System Verilog Changes

Data Types:

SystemVerilog introduced the *logic* data type to replace Verilog’s *reg* data type. In Verilog, if a signal appears on the left hand side of a nonblocking or blocking ( <= or = ) assignment inside of an *always* block, it must be declared as a *reg*. Otherwise, it should be declared as a *wire*. A *reg* signal is typically the output of a flip-flop, a latch, or combinational logic that appears in an *always @(*) block. If a specific data type is not declared, a signal defaults to *wire*.

In the following example, *clk* and *d* are of type *wire* and *q* is explicitly declared as *reg* because it is assigned a value inside of an *always* block.

```verilog
module flop( input clk, input [3:0] d, output reg [3:0] q);
    always @(posedge clk )
        q <= d;
endmodule
```

Always Blocks:

In SystemVerilog, two new *always* blocks were introduced, *always_ff* and *always_comb*. This change was intended to increase clarity by explicitly saying whether the *always* block was sequential logic or combinational logic. In Verilog, only one *always* block exists. To distinguish between synchronous sequential logic and combinational logic in Verilog, the sensitivity list is used. In sequential logic, the signals that trigger the *always* block are put in the sensitivity list. The example located above has *posedge clk* in the sensitivity list. To model combinational logic, the sensitivity list is set to trigger when any signals change with @(*).

Use *always @(posedge clk)* and nonblocking assignments to model synchronous sequential logic.

```verilog
always @(posedge clk )
    begin
        n1 <= d;
        q <= n1;
    end
```
Use *always @(*)* and blocking assignments to model complicated combinational logic. For simple combinational logic, *assign* statements may be used.

```verilog
classic always @(*)
    begin
        p = a ^ b;
        g = a & b;
        s = p ^ cin;
        cout = g | (p & cin);
    end
```

More information on SystemVerilog can be found in Section 4.7.1 of *Digital Design and Computer Architecture*.

---

**HDL Examples:**

**Example 4.5 2:1 Multiplexer**

```verilog
module mux2( input [3:0] d0,
             input [3:0] d1,
             input s,
             output [3:0] y );
    assign y = s ? d1 : d0;
endmodule
```
Example 4.31 Pattern Recognizer Moore Finite State Machine

module patternMoore(  
  input clk,  
  input reset,  
  input a,  
  output y );

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

reg [2:0] state;  
reg [2:0] nextstate;

// state register.  
// reset in the sensitivity list makes an asynchronous reset.  
always @(posedge clk, posedge reset)  
begin  
  if( reset ) state <= S0;  
  else state <= nextstate;
end

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

  // output logic  
  assign y = ( state == S2 );
endmodule
Example 4.30 Divide by 3 Finite State Machine

module divideby3( input clk,
                   input reset,
                   output y );

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

reg [2:0] state;
reg [2:0] nextstate;

// state register.
// reset in the sensitivity list makes an asynchronous reset.
always @(posedge clk, posedge reset)
begin
  if (reset) state <= S0;
  else           state <= nextstate;
end

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

// output logic
assign y = (state == S0);
endmodule
Example 4.24 Seven-Segment Display Decoder

module sevenseg( input [3:0] data,
                  output reg [6:0] segments );

always @(*)
  case( data ) // 7'bABCDEFG
    0:    segments = 7'b1111110;
    1:    segments = 7'b0110000;
    2:    segments = 7'b1101101;
    3:    segments = 7'b1111001;
    4:    segments = 7'b0110011;
    5:    segments = 7'b1011011;
    6:    segments = 7'b1011111;
    7:    segments = 7'b1110000;
    8:    segments = 7'b1111111;
    9:    segments = 7'b1110011;
    default: segments = 7'b0000000;
  endcase

endmodule