Introduction to Digital Design Verification

What is Design Verification?

Verification complexities

What is Design Verification?

What is Design Verification?

"Design Verification is the process used to gain confidence in the correctness of a design w.r.t. the requirements and specification."

Types of verification:

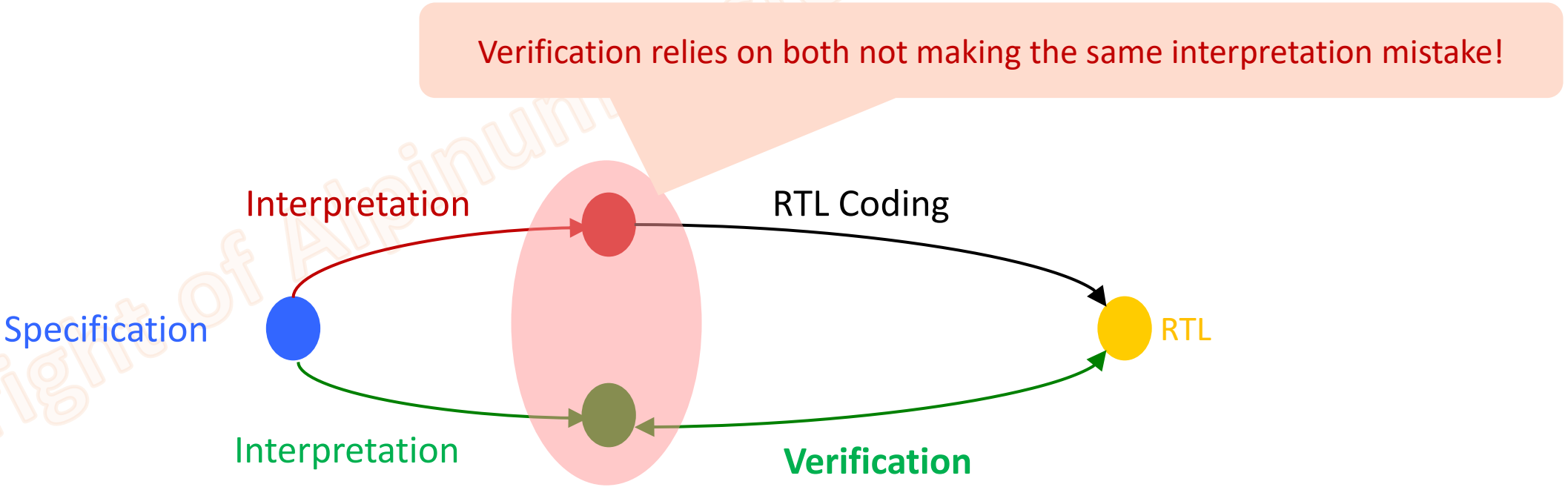
- Functional verification
- Performance verification
- Timing verification
- Physical verification

Verification vs. Validation

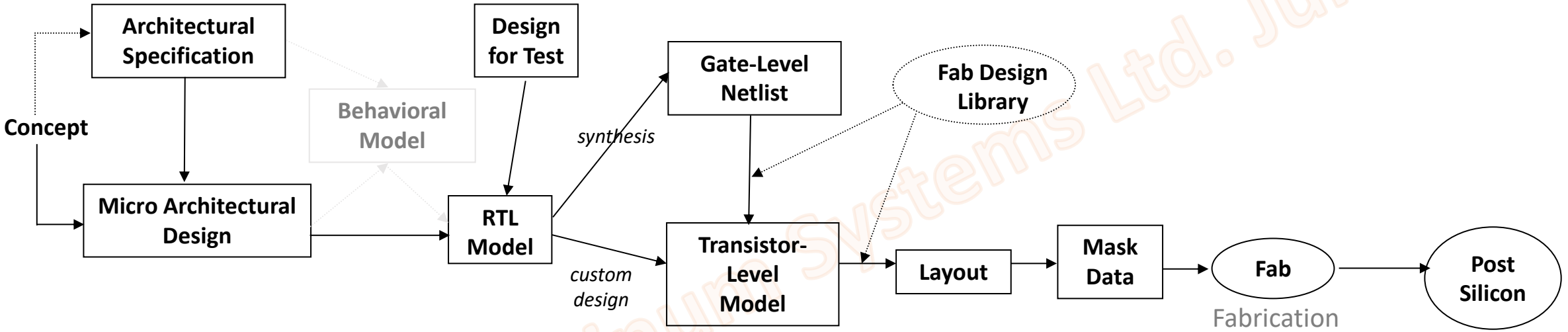
- Verification:
  - Gain confidence in the correctness of a design w.r.t. the requirements and specification.
- Validation:
  - Have we got the correct requirements and specification according to customer needs/desires.
- Building the right thing (validation) vs. building the thing right (verification)
- Note for some companies (e.g. Intel, Arm), validation means “verification”
- Other meanings of “validation”:
  - Confirms that the physical realisation (e.g. silicon) of the design meets the physical (e.g. electrical parameters) required (“*in the lab*”)
  - Confirms that the design functions correctly when deployed in its target environment, e.g. as a component of an embedded system (“*in the field*”)

Introduction: Verification Independence

- There should be verification independent from design
  - Verification engineers refer to the specification in disputes with the design team
  - Designers and Verification engineers both interpret the specification



Verification in Context: The IC Design Process



Fabrication = Physical Realisation

- Manufacturing test
- Bench analysis
- Qualification
- Post-Silicon Validation
- Product Validation

(Specification) Validation

Functional Verification

Equivalence Checking

Physical Verification

Timing Analysis

Performance Verification, Safety verification

Security Verification, Power analysis

Non-functional verification

Functional verification is only a subset of the verification task

Performance verification

- Must also achieve a required level of performance
  - Required throughput normally measured by running an agreed benchmark
- Incorrect performance can point to functional errors
  - A lot of functionality is added to improve performance
    eg: Out of order completion, caches, pre-fetching, pipelining & branch prediction
- Quality of service (QoS) are performance guarantees
  - Real time CPU required to guaranteed response time for interrupts
  - A GPU may be required to guarantee a minimum frame rate
  - An interconnect may be required to guarantee bandwidth and latency
- How can we verify these requirements?
  - Comparison to a reference model eg: check total run time compared to predicted
  - Assertions and monitors eg: counting number of cycles latency of transactions
  - Formal properties can find shortest trace that covers a sequence of events i.e.: best case performance

Security verification

- Ensure secure data cannot leak to insecure areas
- Ensure no insecure access to secure data

Safety verification

- Big impact on the design and verification process
  - Well-defined and documented requirements
  - Traceability to ensure all requirements implemented
    - and NO other code/functions in the design
- Verify that the design can detect intermittent errors

Power verification

- Verification of power saving features
  - E.g. power islands, clock gating
- Power aware simulations to measure expected power dissipations

Do some of these involve functional verification?
For example - verifying the “features” that support security might be considered as functional verification

Why is Verification important?

Verification is a major influence over 3 key project metrics:

- **Increase Quality (Effective Verification)**
  - Improved verification to find more bug
  - And create “right-first-time” products
  - Note quality is more than just verification
- **Decrease Cost (Efficient Verification)**
  - Verification absorbs most resources – improving efficiency reduces costs
  - Re-spins costs millions of dollars
  - Potential opportunity costs.
- **Shorter Timing/Schedule (Efficient Verification)**
  - Verification absorbs most resources – efficiency reduces costs time-to-market.
  - Re-spins take months
  - Late to market means lost opportunity

Siemens Bi-Annual Survey (ASIC) here

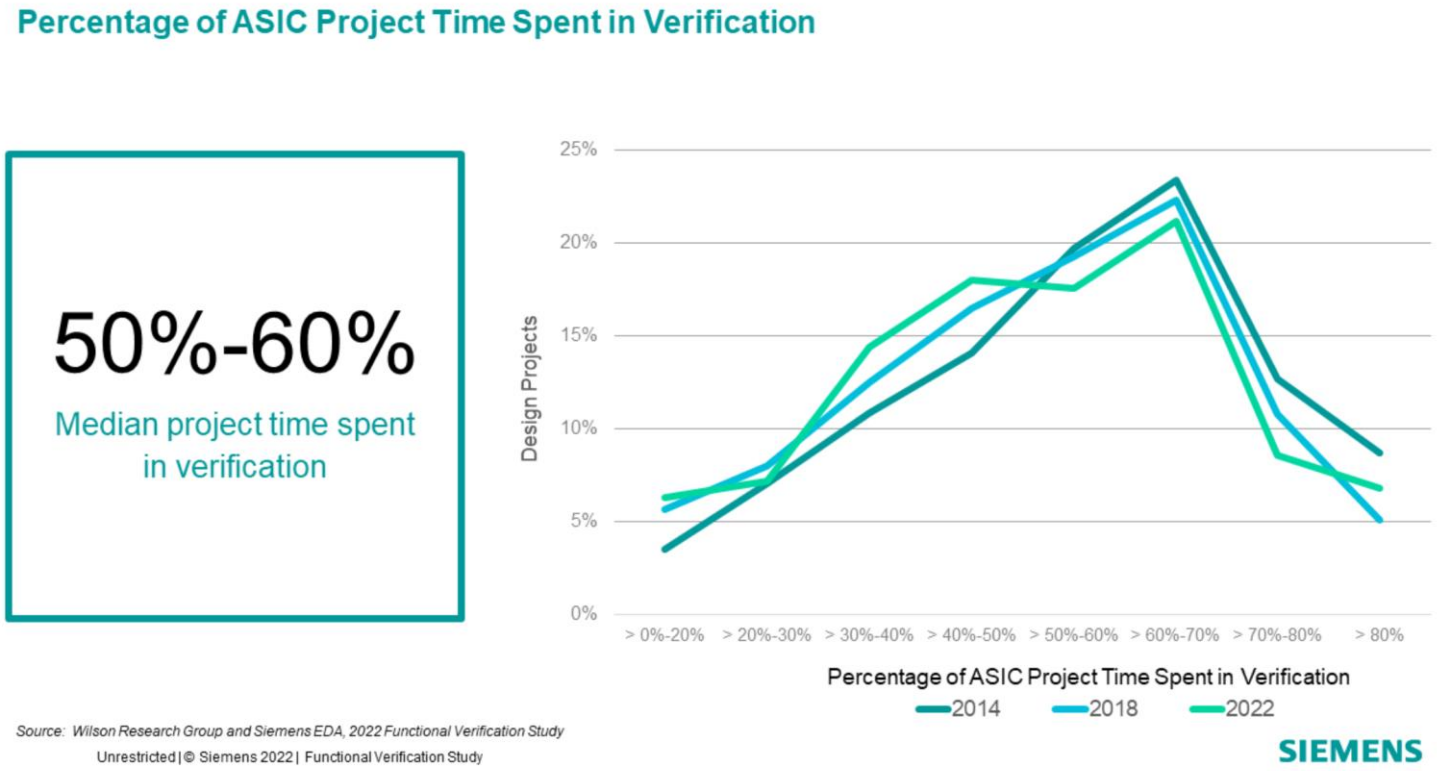


Figure 8-1. Percentage of IC/ASIC Project Time Spent in Verification

Verification Complexities

First design: a 4-bit asynchronous counter (Verilog)

- Clock and reset input
- Four 1-bit state bits
- 4-bit output

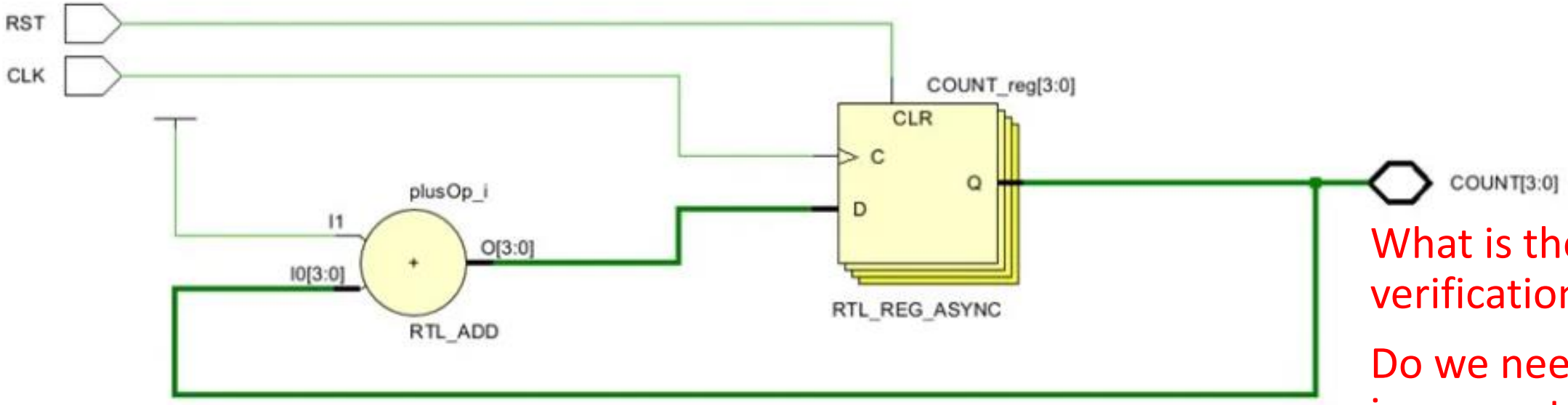
Example design and test bench in Verilog

```
module up_counter(input clk, reset, output[3:0] counter);
reg [3:0] counter_up;

// up counter
always @(posedge clk or posedge reset)
begin
if(reset)
 counter_up <= 4'd0;
else
 counter_up <= counter_up + 4'd1;
end
assign counter = counter_up;
endmodule
```

```
module upcounter_testbench();
reg clk, reset;
wire [3:0] counter;

up_counter dut(clk, reset, counter);
initial begin
clk=0;
forever #5 clk=~clk;
end
initial begin
reset=1;
#200;
reset=0;
end
endmodule
```

Reset is asynchronous because independent of the clock

First design: a 4-bit asynchronous counter



- Input space
  - CLK
  - RST
- State space
  - 4 bits
  - How many combinations?
- Output space
  - 4 bits
  - How many combinations?

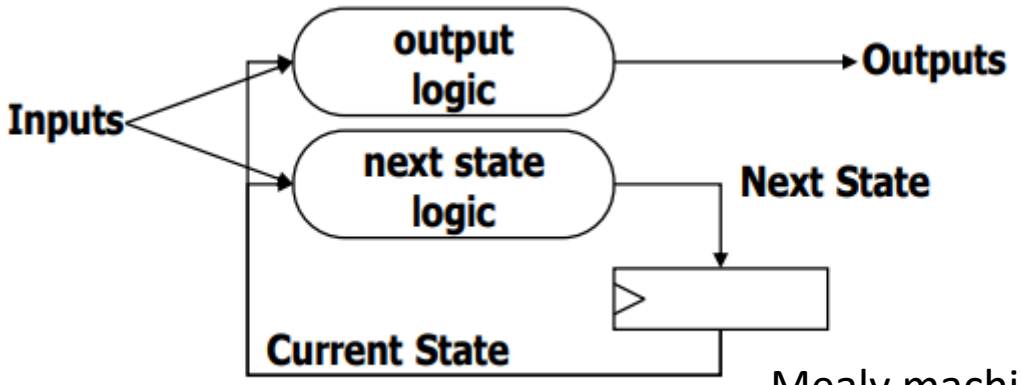
What is the minimum verification plan?

Do we need to reset in every state?

Do we need to vary the clk speed?

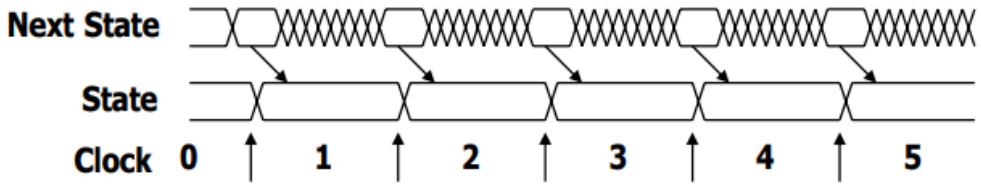
We can represent the design as a state machine

- Values stored in registers represent the state of the circuit
- Combinational logic computes:
  - next state
    - function of current state and inputs
  - outputs
    - function of current state and inputs (Mealy machine)
    - function of current state only (Moore machine)



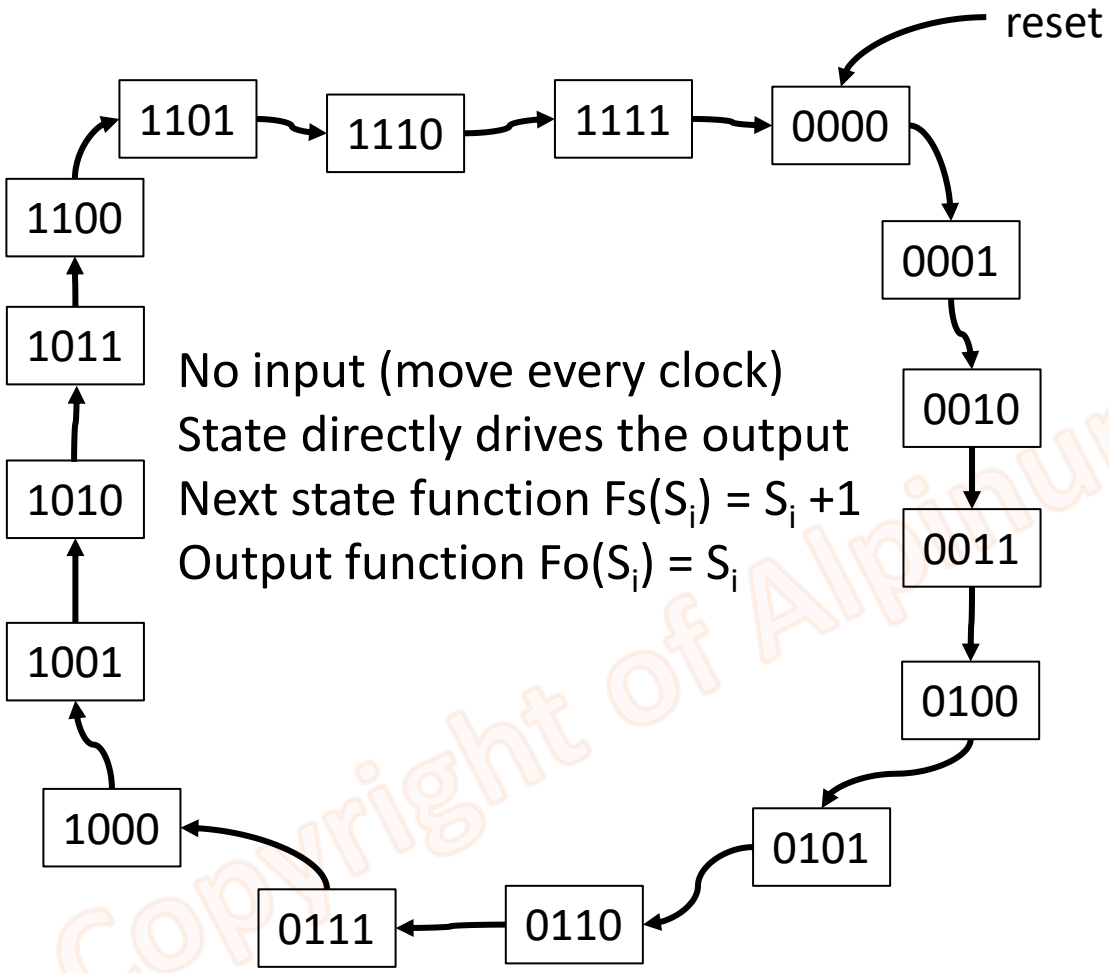
Mealy machine

- States: $S_1, S_2, \ldots, S_k$
- Inputs: $I_1, I_2, \ldots, I_m$
- Outputs: $O_1, O_2, \ldots, O_n$
- Transition function: $Fs(S_i, I_j)$
- Output function:
  - $Fo(S_i)$ Moore
  - or $Fo(S_i, I_j)$ Mealy

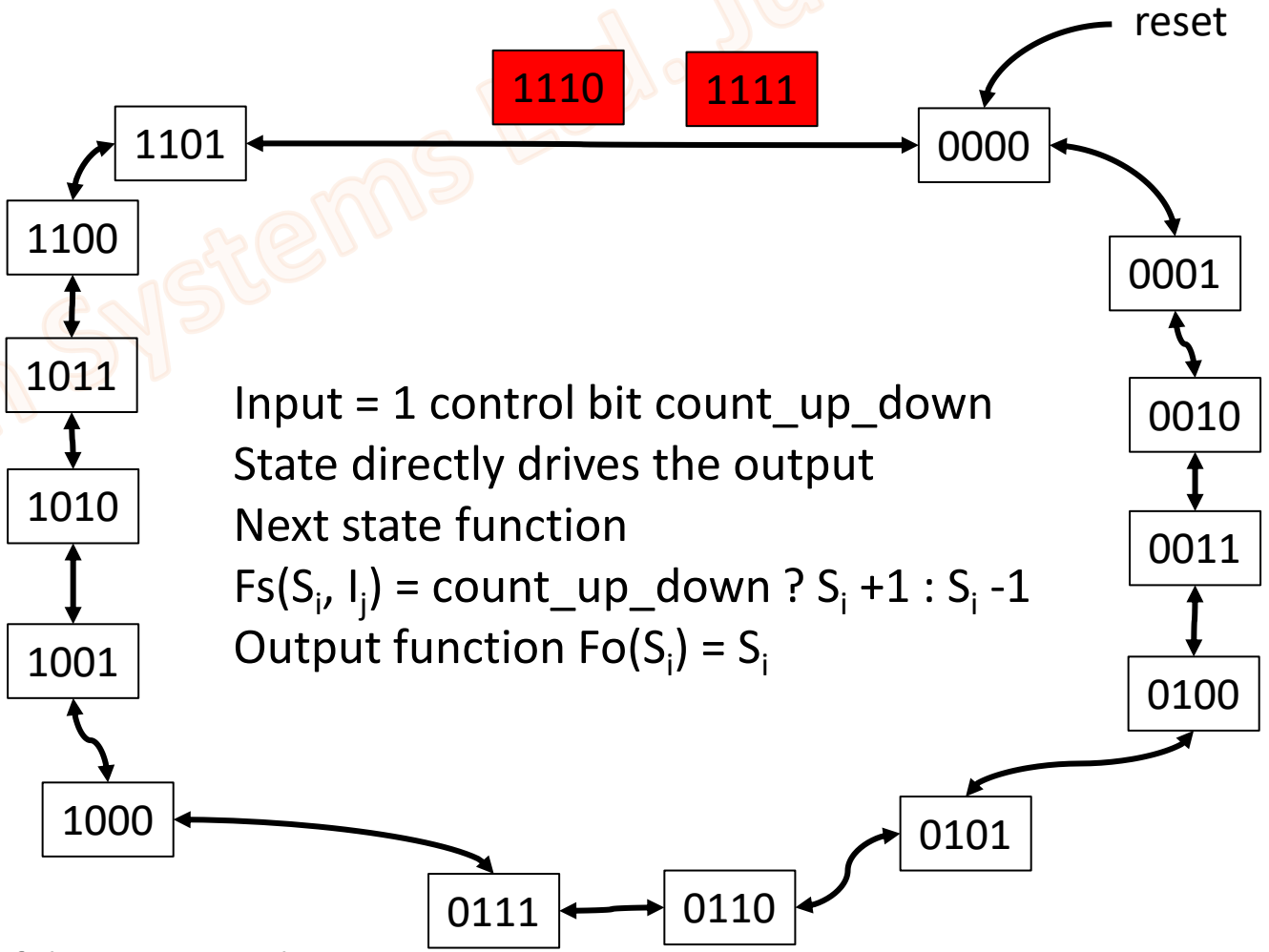


State Machines representations

- 4-bit asynchronous counter



- 4-bit asynchronous up-down modulo 14 counter



State space explosion?

- Consider a 32-bit adder
  - Assuming combinatorial (no internal state)
  - Input space = 2^{64}
  - Output space (consider the carry bit) = 2^{33}
    - ShoOf course, covered if we cover the full input space
  - Assuming unique input on every simulation cycle then
    - Number of cycles to exhaustively verify a 32-bit adder: approx. 18 billion billion
    - Approx. 10 million billion years at 1 simulation per second
      - Of course, simulation runs more quickly, and we can run them in parallel
      - But hopefully you get the point – this is NOT tractable
    - We would also need to check the output on every cycle
- Consider RiscV CPU design
  - Input space = 32-bit instruction
  - Internal architectural State
    - 32 x 32-bit registers
  - Micro-architectural (design related) state
    - For example, a Cache (which can be very large)

Main challenges:

- Input space and architectural state space are too large to cover
  - Not all combinations are possible
  - For a CPU, not all of the 2^{32} input bit combinations (=instructions) are valid
  - For each CPU register, not all the 2^{32} combinations of bit values are valid
  - And even less combinations of all 32 x 32 regsiters
- Most CPU designs have large micro-architectural state too
  - Cache, shadow registers, etc
- The “outputs” are effectively register values and memory
  - so how do we check these values?

Any suggestions for solutions?

Possible solutions:

- Identify the valid input and architectural state bit combinations
  - This can be VERY challenging
- And further identify the ones of main interest
- For checking purposes, check the register and memory contents
  - “Grey box” probes
  - Memory read instructions

We will investigate such solutions during the course

State space exploration example

Possible combination of state bits

Simulation trace

Illegal states

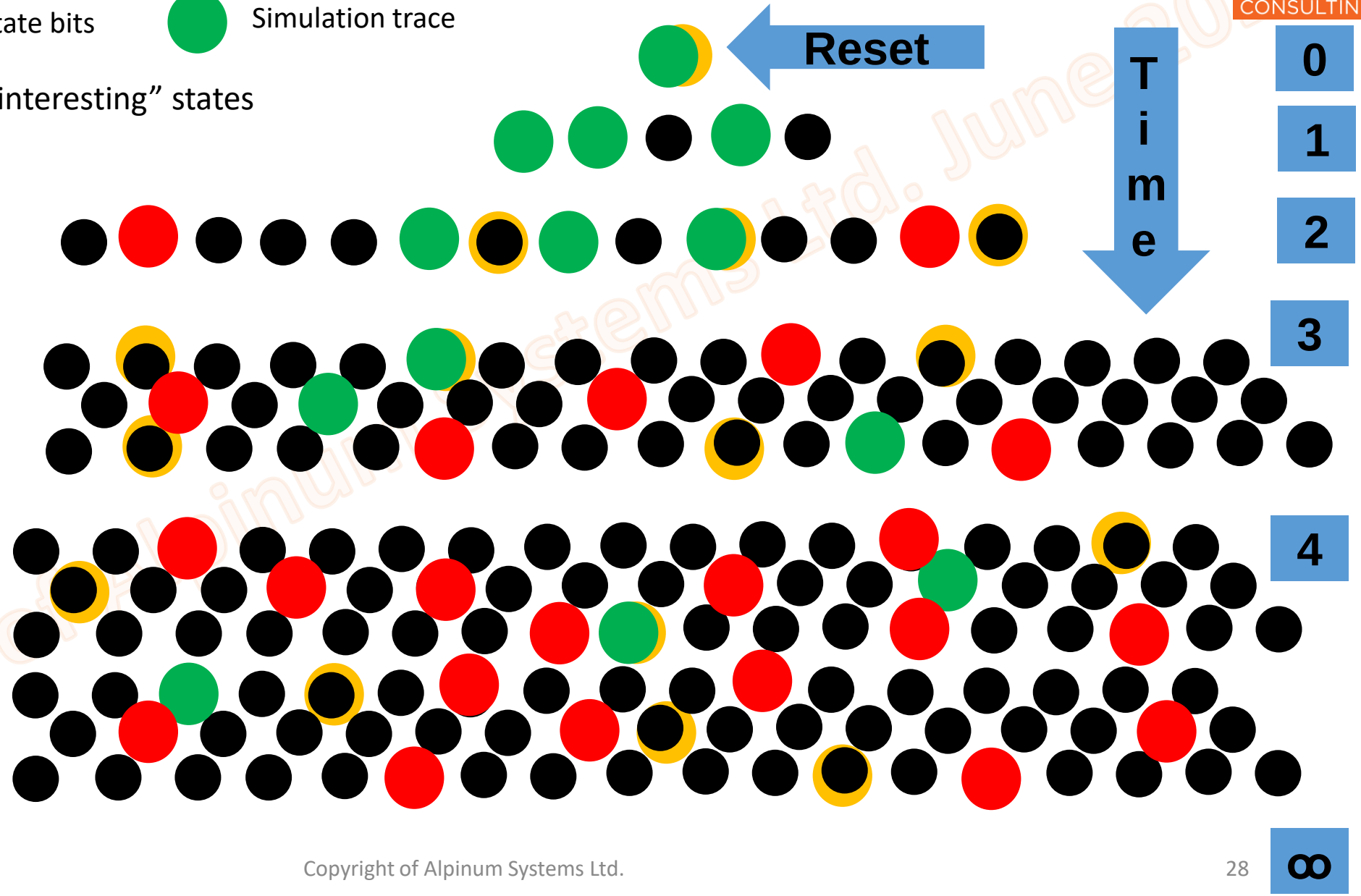
"interesting" states

For a 4 bit, modulo 14 accumulator

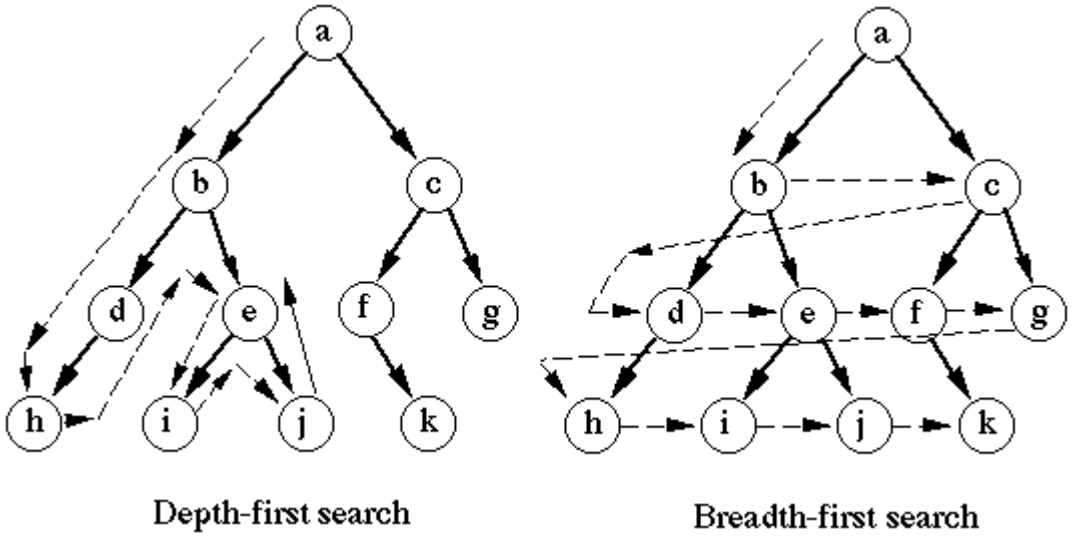
- 16 black cycles {0000,0001,0010, 0011, 0100, 0101,…,1110,1111}
- 2 red circles {1110, 1111}
- Interesting circles? {0000, 1101}

For the RiscV CPU

- There are 32 x 32-bitregister (=2^{1024}) black circles per layer
- Red circles are hard to determine
- Orange circles are hard to determine



Depth-first vs. Breadth-first state space search



This difference in search strategy summarises the difference between simulation and formal verification

- As before the nodes are states in the search space
  - As per the previous slide
- And the arcs are clocked transitions
- These trees are
  - Very wide and Very deep

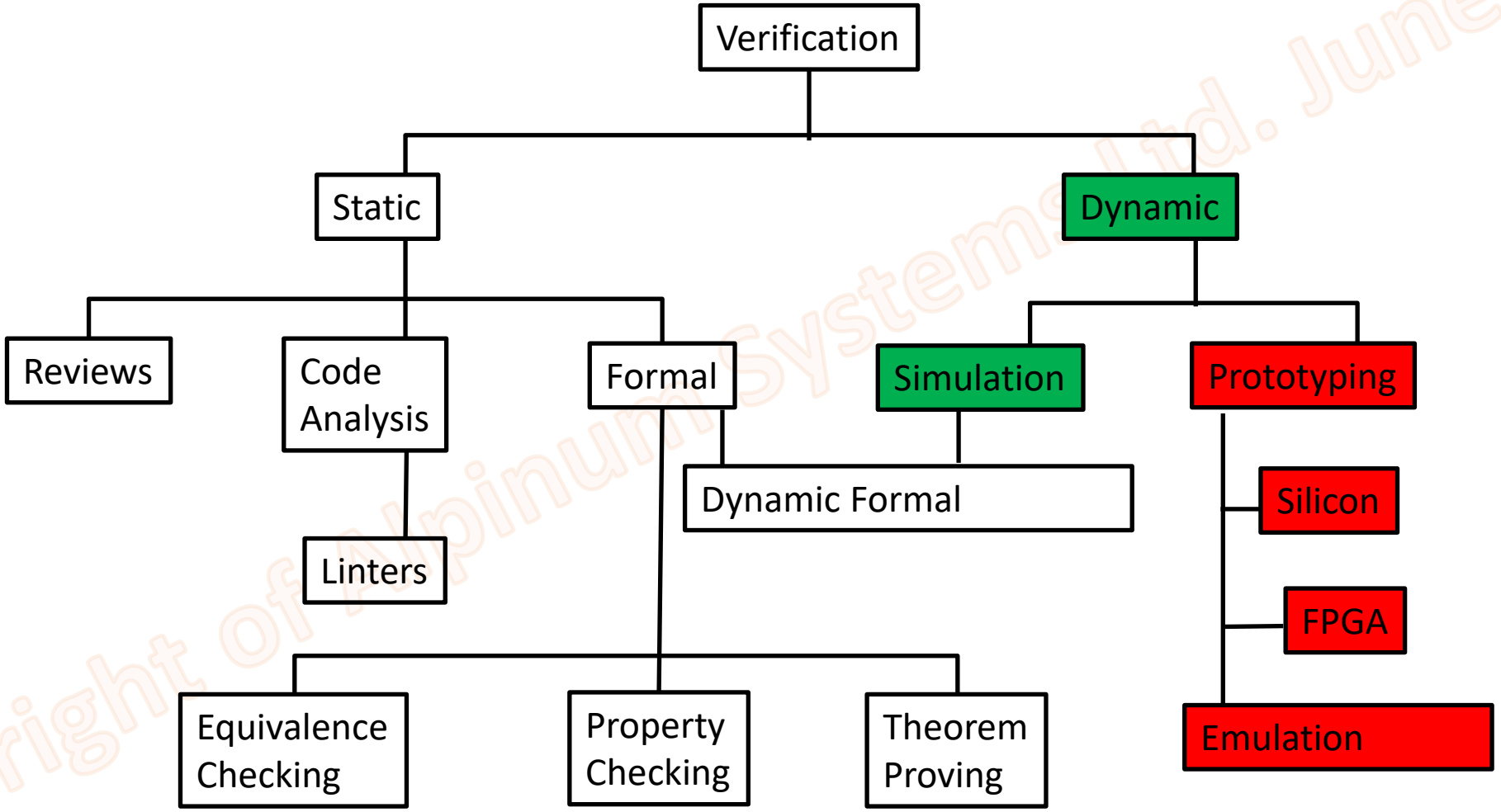
Simulation-based Verification

Driving stimulus and Checking responses

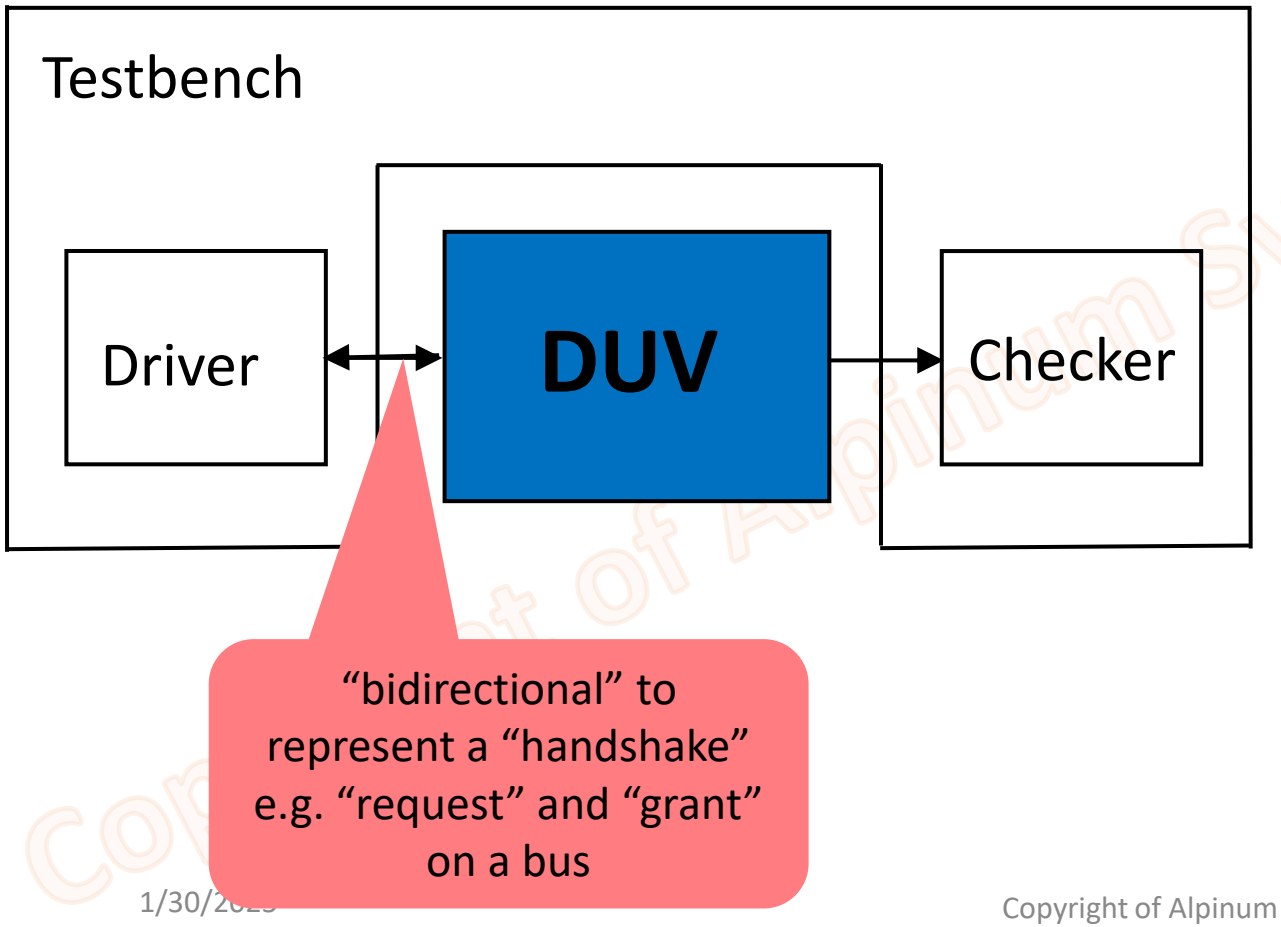
Black box vs. white box

Verification hierarchy

Functional Verification Approaches
Putting simulation-based approaches in context



Dynamic simulation



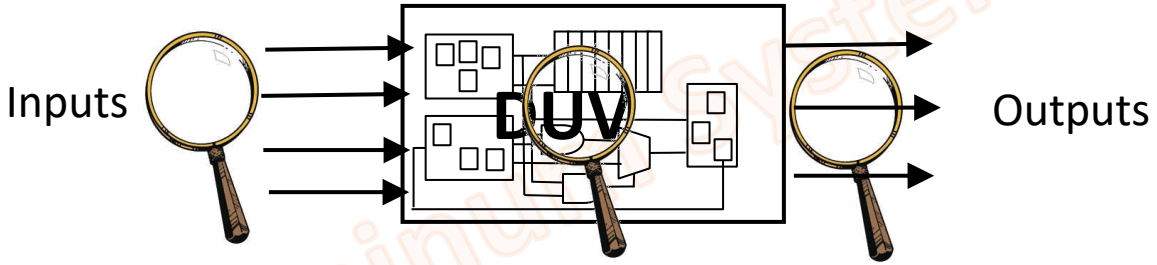
Basic TB architecture

- Drivers
  - Drive stimulus into the DUV
  - Also need to receive and process feedback from DUV
    - For example, protocol handshakes
- Checkers
  - collect the response and check

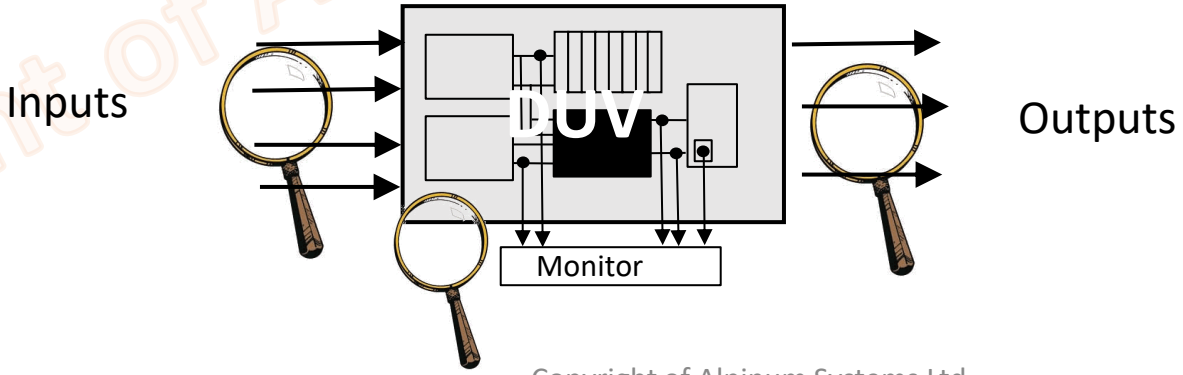
Levels of Observability

- Black Box

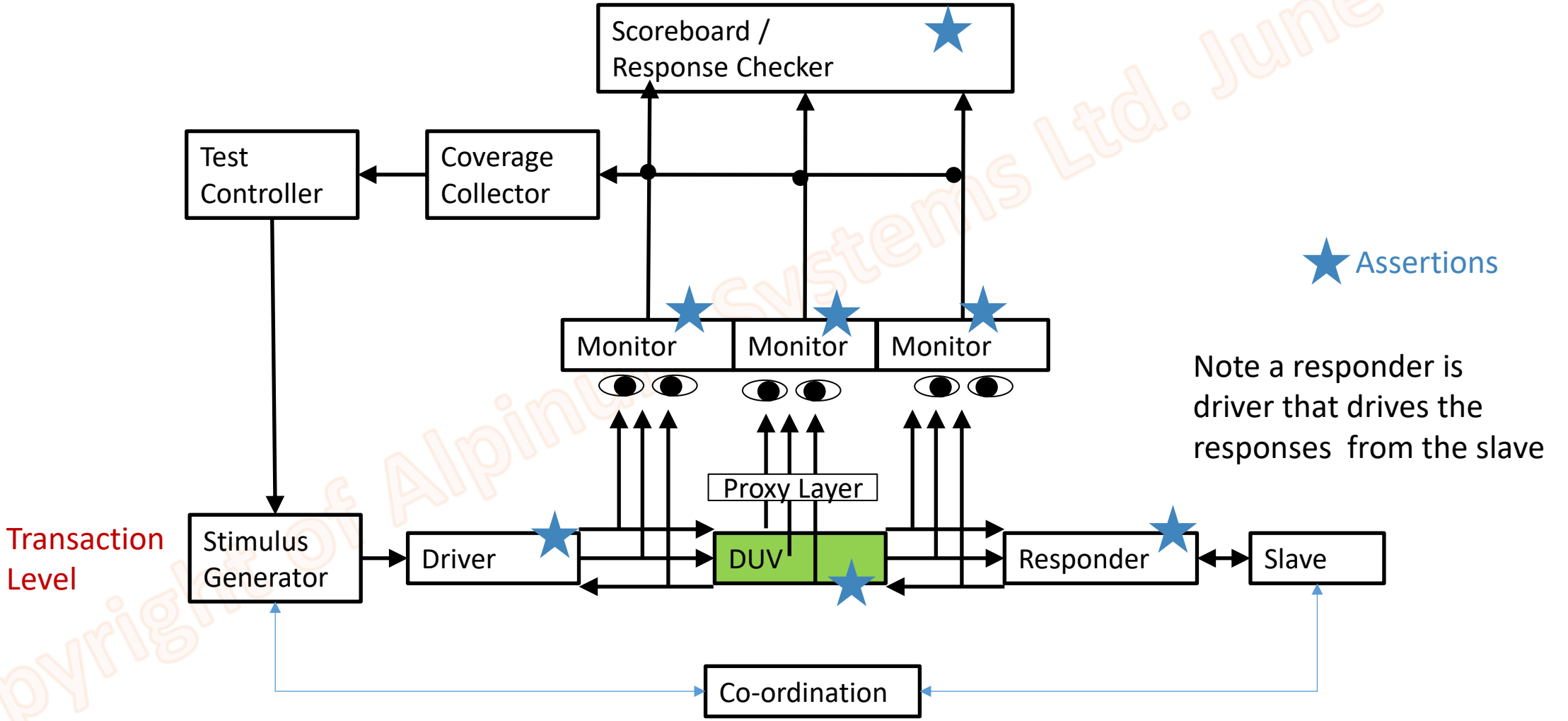
- White Box



- Grey Box



Contemporary Testbench Architecture

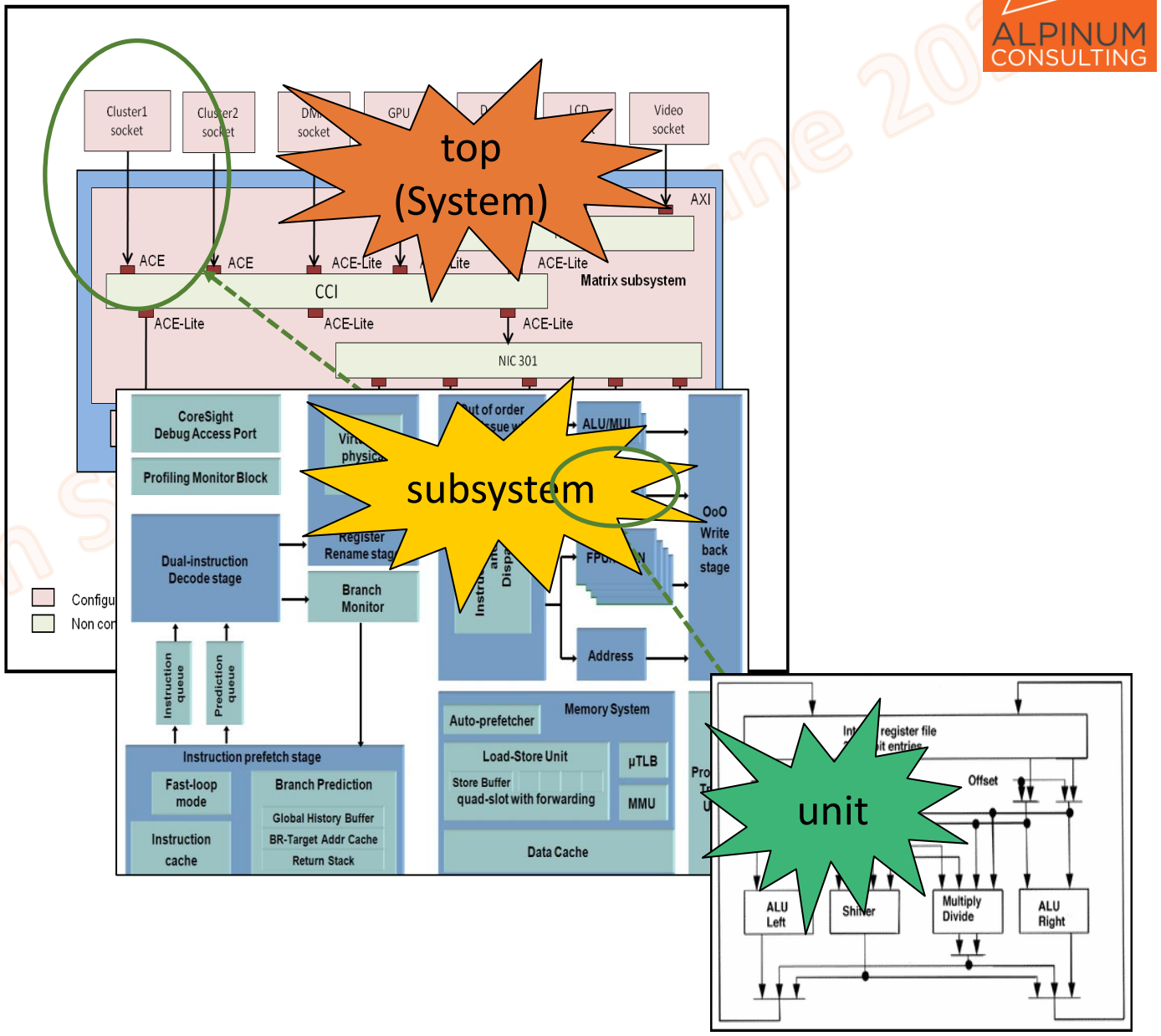


H. Foster: "Response checkers, monitors and asserstions". In Practical Design Verification by Pradhan and Harris (editors). Cambridge, 2009.

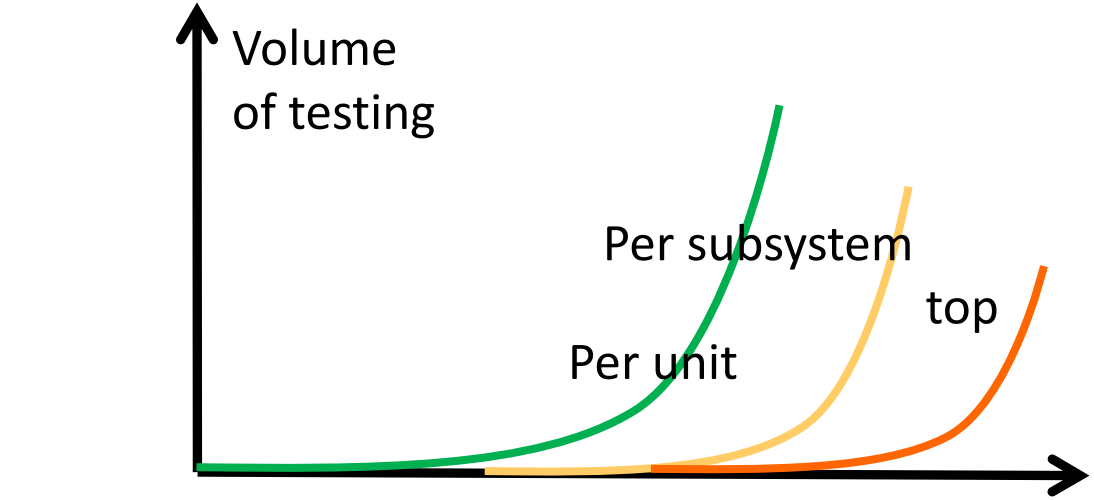
Levels of Verification

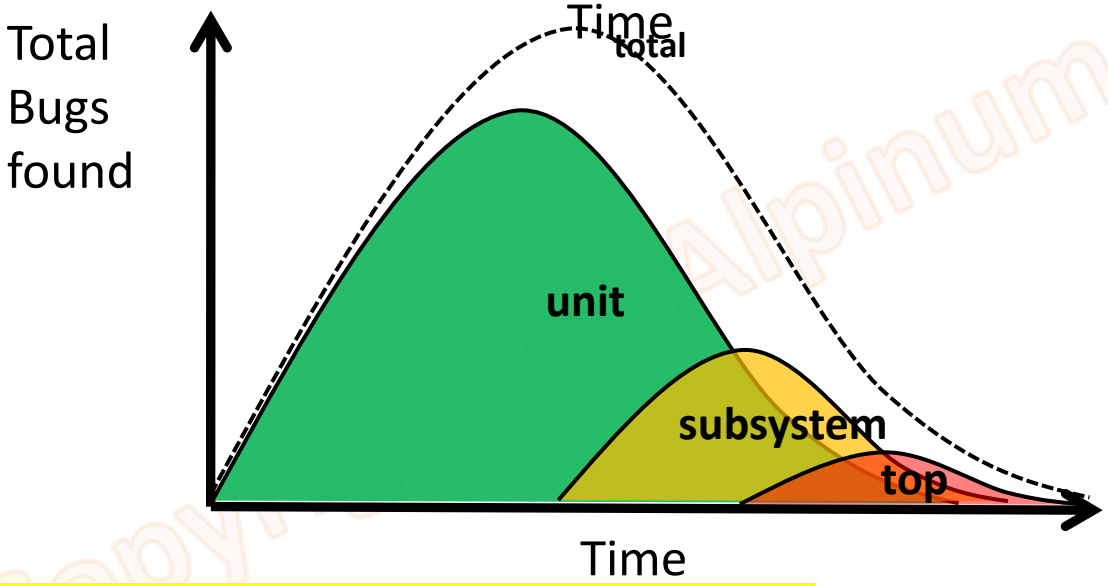
Design Hierarchy

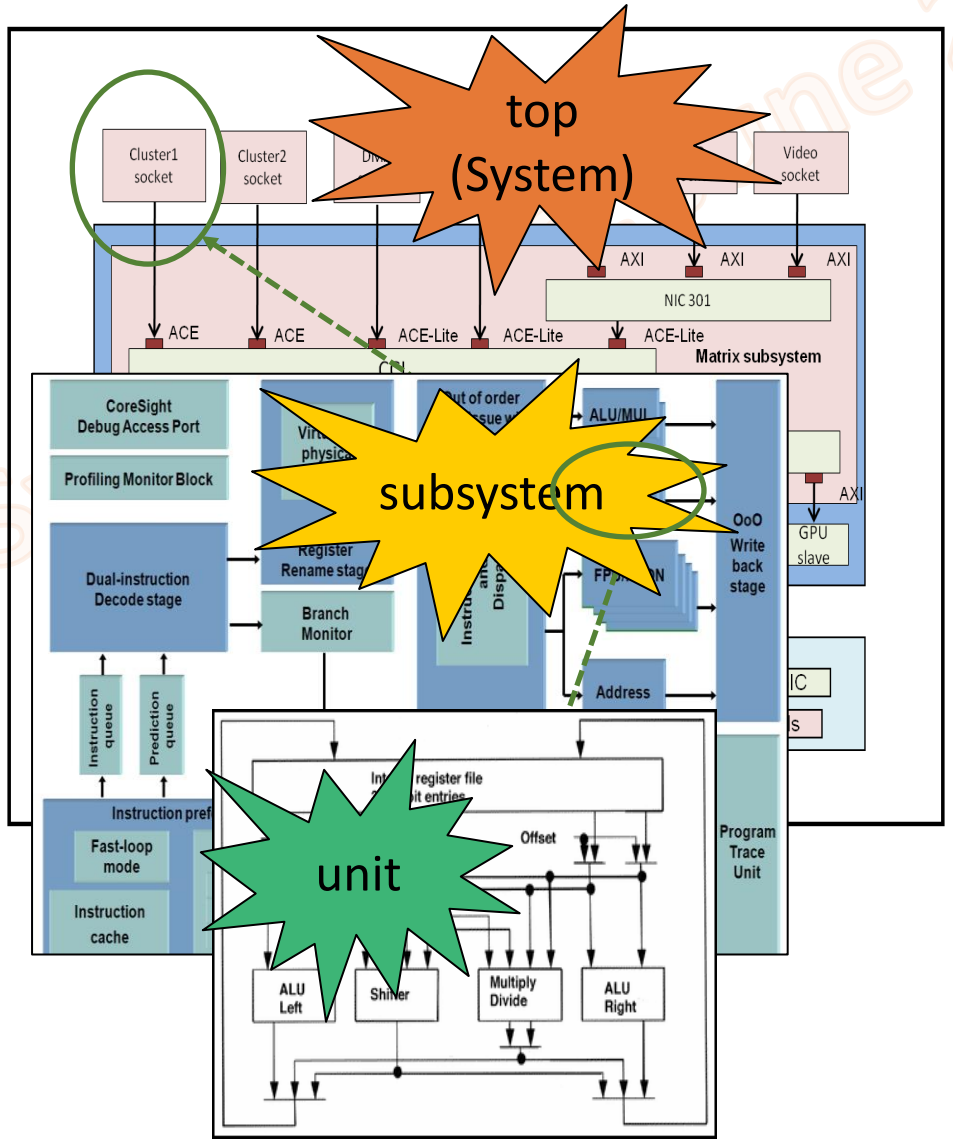
- Chips/systems are divided into separate functional units
  - Usually defined during specification / high-level design
    - *The chip architecture*
  - This practice is called hierarchical design
  - There needs to be a way to connect the functional units
    - This is typically a “bus” infrastructure
      - e.g. AHB or APB, Wishbone



Verification at different Design Levels







Poll https://forms.office.com/r/6YNYwdA8cn

Summary

- What is Design Verification?
  - Clarification vs. validation
  - Many types of verification – we are focused on functional verification
- Verification complexities
  - Increasing design complexity
  - State spaces are too large
    - Are bug free designs possible?

DV Training Courses (online, pre-recorded)

https://alpinumconsulting.com/training/

- Universal Verification Methodology (UVM) Introduction Training
- Advanced Universal Verification Methodology (UVM) Training
- System Verilog Training
- Formal Verification Training
- Design Verification for SV/UVM Training
- Design Verification for VHDL/OSVVM Training
- RISC-V Verification Training