ECE 351

Professor Greenwood

04/09/19

* Module names should be descriptive of the circuitry being described.
* Every block of circuitry is described by a module.
* **Module Structure**
  + Module Name
  + Port List
  + Port Declarations (if ports present)
    - **Testbench** doesn’t have a port list
  + Parameters (optional)
    - Highly recommended for reusability.
  + Declarations of **wires** (local to module only, undefined elsewhere)
    - **wire** data types are internal to the module.
    - There is no such thing as a global variable in Verilog.
  + Declarations of **regs**
  + Declarations of other variables
  + Data flow statements
  + Instantiation of lower level modules
    - All the way down to primitives
  + Always blocks
  + Initial blocks
    - All behavioral statements go in these blocks.
  + Tasks and Functions
    - Contained within the module in which they are described.
  + endmodule statement
* **output**: Declares a net type variable.
  + Example:

module foo(a, c);

input a; //net

**output** c; //net

…

Endmodule

* + If ‘c’ was to be a reg variable, you must re-declare it as such.
* Figure 4.4: **Port Connection Rules**
  + Designs that do not follow the connection rules of this figure will not work.
  + **output** types are initially net data types.
    - The next port must be a net type variable.
      * This also applies to testbenches where the connection variables cannot be a reg, and must be a wire, otherwise the outputs cannot be connected to a reg variable.
  + Initial synthesis issues are due to incorrect port connections.
* **Connecting** **modules**
  + Instantiation
  + The port list
    - Signals can be connected two ways:

1. **Connection by position.**
2. **Connection by name.** (preferred)
   * **By Position**
     + Is a one-to-one mapping of connections between modules.
     + Similar to subroutine calls in C.
   * **By Name**
     + The preferred method (use this).
     + Uses a “.” Followed by the name of the module template, followed by a parentheses containing the instantiating modules variable.
       - Example:

fa\_byname(.c\_out(C\_OUT), …)

* + - Advantages of order **by Name**:
      * Order no longer matters.
      * Keeping the order straight of a large ordered list is troublesome.
      * Doesn’t prevent anyone from working on code that is dependent on your work, it can be unfinished but as long as the other designer knows your signal names and their declaration, their work can continue.
* **Abstraction**: A description that hides unimportant information to make the description less complex.
  + How to describe a digital design:

1. Schematic (less abstract, precise schematic with components)
2. Boolean equations (more abstract, describes functionality)
3. Truth table (most abstract, inputs & outputs only)
   * The synthesizer takes these views:
4. Defined way to build the circuit.
   1. The synthesizer must be creative on how to implement the same circuit with a given library.
5. The functional view still restricts the synthesizer.
6. The Truth table with inputs and outputs lets the synthesizer dictate how the logic gates are implemented.

* **Verilog** has 3 levels of abstraction.

1. Gate Level (lowest level of abstraction)
2. Data Flow
3. Behavioral (highest level of abstraction)
   * The **Verilog Standard** actually has 4 levels of abstraction.
     + The 4th level of abstraction (lowest) is the transistor level.
     + We will not reach the 4th level of abstraction.

* A module may contain **multiple levels of abstraction**.
* Any module written solely in “ \* “ is called “RTL coding”.
* If you have a design which can barely fit in the FPGA, coding in the high-level behavioral level of abstraction, you put the majority of the work on the FPGA, but lose control over how the logic is synthesized.
* Gate Level Modeling: **Primitive Gate**
  + The basic building block of a Verilog program is a module.
  + **Primitive Gates** are predefined logic gates in Verilog.
  + Basic **primitive gates**:
    - **and**, **nand**, **or**, **nor**, **xor**, **xnor**, **buf**, **not**,
  + Format:

gate\_type gate-identifier(Output Name, in0, in1, in2, …, ink);

* + - The gate type is “and”, “nor”, etc.
    - gate-identifier is the Circuit Reference Designator (CRD).
    - Only one Output Name
    - Limitless inputs.
    - Order only matters for the first entry, the “output”, the inputs can be entered in any order.
  + Using multiple instances of the same gate primitive.
    - Example:

nand G1(Y1, A, B);

nand G2(Y2, A, B);

nand G3(Y3, A, B);

* + Alternative method for multiple instances of the same gate primitive.
    - Example:

nand G1(Y1, A, B), G2(Y2, A, B), G3(Y3, A, B);

* + Both methods work.
* **Truth Tables**
  + Simulators use truth tables with inputs that use values:
    - 0
    - 1
    - X
    - Z
* **Buffers:** Keyword **buf**
  + All ports but the last port in the port list are outputs.
* **Tri-state Buffers:** Keyword **bufif1, bufif0, notif1, notif0**
  + See figure 5.3 for details.
* Separating 1-bit inputs for functionality on different lines makes the code **more readable**.
  + Data inputs go on one line of declared inputs.
  + Select lines go on one separate line of declared inputs.
  + Both declaration lines are commented accordingly. (See example 5.5)
* **Execution is in parallel**, not sequential (as in C).
  + It doesn’t matter what order gate instantiations are listed.
  + Inputs and outputs are already connected appropriately in the port list.
  + However, you must **maintain readable code**, so being neat with your code is important.
* **An alternative Instantiation for example 5.5**.
  + Example:

module mux\_4\_1(out, I, s);

Input [3:0]I;

Input [1:0]s;

…

Endmodule

* + Get in the habit of coding in this style.
* Next weeks assignment will include a simulation.
  + Must complete the **simulation tutorial** by next week.
    - The tutorial itself.
    - A design that you will simulate.
    - Make a hard-copy.