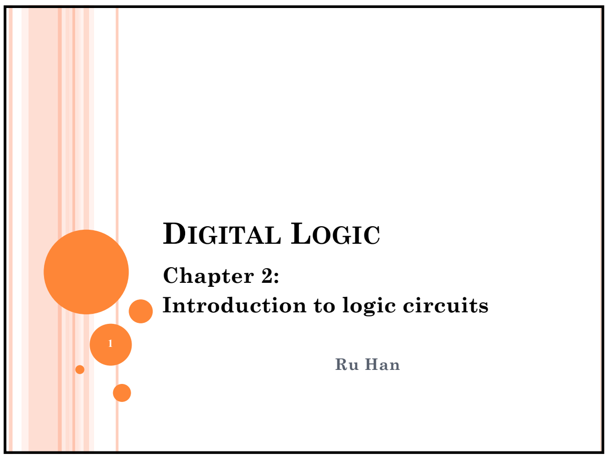
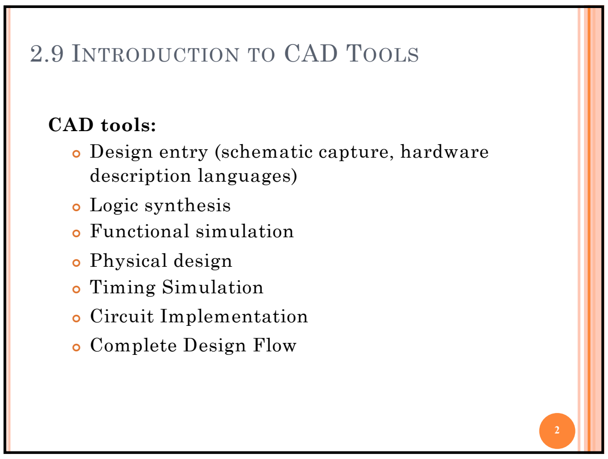


DIGITAL LOGIC

Chapter 2:
Introduction to logic circuits

Ru Han

1

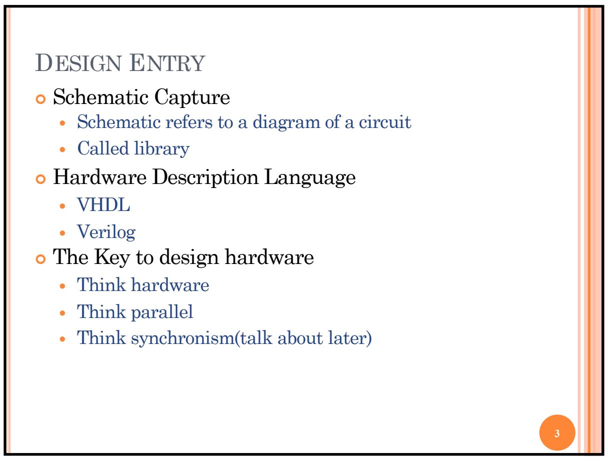


2.9 INTRODUCTION TO CAD TOOLS

CAD tools:

- Design entry (schematic capture, hardware description languages)
- Logic synthesis
- Functional simulation
- Physical design
- Timing Simulation
- Circuit Implementation
- Complete Design Flow

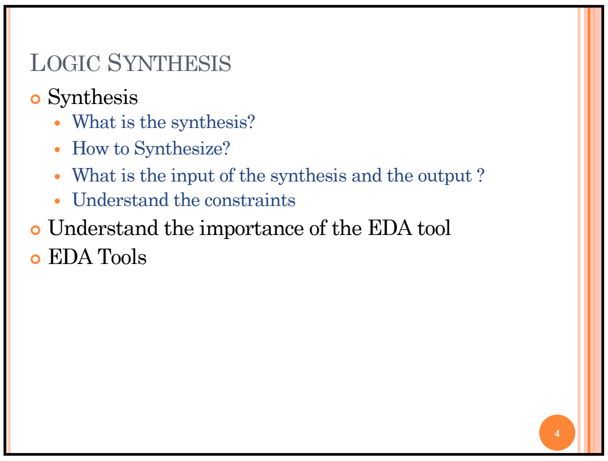
2



DESIGN ENTRY

- Schematic Capture
  - Schematic refers to a diagram of a circuit
  - Called library
- Hardware Description Language
  - VHDL
  - Verilog
- The Key to design hardware
  - Think hardware
  - Think parallel
  - Think synchronism(talk about later)

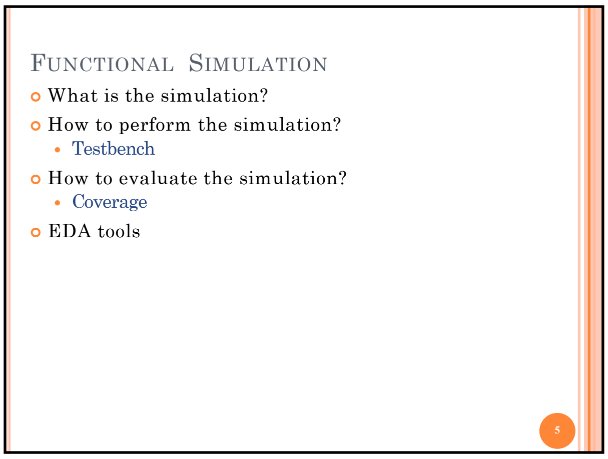
3



LOGIC SYNTHESIS

- Synthesis
  - What is the synthesis?
  - How to Synthesize?
  - What is the input of the synthesis and the output ?
  - Understand the constraints
- Understand the importance of the EDA tool
- EDA Tools

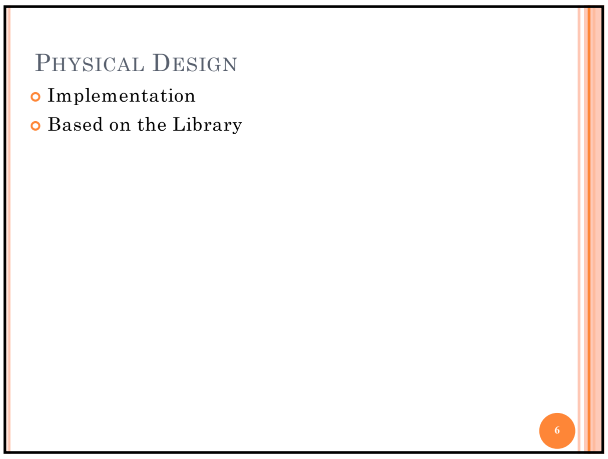
4



FUNCTIONAL SIMULATION

- What is the simulation?
- How to perform the simulation?
  - Testbench
- How to evaluate the simulation?
  - Coverage
- EDA tools

5



PHYSICAL DESIGN

- Implementation
- Based on the Library

6

Timing Simulation

- What is the input of this step?
- Propagation Delay(cell delay + net Delay)
- Try to simulate the True Circuit based on the library

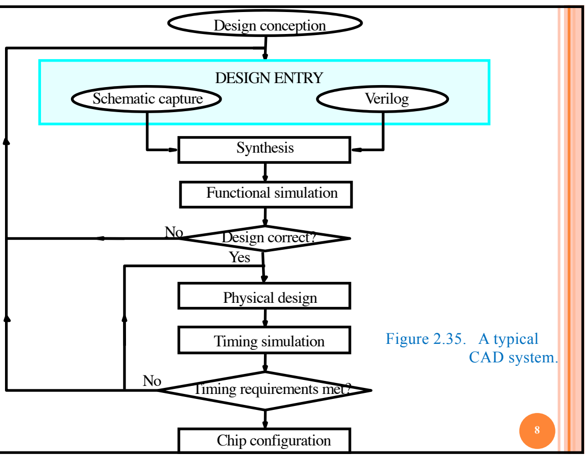


Figure 2.35. A typical CAD system.

2.10 Introduction to Verilog

- **Representation of Digital Circuits in Verilog**

- Verilog allows the designer to describe a desired circuit in a number of ways:

(1) *Structural* representation of logic circuits

Use Verilog constructs that describe the structure of the circuit in terms of circuit elements, such as logic gates. A larger circuit is defined by writing code that connects such elements together.

(2) *Behavioral* representation of logic circuits

Describe a circuit more abstractly, by using logic expressions and Verilog programming constructs that define the desired behavior of the circuit, but not its actual structure in terms of gates.

Structural Specification of Logic Circuits

- Verilog includes a set of *gate-level primitives* that correspond to commonly-used logic gates. A gate is represented by indicating its functional name, output, and inputs.

and (y, x1, x2);

or (y, x1, x2, x3, x4);

not (y, x);

Structural Specification of Logic Circuits

- A logic circuit is specified in the form of a *module* that contains the statements that define the circuit.
- A module has inputs and outputs, which are referred to as its *ports*.
- Example 1:

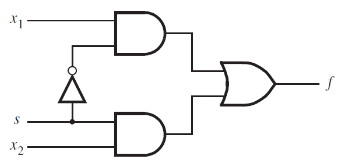
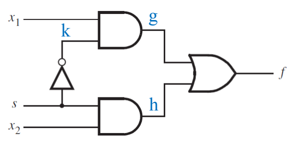


Figure 2.36. The logic circuit for a multiplexer.

Structural Specification of Logic Circuits



```
module  example1 (x1, x2, s, f);
    input  x1, x2, s;
    output  f;

    not (k, s);
    and (g, k, x1);
    and (h, s, x2);
    or (f, g, h);

endmodule
```

Structural Specification of Logic Circuits

- Example 2:

```
module example2 (x1, x2, x3, x4, f, g, h);
        input x1, x2, x3, x4;
        output f, g, h;

        and (z1, x1, x3);
        and (z2, x2, x4);
        or (g, z1, z2);
        or (z3, x1, ~x3);
        or (z4, ~x2, x4);
        and (h, z3, z4);
        or (f, g, h);

endmodule
```

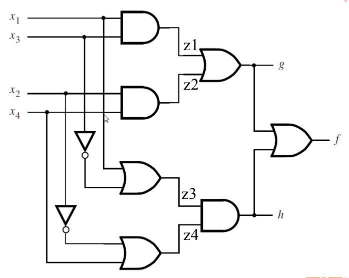


Figure 2.39. Logic circuit for the code in Figure 2.38.

Structural Specification of Logic Circuits

- **Verilog Syntax**
- The names of modules and signals in Verilog code follow two simple rules:

(1) The name must start with a letter, and it can contain any letter or number plus the "_" underscore and "$" characters.

(2) Verilog is case sensitive.

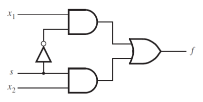
A comment begins with the double slash "// " and continues to the end of the line.

14

Behavioral Specification of Logic Circuits

- Use more abstract expressions and programming constructs to describe the behavior of a logic circuit.
- Example 3:

$$f = \overline{s}x_1 + sx_2$$



```
module  example3 (x1, x2, s, f);
   input x1, x2, s;
   output f;

   assign  f = (~s & x1) | (s & x2);

endmodule
```

Figure 2.40. Using the continuous assignment to specify the circuit in Figure 2.36.

15

Behavioral Specification of Logic Circuits

- Example 4:

```
module example4 (x1, x2, x3, x4, f, g, h);
        input x1, x2, x3, x4;
        output f, g, h;

        assign g = (x1 & x3) | (x2 & x4);
        assign h = (x1 | ~x3) & (~x2 | x4);
        assign f = g | h;

endmodule
```

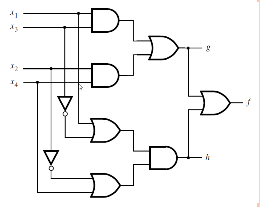
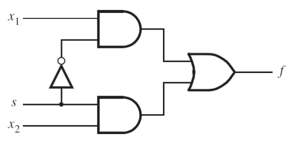


Figure 2.41. Using the continuous assignment to specify the circuit in Figure 2.39.

16

Behavioral Specification of Logic Circuits

- Example 5:



```
// Behavioral specification
module  example5 (x1, x2, s, f);
   input  x1, x2, s;
   output f;
   reg f;

   always @(x1 or x2 or s)
      if (s == 0)
         f = x1;
      else
         f = x2;

endmodule
```

Figure 2.42. Behavioral specification of the circuit in Figure 2.36.

17

Behavioral Specification of Logic Circuits

- The part of the **always** block after the @ symbol, in parentheses, is called the *sensitivity list*.
- The statements inside an **always** block are executed by the simulator only when one or more of the signals in the sensitivity list changes value.
- If a signal is assigned a value using procedural statements, then Verilog syntax requires that it be declared as a *variable*; this is accomplished by using the keyword **reg** in Figure 2.42.

```
output reg f;
always @(*)
always @*
```

18

Behavioral Specification of Logic Circuits

- The **if-else** statement is an example of a Verilog *procedural* statement.
- Verilog syntax requires that procedural statements be contained inside a construct called an **always** block.
- An important property of the **always** block is that the statements it contains are evaluated in the order given in the code.
- The continuous assignment statements, which are evaluated concurrently and hence have no meaningful order.

Behavioral Specification of Logic Circuits

Should be declared as reg explicitly

```
// Behavioral specification
module example5 (x1, x2, s, f);
   input x1, x2, s;
   output f;
   reg f;

   always @(x1 or x2 or s)
      if (s == 0)
         f = x1;
      else
         f = x2;

endmodule
```

Figure 2.42. Behavioral specification of the circuit in Figure 2.36.

```
// Behavioral specification
module example5 (input x1, x2, s, output reg f);

   always @(x1, x2, s)
      if (s == 0)
         f = x1;
      else
         f = x2;

endmodule
```

Figure 2.43. A more compact version of the code in Figure 2.42.

Hierarchical Verilog Code

- For larger designs, it is often convenient to create a hierarchical structure in the Verilog code, in which there is a *top-level* module that includes multiple instances of *lower-level* modules.

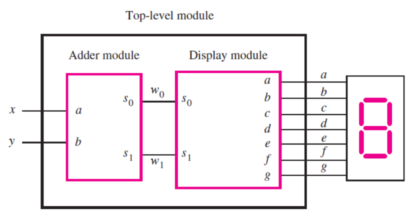


Figure 2.44. A logic circuit with two modules.

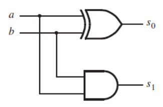
Hierarchical Verilog Code

$$\begin{array}{r} a \\ +\,b \\ \hline s_1 s_0 \end{array} \qquad \begin{array}{r} 0 \\ +\,0 \\ \hline 0\,0 \end{array} \quad \begin{array}{r} 0 \\ +\,1 \\ \hline 0\,1 \end{array} \quad \begin{array}{r} 1 \\ +\,0 \\ \hline 0\,1 \end{array} \quad \begin{array}{r} 1 \\ +\,1 \\ \hline 1\,0 \end{array}$$

(a) Evaluation of $S = a + b$

a	b	s_1	s_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table



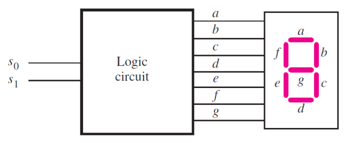
(c) Logic network

```
// An adder module
module adder (a, b, s1, s0);
   input a, b;
   output s1, s0;

   assign s1 = a & b;
   assign s0 = a ^ b;

endmodule
```

Default type is wire

Figure 2.45. Verilog specification of the circuit in Figure 2.12.

Number Display



(a) Logic circuit and 7-segment display

	s_1	s_0	a	b	c	d	e	f	g
0	0	0	1	1	1	1	1	1	0
1	0	1	0	1	1	0	0	0	0
2	1	0	1	1	0	1	1	0	1

(b) Truth table

$a = d = e = \overline{s_0}$
$b = 1$
$c = \overline{s_1}$
$f = \overline{s_1}\,\overline{s_0}$
$g = s_1 \overline{s_0}$

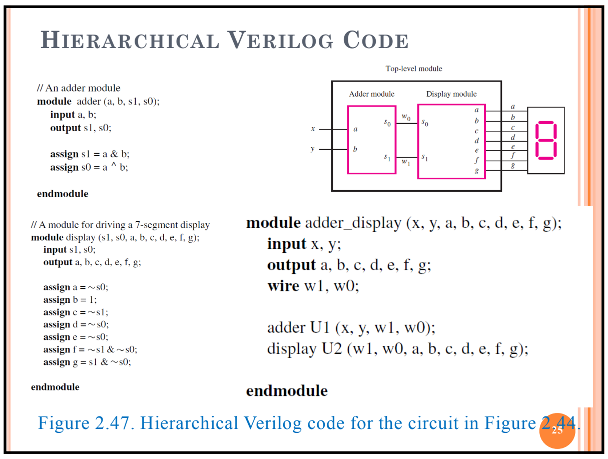
Figure 2.34. Display of numbers.

Hierarchical Verilog Code

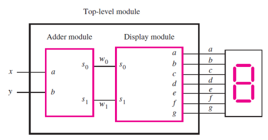
$a = d = e = \overline{s_0}$
$b = 1$
$c = \overline{s_1}$
$f = \overline{s_1}\,\overline{s_0}$
$g = s_1 \overline{s_0}$

```
// A module for driving a 7-segment display
module display (s1, s0, a, b, c, d, e, f, g);
   input s1, s0;
   output a, b, c, d, e, f, g;

   assign a = ~s0;
   assign b = 1;
   assign c = ~s1;
   assign d = ~s0;
   assign e = ~s0;
   assign f = ~s1 & ~s0;
   assign g = s1 & ~s0;

endmodule
```

Figure 2.46. Verilog specification of the circuit in Figure 2.34.



Hierarchical Verilog Code

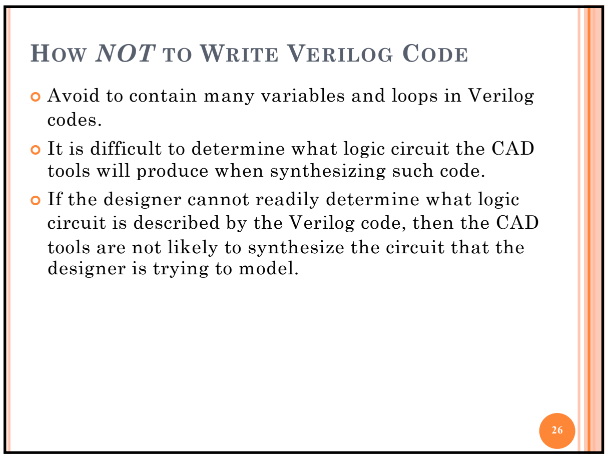


```
// An adder module
module  adder (a, b, s1, s0);
    input a, b;
    output s1, s0;

    assign s1 = a & b;
    assign s0 = a ^ b;

endmodule
```

```
// A module for driving a 7-segment display
module display (s1, s0, a, b, c, d, e, f, g);
   input s1, s0;
   output a, b, c, d, e, f, g;

   assign a = ~s0;
   assign b = 1;
   assign c = ~s1;
   assign d = ~s0;
   assign e = ~s0;
   assign f = ~s1 & ~s0;
   assign g = s1 & ~s0;

endmodule
```

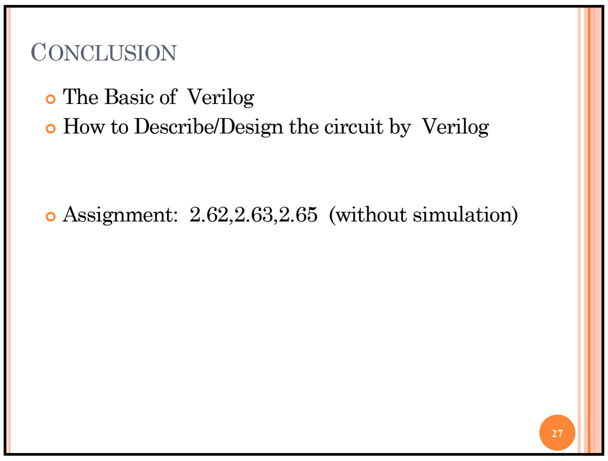
```
module adder_display (x, y, a, b, c, d, e, f, g);
   input x, y;
   output a, b, c, d, e, f, g;
   wire w1, w0;

   adder U1 (x, y, w1, w0);
   display U2 (w1, w0, a, b, c, d, e, f, g);

endmodule
```

Figure 2.47. Hierarchical Verilog code for the circuit in Figure 2.44.



How *NOT* to Write Verilog Code

- Avoid to contain many variables and loops in Verilog codes.
- It is difficult to determine what logic circuit the CAD tools will produce when synthesizing such code.
- If the designer cannot readily determine what logic circuit is described by the Verilog code, then the CAD tools are not likely to synthesize the circuit that the designer is trying to model.



Conclusion

- The Basic of Verilog
- How to Describe/Design the circuit by Verilog

- Assignment: 2.62,2.63,2.65 (without simulation)