Digital Design Verification

Course Instructor:

Debdeep Mukhopadhyay

Dept of Computer Sc. and Engg.

Indian Institute of Technology Madras

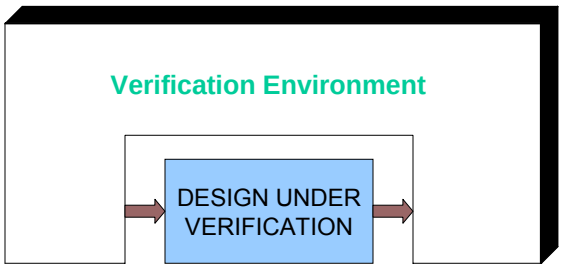
Verification ???

- What is meant by

  "Formal Property Verification"?
- *Options :*
  1. *Formal method of verifying a property*
  2. *Verifying of Formal Properties*
- *Ambiguity of English (natural) Language*
- *Formal Specifications*
- *Bugs are more costly than transistors !!!*

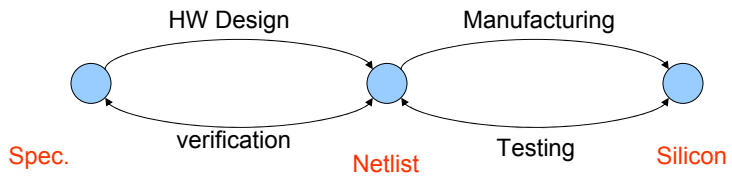
Verification??

- *Process* used to demonstrate that the *intent of design* is preserved in its implementation
- 70% of design effort goes behind verification



Testing vs Verification

- Testing verifies that the design was manufactured properly
- Verification ensures that a design meets its functional intent



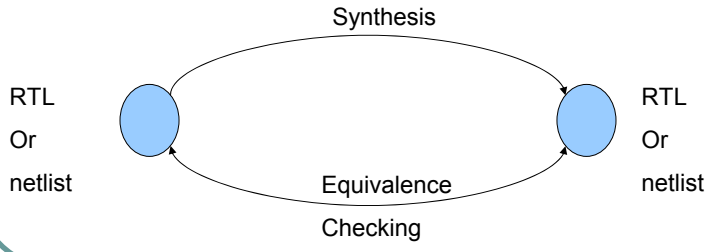
Reconvergence Model: Conceptual representation of the verification process

Types of Verification

- Formal Property Verification
  - Formal Technique to verify formal properties
  - Verifies *all* properties of the design satisfy the properties
  - *Static Property Verification*
- Assertion Based Verification
  - Properties checked during simulation
  - Verification confined to those areas that are encountered during simulation
  - *Dynamic Property Verification*

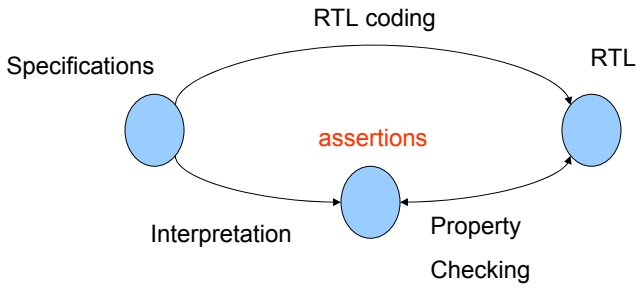
What is being verified?

- Equivalence Checking
  - Mathematically proves that the origin and output of a transformation of a netlist are logically equivalent



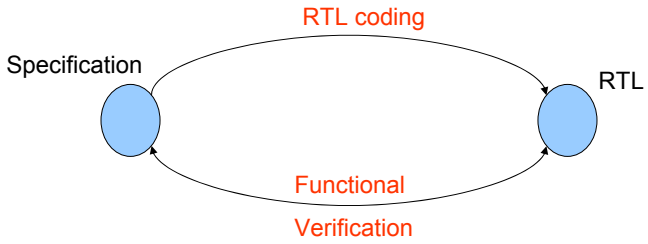
Property Checking

Assertions: Characteristics of a design



Functional Verification

- Ensures that a design implements intended functionality
- Can show but not prove



What is a test-bench?

- Simulation code used to create a pre-determined input sequence to a design and check the output response
- Verification Challenge:
  - What input patterns are to be applied to the DUV
  - What is the expected output response of a proper design under the applied stimuli

Types of mistakes

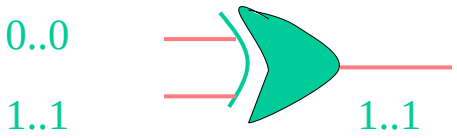
	Fail	Pass
Bad Design		Type II (False Positive)
Good Design	Type I (False negative)	

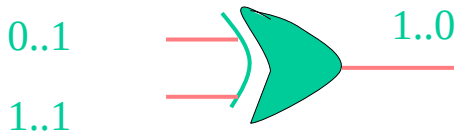
Verification Methodolgies

- Linting
- Simulation: Most common tool for verification
- Approximation of reality: 0, 1, x, z
- Requires stimulus
- Responses are validated against design intends

Event Driven Simulation

- Simulators are always slow
- Outputs change only when an input changes





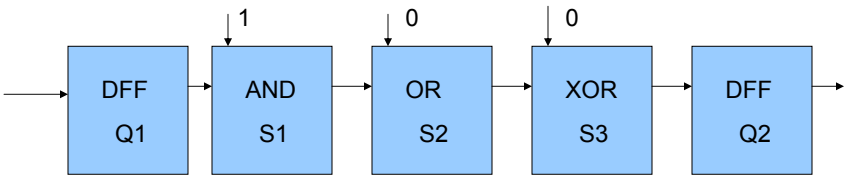
The simulator could execute only when one of the inputs change

assign out = in1 ^ in2; //verilog code snipet

What if both the inputs change?

- Logical world vs physical world
- Unknown or ‘x’ state
- Black box simulation

Cycle Based Simulation



- Assume Q1 holds a zero and Q2 holds a 1 initially
- An Event Driven simulation requires 6 events and 7 models
- If we are interested only in the final states of Q1 and Q2, the simulation could be optimized by acting only on the events for Q1 and Q2
- Simulation is based on clock cycles

CBS

- When the circuit description is compiled all combinatorial functions are collapsed into a single expression that can be used to determine all the ff values depending on the current state of the fan-in flops
- Ex: S3 = Q1 (check)

- During simulation, whenever the clock input rises the value of the ff-s are updated using the input value returned by the pre-compiled combinatorial input functions
- CBS requires generation of 2 events and execution of one model
- The number of logical computations does not change

- Gain when time required to perform logic computations are smaller than that required to schedule intermediate events
- Thumb rule: Large number of registers changing state at every clock cycle
- Loss: All timing and delay information is lost
- Assumes that setup and hold time are met
- Use a static timing analyzer
- Dynamic and static timing analysis

- Synchronous
- Asynchronous inputs, latches or multiple-clock domains cannot be simulated accurately

Few other points about simulators

- Co-simulators
- Avoid wave-form viewers
- Use assertions

Coverage

- Code Coverage
- Statement Coverage
- Path Coverage

Tasks of a Verification Engineer

- Development of the formal property specification
- Check the consistency and completeness of the specifications
- Verifying the implementation against the formal property specifications

Example of a priority arbiter

- mem-arbiter(input r_1,r_2,clk,output g_1,g_2)
- Design Intent:
  1. Request r_1 has a higher priority. When r_1 goes high, grant g_1 goes high for the next two clock cycles
  2. When none of the request lines are high, g_2 is high in the next clock cycle
  3. The grant lines g_1 and g_2 are mutually exclusive

Writing Formal Specifications

- Lots of languages
- Temporal Languages
  - Propositional logic
  - Temporal Operators: truth of propositions over time
  - Concept of time

Linear Temporal Language (LTL)

- X: The Next Time Operator
  - The property Xφ is true at a state if φ is true in the next cycle, where φ may be another temporal property or boolean property.
- F: The Future Operator
  - The property Fφ is true at a state if φ is true at some time in the future

LTL (contd.)

- G: Global Operator
  - The property Gφ is true at a state if the property φ is always true
- U: Until Operator
  - The property φUΨ is true at a state, if Ψ is true at some future state t, and φ is true at all states leading to t.

Property 1 in LTL

1. Request r_1 has a higher priority. When r_1 goes high, grant g_1 goes high for the next two clock cycles

LTL Spec:

G[r_1 => $Xg_1 \wedge XXg_1$]

G : The property must hold at all states

Property 2 & 3 in LTL

2. When none of the request lines are high, g_2 is high in the next clock cycle:

$$G[\neg r_1 \wedge \neg r_2 \Rightarrow Xg_2]$$

3. The grant lines g_1 and g_2 are mutually exclusive:

$$G[\neg g_1 \vee \neg g_2]$$

Specification of correctness?

- Very difficult to check.
- No formal property to check against
- However we may check for contradiction among the properties

In-consistencies

$G[\, r_1 \Rightarrow Xg_1 \wedge XXg_1]$

$G[\neg r_1 \wedge \neg r_2 \Rightarrow Xg_2]$

$G[\neg g_1 \vee \neg g_2]$

Model:

GAME

Environment: r_1 is high at time t but low at time (t+1), r_2 is low at time t and (t+1)

Hence, g_1 should be high at time (t+2), by property 1

g_2 should be high at time (t+2), by property 2

Contradicts property 3.

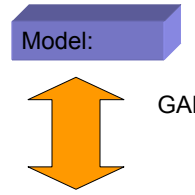
Environment Wins

Removing the In-consistency

$G[r_1 \Rightarrow Xg_1 \wedge XXg_1]$

$G[\neg g_1 \Rightarrow g_2]$

$G[\neg g_1 \vee \neg g_2]$



Environment: r_1 is high at time t but low at time (t+1), r_2 is low at time t and (t+1)

Hence, g_1 should be high at time (t+2), by property 1

g_2 should be low at time (t+2), by property 2

Does not contradict property 3.

Environment Does not Win

Is the specification complete?

- Chicken and egg problem
- Formal vs structural coverage
- Look back at:

$G[r_1 \Rightarrow Xg_1 \wedge XXg_1]$

$G[\neg g_1 \Rightarrow g_2]$

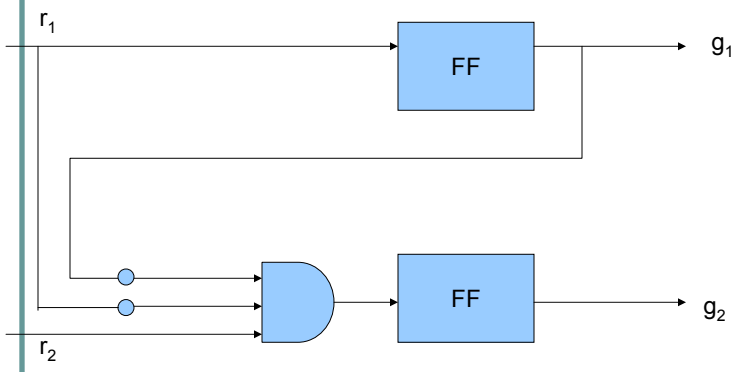
$G[\neg g_1 \vee \neg g_2]$

$G[\neg r_1 \wedge X\neg r_1 \Rightarrow XX\neg g_1]$

Ask the following questions

1. Is g_1 ever high?
2. Is g_2 ever high?
3. Is r_1 required?
4. Is r_2 required?

Design under verification



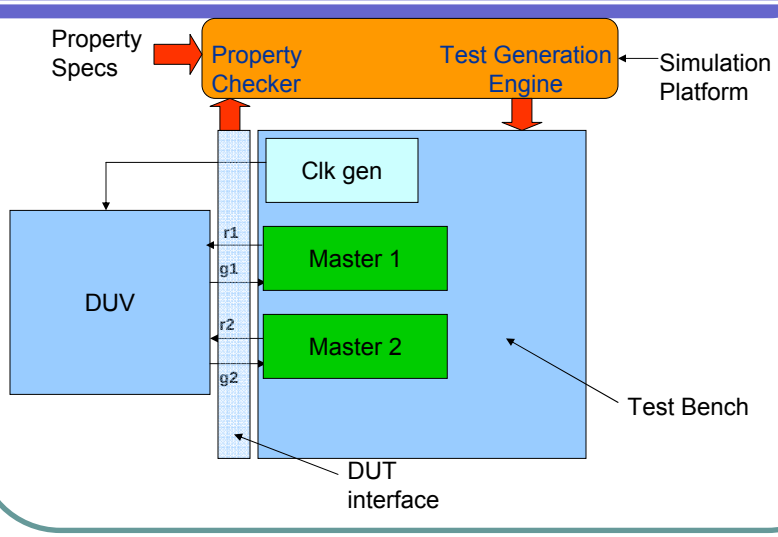
Verilog Code Snipet

```
module arbiter(r1,r2,g1,g2,clk);
  input clk, r1, r2;
  output g1, g2;
  reg g1, g2;
  always @(posedge clk)
  begin
      g2<=r2 & ~r1 & ~g1;
      g1<=r1;
   end
endmodule
```

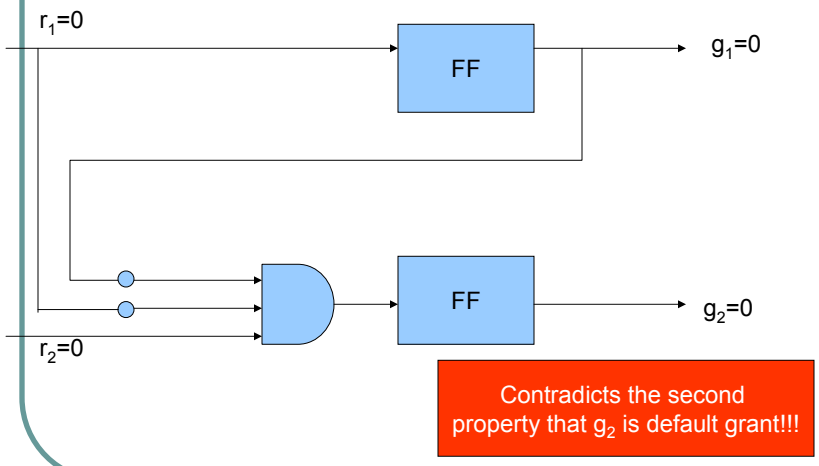
How do you verify??

- Assertion based verification (ABV)
  1. Simulation based verification
  2. More close to the designer (as he has to learn less new techniques)
  3. More close to the old simulation framework
- Formal based verification (FBV)
  1. Formal techniques to verify properties
  2. Mathematical Techniques involved

ABV



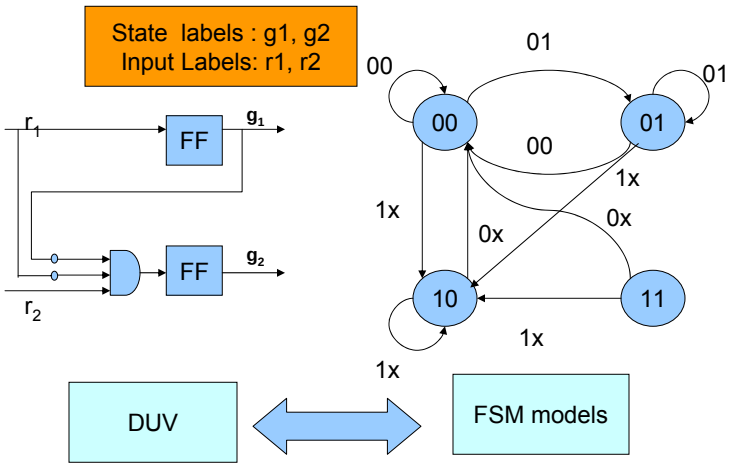
Design under verification



Hurdles of ABV

- Generating the test cases which lead to all the scenarios
- Directed Testing vs Randomized Testing
- We shall see one such language, called “e” in this course

FBV (FSM Extraction)



FBV (contd.)