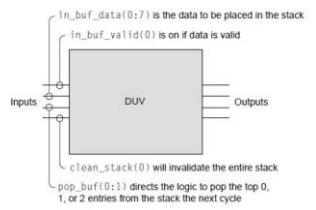
L8 – VERIFICATION TEST PLAN

Michael Quinn (original slides by Dr Aakash Tyagi)
CSCE 689-698 Advanced HW Verification Spring 2018

Theme of Today's Lecture

- Composition and Structure of a Verification Testplan
- Slide Material from:
  - [1] Cameron Wilde, Portland State University, The Verification Plan, ECE510, S4
  - [2] Janick Bergeron, IBM/NCSU Verification Course, Chapter 3
- Additional Content Sources:
  - [3] Hardware Design Verification – Simulation and Formal Method-Based Approaches, William K. Lam, Chapter 5, pp. 211-231
  - [4] Comprehensive Functional Verification: The Complete Industry Cycle. Bruce, Wile, Goss, John, Roesner, Wolfgang. Morgan Kaufmann, 2005 : Chapter 4, pp. 103-120
  - [5] Building A Verification Test Plan: Trading Brute Force For Finesse, Panel Session, DAC 2006, pp 805-806

Prologue



Known

- There is a stack inside.
- The stack is at least two deep.
- The stack is 8 bits wide.
- Only one entry can be written at a time.

Unknown

- How deep is the stack?
- What conditions indicate a full stack?
- When do the contents become valid?
- What is the behavior if a read and write occur on the san Is that even allowed?
- How long does it take to reset the stack?
- Do the entries get zeroed-out or just marked invalid?
- What happens if a read operation occurs when the stack i:
- In case of reading two entries, how are the two data items returned Back to back or one at a time?
- What happens if pop_buf(0:1) is set to "11"b (e.g., three reads)?

Blackbox Testplan

Stimulus cases	Description
Writing and not writing	Writing data to the stack is obvious. However, the verification engineer must include in the test plan cases in which data are not written to the stack for multiple cycles.
Writing and reading	The inputs appear to allow for simultaneous reads and writes to the stack.
All three possible reads	There are three different decodes (none, 1, or 2) described for pop_buf(0:1) and the verification engineer must create all three cases.
Reading when there is nothing in the stack	The correct function of the design in this case has not been defined yet. Regardless of whether or not the design will return no data or an error, the verification engineer must include this case in the test plan.
Writing when the stack is full	At this point, both the depth of the stack and the correc function when data is written to a full stack is not defined. The test plan still must include this scenario.
Reading from the stack and resetting it (clean_stack)	Concurrent conflicting operations, such as reading from the stack and resetting the contents of the stack are often a source of bugs.
Writing to the stack and resetting it (*clean_stack*)	Again, this concurrent conflicting operation must be included in the test plan.
All bits of data	The test plan should include verifying that each bit of the width of data (8 bits) can be either '0'b or '1'b, and that each line in the stack correctly holds the 8 bit value.
Temporal cases such as writing back-to-back with a double read on the first cycle (starting with an empty stack).	Temporal cases, where stimulus evolves over multiple cycles, are always the most difficult to create—and most often where bugs lurk.

Outline

- Background and Motivation
- Evolution of the Verification Plan
- Contents of the Verification Plan

Planning is Essential[1]

5

"Being busy does not always mean real work. The object of all work is production or accomplishment and to either of these ends there must be forethought, system, planning, intelligence, and honest purpose, as well as perspiration. Seeming to do is not doing." -- Thomas A. Edison

"Let our advance worrying become advance thinking and planning." -- Winston Churchill

"A goal without a plan is just a wish." -- Antoine de Saint-Exupery

"He who every morning plans the transaction of the day and follows out that plan, carries a thread that will guide him through the maze of the most busy life. But where no plan is laid, where the disposal of time is surrendered merely to the chance of incidence, chaos will soon reign." -- Victor Hugo

A Verification Plan is[1,2,3,4] .....

6

- This is the specification for the verification effort. It gives the **WHAT** am I verifying and **HOW** am I going to do it!
- A description of the effort needed to accomplish the task
- Owned by everyone (architecture, design, validation)
- Done for every *selected* element at each hierarchical level of integration
- A living document
- Built on the foundations of Design Architecture and Micro-Architecture Specs

Role of the Verification Plan[2]

7

- Calls out the Verification approach and effort
- Lays the foundations and establishes the expectations for 1st time success (ideally)

Specifying the Verification Plan[2]

8

- The specification determines the "what has to be done"!
- The specification does NOT determine the following:
  - How are you doing what needs to be done? [Verification Plan]
  - When are you done? Metrics [Project Plan]
  - How long will it take? Schedule [Project Plan]
  - How many people will it take? Resource Plan [Project Plan]
- Verification & Project Plans define these!

Specifying the Verification Plan[2] (contd)

9

- The verification plan starts from the design specification – why – The reconvergence model!!
- Must exist in written form
  - "the same thing as before, but at twice the speed, with these additions…" – Not acceptable specification (design or verification)!

Spec is Golden

10

- Observe the following example: "The response R shall occur when A occurs after B or C"[4]
- Describes what and how something should be implemented
- When there is a difference between Design and Validation, how do you decide who is correct?
- What do you do when the specification is vague? Who is right?
- The specification is a golden document
  - It is the common source for the verification efforts and the implementation efforts.
- Spec is THE reference and must be ready by the end of TR so as to enable the Planning and Creation of the Verification Plan

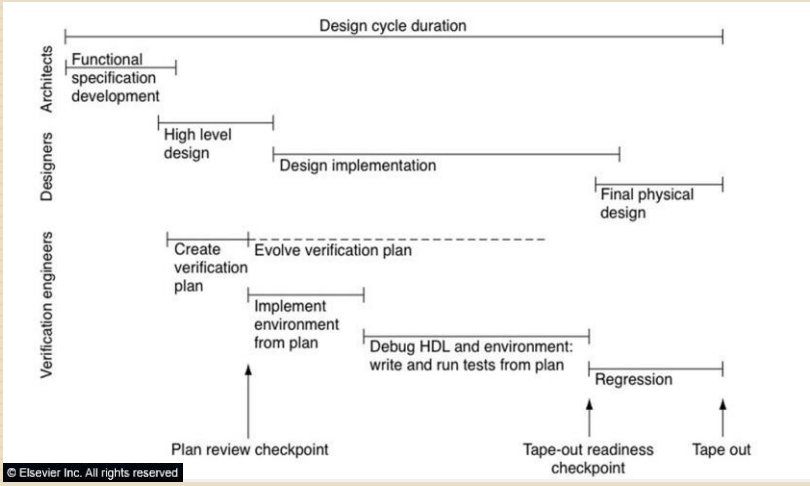
Defining 1st time success[2]

- The plan determines what is to be verified
  - If, and only if, it is in the plan will it be verified
  - The plan provides the forum for the entire design team to determine 1st time success
  - For 1st time success, every feature must be identified and under what conditions as well as expected response
- Plan documents these features (as well as optional ones) and prioritizes them.
  - This way informed risk (i.e. ZBB of some plan items) can be used to bring in the schedule or reduce cost.

Defining 1st time success[2] (contd.)

- Plan creates a well defined (ZBB) line, that when crossed can endanger the whole project and its success in the market
- It defines:
  - How many test scenarios (testbenches/testcases) must be written
  - How complex they need to be
  - Their dependencies
- From the plan, a detailed project plan and schedule can be produced:
  - Number of resources it will take – people, machines, etc.
  - Number of tasks that needs to be performed
  - Time it will take with current resources

Evolution of the Plan[4]



Contents of the Plan[1,4]

- Technical Requirements
  - Description of the Verification Levels
  - Features to be Verified
  - Specific Test Methods
  - Coverage Requirements
  - Test Scenarios
- Project Management
  - Required Tools
  - Risks and Dependencies
  - Resource Requirements
  - Schedule Details

In your project testplan, we only expect technical requirements

Description of the Verification Levels[1]

15

- How do you break down the design?
- What is the verification complexity of each level?
- What reuse do you have between the levels?
- What about the specification for each level?
- What about control and observability?
- Create a PLAN PER LEVEL (or Levels/Plan-Sections)
  - When to test at Block/IP level versus when to migrate the testing to next level of integration (all the way to Sub-System and SoC levels)
  - Must be chosen carefully to weigh Risk and ROI

Features to Be Verified[1,4]

16

- Use the Specification to generate a list of items to be verified
- Need to include global functions: e.g. clock, reset, error handling, and debug functions
- Three types of features (at least) to be included
  - **Critical Features**: Things that must be checked or the design will be Dead on Arrival. In other words, if these things don't work, the chip won't work AT ALL. [Establishes ZBB line]
  - **Secondary Features**: Things that aren't critical to the chip working. They are still important but it allows for verification at the next level.
  - **Non-Verified Features**: Things that will NOT be verified at this level due to it being fully done at a previous level and/or sanity checks will be performed later. It may be too hard to check at this level.

Specific Test Methods[1,4]

17

- Black, White, or Grey Box – ramifications of these decisions (controllability vs. observability)?
- Stimulus: Random, Directed, Pseudo-Random, and/or Formal Verification
- Things to consider:
  - Areas that need targeted tests
  - Coverage of tests (can all possible permutations be hit?)
  - Reuse at higher levels of integration
- Checking: Golden vectors, Reference model, Transaction-based, Micro-Architectural, Architectural, Cycle-Accurate, …

Coverage Requirements[1,3,4]

18

- Feedback that tells you how good your stimulus is
- Why is it needed? What do you lose without it?
- Covers commands, data, concurrent stimulus, errors, illegal conditions (type of checking), …

Test Case Scenarios[1,4]

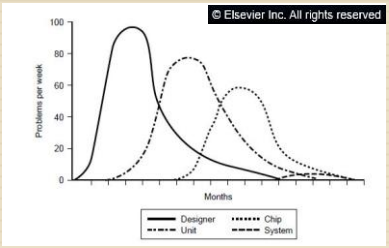
19

- Lists of tests to be written (directed, pseudo-directed). Should list intent and description
- Start with basic and then move to more complex
- Can cross tests cases (to make a very large number of tests)
- Can use random tests to hit many of these cases
- Covers legal, illegal, and corner cases
- IMPORTANT!! Clearly indicate which Features to be Verified are covered by each Test Case

Three Verification Commandments[2]

20

- Thou shalt stimulate/stress thy logic harder than it will ever be stimulated/stressed again!
- Thou shalt place checking upon all things!
- Thou shalt not integrate into a higher platform until coverage is high and bug rate is low!



21

Calculator Overview

Functional Verification

Calculator Design

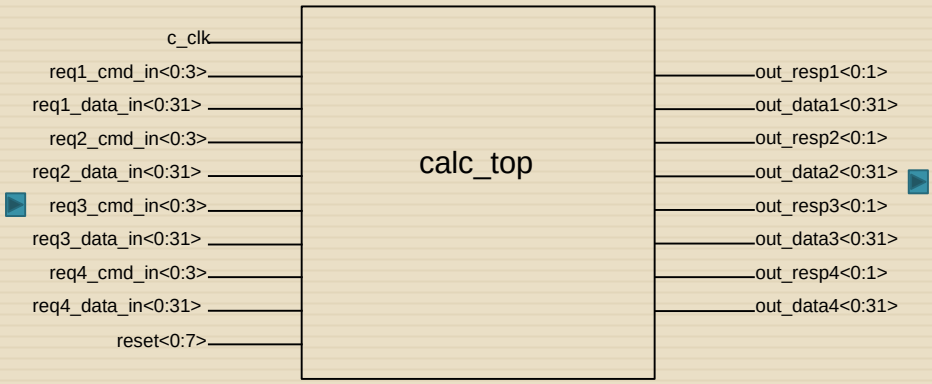
22

- Calculator has 4 functions:
  - Add
  - Subtract
  - Shift left
  - Shift right
- Calculator can handle 4 requests in parallel
  - All 4 requestors use separate input signals
  - All requestors have equal priority
  - Each port must wait for its response prior to sending the next command

Calculator design

23

- Input/Output description



Calculator Design

24

- I/O Description
  - Input commands:
    - 0 - No-op
    - 1 - Add operand1 and operand2
    - 2 - Subtract operand2 from operand1
    - 5 - Shift left operand1 by operand2 places
    - 6 - Shift right operand1 by operand2 places
  - Input Data
    - Operand1 data arrives with command
    - Operand2 data arrives on the following cycle

Calculator Design

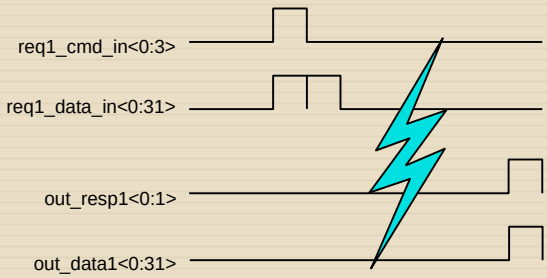
25

- Outputs
  - Response line definition
    - 0 - no response
    - 1 - successful operation completion
    - 2 - invalid command or overflow/underflow error
    - 3 - Internal error
  - Data
    - Valid result data on output lines accompanies response (same cycle)

Calculator Design

26

- Input/Output timing



Each port must wait for its response prior to sending the next command!

Calculator Design

27

- Other information
  - Clocking
    - When using a cycle simulator, the clock should be held high (c_clk in the calculator model)
    - The clock should be toggled when using an event simulator
  - Calculator priority logic
    - Priority logic works on first come first serve algorithm
    - Priority logic allows for 1 add or subtract at a time and one shift operation at a time

Calculator Design

28

- Other information (con't)
  - Resets
    - Hold reset(1:7) to '1111111'b at start of testcase for seven cycles.
    - During the reset period, outputs of the calculator should be ignored
  - Shift operation
    - Only the low order 5 bits of the second operand are used
  - Arithmetic operations are unsigned

Calc1 Testplan

29

- Handout posted on Piazza

Back-Up Slides

30

Required Tools[1,4]

31

- NOT PART OF VERIFICATION PLAN! PART OF PROJECT PLAN
- What tools do you need to do your job?
- Is a methodology in place or do you need new tools?
- Software and Tools
  - Linter
  - Assertion Based
  - Debuggers
  - Formal Verification
- Languages
  - Verification languages (Specman, SystemC, Vera)
  - Libraries
- Hardware
  - Emulation
  - Acceleration
  - Co-Simulation

Risks and Dependencies[1,4]

32

- NOT PART OF VERIFICATION PLAN! PART OF PROJECT PLAN
- New/Buggy Tools
- Integration with established tools
- Education/Training
- Late RTL
- Dependence on an outside vendor (e.g. IPs)
- Poorly verified code
- Poor Architecture/Specification
- Resources – Machines, licenses, and personnel

Resource Requirements[1,4]

33

- NOT PART OF VERIFICATION PLAN! PART OF PROJECT PLAN
- People
  - Environment targeted – New vs. Reuse
  - Experience of employees
  - Staff lowest level and then move up
  - Test writers, Debuggers, Environment, Level Specific
- License
  - Tools to be run * Number of Machines operating at the same time
- Compute
  - How many machines do you need?
  - What type of machines?

Schedule Details[1,4]

34

- NOT PART OF VERIFICATION PLAN! PART OF PROJECT PLAN
- How much change is there?
- Order things that need to be done
- Compute resources affect schedule
- When will the design be ready to be tested?
- When does the product need to hit the market to be impactful?
- How much time does post-silicon need to verify the product?
- Use history as an aide and predict with accuracy when the product will be ready