HDL Coding (Verilog) for FPGA Design

Hardware Description Language and Design Flow

Start

HDL Coding

Test benches

Simulations: Fail?

Synthesis

APR
Example: 2 to 1 Multiplexer

\[
\text{if } (\text{select} \equiv 0) \quad \text{then}
\]
\[\text{out} = x; \]
\[\text{else}
\]
\[\text{out} = y;\]

Three main Verilog Styles that we are concerned with

→ RTL

Behavioral

Structural

module sample (\(\overline{A}, B, C, D, \overline{out}\));
input \(A, B, C, D\);
output \(\overline{out}\);
wire \(E\);
\[\text{UCAND A1} (\overline{A}(A), \overline{B}(E), \overline{C}(B), D(\overline{out}));\]
\[\text{UCXOR X1} (\overline{A}(C), B(D), Z(E));\]
endmodule
module SampleRTL(A, B, C, D, out);
    input A, B, C, D;
    output out;
    reg out;
    always @ (A or B or C or D)
    begin
        if (A & B & ¬D) 
            out = C;
        else if (A & D & ¬C)
            out = B;
        else
            out = 0;
    end
endmodule
module sample 2 (a, out, clk);
  output out;
  input a, clk;
  reg e1, e2, out;
  always @(posedge clk)
    begin
      out <= e2;
      e2 <= e1;
      e1 <= a;
    end
endmodule

module sample-structural (a, out, clk);
  output out;
  input a, clk; wire one;
  DF1 instance1 (D(a), @one, C(clk));
  DF2 instance2 (D(one), @two, C( clk) );
  instance3 (               )
Port Types
1-input  2-output  3-inout

Designing a module

module YourName (A,B);
    input Ai;
    output B;
endmodule

wire x;

Remark: "input" and "output" port types are automatically defined as wires. So, "input wire x", while still correct, is redundant.
wire a;
wire [2:0] a;

\[ \text{Definition of vectors} \]

\[ \text{tmp} = 3'b001; \quad \text{tmp}[0] = 1 \]
\[ \text{tmp}[1] = 0 \]
\[ \text{tmp}[2] = 0 \]

→ Utilizing vectors

→ Remark [reg] → a node in the circuit

→ [integer] → used for loop counters only.

→ Remark: "reg" does NOT necessarily correspond to a register.

\[ \text{reg C;} \]
\[ \text{always @ (A, B)} \]
\[ C = A + B; \]

\[ \text{reg C always @ (posedge clk)} \]
\[ C \leftarrow A + B \]
module sample...

-- input A;
-- wire B;
-- assign B = A;

→ Remark: [assign] → Assigning using “assign”

Assign using “blocking =”
ls outside always block

always block for

Assign using “nonblocking <=”
ls for sequential always

→ Remark: Naming your Signals

Digits, letters, → $  

① should not ? → 25 : is NOT allowed
Start with
Digit

② It should not be a verilog

③ Verilog is case sensitive.
Values:

0, 1, \(X\): unknown value, \(Z\): Tri-State (high impedance)

Value Assignments

```
reg [7:0] sample;
Sample = 8'h09;
Sample[7]
```

h: hexadecimal

```
Sample = 8'd3;
Sample'00000011
```

d: decimal

```
"b": binary
"octal": "0"
```


wire [7:0] sample;
wire [15:0] b;

assign d = sample [6:5]
Remark: [parameter] is very important in designing FSM module.

module sample;
    input [n-1:0] si;
    parameter n = 3;

Memory
Sample: define 10 byte memory:

    reg [7:0] M [9:0];

1D memory

    reg [7:0] M [9:0][2:0];

2D memory

    A = M[3][0]
Operations

Bitwise

\[1001 \&\ 1000 = 1000\]
\[1001 \mid\ 1000 = 1001\]
\[\sim 1001 = 0110\]
\[1001 \land 1000 = 0001\]

Logical Operators

\[1000 \&\ 1001 = 1\]
\[1000 \mid\ 1001 = 1\]
\[\neg 0010 = 0\]

Reduction

\[\&\ 1100 = 0\]
\[\&\ 11111 = 1\]
\[\land\ 0100 = 1\]
Relational

\[ A = 2'b11; \]

\[(A == 2'b01) \sim 0\]

\[(A == 2'b11) \sim 1\]

Remark: "==" which is used in case of \( x \) and \( z \) values

\[ B = 2'b10 \]

\[ B \leq A \Rightarrow 1 \]
Concatenation

say \( A = 3\,\!b\,101 \) and \( B = 2\,\!b\,01 \);

\[
\begin{align*}
\{ A, B \} &= 5\,\!b\,101\,101 \\
A & \leftarrow \sim B
\end{align*}
\]

\[
\{ 2\,\!b\,A, 2\,\!b\,B \} = 6\,\!b\,101\,101
\]
Logical Shift

\[ A = 9\_b\overbrace{010000100} \]

\[ A \gg 2^{\overbrace{1001001}} \]

Diagram:

\[ D = S \? B : C \]

\[ C \rightarrow D \]

\[ B \rightarrow S \]
Procedural & Concurrent Verilog Statements

```verilog
always @ (a, y)
begin
    s = a & y;
    c = a & y;
end

case is very useful for FSM
```