Lecture 6
Verilog HDL, Part 1

Xuan ‘Silvia’ Zhang
Washington University in St. Louis

http://classes.engineering.wustl.edu/ese461/
Outline

Linux Basics

VCS Simulator

Verilog Intro
Useful Commands

• **Navigation**
  - `ls`, `cd`, `pwd`, `ln`, `find`, `less`, `more`

• **Manipulation**
  - `cat`, `cp`, `mv`, `grep`, `mkdir`, `touch`
  - `chmod`, `chown`

• **Check status**
  - `df`, `du`, `quota`, `uname`, `history`

• **Set Environment**
  - Unix shell: Bash, C shell (`csh`, `tcsh`), etc.
  - `source .bashrc`
  - `module avail`
Outline

Linux Basics

VCS Simulator

Verilog Intro
Synopsys VCS

- Electronic design automation (EDA)
  - computer-aided design (CAD), autodesk
  - eco-system: Synopsys, Cadence, Mentor Graphics

- Functional verification solution
  - compile time
  - runtime
Homework #3

- Please design 3 homework questions/problem sets that are representative of the review materials we have covered in lectures and quiz.
- Please include the solutions of all the questions.
- Please submit your homework electronically to the TA’s email (dengxue.yan@wustl.edu)
- Due 09/21 (Wednesday)
- no grace period
Lab #1: Verilog Basics

- (HW1, Q2) 74HC138 MSI CMOS circuit
- (HW2, Q2) Traffic light control
- (HW2, Q5) FSM of string processing
Outline

Linux Basics

VCS Simulator

Verilog Intro
Hardware Description Language (HDL)

- Specialized computer language for circuits
  - precise and formal description of structure and behavior
  - allow for automated simulation and synthesis
  - similar to C: expression, statement, control structure
  - key difference from C: explicit notion of time

- VHDL vs Verilog
  - syntax, ease of use, readability

- Higher level languages
  - abstraction level (assembly)
  - SystemC
  - high-level synthesis: Vivado HLS, Catapult, etc.
Verilog Design Styles

• Bottom-up design
  - traditional method, not popular anymore
  - use gate-level structure

• Top-down design
  - hierarchical design method
  - design steps
  - specification, high-level design, low-level design

• Abstract levels
  - behavioral model (test bench)
  - RTL model (register level)
  - structural model (register and gate level)
Verilog Structure

- **Modules**
  - structural and behavioral statements
  - structural: logic gates, counters, etc.
  - behavioral: loops, if-then, stimulus vectors

```verilog
module d_ff (d, clk, q, q_bar);
  input d, clk;
  output q, q_bar;
  wire d, clk;
  reg q, q_bar;

  always @(posedge clk)
  begin
    q <= d;
    q_bar <= !d;
  end

endmodule
```
Circuit and Test Bench Modules

• An example
  - Initial: execute only once at time zero
  - delay
  - input, output
  - reg, wire
  - $display

```verilog
`timescale 1ns / 1ps
//create a NAND gate out of an AND and an Invertor
module some_logic_component (c, a, b);
    // declare port signals
    output c;
    input a, b;
    // declare internal wire
    wire d;
    // instantiate structural logic gates
    and a1(d, a, b); //d is output, a and b are inputs
    not n1(c, d); //c is output, d is input
endmodule

//test the NAND gate
module test_bench; //module with no ports
    reg A, B;
    wire C;

    //instantiate your circuit
    some_logic_component S1(C, A, B);

    //Behavioral code block generates stimulus to test circuit
    initial
        begin
            A = 1'b0; B = 1'b0;
            #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
            A = 1'b0; B = 1'b1;
            #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
            A = 1'b1; B = 1'b0;
            #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
            A = 1'b1; B = 1'b1;
            #50 $display("A = %b, B = %b, Nand output C = %b \n", A, B, C);
            end
endmodule
Structural Data Types

• wire vs reg
  - wire nets act like real wires in circuits
  - reg holds its value until set to another value
  - reg initialized to x; wire to z

• in, out, inout
  - modules communicate through ports
  - parameter list at the top of module

• Rules for matching data type and port type
  - reg as output of behavioral blocks
  - use wire for inputs, inouts, and most output of structural elements
  - special strength
Structural Design with Modules

- Module name, endmodule
- Port declaration, pin width
- Internal structure
- Parameterized delay

```verilog
module d_ff (d, clk, q, q_bar);
  input d, clk;
  output q, q_bar;
  wire d, clk;
  reg q, q_bar;

  always @(posedge clk)
  begin
    q <= d;
    q_bar <= !d;
  end
endmodule
```

```verilog
module some_logic_component (c, a, b);
  ... //some code
  parameter andDelay = 2; //default delays
  parameter invDelay = 2;
  and #andDelay a1(d, a, b); //using parameter delays
  not #invDelay n1(c, d);
endmodule

module test_bench; //module with no ports
  ...
  some_logic_component #(5, 4) S3(E, A, B); //override andDelay=5, invDelay=4
  some_logic_component #(5) S2(D, A, B); //override andDelay=5, invDelay=2
  some_logic_component S1(C, A, B); //uses default delays
  ...
endmodule
```
Gate Primitives and Delay Operator

• Scalable inputs
  - nand, nor, and, or, xor, xnor
    ```
    nand a1(out1, in1, in2); //2-input NAND gate
eq nand a2(out1, in1, in2, in3, in4, in5); //5-input NAND gate
    ```

• Single input
  - not, buf
    ```
    not inv1(c,a);
    ```

• Gate delay
  - #(rise, fall)
  - #(rise, fall, off)
  - min:typ:max
    ```
    notif0 #(10,11,27) inv2(c,d,control) //rise=10, fall=11, off=27(not if control=0)
nor  #(10,11) nor1(c,a,b);  //rise=10, fall=11 (nor gate)
xnor #(10) xnor1(i,g,h);  //rise=10, fall=10 (xnor gate)
    ```

// min:typ:max values defined for the (rise, fall) delays
or  #(8:10:12, 10:11:13) or1(c,a,b);
Behavioral Data Types

- integer and real
- time

..... //code fragment from inside a module

integer i, y;
real a;
real b = 3.5;
real c = 4;
time simulationTime;
initial
    begin
        y = 4;
        i = 5 + y;
        c = c + 3.5;
        a = 5.3e4;
        simulationTime = $time;
        $display("integer y = %d, i = %f \n", y, i);
        $display("reals c = %f, a = %e, b= %g \n", c, a, b);
        $display("time simulationTime = %t \n", simulationTime);
    end
Number Syntax

- Binary (‘b, ‘B)
- Hexadecimal (‘h, ‘H)
- Decimal (‘d, ‘D)
- Octal (‘o, ‘O)
- Strings of digits (0-9, A-F, a-f)
- Unknown (x, X)
- High impedance (z, Z)

<table>
<thead>
<tr>
<th>Numbers</th>
<th>Bits</th>
<th>Base</th>
<th>Val</th>
<th>Stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>3'b101</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>'b11</td>
<td>?</td>
<td>2</td>
<td>3</td>
<td>000...0011</td>
</tr>
<tr>
<td>8'b11</td>
<td>8</td>
<td>2</td>
<td>3</td>
<td>00000011</td>
</tr>
<tr>
<td>8'b1010_1011</td>
<td>8</td>
<td>2</td>
<td>171</td>
<td>10101011</td>
</tr>
<tr>
<td>3'd6</td>
<td>3</td>
<td>10</td>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>6'o42</td>
<td>6</td>
<td>8</td>
<td>34</td>
<td>100010</td>
</tr>
<tr>
<td>8'hAB</td>
<td>8</td>
<td>16</td>
<td>171</td>
<td>10101011</td>
</tr>
<tr>
<td>42</td>
<td>?</td>
<td>10</td>
<td>42</td>
<td>00...0101010</td>
</tr>
<tr>
<td>'1</td>
<td>?</td>
<td>n/a</td>
<td></td>
<td>11...111</td>
</tr>
</tbody>
</table>
Operators

- **Arithmetic**
  - *, /, +, -, %

- **Logical**
  - !, &&, ||

- **Relational**
  - >, <, >=, <=, ==, !=

- **Bit operation**
  - ~, &, |, ^

- **Shift**
  - >>, <<

- **Concatenation**
  - {}

- **Conditional**
  - ?

<table>
<thead>
<tr>
<th>Op</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>NOT</td>
</tr>
<tr>
<td>*, /, %</td>
<td>MUL, DIV, MOD</td>
</tr>
<tr>
<td>+, -</td>
<td>PLUS, MINUS</td>
</tr>
<tr>
<td>&lt;&lt;, &gt;&gt;</td>
<td>Logical Left / Right Shift</td>
</tr>
<tr>
<td>&lt;&lt;&lt;, &gt;&gt;&gt;</td>
<td>Arithmetic Left / Right Shift</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=</td>
<td>Relative Comparison</td>
</tr>
<tr>
<td>==, !=</td>
<td>Equality Comparison</td>
</tr>
<tr>
<td>&amp;</td>
<td>AND, NAND</td>
</tr>
<tr>
<td>^</td>
<td>XOR, XNOR</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>Conditional</td>
</tr>
</tbody>
</table>
Control Flow

- if-else
  - if-elseif-else
- Case

```verilog
reg Q;
always@(posedge clock or posedge reset)
begin
  if(reset) Q<=0;
  else Q<=D;
end
```

```verilog
case(address)
  0 : $display ("It is 11:40PM");
  1 : $display ("I am feeling sleepy");
  2 : $display ("Let me skip this tutorial");
  default : $display ("Need to complete");
endcase
```

- While

```verilog
always @(posedge clk or posedge rst)
if (rst) begin
  count <= 0;
end else begin : COUNT
  while (enable) begin
    count <= count + 1;
    disable COUNT;
  end
end
```
Control Flow

- For
- Repeat

```plaintext
for (i = 0; i < 16; i = i + 1) begin
  $display ("Current value of i is %d", i);
end

repeat (16) begin
  $display ("Current value of i is %d", i);
  i = i + 1;
end
```
A Verilog Counter

- 50MHz clock
- Count from 0 to 7, reset, then start from 0 again
Questions?

Comments?

Discussion?