CS204: 數位系統設計

Combinational Logic

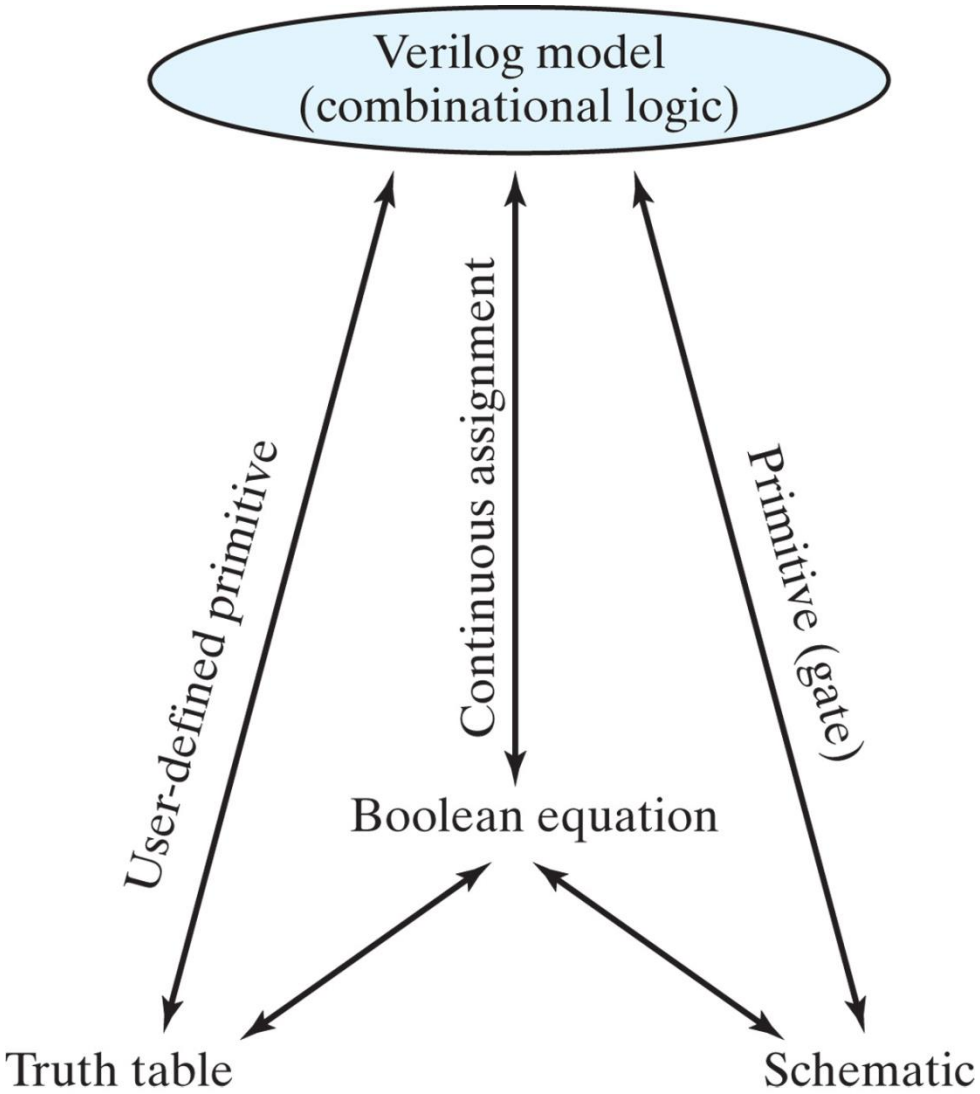
Outline of Chapter 4

- 4.1 Introduction
- 4.2 Combination Circuits
- 4.3 Analysis Procedure
- 4.4 Design Procedure
- 4.5 Binary Adder-Subtractor
- 4.6 Decimal Adder
- 4.7 Binary Multiplier
- 4.8 Magnitude Comparator
- 4.9 Decoders
- 4.10 Encoders
- 4.11 Multiplexers
- **4.12 HDL Models of Combination Circuits**

4-12 HDL Models of Combinational Circuits

- **Modeling Styles**
  - **Gate-level modeling using instantiations of predefined and user-defined primitive gates**
  - **Dataflow modeling using continuous assignment statements with the keyword** assign
  - **Behavioral modeling using procedural assignment statements with the keyword** always

Relationship of Verilog Constructs



Gate-level Modeling

- **The four-valued logic truth tables for the** and**,** or**,** xor**, and** not **primitives**

Table 4.9
Truth Table for Predefined Primitive Gates

and	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

or	0	1	x	z
0	0	1	x	x
1	1	1	1	1
x	x	1	x	x
z	x	1	x	x

Controlling value

xor	0	1	x	z
0	0	1	x	x
1	1	0	x	x
x	x	x	x	x
z	x	x	x	x

not	input	output
	0	1
	1	0
	x	x
	z	x

X is unknown
Z is high impedance

Gate-level Modeling

- **Example:**

MSB LSB

```
output  [0: 3]  D;
wire    [7: 0]  SUM;
```

- **The first statement declares an output vector D with four bits, 0 through 3**
- **The second declares a wire vector SUM with eight bits numbered 7 through 0**

HDL Example 4-1

- **Two-to-four-line decoder**

HDL Example 4.1

```
// Gate-level description of two-to-four-line decoder
// Refer to Fig. 4.19 with symbol E replaced by enable, for clarity.

module decoder_2x4_gates (D, A, B, enable);
  output      [0: 3]     D;
  input                  A, B;
  input                  enable;
  wire                   A_not, B_not, enable_not;

  not
    G1  (A_not, A),
    G2  (B_not, B),
    G3  (enable_not, enable);
  nand
    G4  (D[0], A_not, B_not, enable_not),
    G5  (D[1], A_not, B, enable_not),
    G6  (D[2], A, B_not, enable_not),
    G7  (D[3], A, B, enable_not);
endmodule
```

HDL Example 4-2

- **Four-bit adder: bottom-up hierarchical description**

HDL Example 4.2

```
// Gate-level description of four-bit ripple carry adder
// Description of half adder (Fig. 4.5b)

//  module half_adder (S, C, x, y);                    // Verilog 1995 syntax
//  output  S, C;
//  input    x, y;

module half_adder (output S, C, input x, y);           // Verilog 2001, 2005 syntax
 // Instantiate primitive gates
  xor (S, x, y);
  and (C, x, y);
endmodule

// Description of full adder (Fig. 4.8)                // Verilog 1995 syntax
//  module full_adder (S, C, x, y, z);
//  output          S, C;
//  input           x, y, z;
```

HDL Example 4-2 (continued)

```
module full_adder (output S, C, input x, y, z);          // Verilog 2001, 2005 syntax
  wire S1, C1, C2;

// Instantiate half adders
  half_adder HA1 (S1, C1, x, y);
  half_adder HA2 (S, C2, S1, z);
  or G1 (C, C2, C1);
endmodule

// Description of four-bit adder (Fig. 4.9)                  // Verilog 1995 syntax
//  module ripple_carry_4_bit_adder (Sum, C4, A, B, C0);
//  output [3: 0]  Sum;
//  output         C4;
//  input  [3: 0]  A, B;
//  input          C0;
// Alternative Verilog 2001, 2005 syntax:

module ripple_carry_4_bit_adder ( output [3: 0] Sum, output C4,
  input [3: 0]  A, B, input C0);
  wire           C1, C2, C3;                  // Intermediate carries
// Instantiate chain of full adders
  full_adder     FA0 (Sum[0], C1, A[0], B[0], C0),
                 FA1 (Sum[1], C2, A[1], B[1], C1),
                 FA2 (Sum[2], C3, A[2], B[2], C2),
                 FA3 (Sum[3], C4, A[3], B[3], C3);
endmodule
```

Three-State Gates

- **Statement:** gate name (output, input, control);

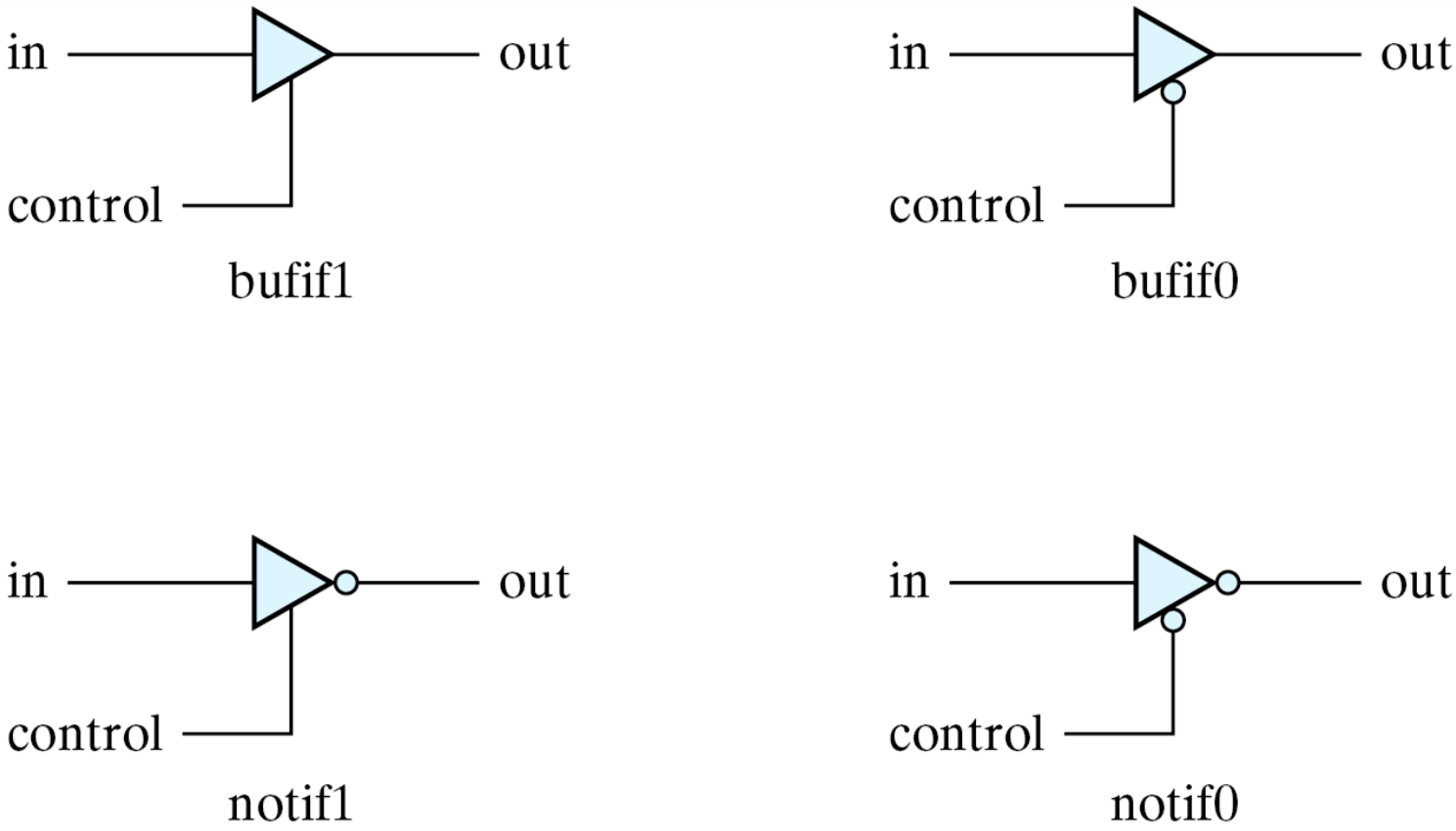


Fig. 4.31 Three-state gates

Three-State Gates

- **Examples of gate instantiation**

```
bufif1  (OUT, A, control);
notif0  (Y, B, enable);
```

The HDL description must use a **tri** data type for the output:

```
// Mux with three-state output

module mux_tri (m_out, A, B, select);
  output m_out;
  input  A, B, select;
  tri    m_out;

  bufif1 (m_out, A, select);
  bufif0 (m_out, B, select);
endmodule
```

Keywords **wire** and **tri** are examples of a set of data types called *nets*, which represent connections between hardware elements. In simulation, their value is determined by a continuous assignment statement or by the device whose output they represent. The word *net* is not a keyword, but represents a class of data types, such as **wire**, **wor**, **wand**, **tri**, **supply1**, and **supply0**. The **wire** declaration is used most frequently. In fact, if an identifier is used, but not declared, the language specifies that it will be interpreted (by default) as a **wire**. The net **wor** models the hardware implementation of the wired-OR configuration (emitter-coupled logic). The **wand** models the wired-AND configuration (open-collector technology; see Fig. 3.28). The nets **supply1** and **supply0** represent power supply and ground, respectively. They are used to hardwire an input of a device to either 1 or 0.

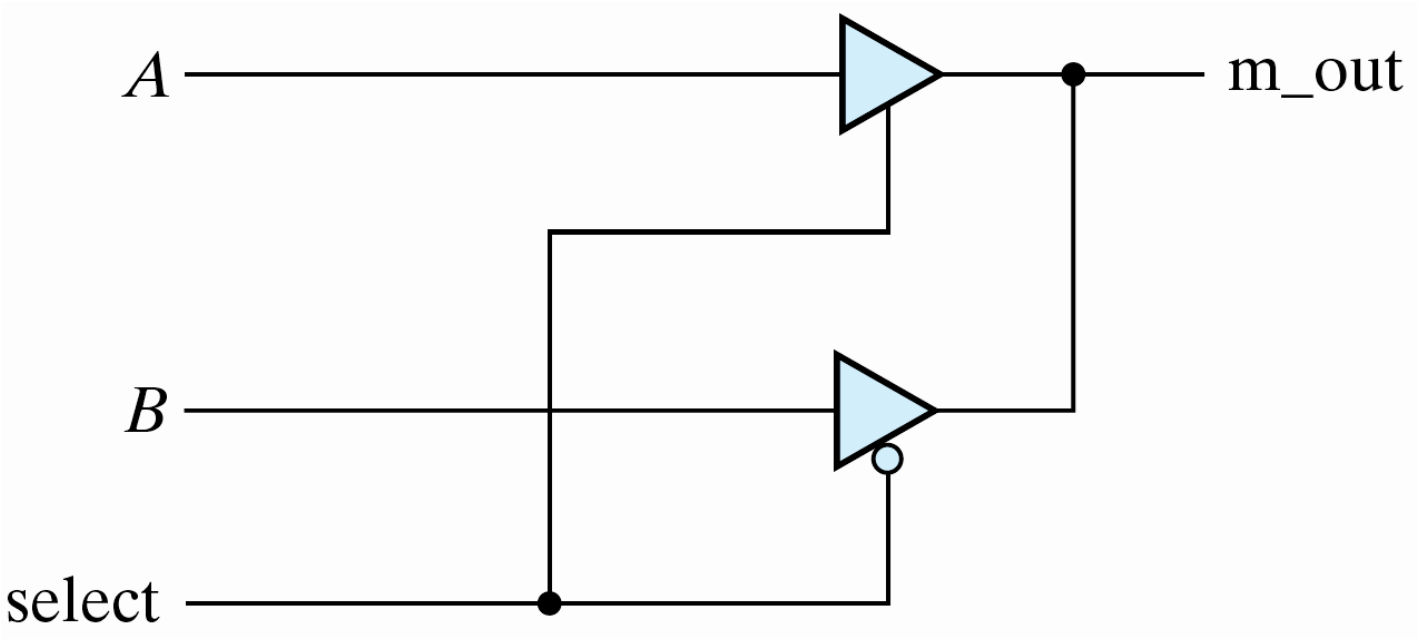


Fig. 4.32 Two-to-one-line multiplexer with three-state buffers

Dataflow Modeling

Verilog HDL operators

Example:

assign Y = (A & S) | (B & ~S)

Table 4.10
Some Verilog HDL Operators

Symbol	Operation	Symbol	Operation
+	binary addition		
–	binary subtraction		
&	bitwise AND	&&	logical AND
|	bitwise OR	||	logical OR
^	bitwise XOR		
~	bitwise NOT	!	logical NOT
= =	equality		
>	greater than		
<	less than		
{}	concatenation		
?:	conditional		

HDL Example 4.3

- **Dataflow description of a 2-to-4-line decoder**

HDL Example 4.3

```
// Dataflow description of two-to-four-line decoder

// See Fig. 4.19. Note: The figure uses symbol E, but the
// Verilog model uses enable to clearly indicate functionality.

module decoder_2x4_df (                    // Verilog 2001, 2005 syntax
  output      [0: 3]      D,
  input                   A, B,
                          enable
);

  assign      D[0] = ~(~A & ~B & ~enable),
              D[1] = ~(~A & B & ~enable),
              D[2] = ~(A & ~B & ~enable),
              D[3] = ~(A & B & ~enable);
endmodule
```

HDL Example 4-4

- **Dataflow description of 4-bit adder**

HDL Example 4.4

```
// Dataflow description of four-bit adder

// Verilog 2001, 2005 module port syntax

module binary_adder (
  output [3: 0]          Sum,
  output                 C_out,
  input  [3: 0]          A, B,
  input                  C_in
);

  assign {C_out, Sum} = A + B + C_in;
endmodule
```

HDL Example 4-5

- **Dataflow description of 4-bit magnitude comparator**

HDL Example 4.5

```
// Dataflow description of a four-bit comparator        //V2001, 2005 syntax

module mag_compare
( output          A_lt_B, A_eq_B, A_gt_B,
  input [3: 0]    A, B
);
  assign A_lt_B = (A < B);
  assign A_gt_B = (A > B);
  assign A_eq_B = (A == B);
endmodule
```

HDL Example 4-6

- **Dataflow description of a 2-to-1-line multiplexer**

HDL Example 4.6

```
// Dataflow description of two-to-one-line multiplexer

module mux_2x1_df(m_out, A, B, select);
  output        m_out;
  input         A, B;
  input         select;

  assign m_out = (select)? A : B;
endmodule
```

- **Conditional operator (?:)**

Condition ? True-expression : false-expression

Example: continuous assignment

assign *OUT = select ? A : B*

Behavioral Modeling

- **Behavioral modeling represents digital circuits at a functional and algorithmic level**
  - **Both sequential and combinational circuits**
  - **Keyword: always, followed by an optional event control @ expression**
  - **reg data type: retains its value until a new value is assigned**
    - **Pitfall: reg data type does not imply a register!**
    - **wire data type: continuously updates**

HDL Example 4-7

- **if statement:**
  - if **(select) OUT = A;**

- **HDL Example 4-7**
  - **Behavioral description of a 2-to-1-line multiplexer**

HDL Example 4.7

```
// Behavioral description of two-to-one-line multiplexer

module mux_2x1_beh (m_out, A, B, select);
  output      m_out;
  input       A, B, select;
  reg         m_out;

  always      @(A or B or select)
    if (select == 1) m_out = A;
    else m_out = B;
  endmodule
```

HDL Example 4-8

- **Behavioral description of a 4-to-1-line multiplexer**

HDL Example 4.8

```
// Behavioral description of four-to-one line multiplexer

// Verilog 2001, 2005 port syntax

module mux_4x1_beh
( output reg  m_out,
  input       in_0, in_1, in_2, in_3,
  input [1: 0]  select
);
  always @ (in_0, in_1, in_2, in_3, select)        // Verilog 2001, 2005 syntax
    case (select)
      2'b00:          m_out = in_0;
      2'b01:          m_out = in_1;
      2'b10:          m_out = in_2;
      2'b11:          m_out = in_3;
    endcase
  endmodule
```

Writing a Simple Test Bench (1/4)

- Initial **block**

```
initial
  begin
          A = 0; B = 0;
   #10  A = 1;
   #20  A = 0; B = 1;
  end
```

```
initial
  begin
    D = 3'b000;
    repeat (7)
      #10  D = D + 3'b001;
  end
```

→ Three-bit truth table

Writing a Simple Test Bench (2/4)

- **Interaction between stimulus and design modules**

```
module t_circuit;
  reg t_A, t_B;
  wire t_C;
  parameter stop_time = 1000 ;

 circuit M ( t_C, t_A, t_B );

// Stimulus generators for
// t_A and t_B go here
  initial # stop_time $finish;
endmodule
```

```
module circuit ( C, A, B )

  input        A, B;

  output       C;

// Description goes here
endmodule
```

Writing a Simple Test Bench (3/4)

- **Stimulus module**

```
module test_module_name;
  // Declare local reg and wire identifiers.
  // Instantiate the design module under test.
  // Specify a stopwatch, using $finish to terminate the simulation.
  // Generate stimulus, using initial and always statements.
  // Display the output response (text or graphics (or both)).
endmodule
```

- **System tasks for display**

$display—display a one-time value of variables or strings with an end-of-line return,
$write**—same as **$display, but without going to next line,
$monitor—display variables whenever a value changes during a simulation run,
$time—display the simulation time,
$finish—terminate the simulation.

Writing a Simple Test Bench (4/4)

- **Syntax for** $dispaly, $write, **and** $monitor**:**

Task-name (format specification, argument list);

- **Example:**

```
$display ("%d %b %b", C, A, B);
```

- **Example:**

```
$display ("time = %0d A = %b B = %b", $time, A, B);
```

→

```
time = 3  A = 10  B = 1
```

HDL Example 4-9

- **Stimulus module**

HDL Example 4.9

```
// Test bench with stimulus for mux_2x1_df

module t_mux_2x1_df;
  wire        t_mux_out;
  reg         t_A, t_B;
  reg         t_select;
  parameter   stop_time = 50;

  mux_2x1_df  M1 (t_mux_out, t_A, t_B, t_select);     // Instantiation of circuit to be tested

initial # stop_time $finish;

initial begin                                          // Stimulus generator
        t_select = 1; t_A = 0; t_B = 1;
  #10   t_A = 1; t_B = 0;
  #10   t_select = 0;
  #10   t_A = 0; t_B = 1;
end
```

HDL Example 4-9 (Continued)

```
  initial begin                                          // Response monitor
    // $display ("    time   Select  A     B    m_out");
    // $monitor ($time,, "   %b    %b    %b    %b", t_select, t_A, t_B, t_m_out);
    $monitor ("time=", $time,, "select = %b A = %b B = %b OUT = %b",
    t_select, t_A, t_B, t_mux_out);
  end
endmodule

//  Dataflow description of two-to-one-line multiplexer

//  from Example 4.6
module mux_2x1_df (m_out, A, B, select);
  output        m_out;
  input         A, B;
  input         select;

  assign m_out = (select)? A : B;
endmodule
```

Simulation log:

select = 1 A = 0 B = 1 OUT = 0 time = 0
select = 1 A = 1 B = 0 OUT = 1 time = 10
select = 0 A = 1 B = 0 OUT = 0 time = 20
select = 0 A = 0 B = 1 OUT = 1 time = 30

HDL Example 4-10

- **Gate-level description of a full adder**

HDL Example 4.10

```
// Gate-level description of circuit of Fig. 4.2
module Circuit_of_Fig_4_2 (A, B, C, F1, F2);
  input  A, B, C;
  output F1, F2;
  wire   T1, T2, T3, F2_b, E1, E2, E3;
  or   g1 (T1, A, B, C);
  and g2 (T2, A, B, C);
  and g3 (E1, A, B);
  and g4 (E2, A, C);
  and g5 (E3, B, C);
  or   g6 (F2, E1, E2, E3);
  not  g7 (F2_b, F2);
  and g8 (T3, T1, F2_b);
  or   g9 (F1, T2, T3);
endmodule
```

HDL Example 4-10 (Continued)

```
// Stimulus to analyze the circuit

module test_circuit;
  reg [2: 0] D;
  wire F1, F2;
  Circuit_of_Fig_4_2   M_F4_32 (D[2], D[1], D[0], F1, F2);
  initial
    begin
      D = 3'b000;
      repeat (7) #10 D = D + 1'b1;
    end
  initial
  $monitor ("ABC = %b F1 = %b F2 =%b ",  D, F1, F2);

endmodule
```

Simulation log: ABC = 000 F1 = 0 F2 =0
ABC = 001 F1 = 1 F2 =0 ABC = 010 F1 = 1 F2 =0
ABC = 011 F1 = 0 F2 =1 ABC = 100 F1 = 1 F2 =0
ABC = 101 F1 = 0 F2 =1 ABC = 110 F1 = 0 F2 =1
ABC = 111 F1 = 1 F2 =1