

Introduction to Digital Logic

Prof. Nizamettin AYDIN

naydin@yildiz.edu.tr
naydin@ieee.org

1

Verilog

Basics and Types of Descriptions

2

Overview

- Part 1
 - Verilog Basics
 - Notation
 - Keywords & Constructs
 - Operators
 - Types of Descriptions
 - Structural
 - Dataflow
 - Boolean Equations
 - Conditions using Binary Combinations
 - Conditions using Binary Decisions
 - Combinational Circuit Testbench

3

Verilog Notation - 1

- Verilog is:
 - Case sensitive
 - Based on the programming languages C and ADA
- Comments
 - Single Line
 - // [end of line]
 - Multiple Line
 - /* */
- List element separator:
- Statement terminator:

4

Verilog Notation - 2

- Binary Values for Constants and Variables
 - 0
 - 1
 - X or x – Unknown
 - Z or z – High impedance state (open circuit)
- Constants
 - n'b[integer]: 1'b1 = 1, 8'b1 = 00000001, 4'b0101=0101, 8'bxxxxxxxx, 8'bxxxx = 0000xxxx
 - n'h[integer]: 8'hA9 = 10101001, 16'hf1=0000000011110001
- Identifier Examples
 - Scalar: A,C,RUN,stop,m,n
 - Vector: sel[0:2], f[0:5], ACC[31:0], SUM[15:0], sum[15:0]

5

Verilog Keywords & Constructs - 1

- Keywords are lower case
- **module** – fundamental building block for Verilog designs
 - Used to construct design hierarchy
 - Cannot be nested
- **endmodule** – ends a module
 - not a statement, no “;”
- Module Declaration
 - **module** *module_name* (*module_port*, *module_port*, ...);
 - Example:
 module full_adder (A, B, c_in, c_out, S);

6

Verilog Keywords & Constructs - 2

- Input Declaration
 - Scalar
 - `input list of input identifiers;`
 - Example:


```
input A, B, c_in;
```
 - Vector
 - `input [range] list of input identifiers;`
 - Example:


```
input[15:0] A, B, data;
```
- Output Declaration
 - Scalar Example:


```
output c_out, OV, MINUS;
```
 - Vector Example:


```
output[7:0] ACC, REG_IN, data_out;
```

Verilog Keywords & Constructs - 3

Primitive Gates

- `buf, not, and, or, nand, nor, xor, xnor`
- Syntax: `gate_operator instance_identifier (output, input_1, input_2, ...)`
- Examples:


```
and A1 (F, A, B); //F = AB
or O1 (w, a, b, c),
O2 (x, b, c, d, e); //w=a+b+c,x=b+c+d+e
```

8

Verilog Operators - 1

- Bitwise Operators

<code>~</code>	NOT
<code>&</code>	AND
<code> </code>	OR
<code>^</code>	XOR
<code>^~ or ~^</code>	XNOR
- Example:


```
input[3:0] A, B;
output[3:0] Z ;
assign Z = A | ~B;
```

9

Verilog Operators - 2

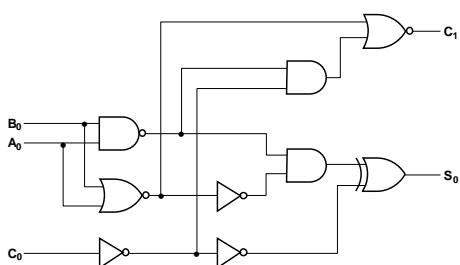
- Arithmetic Operators
 - `+, -, (plus others)`
- Logical & Relational Operators
 - `!, &&, ||, ==, !=, >=, <=, >, < (plus others)`
- Concatenation & Replication Operators
 - `{identifier_1, identifier_2, ...}`
 - `{n{identifier}}`
 - Examples:


```
{REG_IN[6:0],Serial_in},{8 {1'b0}}
```

10

1st Example to Illustrate Verilog Description Types

- IC7283 – a 1-bit adder from a commercial IC



11

Structural Verilog

- Circuits can be described by a netlist as a text alternative to a diagram – Example IC7283
- ```
module IC7283 (A0, B0, C0, C1, S0);
 input A0, B0, C0;
 output C1, S0;
// Seven internal wires needed
 wire N1, N2, N3, N4, N5, N6, N7;
// Ports on primitive gates - output port
// is listed first
 not G1 (N3,C0), G2 (N5,N2), G3 (N6,N3);
 nand G4 (N1,A0,B0);
 nor G5 (N2,A0,B0), G6 (C1,N2,N4);
 and G7 (N4,N1,N3), G8 (N7,N1,N5);
 xor G9 (S0,N6,N7);
endmodule
```

12

## Dataflow Verilog - 1

- Circuit function can be described by assign statements using Boolean equations – Example IC7283

```
module IC7283_df1 (A0, B0, C0, C1, S0);
 input A0, B0, C0;
 output C1, S0;
 wire N1, N2;
 assign N1 = ~(A0 & B0); //Note:
// Cannot write ~& for NAND
 assign N2 = ~(A0 | B0);
 assign C1 = ~((N1 & ~C0) | N2);
 assign S0 = (~N2 & N1)^(~(~C0));
endmodule
```

13

## 2nd Example to Illustrate Verilog Description Types

- Priority Encoder

| Inputs |    |    |    |    | Outputs |    |    |   |
|--------|----|----|----|----|---------|----|----|---|
| D4     | D3 | D2 | D1 | D0 | A2      | A1 | A0 | V |
| 0      | 0  | 0  | 0  | 0  | X       | X  | X  | 0 |
| 0      | 0  | 0  | 0  | 1  | 0       | 0  | 0  | 1 |
| 0      | 0  | 0  | 1  | X  | 0       | 0  | 1  | 1 |
| 0      | 0  | 1  | X  | X  | 0       | 1  | 0  | 1 |
| 0      | 1  | X  | X  | X  | 0       | 1  | 1  | 1 |
| 1      | X  | X  | X  | X  | 1       | 0  | 0  | 1 |

14

## Dataflow Verilog - 2

- Circuit function can be described by assign statements using the conditional operator with binary combinations as in a truth table – Example Priority Encoder

```
module priority_encoder_df2 (D, A, V);
 input[4:0] D;
 output[2:0] A;
 output V;
//Conditional:(X) ? Y: Z - if X is true, then Y,else Z
 assign A = (D[4] == 1'b1) ? 3'b100 :
 (D[3] == 1'b1) ? 3'b011 :
 (D[2] == 1'b1) ? 3'b010 :
 (D[1] == 1'b1) ? 3'b001 :
 (D[0] == 1'b1) ? 3'b000 : 3'bxxx;
 assign V = (D == 5'b00000) ? 1'b0 : 1'b1;
endmodule
```

15

## Dataflow Verilog - 3

- Circuit function can be described by assign statements using the conditional operator for binary decisions on inputs

```
module priority_encoder_df3 (D, A, V);
 input[4:0] D;
 output[2:0] A;
 output V;
// Conditional: (X) ? Y: Z If X is true,
// then Y, else Z
 assign A = D[4] ? 3'b100: D[3] ? 3'b011: D[2]
 ? 3'b010: D[1] ? 3'b001: D[0] ? 3'b000: 3'bxxx;
 assign V = (D == 5'b00000) ? 1'b0: 1'b1;
endmodule
```

16

## Combinational Circuit Testbench

- Testbench* – a program that is used to verify the correctness of an HDL representation for a circuit
  - Applies a set of input combinations to the circuit that thoroughly tests the HDL representation
  - Assists in verifying the correctness of the corresponding outputs:
    - By displaying the inputs and outputs for manual verification
    - By comparing the outputs to those of a known correct representation for the circuit

17

## Combinational Testbench Example - 1

- A Verilog testbench for a combinational circuit with **n** inputs and **m** outputs that:
  - applies all possible binary input combinations as stimuli to the circuit
  - provides the circuit outputs for manual verification
- Stimuli Generation
  - Uses a clocked binary up counter to generate stimuli
  - Stops the simulation when all combinations have been applied

18

## Combinational Testbench Example - 2

- Verilog code:

```
'timescale 1 ns / 100 ps //sets time unit /
// simulation interval (ps - picoseconds (10^{-12} sec)
module priority_encoder_testbench;
reg clk; //clock to advance counter
reg[4:0] stim; //stimuli counter storage for D[4:0]
wire[3:0] results; //wires to hold A[1:0],V
//Following instantiates the circuit being tested
priority_encoder_df3 xl (stim[4:0], results [3:1],
 results{0});
initial //initializes clk and stim to 0 and
//stops simulation after 33 combinations applied
begin
 clk <= 0;
 stim <= 4'b0;
 #330 $stop;
end
```

19

## Combinational Testbench Example - 3

- Verilog code (continued):

```
always
begin
#5 //waits 5 time units before toggling clock
forever
#5 clk <= ~clk; //toggles clock every 5 time units
end

always@(posedge clk) //conditions increment on
// the positive edge of clk
begin
stim <= stim + 1; //increments counter by 1
end
endmodule
```

20

## Verilog

### Behavioral and Hierarchical Descriptions

21

## Overview

- Part 2
  - Behavioral Descriptions
    - Using:
      - Assignment statements
      - Higher level operators
  - Verilog Hierarchy
    - Using:
      - Modules
      - Module instantiation
  - Example: Adder-Subtractor
    - Assignment statements using addition operator and vector XOR operator
    - Hierarchy with adder module and 1's completer module

22

## Behavioral & Hierarchical Verilog Example

- Circuit function can be described by assign statements at higher than the logic level:

```
module addsub (A, B, R, sub);
 input [3:0] A, B;
 output [3:0] R;
 input sub; //sub ? subtract : add
 wire[3:0] data_out;
//Instantiate add and M1comp modules
 add A1 (A, data_out, sub, R);
 M1comp C1 (B, data_out, sub);
endmodule
```

23

## Behavioral & Hierarchical Verilog Example (continued)

```
module add (X, Y, C_in, S);
 input [3:0] X, Y;
 input C_in;
 output [3:0] S;
 assign S = X + Y + {3'b0, C_in};
endmodule
module M1comp (data_in, data_out, comp);
 input[3:0] data_in;
 input comp;
 output [3:0] data_out;
 assign data_out = {4{comp}} ^ data_in;
endmodule // {n{x}} means concatenate
// n copies of x
```

24

## Verilog

### Finite State Machines

25

## Overview

- Part 3
  - Process (Procedural) Description
  - Verilog Keywords and Constructs
  - Process Verilog for a Positive Edge-triggered D Flip-flop
  - Process Verilog for Figure 6-19(a)

26

## Process (Procedural) Description

- So far, we have done dataflow and behavioral Verilog using continuous assignment statements (`assign`)
- Continuous assignments are limited in the complexity of what can be described
- A process can be viewed as a replacement for a continuous assignment statement that permits much more complex descriptions
- A process uses procedural assignment statements much like those in a typical programming language

27

## Verilog Keywords & Constructs - 1

- Because of the use of procedural rather than continuous assignment statements, assigned values must be retained over time.
  - Register type: `reg`
  - The `reg` in contrast to `wire` stores values between executions of the process
  - A `reg` type does not imply hardware storage!
- Process types
  - `initial` – executes only once beginning at  $t = 0$ .
  - `always` – executes at  $t = 0$  and repeatedly thereafter.
  - Timing or event control is exercised over an always process using, for example, the `@` followed by an event control statement in  $( )$ .

28

## Verilog Keywords & Constructs - 2

- Process begins with `begin` and ends with `end`.
- The body of the process consists of procedural assignments
  - Blocking assignments**
    - Example: `C = A + B;`
    - Execute sequentially as in a programming language
  - Non-blocking assignments**
    - Example: `C <= A + B;`
    - Evaluate right-hand sides, but do not make any assignment until all right-hand sides evaluated. Execute concurrently unless delays are specified.

29

## Examples

```
always
begin
 B = A;
 C = B;
end
- Suppose initially A = 0, B = 1, and C = 2. After execution, B = 0 and C = 0.

always
begin
 B <= A;
 C <= B;
end
- Suppose initially A = 0, B = 1, and C = 2. After execution, B = 0 and C = 1.
```

30

### Verilog Keywords & Constructs - 3

- Conditional constructs

- The **if-else**

```
if (condition)
 begin procedural statements end
{else if (condition)
 begin procedural statements end}
else
 begin procedural statements end
```
- The **case**
  - case expression
  - {case expression : statements}
  - endcase;

31

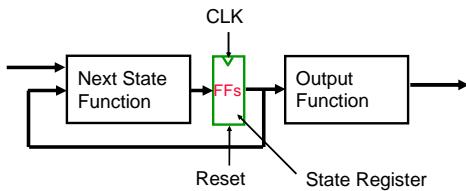
### Verilog for Positive Edge-Triggered D Flip-Flop

```
module dff (CLK, RESET, D, Q)
 input CLK, RESET, D;
 output Q;
 reg Q;
 always@ (posedge CLK or posedge RESET)
 begin
 if (RESET)
 Q <= 0;
 else
 Q <= D;
 end
endmodule
```

32

### Describing Sequential Circuits

- There are many different ways to organize models for sequential circuits. We will use a model that corresponds to the following diagram:



- A process corresponds to each of the 3 blocks in the diagram.

33

### Verilog for Figure 6-19(a) State Diagram

```
module fig_619 (CLK, RESET, X, Z);
 input CLK, RESET, X;
 output Z;
 reg[1:0] state, next_state;
 parameter S0 = 2'b00, S1 = 2'b01,
 S2 = 2'b10, S3 = 2'b11;
//state register
 always@(posedge CLK or posedge RESET)
 begin
 if (RESET == 1)
 state <= S0;
 else
 state <= next_state;
 end
end
```

34

### Verilog State Diagram (continued)

```
//next state function
always@(X or state)
begin
 case (state)
 S0: if (X == 1) next_state <= S1;
 else next_state <= S0;
 S1: if (X == 1) next_state <= S3;
 else next_state <= S0;
 S2: if (X == 1) next_state <= S2;
 else next_state <= S0;
 S3: if (X == 1) next_state <= S2;
 else next_state <= S0;
 default: next_state <= 2'bxx;
 endcase
end
```

35

### Verilog State Diagram (continued)

```
reg Z;
//output function
always@(X or state)
begin
 case (state)
 S0: Z <= 1'b0;
 S1: if (X == 1) Z <= 1'b0;
 else Z <= 1'b1;
 S2: if (X == 1) Z <= 1'b0;
 else Z <= 1'b1;
 S3: if (X == 1) Z <= 1'b0;
 else Z <= 1'b1;
 default: Z <= 1'bx;
 endcase
end
endmodule
```

36

## Verilog

### Registers and Counters

37

## Overview

- Part 4
  - Registers
  - Shift Registers
  - Counters
  - Example
    - 4-bit Left Shift Register with Reset
    - 4-bit Binary Counter with Reset

38

## Verilog for Registers and Counters

- Register – same as flip-flop except multiple bits:

```
reg[3:0] Q;
 input[3:0] D;
 always@(posedge CLK or posedge RESET)
 begin
 if (RESET) Q <= 4'b0000;
 else Q <= D;
 end
```
- Shift Register – use concatenate:  

```
Q <= {Q[2:0], SI};
```
- Counter – use increment/decrement:  

```
count <= count + 1; //increment
count <= count - 1; //decrement
```

39

## Verilog Description of Left Shift Register

```
// 4-bit Shift Register
// with Reset
always@(posedge CLK or
posedge RESET)
begin
 if (RESET)
 Q <= 4'b0000;
 else
 Q <= {Q[2:0], SI};
end
```

40

## Verilog Description of Binary Counter

```
// 4-bit Binary Counter with Reset
module count_4_r_v (CLK, RESET,
EN, Q, CO);
 input CLK, RESET, EN;
 output [3:0] Q;
 output CO;
 reg [3:0] count;
 assign Q = count;
 assign CO = (count == 4'b1111 && EN == 1'b1) ? 1 : 0;
 always@(posedge CLK or
posedge RESET)
 begin
 if (RESET)
 count <= 4'b0;
 else if (EN)
 count <= count +
1;
 end
endmodule
```

41

## Verilog

Algorithmic State Machine Example:  
Binary Multiplier

42

## Overview

- Part 5

- Conversion of ASM into Verilog description
  - Decompose into:
    - Sequencing - synchronous always for state, combinational always for next state
    - Output values - combinational always
    - Register transfers - synchronous always
  - Use parameters to make state assignment
  - Use case for more than two states
  - Use if then else for scalar decisions
  - Use case for vector decisions

-Illustrate using binary multiplier ASM chart in Figure 8-8 of text

## Verilog for Alternative Binary Multiplier – 1

```
//Alternative Binary Multiplier with n = 4
//See Figure 8-8 of text
module alt_bin_multiplier (CLK, RESET, G, LOADB,
 LOADQ, MULT_IN, MULT_OUT);
input CLK, RESET, G, LOADB, LOADQ;
input [3:0] MULT_IN;
output[7:0] MULT_OUT;
reg state, next_state;
parameter IDLE = 0, MUL = 1;
reg [1:0] P;
reg [3:0] A, B, Q;
wire Z;
// Test P for value 0
assign Z = ~| P; // ~| is reduction NOR
// that NORs together all bits of P
```

44

## Verilog for Alternative Binary Multiplier – 2

```
assign MULT_OUT = {A,Q};
//state register
always@(posedge CLK or posedge RESET)
begin
 if (RESET == 1)
 state <= IDLE;
 else
 state <= next_state;
end
//next state function
always@(G or Z or state)
begin
 if (state == IDLE)
 if (G == 1)
 next_state <= MUL;
```

45

## Verilog for Alternative Binary Multiplier – 3

```
else
 next_state <= IDLE;
else // state = MUL
 if (Z == 1)
 next_state <= IDLE;
 else
 next_state <= MUL;
end
//register transfers
always@(posedge CLK or posedge RESET)
begin
 if (LOADB)
 B <= MULT_IN;
 else
 if (LOADQ)
 Q <= MULT_IN;
 else
```

46

## Verilog for Alternative Binary Multiplier – 4

```
if (state == IDLE && G == 1)
begin
 P <= 2'b11;
 A <= 4'b0000;
end
else
if (state == MUL)
begin
 P <= P - 1;
 if (Q[0] == 1)
 {A,Q} <= {{1'b0,A},Q[3:1]};
 //1'b0 is concatenated on the left to acquire the Cout
 //value.
 else
 {A,Q} <= {1'b0,A,Q[3:1]};
end
end
endmodule
```

47