Combinational-Circuit Building Blocks

Chapter 6

Chapter Objectives

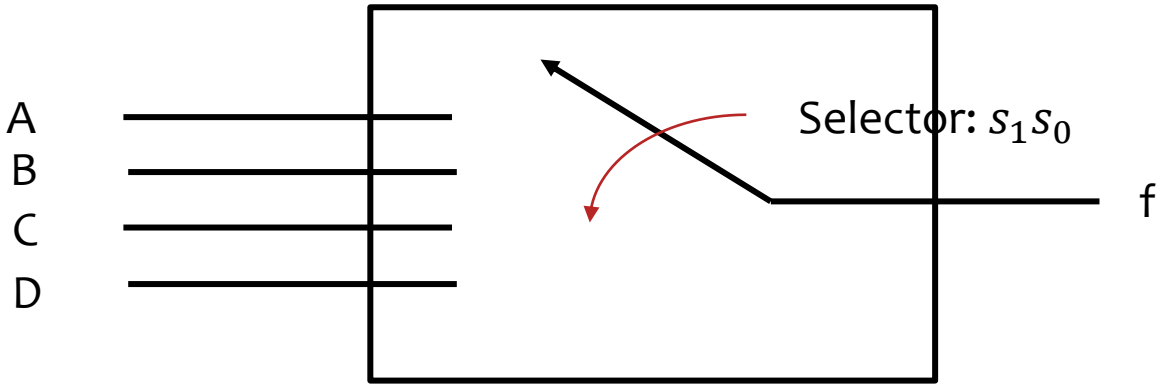
- Commonly used combinational sub-circuits
- Multiplexers, which can be used for selection of signals and for implementation of general logic functions
- Circuits used for encoding, decoding, and code conversion purposes

Contents

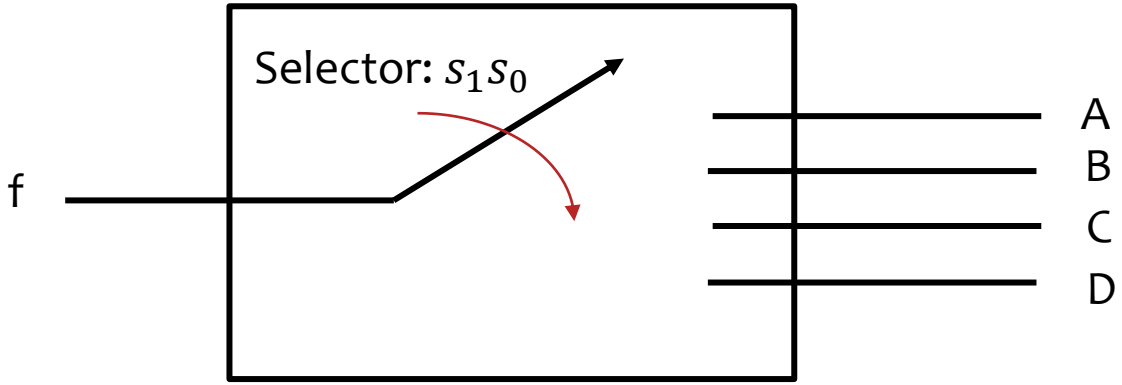
1. Multiplexers
2. Decoders
3. Encoders
4. Code Converters
5. Arithmetic Comparison Circuits

MULTIPLEXERS

Multiplexer Circuit



MUX -Multiplexer



DEMUX - Demultiplexer

Multiplexer Circuits

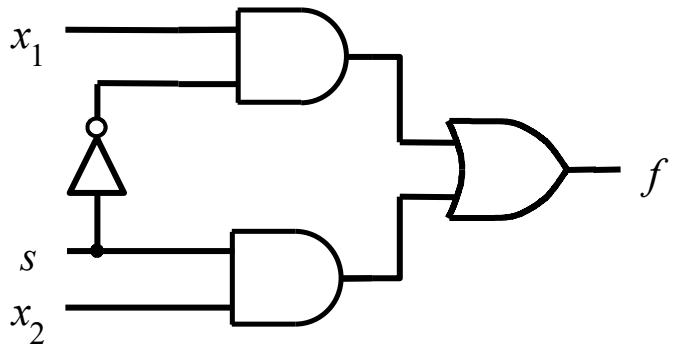
s	x_1	x_2	$f(s, x_1, x_2)$	
0	0	0	0	
0	0	1	0	
0	1	0	1	(1)
0	1	1	1	(2)
1	0	0	0	
1	0	1	1	(3)
1	1	0	0	
1	1	1	1	(4)

Truth Table

$$f = \overset{(1)}{s'x_1x_2'} + \overset{(2)}{s'x_1x_2} + \overset{(3)}{sx_1'x_2} + \overset{(4)}{sx_1x_2}$$

$$= s'x_1(x_2' + x_2) + sx_2(x_1' + x_1)$$

$$= s'x_1 + sx_2$$

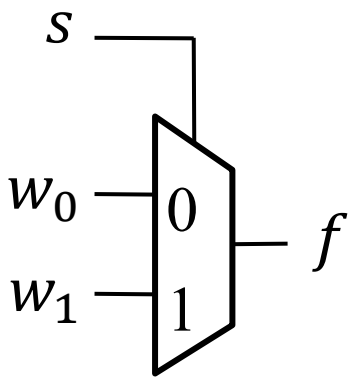


SOP Circuit

2-to-1 Multiplexer

$$f = s'w_0 + sw_1$$

$$f = \begin{cases} w_0 & s = 0 \\ w_1 & s = 1 \end{cases}$$



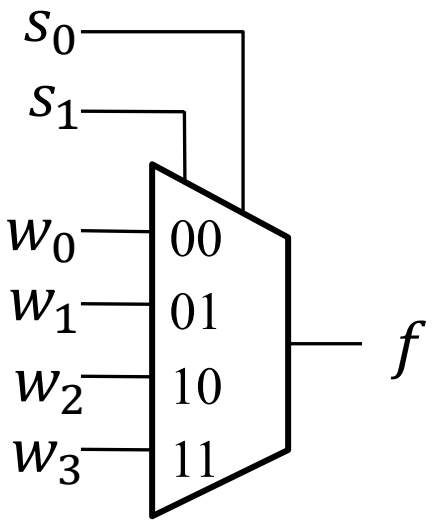
(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table

Figure 6.1 A 2-to-1 multiplexer

4-to-1 Multiplexer



(a) Graphic symbol

s_1	s_0	f	
0	0	w_0	$s_1's_0'w_0$
0	1	w_1	$s_1's_0w_1$
1	0	w_2	$s_1s_0'w_2$
1	1	w_3	$s_1s_0w_3$

(b) Truth table

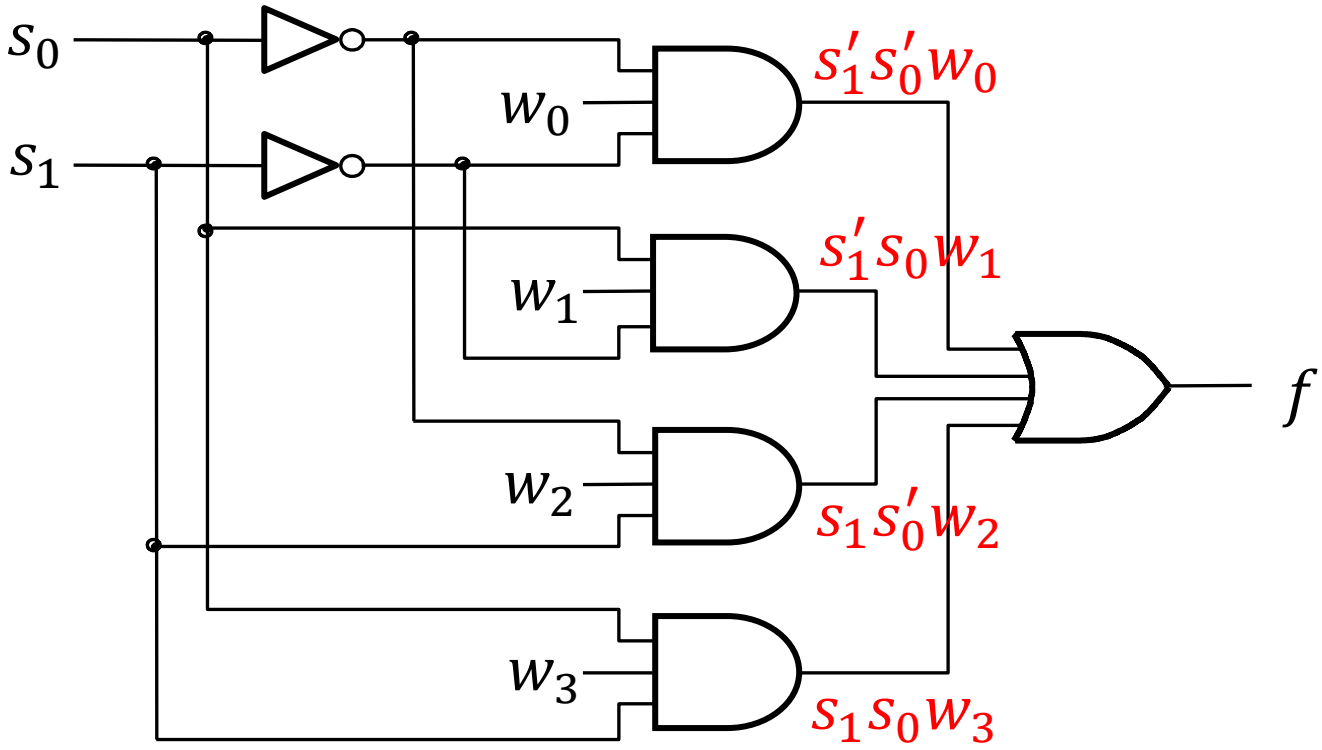
Sum of Products

$$f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_2 + s_1s_0w_3$$

Figure 6.2 A 4-to-1 multiplexer

4-to-1 Multiplexer

Sum of Products



(c) Circuit

Figure 6.2 A 4-to-1 multiplexer

Using 2-to-1 MUX to build a 4-to-1 MUX

s_0	f
0	w_0
1	w_1

(a) Truth table of 2-to-1 MUX

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

(b) Truth table of 4-to-1 MUX

$$f = s_0' w_0 + s_0 w_1$$

$$f = s_1' s_0' w_0 + s_1' s_0 w_1 + s_1 s_0' w_2 + s_1 s_0 w_3$$

$$= s_1'(s_0' w_0 + s_0 w_1) + s_1(s_0' w_2 + s_0 w_3)$$

$$W_0 = s_0' w_0 + s_0 w_1 \qquad W_1 = s_0' w_2 + s_0 w_3$$

$$= s_1' W_0 + s_1 W_1$$

Using 2-to-1 MUX to build a 4-to-1 MUX

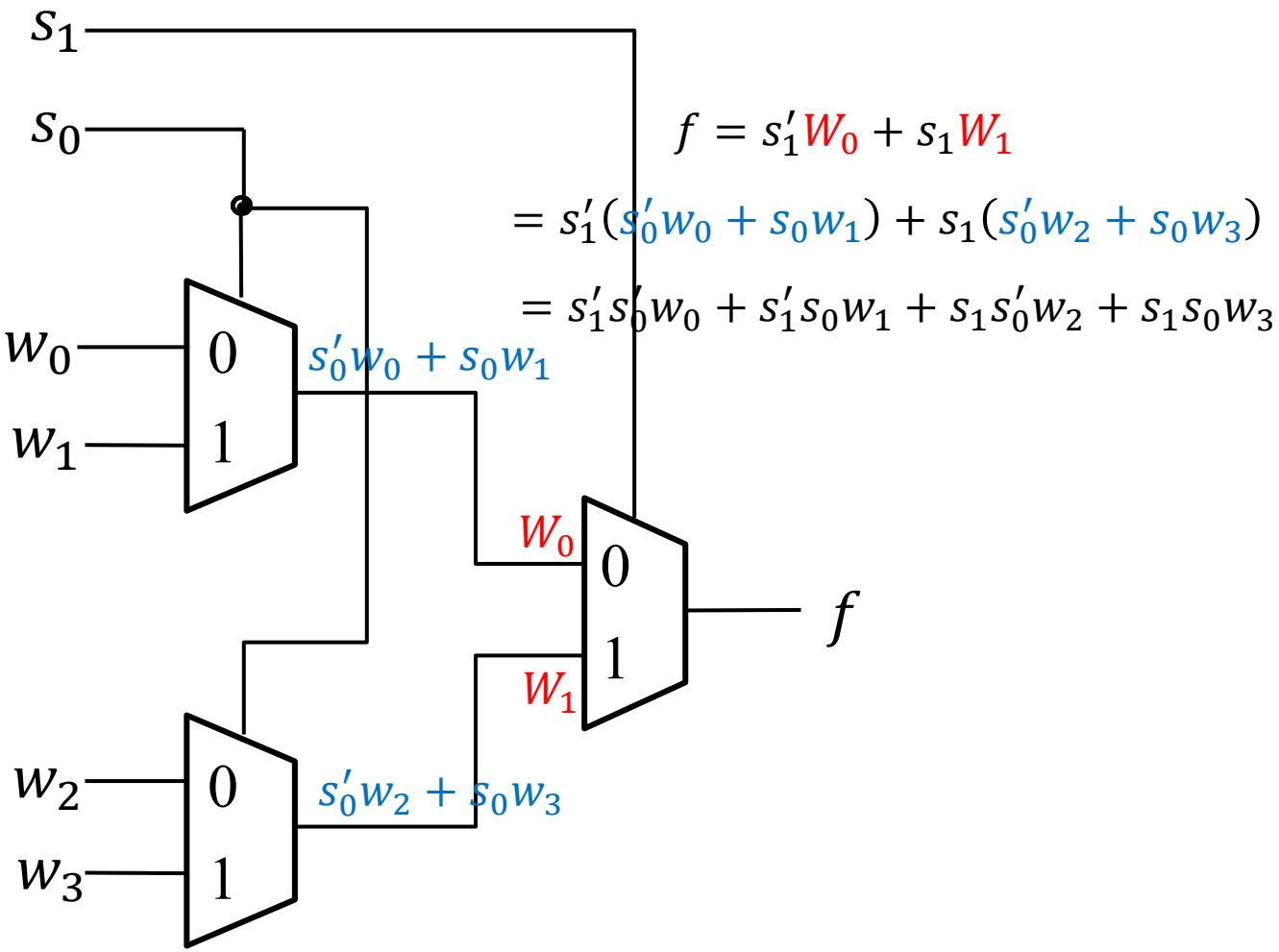


Figure 6.3 Using 2-to-1 multiplexers to build a 4-to-1 multiplexer

16-to-1 MUX using 4-to-1 MUXs

	s_3	s_2	s_1	s_0	f
00	0	0	0	0	w_0
	0	0	0	1	w_1
	0	0	1	0	w_2
	0	0	1	1	w_3
01	0	1	0	0	w_4
	0	1	0	1	w_5
	0	1	1	0	w_6
	0	1	1	1	w_7
10	1	0	0	0	w_8
	1	0	0	1	w_9
	1	0	1	0	w_{10}
	1	0	1	1	w_{11}
11	1	1	0	0	w_{12}
	1	1	0	1	w_{13}
	1	1	1	0	w_{14}
	1	1	1	1	w_{15}

4-to-1 MUX

Truth Table

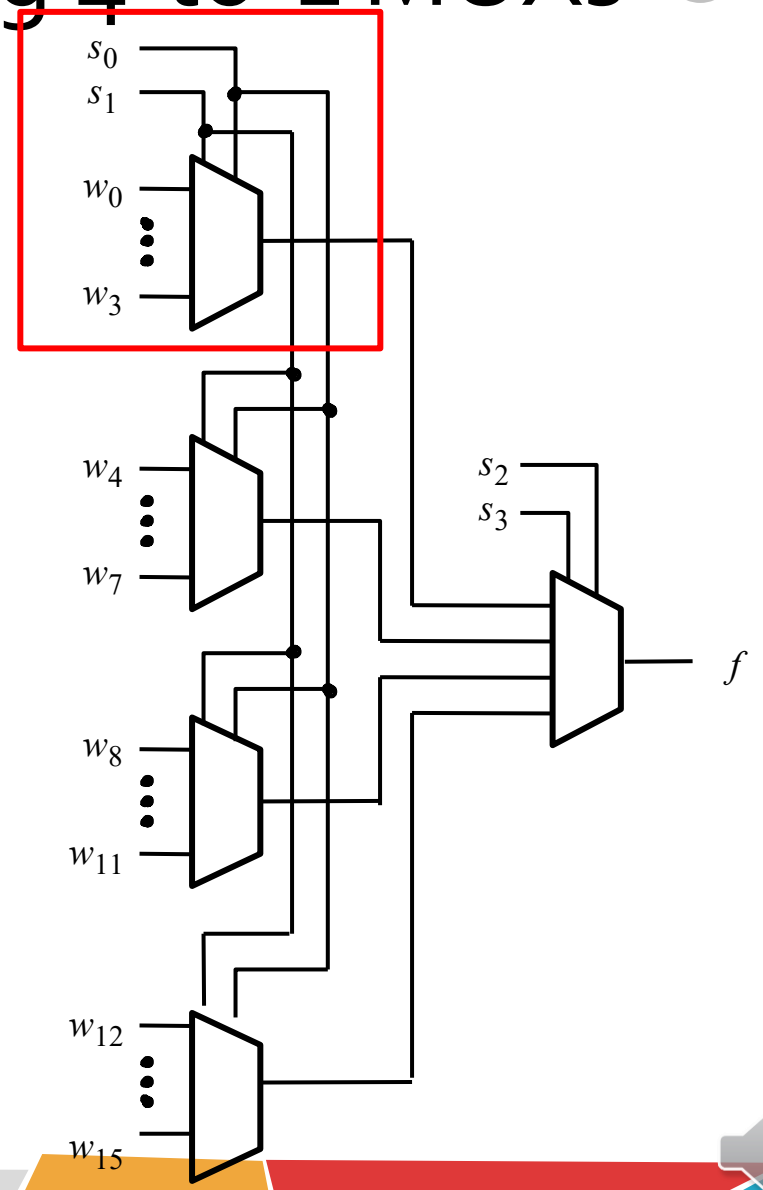
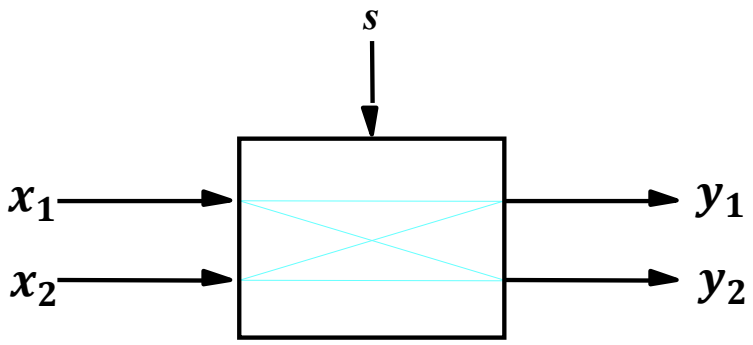
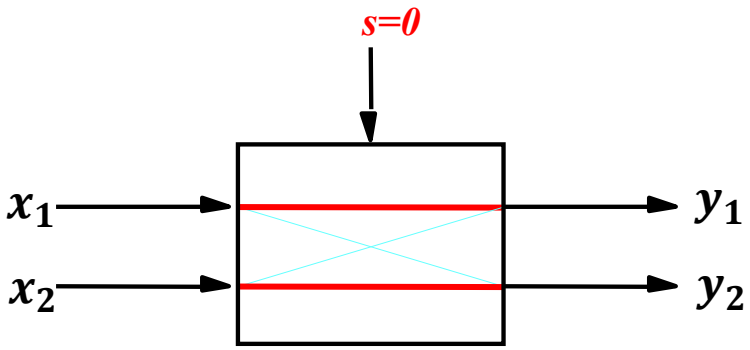


Figure 6.4 A 16-to-1 multiplexer

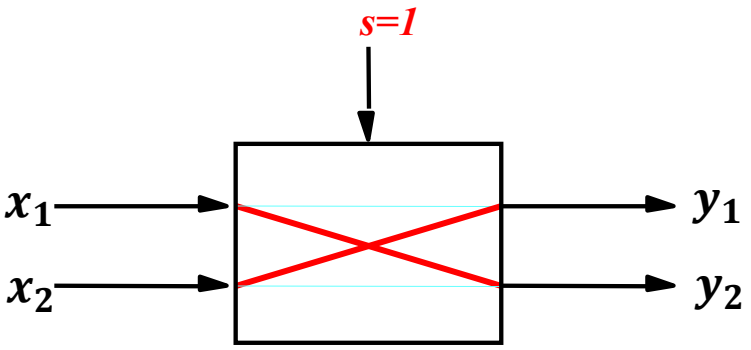
Example 6.1



(a) A 2x2 crossbar switch



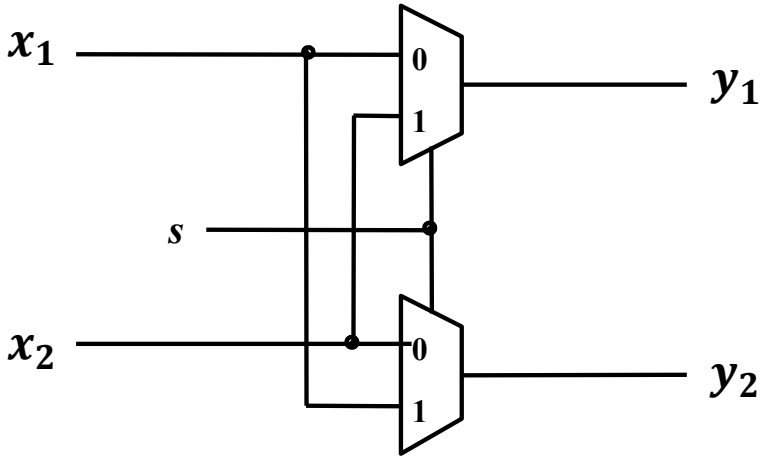
(a) A 2x2 crossbar switch

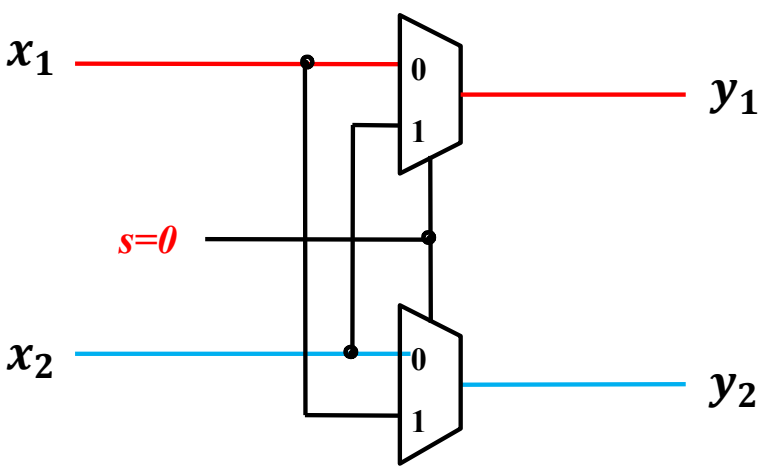


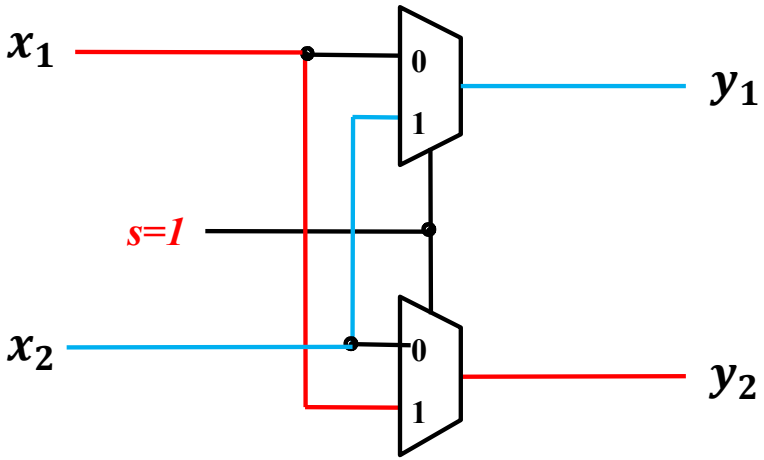
(a) A 2x2 crossbar switch

Figure 6.5 A practical application of multiplexers

Example 6.1



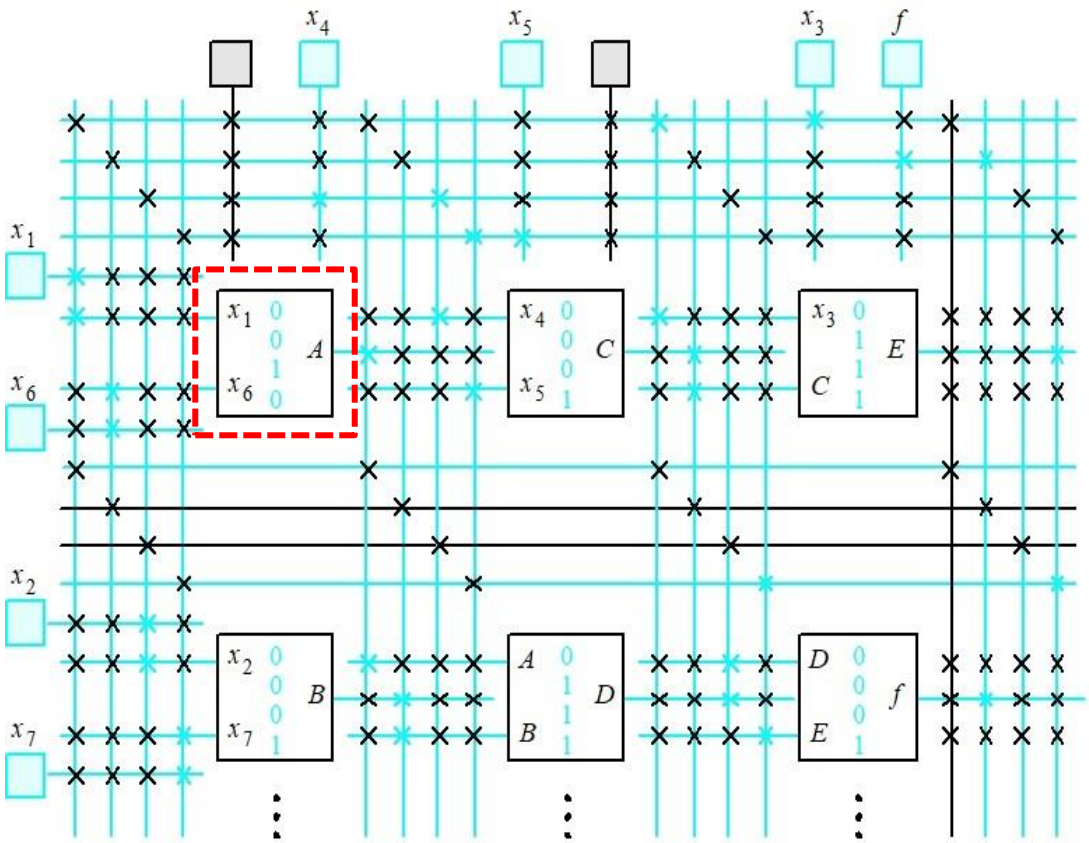




(b) Implementation using multiplexers

Figure 6.5 A practical application of multiplexers

Implementing programmable switches in an FPGA



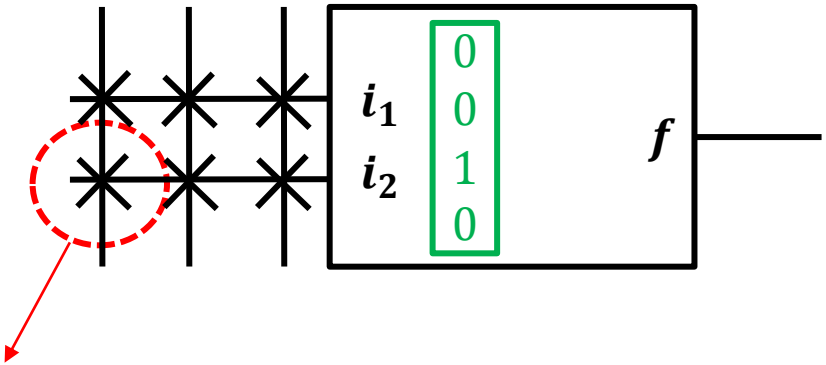
4-to-1 Multiplexer, LUT (Look Up Table), Logic Cell

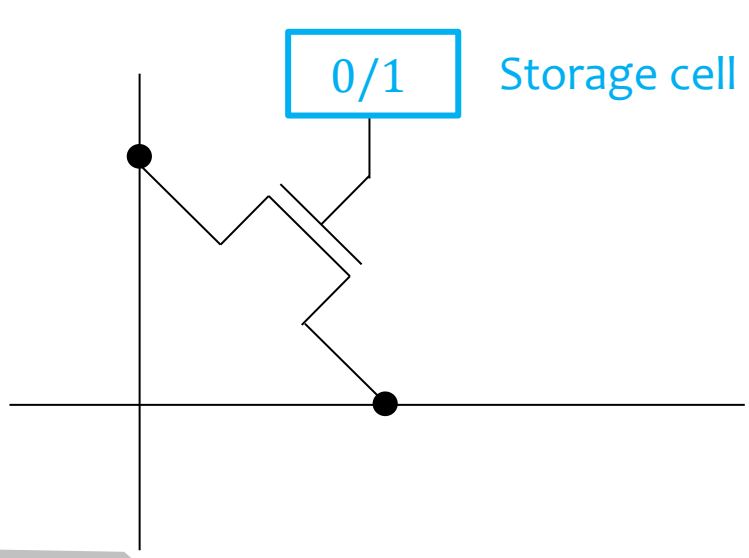
× : disconnected switch

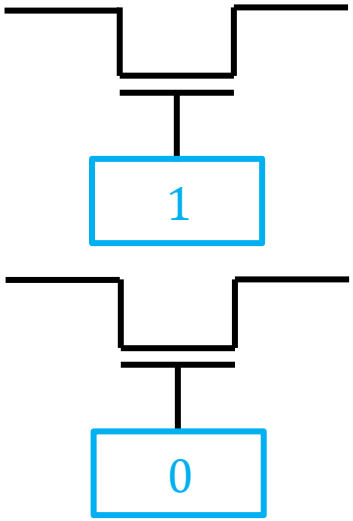
× : connected switch

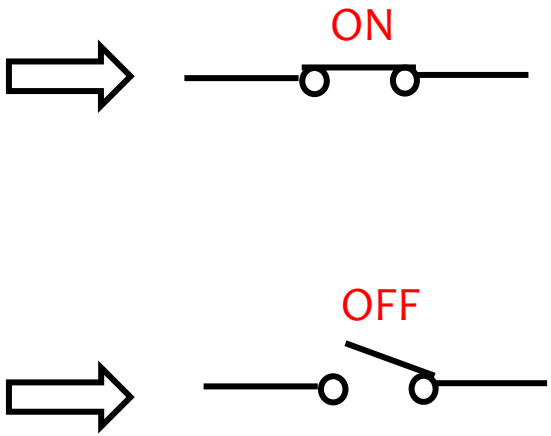
Figure 4.19. Implementation in an FPGA.

Implementing programmable switches in an FPGA



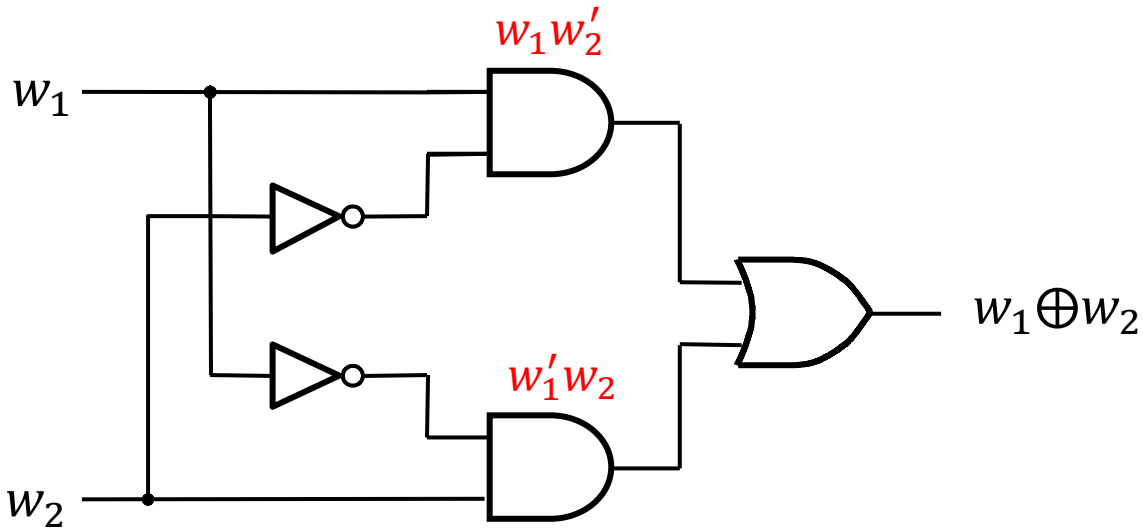






Synthesis of a logic function using MUXs

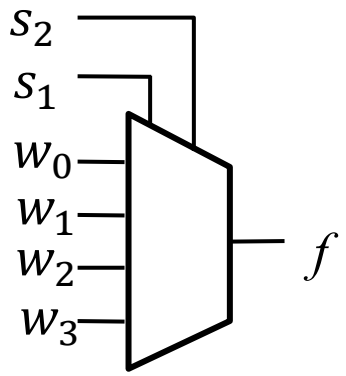
w_1	w_2	f	
0	0	0	
0	1	1	$w_1'w_2$
1	0	1	w_1w_2'
1	1	0	



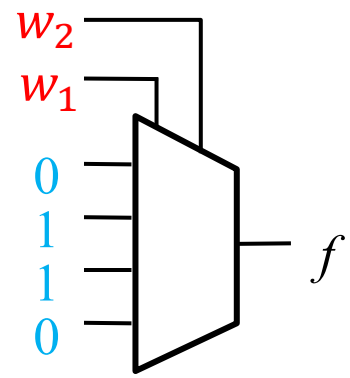
Sum-of-products implementation

Synthesis of a logic function using MUXs

s_1	s_2	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3



w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0



(a) Implementation using a 4-to-1 multiplexer

Figure 6.7 Synthesis of a logic function using multiplexers

Synthesis of a logic function using MUXs

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

$$f(w_1, w_2) = w_1' w_2 + w_1 w_2'$$

w_1	w_2	f
0	0	$f(0,0) = 0$
0	1	$f(0,1) = 1$
1	0	$f(1,0) = 1$
1	1	$f(1,1) = 0$

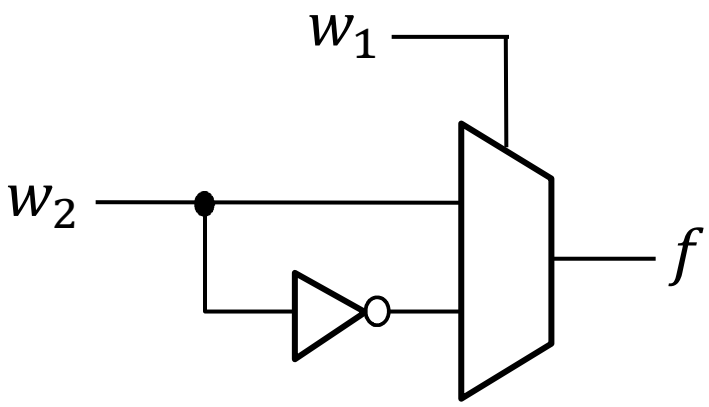
w_1	f
0	$f(0, w_2) = w_2$
1	$f(1, w_2) = w_2'$

Synthesis of a logic function using MUXs

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

w_1	f
0	w_2
1	w_2'

(b) Modified truth table



(c) Circuit

Figure 6.7 Synthesis of a logic function using multiplexers

Implementation of the three-input majority function

Using a 4-to-1 multiplexer

w_1	w_2	w_3	f	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$w_1'w_2w_3$
1	0	0	0	
1	0	1	1	$w_1w_2'w_3$
1	1	0	1	$w_1w_2w_3'$
1	1	1	1	$w_1w_2w_3$

$$f(w_1, w_2, w_3) = w_1'w_2w_3 + w_1w_2'w_3 + w_1w_2w_3' + w_1w_2w_3$$

w_1	w_2	f
0	0	$f(0,0,w_3) = 0$
0	1	$f(0,1,w_3) = w_3$
1	0	$f(1,0,w_3) = w_3$
1	1	$f(1,1,w_3) = 1$

(a) Modified truth table

Implementation of the three-input majority function

Using a 4-to-1 multiplexer

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

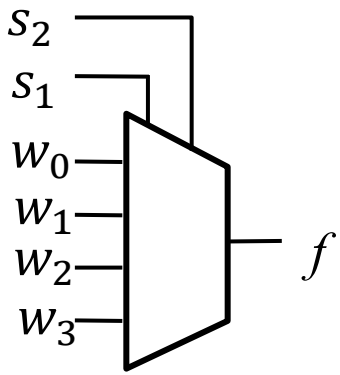
0	0	1	1
0	1	0	1
0	w_3	w_3'	1

(a) Modified truth table

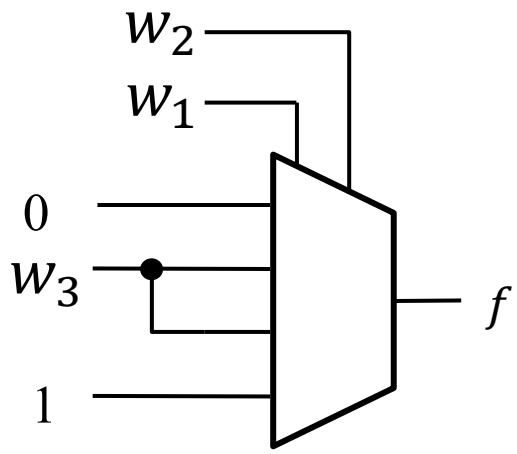
Implementation of the three-input majority function

Using a 4-to-1 multiplexer

s_1	s_2	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3



w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1



(b) Circuit

Three-input XOR implemented with 2-to-1 MUXs

w_1	w_2	w_3	f	
0	0	0	0	$w_2 \oplus w_3$
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	$(w_2 \oplus w_3)'$
1	0	1	0	
1	1	0	0	
1	1	1	1	

$$f(w_1, w_2, w_3) = w_1'w_2'w_3 + w_1'w_2w_3' + w_1w_2'w_3' + w_1w_2w_3$$

w_1	f
0	$f(0, w_2, w_3) = w_2'w_3 + w_2w_3'$
1	$f(1, w_2, w_3) = w_2'w_3' + w_2w_3$

$$f(0, w_2, w_3) = w_2'w_3 + w_2w_3' = w_2 \oplus w_3$$

$$f(1, w_2, w_3) = w_2'w_3' + w_2w_3 = (w_2 \oplus w_3)'$$

(a) Modified Truth table

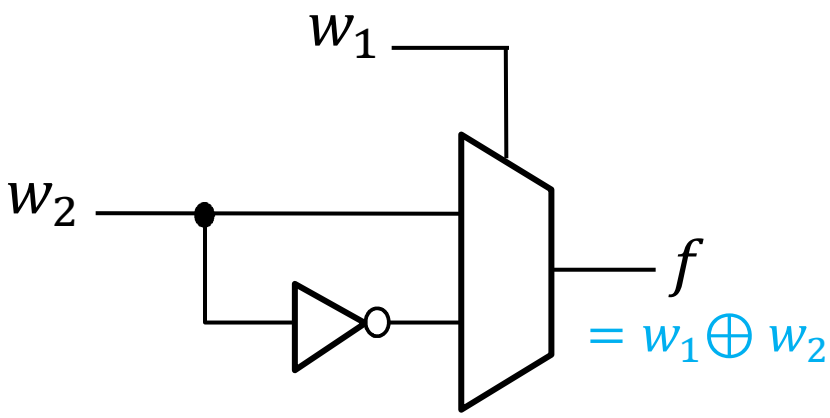
Synthesis of a logic function using MUXs

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

w_1	f
0	w_2
1	w_2'

$w_1 \oplus w_2$

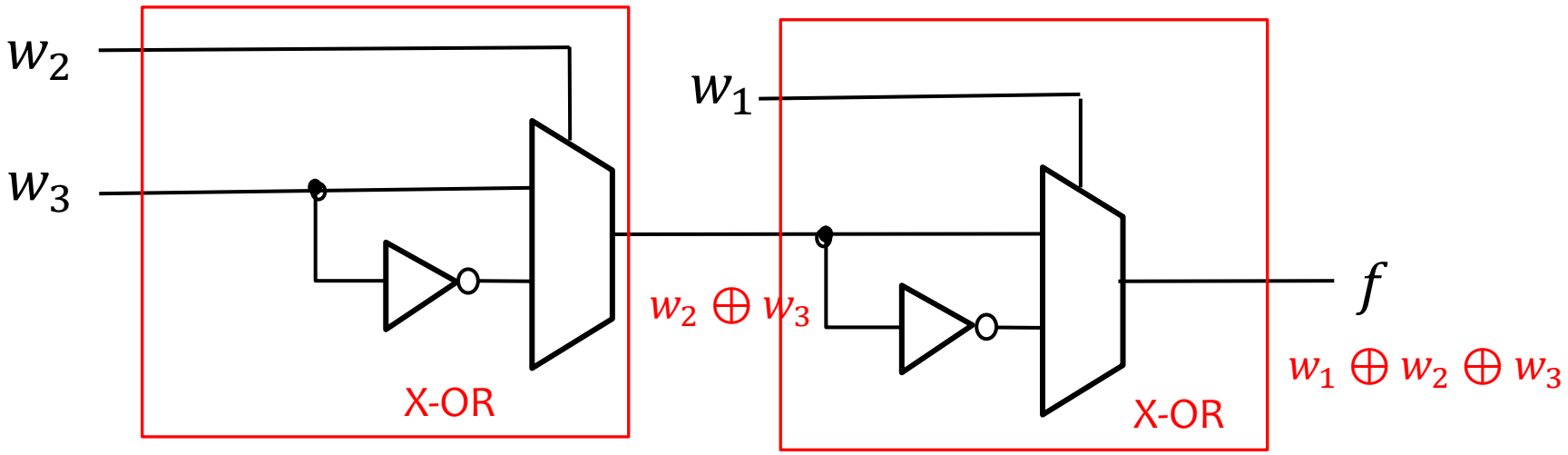
(b) Modified truth table



(c) Circuit

Figure 6.7 Synthesis of a logic function using multiplexers

Three-input XOR implemented with 2-to-1 MUXs



(b) Circuit

Three-input XOR function implemented with a 4-to-1 MUX

w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	w_3'
0	1	1	0	
1	0	0	1	w_3'
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	

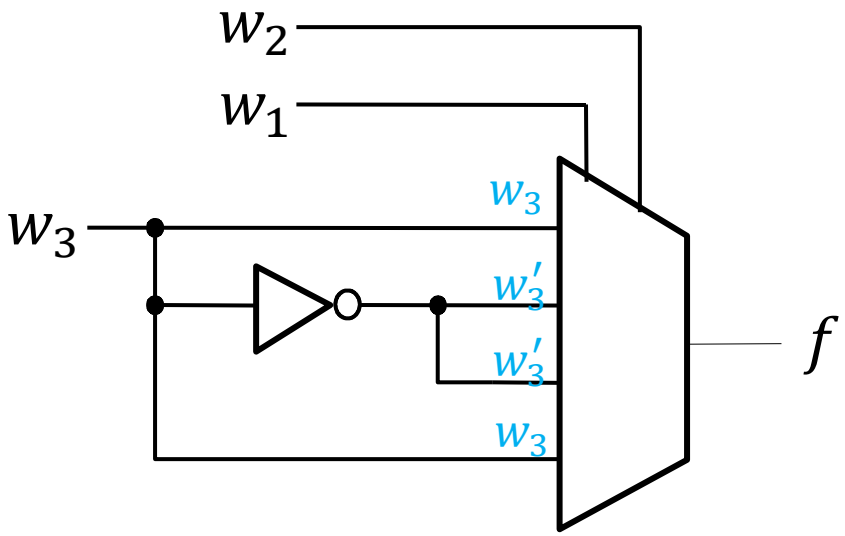
(a) Truth table

w_1	w_2	f
0	0	$f(0,0,w_3) = w_3$
0	1	$f(0,1,w_3) = w_3'$
1	0	$f(1,0,w_3) = w_3'$
1	1	$f(1,1,w_3) = w_3$

Three-input XOR function implemented with a 4-to-1 MUX

w_1	w_2	f
0	0	$f(0,0,w_3) = w_3$
0	1	$f(0,1,w_3) = w_3'$
1	0	$f(1,0,w_3) = w_3'$
1	1	$f(1,1,w_3) = w_3$

(a) Truth table



(b) Circuit

The three-input majority function implemented using a 2-to-1 multiplexer

$$f(w_1, w_2, w_3) = \sum(3, 5, 6, 7)$$

$$= w_1' w_2 w_3 + w_1 w_2' w_3 + w_1 w_2 w_3' + w_1 w_2 w_3$$

$$f(w_1, w_2, w_3) = w_1'(w_2 w_3) + w_1(w_2' w_3 + w_2 w_3' + w_2 w_3)$$

$$w_2' w_3 + w_2 w_3' + w_2 w_3$$
$$= w_2' w_3 + w_2(w_3' + w_3)$$
$$= w_2 + w_2' w_3$$
$$= (w_2 + w_2')(w_2 + w_3)$$
$$= w_2 + w_3$$

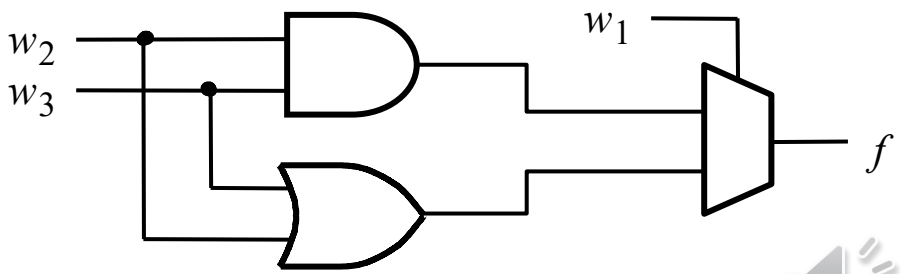
$$f(w_1, w_2, w_3) = w_1'(w_2 w_3) + w_1(w_2 + w_3)$$

$$f(0, w_2, w_3) = w_2 w_3$$
$$f(1, w_2, w_3) = w_2 + w_3$$

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f	
0	$w_2 w_3$	$f(0, w_2, w_3)$
1	$w_2 + w_3$	$f(1, w_2, w_3)$

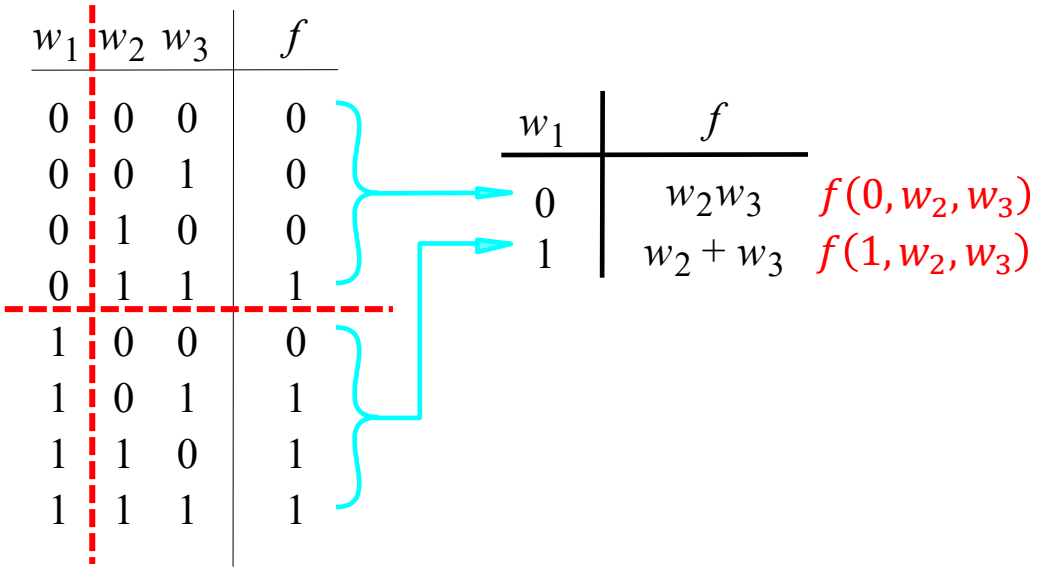
(b) Truth table



(b) Circuit

Shannon's Expansion Theorem

$$f(w_1, w_2, \cdots, w_n) = w_1' f(0, w_2, \cdots, w_n) + w_1 f(1, w_2, \cdots, w_n)$$



w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f	
0	$w_2 w_3$	$f(0, w_2, w_3)$
1	$w_2 + w_3$	$f(1, w_2, w_3)$

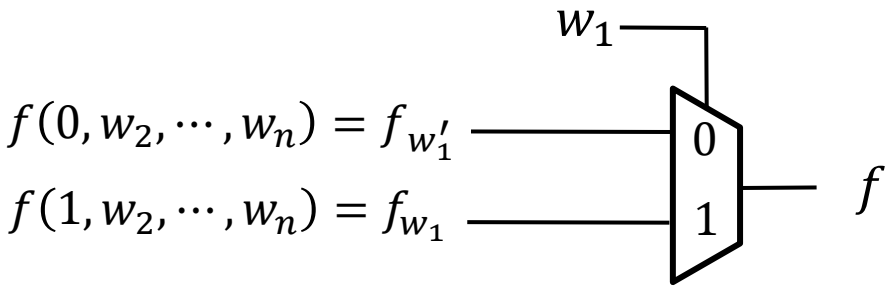
(b) Truth table

$f(0, w_2, \cdots, w_n) = f_{w_1'}$ Cofactor of f with respect to w_1'

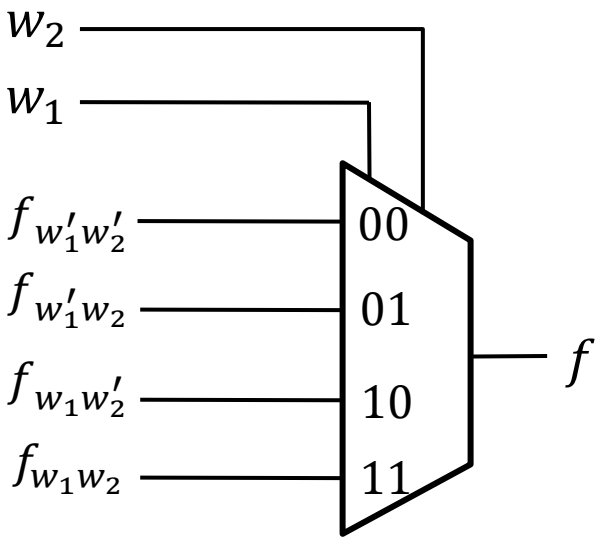
Shannon's Expansion Theorem

$$f(w_1, w_2, \cdots, w_n) = w_1' f_{w_1'} + w_1 f_{w_1}$$ Using 2-to-1 MUX

Using 4-to-1 MUX $= w_1' w_2' f_{w_1' w_2'} + w_1' w_2 f_{w_1' w_2} + w_1 w_2' f_{w_1 w_2'} + w_1 w_2 f_{w_1 w_2}$



Using 2-to-1 MUX

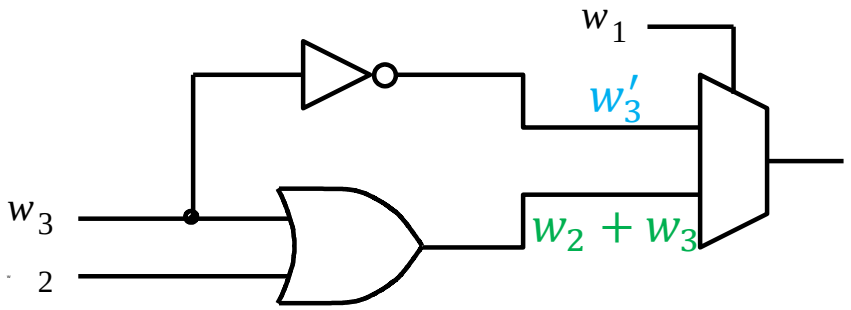


Using 4-to-1 MUX

The circuits synthesized in Example 6.6

$$f(w_1, w_2, w_3) = w_1'w_3' + w_1w_2 + w_1w_3$$

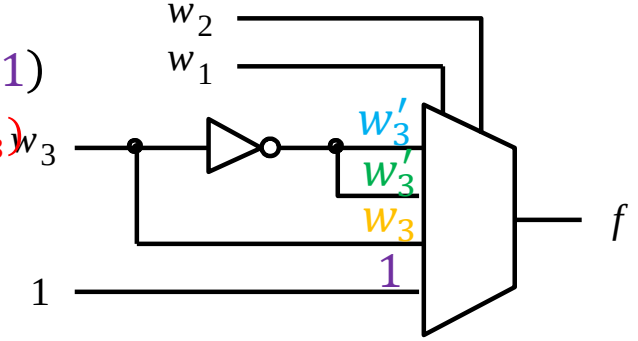
$$f = w_1' f_{w_1'} + w_1 f_{w_1}$$
$$= w_1'(w_3') + w_1(w_2 + w_3)$$
$f(0, w_2, w_3)$ $f(1, w_2, w_3)$



(a) Using a 2-to-1 multiplexer

$$f = w_1'w_2' f_{w_1'w_2'} + w_1'w_2 f_{w_1'w_2} + w_1w_2' f_{w_1w_2'} + w_1w_2 f_{w_1w_2}$$

$$f = w_1'w_2'(w_3') + w_1'w_2(w_3') + w_1w_2'(w_3) + w_1w_2(1)$$
$f(0,0,w_3)$ $f(0,1,w_3)$ $f(1,0,w_3)$ $f(1,1,w_3)$



(b) Using a 4-to-1 multiplexer

The circuit synthesized in Example 6.7

Three input majority function

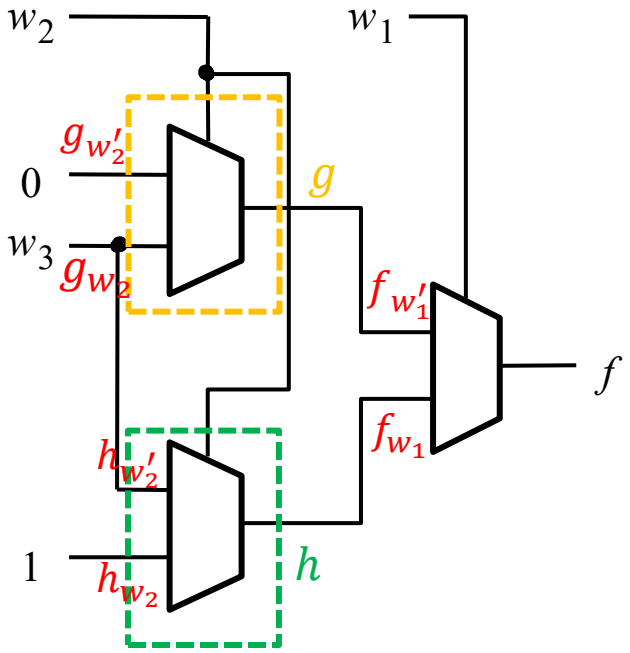
$$f(w_1, w_2, w_3) = w_1w_2 + w_1w_3 + w_2w_3$$

$$\begin{aligned} f &= w_1' f_{w_1'} + w_1 f_{w_1} \\ &= w_1'(w_2w_3) + w_1(w_2 + w_3 + w_2w_3) \\ &= w_1'(w_2w_3) + w_1(w_2 + w_3) \end{aligned}$$

$g(w_2, w_3) = w_2w_3$ $\qquad$ $h(w_2, w_3) = w_2 + w_3$

$$\begin{aligned} g &= w_2' g_{w_2'} + w_2 g_{w_2} \\ &= w_2' g(0, w_3) + w_2 g(1, w_3) \\ &= w_2'(0) + w_2(w_3) \end{aligned}$$

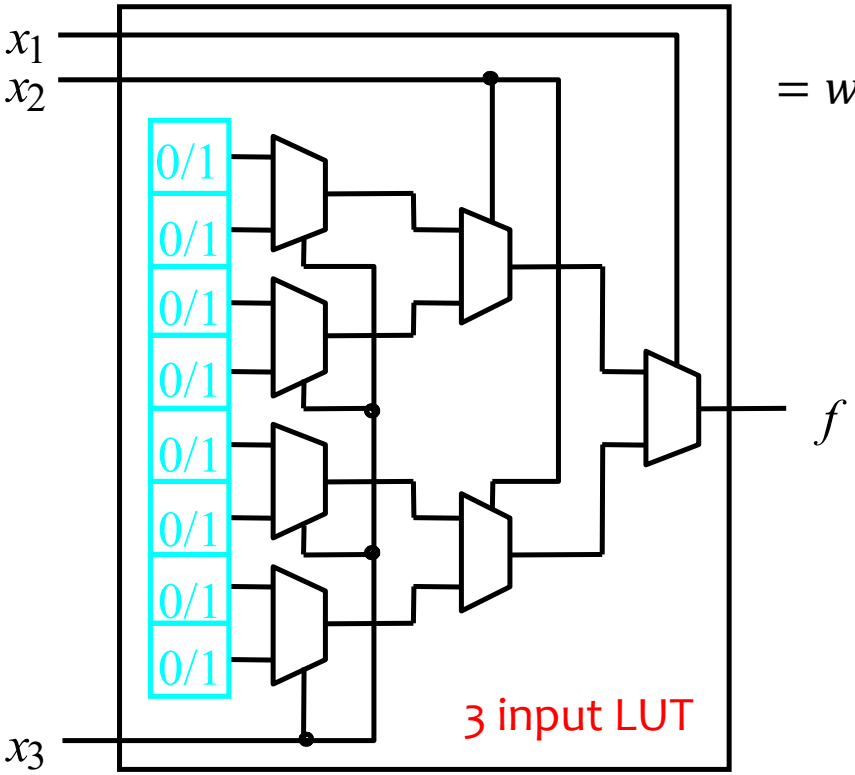
$$\begin{aligned} h &= w_2' h_{w_2'} + w_2 h_{w_2} \\ &= w_2' h(0, w_3) + w_2 h(1, w_3) \\ &= w_2'(w_3) + w_2(1) \end{aligned}$$



Circuits synthesized in Example 6.8

$$f(w_1, w_2, w_3, w_4) = w_2'w_3 + w_1'w_2w_3' + w_2w_3'w_4 + w_1w_2'w_4'$$

$$f = w_1' f_{w_1'} + w_1 f_{w_1} \qquad w_2w_3' + w_2w_3'w_4 = w_2w_3'(1 + w_4)$$
$$= w_1'(w_2'w_3 + w_2w_3' + w_2w_3'w_4)$$
$$+ w_1(w_2'w_3 + w_2w_3'w_4 + w_2'w_4')$$
$$= w_1'(w_2'w_3 + w_2w_3') + w_1(w_2'w_3 + w_2w_3'w_4 + w_2'w_4')$$



$$f_{w_1'} = w_2'w_3 + w_2w_3'$$
$$f_{w_1} = w_2'w_3 + w_2w_3'w_4 + w_2'w_4'$$

$$f_{w_1'} = f(0, w_2, w_3, w_4) = f(0, w_2, w_3, 0)$$

$$f_{w_1} = f(1, w_2, w_3, w_4)$$

$$f = h\left[w_1, f_{w_1'}, f_{w_1}\right]$$

Because it stores a truth table, a 3-LUT can realize any logic function of 3 variables.

Circuits synthesized in Example 6.8

$$f(w_1, w_2, w_3, w_4) = w_2' w_3 + w_1' w_2 w_3' + w_2 w_3' w_4 + w_1 w_2' w_4'$$

$$\begin{aligned} f &= w_2' f_{w_2'} + w_2 f_{w_2} \\ &= w_2'(w_3 + w_1 w_4') + w_2(w_1' w_3' + w_3' w_4) \end{aligned}$$

$$\begin{aligned} f_{w_2'} &= w_3 + w_1 w_4' = (w_1 + w_3)(w_3 + w_4') \\ f_{w_2} &= w_1' w_3' + w_3' w_4 \end{aligned}$$

$x + yz = (x + y)(x + z)$

$f'_{w_2'} = f_{w_2}$

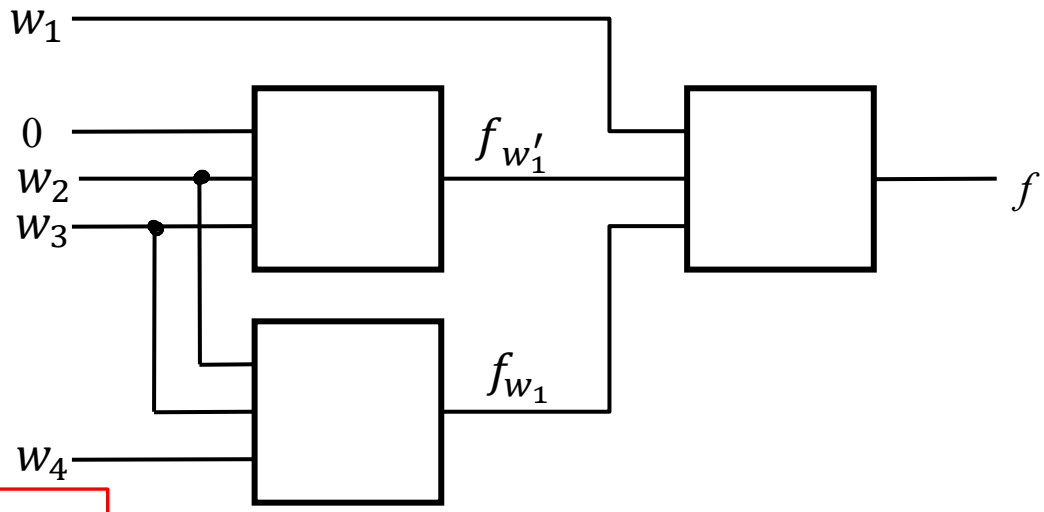
$$\begin{aligned} f'_{w_2'} &= \big((w_1 + w_3)(w_3 + w_4')\big)' \\ &= (w_1 + w_3)' + (w_3 + w_4')' \\ &= w_1' w_3' + w_3' w_4 = f_{w_2} \end{aligned}$$

$$f(w_1, w_2, w_3, w_4) = h\left(w_2, f_{w_2'}, f_{w_2}\right) = h\left(w_2, f_{w_2'}, 0\right)$$

Circuits synthesized in Example 6.8

$$f(w_1, w_2, w_3, w_4) = w_2'w_3 + w_1'w_2w_3' + w_2w_3'w_4 + w_1w_2'w_4'$$

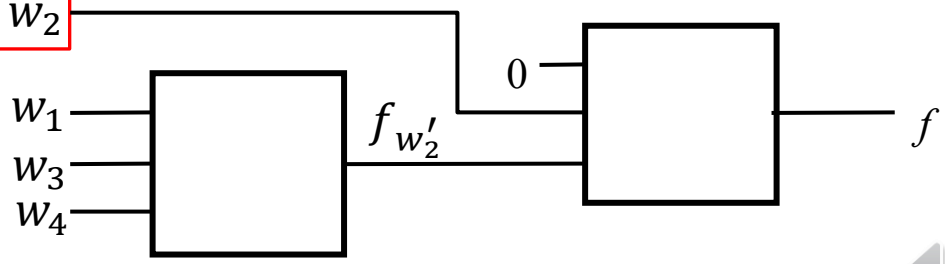
$$f_{w_1'} = w_2'w_3 + w_2w_3'$$
$$f_{w_1} = w_2'w_3 + w_2w_3'w_4 + w_2'w_4'$$



(a) Using three 3-LUTs

$$f_{w_2'} = w_3 + w_1w_4' = (w_1 + w_3)(w_3 + w_4')$$
$$f_{w_2} = w_1'w_3' + w_3'w_4$$
$$f_{w_2'}' = f_{w_2}$$

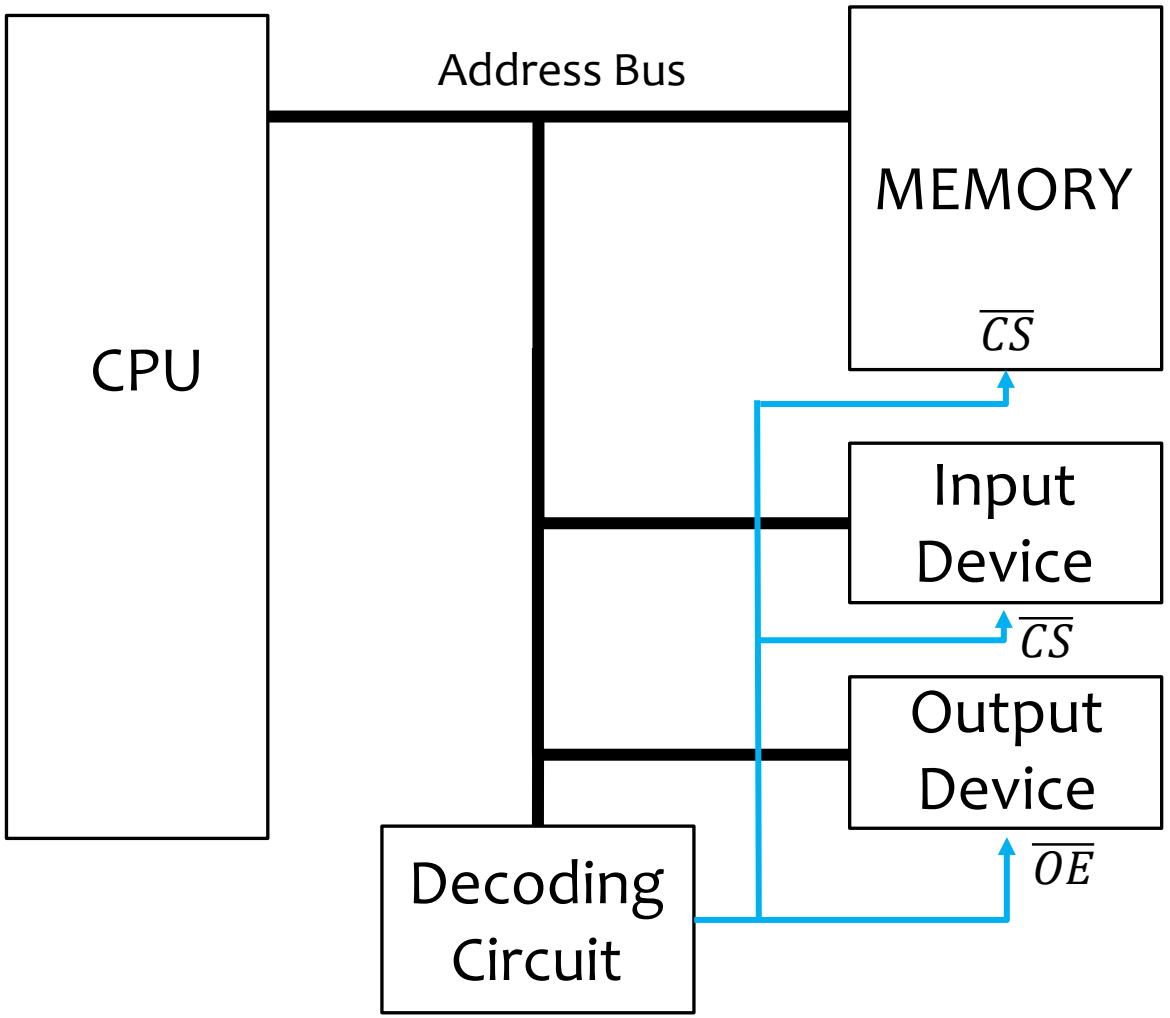
$$f = h\left(w_2, f_{w_2'}, 0\right)$$



(b) Using two 3-LUTs

DECODERS

Application of Decoder



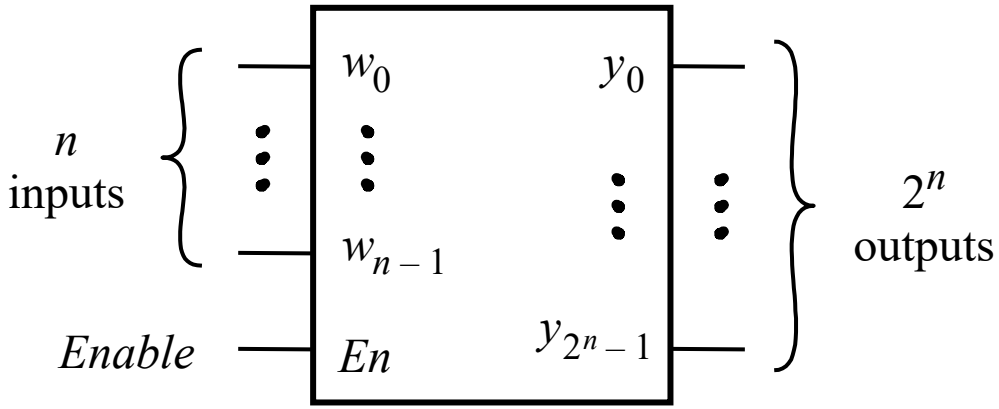
$\overline{CS}$: Chip Select

$\overline{OE}$: Output Enable

$\overline{CS}$: Active Low

CS: Active High

An n-to-2^n binary decoder

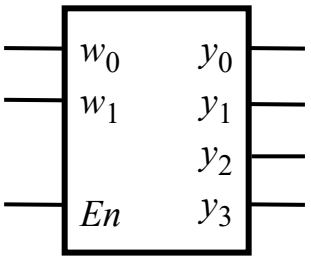


	w_0	w_1	w_2	$\cdots$	w_{n-1}	w_{n-1}	y_0	y_1	y_2	$\cdots$	y_{2^n-1}
	0	0	0	$\cdots$	0	0	1	0	0	$\cdots$	0
	0	0	0	$\cdots$	0	1	0	1	0	$\cdots$	0
2^n *lines*	0	0	0	$\cdots$	1	0	0	0	1	$\cdots$	0
				$\vdots$						$\vdots$	
	1	1	1	$\cdots$	1	1	0	0	0	$\cdots$	1

2-to-4 decoder

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table



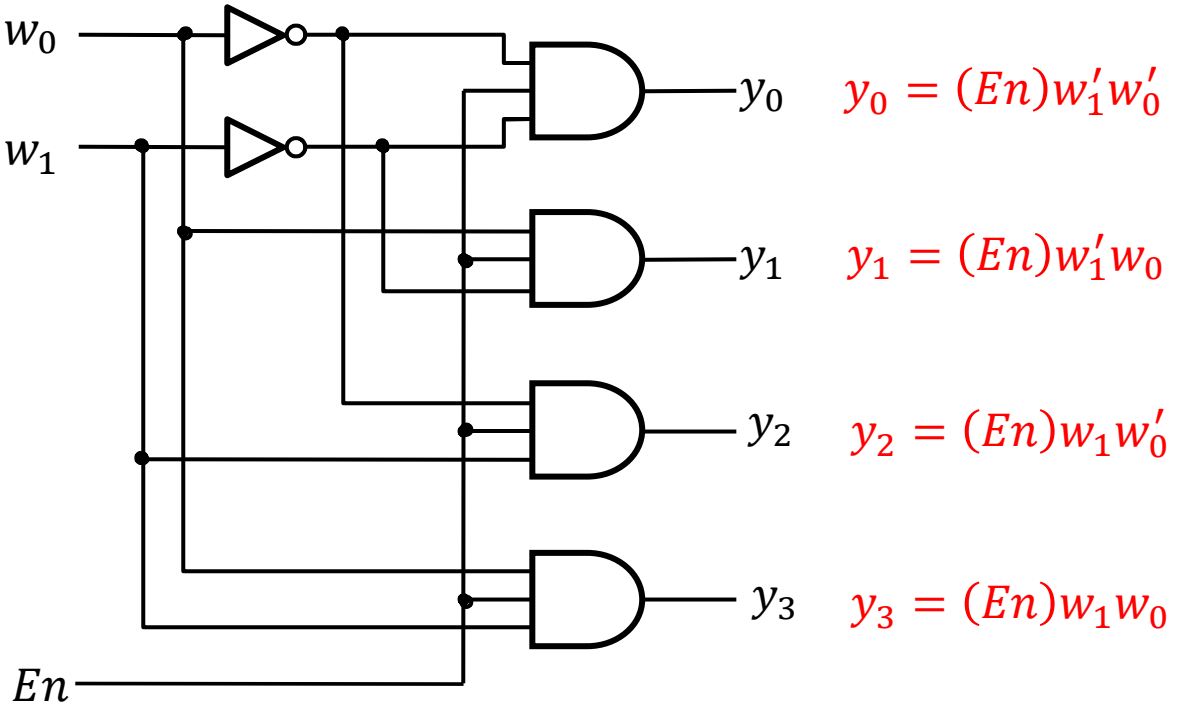
(b) Graphical symbol

$$y_0 = (En)w_1'w_0'$$
$$y_1 = (En)w_1'w_0$$
$$y_2 = (En)w_1w_0'$$
$$y_3 = (En)w_1w_0$$

Figure 6.16. A 2-to-4 decoder.

2-to-4 decoder

SOP Implementation

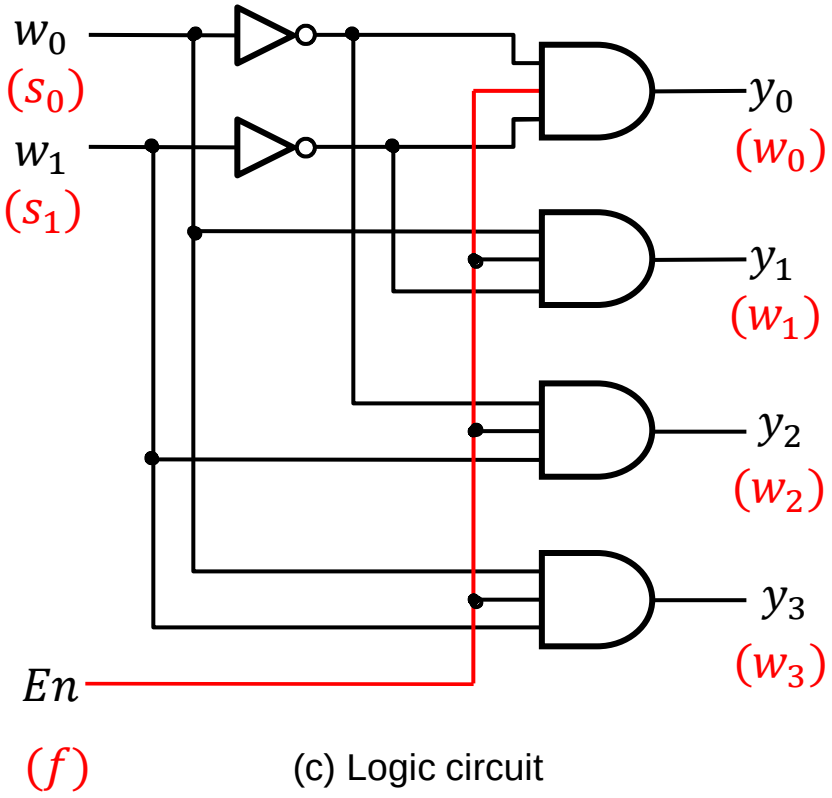


(c) Logic circuit

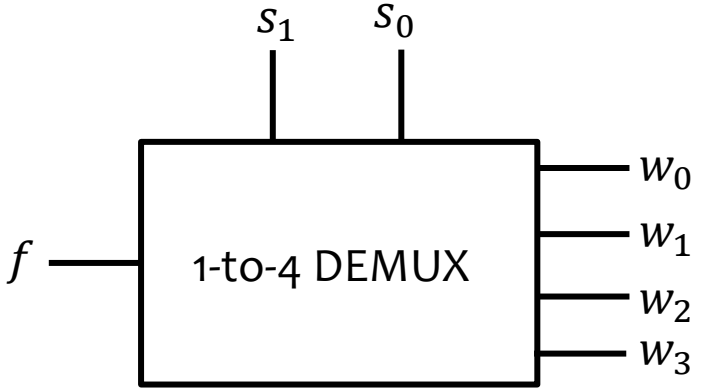
Figure 6.16. A 2-to-4 decoder.

1-to-4 demultiplexer

SOP Implementation



(c) Logic circuit



3-to-8 decoder

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

(a) Truth table

$$y_0 = (En)w_1'w_0'$$
$$y_1 = (En)w_1'w_0$$
$$y_2 = (En)w_1w_0'$$
$$y_3 = (En)w_1w_0$$

En	w_2	w_1	w_0	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	
1	0	0	0	1	0	0	0	0	0	0	0	$y_0 = (Enw_2')w_1'w_0'$
1	0	0	1	0	1	0	0	0	0	0	0	$y_1 = (Enw_2')w_1'w_0$
1	0	1	0	0	0	1	0	0	0	0	0	$y_2 = (Enw_2')w_1w_0'$
1	0	1	1	0	0	0	1	0	0	0	0	$y_3 = (Enw_2')w_1w_0$
1	1	0	0	0	0	0	0	1	0	0	0	$y_4 = (Enw_2)w_1'w_0'$
1	1	0	1	0	0	0	0	0	1	0	0	$y_5 = (Enw_2)w_1'w_0$
1	1	1	0	0	0	0	0	0	0	1	0	$y_6 = (Enw_2)w_1w_0'$
1	1	1	1	0	0	0	0	0	0	0	1	$y_7 = (Enw_2)w_1w_0$
0	×	×	×	0	0	0	0	0	0	0	0	

3-to-8 decoder

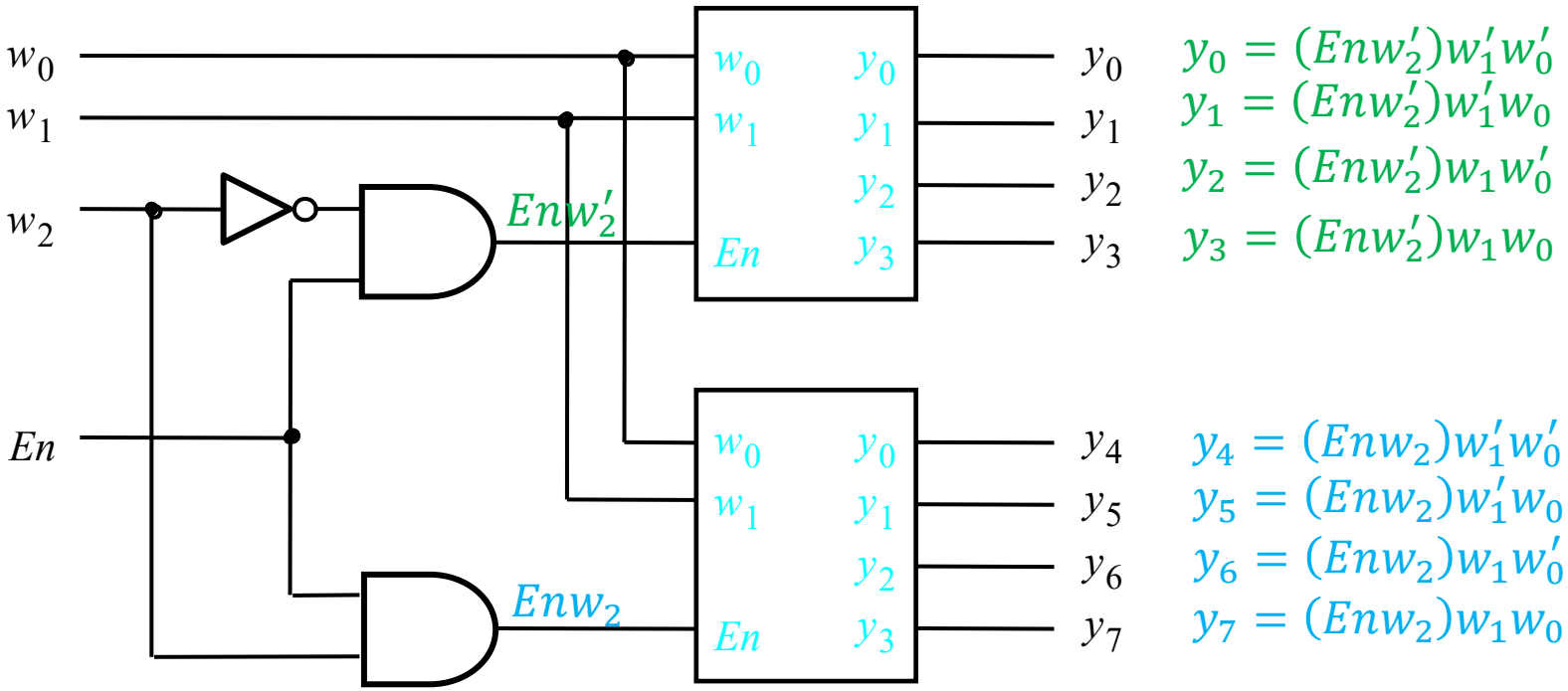


Figure 6.17. A 3-to-8 decoder using two 2-to-4 decoders.

4-to-16 decoder

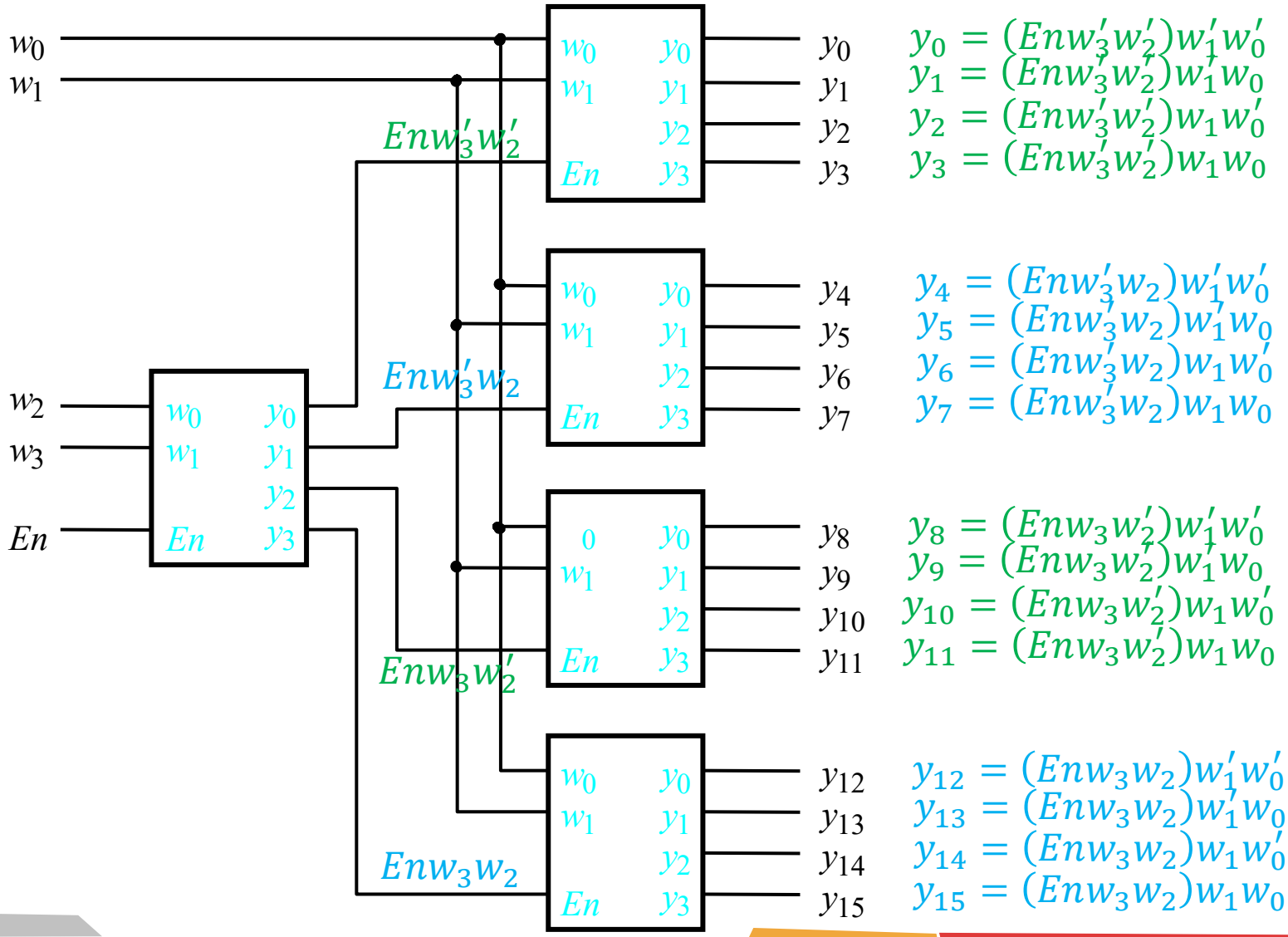


Figure 6.18. A 4-to-16 decoder built using a decoder tree.

4-to-1 multiplexer

$$f = s_1' s_0' w_0 + s_1' s_0 w_1 + s_1 s_0' w_2 + s_1 s_0 w_3$$

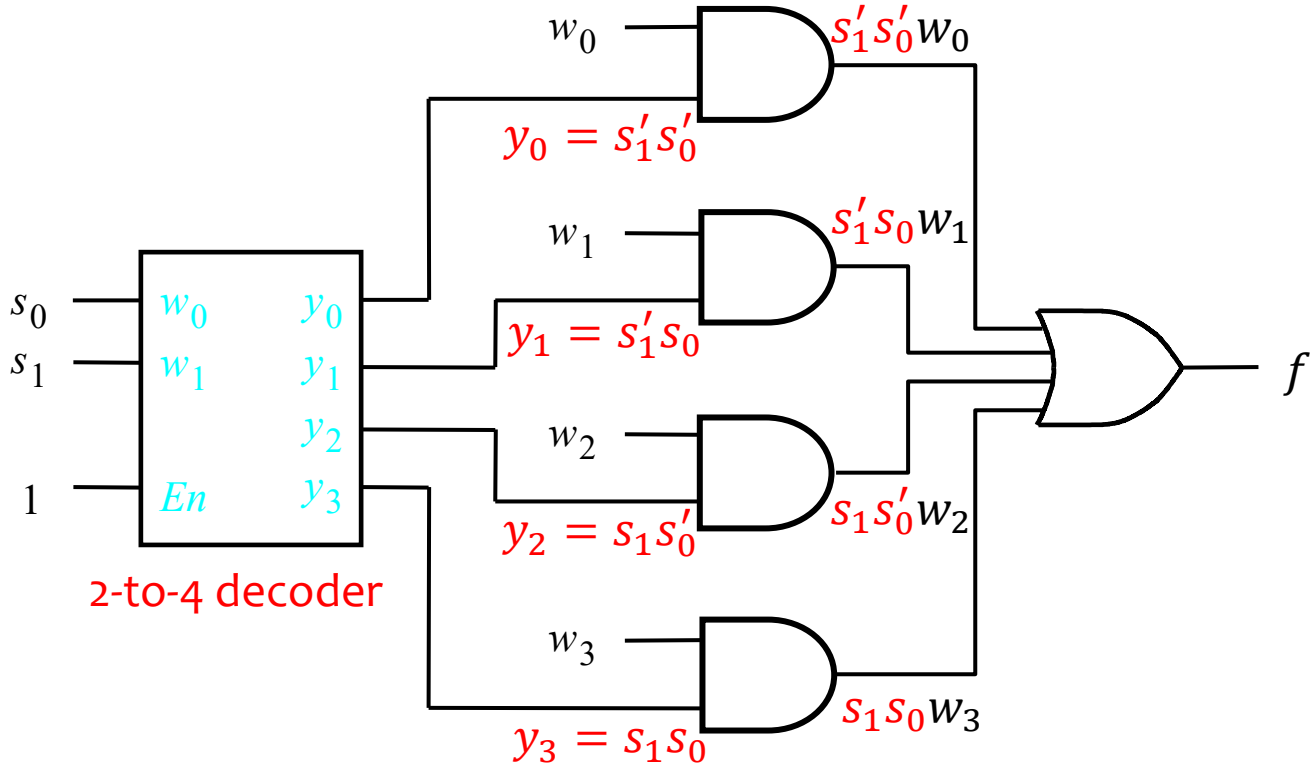
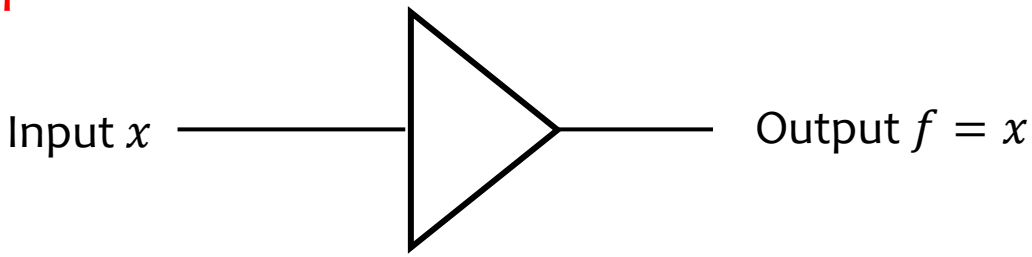


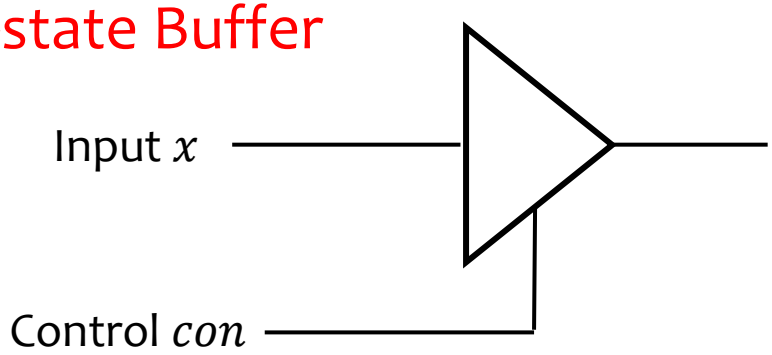
Figure 6.19. A 4-to-1 multiplexer built using a decoder.

Tri-state Buffer

Buffer



Tri-state Buffer



$$f = \begin{cases} x \ \ if \ con = 1 \\ Z \ if \ con = 0 \end{cases}$$

Z : high impedance

$f = 0, 1, Z$: tri-state

4-to-1 multiplexer

$$f = s_1' s_0' w_0 + s_1' s_0 w_1 + s_1 s_0' w_2 + s_1 s_0 w_3$$

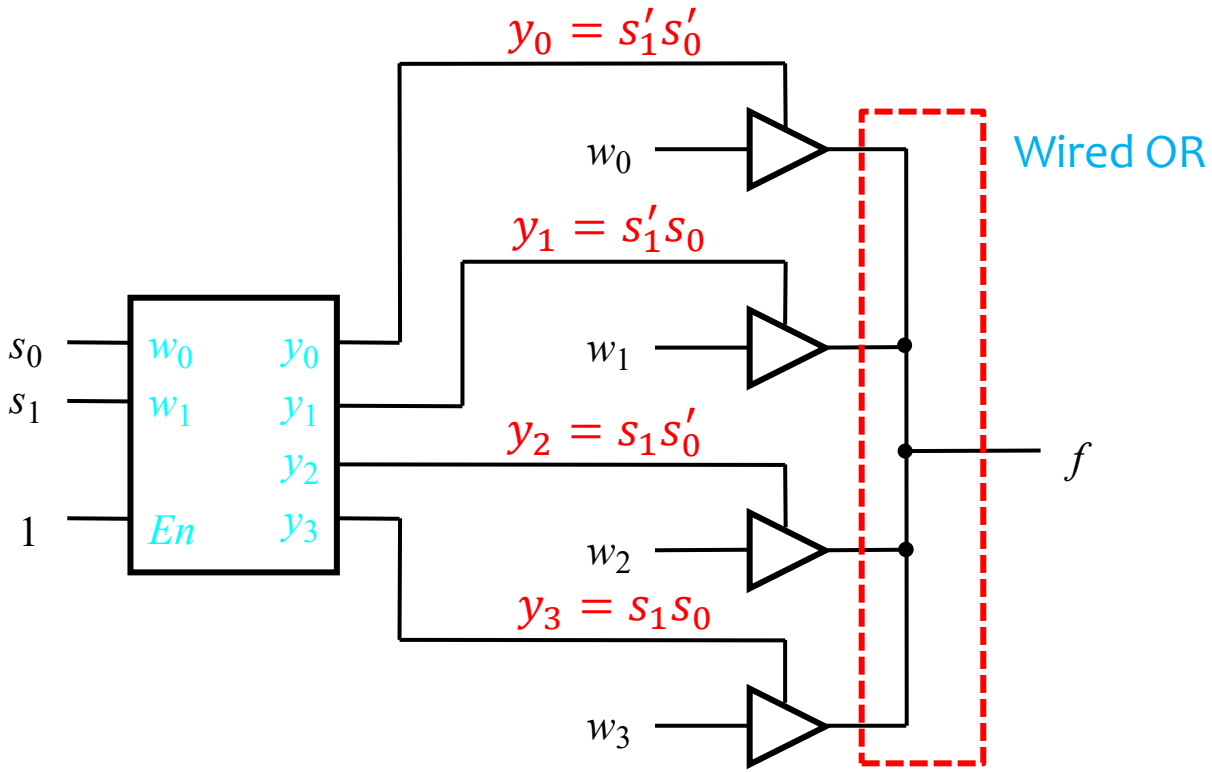


Figure 6.20. A 4-to-1 multiplexer built using a decoder and tri-state buffers.

2^m x n read-only memory (ROM)

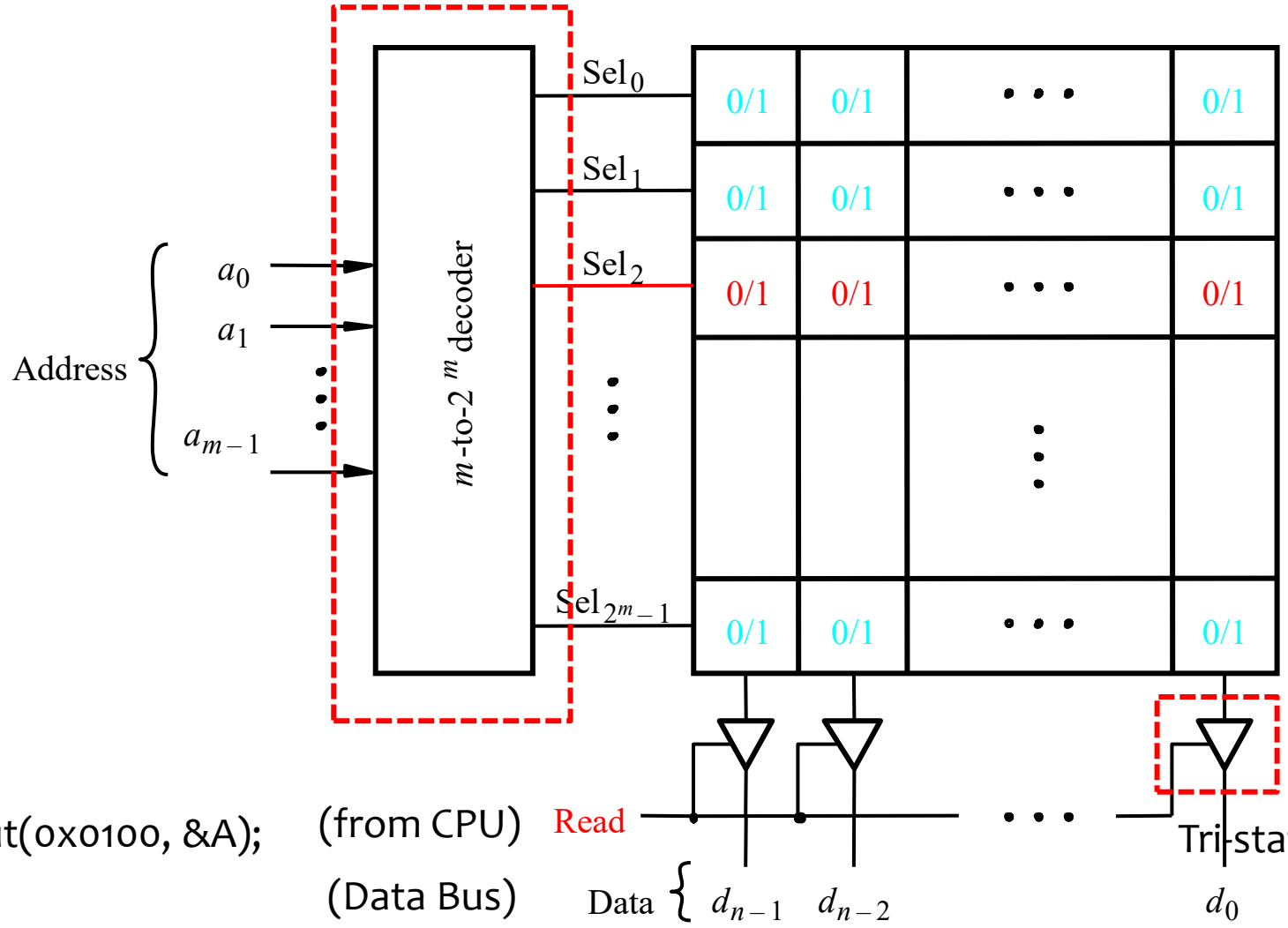


Figure 6.21. A 2^m x n read-only memory (ROM) block.

ENCODERS

2^n–to-n binary encoder

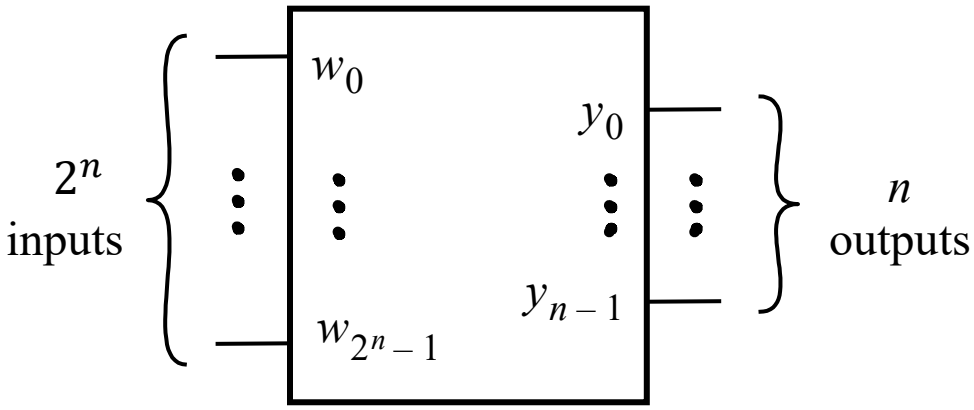
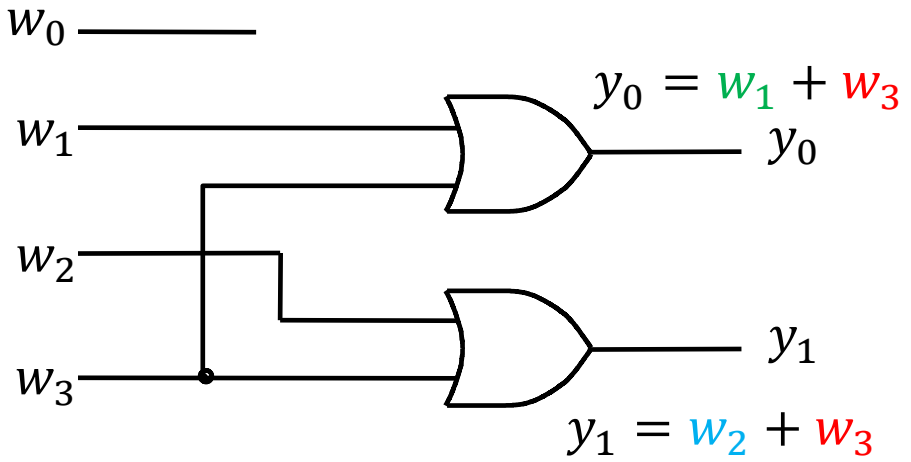


Figure 6.22. A 2^n–to-n binary encoder.

4-to-2 binary encoder

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

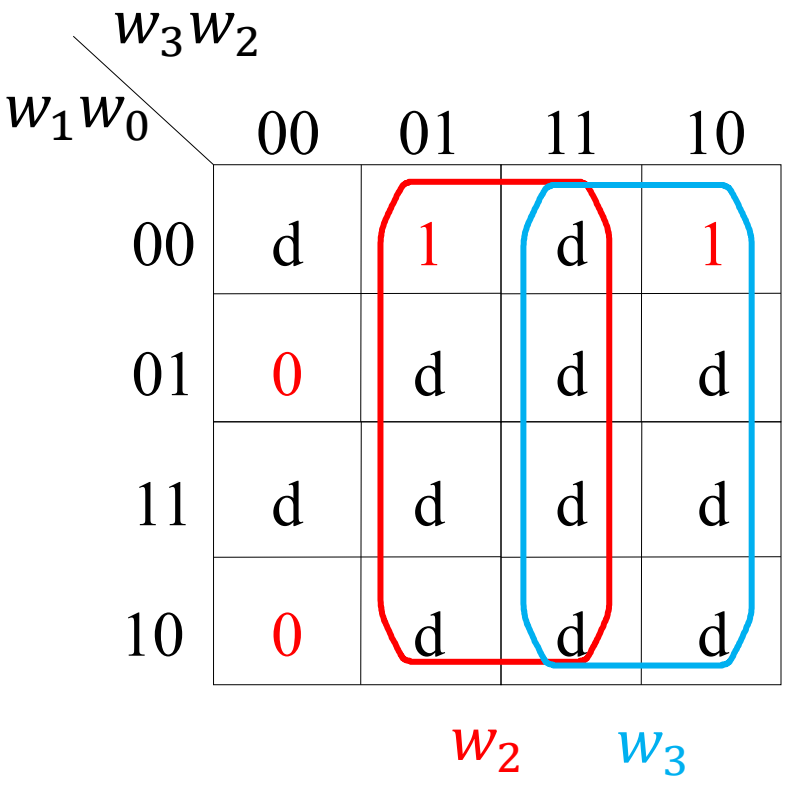
(a) Truth table



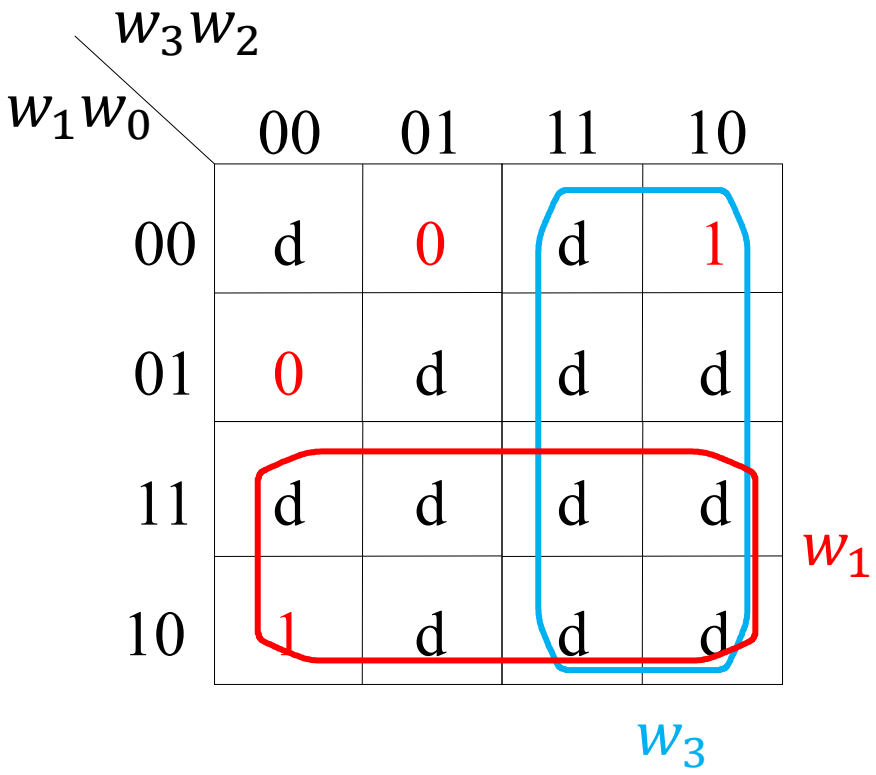
(b) Circuit

Figure 6.23. A 4-to-2 binary encoder.

4-to-2 binary encoder



$$y_1 = w_2 + w_3$$



$$y_0 = w_1 + w_3$$

4-to-2 priority encoder

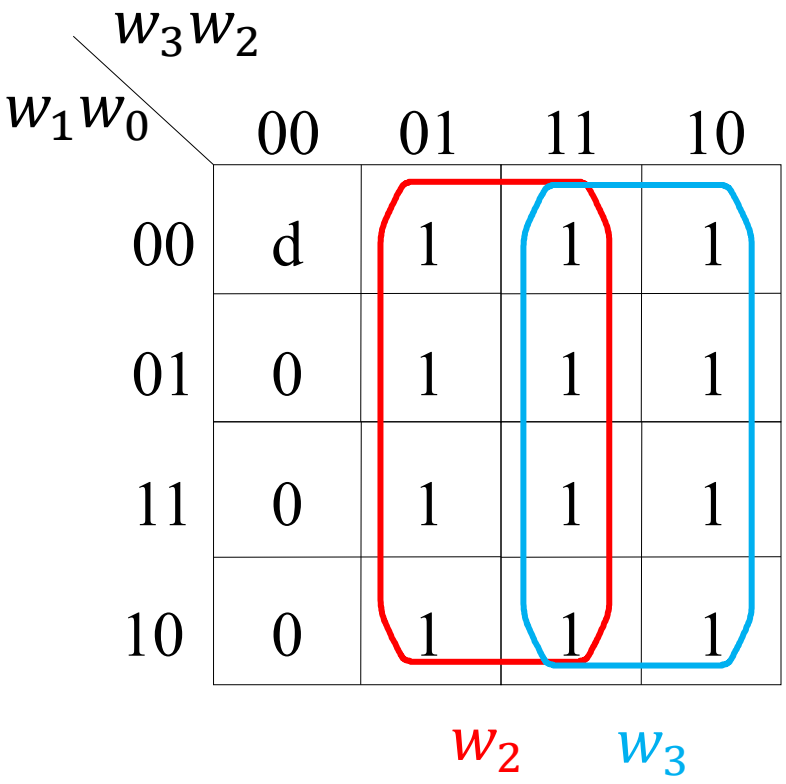
$w_3w_2w_1w_0 = 001\times = \begin{cases} 0010 \\ 0011 \end{cases}$

$w_3w_2w_1w_0 = 01\times\times = \begin{cases} 0100 \\ 0101 \\ 0110 \\ 0111 \end{cases}$

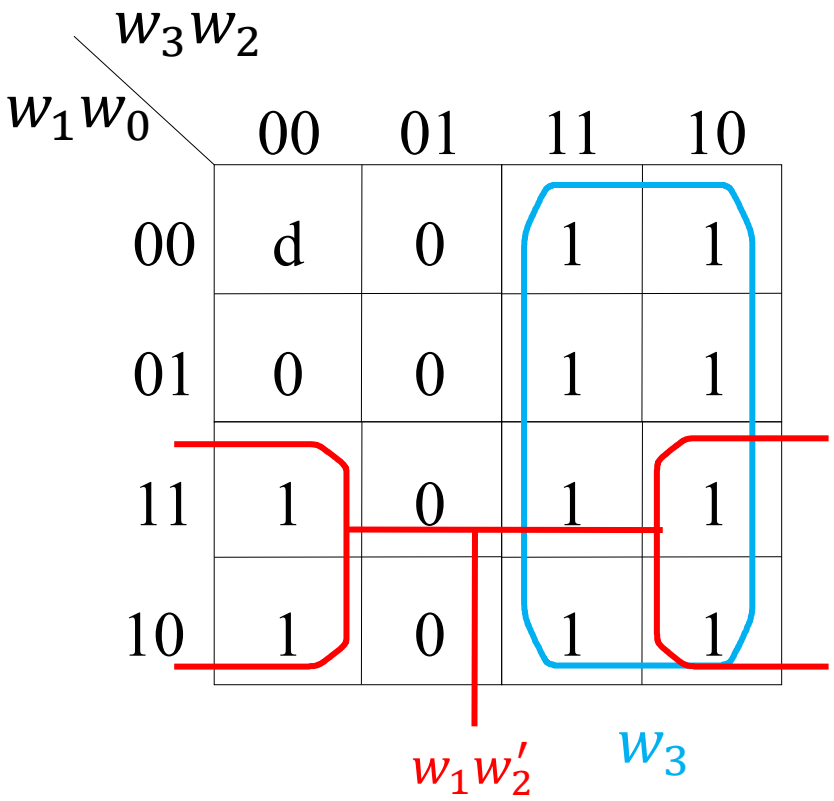
w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Figure 6.24. Truth table for a 4-to-2 priority encoder.

4-to-2 priority encoder

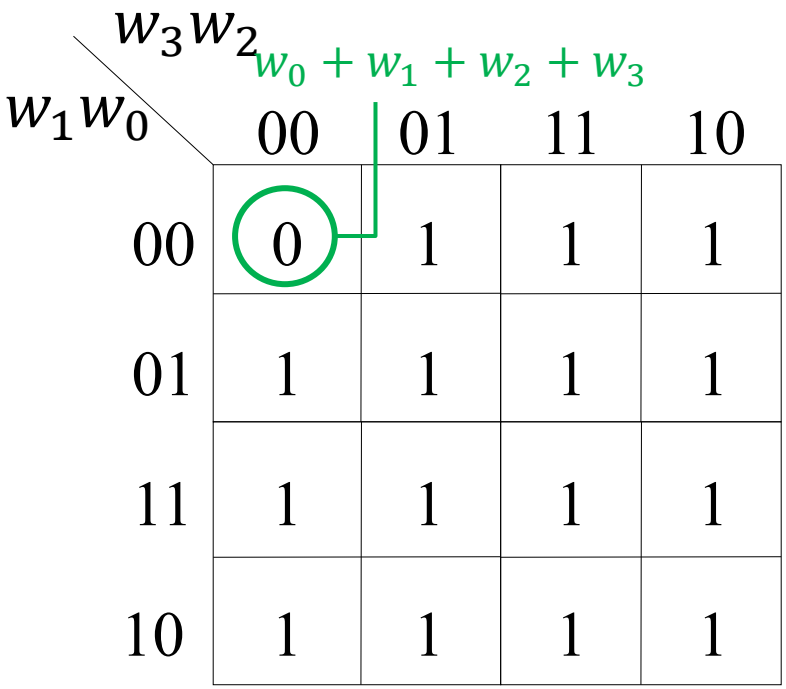


$$y_1 = w_2 + w_3$$



$$y_0 = w_1w_2' + w_3$$

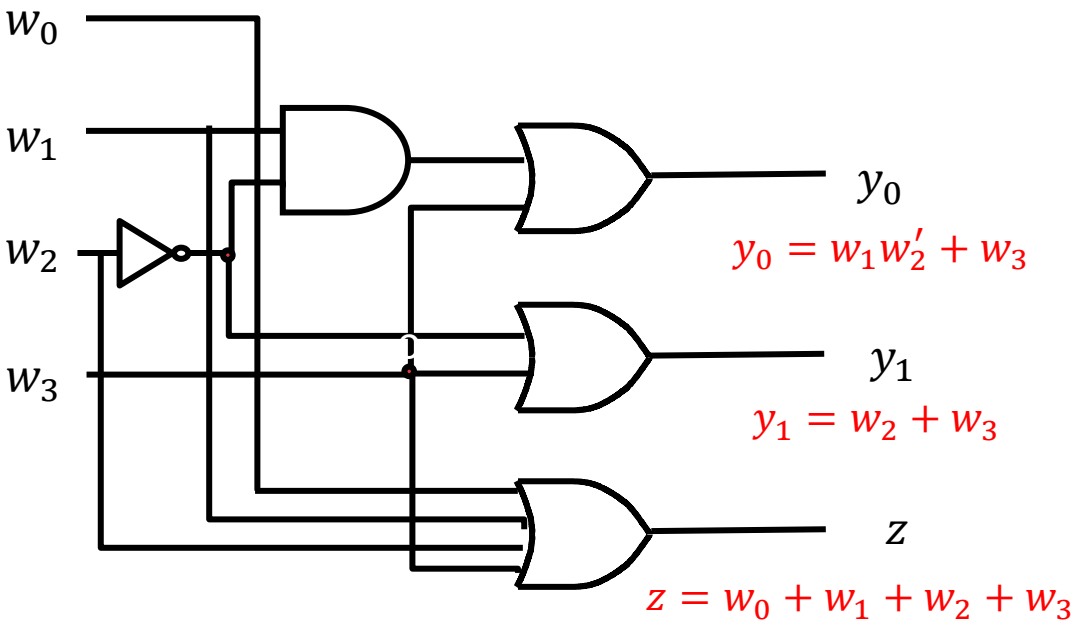
4-to-2 priority encoder



$$z = w_0 + w_1 + w_2 + w_3$$

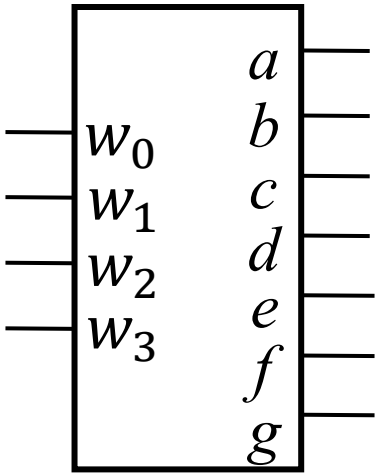
4-to-2 priority encoder

SOP Implementation

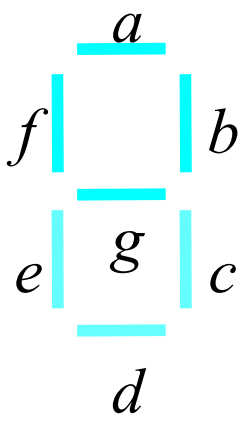


CODE CONVERTERS

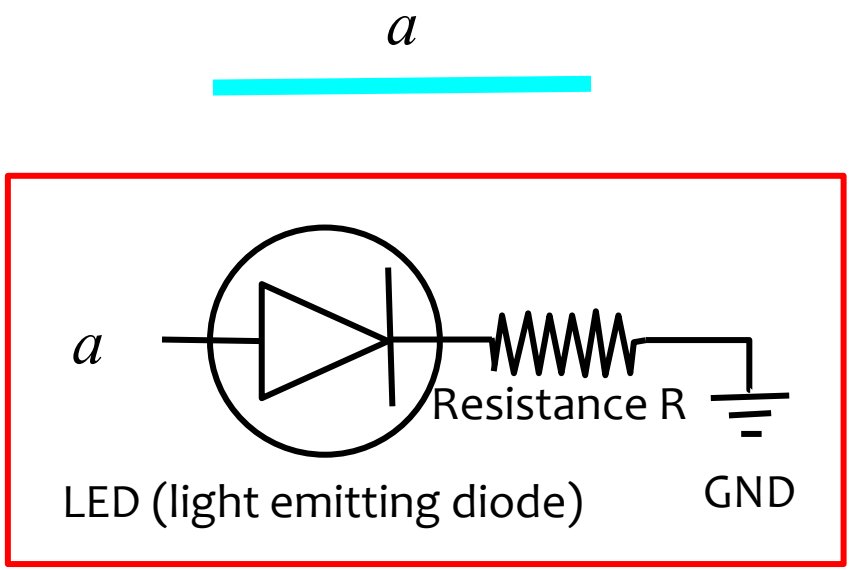
Code Converter

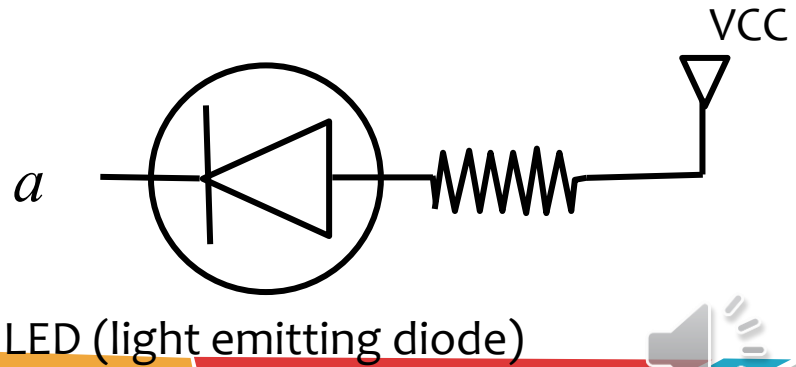


(a) Code converter



(b) 7-segment display





Code Converter

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

address

data

(c) Truth table

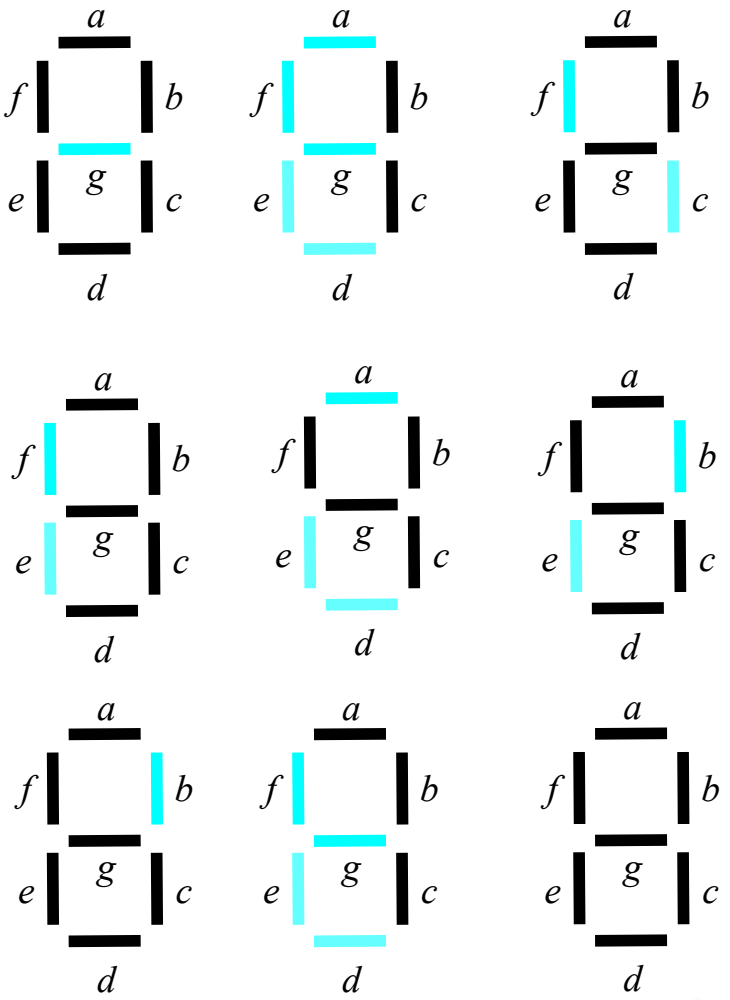
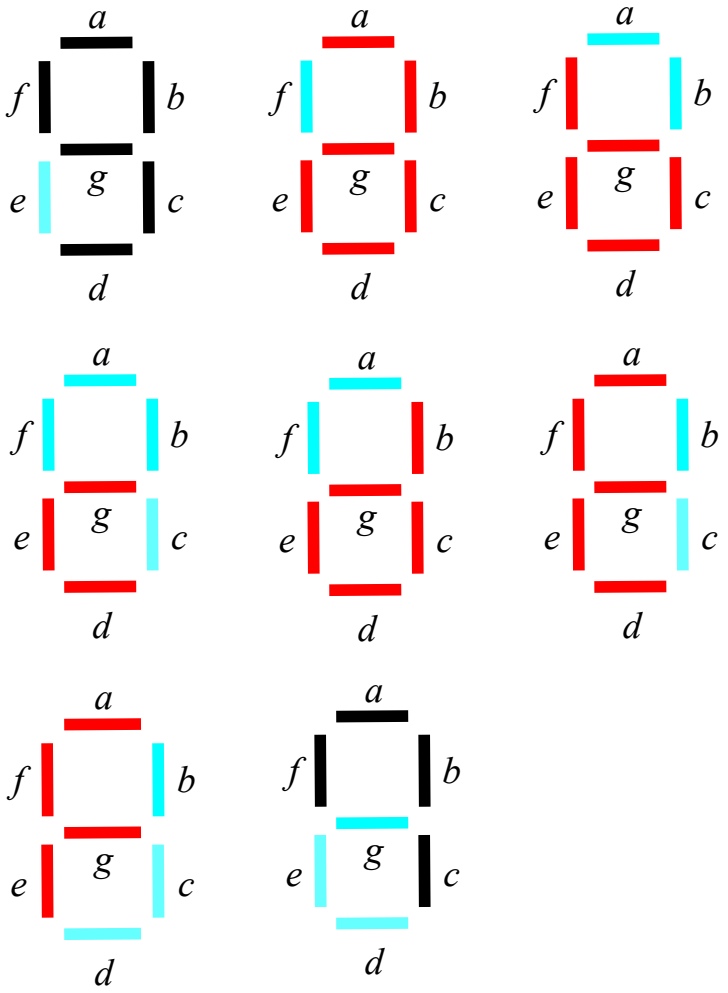


Figure 6.25. A BCD-to-7-segment display code converter.

Code Converter

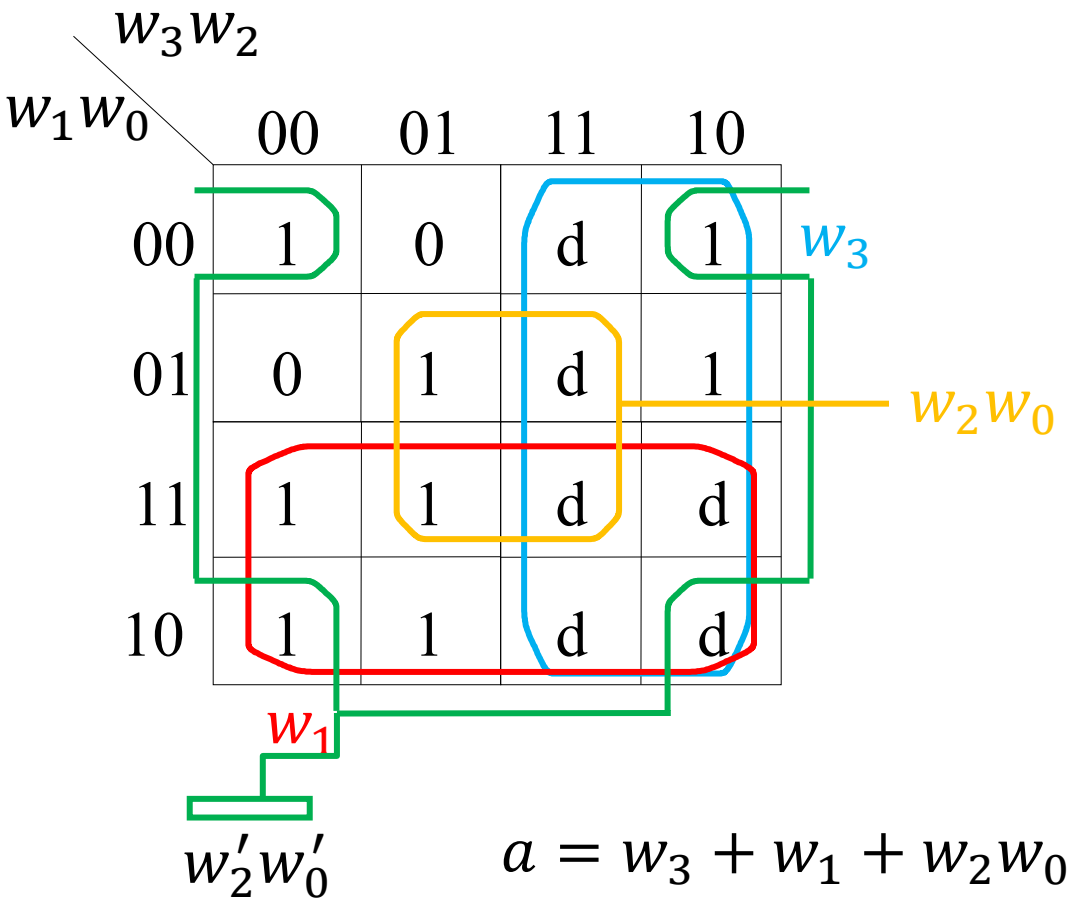
w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1



(c) Truth table

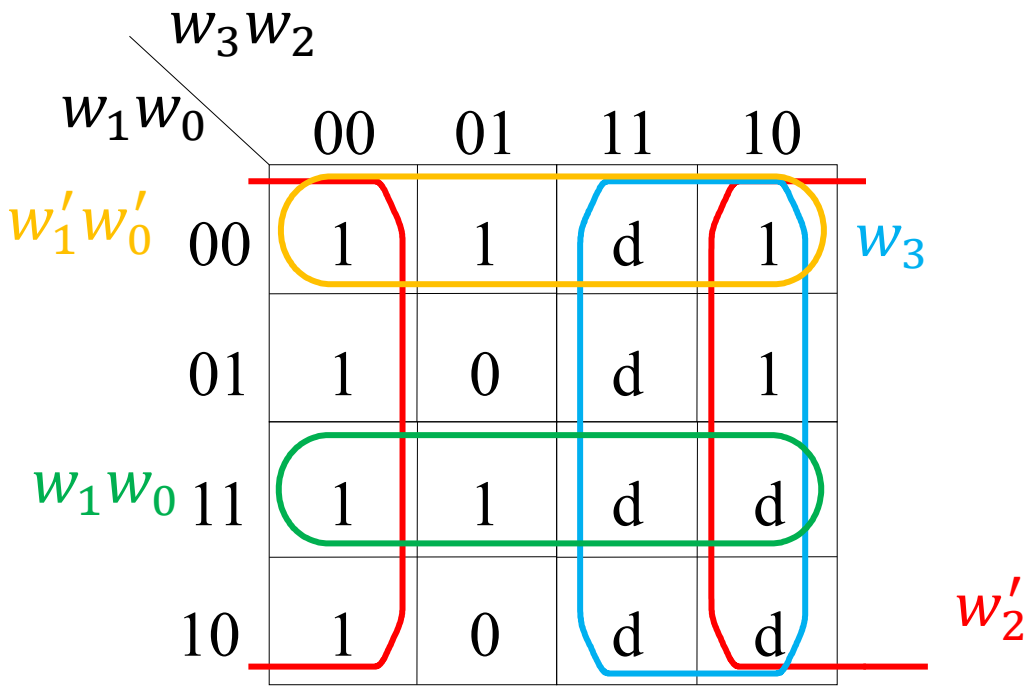
Figure 6.25. A BCD-to-7-segment display code converter.

Implementation of "a"



$$a = w_3 + w_1 + w_2w_0 + w_2'w_0'$$

Implementation of "b"



$$b = w_2' + w_3 + w_1w_0 + w_1'w_0'$$

Implementation of "c"

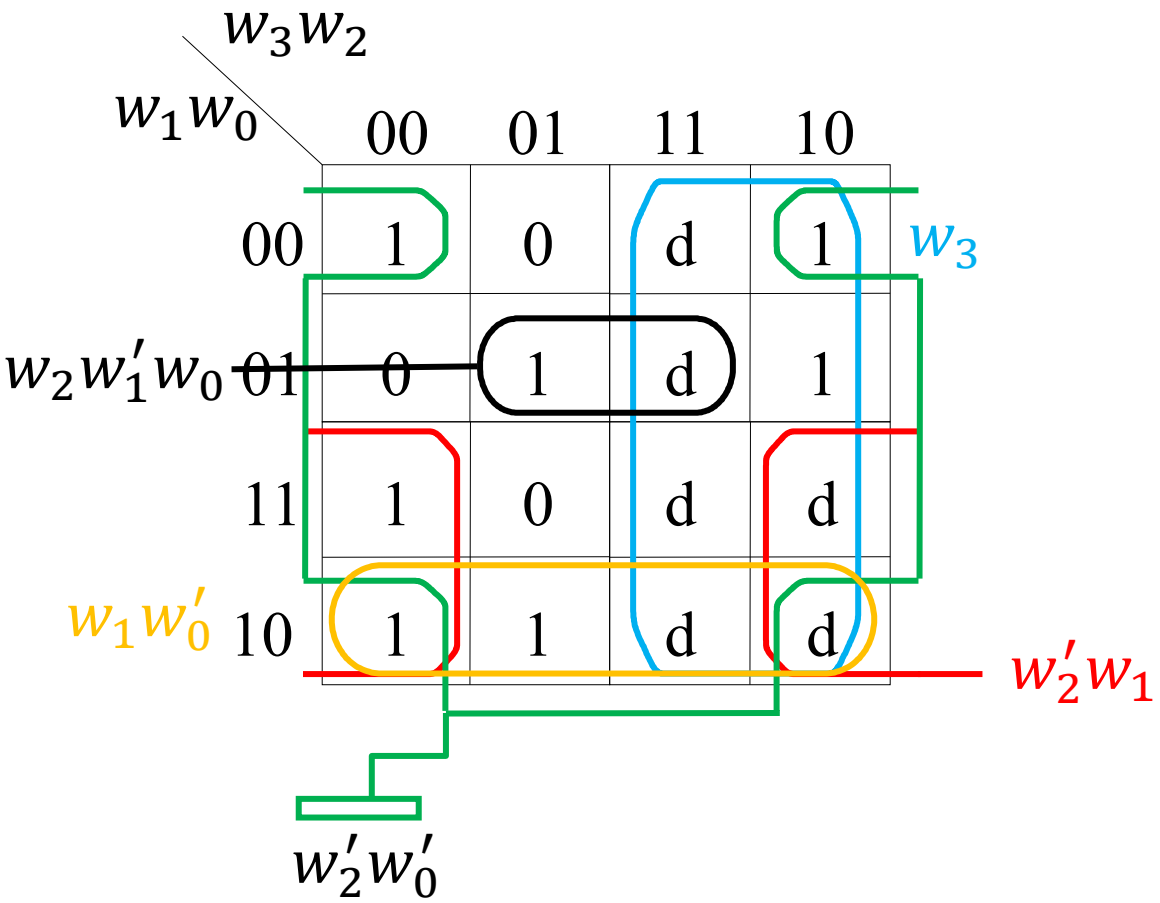
w_1w_0 \ w_3w_2	00	01	11	10
00	1	1	d	1
01	1	1	d	1
11	1	1	d	d
10	0	1	d	d

$w_0 + w_1' + w_2 + w_3$

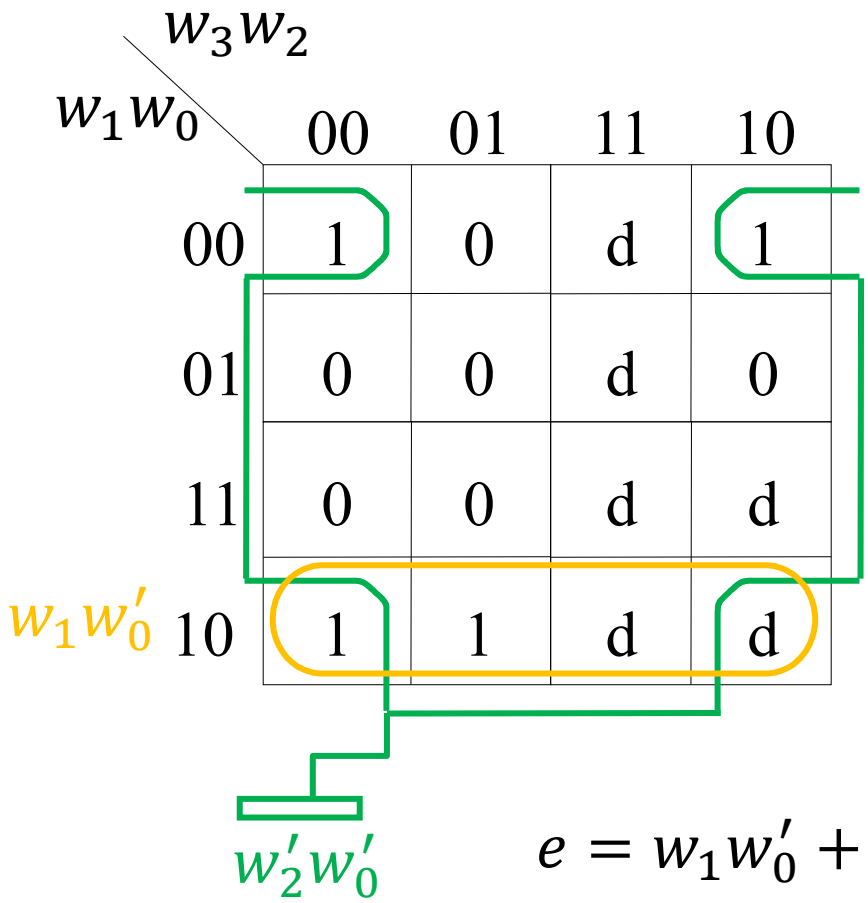
$$c = w_0 + w_1' + w_2 + w_3$$

Implementation of "d"

$$d = w_3 + w_2' w_1 + w_1 w_0' + w_2' w_0' + w_2 w_1' w_0$$

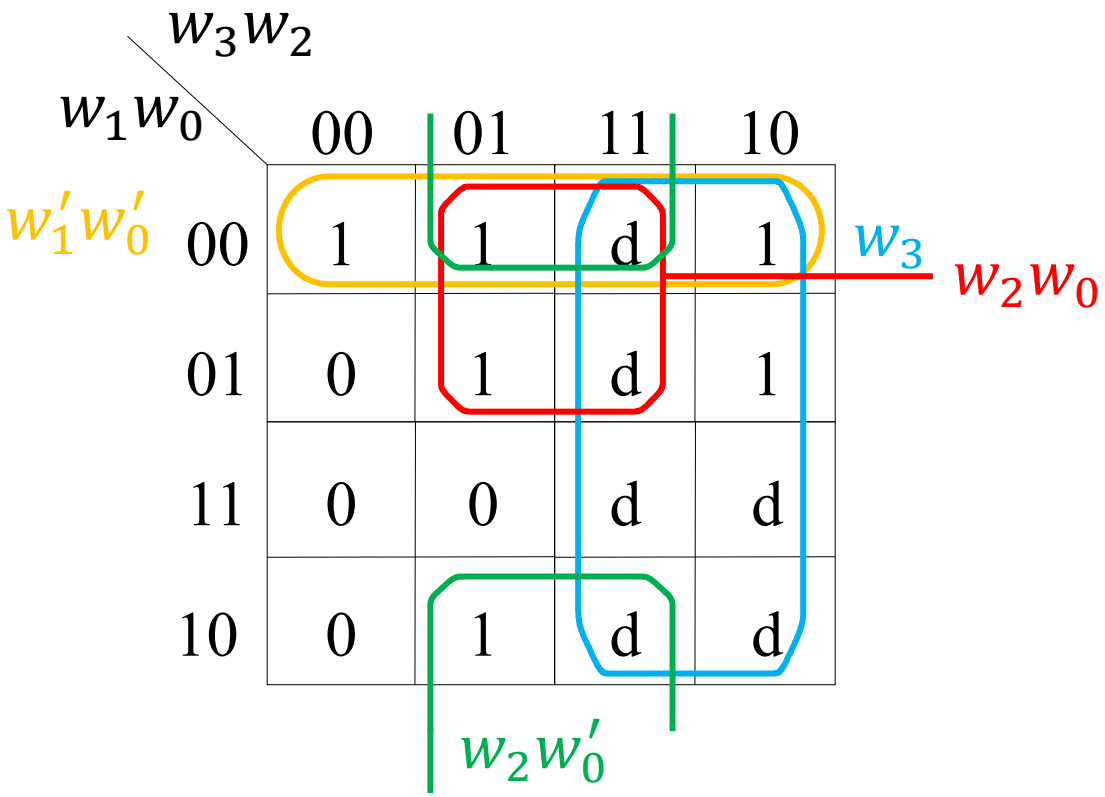


Implementation of "e"



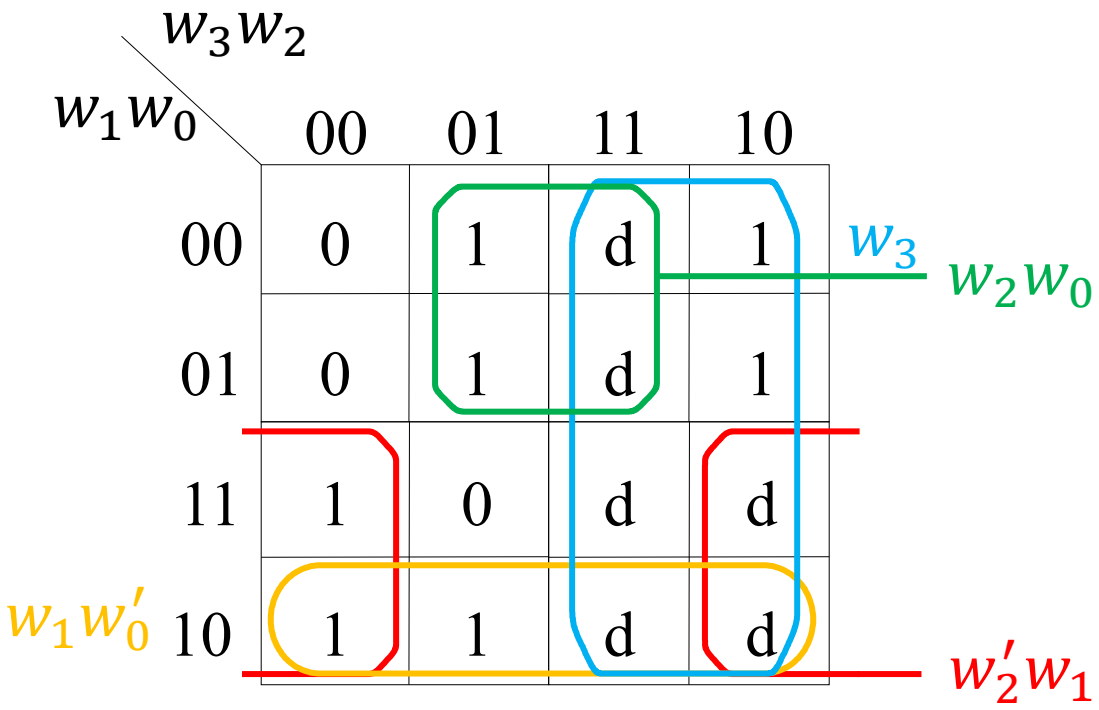
$$e = w_1w_0' + w_2'w_0'$$

Implementation of "f"



$$f = w_3 + w_2w_0 + w_1'w_0' + w_2w_0'$$

Implementation of "g"



$$g = w_3 + w_2w_0 + w_2'w_1 + w_1w_0'$$

Common Factors for Multiple Outputs

$$a = w_3 + w_1 + w_2 w_0 + w_2' w_0'$$

$$b = w_3 + w_2' + w_1 w_0 + w_1' w_0'$$

$$c = w_3 + w_2 + w_1' + w_0$$

$$d = w_3 + w_2' w_1 + w_1 w_0' + w_2' w_0' + w_2 w_1' w_0$$

$$e = w_1 w_0' + w_2' w_0'$$

$$f = w_3 + w_2 w_0 + w_1' w_0' + w_2 w_0'$$

$$g = w_3 + w_2 w_0 + w_2' w_1 + w_1 w_0'$$

ARITHMETIC COMPARISON CIRCUITS

Example 5.10

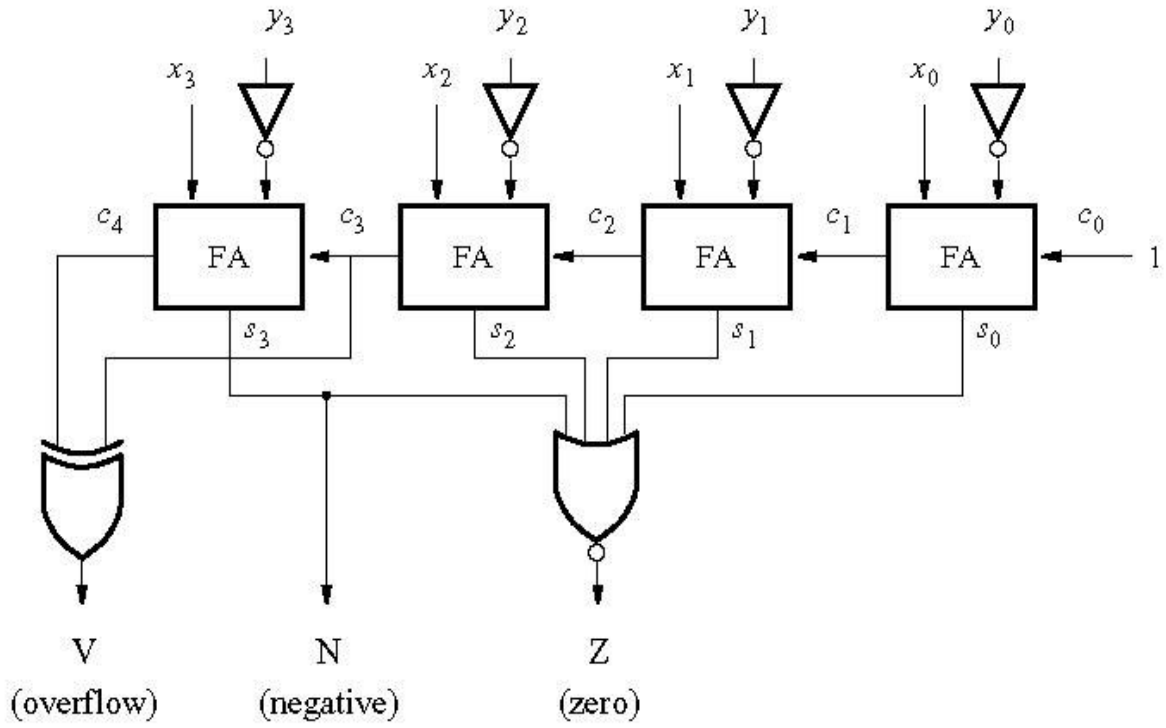


Figure 5.43. A comparator circuit.

$$X = x_3x_2x_1x_0$$
$$Y = y_3y_2y_1y_0$$

$$X = x_3x_2x_1x_0$$
$$-Y = y_3'y_2'y_1'y_0'$$
$$+1$$

$$c_4S = c_4s_3s_2s_1s_0$$

$$S = s_3s_2s_1s_0$$
$$V = c_4 \oplus c_3$$
$$N = s_3$$
$$Z = (s_3 + s_2 + s_1 + s_0)'$$
$$= \begin{cases} 1 \; if \; X - Y = 0 \\ 0 \quad otherwise \end{cases}$$

Example 5.10

```
if (X>Y)
        F=X;
else
        F=Y;
```

$$Z = \begin{cases} 1 \; if \; X = Y \\ 0 \; if \; X \neq Y \end{cases}$$

$$N = \begin{cases} 1 \; if \; X < Y \\ 0 \; if \; X \geq Y \end{cases}$$

$$P = N'Z' = \begin{cases} 1 \; if \; X > Y \\ 0 \; if \; X \leq Y \end{cases}$$

Four-bit Comparator

$A = a_3 a_2 a_1 a_0 \qquad B = b_3 b_2 b_1 b_0$

Using Equivalence (X-NOR)

$i_k = (a_k \oplus b_k)' \; for \; k = 0, 1, 2, 3$

$AeqB = i_3 i_2 i_1 i_0$

Conditions for $AgtB$

1) $if \; a_3 > b_3, that \; is \; a_3 = 1, b_3 = 0$ ⇒ $a_3 b_3'$

2) $if \; a_3 = b_3, \; a_2 > b_2, that \; is \; a_2 = 1, b_2 = 0$ ⇒ $i_3 a_2 b_2'$

3) $if \; a_3 = b_3, a_2 = b_2, a_1 > b_1, that \; is \; a_1 = 1, b_1 = 0$ ⇒ $i_3 i_2 a_1 b_1'$

4) $if \; a_3 = b_3, a_2 = b_2, a_1 = b_1, a_0 > b_0, that \; is \; a_0 = 1, b_0 = 0$ ⇒ $i_3 i_2 i_1 a_0 b_0'$

$AgtB = a_3 b_3' + i_3 a_2 b_2' + i_3 i_2 a_1 b_1' + i_3 i_2 i_1 a_0 b_0'$

$AltB = (AeqB + AgtB)'$

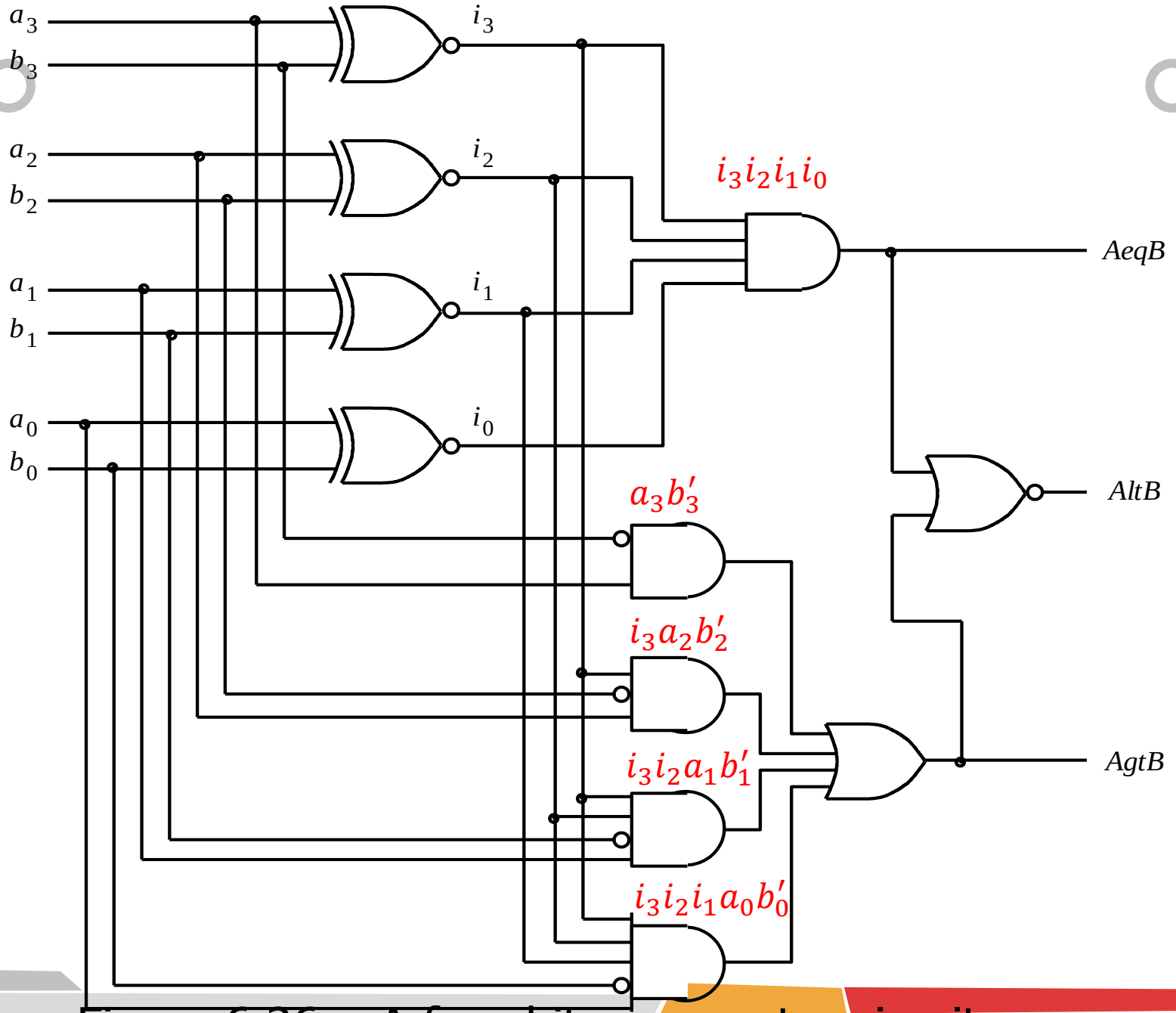


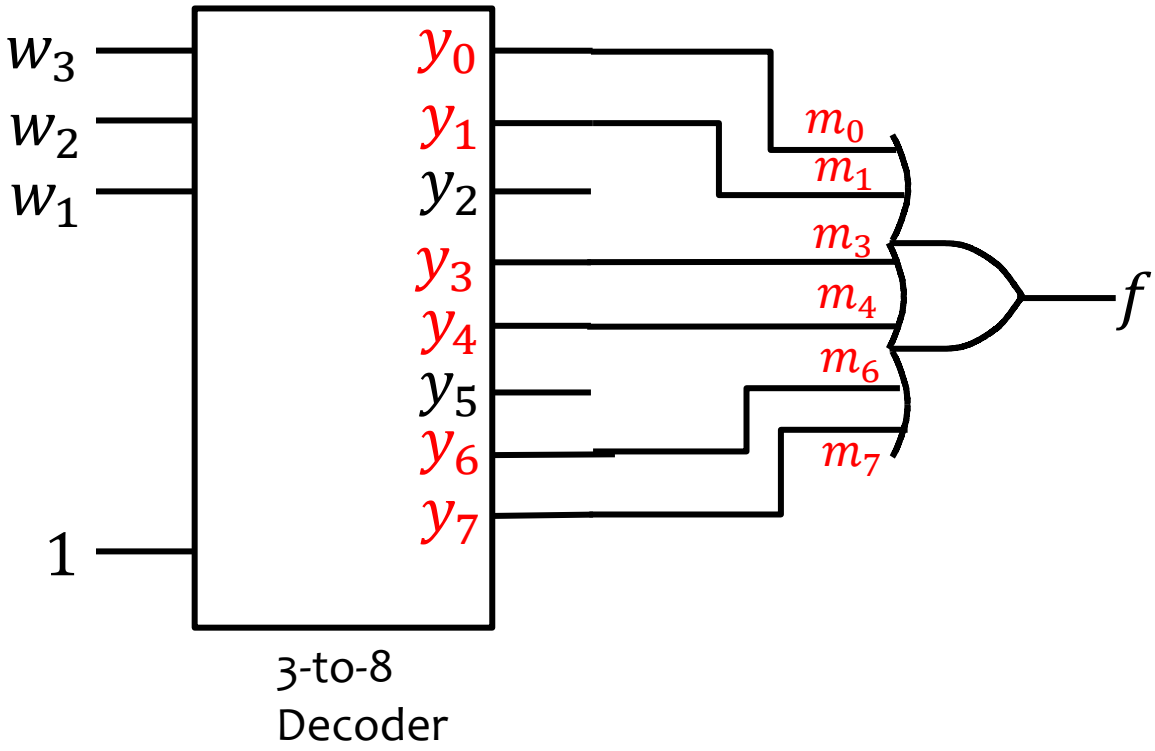
Figure 6.26. A four-bit comparator circuit.

EXAMPLES OF SOLVED PROBLEMS

Example 6.25

Problem: Implement the function $f(w_1, w_2, w_3) = \sum m(0,1,3,4,6,7)$ by using a 3-to-8 binary decoder and an OR gate.

$$y_0 = (En)w_1'w_2'w_3' = (1)w_1'w_2'w_3' = m_0$$



Example 6.26

Problem: Derive a circuit that implements an 8-to-3 binary encoder

w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	y_2	y_1	y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$y_2 = w_4 + w_5 + w_6 + w_7$$

$$y_1 = w_2 + w_3 + w_6 + w_7$$

$$y_0 = w_1 + w_3 + w_5 + w_7$$

Example 6.27

Problem: Implement the function by using a 4-to-1 multiplexer and as few other gates as possible. Assume that only the uncomplemented inputs w_1, w_2, w_3, w_4, and w_5 are available.

$$f(w_1, w_2, w_3, w_4, w_5) = w_1'w_2'w_4'w_5' + w_1w_2 + w_1w_3 + w_1w_4 + w_3w_4w_5$$

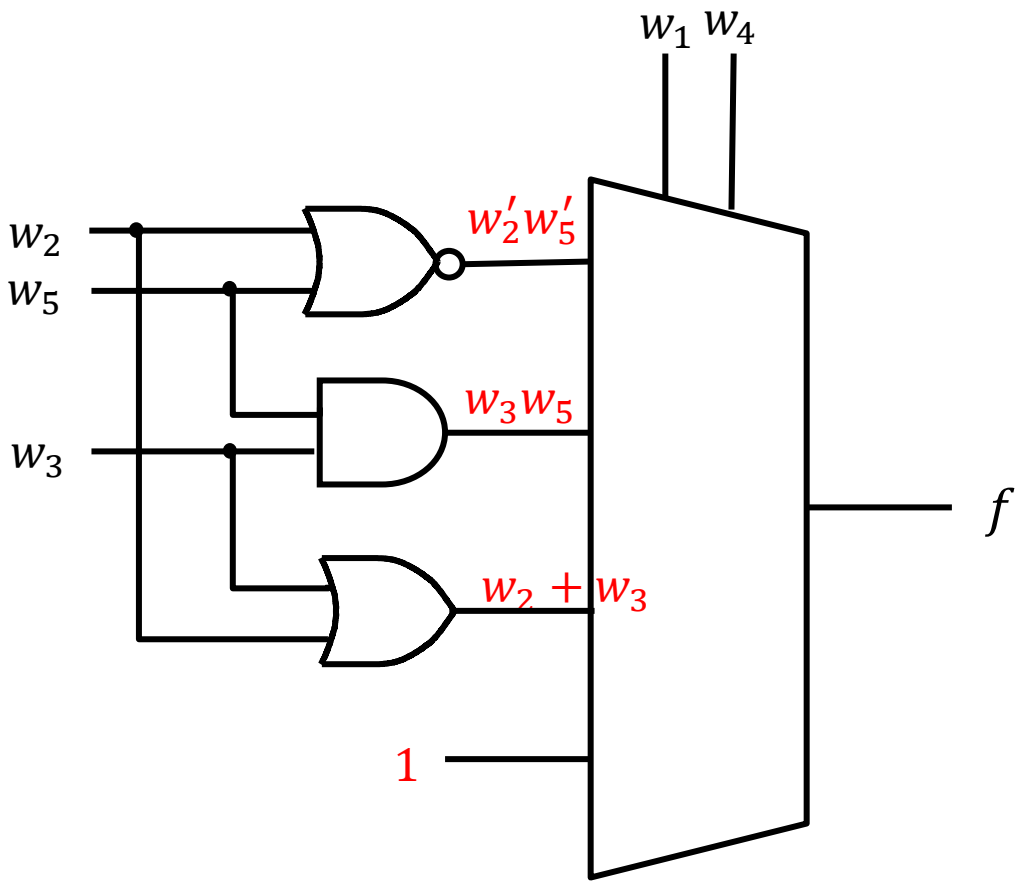
Choose two variables to use 4-to-1 MUX and adopt the Shannon's expansion theorem.

$$f(w_1, w_2, w_3, w_4, w_5) = w_1'w_2'f_{w_1'w_2'} + w_1'w_2f_{w_1'w_2} + w_1w_2'f_{w_1w_2'} + w_1w_2f_{w_1w_2}$$
$$= w_1'w_2'(w_4'w_5') + w_1'w_2(w_3w_4) + w_1w_2'(w_3 + w_4 + w_3w_4w_5) + w_1w_2(1)$$

$$f(w_1, w_2, w_3, w_4, w_5) = w_1'w_4'f_{w_1'w_4'} + w_1'w_4f_{w_1'w_4} + w_1w_4'f_{w_1w_4'} + w_1w_4f_{w_1w_4}$$
$$= w_1'w_4'(w_2'w_5') + w_1'w_4(w_3w_5) + w_1w_4'(w_2 + w_3) + w_1w_4(1)$$

$w_2'w_5' = (w_2 + w_5)'$ NOR

Example 6.27



Example 6.28

Problem: In Chapter 4 we pointed out that the rows and columns of A Karnaugh map are labeled using Gray code. This is a code in which consecutive valuations differ in one variable only. Figure 6.52 depicts the conversion between three-bit binary and Gray codes. Design a circuit that can convert a binary code into Gray code according to the figure.

Example 6.28

b_2	b_1	b_0	g_2	g_1	g_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Figure 6.52 Binary to Gray code conversion

Example 6.28

g_2

b_0 \ b_2b_1	00	01	11	10
0	0	0	1	1
1	0	0	1	1

b_2

$$g_2 = b_2$$

g_1

b_0 \ b_2b_1	00	01	11	10
0	0	1	0	1
1	0	1	0	1

b_2b_1' $b_2'b_1$

$$g_1 = b_2'b_1 + b_2\,b_1' = b_2 \oplus b_1$$

g_0

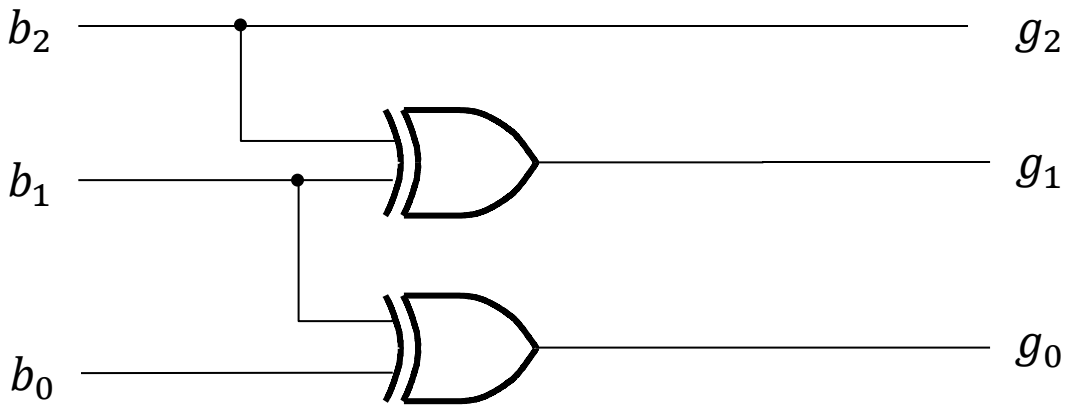
b_0 \ b_2b_1	00	01	11	10
0	0	1	1	0
1	1	0	0	1

b_1b_0' $b_1'b_0$

$$g_0 = b_1'b_0 + b_1\,b_0' = b_1 \oplus b_0$$

Example 6.28

$g_2 = b_2$ $g_1 = b_2 \oplus b_1$ $g_0 = b_1 \oplus b_0$



Example 6.29

Problem: In section 6.1.2 we showed that any logic function can be decomposed using Shannon's expansion theorem. For a four-Variable function, $f(w_1, \cdots, w_4)$, the expansion with respect to w_1 is

$$f(w_1, \cdots, w_4) = w_1' f_{w_1'} + w_1 f_{w_1}$$

A circuit that implements this expression is given Figure 6.53(a).

a. If the decomposition yields $f_{w_1'} = 0$, then the multiplexer in the figure can be replaced by a single logic gate.
Show this circuit.

b. Repeat part (a) for the case where $f_{w_1} = 1$.

Example 6.29

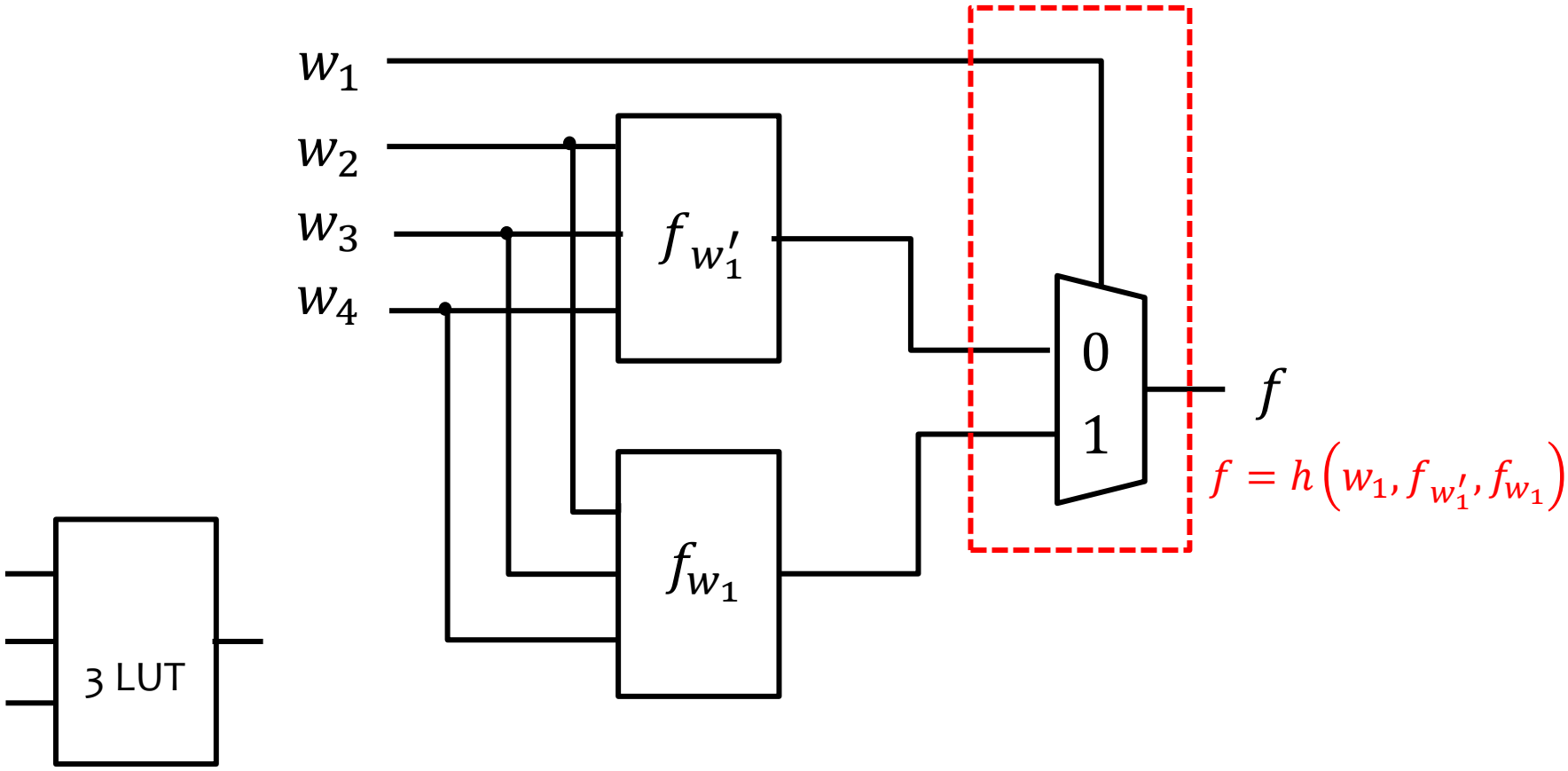


Figure 6.53 (a) Shannon's expansion of the function f

Example 6.29

$$f(w_1, \cdots, w_4) = w_1' f_{w_1'} + w_1 f_{w_1} \qquad if\ f_{w_1'} = 0$$
$$= w_1 f_{w_1}$$

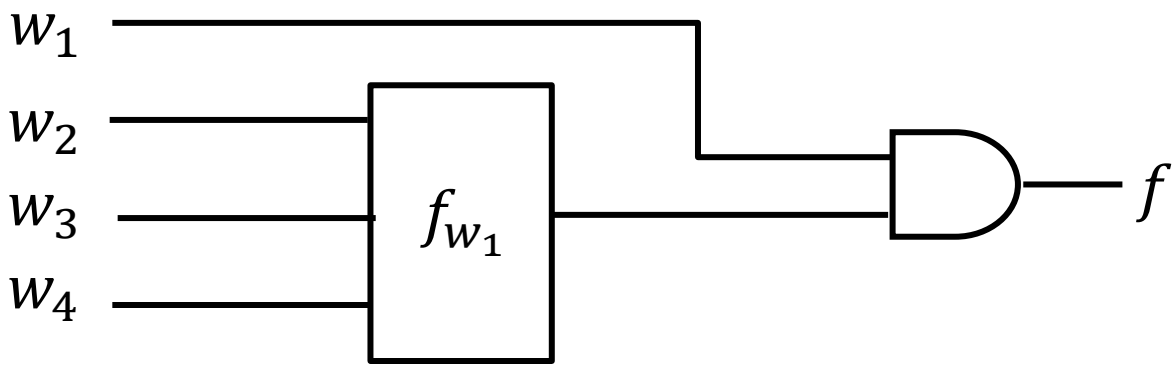


Figure 6.53 (b) Solution for part a

Example 6.29

$$f(w_1, \cdots, w_4) = w_1' f_{w_1'} + w_1 f_{w_1} \qquad if\ f_{w_1} = 1$$
$$= w_1' f_{w_1'} + w_1$$
$$= (w_1 + w_1')\left(w_1 + f_{w_1'}\right)$$
$$= w_1 + f_{w_1'}$$

$x + yz = (x + y)(x + z)$

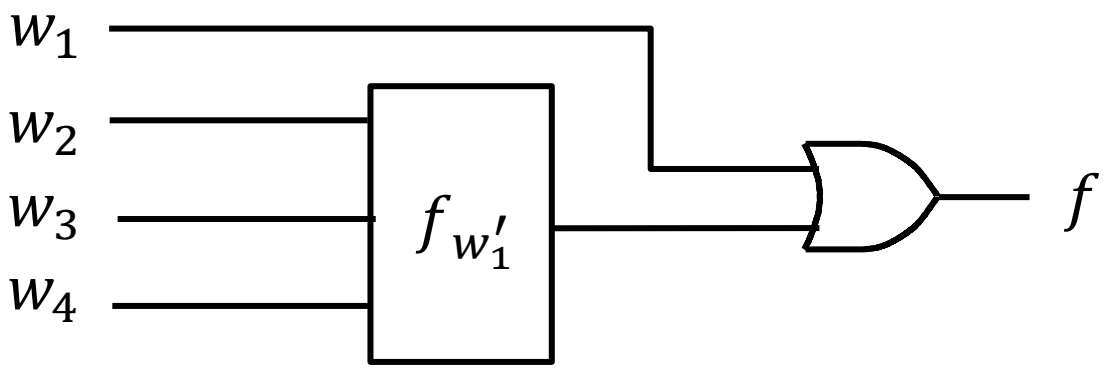


Figure 6.53 (c) Solution for part b

Example 6.30

Problem: In several commercial FPGAs the logic blocks are 4-LUTs. What is the minimum number of 4-LUTs needed to construct a 4-to-1 multiplexer with select inputs s_1 and s_0 and data inputs w_3, w_2, w_1, and w_0?

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

(b) Truth table

$$f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_2 + s_1s_0w_3$$

6 variables

$$g = s_1's_0'w_0 + s_1's_0w_1$$ 4 variables

$$h = s_1s_0'w_2 + s_1s_0w_3$$ 4 variables

$$f = g + h$$ 2 variables

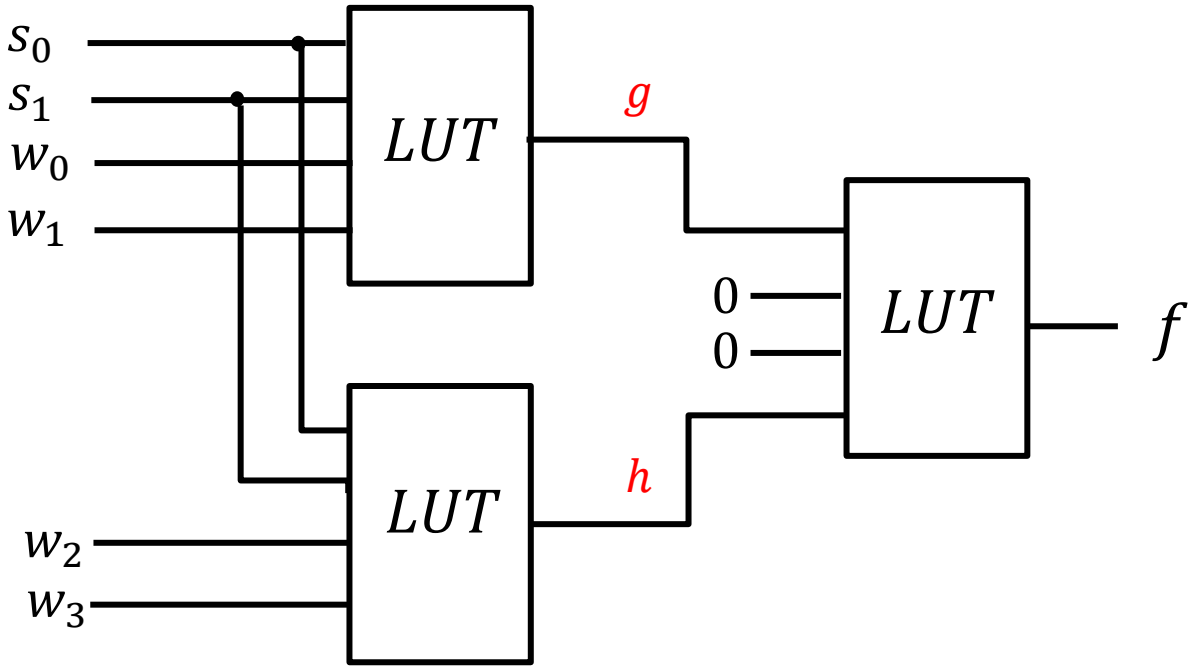
Example 6.30

$$f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_2 + s_1s_0w_3$$ 6 variables

$$g = s_1's_0'w_0 + s_1's_0w_1$$ 4 variables

$$h = s_1s_0'w_2 + s_1s_0w_3$$ 4 variables

$$f = g + h$$ 2 variables



Example 6.30

$k = s_1' s_0' w_0 + s_1' s_0 w_1 + s_1 s_0$ 4 variables

$= s_1'(s_0' w_0 + s_0 w_1) + s_1(s_0)$

$s_1' k = s_1' s_1' s_0' w_0 + s_1' s_1' s_0 w_1 + s_1' s_1 s_0 = s_1' s_0' w_0 + s_1' s_0 w_1$

$s_1 k = s_1 s_1' s_0' w_0 + s_1 s_1' s_0 w_1 + s_1 s_1 s_0 = s_1 s_0$

$k' = (s_1' s_0' w_0 + s_1' s_0 w_1 + s_1 s_0)'$

$= (s_1' s_0' w_0)'(s_1' s_0 w_1)'(s_1 s_0)'$

$= (s_1 + s_0 + w_0')(s_1 + s_0' + w_1')(s_1' + s_0')$

$s_1 k' = (s_1 + s_0 + w_0')(s_1 + s_0' + w_1')(s_1 s_1' + s_1 s_0')$

$= (s_1 + s_0 + w_0')(s_1 + s_0' + w_1')(s_1 s_0')$

$= (s_1 s_0' s_1 + s_1 s_0' s_0 + s_1 s_0' w_0')(s_1 + s_0' + w_1')$

$= (s_1 s_0' + s_1 s_0' w_0')(s_1 + s_0' + w_1')$

$= (s_1 s_0' s_1 + s_1 s_0' s_0' + s_1 s_0' w_1') + (s_1 s_0' w_0' s_1 + s_1 s_0' w_0' s_0' + s_1 s_0' w_0' w_1')$

$= s_1 s_0' + s_1 s_0' w_1' + s_1 s_0' w_0' + s_1 s_0' w_0' w_1'$

$= s_1 s_0'(1 + w_1' + w_0' + w_0' w_1') = s_1 s_0'$

Example 6.30

$$k = s_1' s_0' w_0 + s_1' s_0 w_1 + s_1 s_0$$

4 variables

$$s_1' k = s_1' s_0' w_0 + s_1' s_0 w_1$$

$$s_1 k' = s_1 s_0'$$

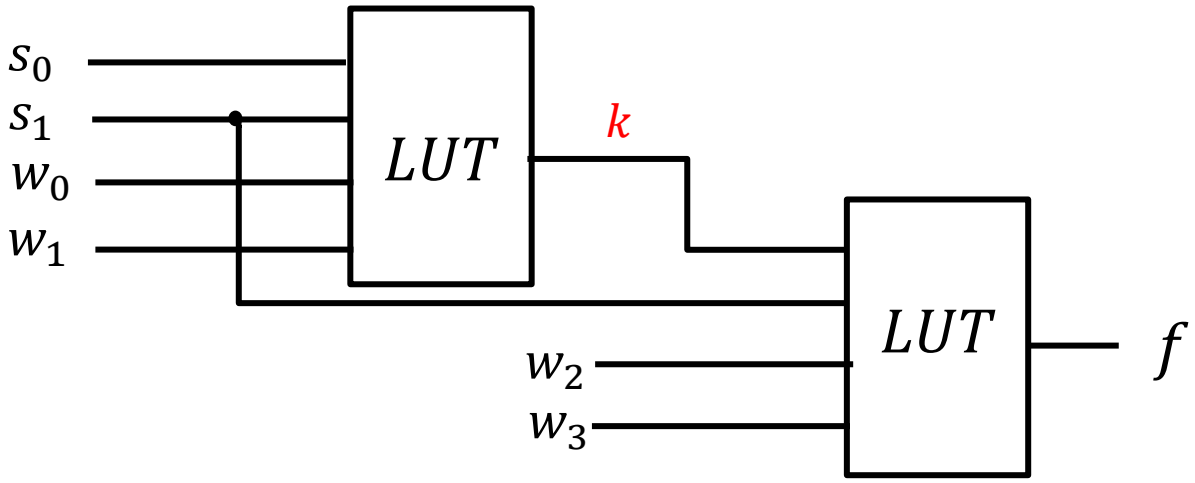
$$s_1 k = s_1 s_0$$

$$f = s_1' s_0' w_0 + s_1' s_0 w_1 + s_1 s_0' w_2 + s_1 s_0 w_3$$

$$= s_1' k + s_1 k' w_2 + s_1 k w_3$$

6 variables

4 variables



Example 6.31

Problem: In digital systems it is often necessary to have circuits that can shift the bits of a vector by one or more bit positions to the left or right. Design a circuit that can shift a four-bit vector $W = w_3w_2w_1w_0$ one bit position to the right when a control signal $Shift$ is equal to 1. Let the outputs of the circuit be a four-bit vector $Y = y_3y_2y_1y_0$ and a signal k, such that if $Shift = 1$ then $y_3 = 0$, $y_2 = w_3, y_1 = w_2, y_0 = w_1, k = w_0$.
If $Shift = 0$ then $Y = W$ and $k = 0$

Example 6.31

$Shift = 0$

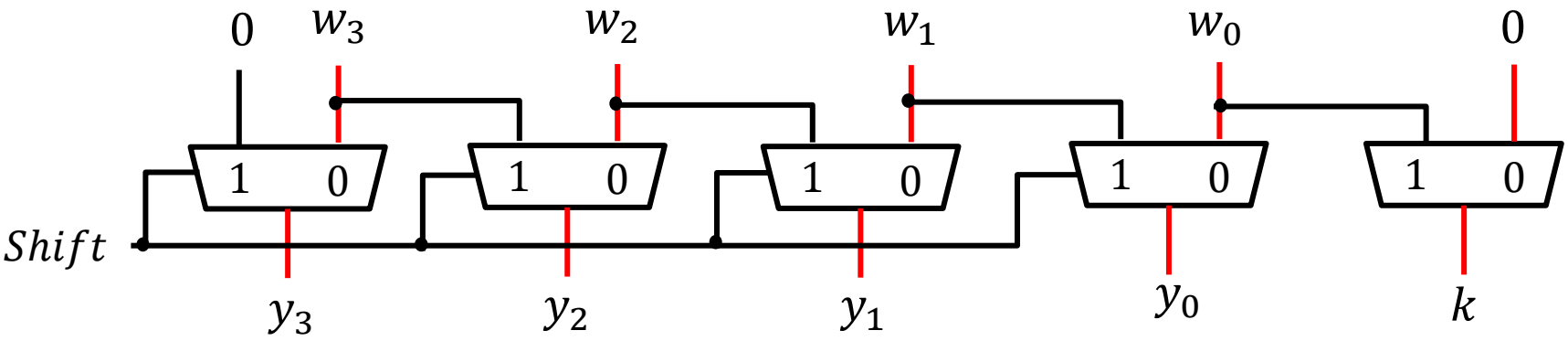


Figure 6.55 A shifter circuit

Example 6.31

$Shift = 1$

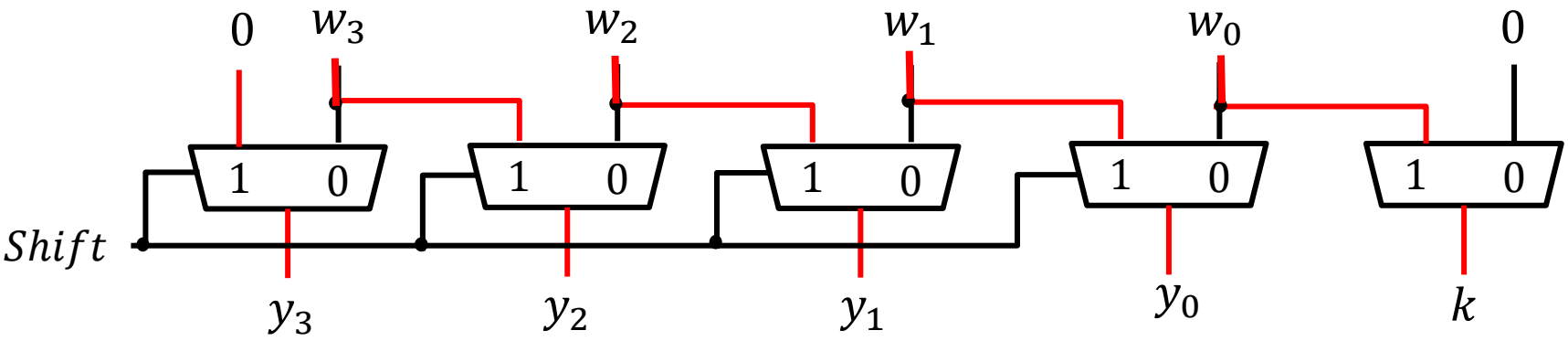


Figure 6.55 A shifter circuit

Example 6.32

Problem: The Shifter circuit in Example 6.31 shifts the bits of an input vector by one bit position to the right. It fills the vacated bit on the left side with 0. A more versatile shifter circuit may be able to shift by more bit positions at a time. If the bits that are shifted out are placed into the vacated positions on the left, then the circuit effectively rotates the bits of the input vector by a specified number of bit positions. Such a circuit is often called a barrel shifter. Design a four-bit barrel shifter that rotates the bits by 0, 1, 2, or 3 bit positions as determined by the valuation of two control signals s_1 and s_0.

Example 6.32

s_1	s_0	y_3	y_2	y_1	y_0	
0	0	w_3	w_2	w_1	w_0	No shift
0	1	w_0	w_3	w_2	w_1	Shift by 1 to the right
1	0	w_1	w_0	w_3	w_2	Shift by 2 to the right
1	1	w_2	w_1	w_0	w_3	Shift by 3 to the right

(a) Truth table

Example 6.32

$s_1 s_0 = 00$

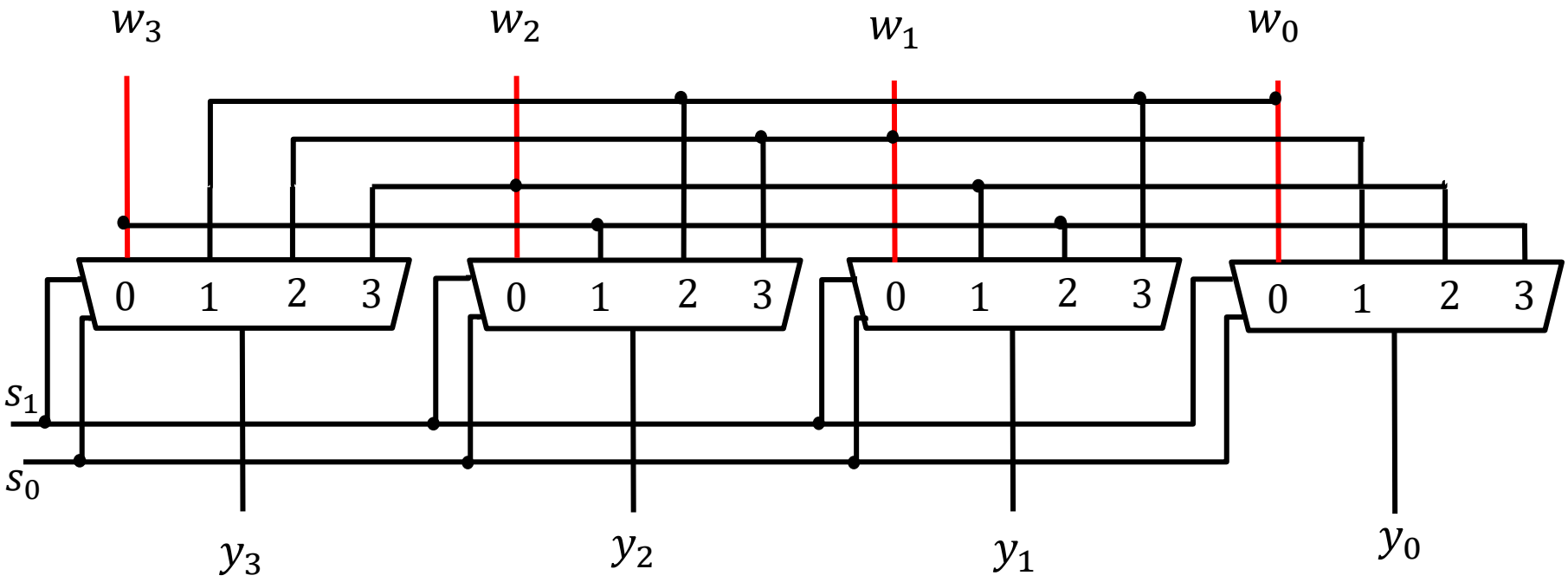


Figure 6.56 A barrel shifter circuit

Example 6.32

$s_1 s_0 = 01$

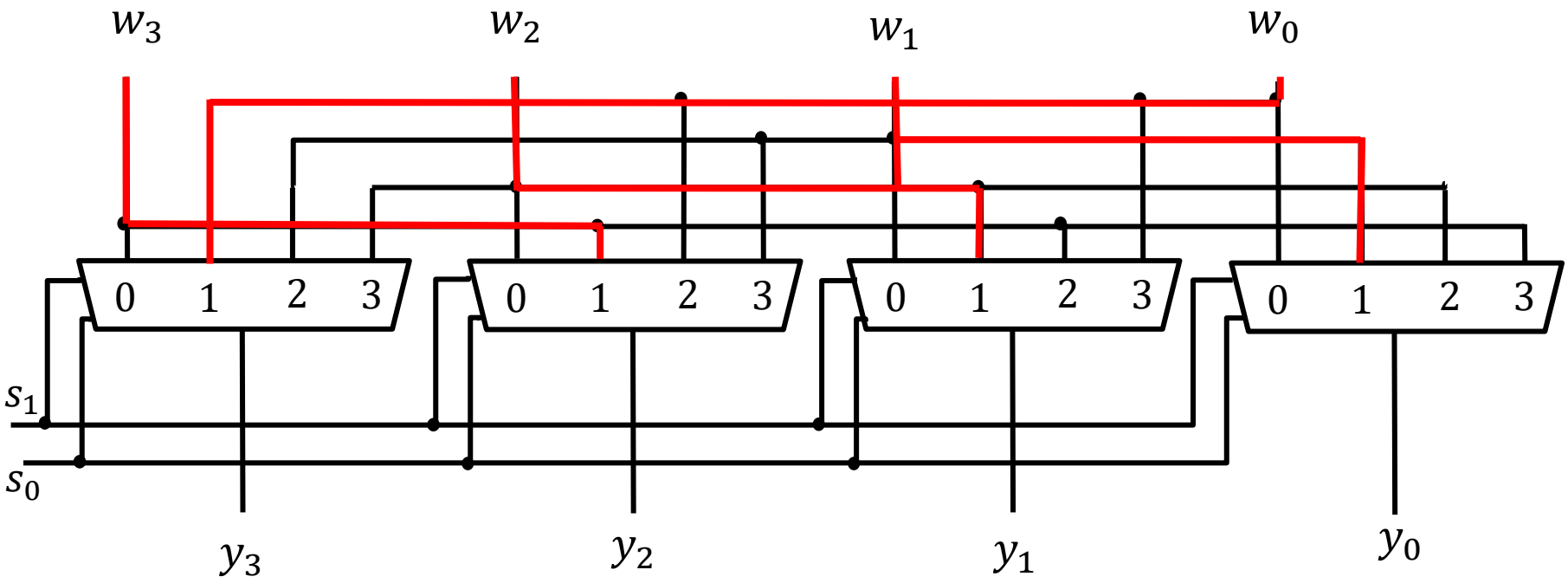


Figure 6.56 A barrel shifter circuit

Summary

- Multiplexers are used as the logic blocks in commercial FPGAs.
- All the logic function is synthesized by using the multiplexers.
- Multiplexer implementations of logic functions require that a given function be decomposed in terms of the variables that are used as the selected inputs based on the Shannon's expansion theorem.
- Decoder circuits are used to decode encoded information.

Summary

- Decoders can be used for address decoding and also for SOP implementation using minterms.
- Encoder encodes given information into a more compact form.
- As a kind of decoder, code converter circuit converts the BCD digit into seven signals that are used to drive the segments in the display.
- Comparator circuit compares the relative sizes of two binary numbers.