Introduction to Verification Process

Krerk Piromsopa, Ph.D.
Department of Computer Engineering
Chulalongkorn University

1

Outline

- Definitions and Terminology
- Design Process
- Test Strategy
- Coverage
- Event Monitor & Assertion Checkers

2

Design Level

- Register Transfer Level (RTL)
  - meaning
    - Register – storage elements
    - Transfer – equations and transformations
      - input-to-register
      - register-to-register
      - register-to-output
    - Level – of abstraction
  - Cycle-by-cycle, or state-by-state
  - No specific timing or physical characteristics

3

Design Level

- Architectural Level
  - Higher level, no detail about cycle-by-cycle.
  - Correspondence design in RTL
  - You will learn next semester
- Gate-Level
  - Timing and physical characteristics of the gate

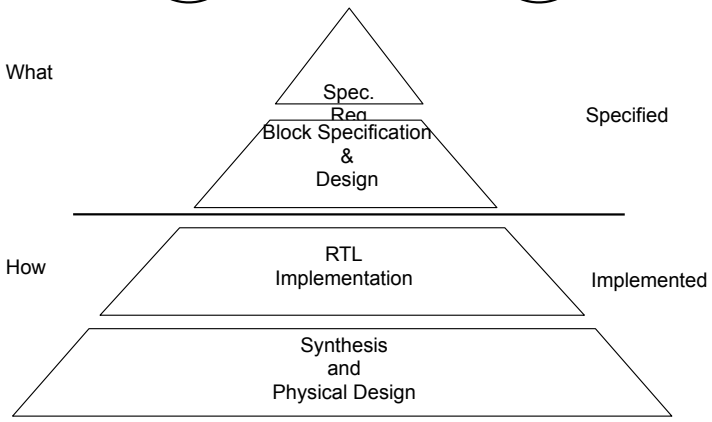
4

Hardware Design

- Schematics
- Boolean equations (1938)
- Block diagrams and timing
- Flip-flop input equations (1952)
- RTL (1968)

5

Life Cycle



6

Fundamental Verification Principle

- Implementation of RTL code mush follow completion of the specification to avoid unnecessarily complex and unverifiable designs

Don't rush to code.

7

High-Level Design

- Modeling Algorithms, not the design implementation details
- Formal Specification, Executable Specification, Table specification
- HDLs are insufficient as specification languages:
  - Lack of expressing environment design assumptions, non-determinism and temporal properties,
  - Tendency to migrate the specification toward a lower-level cycle-by-cycle implementation model

8

Block-Level Specification and Design

- Partitioning and Decomposition
- Advantages
  - Partitioning promotes verification coverage
  - Improve synthesis productivity
  - Facilitate formal verification methodologies
  - Enable Parallel development of block-level test benches during RTL implementation.

9

How (Implementation)

- Well, you have already mastered Verilog and the EDA tools.
- You will learn to use FPGA and other tools next semester.

10

Testing vs. Verification

- During the process
- After the process

11

Functional Test Strategies

- Each level have different objective.
  - Executable specification – algorithmic requirement (order of loads/stores)
  - RTL – divide-and-conquer and bottom up verification approach.
    Transaction based verification

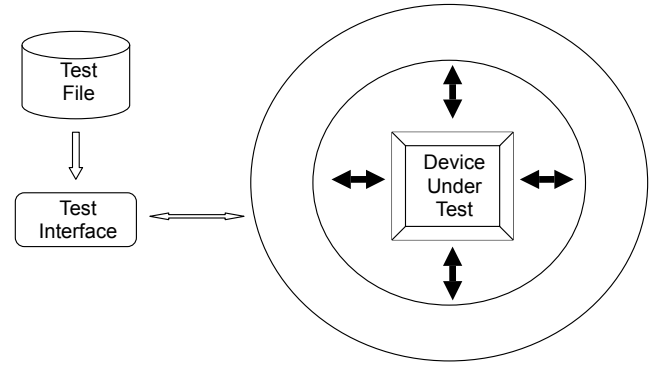
12

Functional Test Strategies

- Deterministic or Directed Test
- Random Test
- Transaction Analyzer Verification

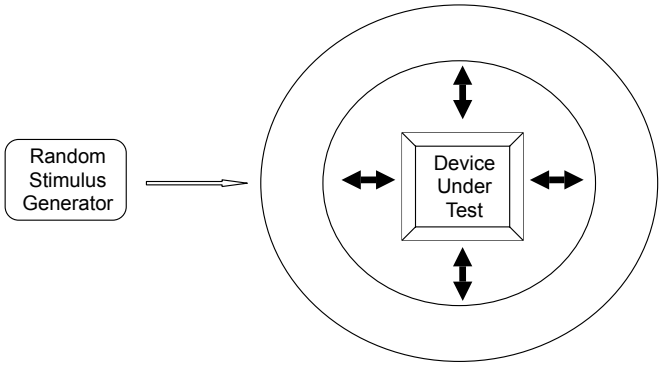
13

Direct Test



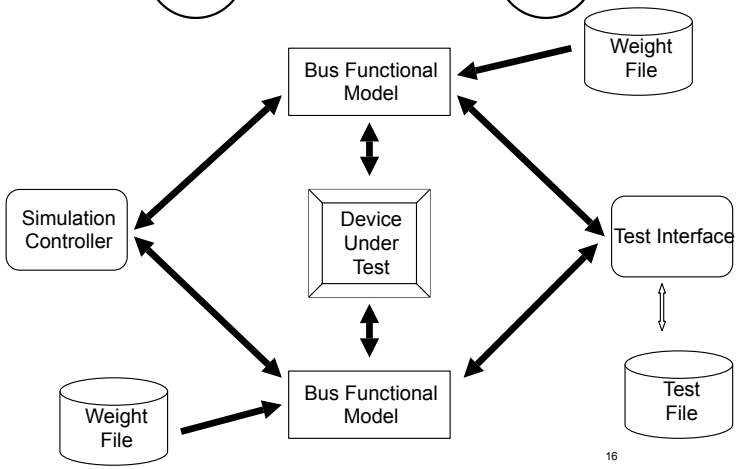
14

Random Test



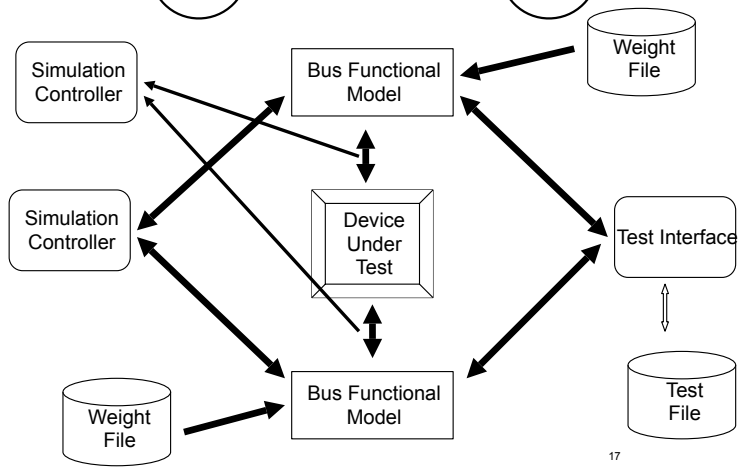
15

Hybrid



16

Transaction Analyzer Verification



17

Others

- Chip Initialization Verification
  - Dead-on-arrival (power-up)
- Verification Coverage

18

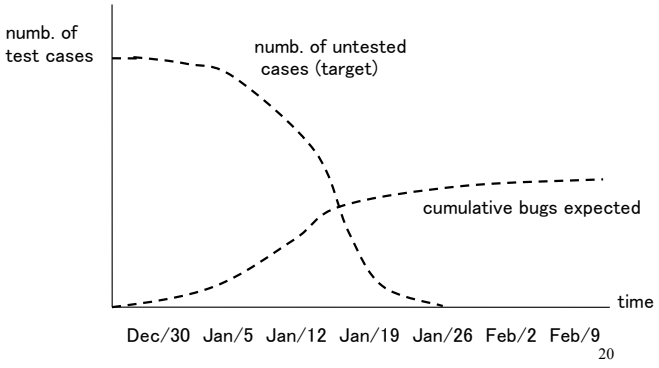
Coverage

- Ad-hoc metrics
  - Bug detection frequency
  - Length of simulation after last bug found
  - Total number of simulation cycles

19

(3) Bug Data Analysis by Project Manager

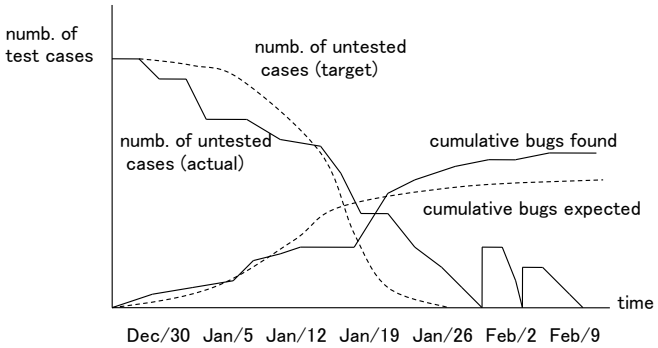
- A project manager drows the « test progress plans» before getting into an actual testing.



20

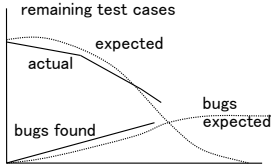
(3) Bug Data Analysis by Project Manager

The Project Manager plots the test progress on the chart:
- **to determine the best strategies to improve the quality**
- **to predict the release date.**

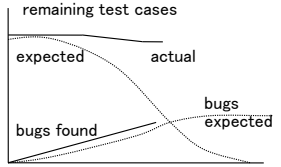


21

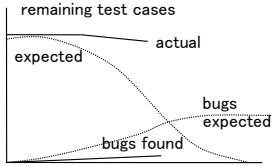
(3) Bug Data Analysis by Project Manager (continued)



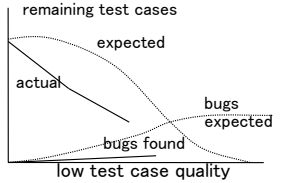
best case



low quality in previous process



inexperienced testers

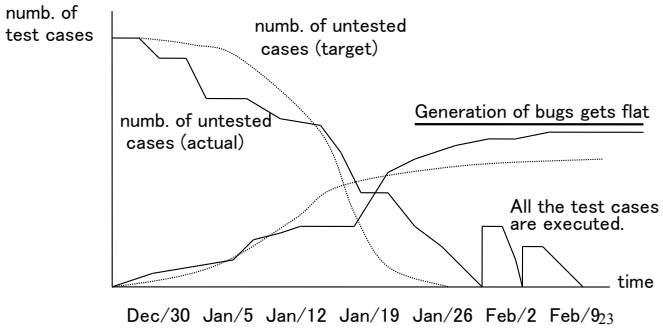


low test case quality

22

(3) Bug Data Analysis by Project Manager

Project Manager thinks it's time release when:
- **All the test cases are executed.**
- **The growth of bugs gets flat.**
- **All the bugs are fixed.**
- **48-hour continuous operation successfully completed.**



23

Coverage

- Programming Code Metrics
  - Line coverage
  - Branch coverage
  - Path coverage
  - Expression coverage
  - Toggle coverage
- Limitation
  - Controllability vs. Observables
  - No qualitative for functional correctness

24

Coverage

- State Machine and Arc Coverage Metrics
- Fault Coverage Metrics
- Regression Analysis and Test Suite Optimization (Agile ??)

25

Event Monitors and Assertion Checkers

- Black-box testing vs. White-box testing
- Measuring functional correctness and detecting erroneous behavior.
- Advantages
  - halts simulation on assertion errors
  - simplifies debugging by localizing the problem to a specific area of code
  - Increases observability
  - Grading coverage
  - Enable the use of formal and semi-formal techniques
  - Capturing and Validating design environment assumptions and constraints

26

Events

- A desirable behavior
- Boundary (corner-case detection)
- Static event (a unique combination of signals at some fixed instance of time)
- Temporal event (a unique sequence of events or state transitions over a period of time)

27

Monitoring Code

•Embedded into RTL or encapsulate in a module

```
`ifdef EVENT_MONITOR
always
   @(c_q_full or c_q_write) begin
   @(negedge ck) begin
        if (c_q_full & c_q_write)
   $display("EVENT%0d:%t:%m",
   `EV_Q_FULL_WR,$time);

   end
End
`endif
```

Directly embedded into RTL

```
event_monitor dv_q_full_write(
   ck,c_q_full & c_q_write,
   `EV_Q1_Q2_FULL);

module event_monitor(…..);
….
…
$display(….)
…

endmodule
```

Encapsulate in a module

28

Assertions

- Check for undesirable behavior during the verification process (illegal event or and invalid design assumption)
- Static and temporal

29

Conclusion

- From DEC alpha 21261 Microprocessor
  - Assertion checkers 34% of bugs
  - Self-Checking directed test 11% of bugs
- Event Monitor Database and Analysis
- How will you apply this to your design?

Learn from experience....

30

Bad Stuffs

- Race Condition

```
always @(posedge clk)
    b=a;

always @(posedge clk)
    c=b;
```

31

Bad Stuffs

- Incomplete Sensitivity List

```
module b (p, w, x, y, z);
input [7:0] w, x, y, z;
output [7:0] p;
wire [7:0] w, x, y, z;
always @(w or x or y)
begin
    r = w | x;
    s = y | z;
    p = r &s;
end
```

32