Logic and Computer Design Fundamentals

Chapter 5 – Digital Hardware Implementation

Part 2 – Programmable Implementation Technologies

Ming Cai
cm@zju.edu.cn
College of Computer Science and Technology
Zhejiang University

Overview

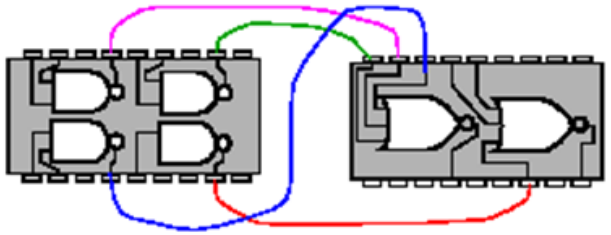
- **Part 1 – The Design Space**
- **Part 2 – Programmable Implementation Technologies**
  - **Why Programmable Logic?**
  - **Programming Technologies**
  - **Read-Only Memories (ROMs)**
  - **Programmable Array Logic (PALs)**
  - **Programmable Logic Arrays (PLAs)**
  - **Field Programmable Gate Array (FPGAs)**

Constructing Digital Circuits

Hand Wired Circuits

Cirri 1970-85

- Make 2 to 4 silicon gates in a package.
- Connect with wires.

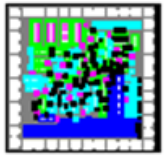


VLSI circuits

Start with a silicon wafer and make:

- the gates
- the interconnections on top

both made together.

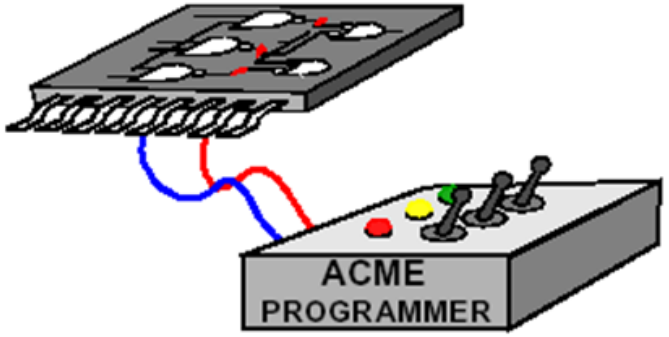


Field Programmable circuits

Start with a silicon wafer and make:

- gates with no connections.
- Make connections later using:

1) electrical means
   - blow fuses, grow anti fuses
   - use memory to hold connections
2) deposit metal lines on top of silicon.



Why Programmable Logic?

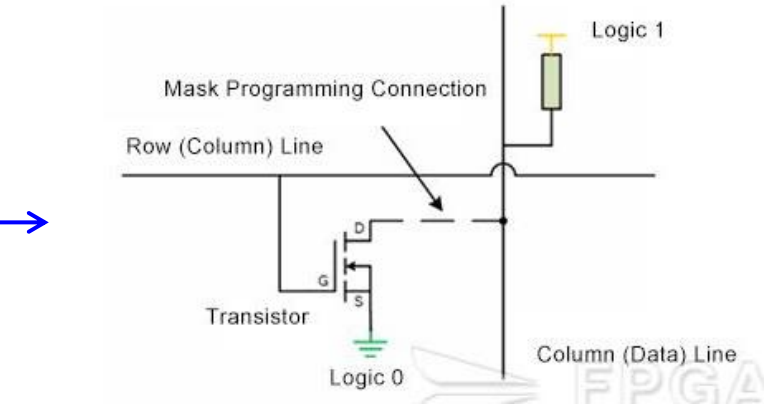
- **Facts:**
  - **It is most economical to produce an IC in large volumes**
  - **Many designs required only small volumes of ICs**
- **Need an IC that can be:**
  - **Produced in large volumes**
  - **Handle many designs required in small volumes**
- **A programmable logic part can be:**
  - **made in large volumes**
  - **programmed to implement large numbers of different low-volume designs**

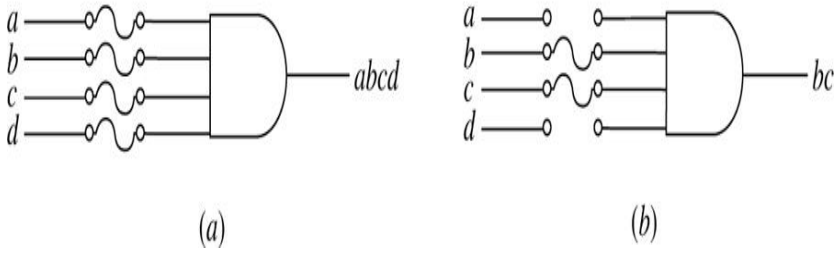
Programmable Logic - More Advantages

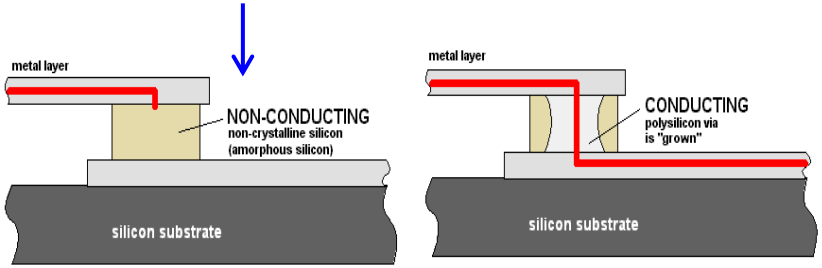
- **Many programmable logic devices are *field-programmable*, which can be programmed outside of the manufacturing environment.**
- **Most programmable logic devices are *erasable* and *reprogrammable*.**
  - **Allows "updating" a device or correction of errors**
  - **Allows reuse the device for a different design - the ultimate in re-usability!**
  - **Ideal for course laboratories**
- **Programmable logic devices can be used to prototype design that will be implemented for sale in regular ICs.**
  - **Complete Intel Pentium designs were actually prototyped with specialized systems based on large numbers of VLSI programmable devices!**

Programming Technologies

- **Three types of programming technologies:**
  - **Control connections**
  - **Control transistor switching**
  - **Build lookup tables**
- **Control connections**
  - **Mask programming**
  - **Fuse**
  - **Antifuse**

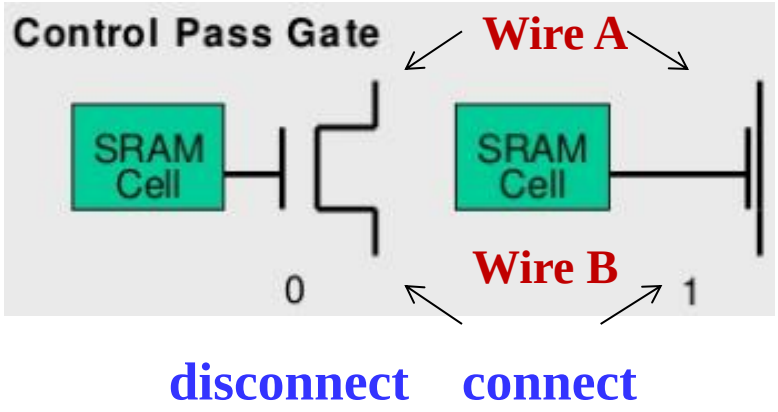


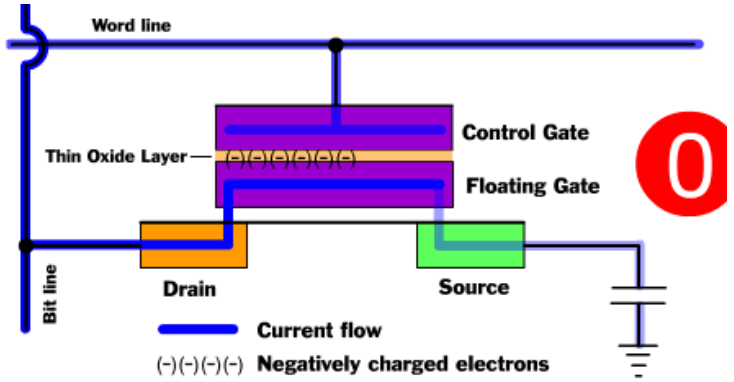




Programming Technologies (continued)

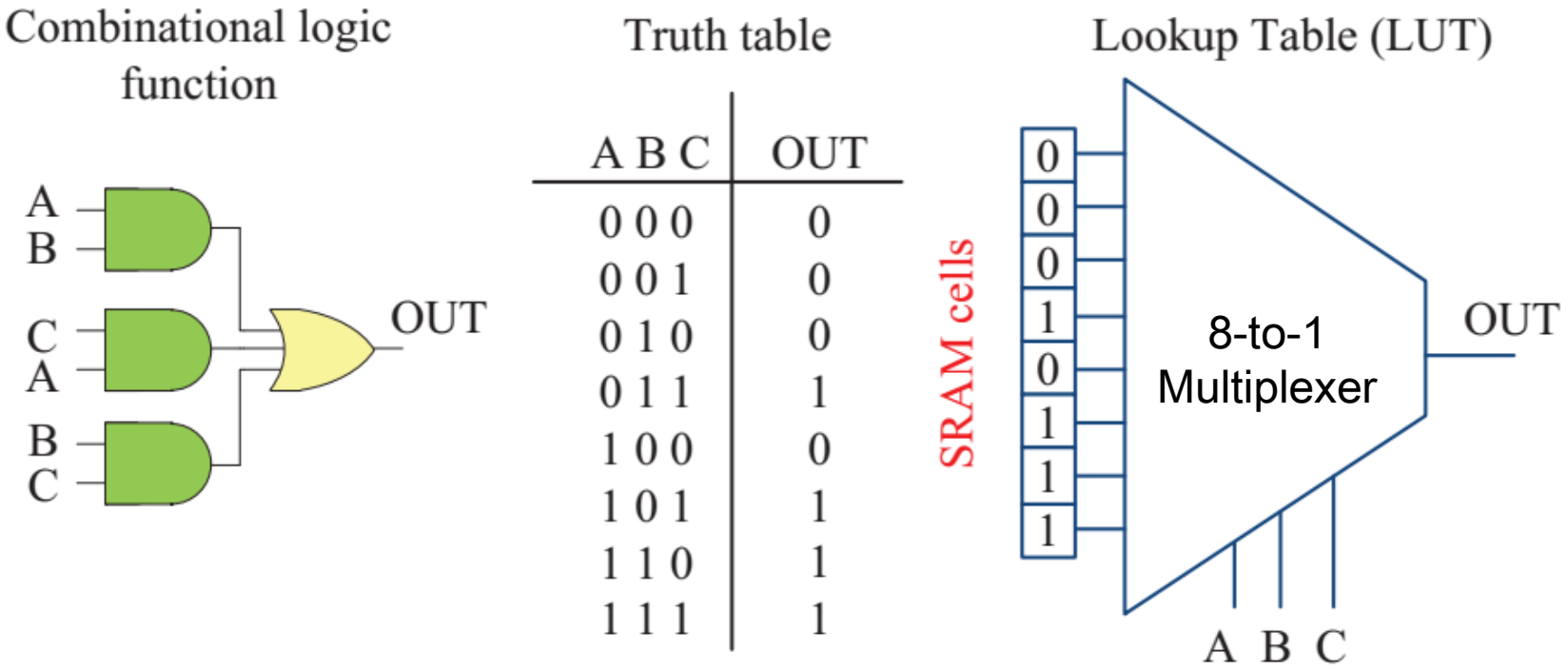
- **Control transistor switching**
  - **Single-bit storage element**
  - **Stored charge on a floating gate**
    - **Erasable**
    - **Electrically erasable**
    - **Flash (as in Flash Memory)**





Programming Technologies (continued)

- **Build lookup tables (LUT)**
  - **Storage elements for the function**

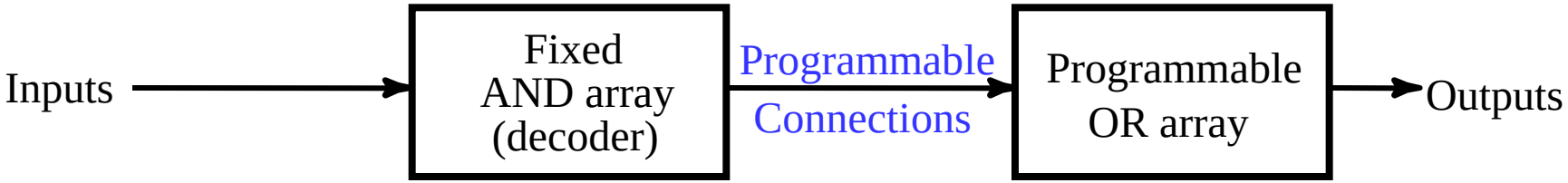


Example of a 3-input LUT implementing a majority voter

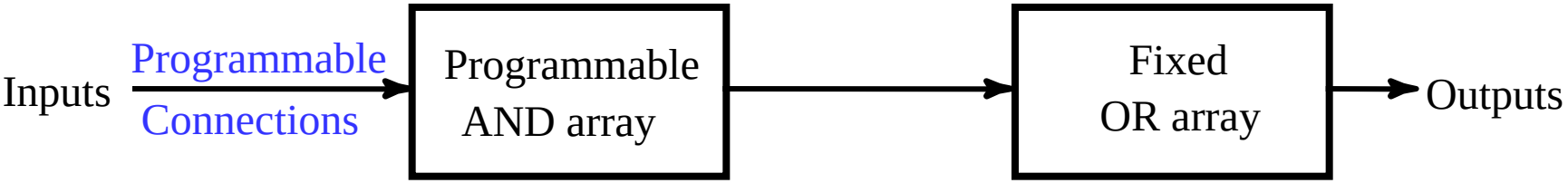
Programmable Logic Device

- ***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**

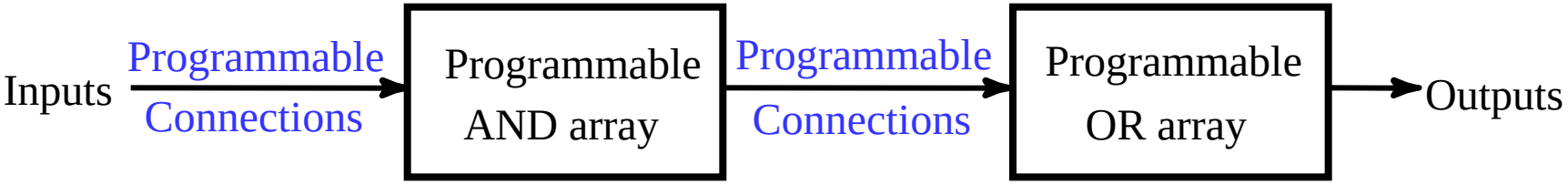
ROM, PAL and PLA Configurations



(a) Programmable read-only memory (PROM) e.g., $F = \overline{A}\,\overline{B} + \overline{A}\,B + A\,\overline{B} + A\,B$

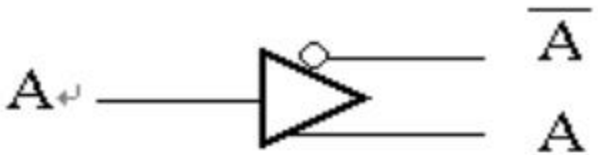


(b) Programmable array logic (PAL®) device e.g., $F = \overline{A}\,\overline{B} + \overline{A}\,C + A\,B\,C$



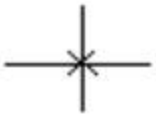
(c) Programmable logic array (PLA) device e.g., $F = \overline{A}\,\overline{B} + \overline{A}\,B\,C + A\,B\,C$

Logical symbols



Buffer

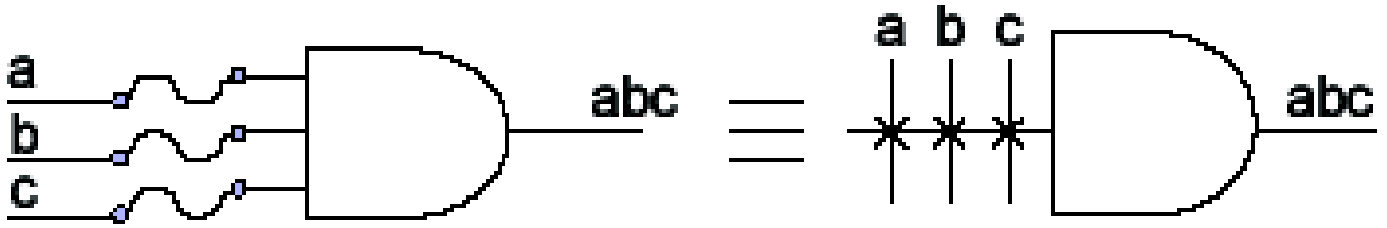
wire connecting
(not programmable)



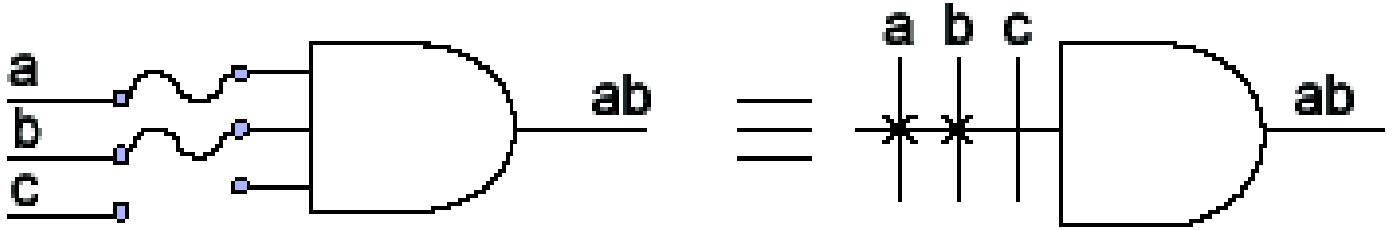
wire connecting
(programmable)

Disconnection
(erasable)

Logical symbols (continued)

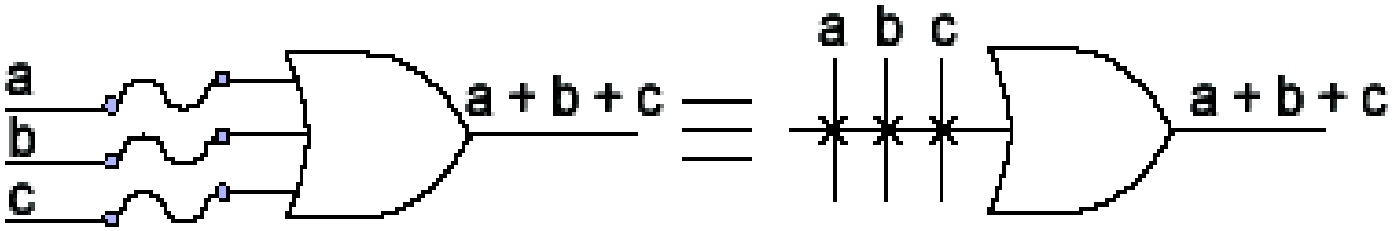


AND gate before programming

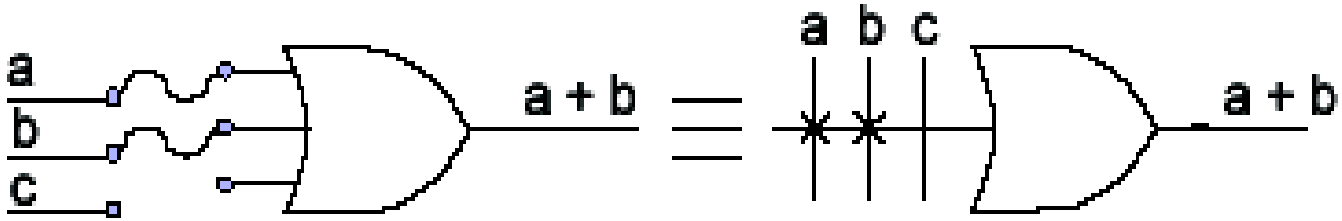


AND gate after programming

Logical symbols (continued)

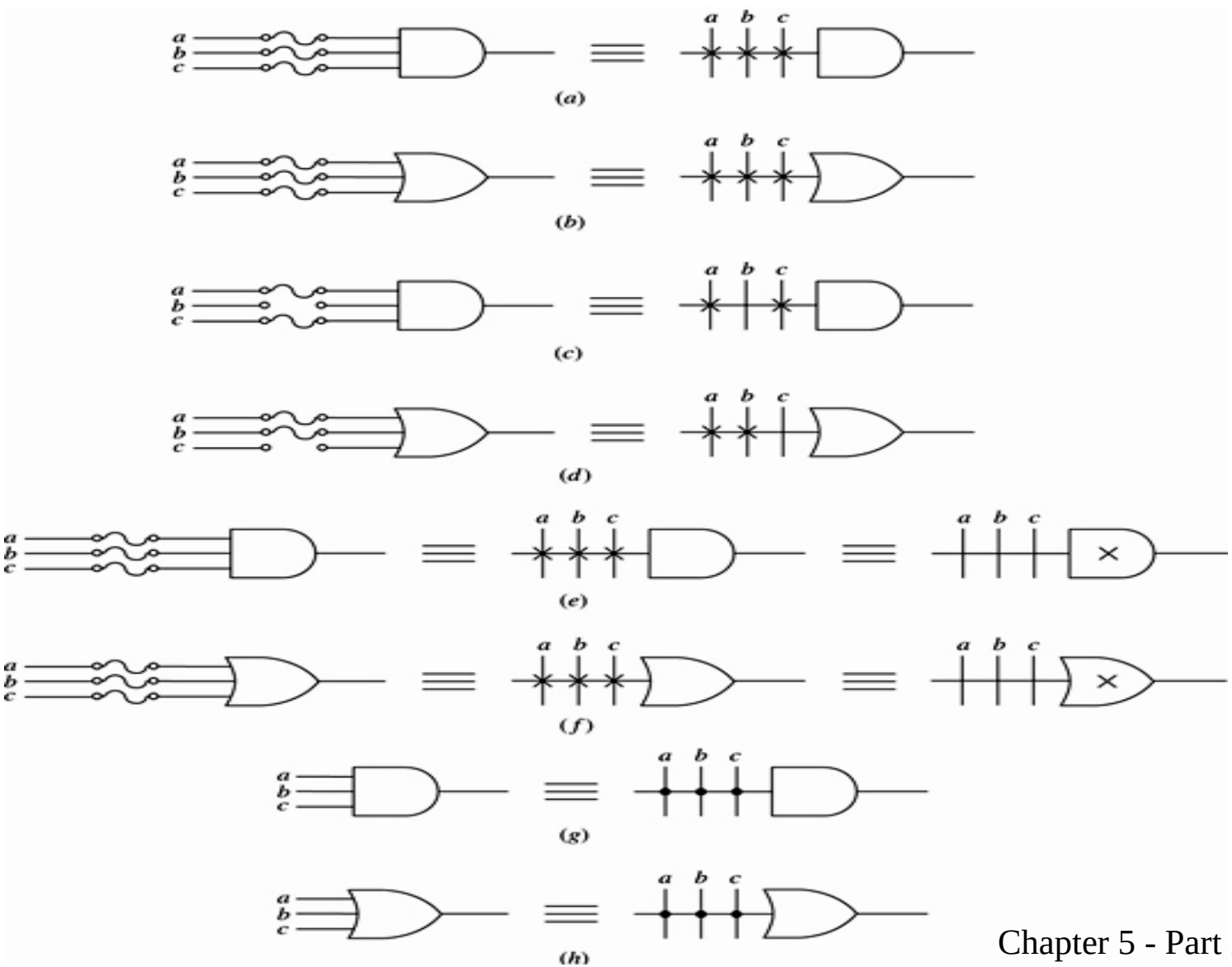


OR gate before programming



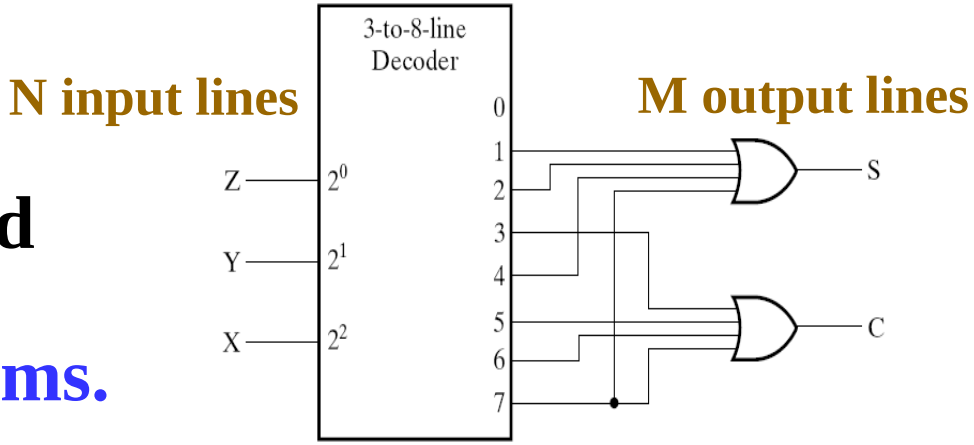
OR gate after programming

Logical symbols (continued)



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.**

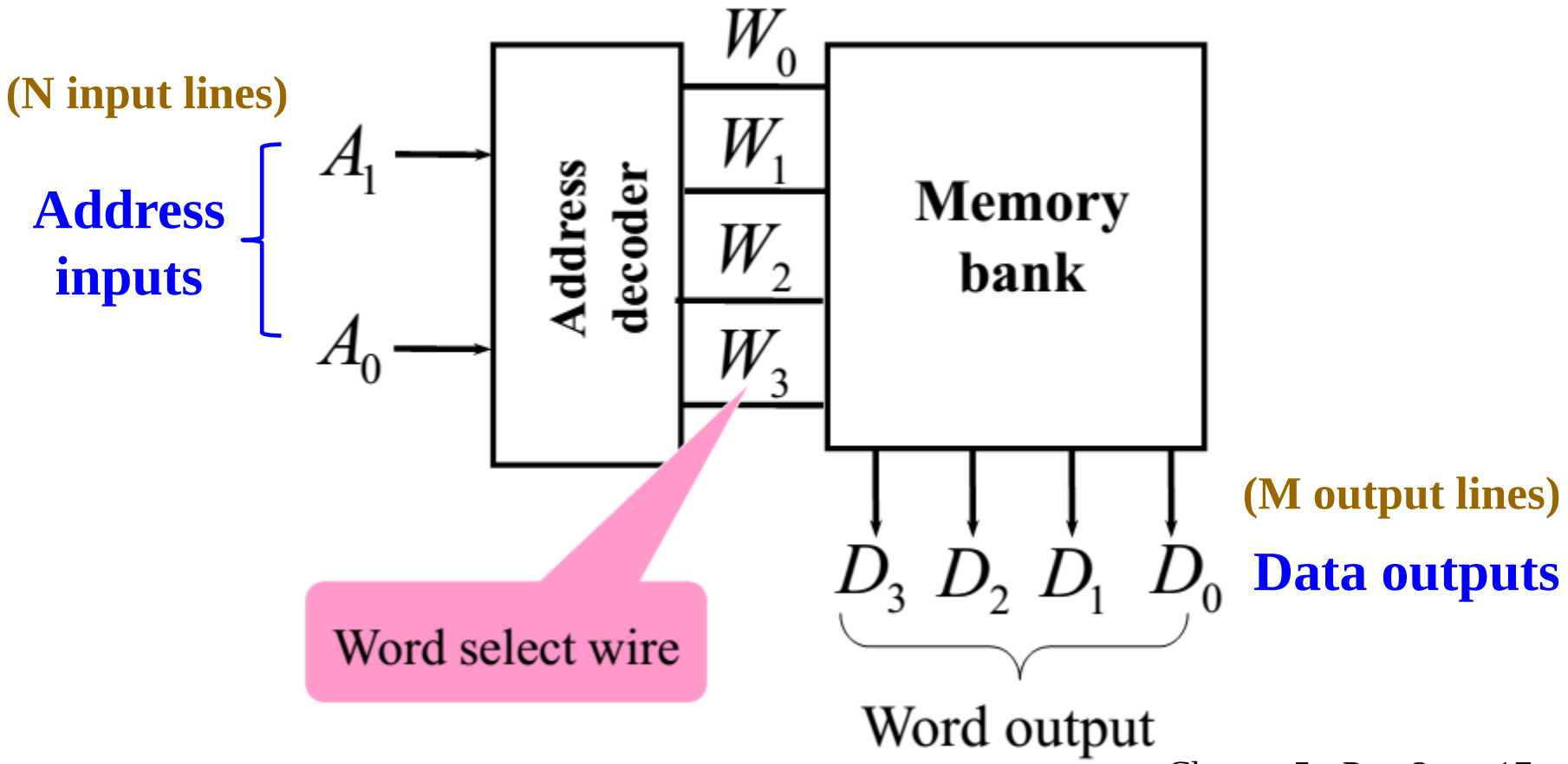
Read Only Memory

- **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 ROM

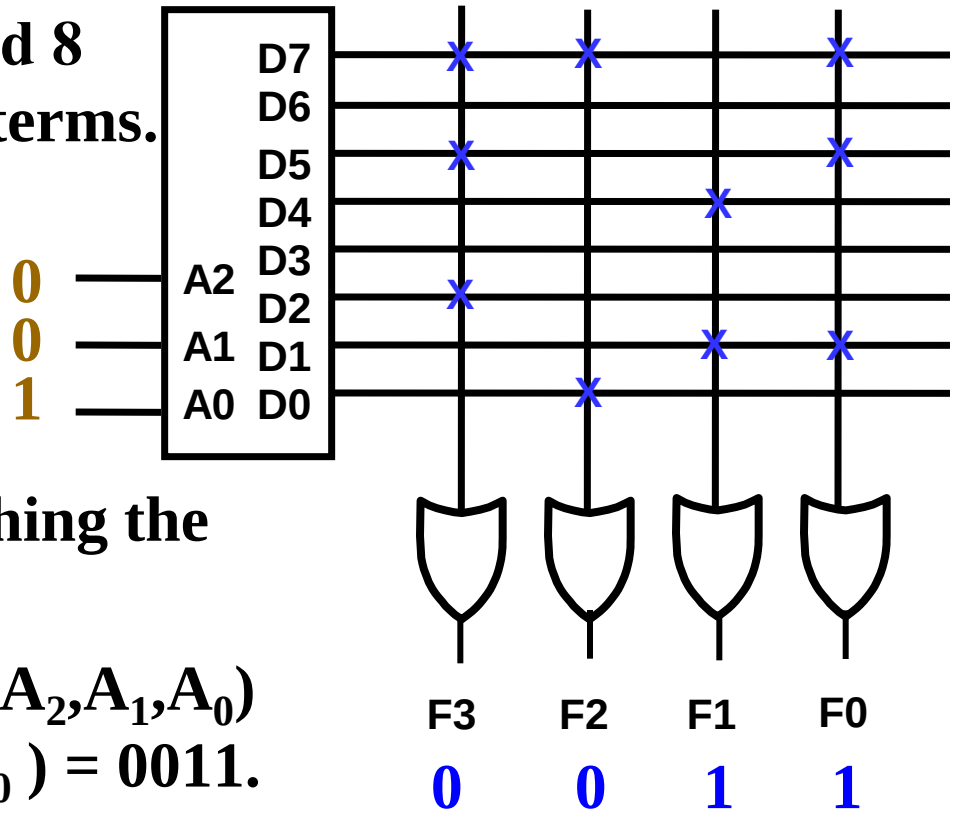
- **ROM size = address width × word width**

  = 2^2 × 4 = 16 bit



Read Only Memory Example

- Example: An 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) = 001, 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)?



Example: Square of 3-bit input number

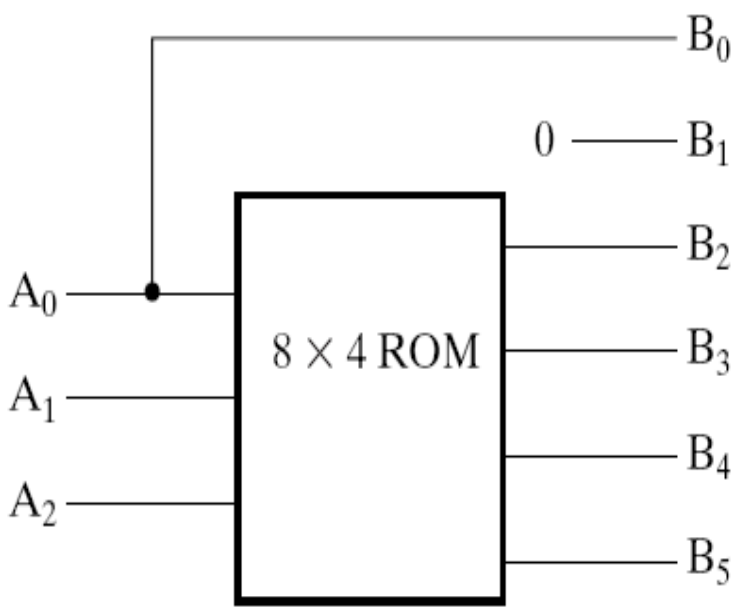
Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

- **B[5:0] = (A[2:0]) ^{2}**
- **How to choose ROM?**
  - **$2^3 \times$ 4 bit ROM: address A[2:0], data B[5:0]**

$B_1 = 0$ $B_0 = A_0$

Example: Square of 3-bit input number

- $2^3 \times 4$ bit ROM are selected



ROM Truth Table

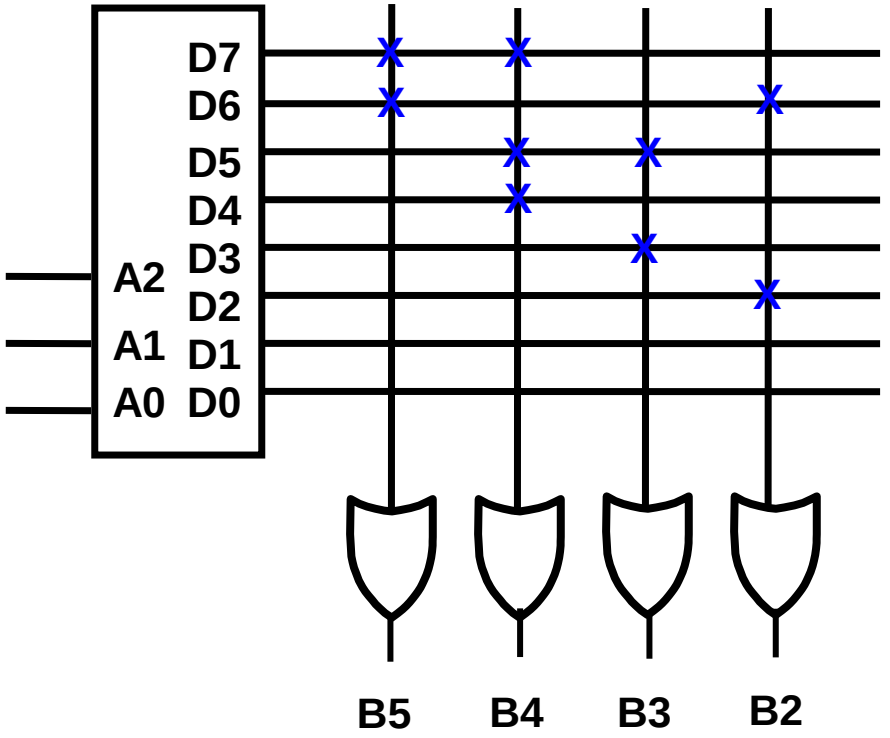
A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

- Are B_0 and B_1 reprogrammable?

Example: Square of 3-bit input number

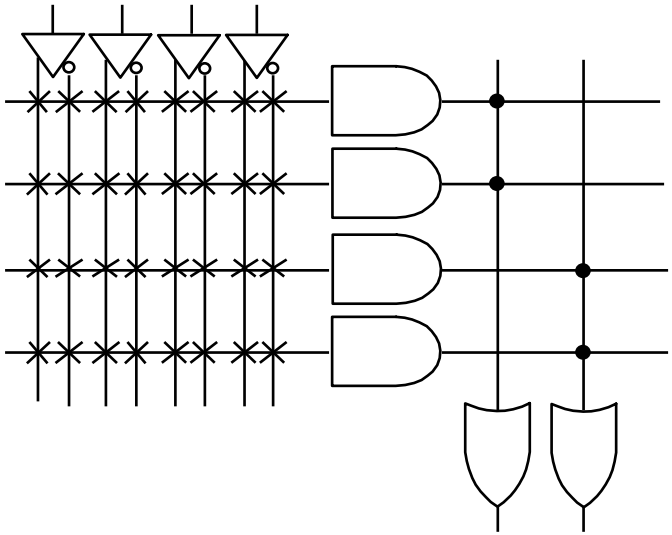
- $2^3 \times 4$ bit ROM

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



Programmable Array Logic (PAL)

- **The PAL is the opposite of the ROM, having a programmable set of ANDs combined with fixed ORs.**
- **PAL does not provide full decoding of the variables and generates part of the minterms.**



- **The decoder is replaced by an array of AND gates that can be programmed to generate product terms of the input variables.**
- **The product terms are then selectively connected to OR gates to provide the sum of products for the required Boolean functions.**

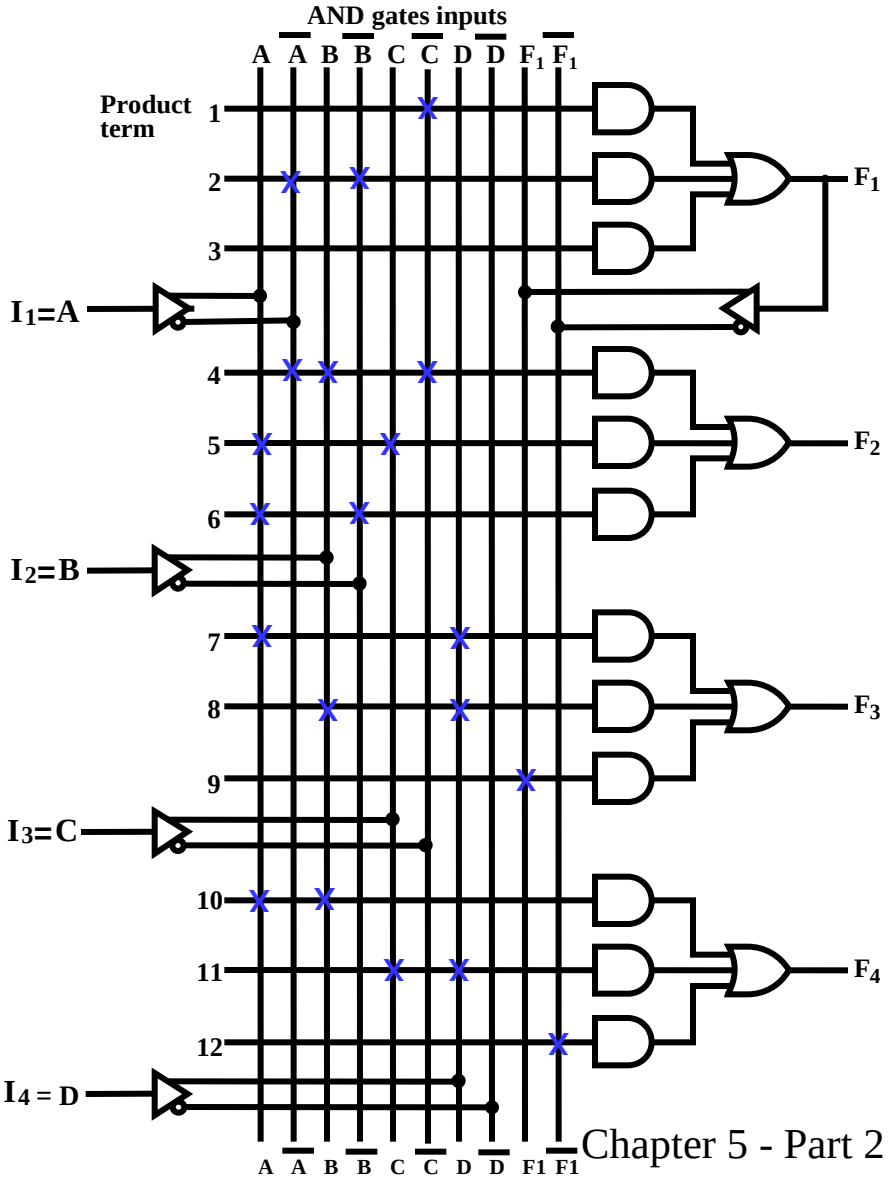
Programmable Array Logic (PAL) (continued)

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

$$F1 = \bar{A}B + \bar{C}$$
$$F2 = \bar{A}B\bar{C} + AC + A\bar{B}$$
$$F3 = AD + BD + \overline{F1}$$
$$F4 = AB + CD + \overline{F1}$$



Programmable Array Logic

- **Design requires fitting functions within the limited number of ANDs per OR gate**
- **Single function optimization is the first step to fitting**
- **Otherwise, if the number of terms in a function is greater than the number of ANDs per OR gate, then factoring is necessary**

Programmable Array Logic Example

- **Equations:** $F1 = A\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}\,\overline{B}C + ABC$

  $F2 = AB + BC + AC$

- **F1 has four terms (>3)**
- **Factor out last two terms as W**

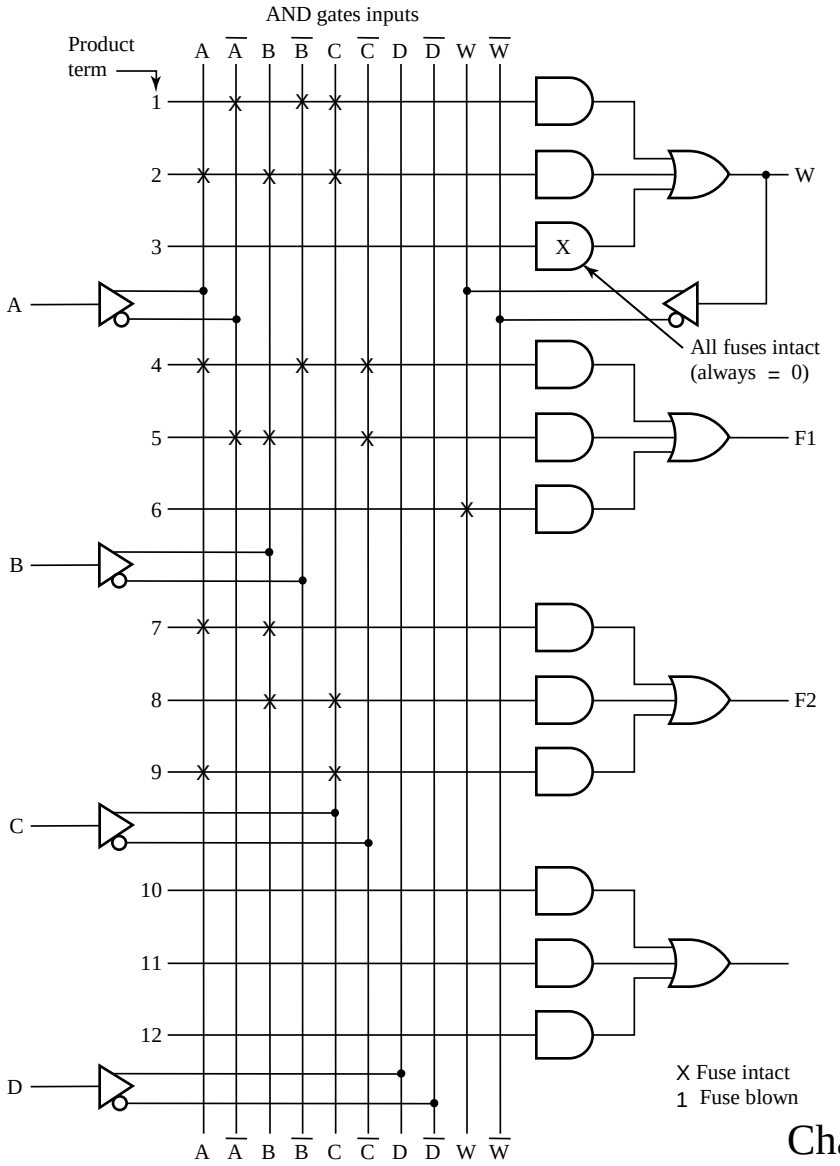
Product term	AND Inputs A	B	C	D	W	Outputs
1	0	0	1	—	—	$W = \overline{A}BC + AB($
2	1	1	1	—	—	
3	—	—	—	—	—	
4	1	0	0	—	—	$F1 = X =$
5	0	1	0	—	—	$A\overline{B}\overline{C} + \overline{A}B\overline{C} + W$
6	—	—	—	—	1	
7	1	1	—	—	—	$F2 = Y =$
8	—	1	1	—	—	$AB + B(\overline{C}) + A($
9	1	—	1	—	—	
10	—	—	—	—	—	
11	—	—	—	—	—	
12	—	—	—	—	—	

Programmable Array Logic Example

$W = \bar{A}BC + AB\bar{C}$

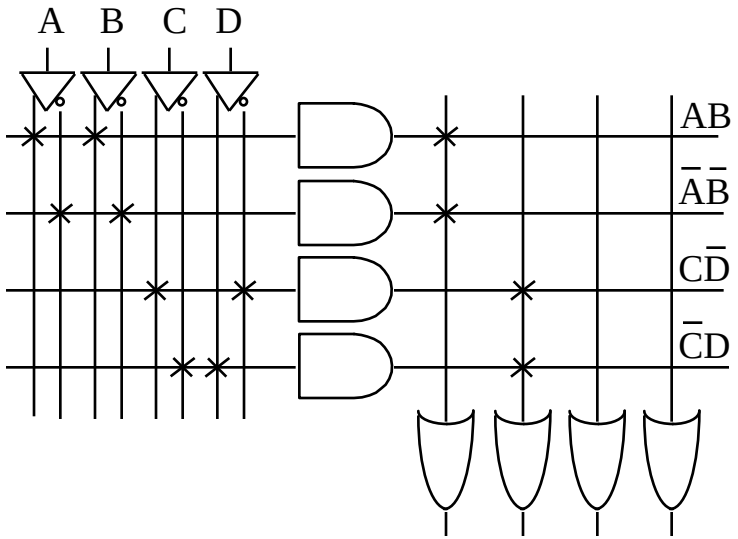
$F1 = X = A\bar{B}C + \bar{A}BC + W$

$F2 = Y = AB\bar{C} + \bar{B}C + A\bar{C}$



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.**

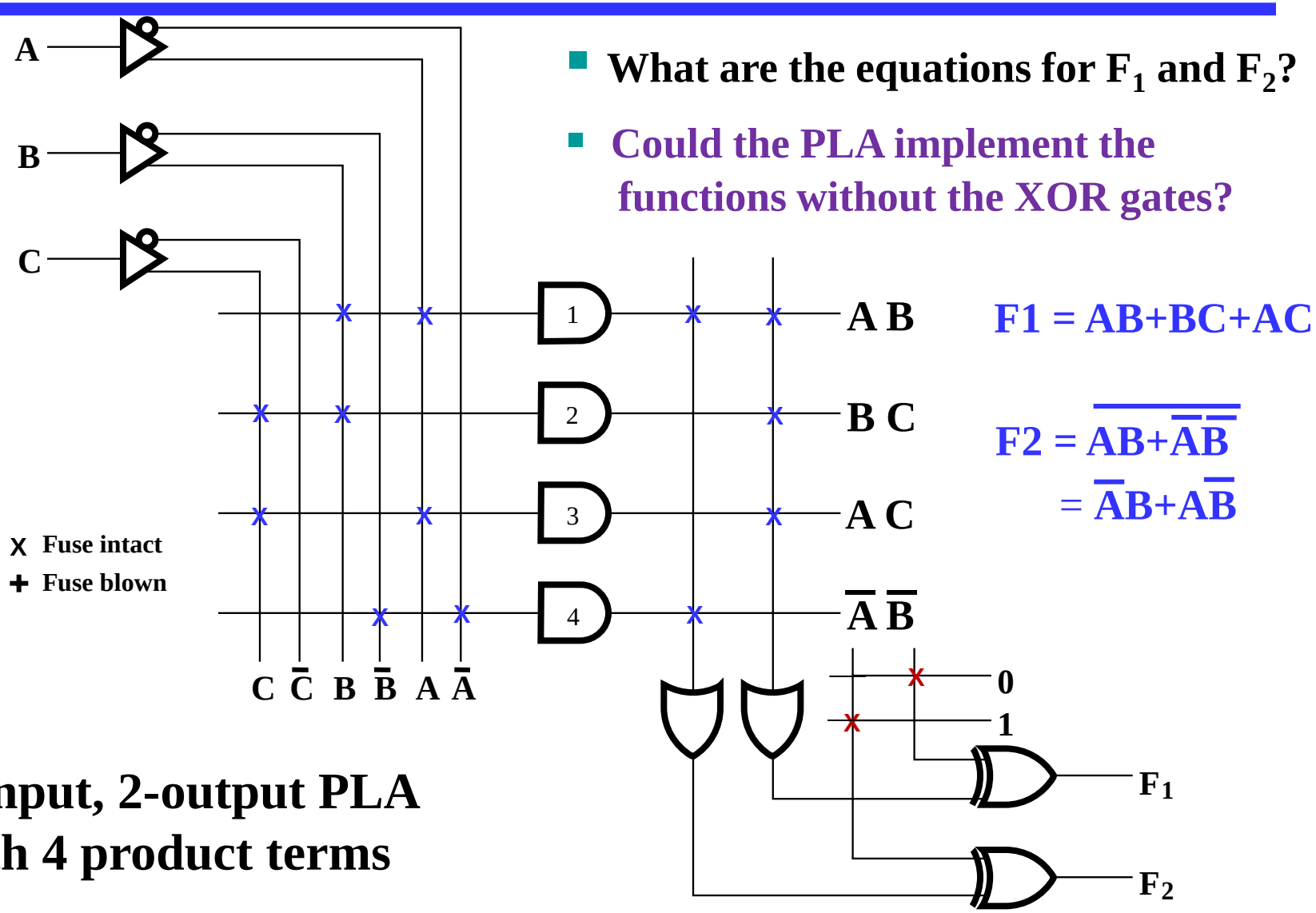


- **Advantages**
  - **A PLA can have large N and M permitting implementation of equations that are impractical for a ROM (because of the number of inputs, N, required)**
  - **A PLA has all of its product terms connectable to all outputs, overcoming the problem of the limited inputs to the PAL ORs**
  - **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



- What are the equations for F_1 and F_2?
- Could the PLA implement the functions without the XOR gates?

$F1 = AB+BC+AC$

$F2 = \overline{AB+\overline{A}\overline{B}}$

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

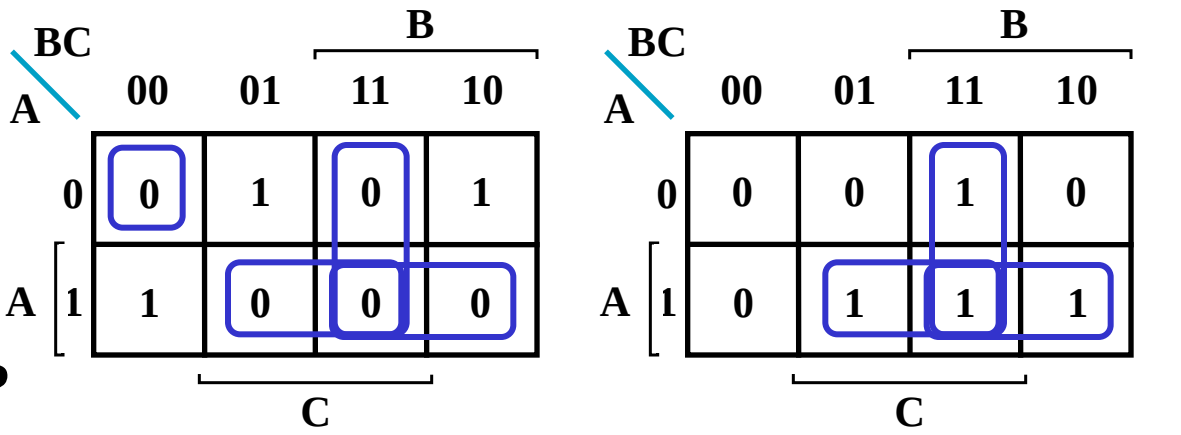
- 3-input, 2-output PLA with 4 product terms

Programmable Logic Array

- The set of functions to be implemented must fit the available number of product terms
- The number of literals per term is less important in fitting
- The best approach to fitting is multiple-output, two-level optimization (which has not been discussed)
- Since output inversion is available, terms can implement either a function or its complement
- For small circuits, K-maps can be used to visualize product term sharing and use of complements
- For larger circuits, software is used to do the optimization including use of complemented functions

Programmable Logic Array Example

- K-map specification
- How can this be implemented with four terms?
- Complete the programming table



$F_1 = \overline{A}\,\overline{B}C + \overline{A}B\,\overline{C} + A\,\overline{B}\,\overline{C}$

$\overline{F}_1 = AB + AC + BC + \overline{A}\,\overline{B}\,\overline{C}$

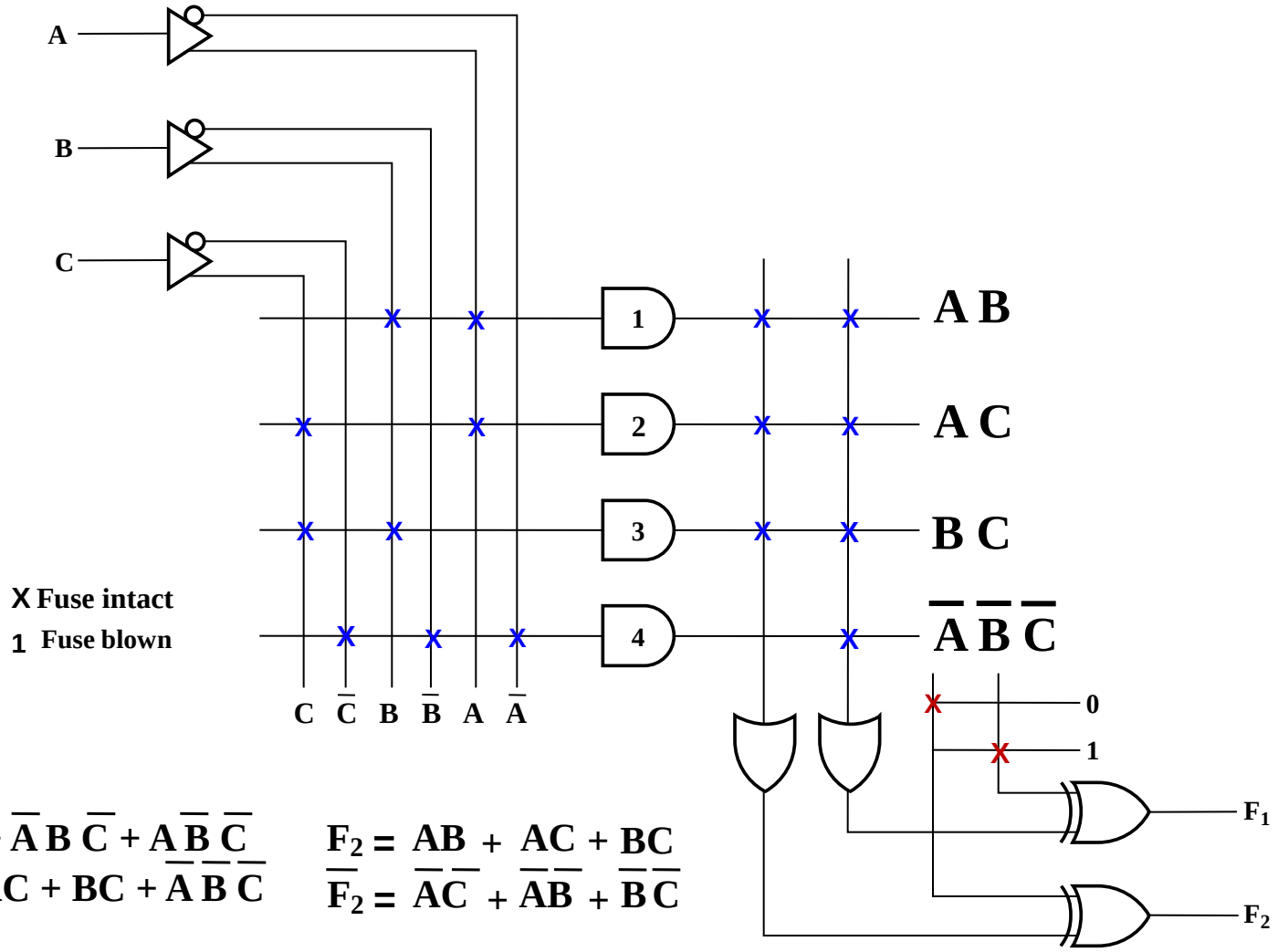
$F_2 = AB + AC + BC$

$\overline{F}_2 = \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$

PLA programming table

	Product term	Inputs A	B	C	Outputs (C) $\overline{F}_1$	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	1
$\overline{A}\overline{B}\overline{C}$	4	0	0	0	1	–

Programmable Logic Array Example



$$F_1 = \overline{A}\,\overline{B}C + \overline{A}B\,\overline{C} + A\overline{B}\,\overline{C}$$

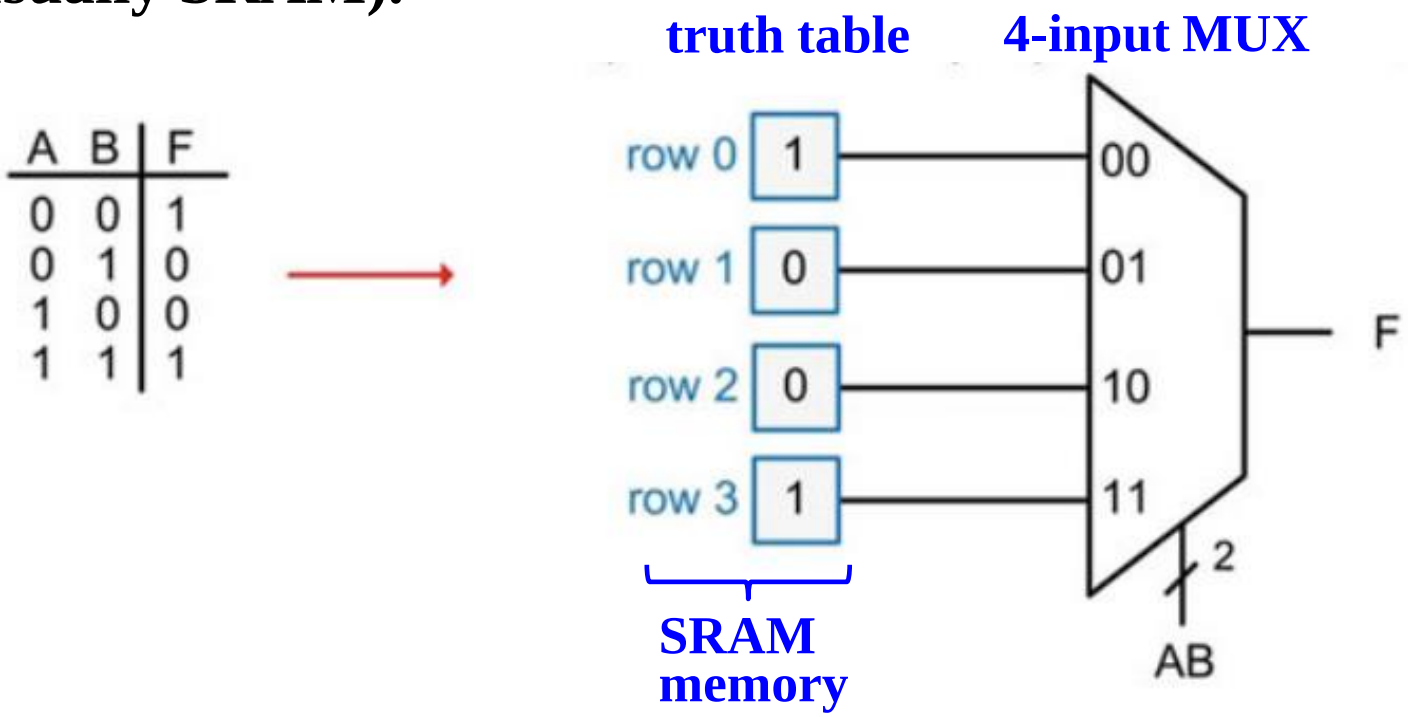
$$\overline{F_1} = AB + AC + BC + \overline{A}\,\overline{B}\,\overline{C}$$

$$F_2 = AB + AC + BC$$

$$\overline{F_2} = \overline{A}\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

Lookup Tables

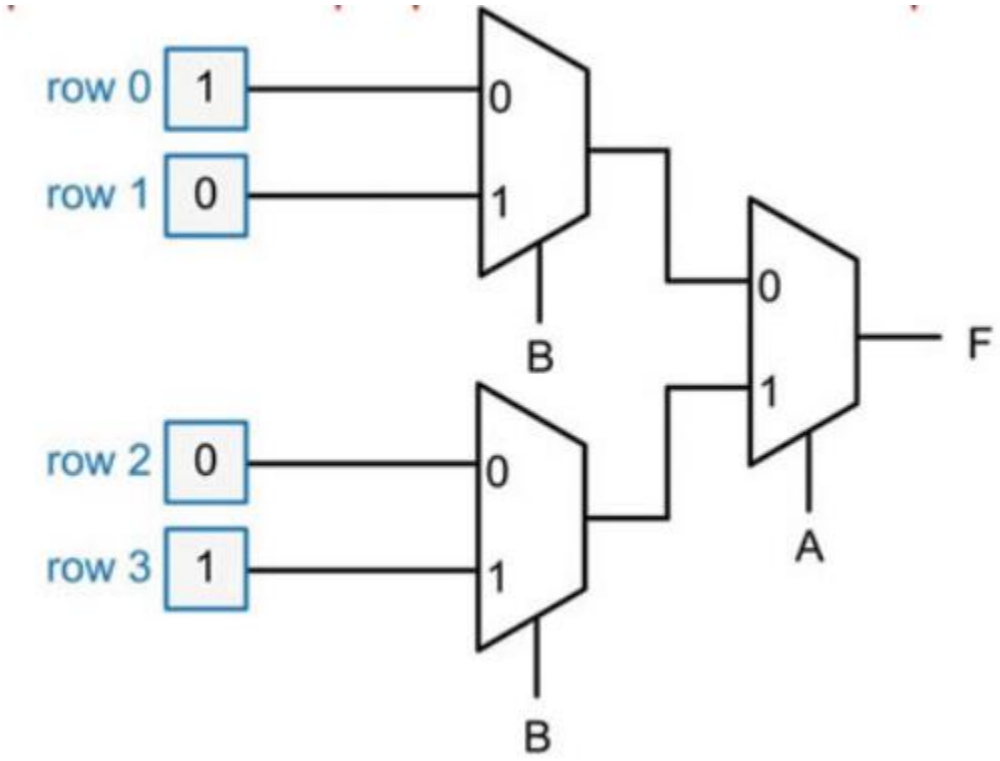
- **Lookup tables (LUTs) are used for implementing logic in Field-Programmable Gate Arrays (FPGAs).**
- **Lookup tables implement truth table in small memories (usually SRAM).**



2-input LUT implemented with a 4-input multiplexer

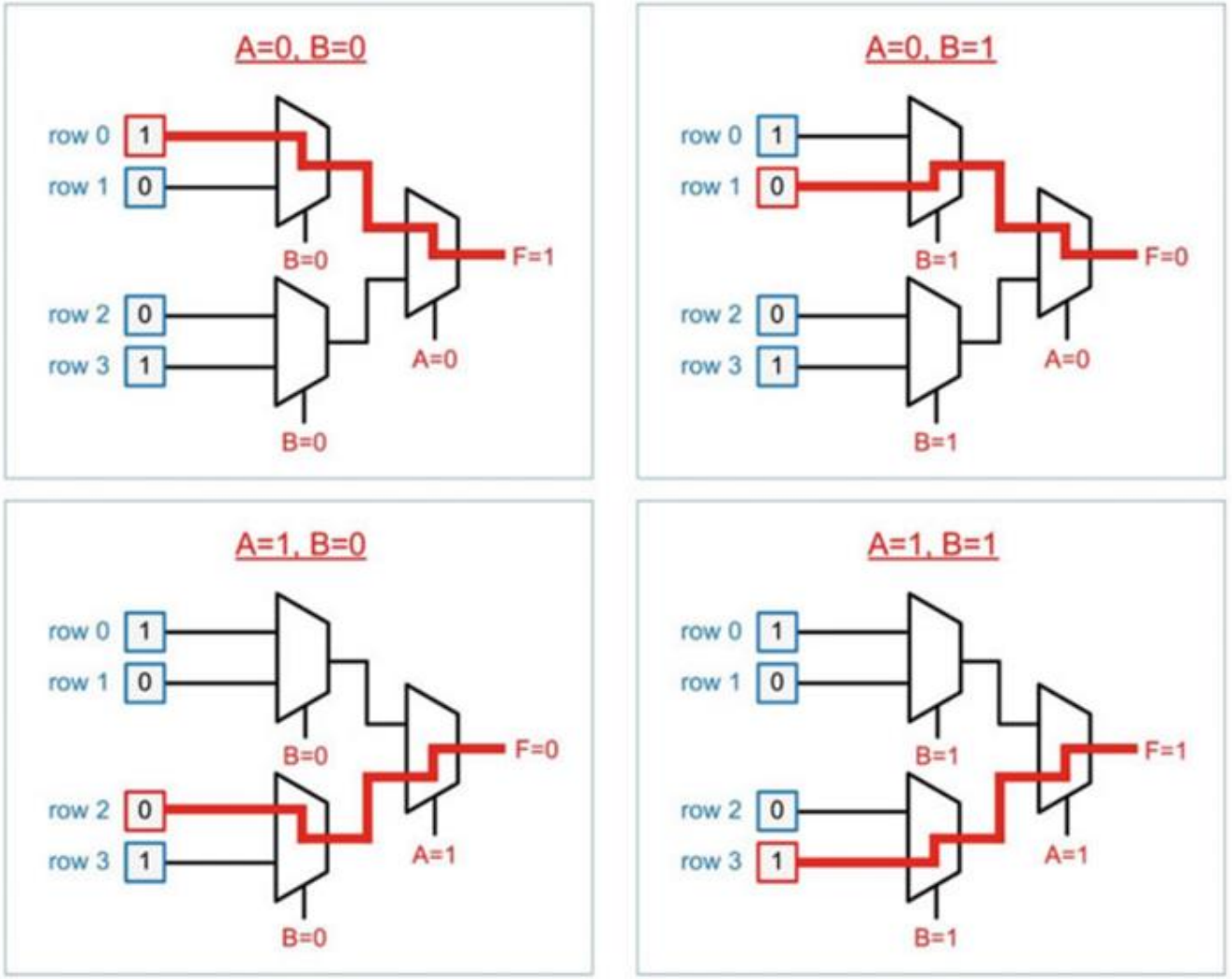
Lookup Tables (continued)

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



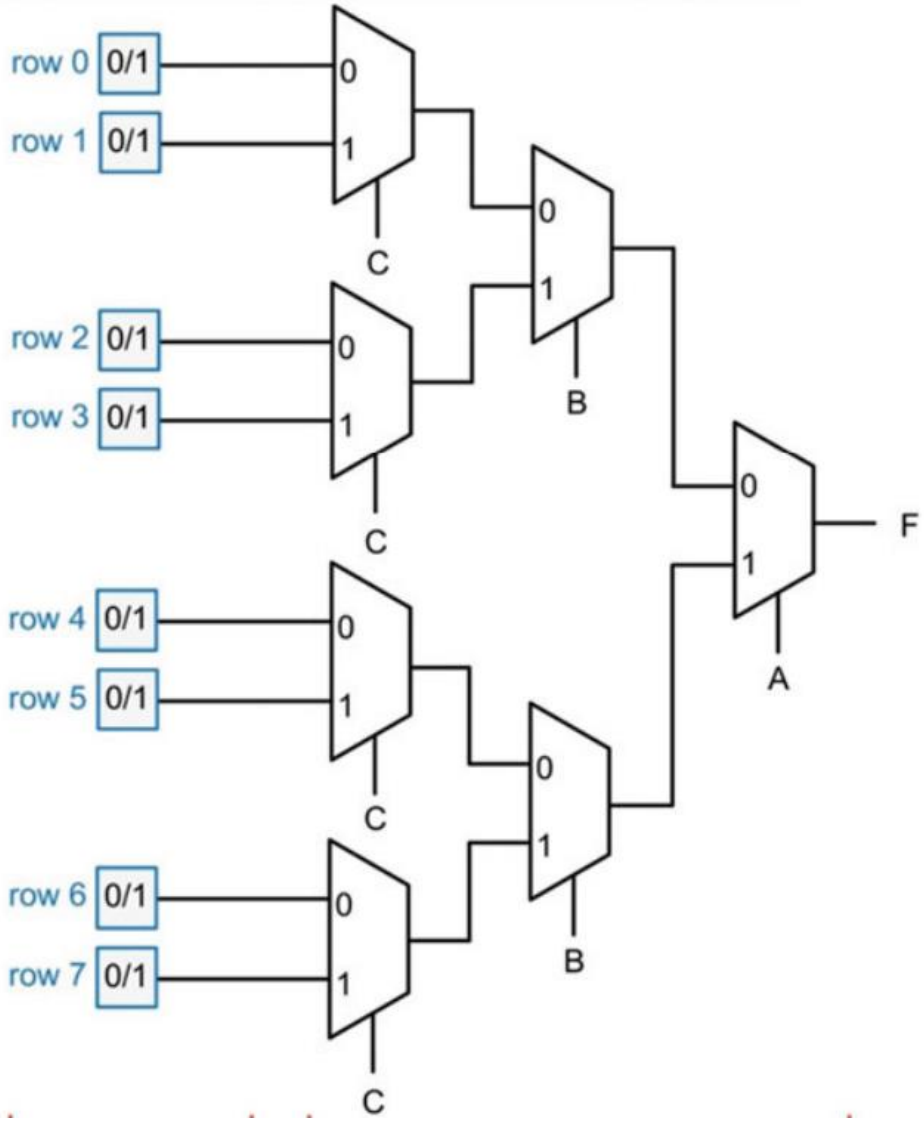
2-input LUT implemented with a two-level cascade of 2-input multiplexers

Lookup Tables (continued)



2-input LUT implemented with a two-level cascade of 2-input multiplexers

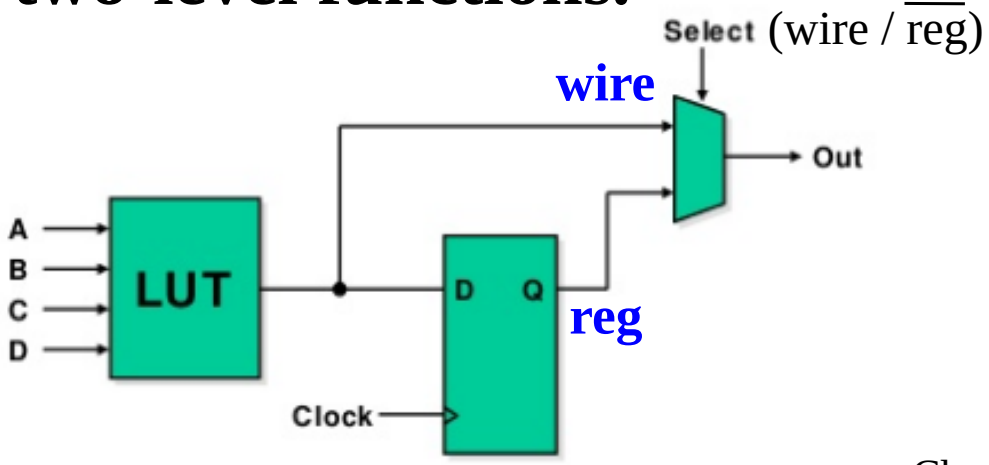
Lookup Tables (continued)



3-input LUT implemented with a three-level cascade of 2-input multiplexers

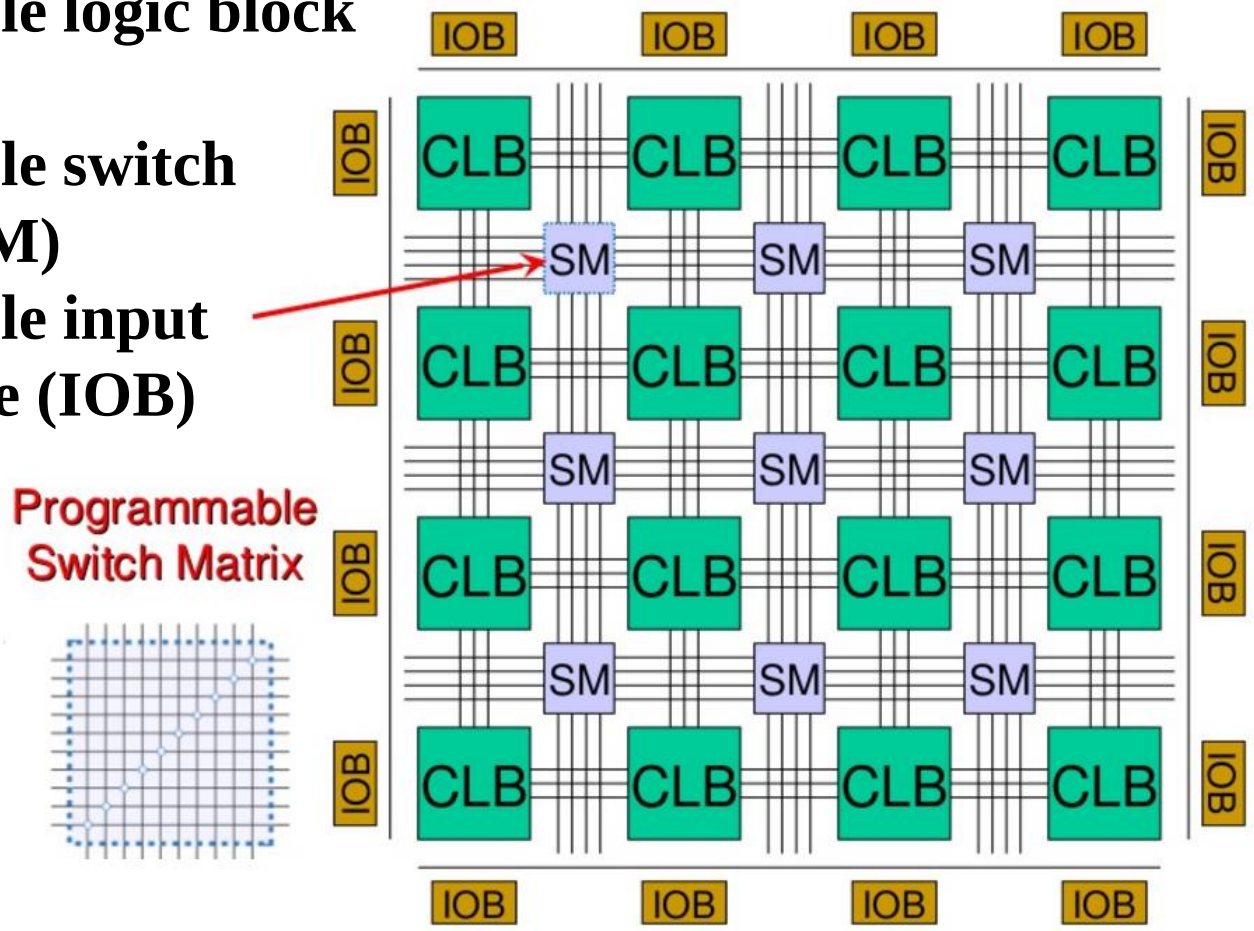
Lookup Tables (continued)

- **Lookup tables are typically small, often with 4 or 6 inputs, one output, and 16 or 64 entries.**
- **Since lookup tables store truth tables, it is possible to implement any 4 or 6-input function.**
- **Thus, the design problem is how to optimally decompose a set of given functions into a set of 4 or 6-input two-level functions.**



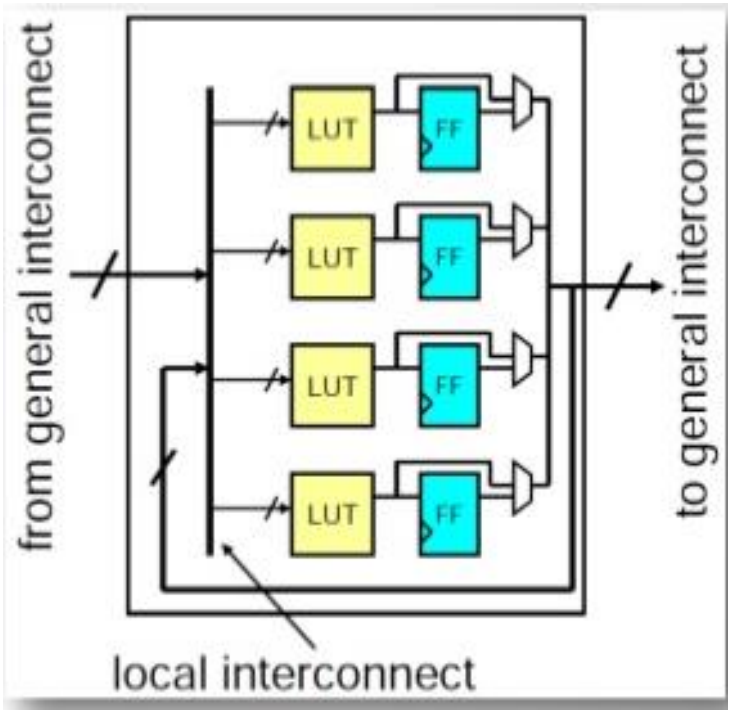
Programmable Gate Array: FPGA

- **The internal structure is composed of three parts:**
  - **Programmable logic block (CLB)**
  - **Programmable switch box/matrix(SM)**
  - **Programmable input output module (IOB)**



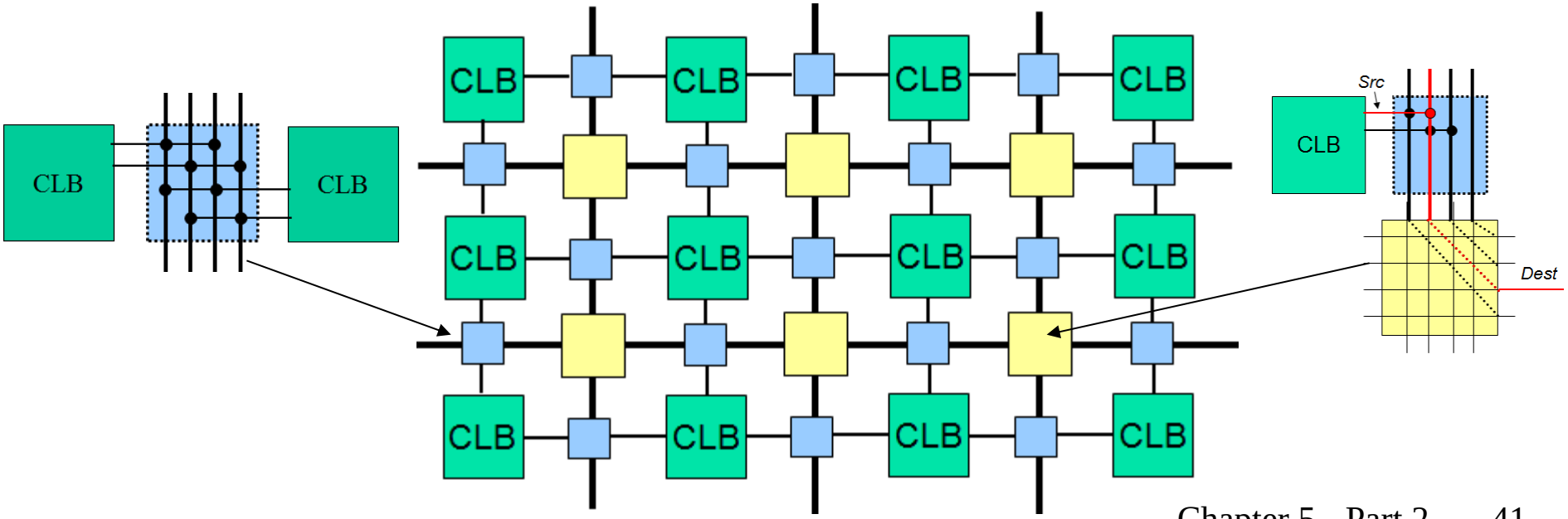
Configurable Logic Block

- **Configuration Logic Block (CLB) is the basic logic unit in a FPGA.**
  - **The storage cells in the LUTS are volatile**
  - **Using PROM to hold data permanently**
  - **The storage cells are loaded from PROM when the chip initializes the switch box/matrix**



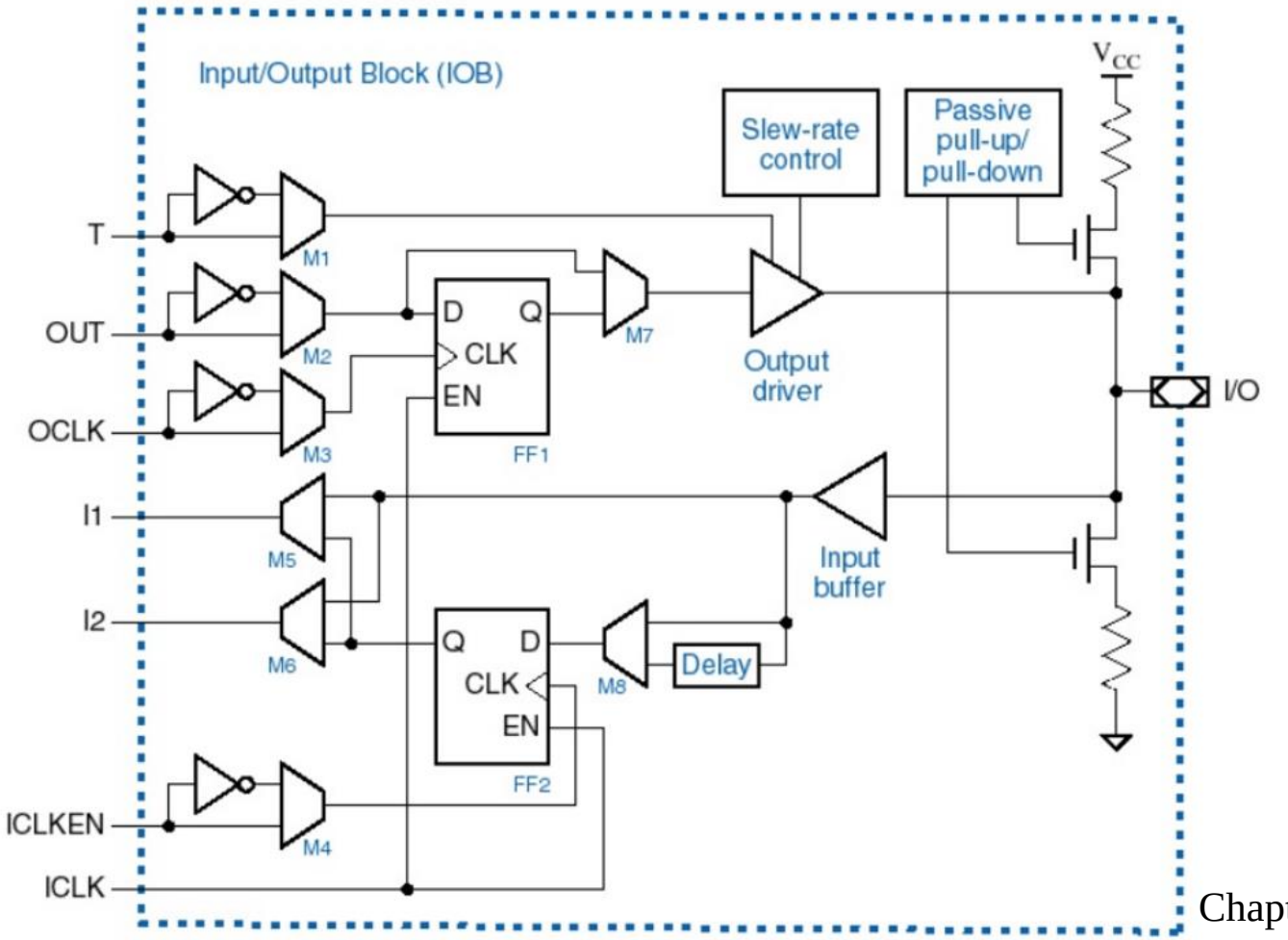
Reconfigurable Interconnect

- **Connection box allows inputs and outputs of CLB to connect to different wires.**
- **Connection box characteristics:**
  - **Flexibility: how many wires a single wire can connect to**
  - **Topology: which wires can be connected**
  - **Routability: how many circuits that can be routed**



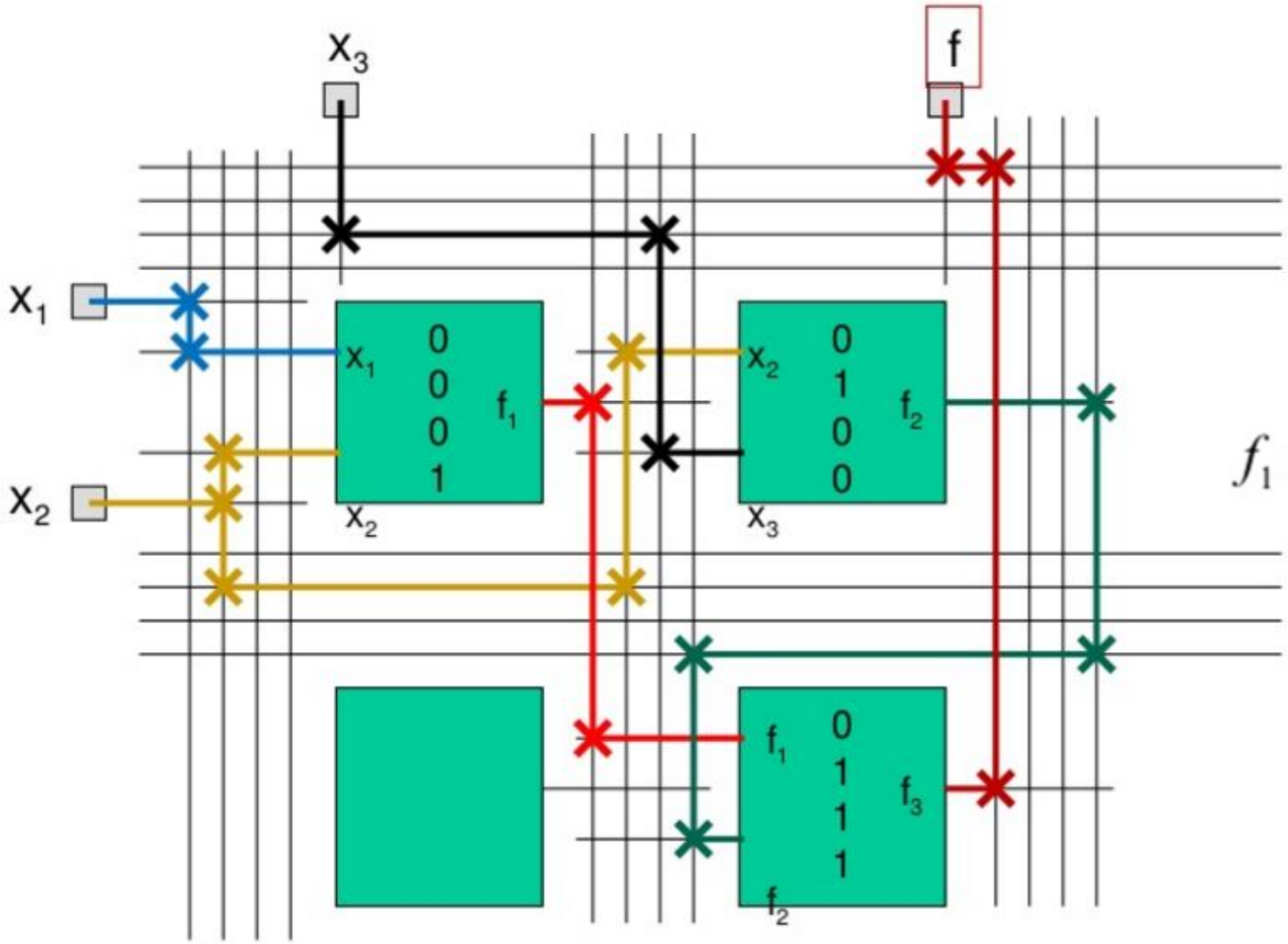
Programmable input/output module

- **I/O blocks are special logic blocks at periphery of device for external connections.**



Programming FPGAs

- **An example of programming an FPGA**



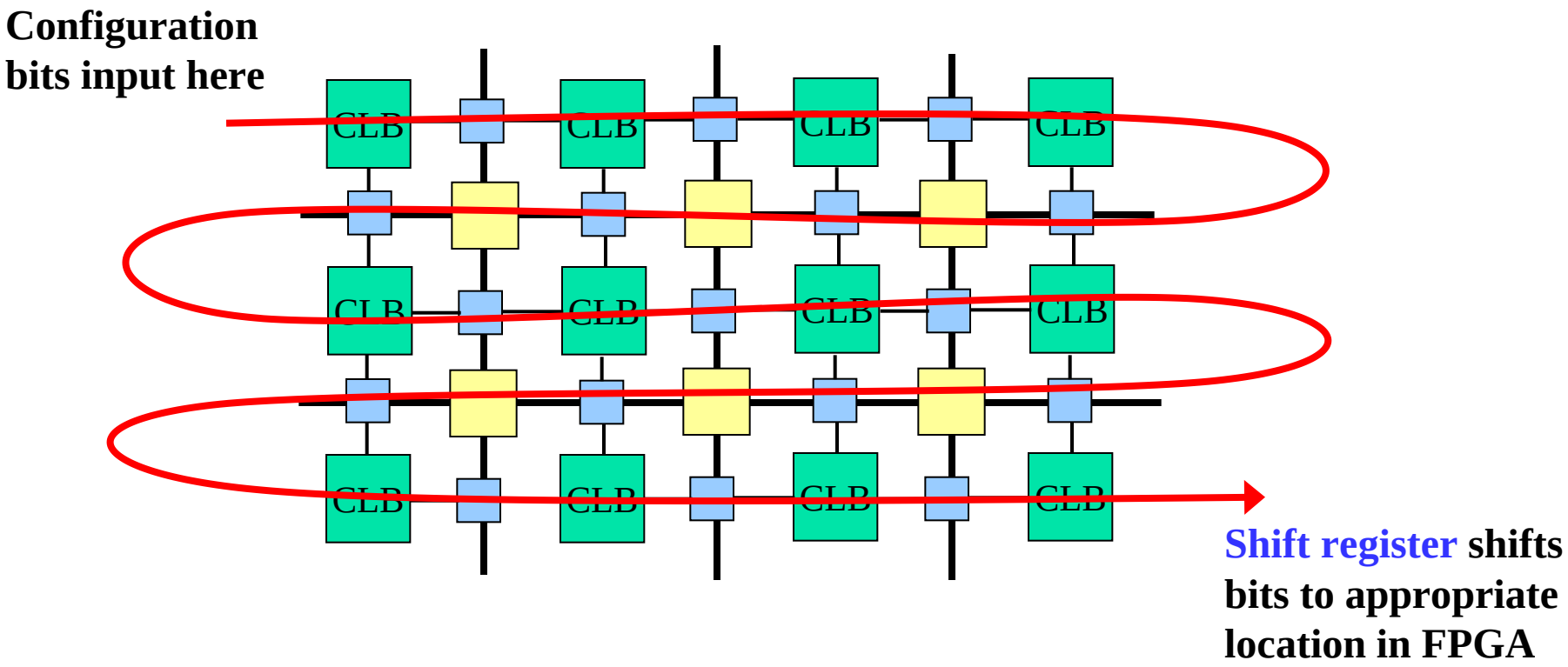
$$f = x_1x_2 + \overline{x_2}x_3$$

$$f_1 = x_1x_2 \qquad f_2 = \overline{x_2}x_3$$

$$f = f_1 + f_2$$

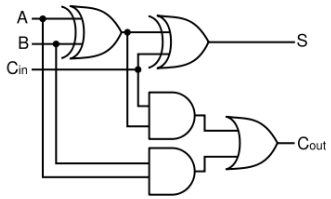
Programming FPGAs (continued)

- **FPGAs are programmed and configured with a "bitfile" that contains bits to fill LUTs.**

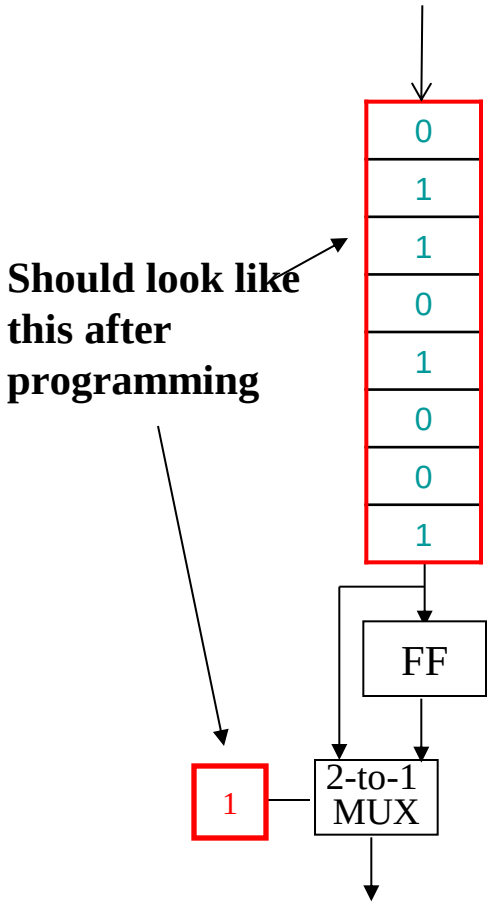


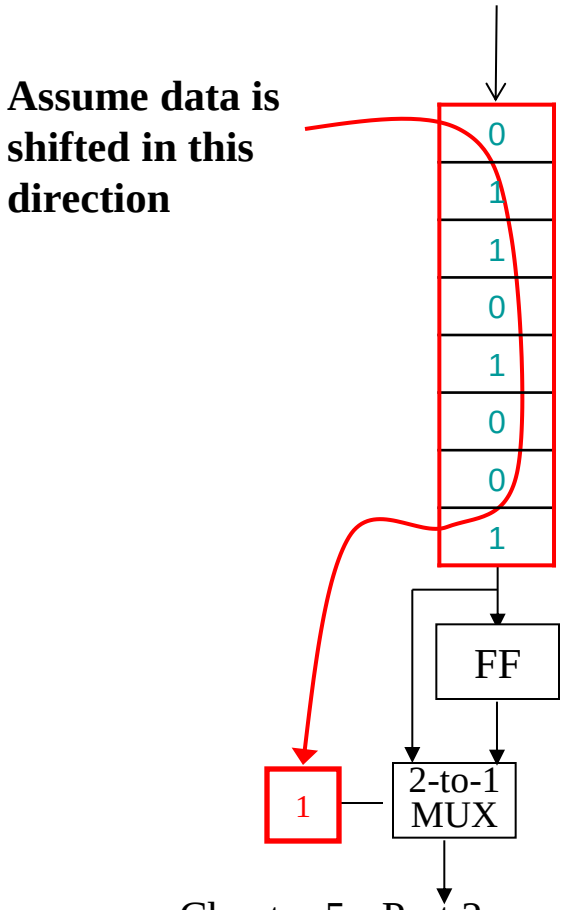
Programming FPGAs (continued)

- **An example of programing CLB with 3-input, 1-output LUT to implement sum output of full adder**



In			Out
A	B	Cin	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1





Combinational Function Implementation

- **Implementation techniques:**
  - **Decoders and OR gates**
  - **Multiplexers**
  - **ROMs**
  - **PALs**
  - **PLAs**
  - **Lookup Tables**
- **Can be referred to as structured implementation methods since a specific underlying structure is assumed in each case.**

Decoder and OR Gates Example

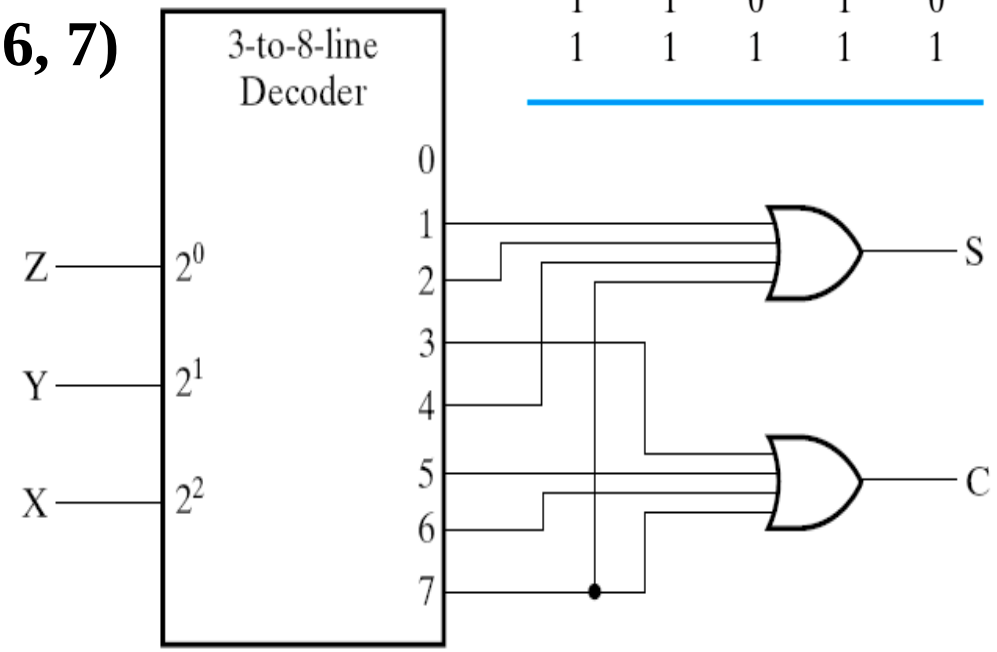
- **Implement a binary Adder**
- **Finding sum of minterms expressions**

$S(X, Y, Z) = \Sigma_m(1, 2, 4, 7)$
$C(X, Y, Z) = \Sigma_m(3, 5, 6, 7)$

Find circuit

Truth Table

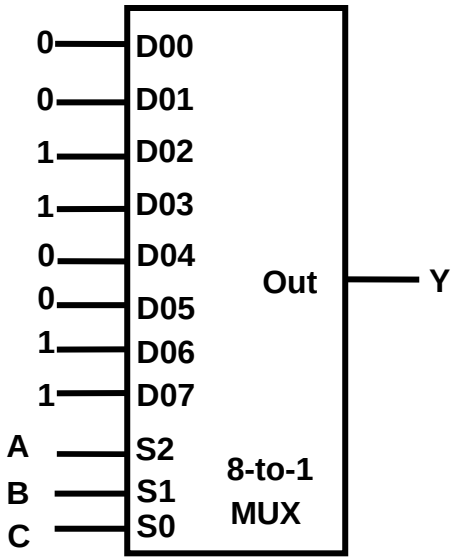
X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Multiplexer Example: Gray to Binary Code

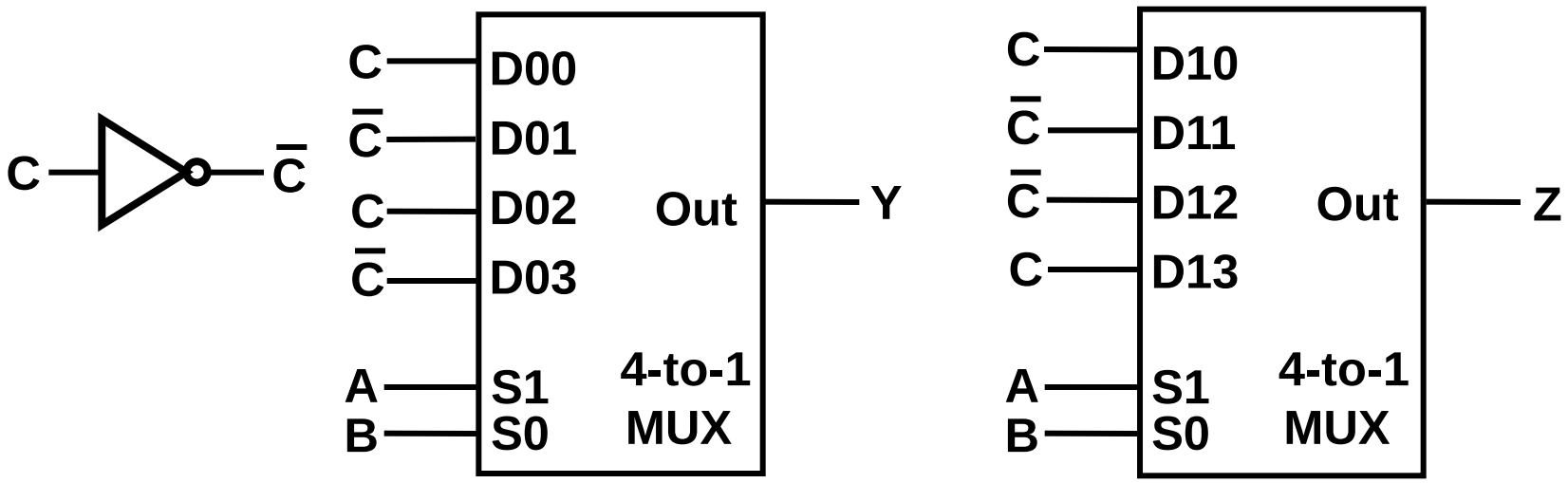
- **Design a circuit to convert a 3-bit Gray code to a binary code.**
- **Label the outputs of the multiplexer with the output variables. Value-fix the information inputs to the multiplexer using the values from the truth table.**

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)

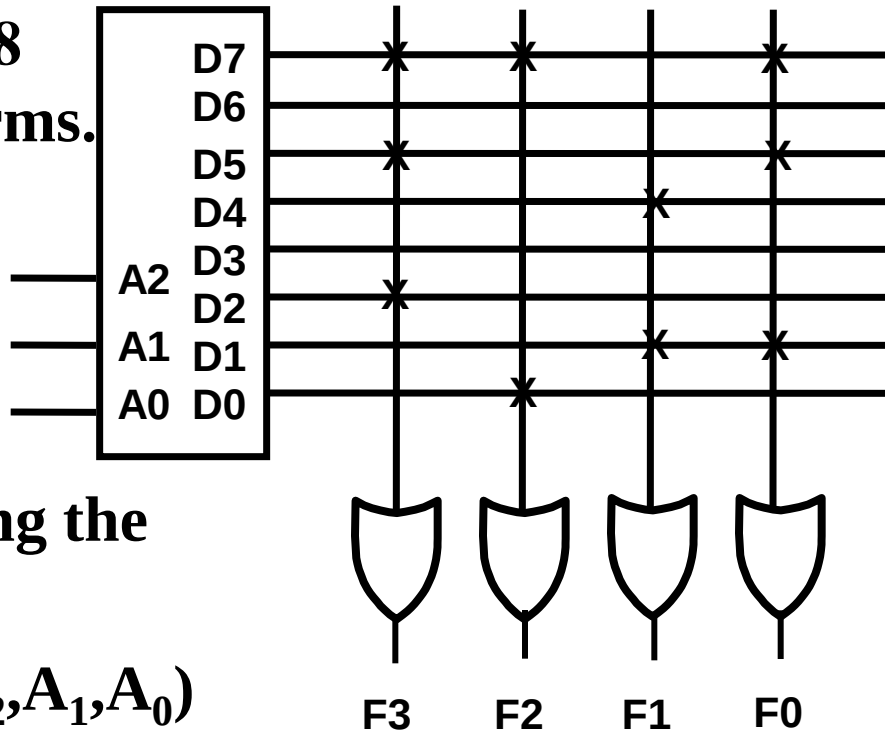
- **Assign the variables and functions to the multiplexer inputs:**



- **This approach reduces the cost by almost half , which requires less control poles for a multiplexer.**
- **This result is no longer ROM-like.**

Read Only Memory Example

- **Example: An 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) = 001, output is (F_3,F_2,F_1,F_0) = 0011.**

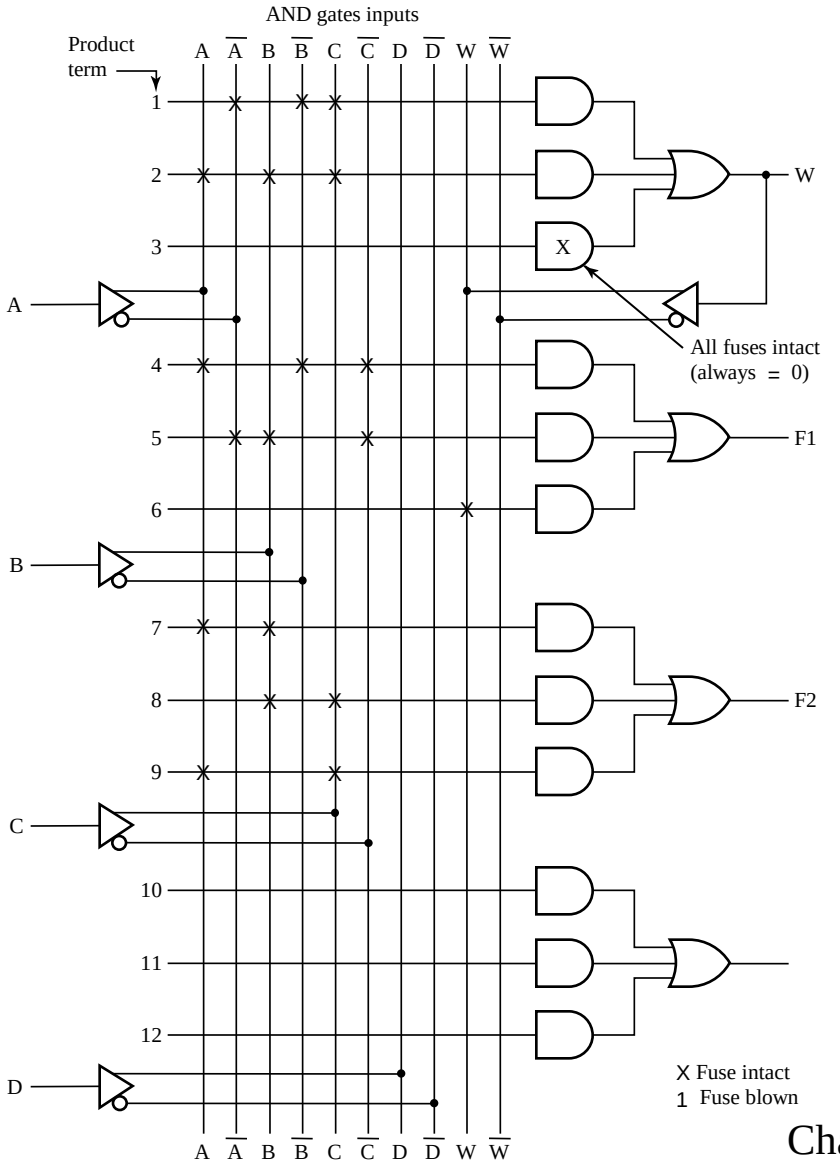


Programmable Array Logic Example

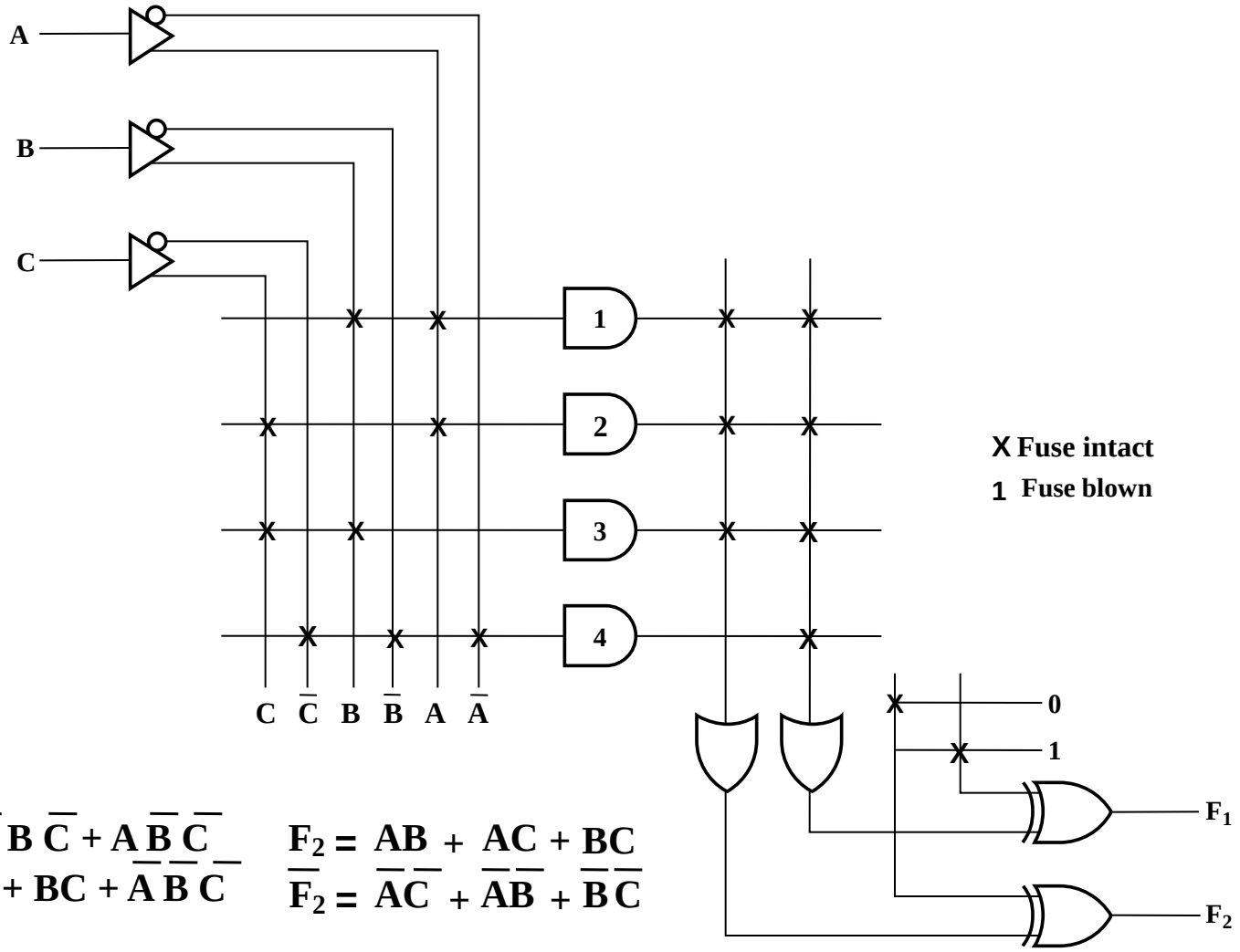
$W = \bar{A}BC + AB(\ldots$

$F1 = X = ABC + A\bar{B}C + W(\ldots$

$F2 = Y = AB\ldots + B\bar{C} + A(\ldots$



Programmable Logic Array Example



$$F_1 = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$$

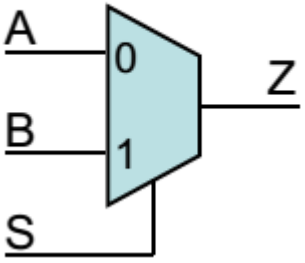
$$\bar{F}_1 = AB + AC + BC + \bar{A}\bar{B}\bar{C}$$

$$F_2 = AB + AC + BC$$

$$\bar{F}_2 = \bar{A}\bar{C} + \bar{A}\bar{B} + \bar{B}\bar{C}$$

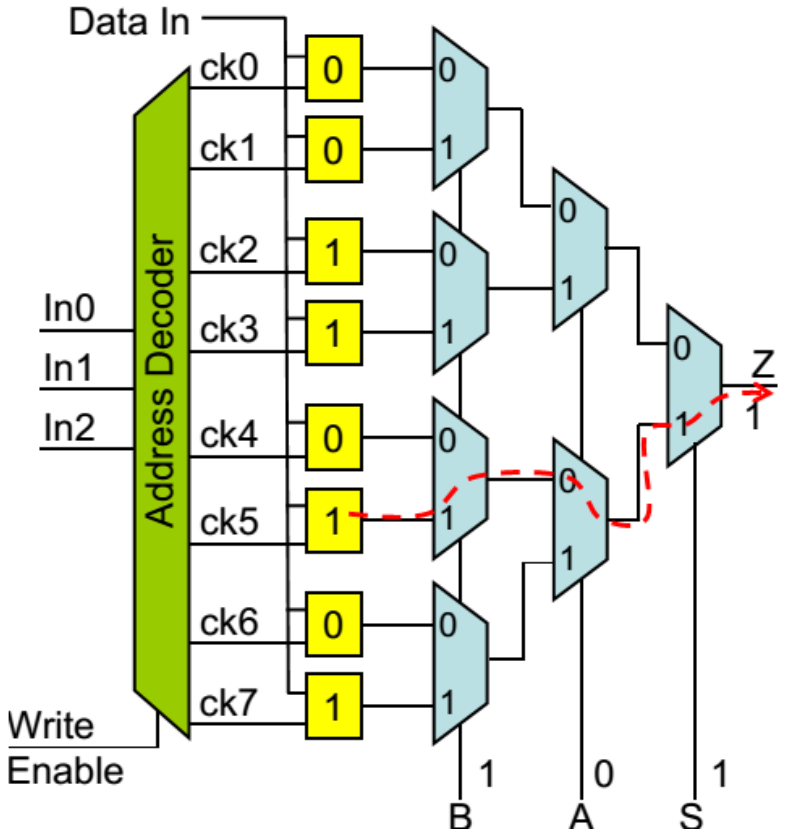
Lookup Tables

- **Lookup tables (LUTs) are used for implementing logic in Field-Programmable Gate Arrays (FPGAs).**
- **Lookup tables implement truth table in small memories (usually SRAM).**



Truth table

S	A	B	Z
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



Assignments

Reading

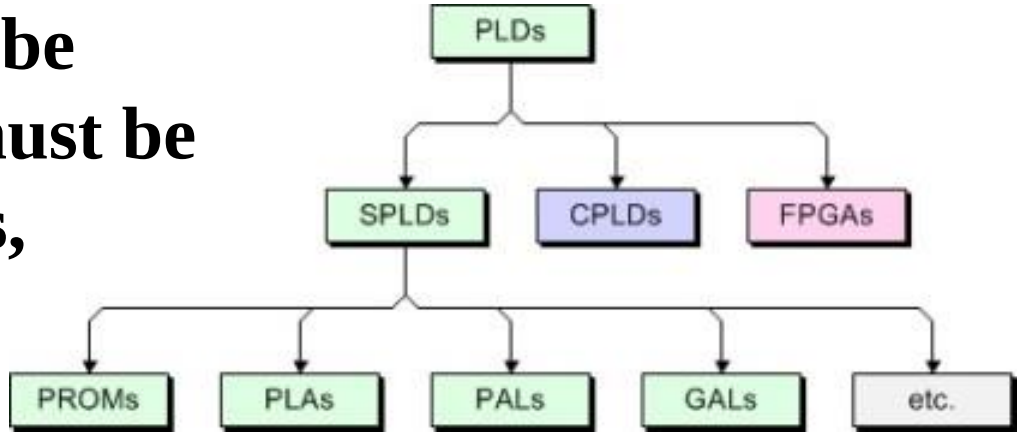
- **5.2**

Problem assignment

- **5-4, 5-12**

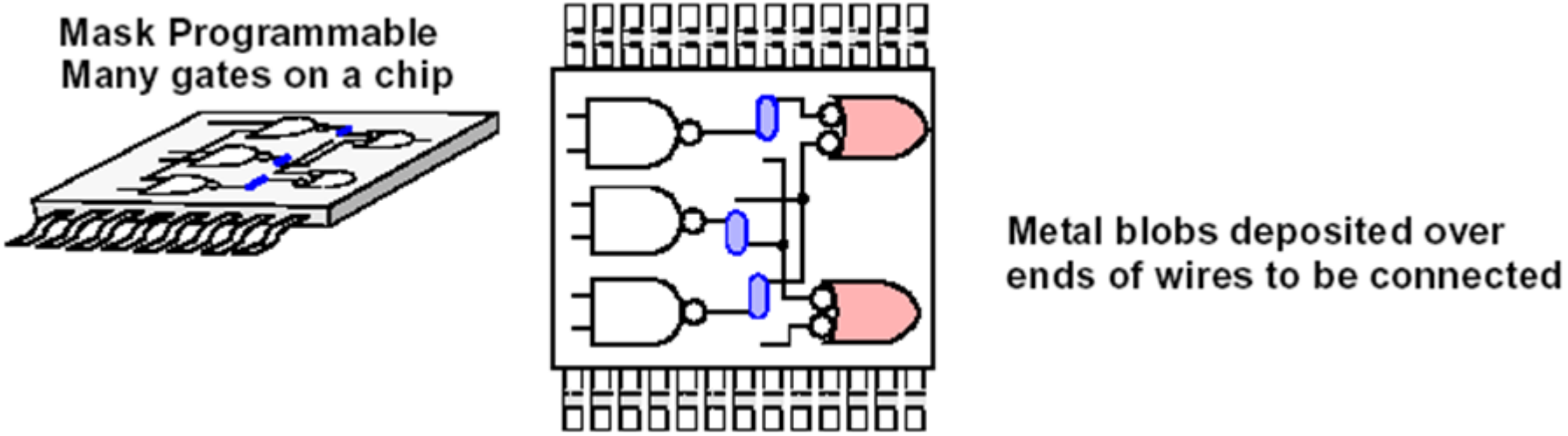
Programmable Logic Device

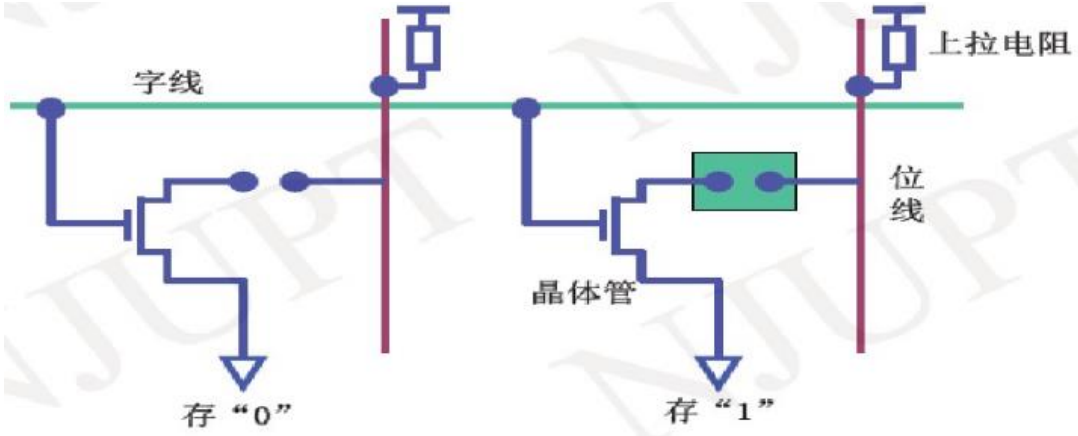
- **A programmable logic device (PLD) is an electronic component used to build reconfigurable digital circuits.**
- **Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture.**
- **Before the PLD can be used in a circuit it must be programmed, that is, reconfigured.**



Technology Characteristics

- **Permanent - Cannot be erased and reprogrammed**
  - **Mask programming (by manufacturers)**



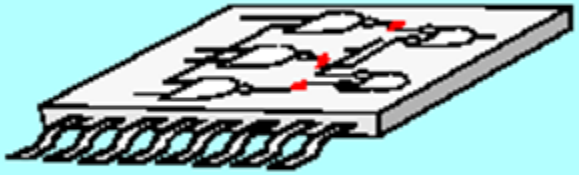


Technology Characteristics

- **Permanent - Cannot be erased and reprogrammed**
  - **Fuse**
  - **Antifuse**

  } **(by users)**

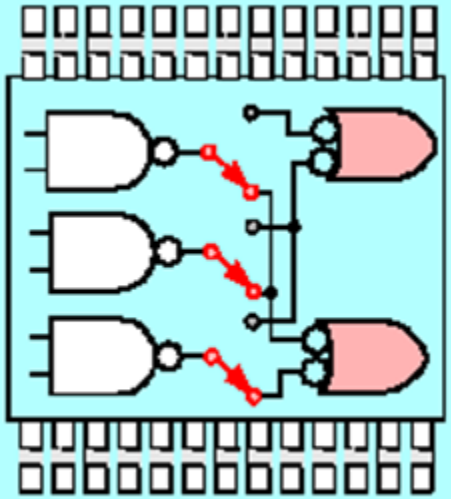
2. Electrically Programmable
Many gates on a chip



Electrically Controlled Switches

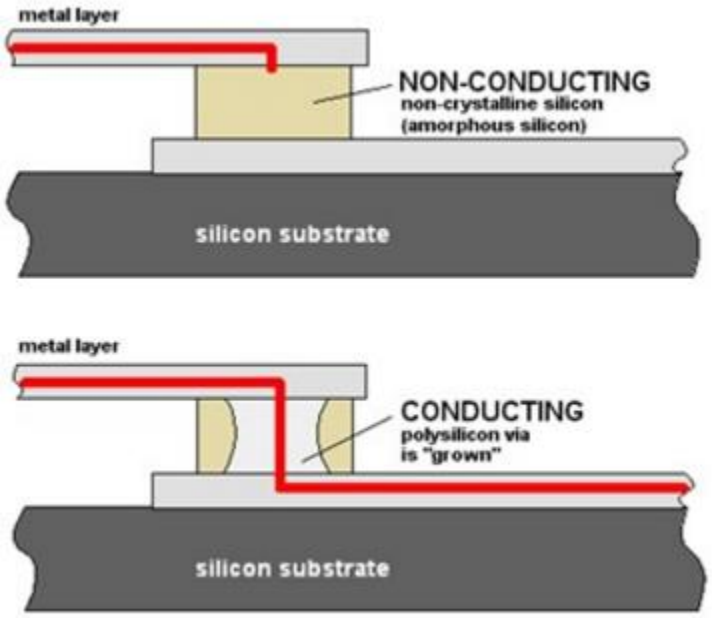
Fuses; blow to remove unwanted connections

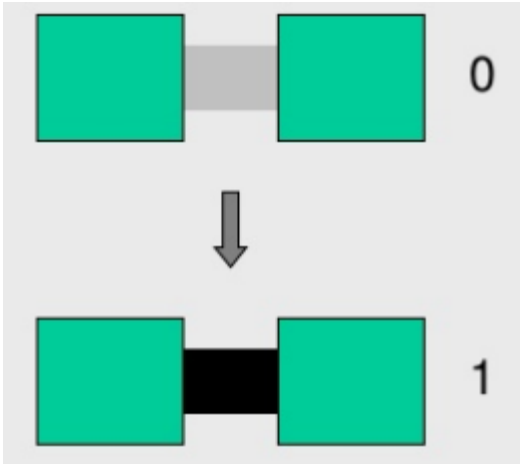
Antifuses; grow to make connections



Antifuse technology

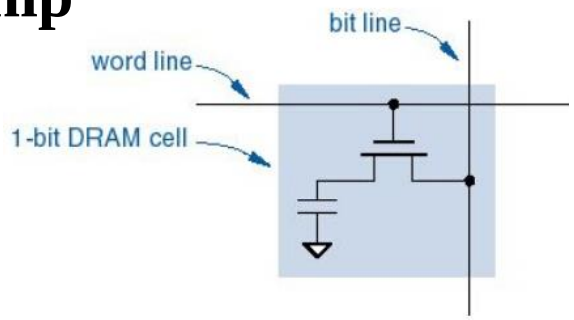
- **Two metal layers sandwich a layer of non-conductive, amorphous silicon. When voltage is applied to this middle layer, the amorphous silicon is turned into polysilicon, which is conductive.**





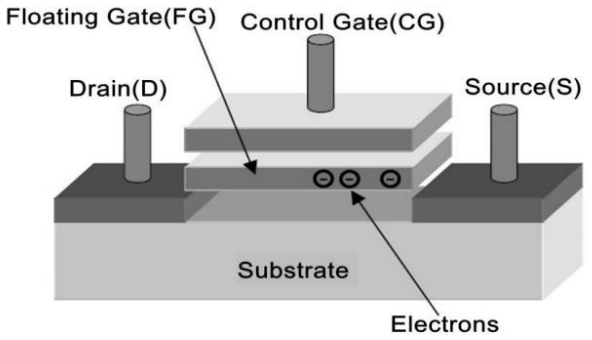
Technology Characteristics

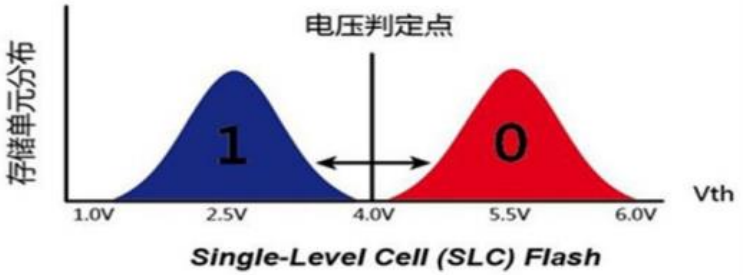
- **Reprogrammable**
  - **Volatile - Programming lost if chip power lost**
    - **Single-bit storage element**
  - **Non-Volatile**
    - **Erasable**
    - **Electrically erasable**
    - **Flash (as in Flash Memory)**



DRAM Cell

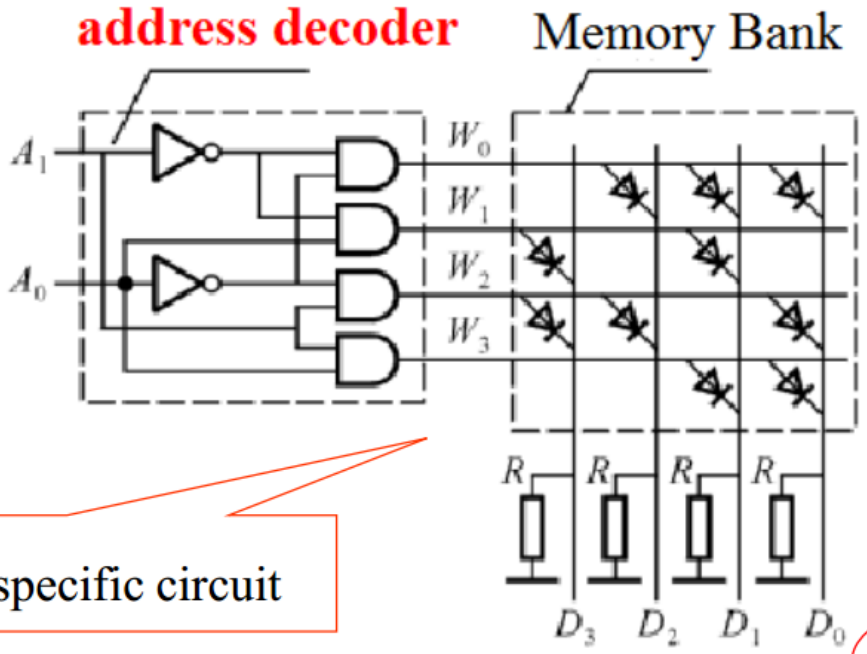
EPROM: Erasable Programmable ROM



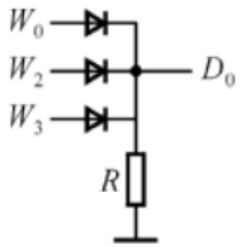


Appendix A: The general structure of ROM

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



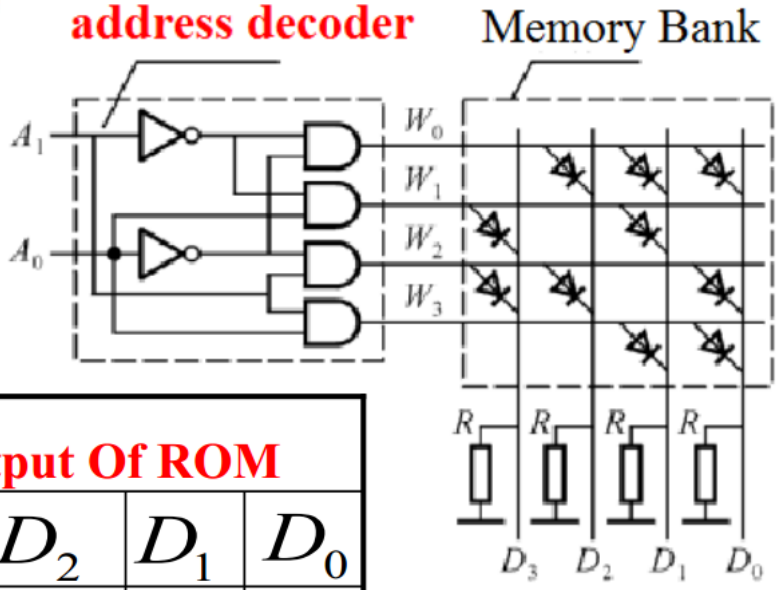
4×4 ROM specific circuit



"D_0" bit wire "OR" gate structure

The general structure of ROM (continued)

- **Output information of ROM:**
- **ROM size = $2^2 \times 4 = 16$bit**



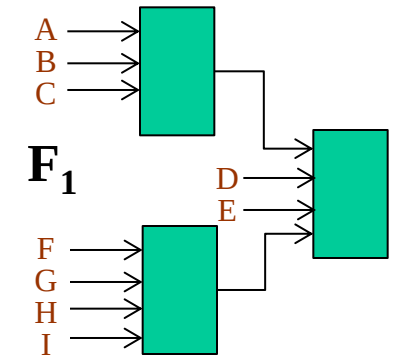
address		Word select wire	Output Of ROM			
A_1	A_0	W	D_3	D_2	D_1	D_0
0	0	W_0	0	1	1	1
0	1	W_1	1	0	1	0
1	0	W_2	1	1	0	1
1	0	W_3	0	0	1	1

Lookup Table Example

- **Equations to be implemented:**

  F1(A,B,C,D,E,F,G,H,I)=ABCDE+FGHIDE

  F2(A,B,C,D,E,F,G,H,I)=ABCE+DFGHI



- **Compute the number of LUT： Factoring k**
  - **Number of inputs = 9 k=[9/4]=3 need 3 LUT**
- **Divide these 2 functions into function group with 4 variables, each group contains 3 functions**
  - **Need at most 6 LUT**
  - **If common LUT exists, the number of LUT can be reduced by 2**

F_1= (ABC)DE+ (FGHI)DE

$X_1(A,B,C)=ABC$

$F_1(D,E,X_1,X_2)=X_1DE+X_2DE$

F_2= (ABC)E+D(FGHI)

$X_2(F,G,H,I)=FGHI$

$F_2(F,X_1,X_2)=X_1E+DX_2$