A Very Basic Intro to Verilog

E15A Lab 2
Model a gate with Verilog
Structural Modeling

module Example1 (SW, LEDY_M);
// Structural modeling (i.e., coding with logic gates)
input [1:0] SW;     //Inputs are the two rightmost switches
output LEDY_M;     //Output is middle yellow LED on E15 board
or    Gate1(LEDY_M,SW[0],SW[1]);  //or gate with instance name "Gate1"
endmodule

Anything following “//” is a comment and is ignored by compiler

Input is two bit “vector”. In definition, range of bits goes in square brackets to left of name.

When using bits, index goes in square brackets to right of name.
module cylon(b, c, d, x);  // Implement a 'cylon' circuit
input b, c, d;             // 3 bit input
output [3:0]x;           // 4 bit output (in a vector)

assign x[0] = ~b & ~c & ~d; // create logic for four led's
assign x[1] = ~c & d;       // as determined by karnaugh
assign x[2] = (c & ~d) | (b & ~d);  // maps.
assign x[3] = c & d;
endmodule

“assign” statement necessary for “dataflow” modeling in this lab.

Bitwise Operators
- &  and
- |  or
- ~  not
- ^  xor

Now that we have the “cylon” module design, how do we test it, and how do we implement it?
Testing with simulation
(Described in tutorial)

Define inputs.
Simulation has not been run yet so outputs are undefined

After Simulation
Testing with Switches and LED (real world)

module cySwitch(SW, LEDR);
input [2:0] SW; // SW[2], SW[1] and SW[0] are inputs
output [3:0] LEDR; // LED's are outputs

// instantiate the "cylon" circuit
// inputs are SW[2], SW[1] and SW[0] (b, c, d)
// outputs are the Red LED's (LEDR[3:0])

cylon myCy(SW[2], SW[1], SW[0], LEDR);
endmodule
Numbers in Verilog (1)

- Numbers follow the syntax: \textit{length'radix} val
- length: number of bits
- radix:
  - b binary 0,1,x,z,_
  - d decimal
  - h hexadecimal
  - o octal
- \( 4'b0010 \) 4 bit binary = 0010_2
- \( 8'h3f \) 8 bit hex = 0011 1111_2
- \( 4'd7 \) 4 bit decimal = 0111_2
- \( 8'b0011_0101 \)
Numbers in Verilog (2)

- length is optional, default = 32
- radix is optional, default = decimal
- 7 (0000 0000 0000 0000 0000 0000 0000 0111)
- negative numbers (2’s complement)
  - $-4’d2$ binary=1110; decimal=-2
- $[3:0] \ x$ x is a “vector”
  - x is a number with 4 bits, $x[3], x[2], x[1], x[0]$
module cyCount(CLOCK_50, LEDR);
  input  CLOCK_50;   // 50 MHz clock input
  output [9:0]LEDR;  // LED's show input and output from "cylon"

  wire [3:0] Q;       // output from counter and input to cylon

  assign LEDR[9:7] = Q[2:0];    // output from counter and input to cylon
  assign LEDR[6:4] = 3'b0;      // turn off these LED's

  // Instantiate counter, max count = 5, output to Q.
  E15Counter1HzB myCounter(CLOCK_50, 4'd5, Q);  // count from 0 to 15.

  // "cylon" has input from Q[2:0], output to LEDR[3:0]
  cylon myCy(Q[2], Q[1], Q[0], LEDR[3:0]);
endmodule
Your lab

- Go through tutorial (do “cylon” design in Verilog)
- Design BCD (Binary Coded Decimal) to 7 segment decoded

<table>
<thead>
<tr>
<th>Number (decimal)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number (binary)</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
</tr>
<tr>
<td>Display</td>
<td>a</td>
<td>a</td>
<td>f</td>
<td>b</td>
<td>f</td>
<td>b</td>
<td>f</td>
<td>b</td>
<td>f</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>g</td>
<td>f</td>
<td>b</td>
<td>g</td>
<td>b</td>
<td>g</td>
<td>c</td>
<td>e</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>