Lecture 16

Synthesizable Verilog: A cheat sheet

Comments

```
//          in-line comment
/*  */      multi-line block comment
```

Constants

```
4'b1010     4-bit binary number with value 1010
8'b1010     8-bit binary number with value 00001010
```

Vector

```
wire [8:0]  8-bit vector LSB is rightmost bit
x[4:2]      3-bit sub-vector LSB with x[4] as MSB and x[2] as LSB
```

Concatenation

```
{3'b10, x[5:4]}   4-bit vector with 10 in MSB and bits 5 and 4 from vector x
```

Logic Function

```
x & y       AND
x | y       OR
x ^ y       XOR
~x          NOT
```

Assignment

```
assign x = 4'b0101;
assign x = y & z;
assign x = y[3] ^ z;
```

Combinational

```
always @(*)
      case (x)
            3'b001:        action_for_x_equal_001;
            3'b010:        action_for_x_equal_010;
            3'b100:        action_for_x_equal_100;
            default:       default_action;
      endcase
```

Module declaration `module moduleName (arg1, arg2, … , argN);`

`module`	is a keyword
`moduleName`	your name for the module
`arg1`	name of an input of output argument

Module arguments - You can use the argument names in the module. An input argument should never be assigned a value and an you should not read the value of an output argument.

`input wire [L-1:0] arg1;`	input to module, wire is type, optional vector length, name
`output wire[M-1:0] arg2;`	output from module, wire is type, optional vector length, name
`output reg [N-1:0] arg3;`	output from module, reg is type, optional vector length, name

Wire vs. Reg

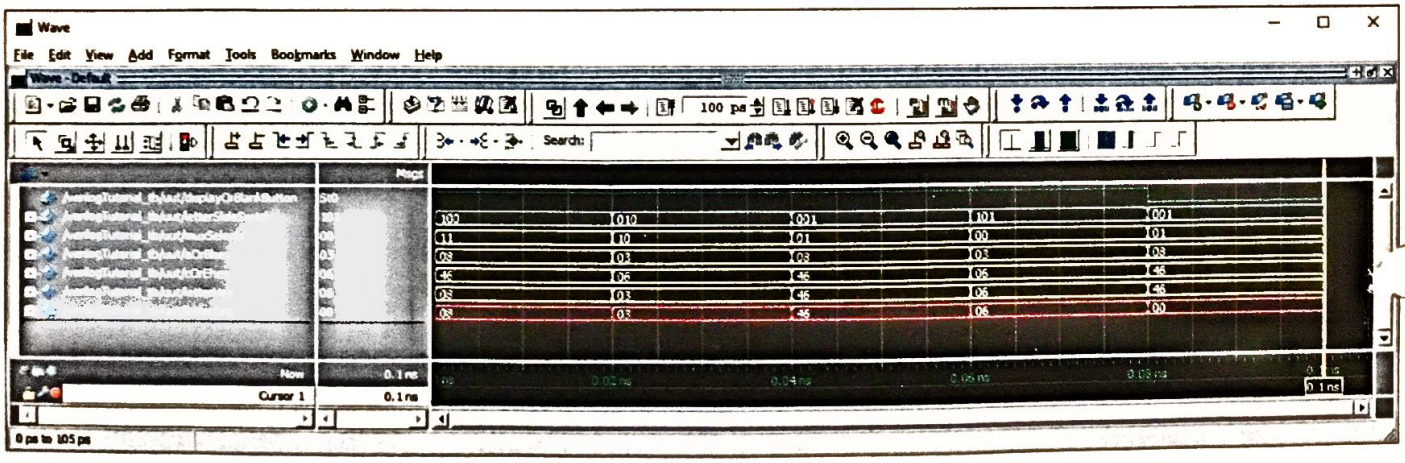
`wire`	For signals that are driven from Logic Function output or a module output
`reg`	For signals are driven by an always/case statement

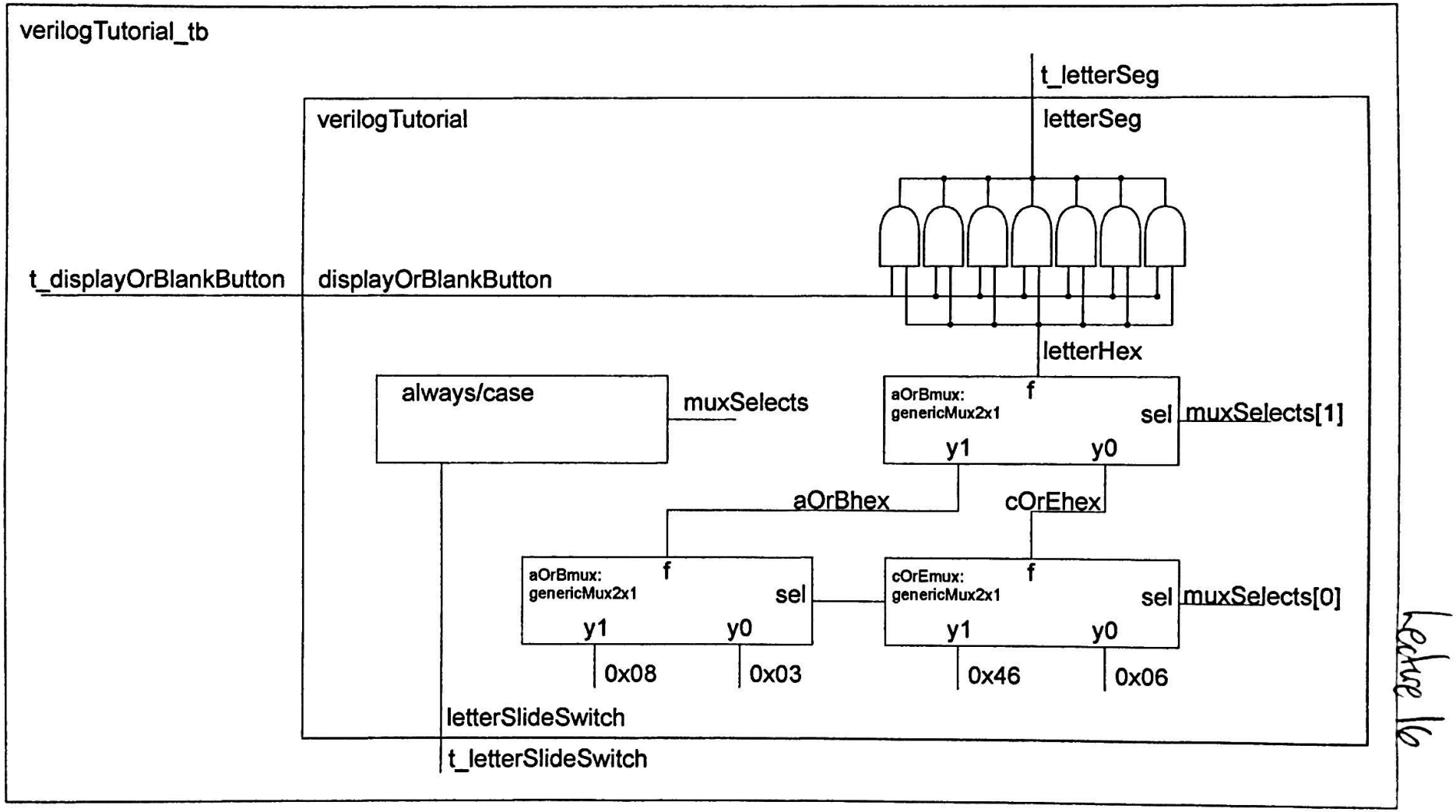
Module instantiation `moduleName #(8) instanceName(arg1, arg2, … , argN);`

`moduleName`	the module name that you want to instantiate
`#(8)`	the value of a generic parameter – depends on module
`instanceName`	a unique name for that module instance
`arg1`	signal from enclosing context that goes in or out of module

```
module verilogTutorial(letterSlideSwitch, displayOrBlankButton, letterSeg);

    input  wire        displayOrBlankButton;        // Ins are always wires
    input  wire  [2:0] letterSlideSwitch;
    output wire  [6:0] letterSeg;                   // Out are wire or reg

    // Signals inside the module never have "input" or "output"
    // Make signals "reg" when they output of an always/case statement
    // Make signals "reg" when they output of a module or logic
    reg          [1:0] muxSelects;
    reg          [6:0] letterA = 7'b0001000;
    wire         [6:0] aOrBhex, cOrEhex, letterHex;

    // If you need multiple statement for a case, use begin/end
    // The assigned value in an always/case must be "reg" type
    always @(*)
        case (letterSlideSwitch)
            4'b100: muxSelects = 2'b11;             // 'A'
            4'b010: muxSelects = 2'b10;             // 'b'
            4'b001: muxSelects = 2'b01;             // 'c'
            default: muxSelects = 2'b00;            // 'E'
        endcase

    // Instantation of generics requires bus size #(7) in this case
    genericMux2x1 #(7) aOrBmux(letterA , 7'h03, muxSelects[0], aOrBhex);
    genericMux2x1 #(7) cOrEmux(7'h46   , 7'h06, muxSelects[0], cOrEhex);

    // The outputs from instantations (letterHex) are wire types
    genericMux2x1 #(7) letterMux(aOrBhex, cOrEhex, muxSelects[1], letterHex);

    // Don't forget to use "assign" when you use AND, OR, NOT
    // Outputs from logic gates (letterSeg) are wire types
    assign letterSeg[6] = letterHex[6] & displayOrBlankButton;
    assign letterSeg[5] = letterHex[5] & displayOrBlankButton;
    assign letterSeg[4] = letterHex[4] & displayOrBlankButton;
    assign letterSeg[2] = letterHex[2] & displayOrBlankButton;
    assign letterSeg[1] = letterHex[1] & displayOrBlankButton;
    assign letterSeg[0] = letterHex[0] & displayOrBlankButton;

endmodule
```

```
module verilogTutorial_tb;

    wire [6:0] t_letterSeg;                         // outputs from instantiations are wires
    reg  [2:0] t_letterSlideSwitch;                 // Inputs to UUT are always reg
    reg        t_displayOrBlankButton;

    verilogTutorial uut (t_letterSlideSwitch, t_displayOrBlankButton, t_letterSeg);

initial
    begin
        t_letterSlideSwitch = 3'b100; t_displayOrBlankButton = 1'b1;       #20
        t_letterSlideSwitch = 3'b010; t_displayOrBlankButton = 1'b1;       #20
        t_letterSlideSwitch = 3'b001; t_displayOrBlankButton = 1'b1;       #20
        t_letterSlideSwitch = 3'b101; t_displayOrBlankButton = 1'b1;       #20
        t_letterSlideSwitch = 3'b001; t_displayOrBlankButton = 1'b0;       #20
        t_letterSlideSwitch = 3'b101; t_displayOrBlankButton = 1'b0;
    end
endmodule
```




verilogTutorial_tb
t_letterSeg
verilogTutorial
letterSeg
t_displayOrBlankButton
displayOrBlankButton
letterHex
always/case
muxSelects
aOrBmux:
genericMux2x1
f
sel
muxSelects[1]
y1
y0
aOrBhex
cOrEhex
aOrBmux:
genericMux2x1
f
sel
y1
y0
cOrEmux:
genericMux2x1
f
sel
muxSelects[0]
y1
y0
0x08
0x03
0x46
0x06
letterSlideSwitch
t_letterSlideSwitch