2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 15-16, 2024

G-QED for Pre-Silicon Verification

Saranyu Chattopadhyay*, Keerthi Devarajegowda+, Mo Fadiheh*

Stanford University*, Siemens EDA+

SIEMENS

accellera SYSTEMS INITIATIVE

Traditional Verification Still a Challenge

Extreme Heterogeneity

2022 Apple M series

Short design-to-deployment

Industry Breaking its Back

>80% projects: Critical bugs missed
[Siemens Wilson Research 2022]

Generalized Quick Error Detection

Sound and Complete

Any *moderate-sized digital design*:

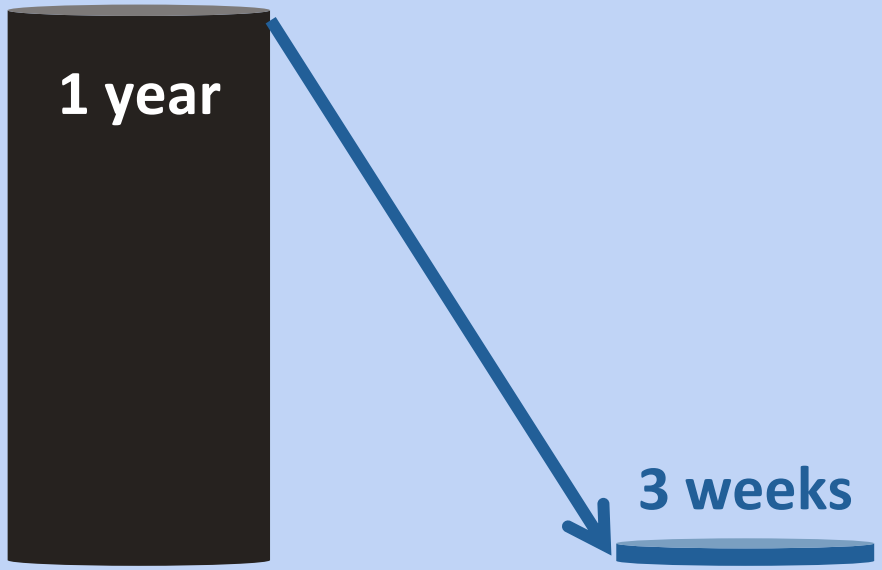
- well-defined ***actions***
- ***arch states*** observable
- can ***idle***

(No gory implementation detail)

9 New Critical Bugs + Rest

AI Hardware Accelerators

(🏭production ready)



Agenda

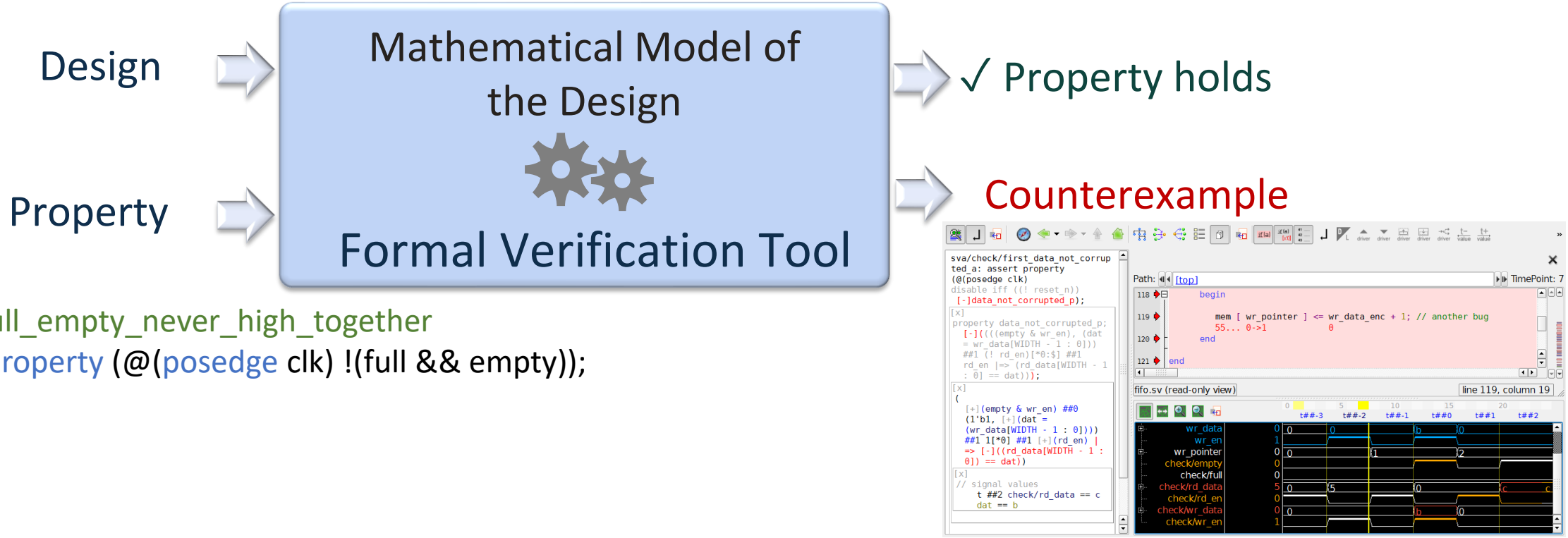
1. What is Formal Verification?
2. G-QED for Designers
3. G-QED Checks Deep Dive
4. G-QED Demo

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

What is Formal Verification?

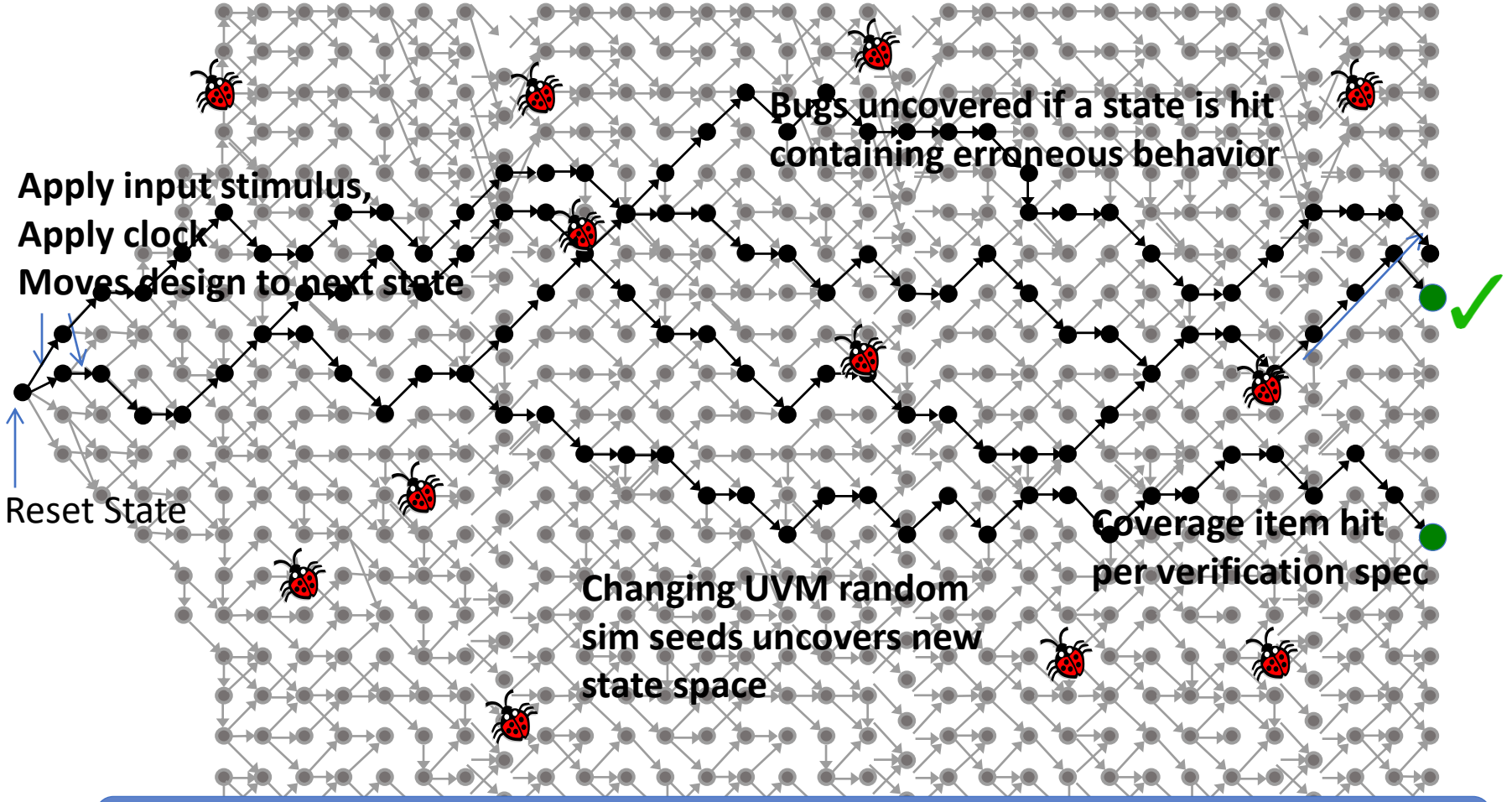
accellera SYSTEMS INITIATIVE

Formal Verification



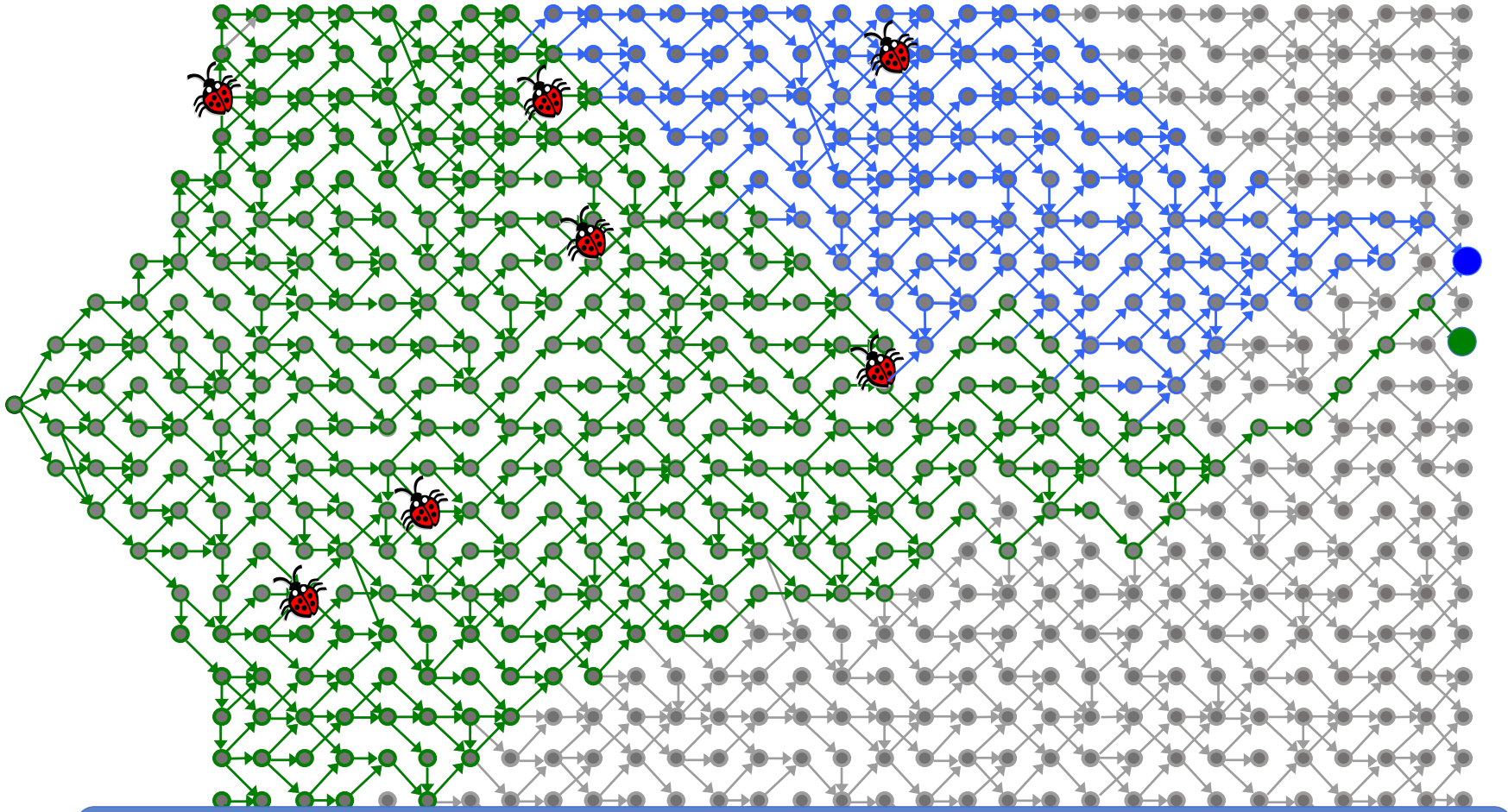

```
//ast_full_empty_never_high_together
assert property (@(posedge clk) !(full && empty));
```

Formal Verification Vs Simulation



But - Even after extensive simulation, significant state space remains uncovered

Formal Verification Vs Simulation



Formal starts with the desired state and works backward to create exhaustive proof, verifying extensive state space in the process

Verification Challenges

Simulation

- Heuristics for coverage metrics
- Miss corner-cases
- AI-assisted: coverage problems

Formal Verification

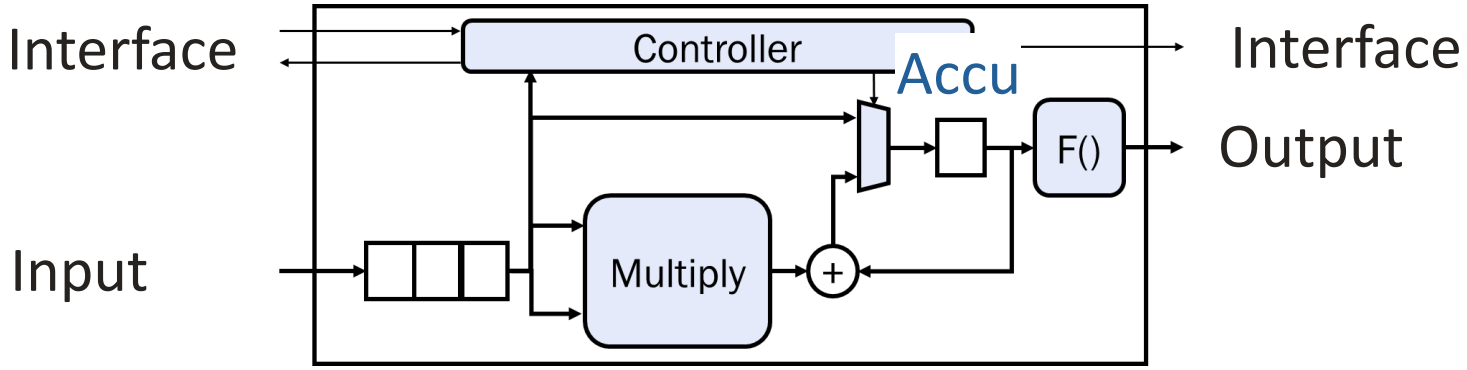
- No thoroughness guarantees
- Time consuming
- Formal tools don’t scale well

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

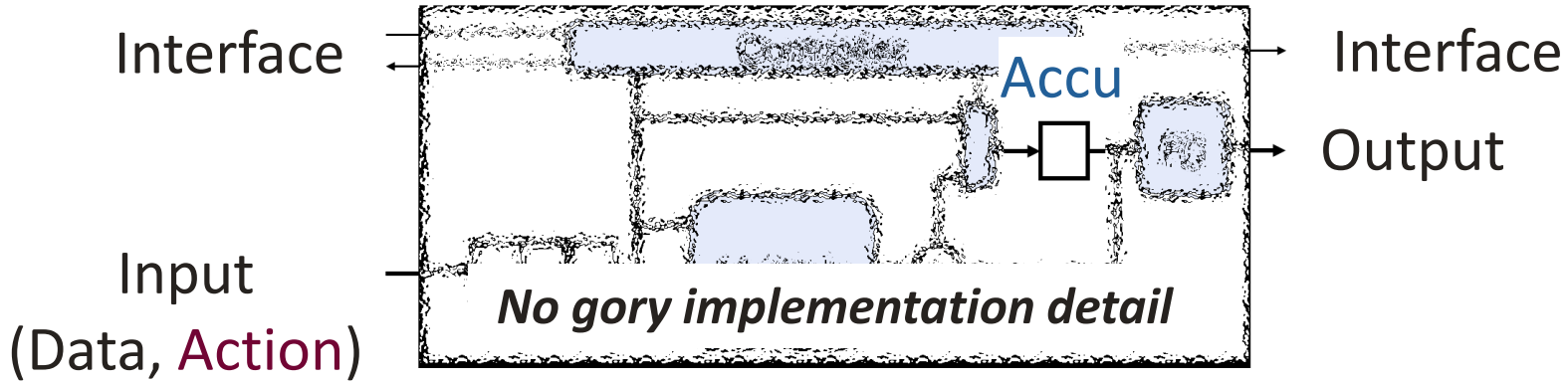
G-QED for Designers

accellera SYSTEMS INITIATIVE

Example Design Under Verification



G-QED for Designers



like Instructions like Software-visible states

Action Set	Arch State Update	Output
SetVal	Accu = Data	
MAC	Accu = Accu+$Data_0$×$Data_1$	
OutGen		*F*(Accu)

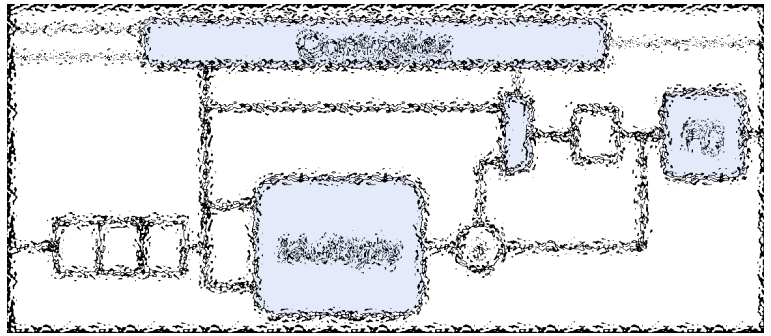
G-QED for Designers

Action Set
Arch State

Upper bound: action execution time T_{UB}

Interface protocol

DUV RTL



G-QED Harness
implements G-QED Checks

Formal Verification Tool

Action sequence failing G-QED checks

G-QED Checks: Sound & Complete

Functional Consistency

Response Bound

Single Action Correctness

G-QED Checks: Sound & Complete

Functional Consistency

Response Bound

Single Action Correctness

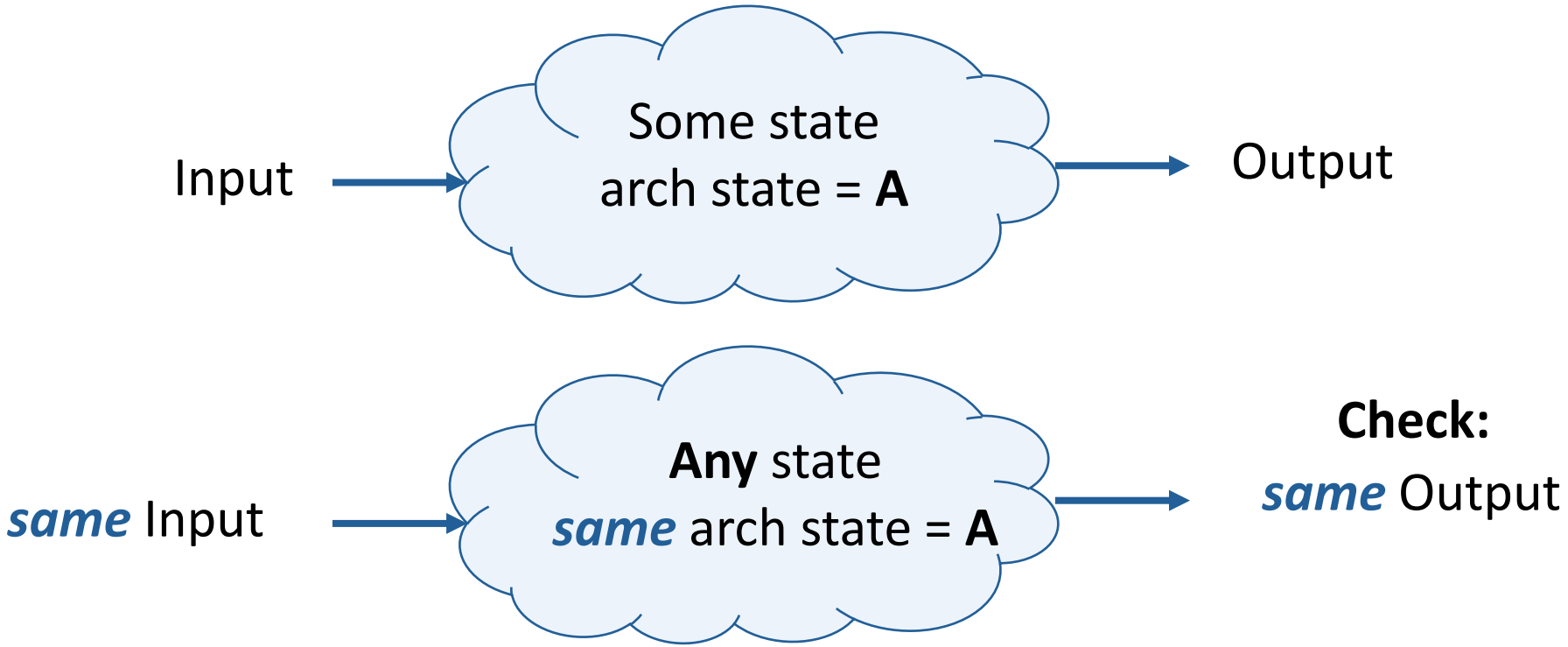
RB, SAC: routinely in industry

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

G-QED Checks Deep Dive

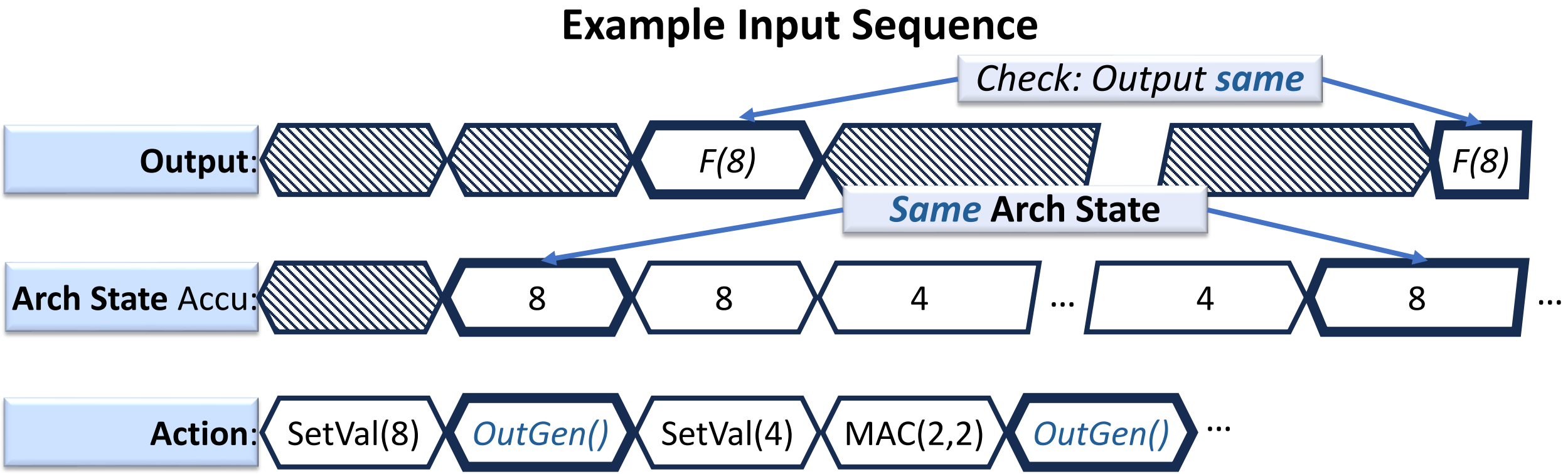
accellera SYSTEMS INITIATIVE

Functional Consistency (FC) Concept



Expected: Output consistent irrespective of context

FC Check on Action OutGen()



Formal Tool: Check for all input sequences exhaustively

FC Implementation Problem

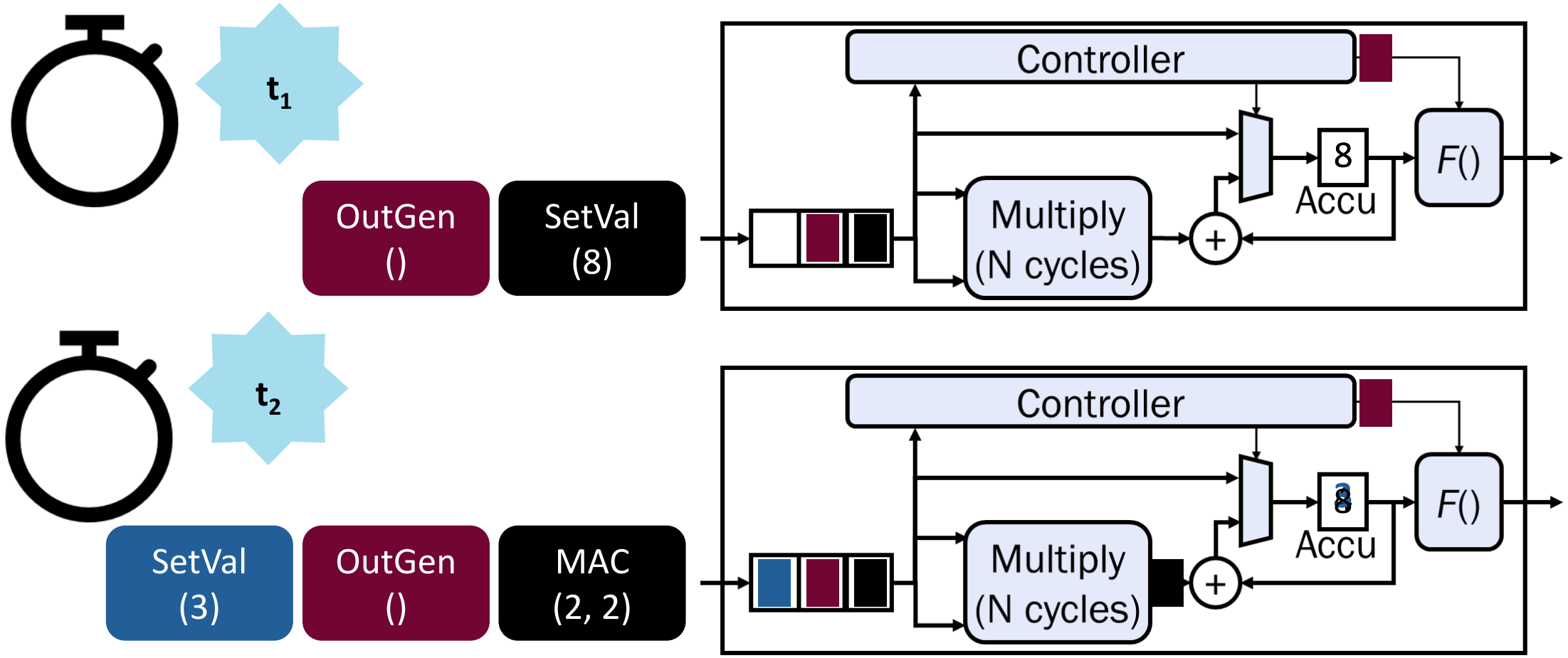
*Difficult to implement **FC** this way*

arch state @ t_1 == **arch state** @ t_2

→ ***same*** **Output**

Hard to specify exact clock cycles t_1 and t_2

Hard to Specify Exact Clock Cycles



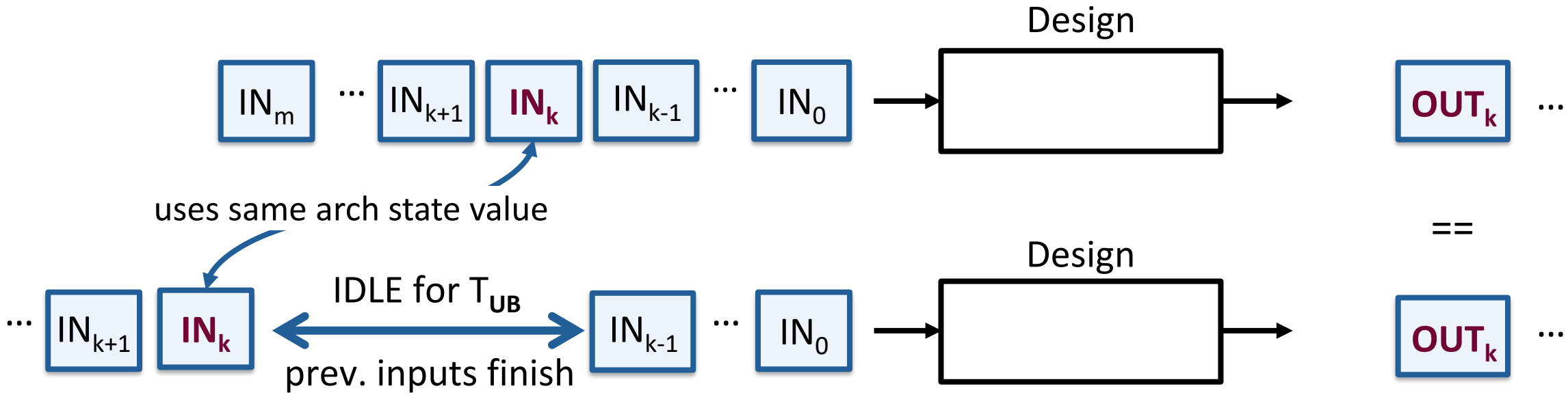
Gets Worse Generalizing for Any Design

Arch state elements read **over multiple cycles**

Read cycles **non-contiguous**

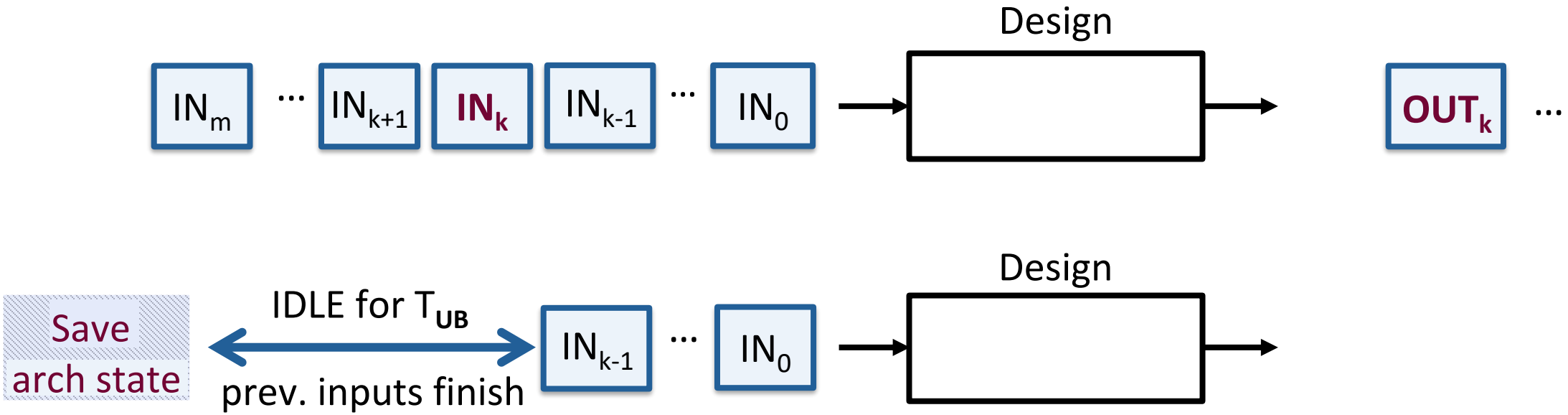
Same arch state element read **multiple times**

Solution: G-QED

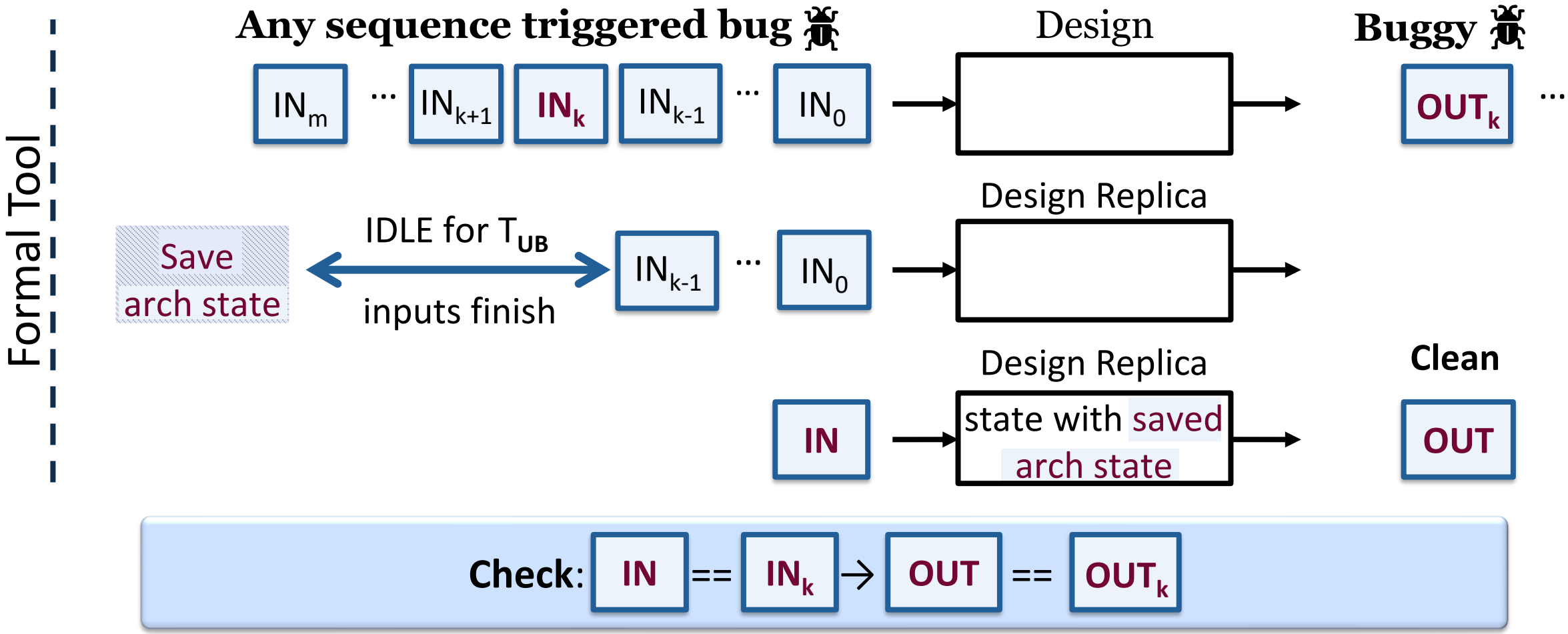


$OUT_i = f(IN_i$, arch state value used by $IN_i)$

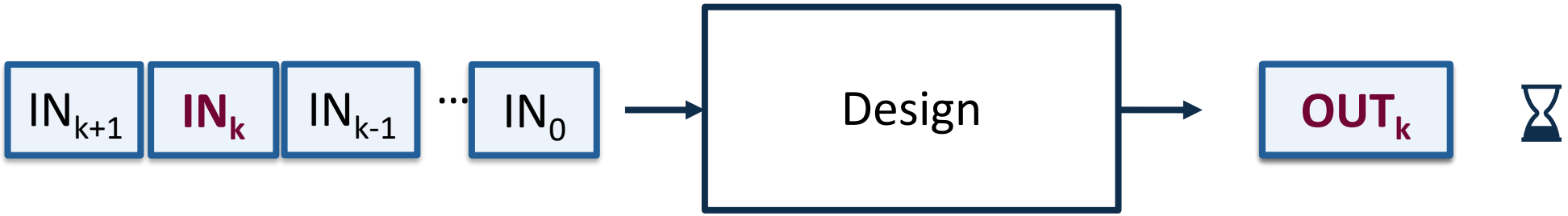
Solution: G-QED



Solution: G-QED

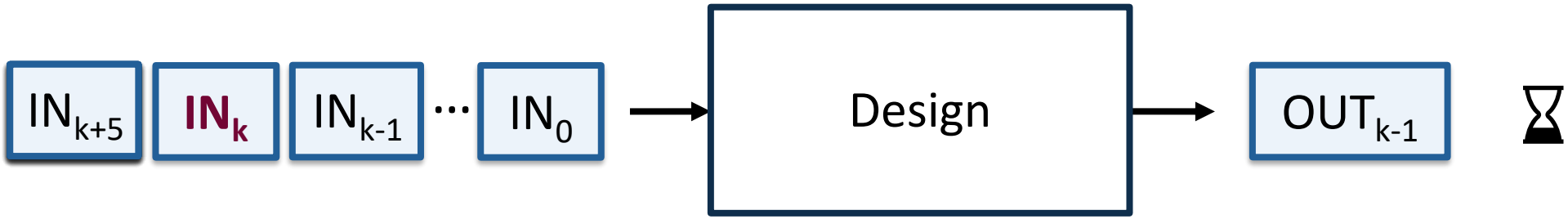


Response Bound



OUT_k produced within T_{UB} since IN_k fed

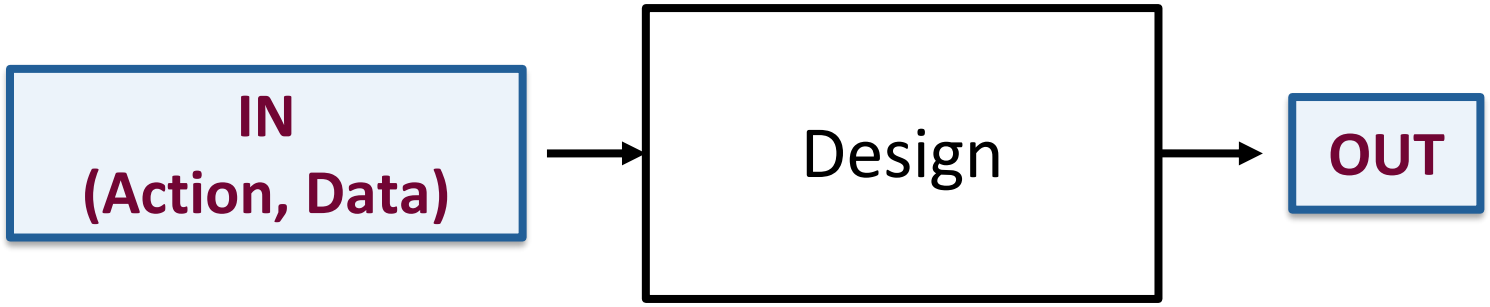
Response Bound Bug



OUT_k not produced within T_{UB}

Single Action Correctness

- FC takes care of bugs related to sequence of instructions or inputs
- Single Action Correctness can be checked from a known reachable state
  - From ***some*** reachable state (e.g., reset state)



Check: OUT == Action(Data, Arch state)

(done routinely in industry)

Single Action Correctness

- Single Action Correctness is straightforward to apply
- However, writing SVA (properties) for complex operations can still be challenging (ex: Floating Point Units)

Single Action Correctness for Floating Point Units

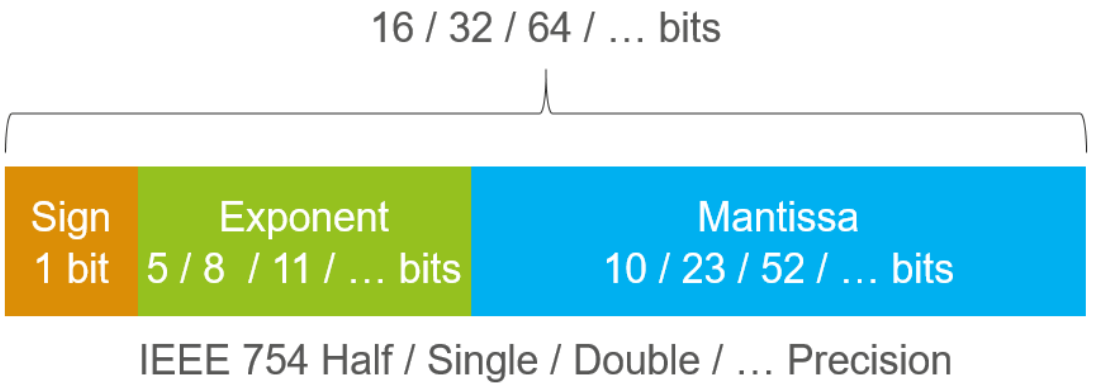
Challenges

- Complex IEEE 754 floating-point specification
  - Arithmetic and comparison operations
  - Bfloat16, half, single, and double precision
  - Five rounding modes
  - Five exception flags
- IEEE 754 design and verification experts in short supply
- Simulation cannot guarantee compliance

Floating-point essential for advanced artificial intelligence applications

Multiply/add loops in convolution layer of a CNN

```
for(r=0; r<R; r++)              //output feature map
  for(q=0; q<Q; q++)            //input feature map
    for(m=0; m<M; m++)          //row in feature map
      for(n=0; n<N; n++)        //column in feature map
        for(k=0; k<K; k++)      //row in convolution kerenel
          for(l=0; l<L; l++)    //column in convolution kernel
            Y[r][m][n]+=W[r][q][k][l]*X[q][m+k][n+l];
```

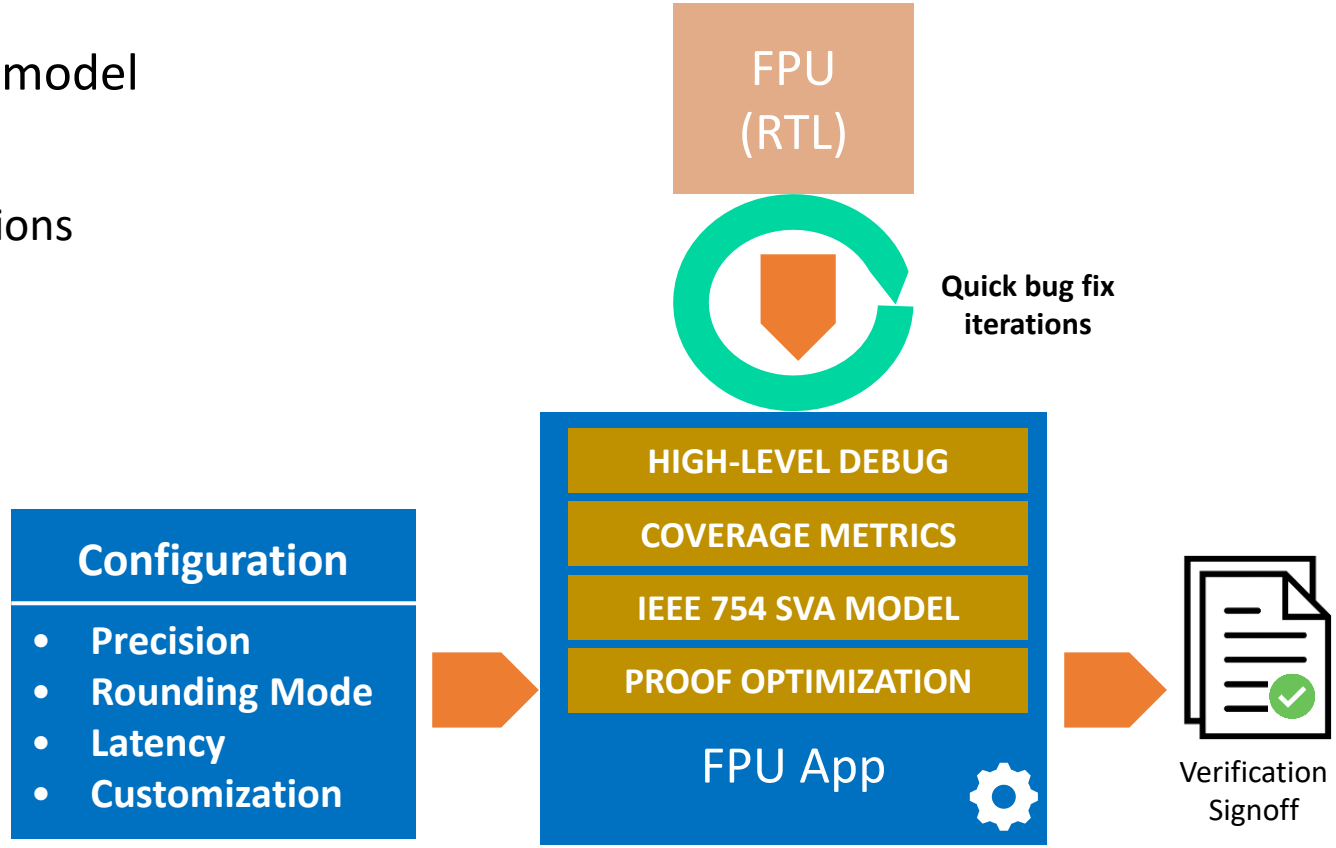


Only formal can prove compliance

Questa FPU

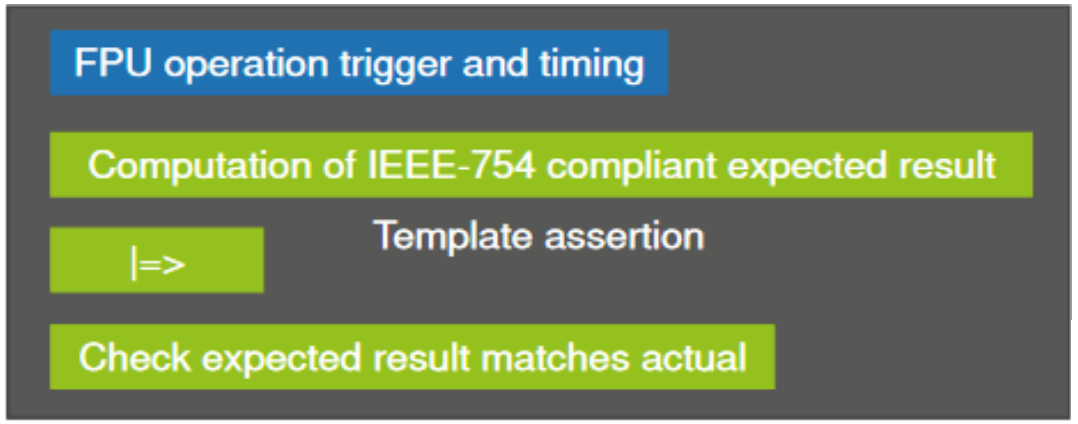
Ensure FPU designs meet expectations

- Prove correctness with easy setup, without C++ model
- Supports broad array of FPU functionality
  - ***Half/single/double, bfloat16*** and ***custom*** precisions
  - ***All 5 rounding modes*** and ***5 exceptions flags***
  - ***ADD, SUB, MUL, DIV, FMA, ABS, NEG*** functions
  - ***Conversion*** and ***comparison*** operations
- Parameters specify ambiguities in the standard
- Easy to model intended deviations from the IEEE-754 standard
- Enables RISC-V configuration



Built-in template assertion

- All standard-derived information part of template
  - User provides mapping of FPU DUT to SVA module
  - Design specifics like encoding of operations, modes, and concrete protocol added by user
  - Extraction supported by numerous covers to extract timing and encoding



green: provided by App
blue : provided by user

Built-in template assertion

fp_template.sv (~/demos/2.11_FPU/sva) - GVIM9 (on orw-gnahum-vm)

File Edit Tools Syntax Buffers Window Help

```
module fp_checker import ieee_754_single_precision::*; #(
    conf_t       conf = '{UNF_BEFORE_ROUNDING: 1'b1, // set to 0 if tinyness is computed after rounding
                          INV_FMA_QNAN: 1'b1, default:1'b1}, // set to 0 if fma of 0*Inf+NaN is not raising an INV exception
    ieee_flags_t supported_flags = '{INX: 1'b1,  // set to 0 to skip 'inexact' exception check
                  INV: 1'b1,  // set to 0 to skip 'invalid' exception check
                  UNF: 1'b1,  // set to 0 to skip 'underflow' exception check
                  OVF: 1'b1,  // set to 0 to skip 'overflow' exception check
                  DIV0: 1'b1}) (// set to 0 to skip 'division_by_0' exception check
    input           clk,
    input           reset_n,
    (* OSS_TCL_VALUE_PRINTER = "ieee754_fp" *)
        input ieee_t    op_a,           // DUT source operands
    (* OSS_TCL_VALUE_PRINTER = "ieee754_fp" *)
        input ieee_t    op_b,           // DUT source operands
    (* OSS_TCL_VALUE_PRINTER = "ieee754_fp" *)
        input ieee_t    op_c='0,        // DUT fma 3rd source operands
    input           start=1'b1,     // DUT start of computation with source
    input[1:0]      rmode='0,       // rounding mode in DUT encoding
// TODO: adapt bit width of fpu operation encoding
    input[2:0]      fpu_op=0,       // signal encoding fpu operation in DUT (please increase bitwidth if needed)
    (* OSS_TCL_VALUE_PRINTER = "ieee754_fp" *)
        input ieee_t    result,         // DUT result of computation
    input           result_valid,   // DUT result valid
sva/fp_template.sv
// end of covers

endmodule

bind fpu fp_checker #(
    .supported_flags('{INV:1'b0, default:1'b1})
) fp_checker (
// leave yet unknown signals (like fpu_op, rmode or flags) unconnected and reverse-engineer with contained covers
    .clk(clk_i),
    .reset_n(1'b1),
    .start(start_i),       // DUT start of computation with source operands
    .fpu_op(fpu_op_i),      // signal encoding fpu operation in DUT
    .op_a(opa_i),        // DUT source operand a
    .op_b(opb_i),        // DUT source operand b
    .op_c(),        // DUT source operand c
    .rmode(rmode_i),       // rounding mode in DUT encoding
    .result(output_o),      // DUT result of computation
    .result_valid(ready_o),// DUT result valid
    .INX_flag(ine_o),    // DUT 'inexact' flag
    .OVF_flag(overflow_o),    // DUT 'overflow' flag
    .UNF_flag(underflow_o),    // DUT 'underflow' flag (aka tiny)
    .DIV0_flag(div_zero_o),    // DUT 'division_by_0' flag
    .INV_flag());   // DUT 'invalid' flag
```

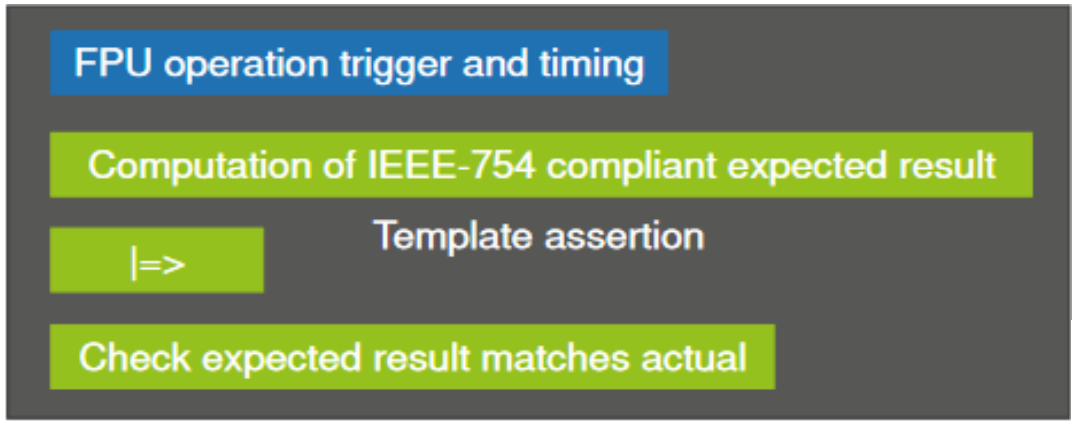
Can mix and match different precisions

IEEE flags definition

Connect to the design via bind statement

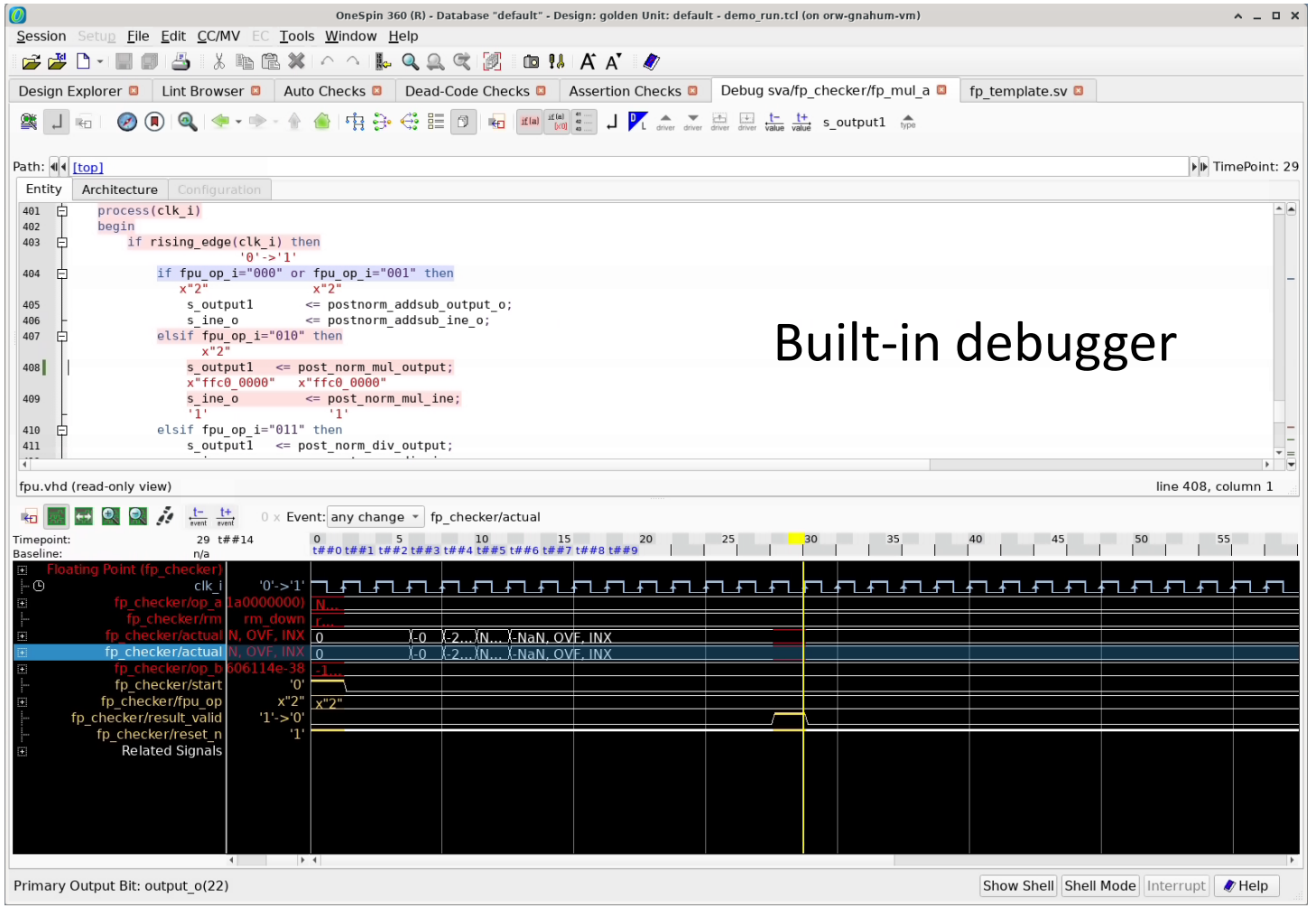
Built-in template assertion

```
sum <= ieee_add(.a(op_a), .b(op_b), .rm(rm), .conf(conf));
property fp_add_p;
    ieee_with_flags_t expected;
    disable iff (~reset_n)
    (start && fpu_op==2'b01, expected = ieee_add(.a(op_a), .b(op_b), .rm(rm), .conf(conf)))
|->
    ##3
    result_valid && ieee_check_result(.expected(expected),
                                      .actual(actual),
                                      .supported_flags(supported_flags),
                                      .conf(conf));
endproperty
fp_add_a : assert property (fp_add_p);
```



green: provided by App
blue : provided by user

Assertions and cover properties



Built-in debugger

Questa FPU

Ensure FPU designs meet expectations

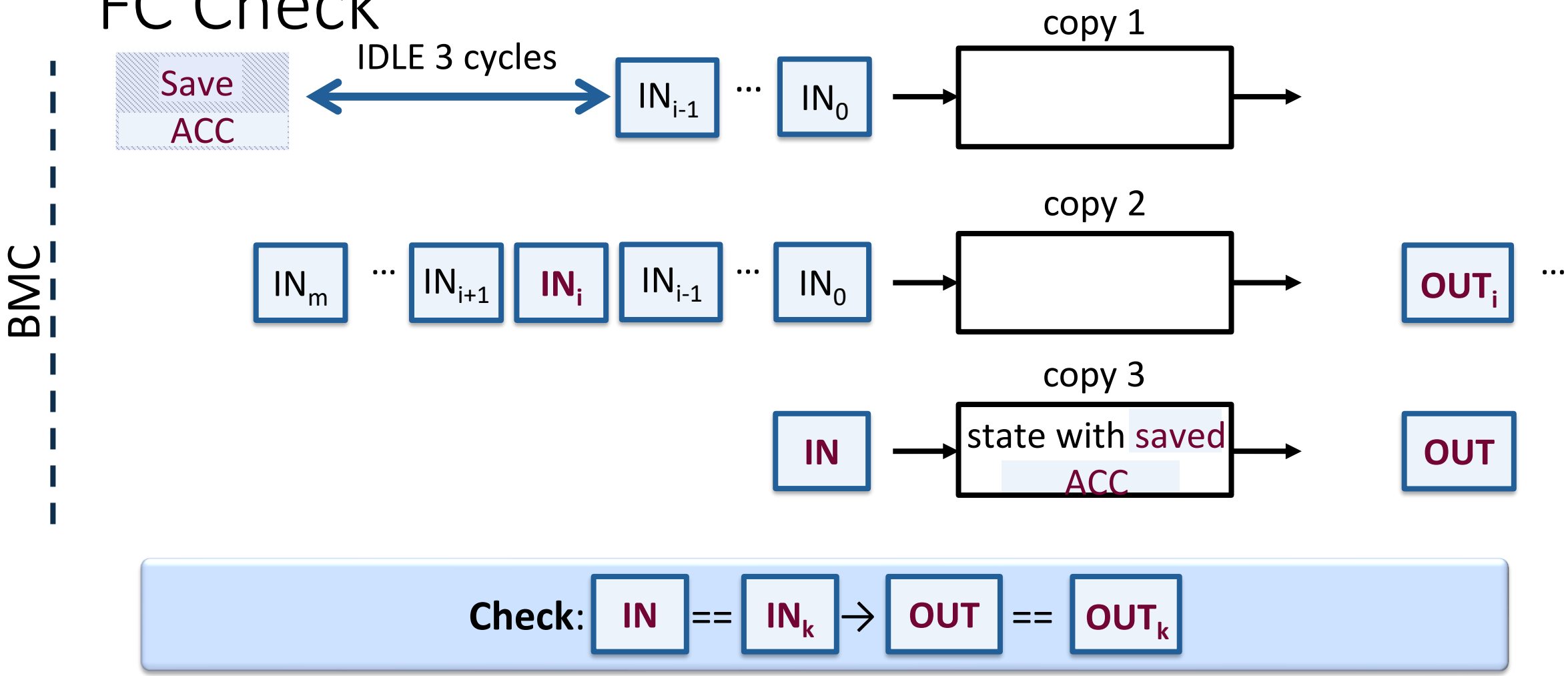
- FPU operations are tedious and difficult to verify using simulation
- Questa FPU solution has the building blocks to construct simple readable properties
- FPU App reduces verification from months down to days
- Provers and disprovers performance enables bug finding in minutes
- Full proof is possible on restricted cases
- Bounded proof is available for all cases
- FPU App provides comprehensive verification
  - Fully compliant external reference model increases verification confidence

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

G-QED DEMO

accellera SYSTEMS INITIATIVE

FC Check


BMC
Save
ACC
IDLE 3 cycles
IN_{i-1}
...
IN_0
copy 1
IN_m
...
IN_{i+1}
IN_i
IN_{i-1}
...
IN_0
copy 2
OUT_i
...
IN
copy 3
state with saved
ACC
OUT
Check: IN == IN_k → OUT == OUT_k

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE 10 YEAR ANNIVERSARY

Questions

accellera SYSTEMS INITIATIVE