EGC 455
SOC Design & Verification

# Functional Verification of Hardware

As presented by IBM

**Baback Izadi**
Division of Engineering Programs
bai@engr.newpaltz.edu

1

## Speakers

- IBM-Z Hardware Verification
  - Shaun Uldrikis – Core Verification Co-lead
  - Divya Joshi - Core Verification Co-lead
  - Luke Buschmann – Unit Verification Co-lead

2

## References

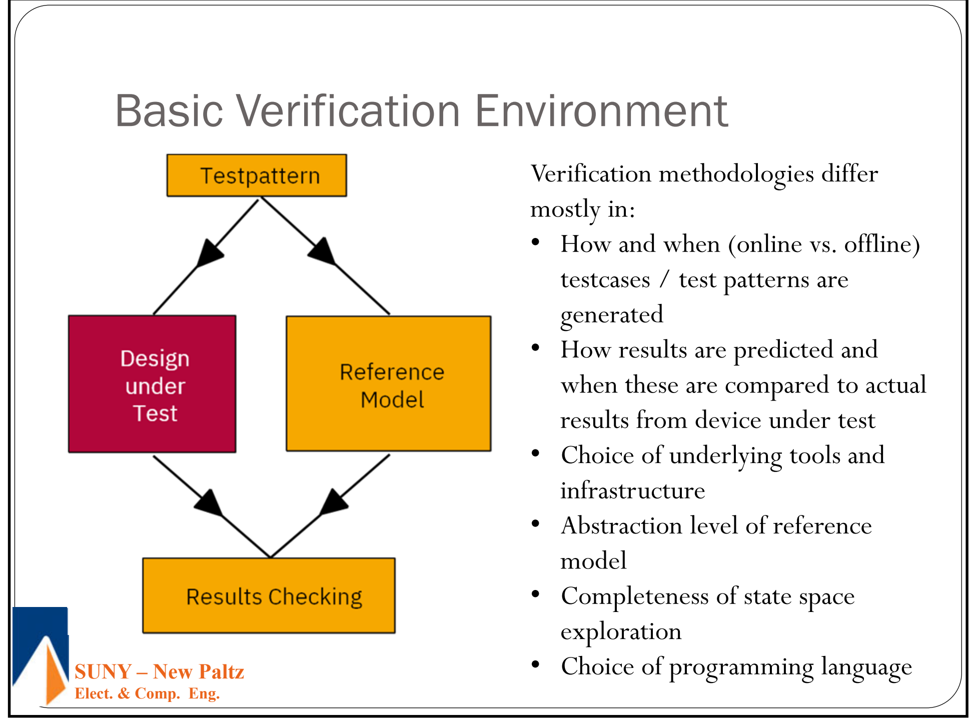
- *Writing Testbenches using SystemVerilog (2006)*
  By Janick Bergeron
- Slide decks
  - John Goss (IBM)
  - Gerrit Koch (IBM)
  - Bruce Wile (IBM)

3

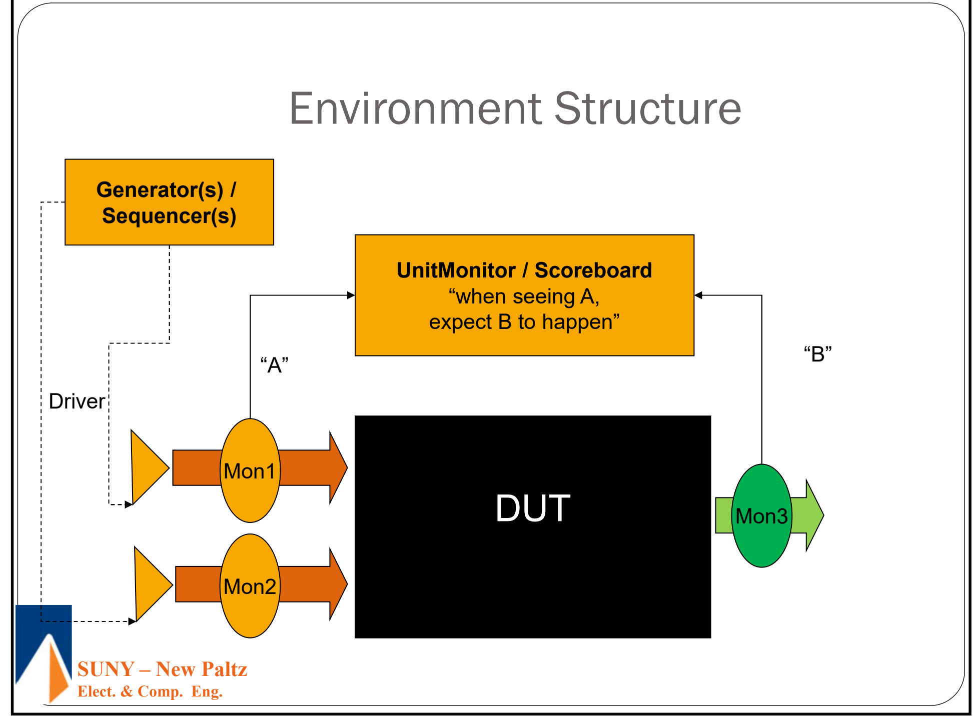
## Day 2

- Testbench options
  - Structure of an environment
  - Constrained random, directed
  - White box, black box, grey box
- Simulation & Regression
- Verification Planning
  - Test Plan
  - What needs to be checked?
  - What metrics indicate you are done?
  - Coverage
  - Who else will use the testbench? (re-use)

4




5




6

## Environment Building Blocks 1

- **Driver** - Low level objects which cover the requirements for accurately exercising the signal(s)/bus. This should adhere to the specification bus protocol rules, unless it needs to support error injection features.
- **Generator** - Creates valid transactions, or sequences of transactions, to be delivered to Drivers for handling.
  - ❖Testbench Generator - Create all testcase operations at time zero, based on bus and DUT specification rules/goals.

    OR
  - ❖Constrained Random Generator - Create transactions on the fly, each cycle, using constraints to build varying, but valid, bus traffic, typically incorporating feedback from the simulation.

7

## Generators (Sequencers)

**Testbench Generator**

- Used if you can fully model the function of the DUT
  - Depending on implementation, it must fully model the starting and ending state of the DUT.
- Useful for targeting unique architectural features, which may not be hit by a more random environment.

**Constrained Random Generator**

- Create interesting transactions across the full range of possible operations using user selected criteria.
- Not required to model the timing of the HW implementation
- Allows the test bench to react to stimulus coming out of the DUT as needed, or to influence future operations.

8

## Generators (Sequencers) cont.

**Directed Testcases**

- Specially designed to reach a specific state in the DUT
- Created when you know of scenarios you must hit, and neither the testbench generator or random generator are likely to hit the case naturally (or often enough), with existing bias controls.

- Examples:
  - Filling up a buffer which is only filled if X writes are driven close together.
  - Forcing address re-use between sequences of writes and reads to validate cache coherency rules.

9

## Environment Building Blocks 2

**Monitor** - A UVM monitor is responsible for capturing signal activity from the design interface and translate it into transaction level data objects that can be sent to other components. (Source: https://www.chipverify.com/uvm/uvm-monitor)

- It should implement low level protocol checks only.
  - Is the combination of signals valid for this cycle?
  - Are the signals functionally valid based previous traffic?
  - Is Parity or ECC correct?
- Higher level checks should be implemented by other objects
- Should monitor signals driven by a Driver, as well as outputs from the design. (source all information from HW signals)
- Each object typically watches signals in only 1 direction

10

## Environment Building Blocks 3

**Unit Monitor / Scoreboard** - Collects transactions from monitors and executes checks to validate function.

- Terms used for the object(s) tracking of operations. Could be used for prediction of results, or looking up previous input transactions to validate an output/response activity (when you can't predict the order or timing of the output)
- Number, and content, of scoreboards depends on how you divide up the checkers.
  - Are some checks independent from others?
  - Can those checks be executed without duplicating work between objects?

11

## Repeatable Environments

- Your environment should utilize a single random seed which all future random numbers are based on.
- This provides a repeatable environment and testcase configuration.
  - All simulations using the same seed should run 100% the same.
- This feature is supported by the simulator.

- But how you use the random numbers may limit your options for validating a design change.

12

## Random Seed example

- You discover a defect, and the designer makes a fix by adding 1 cycle of delay to that operation.
- Your generator environment is calling a random number object every cycle, whether it uses it or not.
- Your testcase now runs differently due to the HW behaving differently by 1 cycle.
- Goal: Validate fixes by re-running the same exact testcase stimulus.
- Therefore, where and how you use random numbers does matter.

13

## Functional verification approaches

Now that you've seen the basic structure….

- Black-Box approach
- White-Box approach
- Grey-Box approach

14

## Black-Box

- The black box has inputs, outputs, and performs some function.
- The function may be well documented…or not.
- To verify a black box, you need to understand the function and be able to predict/allow the outputs based on the inputs.
- The black box can be a full system, a chip, a unit of a chip, or a single macro.

SUNY – New Paltz
Elect. & Comp. Eng.

15

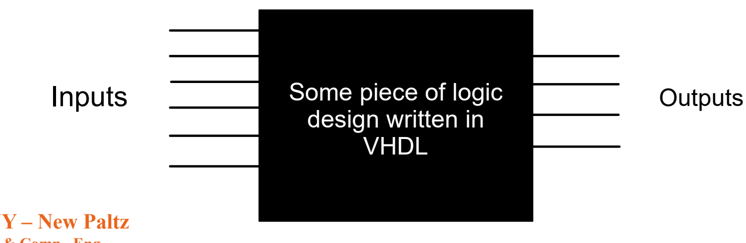
## Black-Box

Pros

- Env written based only on specs. (satisfies reconvergence model)
- Allows for simulation without the verification environment being biased by knowing details of the implementation.

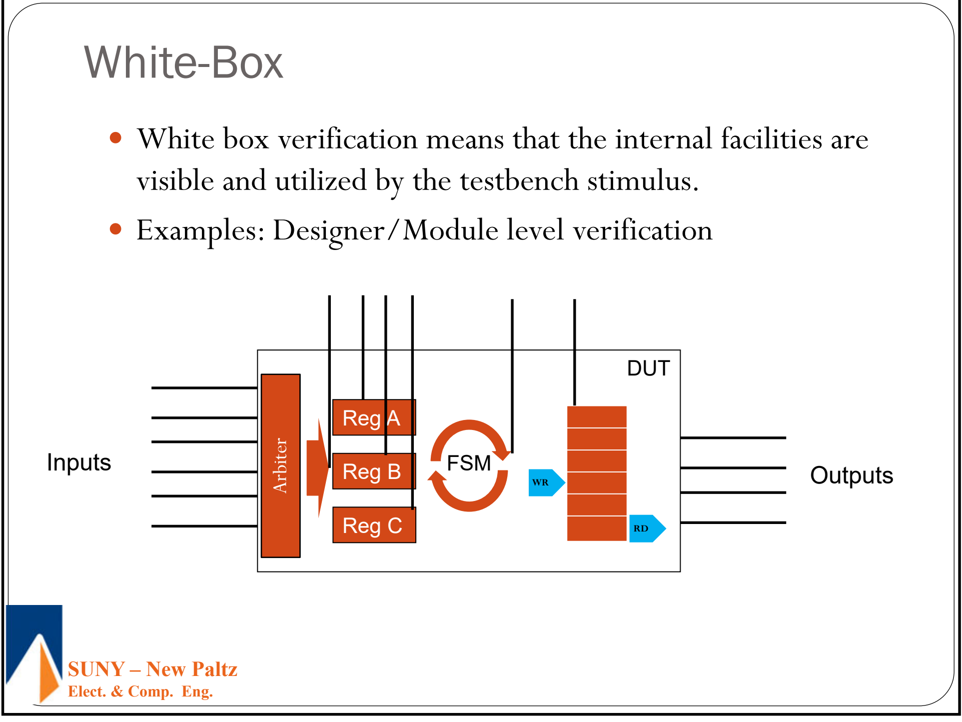
Cons

- May not be able to accurately check function without more information




SUNY – New Paltz
Elect. & Comp. Eng.

16




17

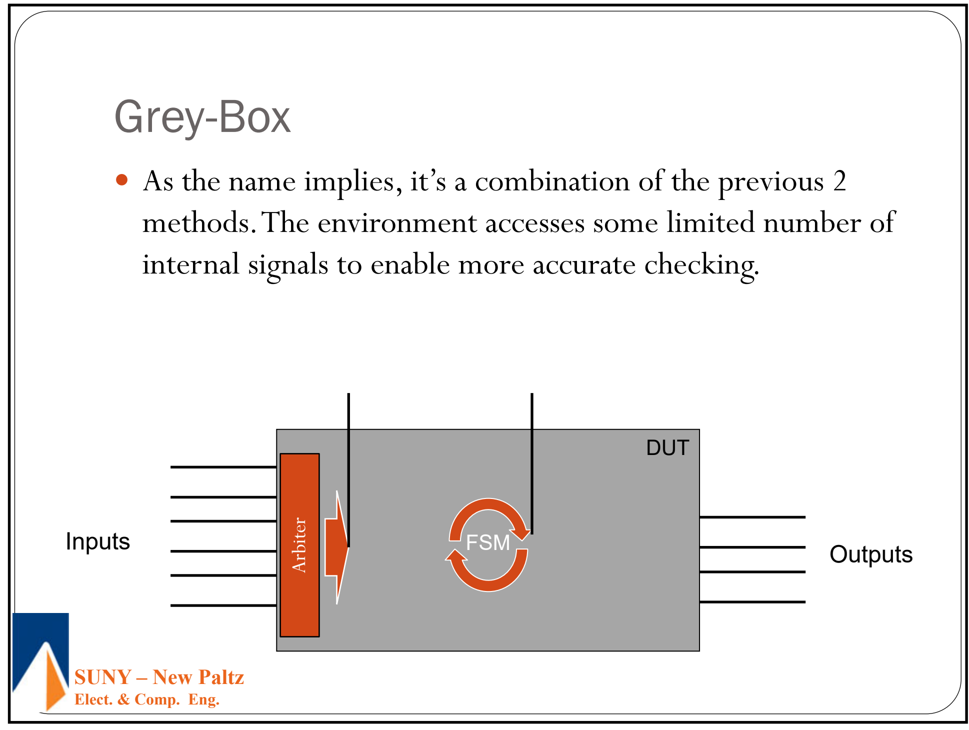
## White-Box

- Pros - Closely monitor execution for cycle accurate checkers
- Cons - Environment construction biased by knowing the exact implementation method. Easy to miss scenarios by using this in-depth knowledge. "The implementation won't handle X, so I'll never drive it."

DUT
Inputs
Arbiter
Reg A
Reg B
Reg C
FSM
WR
RD
Outputs

SUNY – New Paltz
Elect. & Comp. Eng.

18




19

## Grey-Box

- Most environments use this approach! Prediction of correct results on the interface is occasionally impossible without viewing an internal signal.

DUT

Inputs

Arbiter

FSM

Outputs

SUNY – New Paltz
Elect. & Comp. Eng.

20

## Simulation & Regression

- Simulation - Executing a single test/testcase
- Regression:
  1. Executing a suite of tests/testcases
  2. Frequently executing the same testcases with different seeds

So, you developed an environment…..

- You kick off 5 simulations. They each fail with a different error
- When trying to triage the fails, you find multiple transactions happening, making it hard to associate information and find the root cause.
- What is your testing objective at this time?

21

## The Art of Verification

Two simple questions, one huge task.

1. Am I driving all possible input scenarios?
2. How will I know when it fails?

22

## Develop a Test Plan

- The Test Plan is your key to a successful environment. This is developed by you, based on your interpretation of the specification(s).
- It must be detailed to reflect your understanding of the design, and how that design can be stressed.
- It should be written in a way to allow for comprehensive review and feedback from your peers.
- This is your guidebook for developing the environment and executing tests.

23

## Test Plan: Content Requirements

- Define the boundaries of the DUT.
  - What blocks of logic will be included? Will a different environment cover the logic that's not included?
  - Hardware arrays/memories, or use a software behavioral?
    - Any short-cuts should be defined and listed. Maybe a behavioral doesn't act like the real HW. How can you validate that?
- Generation: What features will be exercised? What are the min and max values for that stimulus? Do features need to interact depending on configuration?
- Checkers: List out all the checks for validating each feature. Be detailed.

24

## Test Plan: Define DUT Boundaries

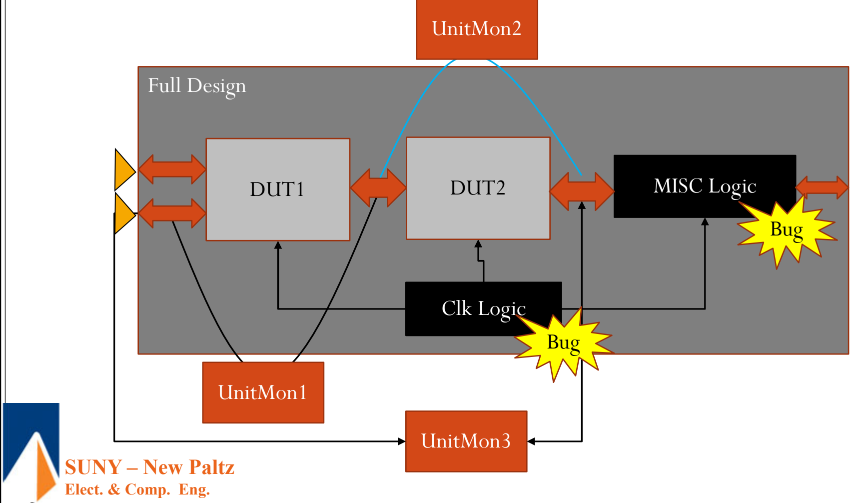
- What logic will you be testing?
- What logic will not be tested (by you)?
- Is there additional logic being left out?

SUNY – New Paltz
Elect. & Comp. Eng.

25

## Test Plan: No Gaps in Testing

- If it wasn't verified, it probably has defects.




SUNY – New Paltz
Elect. & Comp. Eng.

26

## Test Plan: Continued

- Why does this matter?
- Defines options for the physical environment structure.
  - How many generators will you have? Do they need to coordinate at all?
  - Should a configuration object be created to share details across objects?
  - How many monitors? Are they unique? Replicated?
  - How can you implement your checkers? Does some complicated check require additional information?

27

## Test Plan: Initializing the Environment

- How will the design be initialized?
  - Just a reset signal toggling?
  - Functionally writing registers/latches over a service bus?
  - Setting latches directly using the simulator API?
- What will the content be?
  - Do certain values need to be set in the design for it to function?
  - Is there an initialization sequence that needs to be followed?
  - What different configurations are supported?
  - Should we randomize any default values before starting simulation?
  - Are there special settings to enable a certain feature?

28

## Simulation & Regression Revisited

So, you developed an environment…..

- Refer to your Test Plan for what to test first.
- Test a single transaction. If it works….
- Test several transactions of the same type. If it works….
- Try a different transaction type….

- Walk before you Run - It is tempting to fully enable all features of your Generator immediately. But systematically testing individual features will validate your Env and the DUT faster.

29

## Simulation & Regression: Triage

Triage - Analysis of failures to determine root cause & severity

1. What reported the error, a Hardware checker in the design or a verif env checker?
2. If a verif env checker, review the checker to make sure it is correct.
3. Review the testcase and design configuration: Is the design in a good configuration for that testcase? Was the traffic driven by the testcase/generator legal?
4. Did you enable function which was not approved for testing?

Conclusion: Review your environment for correctness before pulling in the designer or other developers.

SUNY – New Paltz
Elect. & Comp. Eng.

30

## Sim & Regr: Tracing / Log Files

- Printing out what your environment is doing is the key to fast and effective debugging.
- Output information so it is searchable!
  - Be consistent in how you log details, following conventions agreed with your team.
  - Prefix lines with unique identifiers for your objects
- Use at least 2 verbosity levels for your messages
  - Low – only required messages are printed
  - High – detailed messages are printed, to give a deeper view about everything your object is doing.

31

## Tracing / Log Files Example Output 1

```
Cycle 2 USB_Drvr0: Start Read
                                  Addr=0x1234ABCD
                                  UID=B8_D2
Cycle 3 USB_Mon0: Start Read
                                  Addr=0x1234ABCD
                                  UID=B8_D2

Cycle 3 USB_Mon1: Start Write Addr=0xAAAA1100 UID=B8_D1 Data=0x11119999
Cycle 4 USB_Mon1: Data=0x7777666655554444
```

32

## Tracing / Log Files Example Output 2

```
Cycle 2 USB_Drvr0: UID=B8_D2 Start Read Addr=0x1234ABCD
Cycle 3 USB_Mon0: UID=B8_D2 Start Read Addr=0x1234ABCD
Cycle 3 USB_Mon1: UID=B8_D1 Start Write Addr=0xAAAA1100
Cycle 3 USB_Mon1: UID=B8_D1 Data=0x1111999988880000
Cycle 3 USB_Mon1: UID=B8_D1 Data=0x7777666655554444
```

- You can easily search (grep) for the Object name, or UID, to see what is happening for a specific bus.
- Clearly formatted log files will speed up the debug time of both the Environment and Design.

33

## Simulation & Regression

Regression – definition 1

- Defect found. Get fix for DUT. Validate fix.
- When testcase is clean, it becomes part of the regression suite.
- This is a set of tests to run to prove the design hasn't somehow gotten worse.
  - Maybe a fix breaks other functionality?
  - Or a newly added feature has unexpected effects?
- Use a suite of regression tests to prove DUT stability it maintained. (Definition 1)

34

## Simulation & Regression

Regression – definition 2

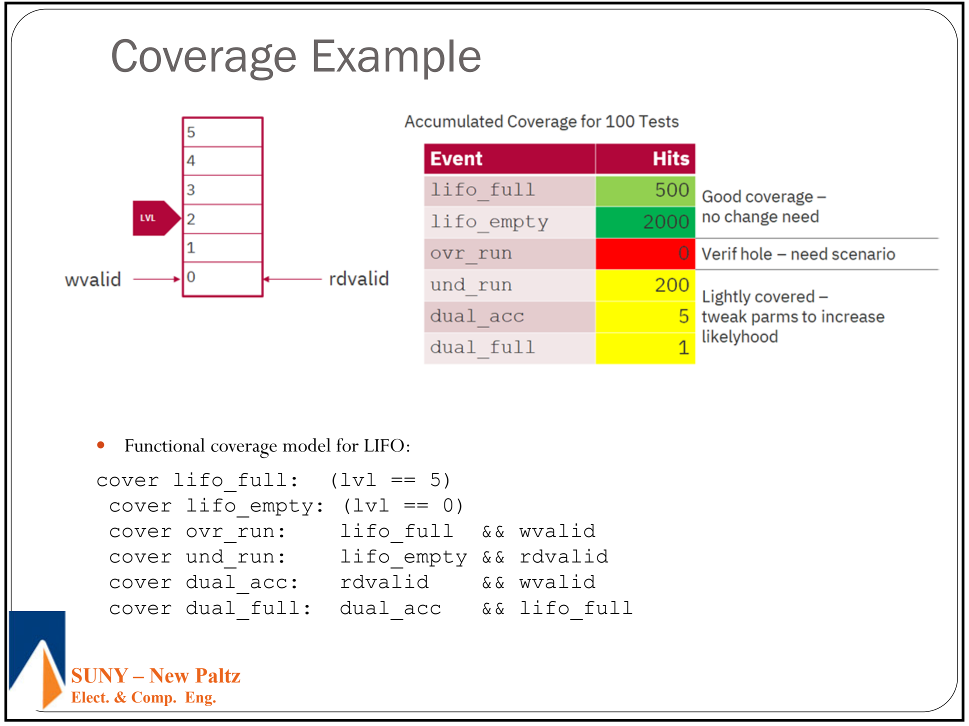
- Large scale execution of defined testcases with randomized seeds.
- Each testcase is executed many times with new randomization, to achieve variance in transaction content and timing.
- This large number of simulations leads to stressing the design, and getting good coverage of the function.

35

## Test Plan: Coverage

- Coverage methods and metrics should be defined in your test plan.
- Coverage events are used to track execution of the logic and are compiled directly into the model.
  - What states did it get into?
  - Did buffers get filled?
  - What features got exercised?
- The data is collected over many simulations, providing a picture of how well the design has been tested.

36




| Event | Hits |
|---|---|
| lifo_full | 500 |
| lifo_empty | 2000 |
| ovr_run | 0 |
| und_run | 200 |
| dual_acc | 5 |
| dual_full | 1 |

- Functional coverage model for LIFO:

```
cover lifo_full:  (lvl == 5)
 cover lifo_empty: (lvl == 0)
 cover ovr_run:    lifo_full  && wvalid
 cover und_run:    lifo_empty && rdvalid
 cover dual_acc:   rdvalid    && wvalid
 cover dual_full:  dual_acc   && lifo_full
```

37

## Test Plan: Coverage

- But wait, there's more!
- Environment coverage:
  - Are generators being fully utilized?
  - Did checkers get executed?
  - Are all testcases still running?
  - Did something in the environment stop working?

38

## Test Plan: Review

- Schedule reviews of the plan, environment, model, coverage and processes used.
- Verification is an iterative process.
- Be prepared to adapt and change your strategy, and environment, based on learning from defects found, or coverage hit/missed.
- And most importantly: Was anything missed?

39

## Are we done yet?

Metrics to know if the design is likely good to be fabricated.

- How much regression was run ?
- What different kinds of tests were run ?
  - Was the full range of settings in the environment actually run?
- How many defects were found? How many is enough? Difficult question.
  - Maybe the designer is that good?
  - Maybe the design isn't complicated?
  - Maybe your environment isn't stressing the DUT.
  - How does the bug rate compare to other units in the design?
  - Ask for a review of your environment from peers.

40

## Are we done yet? (cont.)

- New defect discovery rate
  - Are you still finding new failures often? How often?
  - Are recent defects minor or serious problems?
- Existing defects understood and resolved?
  - Was anything glossed over and ignored?
- Hardware event coverage reviewed
  - Not only were the events hits, but are the defined events interesting?
  - Are their other events which should be added for tracking?
- Did you fully review your environment with team members?

41

## Three Simulation Commandments

- Thou shalt stress thine logic harder than it will ever be stressed again

Thou shalt stress thine logic harder than it will ever be stressed again

Thou shalt place checking upon all things

Thou shalt not move onto a higher platform until the bug rate has dropped off

SUNY – New Paltz
Elect. & Comp. Eng.

42

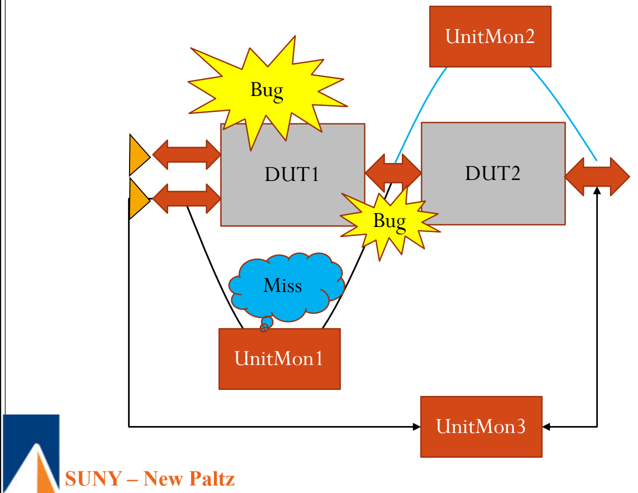
## Multiple Environments

- All the planning has been for a single environment
- Projects typically employ multiple environments to stress the design using several methods.
- As we covered earlier:
  - Designer Sim
  - Unit Sim
  - Element Sim (multiple units)
  - Chip / System Sim
- To reduce chance for human error, we want overlapping environments.

43

## Test Plan: Overlapping environments

- Implement overlapping checkers/envs for redundancy.




SUNY – New Paltz
Elect. & Comp. Eng.

44

## Verification and design reuse 1

- Two aspects of re-use
  - Environment re-use: Your checkers being re-used at a higher level of sim.
  - Design re-use: DUT being imported from, or sent to be used by, a different team/project.

45

## Verification and design reuse 2

Your checkers being re-used at a higher level of sim

- Overlapping environments means other people re-using parts of your environment.
- Your environment needs to be configurable to fit that higher level of sim functionally and structurally.
- Can your environment work if Generators or Drivers were removed? (Hint: The answer should be yes.)
- Can some checker objects be disabled/removed if sim has some need to remove them? (Performance reasons or false fails)

46

## Verification and design reuse 3

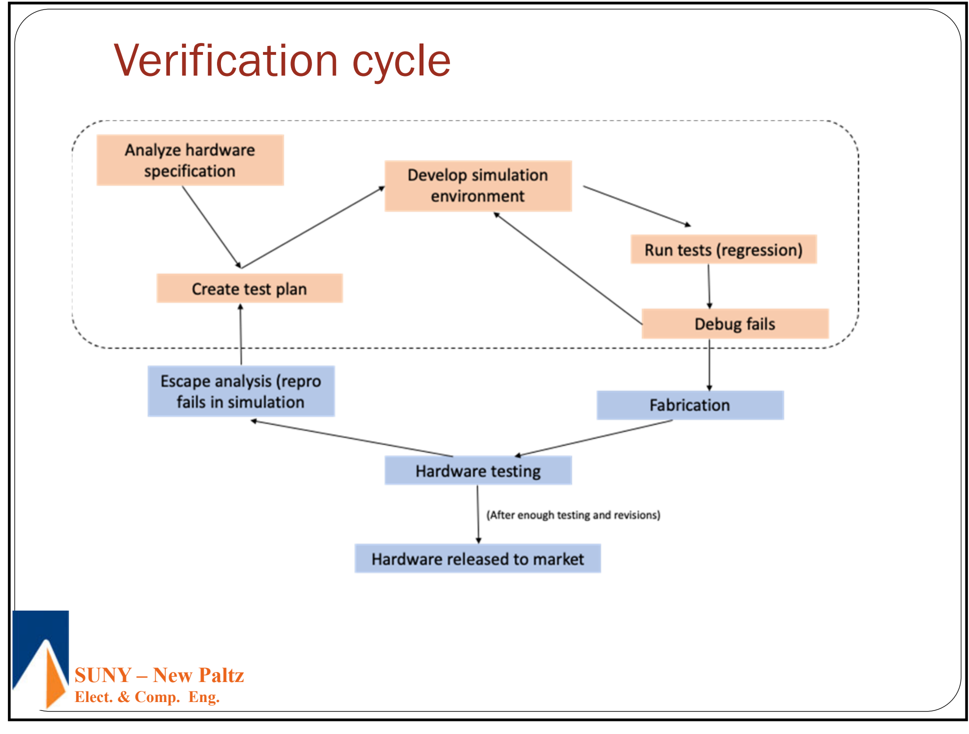
Design re-use requires Trust

- How to trust it?
  - Verify it.
- For reuse, designs must be verified with more strict requirements
  - All claims, possible combinations and uses must be verified.
  - Not just how it is used in a specific environment.

47

## Additional Slides

48




49

## Verification terminology

- Facilities: a general term for named wires (or signals) and latches. Facilities feed gates (and/or/nand/nor/invert, etc) which feed other facilities.
- EDA: Engineering Design Automation--Tool vendors.
- Behavioral: Code written to perform the function of logic on the interface of the design-under-test. A model that emulates the function of the design.
- Macro: 1. A behavioral 2. A piece of logic
- Driver: Code written to manipulate the inputs of the design-under-test. The driver understands the interface protocols.
- Checker/Monitor: Code written to verify the outputs of the design-under-test. A checker may have some knowledge of what the driver has done. A check must also verify interface protocol compliance.
- BFM: The techniques for applying stimulus and monitoring the response of a design is done by abstracting the operations of an interface – these are called bus functional models

50