

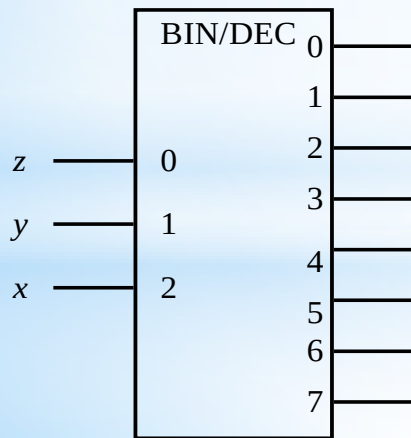
Outline

- 4.1 Logical functional blocks
 - Decoder
 - Encoder
 - Multiplexer
 - Demultiplexer
- 4.2 Programmable logic devices
 - ROM, PLA, PAL, FPGA

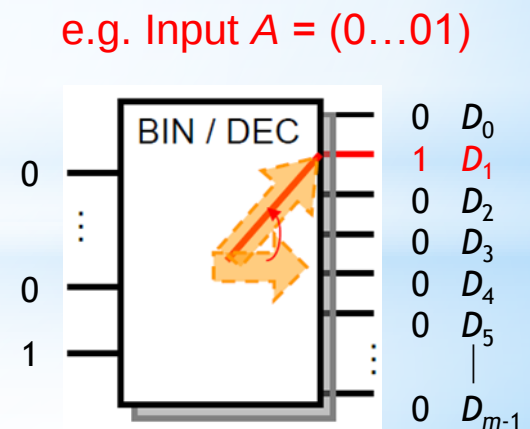
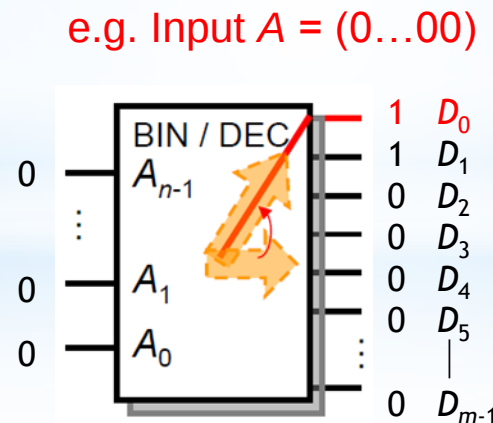
4.1 Logical functional Modules

4.1.1 Decoder

- A decoder is a combinational circuit with n -input and m -output ($0 < n \leq m \leq 2^n$, but usually $m = 2^n$)
- A very important functional blocks as it can be incorporated into many of the other functions



Decoder symbol



1-to-2-Line decoder

Specification:

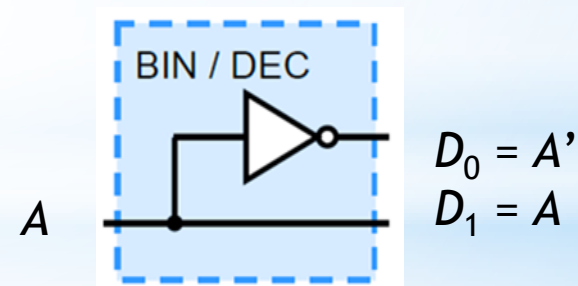
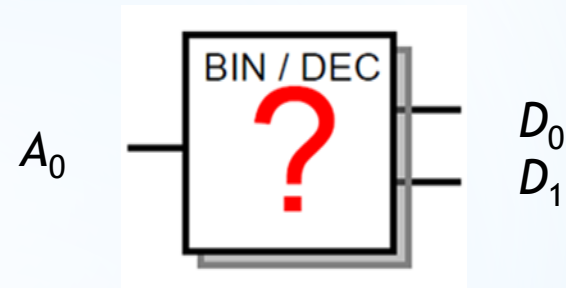
Input: 1-bit (A_0)

Outputs: 2-bit (D_0, D_1)

<i>Input</i>	<i>Outputs</i>	
A	D_0	D_1
0	1	0
1	0	1

$$D_0 = m_0 = A'$$

$$D_1 = m_1 = A$$

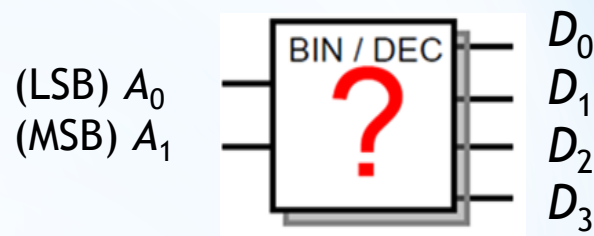


2-to-4-Line Decoder

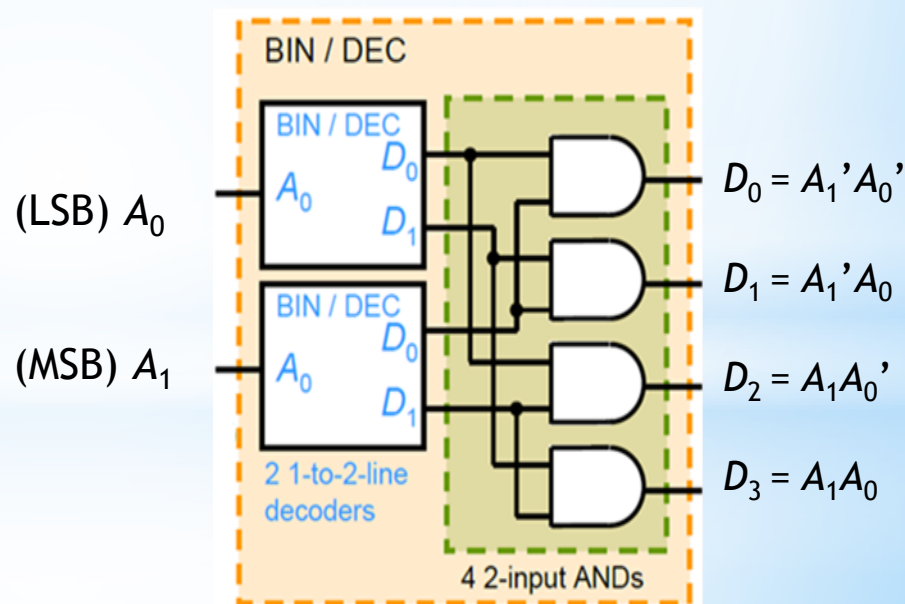
Specification:

Inputs: 2-bit (A_1 , A_0)

Outputs: 4-bit (D_0 , D_1 , D_2 , D_3)



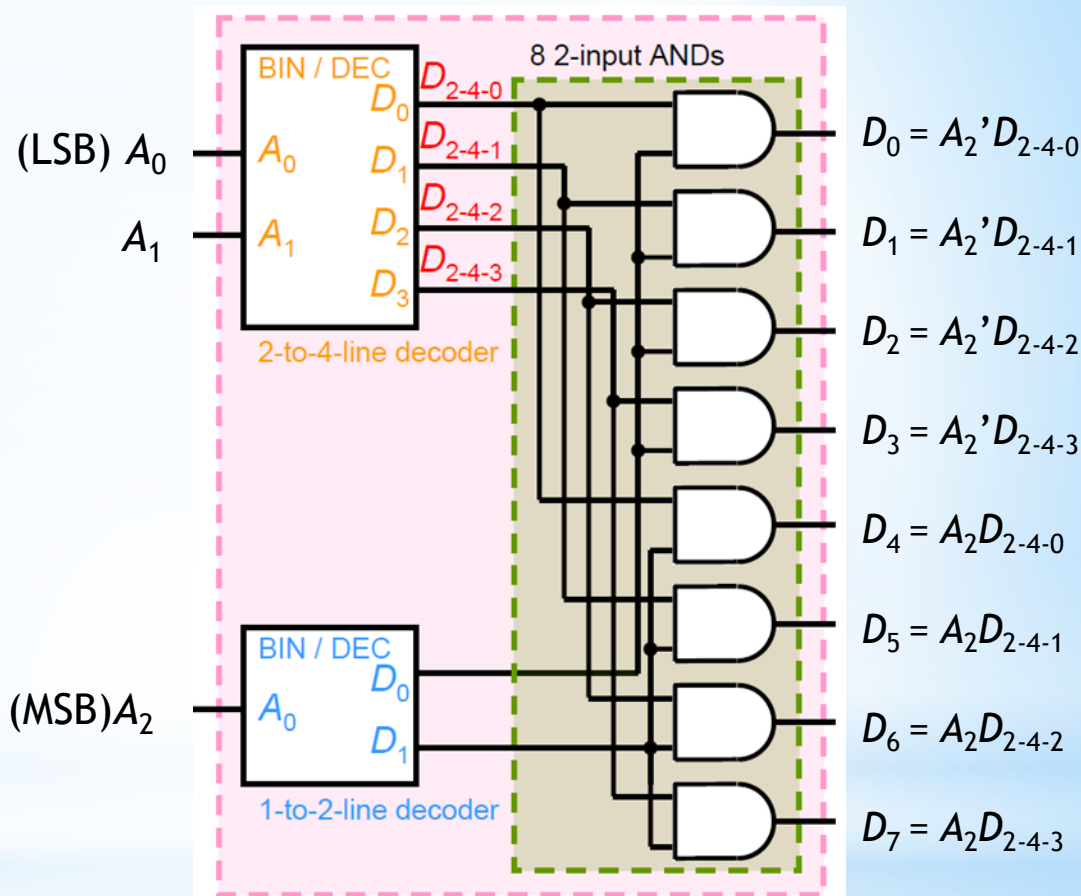
Inputs		Outputs			
A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



3-to-8-Line Decoder

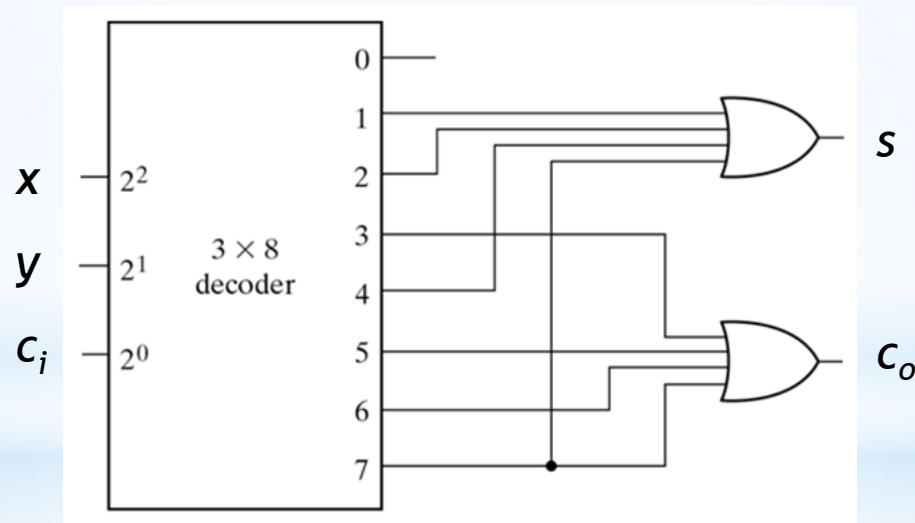
Inputs			Outputs							
A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

- Build the simplest block:
 - 1-to-2-line decoder
 - 2-to-4-line decoder
 - 3-to-8-line decoder
 - n -to- 2^n -line decoder



Full-adder using decoder

- For a full-adder operation, we have:
- $s(x, y, c_i) = \sum m(1, 2, 4, 7)$ and $c_o = \sum m(3, 5, 6, 7)$
- Since there are 3 inputs, we can use a 3-to-8-line decoder to implement the full-adder circuit.



NOTE: You can only produce more than one output with a single decoder !!

Decoder with Enable input

Specification:

A 2-to-4-line decoder with enable input

<i>Inputs</i>			<i>Outputs</i>			
<i>EN</i>	<i>A₁</i>	<i>A₀</i>	<i>D₀</i>	<i>D₁</i>	<i>D₂</i>	<i>D₃</i>
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

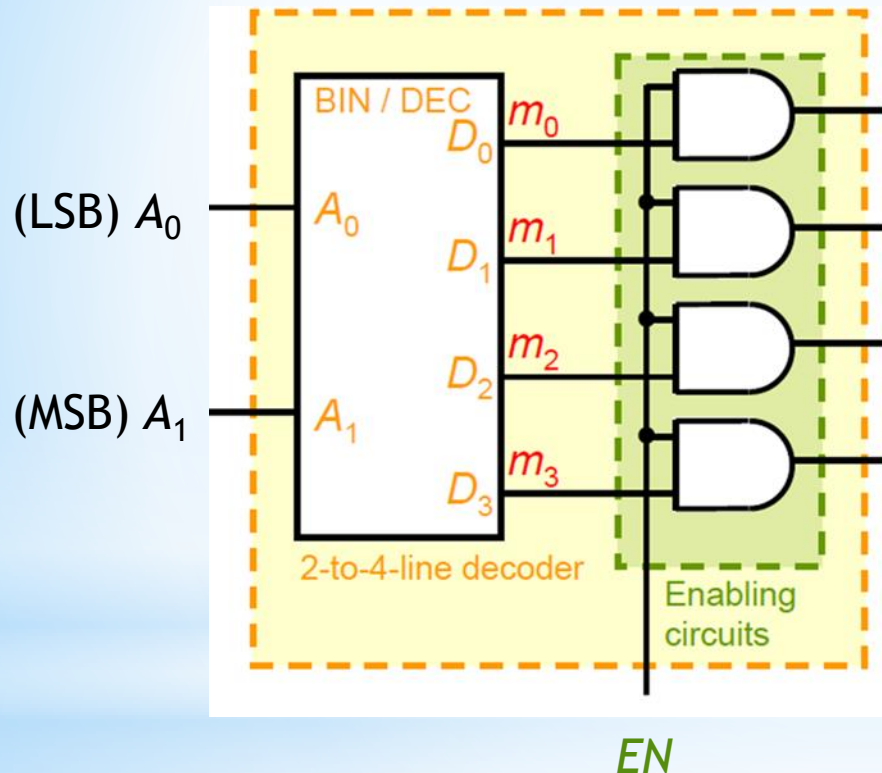
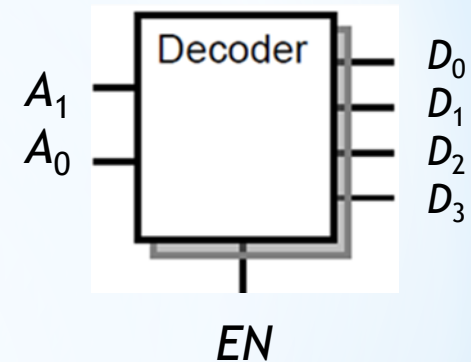
If *EN* to 0 (**disabled**), all outputs are 0

If *EN* to 1 (**enabled**), the output are same as normal decoder (i.e. only the corresponding $D_i = 1$, and all others are 0)

Note: the Xs in the above table are don't care inputs

Decoder with Enable input

$$D_i = EN \cdot m_i$$

 D_0 D_1 D_2 D_3 

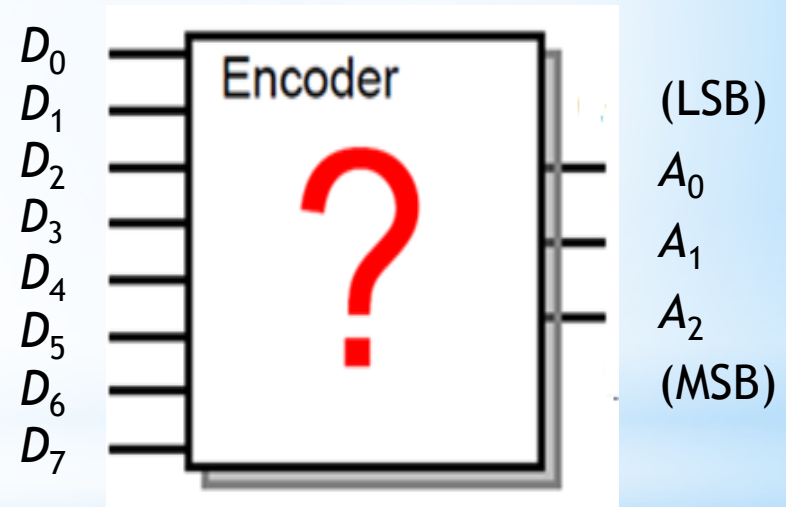
The corresponding symbol of 2-to-4-line decoder with enable input

4.1.2 Encoder

- An encoder is a functional block that performs the inverse operation of a decoder
- m inputs and n outputs
- $0 < n \leq m \leq 2^n$, but usually $m = 2^n$

Example of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A_2	A_1	A_0
1	0	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	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1
All other invalid inputs								X	X	X



Octal-to-binary encoder

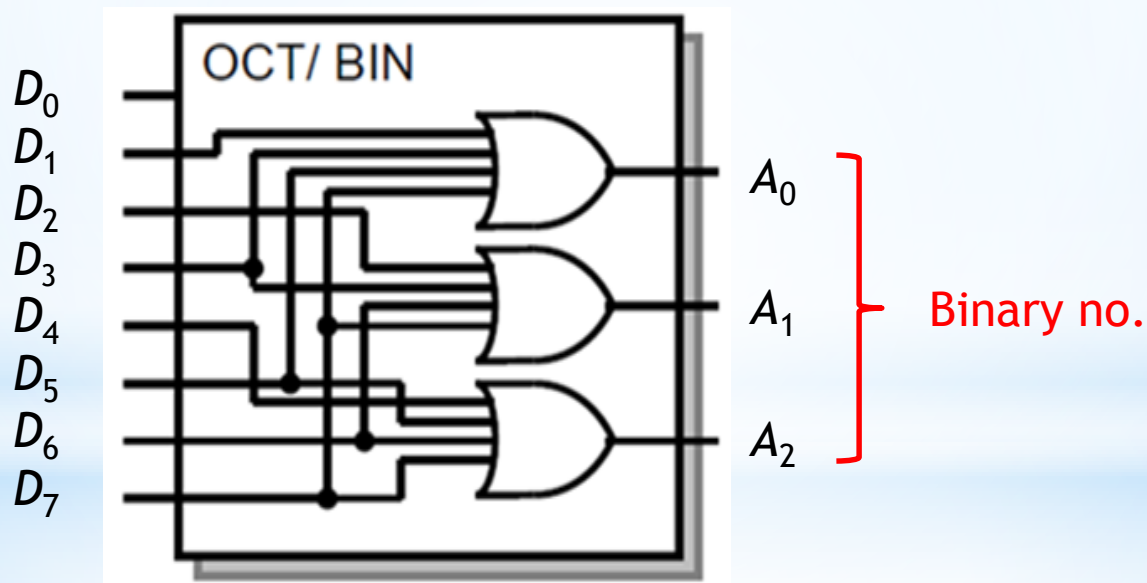
- $m = 8, n = 3$
- i.e. inputs:8, outputs: 3

Octal-to-binary Encoder

$$A_0(D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7) = D_1 + D_3 + D_5 + D_7$$

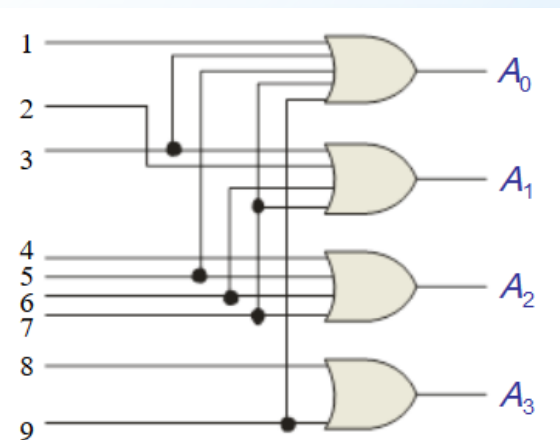
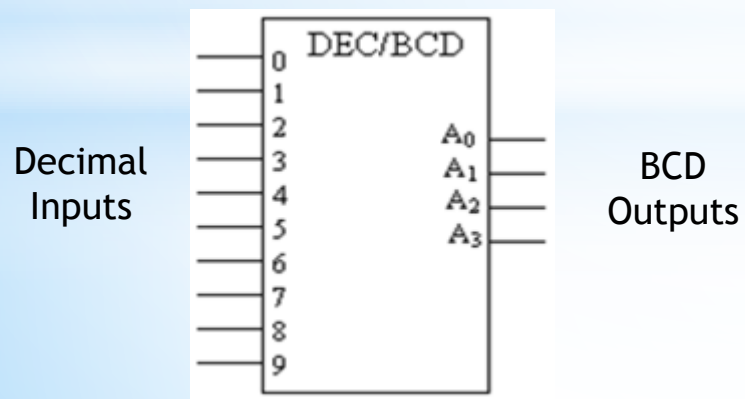
$$A_1(D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7) = D_2 + D_3 + D_6 + D_7$$

$$A_2(D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7) = D_4 + D_5 + D_6 + D_7$$



Decimal-to-BCD encoder

Inputs										Outputs			
0	1	2	3	4	5	6	7	8	9	A_3	A_2	A_1	A_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1



Limitation of Encoders

- Only one input can be active (i.e. 1)
- If two or more inputs active simultaneously, the output produces an incorrect combination
- **Solution:**
 - To resolve this ambiguity, introduce an **input priority**
 - Each input pin has different priority
 - If two or more inputs are 1 at the same time, only consider input that has higher priority
 - This kind of encoder is called **priority encoder**

4.1.3 Priority Encoders

- Design a 4-input priority encoder
- Inputs with higher subscript numbers has higher priority

<i>Inputs</i>				<i>Outputs</i>		
D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	X	X	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

Introduce one more output V (valid output).

If all inputs are 0 (i.e invalid Input), V is 0.

Otherwise V is 1

If D_3 and D_2 are both 1, ignore D_2 (as if D_2 is 0)

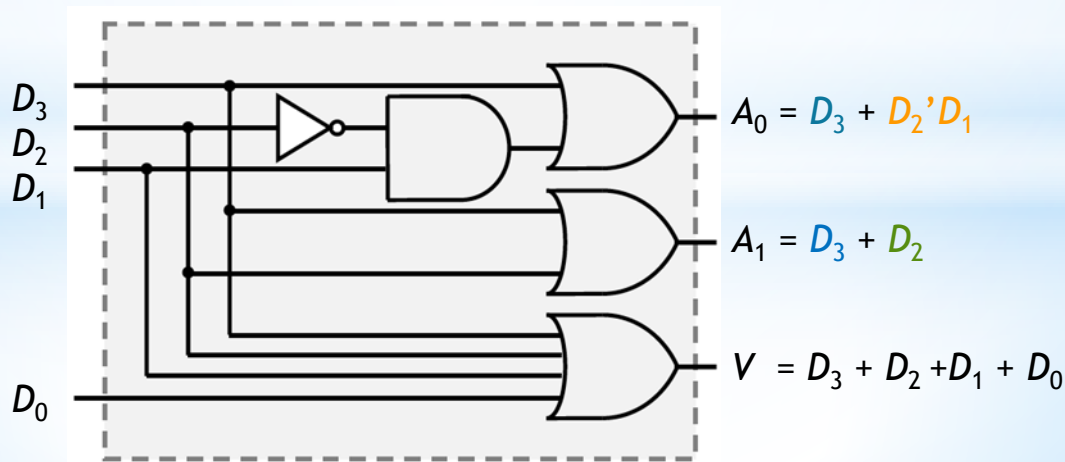
4-Input Priority Encoder

		A_1				A_0				V			
D_1D_0	D_3D_2	00	01	11	10	00	01	11	10	00	01	11	10
	00	x	1	1	1	x		1	1	0	1	1	1
01			1	1	1			1	1	1	1	1	1
11			1	1	1	1		1	1	1	1	1	1
10			1	1	1	1		1	1	1	1	1	1

$A_1(D_3, D_2, D_1, D_0)$
 $= D_3 + D_2$

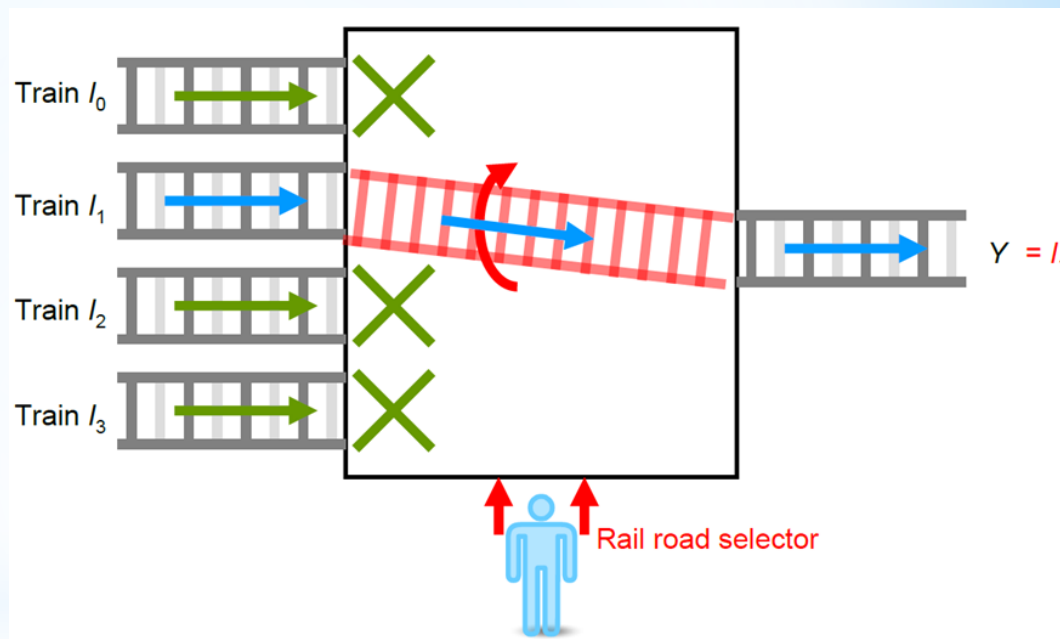
$A_0(D_3, D_2, D_1, D_0)$
 $= D_3 + D_2'D_1$

$V' = D_3'D_2'D_1'D_0'$
 $V = D_3 + D_2 + D_1 + D_0$



4.1.4 Logical functional blocks (Multiplexer -- MUX)

- Combinational circuit performs selection
- To appear I_s on a **single output** (Y) from **many inputs** (I_i)



- A set of selection input variables $(S)_{10} = (S_{n-1} \dots S_1 S_0)_2$

2-to-1-line MUX

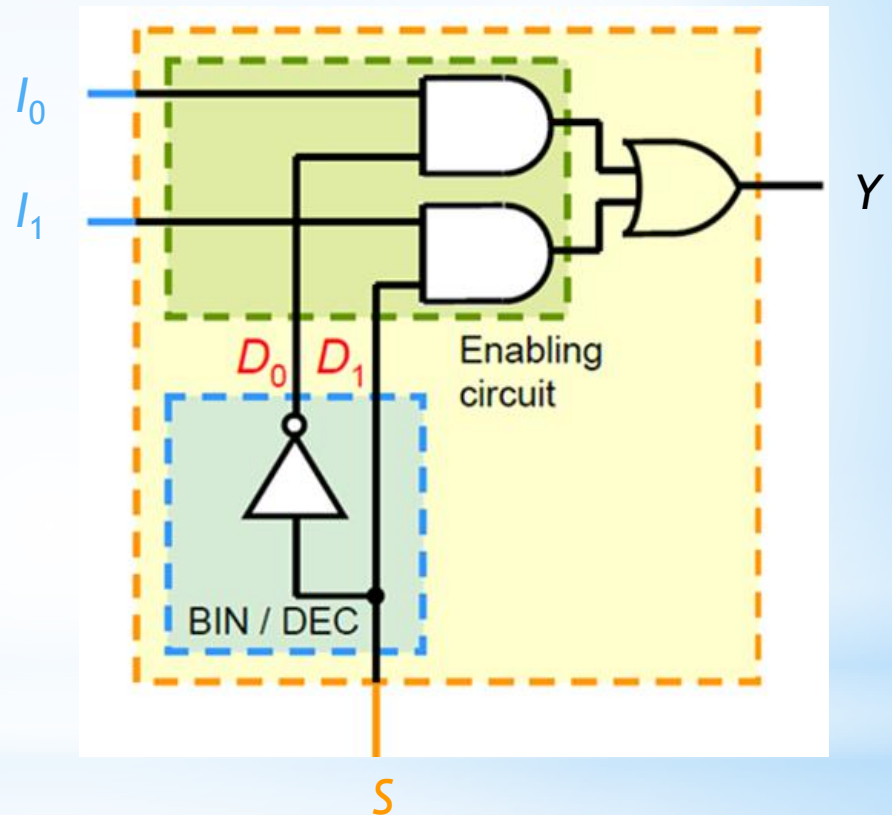
Specification:

$$m = 2$$

$$n = \log_2 m = 1$$

Inputs			Output
I_1	I_0	S_0	Y
X	X	0	I_0
X	X	1	I_1

$$Y(I_0, I_1, S_0) = S_0' I_0 + S_0 I_1$$



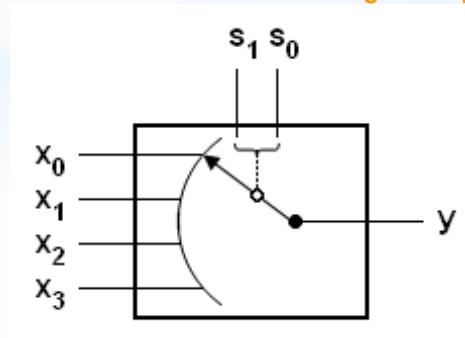
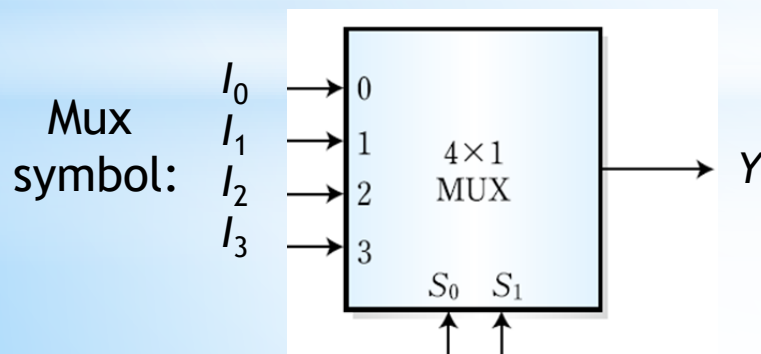
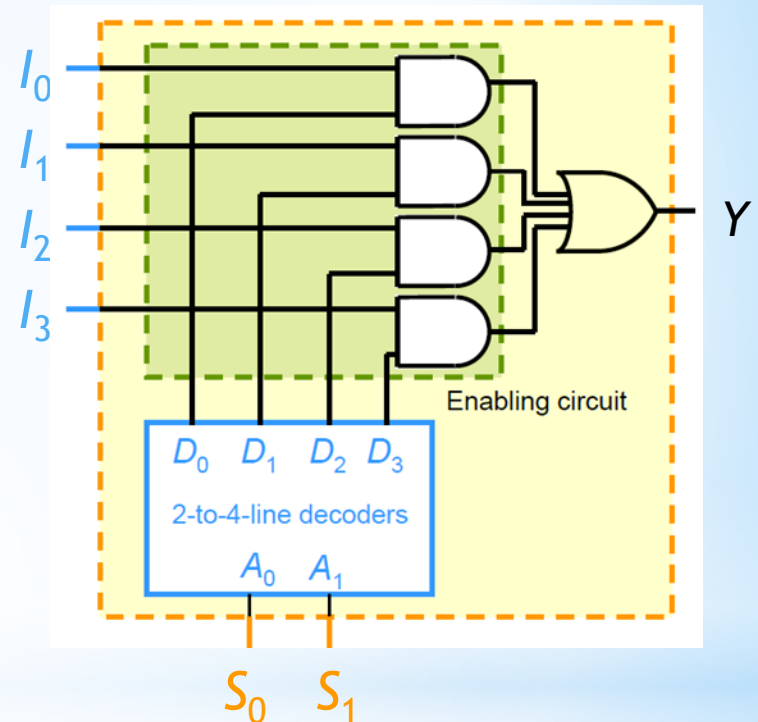
4-to-1-line MUX

$$m = 4$$

$$n = \log_2 m = 2$$

Inputs						Output
I_3	I_2	I_1	I_0	S_1	S_0	Y
X	X	X	X	0	0	I_0
X	X	X	X	0	1	I_1
X	X	X	X	1	0	I_2
X	X	X	X	1	1	I_3

$$Y(I_0, I_1, I_2, I_3, S_1, S_0) = S_1' S_0' I_0 + S_1' S_0 I_1 + S_1 S_0' I_2 + S_1 S_0 I_3$$



Example

Use a 4-input MUX to realize the following function:

$$F(w,x,y,z) = \sum m(1,2,5,7,9,11,13)$$

Step 1: Plot the K-map
(no need to group)

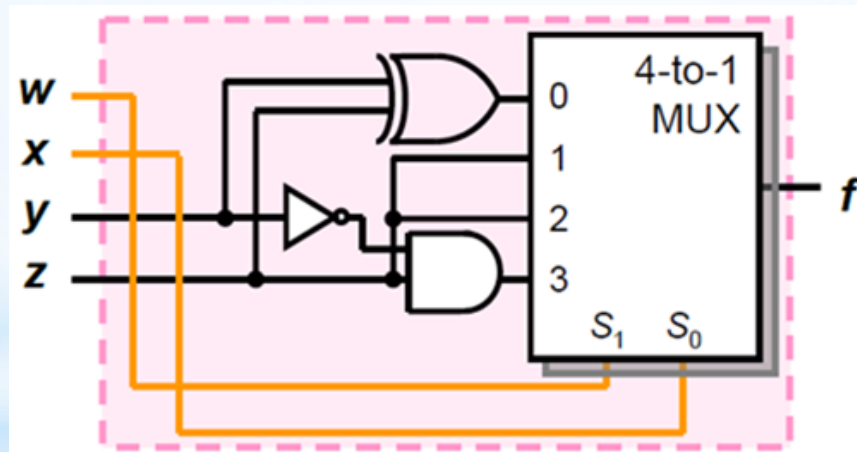
Step 2: Since $n = 2$
Pick w, x as selection variables
(i.e. w as S_1 , x as S_0)
Remaining variables: y, z

wx \ yz	yz			
	00	01	11	10
00		1		1
01		1	1	
11		1		
10		1	1	

Step 3:

		yz				
wx		00	01	11	10	
00	00		1		1	} $wx = 00, f = y'z + yz' = y \oplus z$
	01		1	1		
	11		1			
	10		1	1		

Step 4:



Exercise: instead of using 4-to1-line MUX, try to use 8-to-1-line MUX and 2-to-1-line MUX

NOTE: You can only produce a single output with a single MUX!!

With

2-to-1 Mux

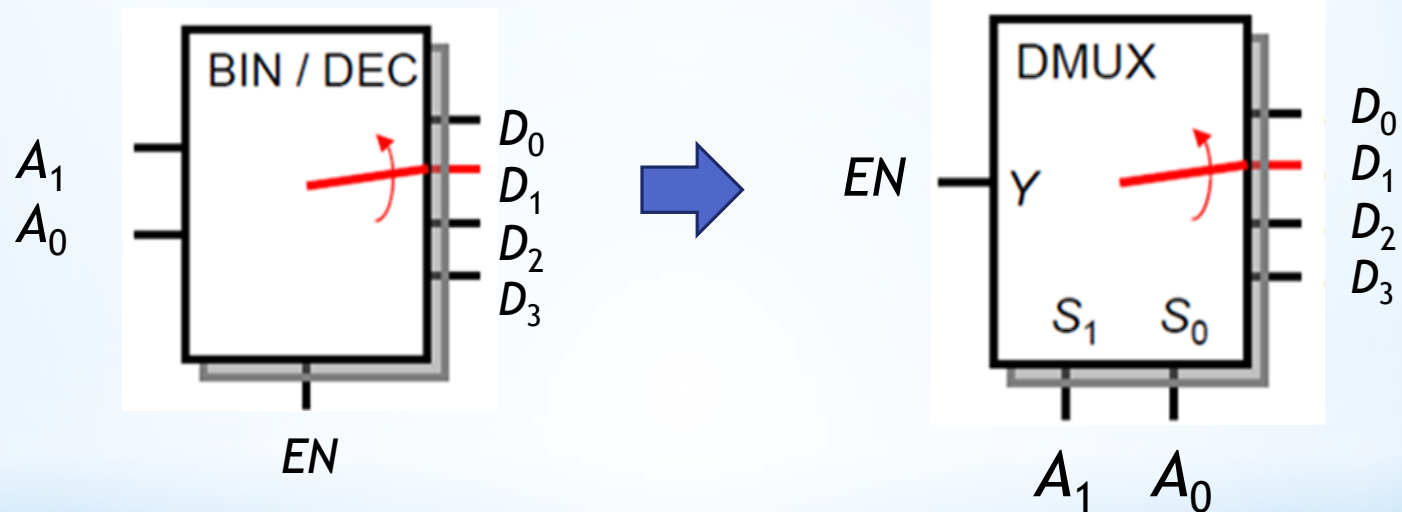
4-to-1 Mux

8-to-1 Mux

wxyz	F	$S_0=w$	$S_1=w, S_0=x$	$S_2=w, S_1=x, S_0=y$
0000	0	$I_0 = xz + y'z + x'yz'$	$I_0 = y \oplus z$	$I_0 = z$
0001	1			$I_1 = z'$
0010	1			
0011	0		$I_1 = z$	$I_2 = z$
0100	0			$I_3 = z$
0101	1			
0110	0			$I_4 = z$
0111	1			$I_5 = z$
1000	0	$I_1 = x'z + y'z$	$I_2 = z$	$I_6 = z$
1001	1			$I_7 = 0$
1010	0			
1011	1		$I_3 = y'z$	
1100	0			
1101	1			
1110	0			
1111	0			

4.1.5 Demultiplexer (DMUX)

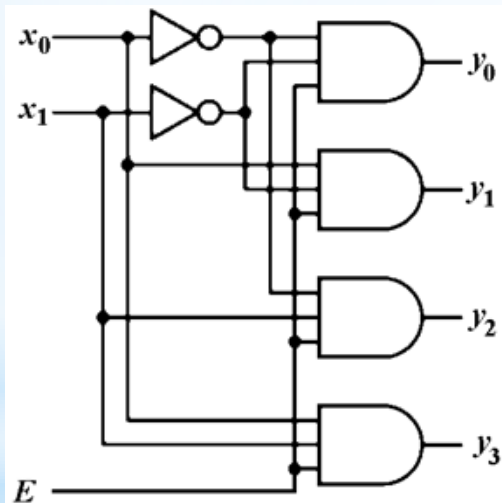
Remember the decoder with enable?



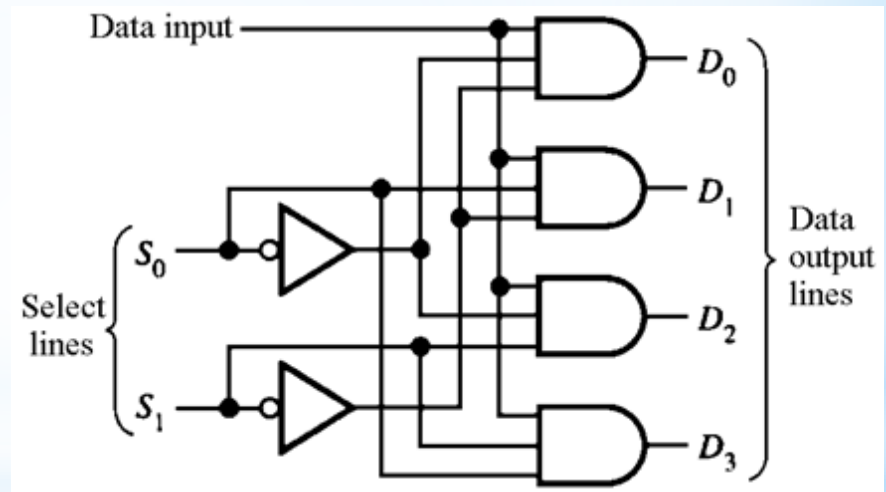
The decoder can perform demultiplexer if we take EN as the input line, A_i (input lines of decoder) as the selection inputs

Demultiplexer (DMUX)

A demultiplexer (DMUX) basically reverses the multiplexing function. A DMUX is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit pattern of n select lines. For this reason, the demultiplexer is also known as a data distributor.



Decoder with enable

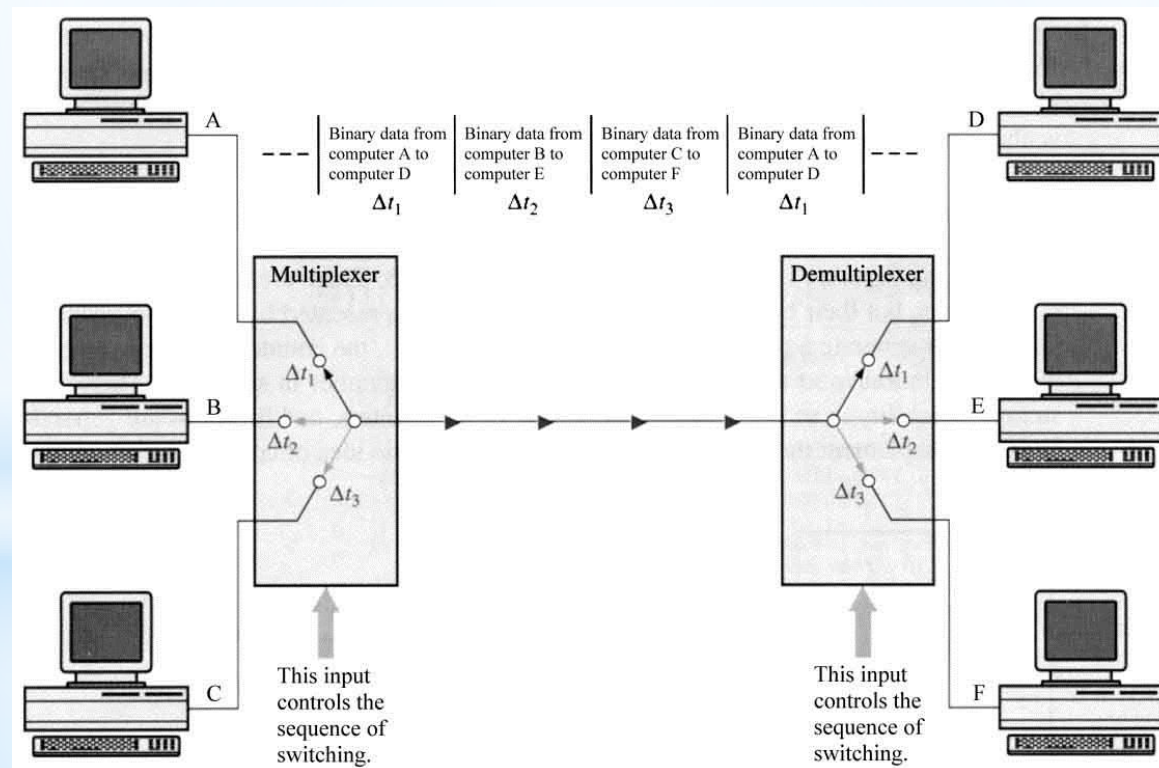


Demultiplexer

Decoder can function as demultiplexer if the E line is taken as a data input line and input lines taken as the selection lines.

Typical MUX and DMUX application

- A MUX is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination and a DMUX basically reverses the multiplexing function.

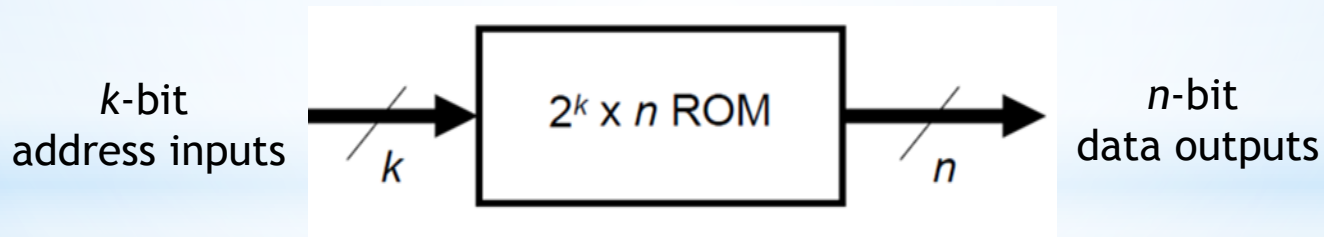


4.2 Programmable logic devices

- Implementation technologies introduced so far are not programmable
 - Fix-wired integrated circuits
- Programmable Logic Device (PLD)
 - Integrated circuit with internal logic gates connected together
 - These internal connections can be reconfigured (erased / programmed) to form a specific logic circuit
 - Basic logic unit whose function can be re-configured through special hardware description language
- Four types of basic PLDs
 - ROM, PLA, PAL, FPGA (this will be covered in detail by Dr. Ray Cheung)

4.2.1 Read-Only Memory (ROM)

- An essential memory device in which **permanent** binary information is stored (even when the power is turned off, the information is still there).
- k -bit address inputs and n -bit data outputs
Inputs: address for the memory
Outputs: the word (n data bits) stored in ROM selected by the k -bit input address
- k address input can specify 2^k words



- Features:
- advantage of simple and less connections
 - disadvantage of doubling the size with an additional input
 - cheap for mass production but expensive for development

Use a 8 x 4 bits ROM to implementing the following functions:

$$D_3(x,y,z) = \sum m(2,3,5), \quad D_2(x,y,z) = \sum m(0,1,3,6,7),$$

$$D_1(x,y,z) = \sum m(1,2,4,5,6), \quad D_0(x,y,z) = \sum m(1,3,5,6)$$

The contents stored inside the ROM is defined in the following table:

Memory location	Address			Data word			
	$A_2 = x$	$A_1 = y$	$A_0 = z$	D_3	D_2	D_1	D_0
m_0	0	0	0	0	1	0	0
m_1	0	0	1	0	1	1	1
m_2	0	1	0	1	0	1	0
m_3	0	1	1	1	1	0	1
m_4	1	0	0	0	0	1	0
m_5	1	0	1	1	0	1	1
m_6	1	1	0	0	1	1	1
m_7	1	1	1	0	1	0	0

With binary assignment of x y z = address of the ROM

Other ROMs

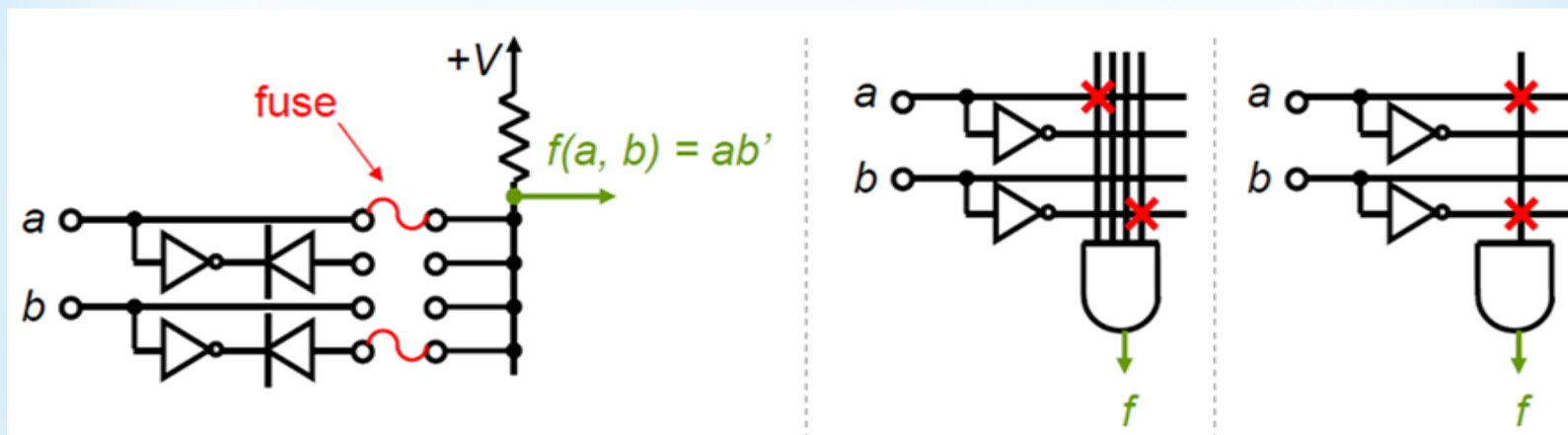
- ROM usually works as a lookup table
- No **data inputs** as ROM does not have a **write** operation (only read operation)
- ROMs that can be written / rewritten
 - PROM
 - EPROM (Erasable PROM)
 - EEPROM (Electrically Erasable PROM)
 - FEPRM (Field Erasable PROM)
 - FLASH (NOR) Memory

PROM

- Programmable Read-Only Memory
- Perform same functions as ROM
- Oldest programmable logic device
- Fuse or anti-fuse programmed
- One-time writing process has been deferred to the end-user
- Advantage: easy programming and inexpensive
- Disadvantage: non-reprogrammable

4.2.2 Fuse-Programmable Arrays

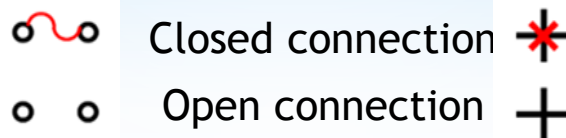
Programmable AND Arrays



Fused-programmable
AND arrays

Conventional
symbol

Array logic
symbol



Example: 8 x 4 PROM

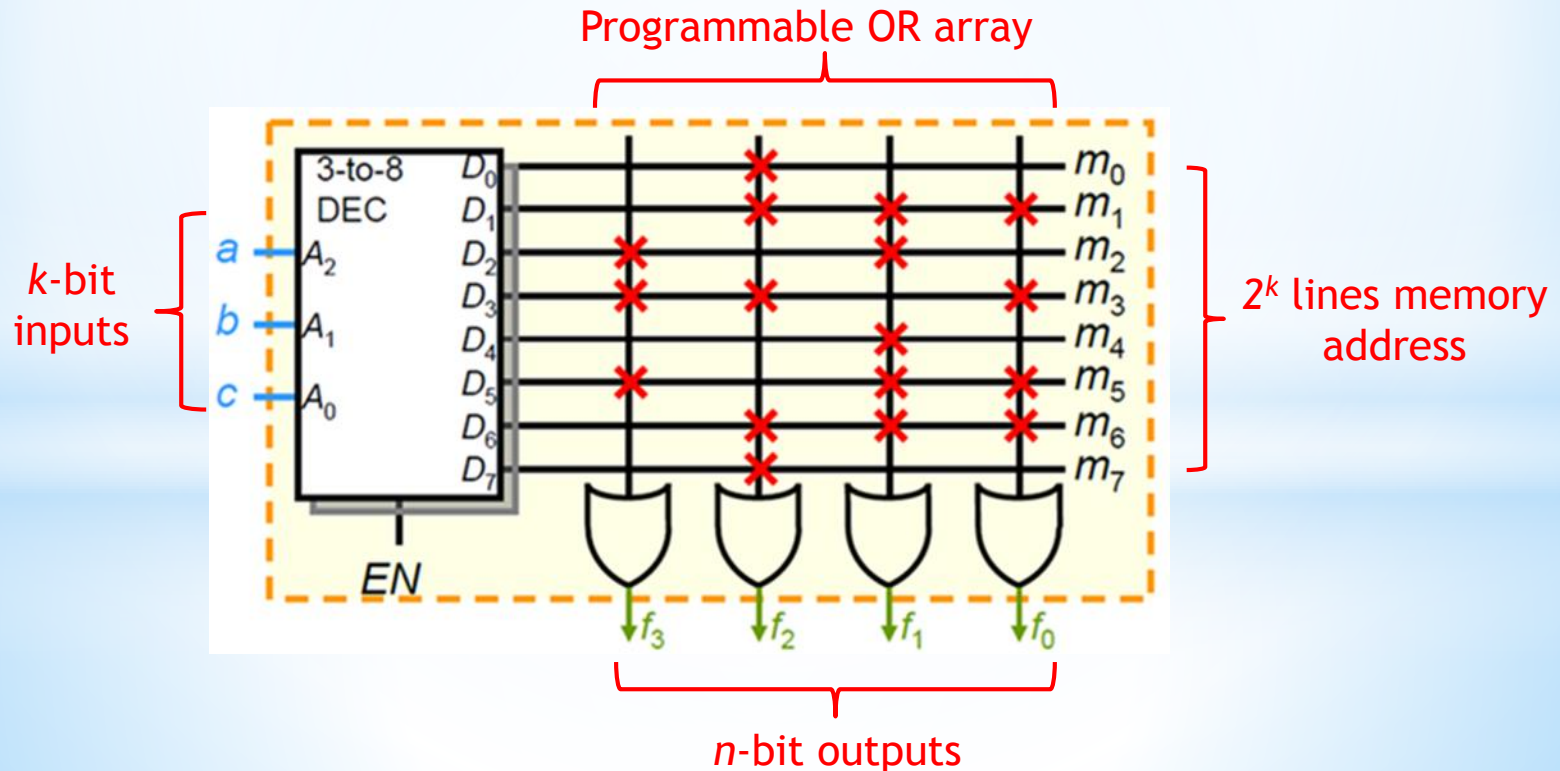
Realize the following functions with a 8 X 4 PROM

$$f_3(a,b,c) = \sum m(2,3,5)$$

$$f_2(a,b,c) = \sum m(0,1,3,6,7)$$

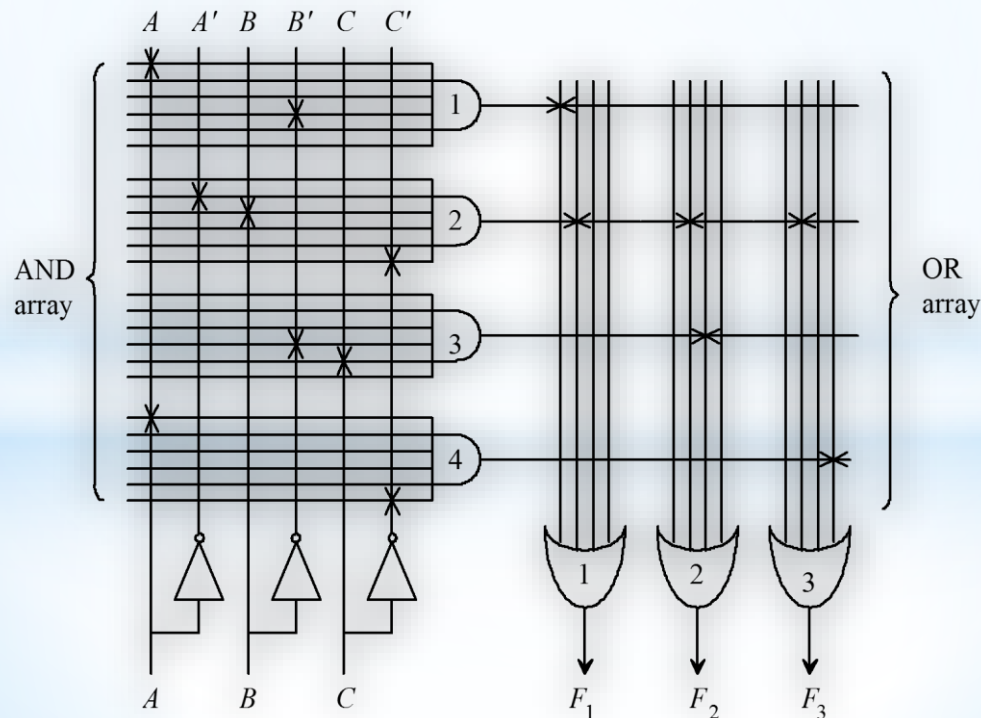
$$f_1(a,b,c) = \sum m(1,2,4,5,6)$$

$$f_0(a,b,c) = \sum m(1,3,5,6)$$



4.2.3 Programming Logic Array (PLA)

- Programmable device similar to PROM but with different internal structure
- The device is used to implement logic function in the form of **sum-of-product** terms

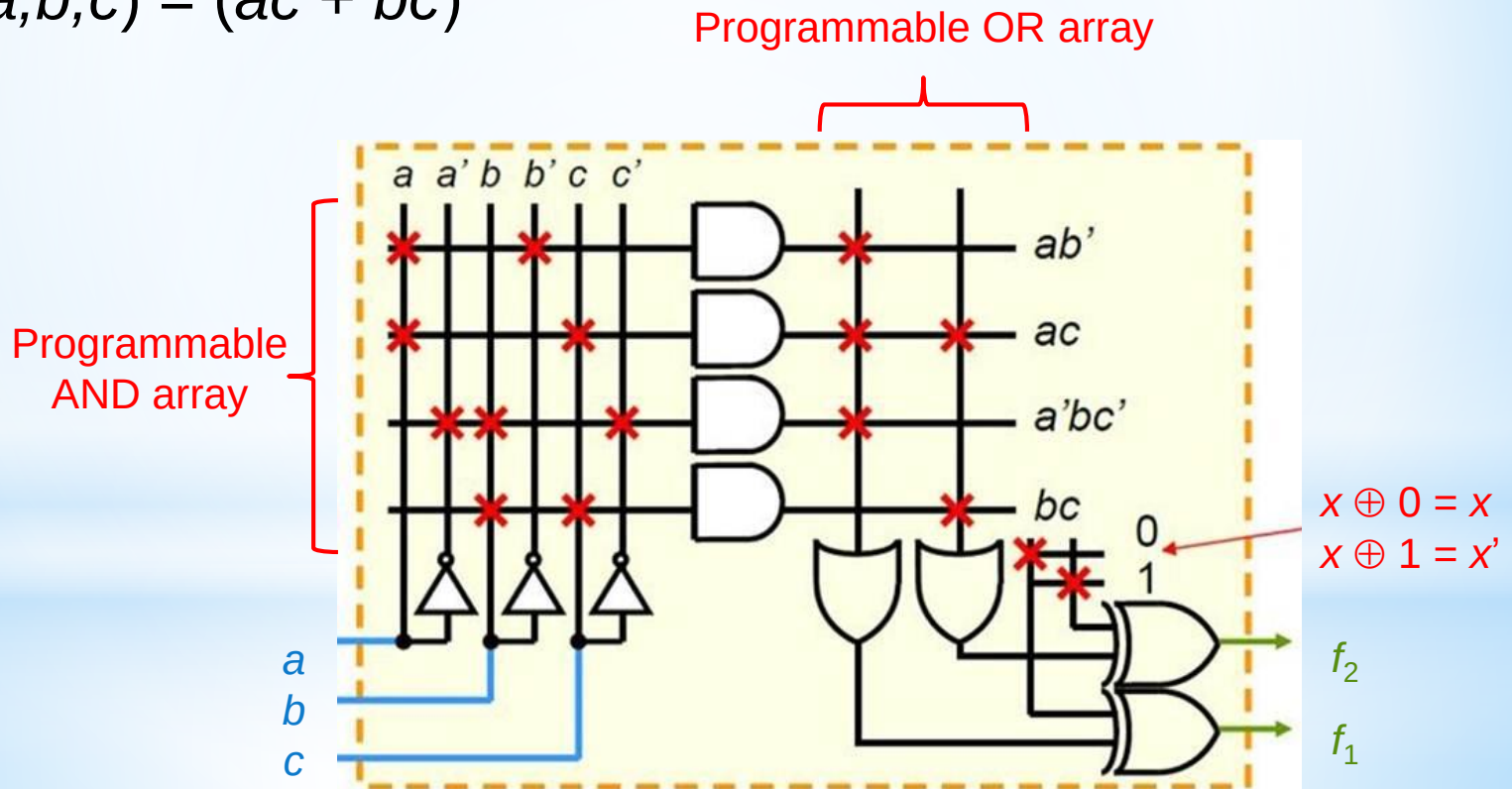


Example: PLA

Implement these 2 functions with a PLA

$$f_1(a,b,c) = ab' + ac + a'bc'$$

$$f_2(a,b,c) = (ac + bc)'$$



- How many product terms for f_1 and f_2 ?

Minimization in PLA

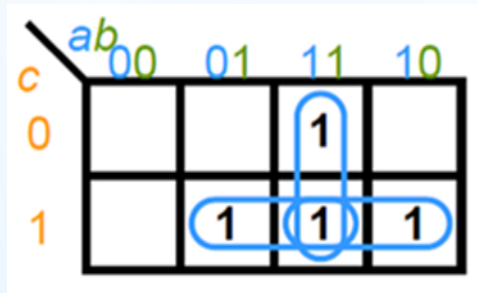
- Reduce the number of distinct product terms, so as to
Minimize the number of AND gates
Minimize the size of PLA
- Step 1) Simplify the Boolean functions, and also their **complementary functions** to minimum number of terms
- Step 2) Find the combination to produce the minimum number of distinct product terms (or maximum common terms)

Example

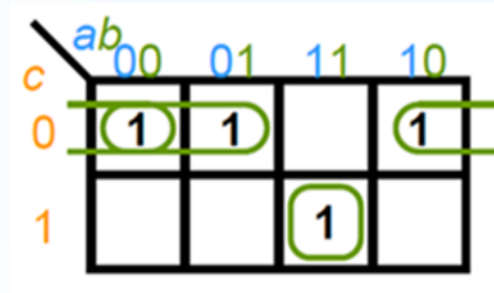
Implement these 2 functions with a PLA

$$f_1(a,b,c) = \sum m(3,5,6,7)$$

$$f_2(a,b,c) = \sum m(0,2,4,7)$$

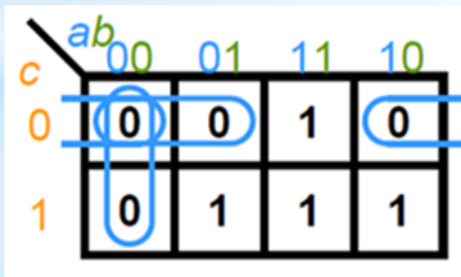


$$f_1 = ac + ab + bc$$

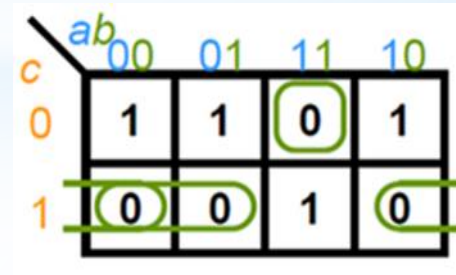


$$f_2 = a'c' + b'c' + abc$$

Now find the simplified complementary functions



$$f_1' = a'c' + a'b' + b'c'$$



$$f_2' = a'c + b'c + abc'$$

- Since there are two functions f_1 and f_2
 - There are 4 complemented and uncomplemented combinations:
 - (i) f_1 and f_2
 - (ii) f_1 and f_2'
 - (iii) f_1' and f_2
 - (iv) f_1' and f_2'
 - We may implement design (i), (ii), (iii) or (iv)
- How to choose?

The design (iii) give the min. no. of distinct product terms (shared 2 common terms)

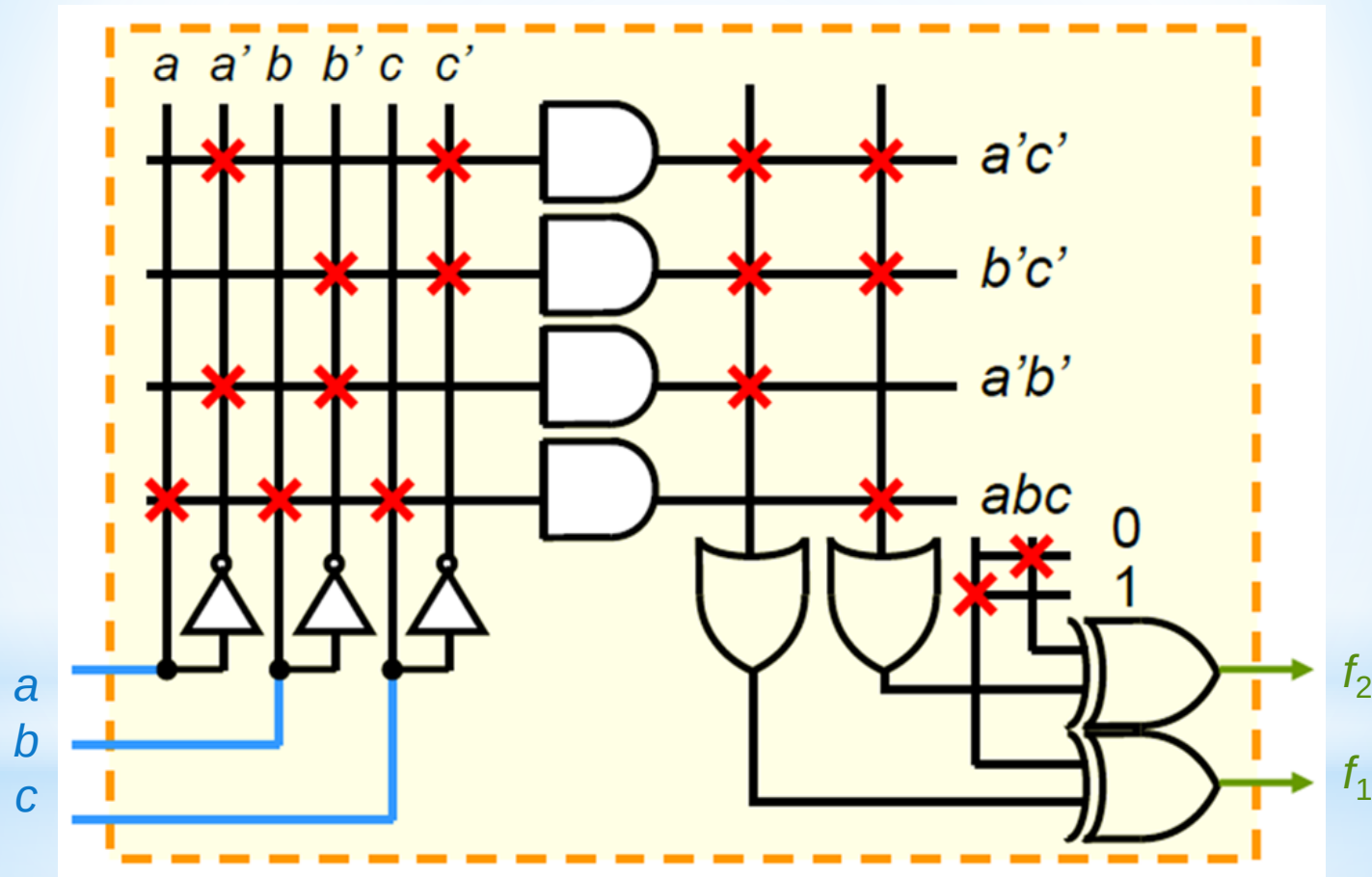
Tabular the result of last step in the PLA programming table

Design (iii): f_1' and f_2

	Product Term	Input a b c	Output f_1 f_2
$a'c'$	1	0 - 0	1 1
$b'c'$	2	- 0 0	1 1
$a'b'$	3	0 0 -	1 -
abc	4	1 1 1	- 1
			C T

T: Uncomplemented output
C: Complemented output

- The final resulting PLA fuse map



4.2.4 Programmable Array Logic (PAL)

- PLA: both the AND and the OR arrays are programmable
- PAL: the AND array is programmable and the OR array is fixed
- PLA is a general design for implementing sum-of-product terms, whereas the PAL has fixed sum-of-product terms
- Design for PAL device is easier, but not as flexible as that for PLA

Use PAL to design a circuit with 4 inputs (a,b,c,d) and 4 outputs (w,x,y,z) .

$$w(a,b,c,d) = \sum m(2,12,13)$$

$$x(a,b,c,d) = \sum m(7,8,9,10,11,12,13,14,15)$$

$$y(a,b,c,d) = \sum m(0,2,3,4,5,6,7,8,10,11,15)$$

$$z(a,b,c,d) = \sum m(1,2,8,12,13)$$

PAL Example

Simplify the functions to minimum number of terms
(steps omitted here)

$$w(a,b,c,d) = abc' + a'b'cd'$$

$$x(a,b,c,d) = a + bcd$$

$$y(a,b,c,d) = a'b + cd + b'd'$$

$$z(a,b,c,d) = abc' + a'b'cd' + ac'd' + a'b'c'd$$

Note: z is sum of 4 product terms

By considering output w as an input variable

$$z(a,b,c,d) = w + ac'd' + a'b'c'd$$

Note: z is SOP of 4 product terms!

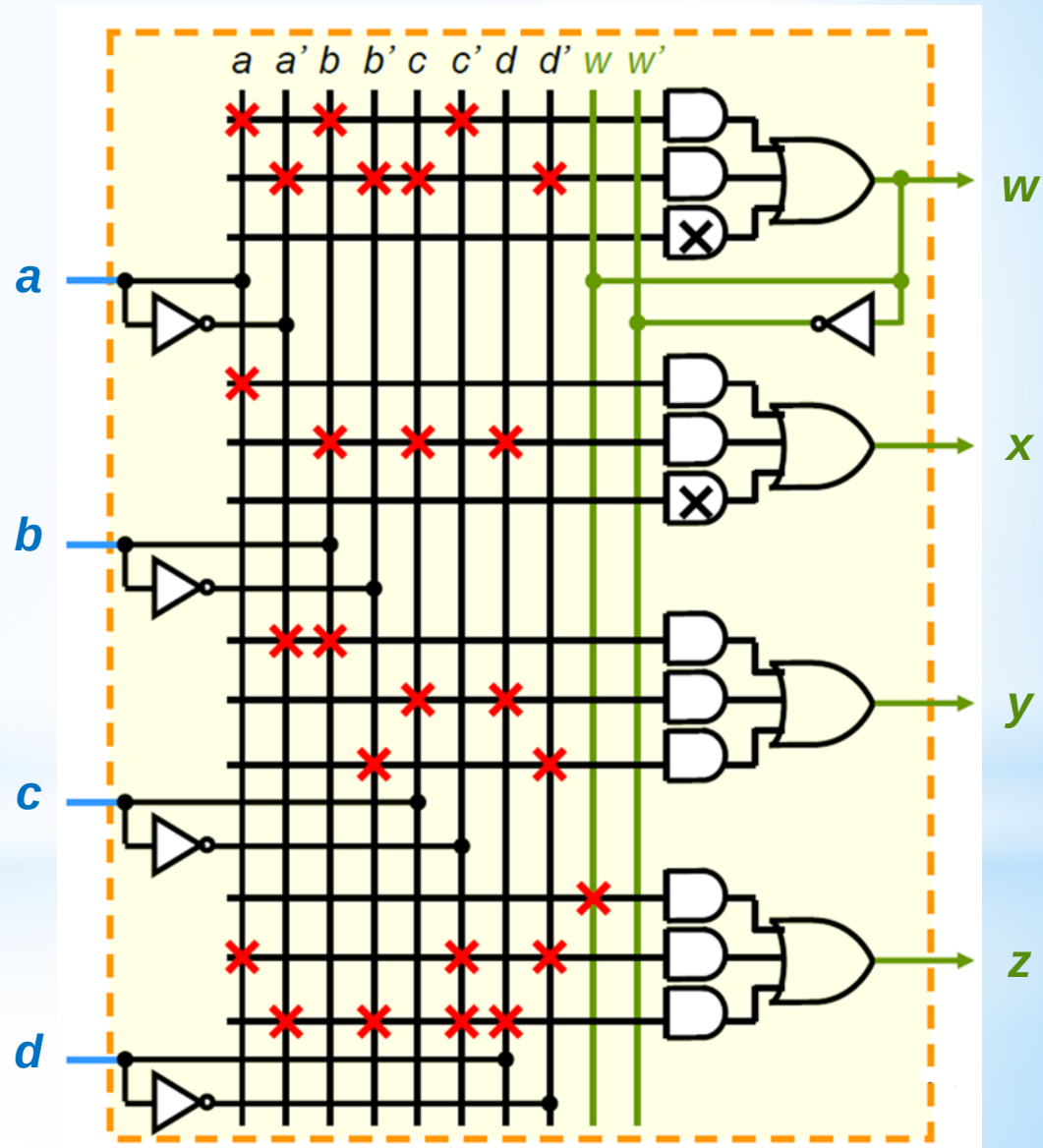
PAL Example

Tabulate the results of last step in the PAL programming table

Product Term		AND Inputs <i>a b c d w</i>	Outputs
1	abc'	1 1 0 – –	$w = abc' + a'b'cd'$
2	$a'b'cd'$	0 0 1 0 –	
3		– – – – –	
4	a	1 – – – –	$x = a + bcd$
5	bcd	– 1 1 1 –	
6		– – – – –	
7	$a'b$	0 1 – – –	$y = a'd + cd + b'd'$
8	cd	– – 1 1 –	
9	$b'd'$	– 0 – 0 –	
10	w	– – – – 1	$z = w + ac'd' + a'b'c'd$
11	$ac'd'$	1 – 0 0 –	
12	$a'b'c'd$	0 0 0 1 –	

PAL Example

- The final result
- Reuse w



PAL 16R6

A PAL device with **16** inputs, **6** outputs with **D-latch** (1-bit memory device), and 2 unbuffered outputs

(8 inputs are feedback from outputs)

This device can be used for designing sequential circuit.

