EE 3613: Computer Organization
Chapter 5: Processor: Datapath & Control - 2
Verilog Tutorial

Avinash Karanth
Department of Electrical Engineering & Computer Science
Ohio University, Athens, Ohio 45701
E-mail: karanth@ohio.edu
Website: http://oucsace.cs.ohiou.edu/~avinashk/ee461a.htm
Acknowledgement: Srinivasan Ramasubramanian

1

Verilog Language

- Describe a system by a set of modules (equivalent to functions in C)
- Keywords e.g. module, are reserved and in all lower case letters
- Operators (some examples)
  - Arithmetic: +, -, *, /
  - Binary operators: &, |, ^, ~, !
  - Shift: << >>
  - Relational: <, <=, >, =>, ==, !=
  - Logical: &&, ||
- Comments start with "//" for one line or /* to */ across lines

2

Number Representation

- Numbers are specified in the traditional form as a series of digits with or without a sign but also in the following form
- <size><base format><number>
  - <size> contains number of bits (optional)
  - <base format>: is a single character ' followed by one of the following characters b, d, o and h, which stand for binary, decimal, octal and hex
  - <number> contains digits which are legal for the <base format>

Declaration	Comments
549	Decimal number
'h 8FF	Hexadecimal number
'o 765	Octal number
4'b 11	4-bit binary number 0011
3'b 10x	3-bit binary number with least significant bit unknown
5'd 3	5-bit decimal number
-4'b 11	4-bit 2's complement of 0011 or equivalently 1101

3

Physical Data Types

- Modeling wires (wire) and registers (reg)
- Register values store the last value that was procedurally assigned
- Wire variables represent physical connections between structural entities such as gates
- The reg and wire data objects may have the following values:

Value	Meaning
0	Logical zero or false
1	Logical one or true
x	Unknown logical value
z	High-impedance of tri-state gate

- reg variables are initialized to 0 at the start of simulation
- wire variable not connected to something has the x value

4

Program Structure

- A digital system as a set of modules
- Each module has an interface to other module (connectivity)
- Good Practice: Place one module per file (not a requirement)
- Modules may run concurrently
- Usually a top-level module which invokes instances of other modules

5

Module

- Represent bits of hardware ranging from simpler gates to complete systems i.e. microprocessor
- Specified behaviorally or structurally or a combination of two
- The structure of a module is the following:

```
module <module name> (<port list>);
  <declarations>
  <module items>
endmodule
```

Declaration	Comments
<module name>	is an identifier that uniquely names the module
<port list>	is a list of input, output ports which are used to connect to other modules
<declarations>	section specifies data objects as registers, memories, and wires as well as procedural constructs such as functions and tasks
<module items>	maybe initial constructs, always constructs, continous assignments, or instances of modules

6

Behavioral Example

- Here is a behavioral specification of module NAND

```
// Behavioral model of a NAND gate
module NAND(in1, in2, out);
input in1, in2;
output out;

// continuous assignment statement
assign out = ~(in1 & in2);

endmodule
```

- All undeclared variables are wires and are one bit wide
- Declare ALL variables!

7

Explanation of NAND Module

- The ports in1, in2, and out are wires
- The continuous assignment statement "assign" continuously watches for changes to variables in its right-hand side and whenever that happens, the right hand side is re-evaluated and the result is immediately propagated to the left hand side (out)
- The continuous assignment statement is used to model combinational circuits where the outputs change when one wiggles the input

```
// Behavioral model of a NAND gate
module NAND(in1, in2, out);
input in1, in2;
output out;

// continuous assignment statement
assign out = ~(in1 & in2);

endmodule
```

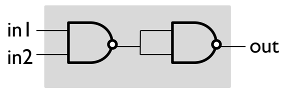
8

Instance of a Module

- The general form to invoke an instance of a module is:
- <module name> <parameter list> <instance name> (<port list>);
  - <parameter list> are values of parameters passed to the instance
  - <instance name> identifies the specific instance of the module
- An example parameter passed would be the delay of the gate
  - Need not be used while designing for this course!
- For our purposes, to invoke an instance of a module
- <module name> <instance name> (<port list>);

9

Structural Example: AND Gate

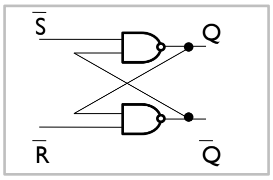


```
// Behavioral model of a AND gate from two NAND gates
module AND(in1, in2, out);
input in1, in2;
output out;
wire w1;

// two instances of the module NAND
NAND NAND1(in1, in2, w1);
NAND NAND2(w1,w1,out);

endmodule
```

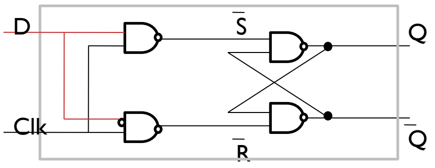
- This module has two instances of the NAND module called NAND1 and NAND2 connected together by an internal wire w1

10

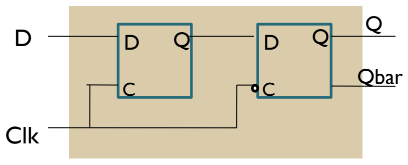
More Structural Examples



```
module SRLatch(S, R, Q, Qbar);
input  S, R;
output  Q, Qbar;
         NAND nand1(S, Qbar, Q);
         NAND nand2(R, Q, Qbar);
endmodule
```



```
module DLatch(Clk, D, Q, Qbar);
input  Clk, D;
output  Q, Qbar;
wire S, R;
         NAND nand1(D, Clk, S);
         NAND nand2(~D, Clk, R);
         SRLatch srlatch1(S, R, Q, Qbar)
endmodule
```



```
module DFlipFlop(Clk, D, Q, Qbar);
input  Clk, D;
output  Q, Qbar;
wire Qint, Qbarint;
   Dlatch dlatch1(Clk, D, Qint, Qbarint);
   Dlatch dlatch2(~Clk, Qint, Q, Qbar);
endmodule
```

Continuous vs. Procedural Assignment

- Continuous statement is used to model combinational logic
  - Continuous assignments drive wire variables
  - Evaluated and updated whenever an input operand changes value
- Procedural assignment changes the state of a register
  - Used for sequential logic or that are clock controlled
  - All procedural statements must be within "always" block

```
reg A;

always begin
    A = B & C;
end
```

Events

- The execution of a procedural statement is triggered
  - Value change on a wire or register
  - Occurence of a named event

```
always @r begin // controlled by any value change in the register r
    A = B & C;
end
```

```
always @(posedge Clk) // controlled by positive edge of the Clk
    A = B & C;
end
```

```
always @(negedge Clk) // controlled by negative edge of the Clk
    A = B & C;
end
```

13

Behavioral Model: D-Flip Flop

- What is the behavioral model of D-flipflop designed earlier?
  - During every positive edge, the input is transferred to the output

```
module DFlipFlop(Clk, D, Q, Qbar);
input  Clk, D;
output  Q, Qbar;

reg Qint;

         // Always is a procedural construct
         // any assignment maybe made only to registers

         always @(posedge Clk)
                 Qint = D;

         assign Q = Qint;
         assign Qbar = ~Qint;

endmodule
```

14

Register Size and Assignments

- Size of a register or wire in the declaration

```
reg [0:7] A, B;  // A and B are 8-bit wide with the most significant bit as 0

wire [0:3] Dataout; // Dataout is a 4-bit wide wire

reg [7:0] C; // C is a 8-bit register with the most significant bit as the 7
```

The last convention will be used in the class!

- Assignments and concatenations

```
A = 8'b 01011010;
B = {A[0:3] | A[4:7], 4'b 0000}
```

- B is set to the first 4 bits of A bitwise or-ed with the last 4 bits of A and then concatenated with 0000, B now holds a value of 11110000

- {} brackets means the bits of 2 or more arguments separated by commas are concatenated

15

Control Constructs

- 2 constructs are available

```
If (A == 4)
    begin
         B = 2;
    end
  else if (A == 2)
    begin
         B = 1;
    end
else
    begin
         B = 4;
    end
```

```
case (A)
4: begin
         B = 2;
   end

2: begin
         B = 1;
   end

default: begin
           B = 4;
         end

endcase
```

16

Control Statement Examples

2 to 1 Multiplexor

```
module mux1bit2to1(a, b, s, out)
input a, b, s;
output out;

assign out = (~s & a) | (s & b);

endmodule
```

OR

```
module mux1bit2to1(a, b, s, out)
input a, b, s;
output out;
reg out; // used in procedural statement

always if (s == 0) out = a;
   else out = b;

endmodule
```

8-bit 4 to 1 Multiplexor

```
module mux8bit4to1(a,b,c,d,s,out)
input [7:0] a,b,c,d;
input [1:0] s;
output [7:0] out;
reg [7:0] out;

always case(s)
  2'b 00: out = a;
  2'b 01: out = b;
  2'b 10: out = c;
  2'b 11: out = d;
endcase

endmodule
```

17

Blocking/Non-Blocking Procedural Statements

- **Blocking assignment statement (= operator) acts much like in traditional programming languages**
  - The whole statement is done before control passes on to the next statement.
- **Non-blocking (<= operator) evaluates all the right-hand sides for the current time unit and assigns the left-hand sides at the end of the time unit**
  - **Example: During every clock cycle**
    - A is ahead of C by 1
    - B is same as D

```
// testing blocking and non-
// blocking assignment
module blocking( Clk, A, B);
input Clk;
output [7:0] A, B;

reg [7:0] A, B;
// as these will be used in
// procedural statements

reg [7:0] C, D
// two internal registers

always @(posedge Clk) begin
// blocking procedural assignment
C = C + 1;
A = C + 1;

// non-blocking procedural
// assignment
D <= D + 1;
B <= D + 1;
end
endmodule
```

18

Some Tips

- Can be downloaded from Xilinx (the webpage is given on the class webpage under tools) or MaxPlus from Altera
- Declare all variables, and one variable (especially input/output) per line
- Write your own test cases – see the example along with Xilinx distribution and help pages
- All the modules must follow the port list defined in the assignment

19