Rutgers University
School of Engineering

Fall 2022

332:231 – Digital Logic Design

Sophocles J. Orfanidis
ECE Department
orfanidi@rutgers.edu

Unit 4 – decoders, encoders, multiplexers, demultiplexers

Course Topics

1. Introduction to DLD, Verilog HDL, MATLAB/Simulink
2. Number systems
3. Analysis and synthesis of combinational circuits
4. Decoders/encoders, multiplexers/demultiplexers
5. Arithmetic systems, comparators, adders, multipliers
6. Sequential circuits, latches, flip-flops
7. Registers, shift registers, counters, LFSRs
8. Finite state machines, analysis and synthesis

Text: J. F. Wakerly, *Digital Design Principles and Practices*, 5/e, Pearson, 2018
additional references on Canvas Files > References

This unit has two parts:

1 .Digital design practices (Wakerly, Ch. 4), provides an overview of some conventions used in practice, such as, block diagrams, gate symbols, signal naming conventions, using active-high vs. active-low levels, bubble-to-bubble designs, layouts and schematics, circuit timing, timing diagrams, and propagation delays.

2. Basic combinational components (Wakerly, Ch. 6 & 7), that are commonly used in practice, such as, read-only-memories (ROMs), decoders, encoders, three-state buffers, priority encoders, multiplexers and demultiplexers, realizing arbitrary combinational functions with ROMs, decoders, and multiplexers.

Comparators are discussed in Sect. 7-4, and in unit-5 of lecture notes.

Unit-4 Contents:

Part 1 – Digital design practices

1. Block diagrams, gate symbols
2. Signal names, active-high, active-low levels
3. Bubble-to-bubble logic design
4. Layouts and schematics
5. Circuit timing, timing diagrams, propagation delays

Part 2 – Basic combinational components

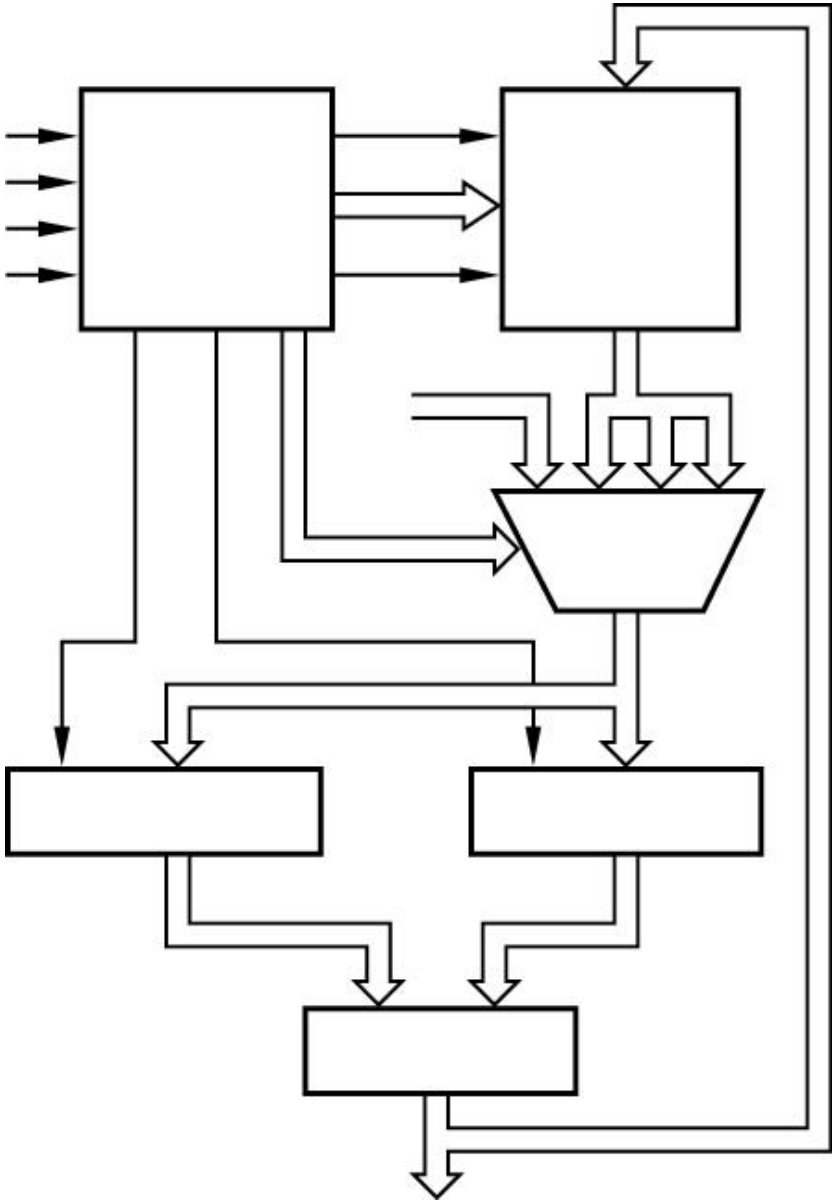
6. ROMs
7. Decoders
8. Realizing arbitrary combinational functions with decoders
9. Encoders
10. Three-state buffers

Contents, continued:

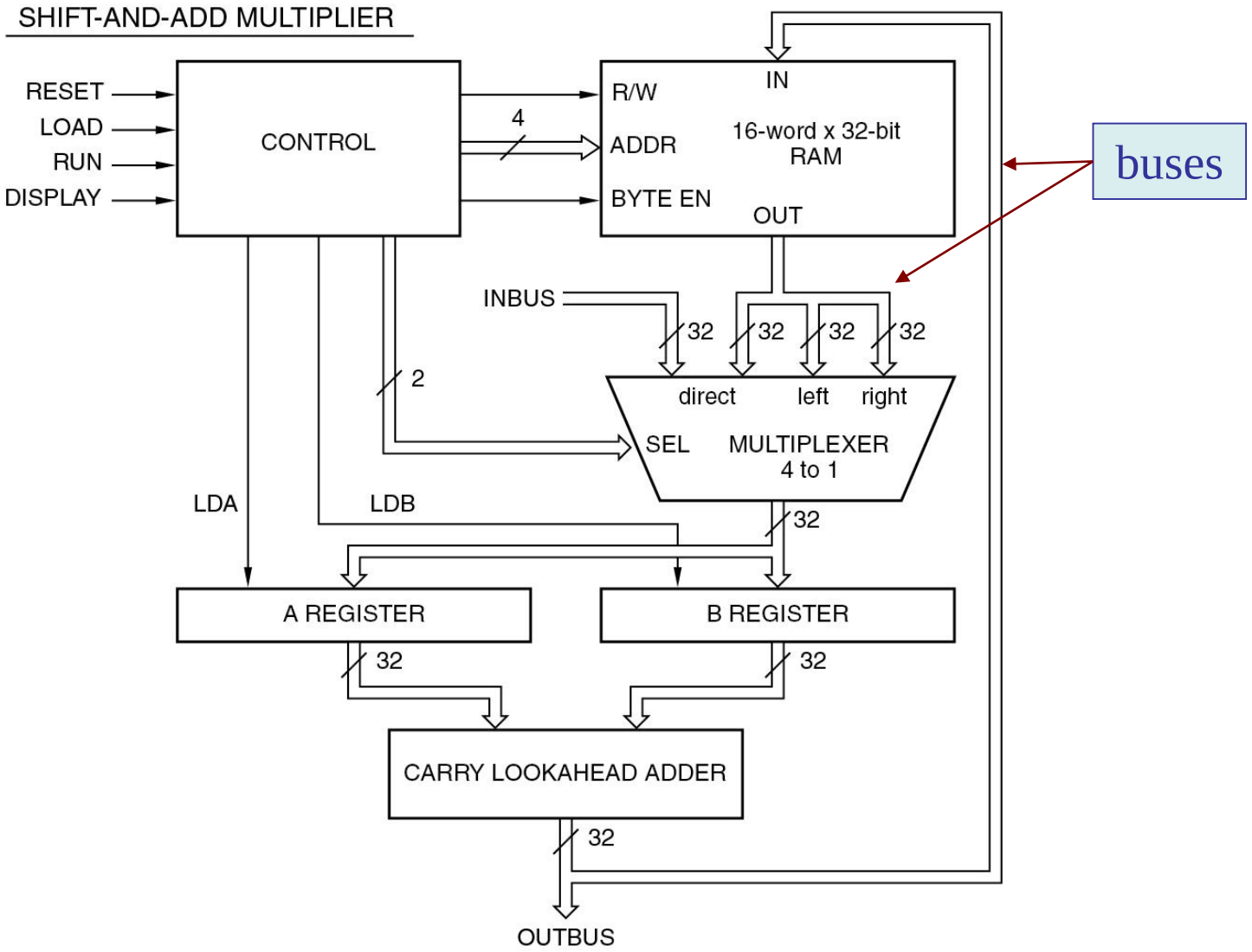
Part 1 – Digital Design Practices

(Wakerly, Ch.4)

block diagram example



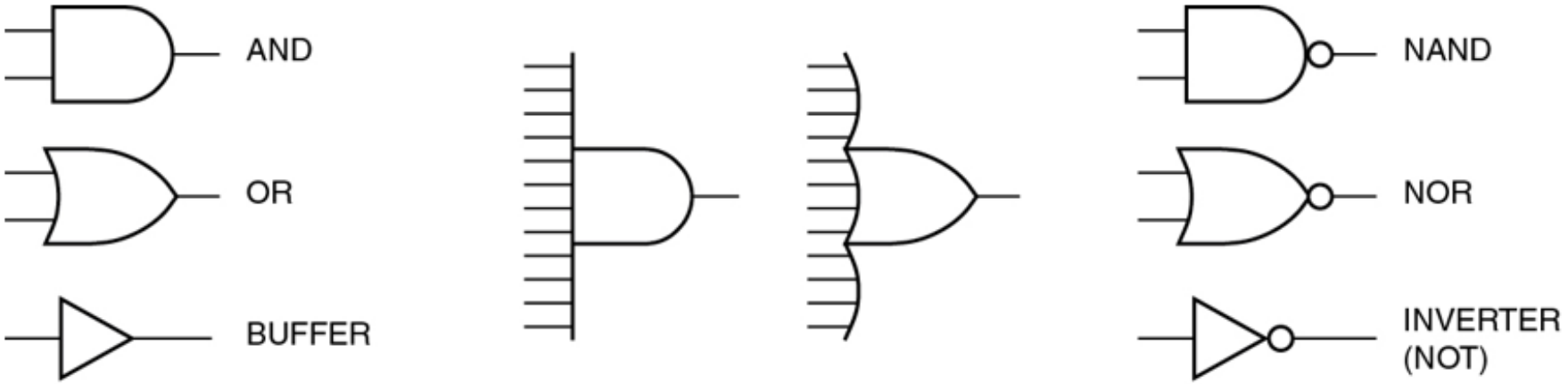
block diagram example



gate symbols

IEEE STANDARD LOGIC SYMBOLS

Together with the American National Standards Institute (ANSI), the Institute of Electrical and Electronic Engineers (IEEE) has developed a standard set of logic symbols. The most recent revision of the standard is ANSI/IEEE Std 91-1984, *IEEE Standard Graphic Symbols for Logic Functions*. The standard allows both rectangular- and distinctive-shape symbols for logic gates. We have been using and will continue to use the distinctive-shape symbols throughout this book, but the rectangular-shape symbols are described in Appendix A.



gate symbols

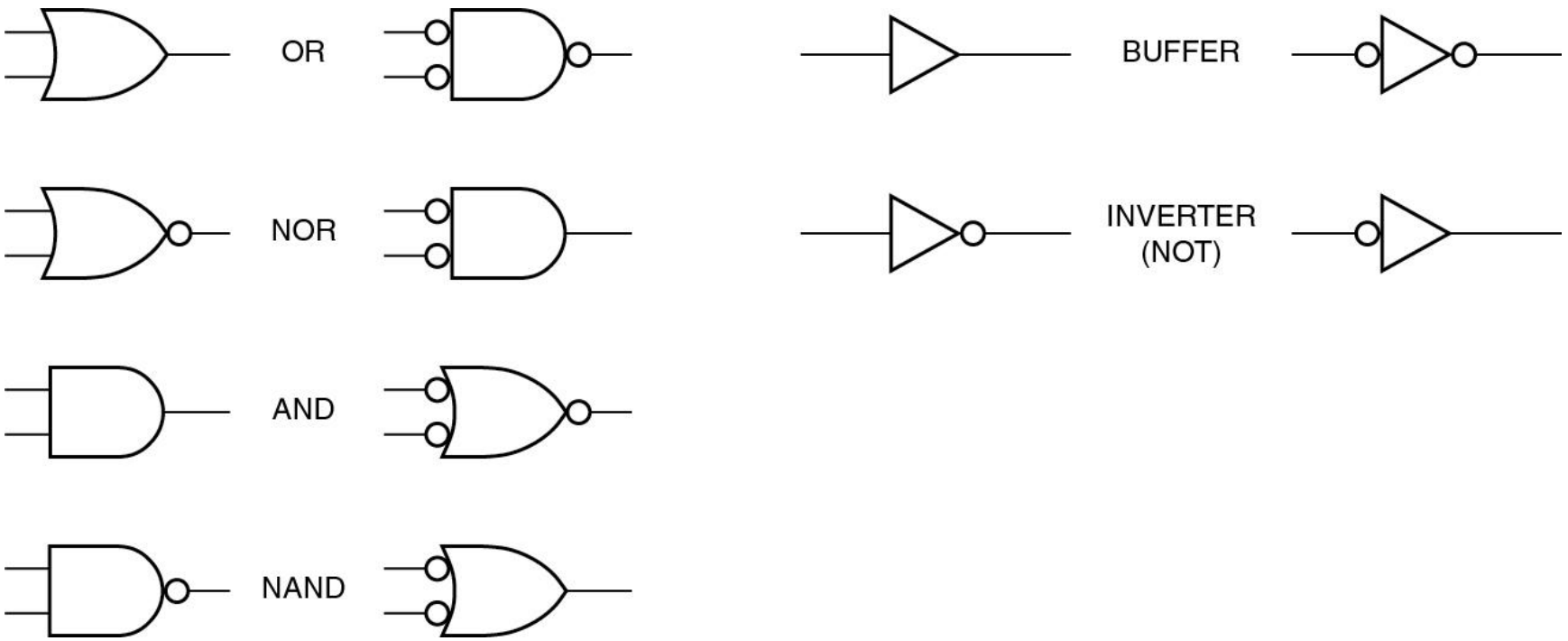


Figure 4-4. Equivalent gate symbols under the De Morgan theorem

signal names, active-high, active-low levels

Active Low	*Active High*
READY–	READY+
ERROR.L	ERROR.H
ADDR15(L)	ADDR15(H)
RESET*	RESET
ENABLE-	ENABLE
-GO	GO
/RECEIVE	RECEIVE
TRANSMIT_L	TRANSMIT

Table 4-1. Each line shows a different naming convention for active levels

active levels for pins

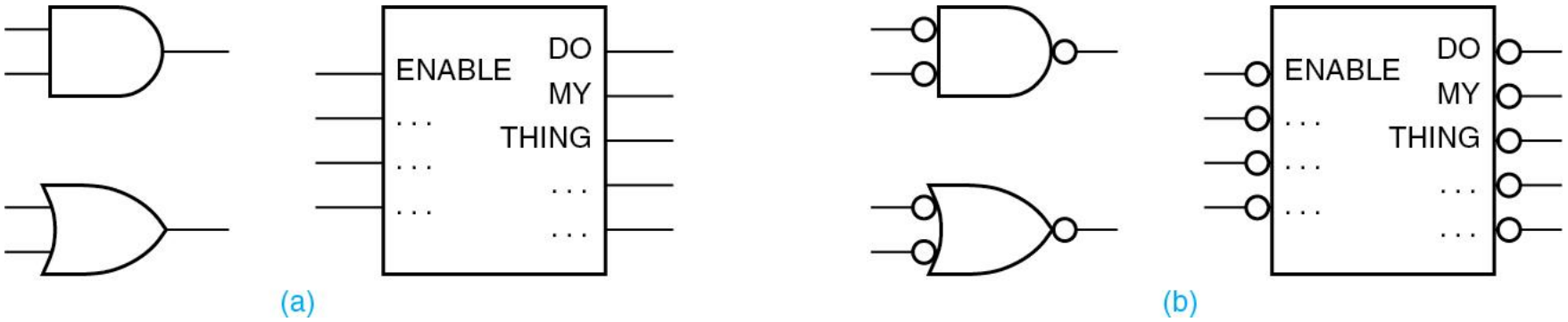


Figure 4-5. Logic symbols.

(a) active-high AND, OR, and a larger-scale logic element

(b) the same elements with active-low inputs and outputs

active levels for pins

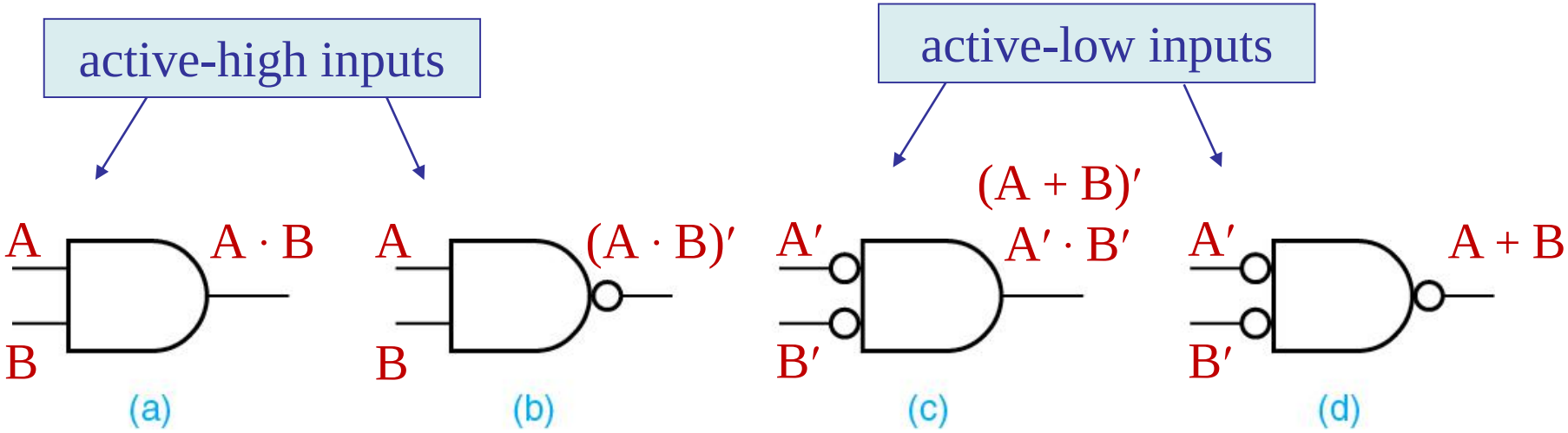


Figure 4-6. Four ways of obtaining an AND function.

(a) AND gate;

(b) NAND gate;

(c) NOR gate;

(d) OR gate

active levels for pins

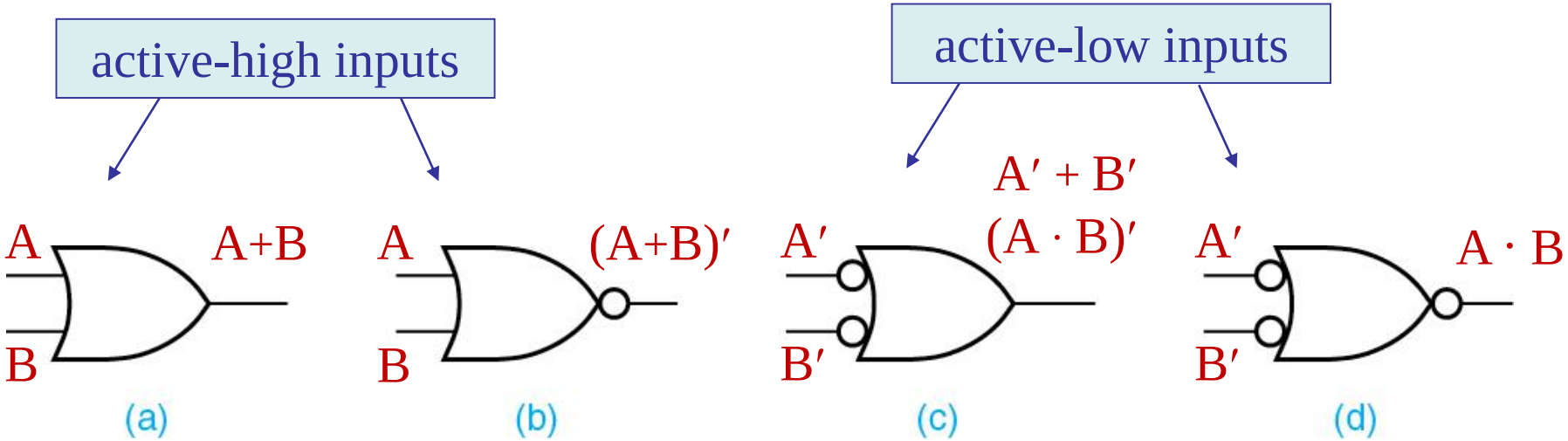


Figure 4-6. Four ways of obtaining an OR function.

(a) OR gate;

(b) NOR gate;

(c) NAND gate;

(d) AND gate

active levels for pins

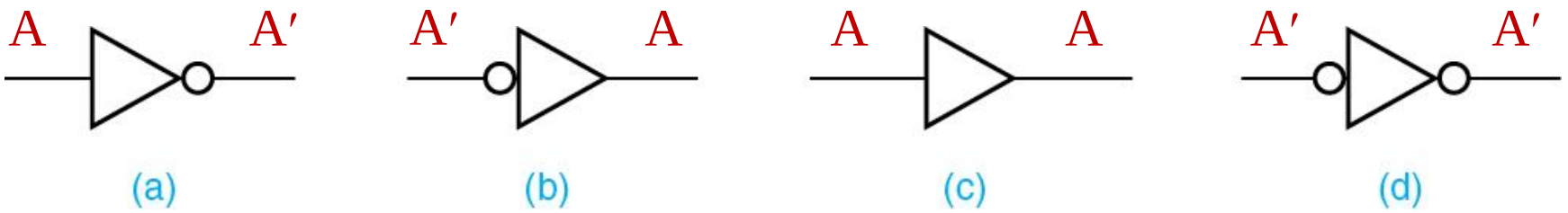


Figure 4-8. Alternative symbols.

(a, b) Inverters

(c, d) Noninverting buffers

active levels for pins

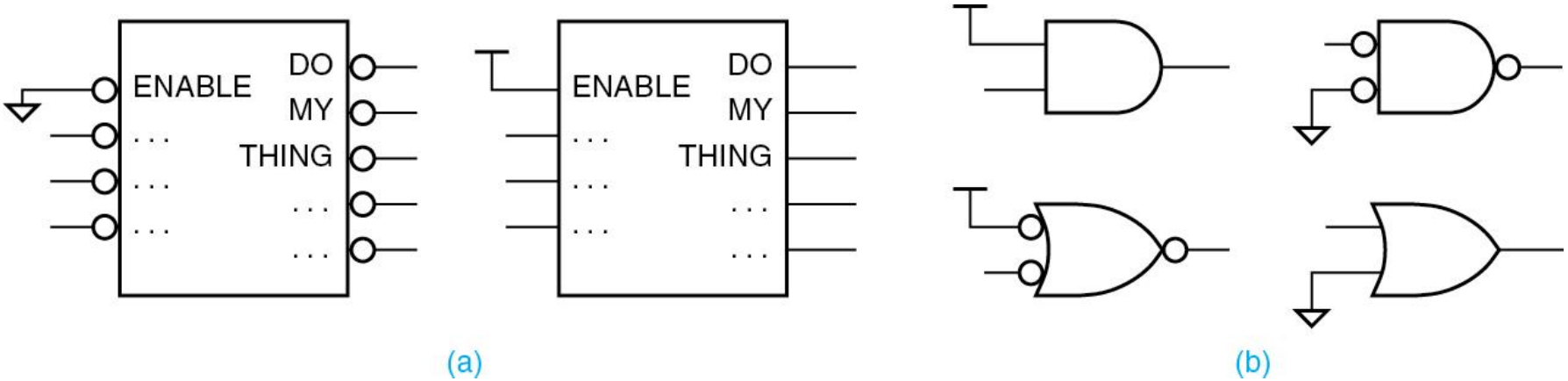


Figure 4-9. Constant 0 and 1 inputs for unused inputs.

(a) with larger-scale logic element

(b) with individual gates

active levels for pins

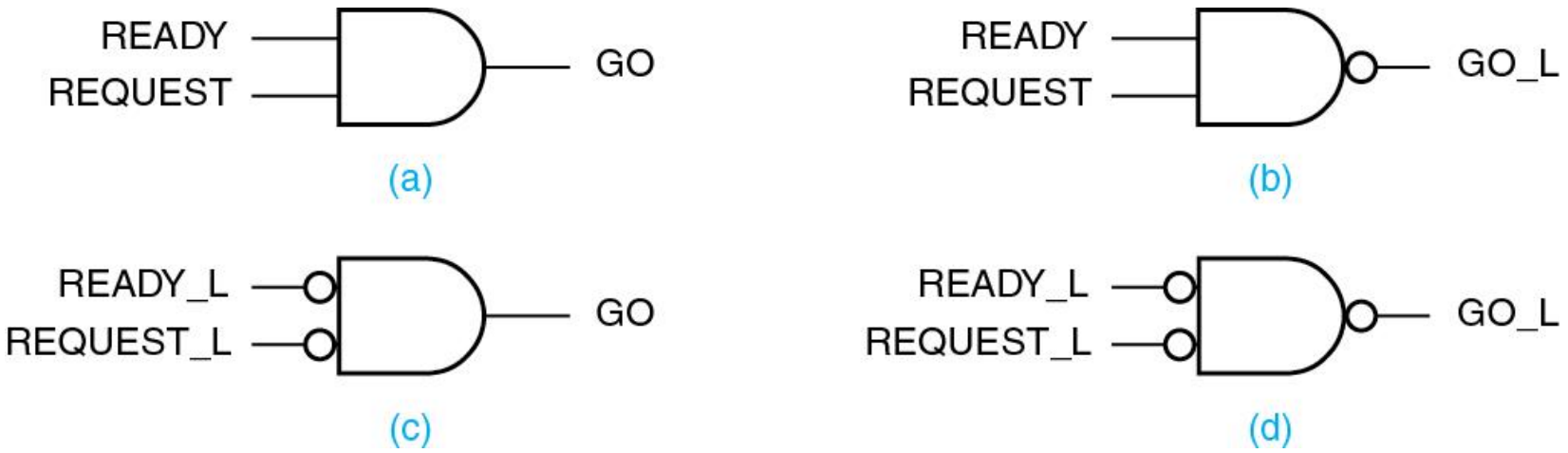


Figure 4-10. Many ways to GO.

(a) Active-high inputs and output

(b) Active-high inputs, active-low output

(c) Active-low inputs, active-high output

(d) Active-low inputs and output

active levels for pins

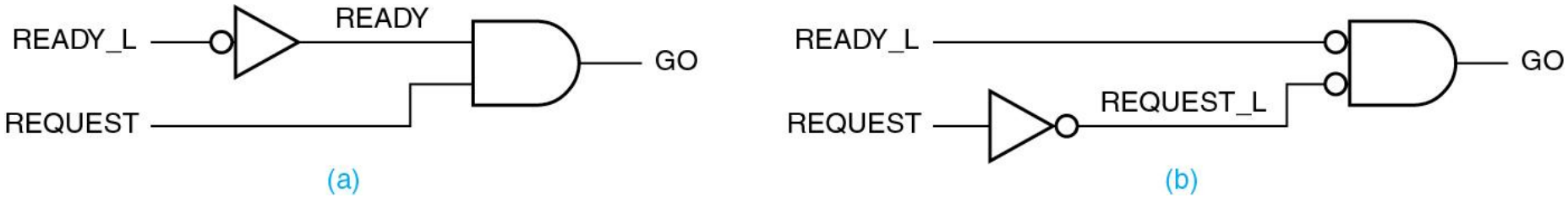


Figure 4-1.1 Two more ways to GO, with mixed input levels.

(a) with an AND gate

(b) with a NOR gate

bubble-to-bubble logic design

BUBBLE-TO-BUBBLE LOGIC DESIGN RULES

The following rules are useful for performing bubble-to-bubble logic design:

- The signal name on a device's output should have the same active level as the device's output pin, that is, active-low if the device symbol has an inversion bubble on the output pin, active-high if not.
- If the active level of an input signal is the same as that of the input pin to which it is connected, then the logic function inside the symbolic outline is activated when the signal is asserted. This is the most common case in a logic diagram.
- If the active level of an input signal is the opposite of that of the input pin to which it is connected, then the logic function inside the symbolic outline is activated when the signal is negated. This case should be avoided whenever possible because it forces us to keep track mentally of a logical negation to understand the circuit.

bubble-to-bubble logic design

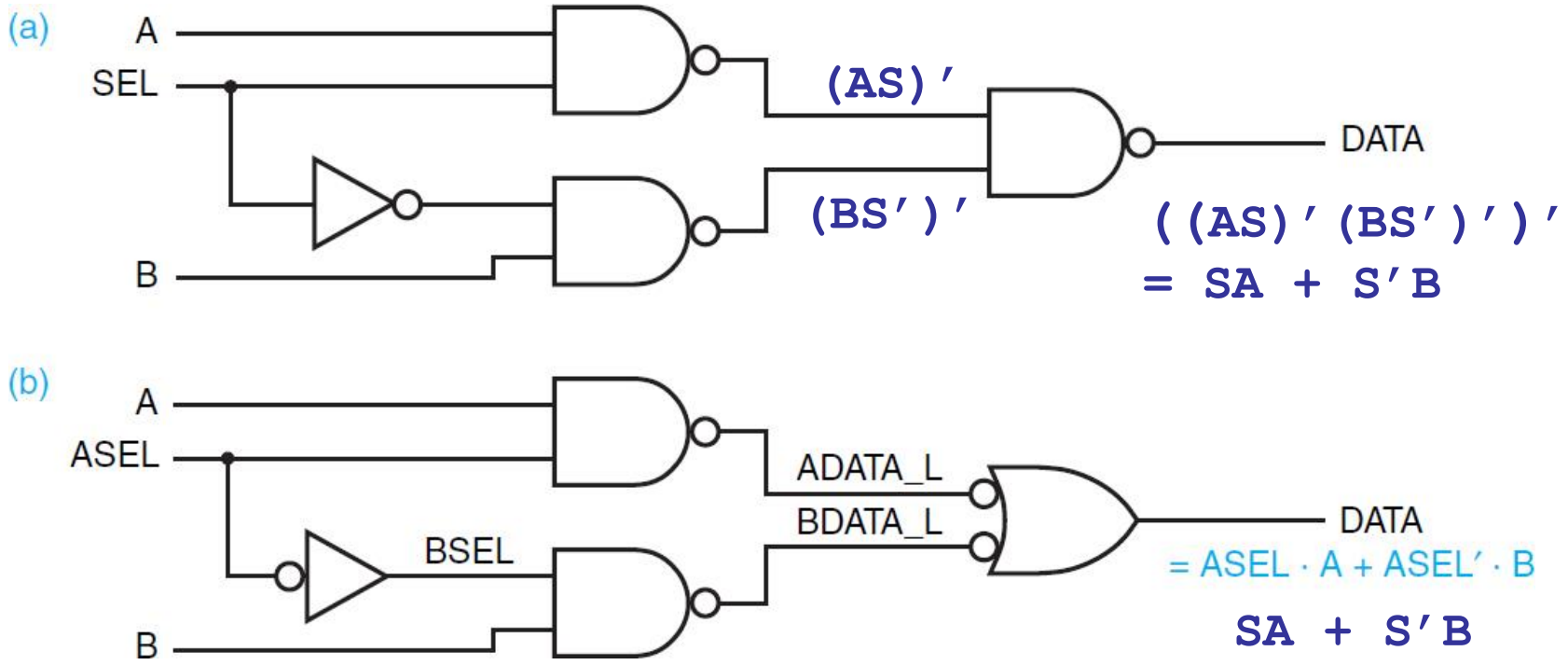


Figure 4-12. Two-Input Multiplexer.

(a) cryptic logic diagram

(b) proper logic diagram with named active levels

bubble-to-bubble logic design

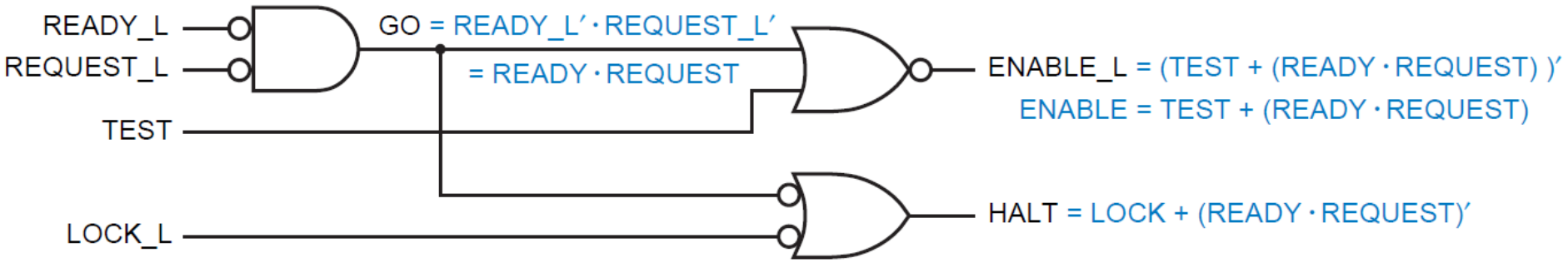


Figure 4-13. Another properly drawn logic diagram

READY = (READY_L)′

REQUEST = (REQUEST_L)′

LOCK = (LOCK_L)′

ENABLE = (ENABLE_L)′

layouts and schematics

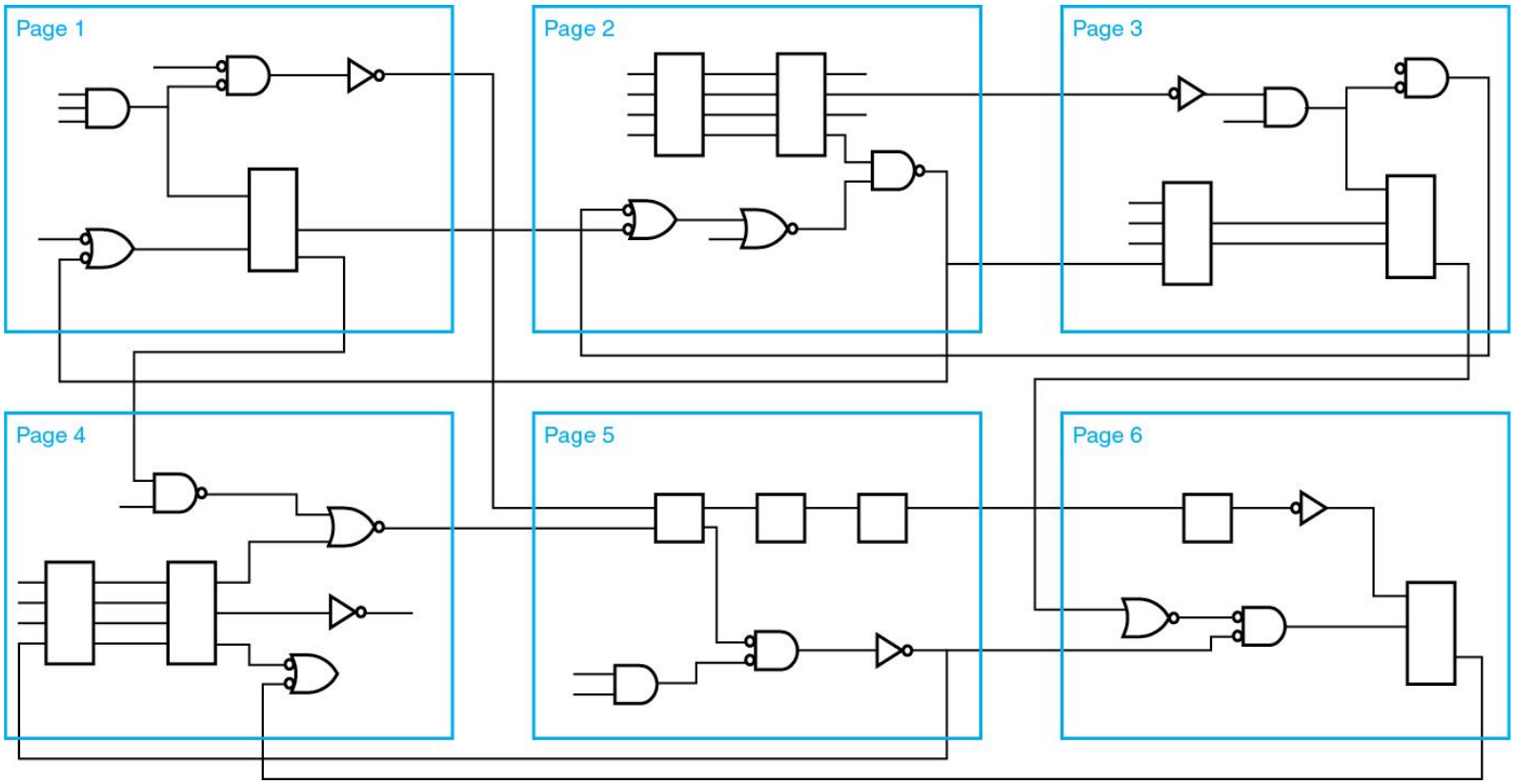


Figure 4-15. Flat schematic structure

layouts and schematics

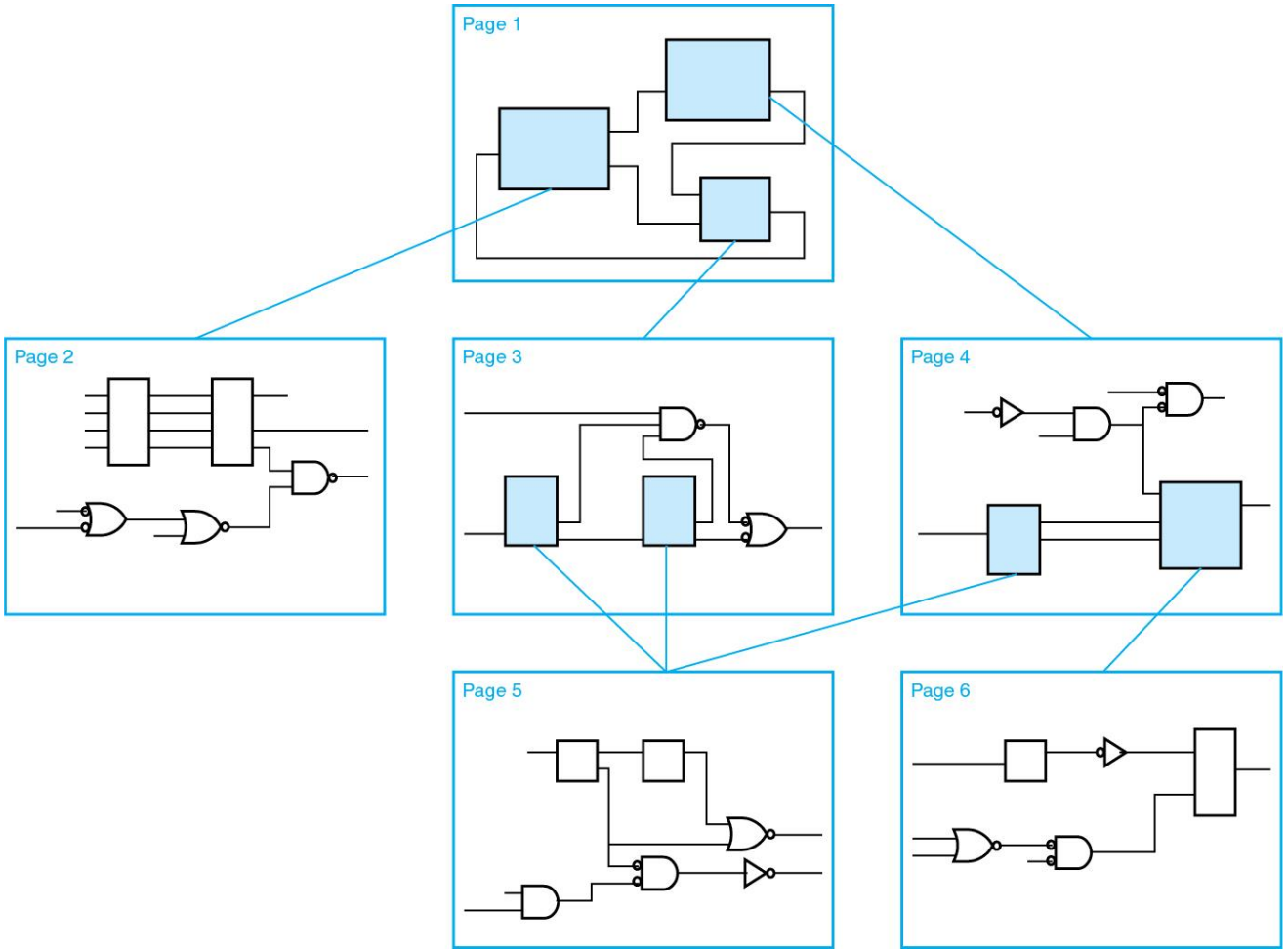


Figure 4-16. Hierarchical schematic structure

layouts and schematics

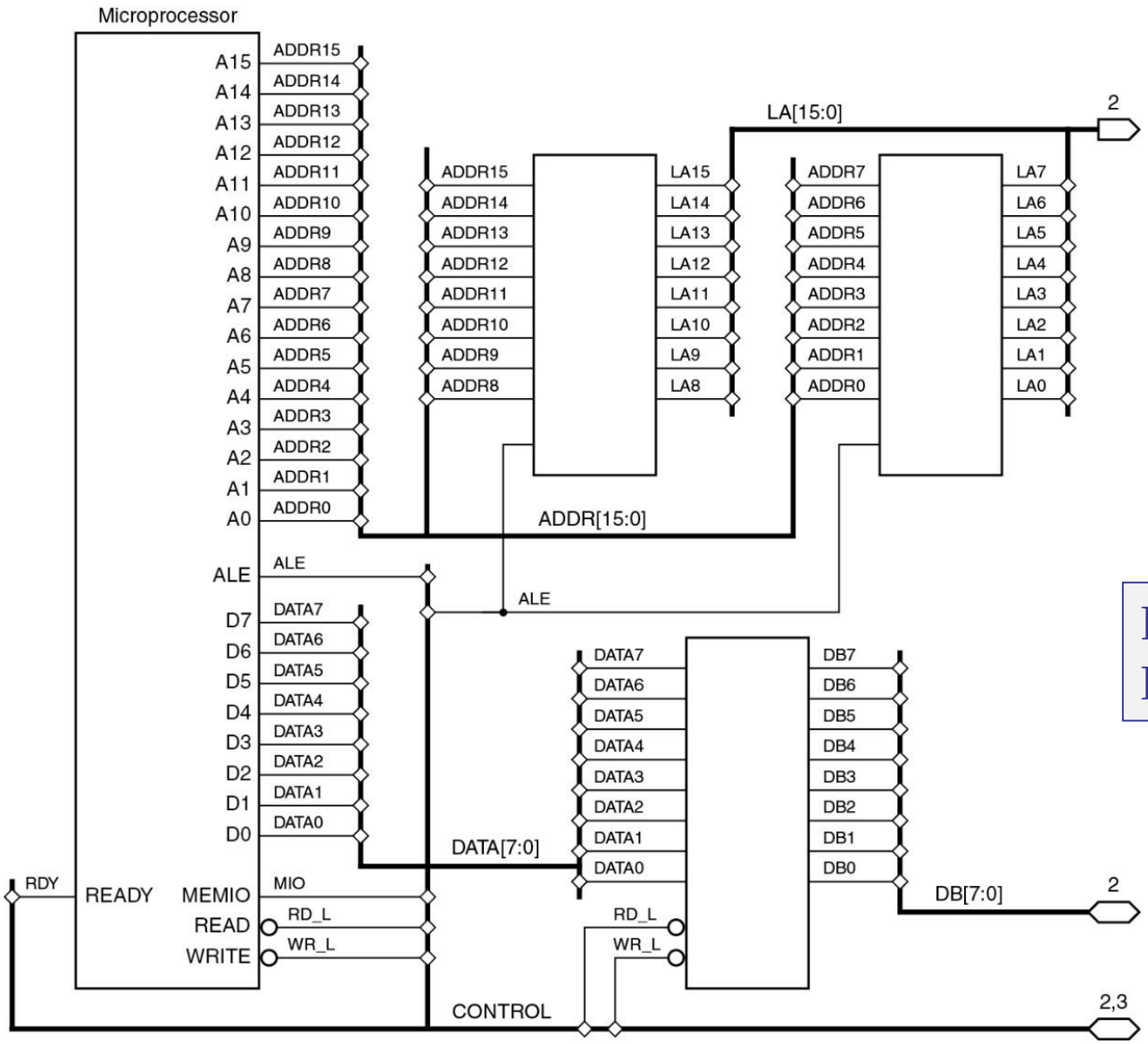


Figure 4-17
Examples of buses

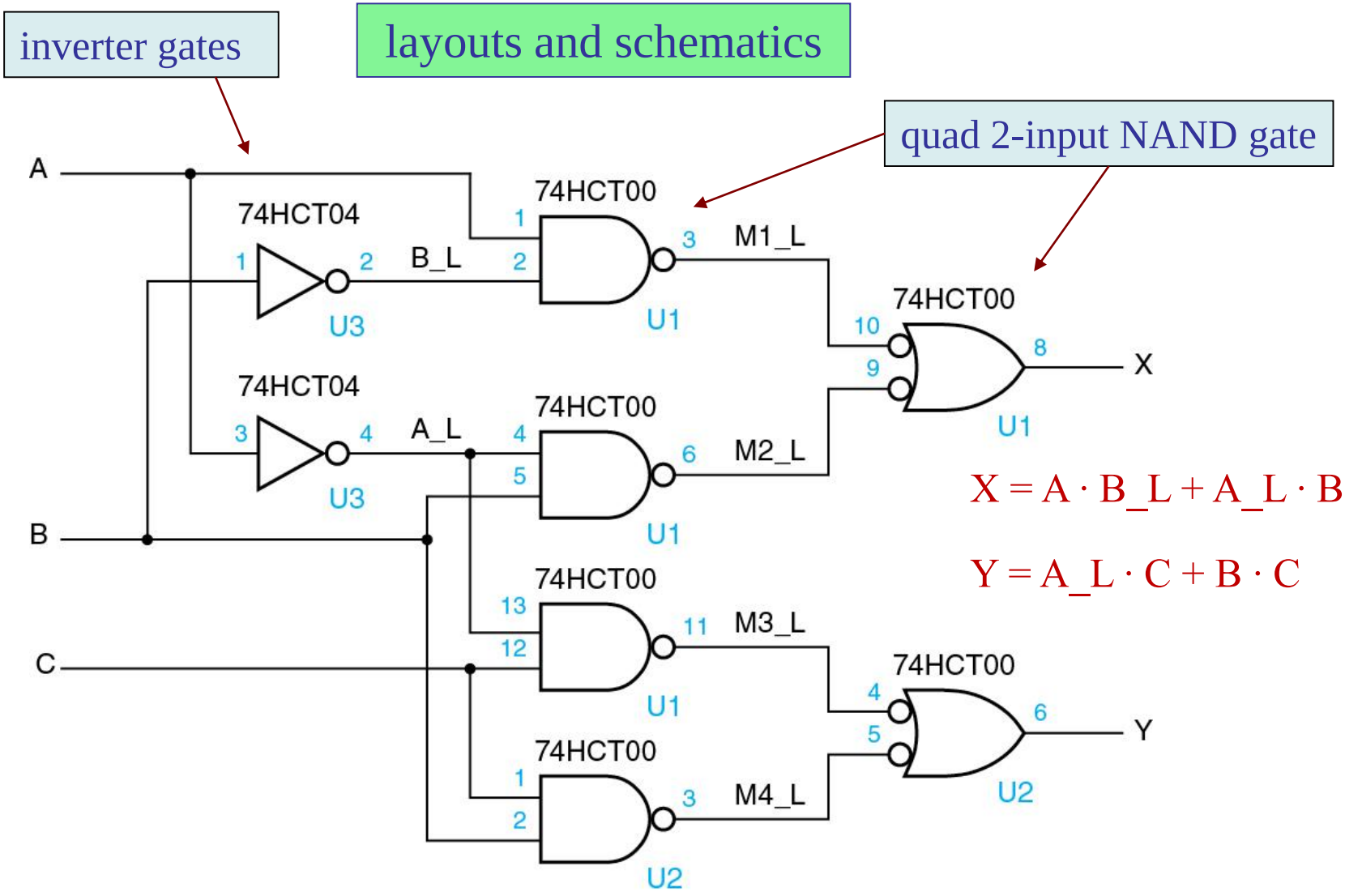


Figure 4-18. Schematic diagram for a circuit using several SSI parts

circuit timing

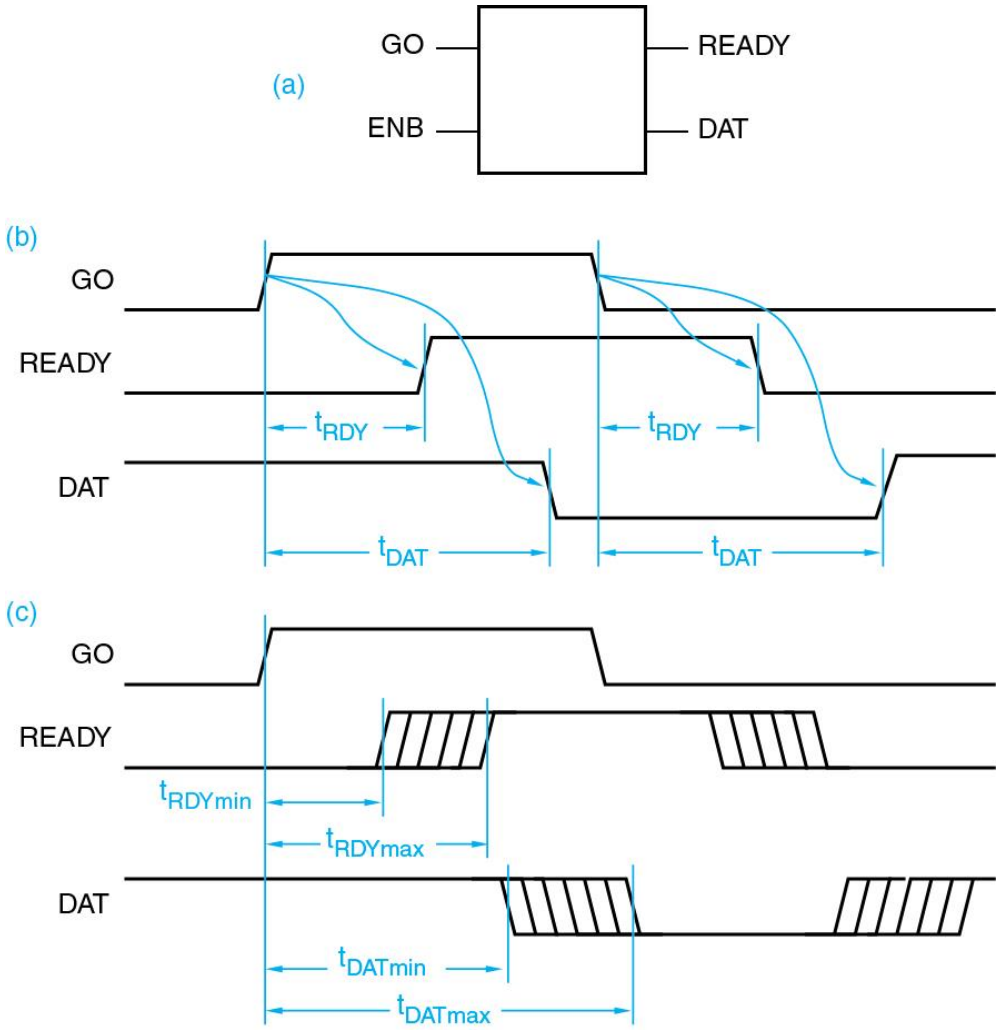


Figure 4-19. Timing diagrams for a combinational circuit.
(a) Block diagram of circuit
(b) Causality, propagation delay
(c) Minimum and maximum delays

Table 4-2. Propagation delay in nanoseconds of selected CMOS SSI parts

circuit timing

Part Number	Function	74AC @ 5.0V				74HC @ 2.0V			74HC @ 4.5V		
		Minimum		Maximum		Typ.	Maximum		Typ.	Maximum	
		t_{pLH}	t_{pHL}	t_{pLH}	t_{pHL}	25°C t_{pd}	25°C t_{pd}	85°C t_{pd}	25°C t_{pd}	25°C t_{pd}	85°C t_{pd}
'00	2-input NAND	1.9	1.9	6.6	6.6	45	90	115	9	18	23
'02	2-input NOR	3.0	3.0	10.4	10.4	45	90	115	9	18	23
'04	Inverter	1.7	1.7	5.9	5.9	45	95	120	9	19	24
'08	2-input AND	1.0	1.0	8.5	7.5	50	100	125	10	20	25
'10	3-input NAND	1.0	1.0	8.0	6.5	35	95	120	10	19	24
'11	3-input AND	1.0	1.0	8.5	7.5	35	100	125	10	20	25
'20	4-input NAND	1.5	1.5	8.0	7.0	45	110	140	14	22	28
'21	4-input AND	1.5	1.5	6.5	7.0	44	110	140	14	22	28
'27	3-input NOR	1.5	1.5	8.5	8.5	35	90	115	10	18	23
'30	8-input NAND	1.0	1.0	9.5	9.5	41	130	165	15	26	33
'32	2-input OR	1.5	1.0	10.0	9.0	50	100	125	10	20	25
'86	2-input XOR	1.0	1.0	9.0	9.5	40	100	125	12	20	25

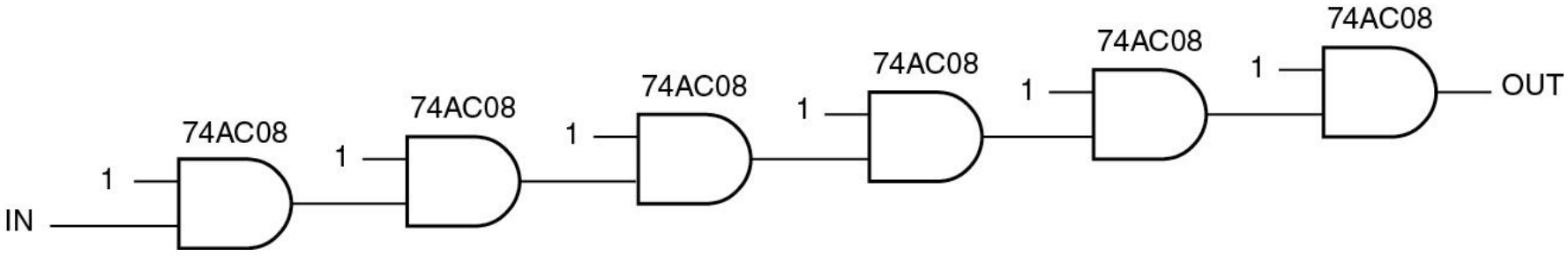
circuit timing

Table 4-3. Propagation delay in nanoseconds of selected CMOS MSI parts

				74AC @ 5.0V		74HC @ 2.0V			74HC @ 4.5V		
				Min.	Max.	Typ.	Maximum		Typ.	Maximum	
Part	Function	From	To	25°C t_{pd}	25°C t_{pd}	25°C t_{pd}	25°C t_{pd}	85°C t_{pd}	25°C t_{pd}	25°C t_{pd}	85°C t_{pd}
'138	3-to-8 binary decoder	any select	output	2.8	10.0	67	180	225	18	36	45
		$\overline{G2A}$, $\overline{G2B}$	output	2.6	9.1	66	155	195	18	31	39
		G1	output	2.8	10.0	66	155	195	18	31	39
'139	dual 2-to-4 binary decoder	any select	output	2.8	9.5	47	175	220	14	35	44
		enable	output	2.8	9.5	39	175	220	11	35	44
'148	8-to-3 priority encoder	$\overline{I1}$-$\overline{I7}$	$\overline{A0}$-$\overline{A2}$			69	180	225	23	36	45
		$\overline{I0}$-$\overline{I7}$	$\overline{EO}$			60	150	190	20	30	38
		$\overline{I0}$-$\overline{I7}$	$\overline{GS}$			75	190	240	25	38	48
		$\overline{EI}$	$\overline{A0}$-$\overline{A2}$			78	195	245	26	39	49
		$\overline{EI}$	$\overline{GS}$			57	145	180	19	29	36
		$\overline{EI}$	$\overline{EO}$			66	165	205	22	33	41
'151	8-to-1 multiplexer	any select	Y	4.7	16.5	94	250	312	30	50	63
		any select	$\overline{Y}$	5.1	17.8	94	250	312	30	50	63
		any data	Y	3.5	12.3	74	195	244	23	39	49
		any data	$\overline{Y}$	3.8	13.5	74	195	244	23	39	49
		enable	Y	3.1	11.1	49	127	159	15	25	32
		enable	$\overline{Y}$	3.5	12.3	49	127	159	15	25	32
'157	2-to-4 multiplexer	select	output	3.8	13.2		145	180	12	29	36
		any data	output	2.2	7.7		125	155	10	25	31
		enable	output	3.6	12.3		135	170	11	27	34
'280	9-input parity circuit	any input	EVEN	5.2	18.2		200	250	17	40	50
		any input	ODD	5.4	19.1		200	250	17	40	50
'283	4-bit adder	C0	any Si	4.5	16.0		230	290	19	46	58
		any Ai, Bi	any Si	4.7	16.5		210	265	18	42	53
		any input	C4	4.5	16.0		195	245	16	39	49
'682	8-bit comp.	any input	output			130	275	344	26	55	69

Figure X4.11

circuit timing



4.11 Determine the exact maximum propagation delay from IN to OUT of the circuit fragment in **Figure X4.11** for both LOW-to-HIGH and HIGH-to-LOW transitions, using the timing information given in **Table 4-2**. Repeat, using a single worst-case delay number for each gate, and compare and comment on your results.

solution

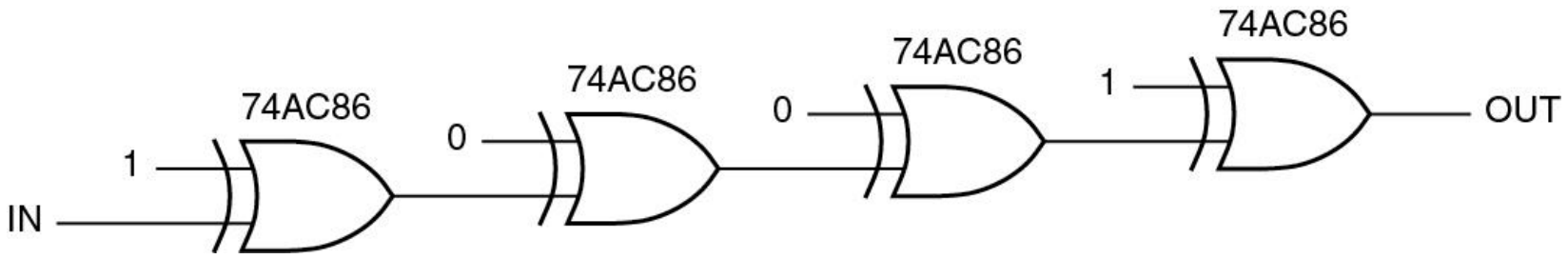
All transitions go in the same direction. Thus,

$$t_{pLH} = 6t_{pLH(AC08)}$$
$$= 6 \cdot 8.5 = 51 \text{ ns}$$
$$t_{pHL} = 6t_{pHL(AC08)}$$
$$= 6 \cdot 7.5 = 45 \text{ ns}$$

Without inverters to "flip" the transition polarity at each stage, the asymmetry between t_{pLH} and t_{pHL} grows with the number of stages. A worst-case calculation yields 51 ns, which is in fact achieved in LOW-to-HIGH transitions.

Figure X4.15

circuit timing



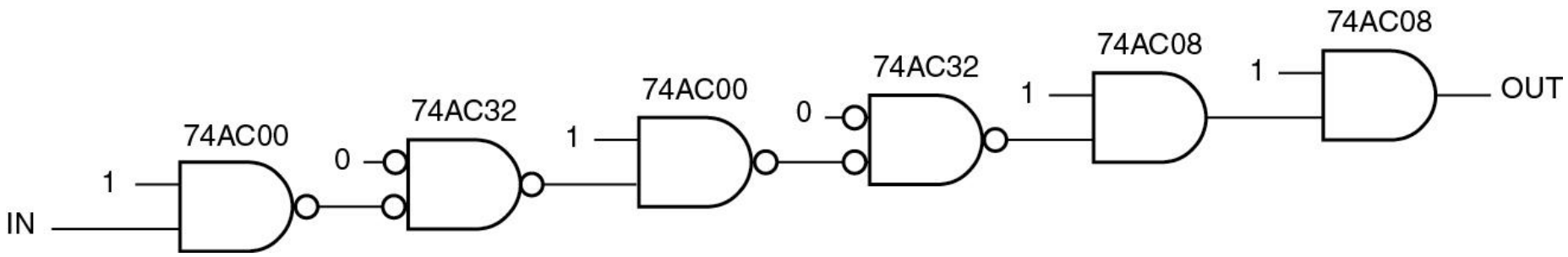
4.15 Estimate the *minimum* propagation delay from IN to OUT for the circuit shown in **Figure X4.15**. Justify your answer.

solution

The minimum delay through one ’AC86 is listed in Table 4–2 as 1.0 ns. Therefore, we estimate the minimum delay through the four gates at 4 ns.

Figure X4.18

circuit timing



4.18 Estimate the *minimum* propagation delay from IN to OUT for the circuit shown in **Figure X4.18**. Justify your answer.

solution

The minimum delays through the ’AC00, ’AC08, and ’AC32 are listed in Table 4–2 as 1.9, 1.0, and 1.5/1.0 (LH/HL) respectively. So, the total minimum delay will be 1.9+1.5/1.0+1.9+1.5/1.0+1.0+1.0 depending on the transition directions through the ’AC32s. However, since there is exactly one inversion between the ’AC32s, the transitions will be in opposite directions, so one will have a minimum delay of 1.5 ns and the other 1.0 ns. The total minimum delay is therefore 1.9+1.5+1.9+1.0+1.0+1.0 = 8.3 ns.

Part 2 – Basic Combinational Components

(Wakerly, Ch. 6 & 7)

Read-only-memories (ROMs), and realizations of combinational functions

Decoders

Realizing arbitrary combinational functions with decoders

Encoders

Three-state buffers

Priority encoders

Multiplexers

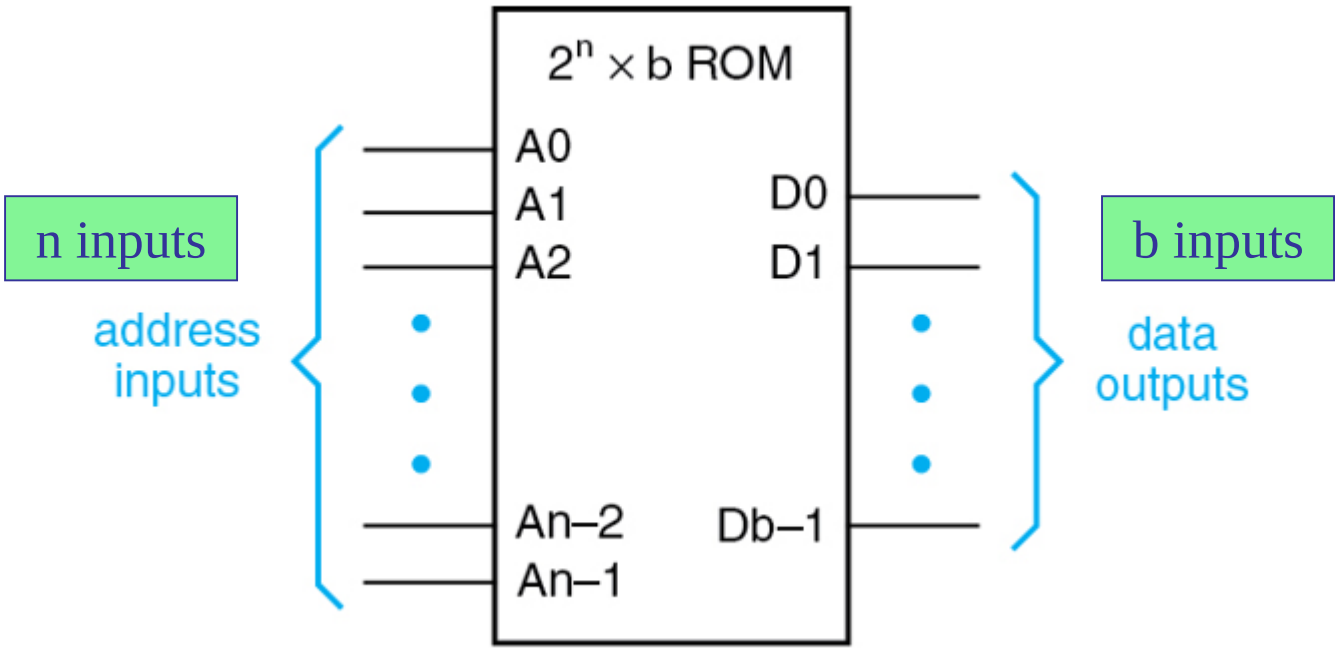
Realizing arbitrary combinational functions with multiplexers

Demultiplexers

Realizing multiplexers and demultiplexers with decoders

ROMs

A read-only-memory (ROM) is a combinational circuit with n address inputs and b data outputs, so that there are 2^n input bit patterns.



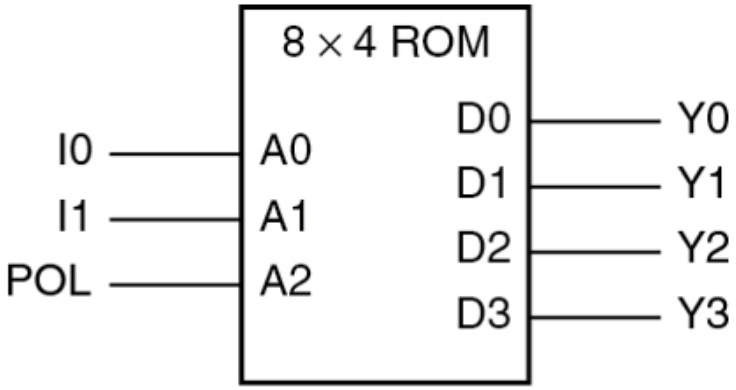
Wakerly Figure 6-14. 2^n x b ROM

ROMs

A ROM may be thought of as a "look-up" table for storing the truth table of an arbitrary combinational circuit that has n inputs and b outputs.

Wakerly Table 6-1. For example, the truth table of a 2-to-4 decoder with an additional output-polarity control input, shown below, can be stored in a 2^3 x 4 or 8x4 ROM

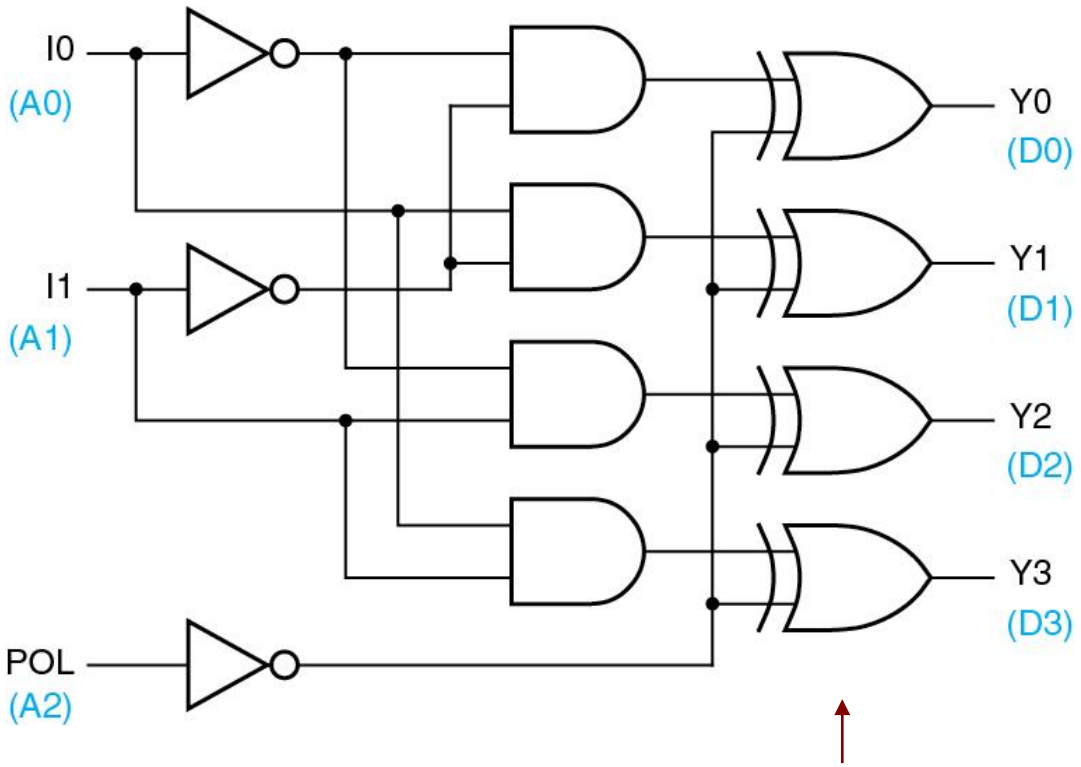
A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
POL	I_1	I_0	D_3	D_2	D_1	D_0



gate-level implementation →

Wakerly Fig. 6-3. Gate-level implementation of a 2-to-4 decoder with output-polarity control (see p.54 for derivations)

A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



$$Y_0 = A_2' \oplus (A_1'A_0')$$

$$Y_1 = A_2' \oplus (A_1'A_0)$$

$$Y_2 = A_2' \oplus (A_1A_0')$$

$$Y_3 = A_2' \oplus (A_1A_0)$$

recall the XOR properties

$0 \oplus Y = Y$

$1 \oplus Y = Y'$

Decoders

A decoder detects a particular code or bit pattern on its input and passes (decodes) that information to its output.

examples to be discussed

1. BCD to 7-segment display
2. n-to-2^n binary decoder
3. 2^n-to-n binary encoder
4. Priority encoders

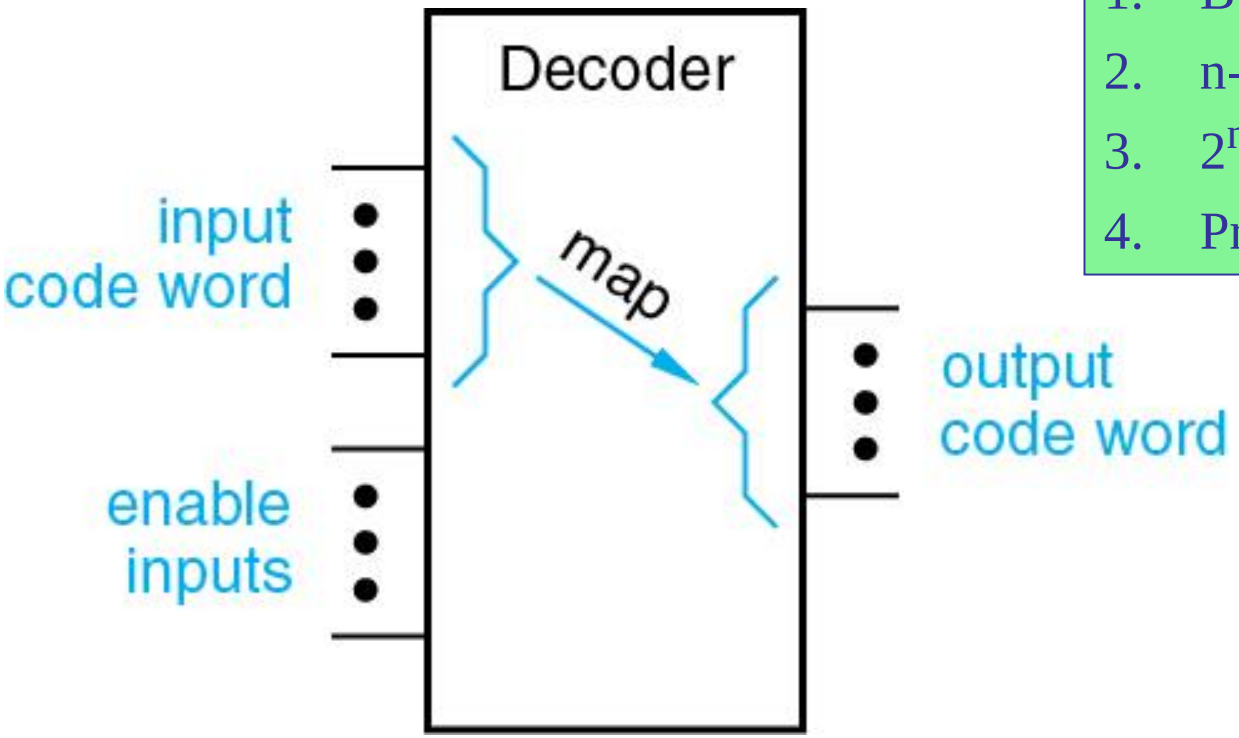
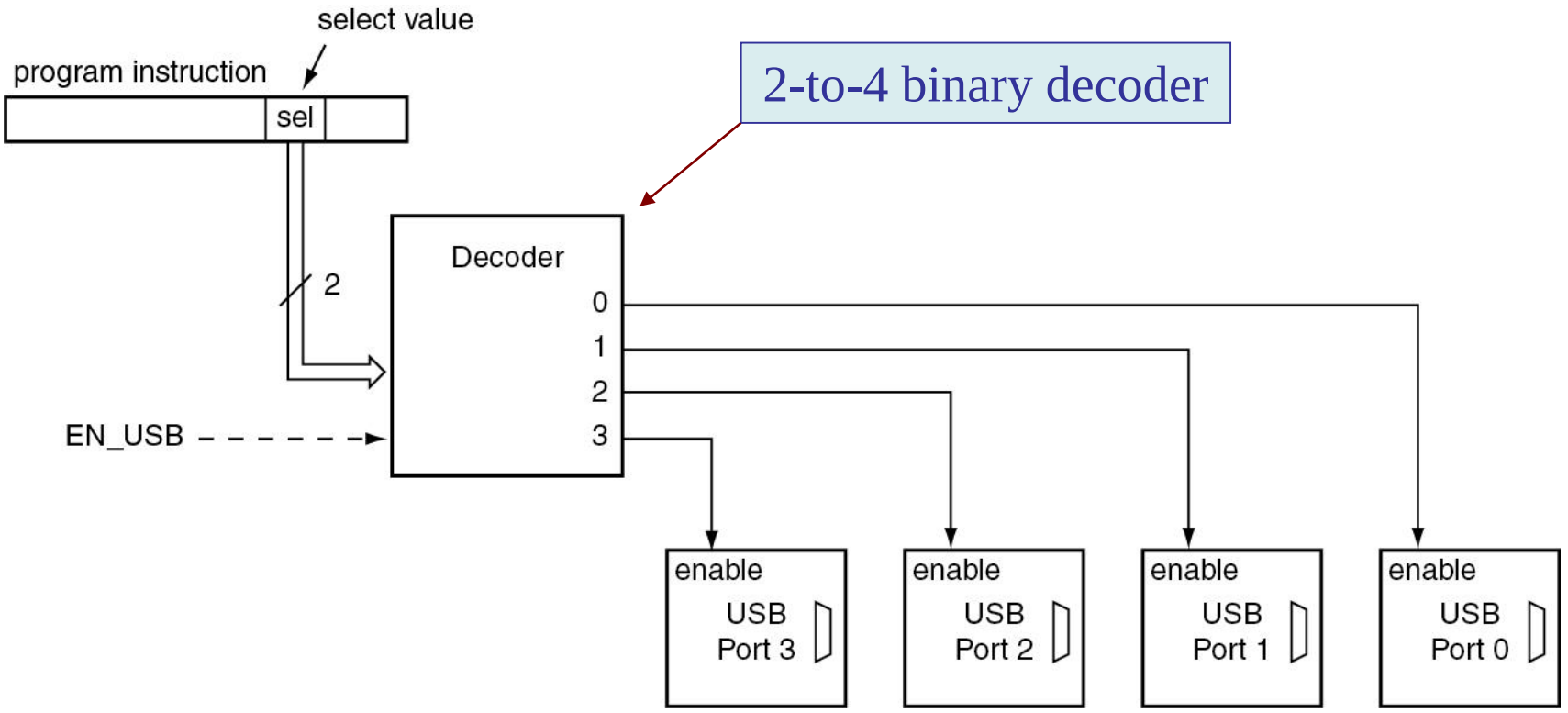


Figure 6-14. Decoder circuit structure

Some decoder application examples

Decoders



2-to-4 binary decoder

Wakerly / Fig.6-12 – Typical decoder application in a computer

BCD to seven-segment LED display decoder

Decoders

binary-coded decimal

light-emitting diode

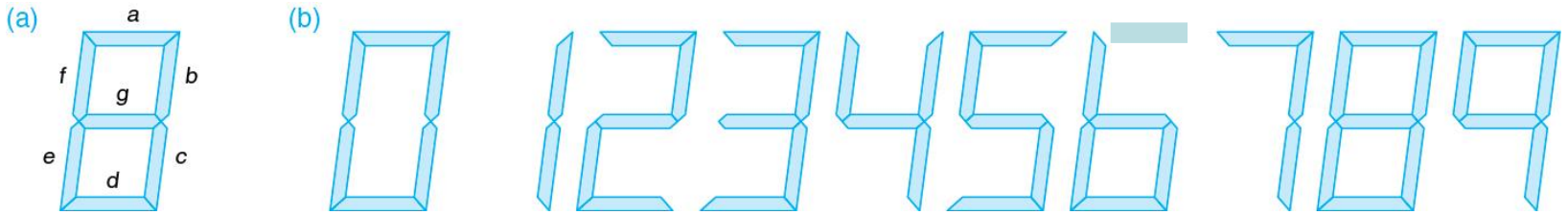
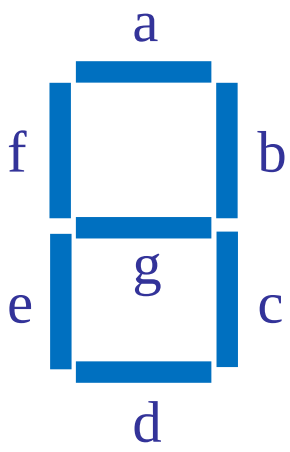


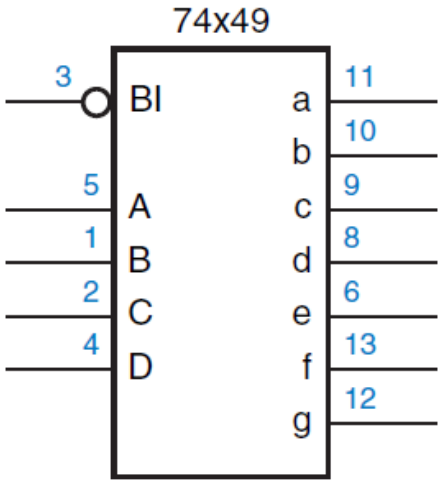
Figure 6-23 Seven-segment LED display.
(a) segment identification
(b) decimal digits

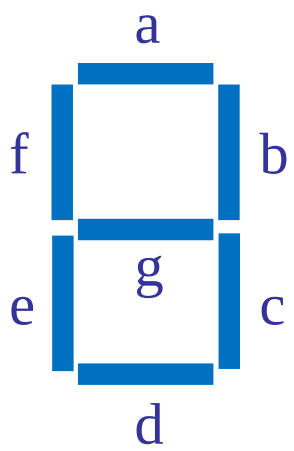


applications: digital clocks & watches
calculator & appliance displays
digital instrumentation displays
digital counters, kitchen timers

Decoders

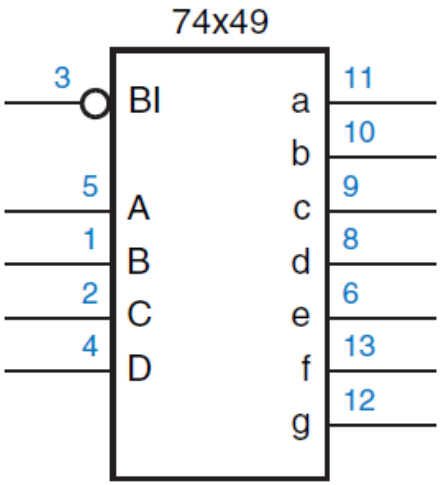
Inputs					Outputs								
BI_L	D (A)	C (B)	B (C)	A (D)		a	b	c	d	e	f	g	
0	x	x	x	x		0	0	0	0	0	0	0	
1	0	0	0	0	0	1	1	1	1	1	1	0	
1	0	0	0	1	1	0	1	1	0	0	0	0	
1	0	0	1	0	2	1	1	0	1	1	0	1	
1	0	0	1	1	3	1	1	1	1	0	0	1	
1	0	1	0	0	4	0	1	1	0	0	1	1	
1	0	1	0	1	5	1	0	1	1	0	1	1	
1	0	1	1	0	6	0	0	1	1	1	1	1	
1	0	1	1	1	7	1	1	1	0	0	0	0	
1	1	0	0	0	8	1	1	1	1	1	1	1	
1	1	0	0	1	9	1	1	1	0	0	1	1	
hex													
A	1	1	0	1	0	0	0	0	1	1	0	1	
b	1	1	0	1	1	0	0	1	1	0	0	1	
C	1	1	1	0	0	0	1	0	0	0	1	1	
d	1	1	1	0	1	1	0	0	1	0	1	1	
E	1	1	1	1	0	0	0	0	1	1	1	1	
F	1	1	1	1	1	0	0	0	0	0	0	0	

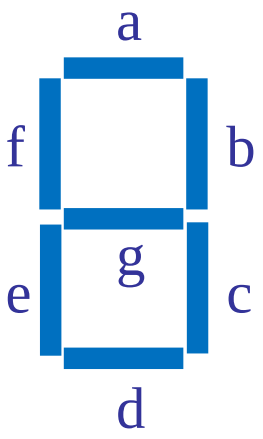




don't care entries, but incorrect for hex

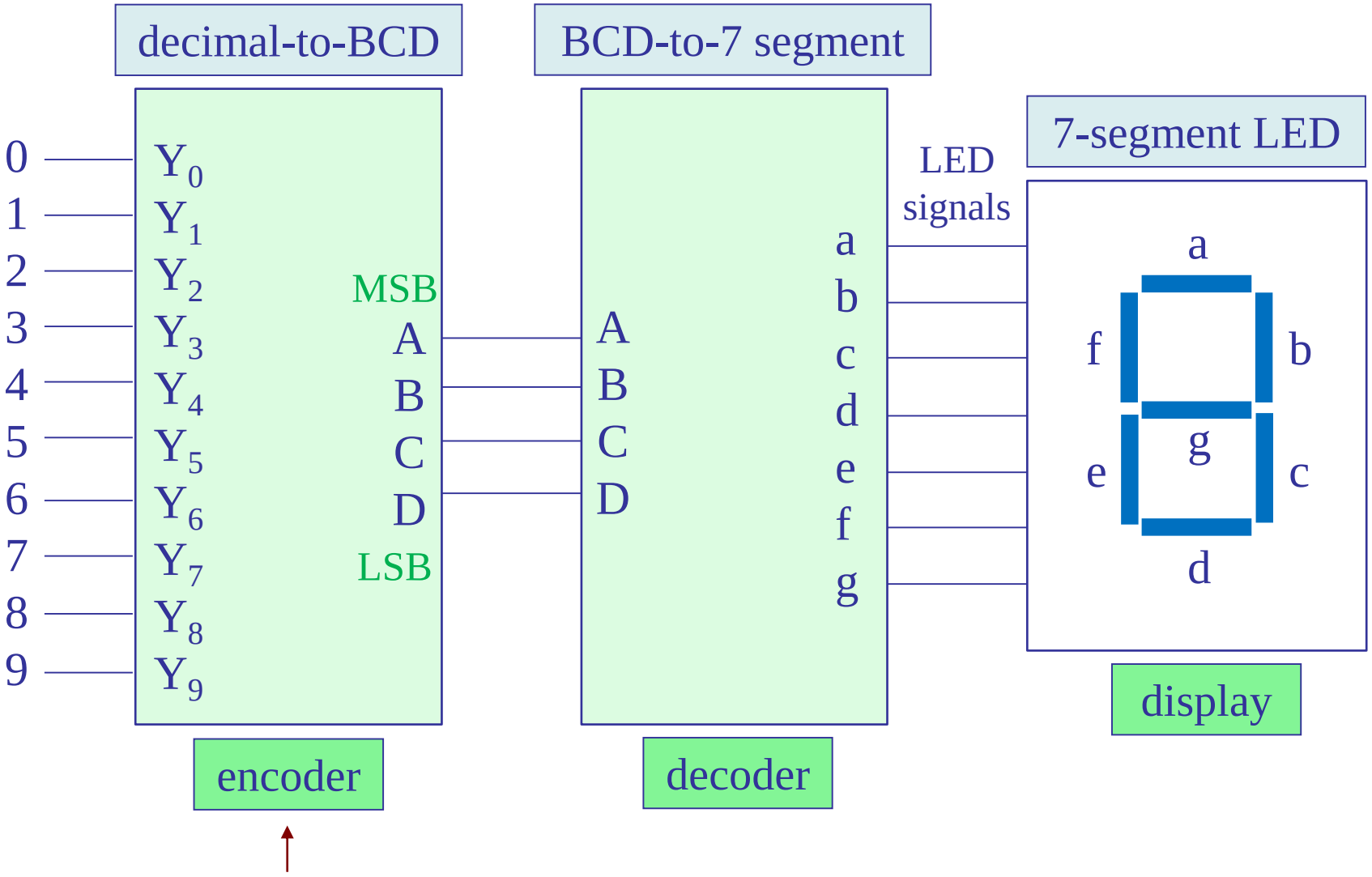
Decoders





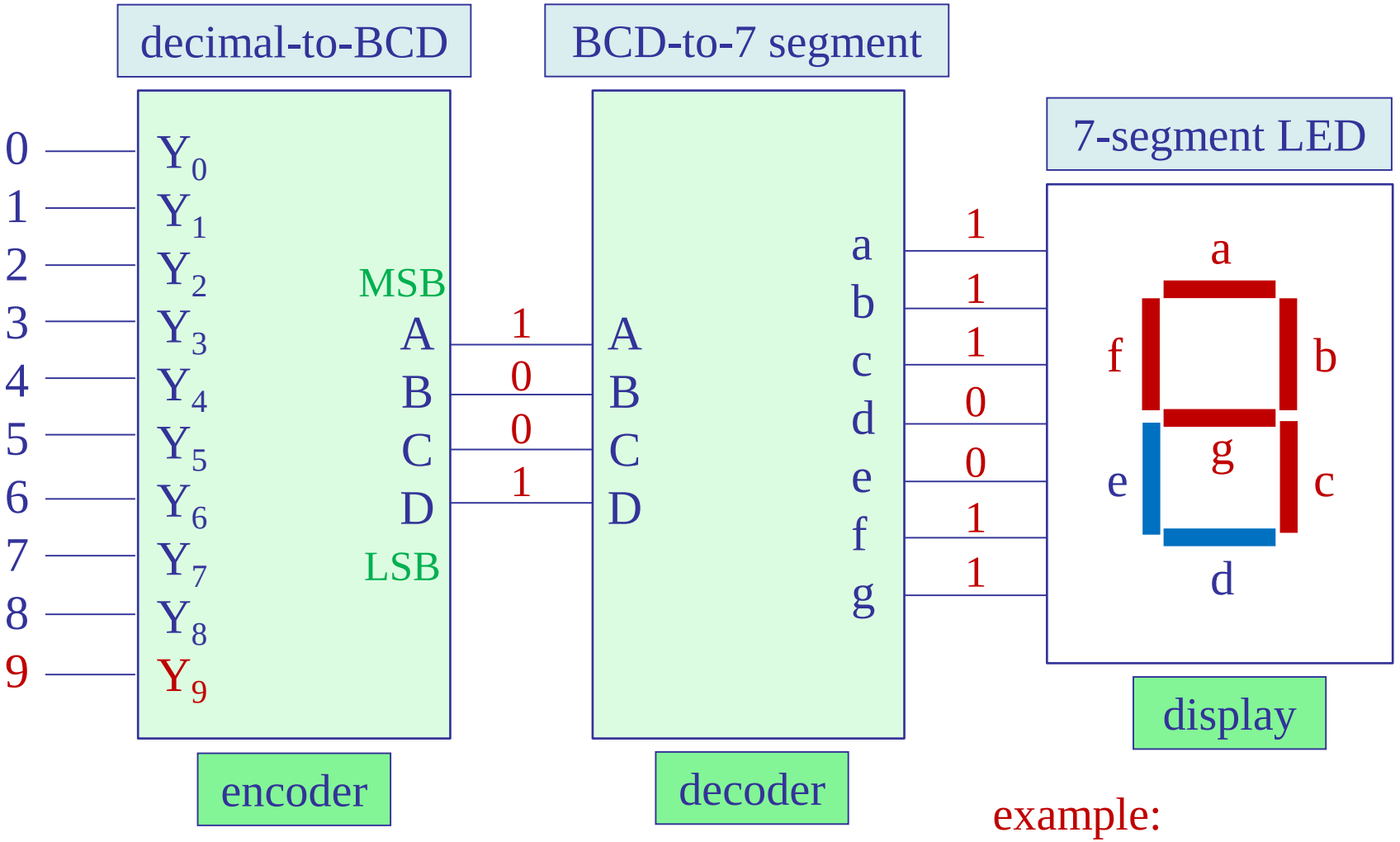
will be explored further in recitation exercises

BCD to seven-segment display encoder/decoder system

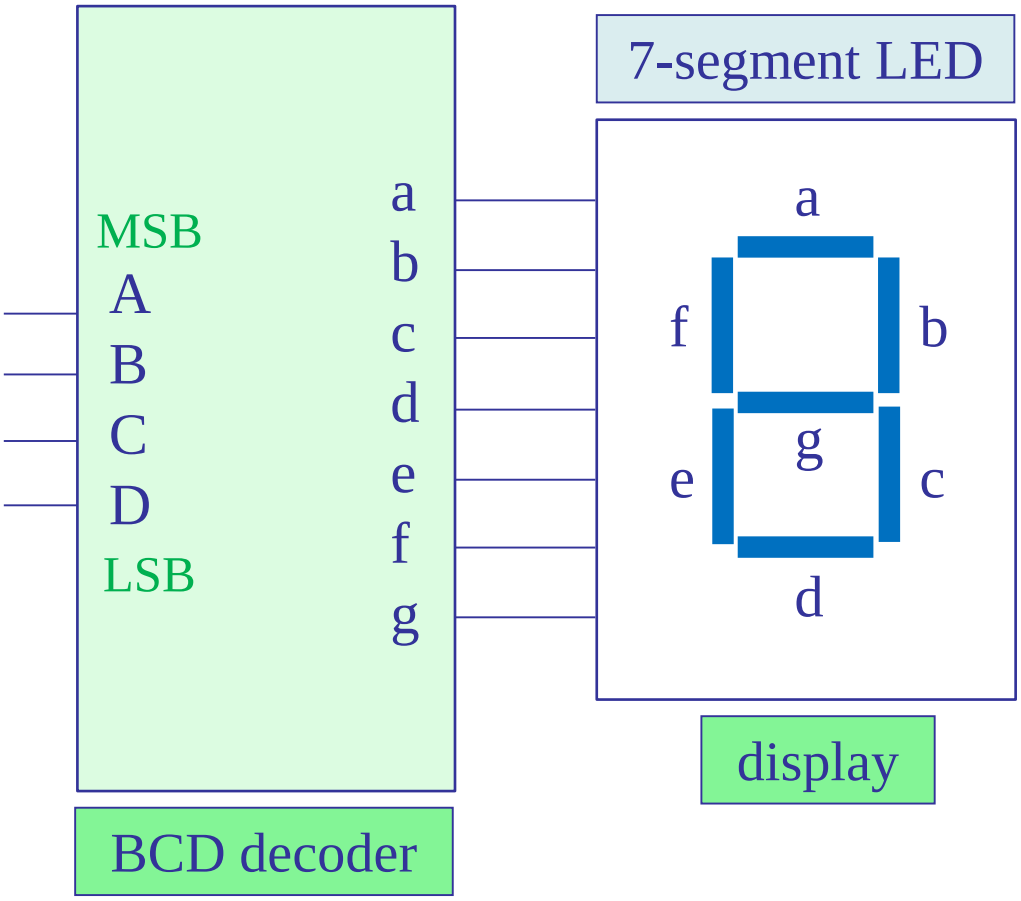


see p.73-79 for details

BCD to seven-segment display encoder/decoder system



example:
displaying the number 9



alternative, d = B′ D′ + CD′ + B′ C + B C′ D + A

see the following link for the case of
[7-segment-hex-decoder]

to be derived in recitations

a = A + C + (B ⊕ D) ′

b = B′ + (C ⊕ D) ′

c = B + C′ + D

d = B′ D′ + CD′ + B′ C + B C′ D

e = B′ D′ + CD′

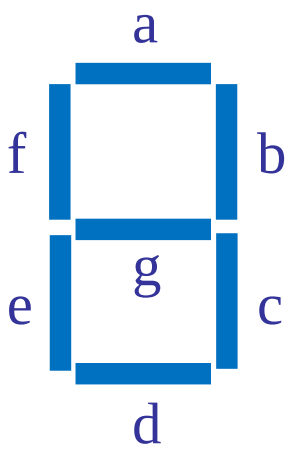
f = A + C′ D′ + B C′ + B D′

g = A + (B ⊕ C) + C D′

BCD decoder truth table

MSB LSB

n	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0	x	x	x	x	x	x	x
11	1	0	1	1	x	x	x	x	x	x	x
12	1	1	0	0	x	x	x	x	x	x	x
13	1	1	0	1	x	x	x	x	x	x	x
14	1	1	1	0	x	x	x	x	x	x	x
15	1	1	1	1	x	x	x	x	x	x	x



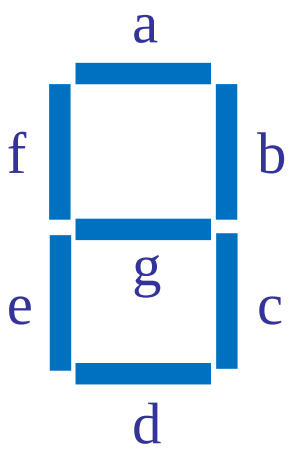
don’t care entries

if 9 has only the e segment off, then use, $d = B' D' + CD' + B' C + B C' D + A$

actual truth table

MSB LSB

n	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
10	1	0	1	0	1	1	0	1	1	1	1
11	1	0	1	1	1	1	1	1	0	1	1
12	1	1	0	0	1	1	1	0	0	1	1
13	1	1	0	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	1	1	1	1	1
15	1	1	1	1	1	1	1	0	0	1	1



why aren't these don't care entries all 1's ?

MATLAB code → and in recitation solutions

$$a = A + C + (B \oplus D)'$$

$$b = B' + (C \oplus D)'$$

$$c = B + C' + D$$

$$d = B'D' + CD' + B'C + BC'D$$

$$e = B'D' + CD'$$

$$f = A + C'D' + BC' + BD'$$

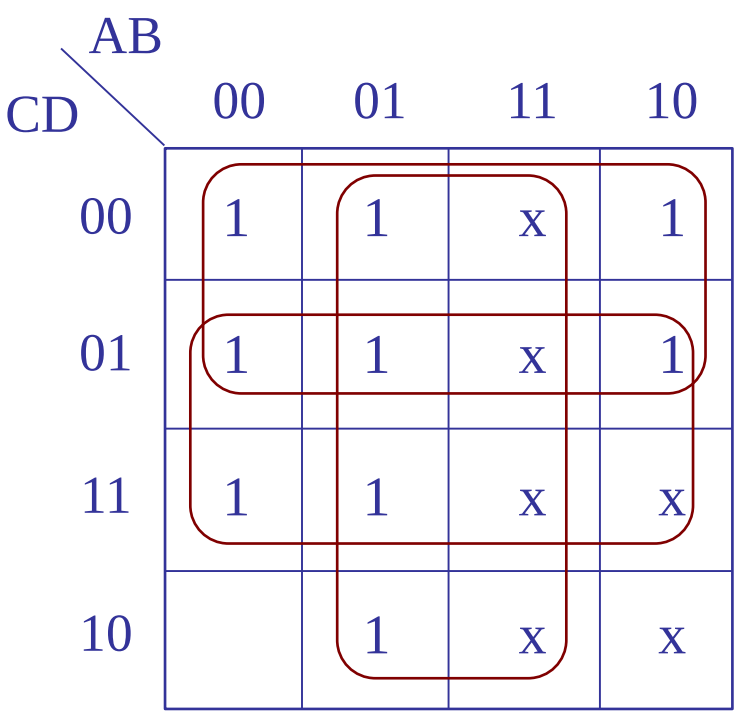
$$g = A + B \oplus C + CD'$$

```
n = (0:15)';

[A,B,C,D] = a2d(n,4);

a = A | C | ~xor(B,D);

b = ~B | ~xor(C,D);

c = B | ~C | D;

d = (~B & ~D) | (C & ~D) | (B & ~C & D) | (~B & C);
% d = (~B & ~D) | (C & ~D) | (B & ~C & D) | (~B & C) | A;

e = (~B & ~D) | (C & ~D);

f = A | (B & ~C) | (~C & ~D) | (B & ~D);

g = A | xor(B,C) | (C & ~D);

[n, A, B, C, D, a, b, c, d, e, f, g]        % truth table
```

alternative version for representing the digit 9

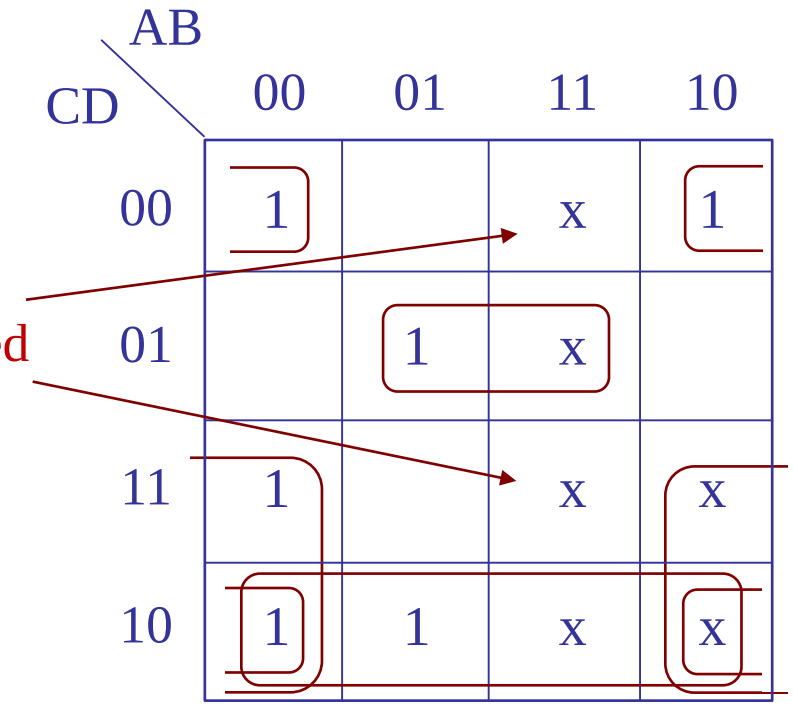
Why aren't all don't care entries equal to 1 even though we take them as 1 in simplifying the K-maps?

Answer: Because in cases (b, c, d, e) not all of the don't care entries were used in the simplification, whereas in cases (a, f, g), all of them were used and they are indeed equal to 1 in the computed truth table



c = B + C′ + D

not used



d = B′ D′ + CD′ + B′ C + BC′ D

n-to-2^n binary decoders

Decoders

called one-hot encoding

in		out			
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

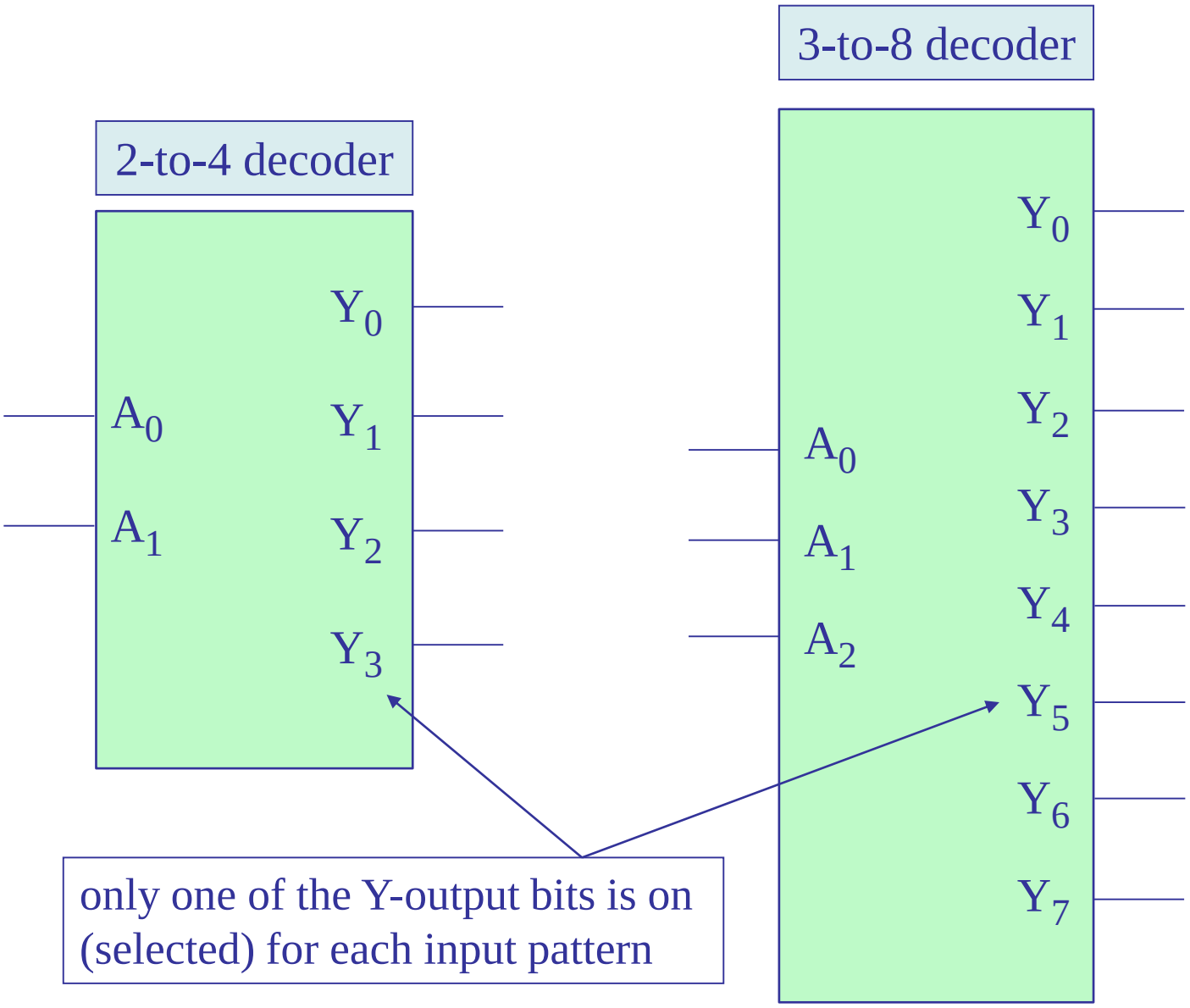
2-to-4 decoder

in			out							
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

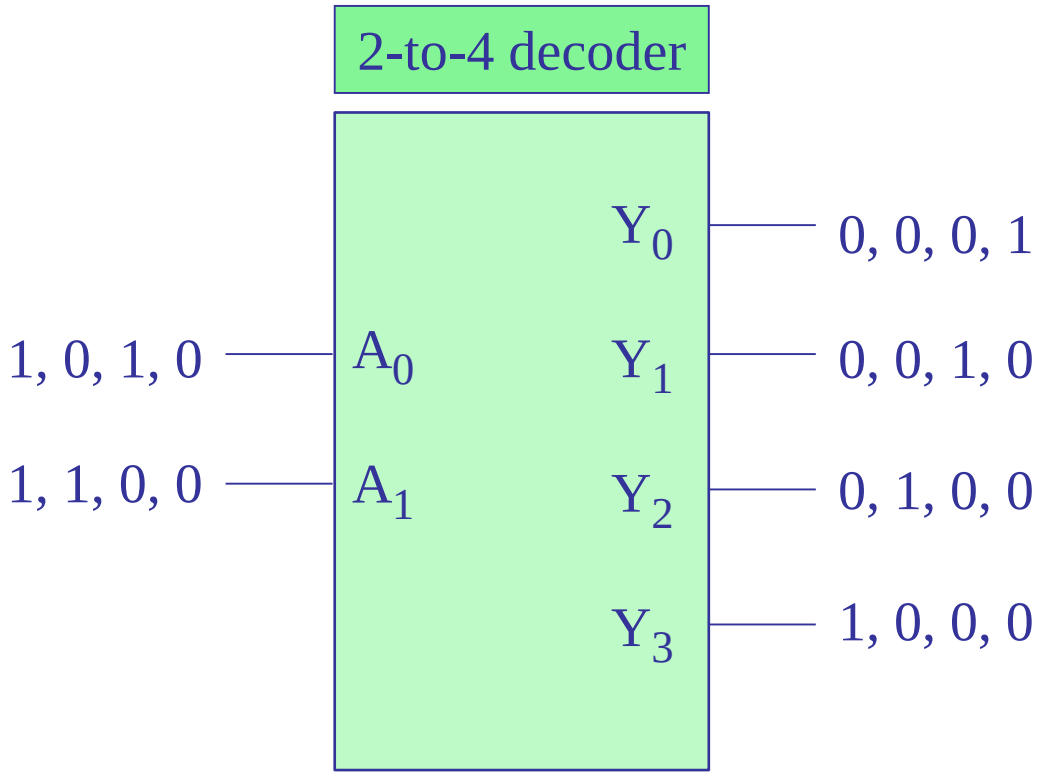
3-to-8 decoder

one-hot encoding realizes all minterms (explained below)

n-to-2^n binary decoders



Decoders



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoders

in	out
0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 1 0 1	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 1 1 0	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 1 1 1	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
1 0 0 0	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 0 1	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 1 0	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 1 1	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 0	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

4-to-16 decoder

Table 6-3. Truth table for a 2-to-4 binary decoder with enable

Inputs				Outputs			
EN	A1	A0		Y3	Y2	Y1	Y0
0	x	x		0	0	0	0
1	0	0	0	0	0	0	1
1	0	1	1	0	0	1	0
1	1	0	2	0	1	0	0
1	1	1	3	1	0	0	0

don’t care entries

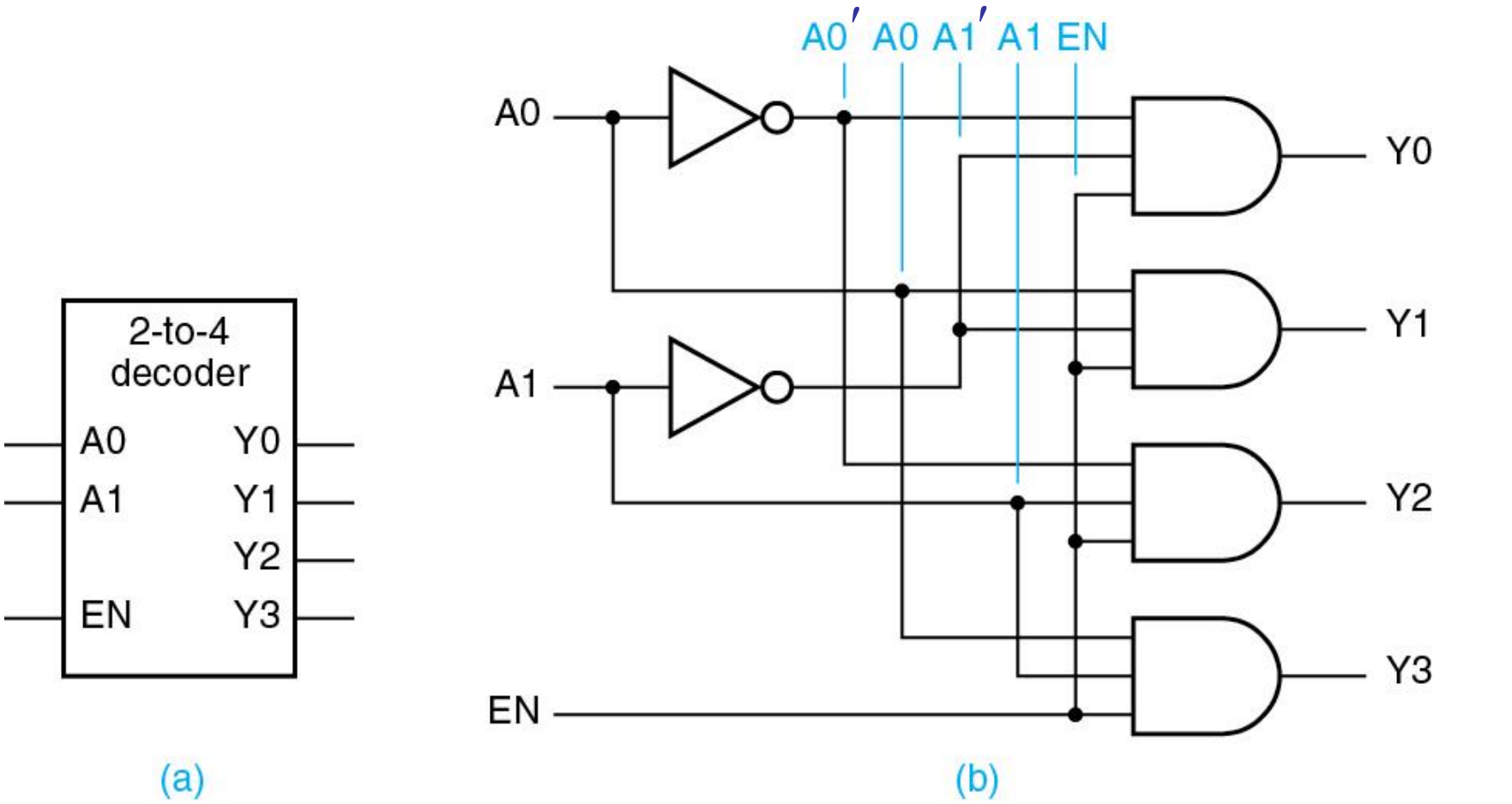
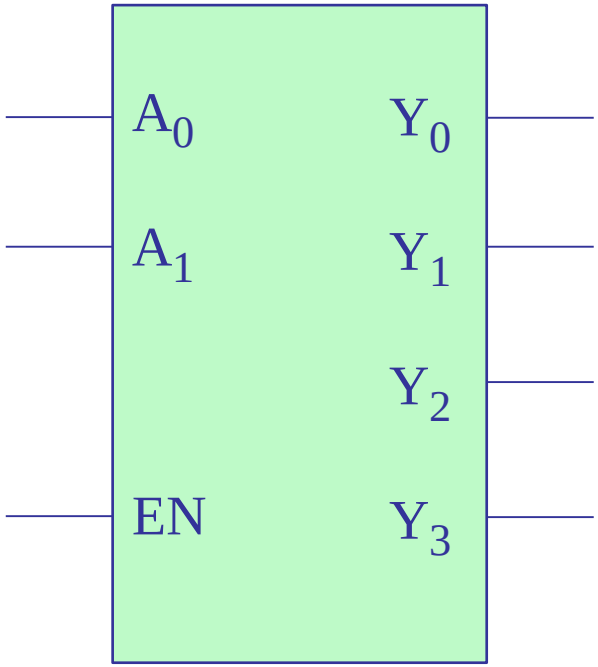


Figure 6-15. 2-to-4 decoder with enable
(a) Inputs and outputs
(b) Logic diagram

Decoders

2-to-4 decoder



when EN=1

$Y_0 = A_1'A_0'$

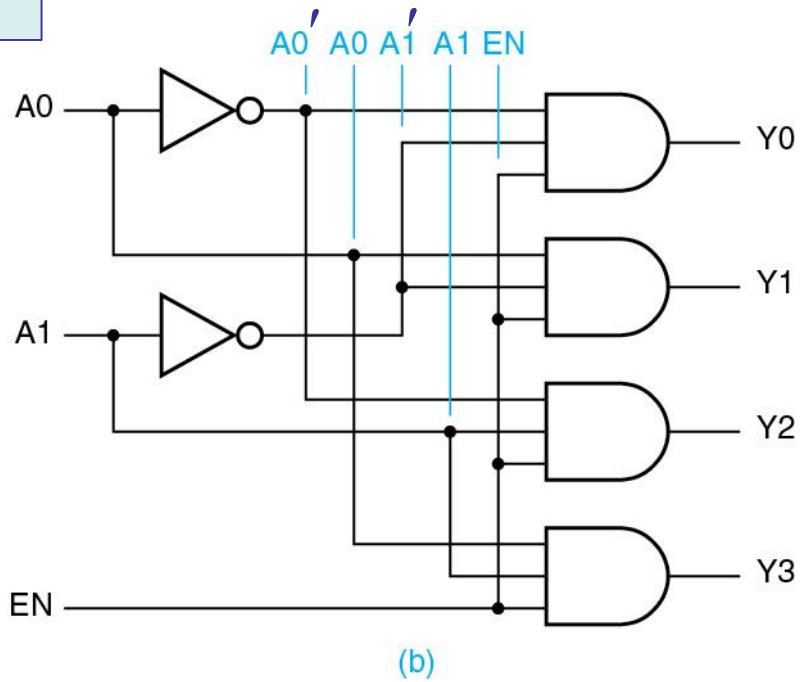
$Y_1 = A_1'A_0$

$Y_2 = A_1A_0'$

$Y_3 = A_1A_0$

all possible minterms

A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



truth table of 3-to-8 decoder

row	$A_2A_1A_0$	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	minterms
0	0 0 0	0	0	0	0	0	0	0	1	$A_2'A_1'A_0'$
1	0 0 1	0	0	0	0	0	0	1	0	$A_2'A_1'A_0$
2	0 1 0	0	0	0	0	0	1	0	0	$A_2'A_1A_0'$
3	0 1 1	0	0	0	0	1	0	0	0	$A_2'A_1A_0$
4	1 0 0	0	0	0	1	0	0	0	0	$A_2A_1'A_0'$
5	1 0 1	0	0	1	0	0	0	0	0	$A_2A_1'A_0$
6	1 1 0	0	1	0	0	0	0	0	0	$A_2A_1A_0'$
7	1 1 1	1	0	0	0	0	0	0	0	$A_2A_1A_0$

MSB

LSB

one-hot encoding

one-hot encoding
realizes all 8 minterms

truth table of 2-to-4 decoder

```
% generating the truth table

 [A1,A0] = a2d(0:3,2);

 Y3 =  A1 &  A0;
 Y2 =  A1 & ~A0;
 Y1 = ~A1 &  A0;
 Y0 = ~A1 & ~A0;

 [A1, A0, Y3, Y2, Y1, Y0]

% 0   0   0   0   0   1
% 0   1   0   0   1   0
% 1   0   0   1   0   0
% 1   1   1   0   0   0
```

$Y_0 = A_1'A_0'$

$Y_1 = A_1'A_0$

$Y_2 = A_1A_0'$

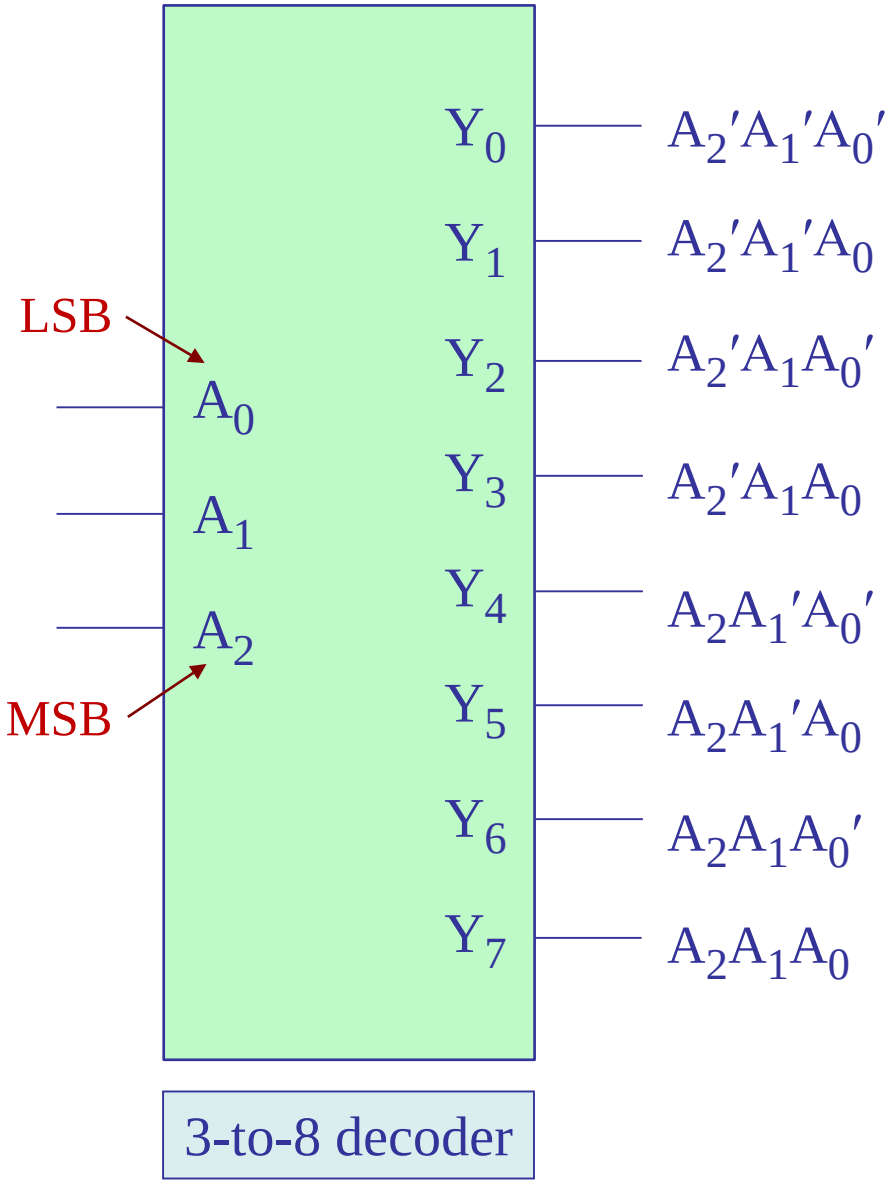
$Y_3 = A_1A_0$

truth table of 3-to-8 decoder

```
% generating the truth table

[A2,A1,A0] = a2d(0:7,3);

Y7 =  A2 &  A1 &  A0;
Y6 =  A2 &  A1 & ~A0;
Y5 =  A2 & ~A1 &  A0;
Y4 =  A2 & ~A1 & ~A0;
Y3 = ~A2 &  A1 &  A0;
Y2 = ~A2 &  A1 & ~A0;
Y1 = ~A2 & ~A1 &  A0;
Y0 = ~A2 & ~A1 & ~A0;

[A2,A1,A0,Y7,Y6,Y5,Y4,Y3,Y2,Y1,Y0]
```

Realizing arbitrary combinational functions with decoders

Decoders



all possible A,B,C minterms

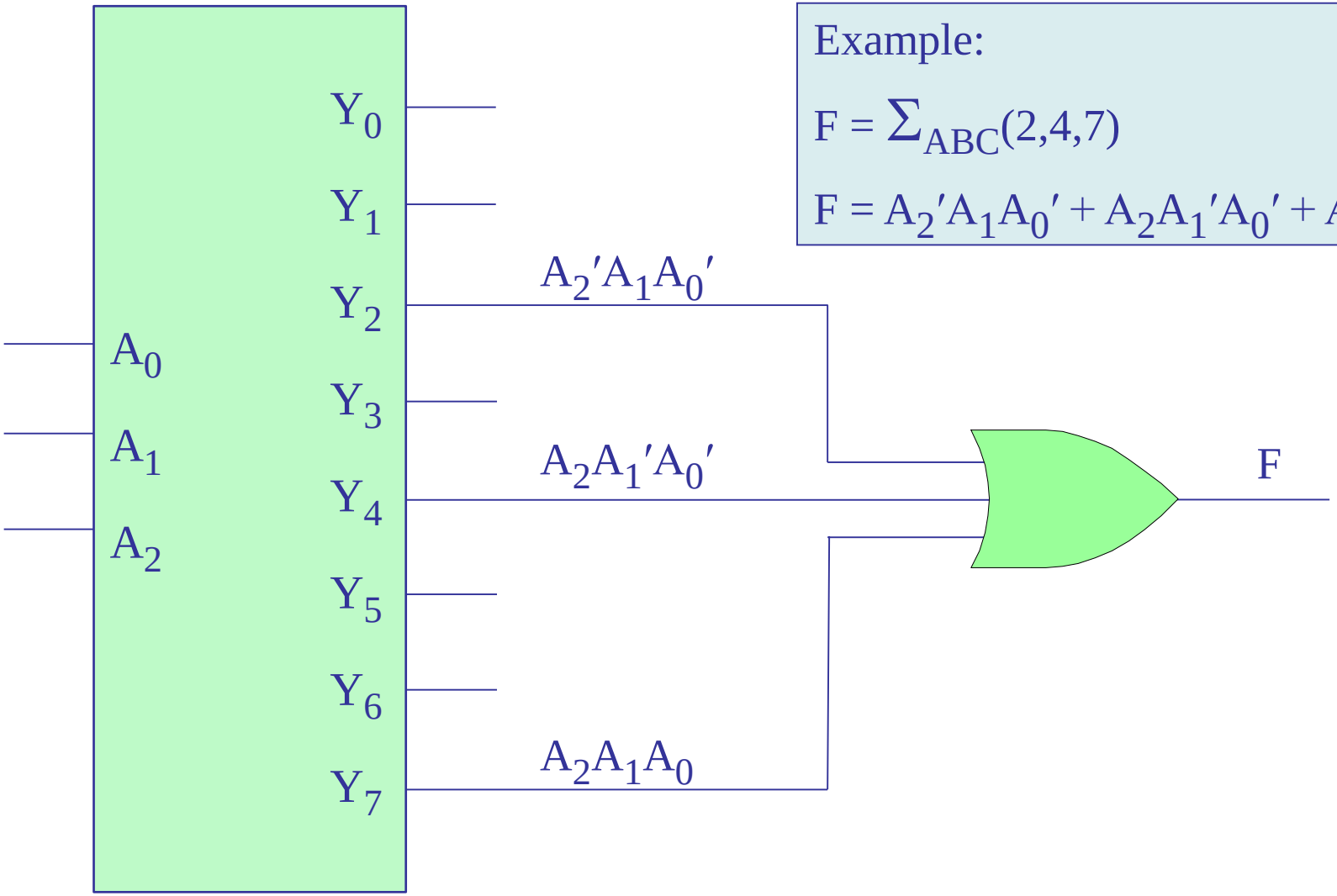
Decoders can implement arbitrary combinational functions, because the decoder outputs are all possible minterms, similar to using ROMs or look-up-tables in FPGAs.

Decoders

Example:

$F = \Sigma_{ABC}(2,4,7)$

$F = A_2'A_1A_0' + A_2A_1'A_0' + A_2A_1A_0$



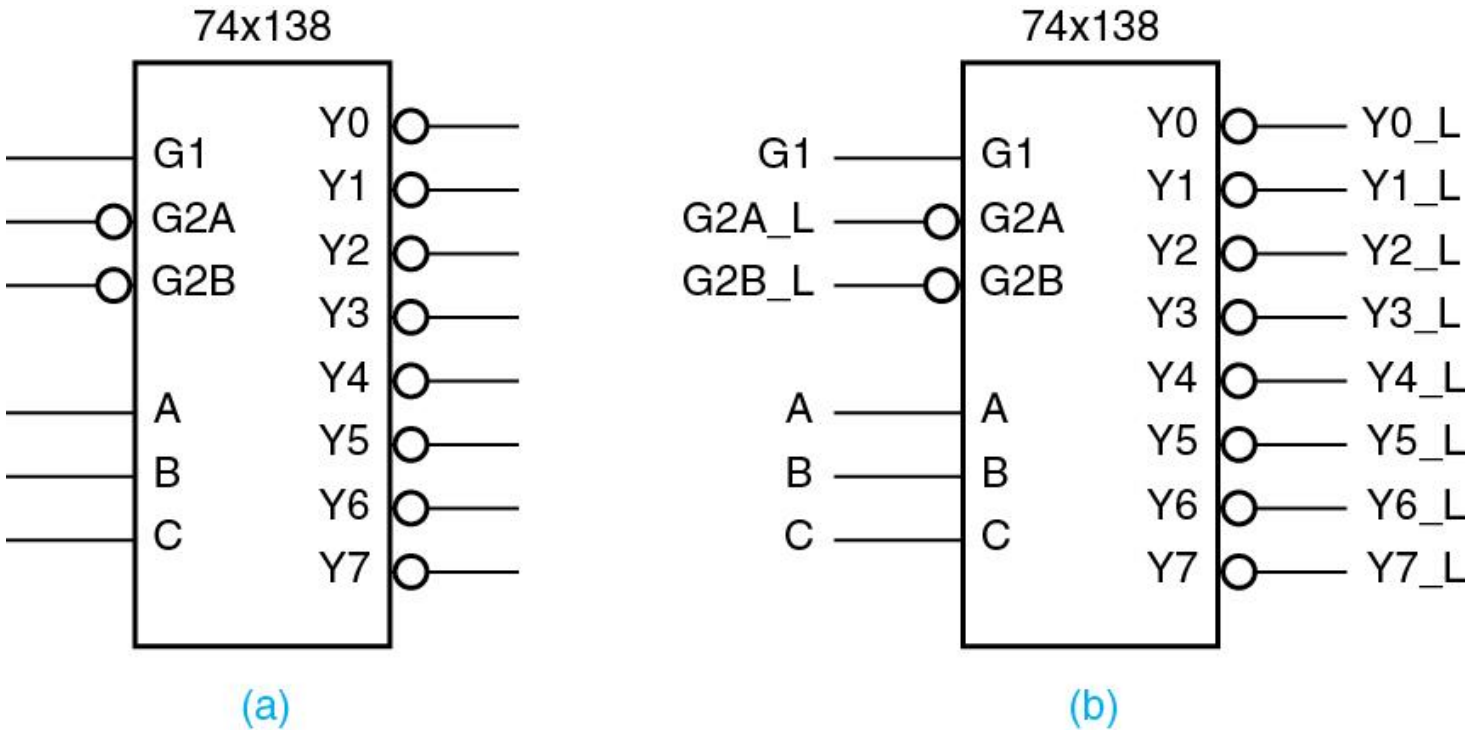
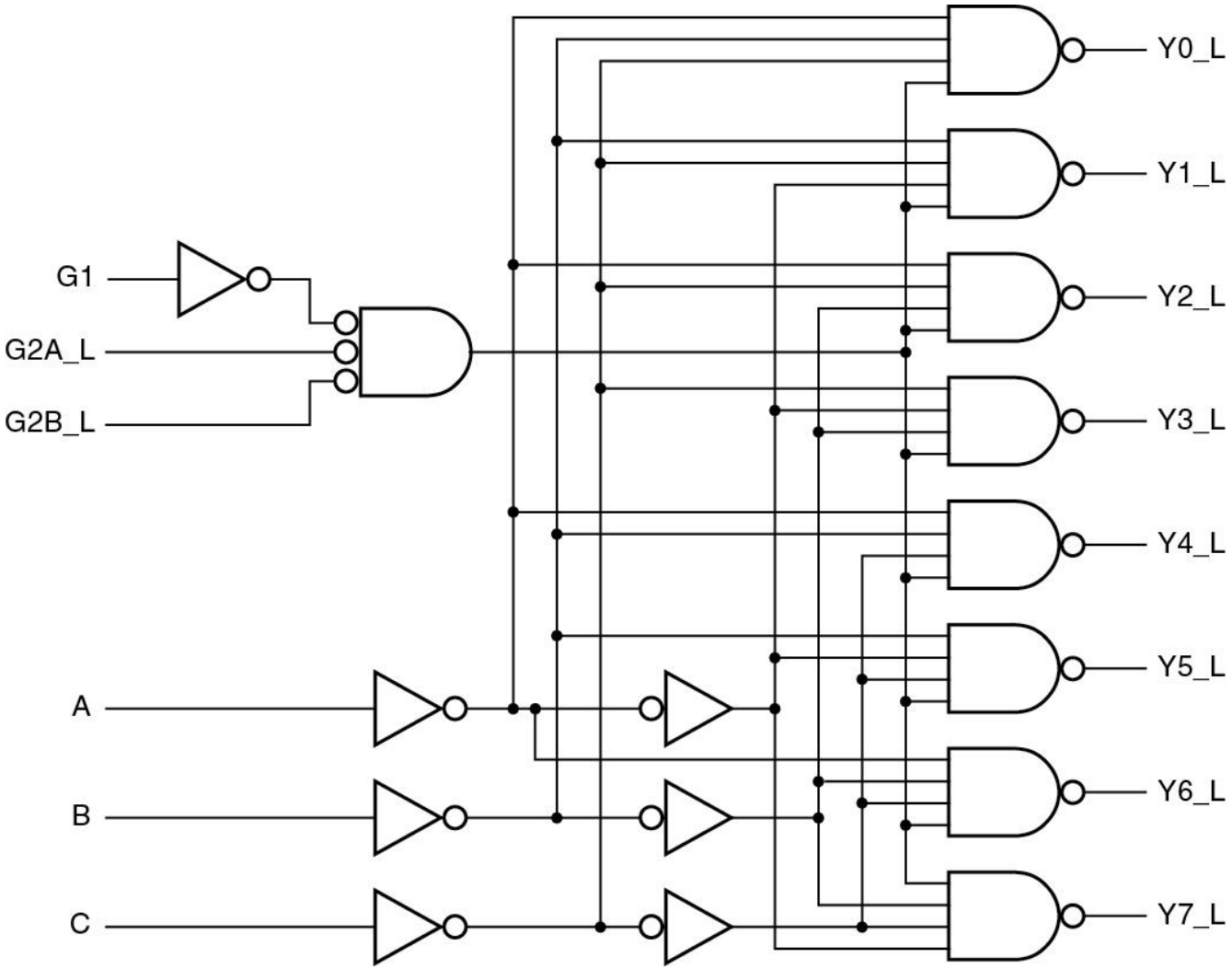


Figure 6-16. Logic symbol for the 74x138 3-to-8 decoder.

(a) conventional symbol

(b) default signal names associated with external pins

Decoders



active-low outputs

Figure 6-17. Logic diagram for the 74x138 3-to-8 decoder

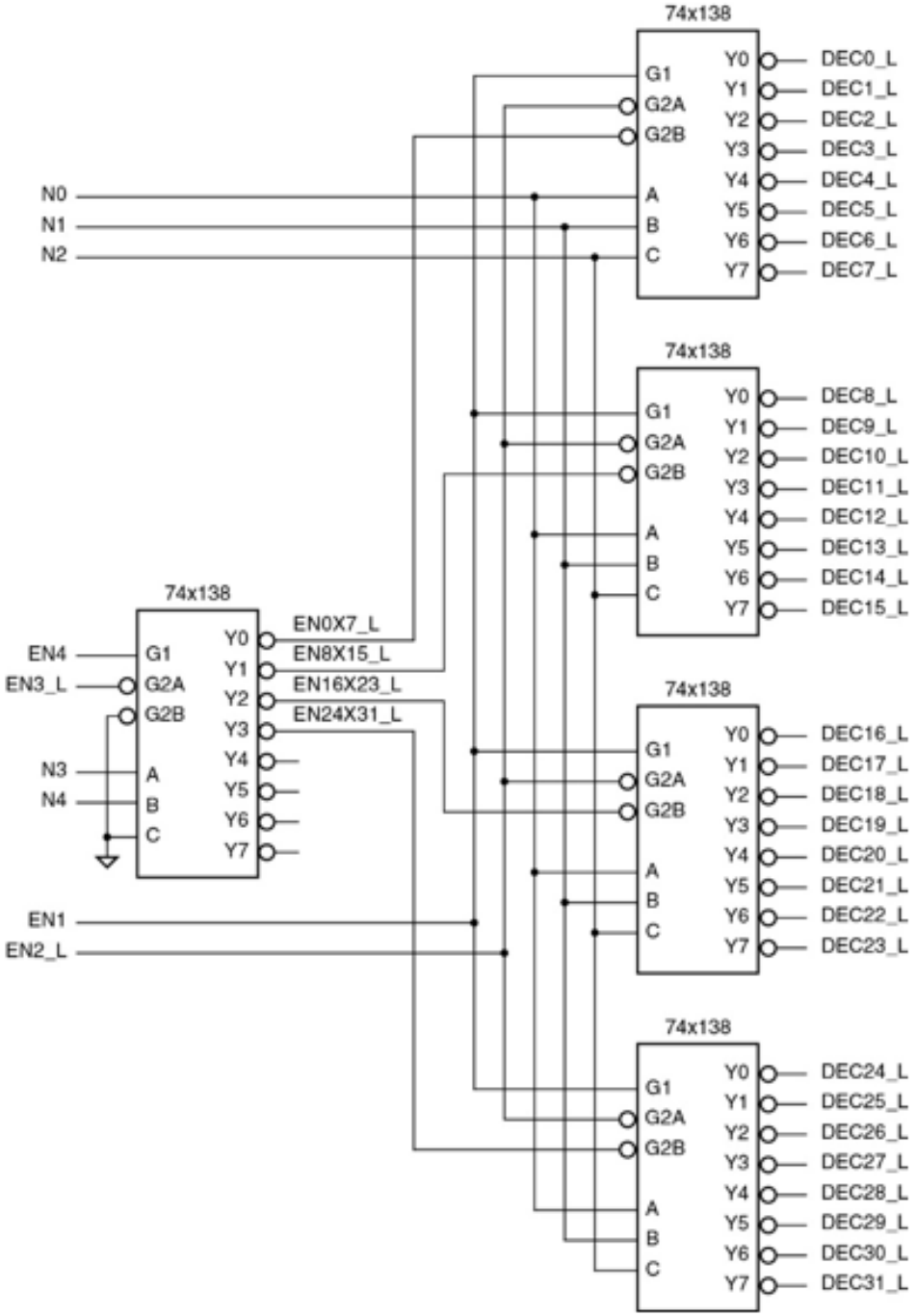
MSB

active-low outputs

Inputs						Outputs							
G1	G2A_L	G2B_L	C	B	A	Y7_L	Y6_L	Y5_L	Y4_L	Y3_L	Y2_L	Y1_L	Y0_L
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

Figure 6-17. Truth table of the 74x138 3-to-8 decoder

Decoders



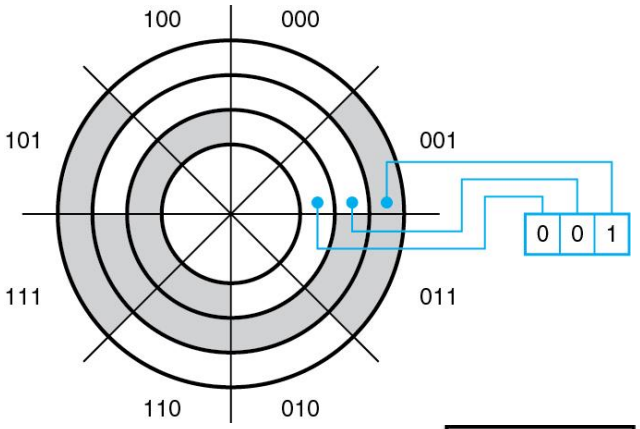
realizing larger decoders from smaller ones

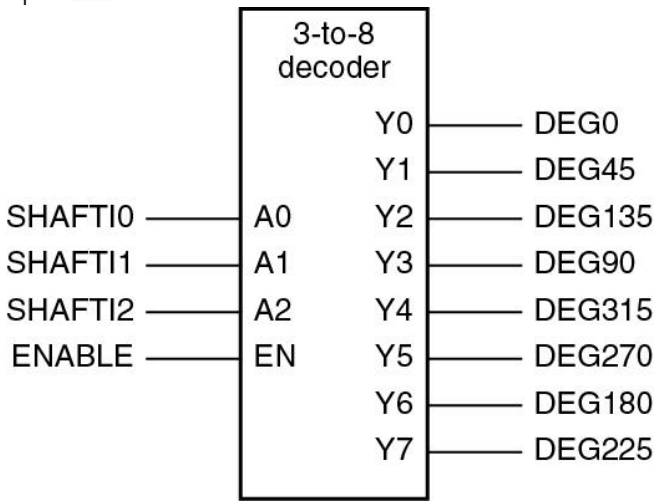
Fig. 6-19. Building a 5-to-32 decoder using five 3-to-8 decoders

Decoders

More generally, it is not necessary to use all of the outputs of a decoder, or even to decode all input combinations (e.g., as in BCD-to-7 segment displays), or that the input combinations are in binary order.

Here is another example of a shaft-position decoder using Gray coding



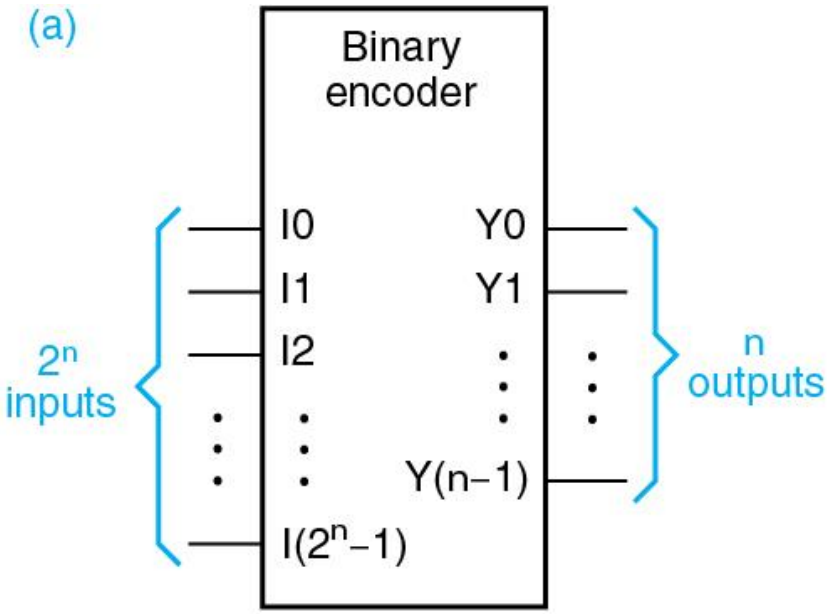


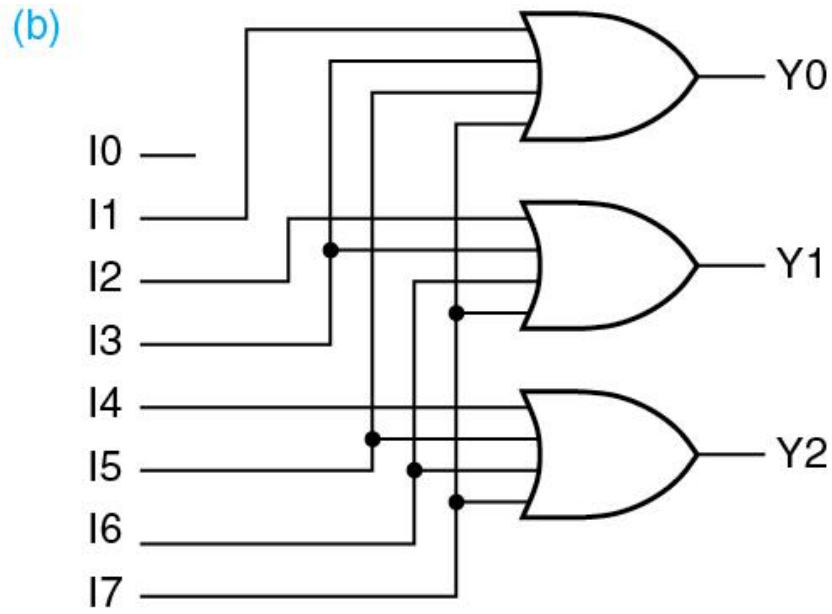
Disk Position	A2	A1	A0	Binary Decoder Output
0°	0	0	0	Y0
45°	0	0	1	Y1
90°	0	1	1	Y3
135°	0	1	0	Y2
180°	1	1	0	Y6
225°	1	1	1	Y7
270°	1	0	1	Y5
315°	1	0	0	Y4

see Wakerly, Ch. 2 & Ch. 6

Encoders

2^n-to-n binary encoders perform the opposite function of n-to-2^n binary decoders





encoding operation →

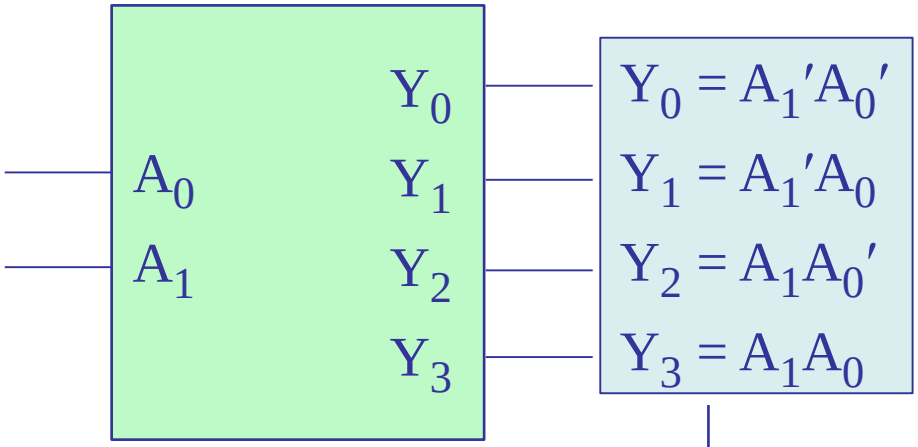
decoding operation ←

Figure 6-24. Binary encoder.
(a) general structure
(b) 8-to-3 encoder

(block diagram to be explained below)

Encoders

2-to-4 decoder



$Y_0 = A_1'A_0'$

$Y_1 = A_1'A_0$

$Y_2 = A_1A_0'$

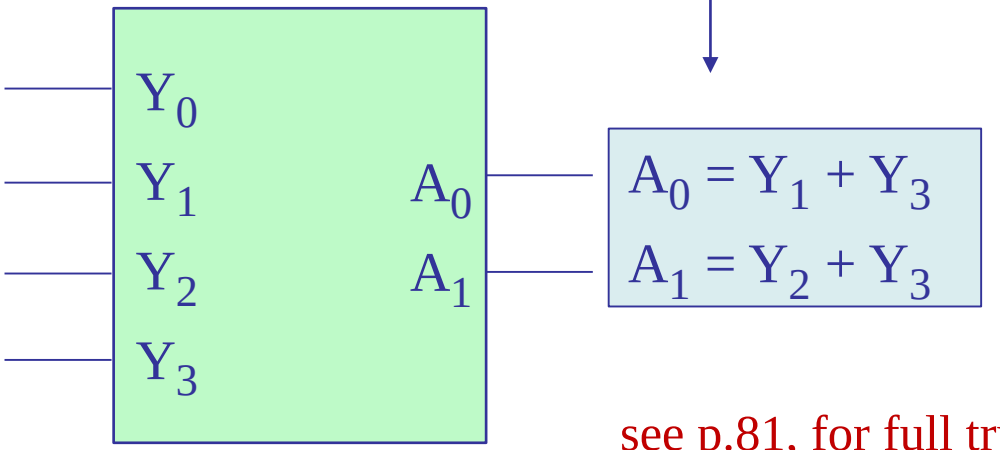
$Y_3 = A_1A_0$

A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

$$Y_1 + Y_3 = A_1'A_0 + A_1A_0 = (A_1' + A_1)\,A_0 = A_0$$

$$Y_2 + Y_3 = A_1A_0' + A_1A_0 = (A_0' + A_0)\,A_1 = A_1$$

4-to-2 encoder



$A_0 = Y_1 + Y_3$

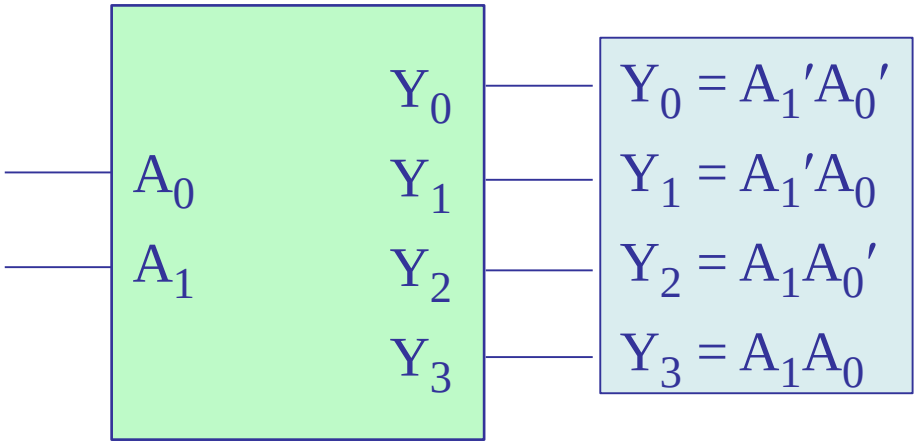
$A_1 = Y_2 + Y_3$

Y_3	Y_2	Y_1	Y_0	A_1	A_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

see p.81, for full truth-table and a problem with encoders

Encoders

2-to-4 decoder



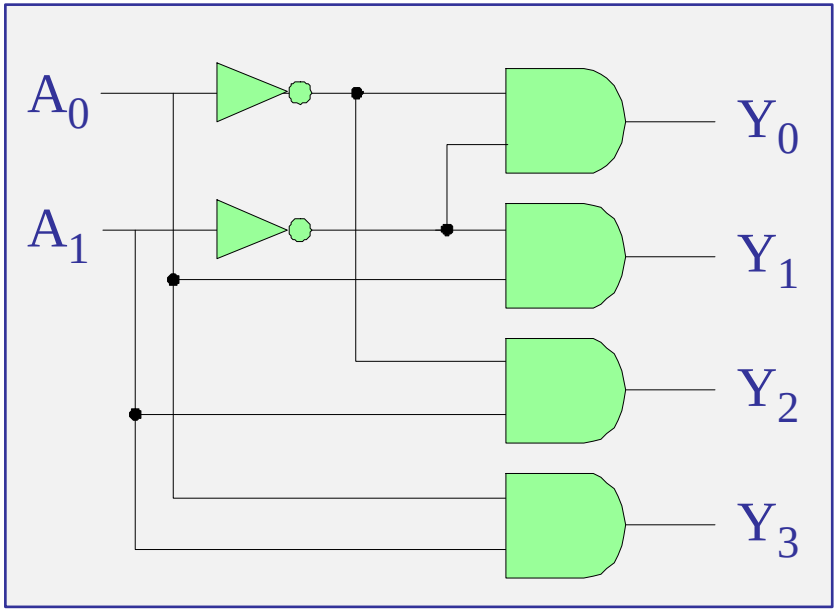
$Y_0 = A_1'A_0'$

$Y_1 = A_1'A_0$

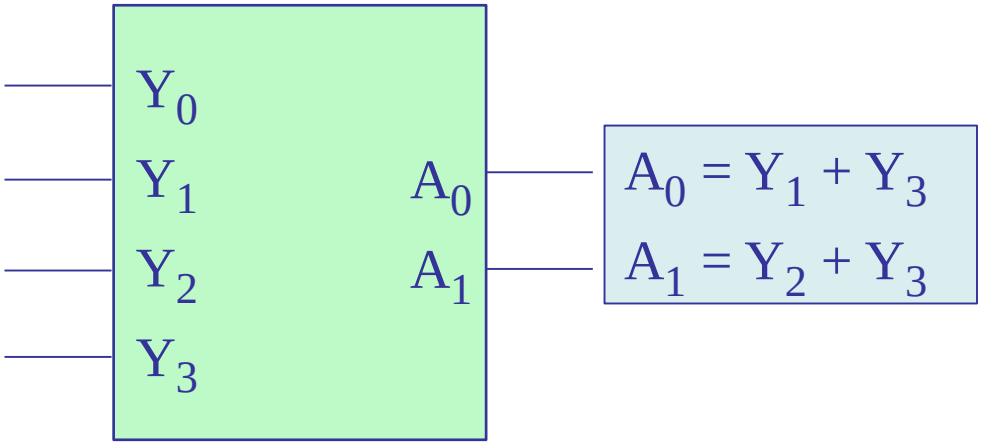
$Y_2 = A_1A_0'$

$Y_3 = A_1A_0$

decoder



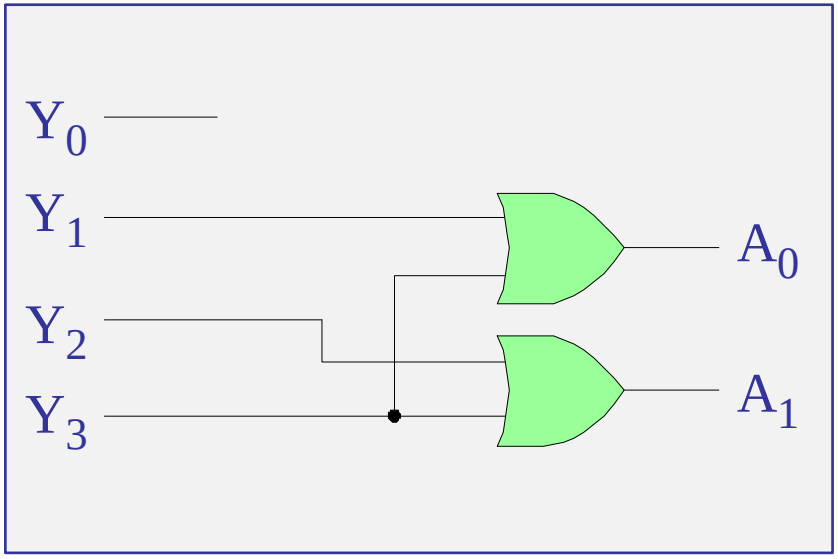
4-to-2 encoder



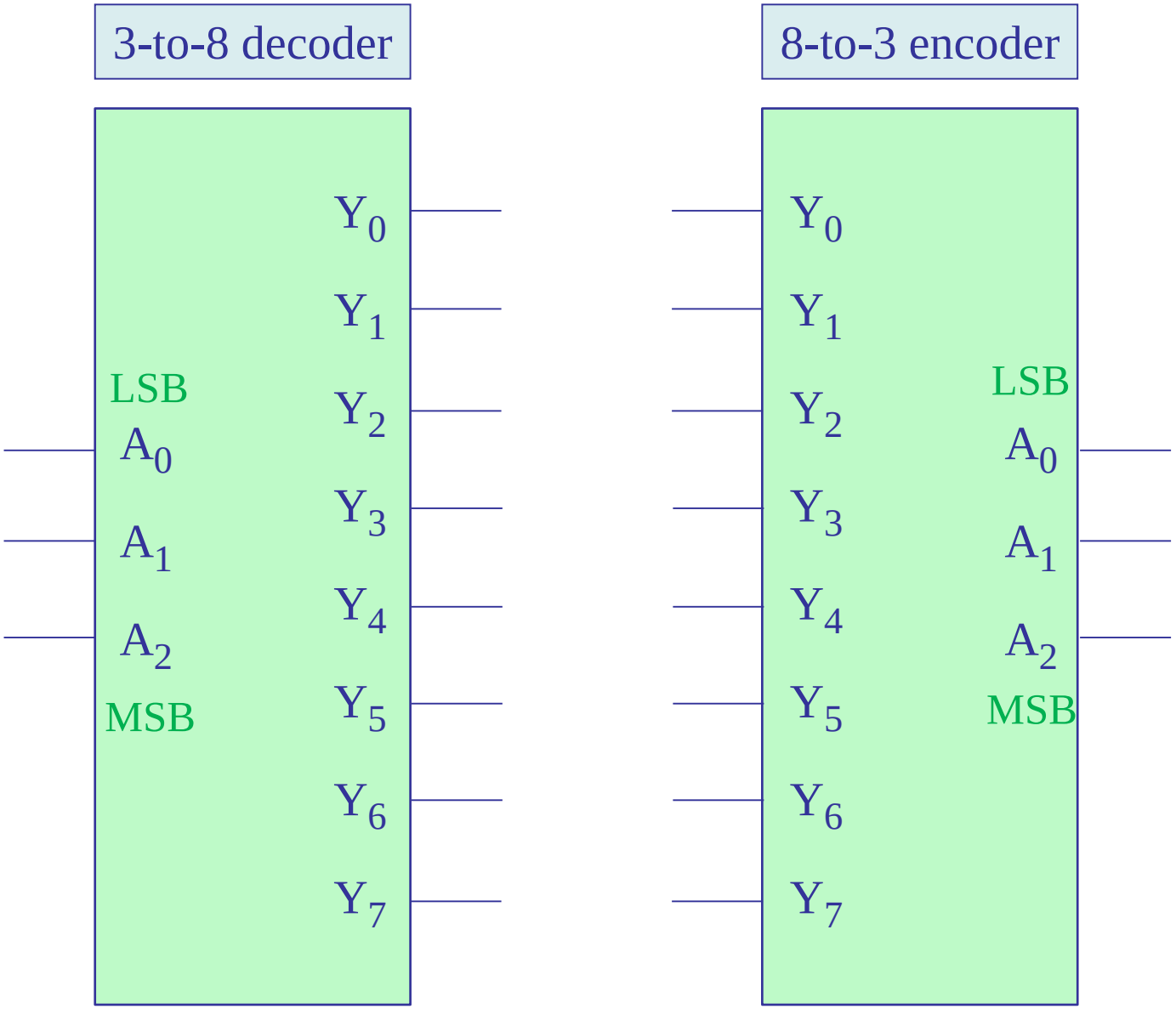
$A_0 = Y_1 + Y_3$

$A_1 = Y_2 + Y_3$

encoder



Encoders



truth table →

truth table of 8-to-3 encoder

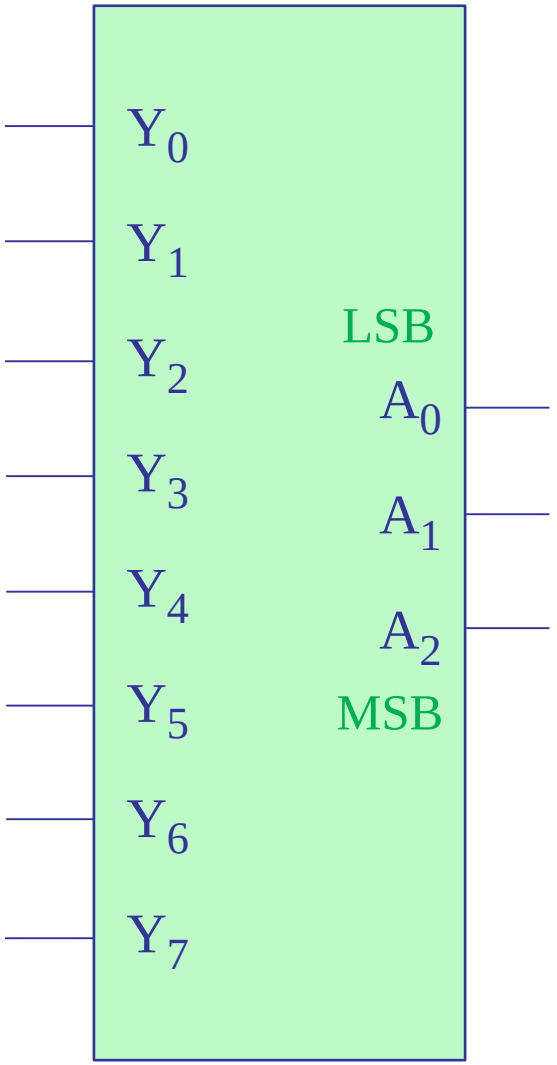
row	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	1
2	0	0	0	0	0	1	0	0	0	1	0
3	0	0	0	0	1	0	0	0	0	1	1
4	0	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	0	0	0	0	1	0	1
6	0	1	0	0	0	0	0	0	1	1	0
7	1	0	0	0	0	0	0	0	1	1	1

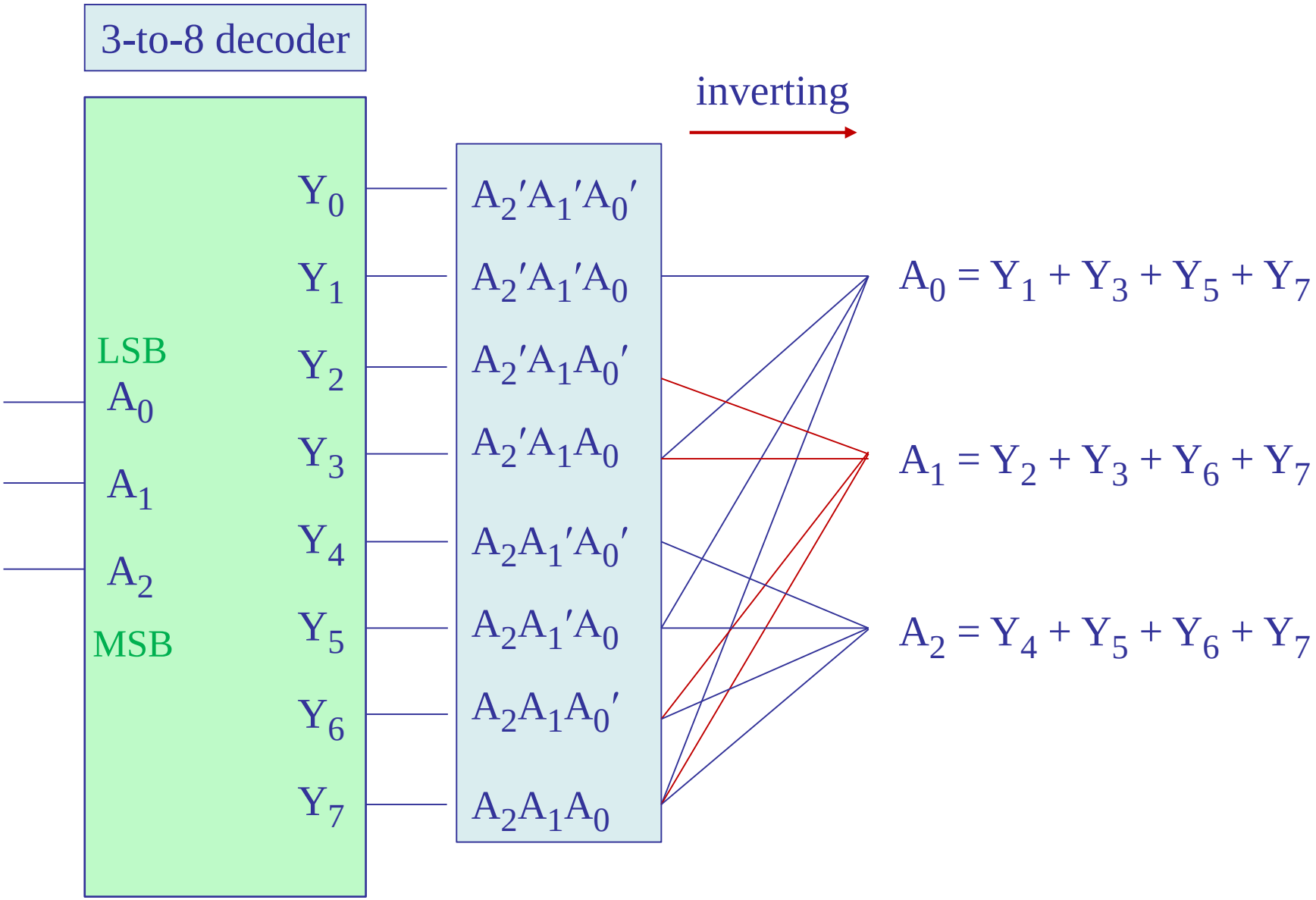
$$A_0 = Y_1 + Y_3 + Y_5 + Y_7$$

$$A_1 = Y_2 + Y_3 + Y_6 + Y_7$$

$$A_2 = Y_4 + Y_5 + Y_6 + Y_7$$

8-to-3 encoder




3-to-8 decoder
inverting
LSB
A_0
A_1
A_2
MSB
Y_0
Y_1
Y_2
Y_3
Y_4
Y_5
Y_6
Y_7
$A_2'A_1'A_0'$
$A_2'A_1'A_0$
$A_2'A_1A_0'$
$A_2'A_1A_0$
$A_2A_1'A_0'$
$A_2A_1'A_0$
$A_2A_1A_0'$
$A_2A_1A_0$
$A_0 = Y_1 + Y_3 + Y_5 + Y_7$
$A_1 = Y_2 + Y_3 + Y_6 + Y_7$
$A_2 = Y_4 + Y_5 + Y_6 + Y_7$

demonstration of simplification steps:

$$Y_1 + Y_3 + Y_5 + Y_7 = A_2'A_1'A_0 + A_2'A_1A_0 + A_2A_1'A_0 + A_2A_1A_0$$

$$= A_2'(A_1' + A_1)A_0 + A_2(A_1' + A_1)A_0$$

$$= A_2'A_0 + A_2A_0 = (A_2' + A_2)A_0 = A_0$$

alternatively, we have,

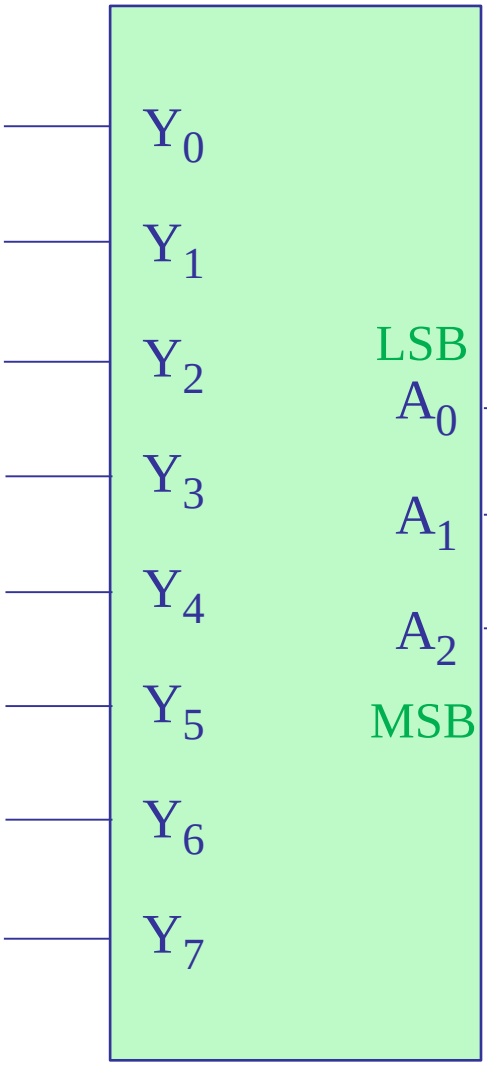
$$Y_1 + Y_3 + Y_5 + Y_7 = A_2'A_1'A_0 + A_2'A_1A_0 + A_2A_1'A_0 + A_2A_1A_0$$

$$= (A_2'A_1' + A_2'A_1 + A_2A_1' + A_2A_1)\,A_0$$

$$= (A_2' + A_2)(A_1' + A_1)\,A_0 = A_0$$

will be explored further in recitation exercises

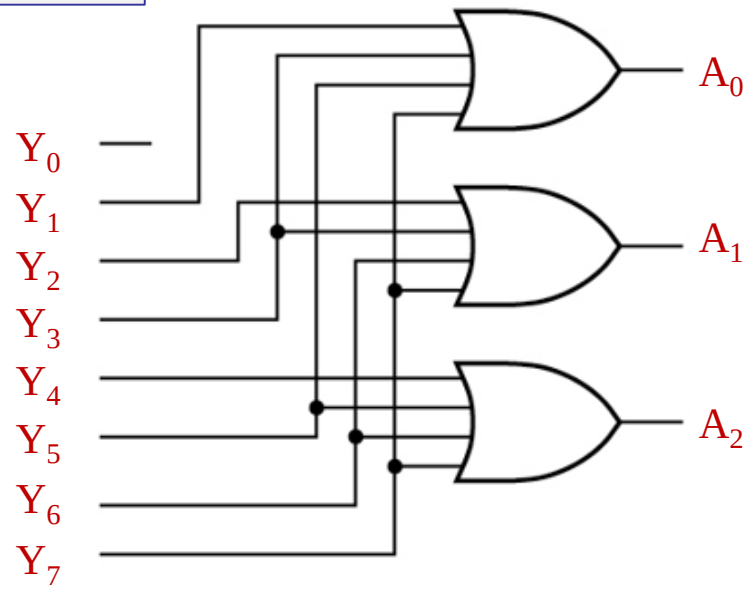
8-to-3 encoder



$A_0 = Y_1 + Y_3 + Y_5 + Y_7$

$A_1 = Y_2 + Y_3 + Y_6 + Y_7$

$A_2 = Y_4 + Y_5 + Y_6 + Y_7$

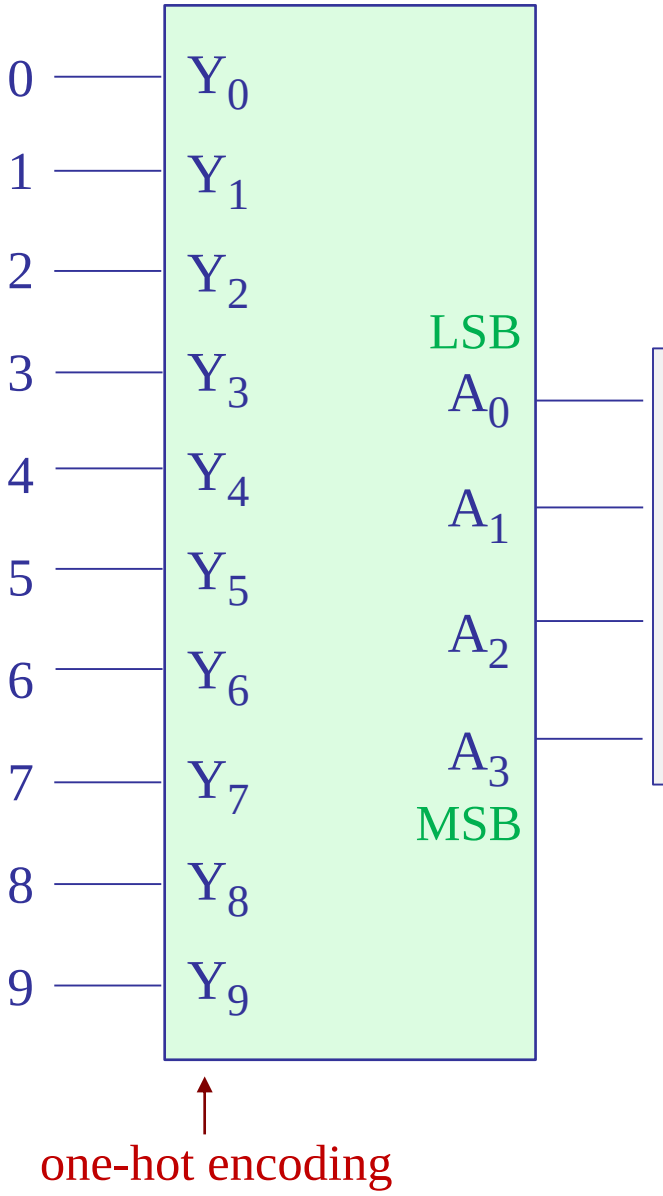


	inputs										BCD code			
n	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	1
2	0	0	0	0	0	0	0	1	0	0	0	0	1	0
3	0	0	0	0	0	0	1	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0	0	0	0	1	0	0
5	0	0	0	0	1	0	0	0	0	0	0	1	0	1
6	0	0	0	1	0	0	0	0	0	0	0	1	1	0
7	0	0	1	0	0	0	0	0	0	0	0	1	1	1
8	0	1	0	0	0	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0	0	1	0	0	1

one-hot encoding of the ten integers 0,1,…,9

see p.75, on how to generate such table with MATLAB

decimal-to-BCD encoder



$$A_0 = Y_1 + Y_3 + Y_5 + Y_7 + Y_9$$

$$A_1 = Y_2 + Y_3 + Y_6 + Y_7$$

$$A_2 = Y_4 + Y_5 + Y_6 + Y_7$$

$$A_3 = Y_8 + Y_9$$

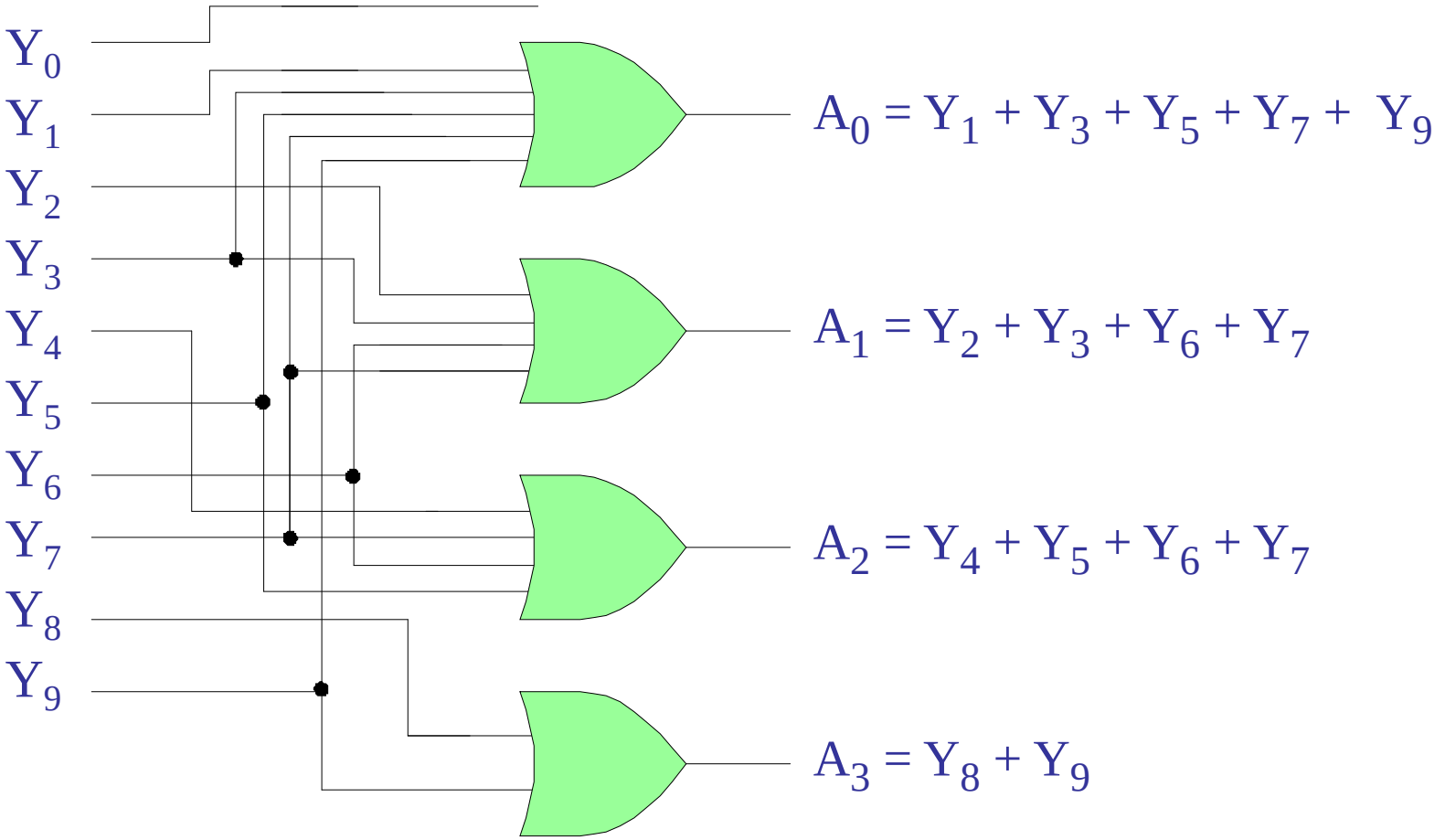
see p.75, for MATLAB code
and p.76, for gate-level implementation
and p.77, for chip-level realization
and p.41 & 78, for a complete system

decimal-to-BCD encoder

```
% decimal-to-BCD truth table on p.73

Y = fliplr(eye(10));

Y9 = Y(:,1);  Y8 = Y(:,2);  Y7 = Y(:,3);
Y6 = Y(:,4);  Y5 = Y(:,5);  Y4 = Y(:,6);
Y3 = Y(:,7);  Y2 = Y(:,8);  Y1 = Y(:,9);
Y0 = Y(:,10);

A3 = Y8 | Y9;
A2 = Y4 | Y5 | Y6 | Y7;
A1 = Y2 | Y3 | Y6 | Y7;
A0 = Y1 | Y3 | Y5 | Y7 | Y9;

[Y,A3,A2,A1,A0]              % print table
```

$$A_0 = Y_1 + Y_3 + Y_5 + Y_7 + Y_9$$
$$A_1 = Y_2 + Y_3 + Y_6 + Y_7$$
$$A_2 = Y_4 + Y_5 + Y_6 + Y_7$$
$$A_3 = Y_8 + Y_9$$

decimal-to-BCD encoder



will be explored further in recitation exercises

Encoders

Decimal-to-BCD Encoder
High–Performance Silicon–Gate CMOS

The MC74HC147 is identical in pinout to the LS147. The device inputs are compatible with standard CMOS outputs; with pullup resistors, they are compatible with LSTTL outputs.

This device encodes nine active–low data inputs to four active–low BCD Address Outputs, ensuring that only the highest order active data line is encoded. The implied decimal zero condition is encoded when all nine data inputs are at a high level (inactive).

- Output Drive Capability: 10 LSTTL Loads
- Outputs Directly Interface to CMOS, NMOS, and TTL
- Operating Voltage Range: 2 to 6 V
- Low Input Current: 1 μA
- High Noise Immunity Characteristic of CMOS Devices
- In Compliance with the Requirements Defined by JEDEC Standard No. 7A
- Chip Complexity: 136 FETs or 34 Equivalent Gates

MC74HC147

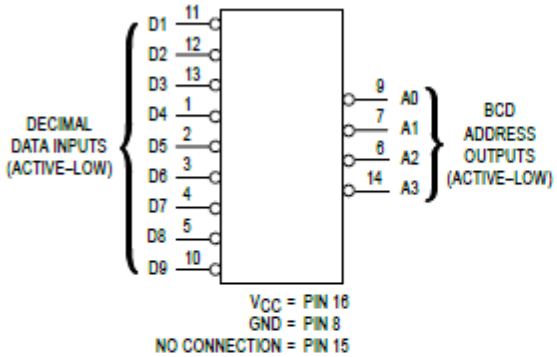
N SUFFIX
PLASTIC PACKAGE
CASE 648–08

D SUFFIX
SOIC PACKAGE
CASE 751B–05

ORDERING INFORMATION

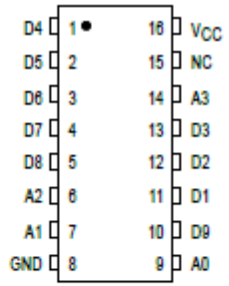
MC74HCXXXN	Plastic
MC74HCXXXD	SOIC

LOGIC DIAGRAM



V_{CC} = PIN 16
GND = PIN 8
NO CONNECTION = PIN 15

PIN ASSIGNMENT

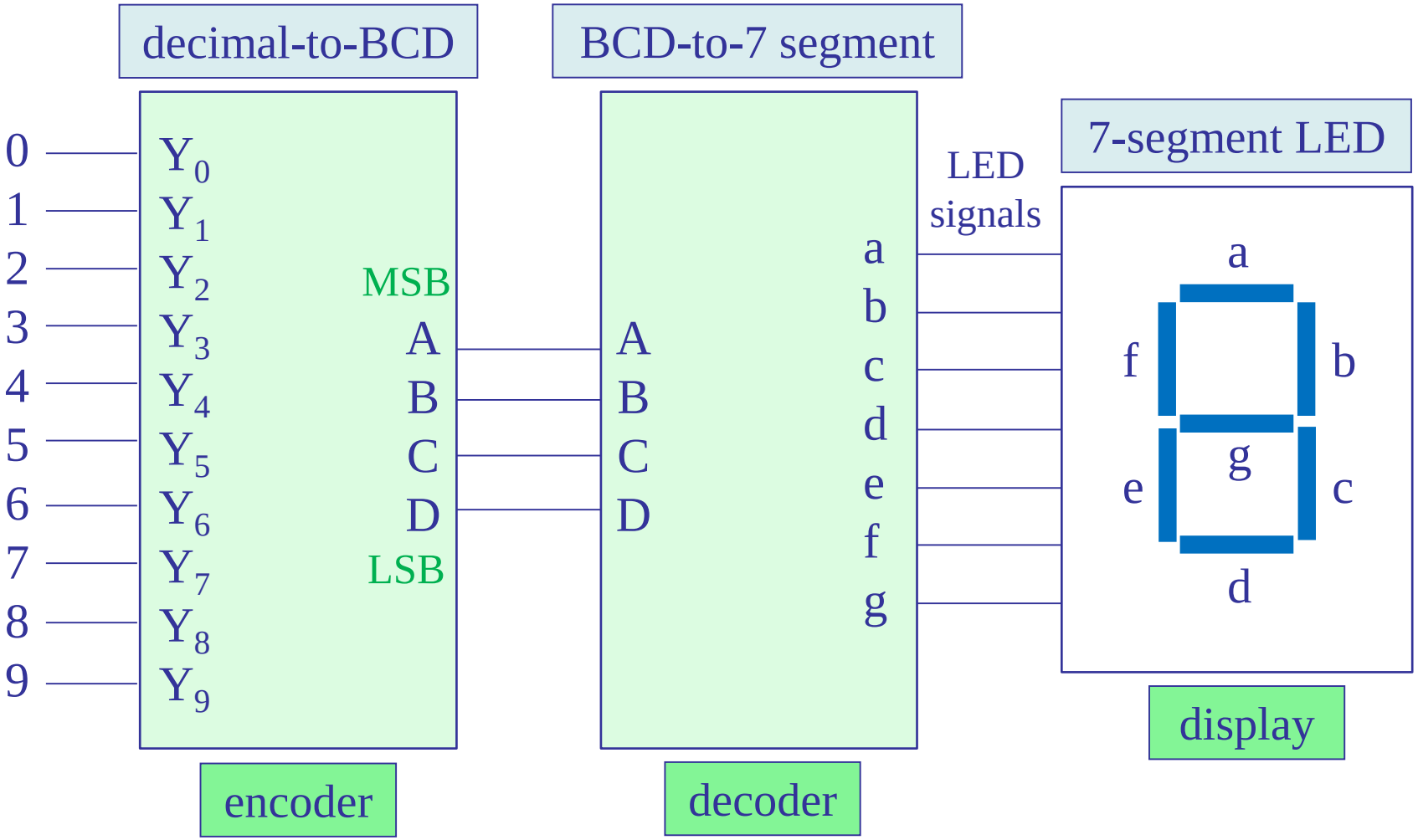


NC = NO CONNECTION

FUNCTION TABLE

Inputs									Outputs			
D9	D8	D7	D6	D5	D4	D3	D2	D1	A3	A2	A1	A0
H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	L	H	H	H	L
H	H	H	H	H	H	H	L	X	H	H	L	H
H	H	H	H	H	H	L	X	X	H	H	L	L
H	H	H	H	H	L	X	X	X	H	L	H	H
H	H	H	H	L	X	X	X	X	H	L	H	L
H	H	H	L	X	X	X	X	X	H	L	L	H
H	H	L	X	X	X	X	X	X	H	L	L	L
H	L	X	X	X	X	X	X	X	L	H	H	H
L	X	X	X	X	X	X	X	X	L	H	H	L

BCD to seven-segment display encoder/decoder system



three-state buffers

Encoders

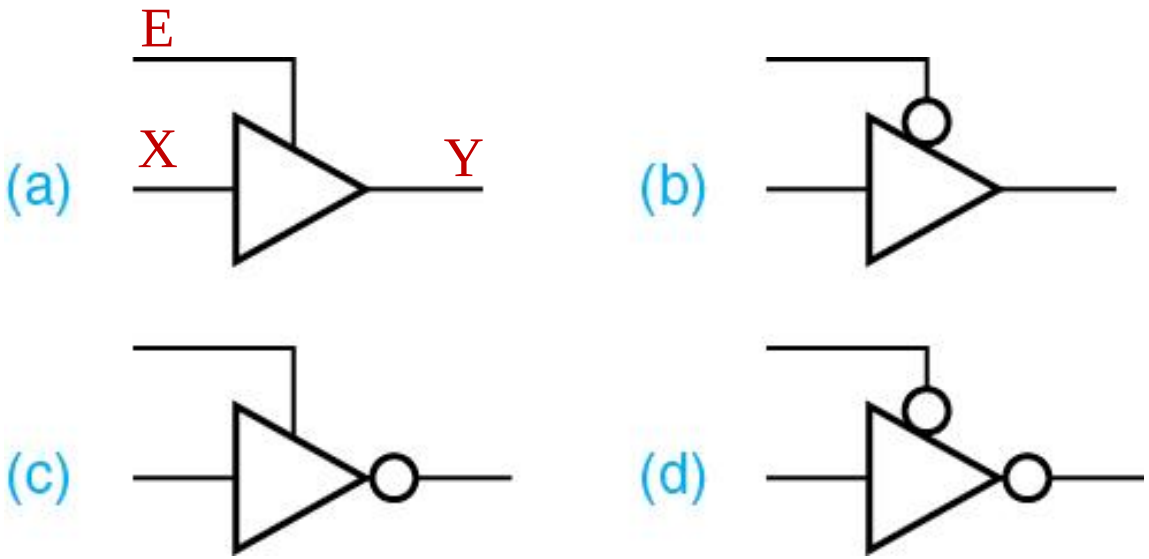


Figure 7-1. Various three-state buffers

(a,b) non-inverting

(c,d) inverting

(a,c) active-high enable

(b,d) active-low enable

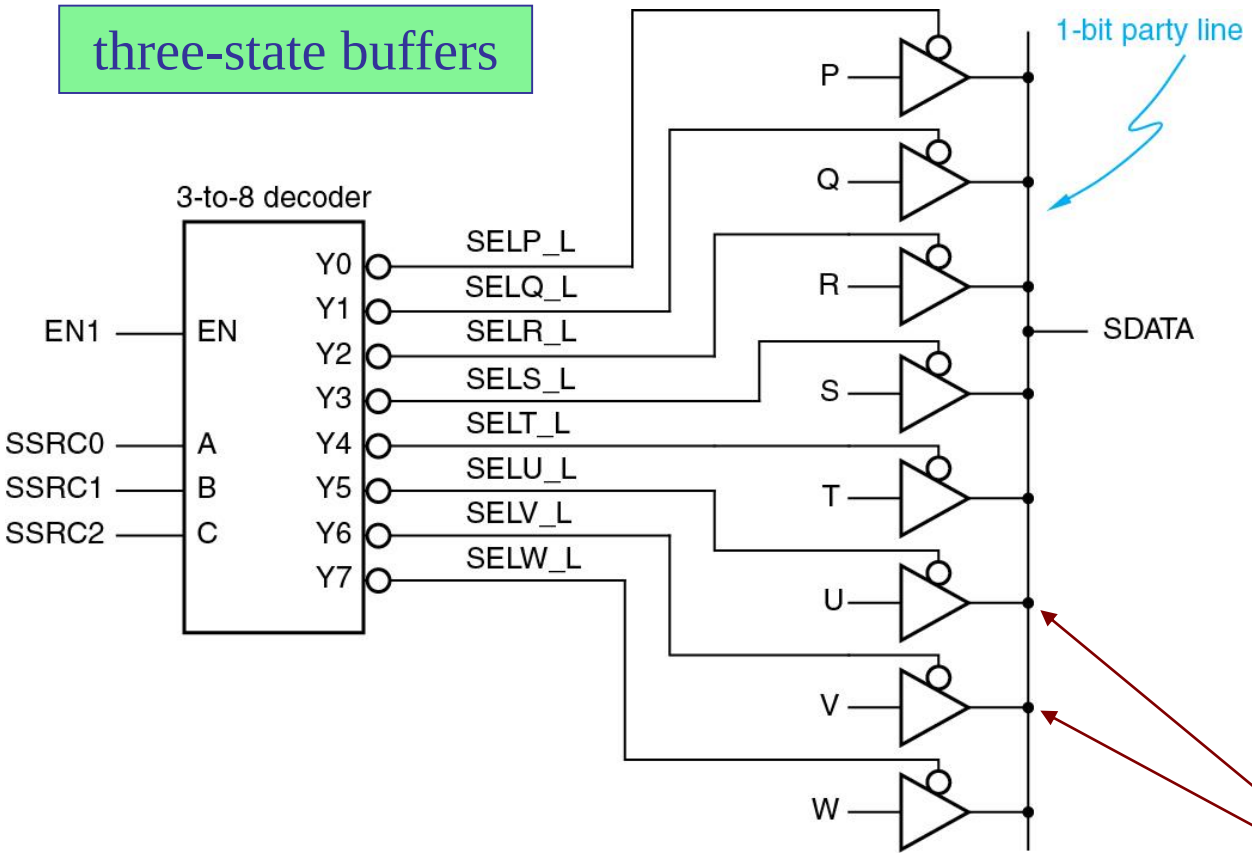
E	X	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

truth table for (a)

high-impedance state, with X effectively disconnected from Y

three-state buffers

Encoders



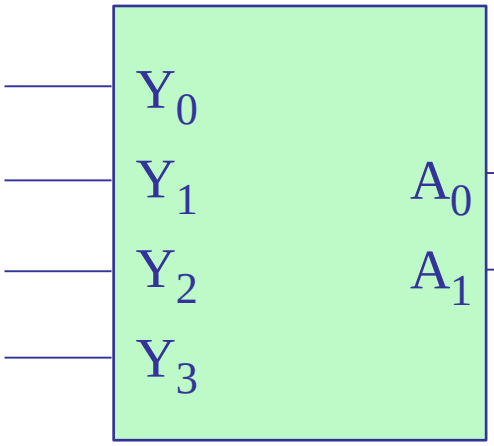
normally, one cannot connect the outputs like that, however, here, only one of the outputs is active at a time, the others being disabled by going into their high-impedance state

Figure 7-2. Eight sources sharing a single three-state party line

problem with encoders when multiple inputs are asserted

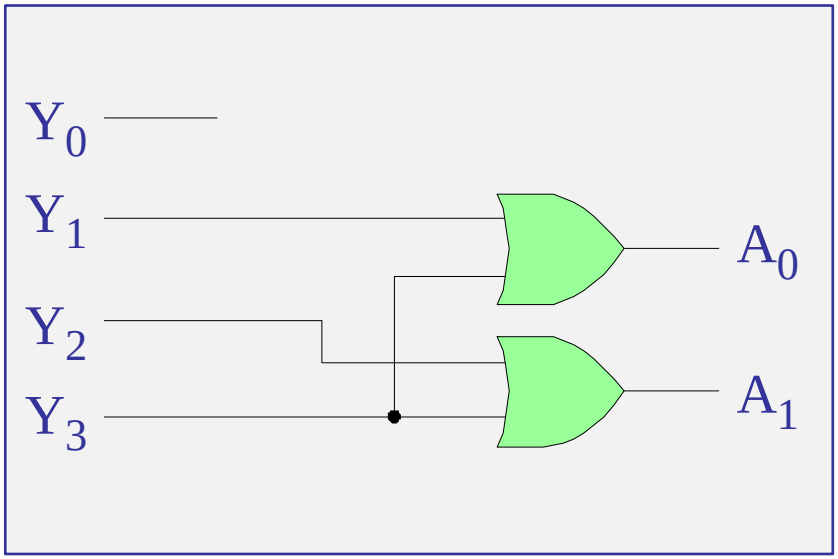
Priority encoders

4-to-2 encoder



$A_0 = Y_1 + Y_3$

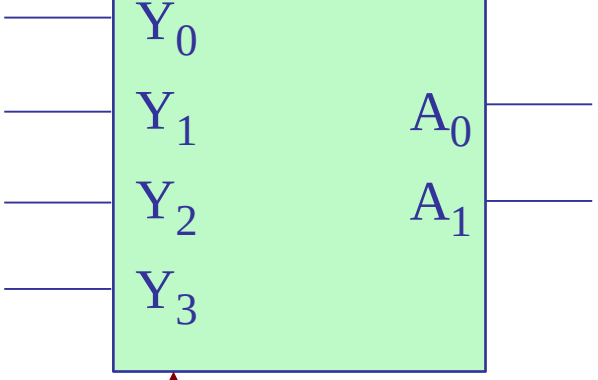
$A_1 = Y_2 + Y_3$



Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	1	0
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

Priority encoders

4-to-2 encoder



$A_0 = Y_1 + Y_3$

$A_1 = Y_2 + Y_3$

Y_3	Y_2	Y_1	Y_0	A_1	A_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

the problem can be resolved by making
the encoder inputs unique, i.e., mutually exclusive,
so that none of the other entries of the full truth table can occur,

this can be accomplished by prioritizing the inputs and sending
the prioritized signals as inputs to the encoder

Priority encoders

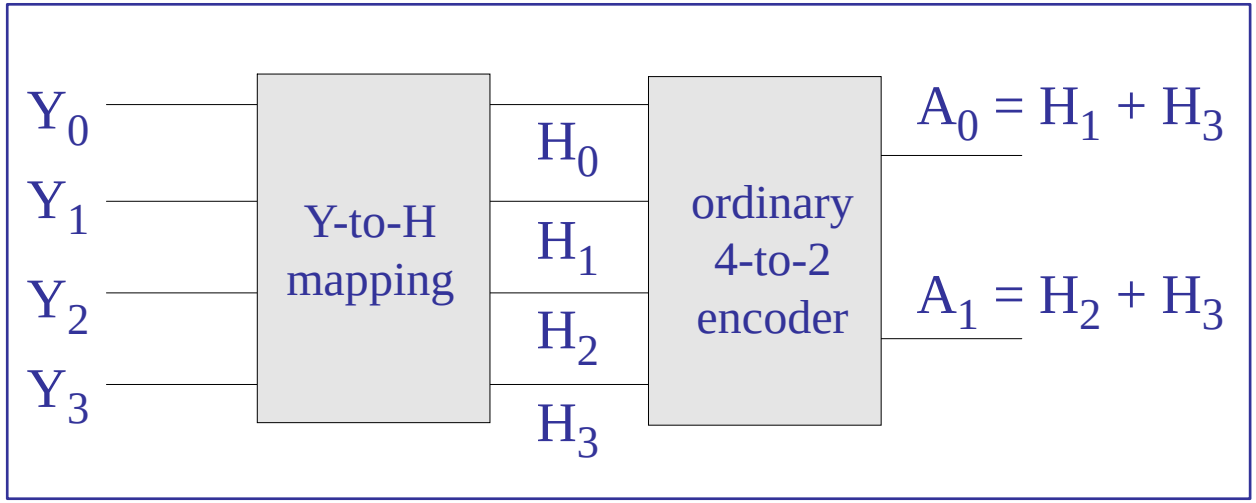
we assign highest priority to the input Y_3, and then to Y_2, Y_1, Y_0, and we construct the following high-priority signals, H_3, H_2, H_1, H_0, and then pass them to an ordinary 4-to-2 encoder to generate the encoded A_1, A_0 outputs

$$H_3 = Y_3$$

$$H_2 = Y_2 Y_3'$$

$$H_1 = Y_1 Y_2' Y_3'$$

$$H_0 = Y_0 Y_1' Y_2' Y_3'$$



if Y_3 is not ON, then Y_2 has next priority

if neither Y_3 nor Y_2 are ON, then Y_1 has next priority

and if none of Y_3,Y_2,Y_1 are ON, then Y_0 has next priority

Priority encoders

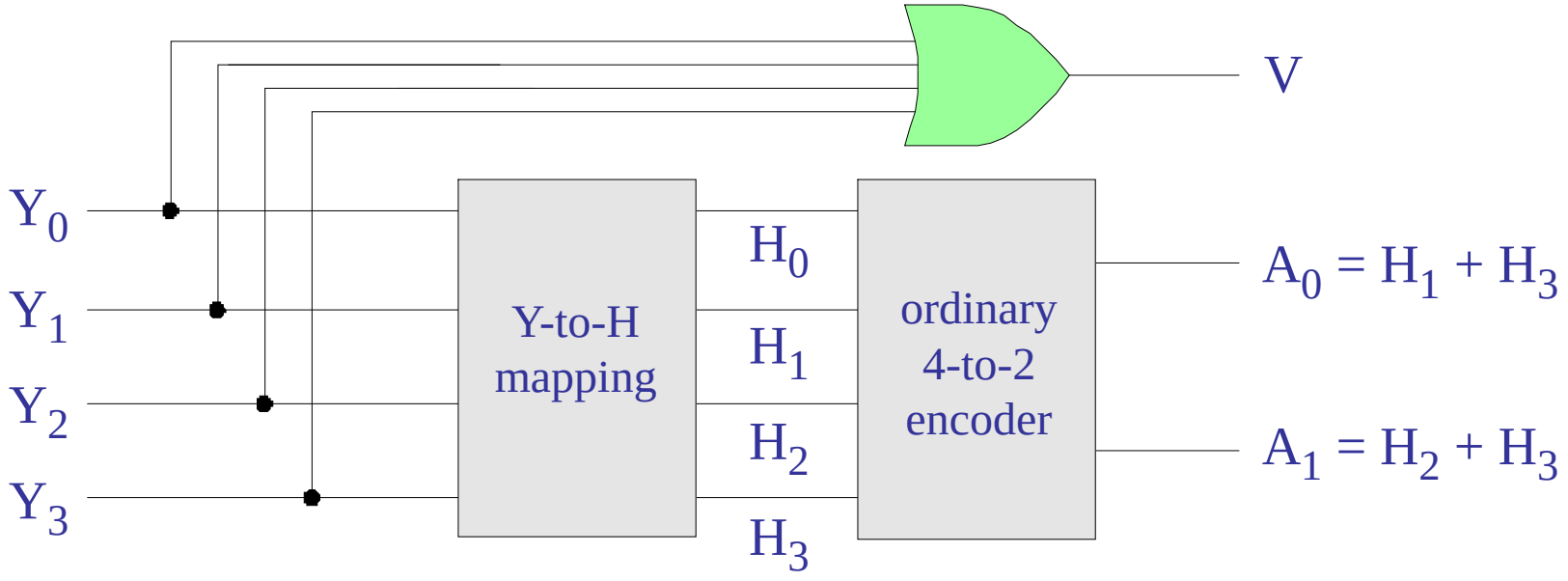
it is common also to define the "validation" signal

$$V = Y_3 + Y_2 + Y_1 + Y_0$$

that is asserted if any of the inputs is ON,

note that the Wakerly text uses active-low logic and uses the complement of V, called IDLE,

$$IDLE = V' = Y_3' Y_2' Y_1' Y_0'$$



further simplifications

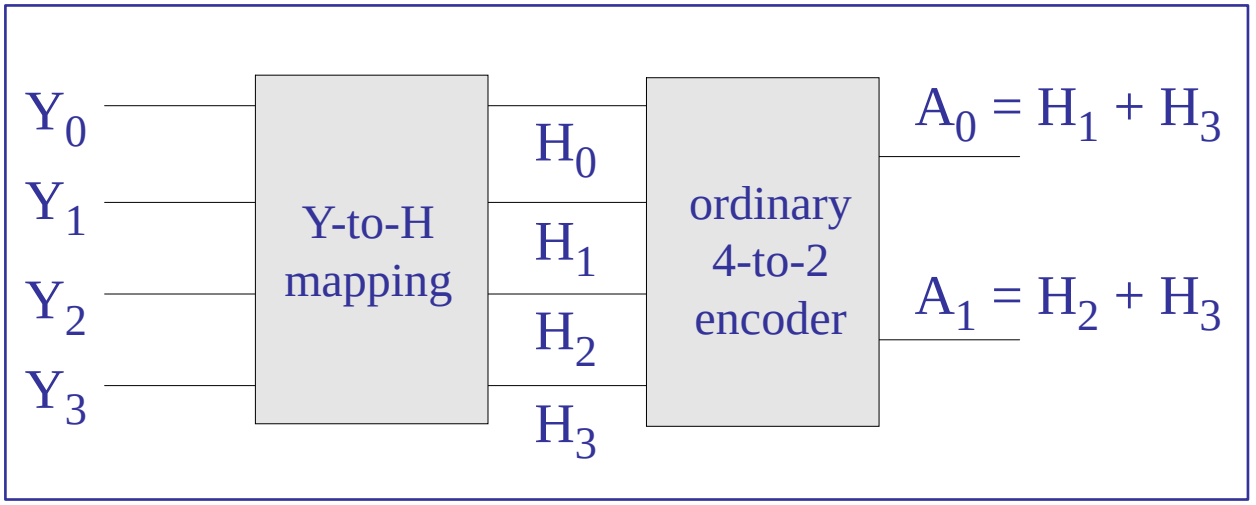
Priority encoders

$H_3 = Y_3$

$H_2 = Y_2\, Y_3'$

$H_1 = Y_1\, Y_2'\, Y_3'$

$H_0 = Y_0\, Y_1'\, Y_2'\, Y_3'$

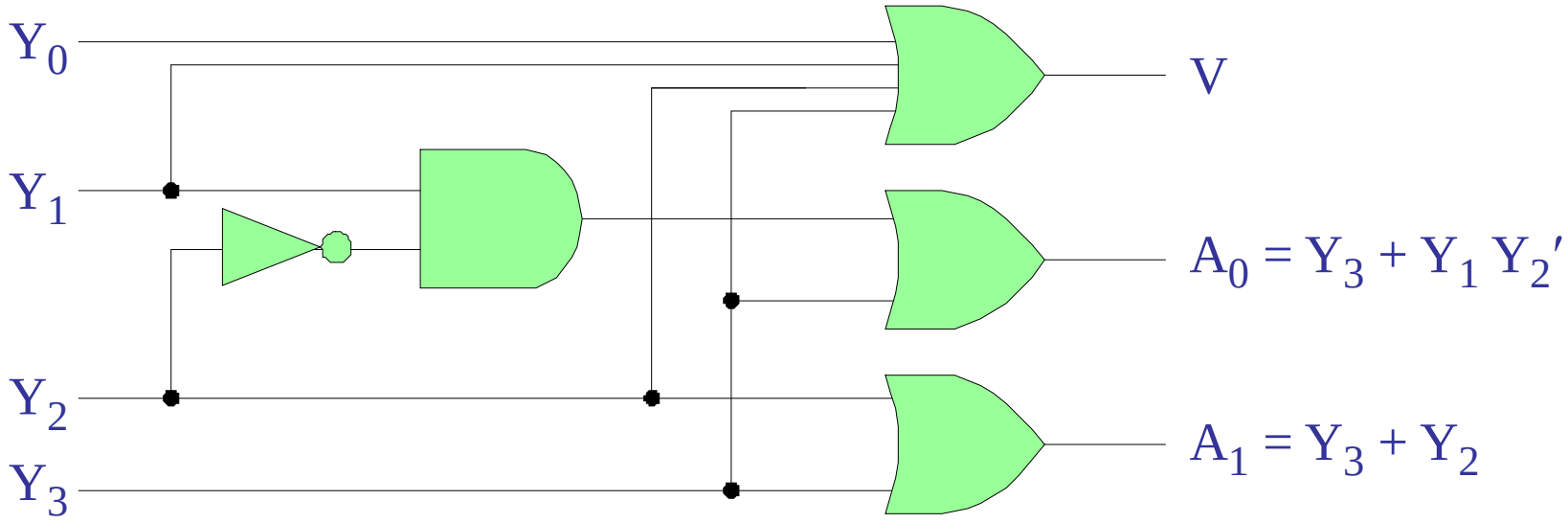


using the distributive property, $A + BC = (A+B)(A+C)$

$$A_0 = H_1 + H_3 = Y_3 + Y_1\, Y_2'\, Y_3' = (Y_3 + Y_3')\,(Y_3 + Y_1\, Y_2'\,) = Y_3 + Y_1\, Y_2'$$

$$A_1 = H_2 + H_3 = Y_3 + Y_2\, Y_3' = (Y_3 + Y_3'\,)\,(Y_3 + Y_2) = Y_3 + Y_2$$

gate-level realization of a 4-to-2 priority encoder

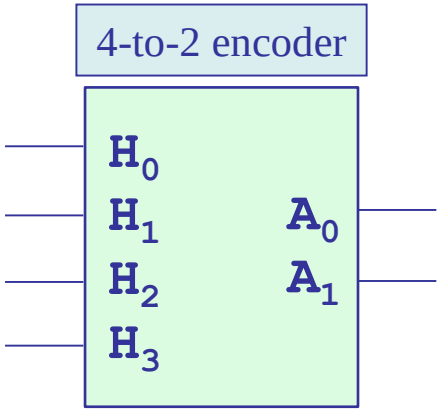


we may verify that this works as expected by computing the full truth table of the Y-inputs, H-inputs, and A-outputs, shown on the next page, and computed with MATLAB on p.89

Priority encoders

Y_3	Y_2	Y_1	Y_0	H_3	H_2	H_1	H_0	A_1	A_0	V
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	0	1	1
0	0	1	1	0	0	1	0	0	1	1
0	1	0	0	0	1	0	0	1	0	1
0	1	0	1	0	1	0	0	1	0	1
0	1	1	0	0	1	0	0	1	0	1
0	1	1	1	0	1	0	0	1	0	1
1	0	0	0	1	0	0	0	1	1	1
1	0	0	1	1	0	0	0	1	1	1
1	0	1	0	1	0	0	0	1	1	1
1	0	1	1	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0	1	1	1
1	1	0	1	1	0	0	0	1	1	1
1	1	1	0	1	0	0	0	1	1	1
1	1	1	1	1	0	0	0	1	1	1

H-outputs
are exclusive, and
correspond to a plain
4-to-2 binary encoder



H_3	H_2	H_1	H_0	A_1	A_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

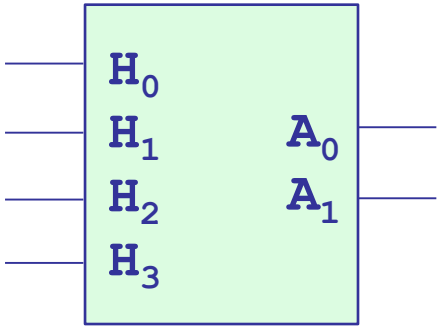
Priority encoders

compressed truth table

Y_3	Y_2	Y_1	Y_0	H_3	H_2	H_1	H_0	A_1	A_0	V
0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	0	0	1
0	0	1	x	0	0	1	0	0	1	1
0	1	x	x	0	1	0	0	1	0	1
1	x	x	x	1	0	0	0	1	1	1

H-outputs
are exclusive, and
correspond to a plain
4-to-2 binary encoder

4-to-2 encoder



if Y_3 is ON, then, $A_1A_0 = 11$, regardless of the values of $Y_2\ Y_1\ Y_0$

if Y_2 is ON, but Y_3 is OFF, then, $A_1A_0 = 10$, regardless of the values of $Y_1\ Y_0$

if Y_1 is ON, but Y_2 and Y_3 are OFF, then, $A_1A_0 = 01$, regardless of the values of Y_0

if onlyY_0 is ON, then, $A_1A_0 = 00$

H_3	H_2	H_1	H_0	A_1	A_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

Priority encoders

```
% 4-to-2 priority encoder truth table

[Y3,Y2,Y1,Y0] = a2d(0:15, 4);          % inputs

H3 = Y3;                               % intermediate
H2 = Y2 & (~Y3);                       % inputs to
H1 = Y1 & (~Y2) & (~Y3);               % plain 4-to-2
H0 = Y0 & (~Y1) & (~Y2) & (~Y3);       % binary encoder

A1 = H2 | H3;                          % outputs
A0 = H1 | H3;

% A1 = Y2 | Y3;                        % alternative
% A0 = (Y1 & (~Y2)) | Y3;              % calculation

V = Y0 | Y1 | Y2 | Y3;                 % valid output

[Y3,Y2,Y1,Y0,H3,H2,H1,H0,A1,A0,V]      % truth table
```

the construction of higher order priority encoders is straightforward.

For example, in the 8-to-3 case, the high-priority signals are constructed as follows, assigning higher to lower priority in the order of,
$Y_7, Y_6, Y_5, Y_4, Y_3, Y_2, Y_1, Y_0$,

8-to-3 priority encoder

$$H_7 = Y_7$$

$$H_6 = Y_6 Y_7'$$

$$H_5 = Y_5 Y_6' Y_7'$$

$$H_4 = Y_4 Y_5' Y_6' Y_7'$$

$$H_3 = Y_3 Y_4' Y_5' Y_6' Y_7'$$

$$H_2 = Y_2 Y_3' Y_4' Y_5' Y_6' Y_7'$$

$$H_1 = Y_1 Y_2' Y_3' Y_4' Y_5' Y_6' Y_7'$$

$$H_0 = Y_0 Y_1' Y_2' Y_3' Y_4' Y_5' Y_6' Y_7'$$

$$V = Y_0 + Y_1 + Y_2 + Y_3 + Y_4 + Y_5 + Y_6 + Y_7$$

standard 8-to-3 encoder from p.72

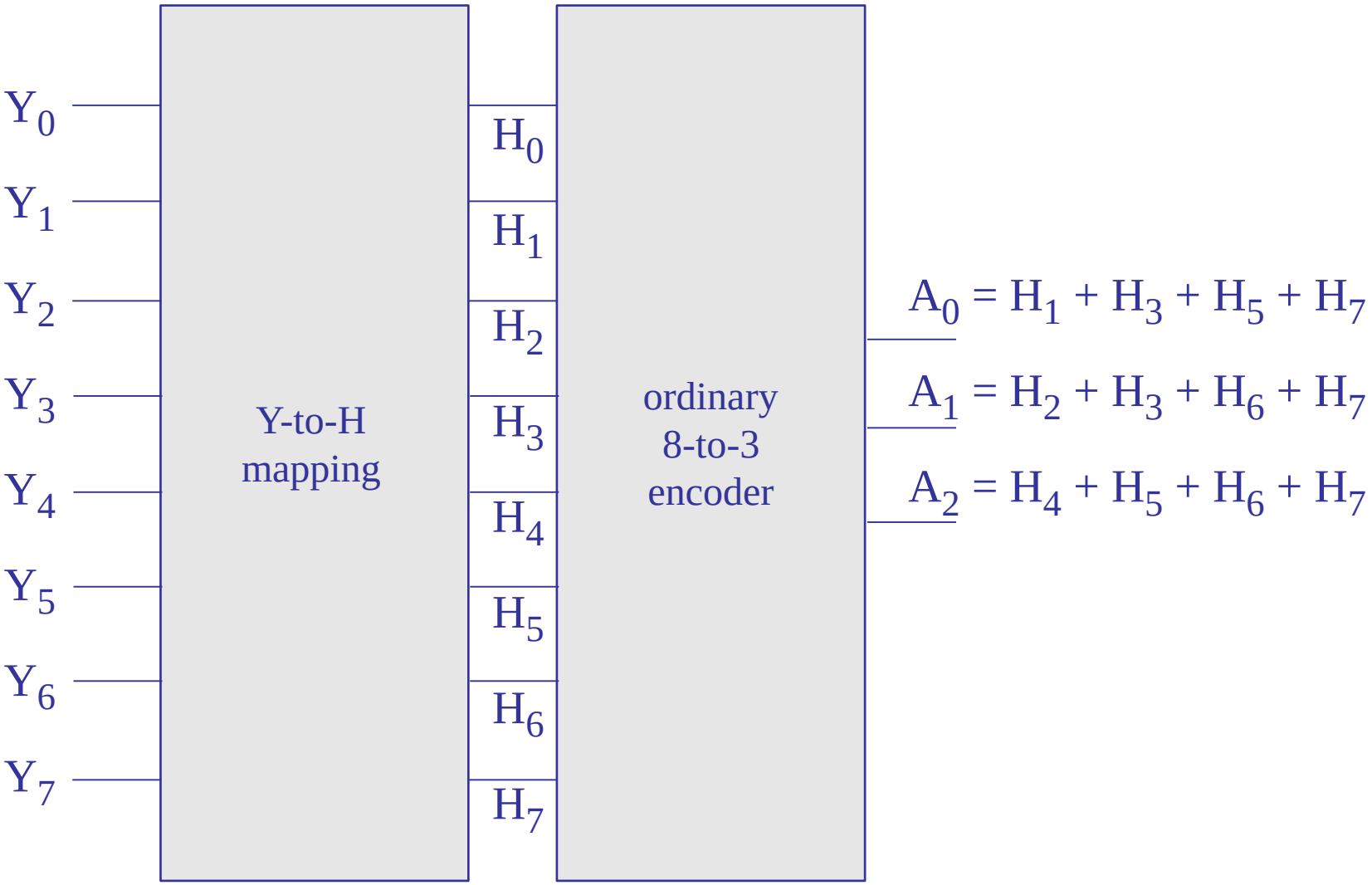
$$A_0 = Y_1 + Y_3 + Y_5 + Y_7$$

$$A_1 = Y_2 + Y_3 + Y_6 + Y_7$$

$$A_2 = Y_4 + Y_5 + Y_6 + Y_7$$

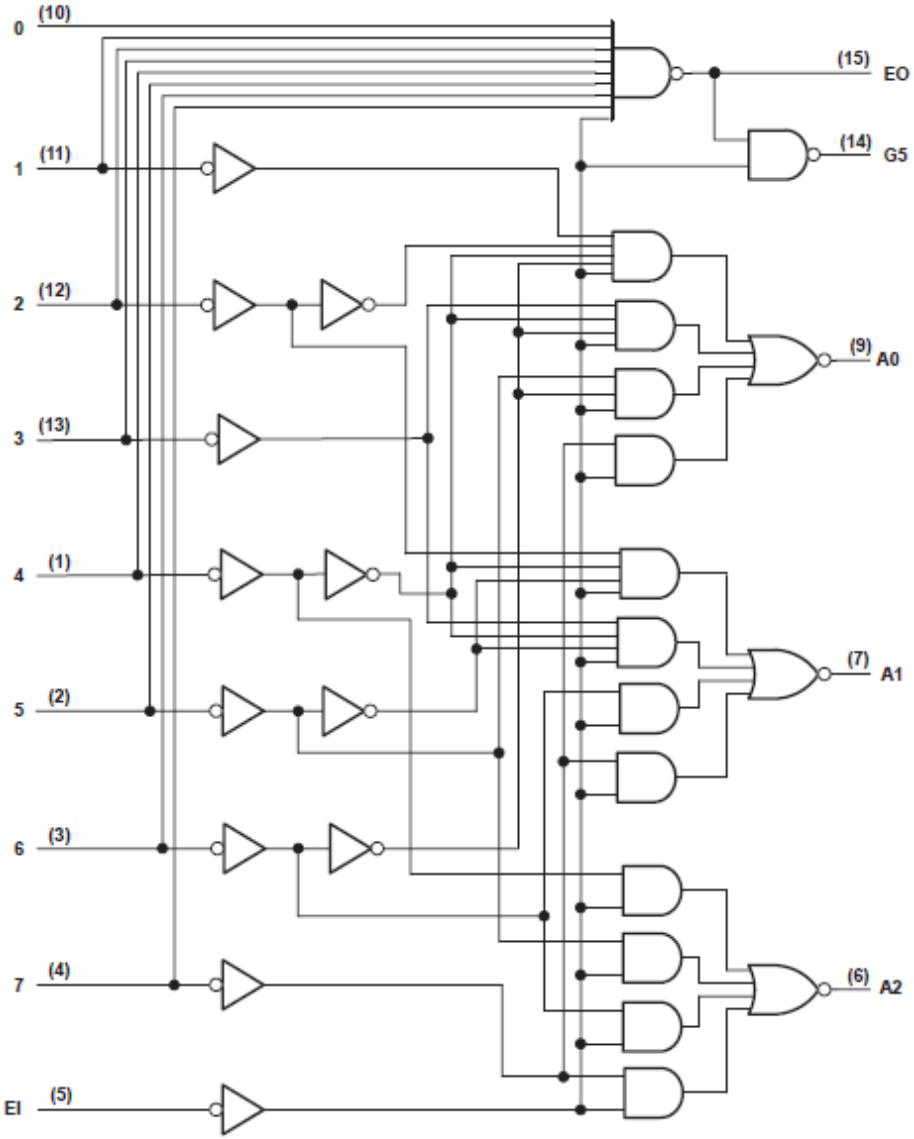
Priority encoders

8-to-3 priority encoder



Priority encoders

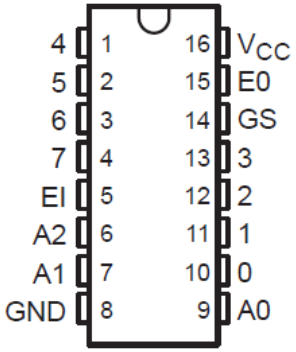
'148, 'LS148 logic diagram (positive logic)



Pin numbers shown are for D, J, N, NS, and W packages.

74148 chip family

SN74148, SN74LS148 . . . D, N, OR NS PACKAGE
(TOP VIEW)



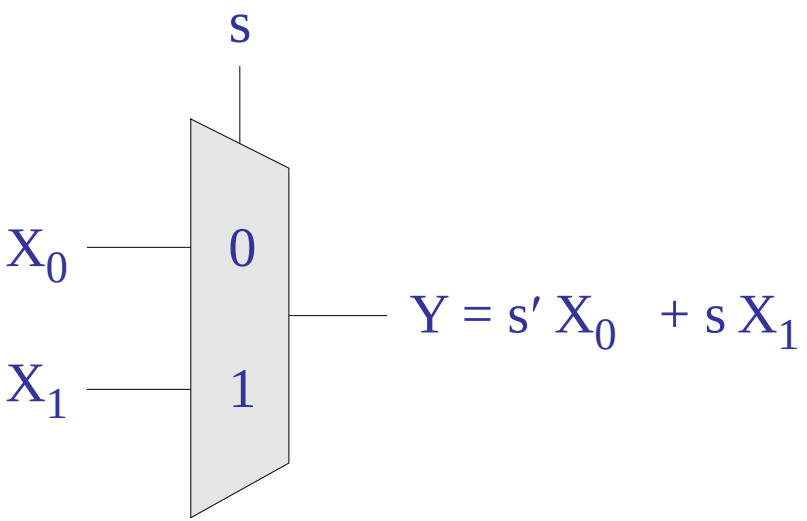
active-low

Multiplexers

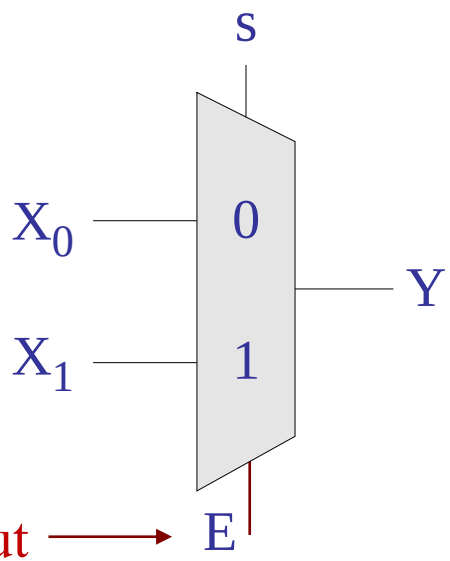
2-to-1 multiplexer

s	Y
0	X_0
1	X_1

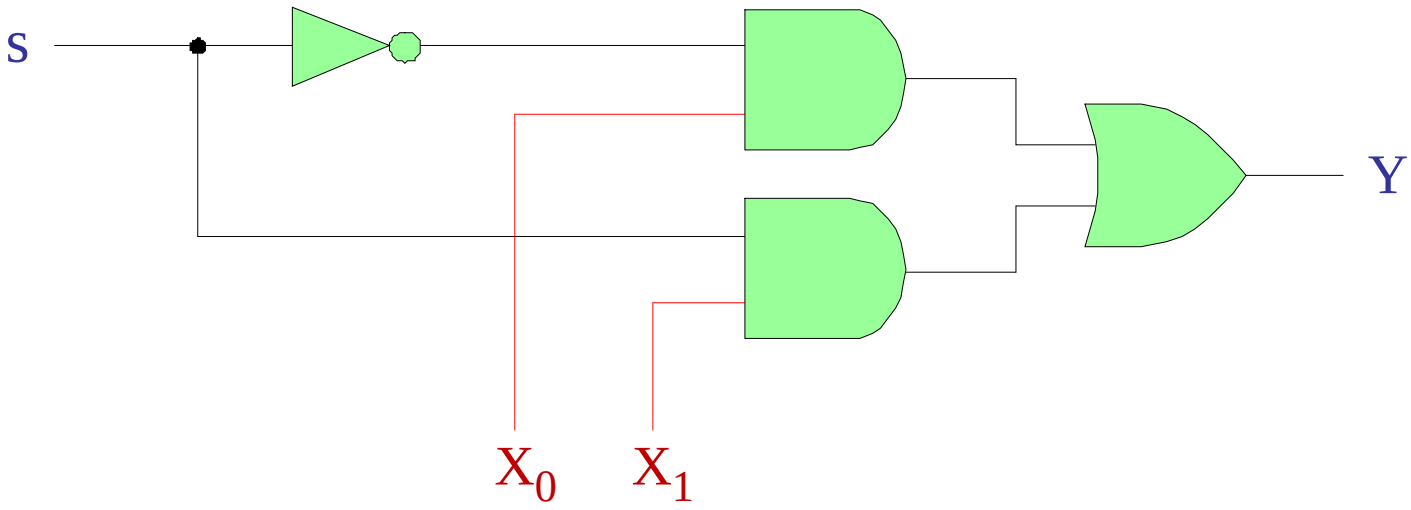
simplified truth table



alternative symbol



2-to-1 multiplexer gate-level realization



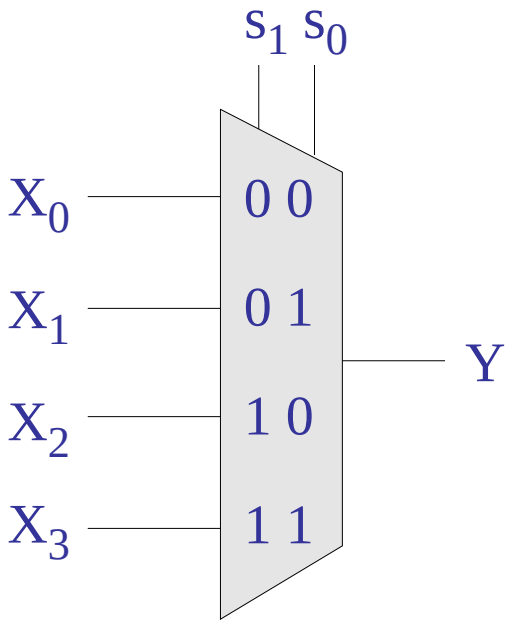
$$Y = s' X_0 + s X_1$$

4-to-1 multiplexer

$$Y = s_1's_0' X_0 + s_1's_0 X_1 + s_1s_0' X_2 + s_1s_0 X_3$$

s_1	s_0	Y
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3

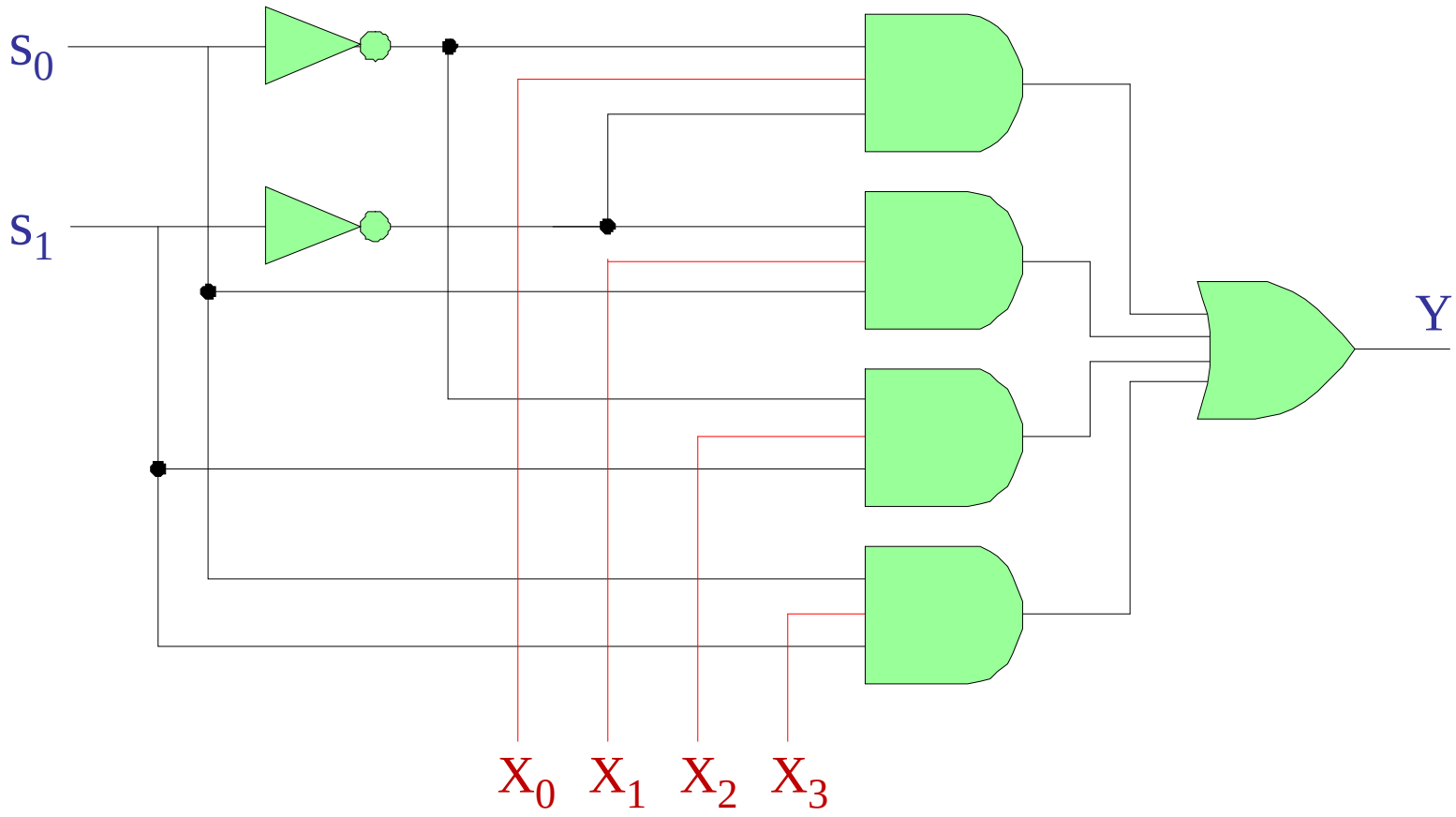
simplified truth table



alternative symbol

can also have an enable input

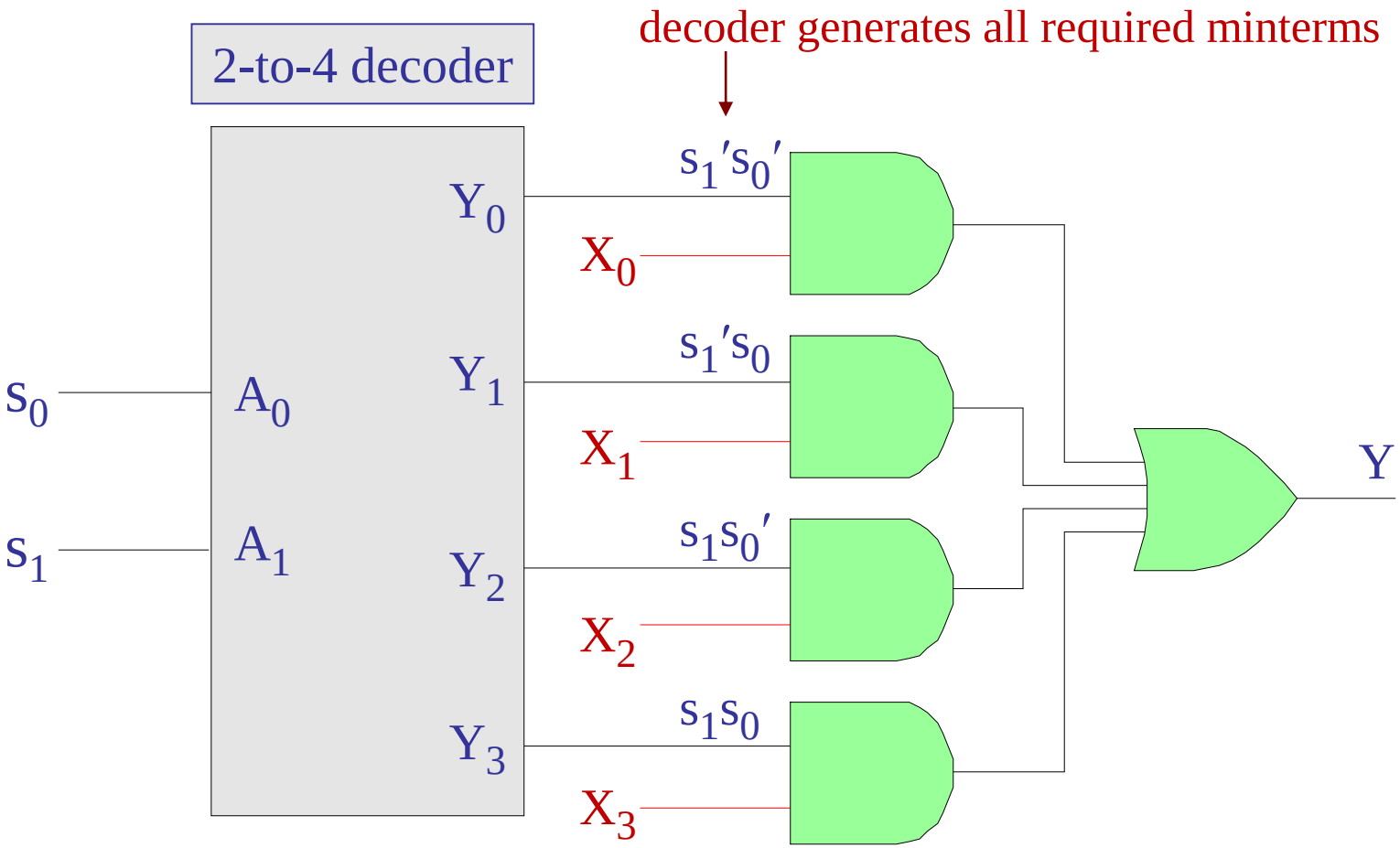
4-to-1 multiplexer gate-level realization



$$Y = s_1's_0' X_0 + s_1's_0 X_1 + s_1s_0' X_2 + s_1s_0 X_3$$

multiplexer realization with decoders

Multiplexers

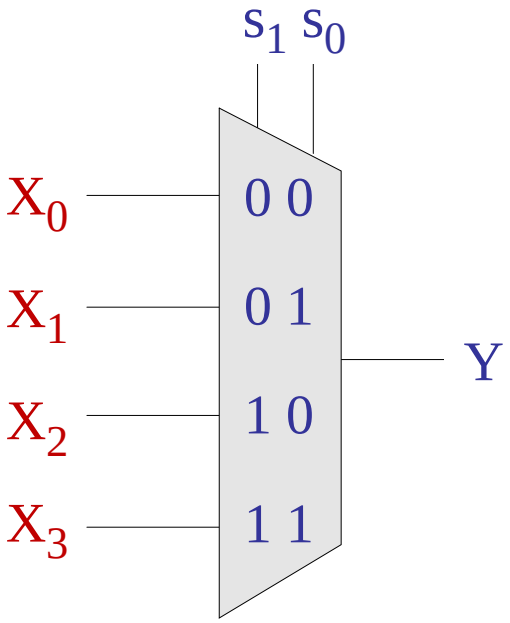


$$Y = s_1's_0' X_0 + s_1's_0 X_1 + s_1s_0' X_2 + s_1s_0 X_3$$

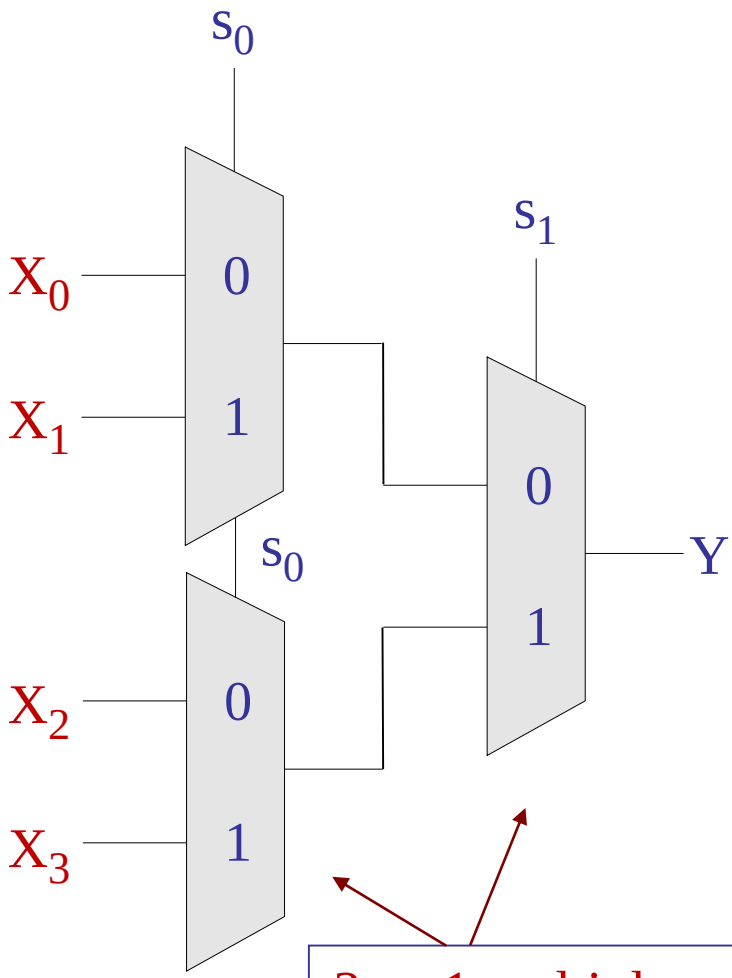
realizing larger multiplexers from smaller ones

Multiplexers

s_1	s_0	Y
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3



4-to-1 multiplexer



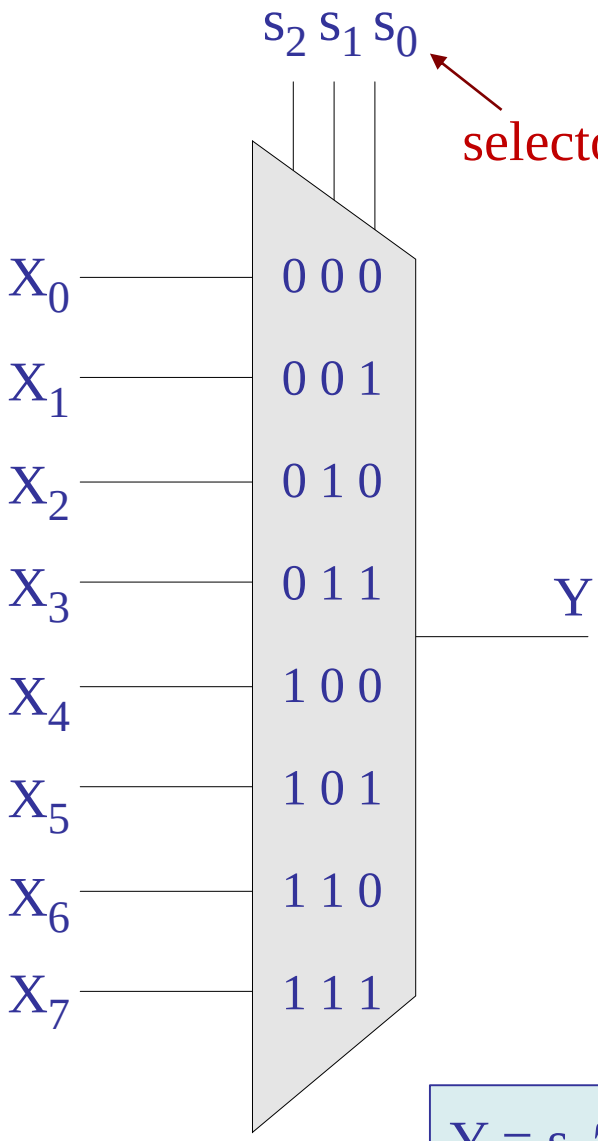
2-to-1 multiplexers

$Y = s_1' s_0' X_0 + s_1' s_0 X_1 + s_1 s_0' X_2 + s_1 s_0 X_3$

discussed further in recitations

Multiplexers

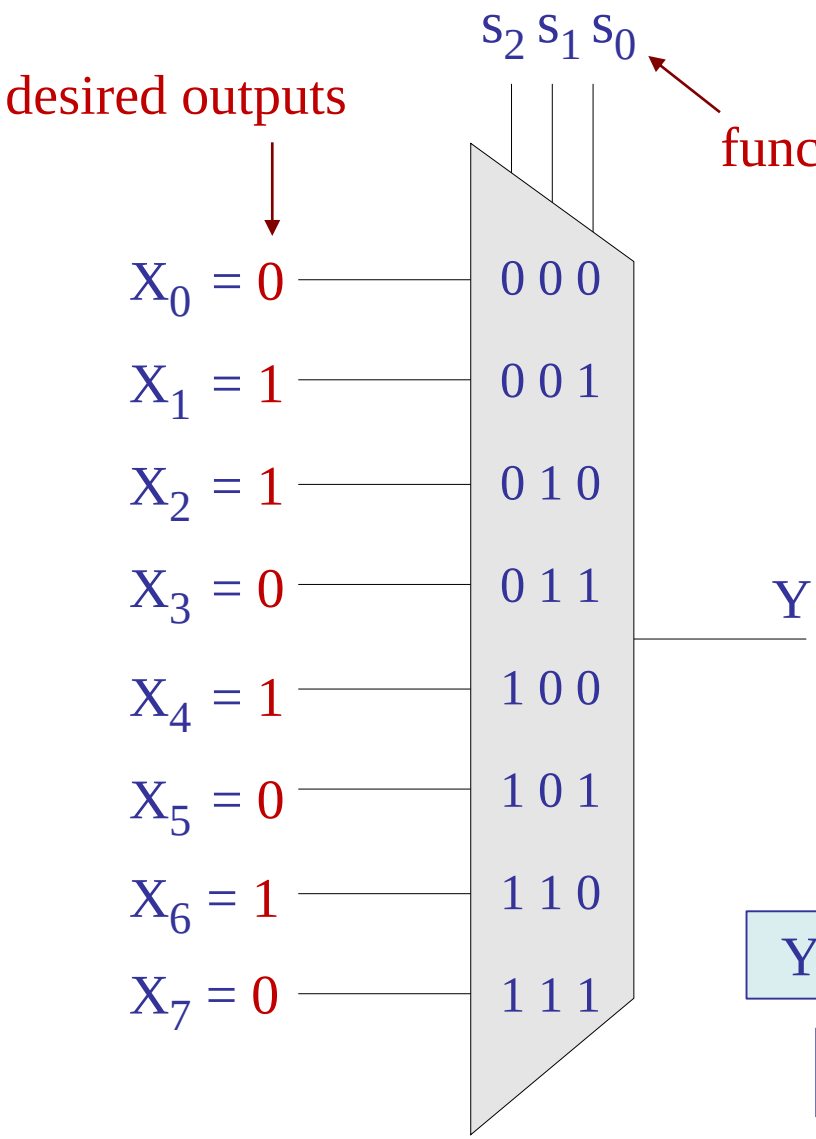
8-to-1 multiplexer



$$Y = s_2's_1's_0' X_0 + s_2's_1's_0X_1 + s_2's_1s_0' X_2 + s_2's_1s_0X_3 + s_2s_1's_0' X_4 + s_2s_1's_0X_5 + s_2s_1s_0' X_6 + s_2s_1s_0X_7$$

realizing combinational functions with multiplexers

Multiplexers

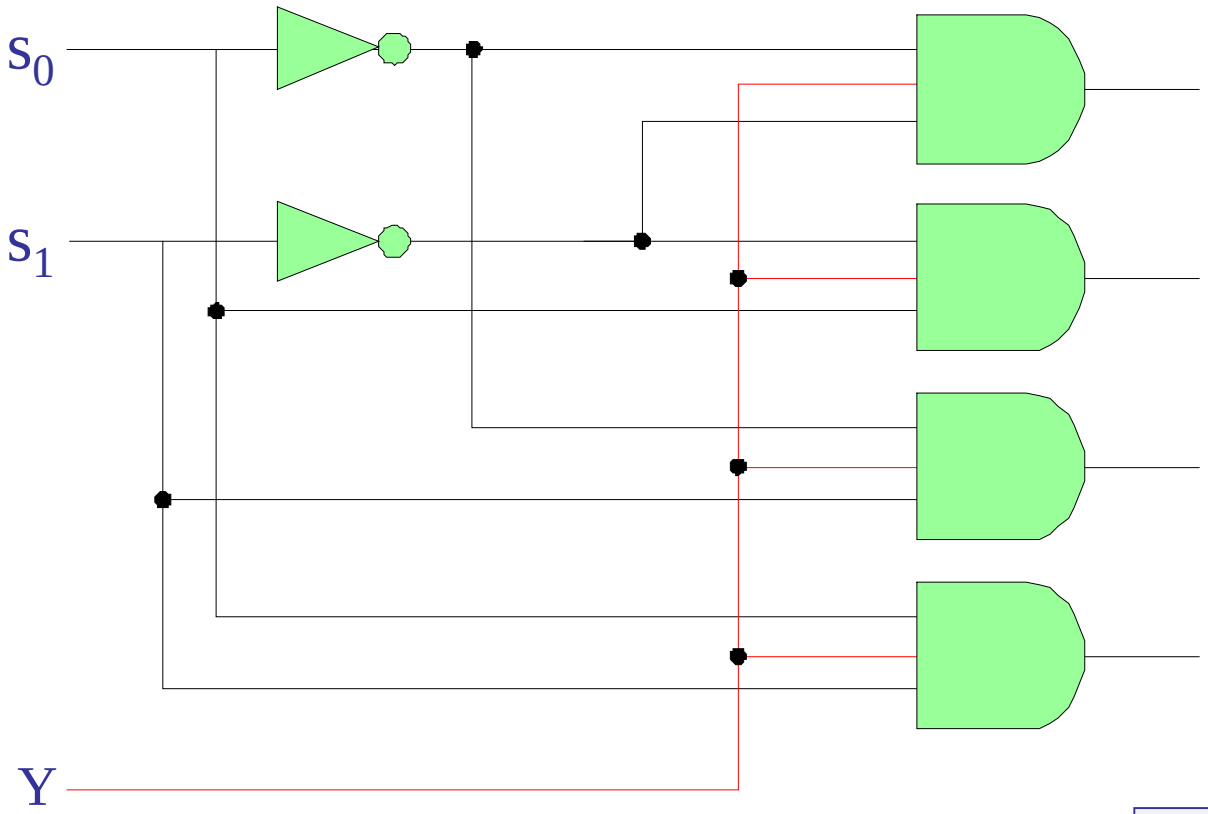


s_2	s_1	s_0	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$$Y = s_2' s_1' s_0 + s_2' s_1 s_0' + s_2 s_1' s_0' + s_2 s_1 s_0'$$

apply the truth table values to the inputs

Demultiplexers

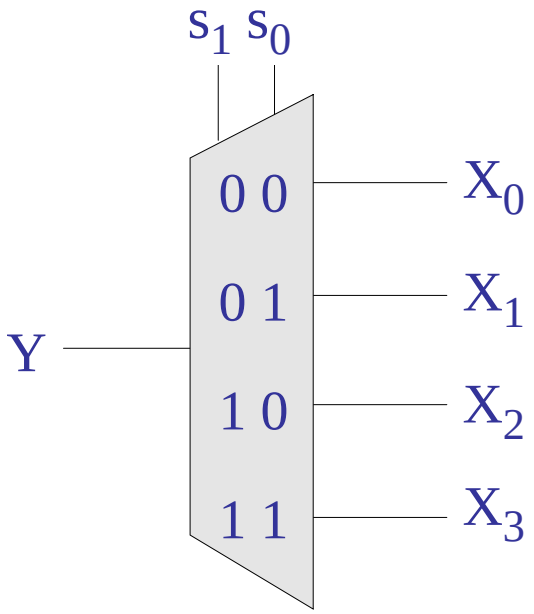


s_1	s_0	X_0	X_1	X_2	X_3
0	0	Y	0	0	0
0	1	0	Y	0	0
1	0	0	0	Y	0
1	1	0	0	0	Y

Demultiplexers

s_1	s_0	X_0	X_1	X_2	X_3
0	0	Y	0	0	0
0	1	0	Y	0	0
1	0	0	0	Y	0
1	1	0	0	0	Y

demultiplexer truth table



alternative symbol

can also have an “enable” input

demultiplexer realization with decoders