

MEMORY AND PROGRAMMABLE LOGIC

- *Memory* is a device where we can store and retrieve information
 - It can execute a *read* and a *write*
- *Programmable Logic* is a device where we can store and retrieve information
 - It can also execute a *read* and a *write*
 - The *write* is usually tricky

Types of Memory

- So confusing that it is funny!
- There are two types: RAM and ROM
- RAM stands for Random Access Memory, can perform both read and write and forgets everything when unplugged from the power
- Then why is it called RAM?

Go Figure

- The name RAM distinguishes it from a short lived and now extinct serial access memory (not to be confused with serial RAM)
- Sometimes it is called core memory (core is the iron in an electromagnet or transformer!)

What about ROM

- It is not a museum
- Stands for Read Only Memory
- It is not a memory. It is a programmable device
- Behaves more like a combinational circuit!

Writing and Programming

- Writing is for memory
- Programming is for Programmable Logic Devices
- Writing is (about) as fast, repeatable and convenient as reading
- Programming is not

Types of Programming

- ROM: Mask Programming; done at the factory
- PROM: can be Programmed once by the user
- EPROM: can be Erased (with UV) and reprogrammed
- EEPROM: can be Electrically Erased

Flash?

- It is EEPROM on hormones
 - Huge capacity
 - Can be erased section by section
 - Can be erased many many times
 - Threatens the Kingdom of Disk

Other PLDs?

- ROM: Read Only Memory
- PLA: Programmable Logic Array
- PAL: Programmable Array Logic
 - Lousy name, eh?
- FPGA: Field Programmable Gate Arrays

ROM

- Can implement a truth table, bit by bit
- Can implement any logic function that has as many input as the ROM has address lines and as many outputs as the word size of the ROM

PLA

- Can implement simplified logic functions
- Has a programmable AND-OR network
- Can fit functions with up to a certain number of prime implicants

PAL

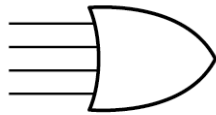
- Like PLA but the OR part of the network is not programmable.
- Less flexible than PLA, but this rarely causes problems

Gate Arrays

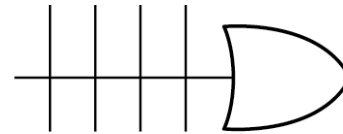
- The mother of all PLDs
- Several regular PLDs with
 - F-F connected to the outputs (bypassable)
 - Interconnected to a programmable network
 - Can contain many components like MUX, table-look-ups etc

New Symbol

- We need a new symbol to accommodate gates with large (possible) inputs
- Can be an AND or OR gate (or NAND, NOR)
- Instead of the familiar octopus we use half fish bone



(a) Conventional symbol



(b) Array logic symbol

Fig. 7-1 Conventional and Array Logic Diagrams for OR Gate

RAM

- There are two main types
 - Static RAM, using regular latches and is optimized for speed, typically used for DSP and real time applications
 - Dynamic RAM, using tiny capacitors, needs periodic refreshing, optimized for size and used for general computing

What about the others?

- SDRAM or SDR RAM: Single Data Rate RAM, one datum per cycle when streaming
- DDR RAM: Double Data Rate, both posedge and negedge output when streaming
- DRDRAM (Direct Rambus): more sensing amplifiers, can stream longer

Memory Unit

- n data inputs
- n data outputs
- READ, WRITE, ENABLE
- k address lines

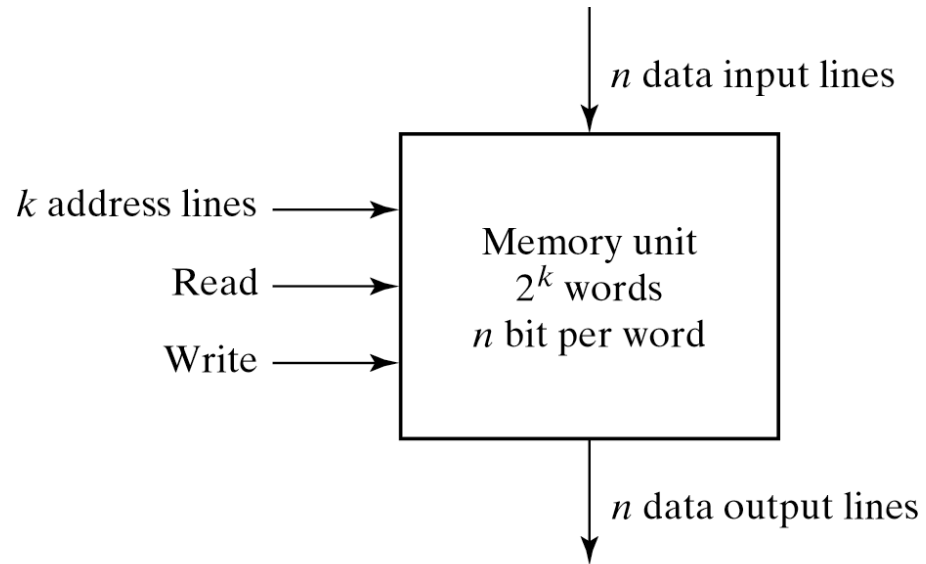


Fig. 7-2 Block Diagram of a Memory Unit

Memory Contents

- Organized in 2^k lines of n bytes each
- Looks suspiciously like a truth table

Memory address		Memory content
Binary	decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Fig. 7-3 Content of a 1024×16 Memory

Read and Write Operations

- Read:
 - Apply binary address
 - Activate READ (and wait?)
- Write:
 - Apply binary address
 - Apply data
 - Activate write

Timing

- Timing is complex
- Memory is slower and might not use the system clock

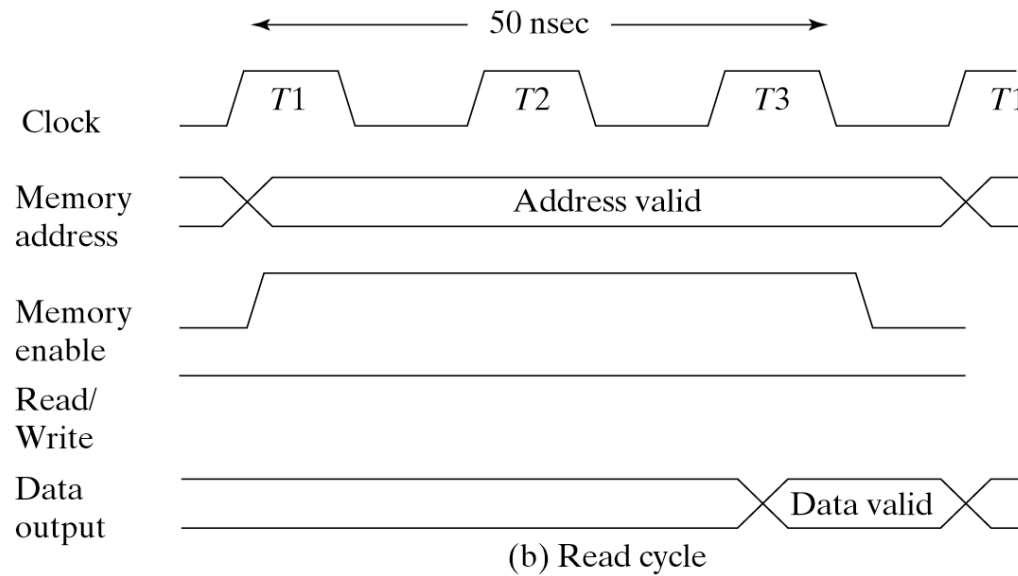
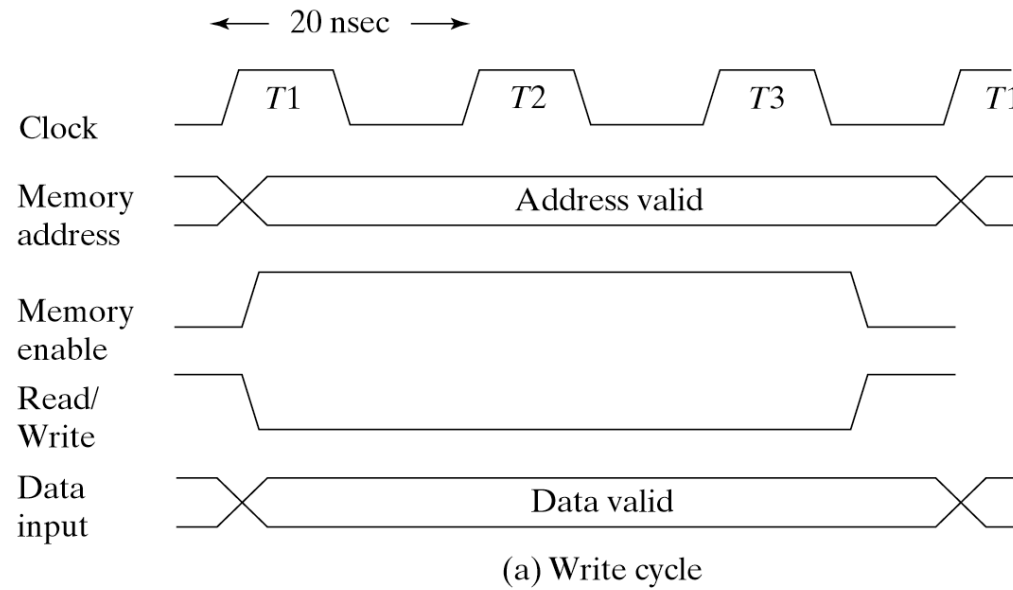


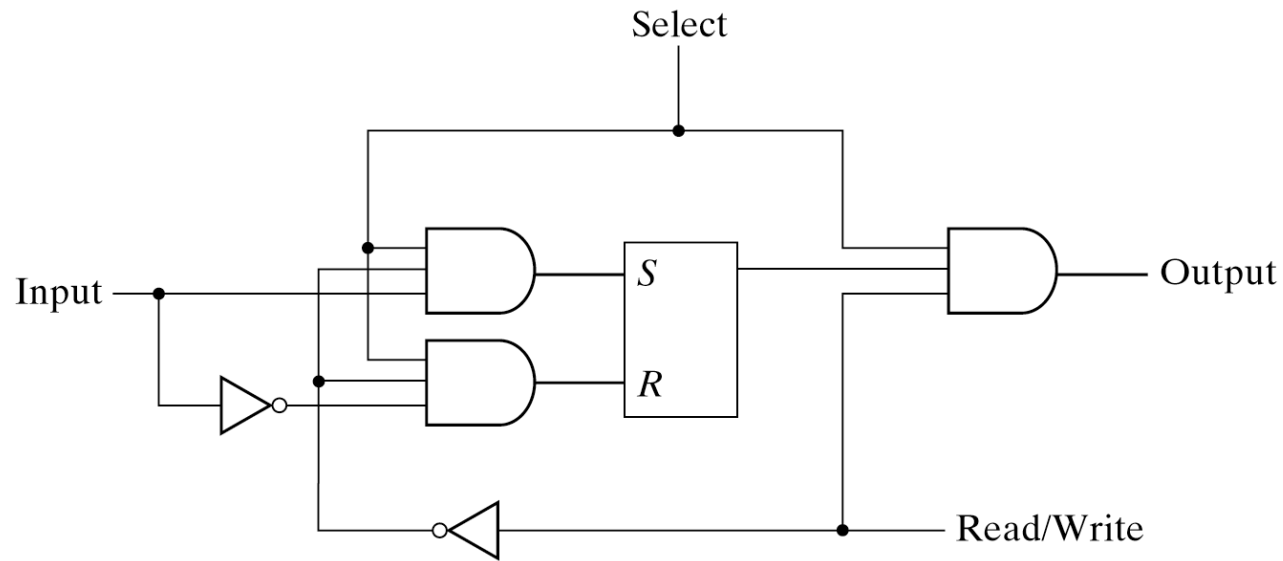
Fig. 7-4 Memory Cycle Timing Waveforms

Commercial Systems

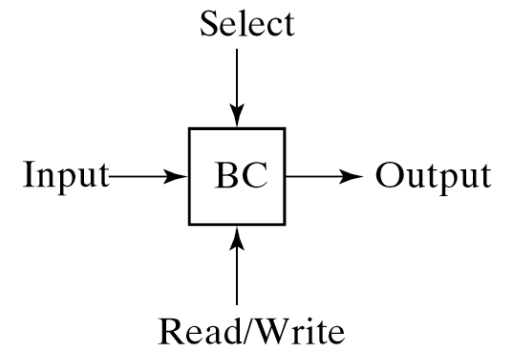
- Have much more complex timing structure and protocols
 - multiple accesses (read over write, etc)
 - dual port
 - streaming/strobing
 - cache
 - virtual memory

Basic Memory Cell

- The memory cell of SRAM has a latch
- has data input and output
- has read/write and enable control inputs



(a) Logic diagram



(b) Block diagram

Fig. 7-5 Memory Cell

Internal Arrangement

- The cells are arranged in matrix
- The enables are connected to the output of a decoder
- the read/write is shared by all
- the outputs of every column are ORed together

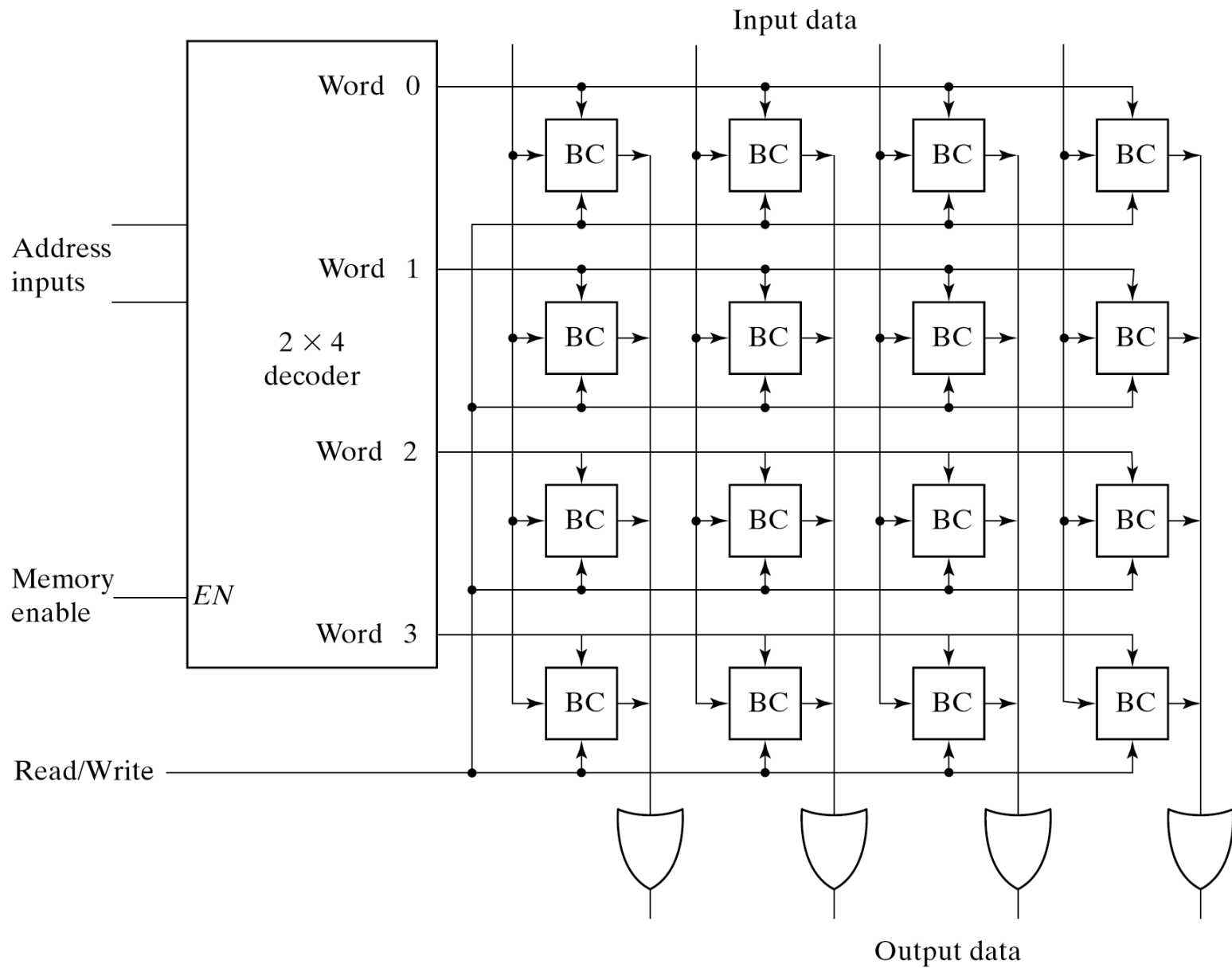


Fig. 7-6 Diagram of a 4×4 RAM

Coincident Decoding

- We can split the decoder in two
- Use several two input AND gates

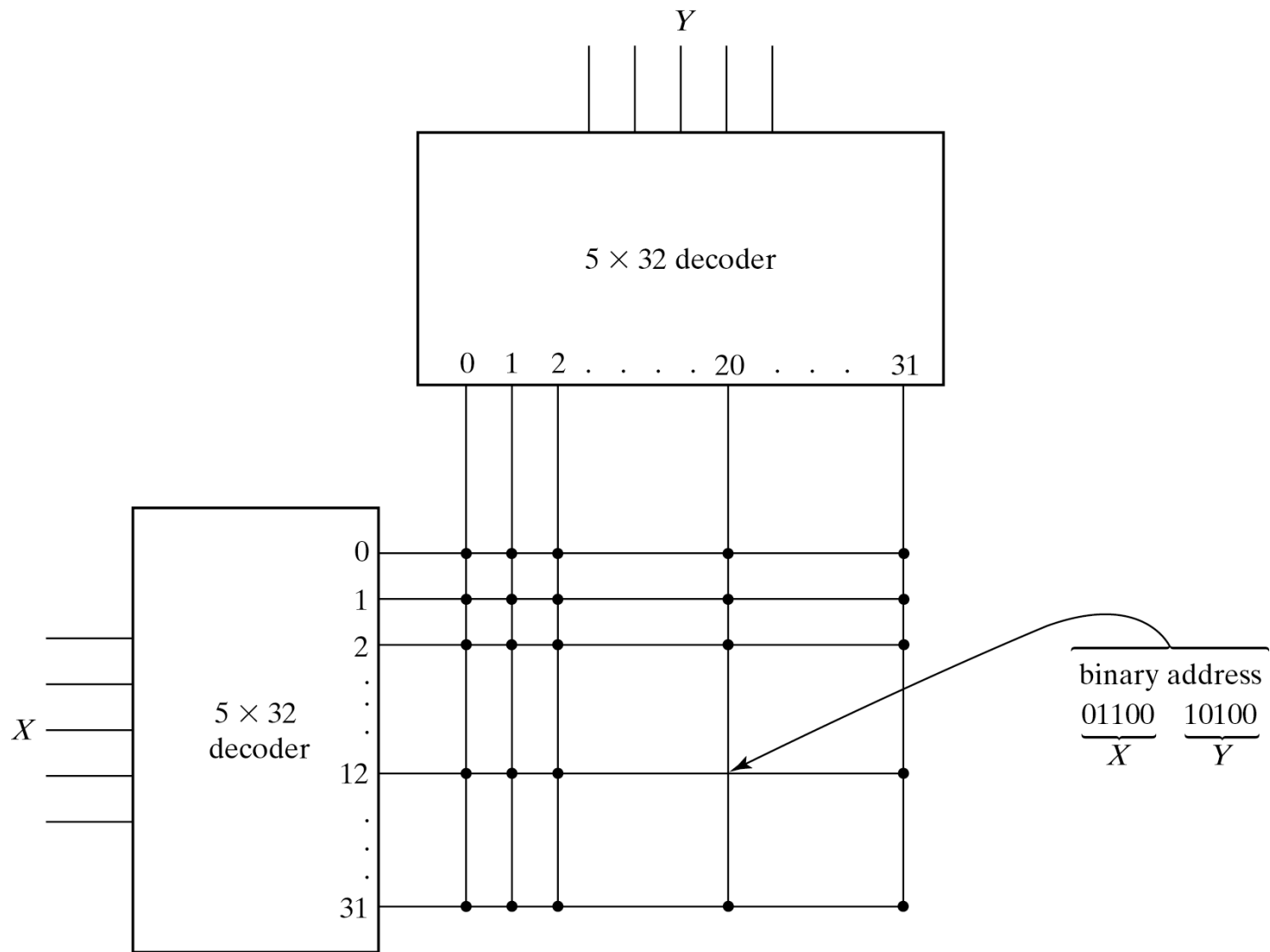


Fig. 7-7 Two-Dimensional Decoding Structure for a 1K-Word Memory

Address Multiplexing

- To reduce the number of address pins we can multiplex address pins
- We have to indicate if we provide column or address bits
 - Column Address Strobe
 - Row Address Strobe

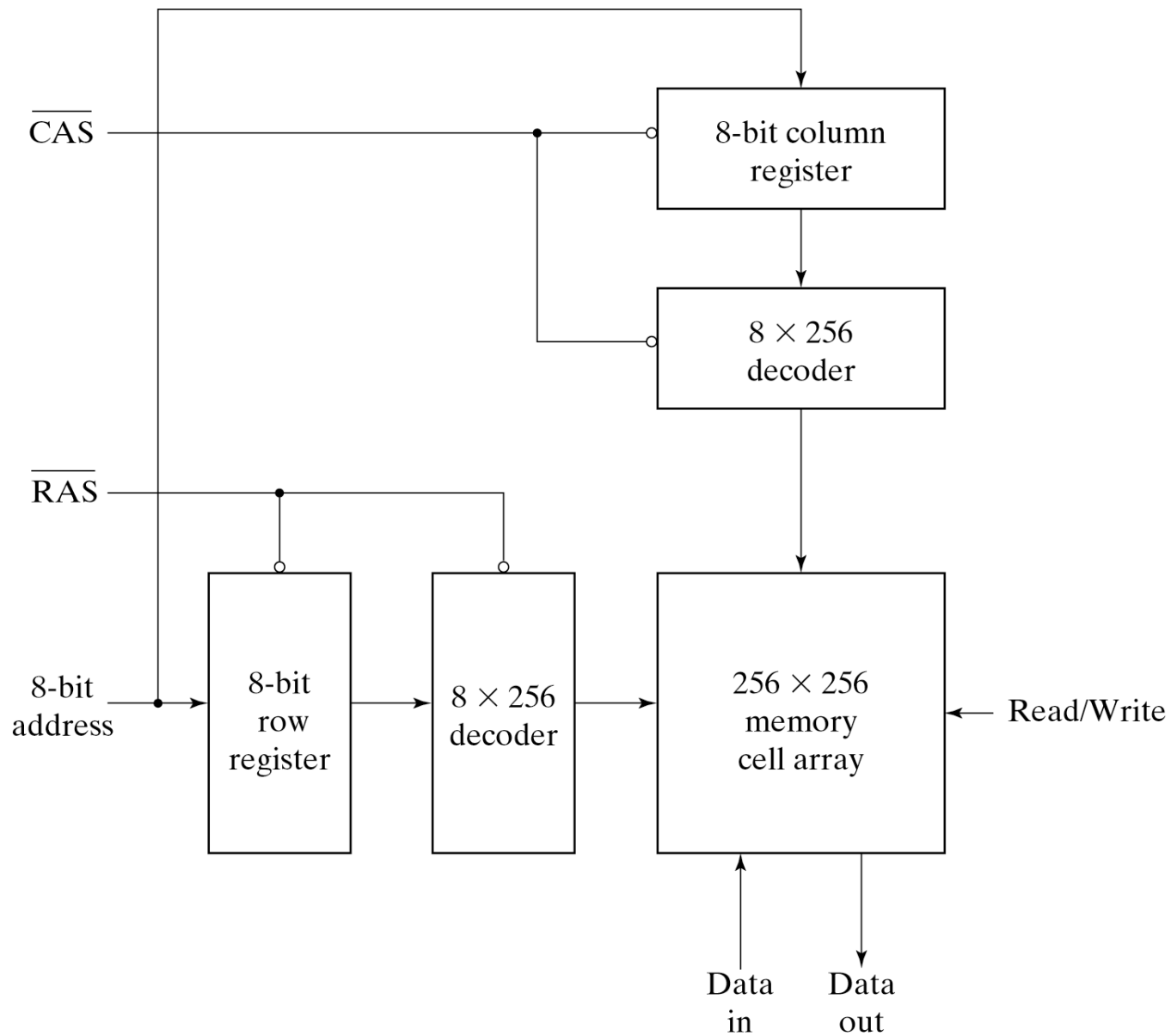


Fig. 7-8 Address Multiplexing for a 64K DRAM

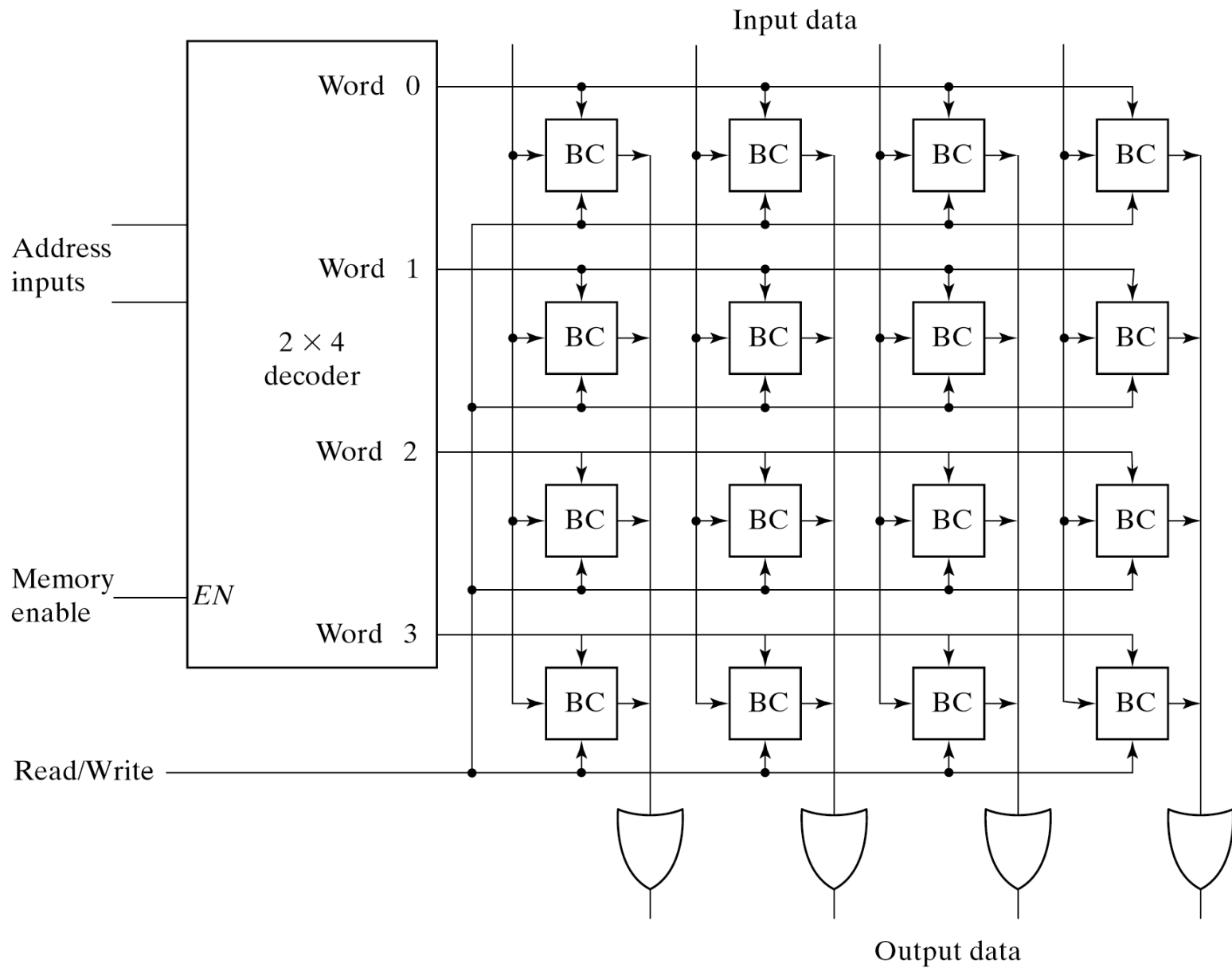


Fig. 7-6 Diagram of a 4×4 RAM

Error Detection and Correction

- Bits in memory are known to flip spontaneously
- Maybe cosmic rays, maybe radioactive isotopes in the material that goes in
- Miniaturization does not help

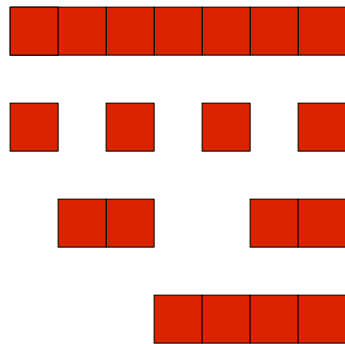
ECC

- Technology developed mainly for communications
- Also used for CDs, hard disks
- Great variety of methods
- Interesting mathematics
- Most are related to parity

Hamming Code

- One of the oldest and most used
- Good for spontaneous errors
 - not good for burst errors
- In the simplest form can correct one error
 - and get confused with two errors

Example of a Hamming Code



First, second and fourth bit are parities

Even Parity

Even parity

Even Parity

Encoding

- Place the data bits in the 3rd 5th 6th and 7th position
- Compute the values in the 1st, 2nd and 4th position
- We can compute each one independently.

Decoding

- Check the three parities
 - write 0 if the parity is correct
 - write 1 if the parity is incorrect
- Write the third one first, the first one third
- If the parities form a 000, the 7 bits were stored and retrieved correctly

Correction

- 001 the 1st bit is wrong
- 010 the 2nd bit is wrong
- 011 the 3rd bit is wrong
- 100 the 4th bit is wrong
- 101 the 5th bit is wrong
- 110 the 6th bit is wrong
- 111 the 7th bit is wrong

Easy to trick

- If there are two errors say 1st and 2nd bits
- The parities will be 011
- The system will smile and correct 3rd bit
- We end up with three errors
- Rats!

Do not Despair

- How about if we add another bit that checks the parity of the whole word
- If the parity is 0000, everything is OK
- If the parity is 1000 the new bit is wrong
- If the parity is 0xxx we have 2 errors
- If the parity is 1xxx we correct 1 error

How to Generalize

- It is quite complex but the idea is simple
- Define the *Hamming Distance* between two words as:
 - the number of bits they differ
- If there is an error in a word then it is 1 H.D. from the original

How to Generalize

- Define a code such that the valid (correct) words are at least 3 H.D. apart to correct 1 error, at least 5 H.D. apart to correct two errors etc

ROM

- Looks like RAM
- Once programmed it is like a combinational circuit
- We can store the truth table on it and have a ready made combinational circuit
- Unfortunately, with every new input we double the size

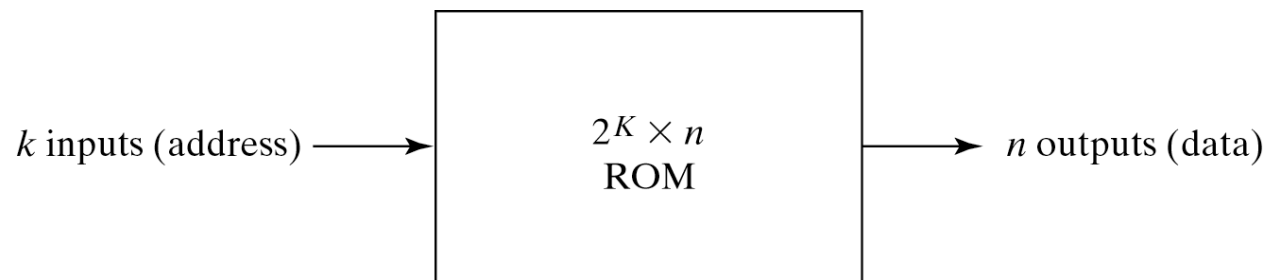


Fig. 7-9 ROM Block Diagram

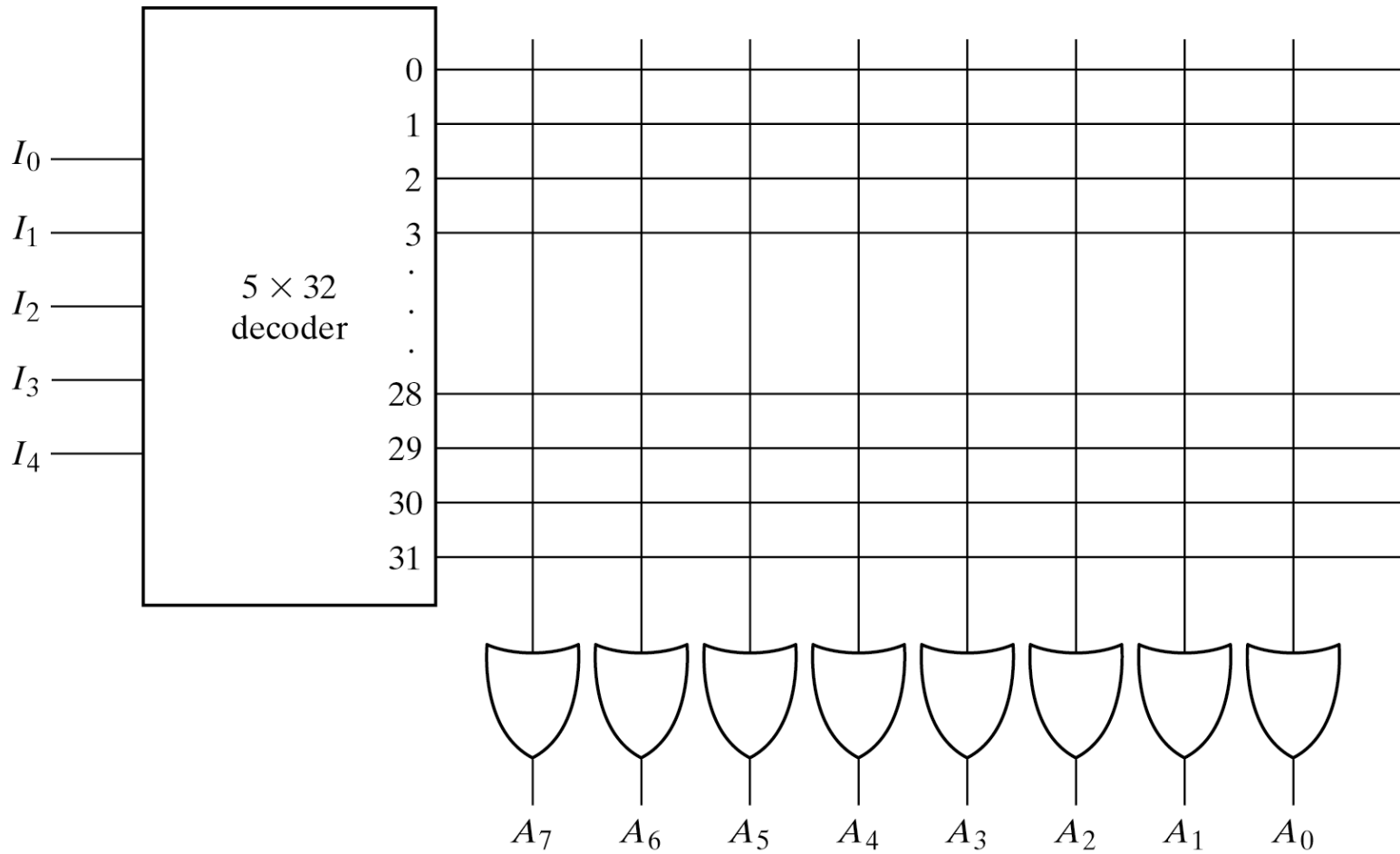


Fig. 7-10 Internal Logic of a 32×8 ROM

ROM Truth Table (partial)

Inputs					Outputs							
I4	I3	I2	I1	I0	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

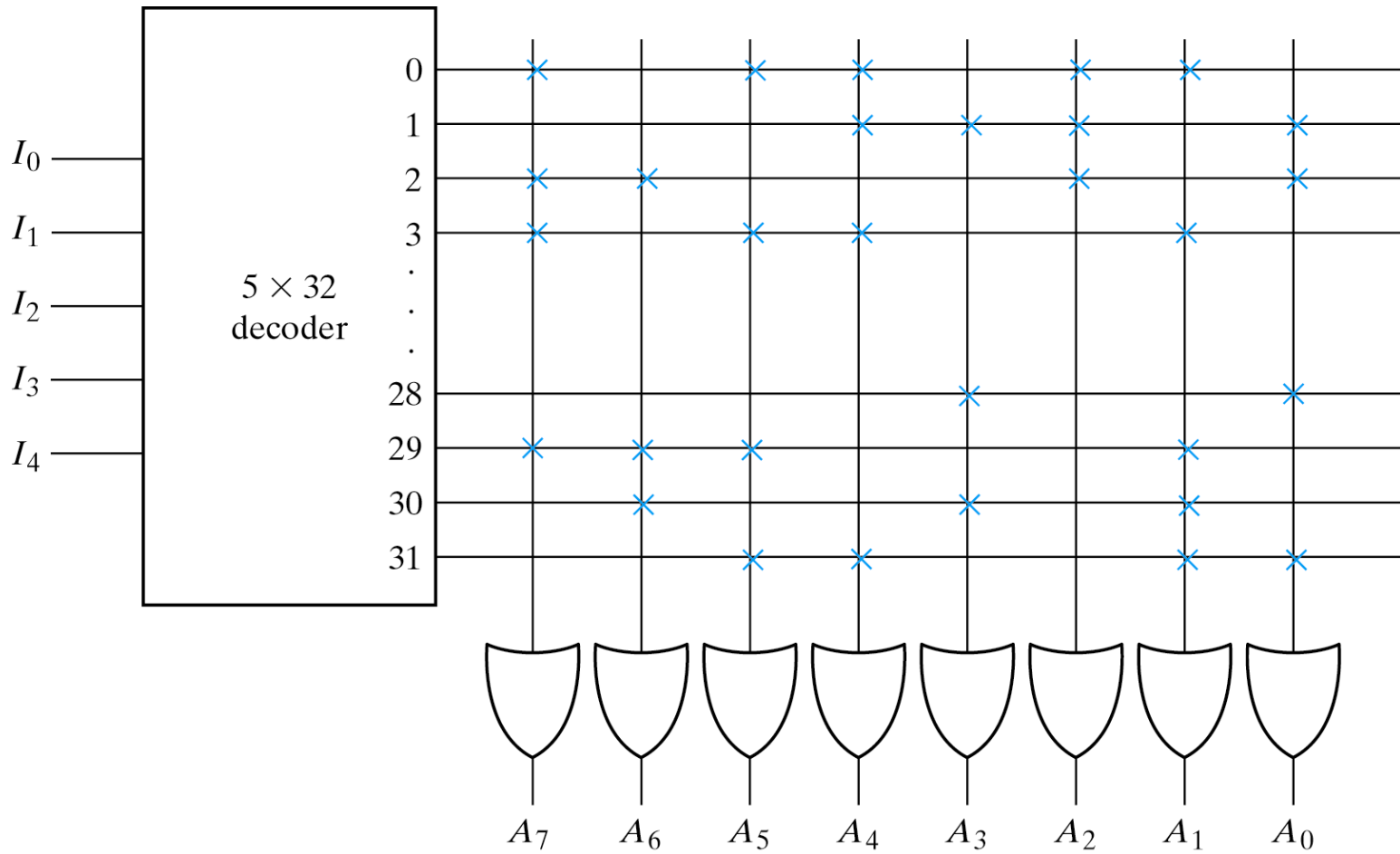
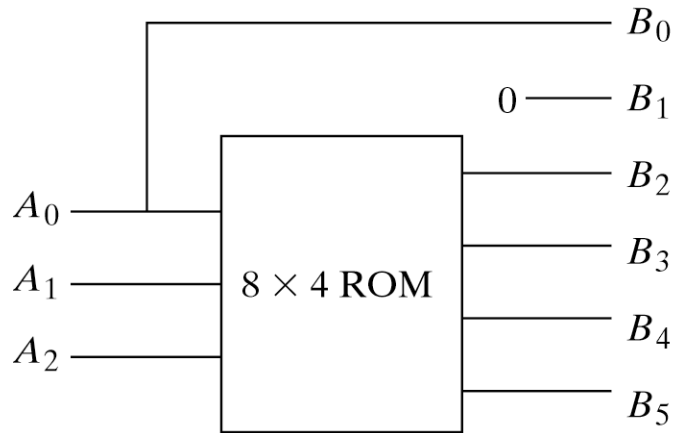


Fig. 7-11 Programming the ROM According to Table 7-3

We can Mix and Match

- Sometimes we can implement part of the function as regular combinational circuit and part of the function (the one that is not amenable to simplification) using ROM
- Here B_0 and B_1 are very simple and can be implemented trivially



(a) Block diagram

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

(b) ROM truth table

Fig. 7-12 ROM Implementation of Example 7-1

PLDs

- ROM is a PLD
- We can think of it as having two parts
 - The AND part is complete (whole decoder) and not programmable
 - The OR part is programmable.

Other PLDs

- Can have the AND programmable and the OR fixed (PAL)
- Can have both programmable (PLA)

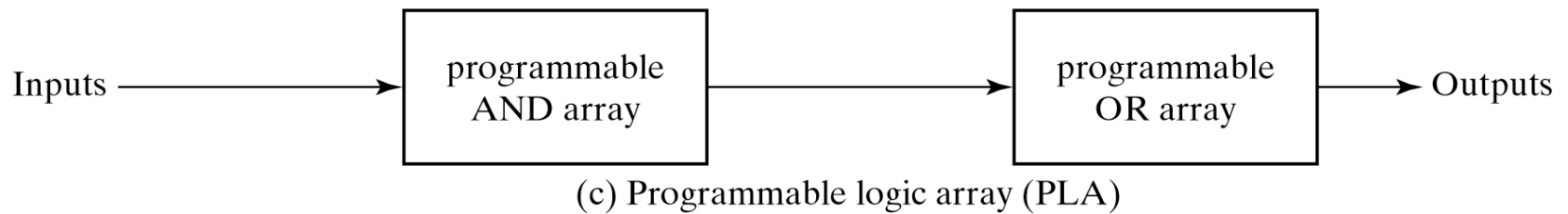
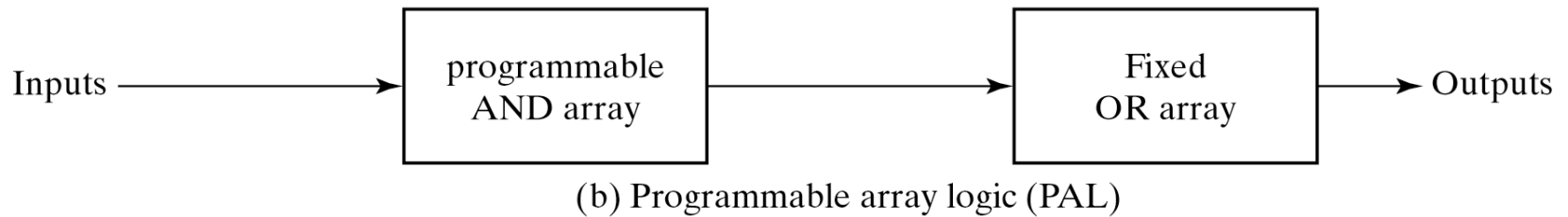
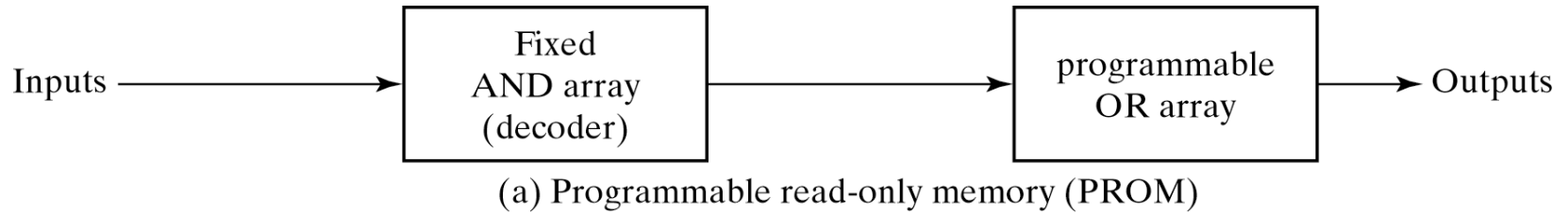


Fig. 7-13 Basic Configuration of Three PLDs

Sharing Implicants

- We try to have the two functions share implicants
- We attempt to minimize both the function and its complement (PLAs usually have either NAND-NAND or NOR-NOR implementation, but not both)

Calculating the Cost

- The bottom line of the cost is the price of the chip that fits the design.
- One has to try all (or at least many) technologies
- Take into account non-technical restrictions

Example

- The two functions we need to implement are
 - $F1 = AB' + AC + A'BC'$
 - $F2 = (AC + BC)'$

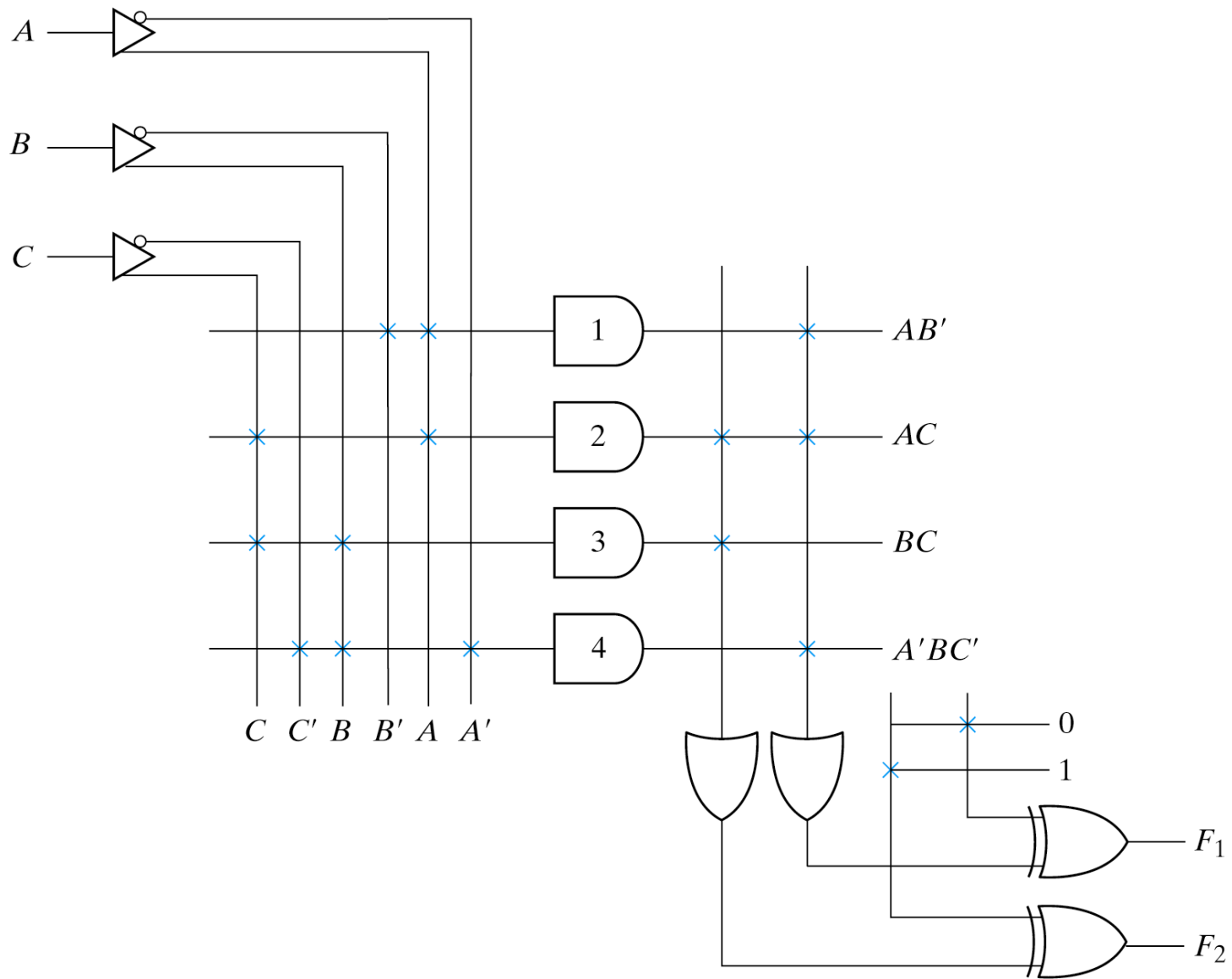


Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

Another Example

- The two functions are
 - $F1(A,B,C) = \text{Sum}(0,1,2,4)$
 - $F2(A,B,C) = \text{Sum}(0,5,6,7)$
- The solution is
 - $F1 = (AB + AC + BC)'$
 - $F2 = AB + AC + A'B'C'$

The Reason is

- They share two implicants
- Which makes it easier to fit on a smaller chip
- If this chip did not exist or was too slow then we should try something else
- Hardware compilers allow us to do it easily

		BC		B	
		00	01	11	10
A	0	1	1	0	1
A	1	1	0	0	0
		C			

$$F_1 = A'B' + A'C' + B'C'$$

$$F_1 = (AB + AC + BC)'$$

		BC		B	
		00	01	11	10
A	0	1	0	0	0
A	1	0	1	1	1
		C			

$$F_2 = AB + AC + A'B'C'$$

$$F_2 = (A'C + A'B + AB'C')'$$

PLA programming table

	Product term	Outputs				
		Inputs			(C)	(T)
		A	B	C	F_1	F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$A'B'C'$	4	0	0	0	–	1

Fig. 7-15 Solution to Example 7-2

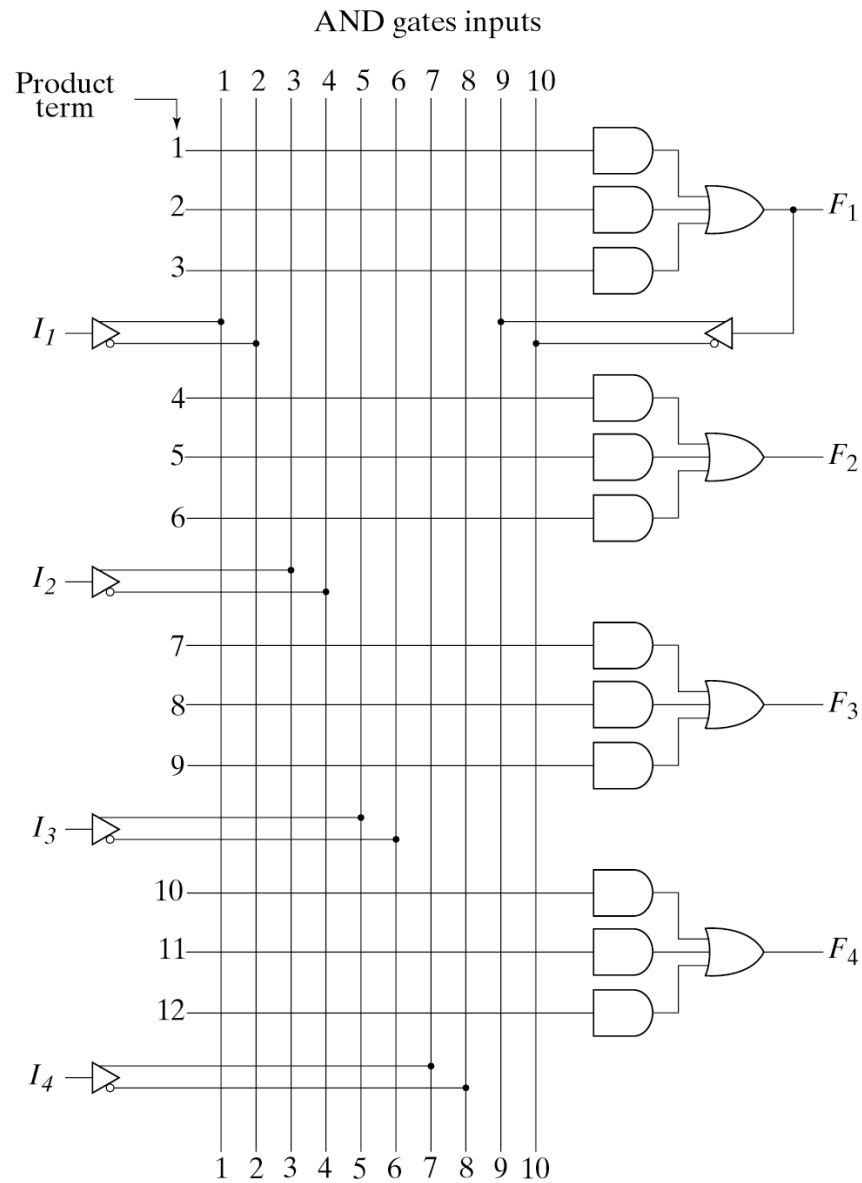


Fig. 7-16 PAL with Four Inputs, Four Outputs, and Three-Wide AND-OR Structure

Example for PAL

- The four functions are:
 - $w(A,B,C,D) = \text{Sum}(2,12,13)$
 - $x(A,B,C,D) = \text{Sum}(7,8,9,10,11,12,13,14,15)$
 - $y(A,B,C,D) = \text{Sum}(0,2,3,4,5,6,7,8,10,11,15)$
 - $z(A,B,C,D) = \text{Sum}(1,2,8,12,13)$

The Best Solution

- $w = ABC' A'B'CD'$

- $x = A + BCD$

- $y = ABC' + A'B'CD' + AC'D' + A'B'C'D$

- $z = w + AC'D' + A'B'C'D$

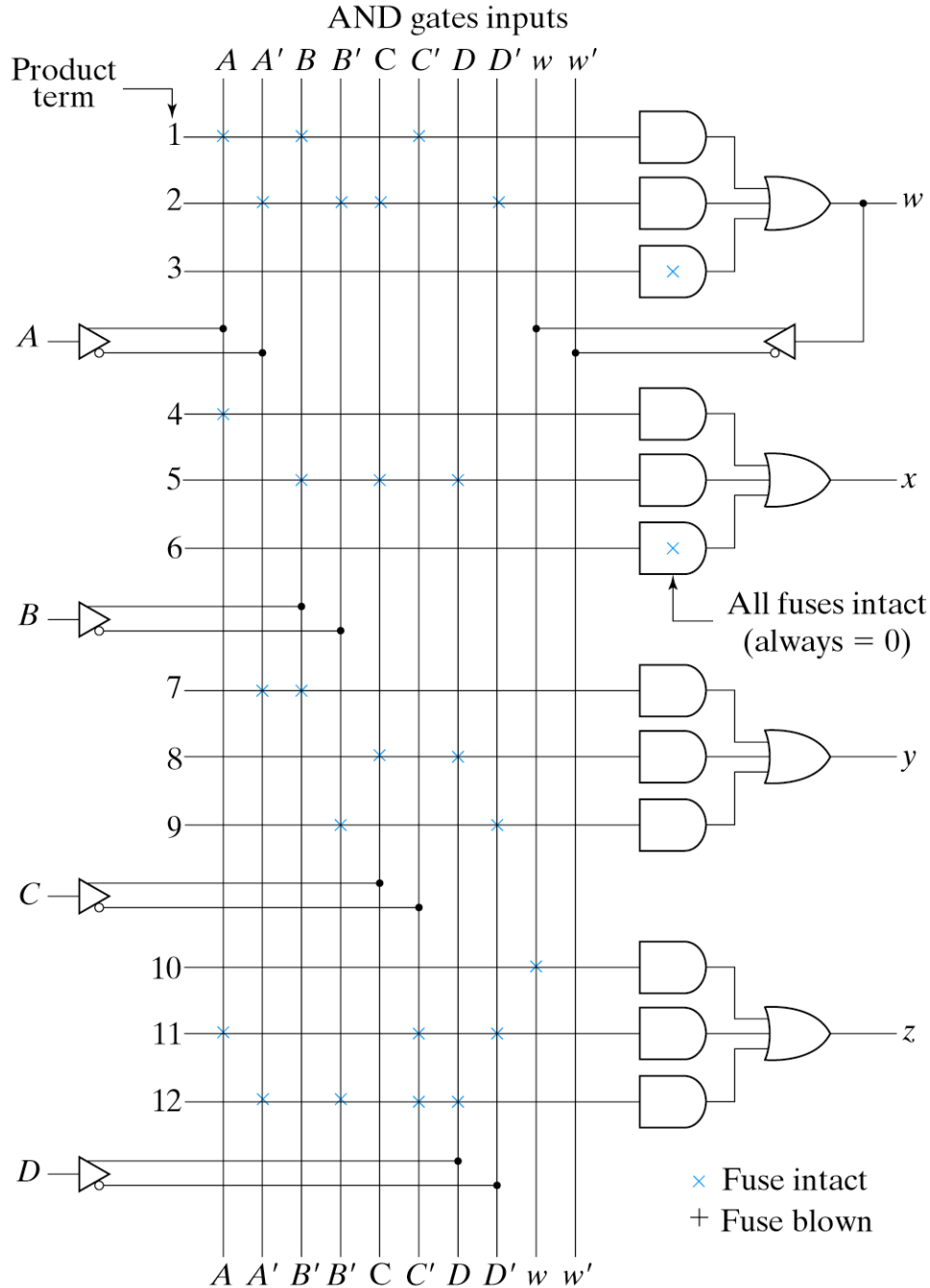


Fig. 7-17 Fuse Map for PAL as Specified in Table 7-6

Sequential PLDs

- Why not add a few flip-flops to the mix
- Sequential circuits consist of a combinational circuit and a few flip-flops
- This is easy to do but then we got greedy and added a lot of things
- Too many things to handle w/o software

The Major Players

- Sequential (or Simple) PLD (SPLD)
- Complex PLD (CPLD)
- Field Programmable Gate Array (FPGA)

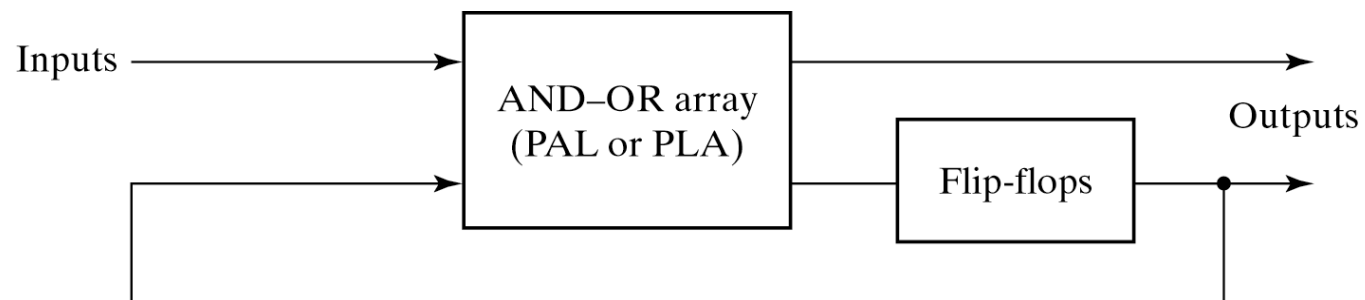


Fig. 7-18 Sequential Programmable Logic Device

Macrocell

- The basic component of most of them is the macrocell
- Consists of a F-F and its attendant combinational logic
- To make life more complex they can share implicants etc

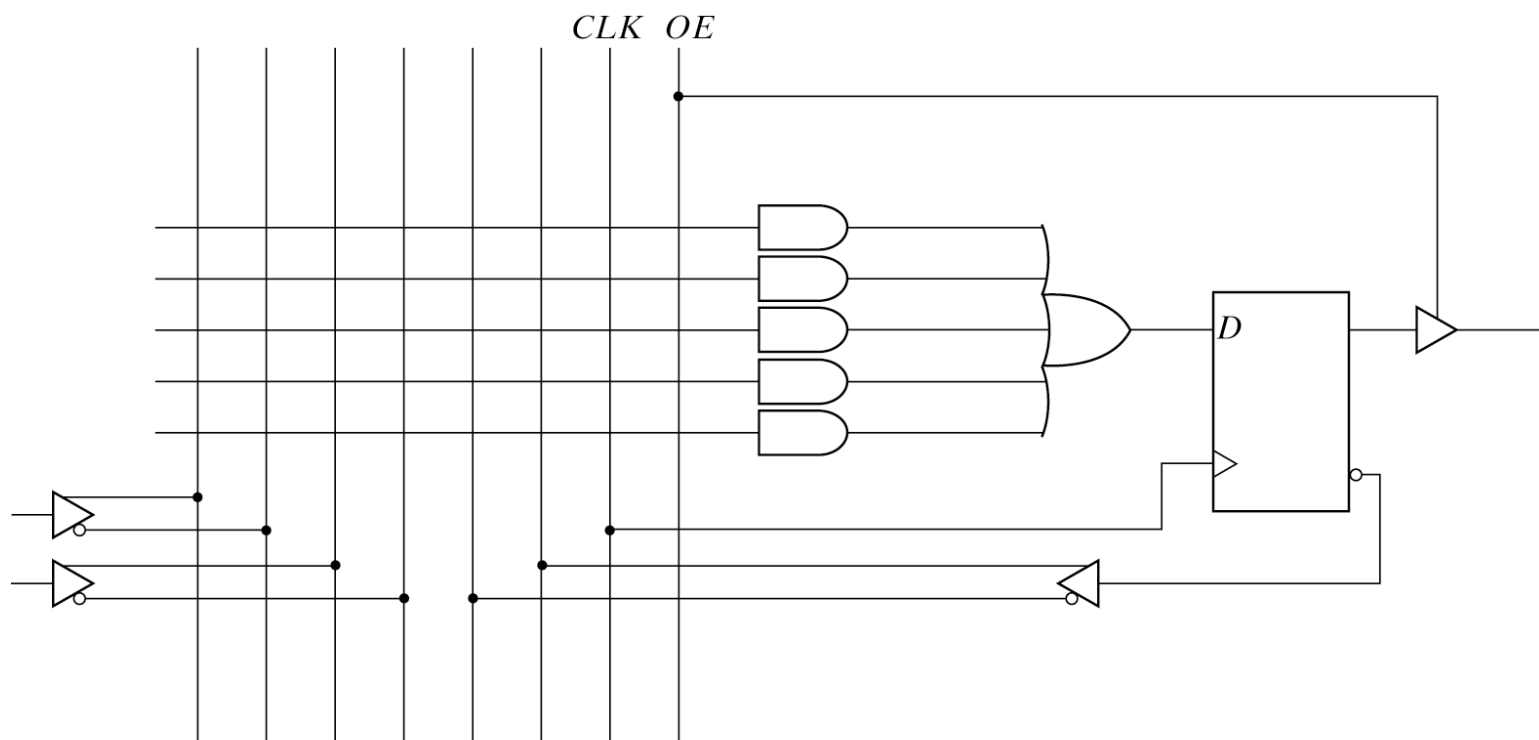


Fig. 7-19 Basic Macrocell Logic

The CPLD

- Many SPLDs connected to a programmable switch matrix
- I/O block (programmable)

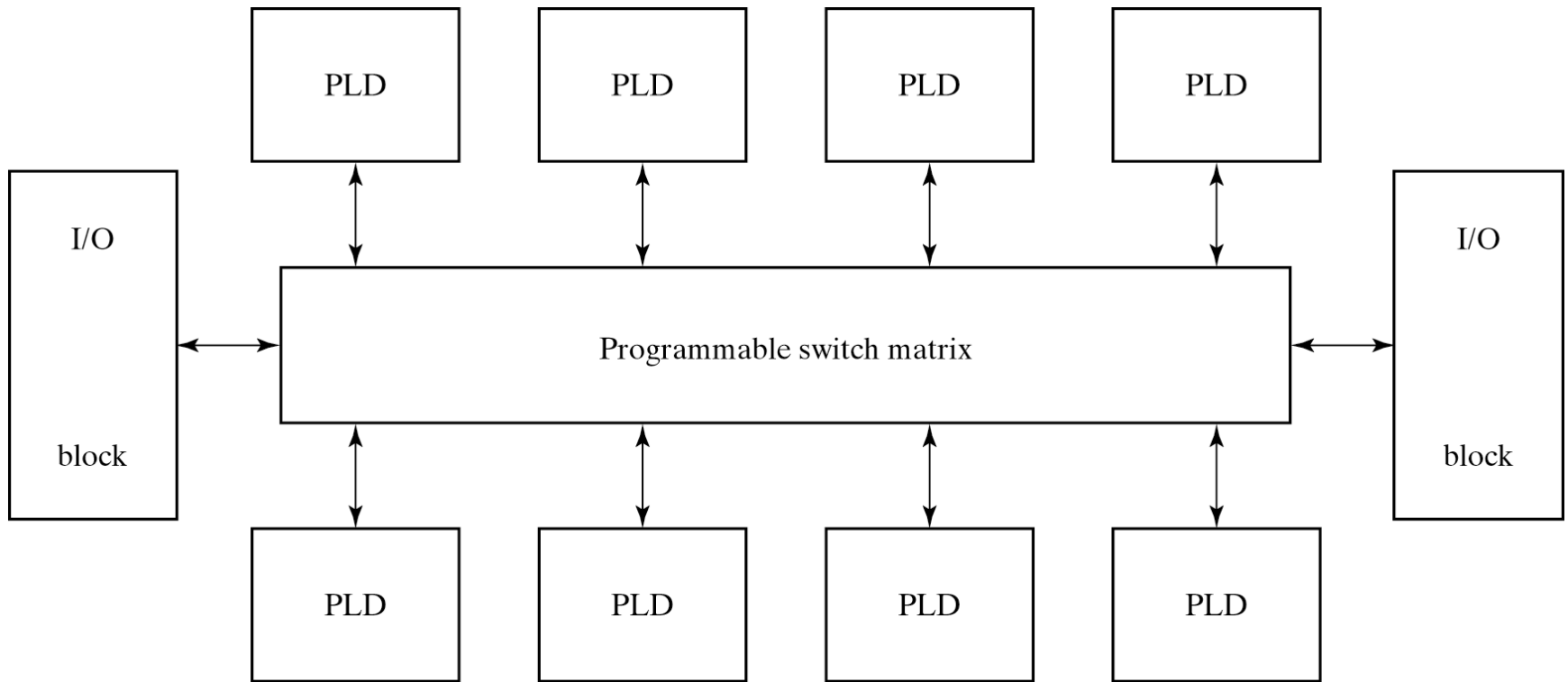


Fig. 7-20 General CPLD Configuration

FPGA

- A gate array with many extra circuits like look-up-tables, decoders etc
- Added components like PCI or USB controllers.

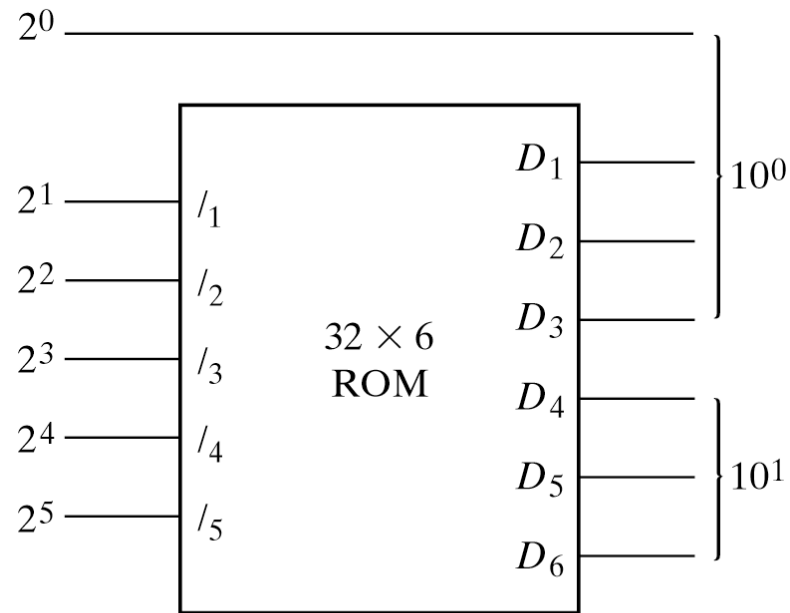


Fig. P7-17