Chapter 4 :: Hardware Description Languages

Digital Design and Computer Architecture

David Money Harris and Sarah L. Harris
Other Behavioral Statements

• Statements that must be inside *always* statements:
  - *if/else*
  - *case, casez*

• *Reminder:* Variables assigned in an *always* statement must be declared as *reg* (even if they’re not actually registered!)
Combinational Logic using `always`

```verilog
// combinational logic using an always statement
module gates(input [3:0] a, b,
             output reg [3:0] y1, y2, y3, y4, y5);
    always @(*) // need begin/end because there is
     begin       // more than one statement in always
        y1 = a & b; // AND
        y2 = a | b; // OR
        y3 = a ^ b; // XOR
        y4 = ~(a & b); // NAND
        y5 = ~(a | b); // NOR
    end
endmodule
```

This hardware could be described with assign statements using fewer lines of code, so it’s better to use assign statements in this case.
module sevenseg(input [3:0] data,
               output reg [6:0] segments);

always @(*)
  case (data)
    //                abc_defg
    0: segments = 7'b111_1110;
    1: segments = 7'b011_0000;
    2: segments = 7'b110_1101;
    3: segments = 7'b111_1001;
    4: segments = 7'b011_0011;
    5: segments = 7'b101_1011;
    6: segments = 7'b101_1111;
    7: segments = 7'b111_0000;
    8: segments = 7'b111_1111;
    9: segments = 7'b111_1011;
    default: segments = 7'b000_0000; // required
  endcase
endmodule
Combinational Logic using `case`

- In order for a `case` statement to imply combinational logic, all possible input combinations must be described by the HDL.
- Remember to use a `default` statement when necessary.
Combinational Logic using `casez`

```verilog
module priority_casez(input [3:0] a, 
                      output reg [3:0] y);

always @(*)
casez(a)
  4'b1???: y = 4'b1000; // ? = don’t care
  4'b01??: y = 4'b0100;
  4'b001?: y = 4'b0010;
  4'b0001: y = 4'b0001;
  default: y = 4'b0000;
endcase

endmodule
```
Blocking vs. Nonblocking Assignments

- <= is a “nonblocking assignment”
  - Occurs simultaneously with others
- = is a “blocking assignment”
  - Occurs in the order it appears in the file

```verilog
// Good synchronizer using nonblocking assignments
module syncgood(input clk, input d, output reg q);
reg n1;
always @(posedge clk)
beg
  n1 <= d; // nonblocking
  q <= n1; // nonblocking
end
dendmodule

// Bad synchronizer using blocking assignments
module synccbad(input clk, input d, output reg q);
reg n1;
always @(posedge clk)
beg
  n1 = d; // blocking
  q = n1; // blocking
end
dendmodule
```
Rules for Signal Assignment

- Use `always @ (posedge clk)` and nonblocking assignments (`<=`) to model synchronous sequential logic:
  ```verilog
classic always @ (posedge clk)  
q <= d;  // nonblocking
```

- Use continuous assignments (`assign ...`) to model simple combinational logic:
  ```verilog
assign y = a & b;
```

- Use `always @ (*)` and blocking assignments (`=`) to model more complicated combinational logic where the `always` statement is helpful.

- Do not make assignments to the same signal in more than one `always` statement or continuous assignment statement.
Finite State Machines (FSMs)

- Three blocks:
  - next state logic
  - state register
  - output logic
FSM Example: Divide by 3

S0
S1
S2

The double circle indicates the reset state
module divideby3FSM (input  clk,
             input  reset,
             output q);
    reg  [1:0] state, nextstate;

    parameter S0 = 2'b00;
    parameter S1 = 2'b01;
    parameter S2 = 2'b10;

    // state register
    always @ (posedge clk, posedge reset)
      if (reset) state <= S0;
      else       state <= nextstate;

    // next state logic
    always @ (*)
      case (state)
        S0:      nextstate = S1;
        S1:      nextstate = S2;
        S2:      nextstate = S0;
        default: nextstate = S0;
      endcase

    // output logic
    assign q = (state == S0);
endmodule
2:1 mux:

module mux2
    #(parameter width = 8) // name and default value
    (input  [width-1:0] d0, d1,
     input             s,
     output  [width-1:0] y);
    assign y = s ? d1 : d0;
endmodule

Instance with 8-bit bus width (uses default):
    mux2 mux1(d0, d1, s, out);

Instance with 12-bit bus width:
    mux2 #(12) lowmux(d0, d1, s, out);