WEEK 4-2017

Combinational circuit III

Design Procedure

- **Design procedure often involves the following steps:**

1. **Specification - textual or HDL description of the desired circuit. It includes respective symbols or names for inputs and outputs.**

2. **Formulation – involves converting the specification into boolean expression or truth table.**

3. **Optimization – producing simplified boolean expression using algebraic manipulation, K-map method, or computer based optimization tools. It also involves drawing a logic diagram or a netlist using AND, OR and NOT.**

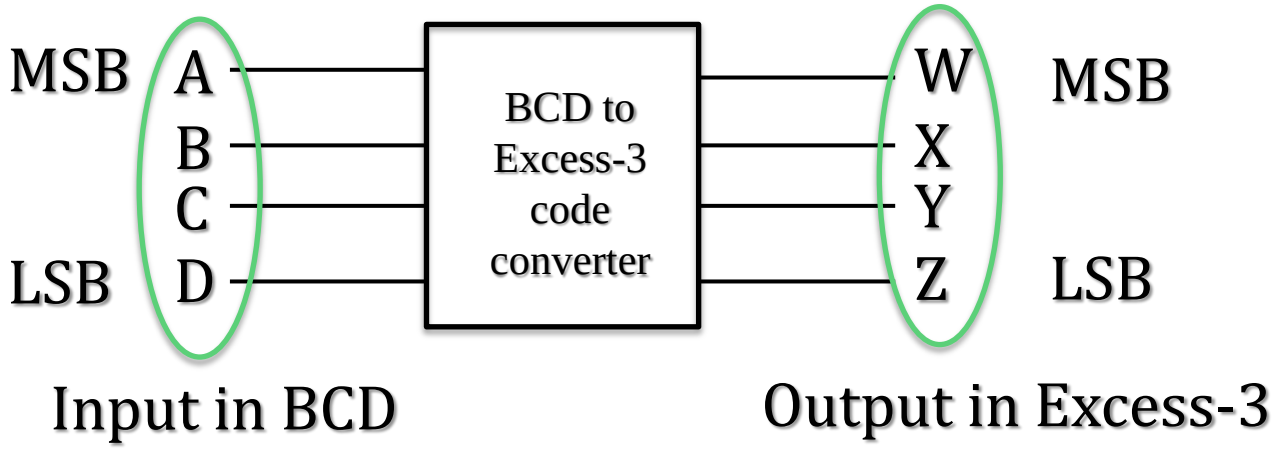
Design Procedure

4. **Technology mapping – transforming the logic diagram or netlist based on the available implementation technology**

5. **Verification – to determine the correctness of the design. Manual logic analysis and computer simulation based logic analysis.**

Design Example 1

Design of a BCD to Excess-3 Code Converter



Specification – the excess -3 code for a decimal digit is the binary combination corresponding to the decimal digit plus 3.

Design Example 1

Design of a BCD to Excess-3 Code Converter

Formulation:

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

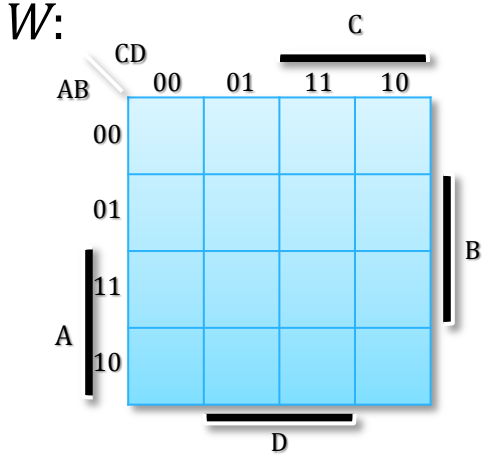
Design Example 1

Design of a BCD to Excess-3 Code Converter

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Optimization:

W:



$$W = A + BC + BD$$

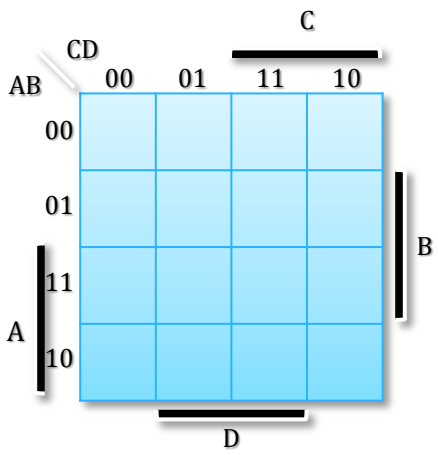
Design Example 1

Design of a BCD to Excess-3 Code Converter

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Optimization:

X:



$$X = \bar{B}C + \bar{B}D + B\overline{C}\,\overline{D}$$

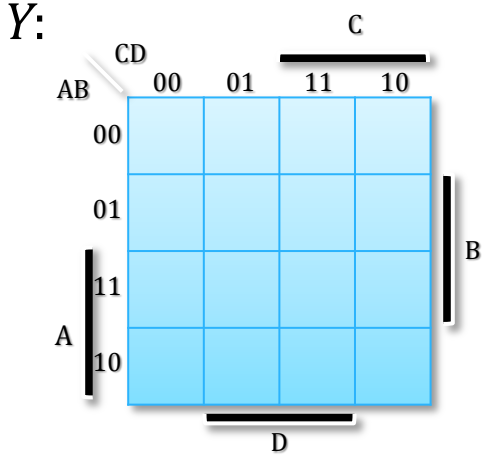
Design Example 1

Design of a BCD to Excess-3 Code Converter

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Optimization:

Y:



$$Y = CD + \bar{C}\bar{D}$$

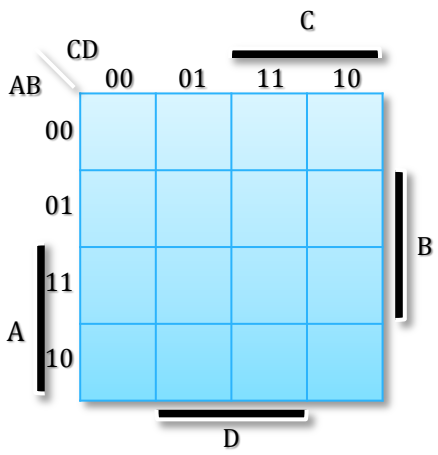
Design Example 1

Design of a BCD to Excess-3 Code Converter

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Optimization:

Z:



$$Z = \overline{D}$$

Design Example 1

Design of a BCD to Excess-3 Code Converter

Optimization:

- Two level implementation as obtained from the map

$$W = A + BC + BD$$

$$X = \bar{B}C + \bar{B}D + B\overline{C}\,\overline{D}$$

$$Y = CD + \bar{C}\,\overline{D}$$

$$Z = \overline{D}$$

- will have gate input costs of 26

Design Example 1

Design of a BCD to Excess-3 Code Converter

Optimization:

- Multi-level implementation

$$T_1 = C + D \quad T_1 = \overline{C + D} = \bar{C}\bar{D}$$

$$W = A + BC + BD = A + BT_1$$

$$X = \bar{B}(C + D) + B\bar{C}\bar{D} = \bar{B}T_1 + B\bar{T}_1$$

$$Y = CD + \bar{C}\bar{D} = CD + \bar{T}_1$$

$$Z = \bar{D}$$

- will have gate input costs of 19

Design Example 1

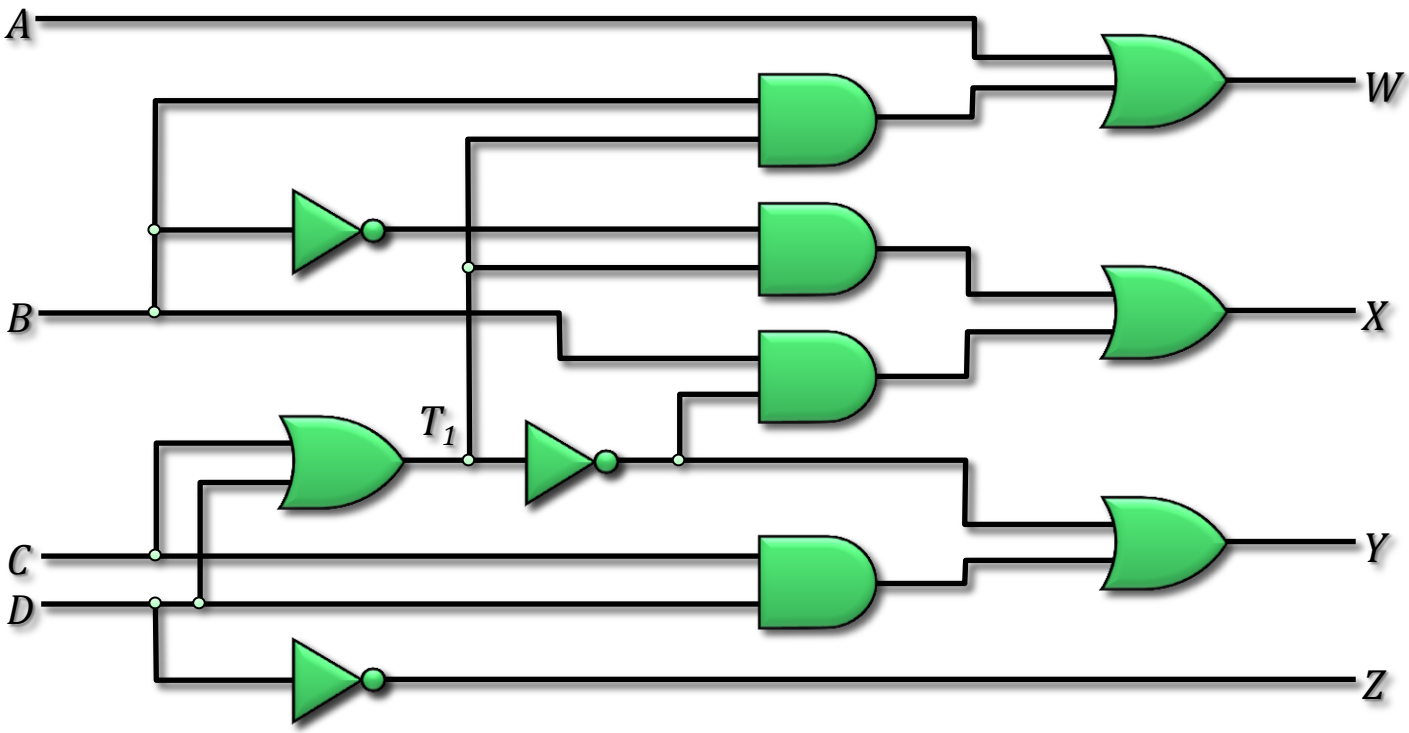
Design of a BCD to Excess-3 Code Converter

Optimization: Draw logic diagram

$$T = C + D$$

$$W = A + BT_1 \qquad X = B\overline{T}_1 + \overline{B}T_1$$

$$Y = CD + \overline{T}_1 \qquad Z = \overline{D}$$

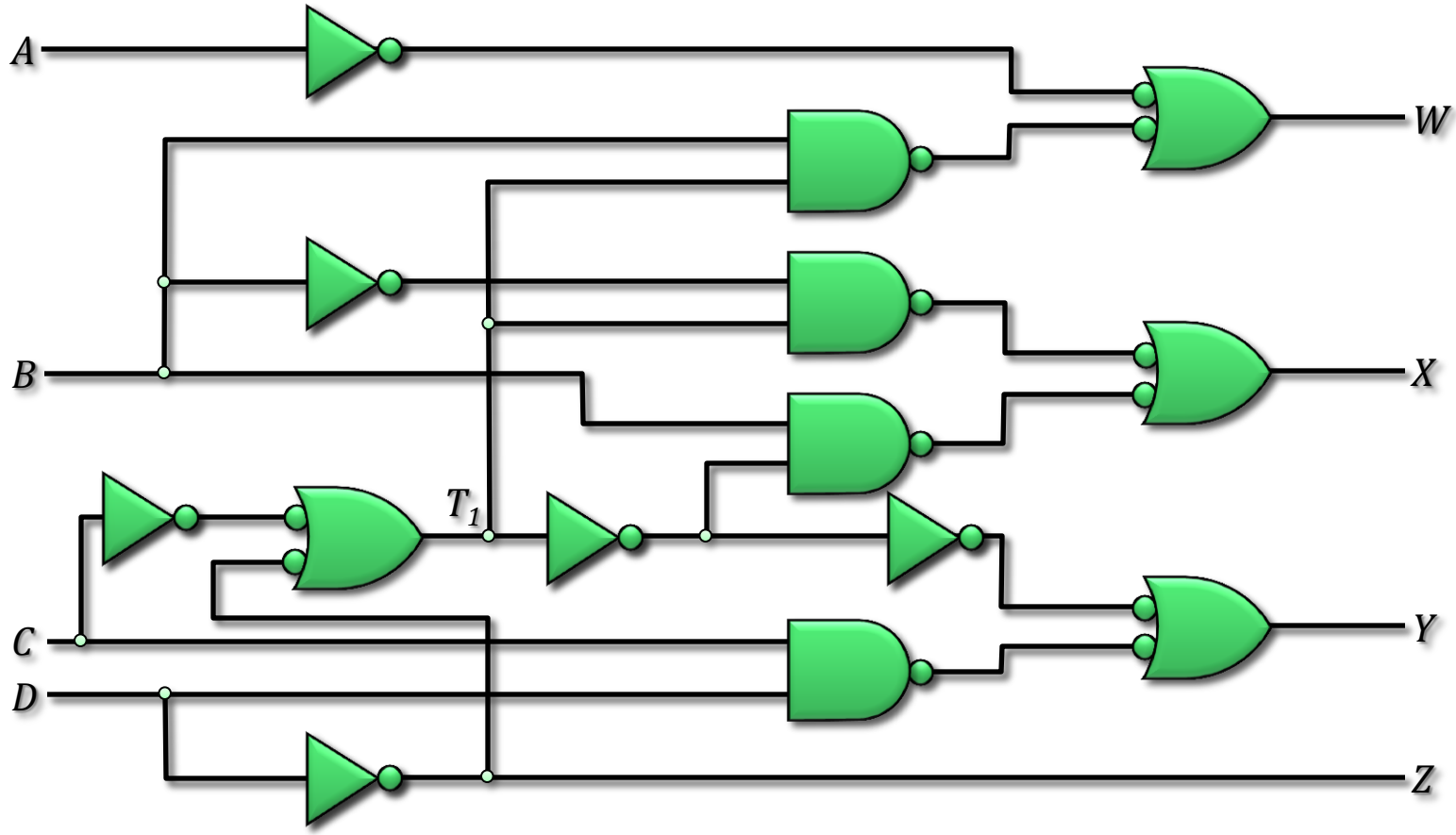


Design Example 1

Design of a BCD to Excess-3 Code Converter

Technology Implementation: NAND ONLY implementations

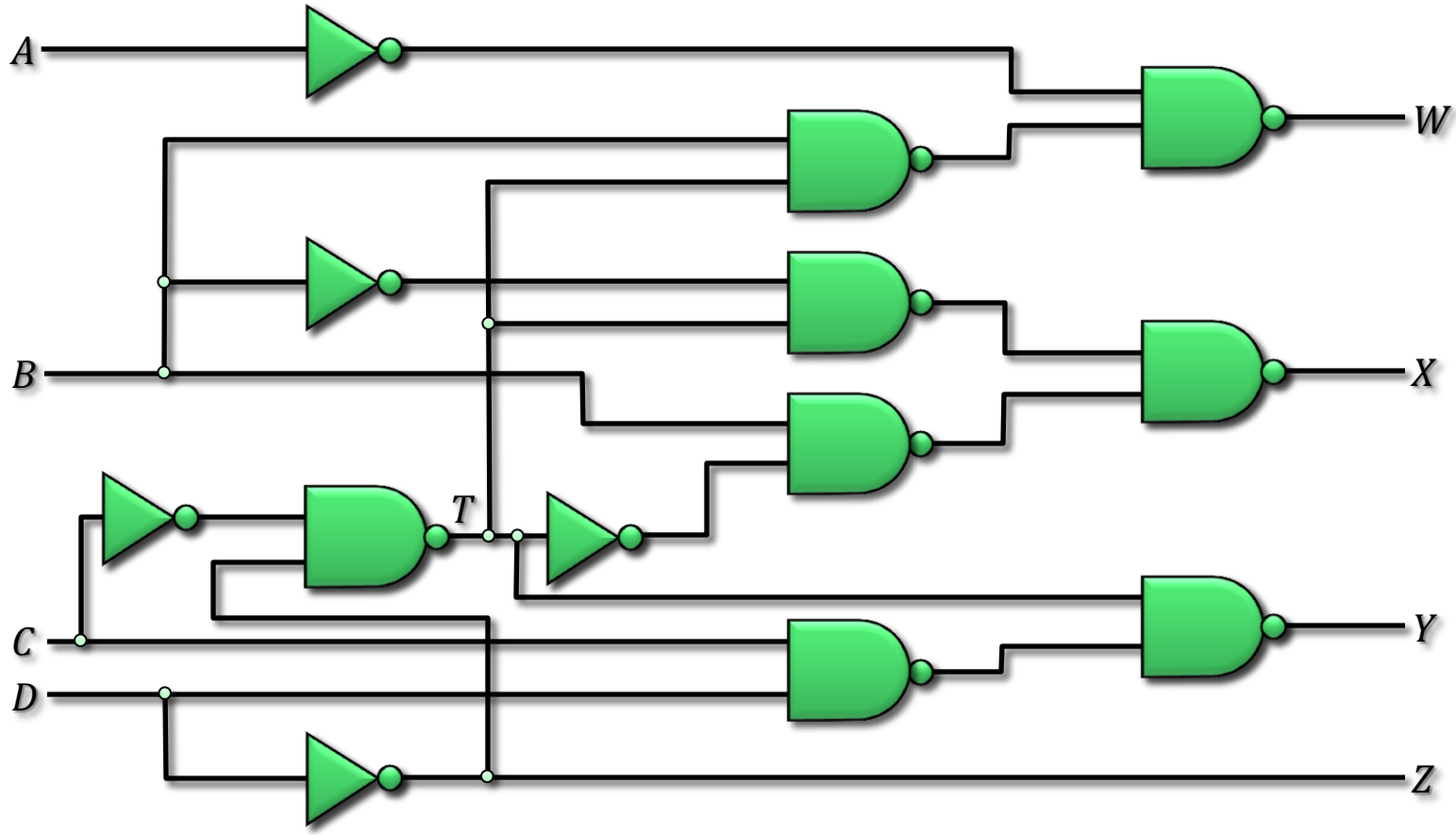
-Add Bubbles



Design Example 1

Design of a BCD to Excess-3 Code Converter

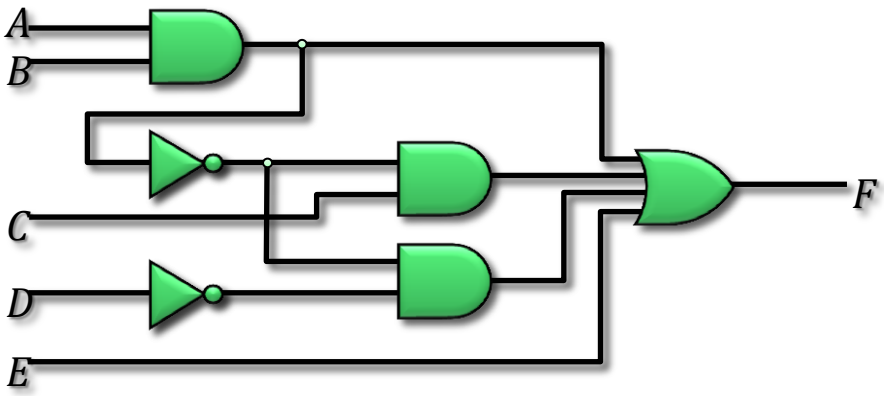
Technology Implementation: NAND ONLY logic diagram



NAND ONLY Implementation

Technology Implementation: NAND ONLY logic diagram

$$F = AB + (\overline{AB})C + (\overline{AB})\overline{D} + E$$



NAND ONLY Implementation

Technology Implementation: NAND ONLY logic diagram

$$F = AB + (\overline{AB})C + (\overline{AB})\overline{D} + E$$

NOR ONLY Implementation

Technology Implementation: NOR ONLY logic diagram

$$F = AB + (\overline{AB})C + (\overline{AB})\overline{D} + E$$

Design Example 1

Design of a BCD to Excess-3 Code Converter

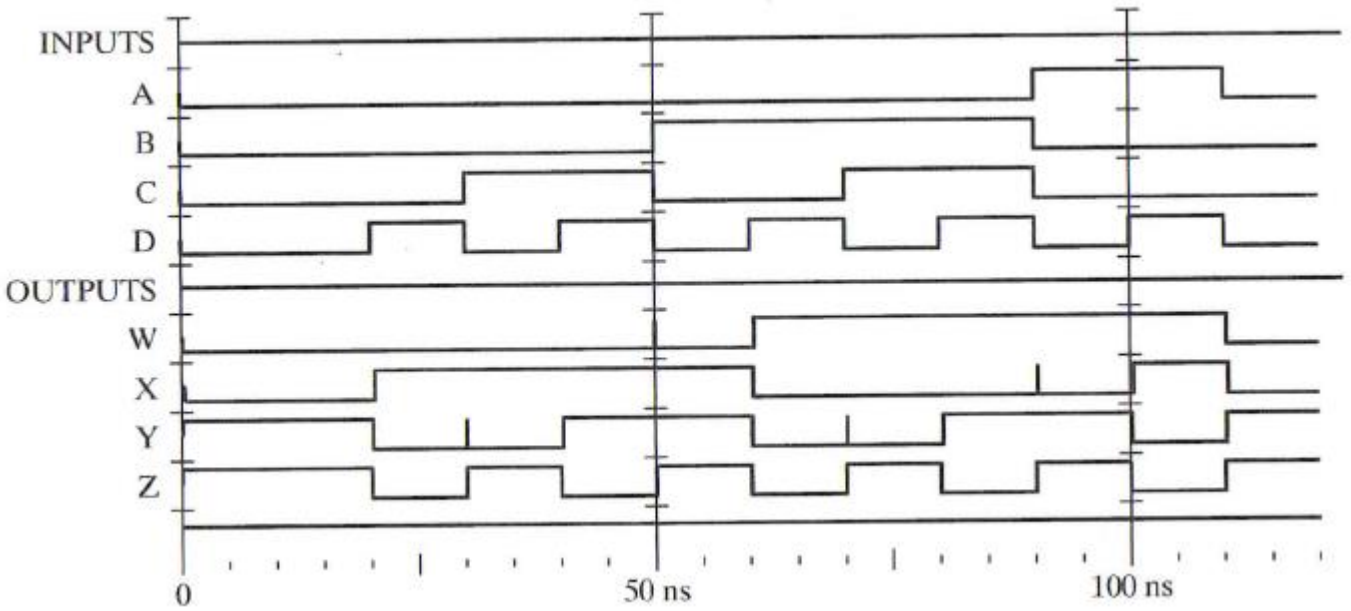
Verification: (Manual)

Decimal Digit	Input BCD				Output Excess-3			
	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1				
2	0	0	1	0	0	1	0	1
3	0	0	1	1				
4	0	1	0	0	0	1	1	1
5	0	1	0	1				
6	0	1	1	0	1	0	0	1
7	0	1	1	1				
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Design Example 1

Design of a BCD to Excess-3 Code Converter

Verification:
(CAD)

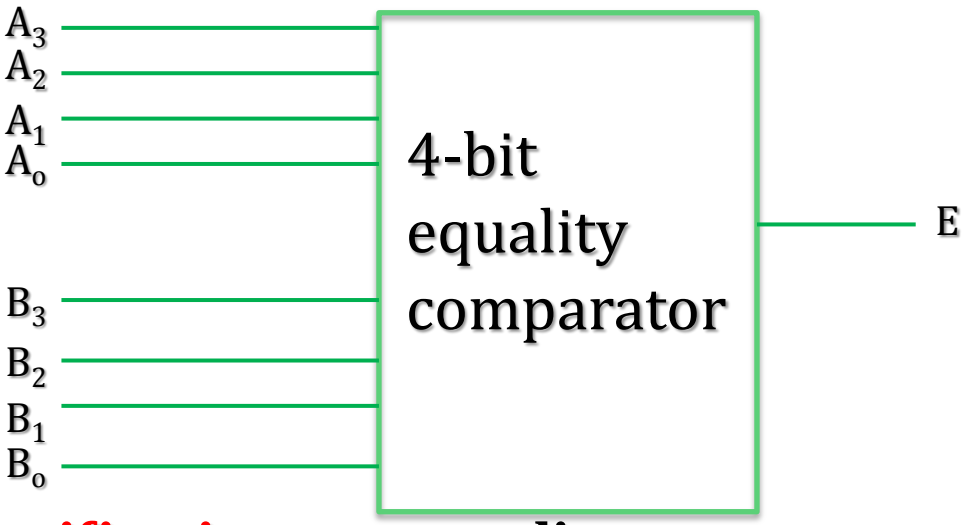


Hierarchical design

- **Complex or large digital circuits will be broken down into smaller and simpler circuits called blocks.**
- **The blocks are interconnected to form the complex circuit.**
- **The circuit formed by interconnecting the blocks obeys the initial circuit specification.**
- **A block can further be broken down if found large to be designed as a single entity.**

Hierarchical design- Example

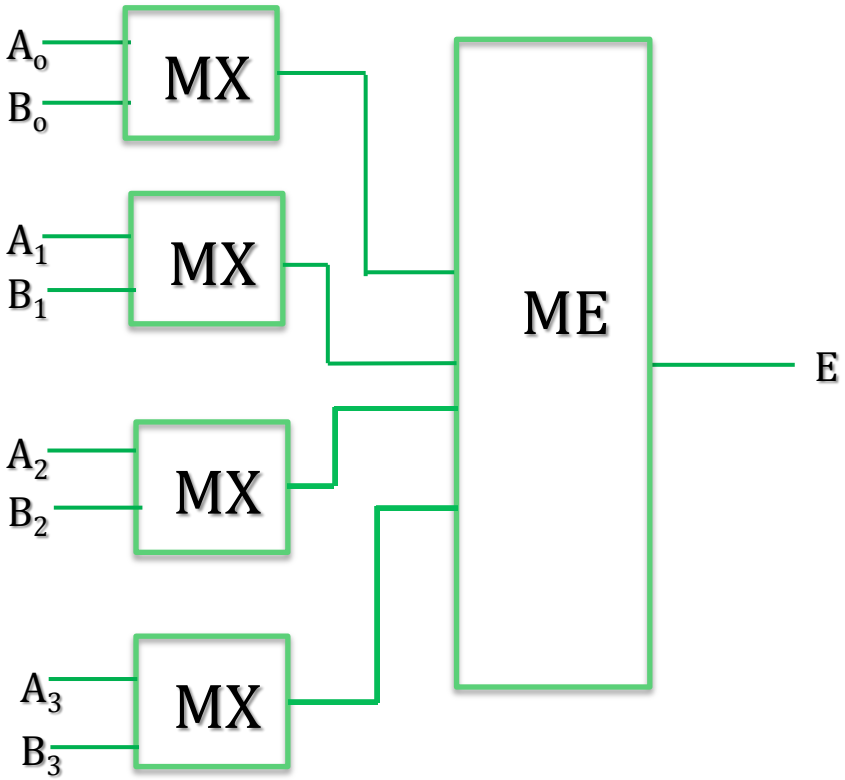
- **Design a 4-bit Equality comparator**



Specification: an equality comparator is a circuit that compares two binary vectors to determine whether they are equal or not. The inputs to this specific circuit consists of two vectors: A(3:0) and B(3:0). The output of the circuit is a single bit variable E. Output E is equal to 1 if vectors A and B are equal and equal to 0 if vectors A and B are unequal.

Hierarchical design- Example

- **Design a 4-bit Equality comparator**
- **The use of truth table and K-map are not much of a help because we will need a truth table with 256 rows and K-map for eight variables.**
- **Use hierarchical approach- break the function into four identical blocks of MX, one-bit comparator, and additional circuit ME that combines the output of each of the comparator into a single output E**



Hierarchical design- Example

- **Design a 1-bit Equality comparator**

Specification : a 1 bit Equality comparator compares if two bits are the same or not. It will have two inputs, say A_i and B_i, and one output, N_i , producing binary 1 if the two bits A_i and B_i are the same.

Formulation : draw truth table and find boolean function

A_i	B_i	N_i
0	0	1
0	1	0
1	0	0
1	1	1

$$N_i = \overline{A_i}\,\overline{B_i} + A_i B_i$$

Hierarchical design- Example

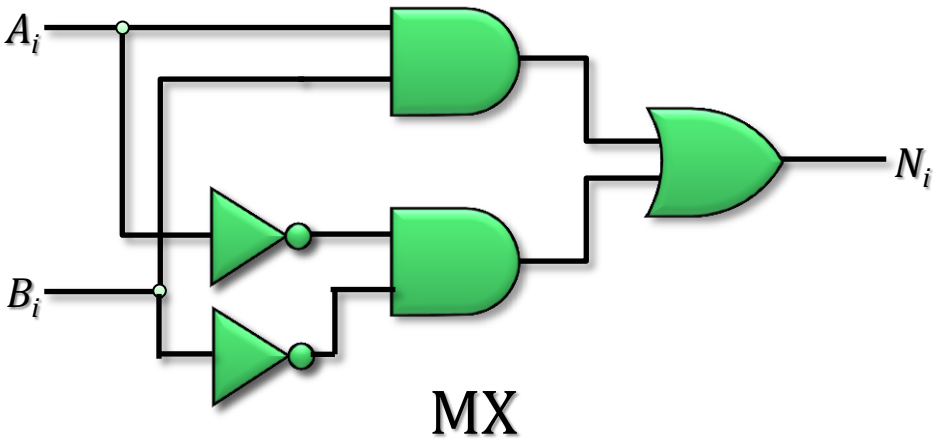
- **Design a 1-bit Equality comparator**

Optimization : use two variable K-map

$$N_i = \overline{A_i}\,\overline{B_i} + A_i B_i$$

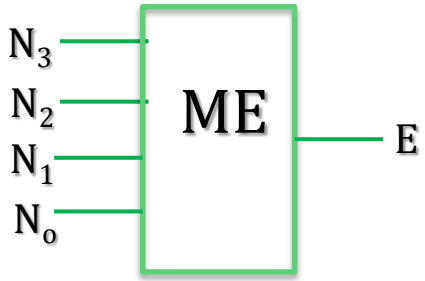
A \ B	0	1
0	1	0
1	0	1

Logic diagram



Hierarchical design- Example

- **Design ME, a circuit that combines the outputs of four 1-bit comparator.**



Specification : E is 1 if the four inputs, N_o,N_1,N_2 and N_3 are all 1;Otherwise E is 0.

Formulation: simply obtain boolean function

$$E = N_o N_1 N_2 N_3$$

Optimization: cannot be optimized further as there is only one "1" in a four variable K-map. Hence,

$$E = N_o N_1 N_2 N_3$$

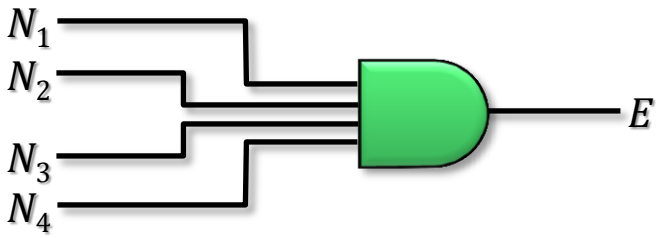
Hierarchical design- Example

- **Design ME, a circuit that combines the outputs of four 1-bit comparator.**

Optimization: cannot be optimized further as there is only one "1" in a four variable K-map. Hence,

$$E = N_o N_1 N_2 N_3$$

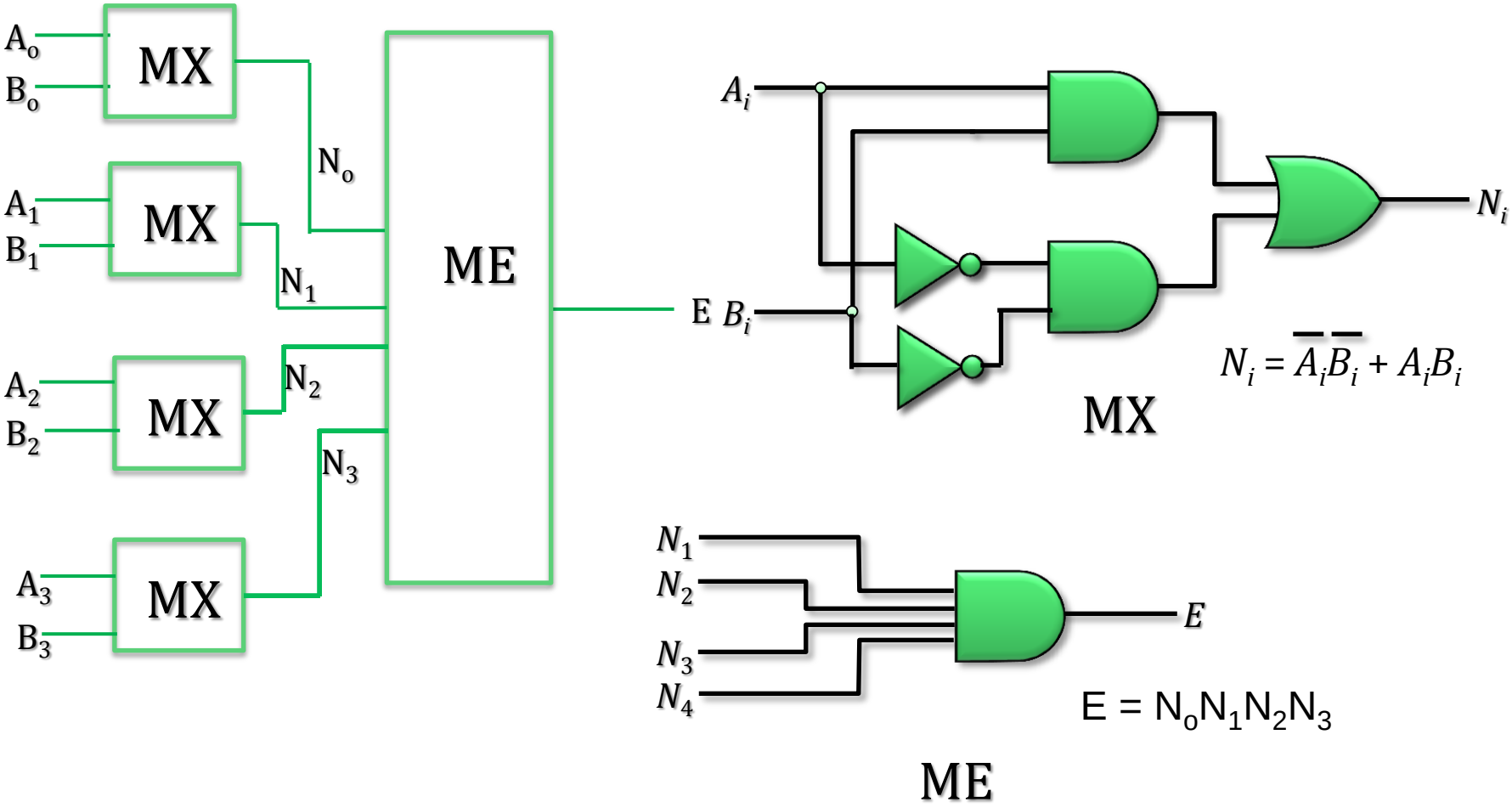
Logic diagram:



ME

Hierarchical design- Example

- **A 4-bit Equality comparator**



$N_i = \overline{A_i}\overline{B_i} + A_iB_i$

$E = N_0N_1N_2N_3$

Hierarchical design

- **Reduces the complexity required to represent the schematic diagram circuit**

- **Allows the reuse of blocks. In previous example, only one of the four one-bit comparator needs to be designed. A copy of the design can be used for the others.**

- **Designer needs to look for regularities in the circuit. The more regular the designer can abstract a system, the easier it becomes to design as less no. of blocks are required to be designed.**

- **The appearance of a block in a design is called an instance.**

Hierarchical design

- **Circuits consisting of million gates are designed using hierarchical design using CAD design tools.**

- **There are predefined reusable blocks that are used for such purpose. These blocks are called functional blocks, and they are a predefined collection of interconnected gates.**

- **Functional blocks with specific combinational functions are available as instances in most CAD tools as well as discrete ICs. These include Decoders, Encoders, and Multiplexers.**

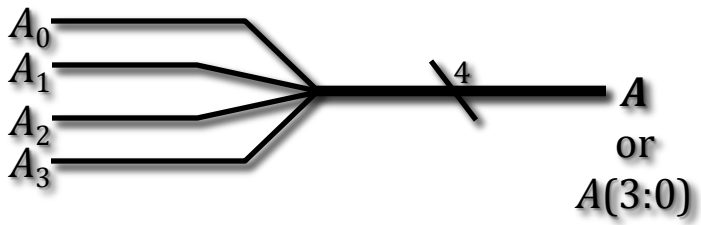
Rudimentary Logic Functions

- **For a single variable, *X*, four different functions are possible:**

X	F = 0	F = X	$F = \overline{X}$	F = 1
0	0	0	1	1
1	0	1	0	1
	0-Value Fixing	Transferring	Inverting	1-Value Fixing
				

Multi-Bit Variables

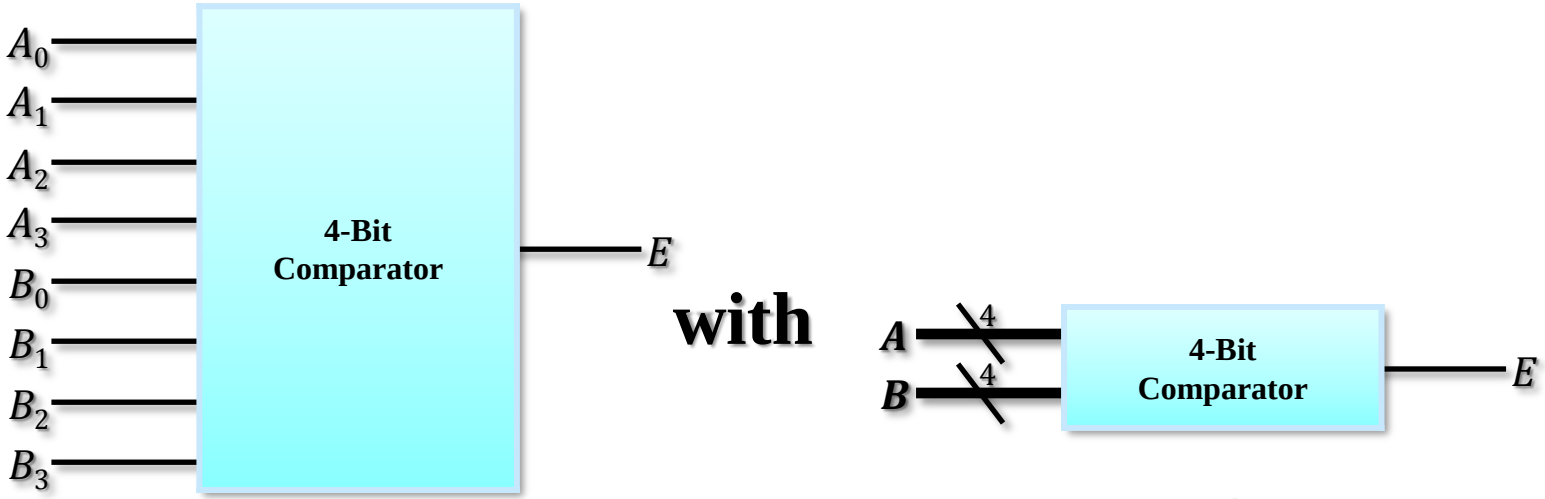
- **A group of single-bit variables that are used together can be gathered into a *bus* which is a vector signal**
- **Example: the input signals to the 4-bit equality comparator – $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ can be expressed as $A(3:0)$ and $B(3:0)$**
- **Buses are represented graphically using thicker lines and an integer to specify their size:**



Bus Signals

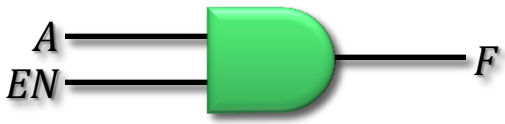
- **Buses make circuits clearer to read**
- **They also significantly simplify design and simulation in software**

- **Compare:**



Enabling Circuits

- **When disabled, output 0:**



EN	F
0	0
1	*A*

- **When disabled, output 1:**

EN	F
0	1
1	*A*

Decoding

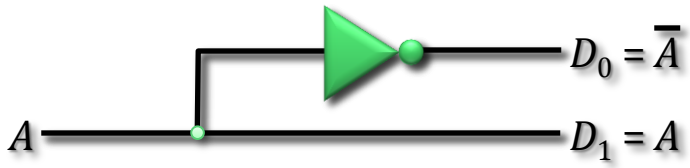
- ***Decoders*** **are circuits that convert *n*-bit input code to *m*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code**

- **Functional blocks for decoding are called *n*-to-*m*-line decoders**

- **The decoder generates 2^n (or fewer) minterms for the *n* input variables**

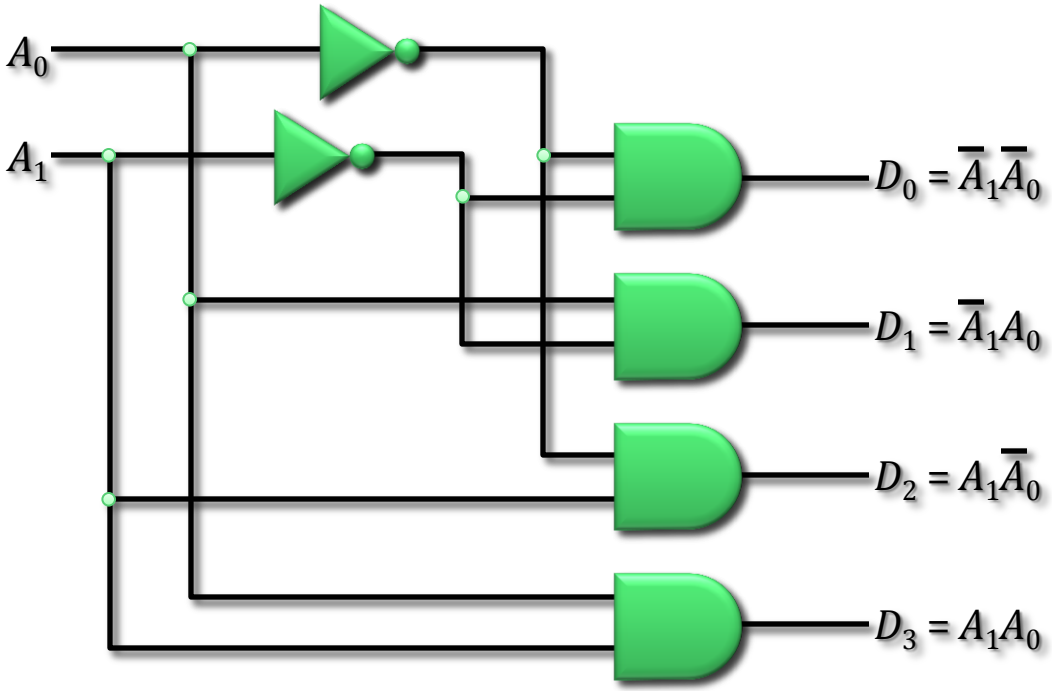
Decoder Examples

- **1-to-2-line decoder:**



A	D_0	D_1
0	1	0
1	0	1

- **2-to-4-line decoder:**



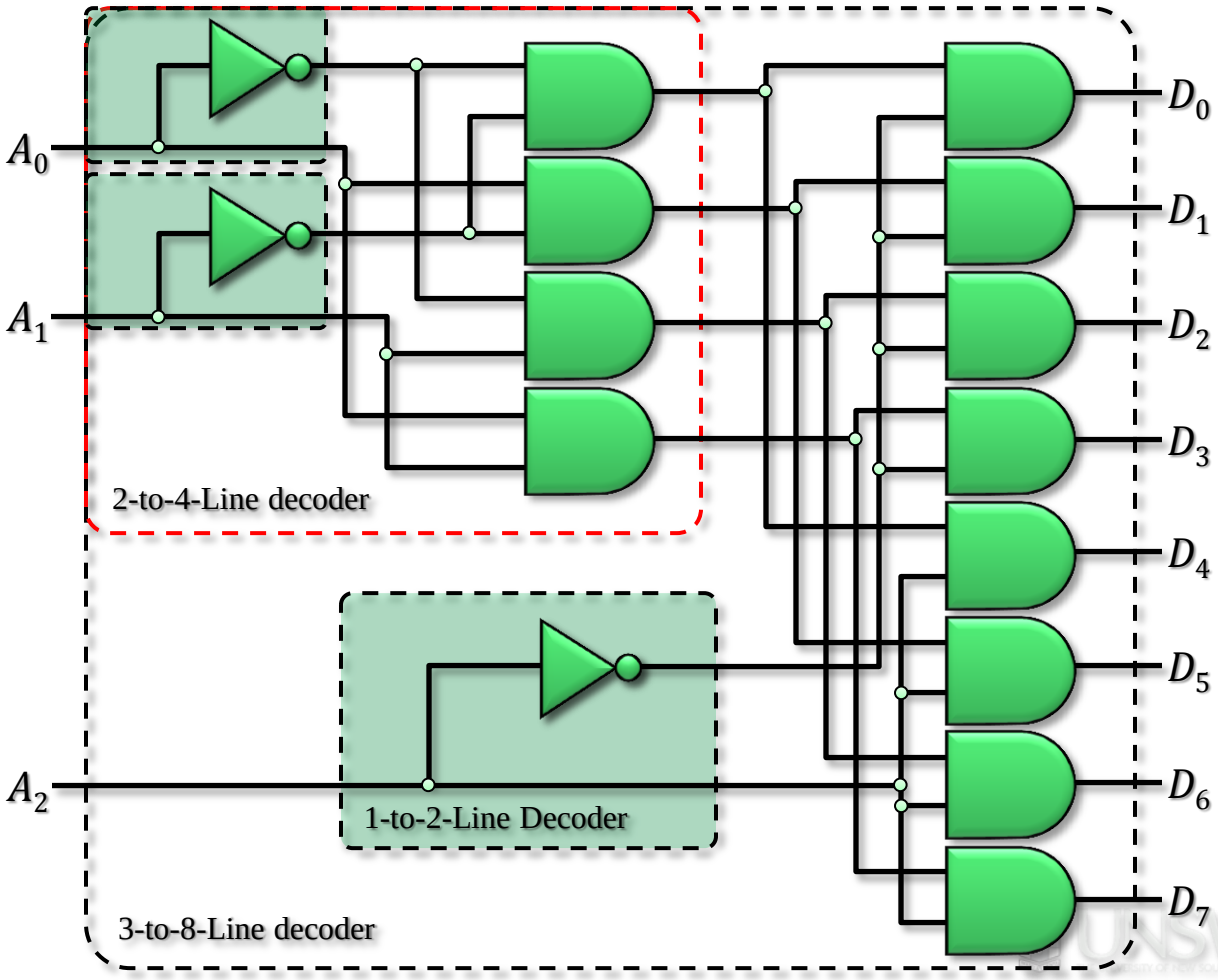
A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoder Examples

- **3-to-8-line decoder:**

Decoder Expansion

- **Decoders can be cascaded together to generate larger decoders using only 2-input AND gates:**



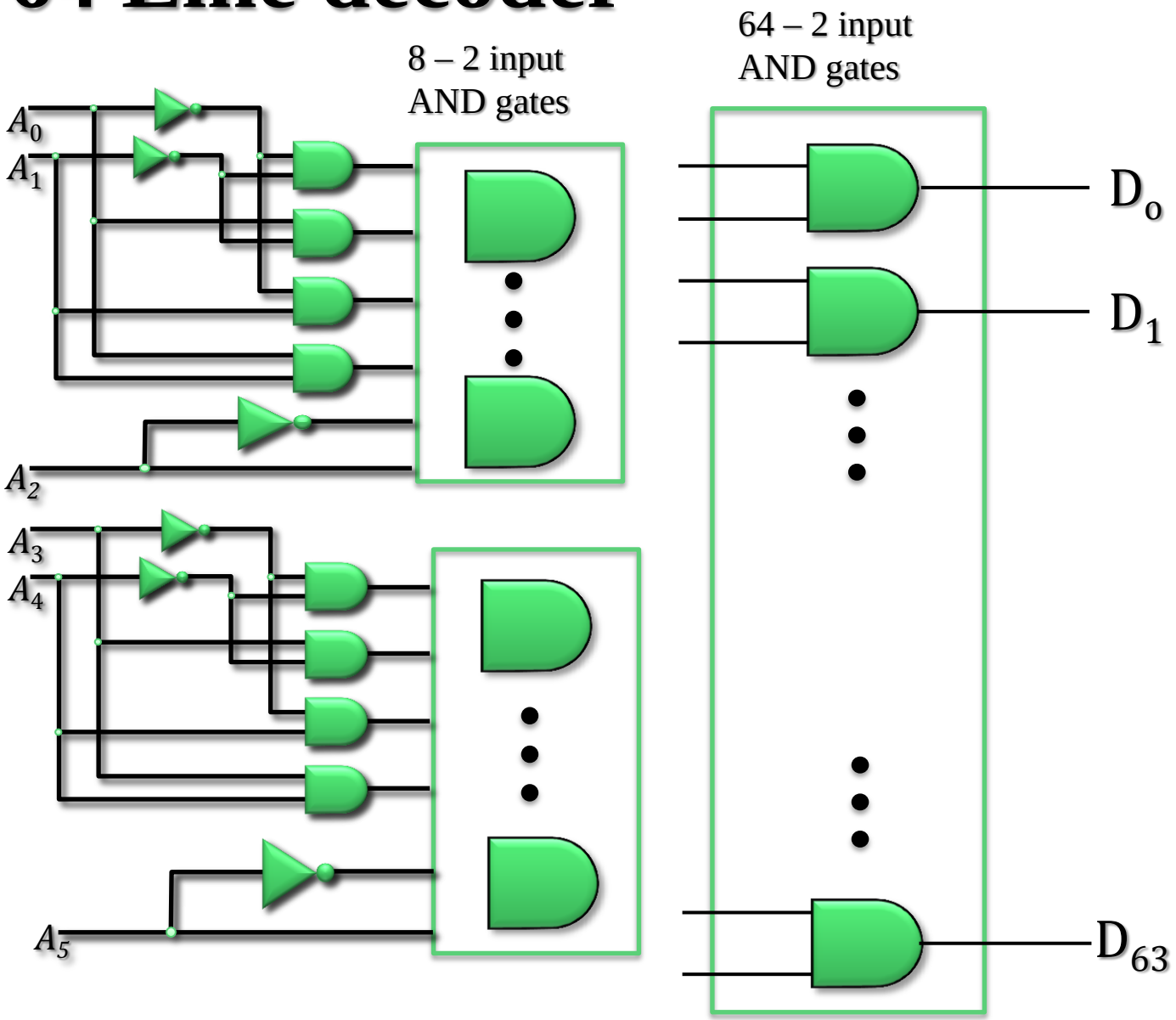
Decoder Expansion Algorithm

- **The general procedure to construct *n*-bit decoder using 2-input AND gates only:**

1. **Let $k = n$.**
2. **If k is even, divide k by 2 to obtain $k/2$. Use 2^k AND gates driven by two decoders of output size $2^{k/2}$. If k is odd, obtain $(k + 1)/2$ and $(k – 1)/2$. Use 2^k AND gates driven by a decoder of output size $2^{(k + 1)/2}$ and a decoder of output size $2^{(k – 1)/2}$.**
3. **For each decoder resulting from step 2, repeat step 2 with k equal to the values obtained in step 2 until $k = 1$. For $k = 1$, use a 1-to-2 decoder.**

6-to-64 Line decoder

- **k = 6**



6-to-64 decoder

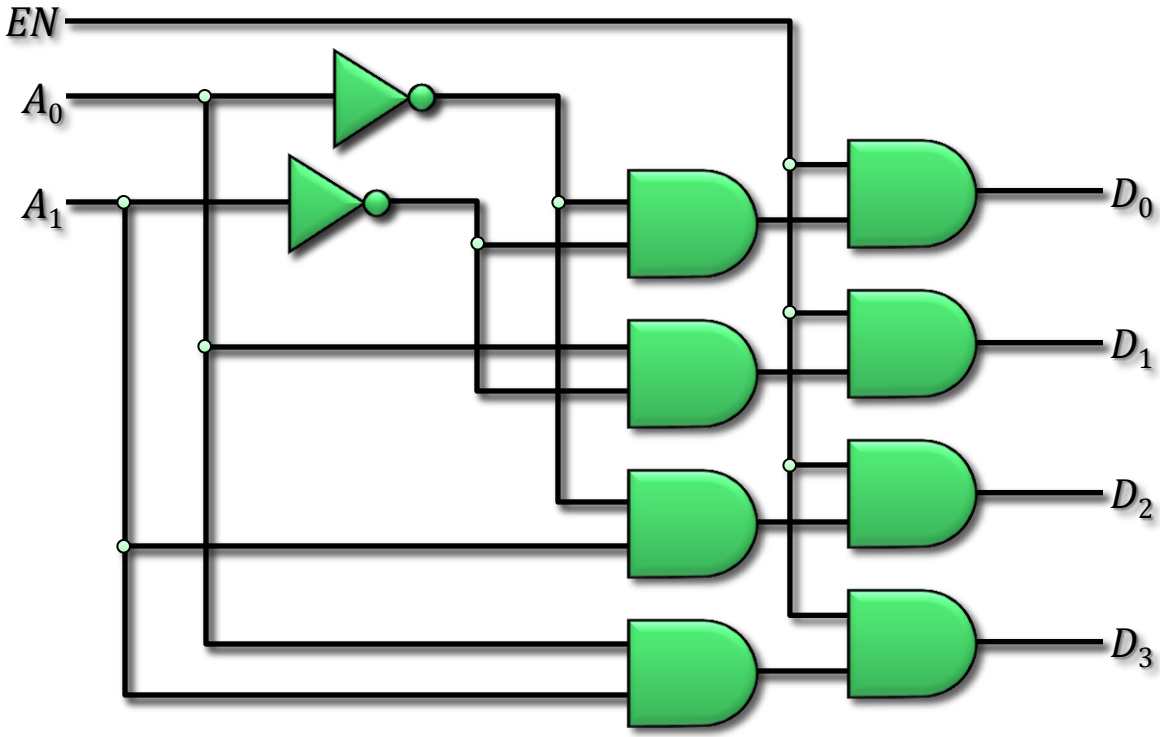
- **Using a two input AND approach, the total cost = 6 + 4x2x2 + 8x2x2 + 64x2 = 182**

- **Had the decoder been realized using 6-input AND gates, the total cost would have been 6 + 64x6 = 390 input gates.**

- **Substantial gate input cost reduction has been achieved.**

Decoder with Enable

- Add an *Enable* input to the decoder
- Allows to set all outputs to 0 when disabled



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-Based Combinational Circuits

- **Any Boolean function can be implemented using a decoder and an OR gate**
- **Example – 1-bit binary adder:**

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

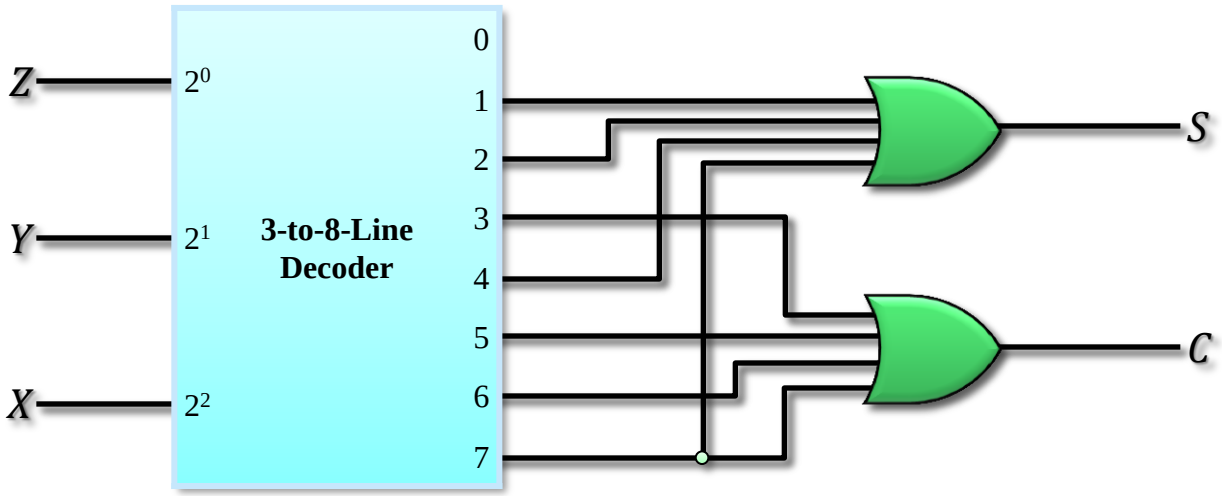
$$S(X, Y, Z) = \sum m(1, 2, 4, 7)$$

$$C(X, Y, Z) = \sum m(3, 5, 6, 7)$$

Decoder-Based 1-Bit Binary Adder

$$S(X, Y, Z) = \sum m(1, 2, 4, 7)$$

$$C(X, Y, Z) = \sum m(3, 5, 6, 7)$$



Encoding

- ***Encoders*** **perform the inverse operation to decoders**
- **An encoder has 2^n (or fewer) input lines and *n* output lines**
- **The output generates the binary code corresponding to the input value**
- **Typically, only one of the inputs can have a value of 1 at any time**

Octal-to-Binary Encoder

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

$$A_0 = D_1 + D_3 + D_5 + D_7$$

$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$

Priority Encoders

- ***Priority Encoders*** **solve the restriction that only one input can be 1 at any time**

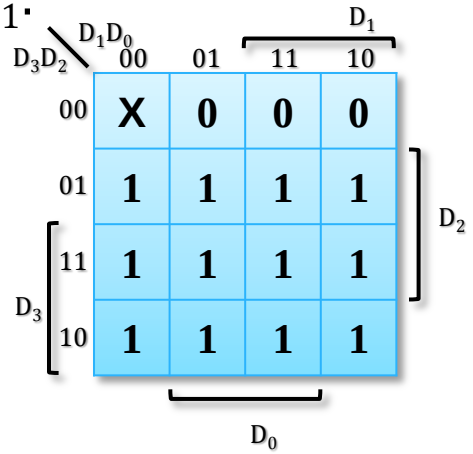
- **The priority encoder outputs the binary value corresponding to the most significant input equal to 1**

- **It also solves the ambiguity between all inputs equal 0 and $D_0 = 1$ by adding a *Valid* output**

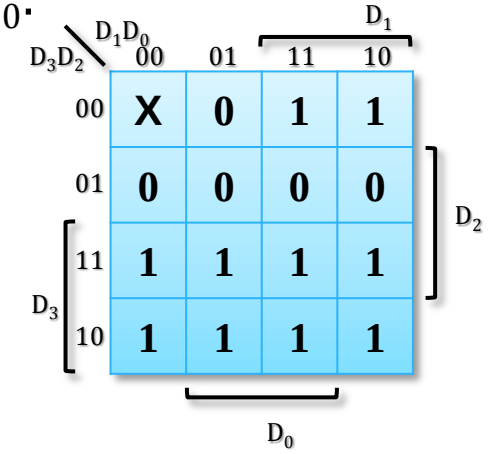
Four-Input Priority Encoder

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

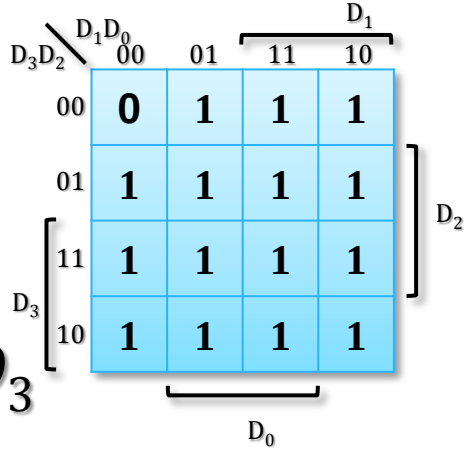
A_1:



A_0:



V:



$$A_0 = D_3 + D_1\bar{D}_2$$

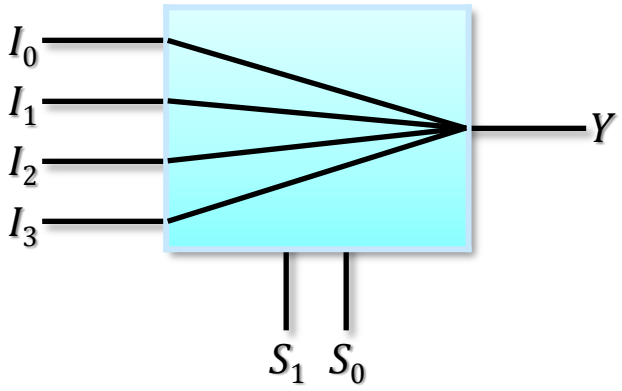
$$A_1 = D_2 + D_3$$

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

Multiplexers (MUX)

- A *Multiplexer* selects information from one of many input lines and directs it to an output line
- A typical multiplexer has *n selection inputs* (S_{n-1}, …, S_0) and 2^n *information inputs* (I_{2^n-1}, …, I_0)

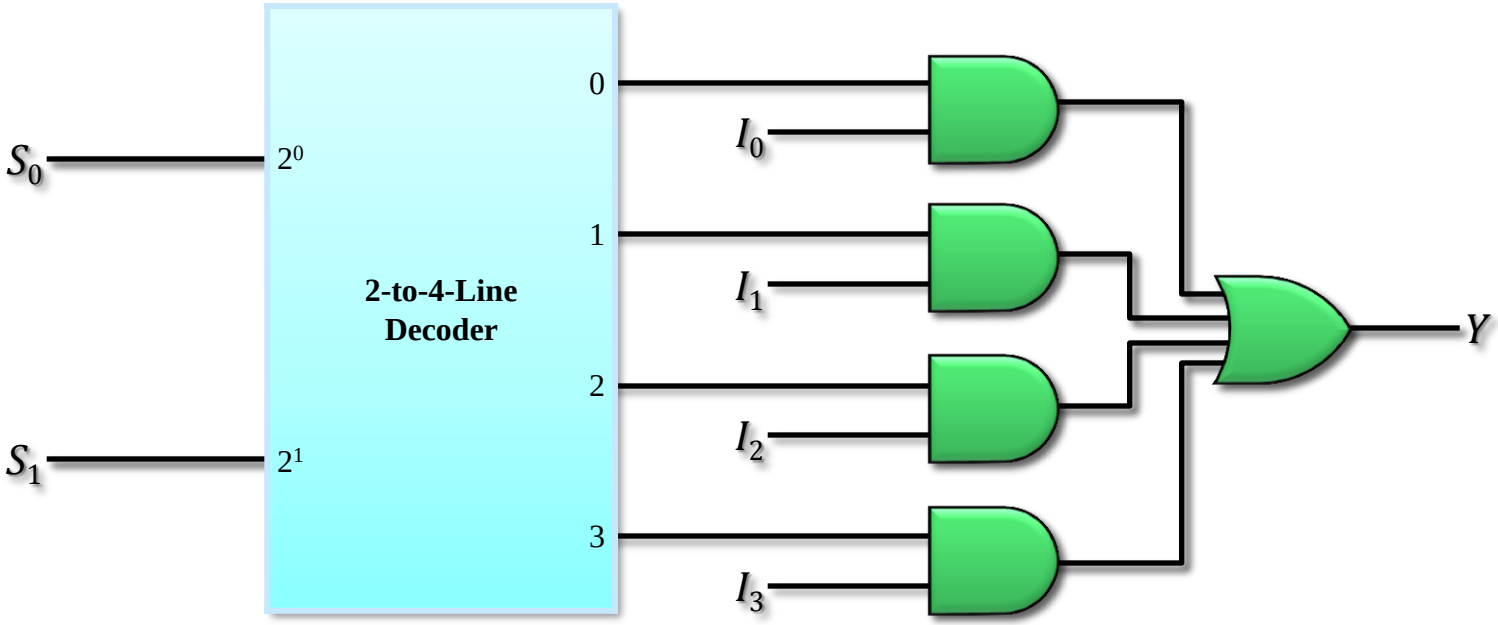
- Example – 4-to-1-line multiplexer:



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

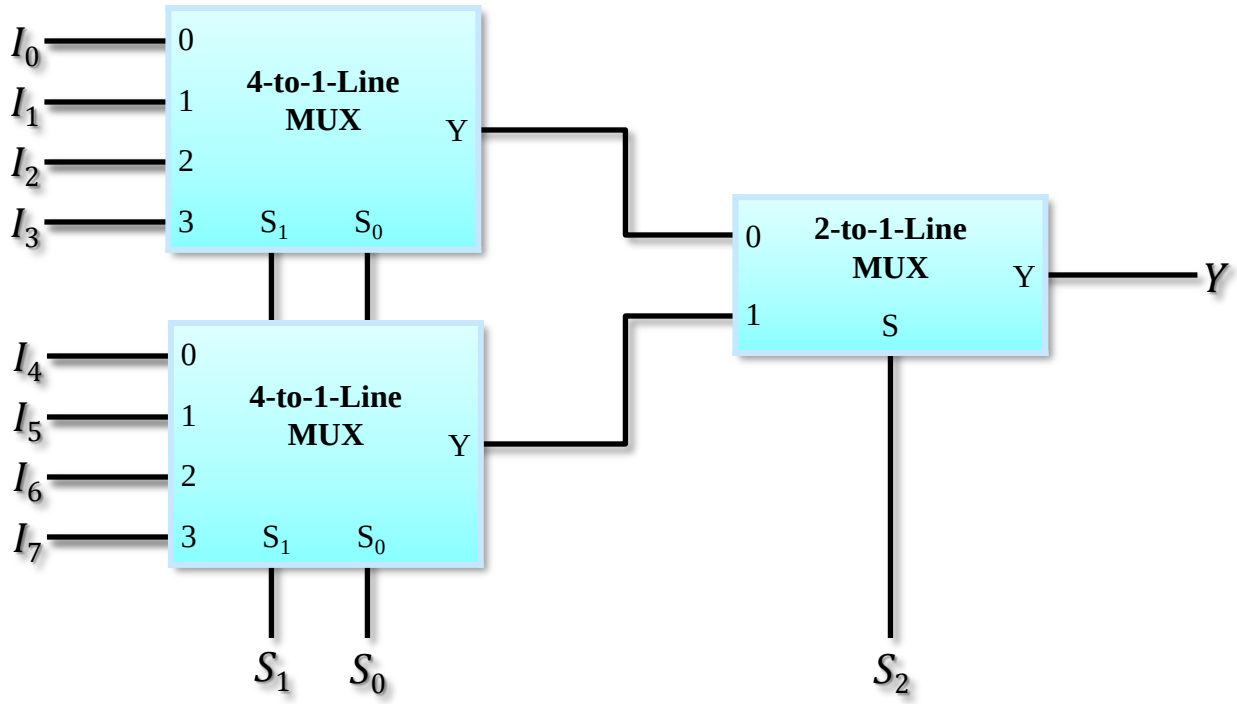
4-to-1-Line MUX Implementation

- **A multiplexer can be implemented using a decoder, 2-input AND gates and multiple-input OR gate:**



Cascading Multiplexers

- **Large multiplexers can also be implemented by cascading smaller ones**
- **Example – 8-to-1-line multiplexer:**



Boolean function implementation

Example: $F(X, Y, Z) = \sum m(1, 2, 6, 7)$

Boolean function implementation

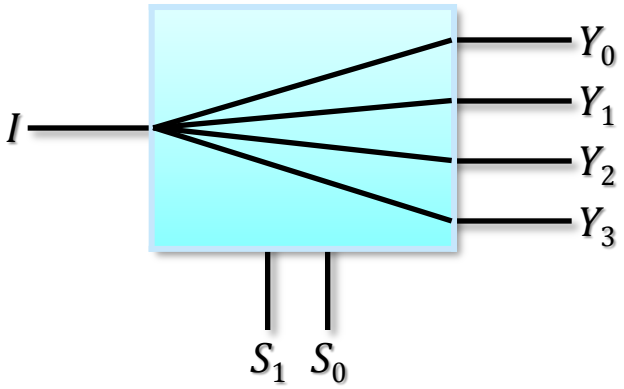
Example:

$F(A,B,C,D) = \sum m(1, 3, 4, 11,12,13,14,15)$

Demultiplexers (DeMUX)

- A *Demultiplexer* directs the information from a single input line to one of 2^n output lines and is controlled by *n* control lines

- Example – 1-to-4-line demultiplexer:



S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	*I*	0	0	0
0	1	0	*I*	0	0
1	0	0	0	*I*	0
1	1	0	0	0	*I*

Demultiplexers and Decoders

- **Demultiplexers are really the same as decoders with enable**

- **Just swap the signal names:**

$$EN \rightarrow I,\ A \rightarrow S \text{ and } D \rightarrow Y$$