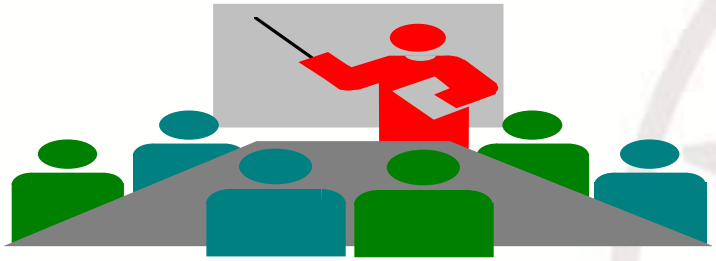


LOGIC and Computer Design Fundamentals

CHAPTER 3

Combinational Logic Design

Commonly Logic Function Design

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Overview

- **Part 1 – Design Procedure**
  - Steps
    - Specification
    - Formulation
    - Optimization
    - Technology Mapping
  - Beginning Hierarchical Design
  - Technology Mapping - AND, OR, and NOT to NAND or NOR
  - Verification
    - Manual
    - Simulation

Overview(continued)

- **Part 2 – Commonly Function Design**
  - Functions and functional blocks
  - Rudimentary logic functions
  - Commonly Logic Function Design
    - Decoding using Decoders
    - Encoding using Encoders
    - Selecting using Multiplexers
  - Implementing Combinational Functions with
    - Decoders and OR-Gate
    - Multiplexers and inverter
    - ROMs
    - PLAs
    - PALs
    - Lookup Tables

Course Outline

- Functions and functional blocks
- Rudimentary logic functions
- Commonly Logic Function Design
- Implementing Methods Of Combinational Functions
- Programming Technologies

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.

Course Outline

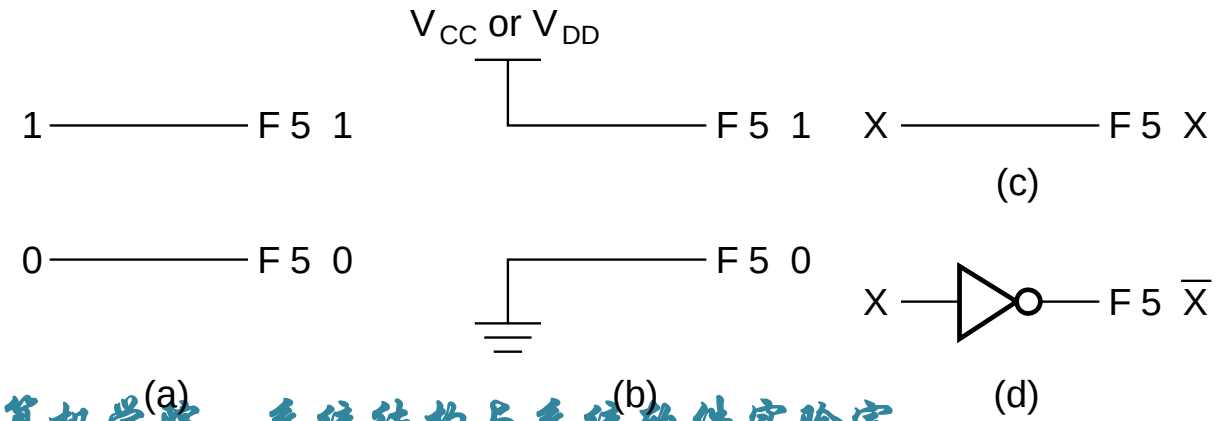
- Functions and functional blocks
- Rudimentary logic functions
- Commonly Logic Function Design
- Implementing Methods Of Combinational Functions
- Programming Technologies

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

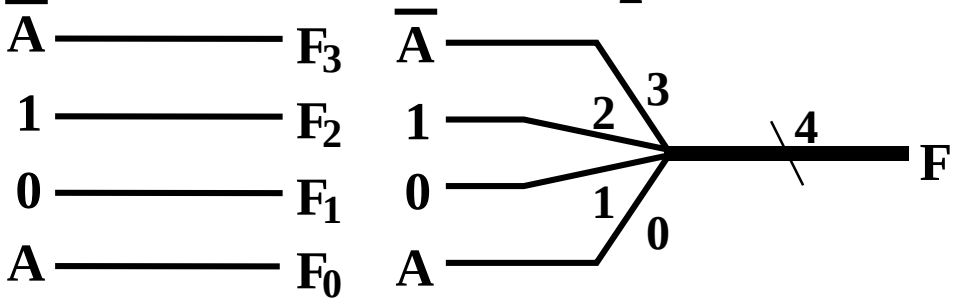
TABLE 4-1
Functions ofOne Variable

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



Multiple-bit Rudimentary Functions

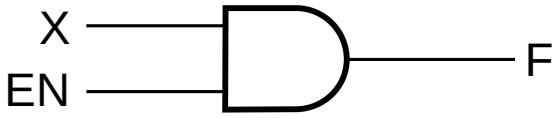
Multi-bit Examples:



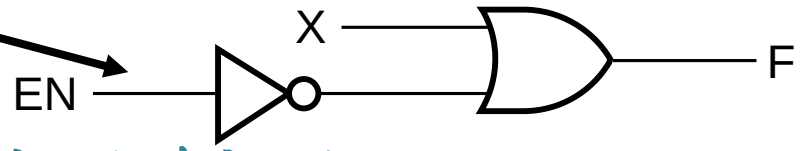
- A wide line is used to represent a *BUS* which is a vector signal
- In (b) of the example, F = (F_3, F_2, F_1, F_0) is a bus.
- The bus can be split into *individual bits* as shown in (b)
- Sets of bits can be split from the bus as shown in (c) for bits 2 and 1 of F.
- The sets of bits need not be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



(a)



(b)

Course Outline

- Functions and functional blocks
- Rudimentary logic functions
- **Commonly Logic Function Design**
- Implementing Methods Of Combinational Functions
- Programming Technologies

Decoder

Decoder

- **Decoder:**
  - Translate an coding into another coding
- **Function:**
  - input variables is convertd 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 $n \leq m \leq 2n$

variety
- Variable decoder
- Code system transform decoder
- Display decoder

Generic Decoder

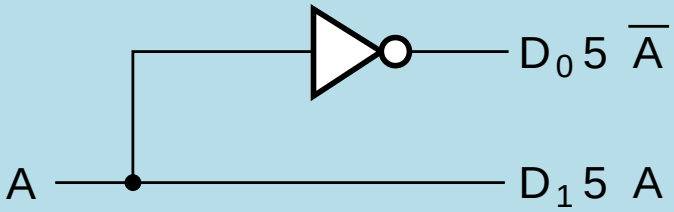
- **Decoding**
  - - the conversion of an *n*-bit input code to an *m*-bit output code with
    $n \leq m \leq 2^n$ such that each valid code word produces a unique output code(Or Signals)
- **Circuits that perform decoding are called *decoders***
- **Here, functional blocks for decoding are**
  - called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

(a)

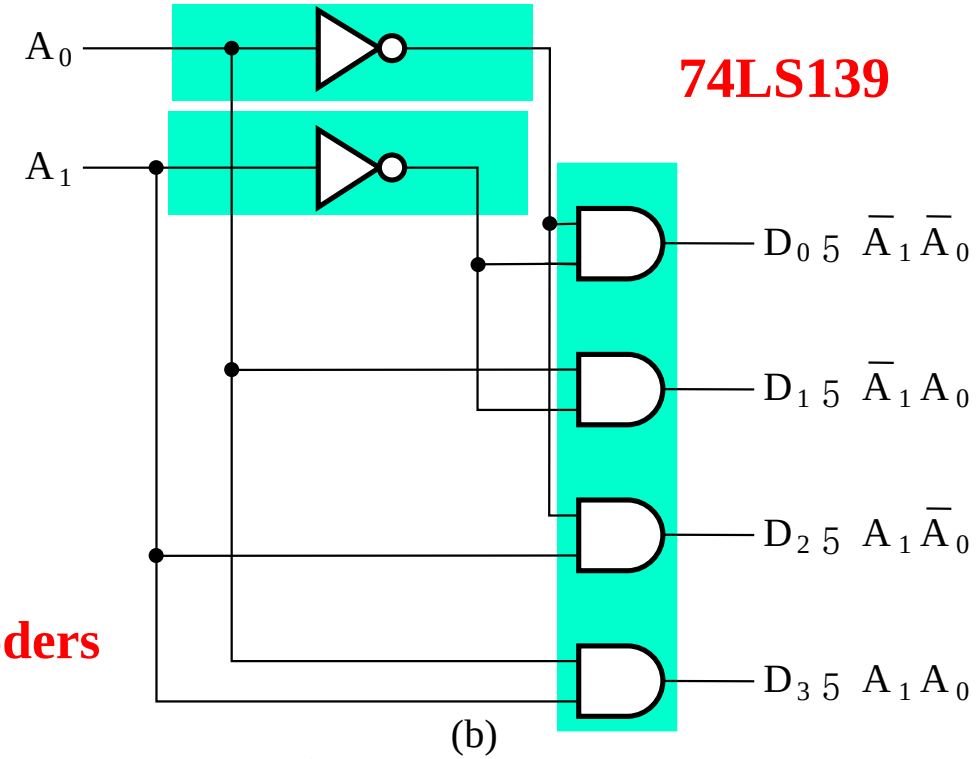


(b)

- 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)



(b)

- 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 $\neq 2^n$

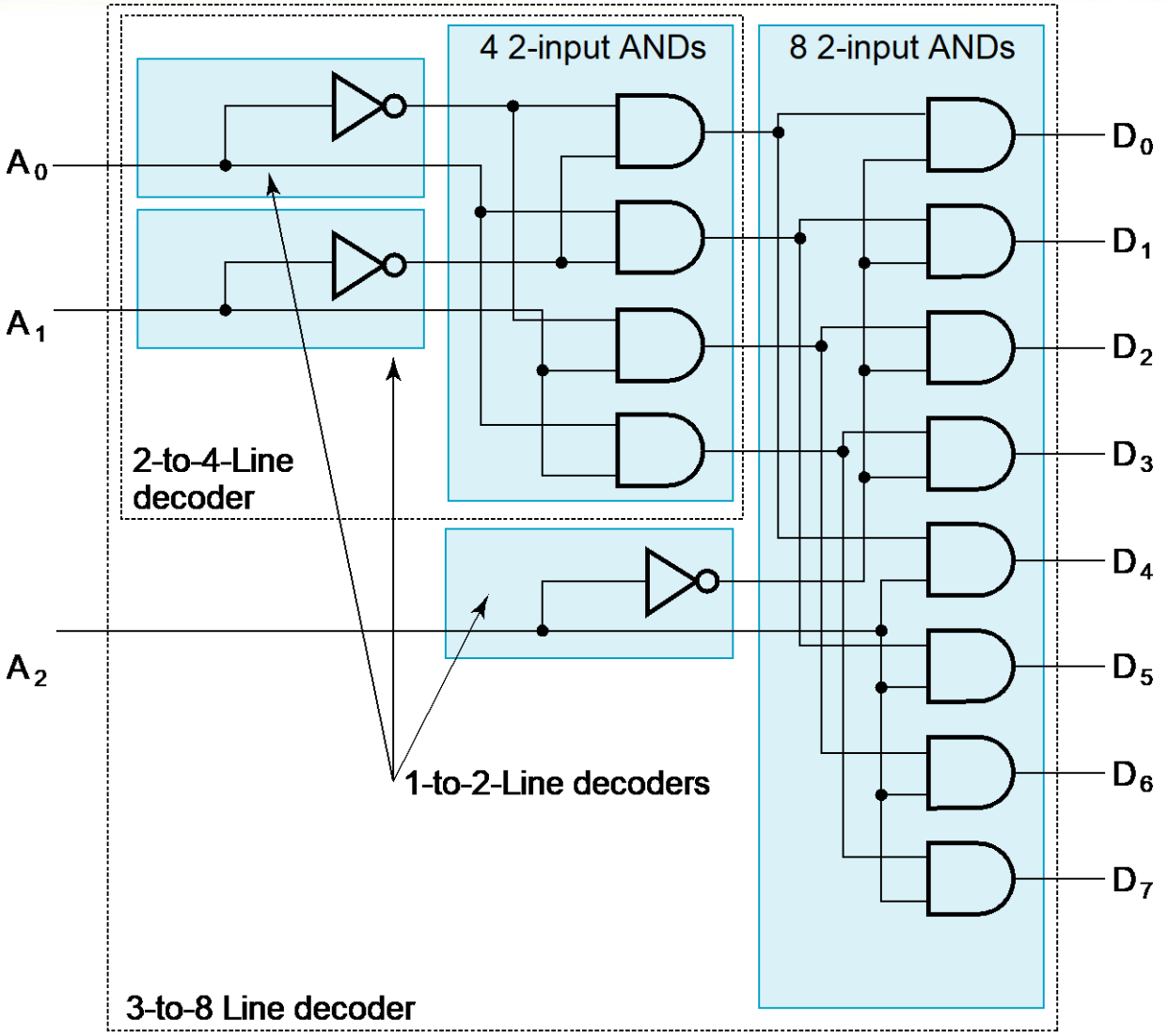
Decoder Expansion - Example 1

- **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



3-8变量译码器的实例(原语)描述

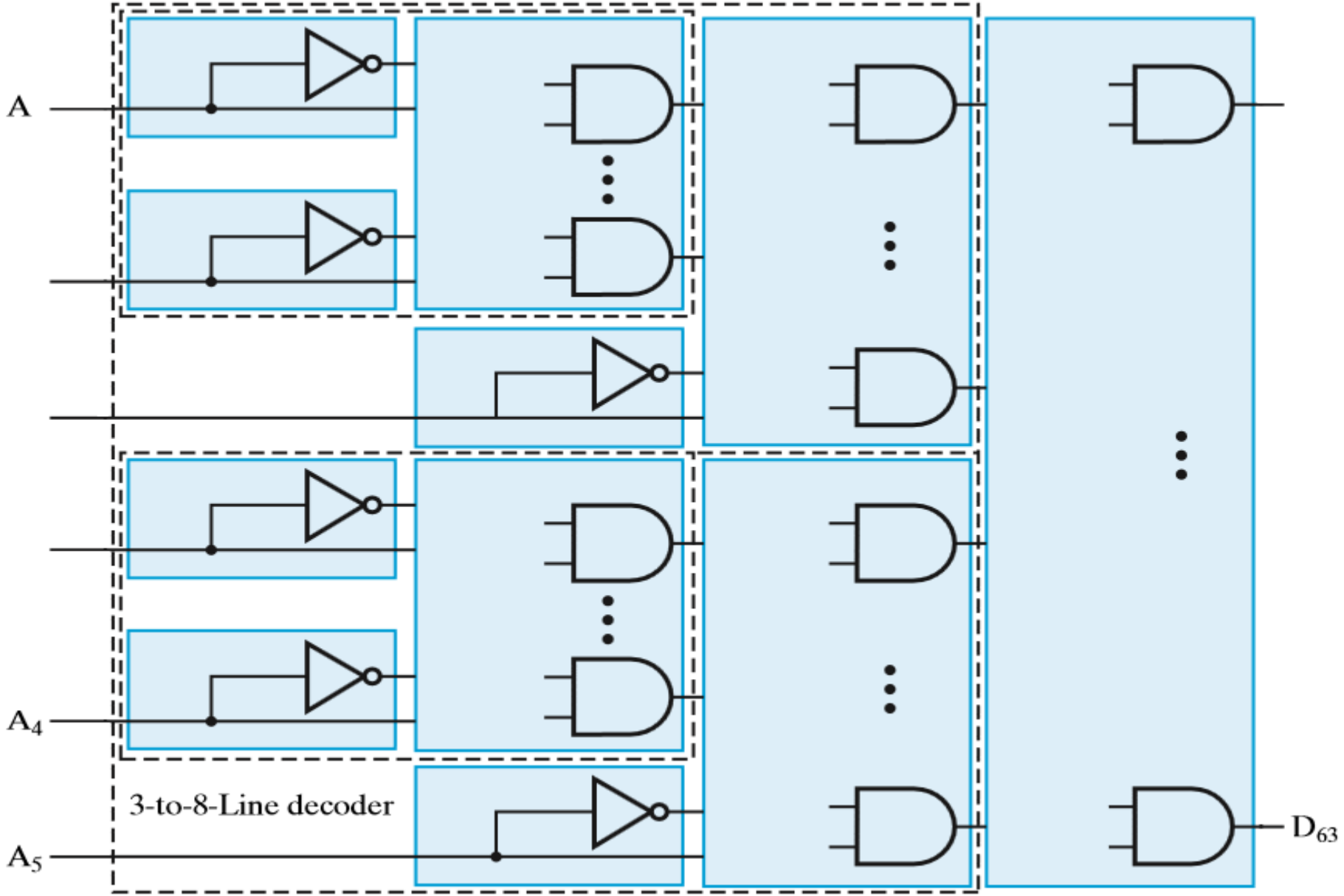
```
module decoder_3_8(C, B, A, G, G2A,G2B, Y);
……
output [7:0]Y;                                          //端口及变量定义
        not     node0(Cn, C);
                node1(Bn, B);
                node3(An, A);
                node4(Gn, G);
        nor     node5(EN, Gn, G2A, G2B);
        and     node_1_0(D0, Bn, An);
                node_1_1(D1, Bn, A);
                node_1_2(D2, B,   An);
                node_1_3(D3, B, A);
        nand    node_2_0(Y[0], EN, D0, Cn);
                node_2_1(Y[1], EN, D1, Cn);
                node_2_2(Y[2], EN, D2, Cn);
                node_2_3(Y[3], EN, D3, Cn);
                node_2_4(Y[4], EN, D0, C);
                node_2_1(Y[1], EN, D1, C);
                node_2_2(Y[2], EN, D2, C);
                node_2_1(Y[3], EN,D3, C);
endmodule
```

请与电路综合后的系统生成代码比较

Decoder Expansion - Example 2

- **6-to-64-line decoder**
  - Number of output ANDs = 64
  - Number of inputs to decoders driving output ANDs = 6
  - Closest possible split to equal (k=n/2=3)
    - 3-to-8-line decoder
    - 3-to-8-line decoder
  - Complete using known 3-8 and 2-to-4 line decoders
- **For gate input cost**
  - 6+2*2*4+2*64=182 Expansion mothed
  - 6+6*64=390 minterm mothed

6-to-64-line decoder

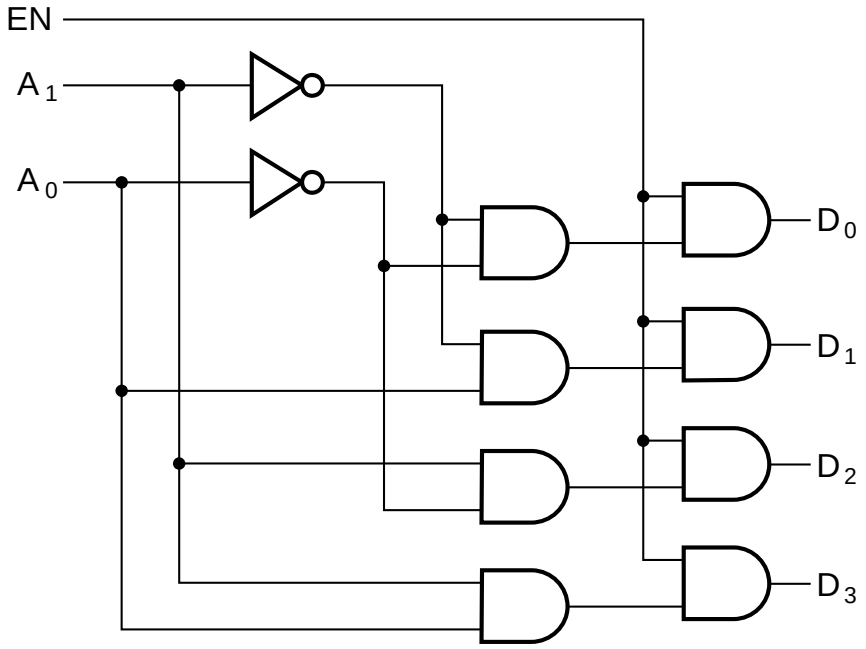


Decoder with Enable

- **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)

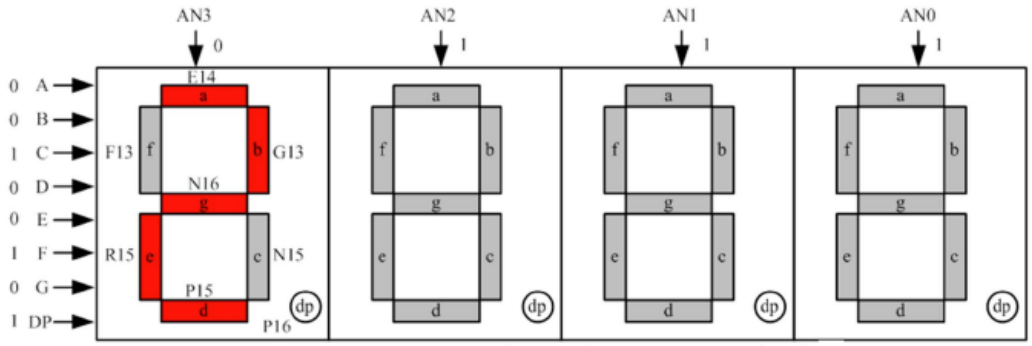


(b)

Display decoder

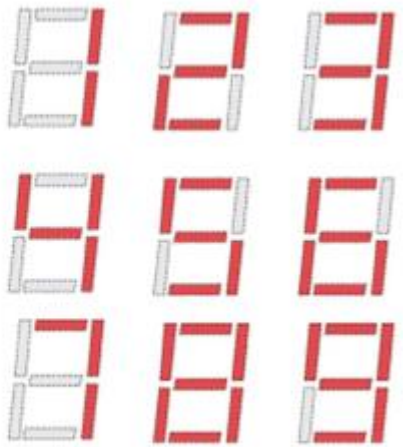
Common electrodes:

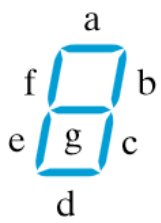
- Common cathode, common anode



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



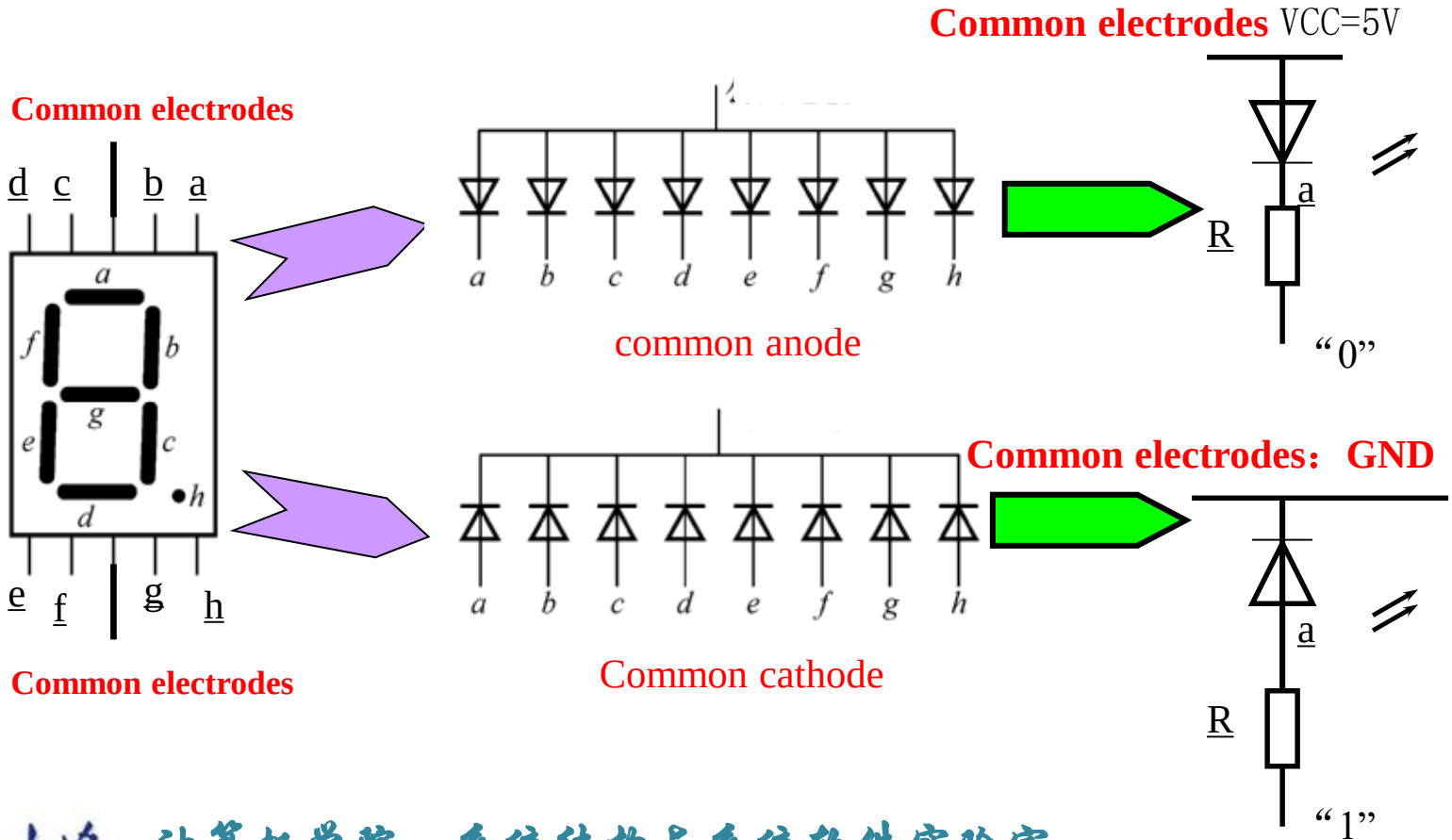


(a) Segment designation

(b) Numeric designation for display

common anode

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

Hex 7- segment decoder

common anode

Hex	$D_3D_2D_1D_0$	BI	a	b	c	d	e	f	g	p
0	0 0 0 0	1	0	0	0	0	0	0	1	p
1	0 0 0 1	1	1	0	0	1	1	1	1	p
2	0 0 1 0	1	0	0	1	0	0	1	0	p
3	0 0 1 1	1	0	0	0	0	1	1	0	p
4	0 1 0 0	1	1	0	0	1	1	0	0	p
5	0 1 0 1	1	0	1	0	0	1	0	0	p
6	0 1 1 0	1	0	1	0	0	0	0	0	p
7	0 1 1 1	1	0	0	0	1	1	1	1	p
8	1 0 0 0	1	0	0	0	0	0	0	0	P
9	1 0 0 1	1	0	0	0	0	1	0	0	P
A	1 0 1 0	1	0	0	0	1	0	0	0	P
B	1 0 1 1	1	1	1	0	0	0	0	0	P
C	1 1 0 0	1	0	1	1	0	0	0	1	P
D	1 1 0 1	1	1	0	0	0	0	1	0	P
E	1 1 1 0	1	0	1	1	0	0	0	0	P
F	1 1 1 1	1	0	1	1	1	0	0	0	P
X	x x x x	0	1	1	1	1	1	1	1	1

a

0	1	0	0
1	0	0	0
0	1	0	0
0	0	1	0

b

0	0	0	0
0	1	0	1
1	0	1	1
0	0	1	0

c

0	0	0	1
0	0	0	0
1	0	1	1
0	0	0	0

d

0	1	0	0
1	0	1	0
0	0	1	0
0	0	0	1

e

0	1	1	0
1	1	1	0
0	0	0	0
0	1	0	0

$a = \bar{D}_3\bar{D}_2\bar{D}_1D_0+\bar{D}_3D_2\bar{D}_1\bar{D}_0+D_3D_2\bar{D}_1D_0+\bar{D}_3D_2\bar{D}_1D_0$

$b = \bar{D}_3D_2\bar{D}_1D_0+D_2D_1\bar{D}_0+D_3D_2\bar{D}_0+D_3D_1D_0$

$c = \bar{D}_3\bar{D}_2D_1\bar{D}_0+D_3D_2\bar{D}_0+D_3D_2D_1$

$d = \bar{D}_3\bar{D}_2\bar{D}_1D_0+\bar{D}_3D_2\bar{D}_1\bar{D}_0+D_2D_1D_0+D_3\bar{D}_2D_1\bar{D}_0$

$e = \bar{D}_3D_0+\bar{D}_3D_2\bar{D}_1+\bar{D}_2\bar{D}_1D_0$

$f = \bar{D}_3\bar{D}_2D_0+\bar{D}_3\bar{D}_2D_1+\bar{D}_3D_1D_0+D_3D_2\bar{D}_1D_0$

$g = \bar{D}_3\bar{D}_2\bar{D}_1+\bar{D}_3D_2D_1D_0+D_3D_2\bar{D}_1\bar{D}_0$

f

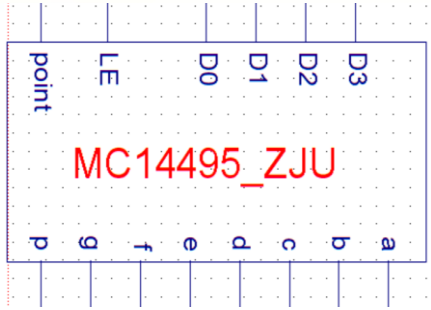
0	1	1	1
0	0	1	0
0	1	0	0
0	0	0	0

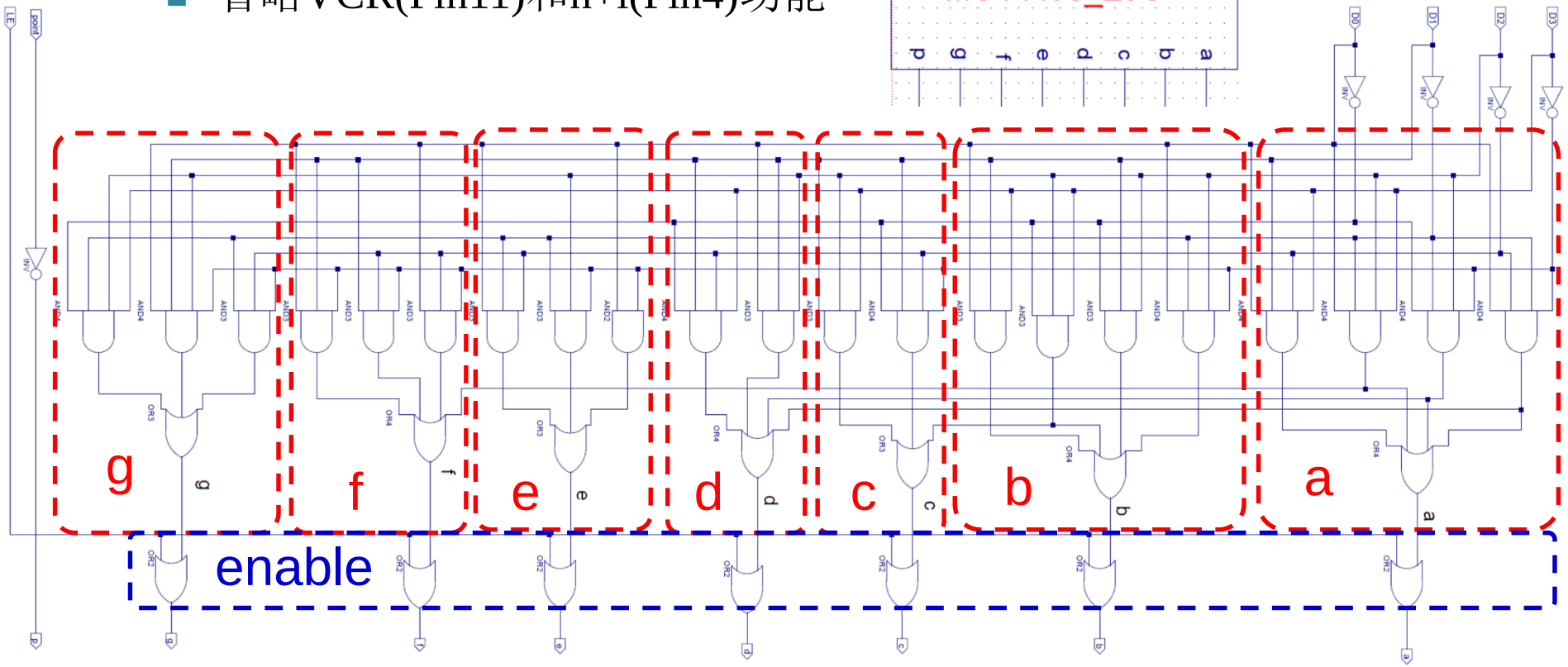
g

1	1	0	0
0	0	1	0
1	0	0	0
0	0	0	0

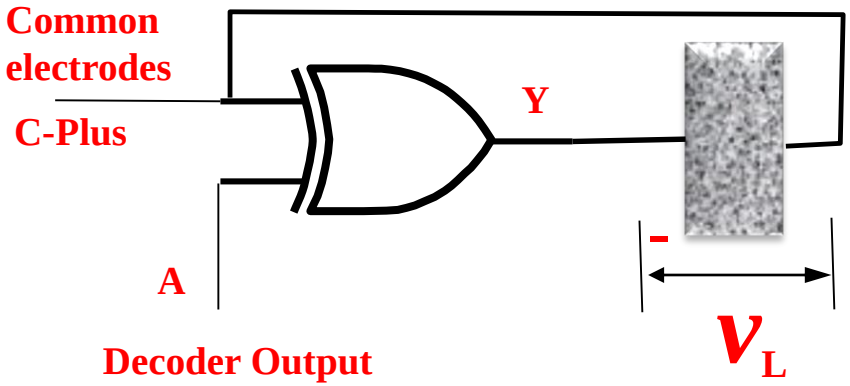
Hex to 7-segment decoder Schematic

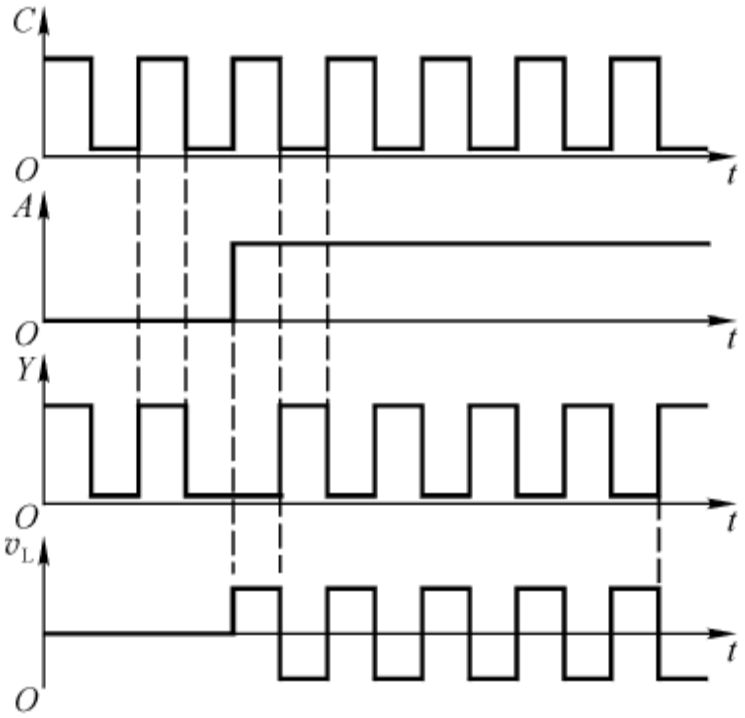
- 兼容MC14495
  - 省略VCR(Pin11)和h+i(Pin4)功能





LCD driving principle





Encoder

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

Encoder Example

- **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**
  - A decimal-to-BCD encoder
    - Inputs: 10 bits corresponding to decimal digits 0 through 9, (D_0, …, 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.

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, …, 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 combi-nations 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	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	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:

$A_2 = D_4$

$A_1 = \overline{\mathbf{D_4}}\, D_3 + \overline{\mathbf{D_4}}\,\overline{\mathbf{D_3}}\, D_2 = \overline{\mathbf{D_4}}\, F_1, \qquad F_1 = (D_3 + D_2)$

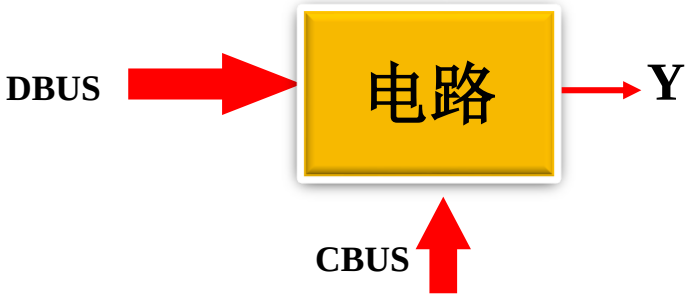
$A_0 = \overline{\mathbf{D_4}}\, D_3 + \overline{\mathbf{D_4}}\,\overline{\mathbf{D_3}}\,\overline{\mathbf{D_2}}\, D_1 = \overline{\mathbf{D_4}}\,(D_3 + \overline{\mathbf{D_2}}\, D1)$

$V = D_4 + F_1 + D_1 + D_0$

Multiplexers

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**

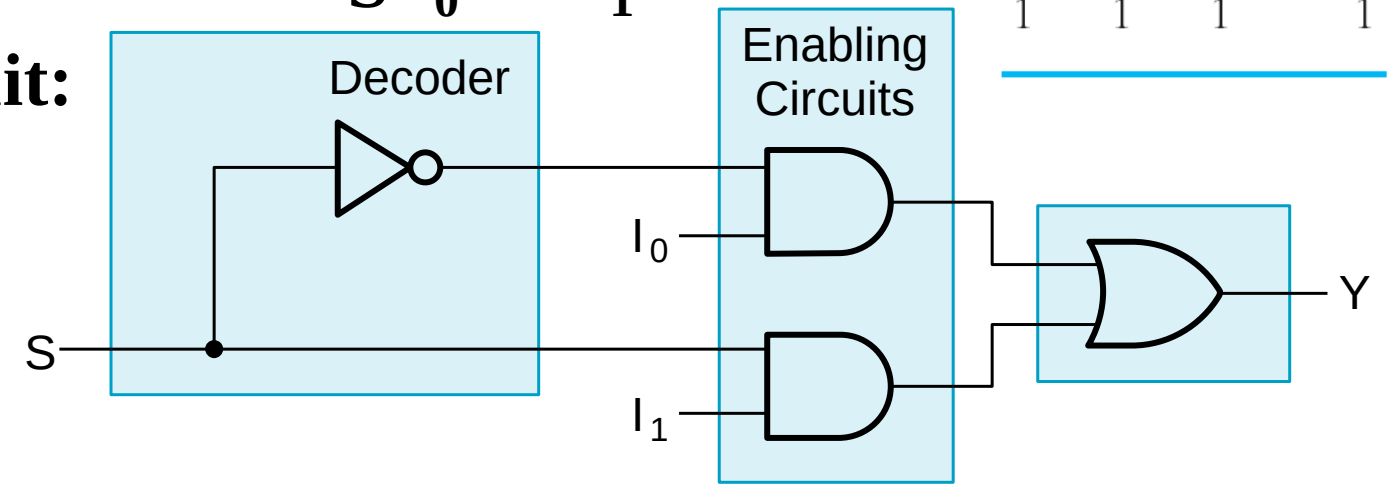
Multiplexers

- **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}, … S_0) called *selection inputs*,
  - 2^n information inputs (I_{2^n-1}, … 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:** $Y = \overline{S} I_0 + S I_1$
- **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

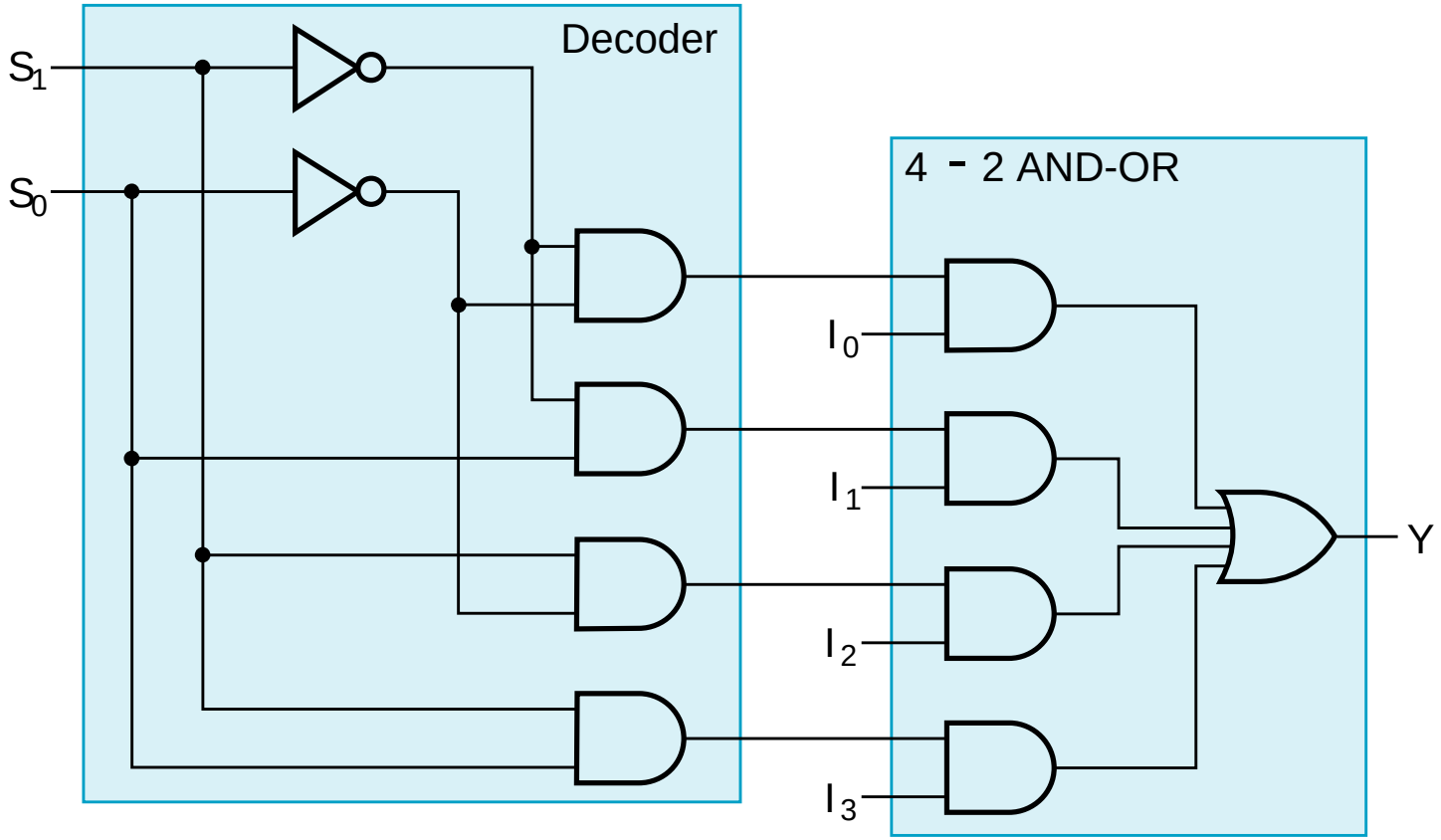


2-to-1-Line Multiplexer (continued)

- **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 ×2 AND-OR

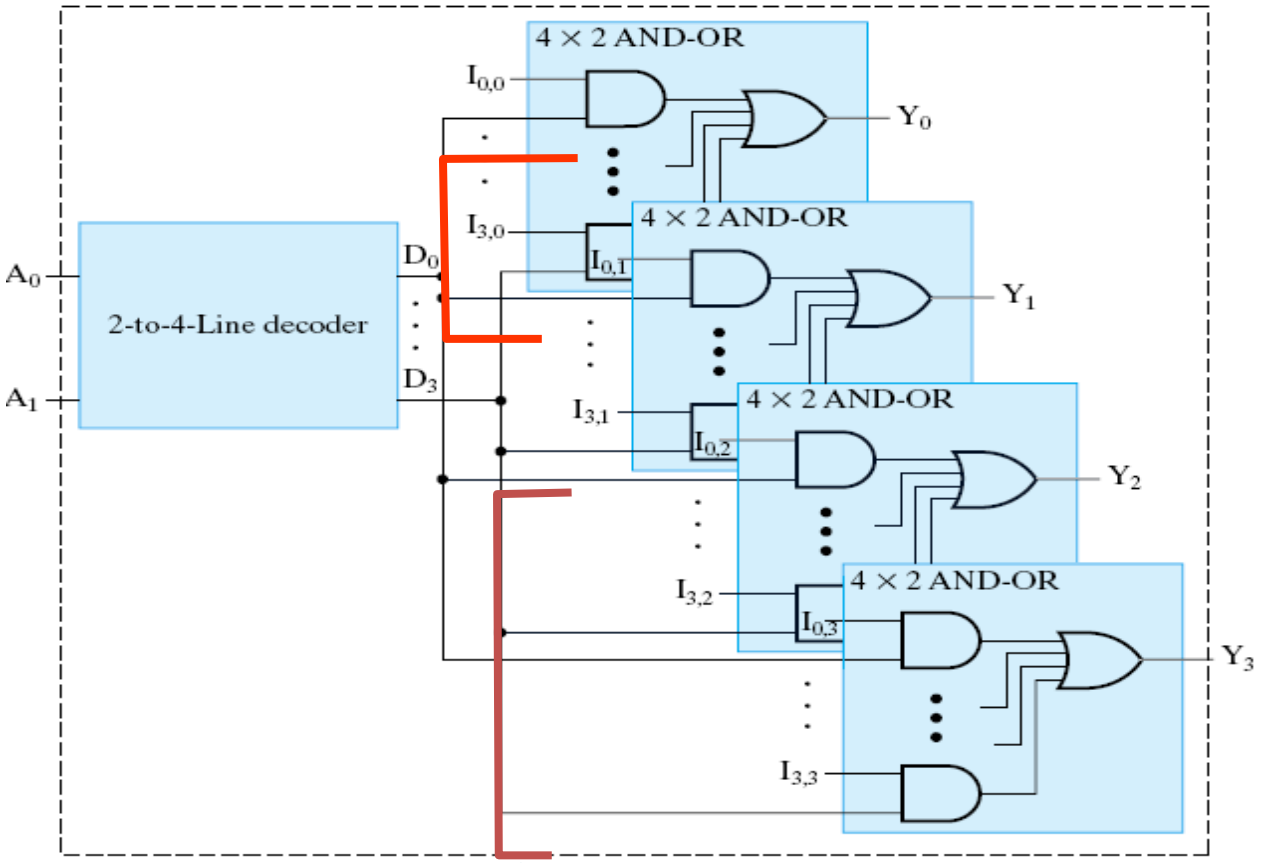
Example: 4-to-1-line Multiplexer

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



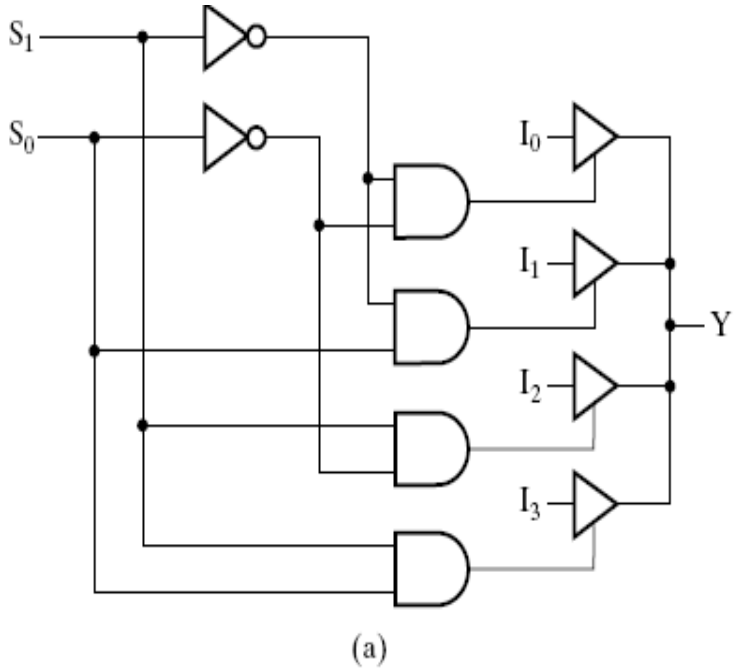
Multiplexer Width Expansion

- 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 multi-plexer

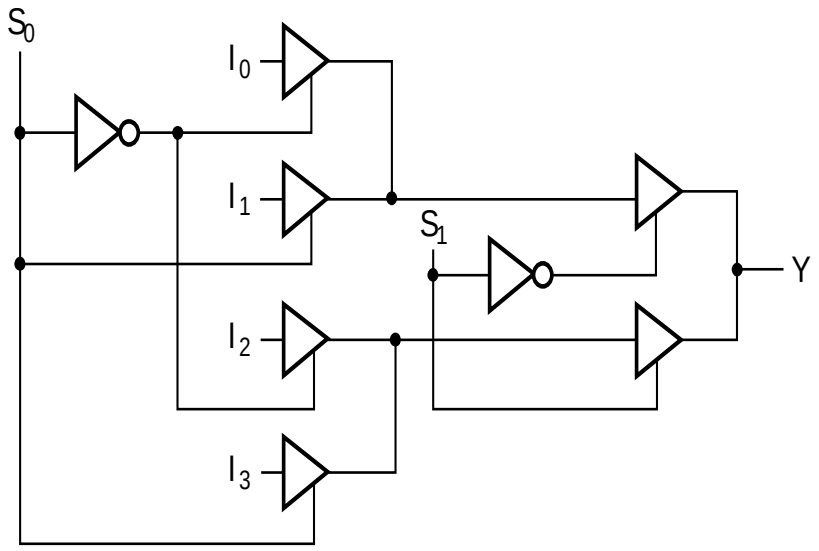


Other Selection Implementations

- Three-state logic in place of AND-OR



(a)



(b)

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

Case Describing

```
module  MUX8T1_32(input [2:0]s,
                  input [31:0]I0,
                  input [31:0]I1,
                  input [31:0]I2,
                  input [31:0]I3,
                  input [31:0]I4,
                  input [31:0]I5,
                  input [31:0]I6,
                  input [31:0]I7,

                  output reg[31:0]o
                  );

      always@*
         case(s)
            3'b000: o = I0;
            3'b001: o = I1;
            3'b010: o = I2;
            3'b011: o = I3;
            3'b100: o = I4;
            3'b101: o = I5;
            3'b110: o = I6;
            3'b111: o = I7;
         endcase

endmodule
```

```
module  MUX2T1_32(input[31:0]I0,
                  input[31:0]I1,
                  input s,
                  output[31:0]o
                   );

   assign o = s ? I1 : I0;

endmodule
```

Course Outline

- Functions and functional blocks
- Rudimentary logic functions
- Commonly Logic Function Design
- **Implementing Methods Of Combinational Functions**
- Programming Technologies

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

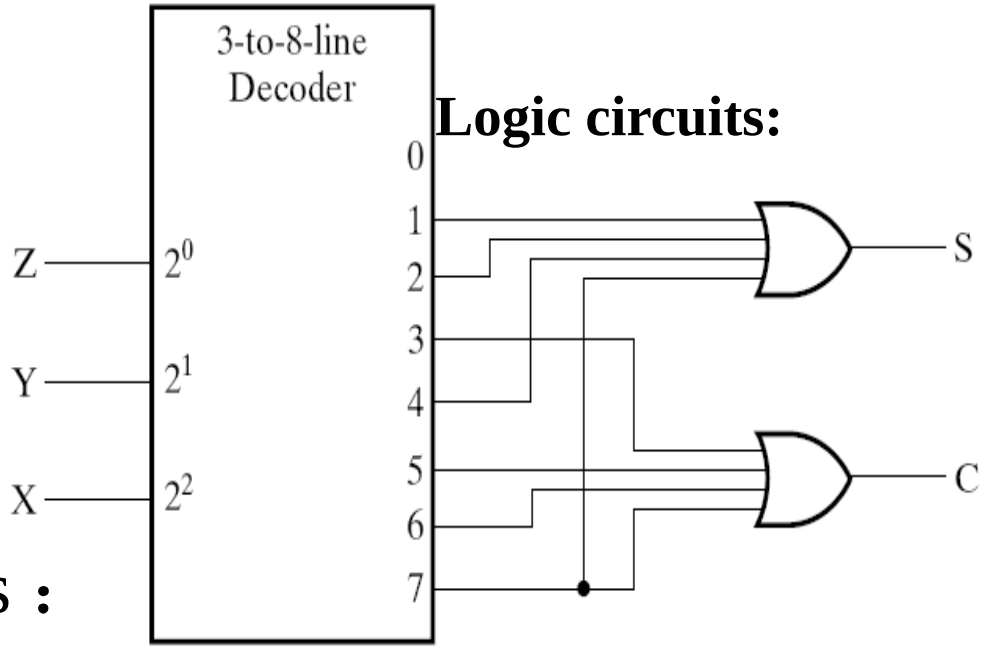
- **Implement *m* functions of *n* variables with:**
  - Sum-of-minterms expressions
  - One n-to-2^n-line decoder
  - m OR gates, one for each output
- **Approach 1:**
  - Find the truth table for the functions
  - Make a connection to the corresponding OR from the corresponding decoder output wherever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

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:



  - Minterms expressions：

S（X，Y，Z）=Σm（1，2，4，7）

C（X，Y，Z）=Σm（3，5，6，7）

Decoder and OR Gates Example-1

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

$P_1 = A_7 \oplus A_5 \oplus A_3$

$P_2 = A_7 \oplus A_6 \oplus A_3$

$P_4 = A_7 \oplus A_6 \oplus A_5$

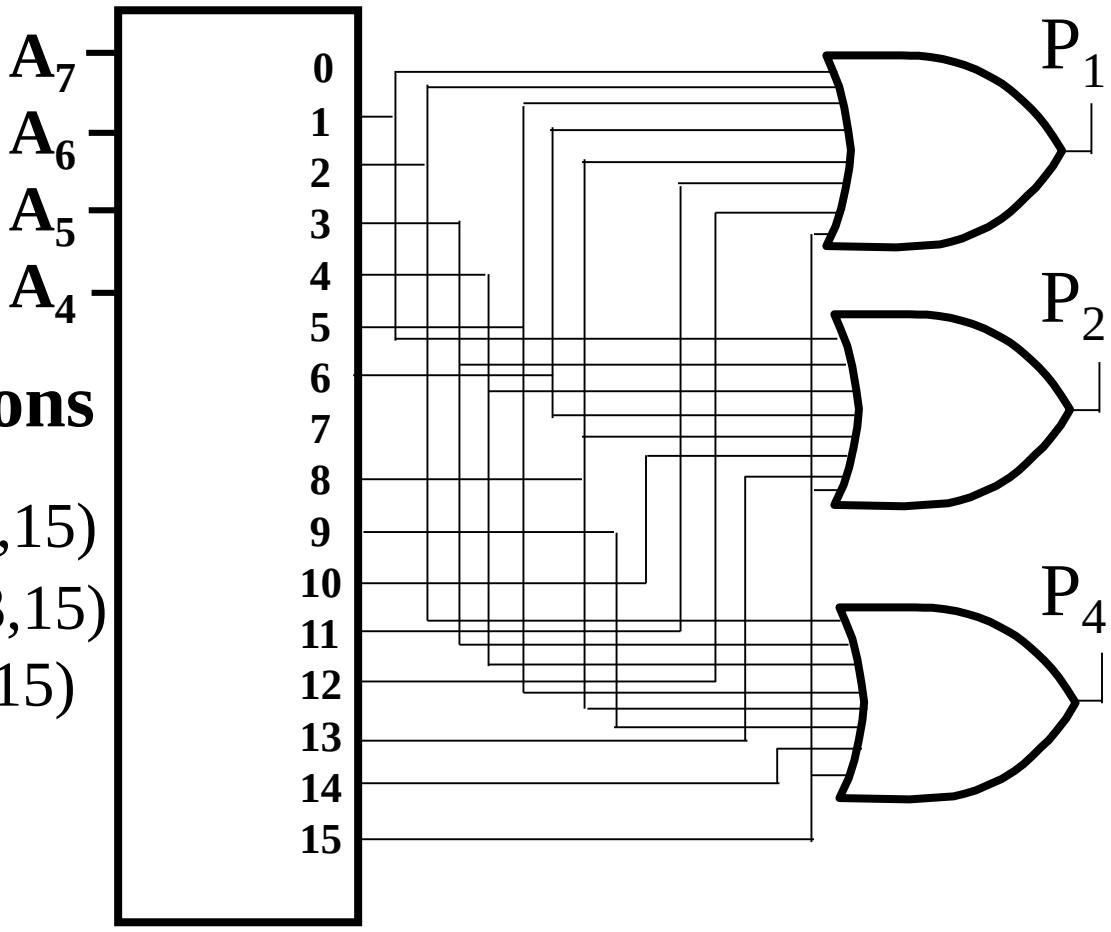
- **Finding sum of minterms expressions**

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

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

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

- **Find circuit**
- **Is this a good idea?**



Combinational Logic Implementation - Multiplexer Approach 1

- **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 $S_{n-1}, \ldots, S_0$
    - 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

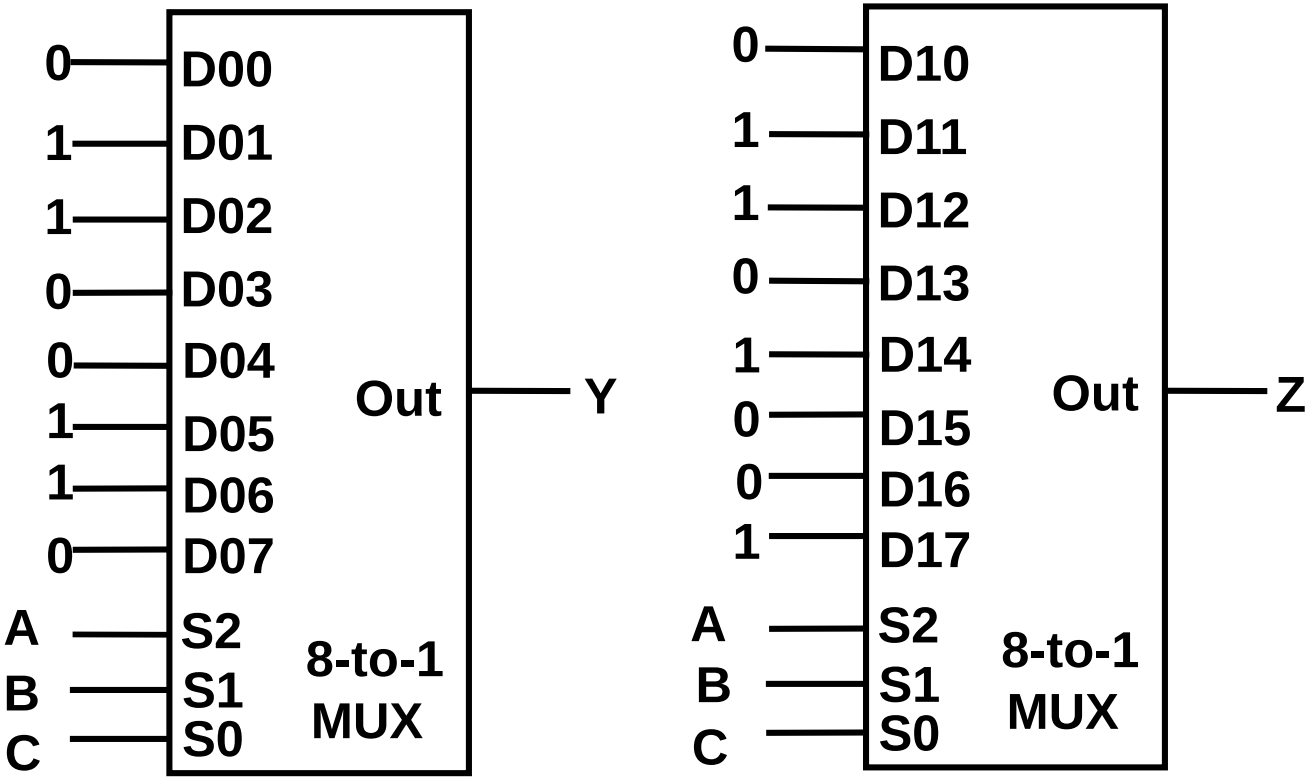
Gray to Binary (continued)

- **Rearrange the table so that the input combinations are in counting order**

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

- **Functions y and z can be implemented using a dual 8-to-1-line multiplexer by:**
  - 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!

相当于ROM

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, $\overline{\mathbf{X}}$) **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 $\overline{\mathbf{X}}$

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

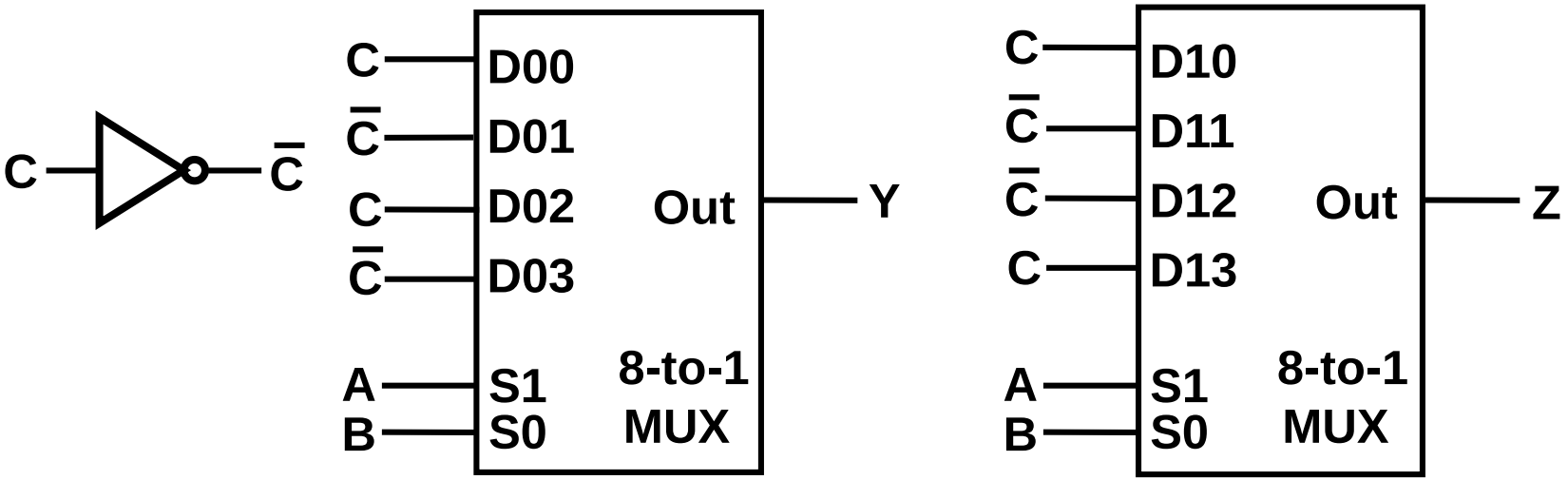
Gray to Binary (continued)

- 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 = $\overline{C}$	F = $\overline{C}$
0 1 1	1 0 0		
1 0 0	0 0 1	F = C	F = $\overline{C}$
1 0 1	1 1 0		
1 1 0	0 1 0	F = $\overline{C}$	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.

Course Outline

- Functions and functional blocks
- Rudimentary logic functions
- Commonly Logic Function Design
- Implementing Methods Of Combinational Functions
- Programming Technologies

Programming Technologies

- **Programming technologies are used to:**
  - Control connections
  - Build lookup tables
  - Control transistor switching
- **The technologies**
  - Control connections
    - Mask programming
    - Fuse
    - Antifuse
    - Single-bit storage element

Programming Technologies

- **The technologies (continued)**
  - Build lookup tables
    - Storage elements (as in a memory)
  - Transistor Switching Control
    - Stored charge on a floating transistor gate
      - Erasable
      - Electrically erasable
      - Flash (as in Flash Memory)
    - Storage elements (as in a memory)

Technology Characteristics

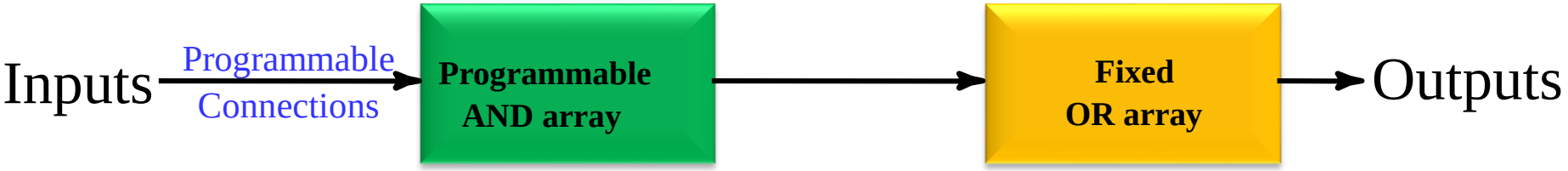
- **Permanent - Cannot be erased and reprogrammed**
  - Mask programming
  - Fuse
  - Antifuse
- **Reprogrammable**
  - Volatile - Programming lost if chip power lost
    - Single-bit storage element
  - Non-Volatile
    - Erasable
    - Electrically erasable
    - Flash (as in Flash Memory)

Programmable Configurations

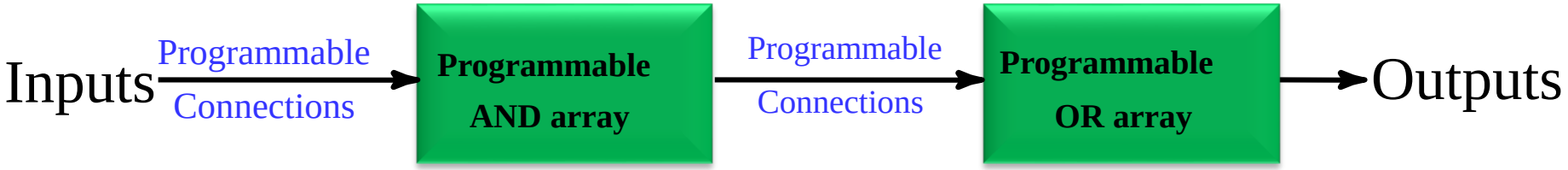
- *Read Only Memory* (*ROM*) - a fixed array of AND gates and a programmable array of OR gates
- *Programmable Array Logic* (*PAL*)® - a programmable array of AND gates feeding a fixed array of OR gates.
- *Programmable Logic Array* (*PLA*) - a programmable array of AND gates feeding a programmable array of OR gates.
- *Complex Programmable Logic Device* (*CPLD*) */Field- Programmable Gate Array* (*FPGA*) - complex enough to be called "architectures" - See VLSI Programmable Logic Devices reading supplement

ROM, PAL and PLA Configurations

(a) Programmable read-only memory (PROM)

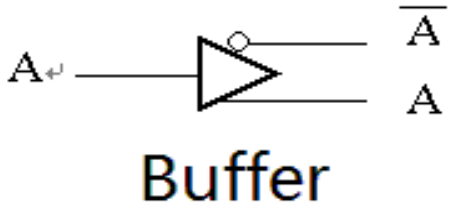


(b) Programmable array logic (PAL) device



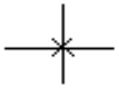
(c) Programmable logic array (PLA) device

Logical symbols



Buffer

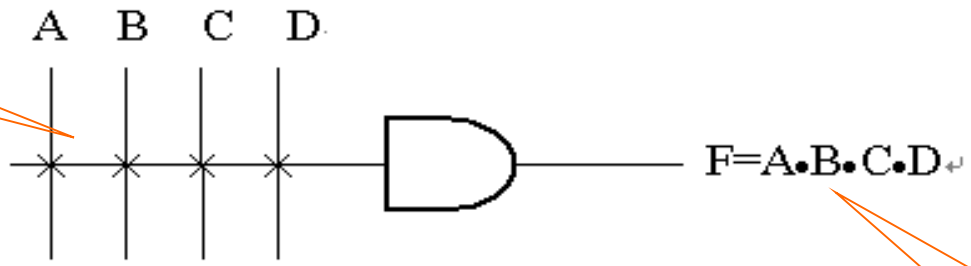
wire connecting
(not programmable)



wire connecting
(programmable)

Disconnection
(erasable)

AND Array



Array of AND **gates**
(programmable)

Product terms

ROM

Read Only Memory

Read Only Memory

- **Read Only Memories (ROM) or Programmable Read Only Memories (PROM) have:**
  - N input lines,
  - M output lines, and
  - 2^N decoded minterms.
- **Fixed AND array with 2^N outputs implementing all N-literal minterms.**
- **Programmable OR Array with M outputs lines to form up to M sum of minterm expressions.**

Read Only Memory-2

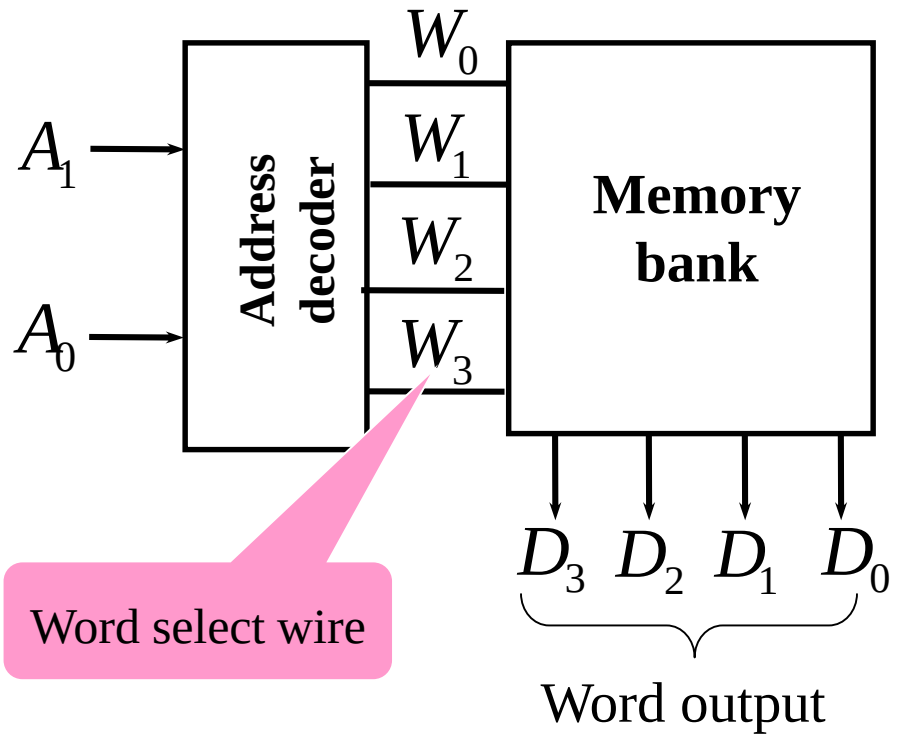
- **A program for a ROM or PROM is simply a multiple-output truth table**
  - If a 1 entry, a connection is made to the corresponding minterm for the corresponding output
  - If a 0, no connection is made
- **Can be viewed as a *memory* with the inputs as *addresses* of *data* (output values), hence ROM or PROM names!**

The general structure of the read-only memory

The general structure of the ROM

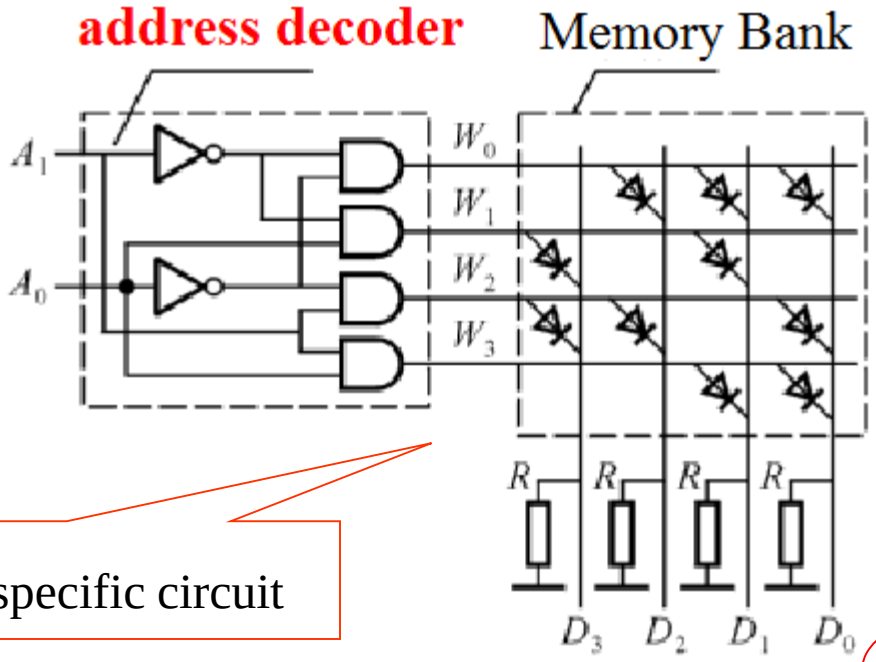
Size=2^2×4 bit=16

ROM structure

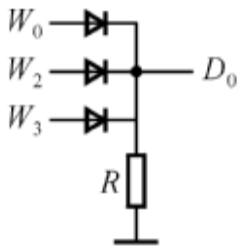


The logical structure of the read-only memory

The **address decoder** is a completely minterms (Full decoder) circuit, that is a non-programmable, “AND" array
The memory bank is a “OR" array

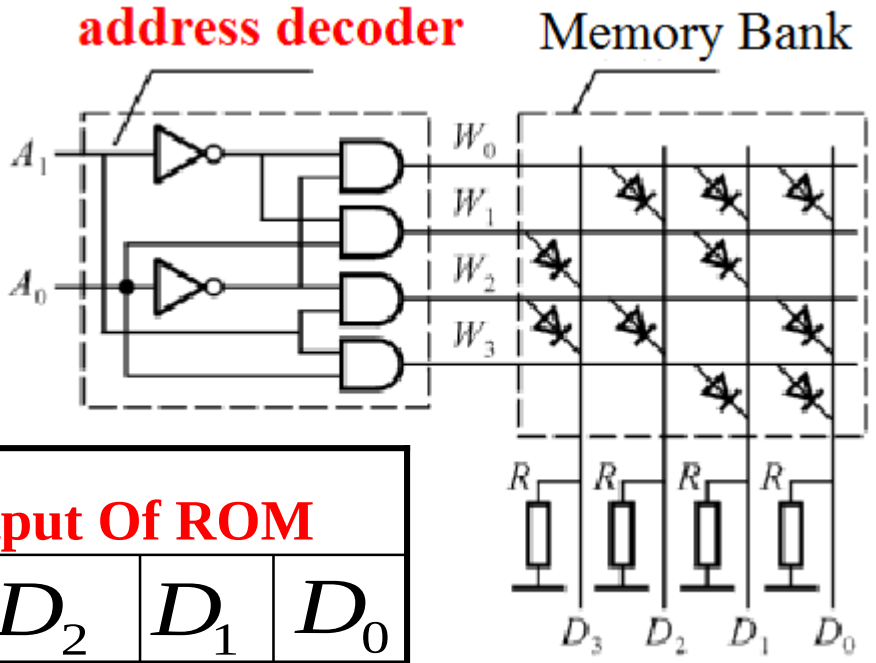


4×4 ROM specific circuit



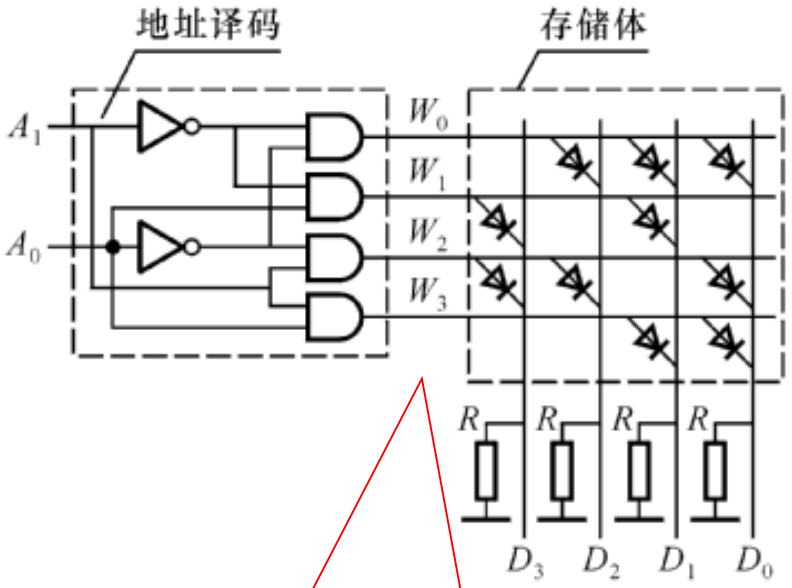
"D_0" bit wire "OR" gate structure

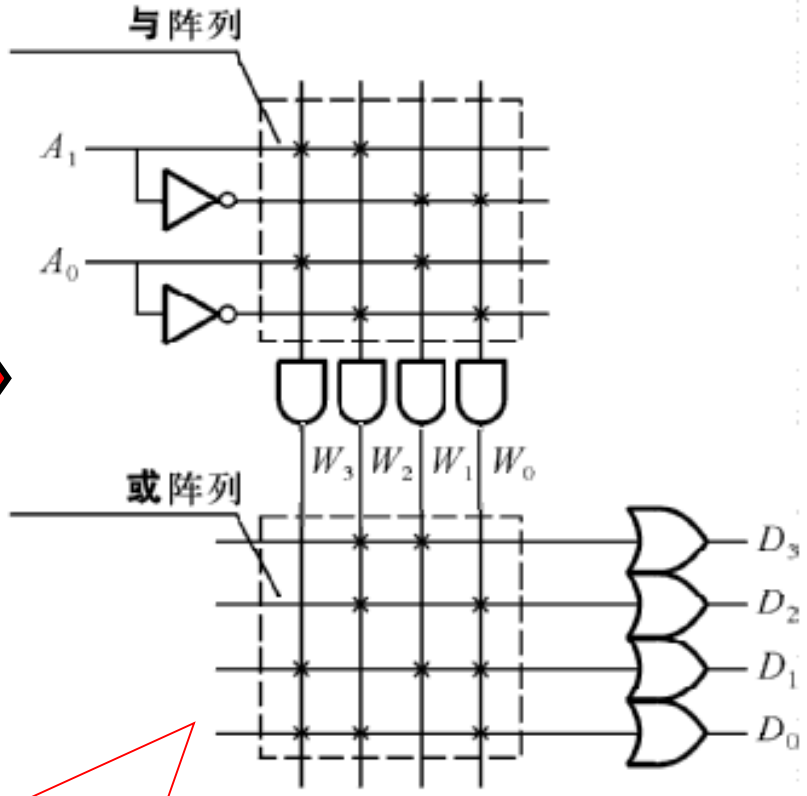
Output information of ROM



address		Word select wire	Output Of ROM			
A_1	A_0	W	D_3	D_2	D_1	D_0
0	0	W_0	0	1	1	1
0	1	W_1	1	0	1	0
1	0	W_2	1	1	0	1
1	0	W_3	0	0	1	1

ROM simplified logic symbol



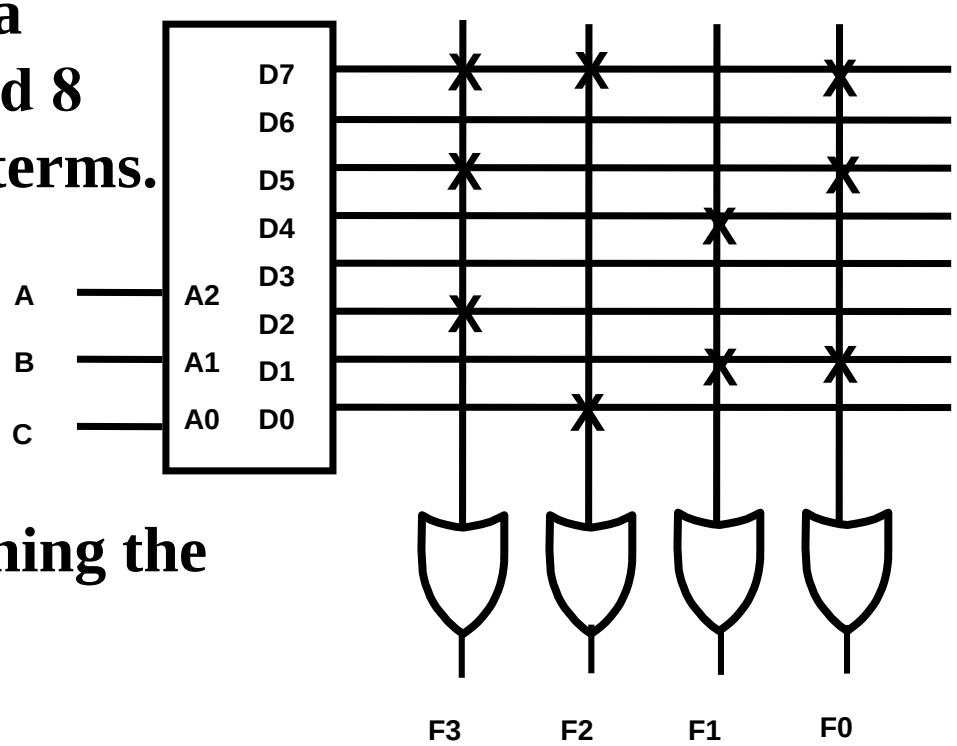


4×4ROM schematic

4×4ROM Simplified logic diagram

Read Only Memory Example

- **Example: A 8 × 4 ROM:**
  - N = 3 input lines, M= 4 output lines
- **The fixed "AND" array is a "decoder" with 3 inputs and 8 outputs implementing minterms.**
- **The programmable "OR" array uses a single line to represent all inputs to an OR gate. An "X" in the array corresponds to attaching the minterm to the OR**



- **Read Example:**
  - **For input $(A_2,A_1,A_0) = 011$, output is $(F_3,F_2,F_1,F_0) = 0011$.**
- **What are functions F_3, F_2 , F_1 and F_0 in terms of (A_2, A_1, A_0)?**

PAL

Programmable Array Logic

Programmable Array Logic (PAL)

- **The PAL is the opposite of the ROM, having a programmable set of ANDs combined with fixed ORs.**
- **Disadvantage**
  - ROM guaranteed to implement any M functions of N inputs. PAL may have too few inputs to the OR gates.
- **Advantages**
  - For given internal complexity, a PAL can have larger N and M
  - Some PALs have outputs that can be complemented, adding POS functions
  - No multilevel circuit implementations in ROM (without external connections from output to input). PAL has outputs from OR terms as internal inputs to all AND terms, making implementation of multi-level circuits easier.

Programmable Array Logic Example

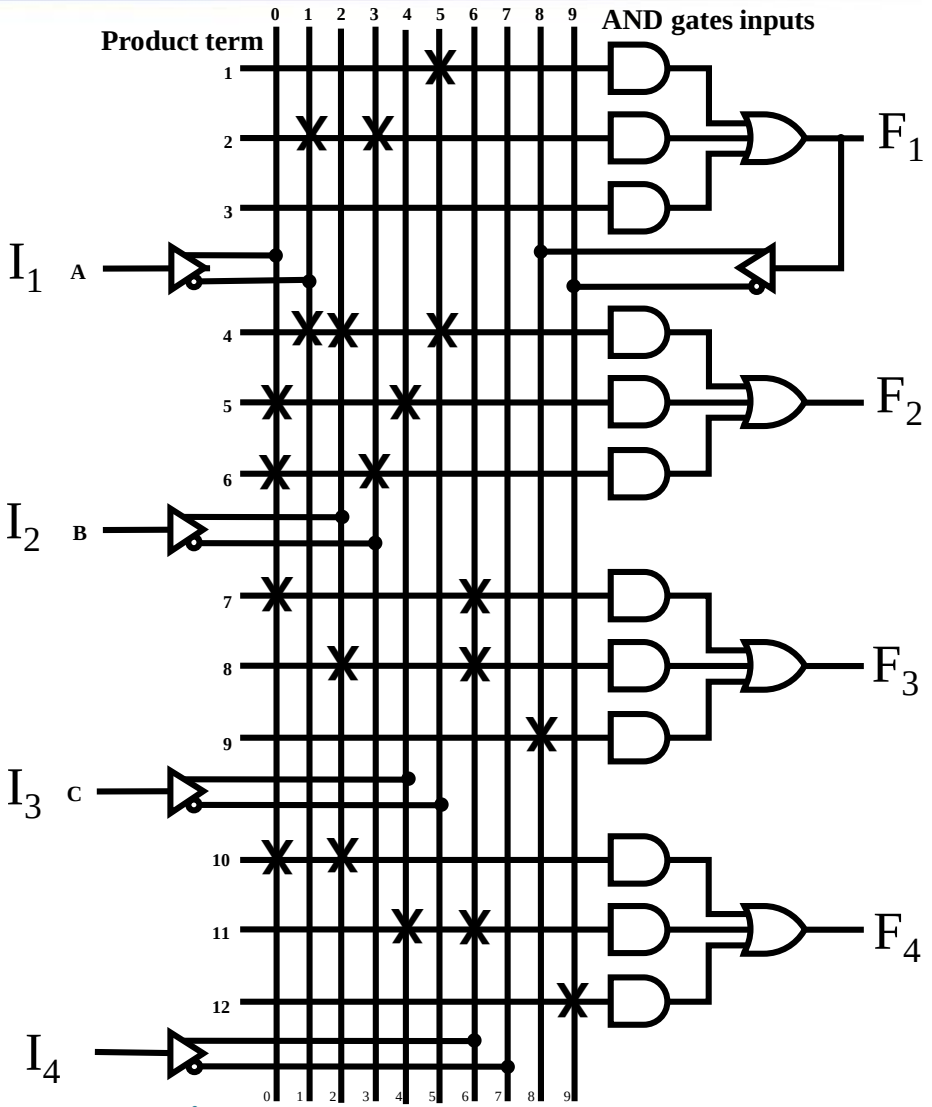
- 4-input, 3-output PAL with fixed, 3-input OR terms
- What are the equations for F1 through F4?

F1 = $\overline{A}\ \overline{B} + \overline{C}$

F2 = $\overline{A}\,B\,\overline{C} + AC + AB$

F3 =

F4 =



PLA

Programmable Logic Array

Programmable Logic Array (PLA)

- **Compared to a ROM and a PAL, a PLA is the most flexible having a programmable set of ANDs combined with a programmable set of ORs.**
- **Advantages**
  - A PLA can have large N and M permitting implement-ation of equations that are impractical for a ROM (because of the number of inputs, N, required
  - A PLA has all of its product terms connectable to all outputs, overcoming the problem of the limited inputs to the PAL Ors
  - Some PLAs have outputs that can be complemented, adding POS functions

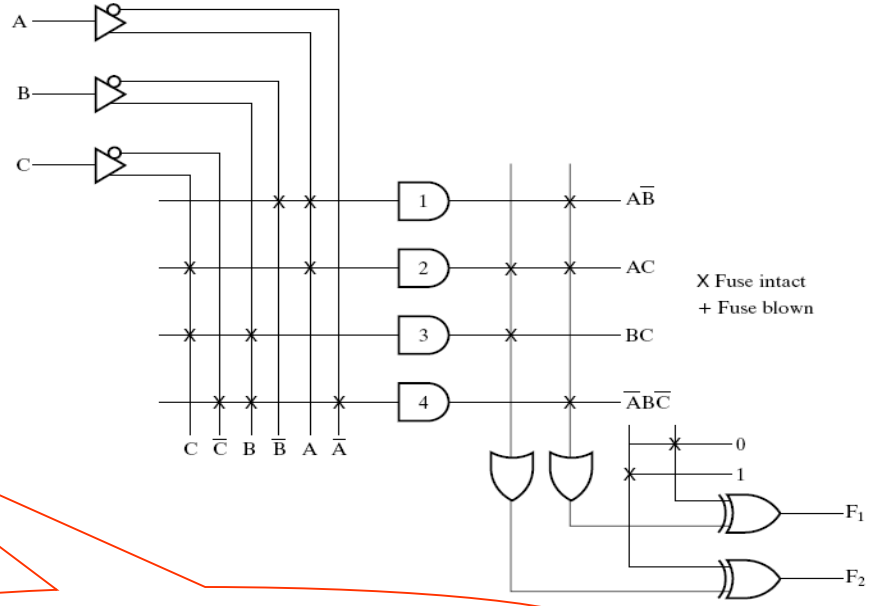
Programmable Logic Array (PLA)

- **Disadvantages**
  - Often, the product term count limits the application of a PLA.
  - Two-level multiple-output optimization is required to reduce the number of product terms in an implementation, helping to fit it into a PLA.
  - Multi-level circuit capability available in PAL not available in PLA. PLA requires external connections to do multi-level circuits.

Programmable Logic Array Example

- **Use of the PLA to implement a set of functions:**
  - First, must be listed AND-Items of function (As much as possible many public items)
  - Specifies whether need to be connected form the input to the AND gate array
  - Specifies whether need to be connected from AND-array- output to the OR-array
  - Whether the output is to be negated

	Product term	Inputs A	B	C	Outputs (T) F_1	(C) F_2
$A\bar{B}$	1	1	0	—	1	—
AC	2	1	—	1	1	1
BC	3	—	1	1	—	1
$\bar{A}B\bar{C}$	4	0	1	0	1	—



build up a program table

GAL & CPLD

Other Programmable Logic

Generic array logic structure GAL

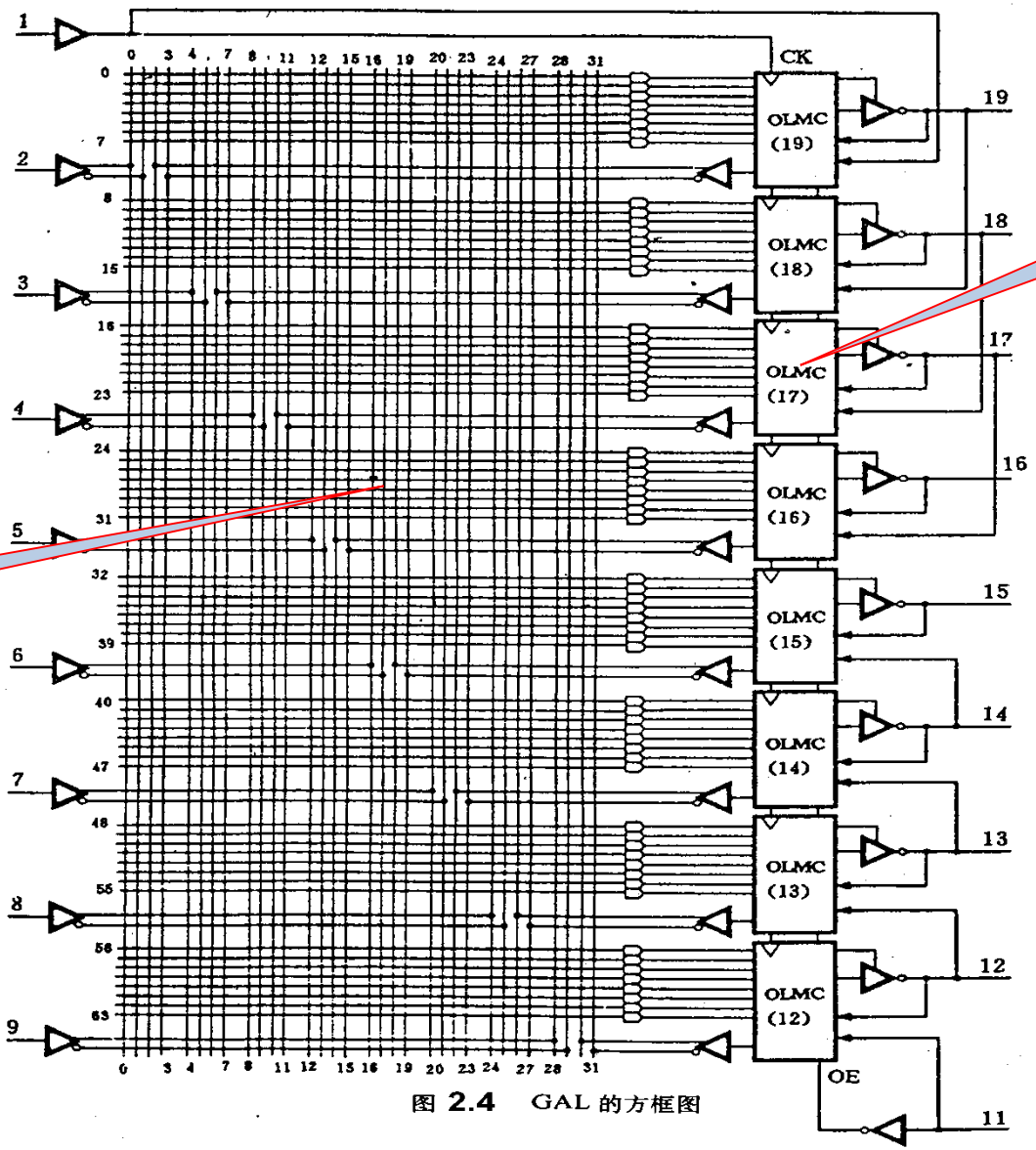
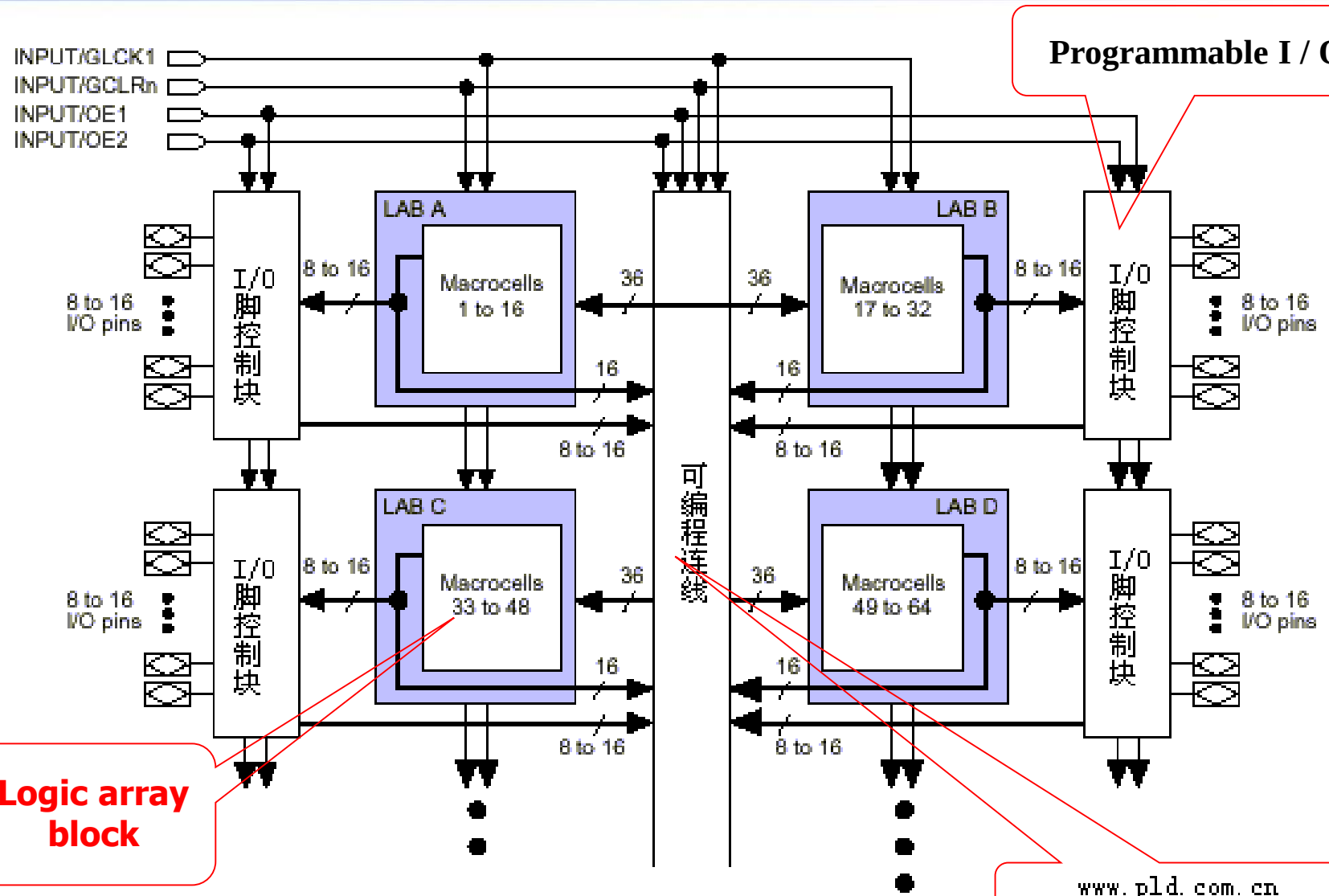


图 2.4 GAL 的方框图

OLMC

Logic macrocells

Complex Programmable Logic Devices CPLD （Altera MAX7000S Series ）



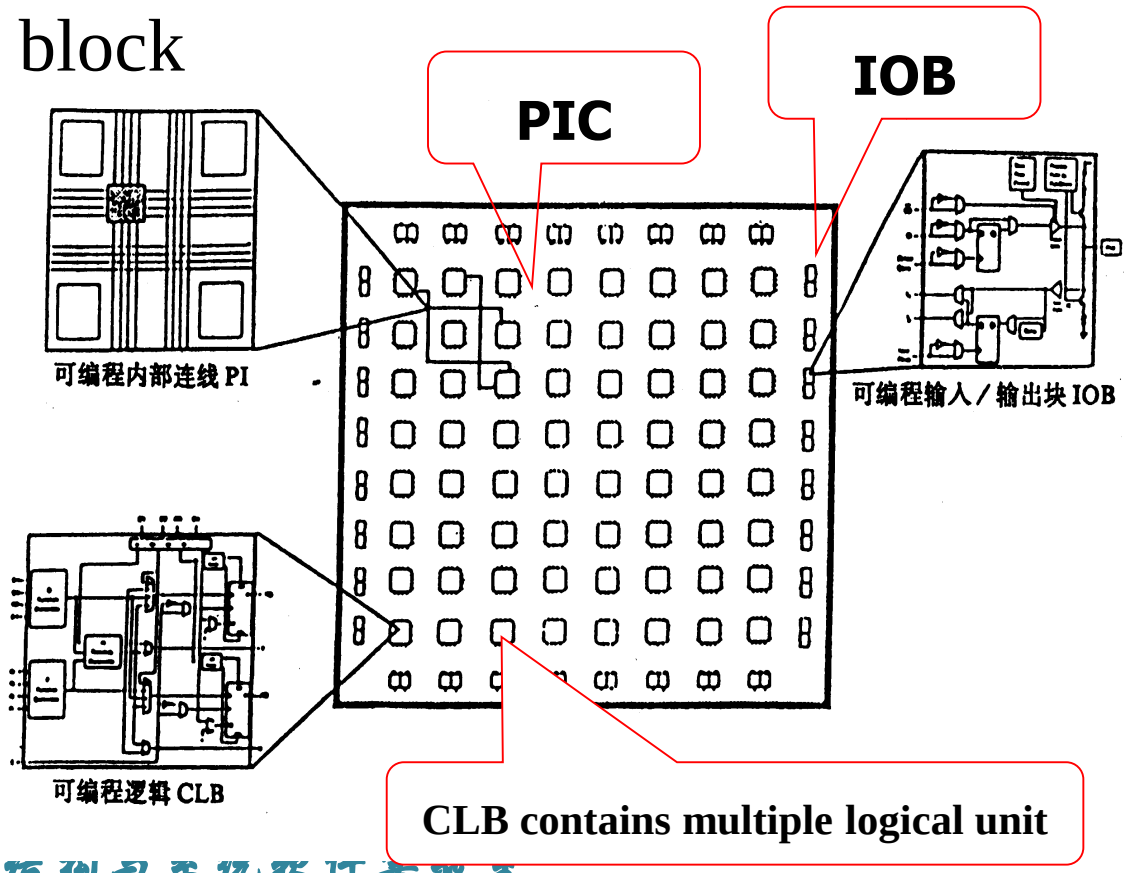
Logic array block contains multiple macrocells

FPGA

Programmable Gate Array

Programmable Gate Array: FPGA

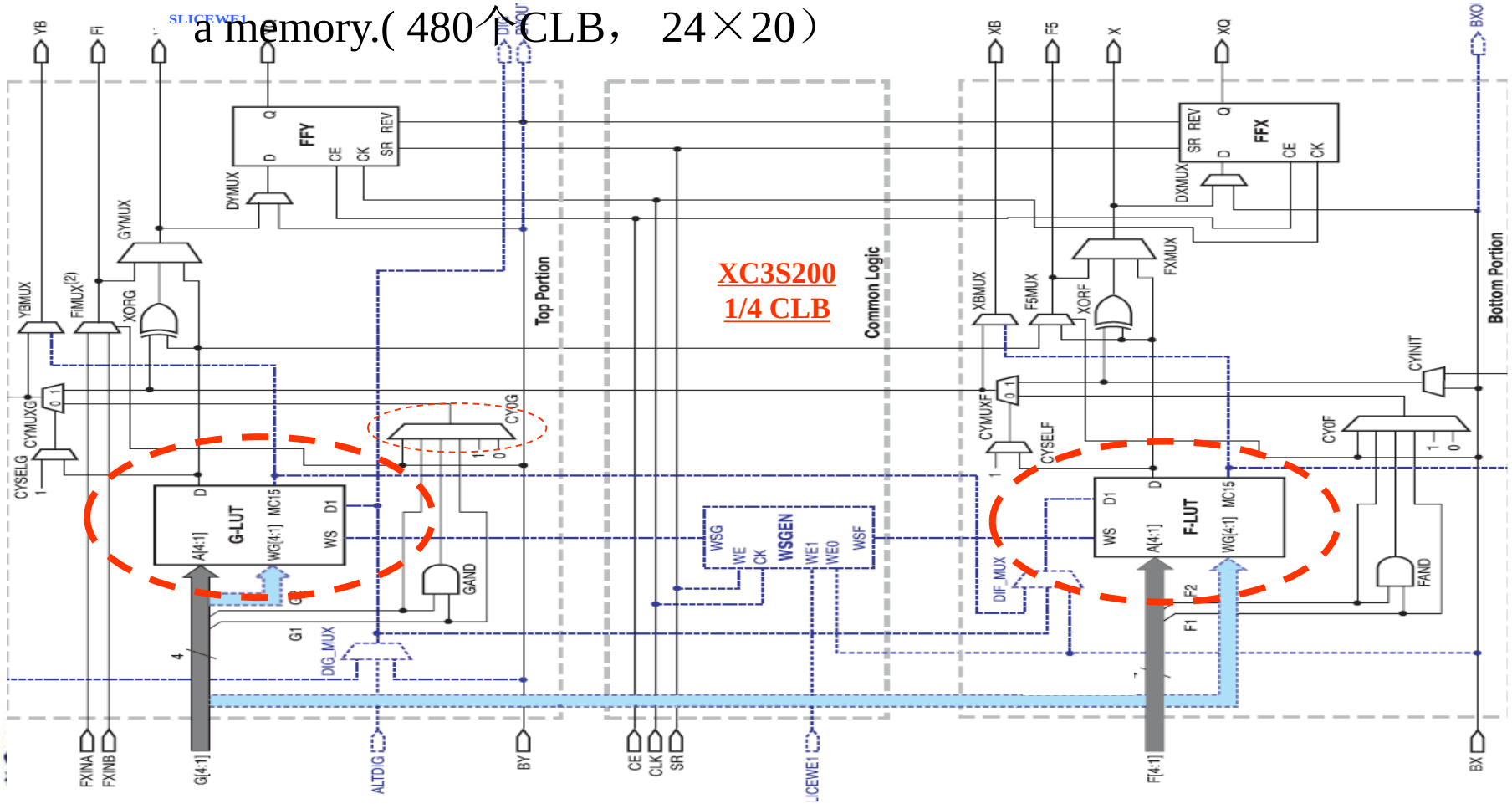
- **The internal structure is known as LCA (Logic Cell Array) is composed of three parts:**
  - Programmable logic block （CLB）
  - Programmable in-put output module (IOB)
  - Programmable internal connection （PIC）



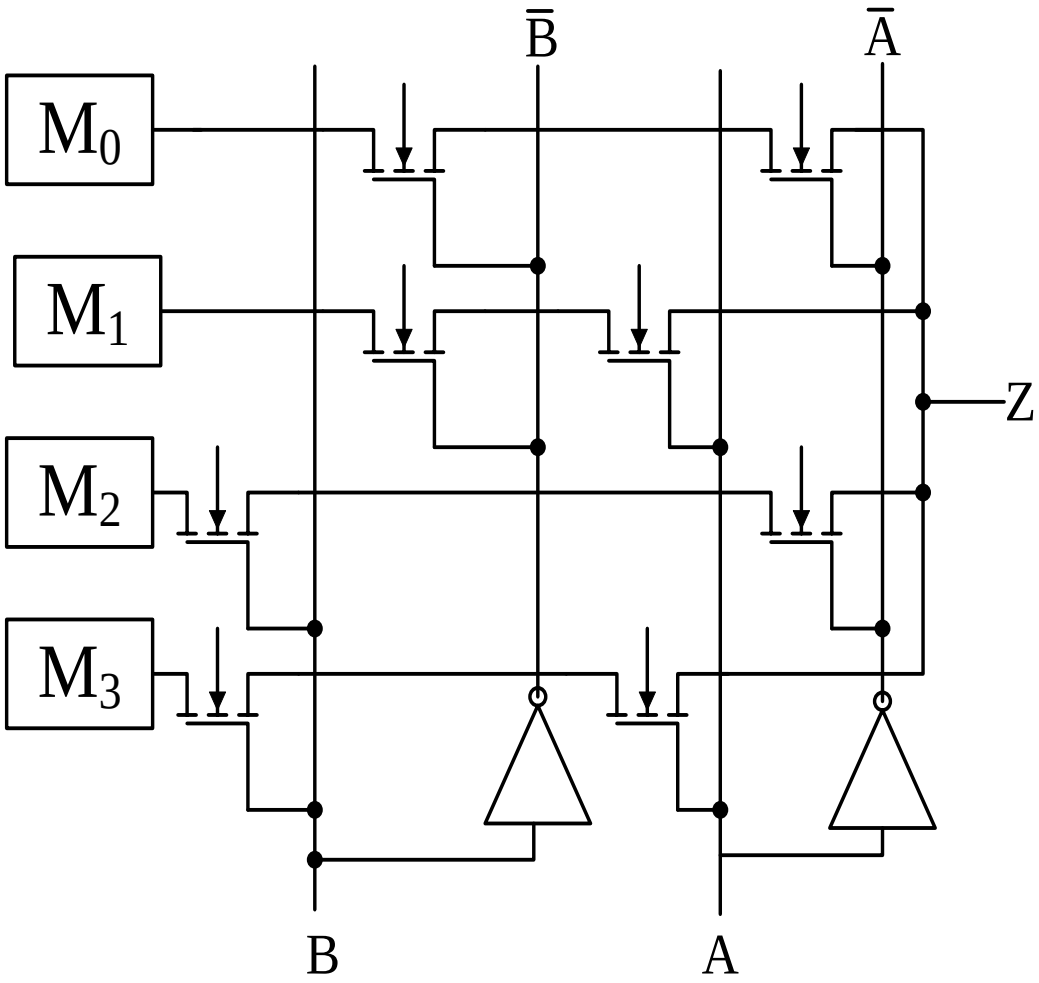
Configurable logic block

CLB（Configuration Logic Block）

- Is The Core of FPGA programmable universal logic function , composed of a RAM-based look-up table (LUT) may also be used as a memory.(480个CLB， 24×20）



Switch array look-up table Principle



Configuration Table

Store	B	A	Z
M_0	0	0	M_0
M_1	0	1	M_1
M_2	1	0	M_2
M_3	1	1	M_3

Combinational Implementation

For Programmable Methods

Combinational Logic Implementation -ROM

- ROM is constituted by the variable decoder and OR gates
- function truth table is stored in the ROM
- Boolean function can be implemented in software

Inputs			Outputs						Square for 3 bits
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

B0=A0

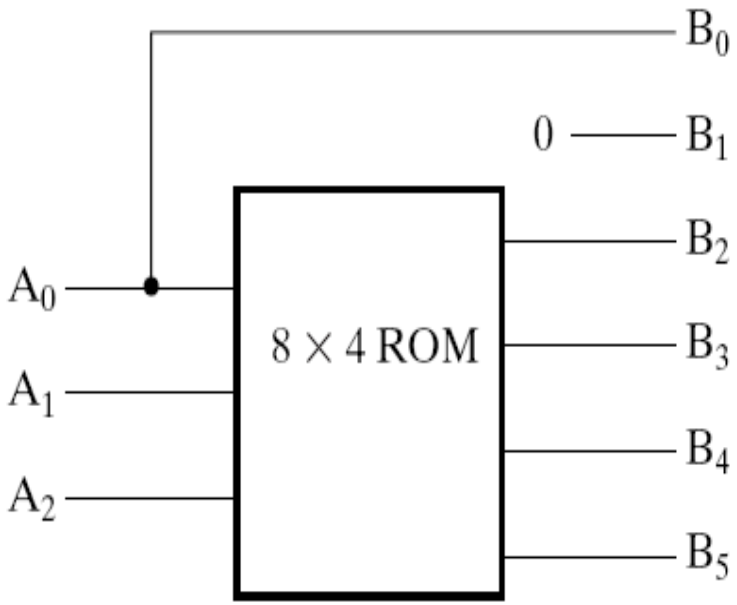
B1=”0”

ROM truth table

can select 8 × 4bit ROM

Function input corresponding to the ROM address

ROM output corresponding to the square value



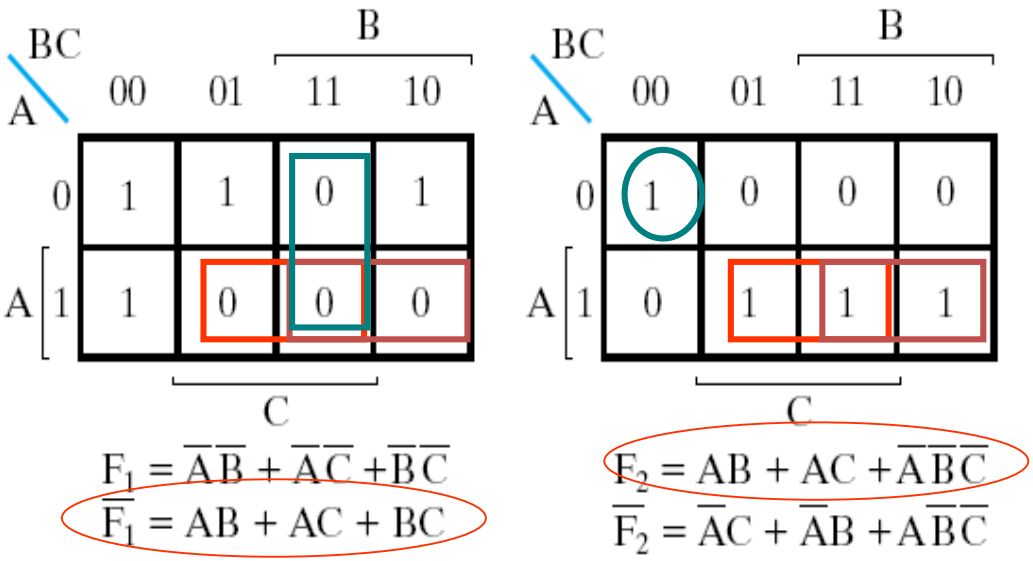
ROM真值表

A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

How to choose the ROM?

Combinational Logic Implementation-PLAs

- Implement functions
  F1(A,B,C)
  $=\Sigma m(0,1,2,4)$
  F2(A,B,C)
  $=\Sigma m(0,5,6,7)$
- Use of K-map simplification
- How the entries at least (shared)?
- Completion of the programming table



PLA programming table

	Product term	Inputs A	B	C	Outputs (C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$\overline{A}\,\overline{B}\,\overline{C}$	4	0	0	0	–	1

Combinational Logic Implementation-PALs

- **Implement functions :**

$W(A,B,C,D)=\Sigma m(1,12,13)$

$X(A,B,C,D)=\Sigma m(7,8,9,10,11,12,13,14,15)$

$Y(A,B,C,D)=\Sigma m(0,2,3,4,5,6,7,8,9,10,11,15)$

$Z(A,B,C,D)=\Sigma m(1,2,8,12,13)$

simplification follows :

$$W=AB\overline{C}+\overline{A}\overline{B}C\overline{D}$$

$$X=A+BCD$$

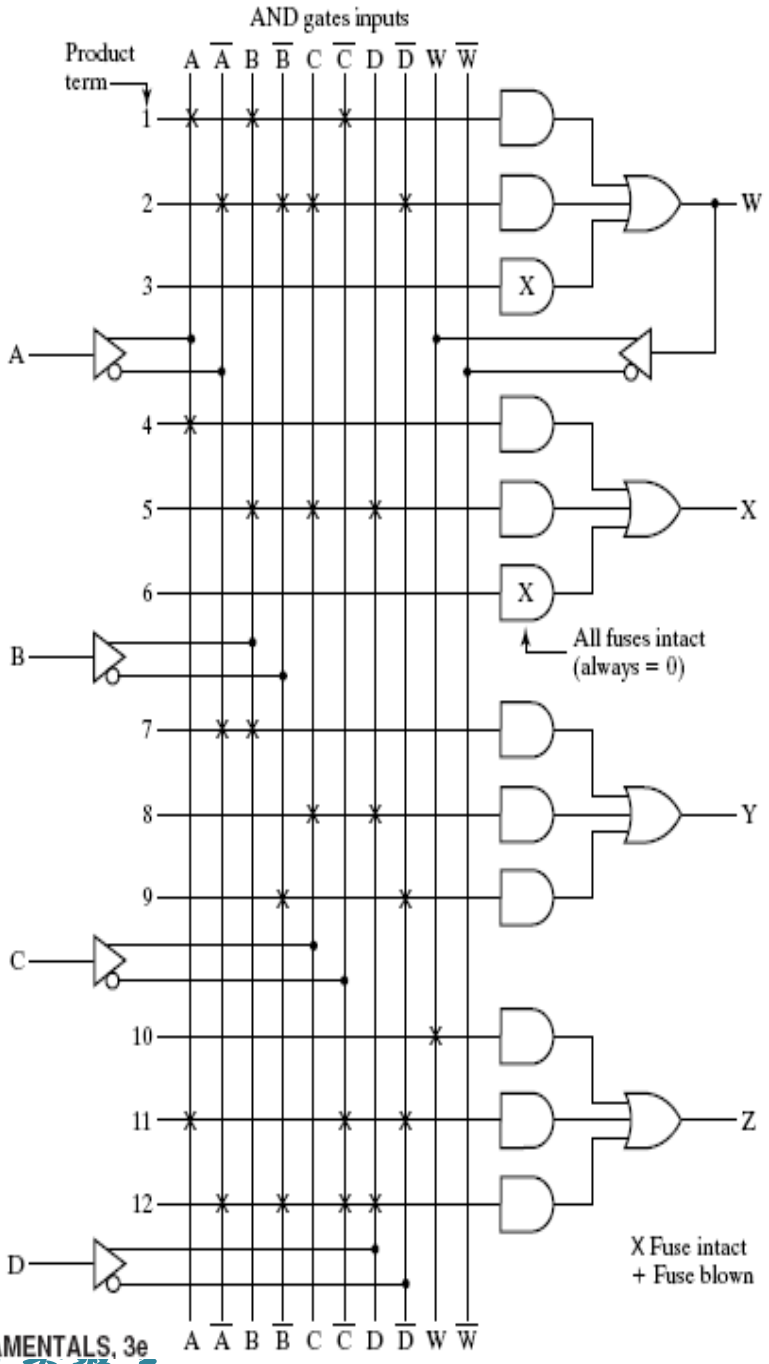
$$Y=\overline{A}B+CD+\overline{B}\overline{D}$$

$$Z=AB\overline{C}+\overline{A}\overline{B}C\overline{D}+A\overline{C}\overline{D}+\overline{A}BC\overline{D}$$

$$=W+A\overline{C}\overline{D}+\overline{A}BC\overline{D}$$

PAL Implement connection

Product term	AND Inputs A	B	C	D	W	Outputs
1	1	1	0	—	—	$W = A B \overline{C}$
2	0	0	1	0	—	$+\overline{A}\,\overline{B} C \overline{D}$
3	—	—	—	—	—	
4	1	—	—	—	—	$X = A$
5	—	1	1	1	—	$+BCD$
6	—	—	—	—	—	
7	0	1	—	—	—	$Y = \overline{A} B$
8	—	—	1	1	—	$+CD$
9	—	0	—	0	—	$+\overline{B}\,\overline{D}$
10	—	—	—	—	1	$Z = W$
11	1	—	0	0	—	$+A \overline{C}\,\overline{D}$
12	0	0	0	1	—	$+\overline{A}\,\overline{B}\,\overline{C} D$



第三章作业2

Ch3

page187-192：3-16，3-27，3-28，3-29，
3-37，3-44，3-47

More and more, It is simple also!


Thank you!