



Digital Logic Design

Chapter 3 – Combinational Logic Design

Commonly Logic Function Design

浙江大学计算机学院 王总辉

zhwang@zju.edu.cn

<http://course.zju.edu.cn>

2023年10月



Overview

- **Part 2 – Commonly Function Design**

- Functions and functional blocks
- Rudimentary logic functions
 - Decoding using Decoders
 - Encoding using Encoders
 - Selecting using Multiplexers
- Implementing Combinational Functions with
 - Decoders and OR-Gate
 - Multiplexers and inverter



Functions and Functional Blocks

- **The functions considered are those found to be very useful in design**
 - Corresponding to each of the functions is a combinational circuit implementation called a *functional block*.
- **In the past, functional blocks were packaged as:**
 - small-scale-integrated circuits (SSI)
 - medium-scale integrated circuits (MSI)
 - and large-scale-integrated circuits (LSI).
- **Today, they are often simply implemented within:**
 - a very-large-scale-integrated (VLSI) circuit.

Rudimentary Logic Functions

● Constant(Value-Fixing)、Transferring、Inverting

- Functions of a single variable x
- Can be used on the inputs to functional blocks
- to implement other than the block's intended function

Functions of one variable

| X | $F = 0$ | $F = X$ | $F = \bar{X}$ | $F = 1$ |
|-----|---------|---------|---------------|---------|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |

1 ————— $F = 1$

0 ————— $F = 0$

(a)

V_{CC} or V_{DD}


————— $F = 1$

————— $F = 0$

(b)

X ————— $F = X$

(c)

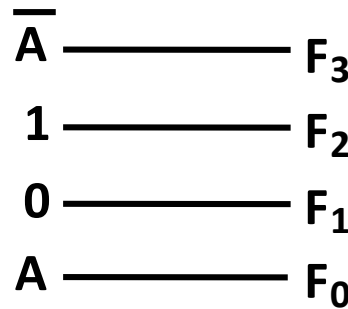
X —  — $F = \bar{X}$

(d)

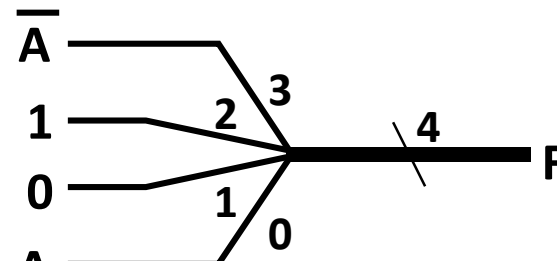
Multiple-bit Rudimentary Functions



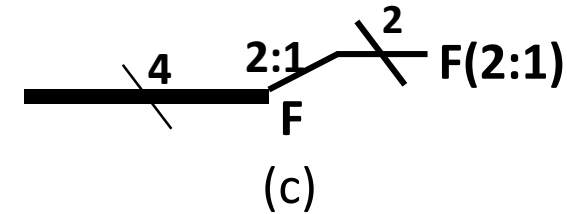
- Multi-bit Examples:



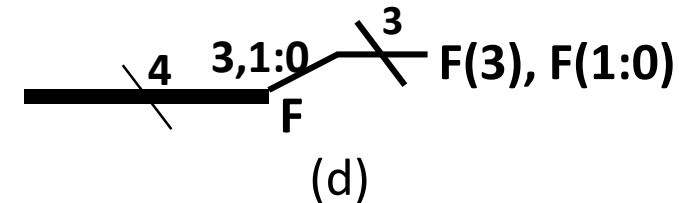
(a)



(b)



(c)



(d)

- A wide line is used to represent a **BUS** which is a vector signal
- In (b) of the example, $F = (F_3, F_2, F_1, F_0)$ is a bus.
- The bus can be split into individual bits as shown in (b)
- Sets of bits** can be split from the bus as shown in (c) for bits 2 and 1 of F .
- The sets of bits need not be continuous as shown in (d) for bits 3, 1, and 0 of F .

Enabling Function

- **Enabling**

- permits an input signal to pass through to an output

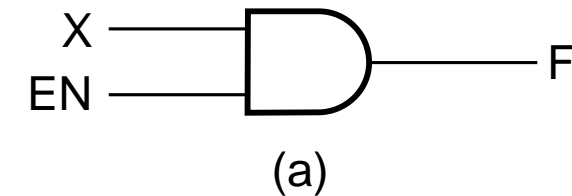
- **Disabling**

- Disabling blocks an input signal from passing through to an output, replacing it with **a fixed value**

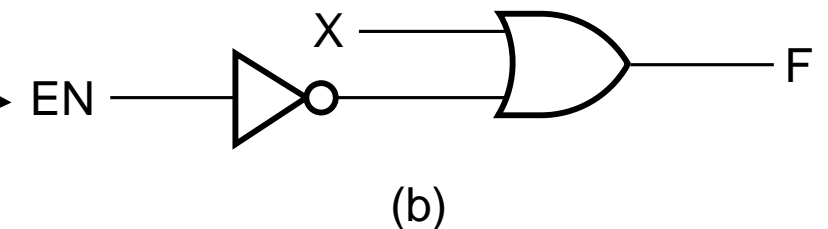
- **The value on the output when it is disable can be**

- **Hi-Z** (as for three-state buffers and transmission gates)
- 0 or 1

- **When disabled, 0 output**



- **When disabled, 1 output**

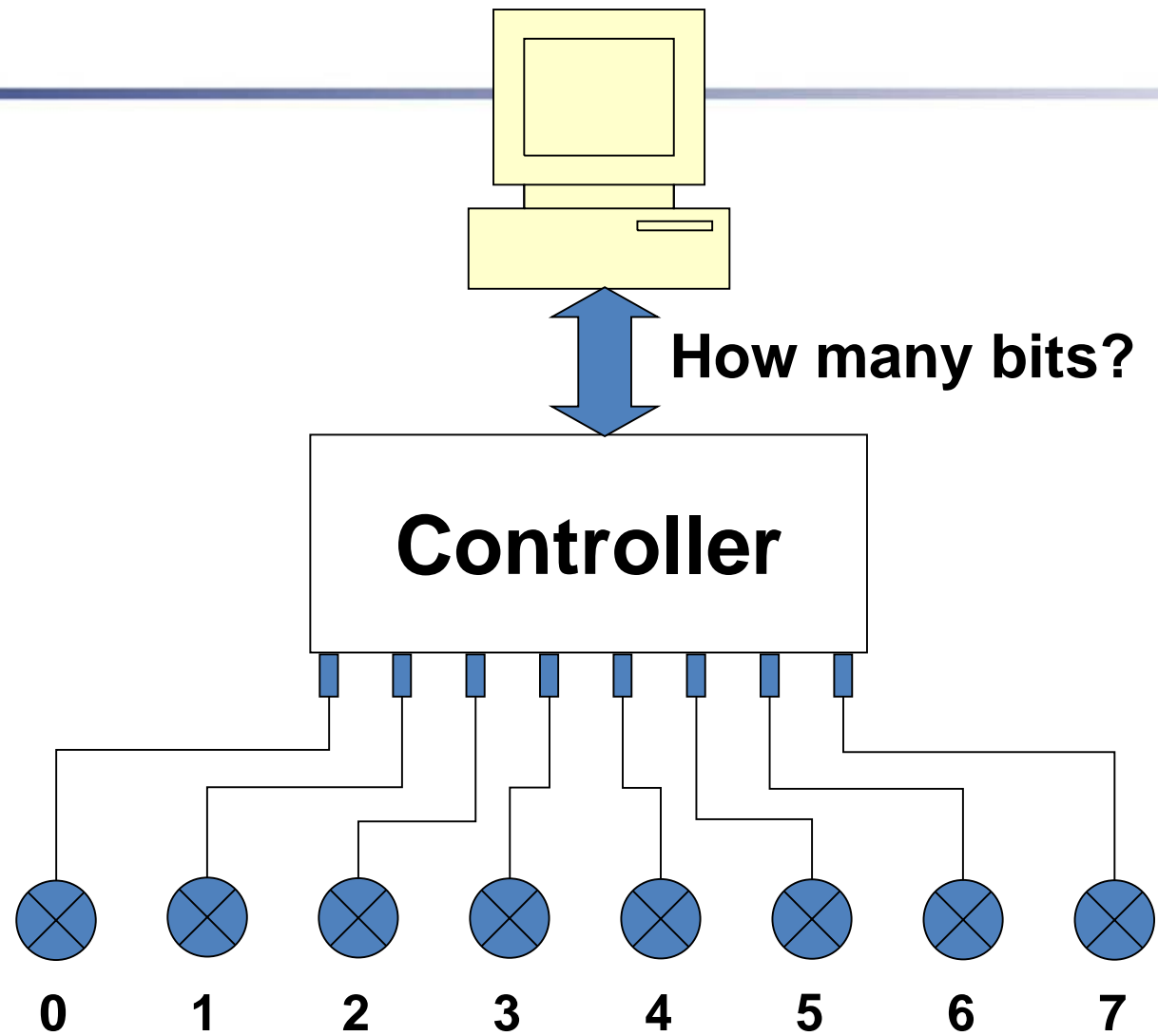




Decoder

- **Decoder:**
 - Translate an coding into another coding
- **Function:**
 - input variables is converted and given a particular output signal
- **Methods:**
 - Combinational circuit. The n-bit input code is converted into the output of the m-bit coded
 - Each group of valid input coded to generate a unique set of output. The general
$$n \leq m \leq 2^n$$

variety {
Variable decoder
Code system transform decoder
Display decoder



| A | B | C | 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 |

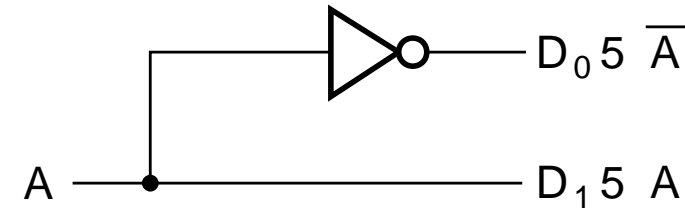
Generic Decoder

- **Decoding** - the conversion of an n -bit input code to an m -bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code
- **Circuits that perform decoding are called *decoders***
- **Here, functional blocks for decoding are**
 - called n -to- m line decoders, where $m \leq 2^n$, and
 - generate 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 |

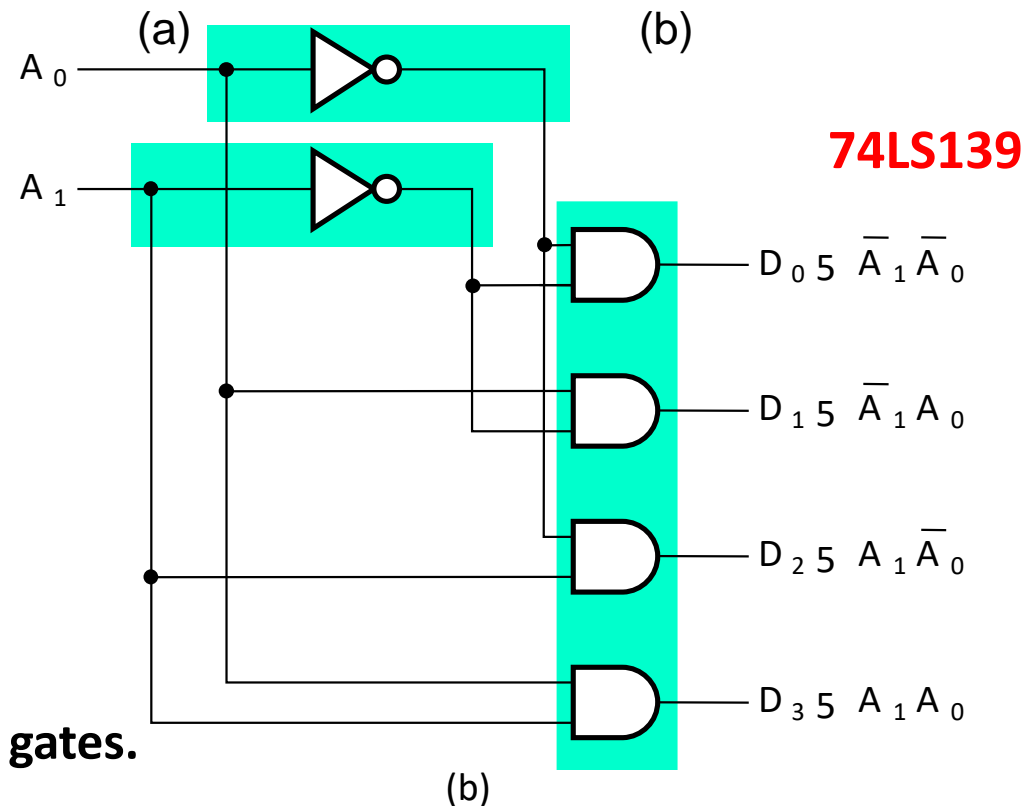


● 2-to-4-Line Decoder

| 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 |

(a)

- Note that the 2-4-line made up of 2 1-to-2-line decoders and 4 AND gates.



Decoder Expansion

- **General procedure given in book for any decoder with n inputs and 2^n outputs.**
- **This procedure builds a decoder backward from the outputs.**
 1. The output AND gates are driven by two decoders with their numbers of inputs either equal or differing by 1.
 2. These decoders are then designed using the same procedure until 2-to-1-line decoders are reached.
- The procedure can be modified to apply to decoders with the number of outputs $\neq 2^n$

Decoder Expansion - Example 1



● 3-to-8-line decoder

74LS138

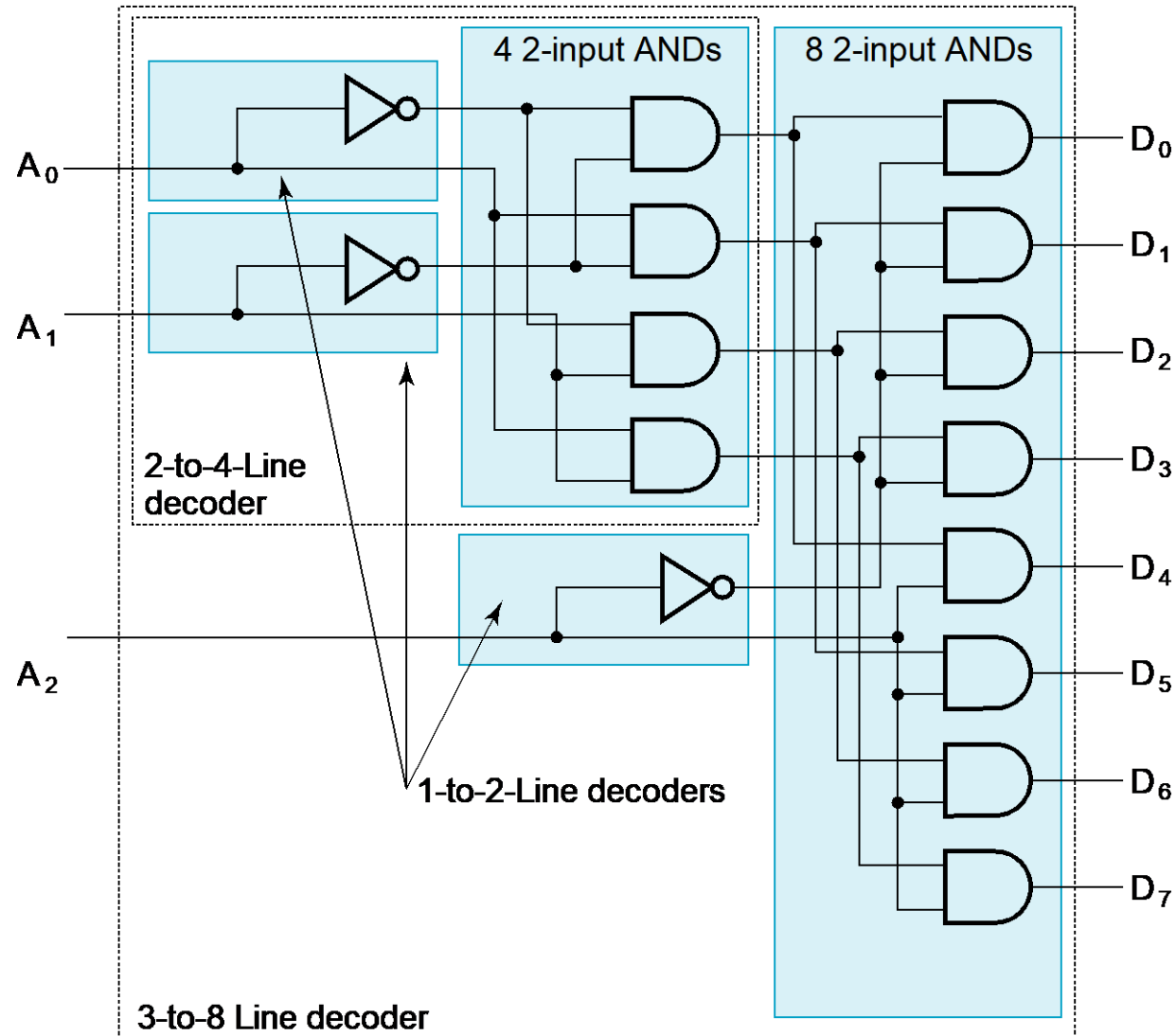
- ❑ Number of output ANDs = 8
- ❑ Number of inputs to decoders driving output ANDs = 3
- ❑ Closest possible split to equal
 - 2-to-4-line decoder
 - 1-to-2-line decoder
- ❑ 2-to-4-line decoder
 - Number of output ANDs = 4
 - Number of inputs to decoders driving output ANDs = 2
 - Closest possible split to equal
 - ❑ Two 1-to-2-line decoders

| input | | | output | | | | | | | |
|-------|----|----|--------|----|----|----|----|----|----|----|
| A0 | A1 | A2 | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Decoder Expansion - Example 1



- Result



Decoder Expansion - Example 2



- **7-to-128-line decoder**

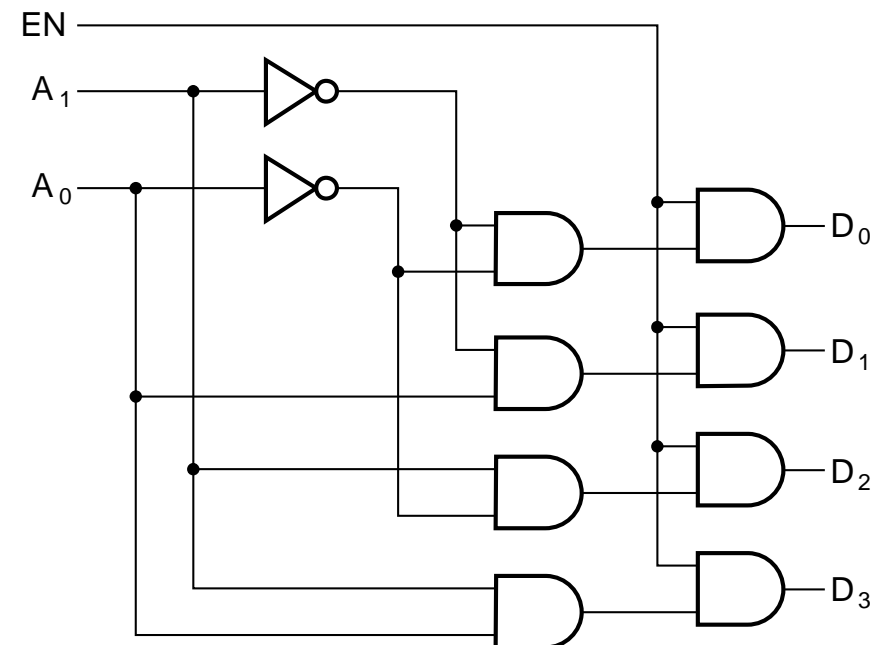
- Number of output ANDs = 128
- Number of inputs to decoders driving output ANDs = 7
- Closest possible split to equal
 - 4-to-16-line decoder
 - 3-to-8-line decoder
- 4-to-16-line decoder
 - Number of output ANDs = 16
 - Number of inputs to decoders driving output ANDs = 2
 - Closest possible split to equal
 - 2 2-to-4-line decoders
- Complete using known 3-8 and 2-to-4 line decoders

Decoder with Enable

- In general, attach m -enabling circuits to the outputs
- See truth table below for function
 - Note use of X's to denote both 0 and 1
 - Combination containing two X's represent four binary combinations
- Alternatively, can be viewed as distributing value of signal EN to 1 of 4 outputs
- In this case, called a *demultiplexer*

| 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 |

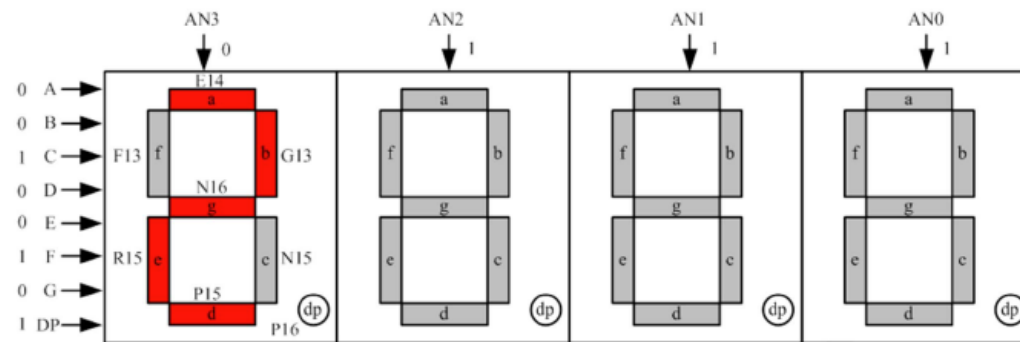
(a)



(b)

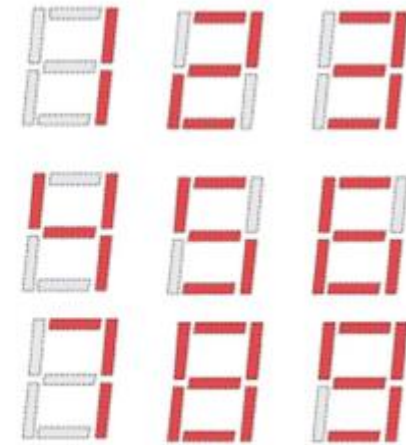
Display decoder

- Common electrodes:
 - Common cathode, common anode



Common anode

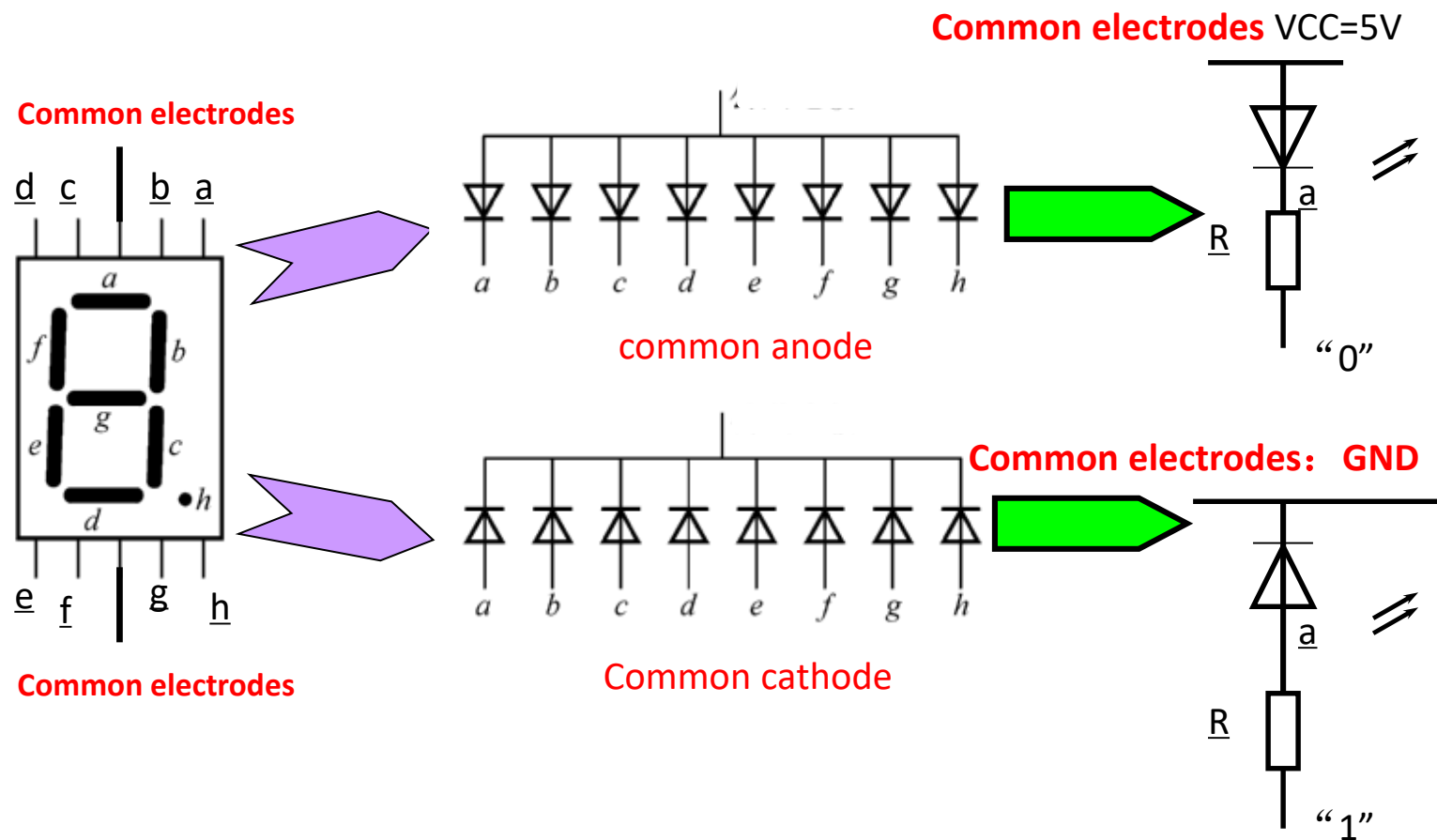
| X | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| A | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| B | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| D | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| E | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| F | 0 | 1 | 1 | 1 | 0 | 0 | 0 |



0 = on

1 = off

LED light emitting diode display

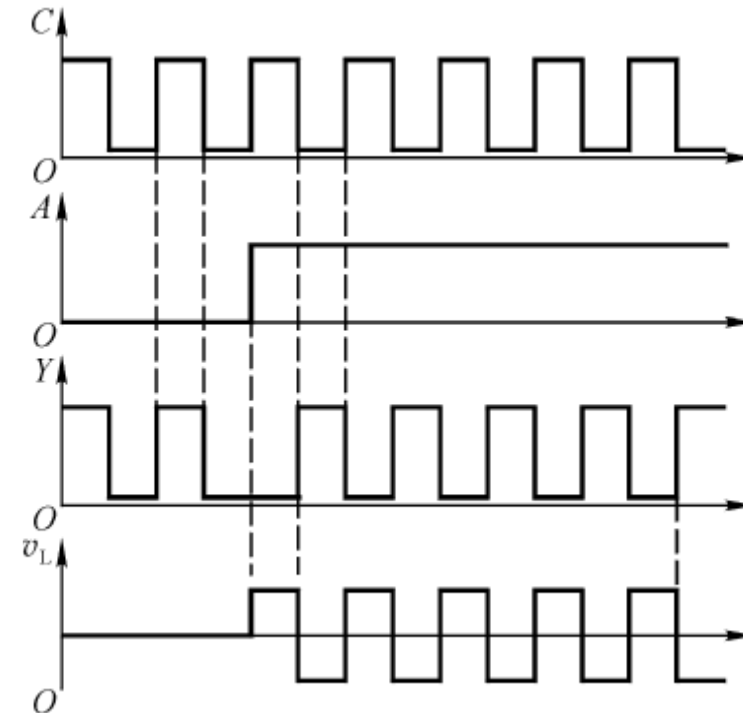
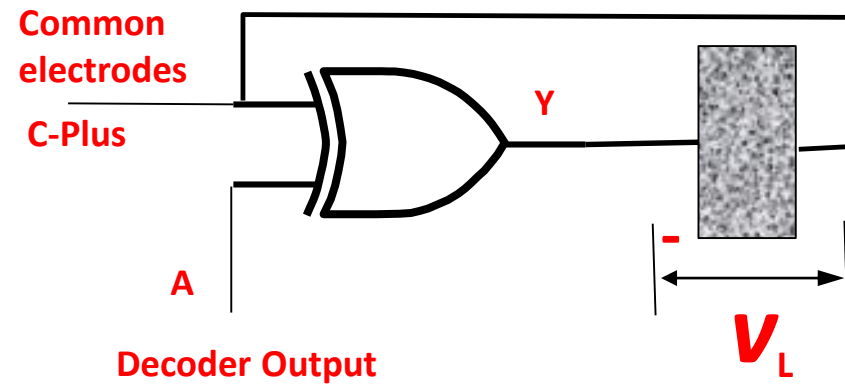


BCD 7- segment decoder

Common cathode

| BCD Input | | | | Seven-Segment Decoder | | | | | | |
|------------------|---|---|---|-----------------------|---|---|---|---|---|---|
| A | B | C | D | a | b | c | d | e | f | g |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| All other inputs | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

LCD driving principle



Encoding

- **Encoding**

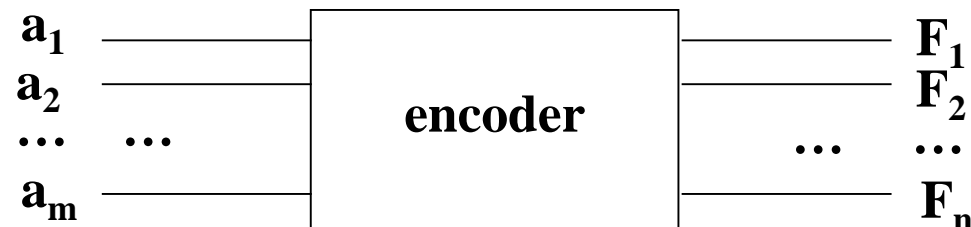
- ❑ the opposite of decoding - the conversion of an m -bit input code to a n -bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code

- **Circuits that perform encoding are called *encoders***

- **An encoder has 2^n (or fewer) input lines and n output lines which generate the binary code corresponding to the input values**

- **Variety**

- ❑ Instruction encoder
- ❑ Priority Encoder
- ❑ decimal-to-BCD



$$n \leq m \leq 2^n$$

A decimal-to-BCD encoder

- an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears
 - Inputs: 10 bits corresponding to decimal digits 0 through 9, (D_0, \dots, D_9)
 - Outputs: 4 bits with BCD codes
 - Function: If input bit D_i is a 1, then the output (A_3, A_2, A_1, A_0) is the BCD code for i
- The truth table could be formed, but alternatively, the equations for each of the four outputs can be obtained directly.

| decimal | A_3 | A_2 | A_1 | A_0 |
|---------|-------|-------|-------|-------|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |

Encoder Example (continued)



- Input D_i is a term in equation A_j if bit A_j is 1 in the binary value for i .

| Input | | | | | | | | | | Output | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|--------|-------|-------|-------|
| D_9 | D_8 | D_7 | D_6 | D_5 | D_4 | D_3 | D_2 | D_1 | D_0 | A_3 | A_2 | A_1 | A_0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

- Equations:

$$A_3 = D_8 + D_9$$

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

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

$$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$$

- $F_1 = D_6 + D_7$ can be extracted from A_2 and A_1 Is there any cost saving?

Priority Encoder

- If more than one input value is 1, then the encoder just designed does not work.
- One encoder that can accept all possible combinations of input values and produce a meaningful result is **a priority encoder**.
- Among the 1s that appear, it selects the **most significant** input position (or the **least significant** input position) containing a 1 and responds with the corresponding binary code for that position.

Priority Encoder Example

- Priority encoder with 5 inputs (D_4, D_3, D_2, D_1, D_0) - **highest priority to most significant** 1 present - Code outputs A2, A1, A0 and V where V indicates at least one 1 present.

| No. of Min-terms/Row | Inputs | | | | | Outputs | | | |
|----------------------|--------|----|----|----|----|---------|----|----|---|
| | D4 | D3 | D2 | D1 | D0 | A2 | A1 | A0 | V |
| 0 | 0 | 0 | 0 | 0 | 0 | X | X | X | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 2-3 | 0 | 0 | 0 | 1 | X | 0 | 0 | 1 | 1 |
| 4-7 | 0 | 0 | 1 | X | X | 0 | 1 | 0 | 1 |
| 8-15 | 0 | 1 | X | X | X | 0 | 1 | 1 | 1 |
| 16-31 | 1 | X | X | X | X | 1 | 0 | 0 | 1 |

- Xs in input part of table represent 0 or 1; thus table entries correspond to product terms instead of minterms. The column on the left shows that all 32 minterms are present in the product terms in the table

Priority Encoder Example (continued)



- **Could use a K-map to get equations**
 - but can be read directly from table and manually optimized if careful:

$$\mathbf{A}_2 = \mathbf{D}_4$$

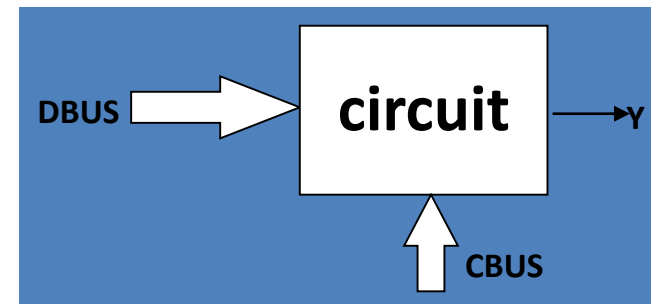
$$\mathbf{A}_1 = \overline{\mathbf{D}_4}\mathbf{D}_3 + \overline{\mathbf{D}_4}\overline{\mathbf{D}_3}\mathbf{D}_2 = \overline{\mathbf{D}_4}\mathbf{F}_1, \quad \mathbf{F}_1 = (\mathbf{D}_3 + \mathbf{D}_2)$$

$$\mathbf{A}_0 = \overline{\mathbf{D}_4}\mathbf{D}_3 + \overline{\mathbf{D}_4}\overline{\mathbf{D}_3}\overline{\mathbf{D}_2}\mathbf{D}_1 = \overline{\mathbf{D}_4}(\mathbf{D}_3 + \overline{\mathbf{D}_2}\mathbf{D}_1)$$

$$\mathbf{V} = \mathbf{D}_4 + \mathbf{F}_1 + \mathbf{D}_1 + \mathbf{D}_0$$

Selecting

- Selecting of data or information is a critical function in digital systems and computers
- Circuits that perform selecting have:
 - ❑ A set of information inputs from which the selection is made
 - ❑ A single output
 - ❑ A set of control lines for making the selection
- Logic circuits that perform selecting are called ***multiplexers***
- Selecting can also be done by three-state logic or transmission gates



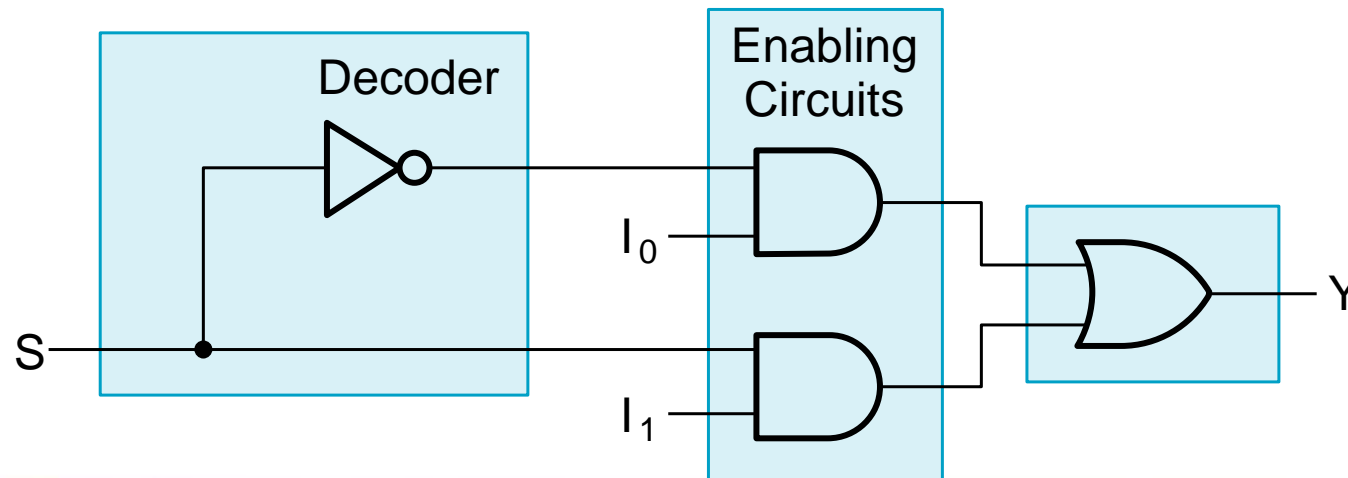
Multiplexers

- A multiplexer selects information from an input line and directs the information to an output line
- A typical multiplexer has
 - n control inputs (S_{n-1}, \dots, S_0) called *selection inputs*,
 - 2^n information inputs (I_{2^n-1}, \dots, I_0),
 - and one output Y
- A multiplexer can be designed to have m information inputs with $m < 2^n$ as well as n selection inputs

2-to-1-Line Multiplexer

- Since $2 = 2^1$, $n = 1$
- The single selection variable S has two values:
 - $S = 0$ selects input I_0
 - $S = 1$ selects input I_1
- The equation: $Y = \bar{S}I_0 + SI_1$
- The circuit:

| S | I_0 | I_1 | Y |
|---|-------|-------|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |



2-to-1-Line Multiplexer (continued)

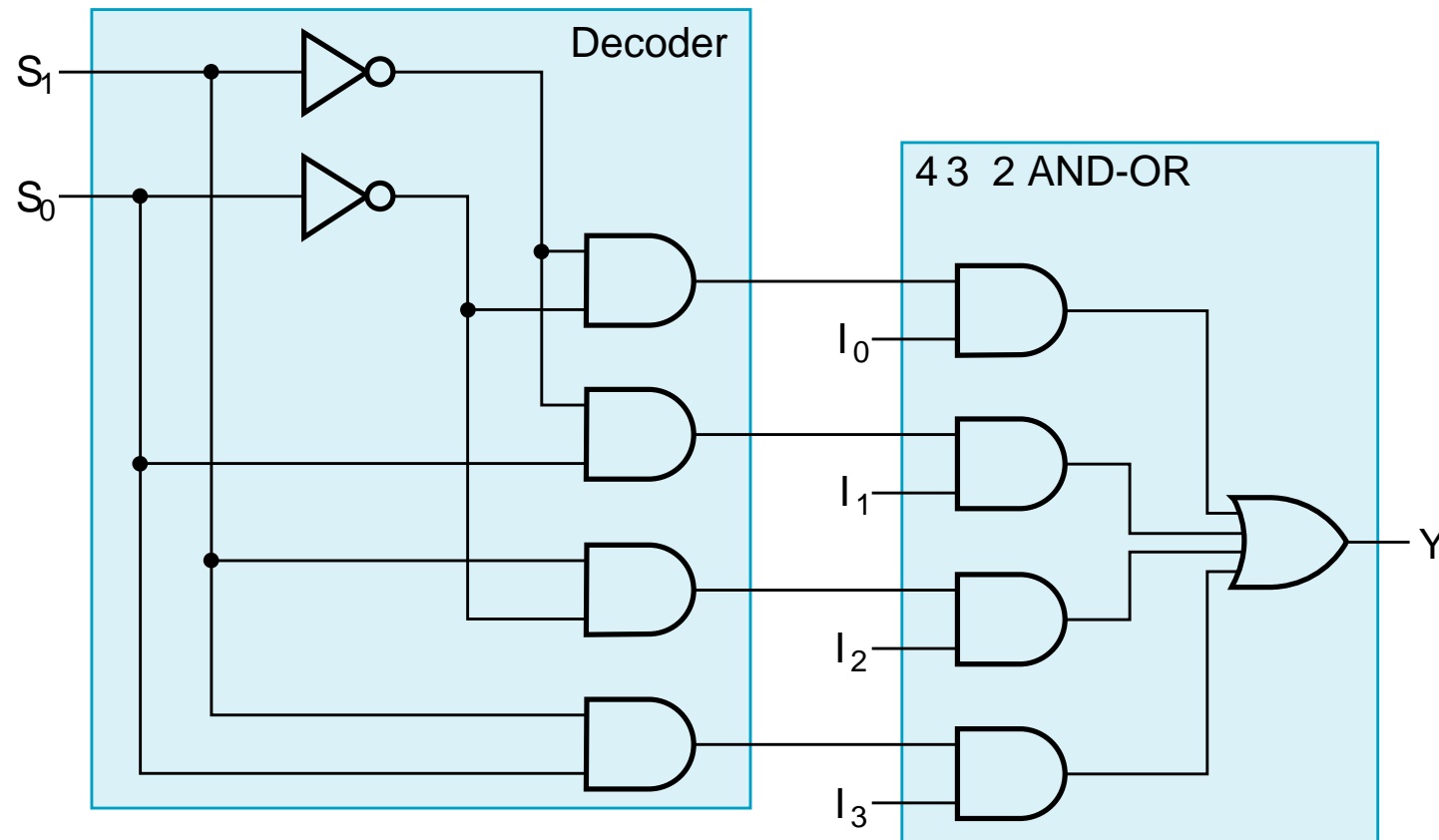


- **Note the regions of the multiplexer circuit shown:**
 - ❑ 1-to-2-line Decoder
 - ❑ 2 Enabling circuits
 - ❑ 2-input OR gate
- **To obtain a basis for multiplexer expansion, we combine the **Enabling** circuits and **OR gate into a 2×2 - AND-OR** circuit:**
 - ❑ 1-to-2-line decoder
 - ❑ 2×2 AND-OR
- **In general, for an 2^n -to-1-line multiplexer:**
 - ❑ n -to- 2^n -line decoder
 - ❑ $2^n \times 2$ AND-OR

Example: 4-to-1-line Multiplexer

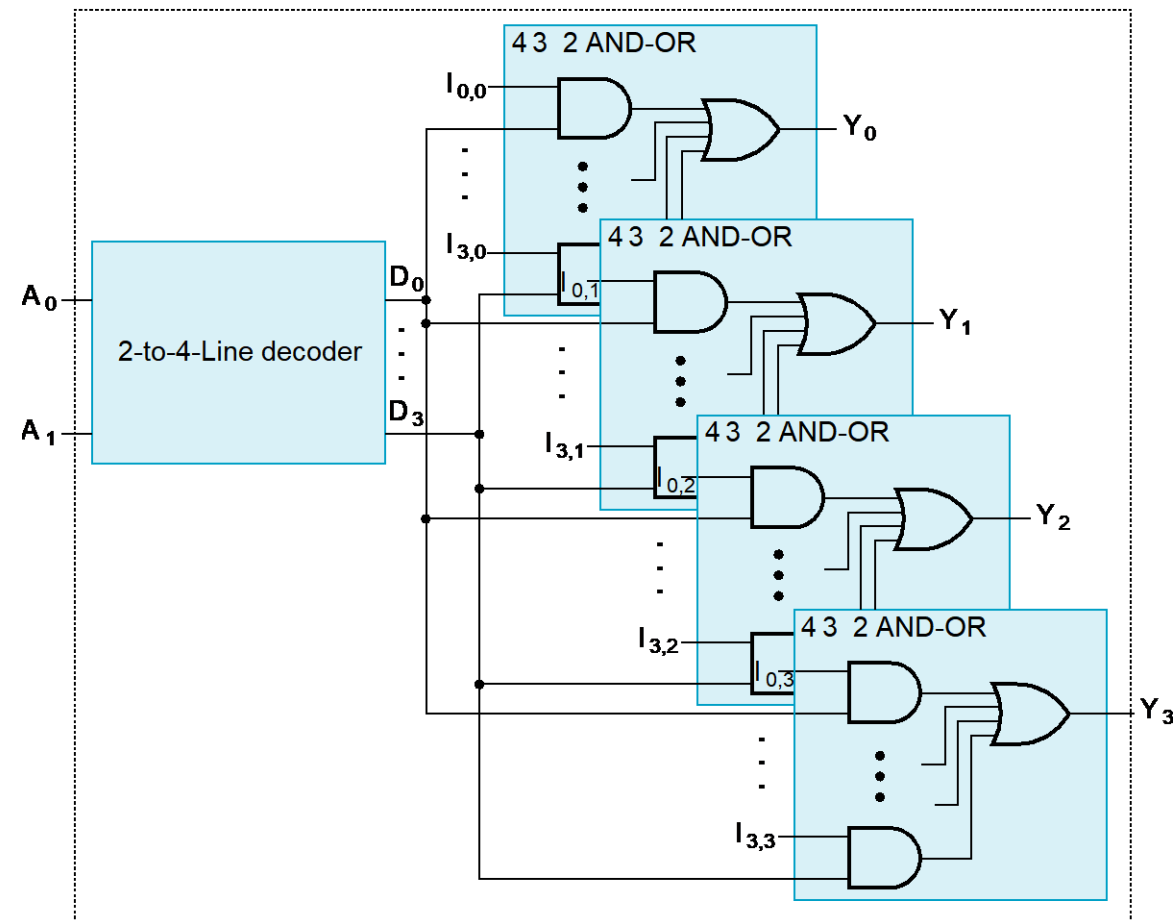


- 2-to- 2^2 -line decoder
- $2^2 \times 2$ AND-OR



Multiplexer Width Expansion

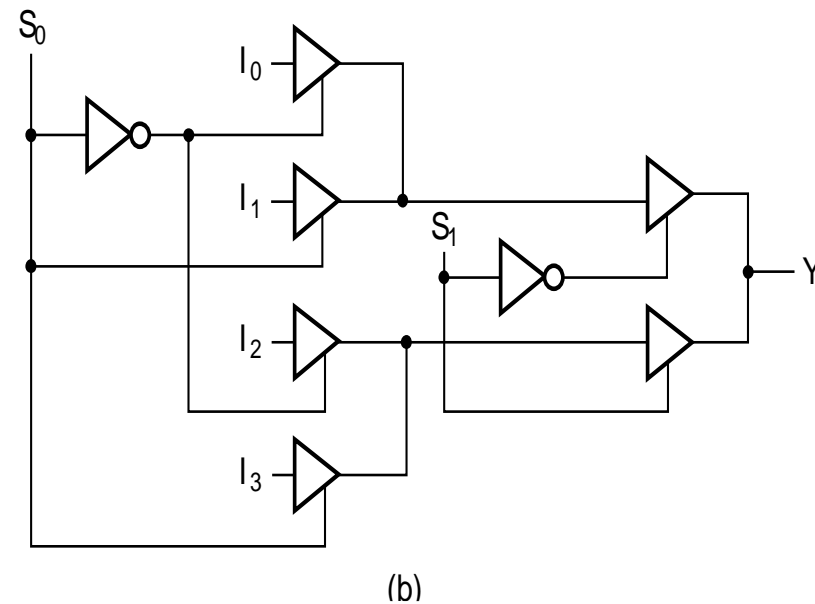
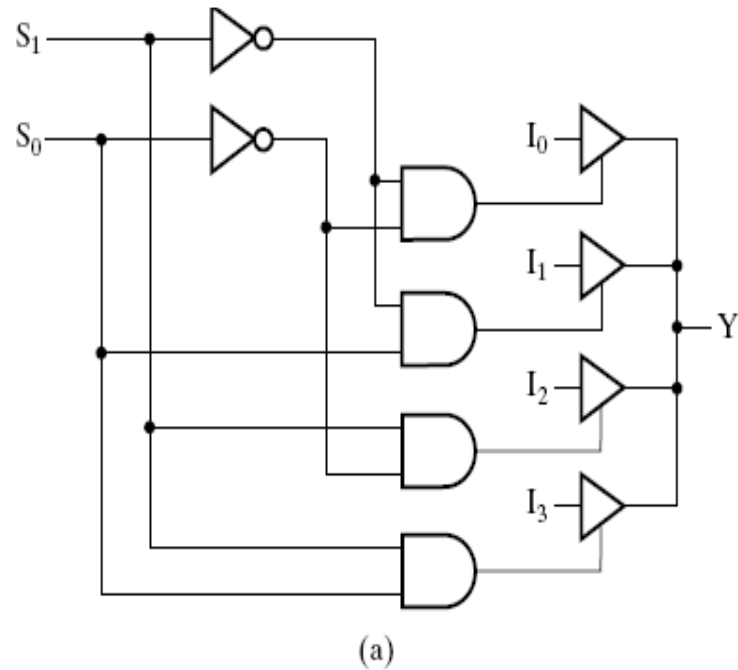
- Select “vectors of bits” instead of “bits”
- Use multiple copies of $2^n \times 2$ AND-OR in parallel
- Example:
4-to-1-line
quad multiplexer



Other Selection Implementations



- **Three-state logic in place of AND-OR**



- **Gate input cost = 14 compared to 22 (or 18 for (a)) for gate implementation**

● Implementing Combinational Functions

- Fundamental gate circuit
- Decoders and OR-Gate
- Multiplexers and inverter
- ROMs
- PLAs
- PALs
- Lookup Tables



Combinational Logic Implementation - Decoder and OR Gates

- **Implement m functions of n variables with:**
 - Sum-of-minterms expressions
 - One n -to- 2^n -line decoder
 - m OR gates, one for each output
- **Approach 1:**
 - Find the **truth table** for the functions
 - Make a connection to the corresponding OR from the corresponding decoder output wherever a 1 appears in the truth table
- **Approach 2**
 - Find the **minterms** for each output function
 - OR the minterms together

Decoder and OR Gates Example-1



- Implement Binary adder for 1 bit
- Find the **truth table** for the functions

□ Truth table at right:

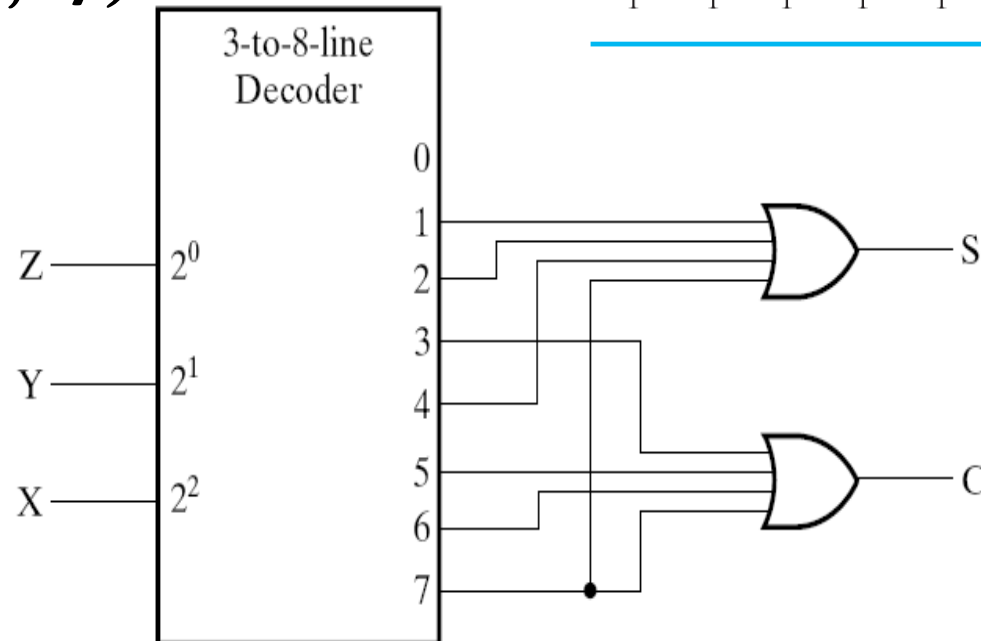
□ Minterms expressions :

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

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

| 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 |

Logic circuits:



Decoder and OR Gates Example-1



- Implement the following set of odd parity functions of (A_7, A_6, A_5, A_3)

$$P_1 = A_7 \oplus A_5 \oplus A_3$$

$$P_2 = A_7 \oplus A_6 \oplus A_3$$

$$P_4 = A_7 \oplus A_6 \oplus A_5$$

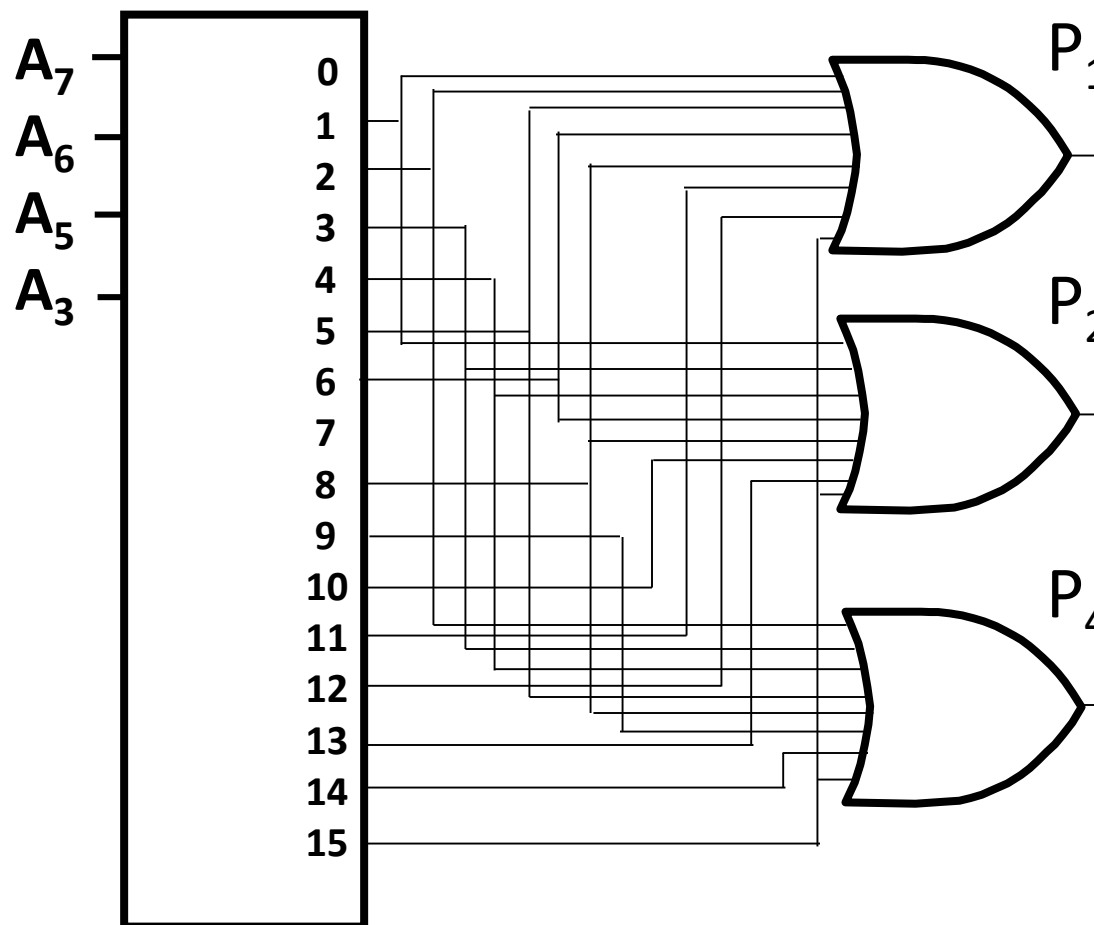
- Finding sum of minterms expressions

$$P_1 = S_m(1,2,5,6,8,11,12,15)$$

$$P_2 = S_m(1,3,4,6,8,10,13,15)$$

$$P_4 = S_m(2,3,4,5,8,9,14,15)$$

- Find circuit
- Is this a good idea?





Combinational Logic Implementation - Multiplexer Approach

- **Implement m functions of n variables with:**
 - Sum-of-minterms expressions
 - An m -wide 2^n -to-1-line multiplexer
- **Design:**
 - Find the truth table for the functions.
 - In the order they appear in the truth table:
 - Apply the function **input variables** to **the multiplexer inputs** S_{n-1}, \dots, S_0
 - Label the outputs of the multiplexer with the output variables
 - Value-fix the information inputs to the multiplexer using the values from the truth table (for don't cares, apply either 0 or 1)

Example: Gray to Binary Code



- Design a circuit to convert a 3-bit Gray code to a binary code
- The formulation gives the truth table on the right
- It is obvious from this table that $X = C$ and the Y and Z are more complex

| Gray A B C | Binary x y z |
|---------------|-----------------|
| 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 1 |
| 1 1 0 | 0 1 0 |
| 0 1 0 | 0 1 1 |
| 0 1 1 | 1 0 0 |
| 1 1 1 | 1 0 1 |
| 1 0 1 | 1 1 0 |
| 0 0 1 | 1 1 1 |

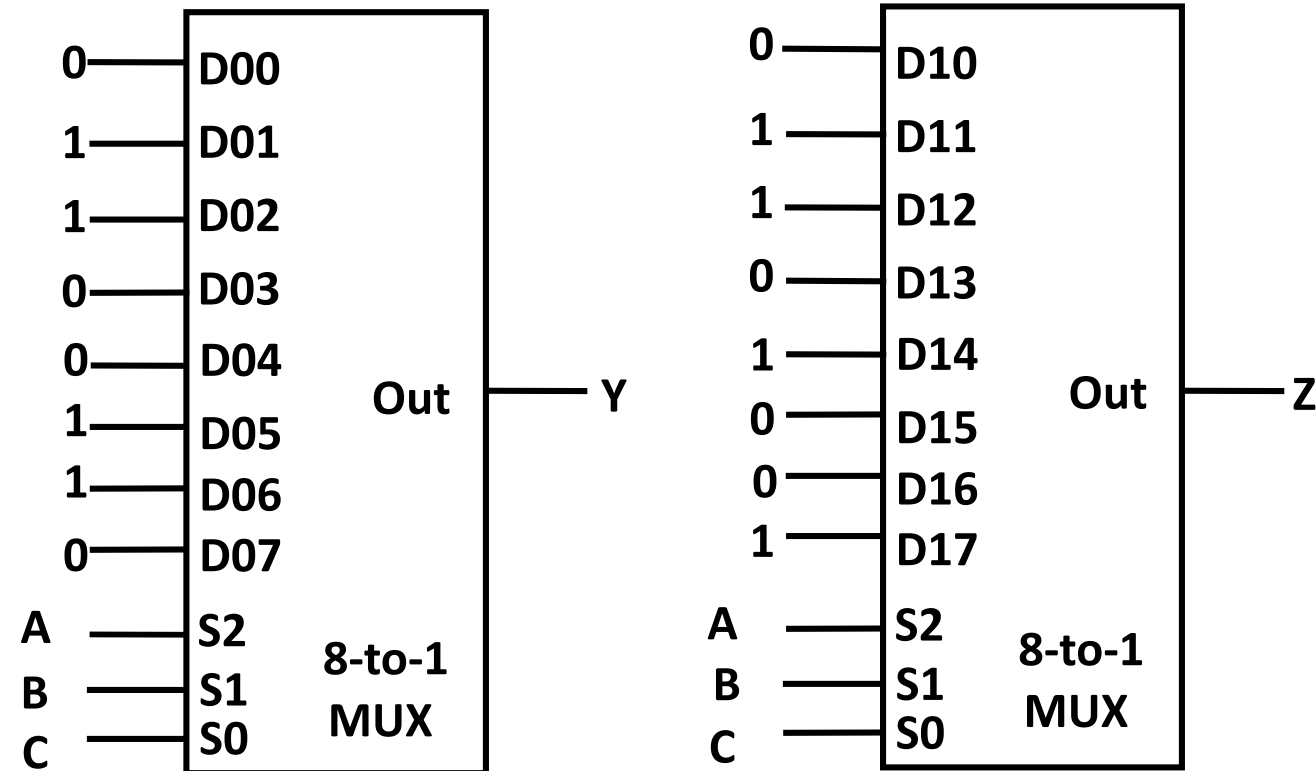
Gray to Binary (continued)

- Rearrange the table so that the input combinations are in counting order
- Functions y and z can be implemented using a dual 8-to-1-line multiplexer by:

| Gray A B C | Binary x y z |
|---------------|-----------------|
| 0 0 0 | 0 0 0 |
| 0 0 1 | 1 1 1 |
| 0 1 0 | 0 1 1 |
| 0 1 1 | 1 0 0 |
| 1 0 0 | 0 0 1 |
| 1 0 1 | 1 1 0 |
| 1 1 0 | 0 1 0 |
| 1 1 1 | 1 0 1 |

- connecting A, B, and C to the multiplexer select inputs
- placing y and z on the two multiplexer outputs
- connecting their respective truth table values to the inputs

Gray to Binary (continued)

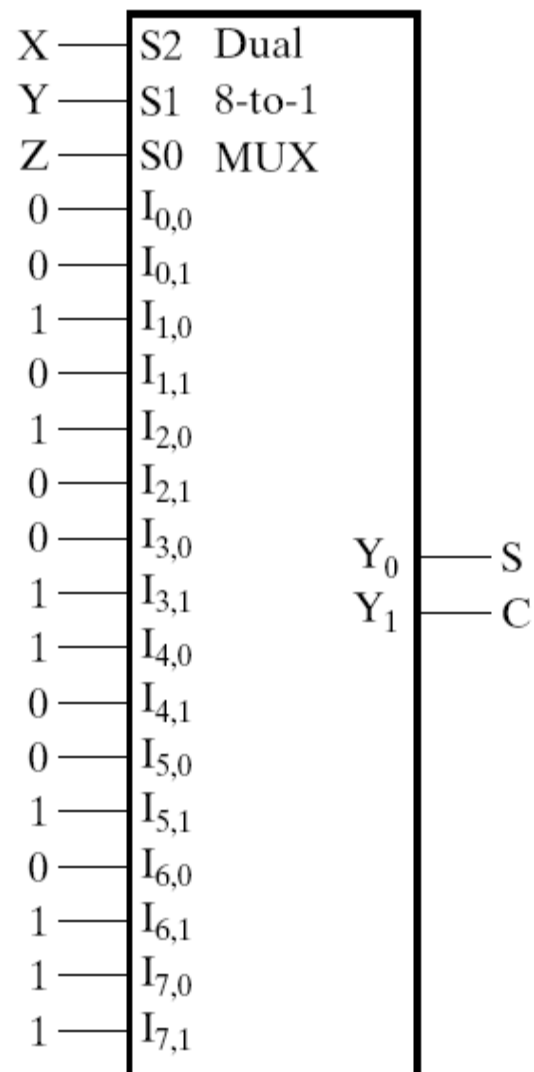


- Note that the multiplexer with fixed inputs is identical to a ROM with 3-bit addresses and 2-bit data!

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 |





Combinational Logic Implementation - Multiplexer Approach 2

- Implement any m functions of $n + 1$ variables by using:
 - An m -wide 2^n -to-1-line multiplexer
 - A single inverter
- Design: **an additional variable**
 - Find the truth table for the functions.
 - Based on the values of the **first n variables**, separate the truth table rows into pairs
 - For each pair and output, **define** a rudimentary **function** of the final variable (0, 1, X , \bar{X}) **Additional equation 1**
 - Using the first n variables as the index, value-fix the information inputs to the multiplexer with the corresponding rudimentary functions
 - Use the inverter to generate the rudimentary function \bar{X}

Example: Gray to Binary Code



- Design a circuit to convert a 3-bit Gray code to a binary code
- The formulation gives the truth table on the right
- It is obvious from this table that $X = C$ and the Y and Z are more complex

| Gray A B C | Binary x y z |
|---------------|-----------------|
| 0 0 0 | 0 0 0 |
| 1 0 0 | 0 0 1 |
| 1 1 0 | 0 1 0 |
| 0 1 0 | 0 1 1 |
| 0 1 1 | 1 0 0 |
| 1 1 1 | 1 0 1 |
| 1 0 1 | 1 1 0 |
| 0 0 1 | 1 1 1 |

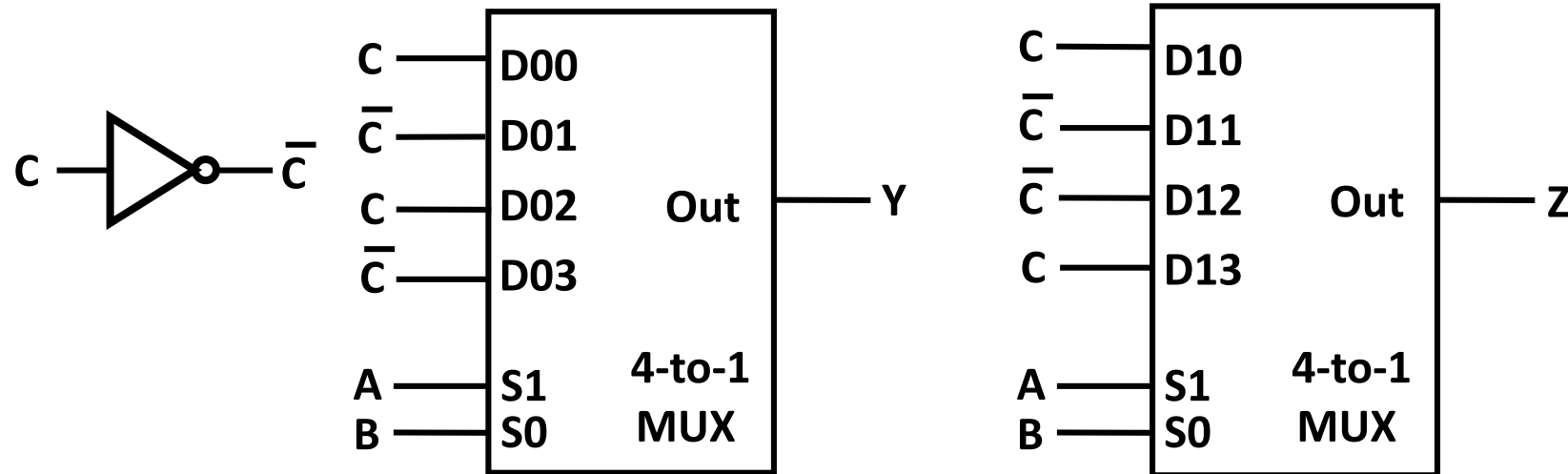
Gray to Binary (continued)

- Rearrange the table so that the input combinations are in counting order, pair rows, and find rudimentary functions

| Gray A B C | Binary x y z | Rudimentary Functions of C for y | Rudimentary Functions of C for z |
|---------------|-----------------|--|--|
| 0 0 0 | 0 0 0 | F = C | F = C |
| 0 0 1 | 1 1 1 | | |
| 0 1 0 | 0 1 1 | F = \bar{C} | F = \bar{C} |
| 0 1 1 | 1 0 0 | | |
| 1 0 0 | 0 0 1 | F = C | F = \bar{C} |
| 1 0 1 | 1 1 0 | | |
| 1 1 0 | 0 1 0 | F = \bar{C} | F = C |
| 1 1 1 | 1 0 1 | | |

Gray to Binary (continued)

- Assign the variables and functions to the multiplexer inputs:

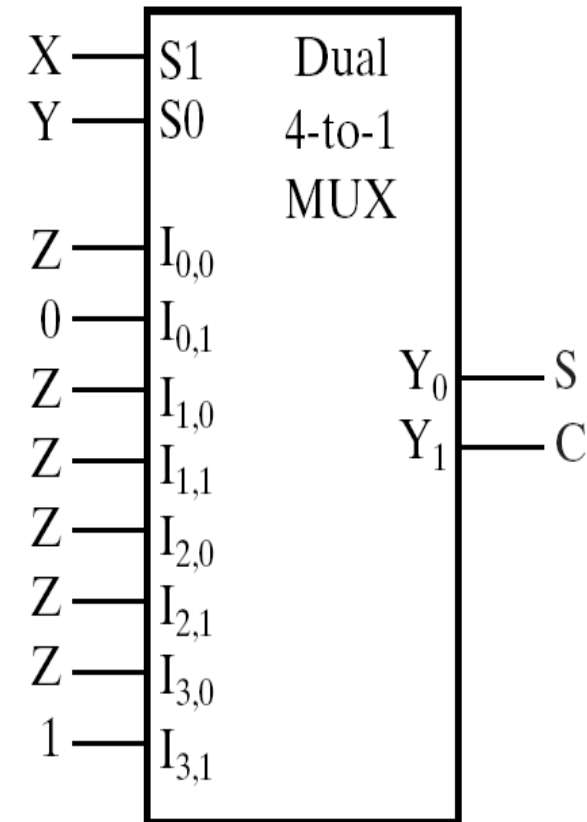


- Note that this approach (Approach 2) reduces the cost by almost half compared to Approach 1.
- This result is no longer ROM-like
- Extending, a function of more than n variables is decomposed into several **sub-functions** defined on a subset of the variables. The multiplexer then selects among these sub-functions.

Full adder

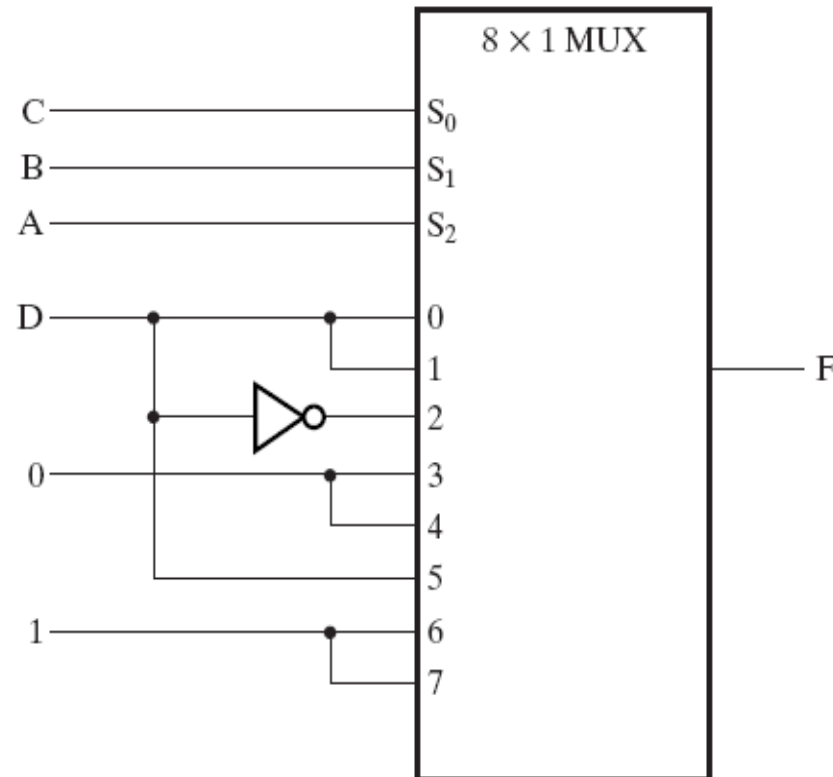
Truth table

| 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 |



| A | B | C | D | F | |
|---|---|---|---|---|---------------|
| 0 | 0 | 0 | 0 | 0 | $F = D$ |
| 0 | 0 | 0 | 1 | 1 | |
| 0 | 0 | 1 | 0 | 0 | $F = D$ |
| 0 | 0 | 1 | 1 | 1 | |
| 0 | 1 | 0 | 0 | 1 | $F = \bar{D}$ |
| 0 | 1 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 0 | $F = 0$ |
| 0 | 1 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 0 | $F = 0$ |
| 1 | 0 | 0 | 1 | 0 | |
| 1 | 0 | 1 | 0 | 0 | $F = D$ |
| 1 | 0 | 1 | 1 | 1 | |
| 1 | 1 | 0 | 0 | 1 | $F = 1$ |
| 1 | 1 | 0 | 1 | 1 | |
| 1 | 1 | 1 | 0 | 1 | $F = 1$ |
| 1 | 1 | 1 | 1 | 1 | |

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





Thank You !