Verification

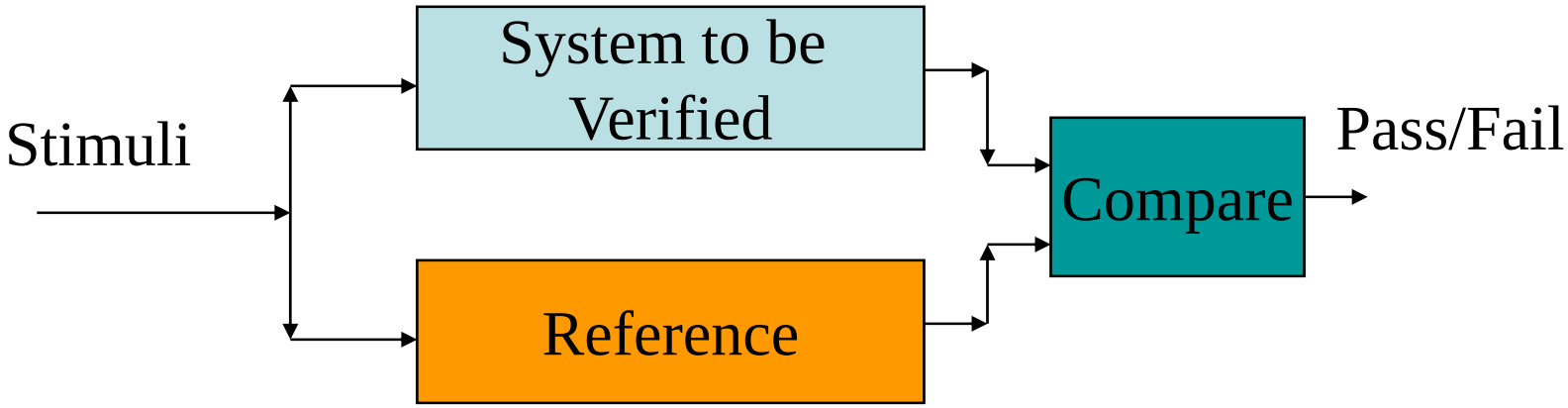
- Simulation is the main method for verifying designs
- Inspecting waveforms is difficult and error-prone
- The principle of Assertion Based Verification is:
  - Describe what you expect to see, and when you expect to see it
  - The simulator will tell you if this doesn’t happen

Writing data

- Verilog has simple output tasks
  - $display - writes data
  - $strobe - writes data after all events have been processed
  - $monitor - continuously monitors variables
- Use formatting commands like C
  - $display("Output = %b", q);

Condition

- Compare with reference



- Reference could be behavioural (simulation) model
- ABV combines reference and comparison

Basic Assertions

```
always @(a, b)
   assert(a == b);
```

- Effectively the same as:

```
always @(a, b)
   if (a != b)
     $strobe("Error a != b");
```

Property Languages

- It is possible to write more complex assertions in Verilog
  - Open Verification Library (OVL)
- Specialised languages are appearing:
  - e
  - OpenVera
  - PSL
  - SystemVerilog Assertions

What is a Property?

- A fact about your design
- In the HDL code or in a separate file
- Properties are used to write
  - Simulation checkers
  - Constraints defining legal simulation inputs
  - Assertions to be proved by static formal property checkers
  - Assumptions made by static formal tools
  - Functional coverage points

Properties and Assertions

- We talk about properties
  - A property is a Boolean expression

```
property equal_ops;
   (out_s == out_b);
endproperty
```

- An assertion is a statement about when that property is true and thus the tool should prove it

```
assert property (equal_ops);
```

Temporal Properties

- So far, no better than Verilog…
- Want to write properties that hold over time:

```
property (@(posedge clk) a |=> b );
```

  b is true in the *cycle* after a is true

- Or we can write sequences:

```
a ##1 b ##1 c |=> d
```

  ##1 means one clock cycle

- `a ##[1:4] b` one to four clock cycles

Complex Properties

- So, it's possible to write complex properties like
  - "If a is true for 3 cycles, b will be true for at least 2 cycles, not more than 2 cycles later"
- In other words, this is the reference model against which we check the behaviour of the system
- As noted, this reference model can be used in simulations and by model checkers

How do we know the property is correct?

- We don't!
- The properties have to be written and debugged at the same time as the HDL model
- But not by the same engineer

Coverage

- A Verilog simulator can measure how often each line is executed (if ever)
  - Known as coverage
  - Built into ModelSim
- Can also measure property coverage
  - How often is each property tested in a simulation?
  - Gives us a measure of how good our simulation is

Including Properties

- As noted, there are 4 property languages and OVL
  - In OVL, assertions are components:

    assert_always a0 (clk, reset_n, out_s==out_b);
- SystemVerilog is a superset of Verilog with an assertion language built in
- Other languages (e.g. PSL) require properties to be in comments or in separate files.

Hardware Verification Languages

- HVLs include e and OpenVera
- Can write properties and assertions
- Also test vectors
  - Thinking of suitable test vectors is hard work
  - Generating random vectors is easy
    - But some random combinations are nonsense
- HVLs allow constrained random vectors
  - Only certain meaningful combinations are allowed
  - Can do this in Verilog, but it's hard work

HVLs

- Current (leading) practice
  - Use HDLs (VHDL/Verilog) for design
  - Use HVLs for testbench, including assertions and constrained random test generation
- SystemVerilog includes many HVL features
  - The way forward?