

EE 231L

Using Verilog to Design Combinational Circuits

Altera Hardware Description Language (AHDL) is a language developed by Altera Corporation for programming programmable logic devices (PLDs). There are many commercial HDLs available which can be used to program PLDs (VHDL and Verilog HDL are two industry standards). In this lab you will learn how to design digital logic circuits using Verilog.

With Verilog, there are two methods for entering designs – a text design file (.v) or a block diagram schematic file (.bdf). Using the graphical design method you enter circuits as you would draw them on a schematic. This method is quite easy to learn and can be used for simple circuits, but becomes very cumbersome for large design. The text design method is flexible, and much easier to use for large designs. We will show several examples using the graphical design method, but will concentrate mainly on the text design method.

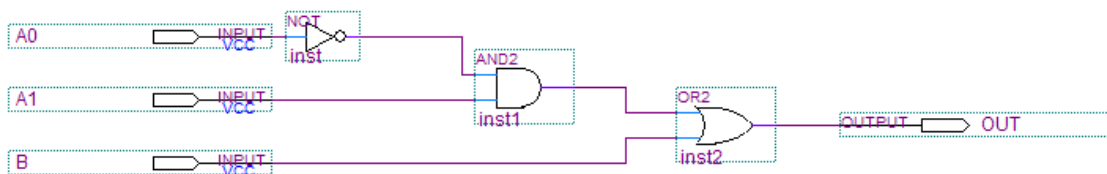
This lab deals with designing combinatorial circuits. This brief tutorial will give you the information you need to design combinatorial circuits using the text design method with Verilog.

Here is a sample text design file which illustrates many features of combinatorial Verilog, Lab1_Verilog1.v:

```
module Lab1_Verilog1 (A0, A1, B, OUT);
    input A0, A1, B;
    output OUT;

    assign OUT = (~A0 & A1) | B;
endmodule
```

This implements the following logic $OUT = (\text{NOT } A0 \text{ AND } A1) \text{ OR } B$;



Key things to note about this:

- A .v file must have a line *module* and *endmodule*. The name of the module must be the same as the file name.
- Then the first statements within this module should be used to describe the inputs and outputs.
- Each line of code ends with a semicolon.

- Outputs must be on the left hand side of an equal sign.
- Inputs must be on the right hand side of an equal sign.
- Boolean expressions are written using ands (& or and), ors (| or or), exclusive ors (^ or xor), not (~ or not), nand (~& or nand) , nor, and exclusive nor (~^ or xnor).

Here is an example of a .v file which implements a Boolean logic using a case statement:

```

module Lab1_Verilog2 (in, out);
    input [3:0] in;
    output reg [7:0] out;

    always @(in)
        case(in)
            4'b1000: out = 8'b01100001; // out = a
            4'b0100: out = 8'b01100010; // out = b
            4'b0010: out = 8'b01100011; // out = c
            4'b0001: out = 8'b01100100; // out = d
            default out = 8'b00111111; // out = ?
        endcase
    endmodule

```

This shows several features in Verilog:

- Multiple inputs and outputs can be grouped together into a group. One can refer to a subset of this group by using brackets –i.e. in[0], or out[7].
- For this program, the input in can have one of 16 values. The program specifies the output for 4 of the possible 16 values. If the inputs are in one of the other possible states, the output should be an ASCII question mark.
- Comments are entered by surrounding them with /* -- */ if the comment spans more than two lines, or use // if the comment takes one line.
- A binary number can be entered with the following notation: 4b'0000. Other ways to enter this number are: decimal (12) and hexadecimal (h'10 or (16)₁₀)

Another way to enter a Boolean expression is with a case statement. Here is code to implement the above logic with if then else statement:

```
module Lab1_Verilog2 (in, out);
    input [3:0] in;
    output reg [7:0] out;

    always @(in)
        if(in == 4'b1000) out = 8'b01100001; // out = a
        else if(in == 4'b0100) out = 8'b01100010; // out = b
        else if(in == 4'b0010) out = 8'b01100011; // out = c
        else if(in == 4'b0001) out = 8'b01100100; // out = d
        else out = 8'b00111111; // out = ?

endmodule
```

- As in C, == is used for comparison – if the left-hand and right-hand side of the expression equal each other, the result is TRUE, and the THEN statement will be implemented.
- Using the else if clause may cause Altera to generate unnecessarily complex logic. This is because the logic not only has to check to see if the current else if statement is true, but also has to verify that the if and all previous else if statements are false. It is usually better to avoid using the else if clause if possible. Use a case statement instead.