Chapter 3
RTL Modeling Gotchas

Gotcha 21: Combinational logic sensitivity lists with function calls

Gotcha: My combinational logic seemed to simulate OK, but after synthesis, the gate-level simulation does not match the RTL simulation.

Synopsis: If combinational logic calls a function, then the combinational sensitivity list must include signals that the function reads. @* does not infer sensitivity to values read by functions called from combinational logic.

Synthesizable RTL modeling style requires that Verilog always procedural blocks have an edge sensitive timing control (the @ token) following the always keyword. This time control is referred to as the block's sensitivity list.

always @(a, b) begin
    sum = a + b;
end

always @(a, b) begin
    prod = mult(a, b);
end

always @(a, b) begin
    out = sel? sum: prod; // missing sel
end

function [15:0] mult (input [7:0] m, n);
    mult = m * n;
endfunction

Note: the code examples in this chapter are contrived in order to illustrate each gotcha using small examples. In real design and verification code, these gotchas might not be as obvious or easy to debug.
When modeling combinational logic, if the sensitivity list is not complete, then the outputs of the block will not be updated for all possible input changes. This behavior models a latch in simulation. However, synthesis will assume a complete sensitivity list and build combinational logic instead of a latch. The simulation results of the RTL model and the synthesized gate-level model will not match. Gotcha!

In the simple examples above, it is easy to manually code a complete sensitivity list, and to see if something is missing. Real designs are not always that simple. A complex decoder, for example, could read several dozen signals, each and every one of which must be listed in the sensitivity list. A very common coding gotcha occurs when a designer, in the process of implementing a design, adds another statement to the complex decode logic that reads an additional variable, and forgets to add that additional signal to the sensitivity list. The functional problem that results can be very difficult to detect and debug. Gotcha!

How to avoid this Gotcha using Verilog

Verilog has an @ wildcardsensitivity list that infers a complete sensitivity list for both simulation and synthesis—most of the time. The @ wildcard will automatically be sensitive to any nets or variables that are read in the always procedural block, including any nets or variables that are passed to a function input. Using @ will fix the gotcha in the example above (either @ or @(*) can be used; they are equivalent).

```
always @* begin
  sum = a + b;
end
```

```
always @* begin
  prod = mult(a, b);
  // call function that reads a, b
end
```

```
always @* begin
  out = sel? sum: prod ;
end
```

```
function [15:0] mult (input [7:0] m, n);
  mult = m * n;
endfunction
```

However, @ has a subtle gotcha that is not widely known. It only infers sensitivity to signals directly referenced in the always block. It will not infer sensitivity to signals that are externally referenced in a function that is called from the always block. That is, the @ will only be sensitive to the signals passed into the function or task. The following example illustrates this gotcha: