

2021-2022学年秋季学期

第四部分-B 组合逻辑电路高级内容

授课团队：宋威

助 教：薛子涵

○期中考试

11月4日下午15:20

405教室

组合逻辑电路高级内容 (属于考试内容)

- 硬件描述语言: Hardware Description Language (HDL)
- 使用Verilog HDL完成组合逻辑电路设计
- 测试与仿真
- 组合逻辑电路的时序分析
- 组合逻辑电路的竞争冒险现象

硬件描述语言 (HDL)

○ 硬件描述语言

现在设计大规模逻辑电路使用程序描述复杂电路。
硬件描述语言即描述数字电路的程序语言。

○ 常见的硬件描述语言

○ Verilog HDL (IEEE 1364)

Cadence (1990), 语法类似于C, 被广泛支持。

○ SystemVerilog (IEEE 1800)

Verilog的超集, 支持面向对象, 支持更多验证特性, 逐步被EDA工具完全支持。

○ VHDL (IEEE 1076)

和Verilog一起是现在最常用的两种HDL之一, 强制类型。

○ Chisel

UCB (2014), 基于scala的DSL。更高的抽象级别。

○ SystemC (IEEE 1666)

基于C++的HDL, 引入HLS (high level synthesis)。

硬件描述语言的抽象级别

- 抽象级别 (abstract level)
 - 描述一个对象时所涉及到的细节级别
- 门级 (网表, 自学)
 - 直接描述电路的门级实现
 - 具体到门的类型和输入信号数
 - 相当于用代码表达电路图
- 寄存器传输级 (Register Transfer Level)
 - 描述寄存器和寄存器间的组合逻辑
 - 精确控制到时钟周期
 - 当前大规模集成电路的设计抽象程度
- 行为级
 - 描述电路的抽象行为、传输协议
 - 让电路综合器 (编译器) 决定实现细节和时钟延迟

○逻辑运算

| 逻辑运算 | 逻辑表达式 | Verilog |
|------|------------------------|---------------------------|
| 与 | $Y = A \cdot B$ | <code>y=a&b</code> |
| 或 | $Y = A + B$ | <code>y=a b</code> |
| 非 | $Y = \bar{A}$ | <code>y=~a</code> |
| 与非 | $Y = \overline{AB}$ | <code>y=~(a&b)</code> |
| 或非 | $Y = \overline{A + B}$ | <code>y=~(a b)</code> |
| 异或 | $Y = A \oplus B$ | <code>y=a^b</code> |
| 同或 | $Y = A \odot B$ | <code>y=~(a^b)</code> |

逻辑运算的例子

○投票电路

$$Y = AB + AC + BC$$

$$y = (a\&b) \mid (a\&c) \mid (b\&c)$$

○双路选择器

$$Y = A\bar{S} + BS$$

$$y = (a\&\sim s) \mid (b\&s)$$

○双路分用器

$$A = I\bar{S}$$

$$B = IS$$

$$a = i\&\sim s$$

$$b = i\&s$$

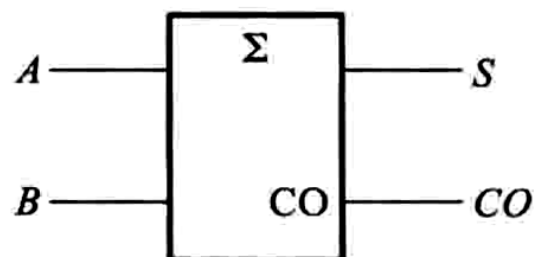
电路模块 (module)

○ 一个模块是一个具有特定输入输出和特定功能的子电路

○ 1位全加器

$$S = A \oplus B \oplus C_I$$

$$C_O = A \cdot B + B \cdot C_I + A \cdot C_I$$

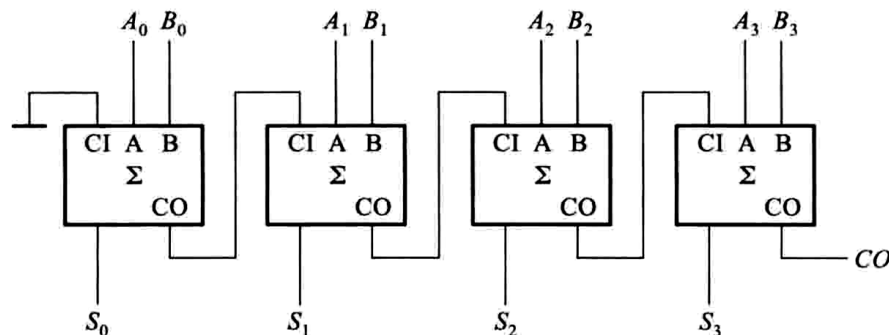


```
module adder(input a, b, ci, output s, co);  
    assign s = a ^ b ^ ci;  
    assign co = (a&b) | (b&ci) | (a&ci);  
endmodule
```


○ 在一个模块中使用另外一个模块（代码复用）

```
module adder(input a, b, ci, output s, co);  
    assign s = a ^ b ^ ci;  
    assign co = (a&b) | (b&ci) | (a&ci);  
endmodule
```

```
module add4(  
    input [3:0] a, b,  
    output [3:0] s,  
    output co );  
    wire [2:0] m; // internal wire  
    adder A0(.a(a[0]), .b(b[0]), .ci(0), .s(s[0]), .co(m[0]));  
    adder A1(.a(a[1]), .b(b[1]), .ci(m[0]), .s(s[1]), .co(m[1]));  
    adder A2(.a(a[2]), .b(b[2]), .ci(m[1]), .s(s[2]), .co(m[2]));  
    adder A3(.a(a[3]), .b(b[3]), .ci(m[2]), .s(s[3]), .co(co));  
endmodule
```



直接使用数学运算符

```
module add4(  
    input [3:0] a, b,  
    output [3:0] s,  
    output co );  
    wire [2:0] m; // internal wire  
    adder A0(.a(a[0]), .b(b[0]), .ci(0), .s(s[0]), .co(m[0]));  
    adder A1(.a(a[1]), .b(b[1]), .ci(m[0]), .s(s[1]), .co(m[1]));  
    adder A2(.a(a[2]), .b(b[2]), .ci(m[1]), .s(s[2]), .co(m[2]));  
    adder A3(.a(a[3]), .b(b[3]), .ci(m[2]), .s(s[3]), .co(co));  
endmodule
```

```
module adder4(  
    input [3:0] a, b,  
    output [3:0] s,  
    output co );  
    assign {co, s} = {1'b0, a} + {1'b0, b};  
endmodule
```

目前已经遇到的关键字

常量表示:

8'b0100_1100 8'h4c 8'd76

逻辑运算符:

& | ~ ^ [] {}

数学运算符:

+ - *

赋值:

=

关键字:

module, endmodule

input, output

wire

assign

更高的抽象级别：敏感代码块(always)

```
module adder(input a, b, ci, output s, co);  
    assign s = a ^ b ^ ci;  
    assign co = (a&b) | (b&ci) | (a&ci);  
endmodule
```

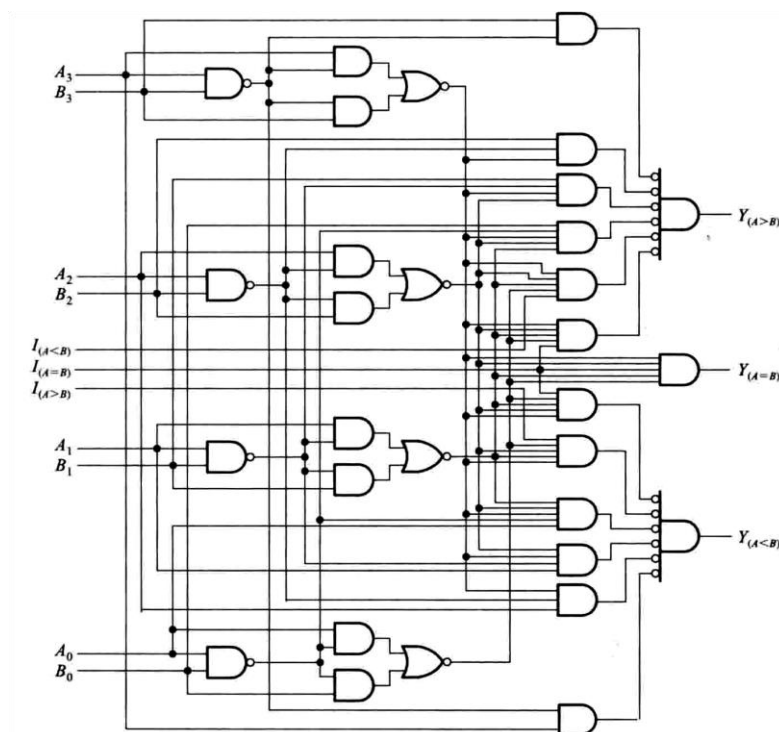
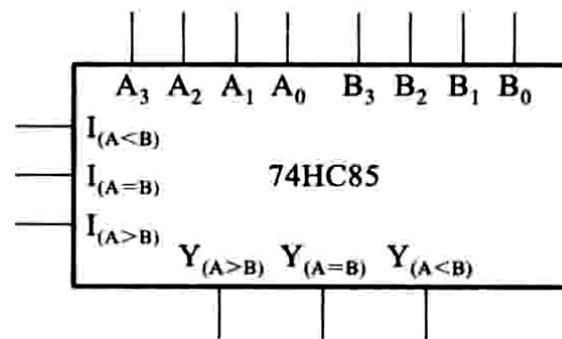
```
module adder_always(input a, b, ci, output s, co);  
    reg s, co;  
    always @(a, b, ci) begin  
        s = a ^ b ^ ci;  
        co = (a&b) | (b&ci) | (a&ci);  
    end  
endmodule
```

你也许觉得，这没什么嘛 😊，看下面的。

引入判断 if-else

4位数值比较器 (74HC85)

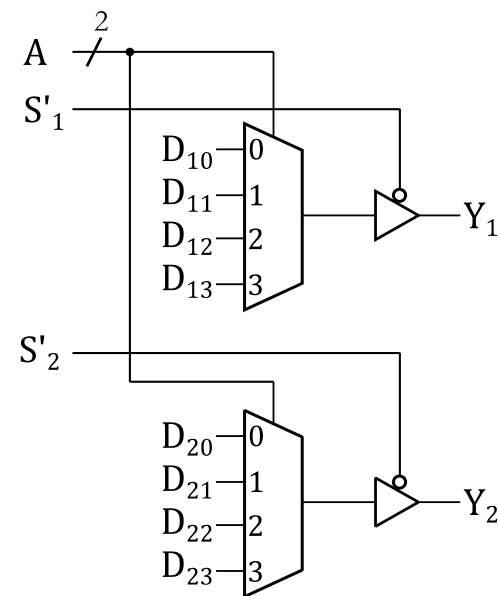
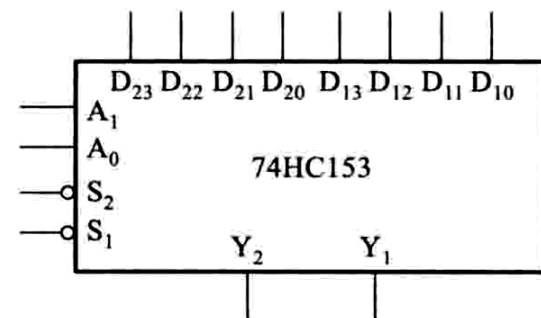
```
module ic74hc85(  
    input [3:0] a, b,  
    input [2:0] i,  
    output [2:0] y);  
  
    reg [2:0] m;  
  
    always @(a, b, i) begin  
        if(a > b)      m = 3'b100;  
        else if(a < b) m = 3'b001;  
        else          m = i;  
    end  
  
    assign y = m;  
endmodule
```



更复杂的 if-else

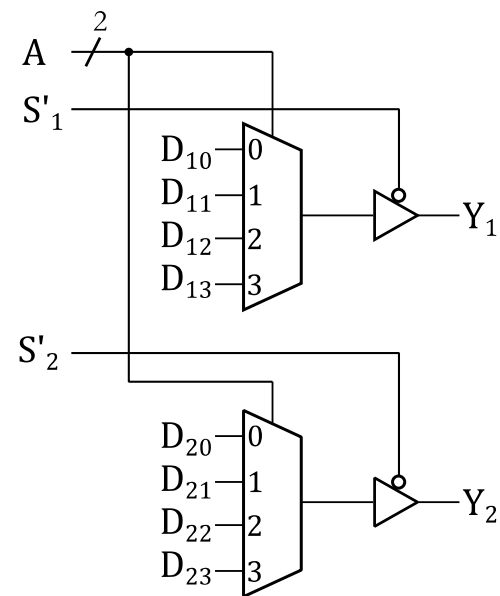
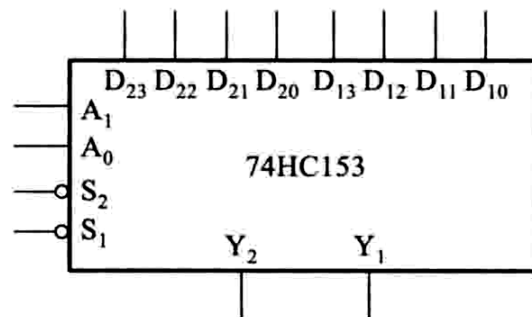
○双4选1数据选择器 (153)

```
module ic74hc153(  
    input [1:0] d1, d2, d3, d4  
    input [1:0] a, s,  
    output [1:0] y);  
  
    reg [1:0] m;  
  
    always @(d1, d2, d3, d4, a) begin  
        if(a == 2'd0)      m = d1;  
        else if(a == 2'd1) m = d2;  
        else if(a == 2'd2) m = d3;  
        else                m = d4;  
    end  
  
    assign y = (~s)&m;  
endmodule
```



○双4选1数据选择器 (153)

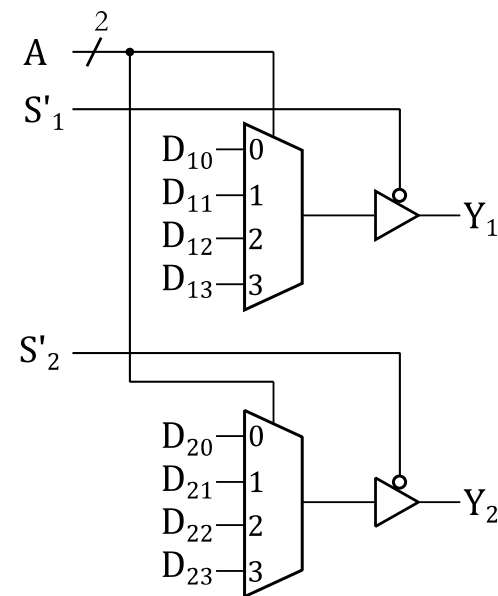
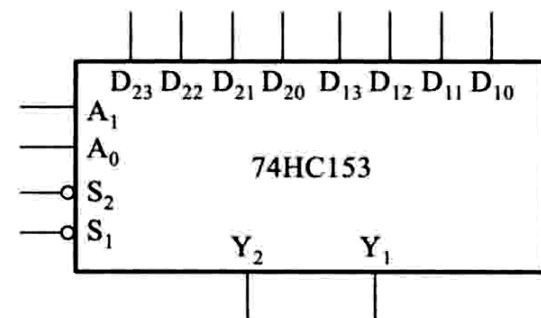
```
module ic74hc153_case(  
    input [1:0] d1, d2, d3, d4  
    input [1:0] a, s,  
    output [1:0] y);  
  
    reg [1:0] m;  
  
    always @(d1, d2, d3, d4, a)  
        case(a)  
            2'd0:    m = d1;  
            2'd1:    m = d2;  
            2'd2:    m = d3;  
            default: m = d4;  
        endcase  
  
    assign y = (~s)&m;  
endmodule
```



直接使用变量为下标

○双4选1数据选择器 (153)

```
module ic74hc153_index(  
    input [3:0] d1, d2,  
    input [1:0] a, s,  
    output [1:0] y);  
  
    wire [1:0] m;  
  
    assign m = {d2[a], d1[a]};  
    assign y = (~s)&m;  
endmodule
```



目前已经遇到的关键字

常量表示:

8'b0100_1100 8'h4c 8'd76

逻辑运算符:

& | ~ ^ [] {}

数学运算符:

+ - * < > == !=

赋值:

=

关键字:

module, endmodule

input, output

wire, reg

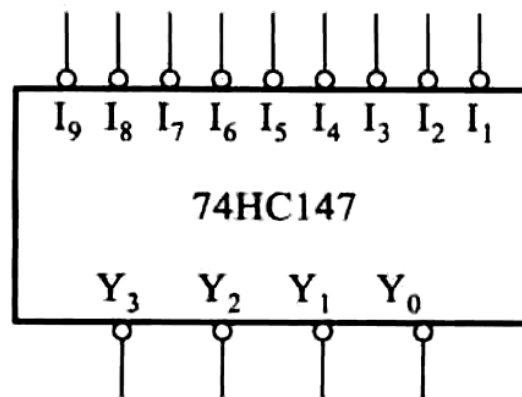
assign

always

if, else, begin, end, case, endcase

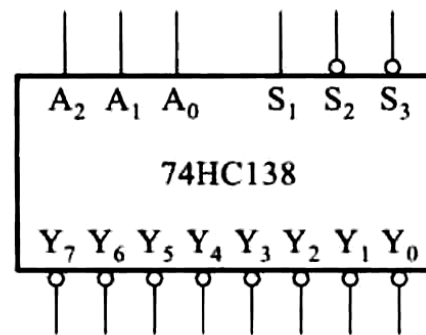
优先编码器74HC147

```
module ic74hc147(  
    input [9:0] i,  
    output [3:0] y);  
  
    reg [3:0] m;  
  
    always @(i)  
        casez(i)  
            10'b0?????????: m = ~(4'd9);  
            10'b10?????????: m = ~(4'd8);  
            10'b110?????????: m = ~(4'd7);  
            10'b1110?????????: m = ~(4'd6);  
            10'b11110?????????: m = ~(4'd5);  
            10'b111110?????: m = ~(4'd4);  
            10'b1111110?????: m = ~(4'd3);  
            10'b11111110???: m = ~(4'd2);  
            10'b111111110?: m = ~(4'd1);  
            default: m = ~(4'd0);  
        endcase  
  
    assign y = m;  
endmodule
```



二进制译码器74HC138

```
module ic74hc138(  
    input [2:0] a, s,  
    output [7:0] y);  
  
    reg [7:0] m;  
  
    always @(a, s) begin  
        m = 8'b1111_1111;  
        if(s == 3'b001)  
            m[a] = 1'b0;  
    end  
  
    assign y = m;  
endmodule
```



目前已经遇到的关键字

常量表示:

8'b0100_1100 8'h4c 8'd76

逻辑运算符:

& | ~ ^ [] {}

数学运算符:

+ - * < > == !=

赋值:

=

关键字:

module, endmodule

input, output

wire, reg

assign

always

if, else, begin, end, case, endcase, **casez**

○设计思想

- 大规模的数字集成电路的设计需要方便的设计途径。
- HDL能够被综合（编译）成目标电路并仿真（执行）。
- 更高的抽象层次有利于设计更复杂的系统

○设计方法

- 模块化，确定模块的输入输出
- 用运算符来表达逻辑表达式
- 直接使用算术运算符
- 条件表达式：if、case
- 向量运算：连接、下标变量

○ 测试程序 (test bench)

- 利用Verilog编写测试来验证被测模块 (DUT) 的功能
- DUT: Design under Test
- 测试程序并不需要是真实的硬件
- 可以直接使用不可综合的代码

○ 测试程序的组成

- 测试的顶层模块
- 被测模块
- 测试激励的产生
- 测试结果的验证

测试加法器

```
`timescale 1ns/1ps  
  
module test;  
  reg [3:0] a, b;  
  wire [3:0] s0, s1;  
  wire      co0, co1;  
  
  add4  ADD0(a, b, 0, s0, co0);  
  adder4 ADD1(a, b, 0, s1, co1);  
  
  initial begin  
    a = 0;  
    b = 0;  
    #10 a = 1;  
    #10 b = 2;  
    #10 a = 3;  
    . . . . .  
    #10;  
    $finish();  
  end  
  
  initial begin  
    $dumpfile("test.vcd");  
    $dumpvars(0);  
  end  
endmodule
```

设定 时间单位/时间精度

测试顶层

测试激励

实例化被测模块

设置测试激励

输出波形文件

测试加法器—输出波形

```
`timescale 1ns/1ps

module test;
  reg [3:0] a, b;
  wire [3:0] s0, s1;
  wire      co0, co1;

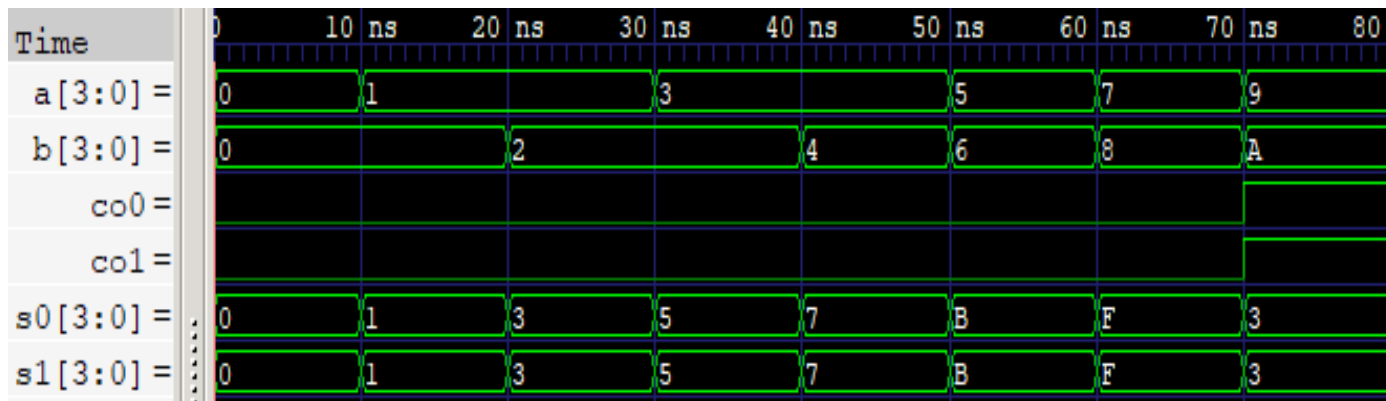
  add4 ADD0(a, b, s0, co0);
  adder4 ADD1(a, b, s1, co1);
```

```
initial begin
```

```
  a = 0;
  b = 0;
  #10 a = 1;
  #10 b = 2;
  #10 a = 3;
  . . . . .
```

```
  #10;
  $finish();
```

```
end
```



```
initial begin
```

```
  $dumpfile("test.vcd");
  $dumpvars(0);
```

```
end
```

```
endmodule
```

两种设计（串行s0和并行s1）的结果一样

○ 测试模块

- 没有输入输出参数
- 测试激励为内部变量
- 需要被直接赋值的信号为reg
- 可以使用initial块为激励赋值

○ initial块

- 只执行一次的代码块
- 使用#time来延时
- \$finish()系统函数终止仿真
- \$dumpfile()设置波形输出文件
- \$dumpvars()设置需要输出波形的信号

```
`timescale 1ns/1ps

module test;
    reg [3:0] a, b;
    wire [3:0] s0, s1;
    wire      co0, co1;

    add4     ADD0(a, b, s0, co0);
    adder4   ADD1(a, b, s1, co1);

    initial begin
        a = 0;
        b = 0;
        #10 a = 1;
        #10 b = 2;
        #10 a = 3;
        . . . . .
        #10;
        $finish();
    end

    initial begin
        $dumpfile("test.vcd");
        $dumpvars(0);
    end
endmodule
```

测试比较器74HC85

```
`timescale 1ns/1ps

module test;
  reg [7:0] a, b;
  wire [2:0] y, z;
  wire [2:0] i;

  ic74hc85 dut(.a(a[7:4]), .b(b[7:4]), .i(i), .y(y));

  assign i = {a[3:0]>b[3:0], a[3:0]==b[3:0], a[3:0]<b[3:0]};
  assign z = {a>b, a==b, a<b};

  initial begin
    a = 0;
    b = 0;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 $finish();
  end

  initial begin
    $dumpfile("test.vcd");
    $dumpvars(0);
  end
endmodule
```

使用计算生成部分激励和估计结果 (z)

使用\$random函数生成随机激励

测试比较器74HC85—输出波形

```
`timescale 1ns/1ps

module test;
  reg [7:0] a, b;
  wire [2:0] y, z;
  wire [2:0] i;

  ic74hc85 dut(.a(a[7:4]),
               .b(b[7:4]),
               .i(i), .y(y));

  assign i = {a[3:0]>b[3:0],
             a[3:0]==b[3:0],
             a[3:0]<b[3:0]};
  assign z = {a>b, a==b, a<b};

  initial begin
    a = 0;
    b = 0;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 a = $random; b = $random;
    #10 $finish();
  end

  initial begin
    $dumpfile("test.vcd");
    $dumpvars(0);
  end
endmodule
```

| Time | 0 | 10 ns | 20 ns | 30 ns | 40 ns | 50 ns | 60 ns | 70 ns | 80 ns |
|--------|-----|-------|-------|-------|-------|-------|-------|-------|-------|
| a[7:0] | 00 | 24 | 09 | 0D | 65 | 01 | 76 | ED | |
| b[7:0] | 00 | 81 | 63 | 8D | 12 | 0D | 3D | 8C | |
| i[2:0] | 010 | 100 | | 010 | 100 | 001 | | 100 | |
| y[2:0] | 010 | 001 | | | 100 | 001 | 100 | | |
| z[2:0] | 010 | 001 | | | 100 | 001 | 100 | | |

被测模块的输出y和测试模块的计算结果z一致。

测试选择器74HC153

```
module test;
  reg [3:0] d1, d2;
  reg [1:0] a, s;
  wire [1:0] y1, y2, y3;

  ic74hc153      dut1(d1, d2, a, s, y1);
  ic74hc153_case dut2(d1, d2, a, s, y2);
  ic74hc153_index dut3(d1, d2, a, s, y3);
```

用同样的激励驱动3种等效设计

```
initial begin
  d1 = 0; d2 = 0; a = 0; s = 0;
  #10 {d1,d2,a,s} = $random;
  #10 {d1,d2,a,s} = $random;
  #10 {d1,d2,a,s} = $random;
  #10 {d1,d2,a,s} = $random;
  . . .
  #10 {d1,d2,a,s} = $random;
  #10 {d1,d2,a,s} = $random;
  #10 $finish();
end
```

使用\$random函数生成随机激励

```
initial begin
  $dumpfile("test.vcd");
  $dumpvars(0);
end
endmodule
```

测试编码器和译码器

```
module test;
  reg [2:0] data_org;
  wire [7:0] data_one_hot;
  wire [3:0] data_gen;

  ic74hc138 decoder(data_org, 3'b001, data_one_hot);
  ic74hc147 encoder({2'b11,data_one_hot}, data_gen);
```

将随机数data_org编码成独热码，然后再用编码器复原随机数data_gen。

```
  initial begin
    data_org = 0;
    #10 data_org = $random;
    #10 data_org = $random;
    . . .
    #10 data_org = $random;
    #10 data_org = $random;
    #10 $finish();
  end
```

```
  wire OK;
  assign OK = {1'b0,data_org} == ~data_gen;
```

在测试模块中直接检验data_org和data_gen是一样的。

```
  initial begin
    $dumpfile("test.vcd");
    $dumpvars(0);
  end
endmodule
```

○测试

- 利用Verilog HDL来测试一个Verilog HDL设计的正确性
- 利用仿真来检验被测模块（黑盒测试）
- 需要使用各种激励的组合来扩大测试覆盖率

○测试方法

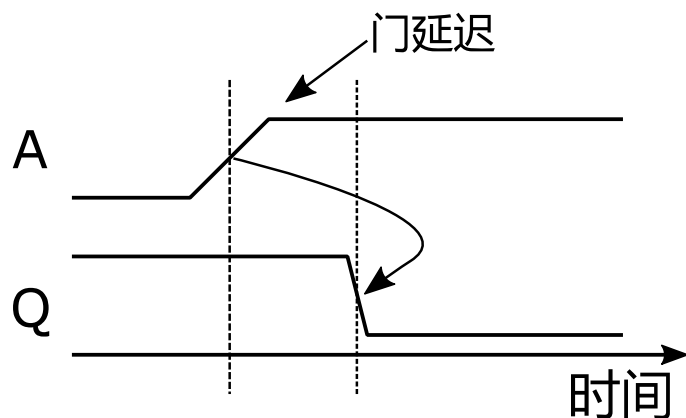
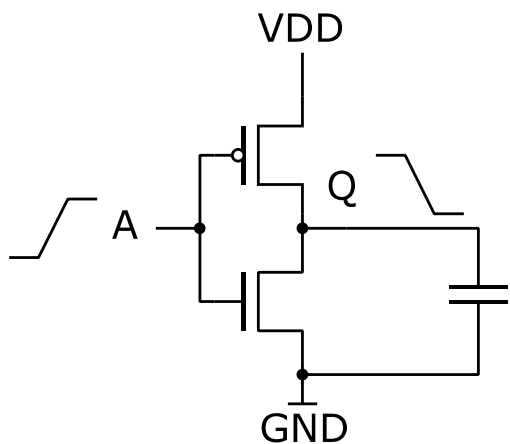
- 尽量使用高级语言特性
- 尽量用少量的代码扩大覆盖（累死机器别累死人）
 - 随机测试
 - 自动结果检验

组合逻辑电路的时序分析

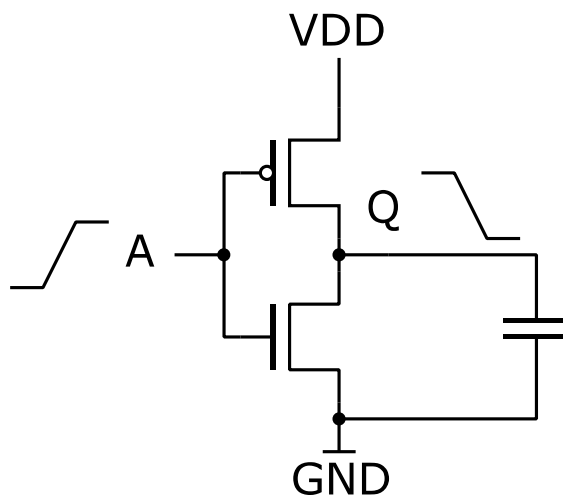
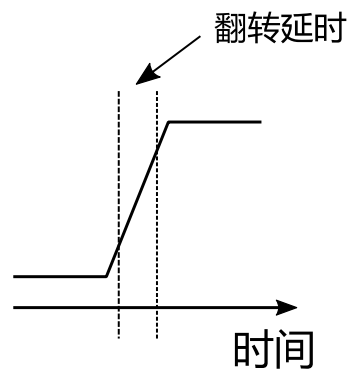
○时序分析

- 抽象逻辑电路的时间模型
- 利用时间模型推导电路的延迟
- 分析一个复杂电路的时间特性
- 计算电路的最大延迟路径（关键路径）
- 利用分析结果优化电路

○门电路的时间分析模型



反门的时序分析



Input transition time
输入翻转延时 t_{ti}

Output transition time
输出翻转延时 t_{to}

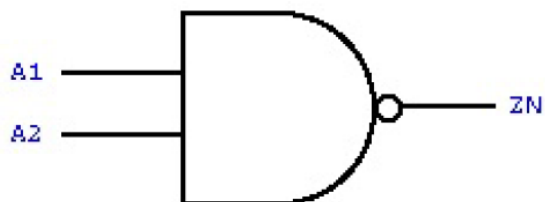
传输延时 t_p

Input capacitance
输入电容 C_i

Output load
输出负载 C_o

与非门的时序分析

| | |
|-----------|---|
| Strength | 1 |
| Cell Area | 0.798 μm^2 |
| Equation | $ZN = \neg(A1 \& A2)$ |
| Type | Combinational |
| Input | A1, A2 |
| Output | ZN |
| PG Pins | VDD (primary_power), VSS (primary_ground) |



| State Table | | |
|-------------|----|----|
| A1 | A2 | ZN |
| L | - | H |
| H | H | L |
| - | L | H |

| Propagation Delay [ns] | | | | | |
|------------------------|------|--------|---------|--------|---------|
| Input Transition [ns] | | 0.0012 | | 0.1985 | |
| Load Capacitance [fF] | | 0.3656 | 59.3567 | 0.3656 | 59.3567 |
| A1 to ZN | fall | 0.01 | 0.13 | 0.01 | 0.21 |
| | rise | 0.01 | 0.15 | 0.04 | 0.25 |
| A2 to ZN | fall | 0.01 | 0.13 | 0.01 | 0.19 |
| | rise | 0.01 | 0.15 | 0.05 | 0.26 |

| Output Transition [ns] | | | | | |
|------------------------|------|--------|---------|--------|---------|
| Input Transition [ns] | | 0.0012 | | 0.1985 | |
| Load Capacitance [fF] | | 0.3656 | 59.3567 | 0.3656 | 59.3567 |
| A1 to ZN | fall | 0.00 | 0.11 | 0.03 | 0.13 |
| | rise | 0.00 | 0.14 | 0.03 | 0.15 |
| A2 to ZN | fall | 0.00 | 0.11 | 0.03 | 0.12 |
| | rise | 0.01 | 0.14 | 0.03 | 0.15 |

| Capacitance [fF] | |
|------------------|--------|
| A1 | 1.5990 |
| A2 | 1.6642 |

| Leakage [nW] |
|--------------|
| 17.39 |

45nm Nangate Open Cell Library

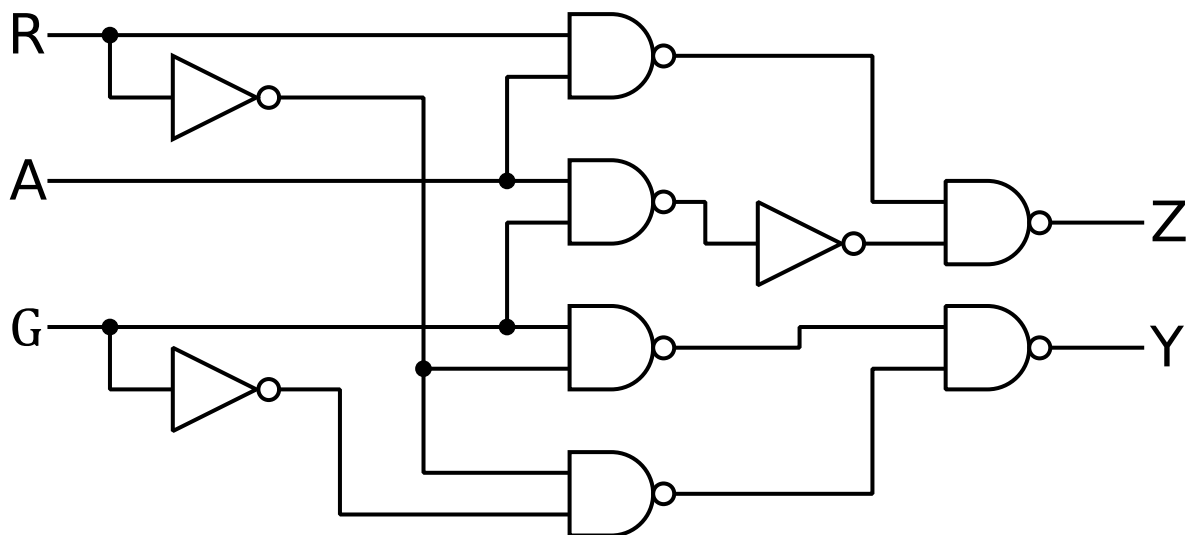
与非门的传输延时和输出翻转时间表

| Propagation Delay [ns] | | | | | |
|------------------------|------|--------|---------|--------|---------|
| Input Transition [ns] | | 0.0012 | | 0.1985 | |
| Load Capacitance [fF] | | 0.3656 | 59.3567 | 0.3656 | 59.3567 |
| A1 to ZN | fall | 0.01 | 0.13 | 0.01 | 0.21 |
| | rise | 0.01 | 0.15 | 0.04 | 0.25 |
| A2 to ZN | fall | 0.01 | 0.13 | 0.01 | 0.19 |
| | rise | 0.01 | 0.15 | 0.05 | 0.26 |

- 不同输入翻转时间
- 不同输出负载
- 不同翻转方向
- 不同信号路径

| Output Transition [ns] | | | | | |
|------------------------|------|--------|---------|--------|---------|
| Input Transition [ns] | | 0.0012 | | 0.1985 | |
| Load Capacitance [fF] | | 0.3656 | 59.3567 | 0.3656 | 59.3567 |
| A1 to ZN | fall | 0.00 | 0.11 | 0.03 | 0.13 |
| | rise | 0.00 | 0.14 | 0.03 | 0.15 |
| A2 to ZN | fall | 0.00 | 0.11 | 0.03 | 0.12 |
| | rise | 0.01 | 0.14 | 0.03 | 0.15 |

简化的时序分析

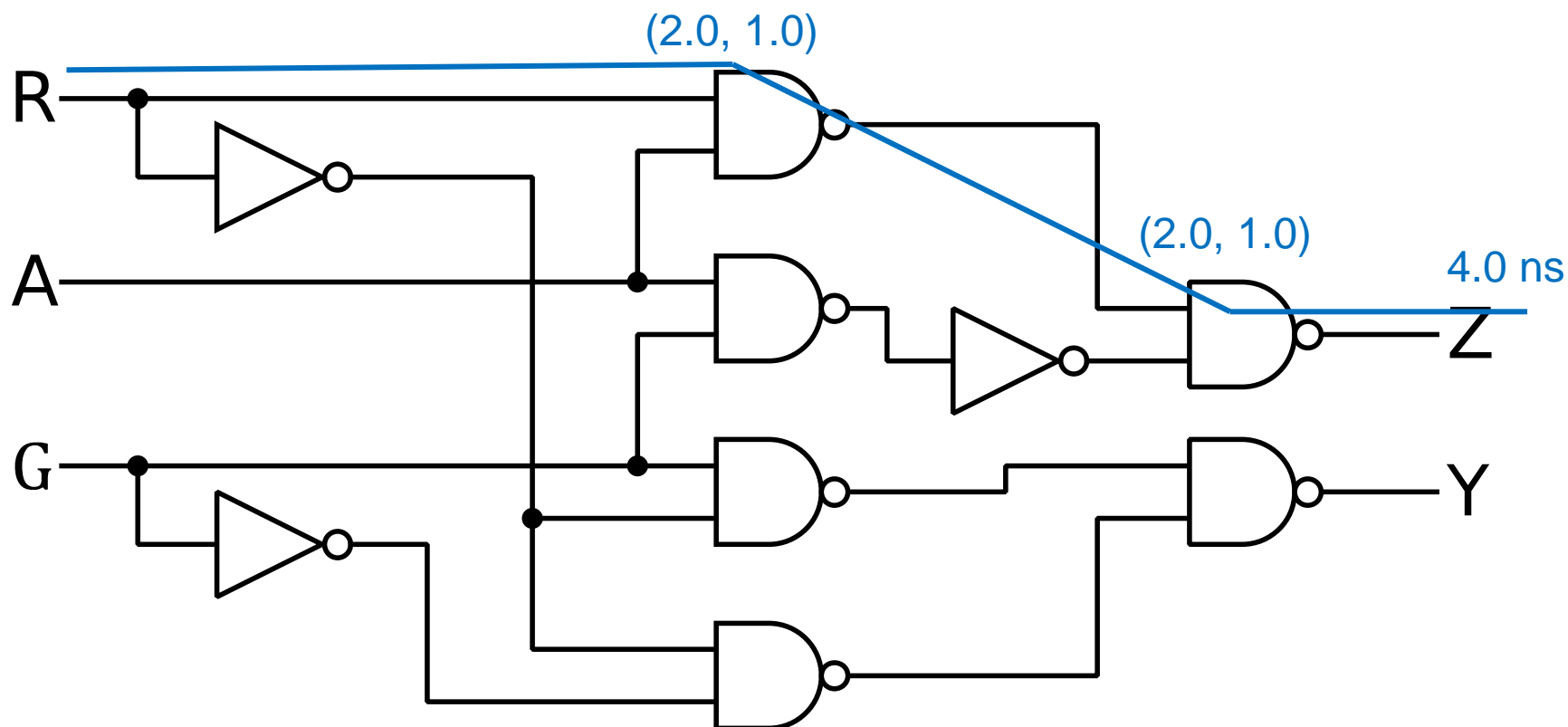


- 不区分翻转方向
- 不区分门的不同输入信号
- 所有门的输入电容: $C_i = 1pF$
- 反门: $t_p = 1t_{ti}C_o, t_{to} = 0.4t_{ti}C_o$
- 与非门: $t_p = 2t_{ti}C_o, t_{to} = 1t_{ti}C_o$
- 输入信号的翻转延时: $t_{ti} = 1ns$
- 输出信号的负载: $C_o = 1pF$

该电路的最快路径和最短路径?

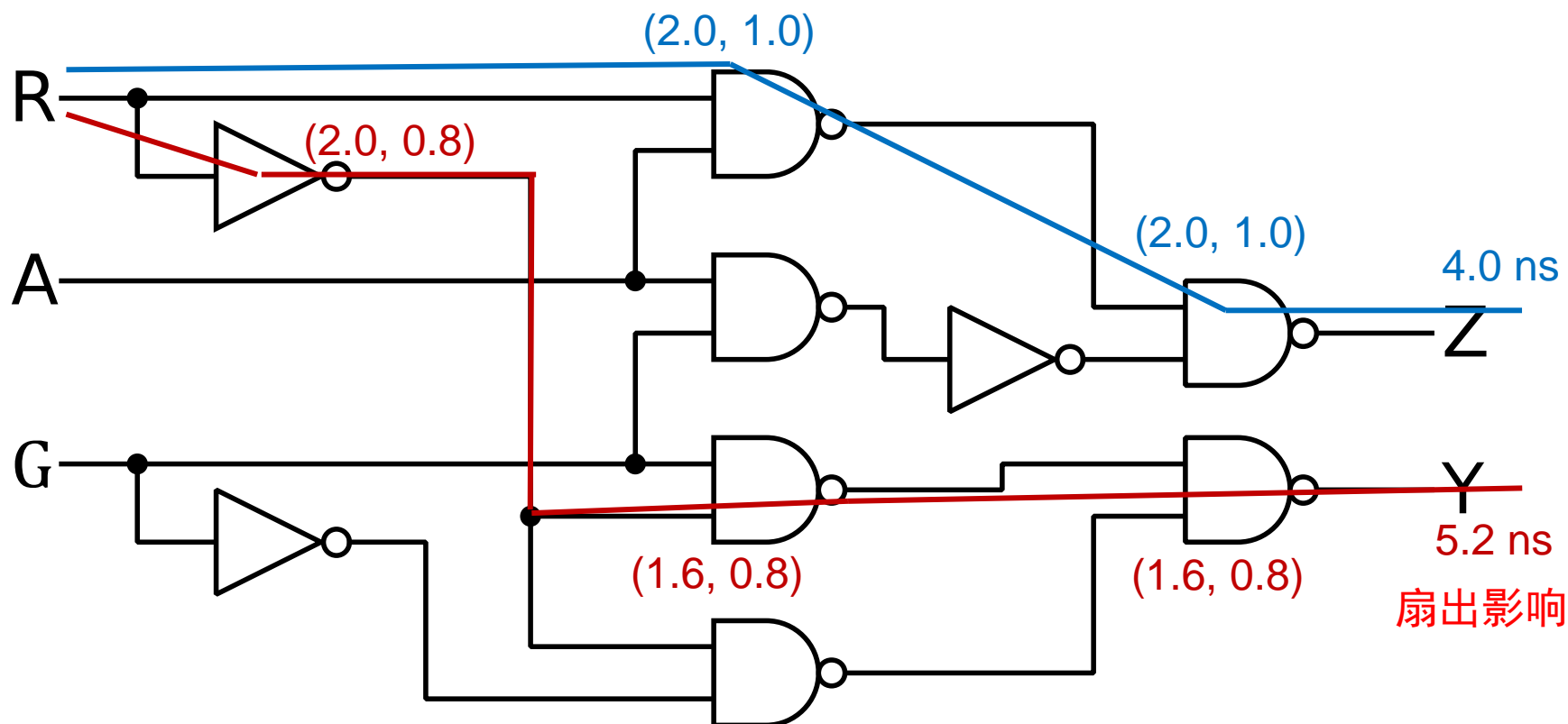
简化的时序分析

- 所有门的输入电容: $C_i = 1pF$
- 反门: $t_p = 1t_{ti}C_o, t_{to} = 0.4t_{ti}C_o$
- 与非门: $t_p = 2t_{ti}C_o, t_{to} = 1t_{ti}C_o$
- 输入信号的翻转延时: $t_{ti} = 1ns$
- 输出信号的负载: $C_o = 1pF$



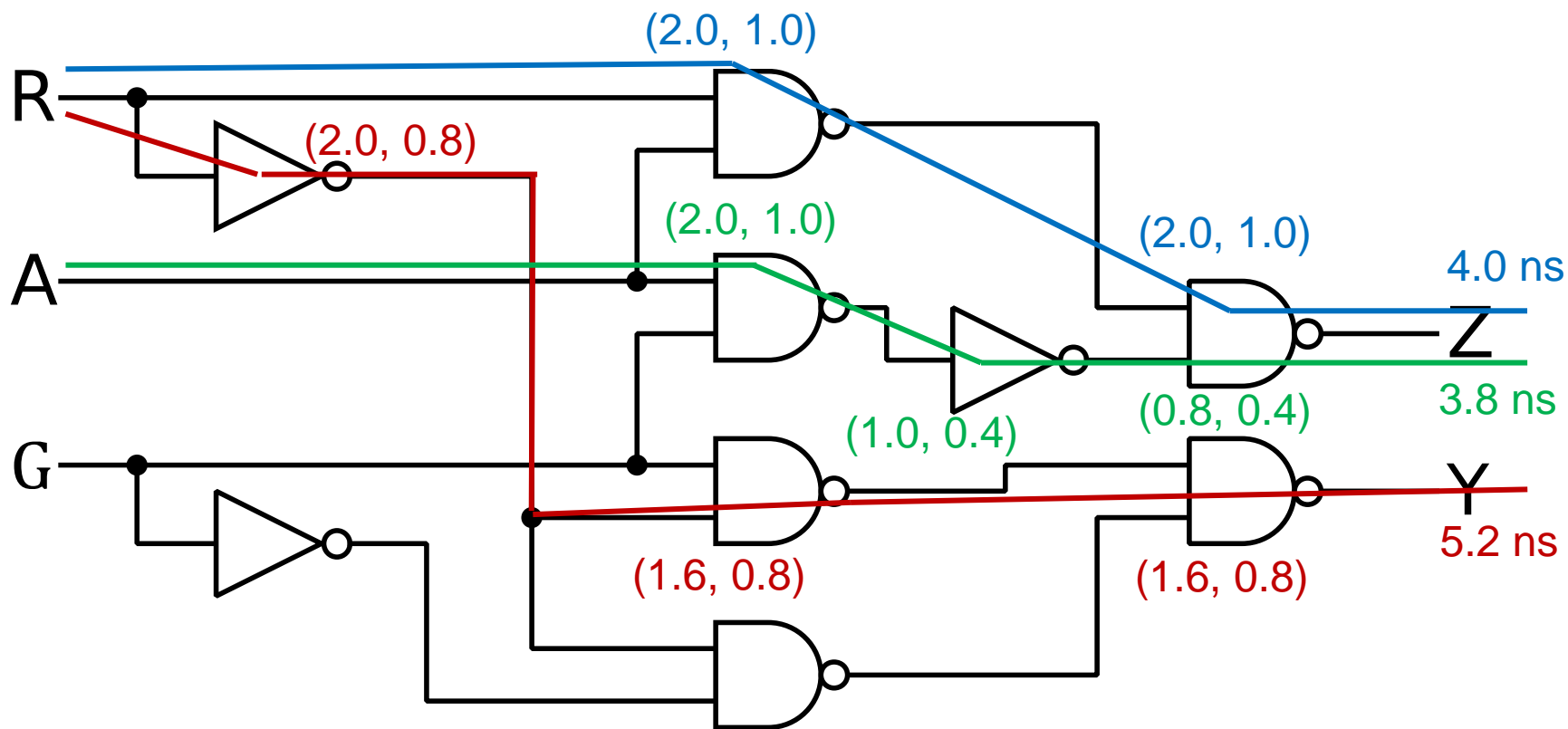
简化的时序分析

- 所有门的输入电容: $C_i = 1pF$
- 反门: $t_p = 1t_{ti}C_o, t_{to} = 0.4t_{ti}C_o$
- 与非门: $t_p = 2t_{ti}C_o, t_{to} = 1t_{ti}C_o$
- 输入信号的翻转延时: $t_{ti} = 1ns$
- 输出信号的负载: $C_o = 1pF$



简化的时序分析

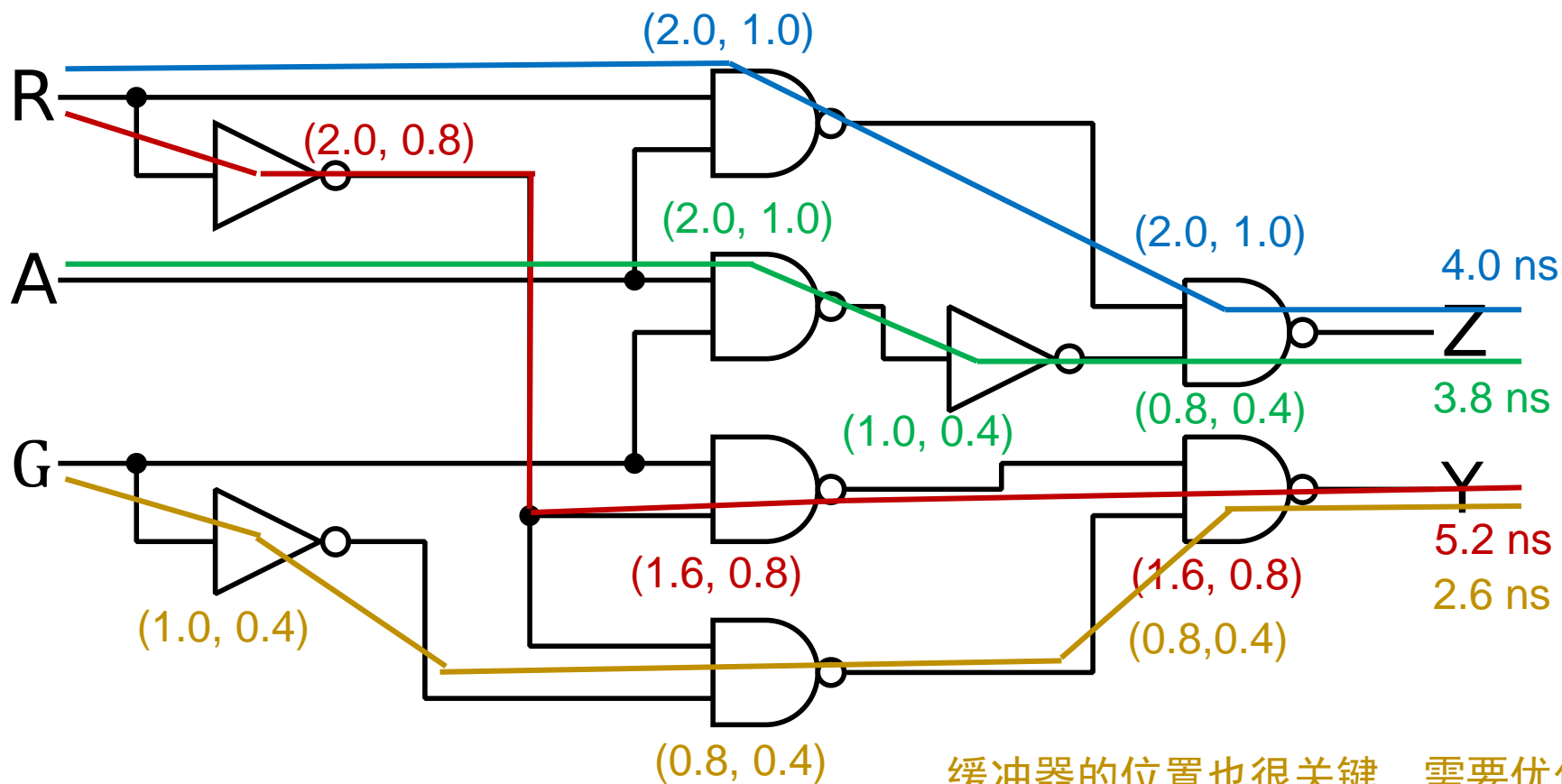
- 所有门的输入电容: $C_i = 1pF$
- 反门: $t_p = 1t_{ti}C_o, t_{to} = 0.4t_{ti}C_o$
- 与非门: $t_p = 2t_{ti}C_o, t_{to} = 1t_{ti}C_o$
- 输入信号的翻转延时: $t_{ti} = 1ns$
- 输出信号的负载: $C_o = 1pF$



反门作为缓冲器降低了延迟

简化的时序分析

- 所有门的输入电容: $C_i = 1pF$
- 反门: $t_p = 1t_{ti}C_o, t_{to} = 0.4t_{ti}C_o$
- 与非门: $t_p = 2t_{ti}C_o, t_{to} = 1t_{ti}C_o$
- 输入信号的翻转延时: $t_{ti} = 1ns$
- 输出信号的负载: $C_o = 1pF$



逻辑组合电路的时序分析总结

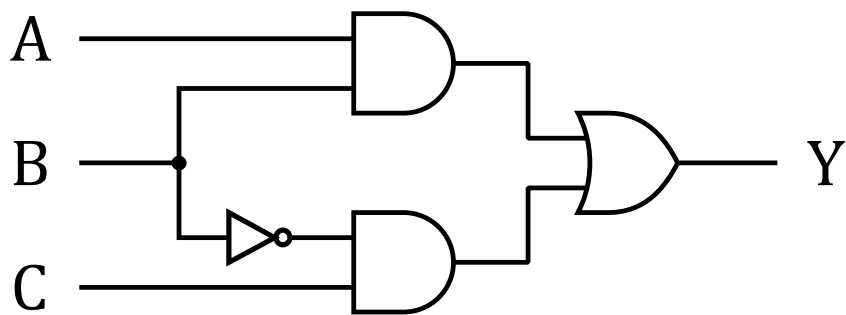
○ 时序分析的原理

- 将CMOS电路简化成简单的电容充电模型
- 再将传输延时简化为线性模型
- 根据输入翻转延时和输出负载计算传输延时和输出翻转延时
- 输出负载为输出端所接所有门的输入电容之和
- 一般用于电路综合时的粗粒度延时计算

○ 一般性常识

- 门越复杂，延时越长，驱动力越弱
- 简单门可用作缓冲器减小延时
- 扇出数量对延时有影响
- 缓冲器位置对延时有影响

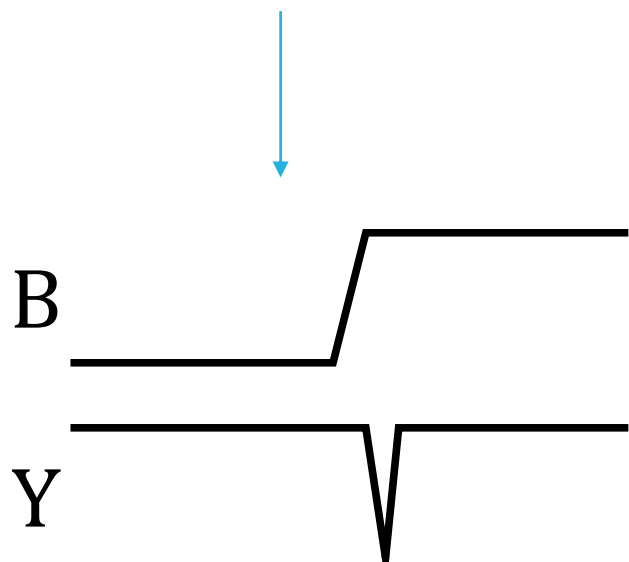
组合逻辑电路的竞争冒险现象



$$Y = AB + \bar{B}C$$

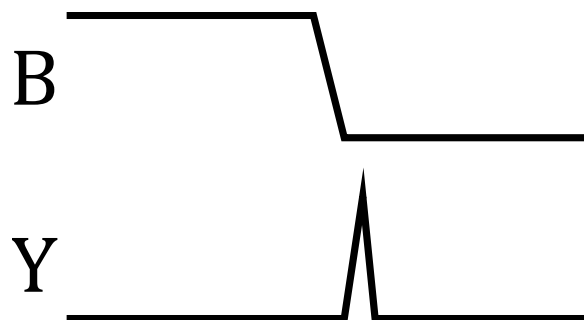
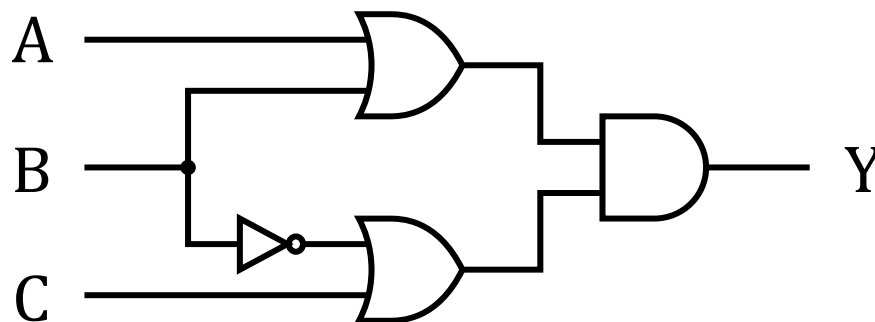
从B到Y的两条路径延时不一致，导致了Y出现了毛刺。

这种毛刺即组合逻辑电路的竞争冒险现象。



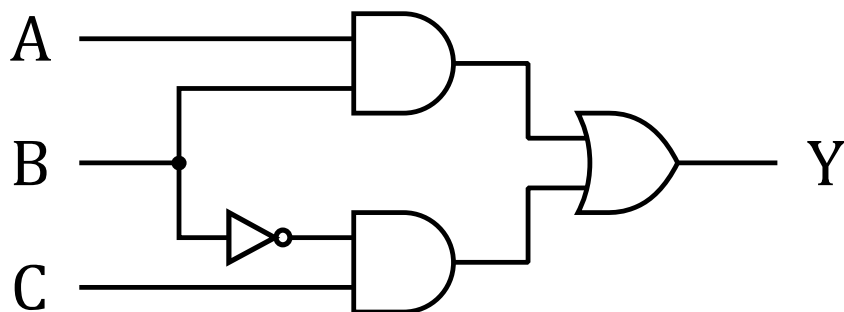
组合逻辑电路的竞争冒险现象

$$Y = (A + B)(\bar{B} + C)$$



或与电路和与或电路一样，也会出现竞争冒险现象。

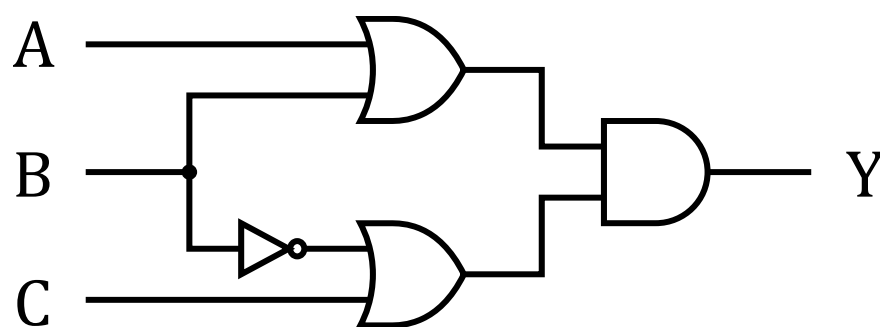
组合逻辑电路的竞争冒险现象检测



$$Y = AB + \bar{B}C$$

如果 $A = 1, C = 1$

$$Y = B + \bar{B}$$



$$Y = (A + B)(\bar{B} + C)$$

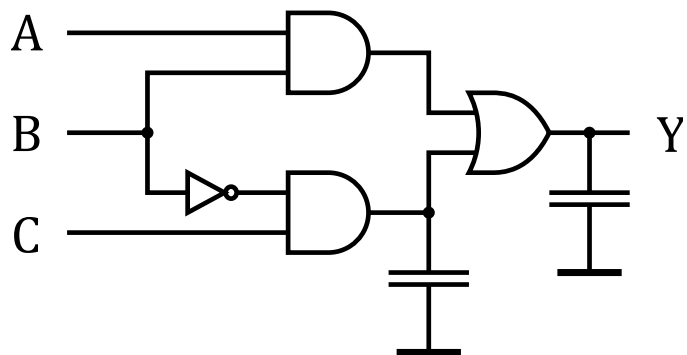
如果 $A = 0, C = 0$

$$Y = B \cdot \bar{B}$$

如果在特定情况下，电路表达式可以被化简成 $Y = B + \bar{B}$ 或者 $Y = B \cdot \bar{B}$ ，则可能出现竞争冒险现象。

竞争冒险现象的解决方法一

○接入滤波电容



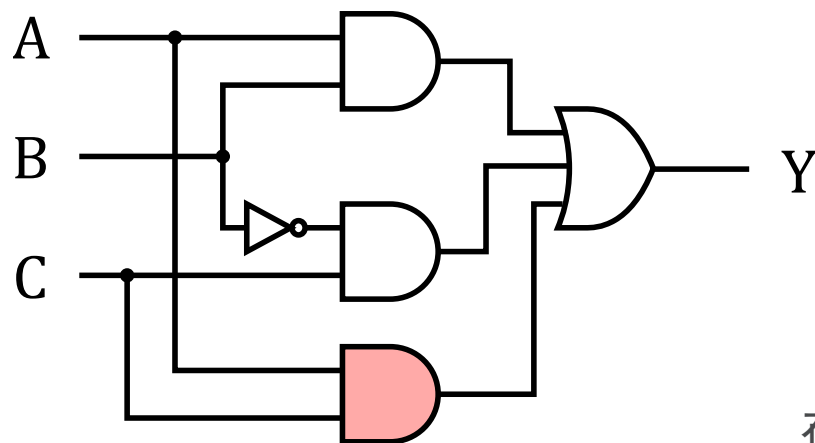
- 芯片内部的数字电路不支持直接添加电容
- 电容导致电路速度下降
- 板级电路常用做法

○选通脉冲

- 等待信号稳定后再采样输出结果
- 利用时钟电路作为采样信号，同步时序电路的常用做法
- 对于特殊的纯组合逻辑电路（无时钟电路不适用）

竞争冒险现象的解决方法二

○添加冗余门



| AB | C | 0 | 1 |
|----|---|---|---|
| 00 | | 0 | 1 |
| 01 | | 0 | 0 |
| 11 | | 1 | 1 |
| 10 | | 0 | 1 |

在卡诺图中，如果出现相邻但是不交叠的两个覆盖区域时，便可能出现竞争冒险现象。用冗余覆盖区域覆盖相邻区域，则可以避免。

总结：考试范围

- Verilog HDL的组合逻辑设计
 - 理解Verilog HDL的基本语法
 - 能使用Verilog HDL设计组合逻辑电路
 - 能理解基本的测试模块
- 组合逻辑电路时序分析
 - 理解简化的门延时模型
 - 能够利用简化模型，推导简单电路的关键路径（最长延时路径）
- 竞争冒险
 - 能够分析一个简单电路是否存在竞争冒险现象
 - 能够利用卡诺图来避免简单的竞争冒险电路

任何问题?

作业 (2页!)

第四章习题:

32

编程题 1:

用Verilog HDL编写一个组合逻辑电路模块完成以下功能:

该模块有8个单比特输入信号 $input [7:0] d$

输出信号 $output [2:0] s$ 给出输入信号中高电平 (1) 的个数。

提交设计电路和测试模块。

编程题 2:

用Verilog HDL编写一个组合逻辑电路模块完成以下功能:

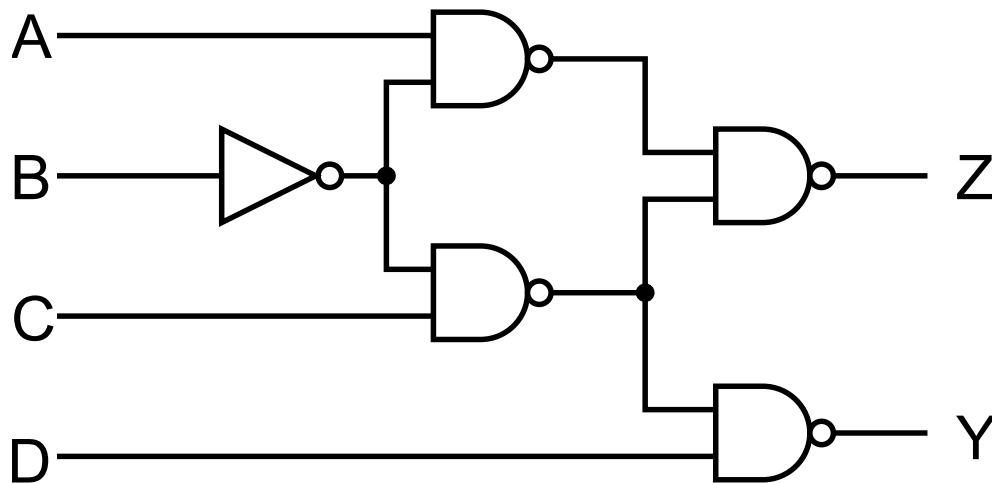
该模块一个4位输入信号 $input [3:0] din$

输出信号 $output [2:0] dout$ 给出 $[din/3]$ (除3的上取整)。

提交设计电路和测试模块。

作业 (2页!)

分析所有路径的延时:



- 不区分翻转方向
- 不区分门的不同输入信号

- 所有门的输入电容: $C_i = 1pF$
- 反门: $t_p = 1t_{ti}C_o, t_{to} = 0.4t_{ti}C_o$
- 与非门: $t_p = 2t_{ti}C_o, t_{to} = 1t_{ti}C_o$
- 输入信号的翻转延时: $t_{ti} = 1ns$
- 输出信号的负载: $C_o = 1pF$