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
SystemVerilog Assertions

• Introduction – not a full tutorial.
  – There's too much to cover.
• SVAs can be used with VHDL/Verilog models and testbenches.
Example – Lift Controller

• Toy lift
  – Three floors
    • Sensors – top, bottom, middle_plus, middle_minus
    • Call buttons
    • Indicator lamps
  – All signals active high
  – Rising edge of clock is active (posedge in Verilog)
  – direction – 1 is up, 0 is down
  – seven states
An invariant is a property that is always true:

```plaintext
always @*
    assert (!top || !bottom)
    else $error("At top and bottom!");
```

- `@*` is a default sensitivity list
- assertion is evaluated whenever top or bottom changes
- Assert that one or both of top and bottom is true
  - i.e. that both cannot be 1 at the same time
- Is this really useful?
  - Better to synchronise with clock!
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
Property

property NotIn2Places;
    (!top || !bottom);
endproperty

• Assert property at clock edge
assert property ( @(posedge clock) NotIn2Places);
Temporal Properties

- So far, no better than Verilog…
- Want to write properties that hold over time:
  ```verilog
  property (@(posedge clk) a |=> b );
  b is true in the cycle after a is true
  ```
- Or we can write sequences:
  ```verilog
  a ##1 b ##1 c |=> d
  ##1 means one clock cycle
  ```
- ```verilog
  a ##[1:4] b  one to four clock cycles
  ```
Temporal Property

• If the lift is at the bottom and the call1 button is pressed, we expect that the lift will go up in the next clock cycle

```
property Call1;
    (bottom && call1 |=>
        enable && direction);
endproperty
```

• $\Rightarrow$ is non-overlapping implication
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
Clocking

• All assertions are tested on rising edge of the clock.
  – We can make this a default condition

```text
default clocking clock_block @ (posedge clock);
endclocking
```

• Hence

```text
assert property (Call1);
```
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
Coverage

• How many times is the property checked?

```java
cover property (Call1);
```

• Just because an assertion does not fail, doesn't mean that it's ever been tested!

• If `(bottom && call1)` is never true, the assertion passes *vacuously*
Overlapping implication

- If the ground floor indicator is lit, the lift should be going down

```vhdl
property Indicator0;
    (indicator0 |-> !direction);
endproperty
```

- `|->` overlapping implication – same clock cycle
Sequences

- If the top indicator is 0 and then goes to 1, the lift should be at the top in the next clock cycle

```property GetTo2;
  (indicator2 ###1 !indicator2 |=> top);
endproperty```

- ## refers to the clocking method
- Could use an overlapping implication here!
Liveness

property Eventually2;
    (bottom && call2 |-> ##[1:$] top);
endproperty

• $[1:$] means a range of 1 to infinite number of clock cycles
• If the lift is at the bottom and the top call button is pressed, the lift gets to the top floor eventually
Liveness & Safety

• "Liveness" means that good things happen eventually
• "Safety" means that bad things never happen
• Liveness properties are not a good idea
  – What does eventually mean?
  – Properties can pass vacuously, so how can we know it's failed?
States

**property** StateChange;

\[(state == \text{floor0}) \&\& \text{call1} \Rightarrow (state == \text{upto1});\]

**endproperty**

- Mapping states is a little convoluted, but possible!
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