Introduction to Combinational Verilog

EECS270 rev 9/25/12
What is Verilog?

- An HDL: Hardware Description Language
- Another way to express logical function.
- Fundamental form is an algebraic equation. For example:

  \[ C = A \& B \]

Schematic equivalent:
Anatomy of a Verilog Module

- Used to organize logical functions
  - Top-Level-Module (just one)
    - Sub-Level-Module 1 (multiple sub modules)
    - Sub-Level-Module 2
    - Sub-Level-Module 3
- Consists of:
  - Module Declarations
  - Input Output Declarations
  - Logic Function
    - output variables = fnc(input variables)
Let's Make a Verilog Half Adder Module

Schematic Half Adder

Verilog Half Adder

```verilog
module add_half (a, b, s, cout);

    input a, b;
    output s, cout;
    wire s, cout;

    assign s = a ^ b;
    assign cout = a & b;

endmodule  // end of half adder module
```
Module Declaration

Module Declaration

Module Name

module add_half (a, b, s, cout);

Inputs (listed first)

Outputs (listed last)

endmodule

Marks end of module
Input and Output Declarations

Input Declaration

Input Connections Listed in Module Declaration

Output Declaration

Output Connections Listed in Module Declaration

Output Data Type Declaration

- Must be declared as either “wire” or “reg”

- **wire**: Does not hold state and depends continuously on input values.

- **reg**: (Register) dependent on previous values or state. This type will be used later in sequential logic labs.

```plaintext
input a, b;
output s, cout;
wire s, cout;
```
Assignments

- Assignments associate output variables to input variables via a logical function.

  ```
  assign s = a ^ b;
  assign cout = a & b;
  ```

- Although the assignment for cout follows s, s and cout respond **simultaneously (in parallel)** to changes in a and b.

- Unlike programming languages such as C, these logical statements are not processed in sequential order.
Verilog Operators Available for Lab 3

Verilog is a very powerful language with high level constructs such as: If Then Else, Case Statements, Arithmetic operators, Etc

For Lab 3 you are only allowed to use:

1. Bitwise AND &
2. Bitwise OR |
3. Bitwise negation ~ Can be combined with other logical functions ie NOR of A, B is ~(A | B).
4. Bitwise XOR ^
Liters

Literals are constants. The syntax for a constant is:

\[
<\text{size}>\langle\text{base format}\rangle<\text{number}>
\]

**size:** size of number in bits, optional

**base:** numeric base, b (binary), d (decimal, default), h (hex)

**number:** Constant to be specified

For example:

- 549  \hspace{1em} \text{decimal number, no size specified}
- 'h8FF \hspace{1em} \text{hex number, no size specified}
- 4'b11 \hspace{1em} \text{4-bit binary number (0011)}
- 5'd3 \hspace{1em} \text{5-bit decimal number (00011)}
Defining constant references can be very helpful and makes your HDL readable.

For example,

```vhdl
`define CONST3 3'b011
a = b & `CONST3;
```
Sets

- Sets are a way to express a collection of signals
- For example, let's say we want to set the seven segment HEX0 display to the pattern 7.
- 1st, declare the seven segments as a set:

  Output [6:0] H0;

- 2nd, set each element to the appropriate level to give the pattern 7.
Sets, continued

<table>
<thead>
<tr>
<th>Member Assignment</th>
<th>Group Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>assign H0[0] = 0;</td>
<td>assign H0 = 7'b1111000</td>
</tr>
<tr>
<td>assign H0[1] = 0;</td>
<td>or</td>
</tr>
<tr>
<td>assign H0[2] = 0;</td>
<td>assign H0 = 7'h78</td>
</tr>
<tr>
<td>assign H0[3] = 1;</td>
<td>or</td>
</tr>
<tr>
<td>assign H0[4] = 1;</td>
<td>`define N7 7'b1111000</td>
</tr>
<tr>
<td>assign H0[5] = 1;</td>
<td>assign H0 = `N7;</td>
</tr>
<tr>
<td>assign H0[6] = 1;</td>
<td></td>
</tr>
</tbody>
</table>

Clearest and easiest
Module Hierarchy

• HDL project are typically organized into one top level module and several supporting sub modules.

• Sub modules generally represent the various functions needed to support your design.

• The top level module defines the inputs and outputs of your application and the organization of the supporting functions or sub modules.
Module Hierarchy Example

Consider a two-bit adder built with a half adder and full adder.

**Figure 1a:** Half adder

**Figure 1b:** Full adder

**Figure 2c:** Two-bit adder built from half adder and full adder
Half Adder Module
Sub Module

module add_half (a, b, s, cout);
    input a, b;
    output s, cout;
    wire s, cout;
    assign s = a ^ b;
    assign cout = a & b;
endmodule // end of half adder module
module add_full (a, b, cin, s, cout);

input a, b, cin;
output s, cout;
wire s, cout;

assign s = a ^ b ^ cin;
assign cout = (a & b) | (a & cin) | (b & cin);

endmodule // end of full adder module
Two Bit Adder Module
Top Level Module

Pass values between modules with “wire” types

module add_2bit (a, b, s, cout);

input [1:0] a, b; // Both a and b are 2 bit inputs
output [1:0] s; // s[1] = MSB of s, s[0] = LSB of s
output cout;
wire [1:0] s;
wire cout;
wire c0; // intermediate carry between adders
add_half a1(a[0], b[0], s[0], c0);
add_full a2(a[1], b[1], c0, s[1], cout);
endmodule

Application IO on top level

module “call” syntax
<module name> <instance name> (<parameter list>);

• Module name is the name declared in module
• Instance name is one you make up. Each must be unique.
• Inputs and Outputs must follow order set in module declaration.
Final Verilog File

```verilog
module add_half (a, b, s, cout);
    input a, b;
    output s, cout;
    wire s, cout;

    assign s = a ^ b;
    assign cout = a & b;
endmodule // end of half adder module

module add_full (a, b, cin, s, cout);
    input a, b, cin;
    output s, cout;
    wire s, cout;

    assign s = a ^ b ^ cin;
    assign cout = (a & b) | (a & cin) | (b & cin);
endmodule // end of full adder module

module add_2bit (a, b, s, cout);
    input [1:0] a, b; // Both a and b are 2 bit inputs
    output [1:0] s;   // s[1] = MSB of s, s[0] = LSB of s
    output cout;
    wire [1:0] s;
    wire cout;
    wire c0;        // intermediate carry between adders

    add_half a1(a[0], b[0], s[0], c0);
    add_full a2(a[1], b[1], c0, s[1], cout);
endmodule
```
Quartus Particulars

• As with schematic projects, the top-level-module and project name must be the same.

• Top-level-module and file (.v) should be the same name.

• Although it is acceptable to place modules in separate files, it is recommended to put the top-level-module and sub modules in the same file.

• FPGA pin assignments are made to IO names established in the top-level-module with either pin assignment editor or editing the qsf file.