Digital Logic Design

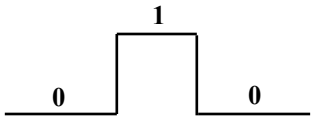
Hajar Falahati

Department of Computer Engineering
IRAN University of Science and Technology

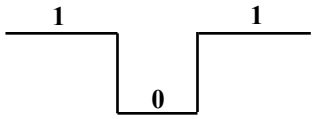
hfalahati@iust.ac.ir

Hazard

- Static Hazard
  - Two-level SOP (AND-OR) circuit
    - It may have static-1 hazards
  - Two-level POS (OR-AND) circuit
    - It may have static-0 hazards

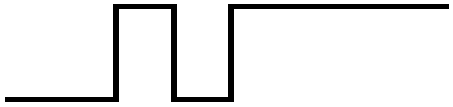


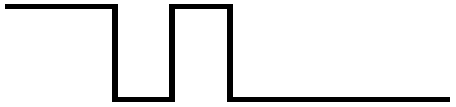
Static-0 Hazard



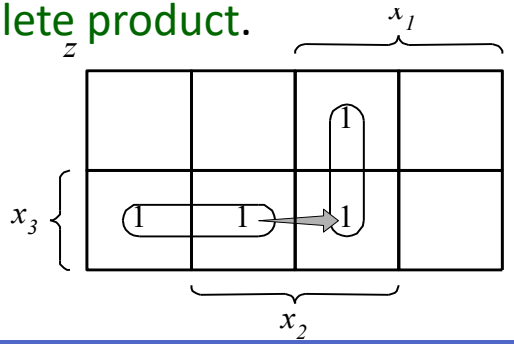
Static-1 Hazard

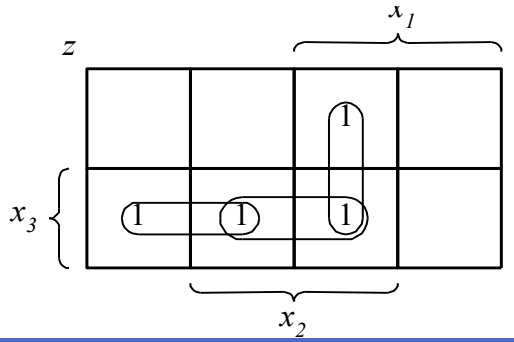
- Dynamic Hazard


(a)


(b)

- Hazard-free realization
  - Use the complete sum or complete product.
  - Do not cover don't cares.





Outline

- Hierarchical Design
- Combinational Logic
- Famous Combinational Logic

Implementation

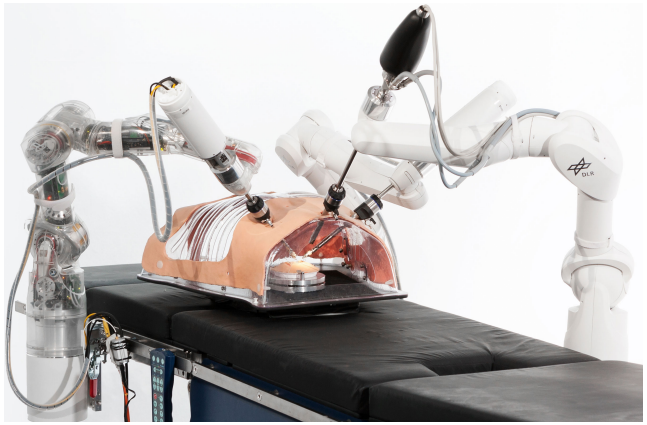
- Digital circuit **design**:
  - Specification
  - Formulation and truth table
  - Optimization
  - Word description of a function

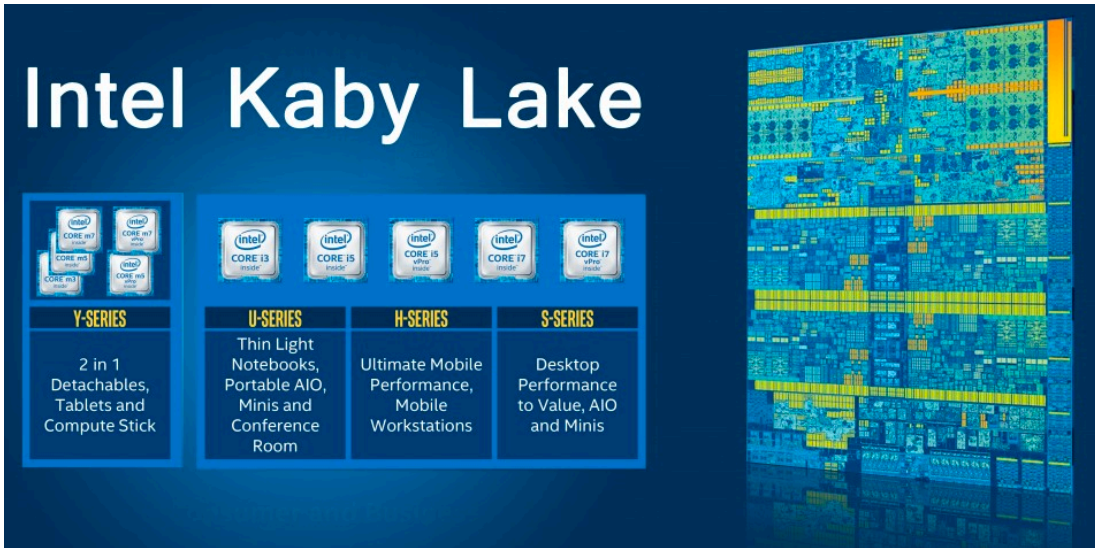
    $\Rightarrow$a set of switching equations
  - Hardware realization
    - Technology Mapping

- Gate-level design
  - Connect Logic gates

Complex Design

- How about complex design?





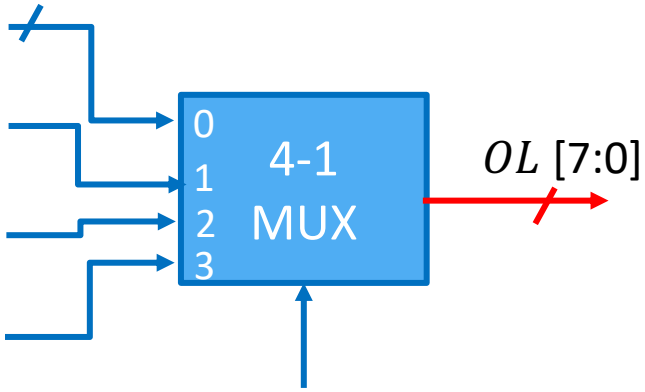
- 64- bit processor
- 4 cores, 8 threads
- 1.75B transistor
- 14 nm

Hierarchical Design

Logic Components

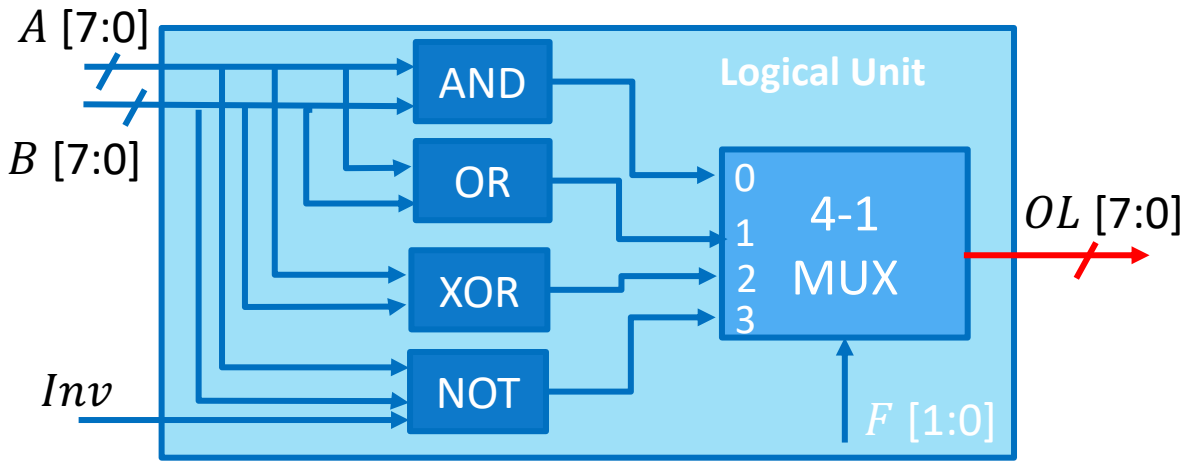
- Logic component or blocks
  - Constructed via logic gates
  - A set of *m* boolean inputs
  - A set of *n* boolean outputs
  - *n* switching functions
    - Each mapping the 2^m input combinations to an output
    - Current output depends only on the current input values

- Types
  - Combinational
    - Does not have memory
  - Sequential
    - Has memory



Component-Level Design

- Connect logic components
- Control complexity of function



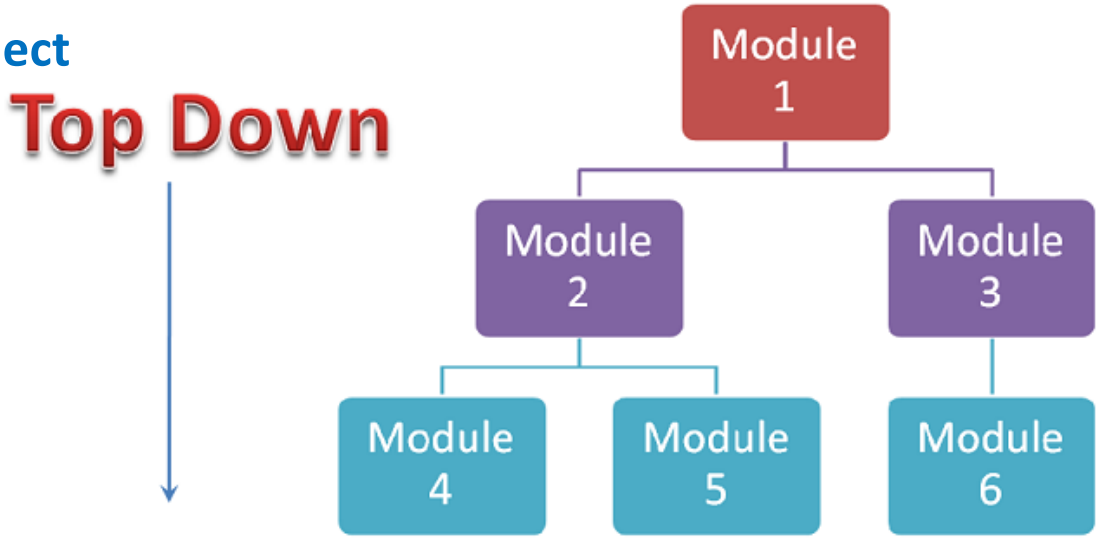
Hierarchical Design

- Step 1. Decompose the function into smaller pieces, a.k.a., *blocks*
- Step2: Decompose each block's function into smaller blocks
- Repeat step2 until all resulted blocks are small enough

- Primitive Block
  - A block that can not be decomposed

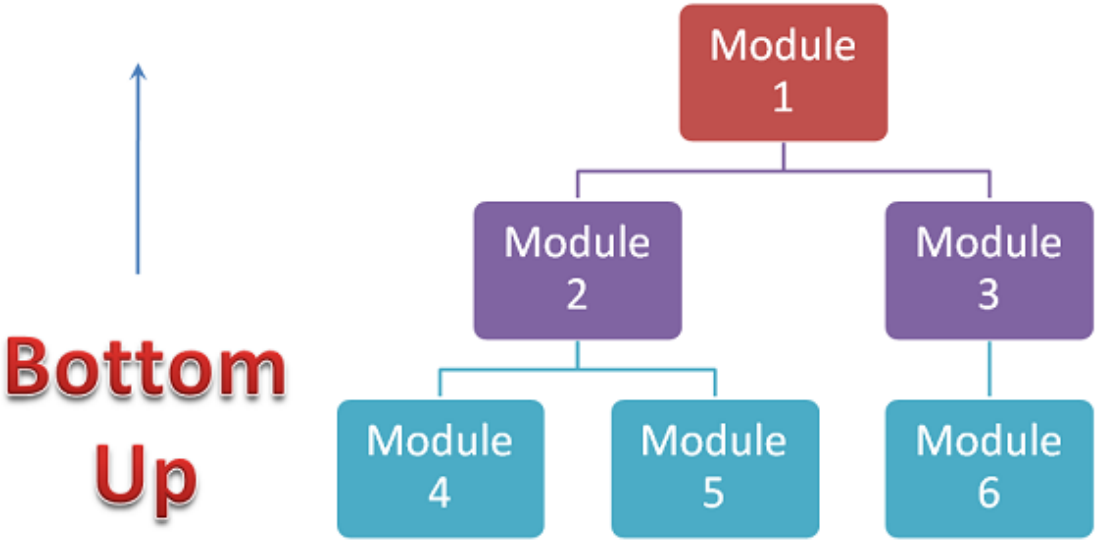
Top-Down

- To gain **insights** into **its** **compositional sub-systems**
  - Reverse engineering fashion
- **Extensive planning** and **research phase**
  - -> Leads to development of a product
- **Structural control of a project**
- **More straightforward**

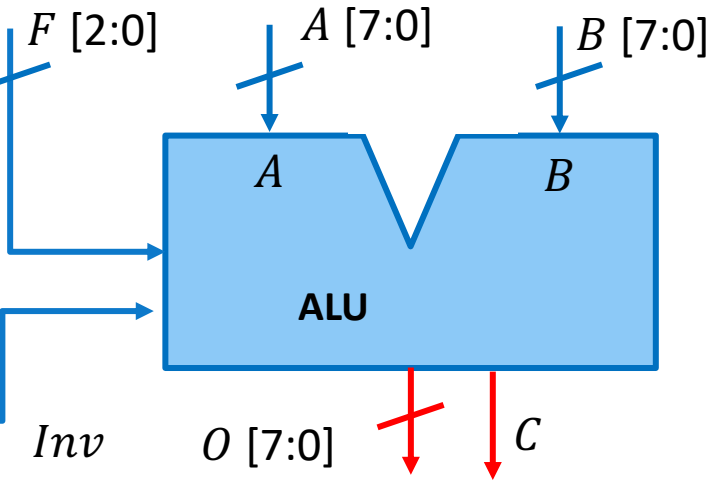


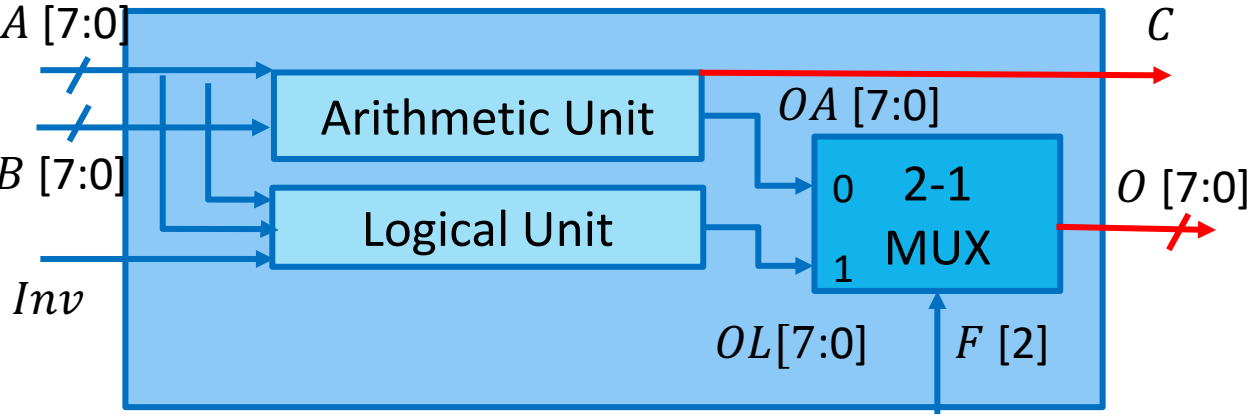
Buttom-Up

- Piecing together of systems to give rise to more complex systems
- More **optimized**
- More **realistic**

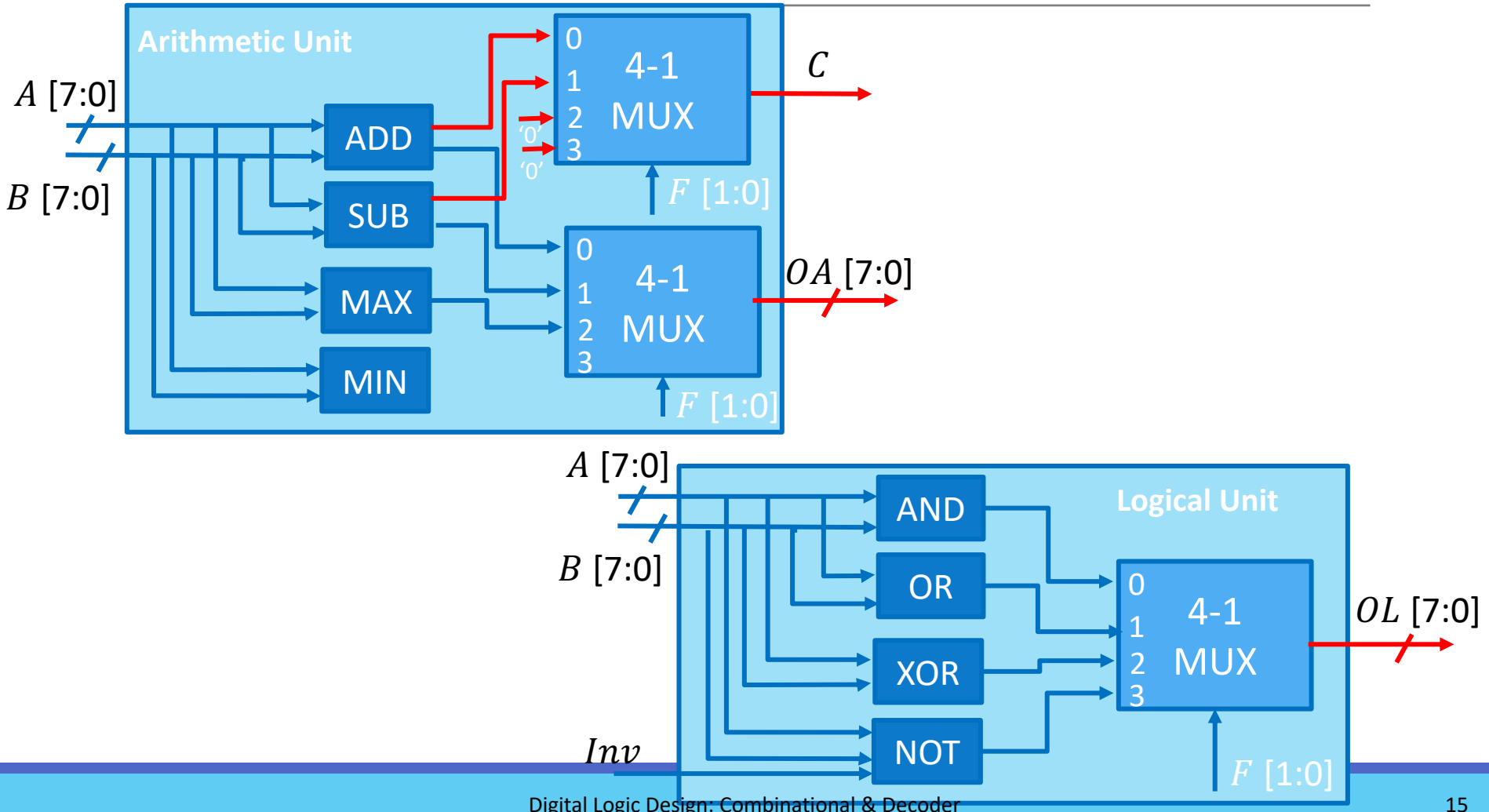


ALU


F [2:0]
A [7:0]
B [7:0]
A
B
ALU
Inv
O [7:0]
C


A [7:0]
B [7:0]
Inv
Arithmetic Unit
Logical Unit
OA [7:0]
OL[7:0]
0
1
2-1
MUX
F [2]
C
O [7:0]

ALU…


Arithmetic Unit
A [7:0]
B [7:0]
ADD
SUB
MAX
MIN
0
1
2
3
4-1
MUX
'0'
'0'
F [1:0]
C
OA [7:0]
Logical Unit
AND
OR
XOR
NOT
Inv
OL [7:0]

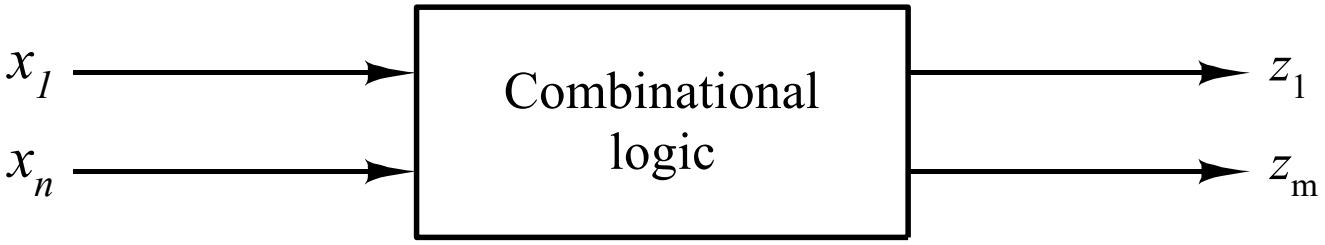
Combinational Logics

Digital Logic Design: Combinational & Decoder 22

Combinational Logics

- **Always Current inputs** produce the **output**
- **Output** is **independent** of
  - **Sequence of inputs**
  - **Time of applying inputs**

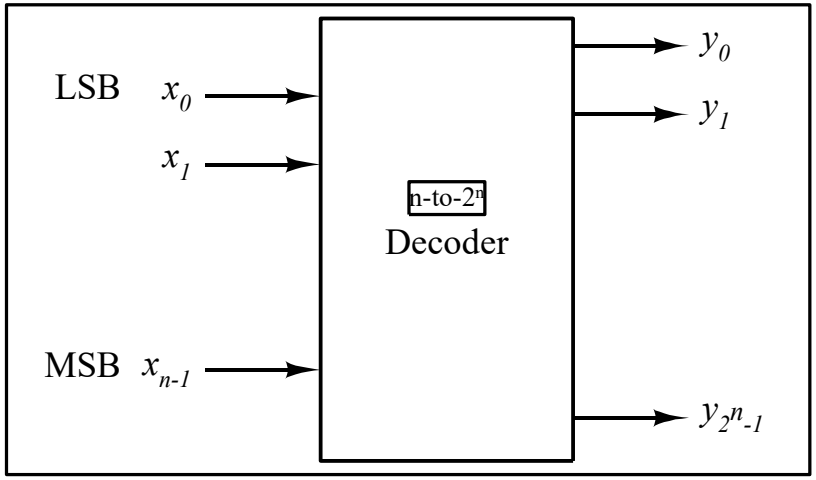
- => **Combinational logics** are memory less
  - **Memory-less circuits** do not contain any **feedback** lines



Decoder

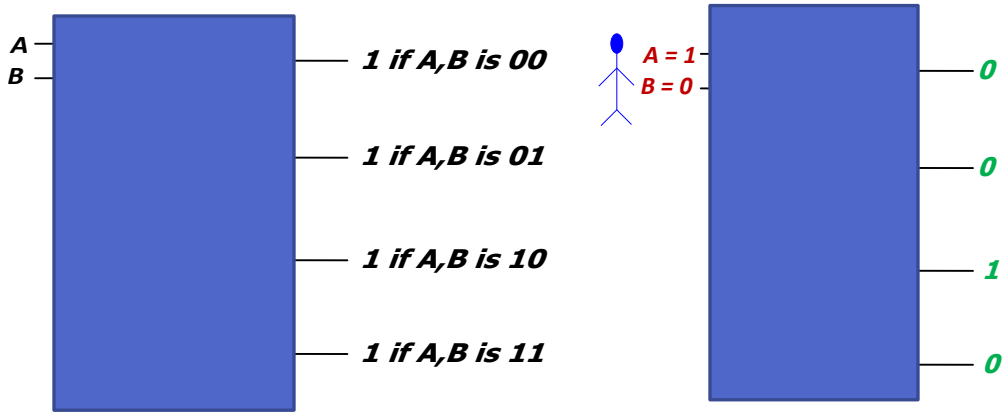
Decoder

- n inputs and 2^n outputs
- Exactly one of the outputs is 1 and all the rest are 0s
- Function
  - Decoding
  - Each valid code word produces a unique output code



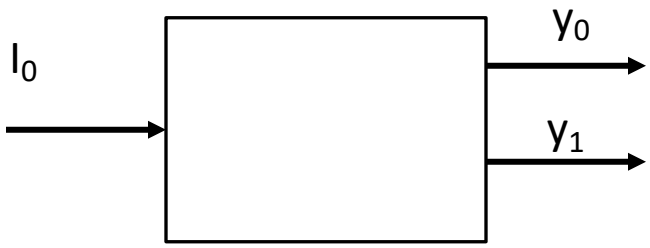
Decoder Implementation

- The one output that is **logically 1** is the output corresponding to the **input pattern** that the **logic circuit is expected to detect**
- Generate 2^n (or fewer) minterms for the *n* input variables



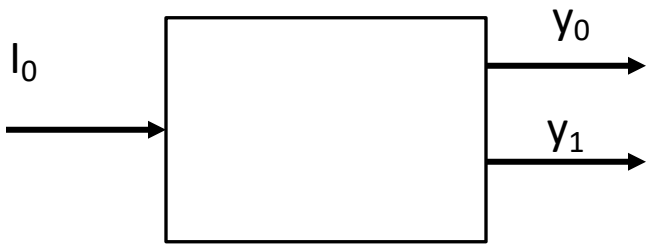
Decoder 1-2

- Input
  - 1 bit
- Output
  - 2 bit

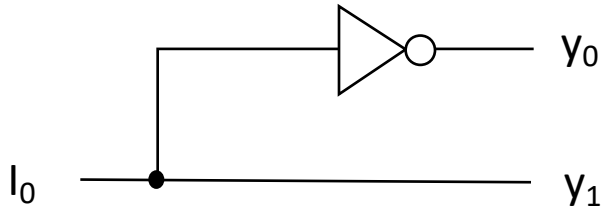


Decoder 1-2

- Input
  - 1 bit
- Output
  - 2 bit

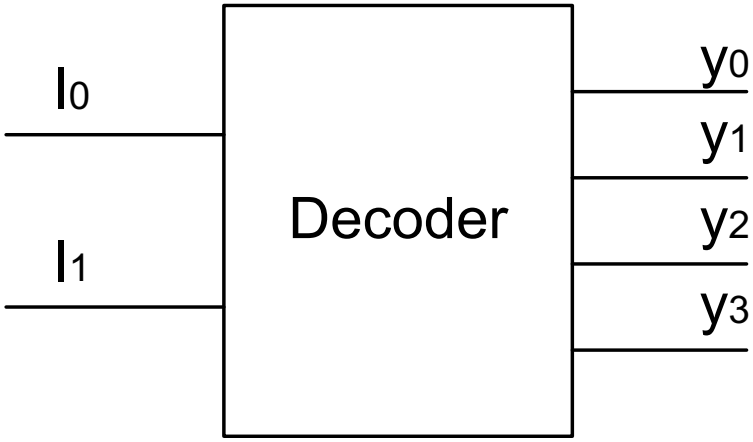


I_0	y_0	y_1
0	1	0
1	0	1



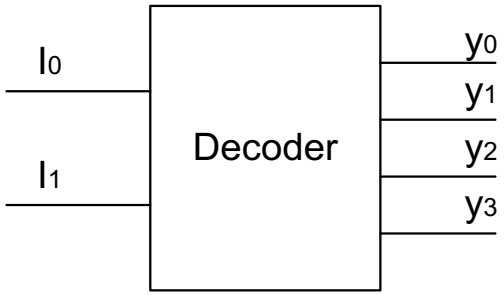
Decoder 2-4

- Input
  - 2 bit
- Output
  - 4-bit



Decoder 2-4

- Input
  - 2 bit
- Output
  - 4-bit

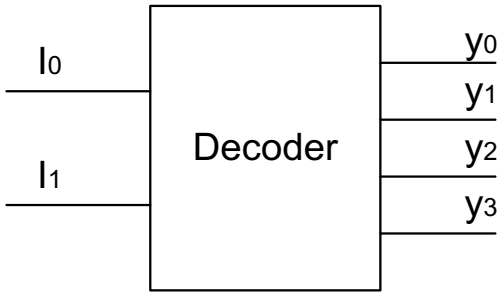


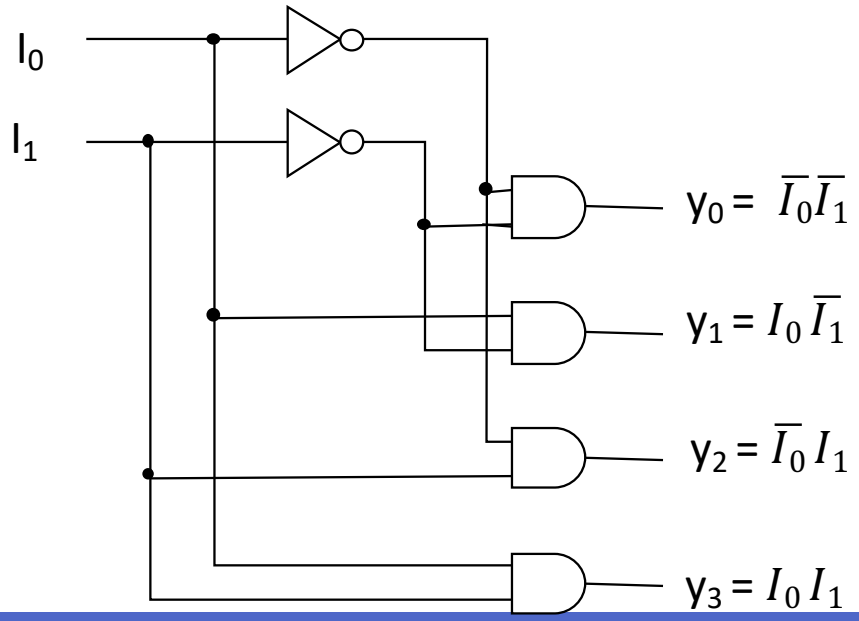
I_1	I_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoder 2-4

- Input
  - 2 bit
- Output
  - 4-bit

I_1	I_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



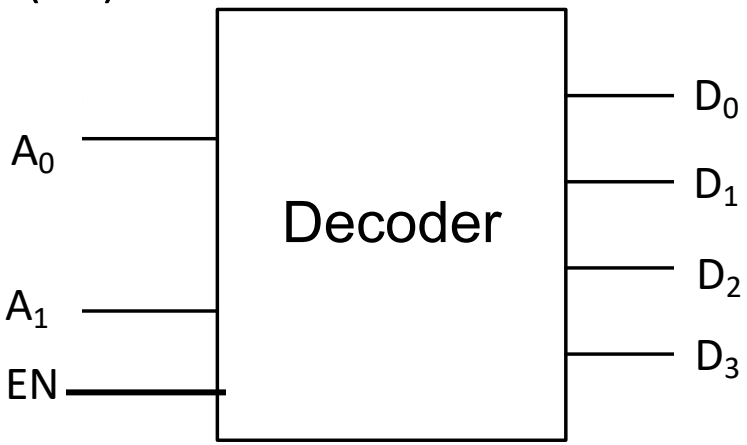


Decoder with Enable

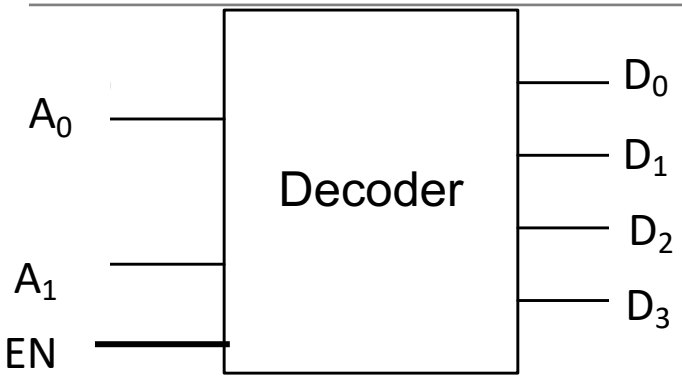
- Input: n-bit code and 1-bit enable (EN)
- Output: m-bit; $n \leq m \leq 2^n$
- Function
  - If EN=0 do nothing
  - If EN=1 decoding

Decoder with Enable: 2-4

- Input: n-bit code and 1-bit enable (EN)
- Output: m-bit; $n \leq m \leq 2^n$
- Function
  - If EN=0 do nothing
  - If EN=1 decoding

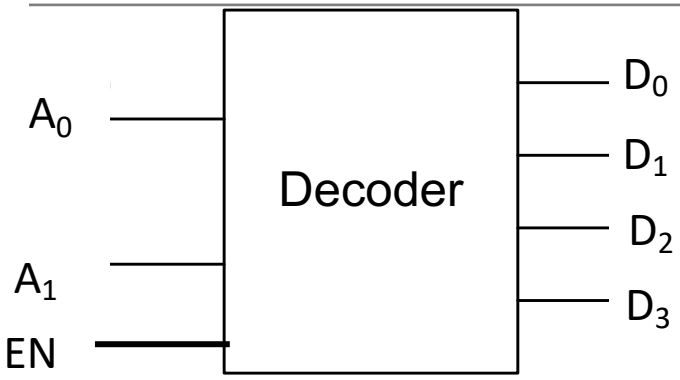


Decoder with Enable: 2-4 (cont'd)

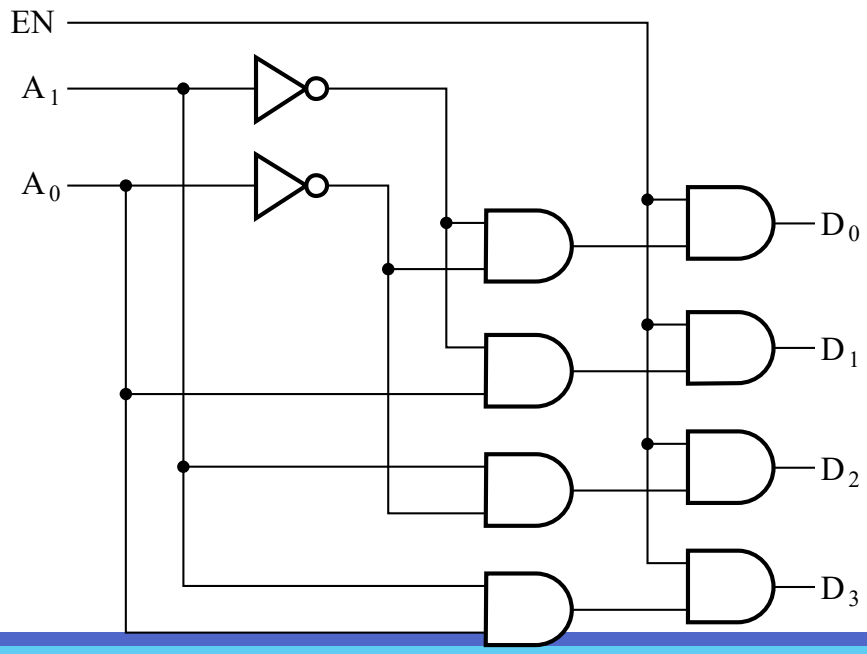


EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

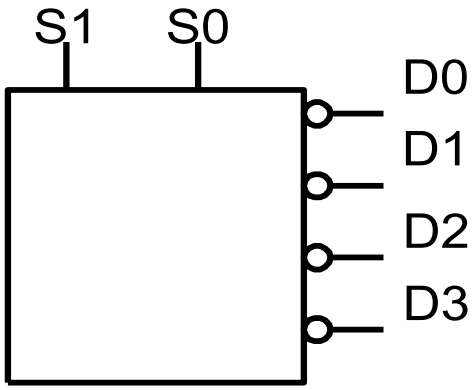
Decoder with Enable: 2-4 (cont'd)



EN	A_1	A_0	D_0	D_1	D_2	D_3
0	X	X	0	0	0	0
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

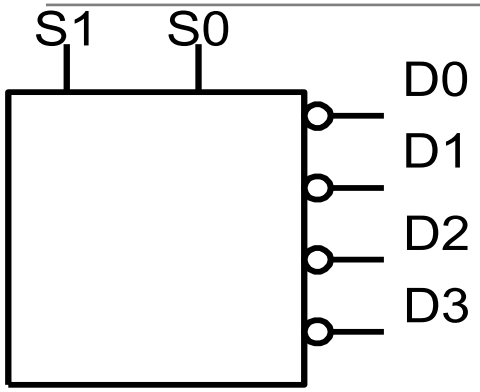


Decoder 2-4: Active Low

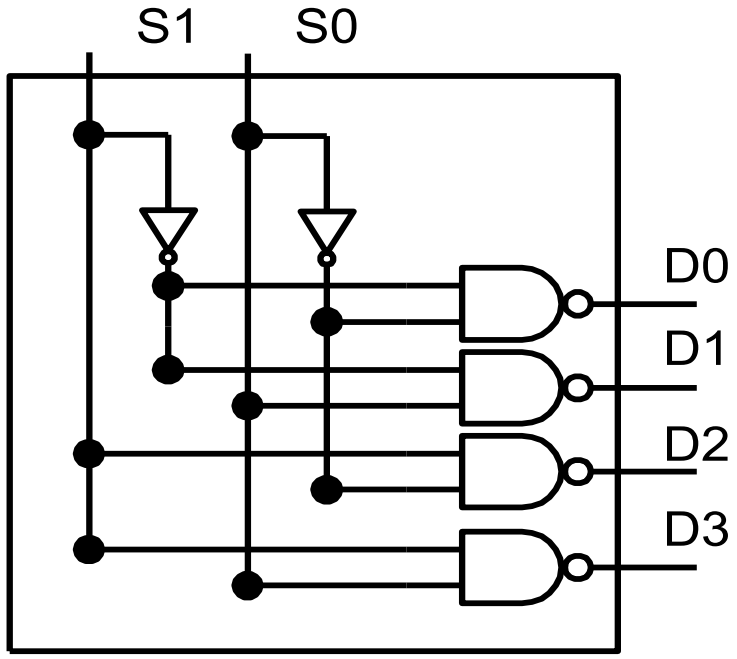


S1	S0	D0	D1	D2	D3
L	L	L	H	H	H
L	H	H	L	H	H
H	L	H	H	L	H
H	H	H	H	H	L

Decoder 2-4: Active Low (cont'd)

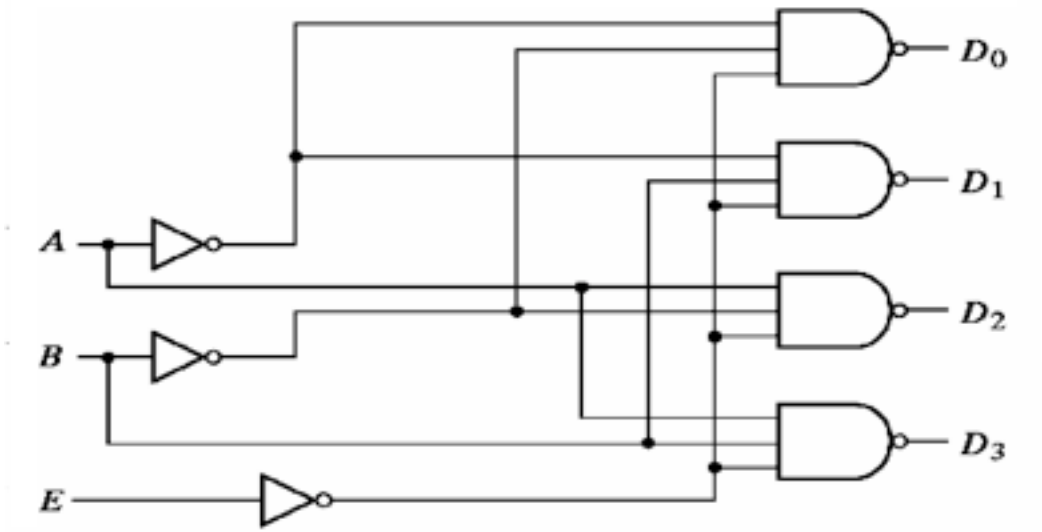


S1	S0	D0	D1	D2	D3
L	L	L	H	H	H
L	H	H	L	H	H
H	L	H	H	L	H
H	H	H	H	H	L



Decoder with enable:2-4: Active Low

- Output
  - Maxterms



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

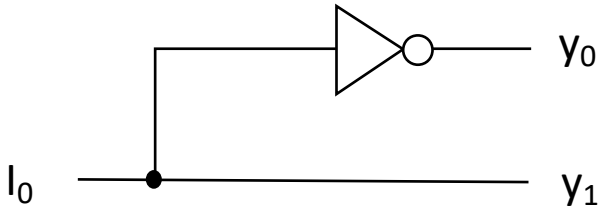
(b) Truth table

How About Larger Decoders?

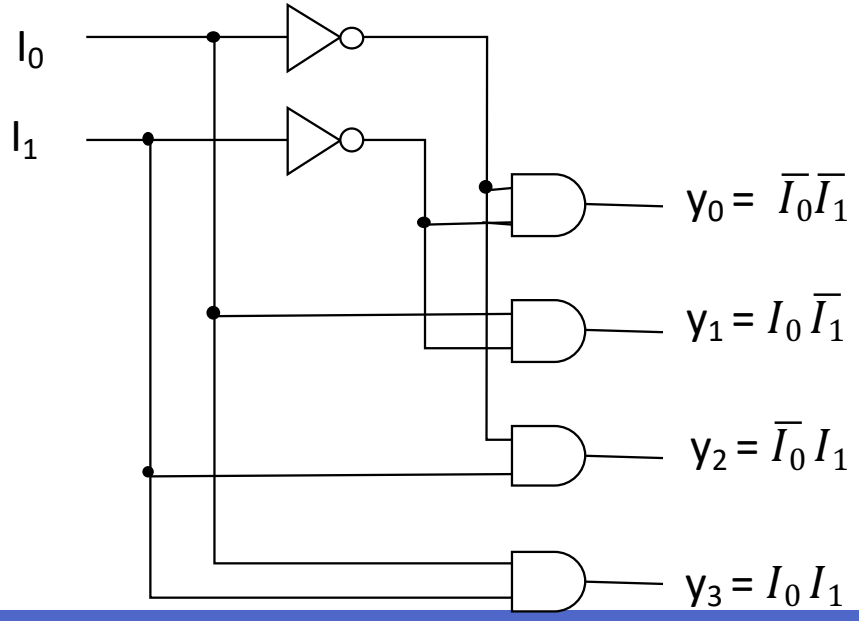
- 3-8 Decoder
- 4-16 Decoder
- 5-32 Decoder
- 10-1024 Decoder
- 11-2048 Decoder

Decoder 1-2 and 2-4

I_0	y_0	y_1
0	1	0
1	0	1



I_1	I_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Decoder 2-4 is made up of 2 decoder 1-2 and 4 AND gates

Expansion

- Combine two or more small circuits to form a larger one
  - Using enable signal
- Decoder expansion
  - E.g., construct decoder 3-8 from decoder 2-4

Decoder 3-8

- Design a 3-8 decoder using 2-4 decoder?

Decoder 3-8: Truth Table

S_1	S_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

S_2	S_1	S_0	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Expansion: Decoder 3-8

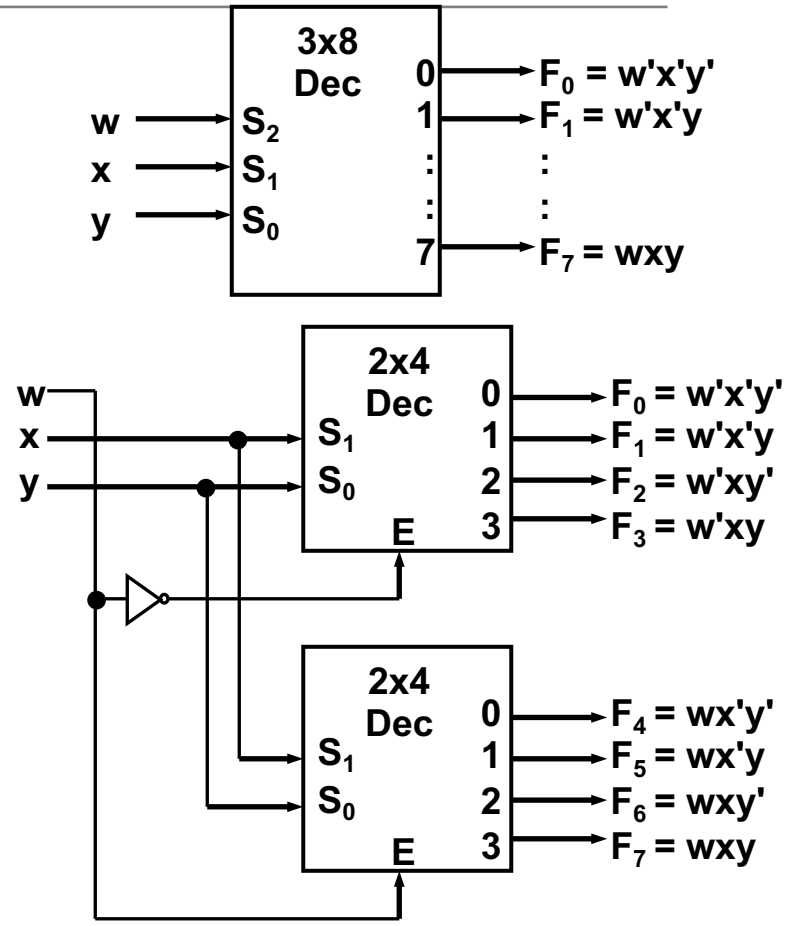
- Decoder 3-8
  - Two 2-4 decoder
  - MSB is connected to the enable inputs
  - if S_2=0, upper is enabled
  - if S_2=1, lower is enabled.

S_1	S_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

S_2	S_1	S_0	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Expansion: Decoder 3-8 (cont'd)

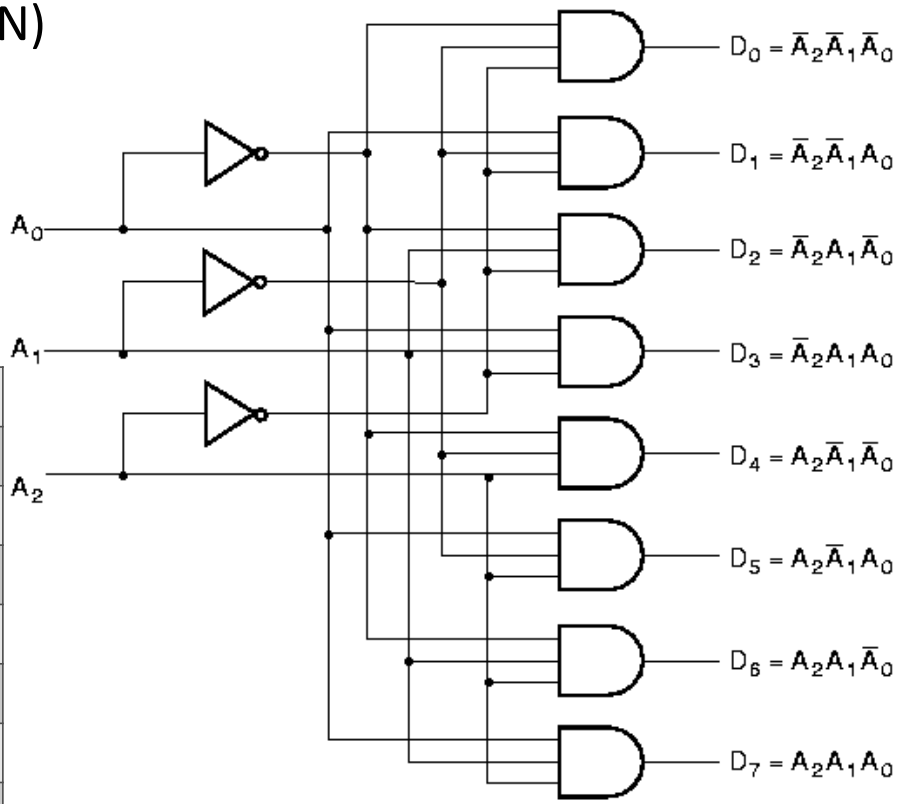
S_2	S_1	S_0	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Decoder 3-8: Logic Diagram

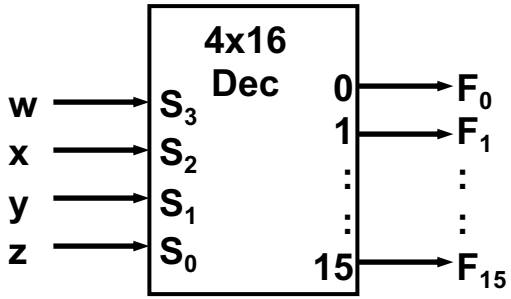
- Input: 3-bit code and 1-bit enable (EN)
- Output: 8-bit

A2	A1	A0	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1							
0	0	1		1						
0	1	0			1					
0	1	1				1				
1	0	0					1			
1	0	1						1		
1	1	0							1	
1	1	1								1



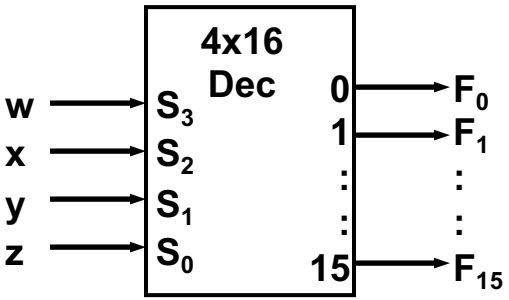
Decoder 4-16 using Decoder 3-8

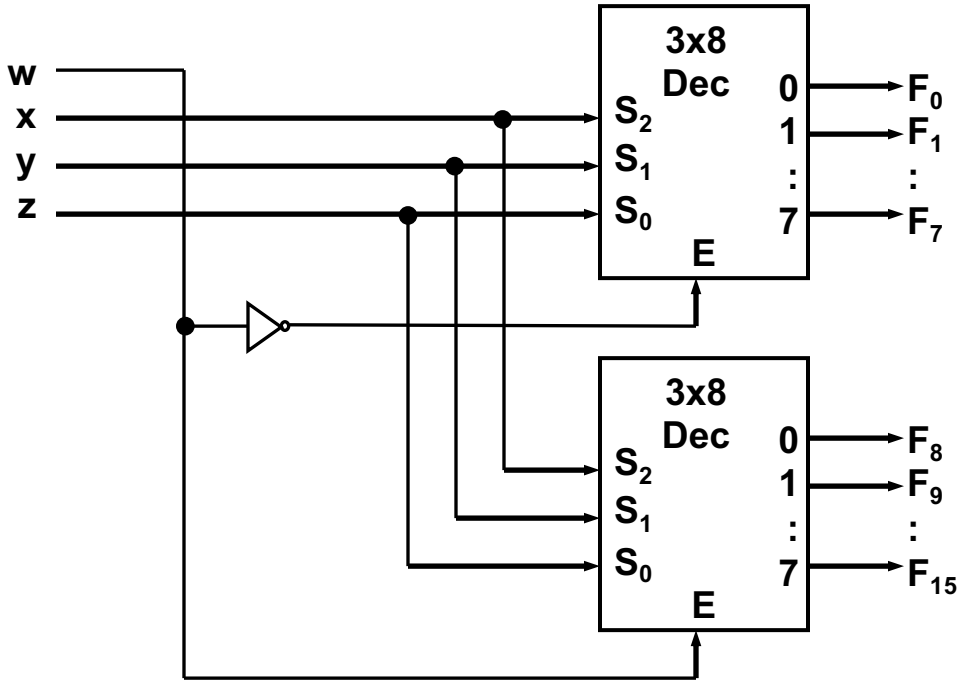
- Implement 4-16 decoder
  - Using 3-8 decoders



Check Yourself ☺

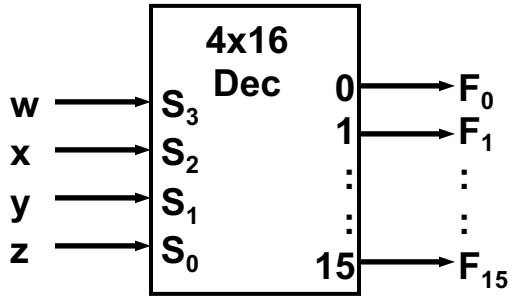
Decoder 4-16 using Decoder 3-8 (cont'd)

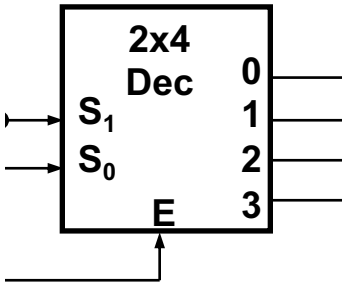




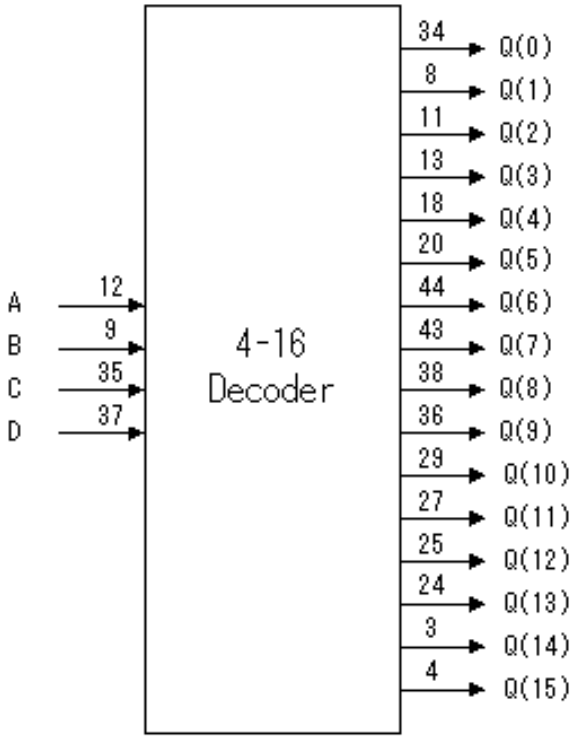
Decoder 4-16 using Decoder 2-4

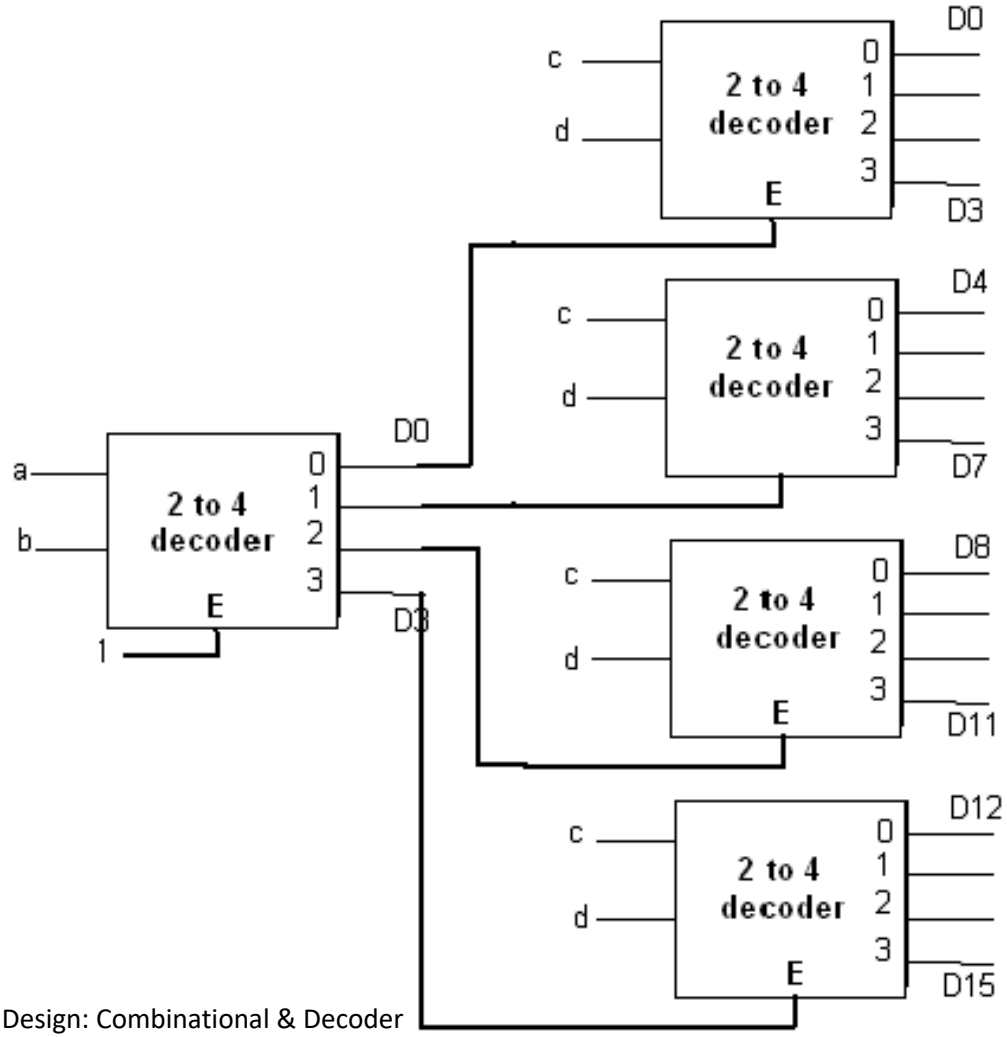
- Implement 4-16 decoder
  - Using 2-4 decoders





Expansion: Decoder 4-16: Sample2 (cont'd)





Implementation Using Decoder

Decoder as a Logic Block

- Implement function f using decoder
  - f(Q,X,P) = ∑m(0,1,4,6,7)

Truth Table

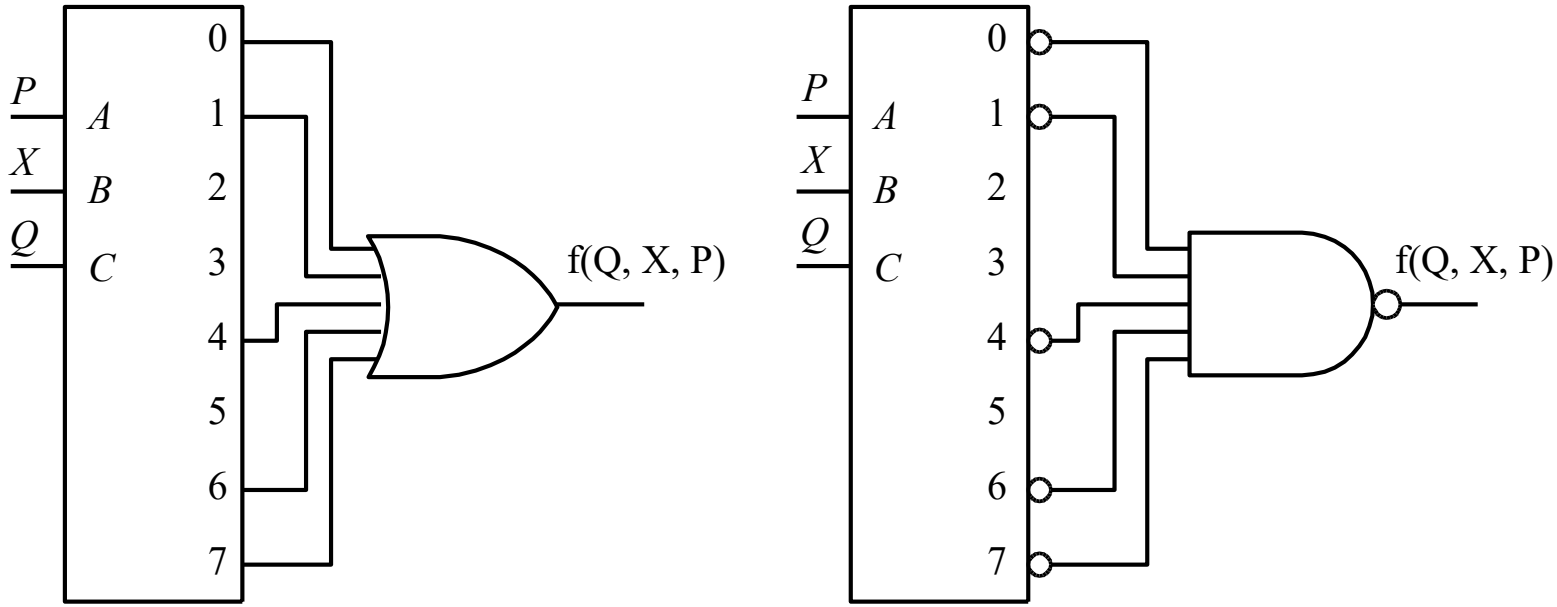
- Truth table for f(Q,X,P) = ∑m(0,1,4,6,7)
- Consider the truth table of a 3-8 decoder

Q	X	P	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Q	X	P	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Realization

- f(Q,X,P) = ∑m(0,1,4,6,7)

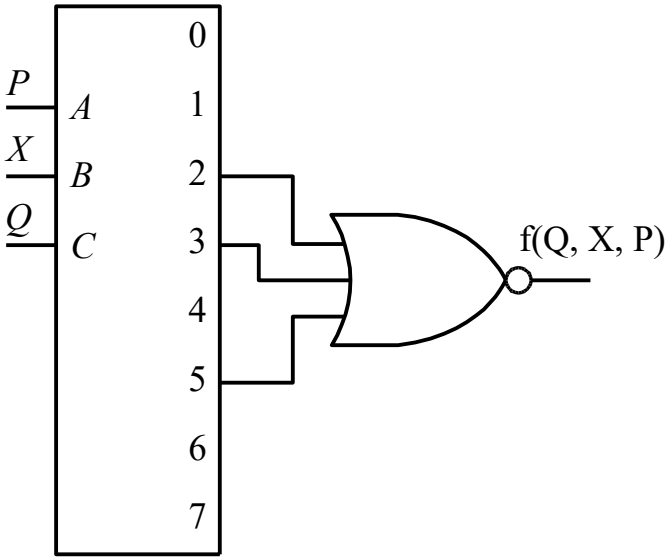


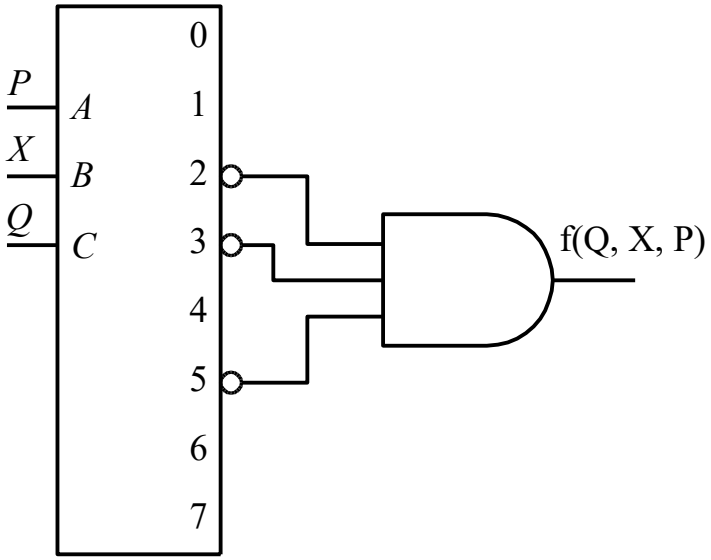
Sample 1

- Implement function f using decoder
  - f(Q,X,P) = $\prod M(2,3,5)$

Sample1: Realization

- Implement function f using decoder
  - f(Q,X,P) = $\prod M(2,3,5)$





Sample 2

- Implement a full adder using decoder

Full Adder: Truth Table

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S(x, y, z) = S\ m(1,2,4,7)$

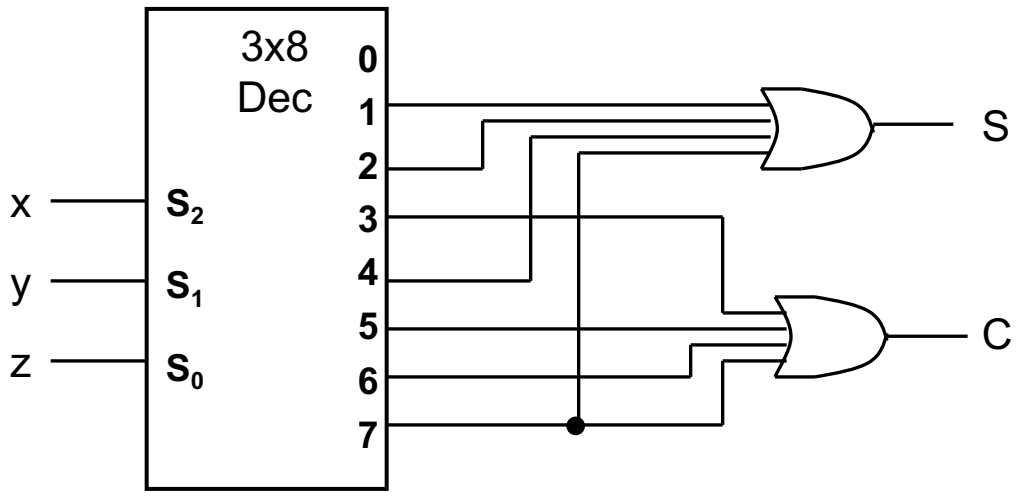
$C(x, y, z) = S\ m(3,5,6,7)$

Full Adder: Realization

S(x, y, z) = S m(1,2,4,7)

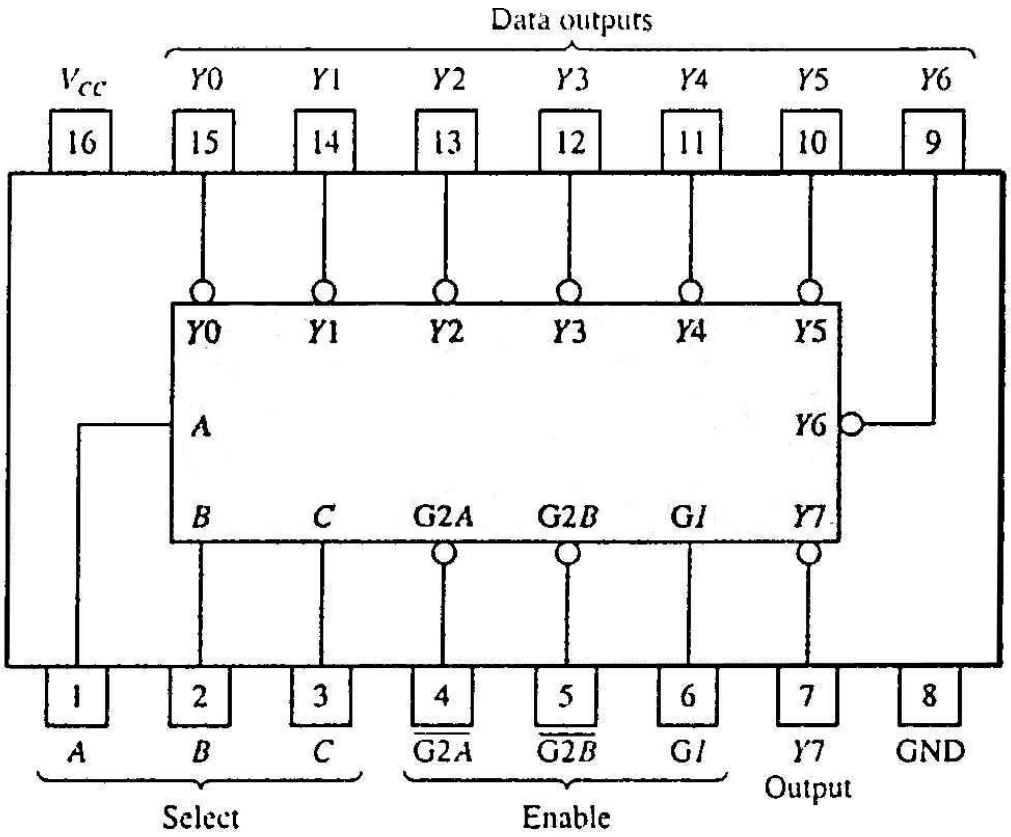
C(x, y, z) = S m(3,5,6,7)

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Decoder Chip

- 74138
  - An standard decoder
  - Active-low outputs
  - Three Enable

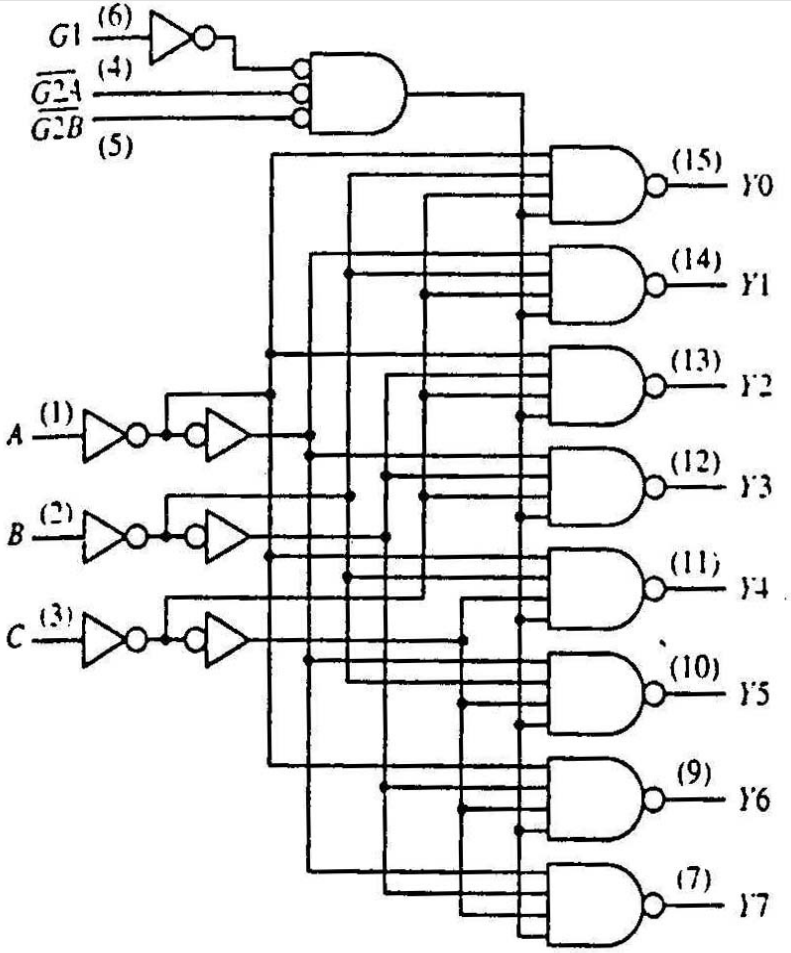


74138: Function Table

Inputs					Outputs							
Enable		Select										
$G1$	$\overline{G2^*}$	C	B	A	$Y0$	$Y1$	$Y2$	$Y3$	$Y4$	$Y5$	$Y6$	$Y7$
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L
x	H	x	x	x	H	H	H	H	H	H	H	H
L	x	x	x	x	H	H	H	H	H	H	H	H

$$\overline{G2^*} = \overline{G2A} + \overline{G2B}$$

74138: Logic Circuit

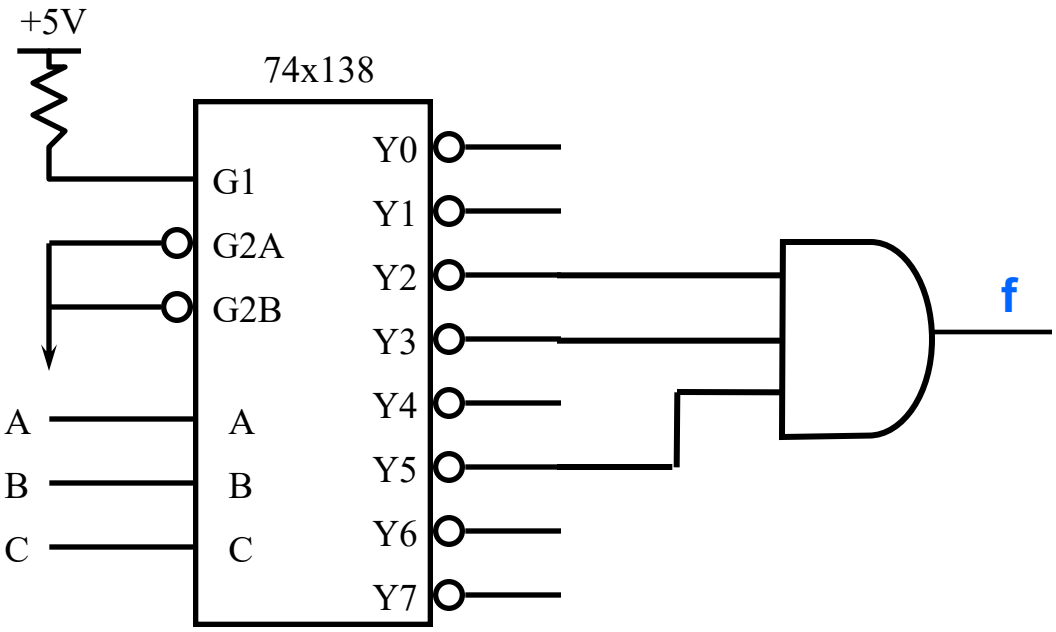


Sample 3

- Implement function f using decoder
  - f(A,B,C) = $\prod M(2,3,5)$

Implement f Using 74X138

- Implement function f using decoder
  - f(A,B,C) = $\prod M(2,3,5)$

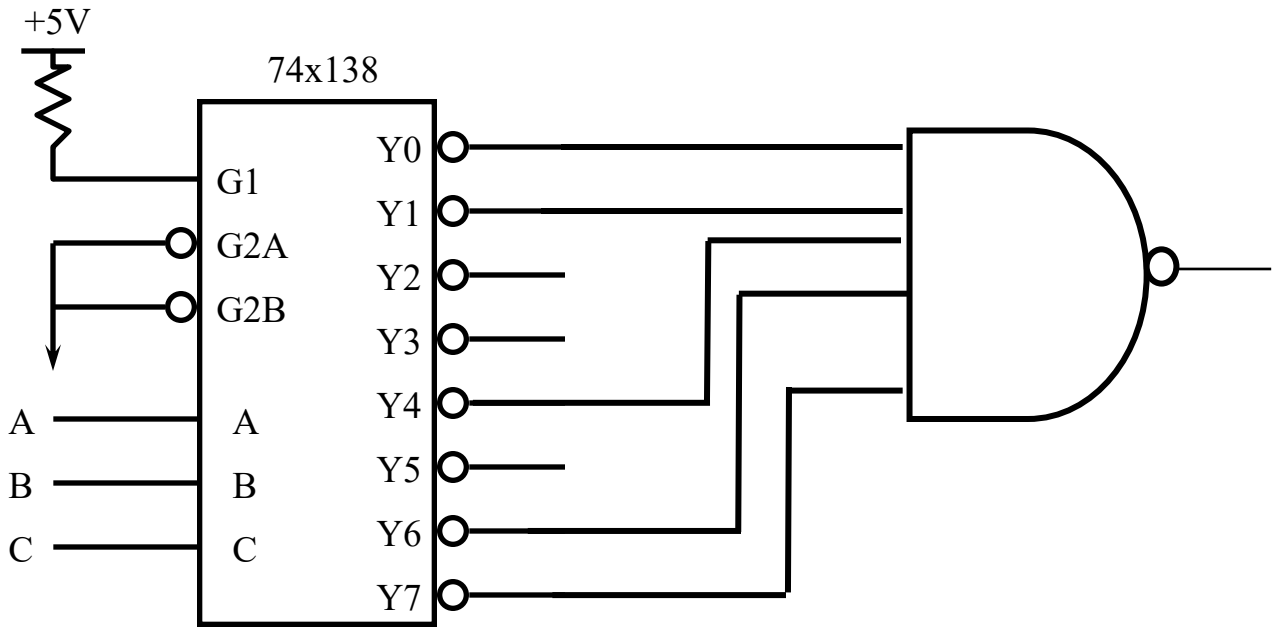


Sample 4

- Implement function f using an active low decoder
  - f(A,B,C) = ∑m(0,1,4,6,7)

Decoder as a logic block: Sample 7 (cont'd)

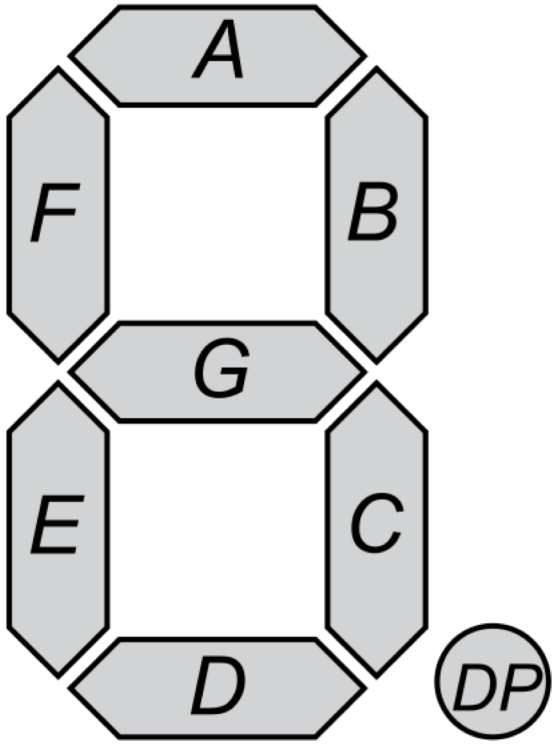
- Implement function f using an active low decoder
  - f(A,B,C) = ∑m(0,1,4,6,7)



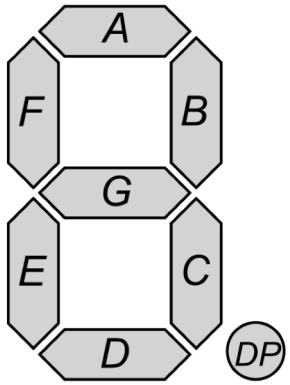
7-Segment Decoder

7-Segment LED Displays

- Display numbers
- 7 light emitting diodes (LED)
- Each LED controlled by an input
  - 1 means “on”
  - 0 mean “OFF”



Digit	Display	gfedcba	abcdefg	a	b	c	d	e	f	g
0	0	0x3F	0x7E	on	on	on	on	on	on	off
1	1	0x06	0x30	off	on	on	off	off	off	off
2	2	0x5B	0x6D	on	on	off	on	on	off	on
3	3	0x4F	0x79	on	on	on	on	off	off	on
4	4	0x66	0x33	off	on	on	off	off	on	on
5	5	0x6D	0x5B	on	off	on	on	off	on	on
6	6	0x7D	0x5F	on	off	on	on	on	on	on
7	7	0x07	0x70	on	on	on	off	off	off	off
8	8	0x7F	0x7F	on	on	on	on	on	on	on
9	9	0x6F	0x7B	on	on	on	on	off	on	on
A	A	0x77	0x77	on	on	on	off	on	on	on
b	b	0x7C	0x1F	off	off	on	on	on	on	on
C	C	0x39	0x4E	on	off	off	on	on	on	off
d	d	0x5E	0x3D	off	on	on	on	on	off	on
E	E	0x79	0x4F	on	off	off	on	on	on	on
F	F	0x71	0x47	on	off	off	off	on	on	on

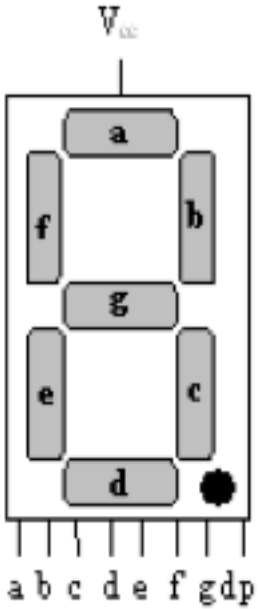


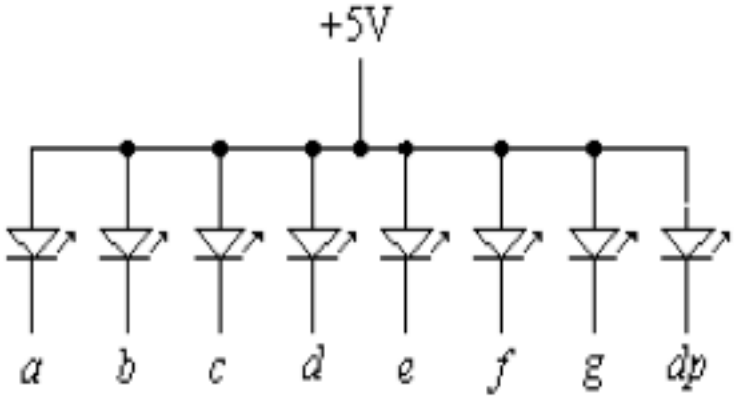
7-Segment : Types

- Two types
  - Common- anode
  - Common-cathode

7-Segment : Common-Anode

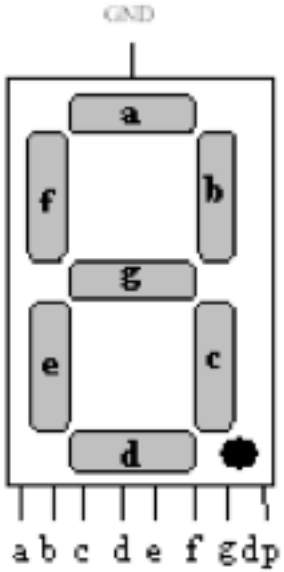
- Anode of all of the LEDs are tied together to power supply

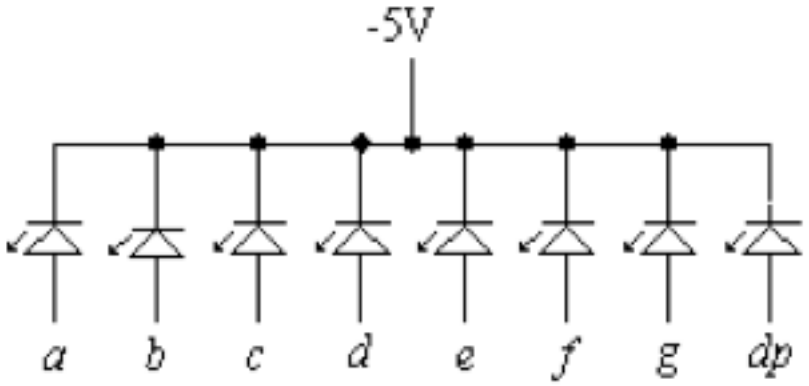




7-Segment: Common-Cathode

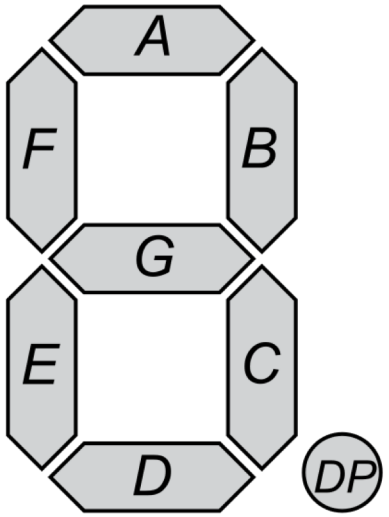
- Cathode of all of the LEDs are tied together to ground





7-Segment Decoder

- Input
  - 4-bit BCD code
- Output
  - 7-bit code
- Function
  - Display decimal code
- Example
  - Input: $(0000)_{BCD}$
  - Output: $(1111110)_{7\text{-segment}}$



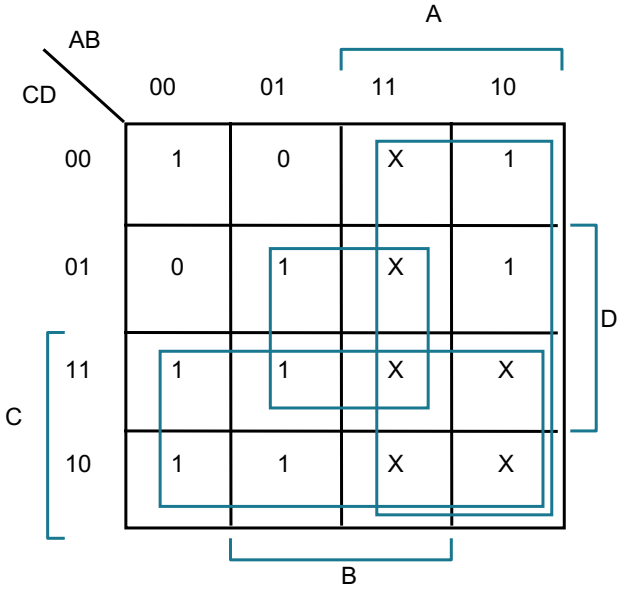
Digit	Display	gfedcba	abcdefg	a	b	c	d	e	f	g
0	0	0x3F	0x7E	on	on	on	on	on	on	off

7-Segment Decoder: Truth Table

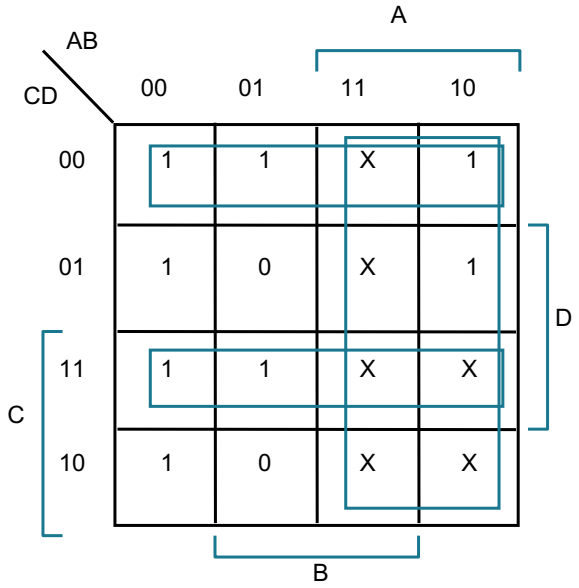
Digit	ABCD	abcdefg
0	0000	1111110
1	0001	0110000
2	0010	1101101
3	0011	1111001
4	0100	0110011
5	0101	1011011
6	0110	1011111
7	0111	1110000
8	1000	1111111
9	1001	1110011

Digit	ABCD	abcdefg
10	1010	XXXXXXX
11	1011	XXXXXXX
12	1100	XXXXXXX
13	1101	XXXXXXX
14	1110	XXXXXXX
15	1111	XXXXXXX

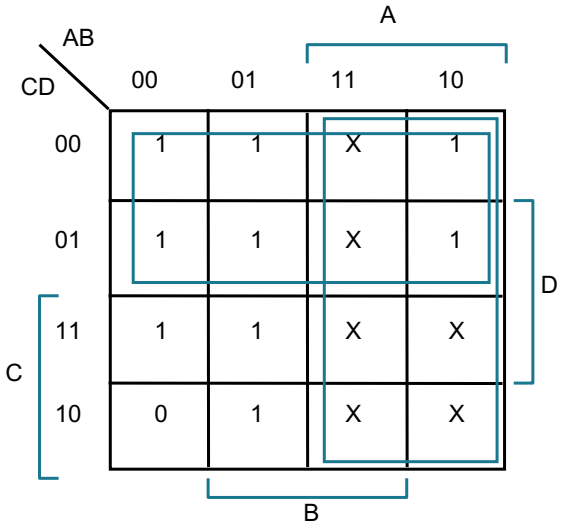
7-Segment Decoder: K-Map



a



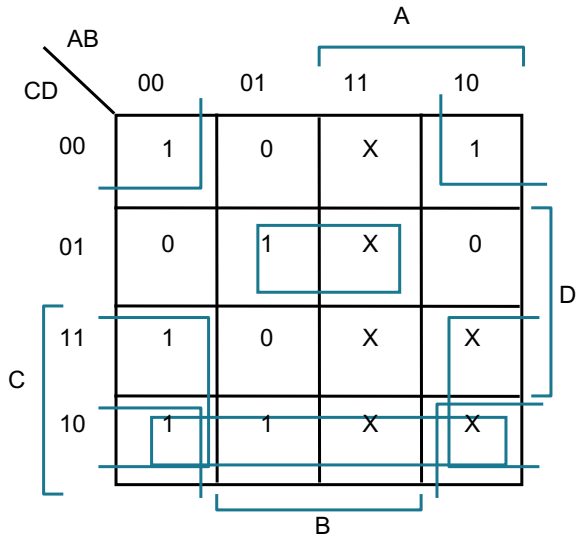
b



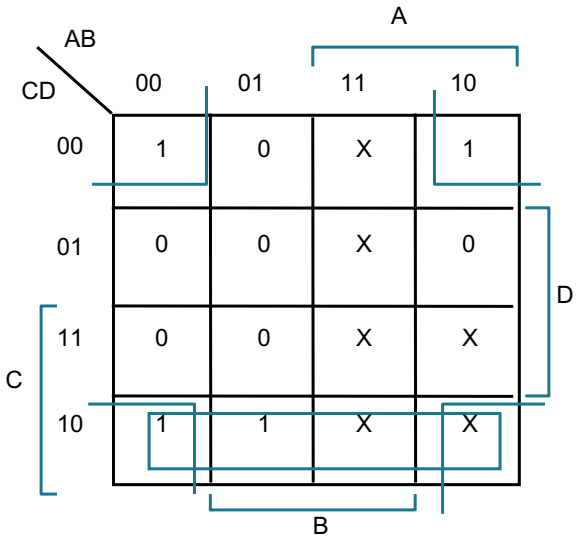
c

a = A + B D + C + B' D'
b = A + C' D' + C D + B'
c = A + B + C' + D

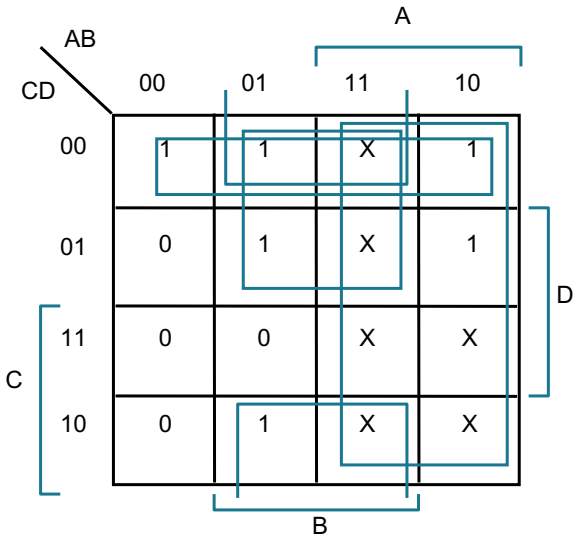
7-Segment Decoder: K-Map



d



e



f

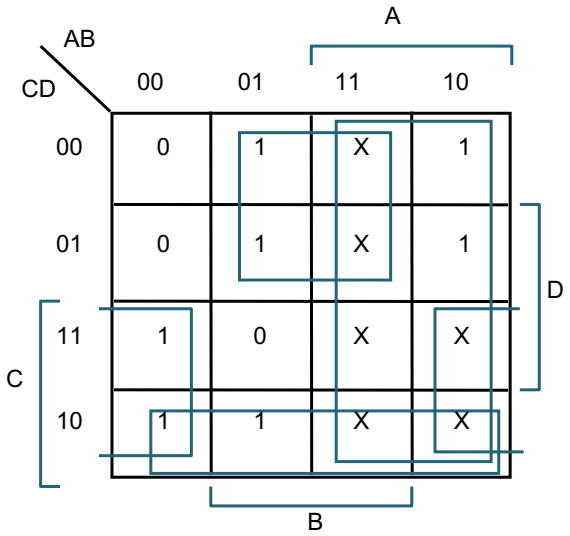
d = B' D' + C D' + B C' D + B' C
e = B' D' + C D
f = A + C' D' + B D' + B C'

7-Segment Decoder: K-Map

a = A + B D + C + B' D'
b = A + C' D' + C D + B'
c = A + B + C' + D

d = B' D' + C D' + B C' D + B' C
e = B' D' + C D
f = A + C' D' + B D' + B C'

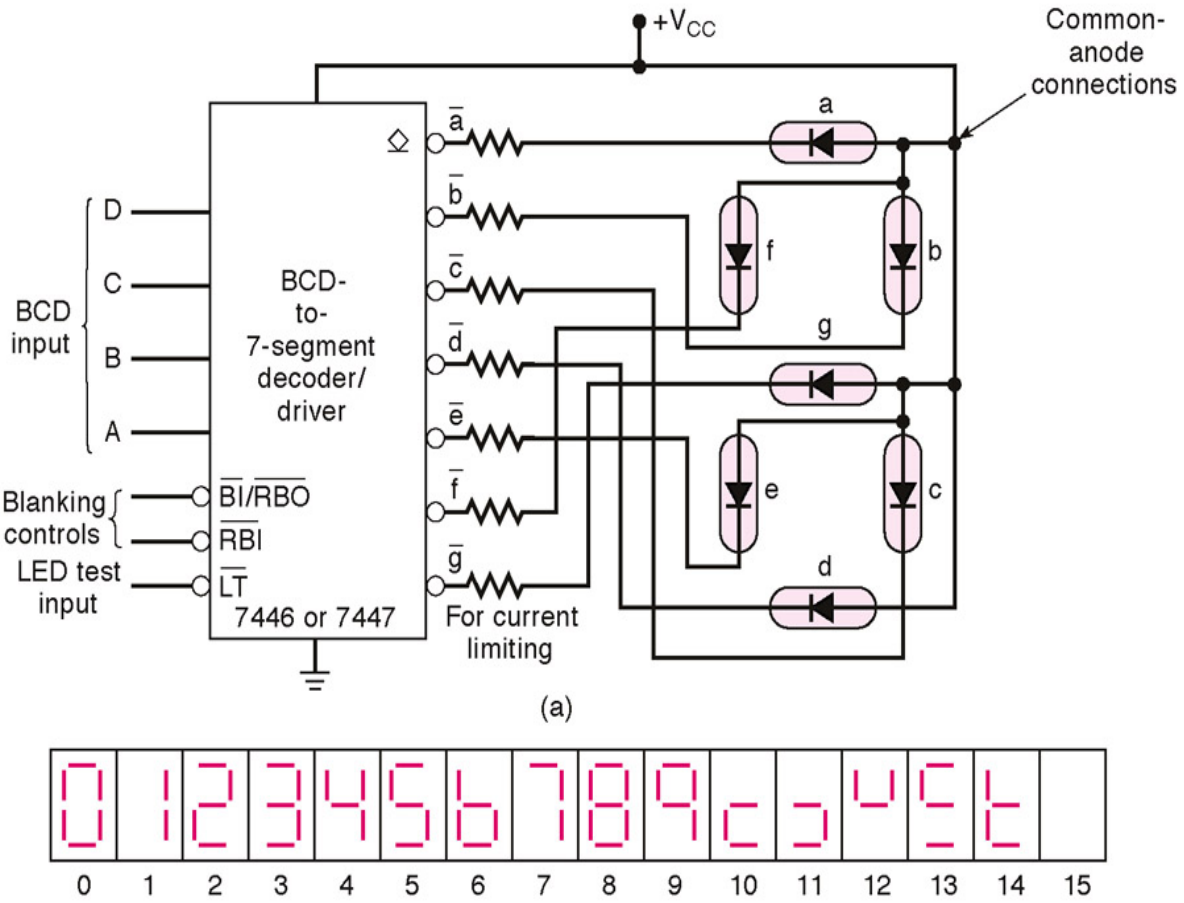
g = A + C D' + B C' + B' C



g

BCD to 7-Segment Decoder

- 74LS47



Thank You