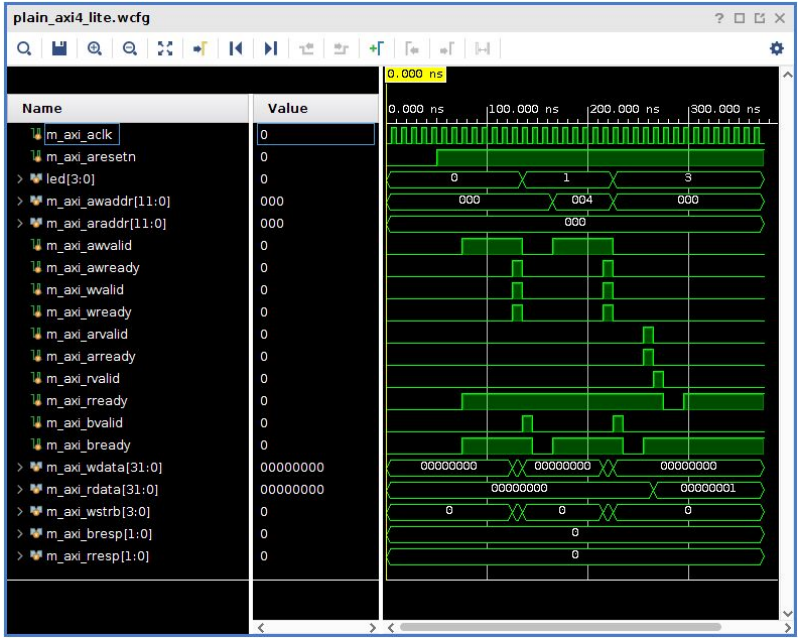
CPE 470 - Simulators

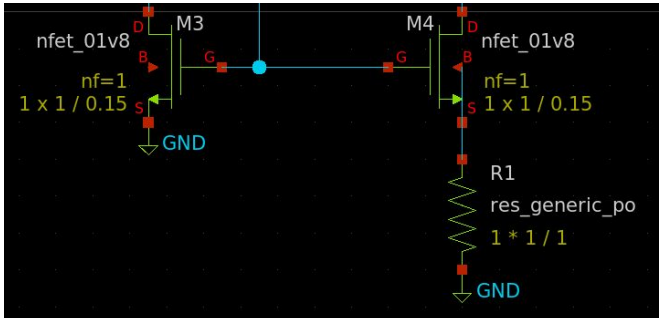
Why Simulation?

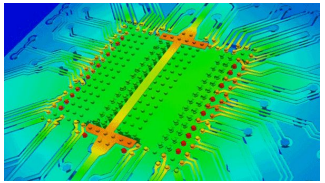
- We will spend most of our design time in simulation
  - Unlike FPGA development, where deployment time is short
- Will simulate across multiple steps of the process
  - Using multiple simulators
- More than just wave forms
  - We use fully separate simulators and waveform viewers
  - Most simulations too complex for human eye analysis
  - Use `assert` statements to do testing programatically



Levels of Simulation

- Behavioral Modeling
  - Build design in higher level language like Python
- **Logic Simulation**
  - HDL-based simulation
  - Using testbenches to test your RTL
- Gate Level Simulation
  - Run testbenches against synthesized and laid-out design
  - Uses gate-level models from PDK
- Transistor-Level Simulation
  - Uses spice tools (ngspice)
  - Uses transistor-level models from PDK
  - Slow
- E&M Simulation
  - Electromagnetic simulation
  - Extremely computationally intensive
  - Often used for wireless and RF chips



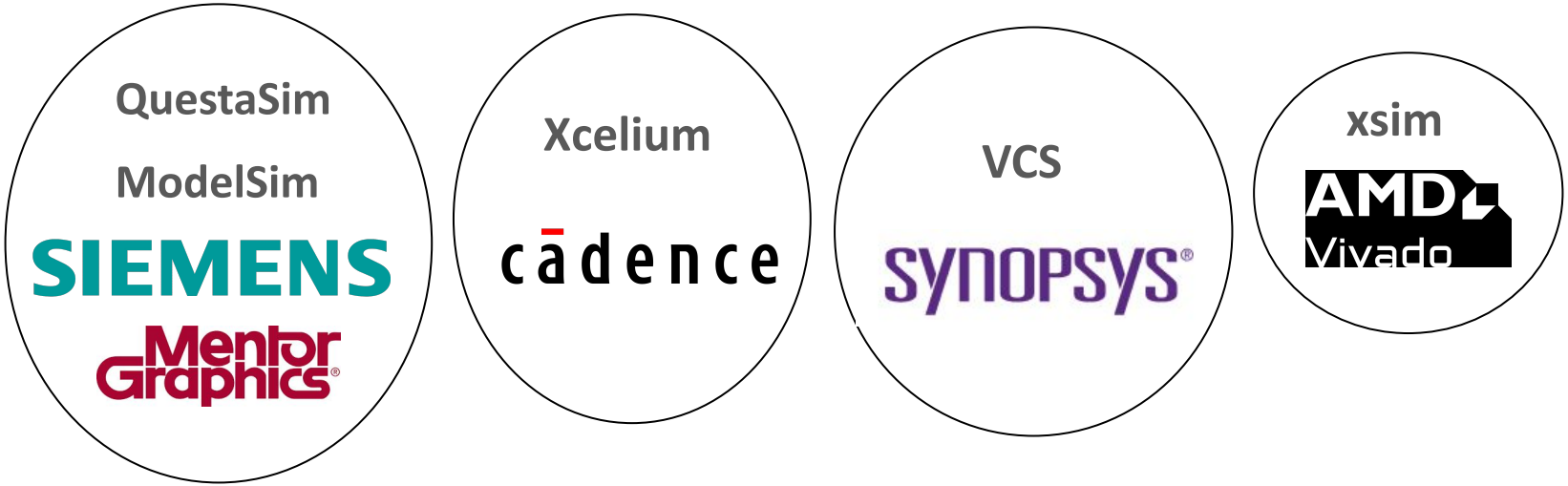


Cycle vs. Event Simulators

	Event Driven	**Cycle Accurate Simulator**
When do transitions occur?	Any Time	Clock Edge
Calculations per Clock Cycle	Many	One
Able to catch timing issues?	Sometimes	No
Simulation Speed	Slow	Fast

Industry Simulators: “The Big 3”

- Industry is dominated by 3 main simulators:
  - QuestaSim originally by Mentor Graphics, bought by Siemens
  - Cadence and Synopsys have competing simulators built into their EDA environments
  - Vivado xsim is used primarily only for FPGA applications

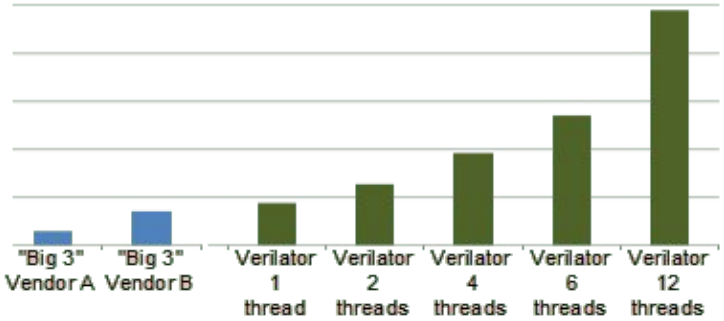


Verilator

Glossary

Linter: a static code analysis tool used to syntax errors

- Doubles as **linter**
  - Tends to be strictest on syntax
  - Enforces syntax such as:
    - All cases having a default
    - <= vs = in different blocks
- Fastest Simulator
  - Multi-threading support
  - Compiled simulator
    - Turns your system verilog into C++ model
    - Compiles into high speed binary



Verilator Tradeoffs

- Binary
  - All signals are 0 or 1
  - **Assumes all unknown signals are 0**
    - Cannot find uninitialized states
  - Does not support **X** or **Z** values for signals
    - Does support Tri-State signals
    - Converts tri-state to two-state
- Cycle Simulator by default
  - Only simulates between clock edges
    - No delays, ie: **#1**
  - Can add **`--timing`** flag to add delay support at the cost of performance
    - This enables IEEE compliant scheduler

Glossary

X: signal is unknown

Z: signal is in High Z or high impedance, meaning it is not actively being driven

```
logic x;
initial begin
    #1
    x = x + 1;
    #1
    $finish();
end
```

Glossary

VPI: Verification Procedural Interface

DPI: Direct Programming Interface

Verilator - Industry Use

- Often used as a secondary simulator
  - Verilator's high speed makes it really good for system-level tests
  - Running firmware

- Used for shipping software models of a device
  - RTL ⟶ C++ ⟶ Binary
  - Provide end users with emulated software model
    - without exposing intellectual property
  - Can interact with external code/drivers
    - **VPI** enables interactions with C code
    - **DPI** enables interactions with other languages

Icarus Verilog

`iverilog`

- Slower, Correctness-focused simulator
- 4-state Simulator
  - 0, 1, X, Z
- Event Driven
  - Intended to be IEEE compliant
  - By default supports delays
- Originally built for only verilog, with system verilog added later
- Developed primarily by one person
  - Stephen Williams

Simulator Differences

- Icarus verilog will catch initialization issues that verilator will not catch
- Verilator can make it easier to past tests
  - False sense of security
  - Need icarus to catch those mistakes
- Make sure to initialize your clocks to 0

```
logic x;
initial begin
    #1
    x = x + 1;
    #1
    $finish();
end
```

```
always begin
    #(CLK_PERIOD/2)
    clk<=~clk;
end
```

VERILATOR

Race Conditions

Glossary

Race Condition: unpredictable outcome due to access of shared resource

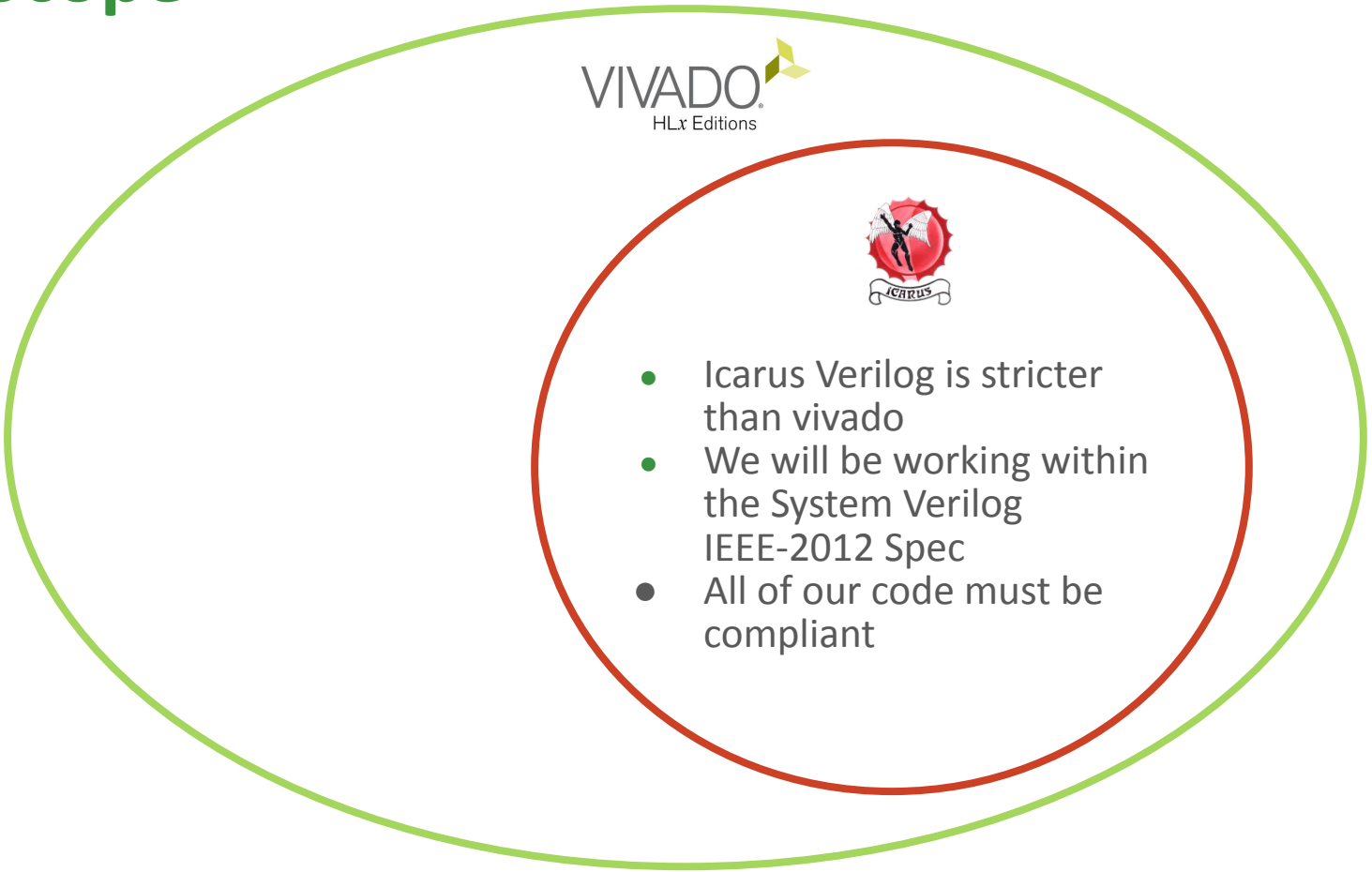
```
byte slam;
bit dunk;
initial begin
  forever begin
    @(posedge clk);
    dunk = ~dunk;
    slam += dunk;
  end
end
always @(posedge clk) basket <= slam + dunk;
```

- **Race Conditions** can occur when using blocking assignment (**=**) on a clock edge:
  - reading the old value or the updated value?
  - In this example: When basket gets calculated, will it use the new or old dunk?
- Always use a non-blocking assignment (**<=**) to drive simulator signals on clock edges
  - Different simulators can handle race conditions differently
  - Using multiple allows catching and fixing these problems.

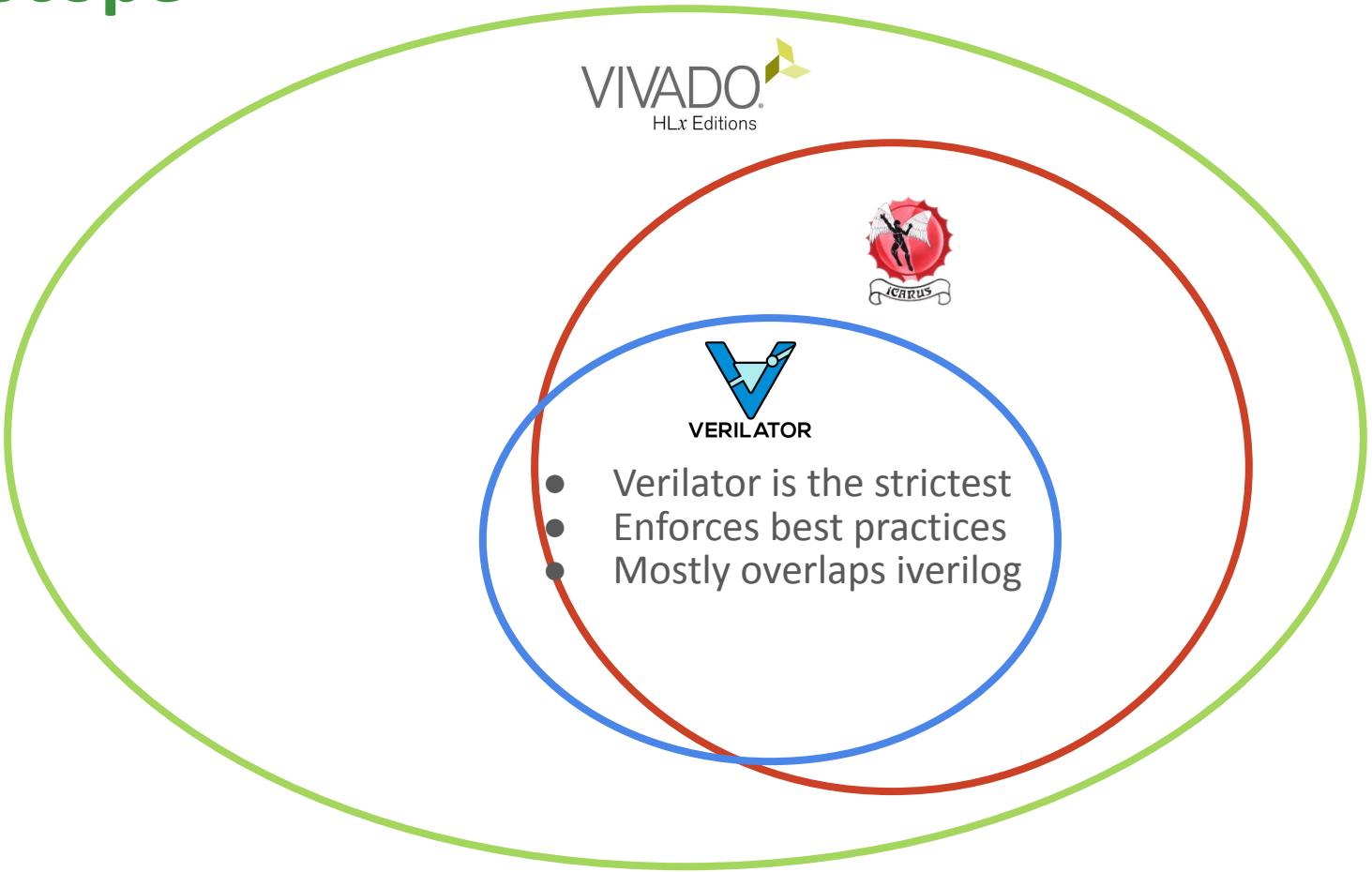
Scope

- We are used to working in the Vivado environment
  - Highly Permissive
  - Does not enforce syntax very strongly
    - <= or = in an initial block? Who cares
    - Incomplete case? Why not
    - Bit widths don’t match? I’ll try my best

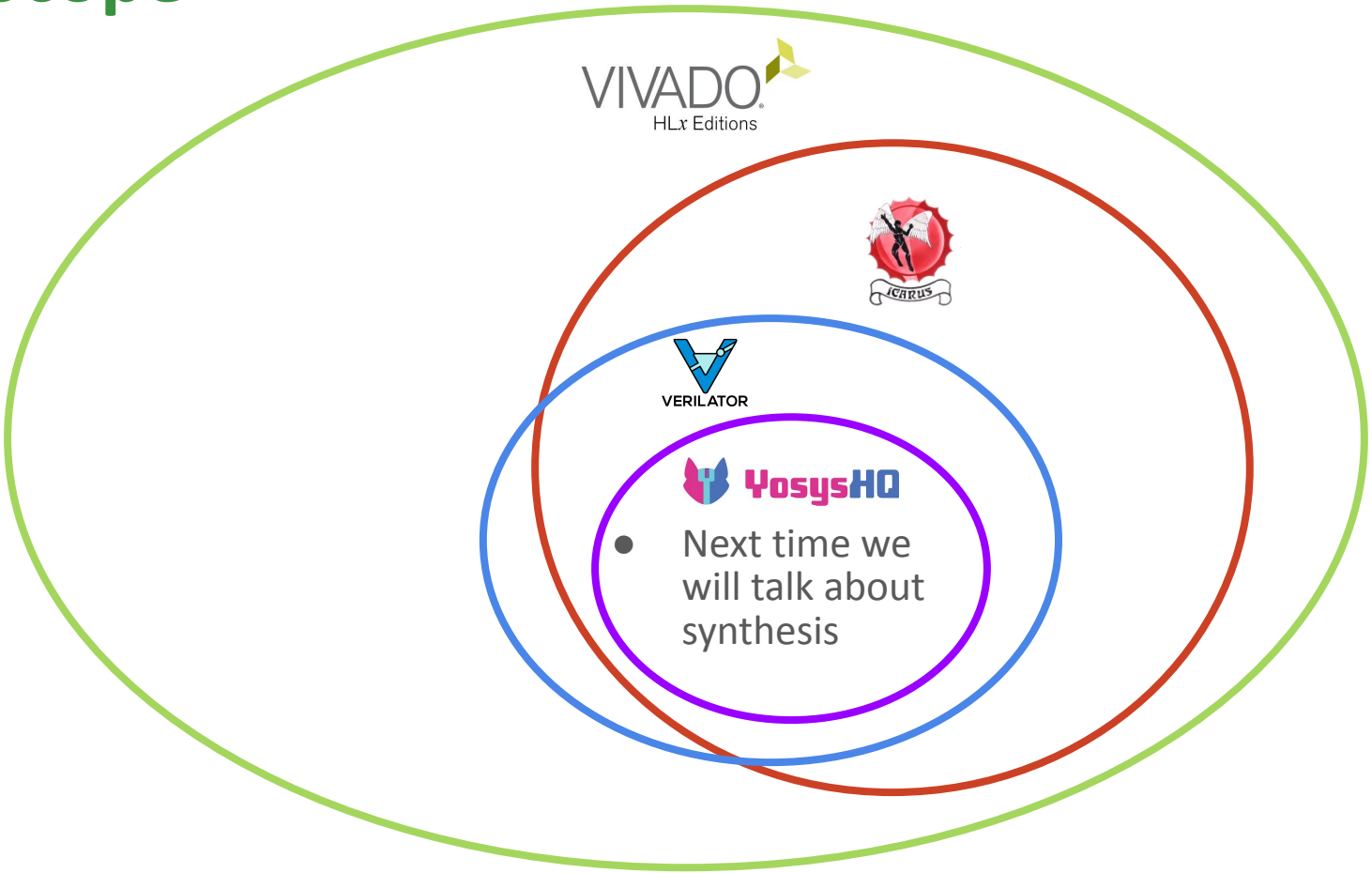
Scope


VIVADO
HLx Editions
ICARUS
Icarus Verilog is stricter than vivado
We will be working within the System Verilog IEEE-2012 Spec
All of our code must be compliant

Scope


VIVADO
HLx Editions
ICARUS
VERILATOR
Verilator is the strictest
Enforces best practices
Mostly overlaps iverilog

Scope


VIVADO
HLx Editions
ICARUS
VERILATOR
YosysHQ
Next time we will talk about synthesis

Compliance

Here are some best practices to help you succeed:

- Files named as `<module name>.sv`, one file per module
- Avoid fancier system verilog features where possible
  - ❌ Interfaces
  - ❌ Modports
  - ❌ Multidimensional inputs/outputs
    - ❌ input [32:0] array [32:0];
- Separate combinational and sequential elements
  - Keep logic in combinational
  - Do next-state assignment in sequential
- Use consistent timescales across modules

```
`timescale <time_unit>/<time_precision>

// Example
`timescale 1ns/1ps
```

References

- https://steveicarus.github.io/iverilog/
- https://www.veripool.org/verilator/
- https://en.wikipedia.org/wiki/Logic_simulation
- https://ieeexplore.ieee.org/document/6469140
- https://blogs.sw.siemens.com/verificationhorizons/2020/08/13/systemverilog-race-condition-challenge-responses/