Lecture 8

Intro. to Combinational Blocks

Revised by WJ Han

Youpyo Hong, Dongguk University

Combinational Logic Design Practices

- **We will learn some basic combinational logic blocks and study how to design them.**

  - Decoder

  - Encoder

  - Multiplexer

  - Demultiplexer

  - Adders/Subtracters

  - Multipliers

Design Procedure

1) Determine the specification.

2) Determine the required inputs and outputs.

3) Derive the truth table.

4) Obtain the simplified Boolean functions.

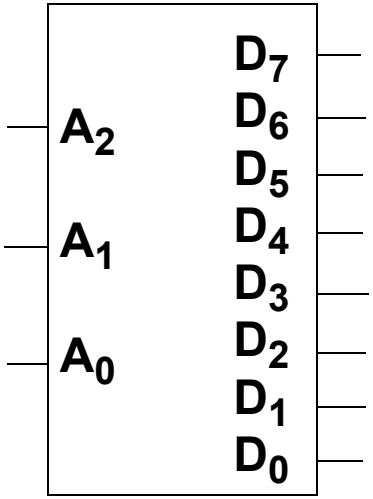
Use K-Map or inspection in this step.

5) Draw the logic diagram.

A Decoder

- **A decoder : a combinational circuit that converts *n* coded binary inputs to *m* (*m* <=2^n) unique outputs. We call it *n*-to-*m* decoder.**

Example. 3-to-8 Decoder



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

We have to make K-Map for each D outputs. But this case is too simple.

$D_0 = \overline{A}_2\overline{A}_1\overline{A}_0$

$D_1 = \overline{A}_2\overline{A}_1A_0$

$D_2 = \overline{A}_2A_1\overline{A}_0$

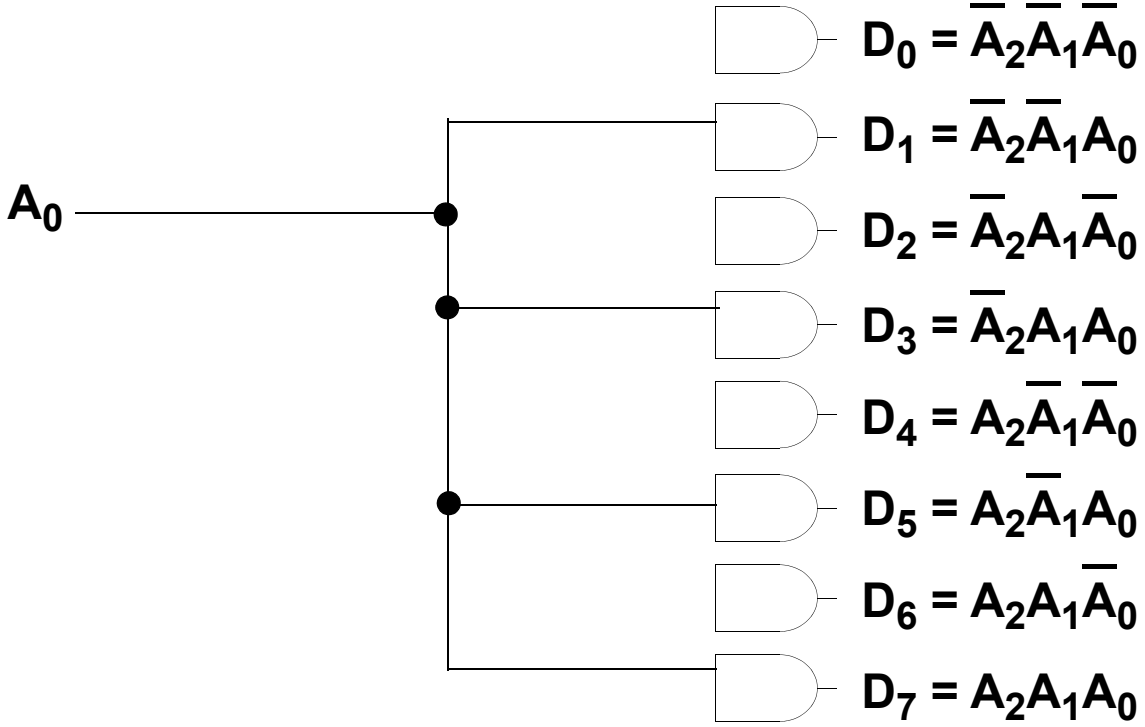
$D_3 = \overline{A}_2A_1A_0$

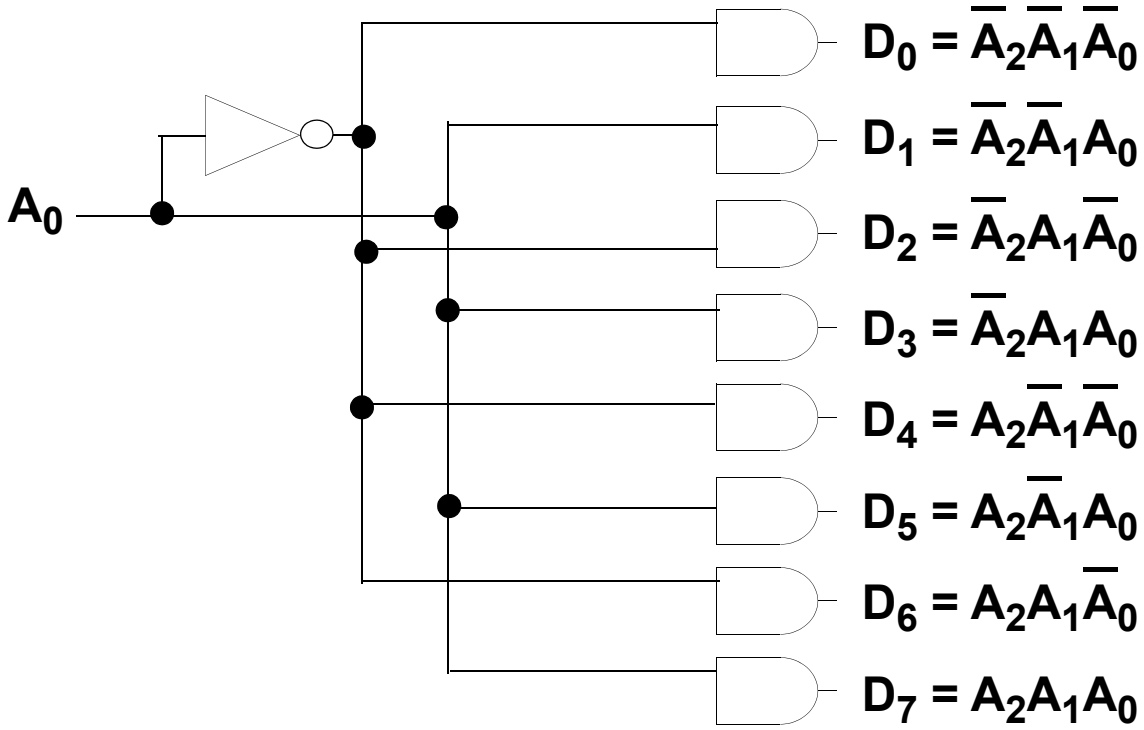
$D_4 = A_2\overline{A}_1\overline{A}_0$

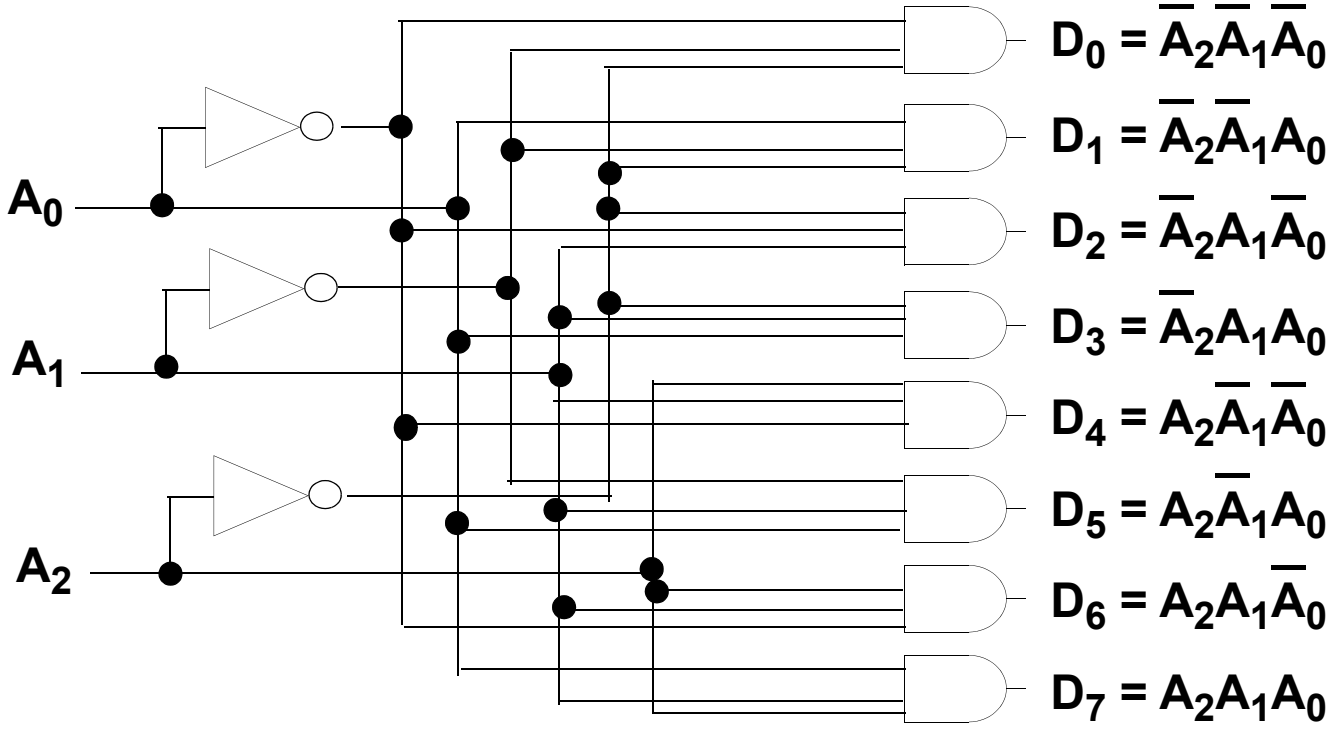
$D_5 = A_2\overline{A}_1A_0$

$D_6 = A_2A_1\overline{A}_0$

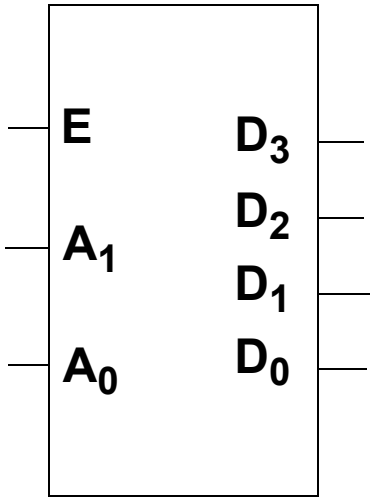
$D_7 = A_2A_1A_0$




A_0
$D_0 = \overline{A}_2\overline{A}_1\overline{A}_0$
$D_1 = \overline{A}_2\overline{A}_1A_0$
$D_2 = \overline{A}_2A_1\overline{A}_0$
$D_3 = \overline{A}_2A_1A_0$
$D_4 = A_2\overline{A}_1\overline{A}_0$
$D_5 = A_2\overline{A}_1A_0$
$D_6 = A_2A_1\overline{A}_0$
$D_7 = A_2A_1A_0$

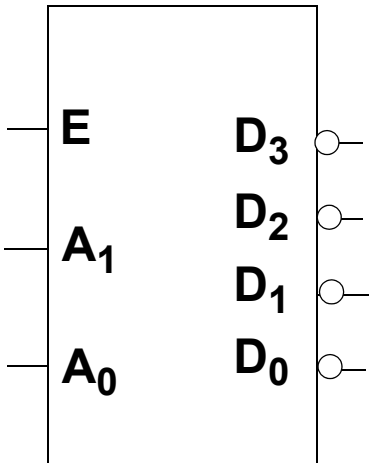

A0
A1
A2
D0 = $\overline{A}_2\overline{A}_1\overline{A}_0$
D1 = $\overline{A}_2\overline{A}_1A_0$
D2 = $\overline{A}_2A_1\overline{A}_0$
D3 = $\overline{A}_2A_1A_0$
D4 = $A_2\overline{A}_1\overline{A}_0$
D5 = $A_2\overline{A}_1A_0$
D6 = $A_2A_1\overline{A}_0$
D7 = $A_2A_1A_0$

2-to-4 Decoder with Enable Input



Inputs			Outputs			
E	A_1	A_0	D_3	D_2	D_1	D_0
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
0	-	-	0	0	0	0

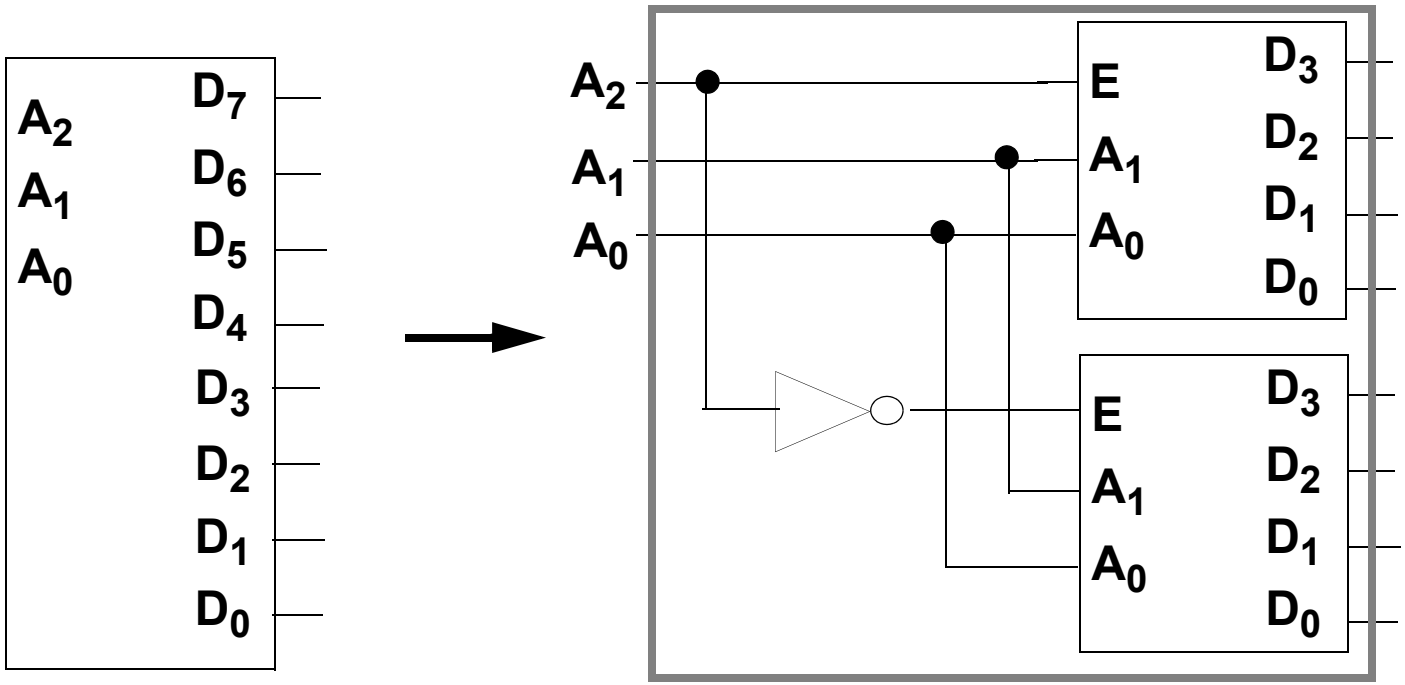
2-to-4 Decoder with Enable Input



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

Decoder Expansion

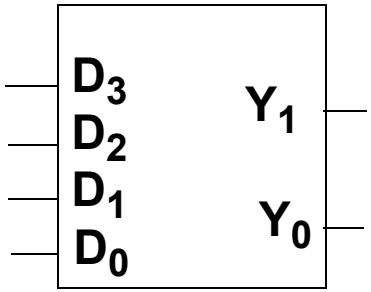
- **When a certain size decoder is needed, if smaller sizes are available, then we need to expand the small size decoders.**



Encoder

- **An encoder : performs the inverse function of decoder, i.e. generate the binary code corresponding to the input value.**

Example. 4-to-2 Encoder



Inputs				Outputs	
D_3	D_2	D_1	D_0	Y_1	Y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$Y_1 = ?$ $Y_0 = ?$

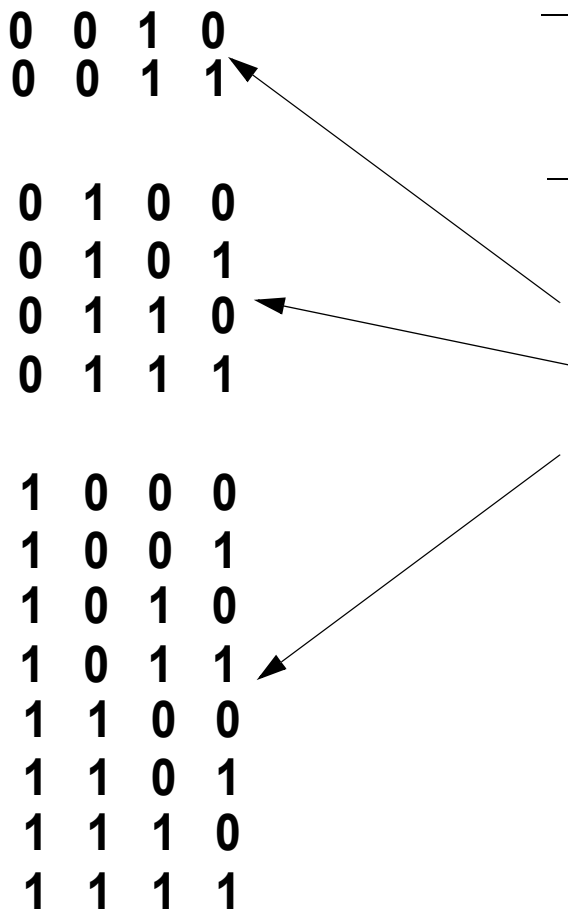
Improved Octal-to-Binary Encoder

- If more than one inputs are simultaneously 1, the output is not defined. --> Priority encoder removes this ambiguity.

- When all inputs are 0, an output of all 0's are generated as if D_0 is 1.

--> Valid output set to on only when at least one input is 1.

Inputs				Outputs		
D_3	D_2	D_1	D_0	Y_1	Y_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	-	0	1	1
0	1	-	-	1	0	1
1	-	-	-	1	1	1



Inputs				Outputs		
D_3	D_2	D_1	D_0	Y_1	Y_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	-	0	1	1
0	1	-	-	1	0	1
1	-	-	-	1	1	1