Verilog Tutorial for ECE260B

Prof. Andrew B. Kahng   TAs: Sherief Reda, Swamy Muddu
18 January 2004

References:

1. Application Specific Integrated Circuits, Michael Smith
2. Verilog Digital Computer Design, Mark Arnold
3. Verilog Digital System Design, Zainalabedin Navabi
Historical background

- Verilog is a hardware description language that was developed by Gateway Design Automation which was later purchased by Cadence.

- IEEE standard number 1364.

- Has similarities to C in syntax

- This is an introductory lecture. You will not be asked to write Verilog models in ECE260B.
Characterizing Hardware Languages

- Timing
- Concurrency
- Modeling hardware
Verilog identifiers

- What is an identifier?
- An identifier may contain any sequence of letters, digits, a dollar sign and the underscore symbol
- An identifier must start with a letter or an underscore
- Verilog is case sensitive
- Examples: clock, case_sensitive, ok123
Verilog logic values

Verilog has a 4 logic-value system: 0, 1, x and z

Verilog data types

- Main types: **nets** and **registers**
- What is the difference between **nets** and **registers**?
- Also **integer**, **real**, **event** and **time**
- Main net types: **wire**, **supply0** and **supply1**
- Register data type: **reg**
- **wire** and **reg** are scalar by default, i.e., one bit, but can be declared as range of bits
Examples of valid Verilog declarations

```verilog
wire pwr_good, pwr_on, pwr_stable;
assign pwr_stable = 1;

integer i;
i = 123;

reg Q; wire D;
assign D = 1;
Q = D;

wire [31:0] Dbus;
reg [7:0] vector;
assign Dbus[1] = 1;
assign Dbus[3:0] = 'b1111;
vector = 'b1010;
```
Numbers

- Constants numbers are integer and real constants
- Integer constants are written as width’radix value
- Radix may be decimal (d or D), hex (h or H), octal (o or O) or binary (b or B)
- Real constants are written using decimal (100.0) or scientific (1e2)

Operators

- Three types of operators: unary, binary and ternary
- Verilog operators are similar to those in the C language
- except there is autoincrement (++) or autodecrement (--)
Hierarchy

➢ The **module** is the basic unit of code in the Verilog language

```verilog
module holiday_1(sat, sun, weekend);
    input sat, sun; output weekend;
    assign weekend = sat | sun;
endmodule
```

➢ Within a module we may instantiate other modules, but we cannot declare other modules
Procedures and Assignments

- A Verilog procedure is an **always** or **initial** statement, a **task**, or a **function**

- Statements in a procedure execute sequentially in the order in which they appear, but the procedure execute concurrently with other procedures

```verilog
module trial
    // some statements
always
    begin
        // some statements
    end
endmodule
```
Timing Controls and Delay

- The statements within a sequential block are executed in order, but, in the absence of any delay, they are executed at the same simulation time.

- What is the difference between:

  (i) \( x = \#1 y \);
  (ii) \( \#1 x = y \);
  (iii) \( \#1; \)
  \( x = y \);
Even Controls

```verilog
always @(clk) Y = X;
always @(posedge clk) Y = X;
always @(negedge clk) Y = X;
always @(negedge clk or posedge clk) Y = X;
```
Tasks and Functions

- A task is a type of procedure called from another procedure. It has both inputs and outputs but does not return a value.

- A function is a procedure used in any expression, has at least one input, no outputs, and returns a single value. A function may not call a task.

- Tasks and functions definitions occur inside modules.

- Tasks may contain timing controls but functions may not.

```plaintext
Call_A_task_And_Wait(input1, input2, output);
Result_immediate = call_a_function(all_inputs);
```
module F_subset_decode;
    reg [2:0] A, B, C, D, E, F;
initial
    begin
        A = 1; B = 0; D = 2; E = 3;
        C = subset_decode(A, B);
        F = subset_decode(D, E);
    end

function [2:0] subset_decode;
    input [2:0] a, b;
begin
    if (a <= b) subset_decode = a; else subset_decode = b;
end
endfunction
endmodule
Control Statements: Case and if Statements

if (switch) Y=1; else Y=0;

```verilog
module test_mux;
reg a, b, select;
wire out;
mux mux_1(a, b, out, select);
initial
begin
#2; select = 0; a = 0; b = 1;
#2; select = 1'bx; #2; select = 1'bz;
#2; select = 1;
end
endmodule
```

```verilog
module mux(a, b, mux_output, muxselect);
input a, b, mux_select;
output mux_output;
reg mux_output;
always begin
  case(mux_select)
  0: mux_output = a;
  1: mux_output = b;
  default: mux_output = 1'bx;
  endcase
#1;
end
endmodule
```
Control Statements: Loop Statements

A loop statement is a **for**, **while**, **repeat**, or **forever** statement

```verilog
module loop_1;
integer i;
reg [31:0] databus;
initial
begin
(i) for(i = 0; I <= 15; i = i+1) databus[i] = 1;
(ii) i=0; while (I <= 15) begin databus[i]=1; i=i+1; end
(iii) repeat(16) begin databus[i]=1; i = i+1; end
end
endmodule
```
Logic-Gate Modeling

- Verilog built-in logic models are the following primitives:

  nand nand_1(n001, n004, n005);
  Nand (n006, n005, n002);

- User-defined primitive (UDP)

  ```verilog
  primitive adder(sum, ina, inb);
  output sum; input ina, inb;
  table
    00 : 0;
    01 : 1;
    10 : 1;
    11 : 0;
  endtable
  endprimitive
  ```
Example 1

```verilog
module top;
integer x, y;
initial
begin
    x = 0;
    while (x <= 4095)
    begin
        for (y=1; y <= 4095; y = y+1)
        begin
            $display("x=%d, y=%d", x, y);
        end
        x = x+1;
    end
$write("all ");
$display("done");
endmodule
```
Example 2

module hard_xor;
    reg a, b;
    wire c;
    wire t1, t2, not_a, not_b;

    not i1(not_a, a);
    not i2(not_b, b);
    and a1(t1, not_a, b);
    and a2(t2, a, not_b);
    or o1(c, t1, t2);
endmodule
Example 3

```verilog
module behavioral_xor;
  reg a, b;
  reg c;
  reg t1, t2, not_a, not_b;

  always begin
    not_a = ~a;
    not_b = ~b;
    t1 = not_a & b;
    t2 = a & not_b;
    c = t1 | t2;
  end
endmodule
```
Example 4

```verilog
module two_blocks_time_control;
  integer x, y;
  initial
    begin
      #4
      a = 1;
      $display("a is one at $time = %d", $time);
    end

  initial
    begin
      #3;
      b=2;
      $display("b is two at $time=\%d", $time);
    end
endmodule
```
Example 5

```verilog
declare module top
integer ia, ib;
reg a, b;
wire c;

xor x1(c, a, b);
initial
begin
for (ia=0; ia <= 1; ia = ia+1)
begin
a = ia;
for(ib=0; ib <= 1; ib = ib+1)
begin
b = ib;
#10 $display("a=%d b=%d c=%d", a, b, c);
end
end
end
endmodule
```
Example 6

```verilog
always @(posedge sysclk)
begin
  if (clr) dout = 0;
  else
    if (ld)
      dout = din;
    else
      begin
        if (cnt) dout = dout + 1;
      end
  end
end
```
Example 7

```plaintext
integer count, sum, prod;
initial
begin
  sum = 0;
  count = 1;
  example(sum, prod, count, 2);
  example(sum, prod, count, 3);
  example(sum, prod, count, 5);
  example(sum, prod, count, 7);
  $display(sum, prod, count);
end

task example;
inout sum_arg;
output prod_arg;
inout count_arg;
input numb_arg;

integer count_arg, numb_arg, sum_arg, prod_arg;

begin
  sum_arg = sum_arg+count_arg;
  prod_arg = sum_arg*count_arg;
  count_arg=count_arg+numb_arg;
  $display(sum_arg, prod_arg);
end
endtask
```