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

Invariant

- An invariant is a property that is always true:

```
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 false
  – i.e. that both cannot be 1 at the same time
- Is this really useful?
  – Better to synchronise with clock!

Property

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

- Assert property at clock edge

```
assert property (
 @(posedge clock) NotIn2Places);
```

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
```

- `|=>` is non-overlapping implication

Clocking

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

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

- **Hence**

```
assert property (Call1);
```

Coverage

- How many times is the property checked?

```
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

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

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

Sequences

- If the top indicator is 1 and then goes to 0, 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 == floor0) && call1
   |=> (state == upto1);
endproperty
```

- Mapping states is a little convoluted, but possible!

Illegal Transition

```
property BadStateChange;
   (state == floor0) |->
   $past(state != upto1);
endproperty
```

- If we are now in one state, we can't have come from another