Verilog Procedural Statements

Verilog Procedural Statements

```
always @(sensitivity_list)
[begin]
    [procedural assignment statements]
    [if-else statements]
    [case statements]
    [while, repeat, and for loops]
    [task and function calls]
[end]
```

Verilog syntax requires that procedural statements be contained inside a construct called an **always** block.

About Always block

- An always block represents a hardware block in HW.

- A typical Verilog design module may include several always blocks
  - each representing a part of the circuit being modeled.
  - all the always blocks are concurrent with respect to one another.

```
always @ (sensitivity list)
      statement;
```

Whenever the event in the sensitivity list occurs, the statement is executed

Always block Sensitivity List

- The part of the always block after the @symbol, in parentheses, is called the sensitivity list.

- The sensitivity list simply tells the Verilog compiler which input signals can directly affect the outputs produced by the always block.

```
always @(x1 or x2 or s)
    if (s == 0)
        f = x1;
    else
        f = x2;
```

- More simply can be written as, always @(*)
  - Tells the compiler to include all input signals in the sensitivity list

Net and Variable Types

- Nets provide a means for interconnecting logic elements, but they do not allow a circuit to be described in terms of its behavior.

- For modeling behavior, Verilog provides ***Variables***

- Two types of variables:
  1. reg
     - the keyword reg does not denote a storage element, or register in hardware!
  2. Integer
     - mostly used as control variables e.g loop index
     - do not directly correspond to nodes in a circuit!

  – Any signal assigned a value inside an always block has to be a variable of type **reg** or **integer**

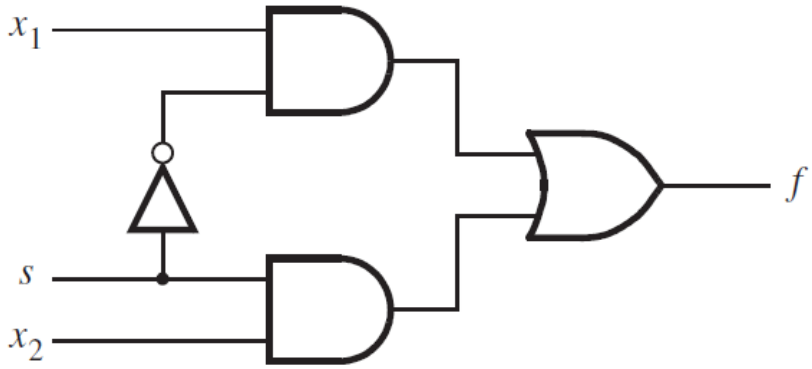
Procedural Statement: If-Else

```
// Behavioral specification
module example5 (x1, x2, s, f);
    input x1, x2, s;
    output f;
    reg f;

    always @(x1 or x2 or s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```

output reg f;



If a signal is assigned a value using procedural statements, Then Verilog syntax requires that it be declared as a variable

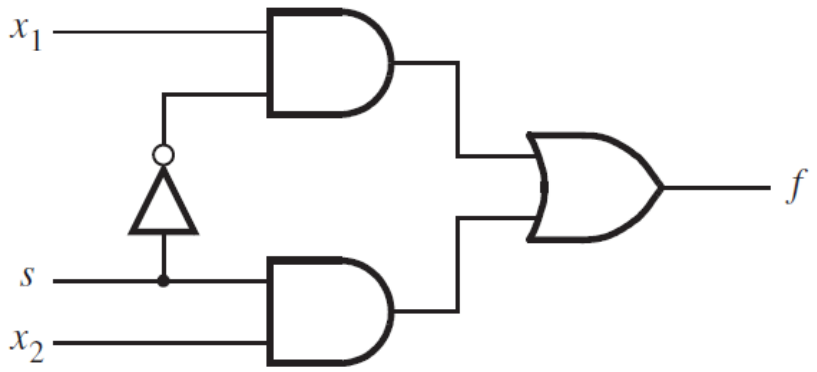
Procedural Statement: If-Else

Alternate style: Note the signal declaration inside the port-list

```
// Behavioral specification
module  example5 (input x1, x2, s, output reg f);

    always @(x1, x2, s)
        if (s == 0)
            f = x1;
        else
            f = x2;

endmodule
```



Procedural Statement: The Case

```
case  (expression)
    alternative1: begin
                      statement;
                  end
    alternative2: begin
                      statement;
                  end
    [default:     begin
                      statement;
                  end]
endcase
```

- The bits in expression, called the controlling expression, are checked for a match with each alternative.

Case: Example

```
module mux (w0, w1, s, f);
  input w0, w1, s;
  output reg f;

  always @(w0, w1, s)
    case (s)
      1'b0: f = w0;
      1'b1: f = w1;
endcase
endmodule
```