Verilog Overview (1/3)

- A Hardware Description Language (HDL) for describing & testing logic circuits.
  - text based way to talk about designs
  - easier to simulate before silicon
  - translate into silicon directly
- No sequential execution, normally hardware just “runs” continuously.
- Verilog: A strange version of C, with some changes to account for time
  - VHDL is alternative; similar and can pick it up easily.
  - Verilog simpler to learn!

Verilog Overview (2/3)

- Verilog description composed of modules:
  - module Name (port list);
    Declarations and Statements;
  - endmodule
- Modules can have instantiations of other modules, or use primitives supplied by language
- Note that Verilog varies from C syntax, borrowing from Ada programming language at times (endmodule)

Verilog Overview (3/3)

- Verilog has 2 basic modes
  1. Structural composition: describes that structure of the hardware components, including how ports of modules are connected together
    - module contents are builtin gates (and, or, xor, not, nand, nor, xnor, buf) or other modules previously declared
  2. Behavioral: describes what should be done in a module
    - module contents are C-like assignment statements, loops

Example: Structural XOR (xor built-in,but..)

```
module xor(X, Y, Z);
  input X, Y;
  output Z;
  Default is 1 bit wide data
  wire notX, notY, “ports” connect components
  XnotY, YnotX;
  not (notX, X),
  (notY, Y);
  and (YnotX, notX, Y),
  (XnotY, X, notY);
  or (Z, XnotX, XnotY);
endmodule
```

Example: Behavioral XOR in Verilog

```
module xorB(X, Y, Z);
  input X, Y;
  output Z;
  reg Z;
  always @(X or Y)
    Z = X ^ Y;
endmodule
```

- Unusual parts of above Verilog
  - always @(X or Y) ”=> whenever X or Y changes, do the following statement
  - “reg” is only type of behavioral data that can be changed in assignment, so must redeclare Z
- Default is single bit data types: X, Y, Z
**Verilog: replication, hierarchy**

- Often in hardware need many copies of an item, connected together in a regular way
  - Need way to name each copy
  - Need way to specify how many copies
- Specify a module with 4 XORs using existing module example

**Example: Replicated XOR in Verilog**

```verilog
module 4xor(A, B, C);
  input[3:0] A, B;
  output[3:0] C;
  xorB My4XOR[3:0] (.X(A), .Y(B), .Z(C) );
endmodule
```

- Note 1: can associate ports explicitly by name,
  ```verilog
  .X (A), .Y(B), .Z(C) 
  ```
- or implicitly by order (as in C)
  ```verilog
  (A, B, C)
  ```
- Note 2: must give a name to new instance of xors (My4XOR)

**Verilog big idea: Time**

- Difference from normal prog. lang. is that time is part of the language
  - part of what trying to describe is when things occur, or how long things will take
- In both structural and behavioral Verilog, determine time with #n: event will take place in n time units
  - structural: not #2 (notX, X) says notX does not change until time advances 2 ns
  - behavioral: Z = #2 A ^ B; says Z does not change until time advances 2 ns
- Default unit is nanoseconds; can change

**Example:**

```verilog
module test(stream);
  output stream;
  reg stream;
  initial
    begin
      stream = 0;
      #2 stream = 1;
      #5 stream = 0;
      #3 stream = 1;
      #4 stream = 0;
    end
endmodule
```

**Testing in Verilog**

- Code above just defined a new module
- Need separate code to test the module (just like C/Java)
- Since hardware is hard to build, major emphasis on testing in HDL
- Testing modules called “test benches” in Verilog, like a bench in a lab dedicated to testing
- Can use time to say how things change

**Testing Verilog**

- Create a test module that instantiates xor:
  ```verilog
  module testxor;
    reg x, y, expected; wire z;
    xor myxor(.x(x), .y(y), .z(z));
    /* add testing code */
  endmodule
  ```
- Syntax: declare registers, instantiate module.
Testing continued

• Now we write code to try different inputs by assigning to registers:

  ```verilog
  initial
  begin
    x=0; y=0; expected=0;
    #10 y=1; expected=1;
    #10 x=1; y=0;
    #10 y=1; expected=0;
  end
  ```

Testing continued

• Pound sign syntax (#10) indicates code should wait simulated time (10 nanoseconds in this case).

• Values of registers can be changed with assignment statements.

• So far we have the xor module and a testxor module that iterates over all the inputs. How do we see if it is correct?

Output

```verilog
x=0, y=0, z=0, exp=0, time=0
x=0, y=1, z=1, exp=1, time=10
x=1, y=0, z=1, exp=1, time=20
x=1, y=1, z=0, exp=0, time=30
```

• Expected value matches actual value, so Verilog works

Peer Instruction

• How many mistakes in this module?

```verilog
module test(X);
output X;
initial
begin
  X = 0;
  #2 X = 1;
end
end
```

Peer Instruction Answer

1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.0
In conclusion

• Verilog allows both structural and behavioral descriptions, helpful in testing

• Syntax a mixture of C (operators, for, while, if, print) and Ada (begin... end, case...endcase, module ...endmodule)

• Some special features only in Hardware Description Languages
  • # time delay, initial, monitor

• Verilog can describe everything from single gate to full computer system; you get to design a simple processor