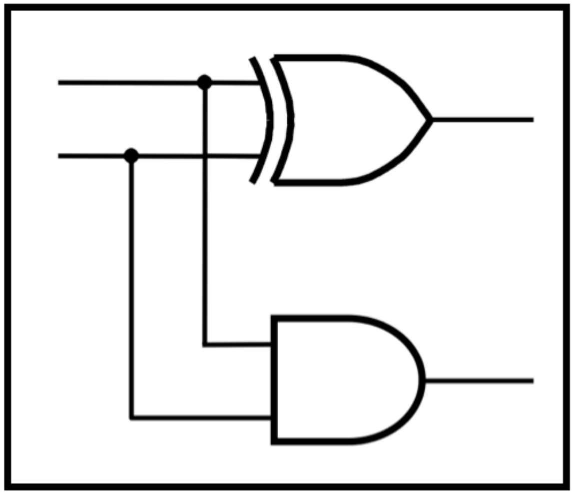


CprE 281: Digital Logic

Instructor: Alexander Stoytchev

http://www.ece.iastate.edu/~alexs/classes/

Intro to Verilog

CprE 281: Digital Logic
Iowa State University, Ames, IA
Copyright © Alexander Stoytchev

Administrative Stuff

- **HW3 is due on Monday Sep 12 @ 4p**

Administrative Stuff

- HW4 is out

- It is due on Monday Sep 19 @ 4pm.

- Please write clearly on the first page (in BLOCK CAPITAL letters) the following three things:

    - Your First and Last Name
    - Your Student ID Number
    - Your Lab Section Letter

- Also, please
    - Staple your pages

Administrative Stuff

TA Office Hours:

- **11:00am-1:00pm on Wednesdays (Jinyuan Jia)**
  Location: TLA (Coover Hall - first floor)

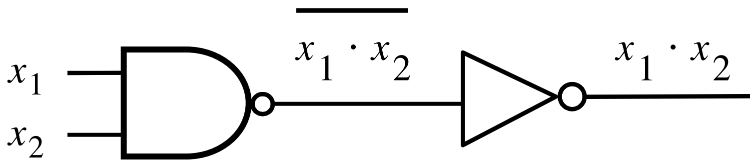
- **9:50am-11:50am on Thursday (Siyuan Lu)**
  Location: TLA (Coover Hall - first floor)

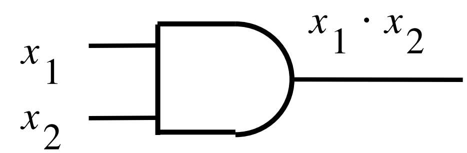
Administrative Stuff

- **Midterm Exam #1**
- **When: Friday Sep 23.**
- **Where: This classroom**
- **What: Chapter 1 and Chapter 2 plus number systems**
- **The exam will be open book and open notes (you can bring up to 3 pages of handwritten notes).**
- **More details to follow.**

Quick Review

NAND followed by NOT = AND





x_1	x_2	f
0	0	1
0	1	1
1	0	1
1	1	0

f
0
0
0
1

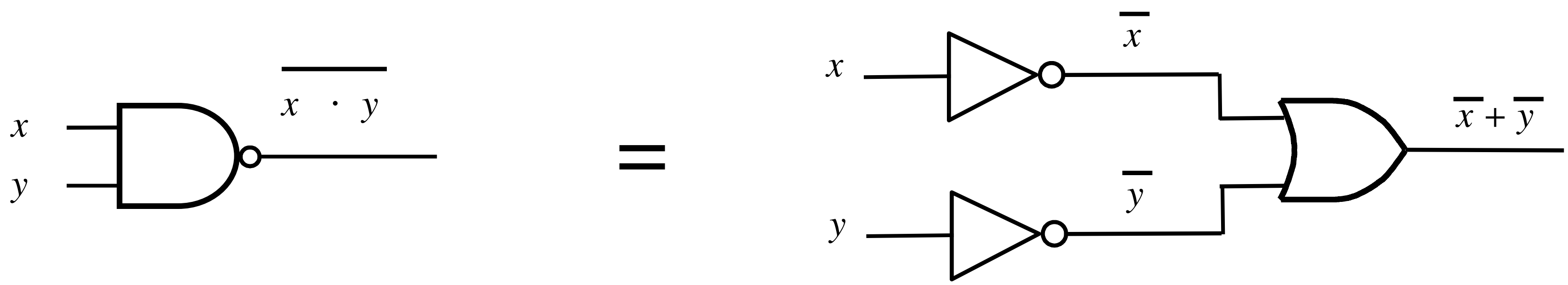
x_1	x_2	f
0	0	0
0	1	0
1	0	0
1	1	1

DeMorgan's Theorem

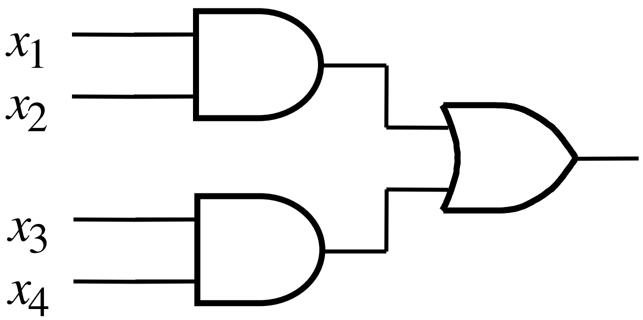
15a. $\overline{x \cdot y} = \overline{x} + \overline{y}$

DeMorgan's Theorem

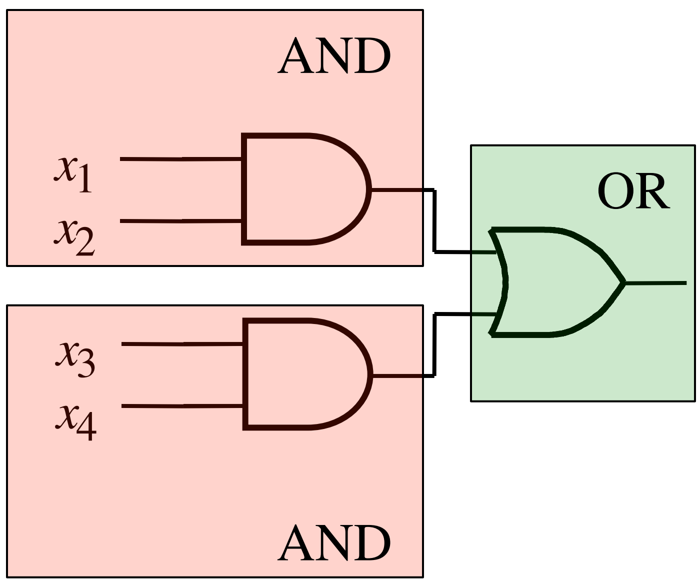
15a. $\overline{x \cdot y} = \overline{x} + \overline{y}$



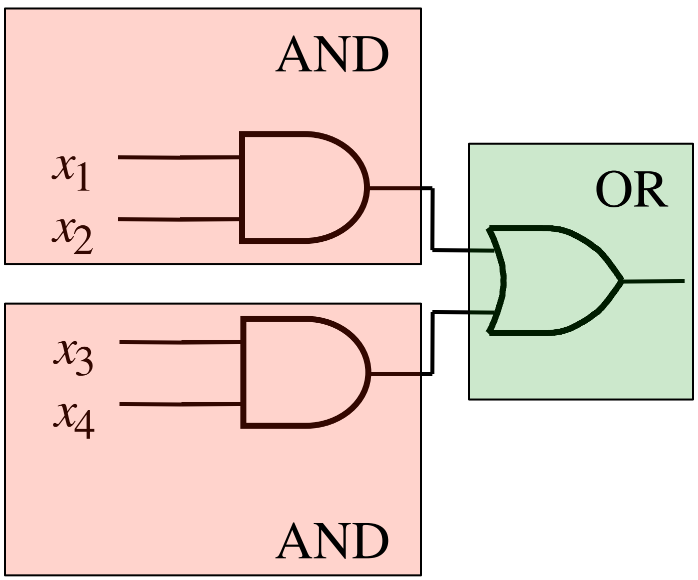
Sum-Of-Products

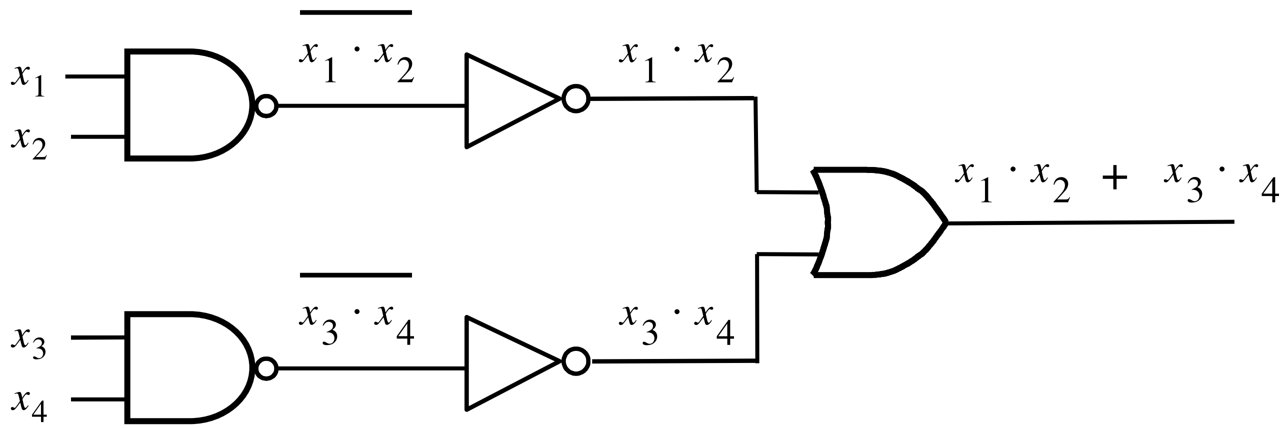


Sum-Of-Products

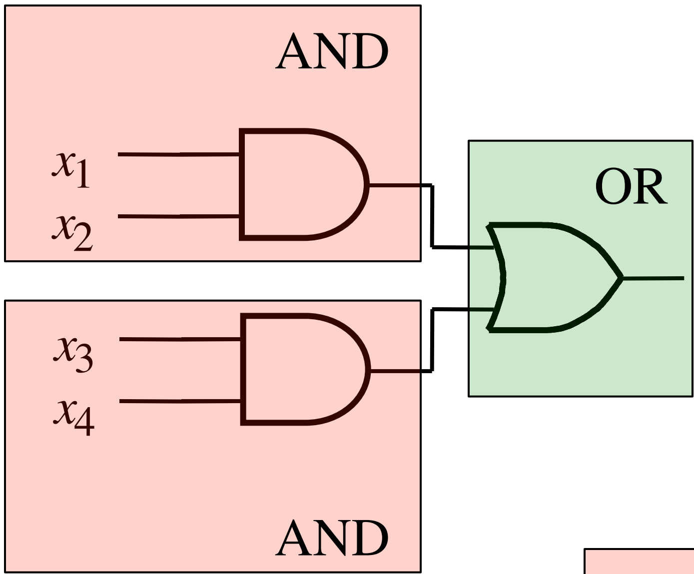


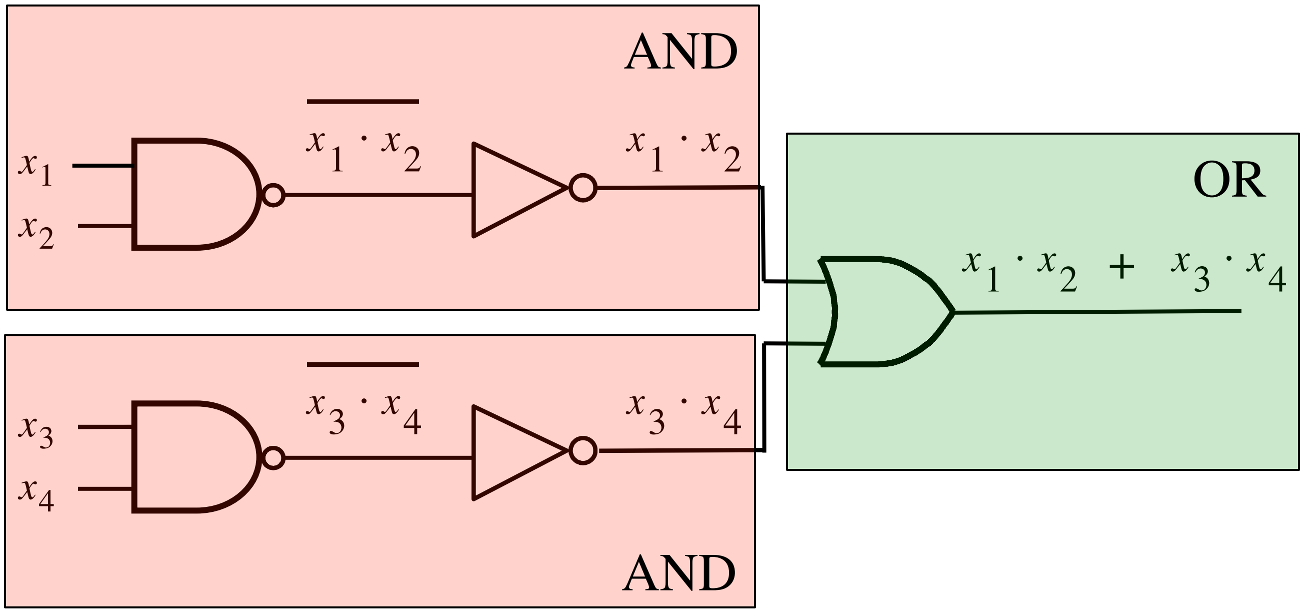
Sum-Of-Products



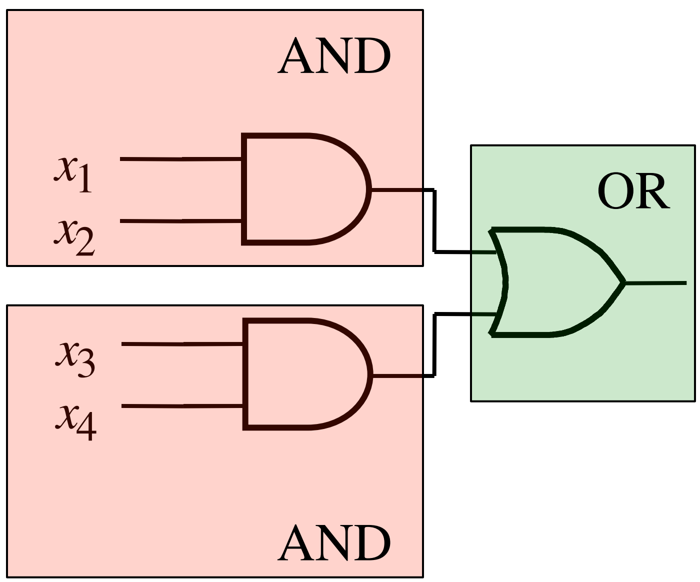


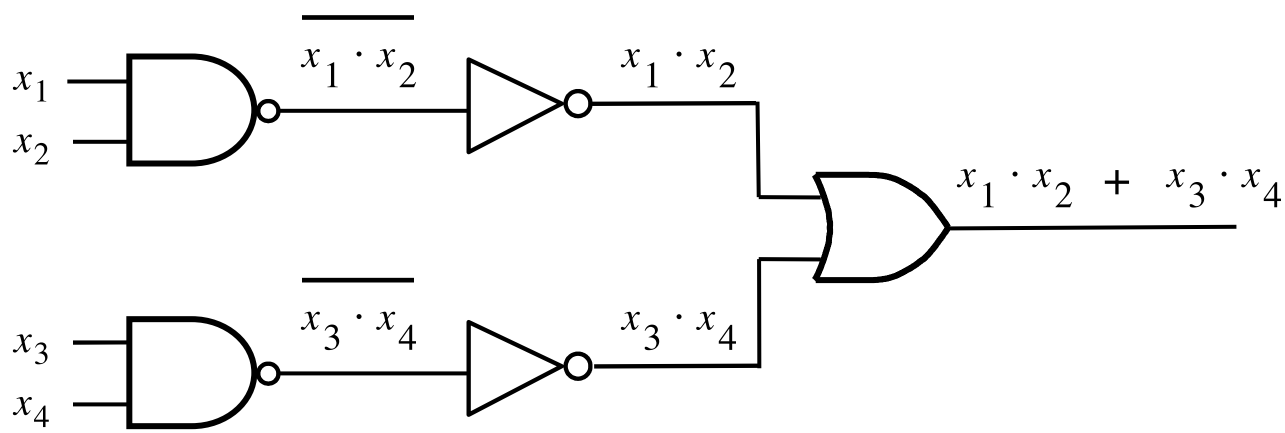
Sum-Of-Products



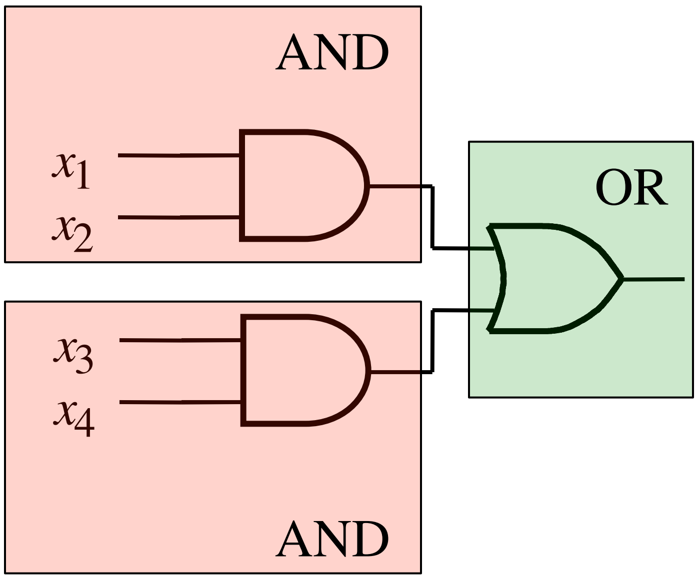


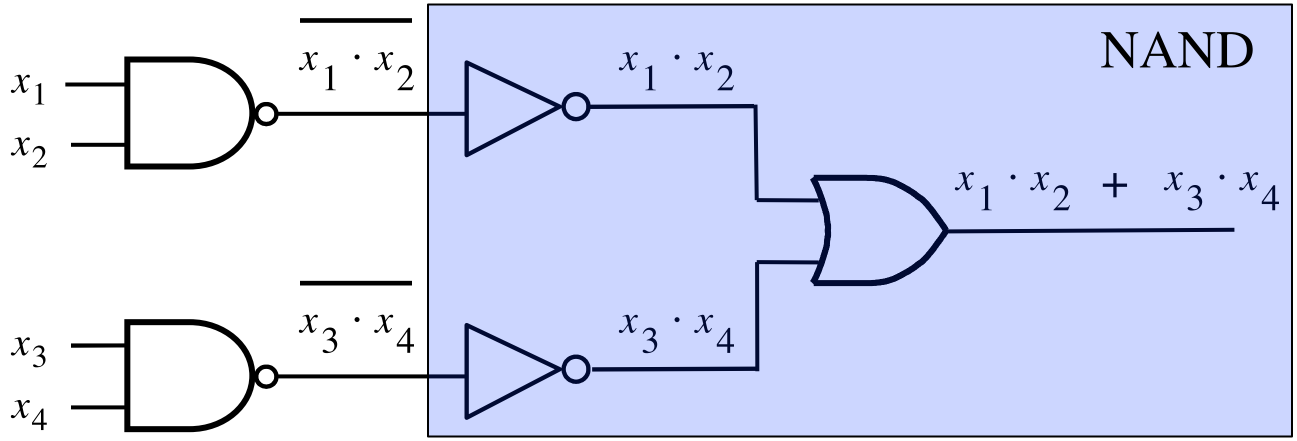
Sum-Of-Products



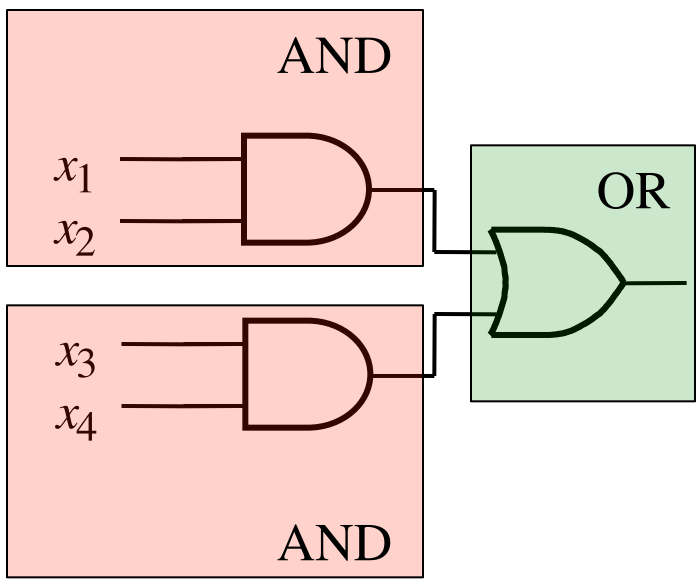


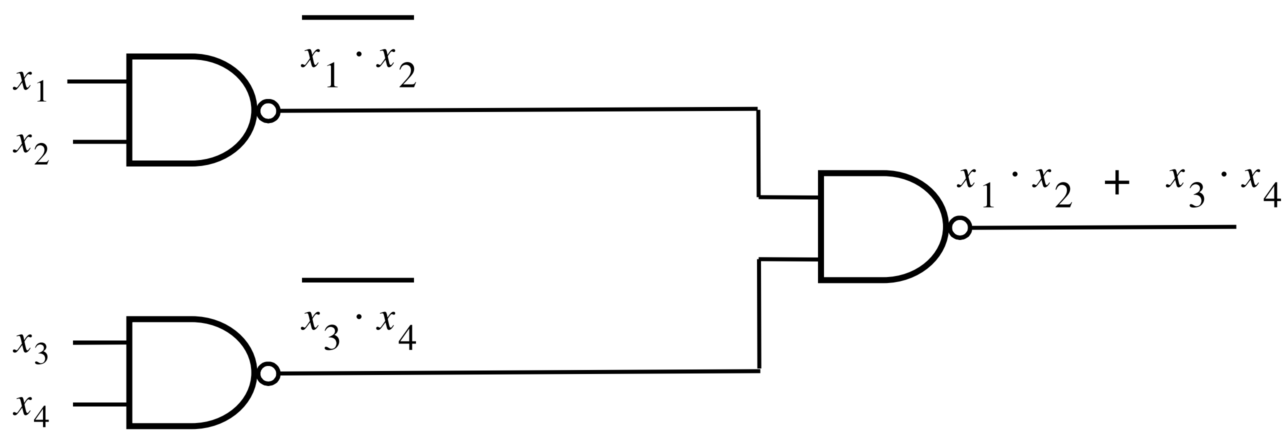
Sum-Of-Products



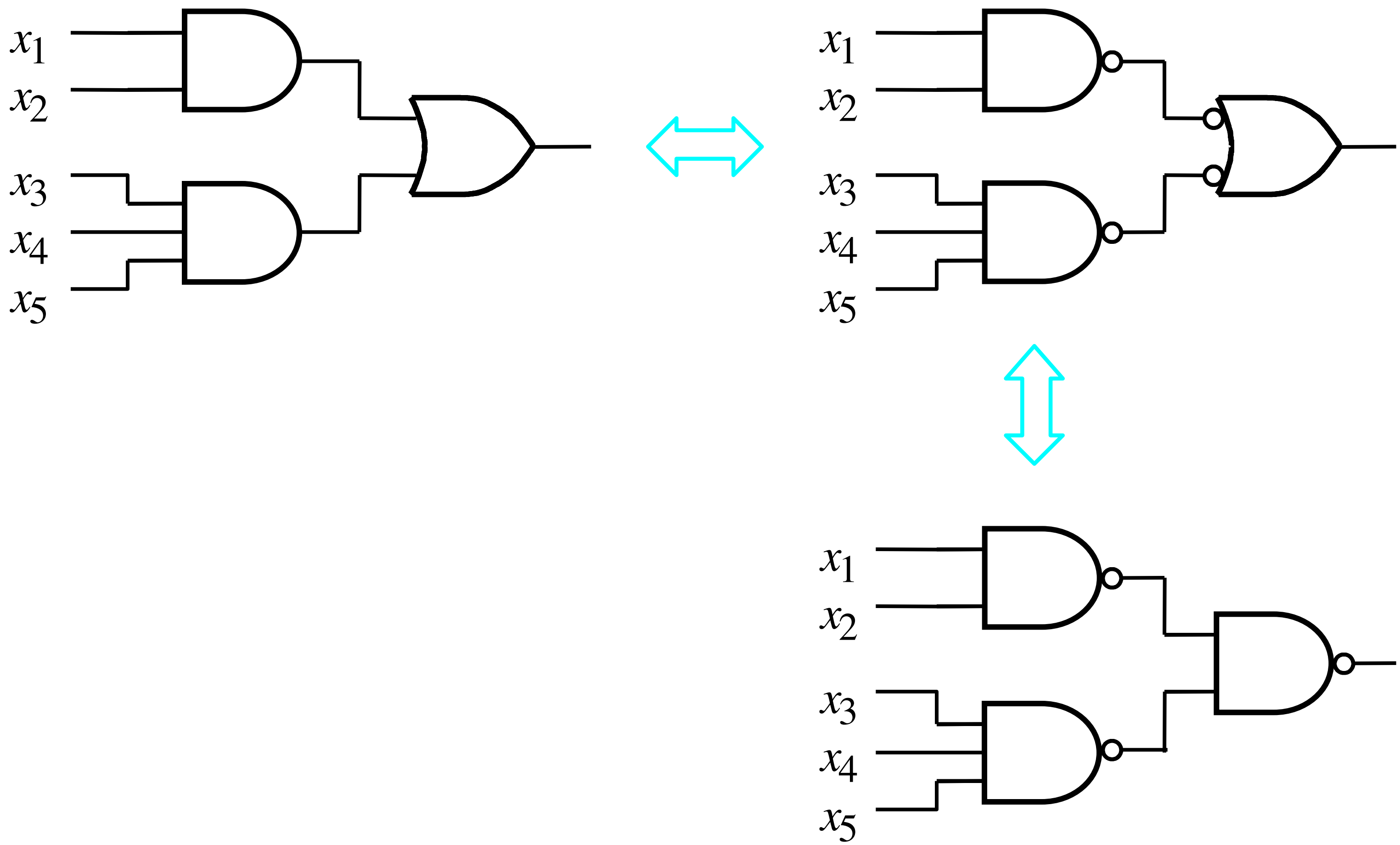


Sum-Of-Products





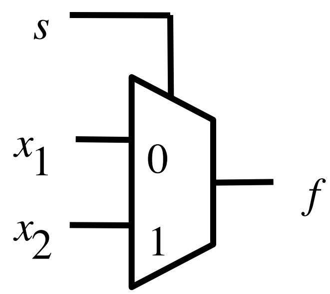
Sum-Of-Products



2-1 Multiplexer (Definition)

- Has two inputs: x_1 and x_2
- Also has another input line s
- If s=0, then the output is equal to x_1
- If s=1, then the output is equal to x_2

Graphical Symbol for a 2-1 Multiplexer



[Figure 2.33c from the textbook]

Let's Derive the SOP form

$s\ x_1\ x_2$	$f(s, x_1, x_2)$	
0 0 0	0	
0 0 1	0	
0 1 0	1	$\overline{s}\ x_1\ \overline{x}_2$
0 1 1	1	$\overline{s}\ x_1\ x_2$
1 0 0	0	
1 0 1	1	$s\ \overline{x}_1\ x_2$
1 1 0	0	
1 1 1	1	$s\ x_1\ x_2$

$$f(s, x_1, x_2) = \overline{s}\ x_1\ \overline{x}_2 + \overline{s}\ x_1\ x_2 + s\ \overline{x}_1\ x_2 + s\ x_1\ x_2$$

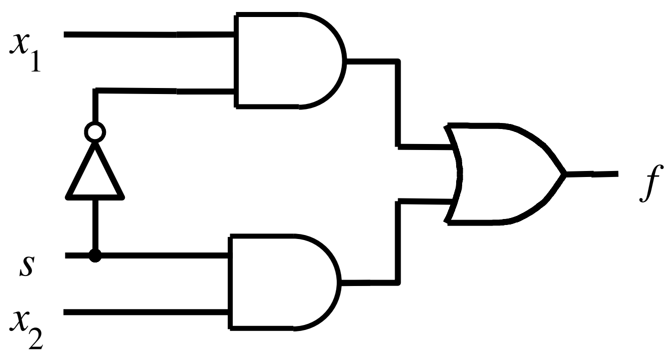
Let's simplify this expression

$$f(s, x_1, x_2) = \overline{s}\, x_1 \overline{x_2} + \overline{s}\, x_1 x_2 + s\, \overline{x_1}\, x_2 + s\, x_1 x_2$$

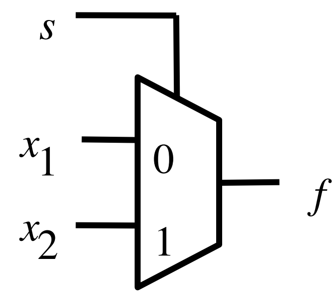
$$f(s, x_1, x_2) = \overline{s}\, x_1 (\overline{x_2} + x_2) + s\, (\overline{x_1} + x_1) x_2$$

$$f(s, x_1, x_2) = \overline{s}\, x_1 + s\, x_2$$

Circuit for 2-1 Multiplexer



(b) Circuit



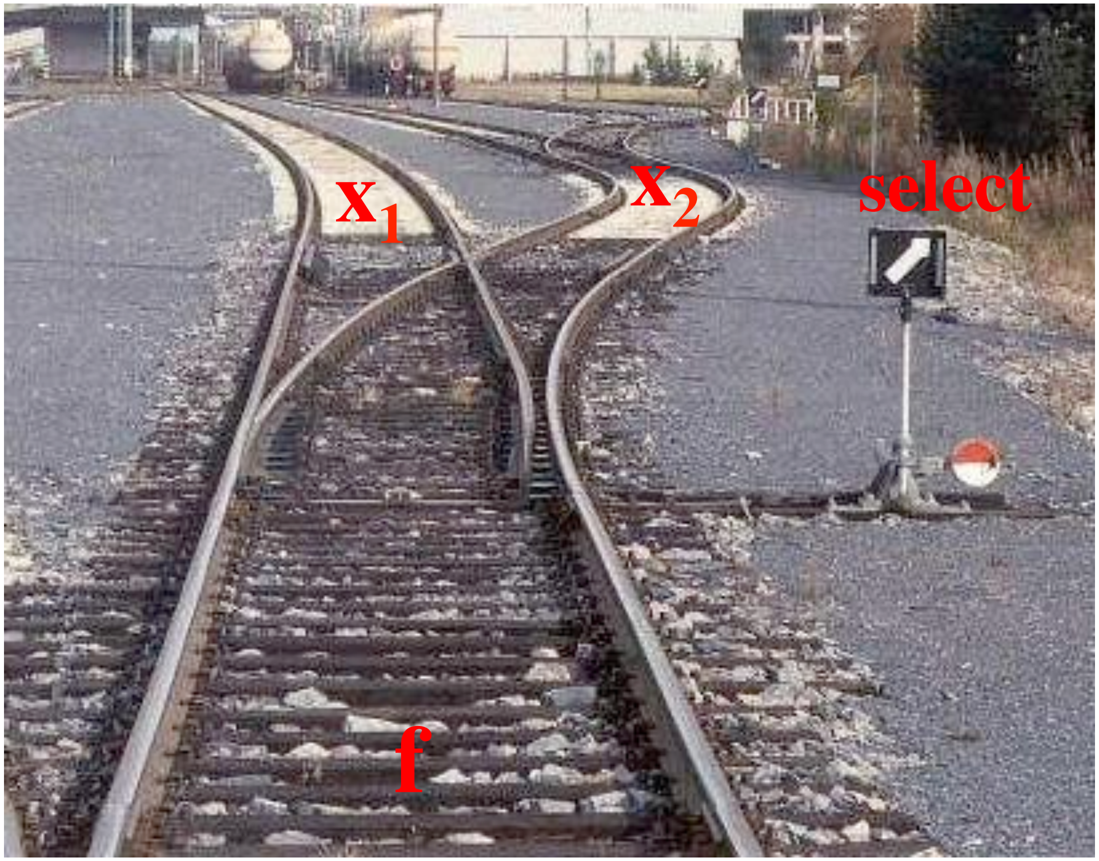
(c) Graphical symbol

[Figure 2.33b-c from the textbook]

Analogy: Railroad Switch

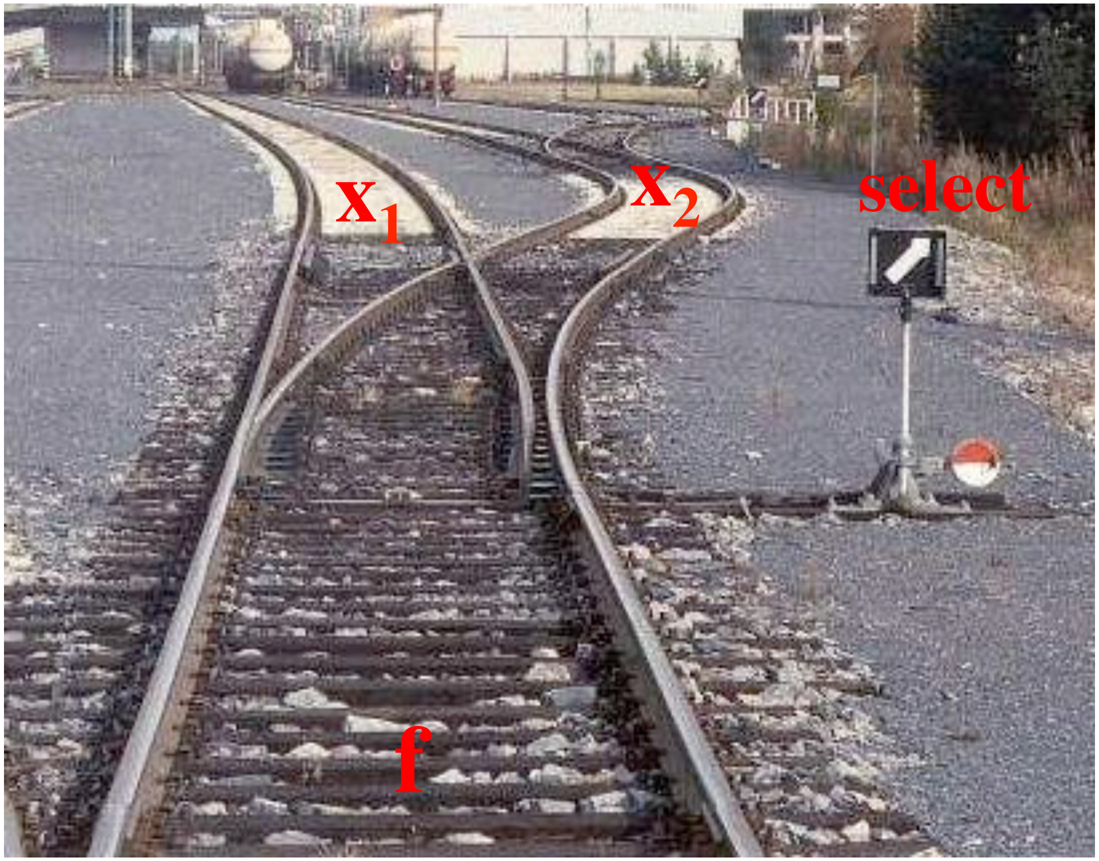
http://en.wikipedia.org/wiki/Railroad_switch]

Analogy: Railroad Switch



http://en.wikipedia.org/wiki/Railroad_switch]

Analogy: Railroad Switch



This is not a perfect analogy because the trains can go in either direction, while the multiplexer would only allow them to go from top to bottom.

http://en.wikipedia.org/wiki/Railroad_switch]

More Compact Truth-Table Representation

$s\ x_1\ x_2$	$f(s, x_1, x_2)$
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

(a)Truth table

s	$f(s, x_1, x_2)$
0	x_1
1	x_2

[Figure 2.33 from the textbook]

4-1 Multiplexer (Definition)

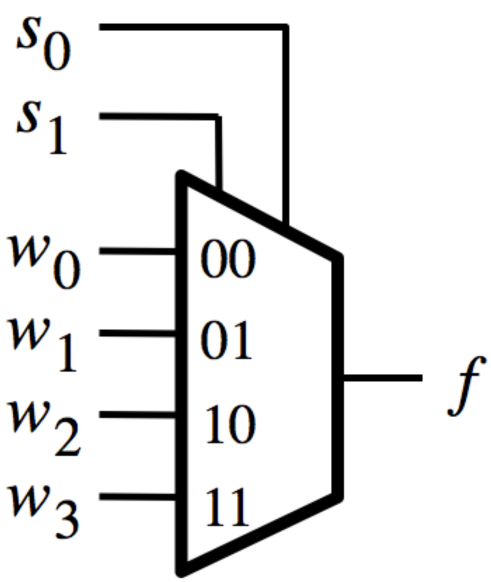
- **Has four inputs: w_0 , w_1, w_2, w_3**

- **Also has two select lines: s_1 and s_0**

- **If s_1=0 and s_0=0, then the output f is equal to w_0**
- **If s_1=0 and s_0=1, then the output f is equal to w_1**
- **If s_1=1 and s_0=0, then the output f is equal to w_2**
- **If s_1=1 and s_0=1, then the output f is equal to w_3**

We'll talk more about this when we get
to chapter 4, but here is a quick preview.

Graphical Symbol and Truth Table



(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) Truth table

[Figure 4.2a-b from the textbook]

The long-form truth table

S_1	S_0	I_3	I_2	I_1	I_0	F
0	0	0	0	0	0	0
		0	0	0	1	1
		0	0	1	0	0
		0	0	1	1	1
		0	1	0	0	0
		0	1	0	1	1
		0	1	1	0	0
		0	1	1	1	1
		1	0	0	0	0
		1	0	0	1	1
		1	0	1	0	0
		1	0	1	1	1
		1	1	0	0	0
		1	1	0	1	1
		1	1	1	0	0
		1	1	1	1	1

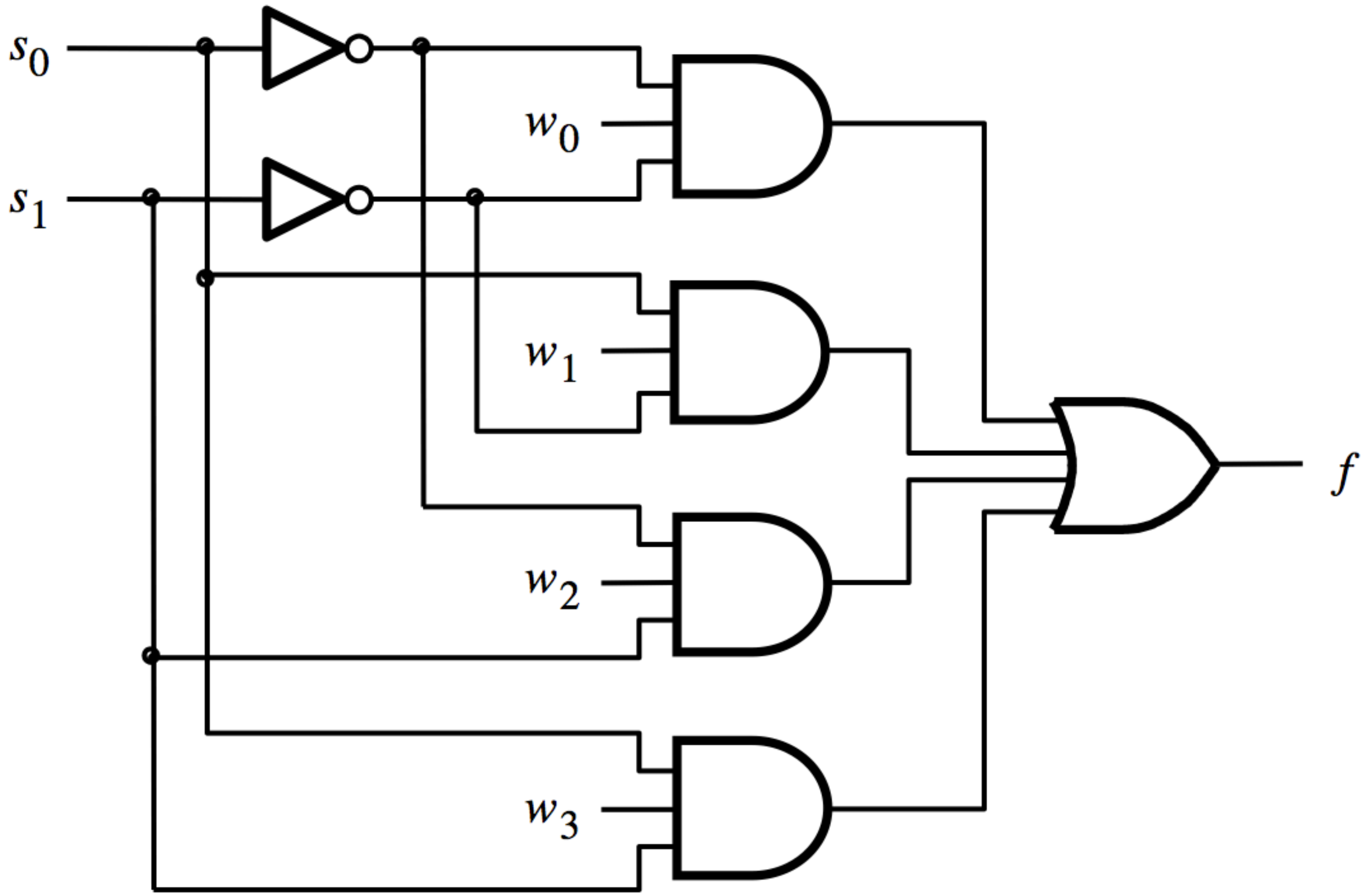
S_1	S_0	I_3	I_2	I_1	I_0	F
0	1	0	0	0	0	0
		0	0	0	1	0
		0	0	1	0	1
		0	0	1	1	1
		0	1	0	0	0
		0	1	0	1	0
		0	1	1	0	1
		0	1	1	1	1
		1	0	0	0	0
		1	0	0	1	0
		1	0	1	0	1
		1	0	1	1	1
		1	1	0	0	0
		1	1	0	1	0
		1	1	1	0	1
		1	1	1	1	1

S_1	S_0	I_3	I_2	I_1	I_0	F
1	0	0	0	0	0	0
		0	0	0	1	0
		0	0	1	0	0
		0	0	1	1	0
		0	1	0	0	1
		0	1	0	1	1
		0	1	1	0	1
		0	1	1	1	1
		1	0	0	0	0
		1	0	0	1	0
		1	0	1	0	0
		1	0	1	1	0
		1	1	0	0	1
		1	1	0	1	1
		1	1	1	0	1
		1	1	1	1	1

S_1	S_0	I_3	I_2	I_1	I_0	F
1	1	0	0	0	0	0
		0	0	0	1	0
		0	0	1	0	0
		0	0	1	1	0
		0	1	0	0	0
		0	1	0	1	0
		0	1	1	0	0
		0	1	1	1	0
		1	0	0	0	1
		1	0	0	1	1
		1	0	1	0	1
		1	0	1	1	1
		1	1	0	0	1
		1	1	0	1	1
		1	1	1	0	1
		1	1	1	1	1

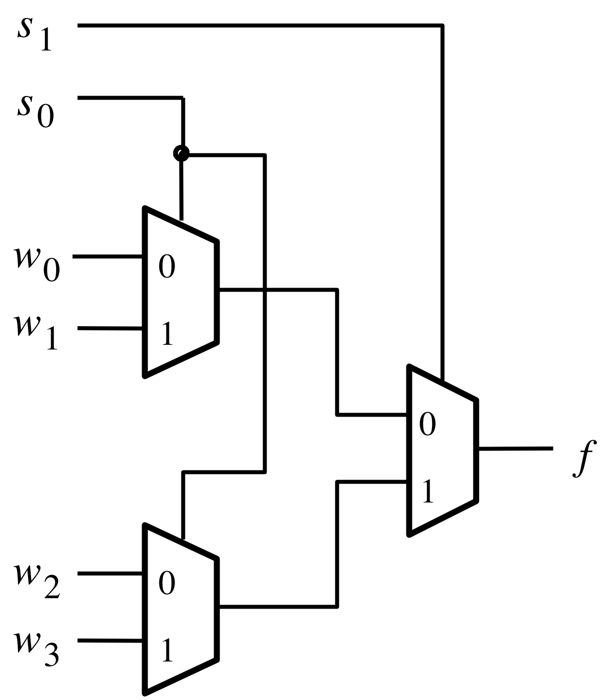
[http://www.absoluteastronomy.com/topics/Multiplexer]

4-1 Multiplexer (SOP circuit)



[Figure 4.2c from the textbook]

Using three 2-to-1 multiplexers to build one 4-to-1 multiplexer

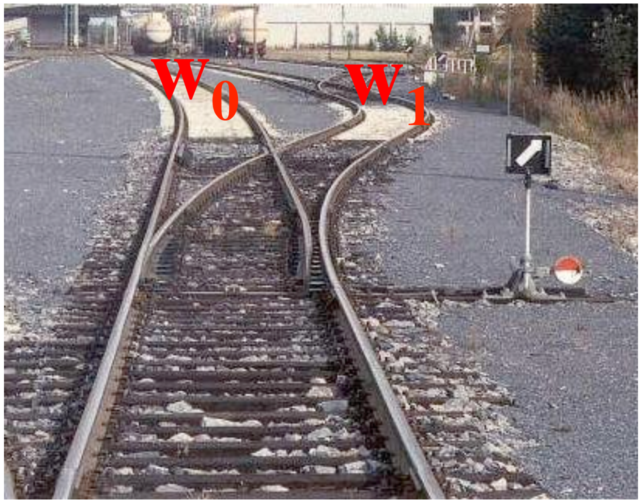


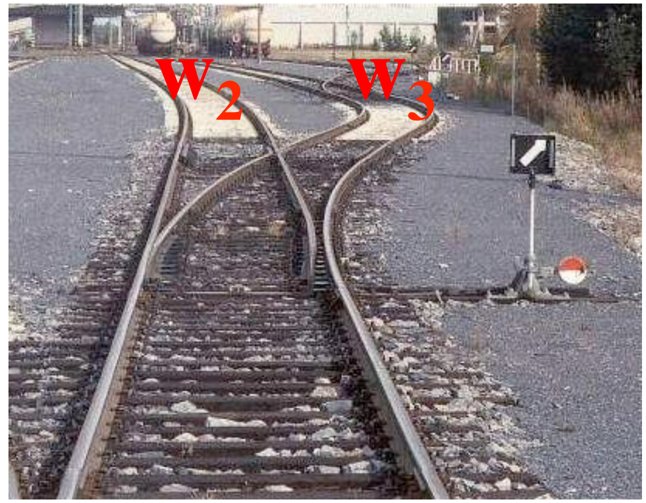
[Figure 4.3 from the textbook]

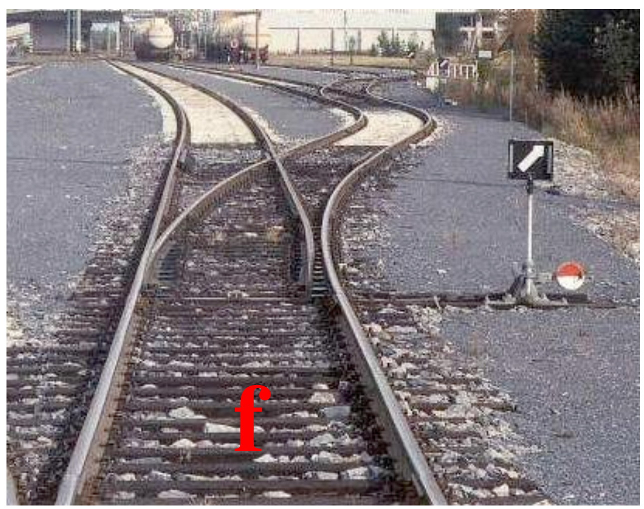
Analogy: Railroad Switches

http://en.wikipedia.org/wiki/Railroad_switch]

Analogy: Railroad Switches



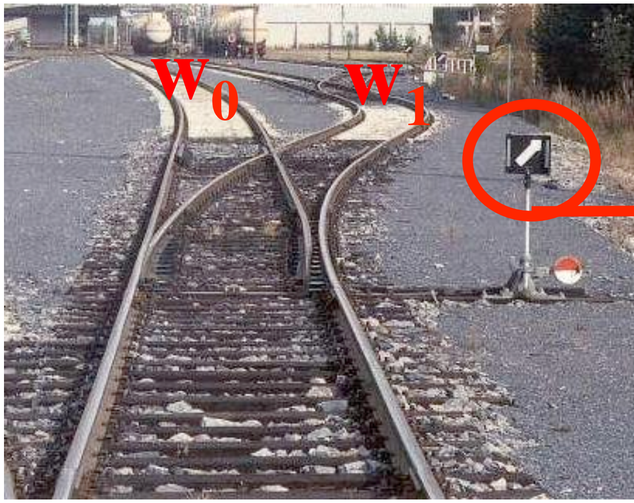


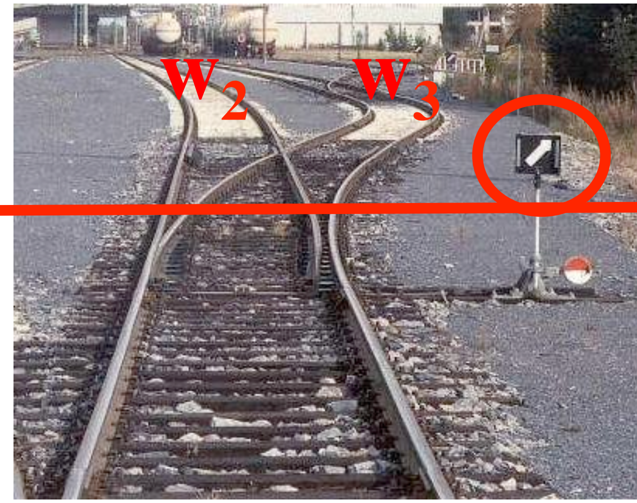


s_1

http://en.wikipedia.org/wiki/Railroad_switch]

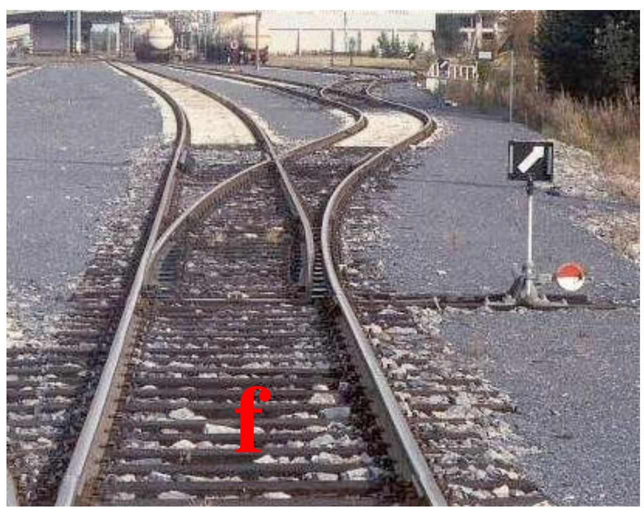
Analogy: Railroad Switches





s_0

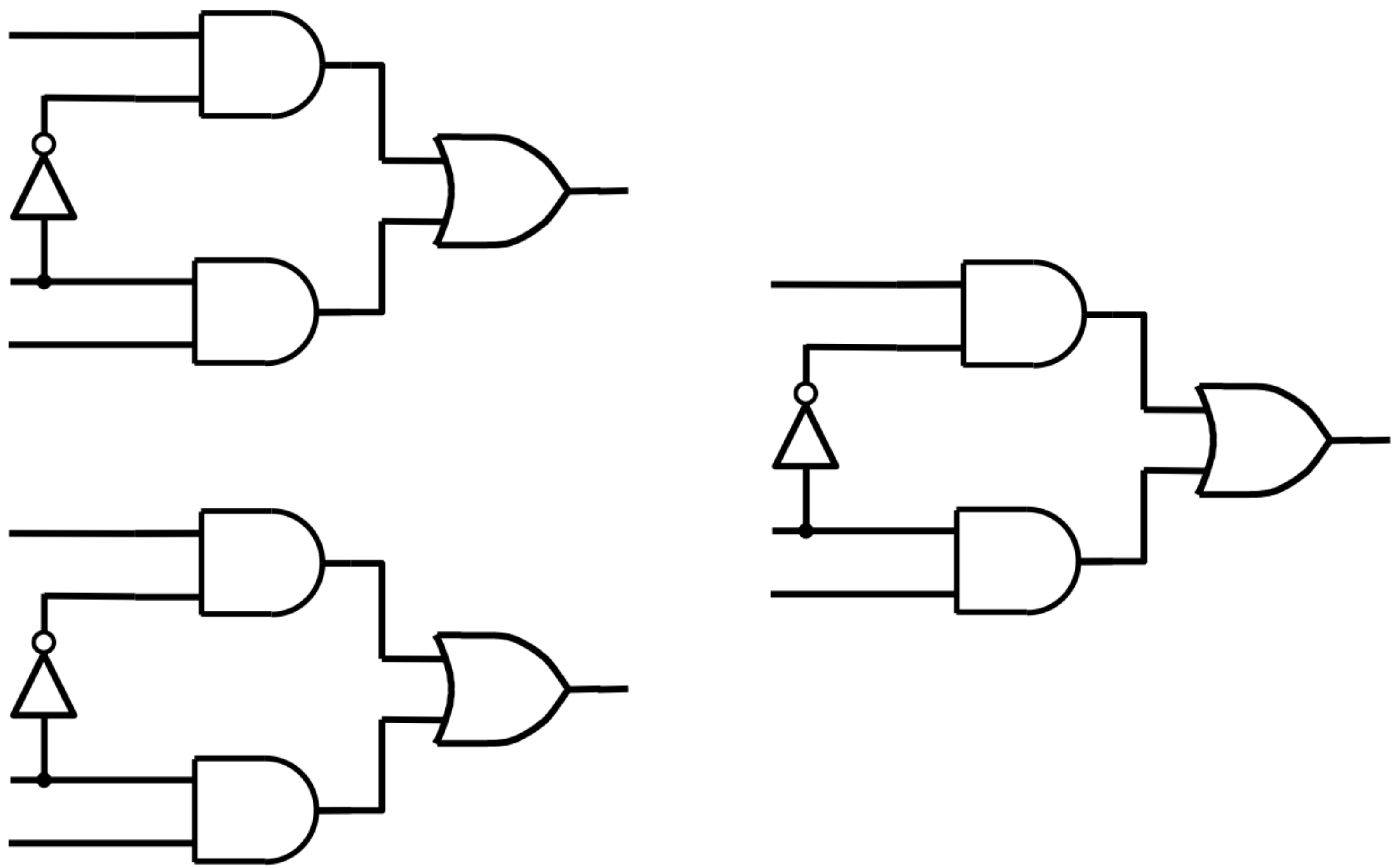
these two switches are controlled together



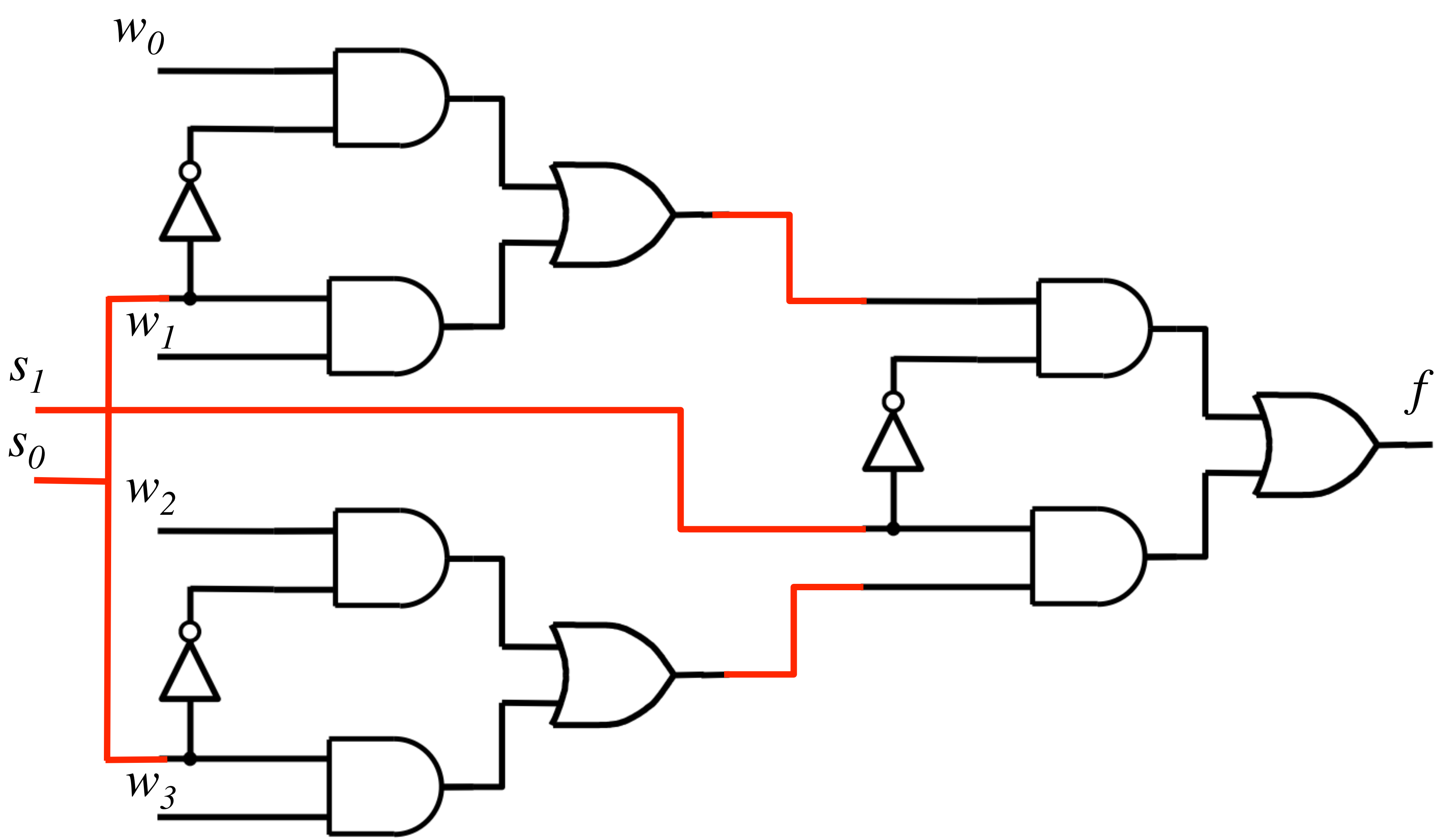
s_1

http://en.wikipedia.org/wiki/Railroad_switch]

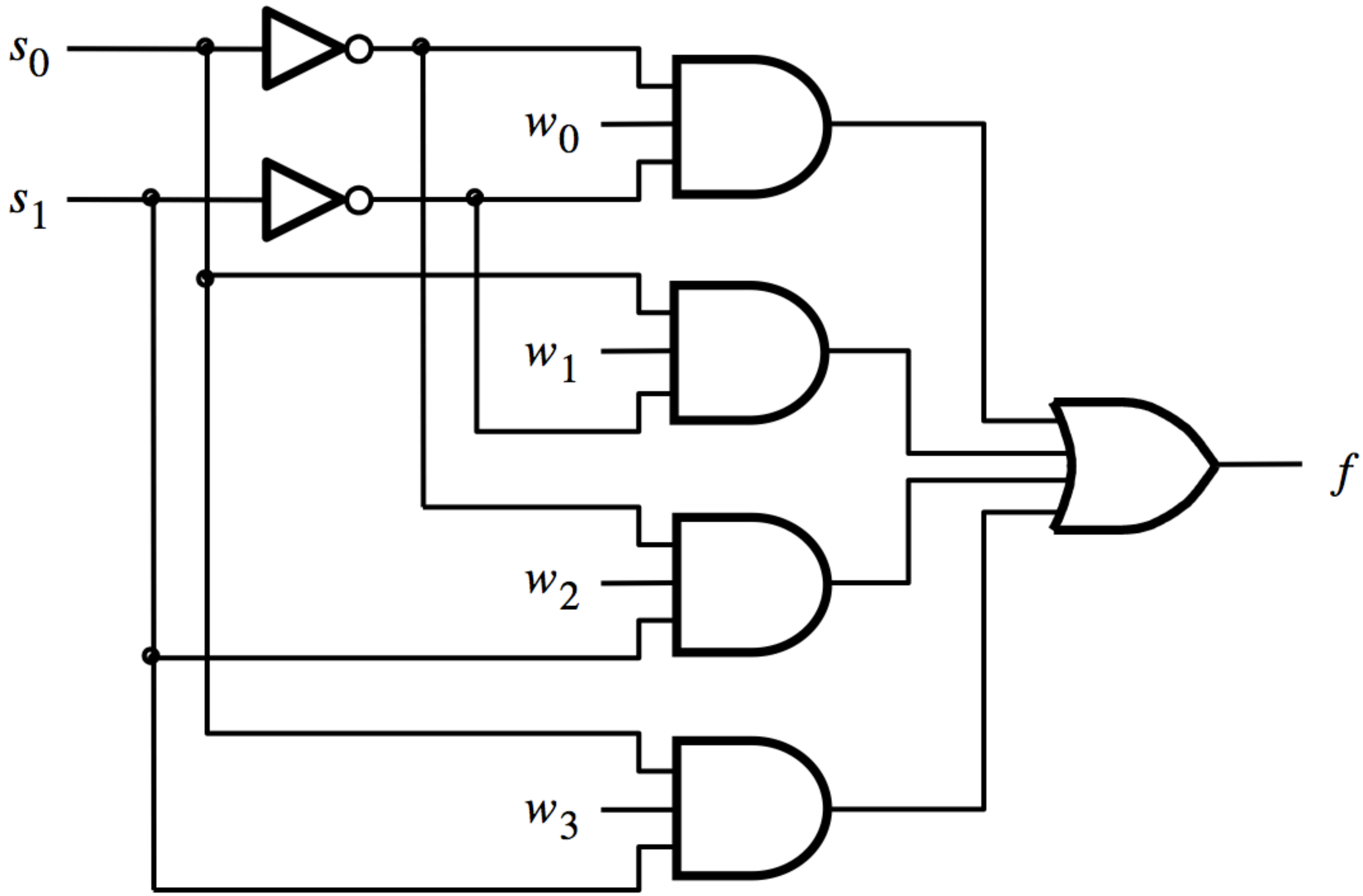
Using three 2-to-1 multiplexers to build one 4-to-1 multiplexer



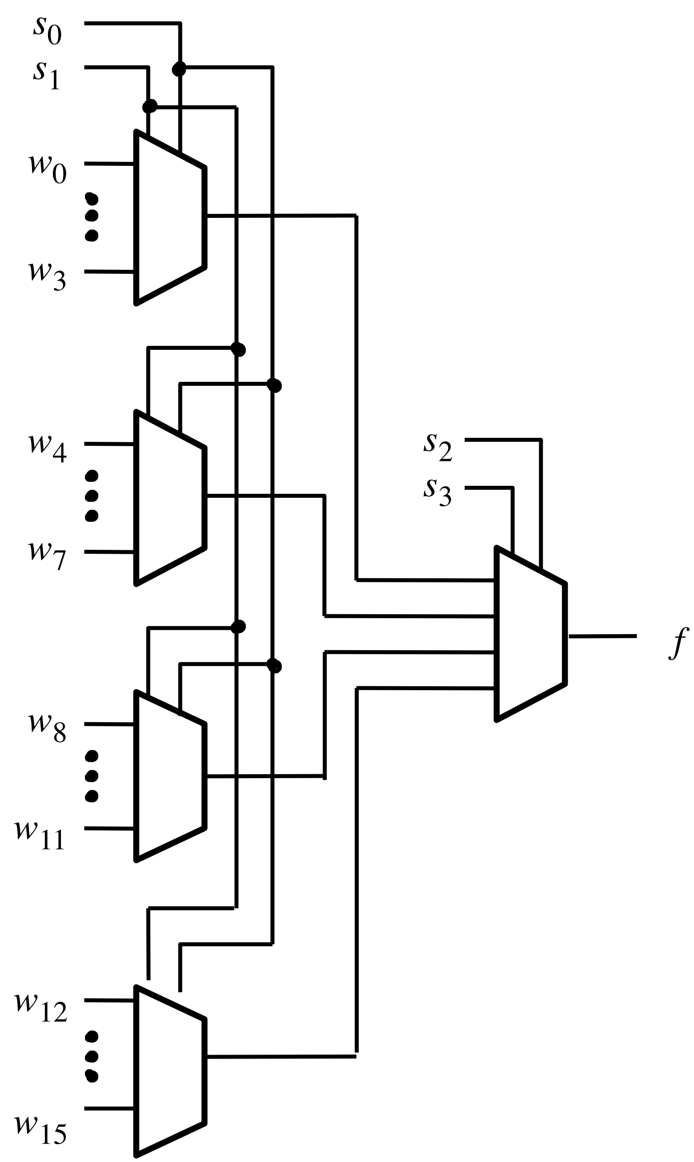
Using three 2-to-1 multiplexers to build one 4-to-1 multiplexer



That is different from the SOP form of the 4-1 multiplexer shown below, which uses less gates



16-1 Multiplexer

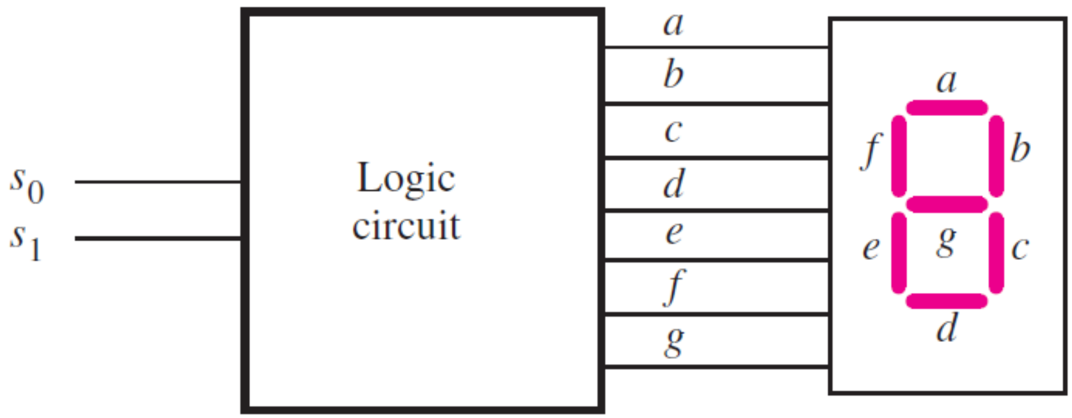


[Figure 4.4 from the textbook]

[http://upload.wikimedia.org/wikipedia/commons/2/26/SunsetTracksCrop.JPG]

7-Segment Display Example

Display of numbers



(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

[Figure 2.34 from the textbook]

Display of numbers

	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

Display of numbers

	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

$a = \overline{s_0}$ $c = \overline{s_1}$ $e = \overline{s_0}$ $g = s_1 \overline{s_0}$

$b = 1$ $d = \overline{s_0}$ $f = \overline{s_1}\,\overline{s_0}$

Intro to Verilog

History

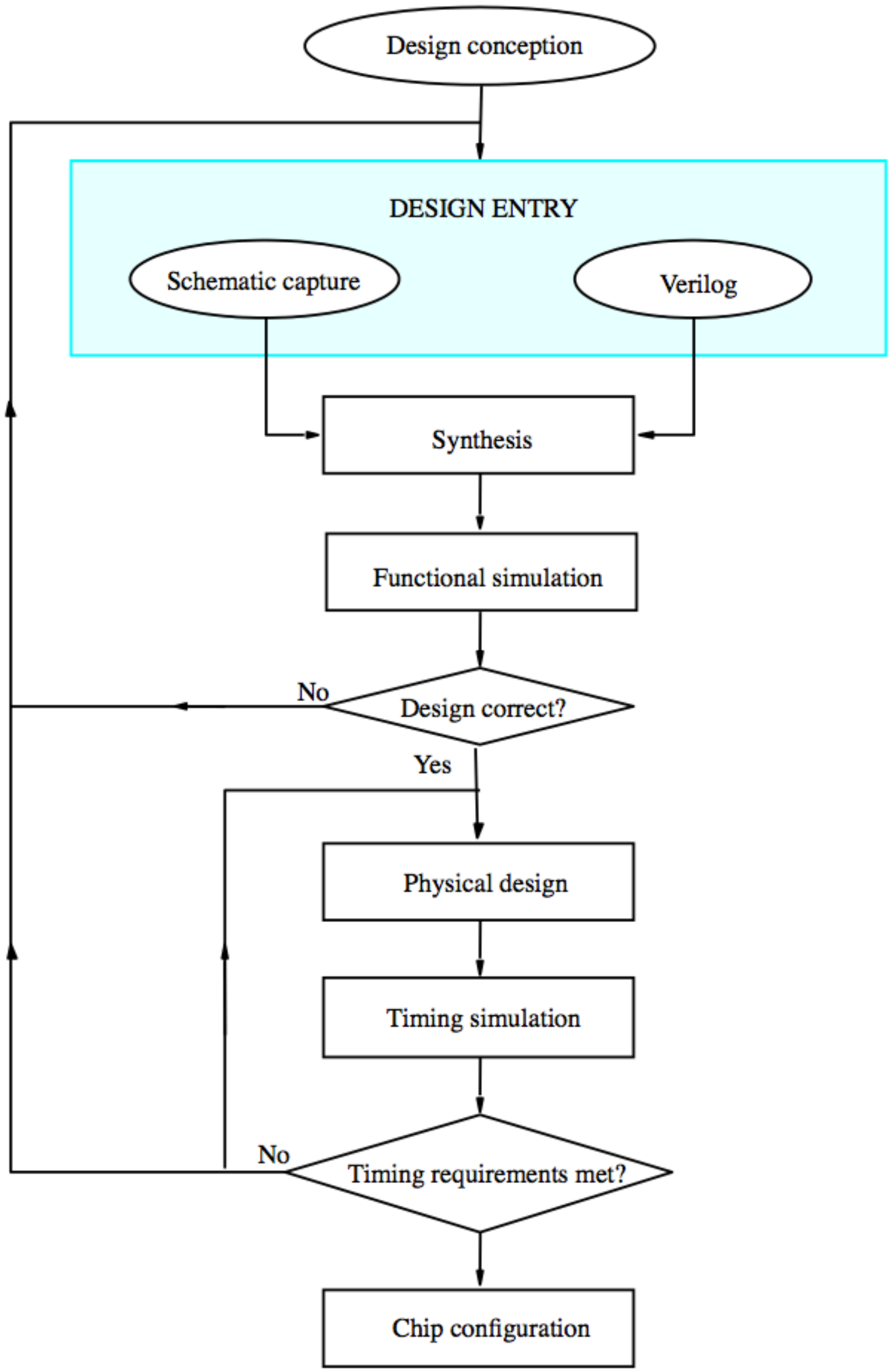
- Created in 1983/1984
- Verilog-95 (IEEE standard 1364-1995)
- Verilog 2001 (IEEE Standard 1364-2001)
- Verilog 2005 (IEEE Standard 1364-2005)
- SystemVerilog
- SystemVerilog 2009 (IEEE Standard 1800-2009).

HDL

- Hardware Description Language
- Verilog HDL
- VHDL

Verilog HDL != VHDL

- These are two different Languages!
- Verilog is closer to C
- VHDL is closer to Ada



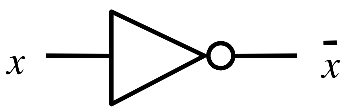
[Figure 2.35 from the textbook]

“Hello World” in Verilog

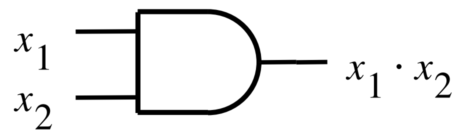
```
module main;
  initial
    begin
      $display("Hello world!");
      $finish;
    end
endmodule
```

[http://en.wikipedia.org/wiki/Verilog]

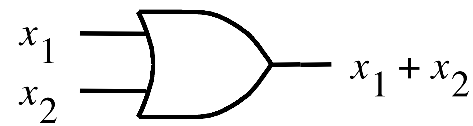
The Three Basic Logic Gates



NOT gate



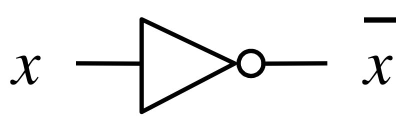
AND gate



OR gate

[Figure 2.8 from the textbook]

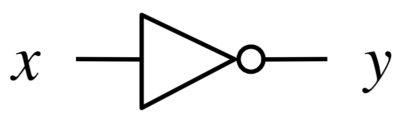
How to specify a NOT gate in Verilog



NOT gate

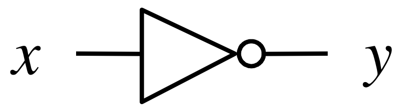
How to specify a NOT gate in Verilog

we'll use the letter y for the output



NOT gate

How to specify a NOT gate in Verilog

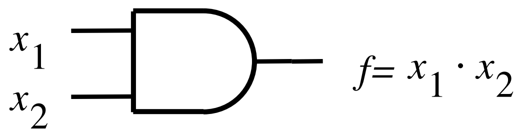


NOT gate

```
not (y, x)
```

Verilog code

How to specify an AND gate in Verilog

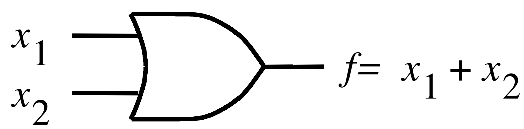


and (f, x1, x2)

AND gate

Verilog code

How to specify an OR gate in Verilog

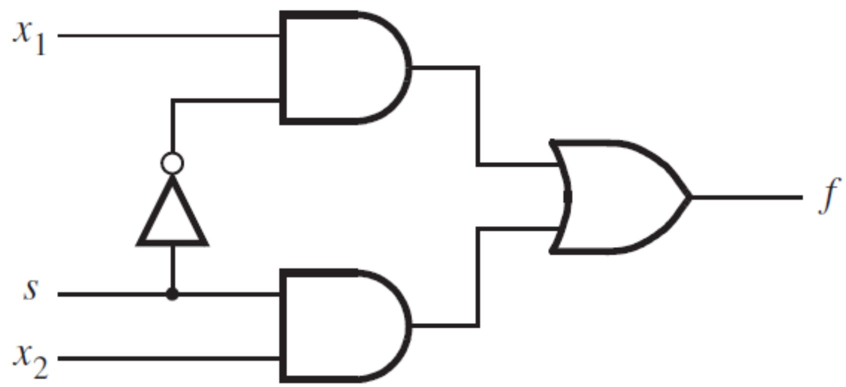


OR gate

or (f, x1, x2)

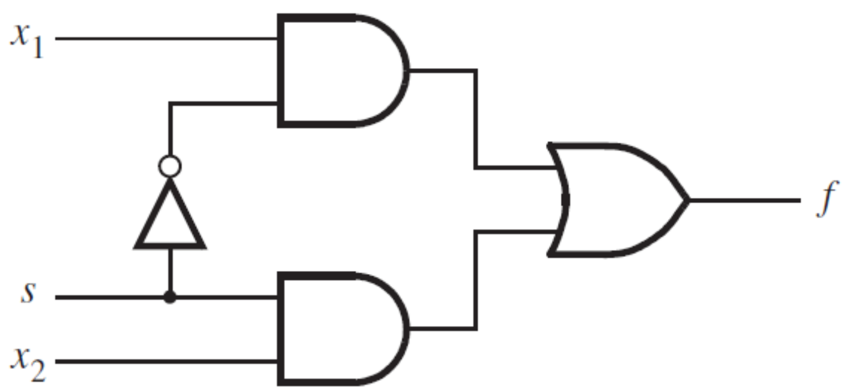
Verilog code

2-1 Multiplexer



[Figure 2.36 from the textbook]

Verilog Code for a 2-1 Multiplexer

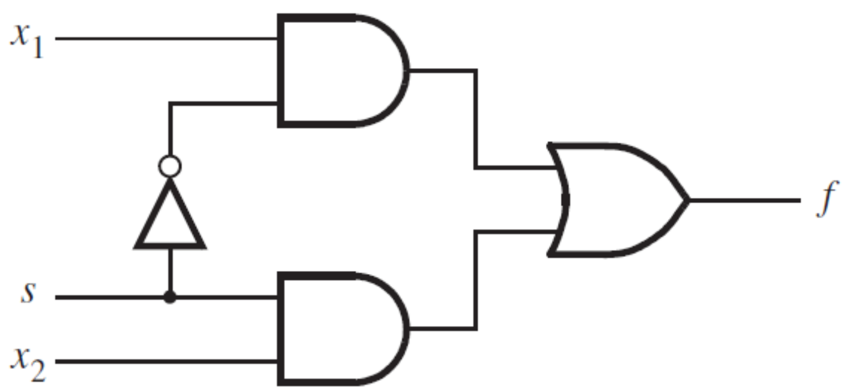


[Figure 2.36 from the textbook]

```
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
```

[Figure 2.37 from the textbook]

Verilog Code for a 2-1 Multiplexer

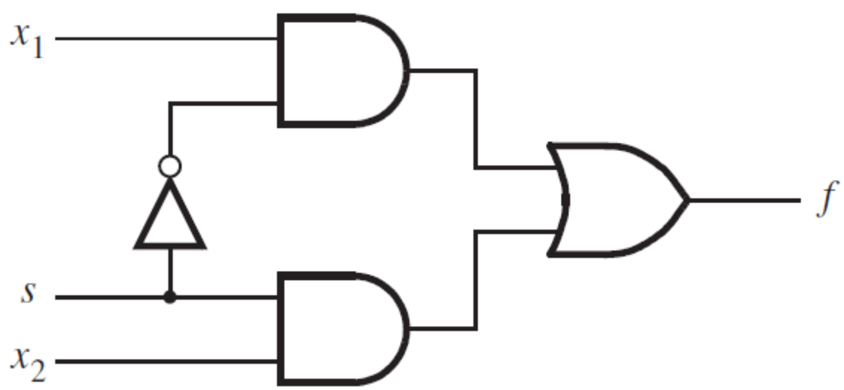



```
module  example3 (x1, x2, s, f);
    input x1, x2, s;
    output f;

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

endmodule
```

[Figure 2.36 from the textbook]

[Figure 2.40 from the textbook]

Verilog Code for a 2-1 Multiplexer

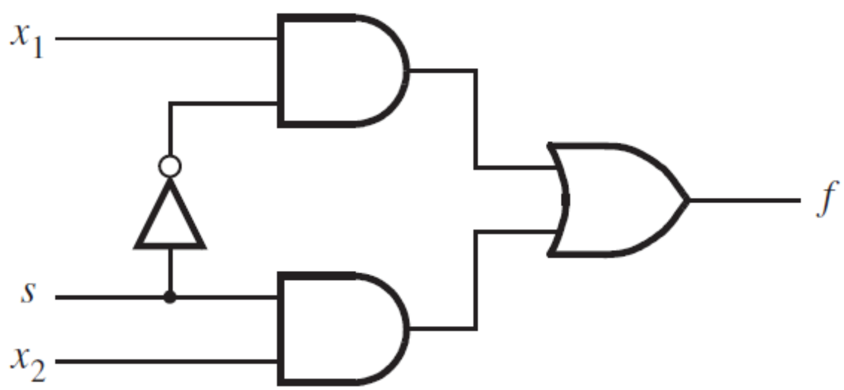


[Figure 2.36 from the textbook]

```
// 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 from the textbook]

Verilog Code for a 2-1 Multiplexer



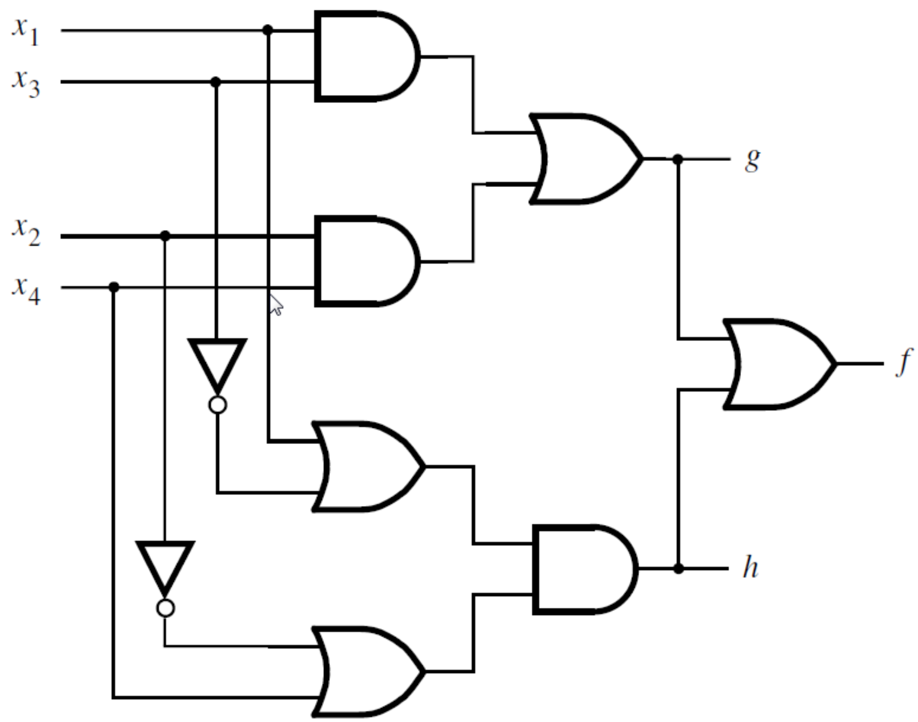

```
// Behavioral specification
module  example5 (input x1, x2, s, output reg f);

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

endmodule
```

[Figure 2.36 from the textbook]

[Figure 2.43 from the textbook]

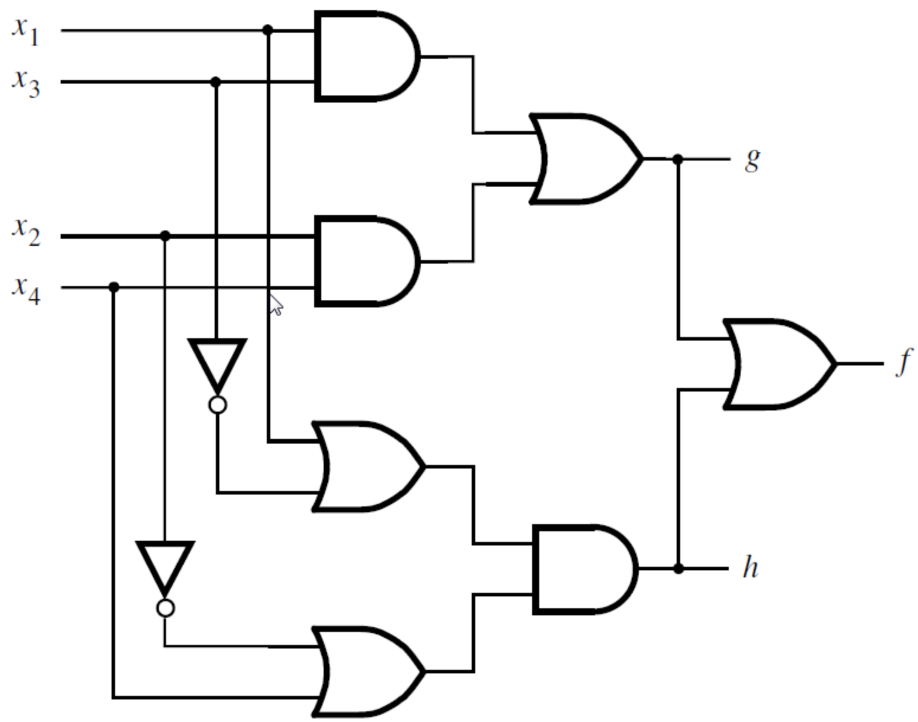
Another Example

Let’s Write the Code for This Circuit



[Figure 2.39 from the textbook]

Let's Write the Code for This Circuit

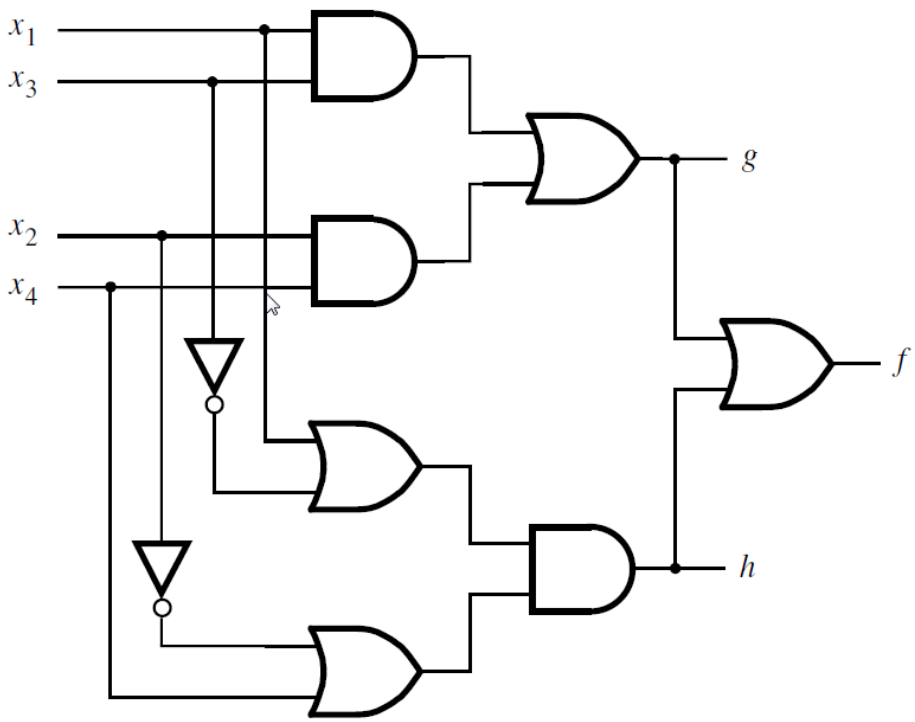



```
module example2 (x1, x2, x3, x4, f, g, h);
        input x1, x2, x3, x4;
        output f, g, h;

        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 from the textbook]

[Figure 2.38 from the textbook]

Let’s Write the Code for This Circuit



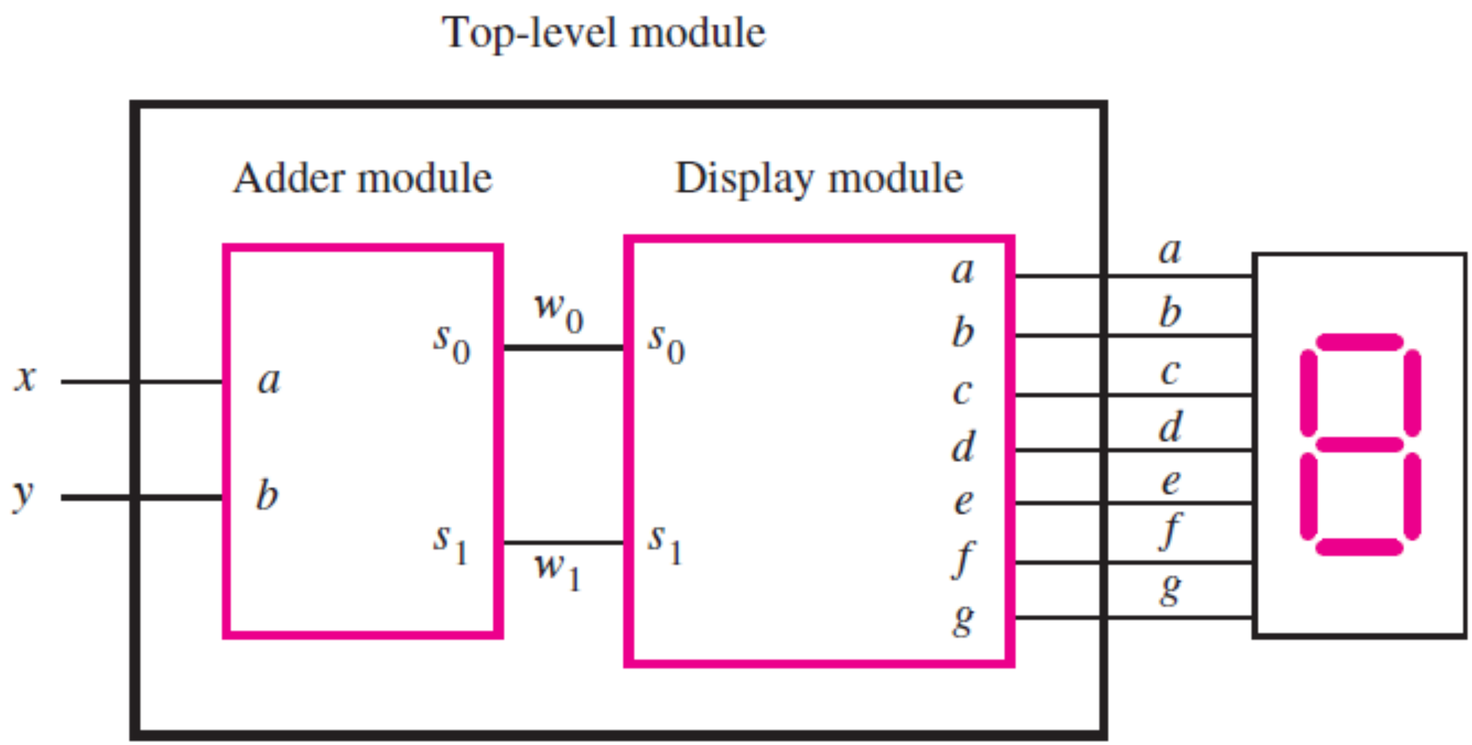

```
module example4 (x1, x2, x3, x4, f, g, h);
        input x1, x2, x3, x4;
        output f, g, h;

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

endmodule
```

[Figure 2.39 from the textbook]

[Figure 2.41 from the textbook]

Yet Another Example

A logic circuit with two modules



[Figure 2.44 from the textbook]

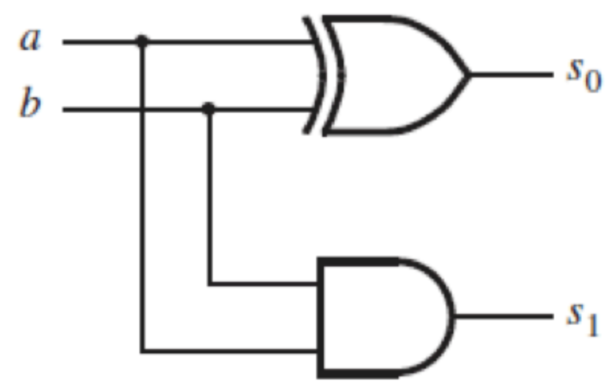
The adder module

a	0	0	1	1
$+\,b$	$+\,0$	$+\,1$	$+\,0$	$+\,1$
$s_1\,s_0$	0 0	0 1	0 1	1 0

(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

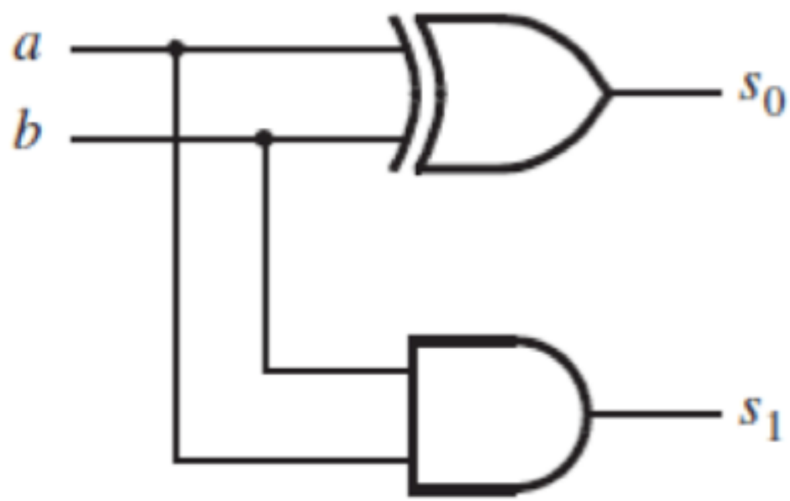
(b) Truth table



(c) Logic network

[Figure 2.12 from the textbook]

The adder module




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

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

endmodule
```

[Figure 2.45 from the textbook]

The display module

	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

$a = \overline{s_0}$ $c = \overline{s_1}$ $e = \overline{s_0}$ $g = s_1\overline{s_0}$

$b = 1$ $d = \overline{s_0}$ $f = \overline{s_1}\,\overline{s_0}$

The display module

$a = \overline{s_0}$

$b = 1$

$c = \overline{s_1}$

$d = \overline{s_0}$

$e = \overline{s_0}$

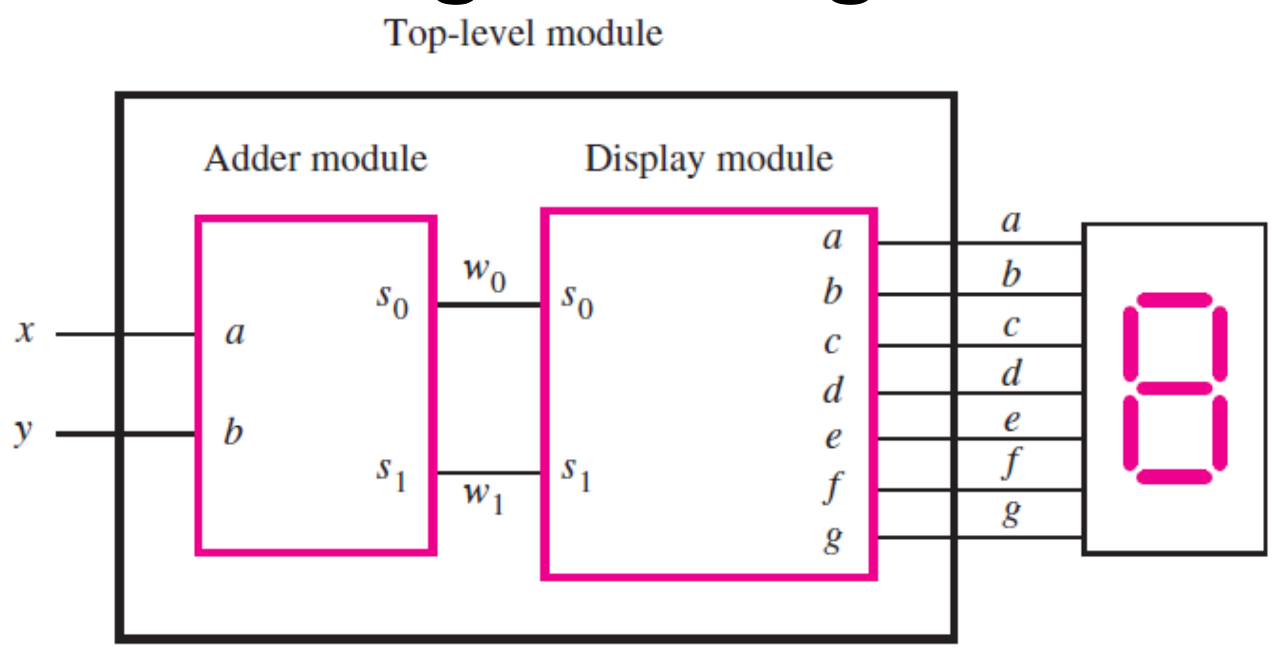
$f = \overline{s_1}\,\overline{s_0}$

$g = s_1 \overline{s_0}$

```
// 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 from the textbook]

Putting it all together




```
// 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
```

Questions?

THE END