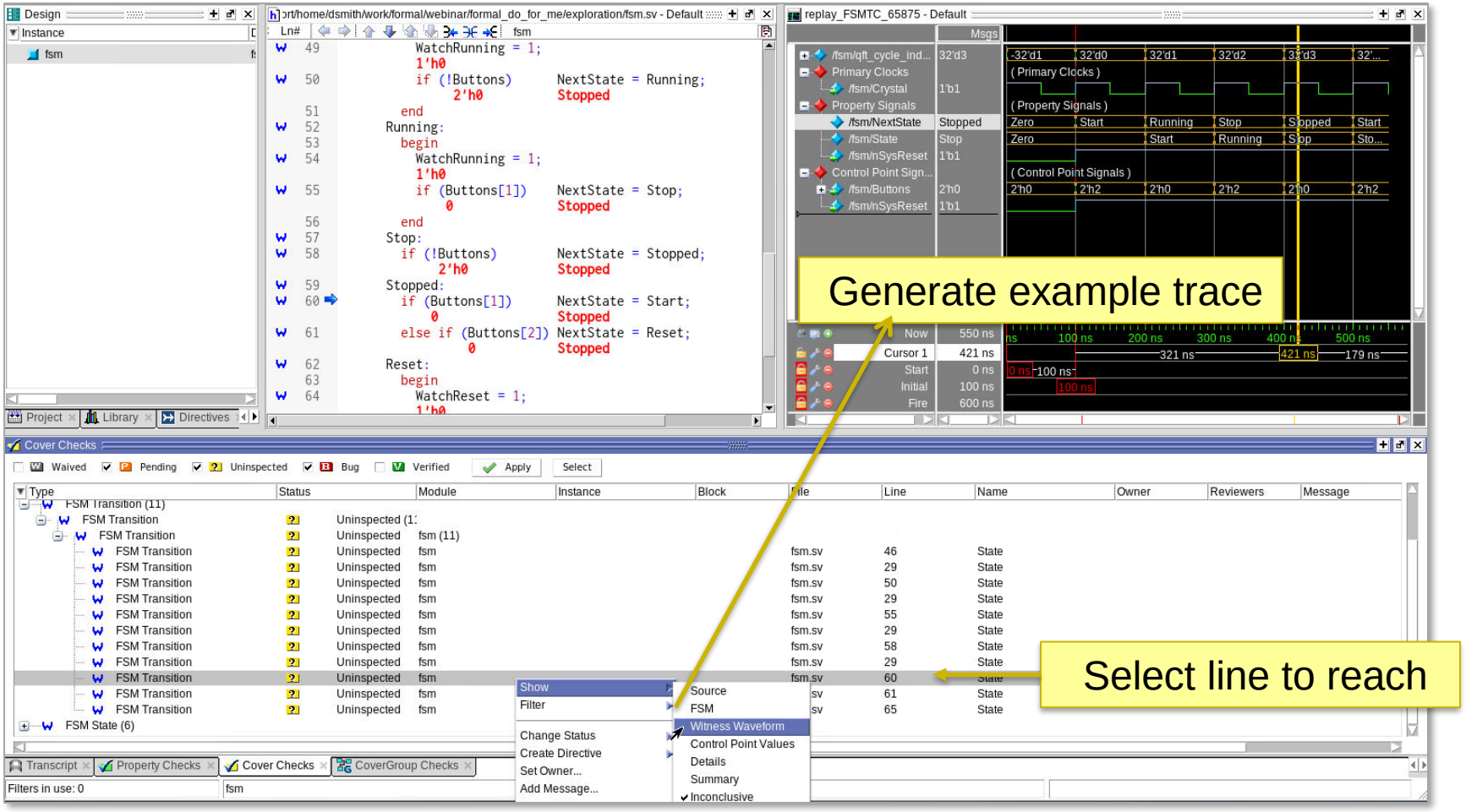
```
cover property ( State == Stopped );
```



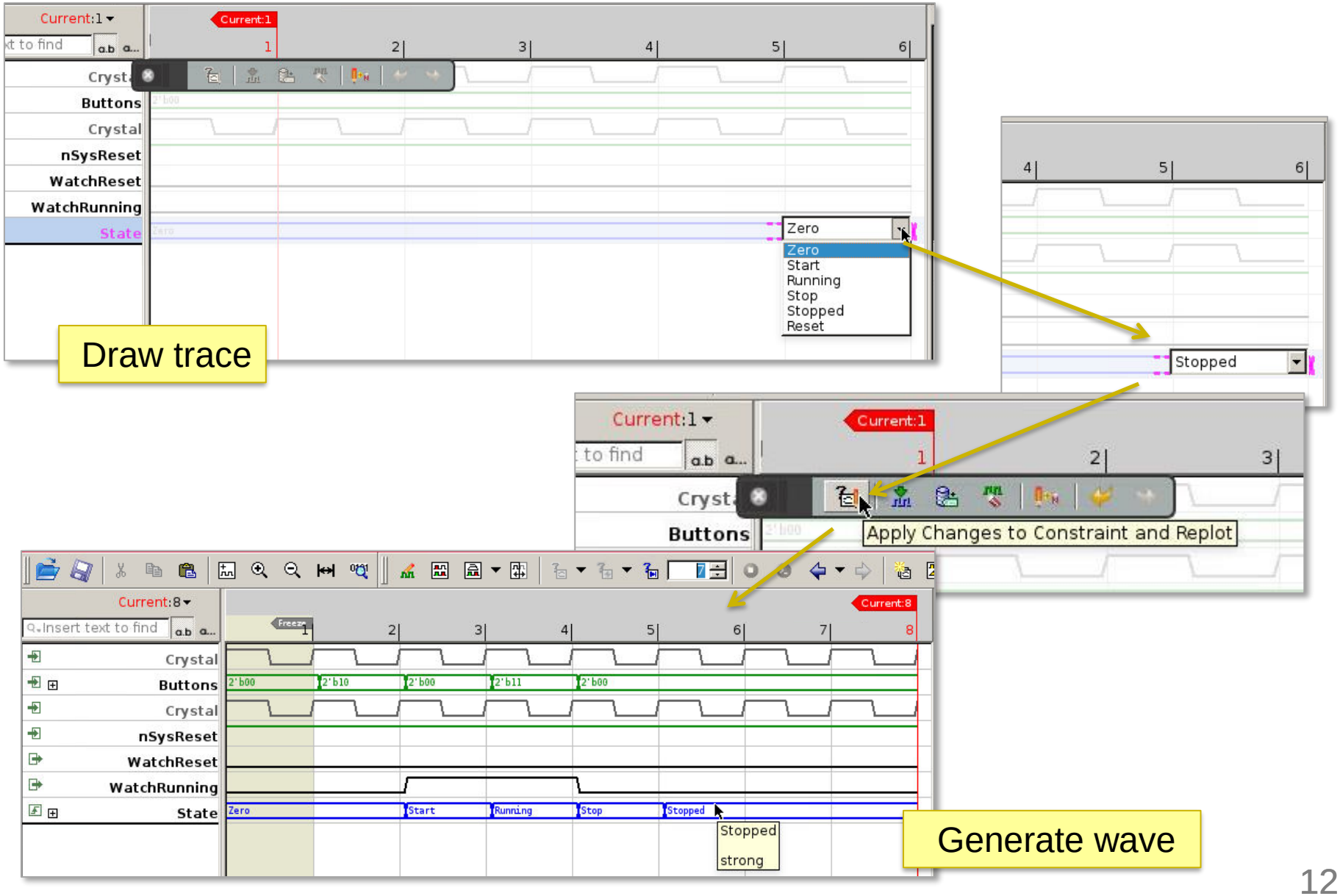
Design visualization with …

- No testbench
- No testcase

Auto Trace from Coverage App



Draw a Scenario



Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

DOULOS

Automatic Property Checking

Array bounds

Arithmetic overflow

Priority and unique case

Set and reset both active

Reachable X assignment

Deadlock / livelock

Incomplete sensitivity lists

... and others

Array Bounds Check

```
logic [7:0] address;
logic [0:3] array;
int k, n;

assign n = address >> 6;

always @(posedge clock)
  if (write)
    array[address] <= data_in;
  else if (read)
    data_out <= array[n];
  else
    data_out <= array[k];

c_k: assume property ( @(posedge clock) k >= 0 && k < 4 );
```

Bounds check fails

Bounds check okay

Bounds check may fail or not

```
a_1: assert property ( @(posedge clock) write |-> address < 4 );
```

Auto

Arithmetic Overflow Check

```
logic [7:0] address;


always @(posedge reset or posedge clock)
begin
  logic [3:0] sum;
  if (reset)
    sum <= 0;
  else
    sum <= sum + address;
end
```

Arithmetic overflow may fail or not

```
assert property ( @(posedge clock) disable iff (reset)
                  sum + address < 16 );
```

Auto

Unique Case Check

```
logic sel, c1, c2,

always @(posedge clock)
  unique case (sel)
    0: out2 <= 0;
    1: out2 <= 1;
  endcase

always @(posedge clock)
  unique case (sel)
    c1: out3 <= 0;
    c2: out3 <= 1;
  endcase
```

full_case and parallel_case both okay

full_case and parallel_case both fail

Auto

```
assert property ( @(posedge clock) c1 | c2 );
assert property ( @(posedge clock) !(c1 & c2) );
```

Other Automated Checking

Clock-domain crossing (CDC)

Reset-domain crossing (RDC)

Low-power UPF checks

Glitch checking

… and others

Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

DOULOS

Model Checking

Formal uses SVA for checking requirements

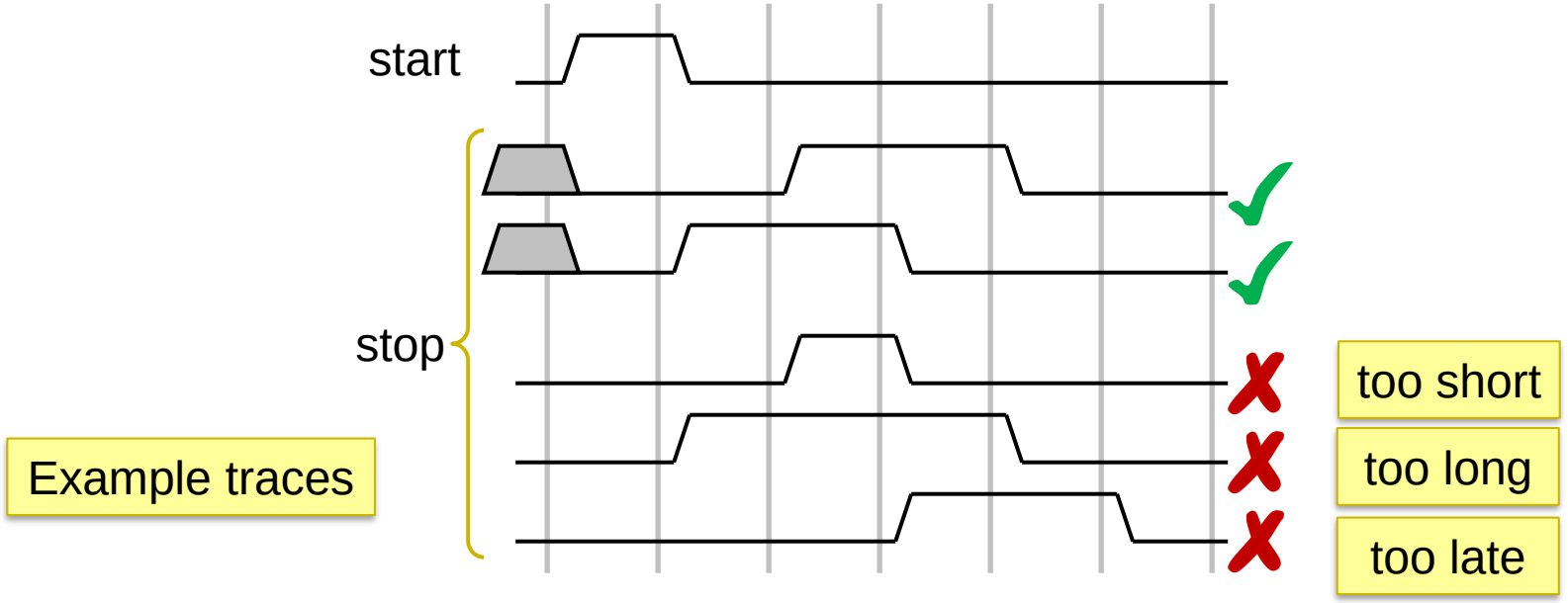
```
assert property ( !(WE & OE) );
assert property ( Size <= Max );
```

```
property incr_size;
  int sz;
  (Wr, sz = Size) ##1 !Ready[*1:$] ##1 Ready |-> Size == sz + 1;
endproperty

assert property ( incr_size );
```

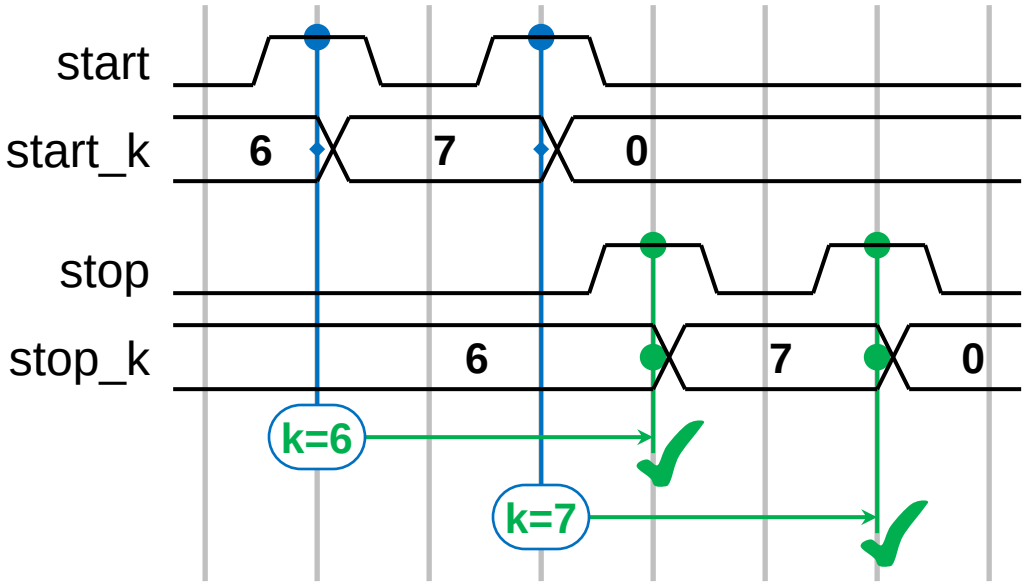
Capturing a Specification

After a *start* pulse, *stop* must go true on the next or second clock, and must remain true for exactly two clocks




```
assert property ( start |-> ##[1:2] stop [*2] );
```

Prove Protocol Correctness




```
bit [2:0] start_k, stop_k;
always @(posedge clk) begin
 if (start)
    start_k <= start_k + 1;
 if (stop)
    stop_k <= stop_k + 1;
end
```

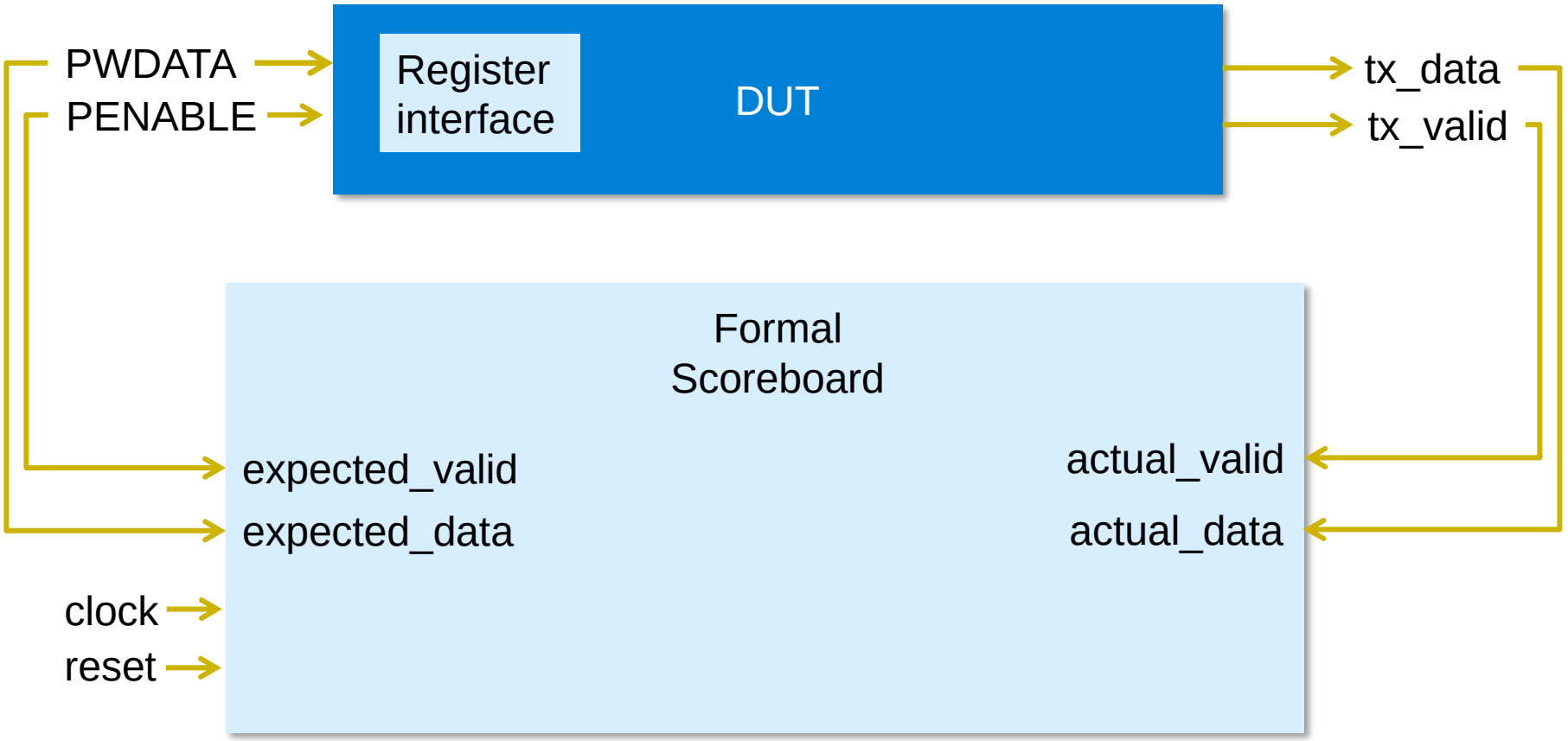
```
property overlap_start_stop;
  bit [2:0] k;
  (start, k = start_k)
  |-> ##[1:4]
  stop && (stop_k == k) ;
endproperty

assert property overlap_start_stop;
```

new variable for each instance of property

local variable assignment

End-to-End Checking



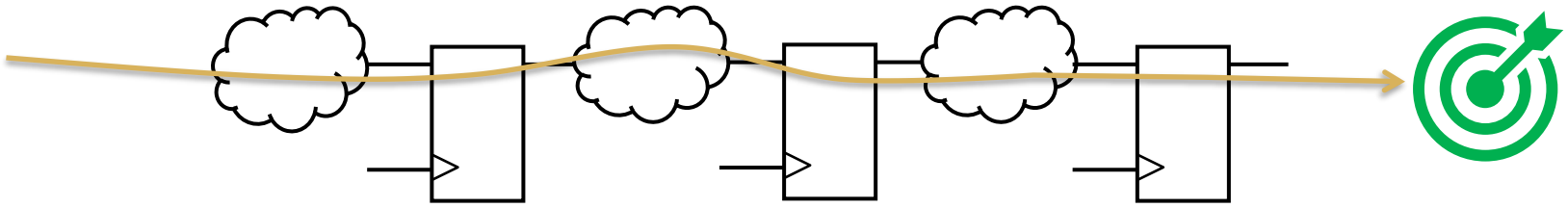
Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

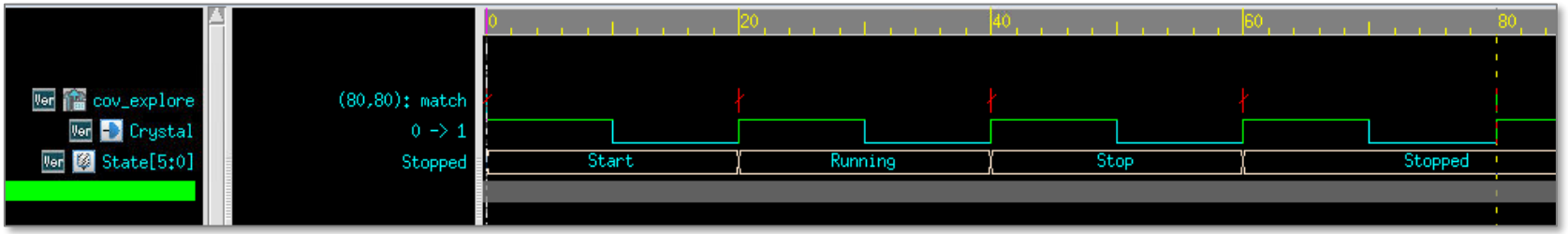
DOULOS

What Is Reachability?

Reachability – given any legal stimulus, is it possible to reach a scenario or line of code?



```
cover property ( State == Stopped );
```



Many Applications

Deadlock

Livelock

Vacuous assertions

Liveness

X-propagation

Connectivity

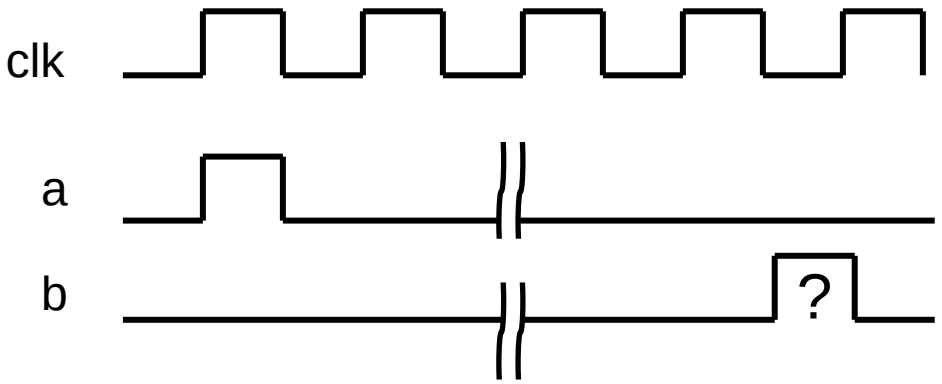
Registers

Security

Liveness

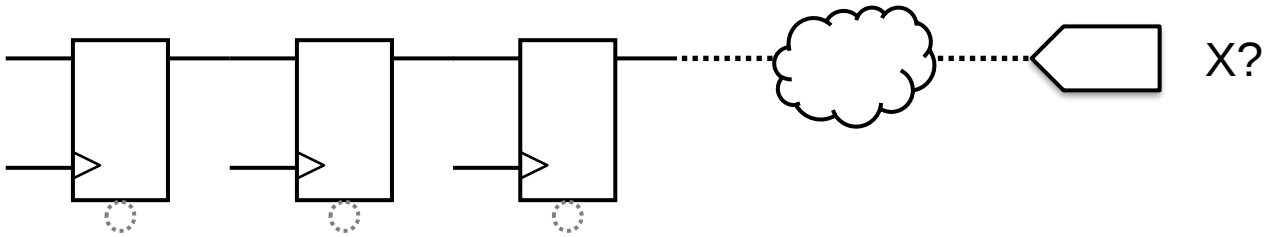
Does something eventually happen?

```
assert property ( a |-> s_eventually ( b ) );
```



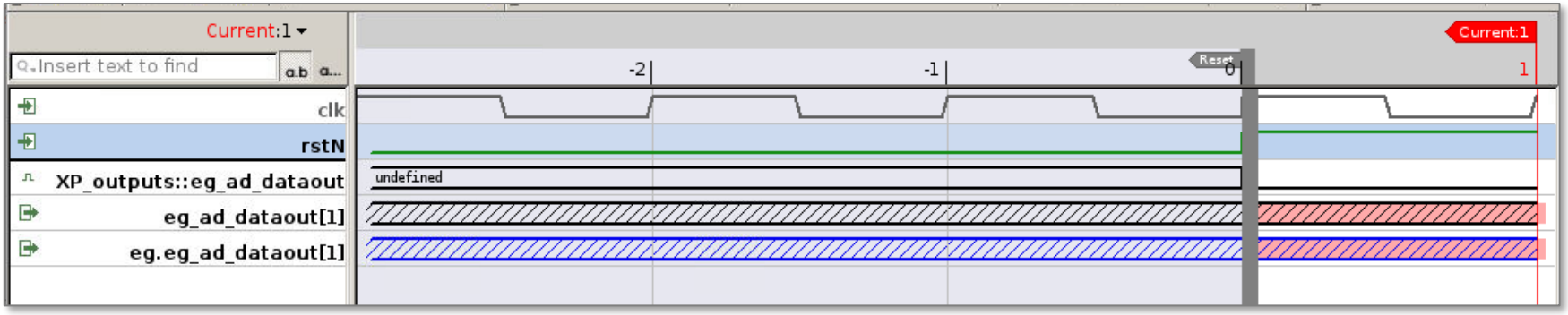
Hard (impossible) to prove in simulation

X Propagation

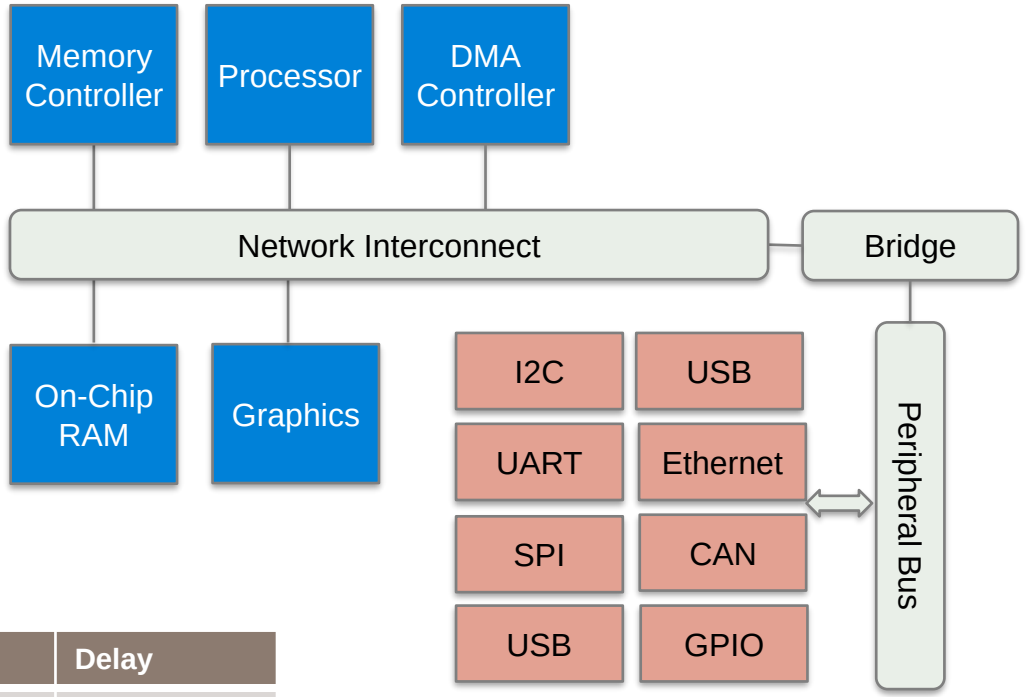


Non-resettable flops

```
cover property ( $isunknown( dataout ) );
```



Connectivity

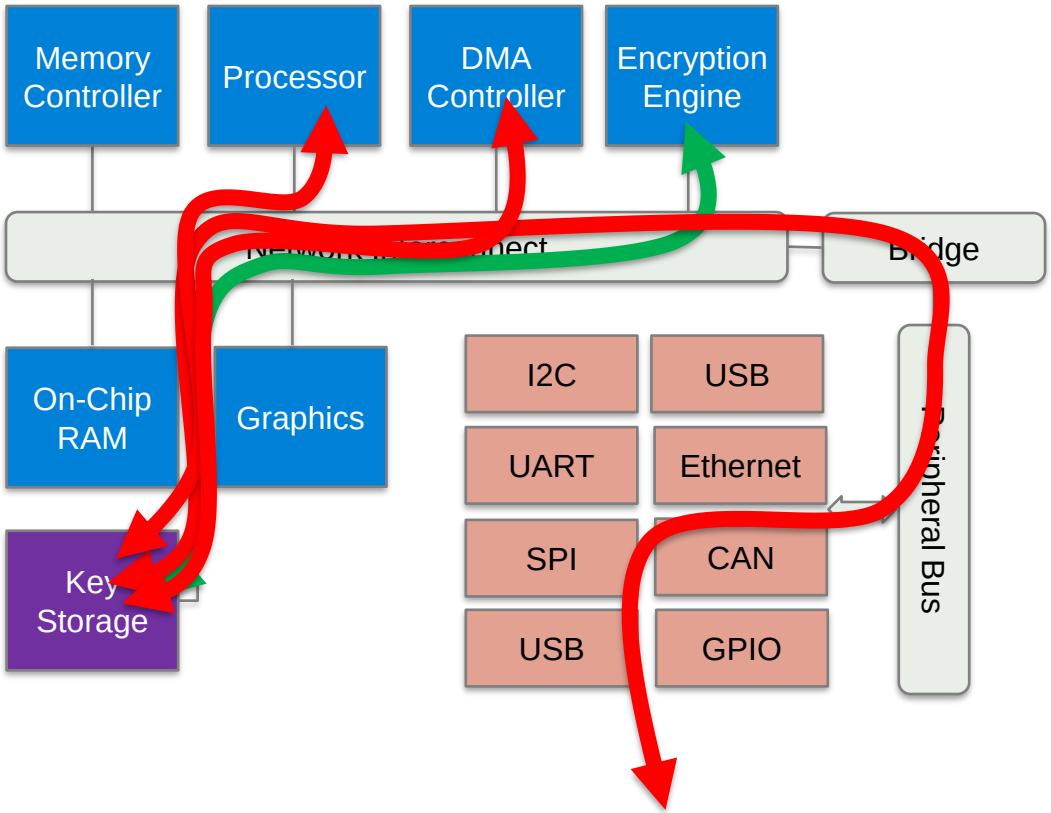


From the spec

Signal	From	To	Delay
PCLK	Processor	Interconnect	0
PWRITE	Processor	Interconnect	0
sig1	Processor	Graphics	3
…	…	…	…

```
assert property ( processor.PCLK == interconnect.PCLK );
assert property ( graphics.sig1 == $past(processor.sig1,3) );
```

Security

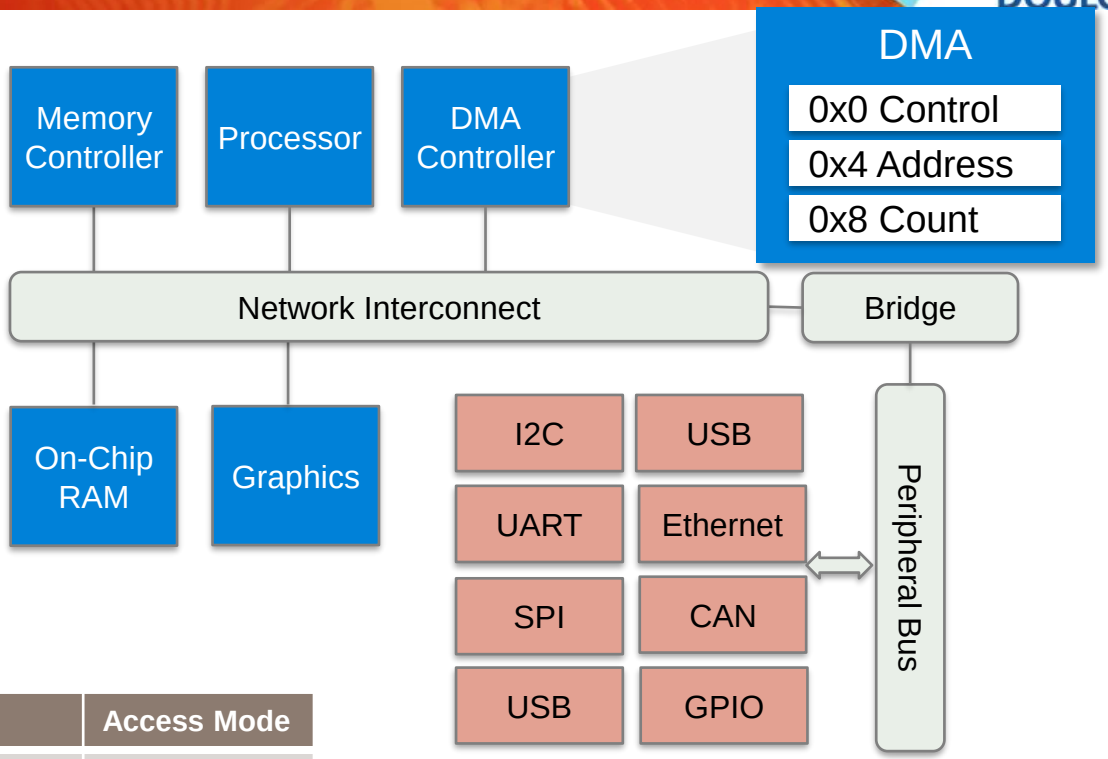


Limited key access?

Keys unreachable from other paths?

Provable by formal

Register Testing



From the spec

Register	Path	Offset	Access Mode
Config	soc.dma.ctrl	0x0	RW
Address	soc.dma.addr	0x4	RW
Count	soc.dma.count	0x8	RW
…	…	…	…

```
assert property ( sel && addr == 0x0 |-> soc.dma.ctrl == ... );
```

Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

DOULOS

Logic Equivalency Checking

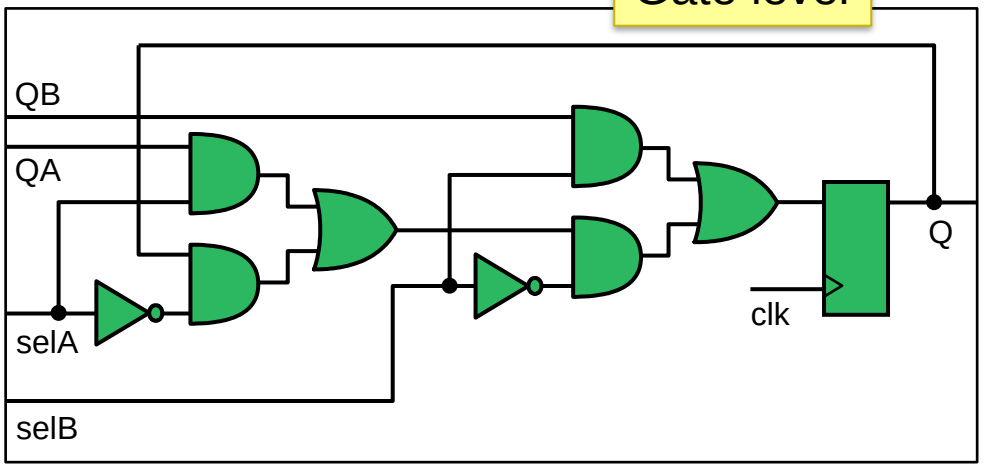
RTL

```
module selAB (
  input  logic clk,
  input  logic QA, selA, QB, selB,
  output logic Q
);

  always @(posedge clk)
    begin
      if (selA) Q <= QA;
      if (selB) Q <= QB;
    end

endmodule
```

Gate level



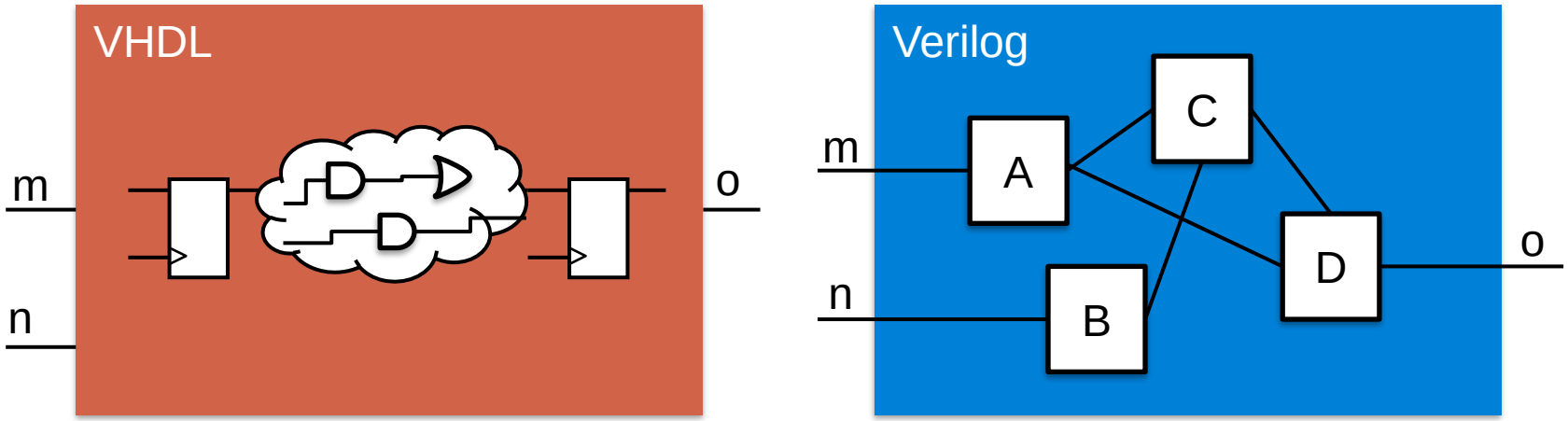
Are they functionally the same?

RTL versus gate-level netlist

Netlist versus netlist

Only works with recognizable equivalency points (signal names)

Sequential Equivalency Checking



Dynamic, not static like LEC – advances the clock

Shows equivalency between different implementations

Equivalency at the port-level

RTL <-> RTL, RTL <-> HLS (SystemC/C/C++)

Many Applications

VHDL <-> Verilog translation

Incremental feature updates (chicken bits)

ECO fixes

Data path verification

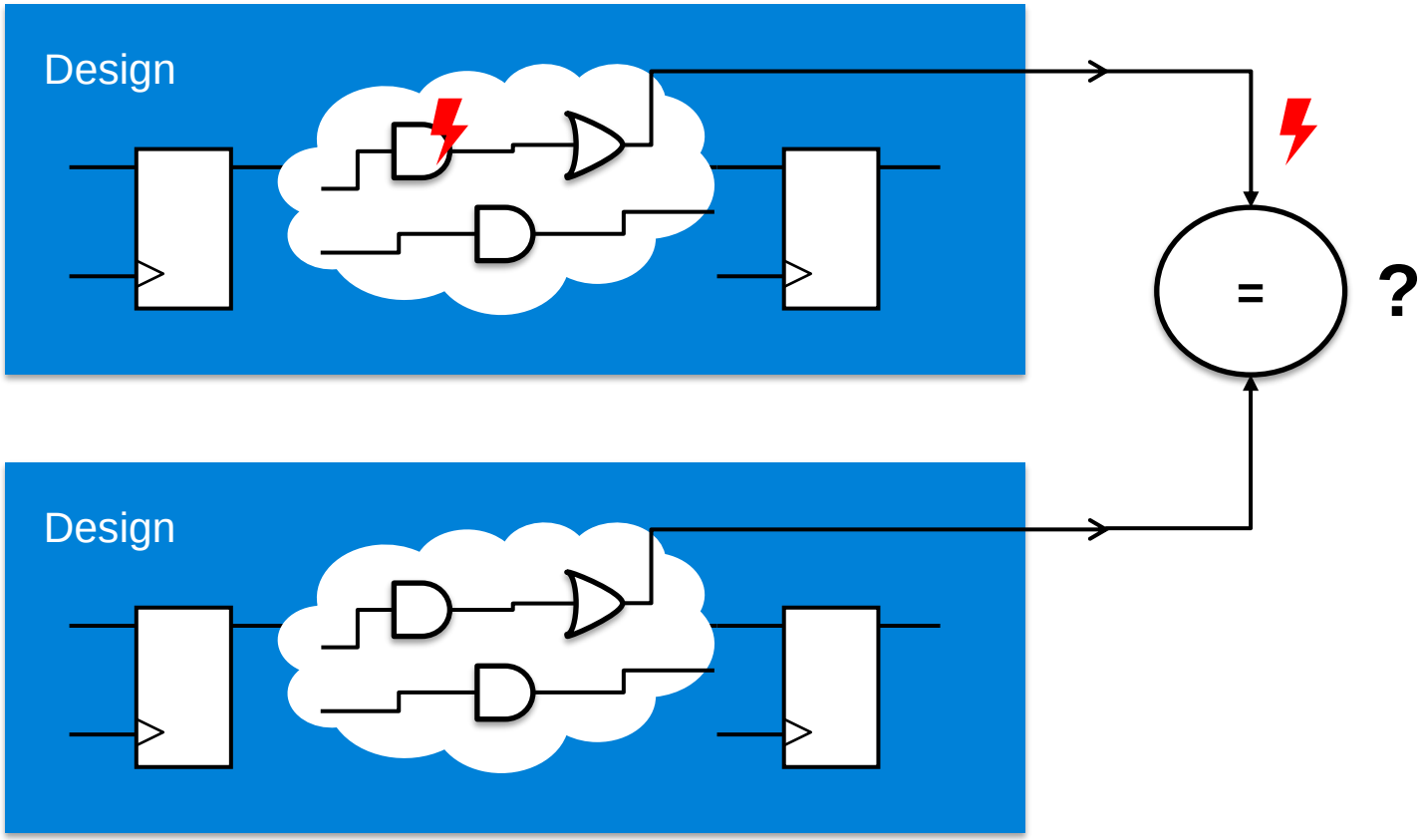
C to RTL equivalence

Functional safety

Fault injection

Safety mechanism insertion

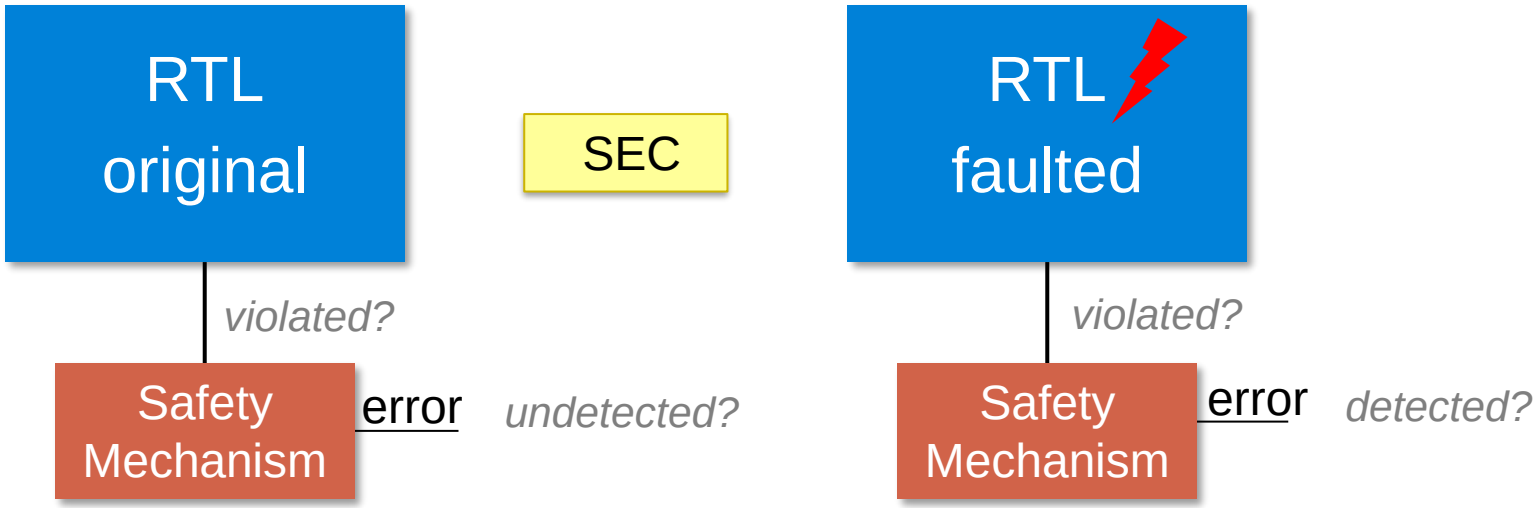
Fault Injection



SEC can traverse through state better than model checking

Simply check if outputs are affected by the injected fault

Functional Safety




```
int fault;
always @($global_clock) begin
   violation = injected && (  original.output != faulted.output );
   detected  = injected && ( !original.error  && faulted.error  );
   ...
```

```
// Inject fault (Tcl pseudo code)
cut faulted.signal -cond { fault == 1 } ...
```

ISO26262

Direct formal to a value

```
// Find residual fault(s)
cover property (( fault == 1 ) && violation && !detected );
```

Data Path Verification

```
// C algorithm
f_product = f16_mul(f_multiplier, f_multiplicand);
...
```

SEC

= ?

```
// RTL
module fmul #(...) ( input  logic [SIZE-1:0] multiplier,
                     input  logic [SIZE-1:0] multiplicand,
                     output logic [SIZE-1:0] product,
...
```

State space too large for model checking

May only be able to verify with formal using SEC

Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

DOULOS

Areas Formal Helps Sign-off

Achieving coverage closure in simulation

Creating simulation testbenches to hit coverage holes

Measure assertion quality

Formal coverage

Testplan and testcase generation

Reachability

Coverage Exclusions for Simulation

Formal finds unreachables and generates exclusions

```
<formal_tool> generate exclude exclude_file.tcl
```

```
coverage exclude -scope
{/tb_axi4lite_2_apb4/dut/u_master_interface/u_apb_master_s
c} -srcfile .../src/vlog/apb_master_sc.v -linerange 88 -
item s 1 -reason "EU"
coverage exclude -scope
{/tb_axi4lite_2_apb4/dut/u_master_interface/u_apb_master_s
c} -srcfile .../src/vlog/apb_master_sc.v -linerange 106 -
item s 1 -reason "EU"
```

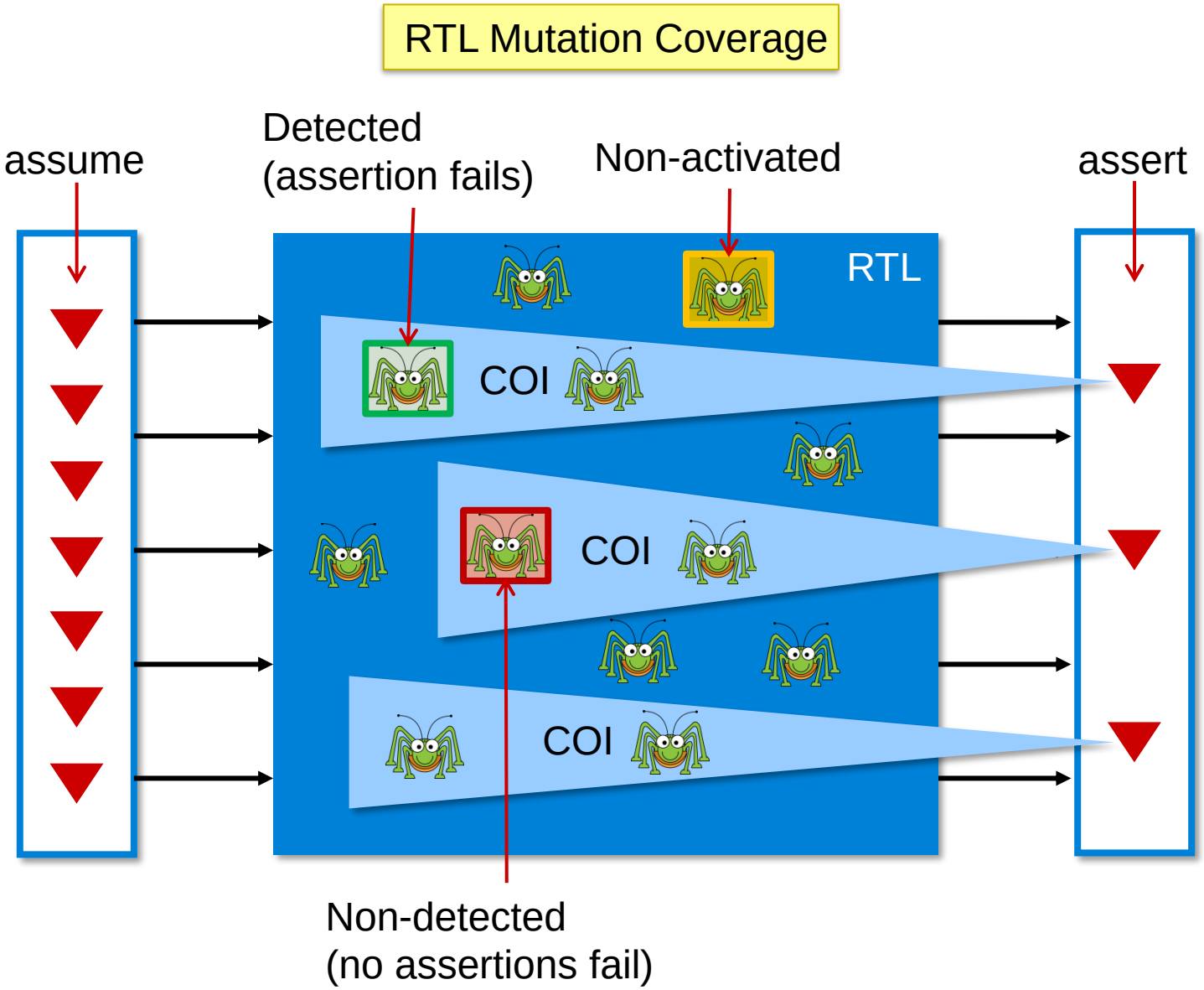
Testbench Generation

Generate stimulus to target coverage holes

```
<formal_tool> generate testbenches
```

```
module replay_vlog;
initial begin
    #1;
    force axi4lite_to_apb4.use_1clk_i = 1'b0;
    force axi4lite_to_apb4.PRESETn_i = 1'b0;
    force axi4lite_to_apb4.PREADY_i = 1'b0;
    force axi4lite_to_apb4.PSLVERR_i = 1'b0;
    force axi4lite_to_apb4.PSELx_i_csr = 1'b0;
    ...
```

Measuring Assertion Quality



Formal Coverage

- Assertion density – are there enough?
  - Cone-of-influence (COI) coverage
  - Proof core coverage
- Code coverage
  - Proof core coverage
- Functional coverage
  - Cover properties
  - Synthesizable covergroups
- Assertion quality
  - Mutation coverage

Merges with simulation coverage

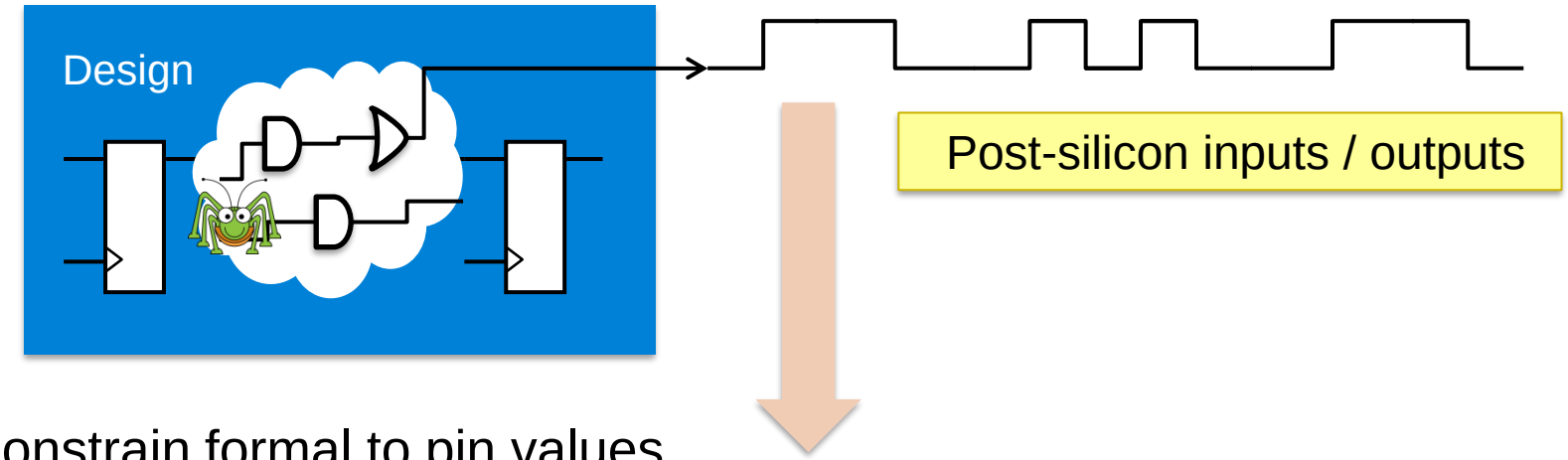
Applications for Formal

- Design exploration
- Automatic design checking
- Model checking
- Reachability
- Equivalence
- Sign-off
- Post-silicon

DOULOS

Post-Silicon Debug

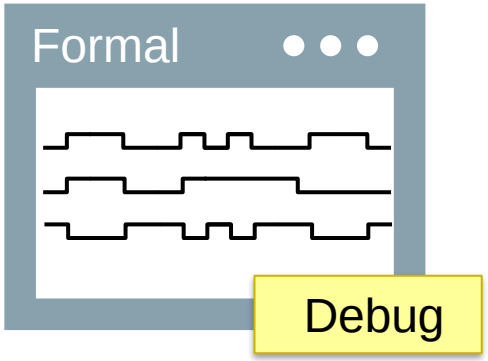
Formal can reproduce post-silicon results for debug



Constrain formal to pin values

```
assume property ( pins == ... );
```

```
cover property ( state == ERROR );
```



What Can Formal Do For Me?

- What is formal?
- Where can formal be used?
- Applications for formal
- ➡ Wrap-up

DOULOS

Summary

Formal complements your simulation flow

Formal verifies scenarios hard or tedious in simulation

Formal can be part of any verification planning and effort

Why would you not take advantage of what formal can do?

Thank you for attending

We hope you found this information helpful!

DOULOS

DOULOS

Delivering KnowHow www.doulos.com

SoC Design & Verification

» SystemVerilog » UVM » Formal
» SystemC » TLM-2.0

FPGA & Hardware Design

» VHDL » Verilog » SystemVerilog
» Tcl » Xilinx » Intel FPGA (Altera)

Embedded Software

» Emb C/C++ » Emb Linux
» Yocto » RTOS » Security » Arm

Python & Deep Learning

SystemVerilog OUT OF THE BOX

UVM

SYSTEMC

yocto PROJECT PARTICIPANT

freeRTOS

SECURE EMBEDDED

arm APPROVED TRAINING PARTNER