Introduction to Verilog and ModelSim

(Part 2 – Verilog Basics)

Basic Verilog Construct

- Verilog code starts with a **module** definition

```
module test (paramA, paramB);
// codes go here
endmodule
```

  - *test* is the name (label) of the module
  - *paramA* and *paramB* are interface ports
  - notice the semi-colon after parameter list
  - must end with **endmodule** keyword
- Also, notice C99/C++-style comments

Verilog Module

- Example of full module declaration
  - original Verilog-95 standard

```
module invert (paramA, paramB);
input paramA;
output paramB;
// code here
endmodule
```

- All parameters must be assigned a direction
  - input, output OR inout
  - some tools require tri-state representation of inout

Verilog Module (cont.)

- If using Verilog 2001 standard
  - input/output can be done inside parentheses

```
module invert (input paramA, output paramB);
// code here
endmodule
```

  - looks nicer if arranged this way...

```
module invert
(
   input paramA,
   output paramB
);
// code here
endmodule
```

Verilog: Wire/Reg

- Driven signals (incl. output) needs type:
  - wire :
    - physical connection that is continuously updated
    - keyword used with **assign** statement
  - reg :
    - signal that is assigned a value under given conditions (usually used to represent latched signal)
    - keyword used with **always** block

Verilog Example: Wire

- Describing simple logic AND

```
module logic_and (paramA, paramB, paramC);
input paramA, paramB;
output paramC;
wire paramC;
assign paramC = paramA & paramB;
endmodule
```

Verilog Example: Reg

- Describing simple logic AND

```
module logic_and (paramA, paramB, paramC);
input paramA, paramB;
output paramC;
reg paramC;
always @ (paramA or paramB)
begin
   paramC = paramA & paramB;
end
endmodule
```

Verilog Operators

- Three basic bitwise operators (gates)
  - ~ (bitwise invert – unary operator)
  - & (bitwise and – binary operator)
  - | (bitwise or – binary operator)
- Bitwise XOR is also frequently used
  - ^ (bitwise XOR binary operator)
- Many more operators but these are the basics
  - The rest can be built from these

Verilog: Numbers (LATER!)

- Can be represented as:
  - Base-N (binary, octal, hex, decimal)
    - integer A = 4'b1010; B = 4'o13; C = 8'h55; D = 7'd101;
  - Integers
    - integer A = 9;
  - Real (floating point) numbers
    - real A = 12.345;
- Usually used for numerical processing
  - Much easier than structurally built block

Your First Verilog Module

- Create a work path – no whitespace(s)!
  - e.g. c:\users\public\modelsim\<your-name>
- Type the following code
  - save as *logic_and2.v* in your work path

```
module logic_and2 (inputA, inputB, outputC);
input inputA, inputB;
output outputC;
reg outputC;
always @ (inputA or inputB)
begin
   outputC = inputA & inputB;
end
endmodule
```

Testbench

- An environment for a Design Under Test (DUT)
  - Generates stimuli (input signals going into)
  - Monitors response (output signals coming out)
- Used to verify the functionality of DUT
  - Basically, MUST consider ALL possible input range
  - Creating a testbench is part of the design flow

Your First Testbench

- Type the following code
  - save as *logic_and2_tb.v* in your work path

```
module logic_and2_tb ();
reg driveA, driveB;
wire monitorC;
integer loop;
initial
begin
   for (loop=0;loop<4;loop=loop+1)
      begin
         {driveA,driveB} = loop;
         #10;
      end
end
logic_and2 mygate (driveA, driveB, monitorC);
endmodule
```

No interface ports needed!

{ } is symbol for concatenation

initial block executed only once

Delay – depends on timescale

Practical Session 2.1

- Implement a Verilog module
  - The required logic will be decided by your instructor
- Create a suitable testbench for the module
  - How many cycles required by the loop?