VERILOG 4:

FOUR COMMON MISTAKES

Common Mistake #1a: `assign` in an `always` block

- There is no "assign" keyword in always blocks!
- The purpose of an always block → to define a *reg*
- See the slide titled Describing Hardware in the *Verilog 2* Handout

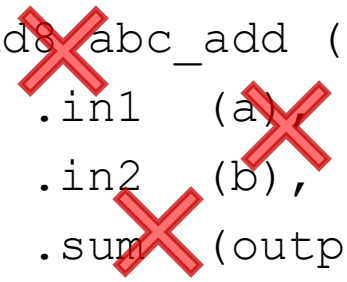
```
wire out2;
reg  out;

always @(a or b) begin
   out = a & b;
   assign out2 = a ^ b;
end
```

Common Mistake #1b: module instantiation in an `always` block

- There are no module instantiations in always blocks!
- The purpose of an always block → to define a reg
- See the slide titled Describing Hardware in the *Verilog 2* Handout

```
reg out;
always @(a or b) begin
   out = a & b;
   add8 abc_add (
      .in1   (a),
      .in2   (b),
      .sum   (output) );
end
```



Common Mistake #2: Setting the same **`reg`** in multiple **`always`** blocks

- Simulators will typically do what you tell them to do
- Synthesis tools and lint checkers will typically give a warning
- Never do this in Hardware verilog
- You might be able to get away with it in testing verilog but don't do it

```
module

  always @(a or b) begin
    x = a ^ b;
  end

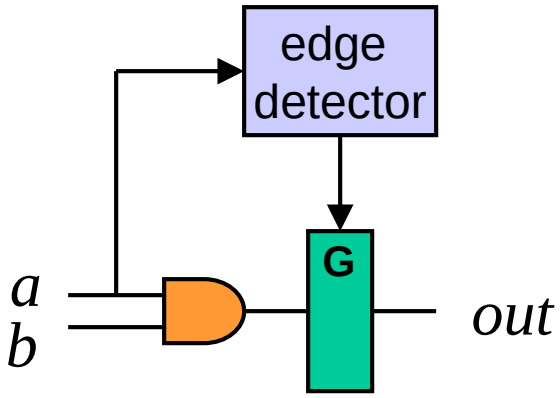
  always @(reset) begin
    x = 1'b0;
  end

endmodule
```

Mistake #3: Inferring State In Combinational Circuits By: An Incomplete Sensitivity List

- ```
  always @(a) begin  // missing b
     out = a & b;
  end
  ```
- *out* updates when *a* changes as expected
- *out* *does not update* when *b* changes!
  - Put another way, *b* can change all it wants but *out* will not update—this requires a memory element to remember the last value of *out*
- Synthesis tools and lint checkers will give a warning
- Using the construct new in Verilog 1364-2001:

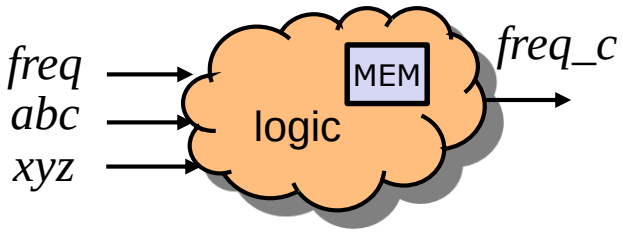
  ```
  always @(*)
  ```
  or
  ```
  always @*
  ```
  eliminates this type of bug, but it is not supported by all modern CAD tools


  ```

Mistake #4: Inferring State In Combinational Circuits By: Not Setting a *reg* In All Paths

```
// this "always" block does not
// instantiate combinational logic
always @(freq or xyz or abc) begin
  if (xyz == 4'b0010) begin
    freq_c = abc;
  end

  case (freq) begin
    3'b000:  freq_c = abc;
    3'b001:  freq_c = abc + 3'b001;
  endcase
end
```



- Example: attempted combinational circuit
  - If `xyz==4'b0010`, *freq_c* is a function of the inputs
  - If `freq==3'b000` or `3'b001`, *freq_c* is a function of the inputs
  - Otherwise, *freq_c* keeps its old value—which implies memory is needed! Which means our combinational circuit is broken.
  - A failure occurs when the inputs of a statement are different from the signals used for the **`if`** or **`case`** condition

Mistake #4: Inferring State In Combinational Circuits By: Not Setting a reg In All Paths

Fixes all

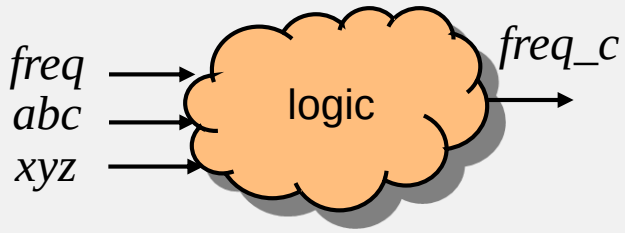
Fixes the *if* block

Fixes the *case* block

```
// this "always" block does instantiate combinational logic
always @(freq or xyz or abc) begin
  // "default" section of always block
  freq_c = freq;   // some default value

  // main logic block
  if (xyz==4'b0010) begin
    freq_c = abc;
  end
  else begin   // perhaps not always applicable
    freq_c = ...;
  end

  case (freq) begin
    3'b000:  freq_c = abc;
    3'b001:  freq_c = abc+3'b001;
    default: freq_c = ...;     // perhaps you know freq should never be anything
  endcase                      // other than 000 or 001, or perhaps you do want
end                            // freq_c equal to something in these six cases
```



- Example: successful combinational circuit
  - Solution: *freq_c* is set regardless of the values of all inputs
  - One solution: always declare default values at beginning of *always* blocks
  - If helpful, declare default case in case statements

For Combinational Circuits: Avoid Inferring State by:

1. Including **all input** variables in the sensitivity list!
   - Use **`always @(*)`**
2. Set the value of all regs in **all** paths through every `always` block
   - A nice solution is to set default values for all output variables immediately after entering an `always` block
     - You will eliminate the chance of this bug if you do