Unit 5

Memory and Programmable Logic

Syllabus: RAM - Memory Decoding - Error Detection and Correction - ROM - Programmable Logic Array - Programmable Array Logic - Sequential Programmable Devices

Definitions:

① **Memory Unit:**

→ a collection of cells capable of storing a large quantity of binary information.

→ Information is transferred to the memory unit for storage and retrieved for processing.

→ There are 2 types of memories used in digital systems - random access memory (RAM) and read only memory (ROM)

② **Memory Write:** The process of storing new information into the memory is referred to as a memory-write operation.

③ **Memory Read:** The process of reading the stored information out of the memory is termed as a memory-read operation.

* **Types of memories: RAM vs. ROM**

(a) Random-Access Memory (RAM)

→ used in operating systems, application programs

→ Data which is in current use resides in the RAM for quick reach by the processor.

(b) Read-Only Memory (ROM)

→ content cannot be changed

→ contents retained even when shut off

→ used to start up computer and load the operating system into the RAM

→ can be used as a Programmable Logic Device (PLD).

→ different types of ROM include Mask Programming, PROM, EPROM, EEPROM, Flash Memory

* Random Access Memory

→ The architecture of RAM is such that data can be selectively retrieved from any of its internal locations.

→ The time taken to transfer information to or from any desired location is always the same - which is why the device is termed as Random Access Memory. This is in contrast to the fact that the time taken to retrieve information on a magnetic tape depends on the location of the data.

Information Storage in RAM

→ The memory unit of a RAM stores information in groups of bits called words

→ Word → a set of bits that moves in and out of storage as a unit.

→ The group of 0's and 1's in a word may represent a number, an instruction, one or more alphanumeric characters, or any other binary-coded information.

→ 8 bits = 1 byte

→ Computer memories use words that are multiples of 8 bits in length.

Capacity of a memory unit : The total number of bytes that the unit can store.

Communication in RAM :

→ Communication between the memory and the environment happens through

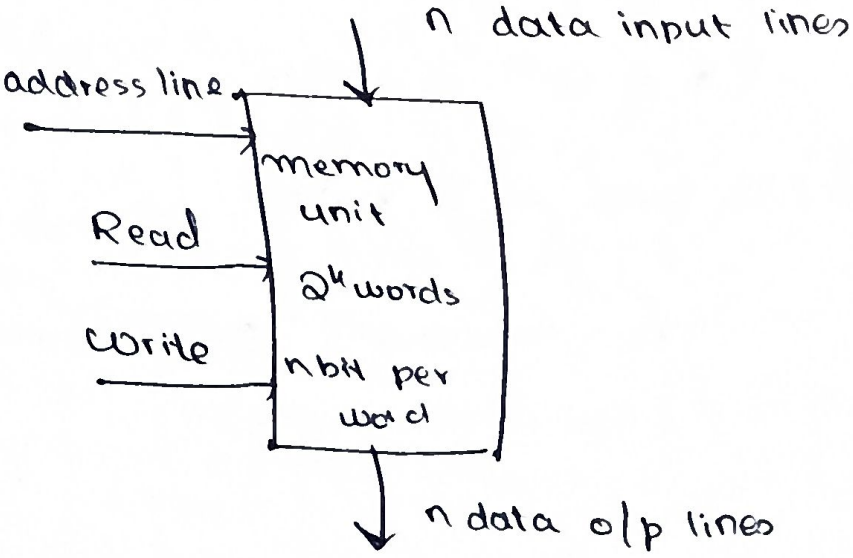
* data input and output lines
* address selection lines
* control lines specifying the direction of transfer

Block Diagram

→ n data i/p and o/p lines

→ k address lines specifies the particular word

→ 2 control inputs specify the direction of transfer



write op ⇒ binary data transferred into memory
read op ⇒ binary data transferred out of memory

Working

→ Each word has an address (like an id no)

→ Each address line selects a particular word to R/W operations

→ Selection of a word is done by applying the k-bit address to the address lines.

→ An internal decoder accepts this address and opens the paths needed to select the word specified.

→ Addresses go from $0-(2^k-1)$, where k is the number of address lines.

→ The number of words in the memory is referred to by

K → kilo → 2^{10}
M → mega → 2^{20}
G → giga → 2^{30}

Example: If a memory unit has a capacity of 1K words of 16 bits each, then

$1K = 2^{10} = 1024$

16 bits = 2 bytes

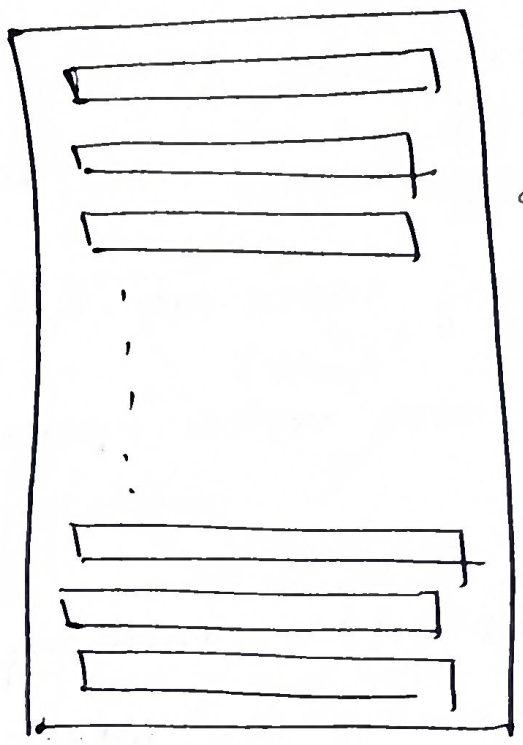
∴ the memory can accomodate 1024×2

$= 2048 = 2K$ bytes

Diagram

Memory Address	
Binary	Decimal
0000 000000	0
	1
	2
	.
	.
	.
1111 111101	1021
1111 111110	1022
1111 111111	1023

Memory content



write some decimal binary nos in boxes to rep words

Write and Read Operations on Ram

(i) To transfer a new word into the memory (WRITE)

1. Place address in address line
2. Place input data in data line
3. Activate the write control signal

(ii) To transfer a stored word out of the memory (READ)

1. Place address in address line
2. Activate the read control signal
3. Output will be in the data line

Control Inputs to Memory Chip

Memory Enable	Read / Write	Memory Operation
0	X	None
1	0	Write
1	1	Read

* Timing Waveforms

→ The operation of the memory unit is controlled by the central processing unit (CPU).

→ The CPU is synchronized by its own clock.

→ The memory does not use an internal clock.

→ Its read and write operations are specified by control inputs.

→ The CPU provides memory control signals in such a way so as to synchronize its internal clock with the read and write operations of the memory.

Consider the following example:

Let the CPU clock frequency be 50MHz

Let the memory cycle time be 50 ns..

$f_{CPU} = 50MHz \quad T_{CPU} = \frac{1}{50 \times 10^6} = 20ns$

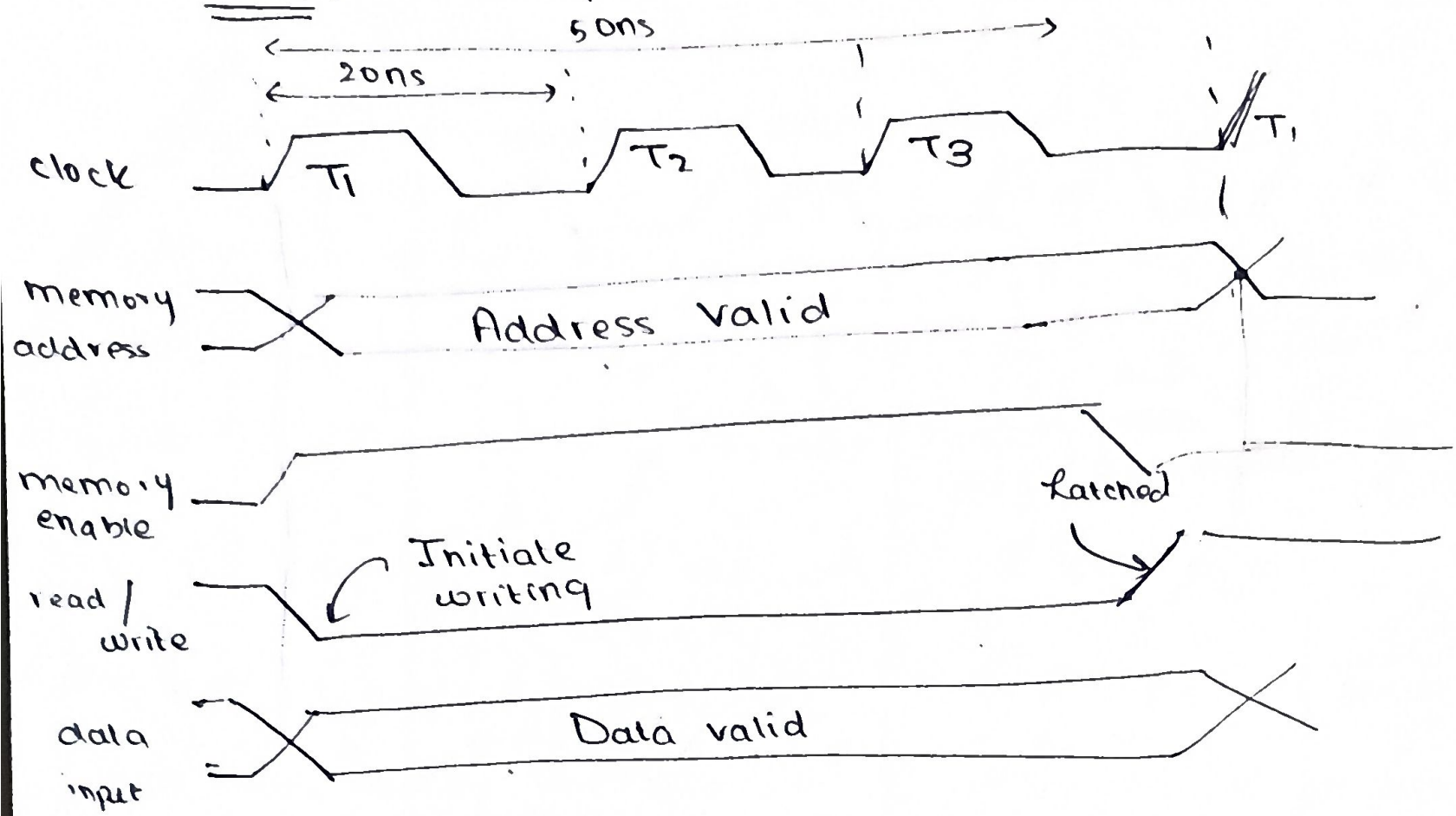
→ A 50ns memory cycle time means that

(i) the write cycle terminates the storage of given word within a 50ns interval

(ii) the read cycle always provides an o/p of the selected word in < 30ns.

→ The time period of the CPU is 20ns => 25 3 clock cycles are needed for each memory request.

Case 1: Write Operations



→ There are 3 20ns cycles T_1, T_2, T_3

→ To write, the address & input data is supplied. The crossed lines that cross in the address & data waveforms indicate a possible change in value.

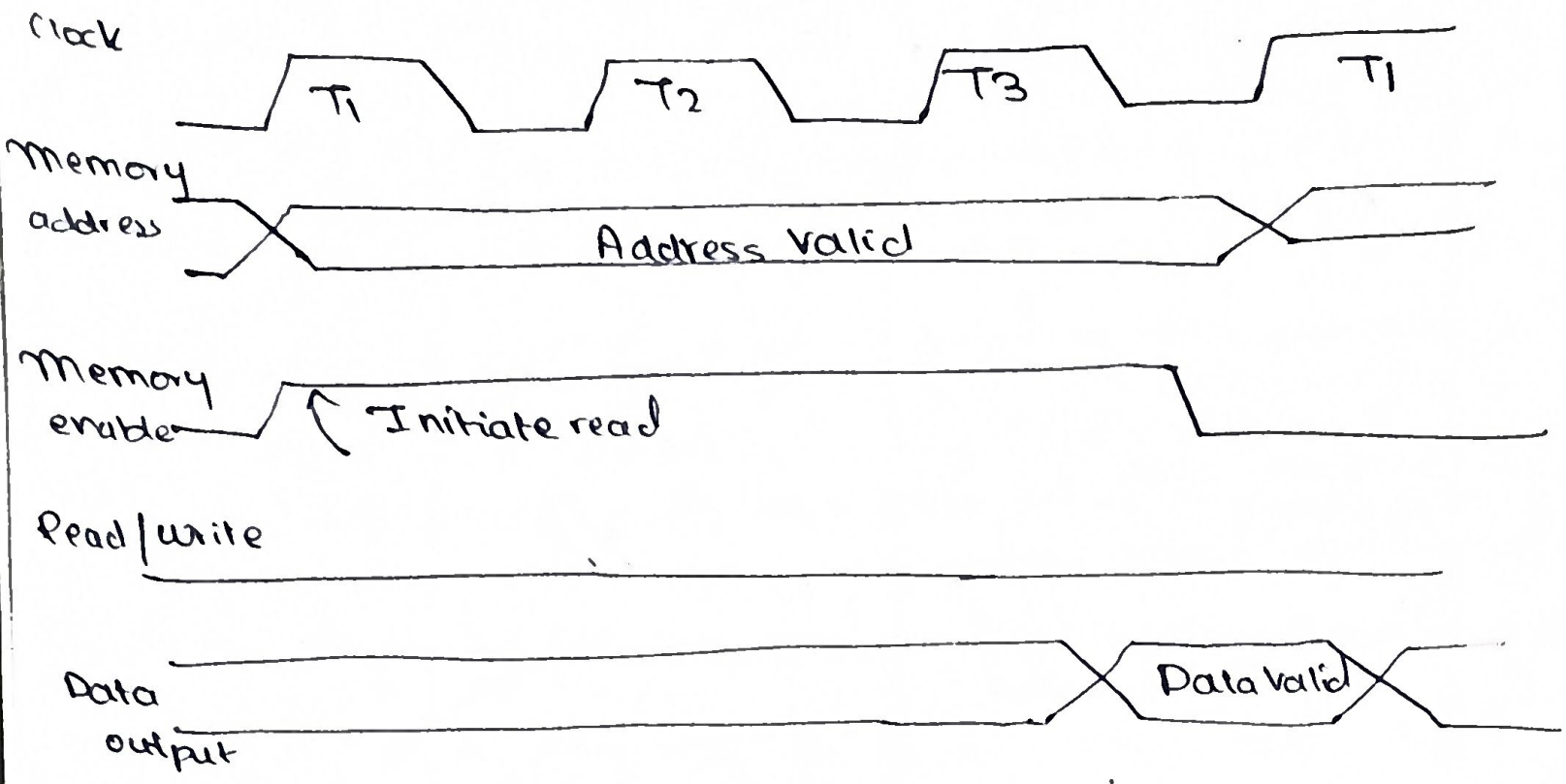
→ The memory enable signal switches to high

→ The read / write signals switches to low (0) to indicate a write operation

→ The address and data signals must remain valid for a short time after the control signals are deactivated.

→ After the third clock cycle is completed, the memory write operation is completed and the CPU can access the memory again from the next T_1 cycle.

Case 2: Read Operations



→ To perform read operations, the memory enable is set to high.

→ The read/write signal is also set to high

→ The memory places the data of the word selected by the address into the output data line within a 50 ns interval or less from the time that the memory enable is activated.

→ The CPU transfers the data into one of its internal registers during the negative transition of T3

→ The next cycle T_i is available for another memory request.

* Definitions

Access time → Time required by the memory to select a word & read it

Cycle Time → Time required to complete a write operation.

Types of Memories

(i) Sequential vs. Random Access memory

Sequential access memory → information stored is not immediately accessible, but is available only at certain intervals of time.

→ a magnetic disk or tape unit is of this type.

→ Each memory location passes the read and write heads in turn, but info. is only read out when the requested memory is reached.

→ i.e Time taken to access a word depends on the position of the word

Random Access memory → access time is always the same

→ word locations may be thought of as being separated by a space, each word occupying one particular location

(ii) Static and Dynamic Ram

Static RAM → has internal latches to store binary information

→ stored info retained as long as there is power supply

→ easier to use, has shorter read and write cycles.

Dynamic RAM → stores binary info in the form of electric charges on capacitors

→ the stored charges tend to discharge with time

→ periodic charging is done by refreshing

→ Refreshing is done by cycling through the coords every few milliseconds

→ offers reduced power consumption and larger storage capacity in a single memory chip.

(iii) Volatile and Non Volatile RAM

Volatile → lose stored information when power is turned off

→ binary cells need external power to maintain the stored information.

Non-Volatile → retains stored information even after removing power

→ includes magnetic disks, ROM, hard disks, CDs.

* Memory Decoding

→ There is a need for decoding circuits to select the memory word specified by the input address.

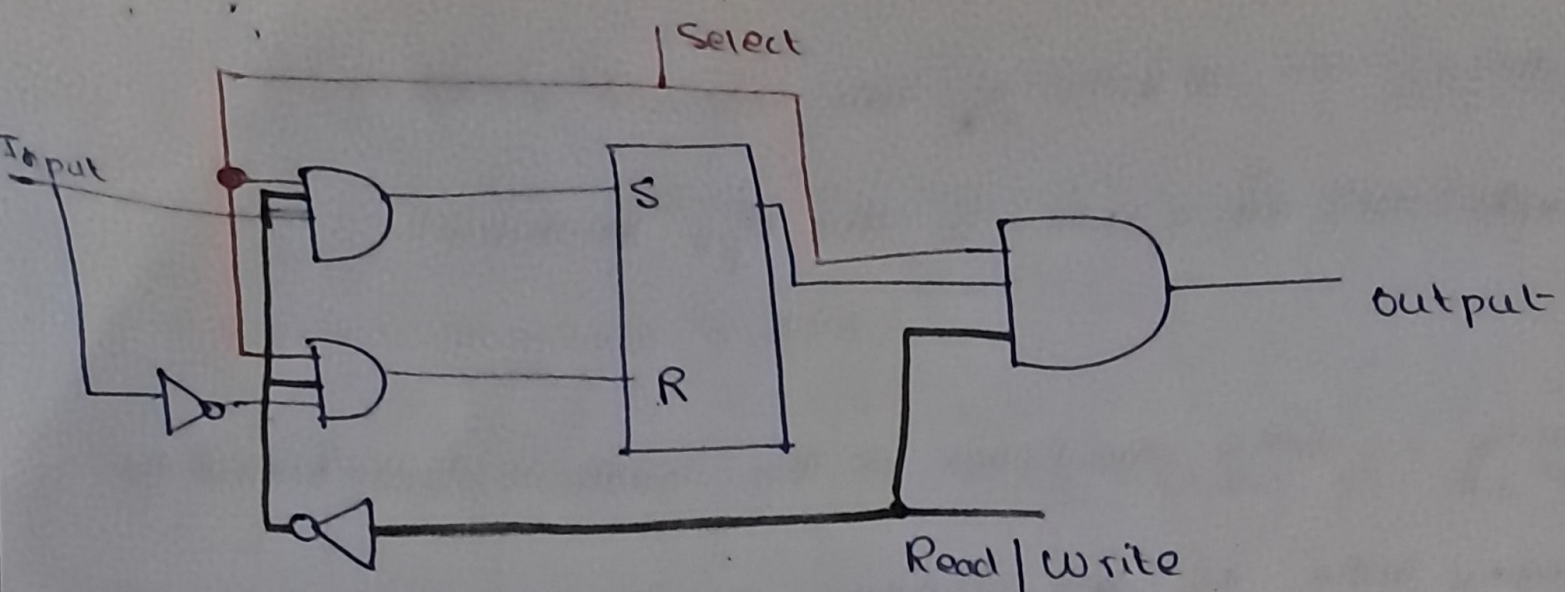
→ The internal construction of RAM with m words and n bits per word consists of m x n binary storage cells

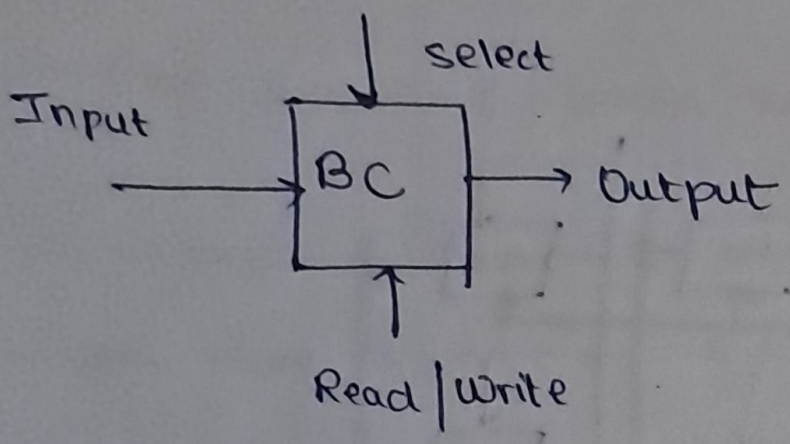
→ The storage part of the binary cell is modeled using an SR latch with the associated gates to form a D latch.

→ The internal latch stores 1 bit of information

→ The select input enables the cell for reading & writing.

→ The read/write input determines the operation of the cell.





Consider the following example:

there are 4 words $(m = 4)$

each word is four bits $(n = 4)$

∴ Total no. of binary cells $= m \times n = 16$

No. of outputs = 4

no. of address lines = 2

→ Each binary cell has 3 inputs and 1 output

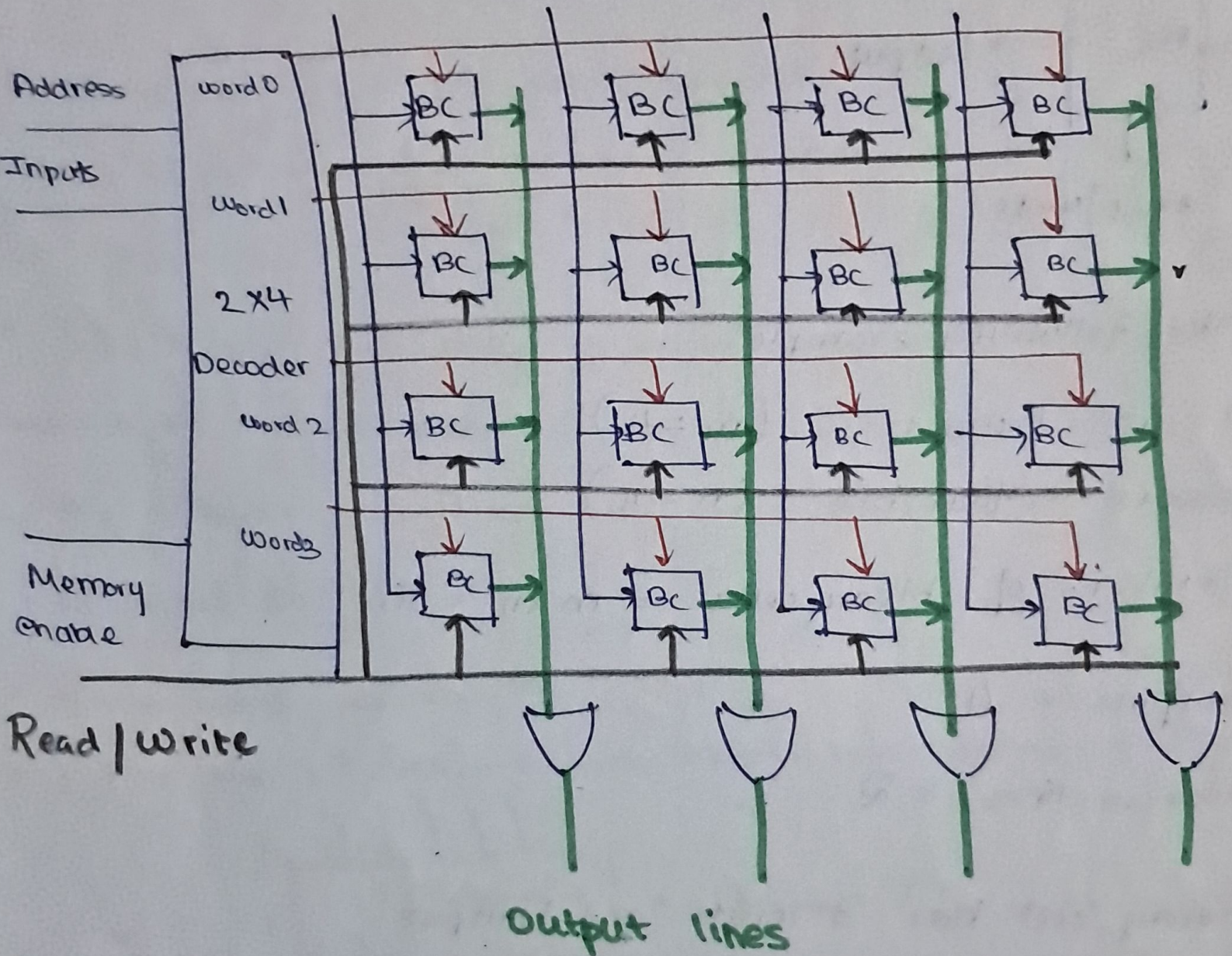
→ When the memory is 0, all outputs of the decoder are 0.

→ When the memory select is 1, one of the words is chosen, depending on the address line values.

→ Once a word is chosen, the read / write input determines the operation.

Read Operations : 4 bits of the selected word go through the OR gates to the o/p terminal

Write Operations : data available in the input is transferred to to the 4 binary cells of the selected word.



* Coincident Decoding

→ A decoder with k inputs would ~~require~~ have 2^k outputs, and thereby require 2^k AND gates in all.

→ The total no. of gates and the no. of inputs per gate can be reduced by employing 2 decoders in a 2 dimensional selection scheme.

→ k/2 input decoders are used instead of one k-input decoder.

→ One decoder performs row selection, the other column selection.

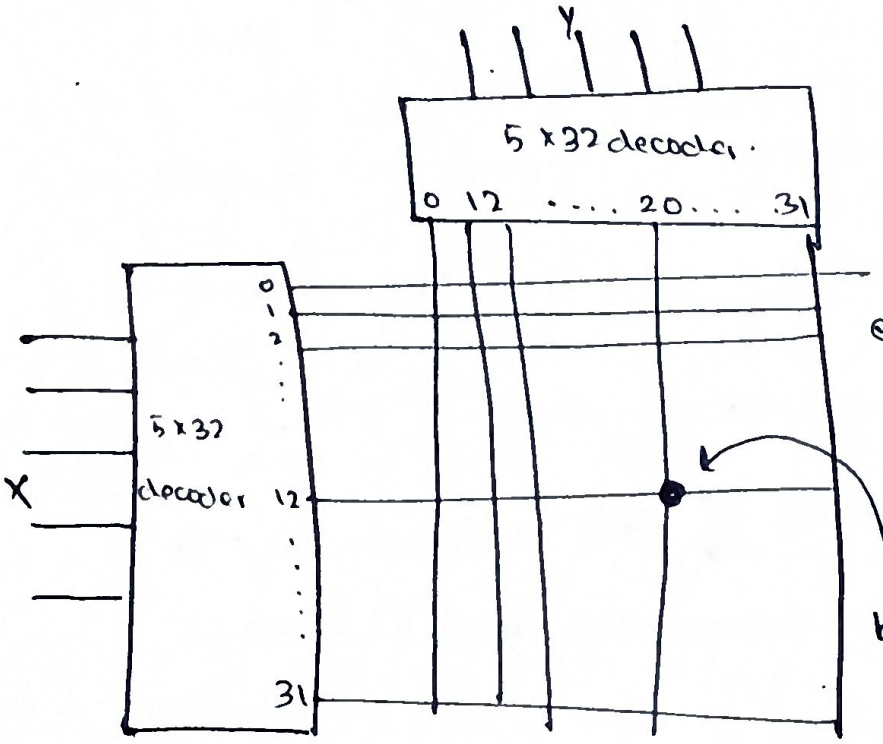
For example, for a 1K word memory:

→ rather than using a 10 x 1024 decoder, 2 5 x 32 decoders be used

→ This means that only 64 AND gates with 5 inputs each would be needed over 1024 AND gates with 10 inputs each.

→ The 5 most significant bits go to input X, and the 5 least significant go to Y.

→ Each word within the memory is selected by the coincidence of 1 X line and 1 Y line, i.e each intersection represents a word



for eg. if a word has the address 404, the 10 bit binary equivalent is 01100 10100

⇨ X = 01100 = 12

Y = 10100 = 20

* Address Multiplexing

→ SRAM memory cells have 6 transistors.

→ To build memories with a higher density, it is necessary to reduce the no. of transistors in a cell.

→ The DRAM cell has a single MOS transistor and a capacitor.

→ The charge stored on the capacitor discharges with time, & the memory cells are periodically recharged by refreshing the memory.

→ The density of DRAMs = 4 X SRAMs

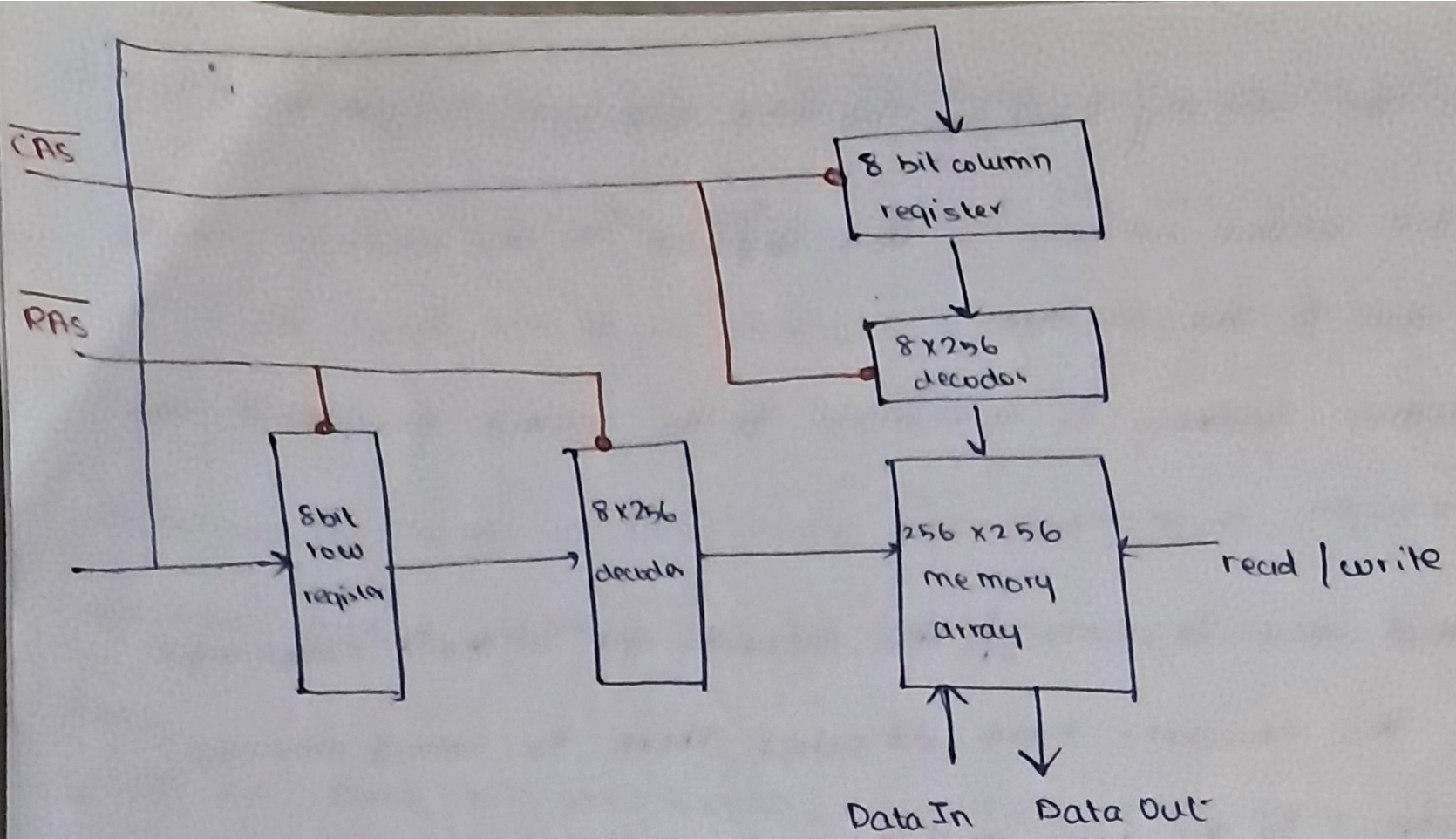
→ DRAMs also cost 4 times less, and have a lower power requirement.

→ Because of the large capacity, the address decoding of DRAMs is arranged into a 2 dimensional array, and larger memories often have multiple arrays.

→ To minimize the no. of pins, designers use address multiplexing whereby one set of address input pins accomodates the address components.

→ In a 2 dimensional array, the address is applied in 2 parts at 2 different parts, first the row address and then the column address.

→ The no. of pins used for both parts of the address are the same. ⇒ the size of the package decreases significantly.



Consider a 64K memory.

→ The memory has a 2D array of cells arranged into 256 rows & columns

→ There are 2 8X256 decoders.

→ There is a single data input line, and a single output line

→ There is also a read/write control.

→ There is an 8 bit address input, and two address strobes to enable the rows and columns addresses into their respective registers.

→ The RAS = Row Access Strobe enables the 8 bit row register

→ The CAS = Column Address Strobe enables the 8 bit column register.

→ The bar above the RAS & CAS indicates that the registers are enabled on the zero level of the signal.

Working : → Initially, both probes are in 1-state

→ The 8 bit row address is applied to the row register address input, RAS is turned to 0.

→ The corresponding decoder is also enabled, and one row is selected

→ After the settling time of the Row selection, it goes back to ...

→ The 8 bit column address is then applied to the address inputs, and CAS goes to the 0 state.

→ The column address is transferred to the column register & the column decoder is enable.

→ Now that the 2 parts of the address are in their respective registers, the decoders have decoded them to select one cell corresponding to the row and column address.

→ A read or write operation can be performed on that cell.

→ CAS must go back to 1, before initiating another memory operation.

Error Detection and Correction - Hamming Code

→ The dynamic physical interaction of the electrical signals affecting the data path may cause occasional errors in storing and retrieving binary information.

Error Detecting & Correcting Schemes

Error Detecting Scheme: parity bit

→ A parity bit is generated and stored along with the data word in the memory.

→ After reading the word, the parity is checked.

→ If the parity bits are correct, then the word is accepted.

→ This technique can only detect errors, not correct them.

Error Correcting Code → Multiple parity check bits are stored with the data word in the memory.

→ Each check bit is a parity over a group of bits in the data word.

→ When the word is read back from the memory, the parity bits are checked with a set of check bits generated from the data.

→ If the check bits are correct, then there is no error. If they do not match, they generate a unique pattern called a syndrone that can be used to identify the bit that is in error.

*Hamming Code

→ Devised by R.W. Hamming

→ If there is an n bit word, k parity bits are added

→ A new word of n+k bits is formed.

→ The relation between n and k is:

$$2^k - 1 \geq n + k$$

→ The positions numbered as powers of 2 are reserved for the parity bits.

→ The remaining bits are data bits.

For example, consider an 8 bit word : 11008100.

$n = 8$

$2^k - 1 \geq n + k$

$2^k - 1 \geq 8 + k$

$\Rightarrow \boxed{k = 4}$

4 parity bits are required.

Total no. of bits = 12

Decimal	P_8C_8	P_4C_4	P_2C_2	P_1C_1
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
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

$C_1 = \text{XOR}(1,3,5,7,9,11)$

$C_2 = \text{XOR}(2,3,6,7,10,11)$

$C_4 = \text{XOR}(4,5,6,7,12)$

$C_8 = \text{XOR}(8,9,10,11,12)$

$P_1 = \text{XOR}(3,5,7,9,11)$

$P_2 = \text{XOR}(3,6,7,10,11)$

$P_4 = \text{XOR}(5,6,7,12)$

$P_8 = \text{XOR}(9,10,11,12)$

The value of each parity bit is calculated as follows (Refer table below)

P_1 = XOR (3, 5, 7, 9, 11) ⇒ 0
1 1 0 0 0

P_2 = XOR (2, 3, 6, 7, 10, 11)
1 0 0 1 0 ⇒ 0

P_4 = XOR (4, 5, 6, 7, 12)
1 0 0 0 ⇒ 1

P_8 = XOR (8, 9, 10, 11, 12)
0 1 0 0 ⇒ 1

Bit position

1	2	3	4	5	6	7	8	9	10	11	12
P_1	P_2	1	P_3	1	0	0	P_4	0	1	0	0
0	0		1				1				

The corresponding Hamming code is

001110010100

Hamming Code Check

C_1 = XOR (1, 3, 5, 7, 9, 11) ⇒ 0
0 1 1 0 0 0

C_2 = XOR (2, 3, 6, 7, 10, 11) ⇒ 0
0 1 0 0 1 0

C_4 = XOR (4, 5, 6, 7, 12) ⇒ 0
1 1 0 0 0

C_8 = XOR (8, 9, 10, 11, 12) ⇒ 0
1 0 1 0 0

$C = C_8 C_4 C_2 C_1 \Rightarrow 0000 \Rightarrow$ No error

Example:

Find the error, if any, and the bit at which the error is present in the following Hamming codes

Ex1 Bit position

1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	0	0	1	0	1	0	0

C_1 = XOR (1, 3, 5, 7, 9, 11) ⇒ 1
(1 1 1 0 0 0)

C_2 = XOR (2, 3, 6, 7, 10, 11) ⇒ 0
(0 1 0 0 1 0)

C_4 = XOR (4, 5, 6, 7, 12) ⇒ 0
(1 1 0 0 0)

C_8 = XOR (8, 9, 10, 11, 12) ⇒ 0
(1 0 1 0 0)

$C = C_8C_4C_2C_1 = 0001$ ⇒ Error in bit 1

Ex2 Bit Position

1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	0	1	0	1	0	0

C_1 = XOR (1, 3, 5, 7, 9, 11) ⇒ 1
(0 1 0 0 0 0)

C_2 = XOR (2, 3, 6, 7, 10, 11) ⇒ 0
(0 1 0 0 1 0)

C_4 = XOR (4, 5, 6, 7, 12) ⇒ 1
(1 0 0 0 0)

C_8 = XOR (8, 9, 10, 11, 12) ⇒ 0
(1 0 1 0 0)

$C = C_8C_4C_2C_1 = 0101$ ⇒ Error in bit 5

* Single Error Correction, Double Error Detection

→ By adding another parity bit to the coded word, the Hamming code can be used to correct a single error and detect multiple errors.

→ The additional parity bit is evaluated as the X-OR of all the other bits.

eg.

If the Hamming code is:

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

then $P_{13} = 1$

This becomes a check for even parity

During checking

if P=0 ⇒ correct

P=1 ⇒ incorrect

The following 4 cases arise:

C	P_{13}	
0	0	→ no error
0	1	→ error in P_{13}
1	0	→ double error
1	1	→ single error

Example Questions

(1) Given the 8 bit word 01011011, generate the 13 bit composite word for the Hamming code that corrects single errors and detects double errors.

P_1 = XOR (3, 5, 7, 9, 11) => 0
0 1 1 1 1

P_2 = XOR (3, 6, 7, 10, 11) => 0
0 0 1 0 1

P_4 = XOR (5, 6, 7, 12) => 1
1 0 1 1

P_8 = XOR (9, 10, 11, 12) => 1
1 0 1 1

Bit position	1	2	3	4	5	6	7	8	9	10	11	12	13
	A	P_2	D	P_4	D	D	D	P_8	D	D	D	D	P_{13}
Data			0		1	0	1		1	0	1	1	
Parity	0	0		1				1					
Hamming Code	0	0	0	1	1	0	1	1	1	0	1	1	1

∴ The 13 bit composite word is 0001 1011 1011 1

(2) Obtain the 15 bit Hamming code word for the 11 bit data word 11001001010

Ans: n = 11

$2^k - 1 \geq n + k$

$2^k - 1 \geq 11 + k$

$k = 4$

∴ Total no. of bits required = $n + k = 11 + 4 = 15$

Dec.

Decimal	P_8C_8	P_4C_4	P_2C_2	P_1C_1
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
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

$C_1 = XOR\ (1,3,5,7,9,11,13,15)$

$C_2 = XOR\ (2,3,6,7,10,11,14,15)$

$C_4 = XOR\ (4,5,6,7,12,13,14,15)$

$C_8 = XOR\ (8,9,10,11,12,13,14,15)$

$P_1 = XOR\ (3,5,7,9,11,13,15)$

$P_2 = XOR\ (3,6,7,10,11,14,15)$

$P_4 = XOR\ (5,6,7,12,13,14,15)$

$P_8 = XOR\ (9,10,11,12,13,14,15)$

Bit Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	P_1	P_2	D	P_4	D	D	D	P_8	D	D	D	D	D	D	D
Data			1		1	0	0		1	0	0	1	0	1	0
Parity	1	0		1				1							
Hamming Code	1	0	1	1	1	0	0	1	1	0	0	1	0	1	0

P_1 = XOR (3,5,7,9,11,13,15) ⇒ 1
1 1 0 1 0 0 0

P_2 = XOR (3,6,7,10,11,14,15) ⇒ 0
1 0 0 0 0 1 0

P_4 = XOR (5,6,7,12,13,14,15) ⇒ 1
1 0 0 1 0 1 0

P_8 = XOR (9,10,11,12,13,14,15) ⇒ 1
1 0 0 1 0 1 0

∴ The 15 bit Hamming Code coord is : 101 110 011 001 010

Ex3 A 12-bit Hamming code word containing 8 bits of data & 4 parity bits is read from memory. What was the og 8 bit data word?

(a) 0000 1010 1010

Bit Position	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	D	P_4	D	D	D	P_4	D	D	D	~~D~~ P_8
	0	0	0	0	1	0	1	0	1	0	1	0

Original word = ~~0 1 0 1 0 1 0 1~~ 0101 1010

(b) 0111 0110 1111

Bit position	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	D	P_4	D	D	D	P_8	D	D	D	D
	0	1	1	1	0	1	1	0	1	1	1	1

original word: 1011 1111

Eg4: How many parity check bits must be included with the data word to achieve single error correction and double error detection. when the data coord contains

(a) 24 bits

$n = 24$

$2^k - 1 \geq n + k$

$2^k - 1 \geq 24 + k$

$\boxed{k = 5}$

$\therefore$ 5 check bits + 1 = 6 parity bits

(b) 32 bits

$n = 32$

$2^k - 1 \geq n + k$

$2^k - 1 \geq 32 + k$

$2^6 - 1 \geq 32 + 6$

$\boxed{k = 6}$ $\Rightarrow$ 7 parity bits

(c) 64 bits

$2^7 - 1 \geq 64 + 7$

= 7 check bits + 1 parity bit

= 8 parity bits

* Read-Only Memory (ROM)

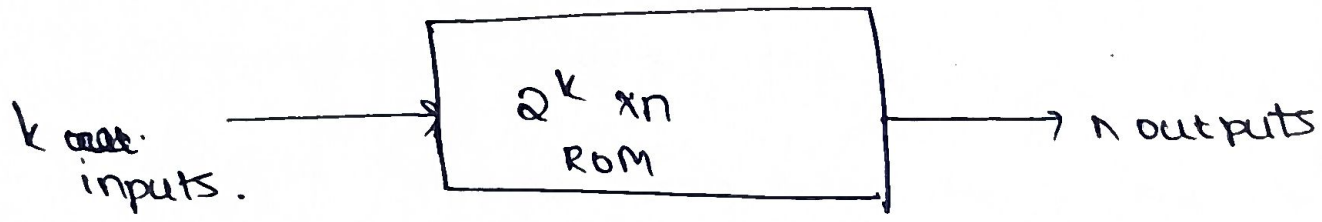
→ A memory device in which permanent binary information is stored.

→ The binary information is specified by the designer and is then embedded in the unit to form the required interconnection pattern.

→ Once the pattern is established, it stays within the unit even when the power is turned off and on again.

→ If there are 2^k words, then k address lines are needed.

→ Each word has n bits.



Example: Consider a 32 x 8 ROM

$n = 8$ = no. of bits per word

no. of words = 32

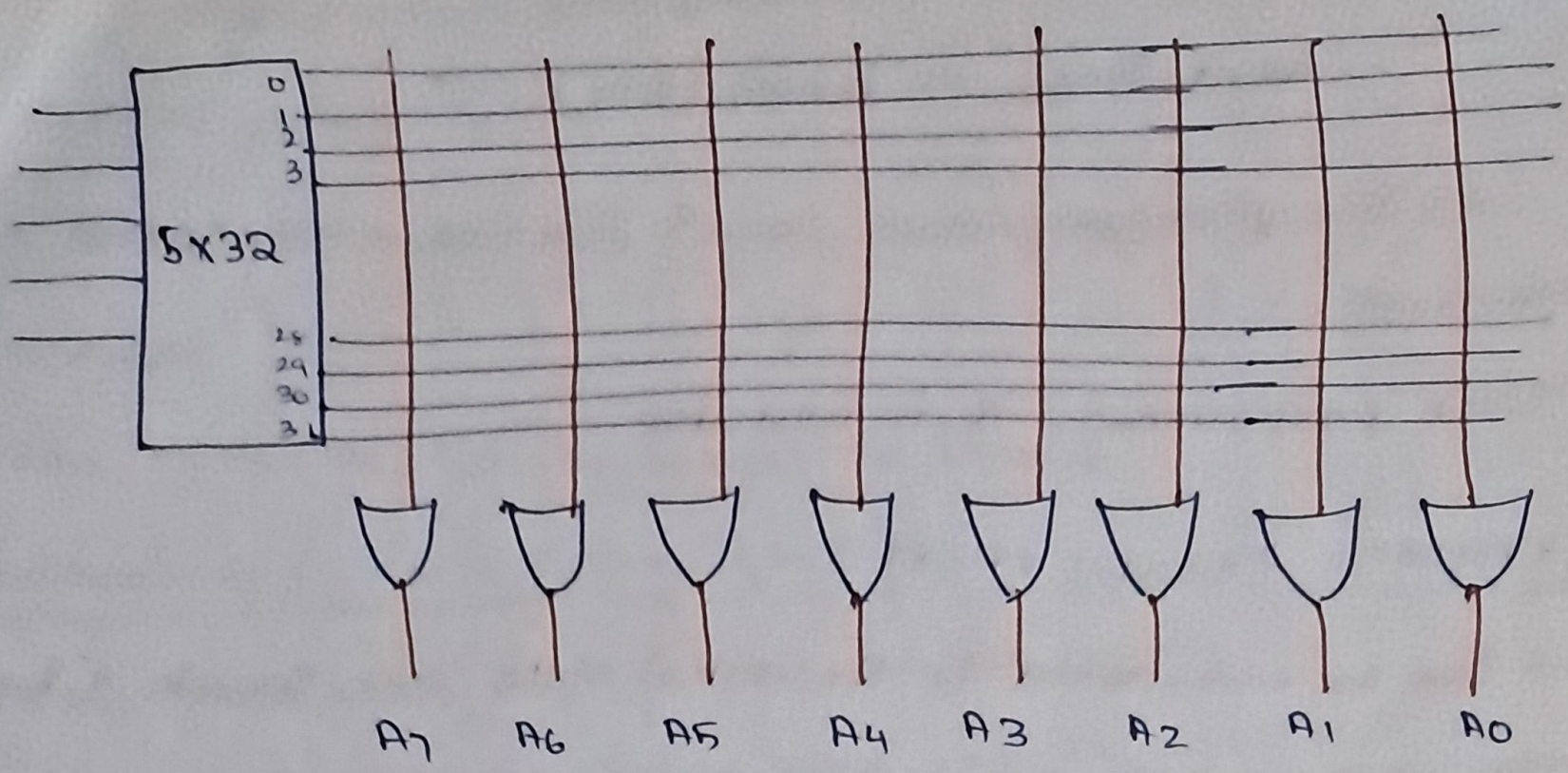
no. of address lines = $k = 5$ $(2^5 = 32)$

→ The 5 inputs are decoded into a 32 outputs using a 5 x 32 decoder.

→ Each of the 32 outputs are connected to 8 OR Gates.

In general

k = no. of inputs
2^k = no. of words
$k \times 2^k$ = decoder
n = no. of outputs = no. of OR Gates



* Types of ROMS

① Mask Programming

→ programming done by the semiconductor company
→ customer fills out truth table he wishes the ROM to satisfy
→ the manufacturer makes the corresponding mask for the paths to produce 0s and 1s according to the truth table
→ Expensive, since the ROM is custom made
→ Economical only if a large no. of the same configuration are ordered.

② PROM - Programmable Read Only Memory

→ Programming done with fuses at the intersections
→ Initially, all the fuses are intact, all are 1.
→ The fuses are blown by the application of a high-voltage pulse through a pin.

→ blown fuse ⇒ binary state 0

Intact fuse ⇒ binary state 1

→ This procedure allows users to program, using special instruments.

→ Once programmed, it is reversible.

③ EPROM - Erasable PROM

→ Can be restructured to the initial state even though it has been programmed previously.

→ When placed under a special UV light, the shortwave radiation discharges the internal floating gates, that serve as programmed connection

→ A new set of values can be then programmed.

④ EEPROM or E^2ROM - Electrically Erasable PROM

→ Similar to an EROM

→ Previously programmed connections can be erased with an electrical signal instead of UV light.

→ The advantage is that the device can be erased without removing it from its socket.

⑤ Flash Memory Devices

→ Similar to EEPROMs, but have additional built-in circuitry to selectively program and erase the circuit, without the need of a special programmer.

→ Have widespread application in modern technology like cell phones, digital cameras, microcontrollers.

→ Attractive storage for laptops, due to their low power consumption.

→ like EEPROMs, can be subject to fatigue.

* Programmable logic Devices (PLD)

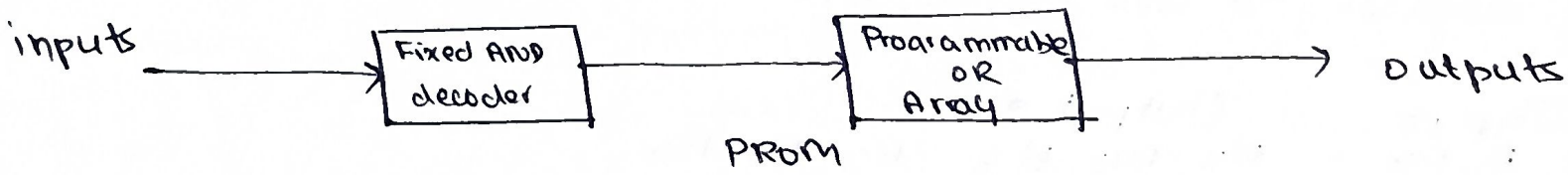
→ Electronic component used to build digital circuits

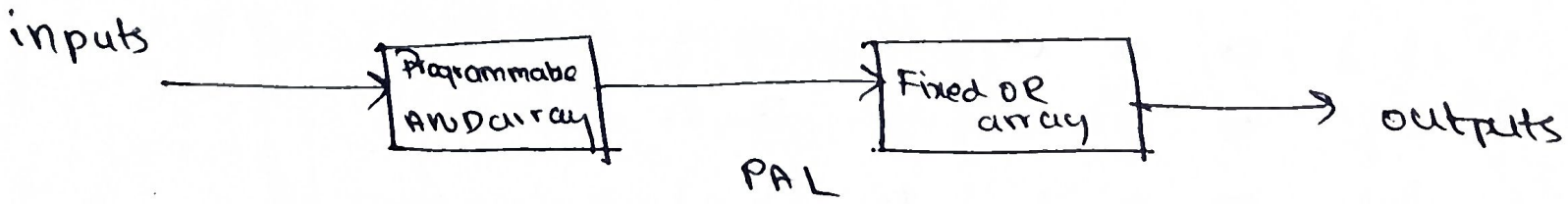
→ It must be programmed before being used in a circuit.

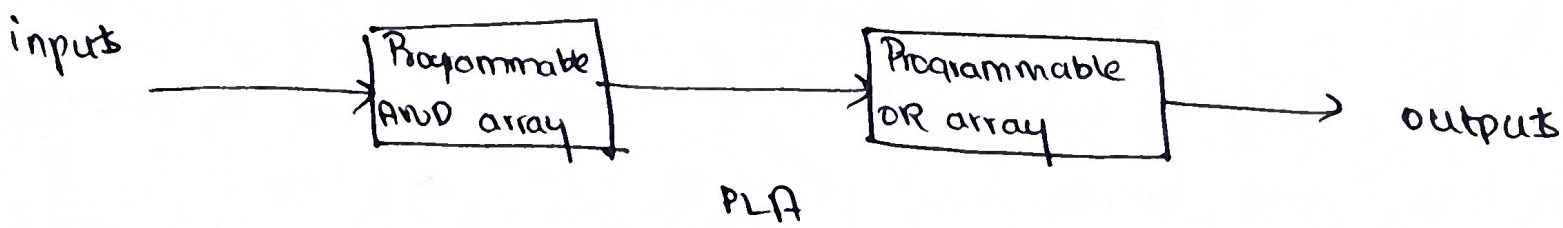
Types of PRDs

- PROM
- Programmable logic Array (PLA)
- Programmable Array logic (PAL)
- Field programmable gate array (FPGA)

Configuration of PLDs







(1) Programming PROMs

→ The intersections of the output of the decoder, and the inputs to the OR gates are programmable.

→ They are equivalent to switches and are called crosspoints.

→ Various physical devices are used to implement crosspoint switches, out of which the simplest is using fuses.

→ Fuses are blown by the application of a high-voltage pulse.

Example : Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and outputs a binary number equal to the square of the input number.

Ans: no. of input lines : 3 $k=3$

no. of output lines : 6

type of decoder to use: $2^k \times n$

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

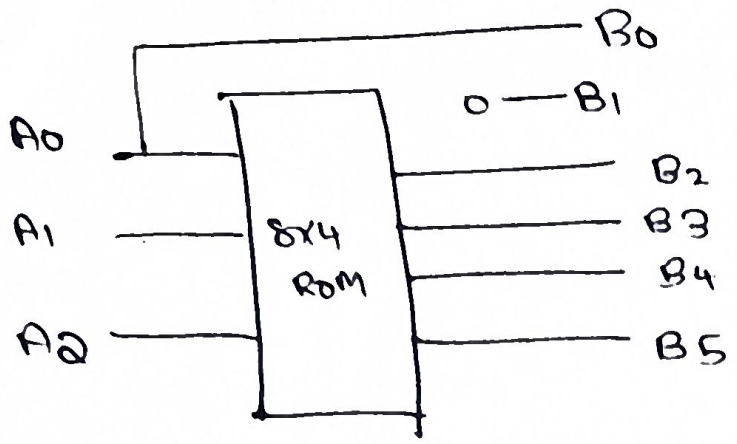
$\Rightarrow$ no. of outputs are 4

Type of ROM

$2^k \times 4$

8 × 4 ROM

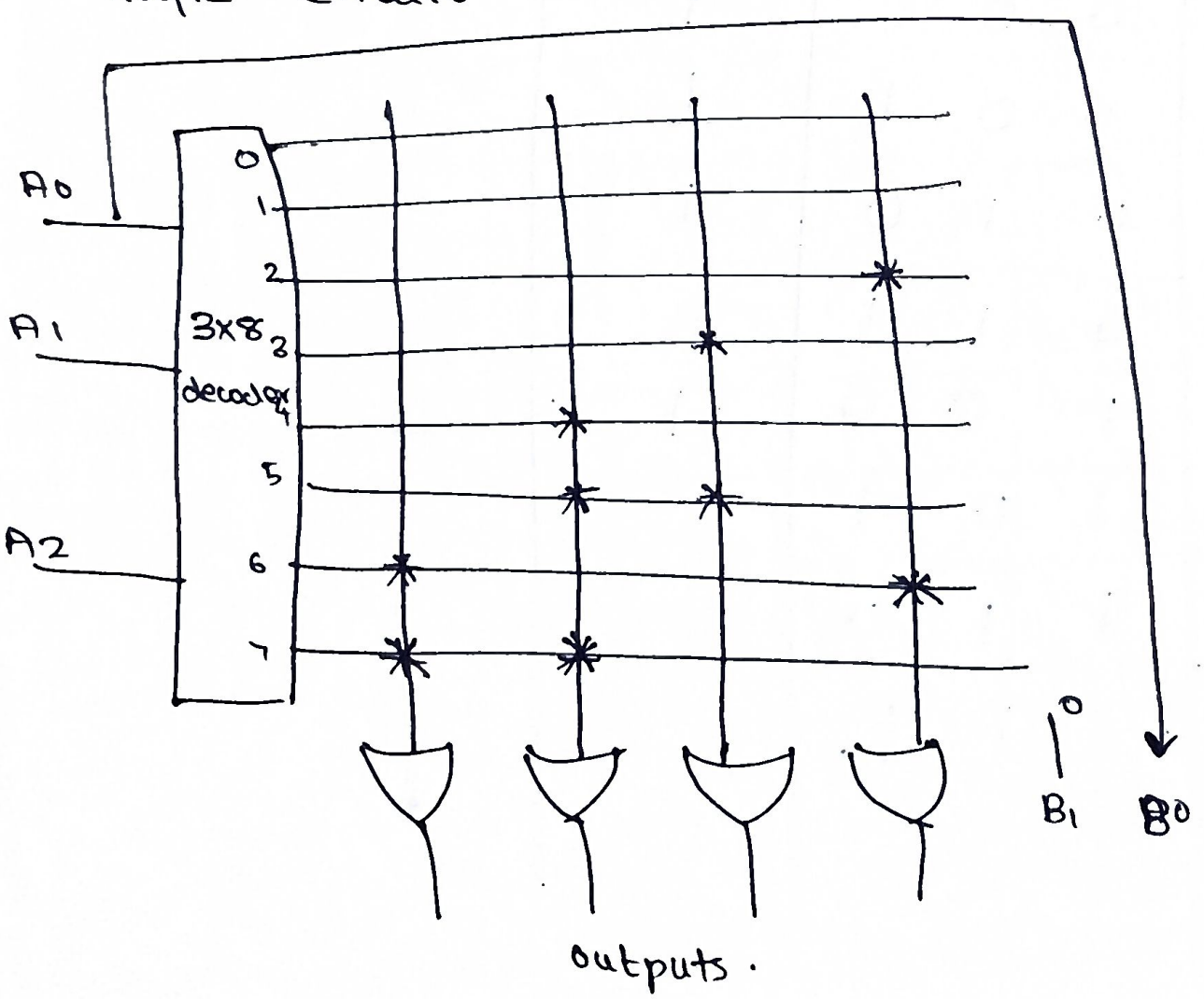
Block Diagram



ROM Truth Table

A_2	A_1	A_0	B_5	B_4	B_2	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	0	0	0

Logic Circuit

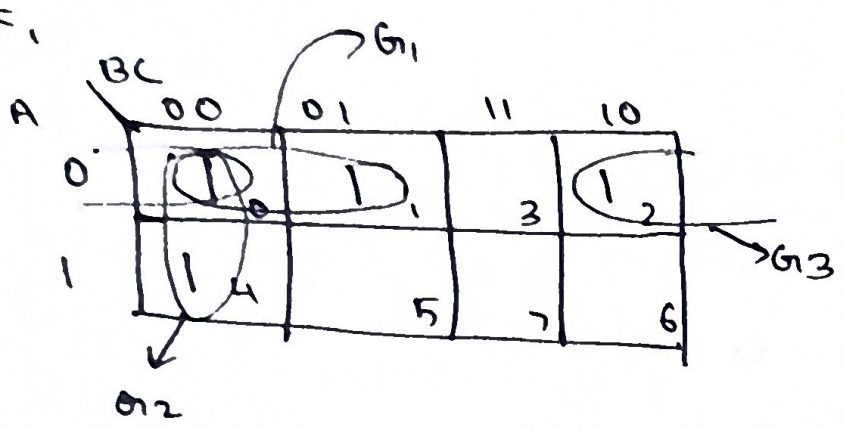


Eg2 Construct a combinational circuit using ROM that implements the following Boolean functions.

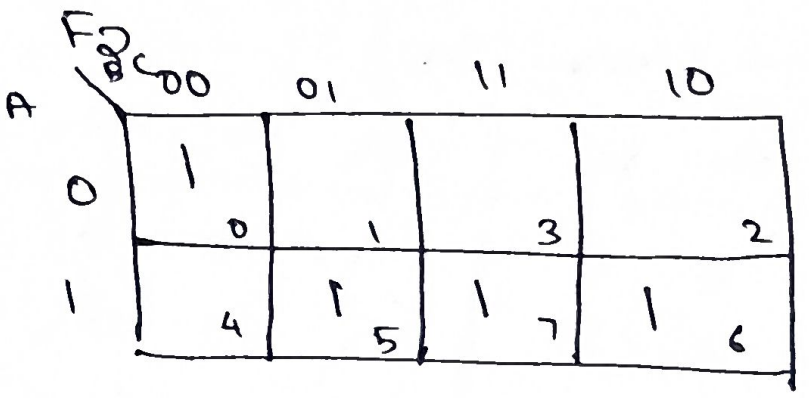
$F_1(A,B,C) = \Sigma(0,1,2,4)$

$F_2(A,B,C) = \Sigma(0,5,6,7)$

F1



F2



G_1: A B C
0 0 0
0 0 1

$\bar{A}\bar{B}$

G_2: A B C
0 0 0
1 0 0

$\bar{B}\bar{C}$

G_3: A B C
0 0 0
0 1 0

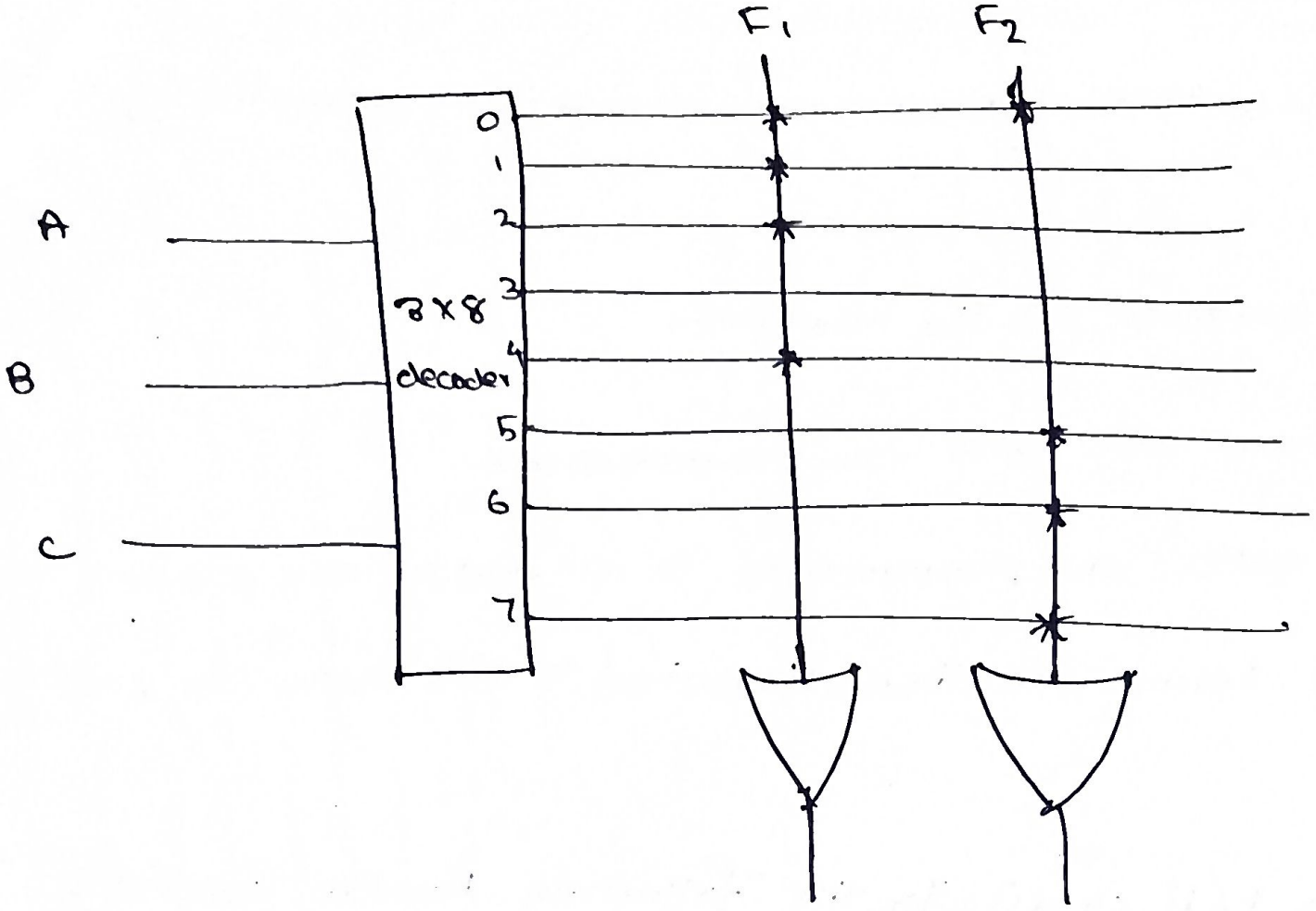
$\bar{A}\bar{C}$

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

Not necessary to simplify

A BC	F_1	F_2
0 0 0	1	1
0 0 1	1	–
0 1 0	1	–
0 1 1	–	–
1 0 0	1	–
1 0 1	–	1
1 1 0	–	1
1 1 1	–	1

ROM Design



Ex3 Design a ROM that implements the following Boolean functions

$A(x,y,z) = \Sigma(1,2,4,6)$

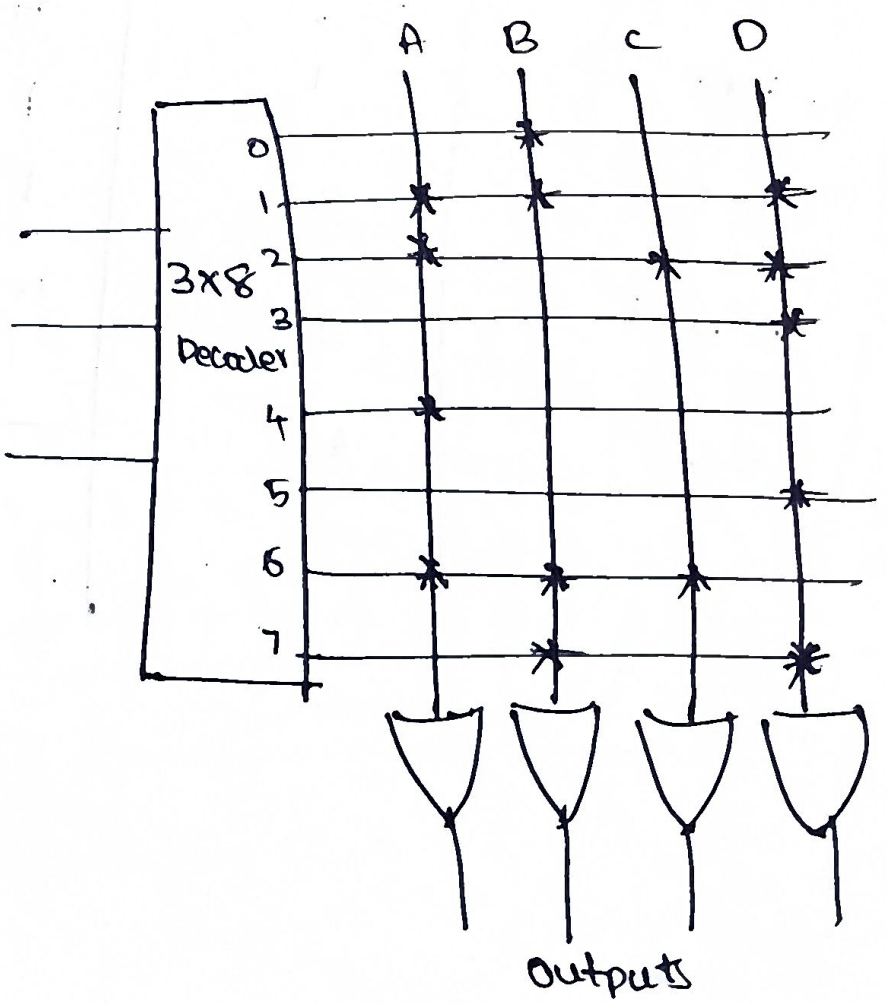
$B(x,y,z) = \Sigma(0,1,6,7)$

$C(x,y,z) = \Sigma(2,6)$

$D(x,y,z) = \Sigma(1,2,3,5,7)$

x y z	A	B	C	D
0 0 0	0	1	0	0
0 0 1	1	1	0	1
0 1 0	1	0	1	1
0 1 1	0	0	0	1
1 0 0	1	0	0	0
1 0 1	0	0	0	1
1 1 0	1	1	1	0
1 1 1	0	1	0	1

ROM Design



② Programmable Logic Array (PLA)

→ Similar to a PROM, but does not provide full decoding of variables.

→ does not generate all the minterms

→ the AND and OR gates are programmable

→ the AND gates are programmed to generate any product term

→ the product terms are then connected to OR gates to provide the SOP

Ex: Design a PLA circuit for the following Boolean functions

$F1 = AB' + AC + A'BC'$

$F2 = (AC + BC)'$

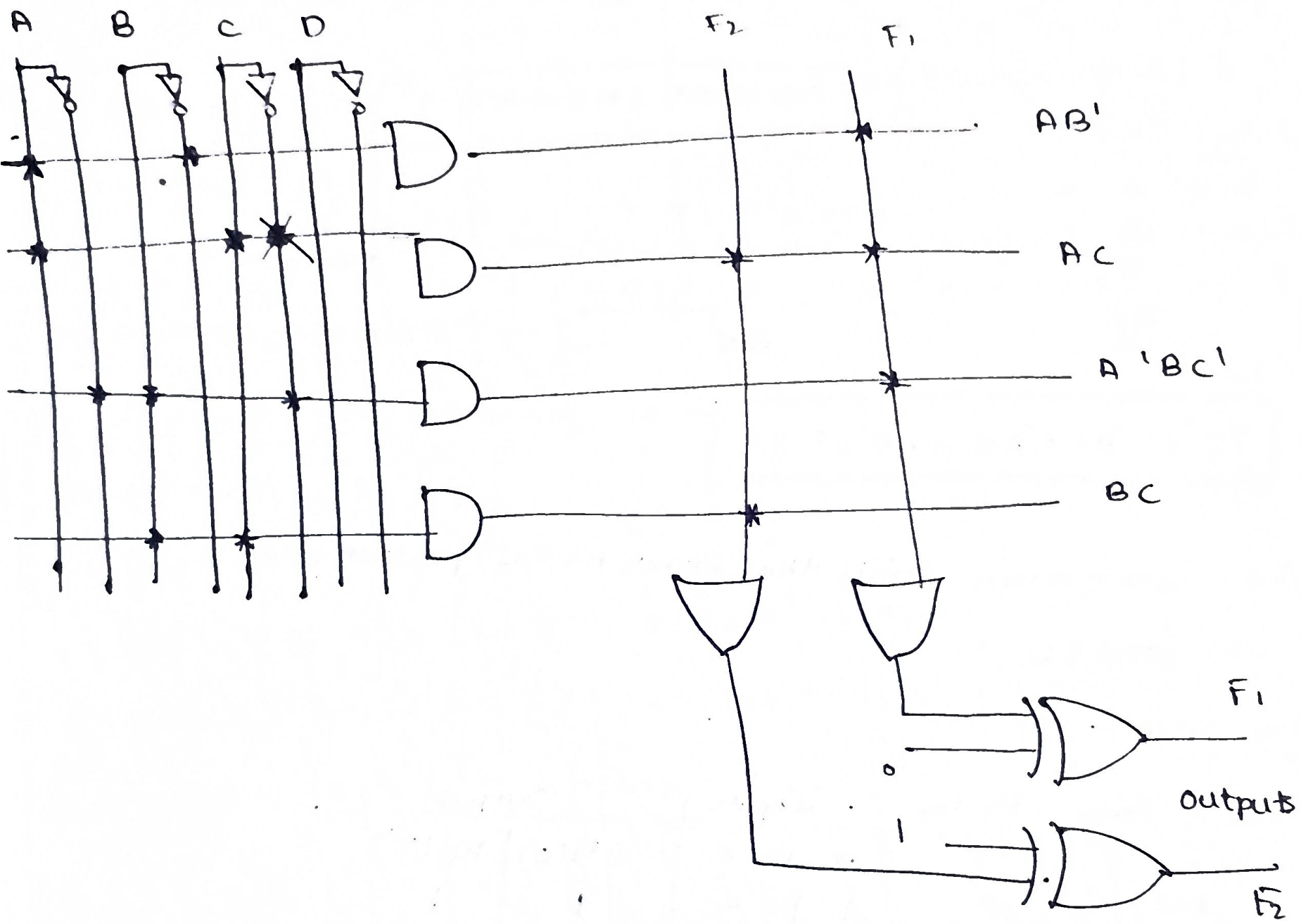
Fuse Map

Product Table	Product No	Inputs			Output	
		A	B	C	F1(T)	F2(C)
AB'	1	1	0	–	1	–
AC	2	1	–	1	1	1
A'BC'	3	0	1	0	1	–
BC	4	–	1	1	–	1

PRA Circuit

No. of AND gates = no. of product terms = 4

No. of OR gates = no. of functions = 2



Ex2 Implement the following 2 Boolean functions with a PRA.

$F_1(A,B,C) = \Sigma(0,1,2,4)$

$F_2(A,B,C) = \Sigma(0,5,6,7)$

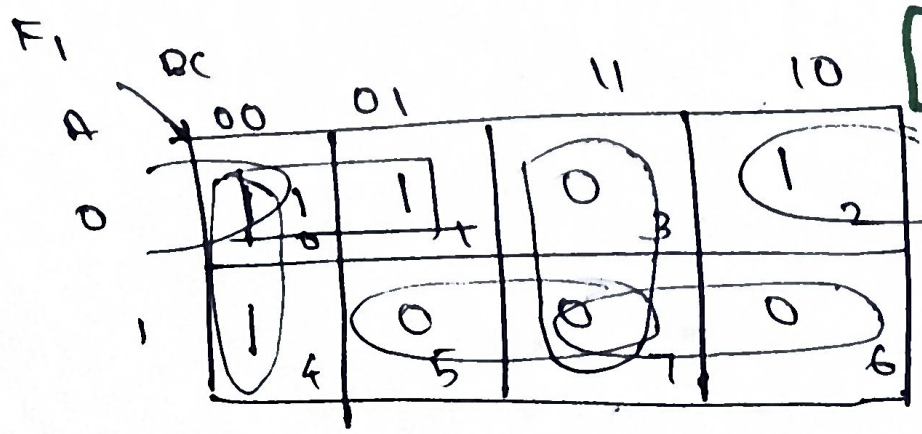
G1 ABC
00 ø
00 x
$\overline{A}\,\overline{B}$

G2 = ABC
ø 0 0
1 0 0
$\overline{B}\,\overline{C}$

G3: A BC
0 0 0
0 1 0
$\overline{A}\,\overline{C}$

G1 = A BC
0 1 1
1 1 1
BC

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



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

G2 = ABC
1 0 1
1 1 1
AC

G3 = ABC
1 1 1
1 1 0
AB

$F_2(A,B,C) = \Sigma(0,5,6,7)$

F2

A \ BC	00	01	11	10
0	1 (0)	0 (1)	0 (3)	0 (2)
1	0 (4)	1 (5)	1 (7)	1 (6)

$G_1 = A\,B\,C = A'B'C'$
0 0 0

$G_2 = A\,B\,C$
1 0 1
1 1 1
$\overline{AC}$

$G_3 = A\,B\,C$
1 1 1
1 1 0
$\overline{AB}$

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

F_2'

$G_1 = A\,B\,C$
0 0 1
0 1 1
$\overline{A}C$

$G_2 = A\,B\,C$
0 1 1
0 1 0
$\overline{A}B$

$G_3 = A\,B\,C$
1 0 0
$AB'C'$

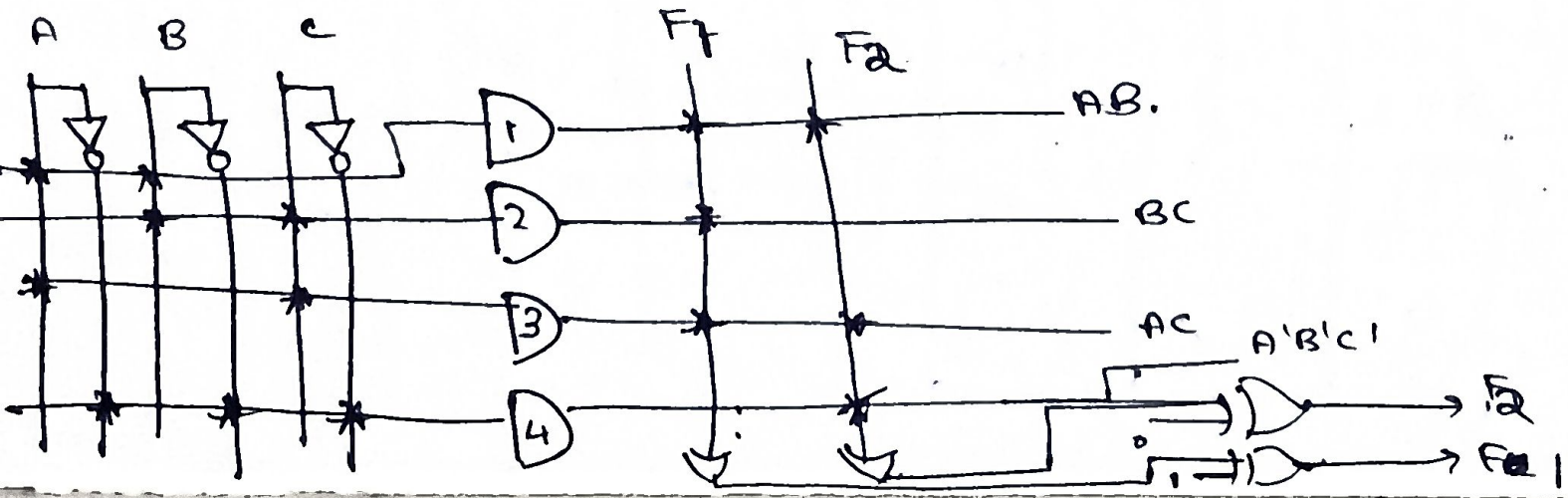
$$F_2' = \bar{A}C + \bar{A}B + AB'C'$$

The combination with the least no. of product terms is F_1' and F_2

Fuse Table

Product Term	Pt No	Inputs A	B	C	Outputs $F_1(C)$	$F_2(T)$
AB	1	1	1	–	1	1
BC	2	–	1	1	1	–
AC	3	1	–	1	1	1
$A'B'C'$	4	0	0	0	–	1

Logic Diagram



③ Programmable Array Logic (PAL)

→ a PLD with a fixed OR gate and a programmable AND array.

→ easier to program, but not as flexible as a PLA.

→ Unlike PLAs, a product term cannot be shared among 2 or more OR gates.

→ This means that each function is simplified by itself w/o regard to common product terms.

Ex1 Simplify the following Boolean Functions and design a PAL for the same.

$w(A,B,C,D) = \Sigma(2,12,13)$

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

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

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

① $w(A,B,C,D)$

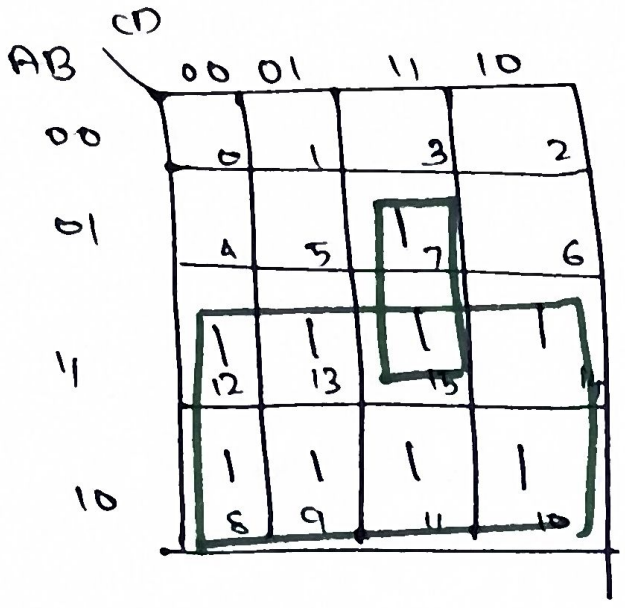
AB \ CD	00	01	11	10
00	0	1	3	1 (2)
01	4	5	7	6
11	1 (12)	1 (13)	15	14
10	8	9	11	10

$G_1 = A\ B\ C\ D$
$0\ 0\ 1\ 0 = A'B'CD'$

$G_2 = A\ B\ C\ D$
$1\ 1\ 0\ 0$
$1\ 1\ 0\ 1$
$= ABC'$

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

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



G_1 =

A	B	C	D
0	1	1	1
1	1	1	1

BCD

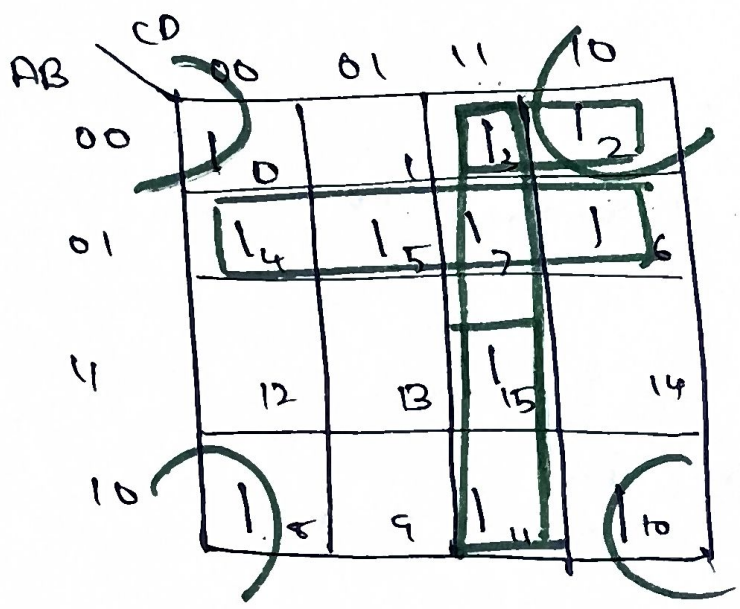
G_2 =

A	B	C	D
1	1	0	0
1	1	1	1
1	1	0	1
1	1	1	0
1	0	0	0
1	0	0	1
1	0	1	1
1	0	1	0

A

$x = A + BCD$

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



G_1 =

A	B	C	D
0	0	0	0
0	0	1	0
1	0	0	0
1	0	1	0

$\bar{B}\bar{D}$

G_3 =

A	B	C	D
0	1	1	1
0	0	1	1
1	1	1	1
1	0	1	1

CD

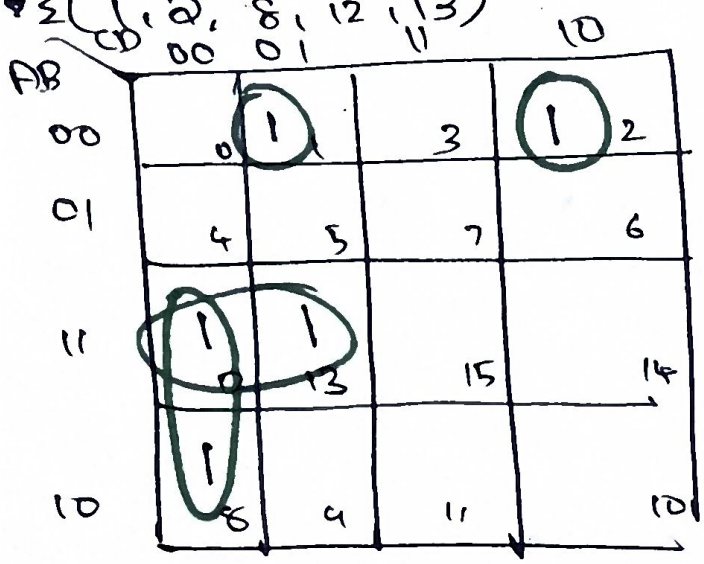
G_2 =

A	B	C	D
0	1	0	0
0	1	0	1
0	1	1	1
0	1	1	0

$\bar{A}B$

$y = A'B + CD + B'D'$

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



$G_1 = A'BCD'$

$G_2 = A'B'C'D$

G_3 :

A	B	C	D
1	1	0	0
1	1	0	1

ABC'

G_4 :

A	B	C	D
1	1	0	0
1	0	0	0

$AC'D'$

$w = A'BCD' + A'B'C'D'$

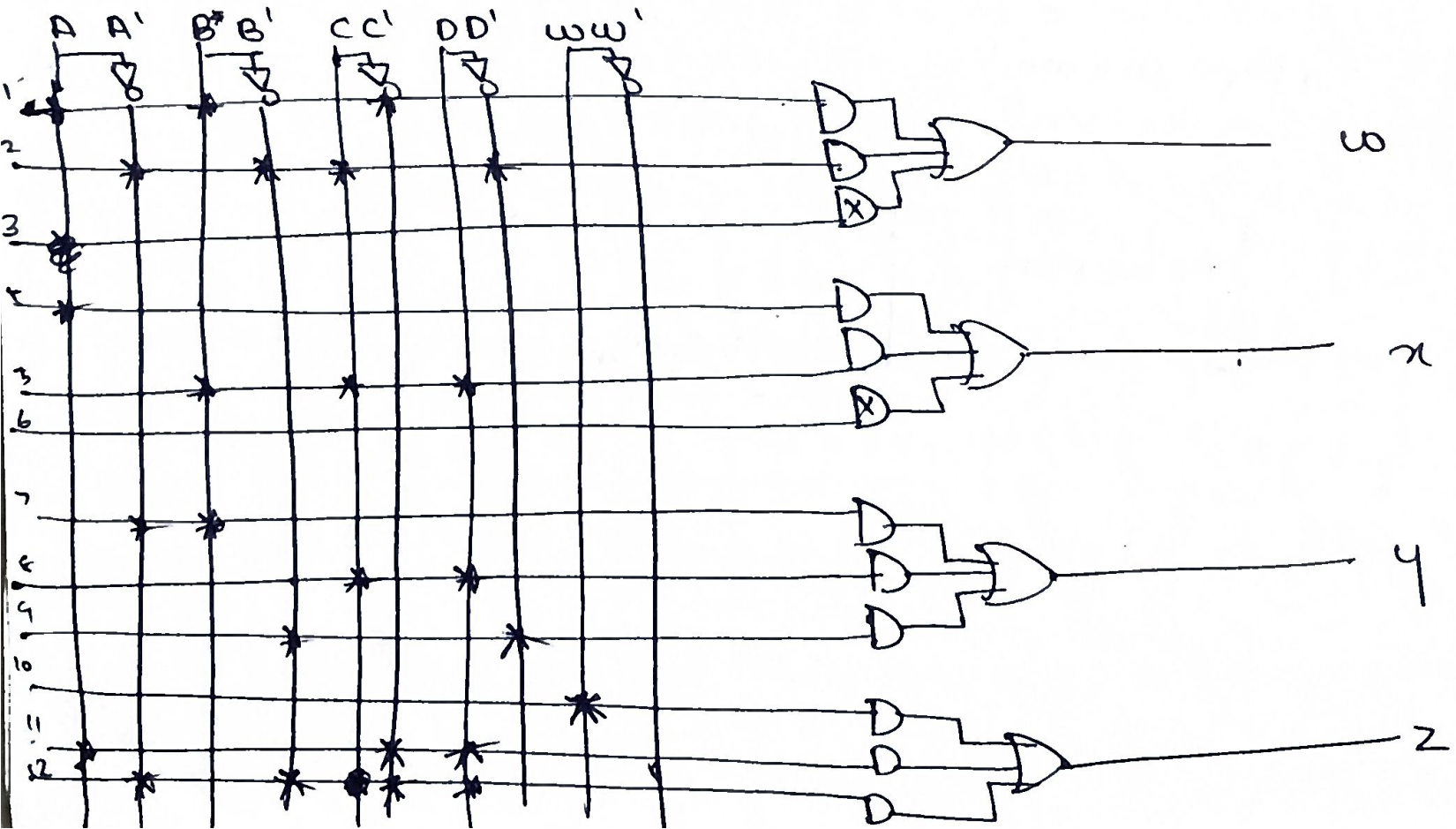
$w = ABC' + AC'D'$

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

∴ No. of AND gates required = 3

Term	Product Term No	Input A	B	C	D	ω	Function Output
1	ABC'	1	1	0	–	–	ω = ABC' + A'B'CD'
2	A'B'CD'	0	0	1	0	–	
3	–	–	–	–	–	–	
4	A	1	–	–	–	–	x = A + BCD
5	BCD	–	1	1	1	–	
6	–	–	–	–	–	–	
7	A'B	0	1	–	–	–	y = A'B + CD + B'D'
8	CD	–	–	1	1	–	
9	B'D'	–	0	–	0	–	
10	ω	–	–	–	–	1	z = ω + AC'D' + A'B'C'D
11	AC'D	1	–	0	D	–	
12	A'B'C'D	0	0	0	1	–	

Circuit



* Sequential Programming Devices

→ Digital systems are designed with flip flops and gates.

→ Combinational PLD consists of only gates

→ It is necessary to include external flip-flops to PLDs.

→ Sequential programming devices include both gates & flip-flops

Major Types

1. Sequential or simple programmable logic device (SPLD)
2. Complex programmable logic device (CPLD)
3. Field programmable gate array (FPGA)

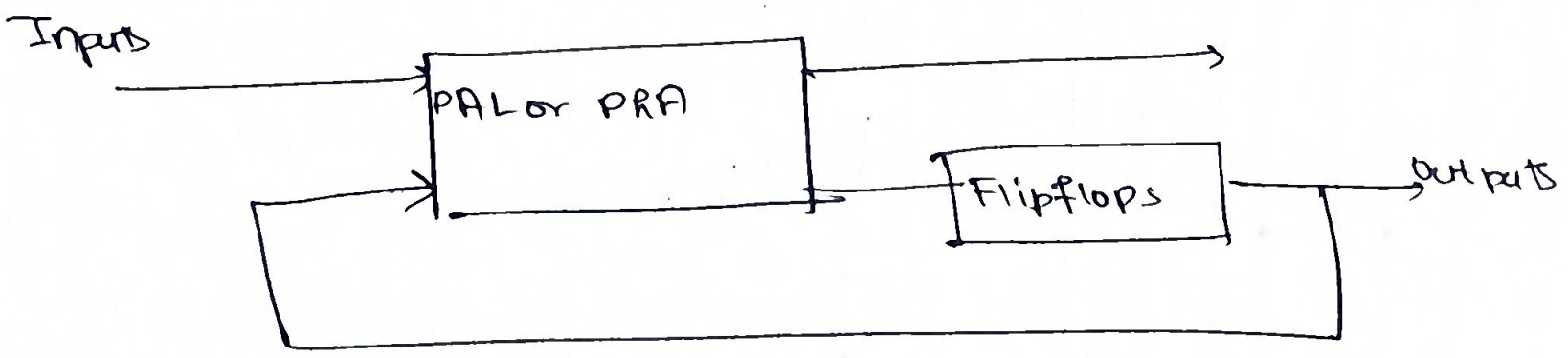
① Sequential / Simple PLD

→ has flip-flops in addition to and-or gates.

→ PAL or PLA combined with flipflops to form a register

→ outputs can be taken from OR gates or flip-flops

→ D or JK flipflops are used.



* FPRS

→ Field programmable Logic sequencer (FPRS)

→ has several outputs to flipflops, ~~the PRA is arqc~~ FPRS is organized around a PAL.

→ flexible, can operate with either the JK or D type.

Disadvantage

→ Too many programmable connections

→ Did not succeed commercially

* Registered PAL

→ a PAL with flip-flops

→ This is to signify that that the device contains flip flops in addition to AND-OR gates.

* Macrocells

→ Each section of an SPRD is called a macrocell.

→ has an SOP combinational Logic function and an optional flip-flop.

Construction → has an AND-OR array as a combinational PAL

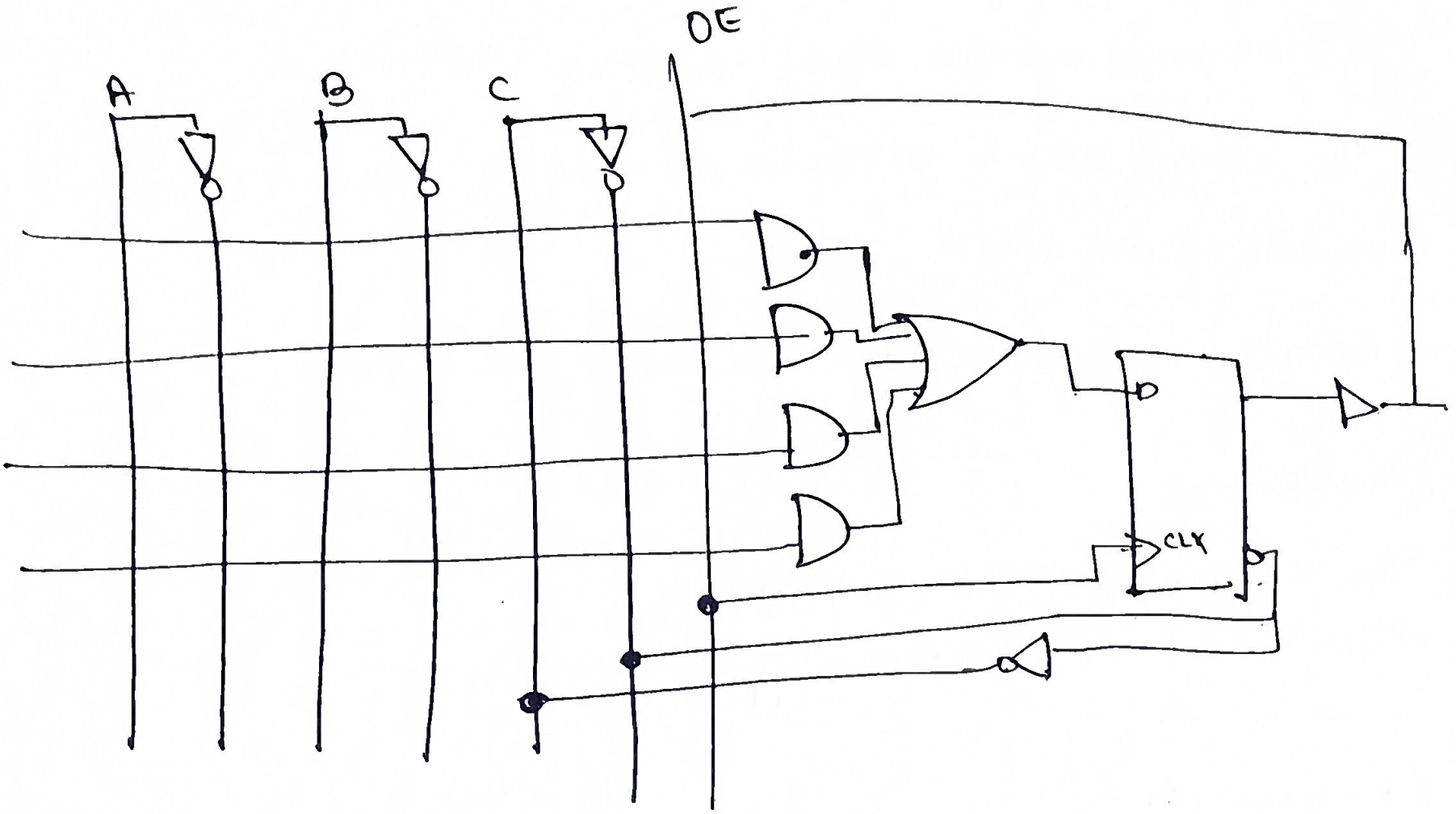
→ output is driven by an edge-triggered D flip-flop

→ Flipflop o/p is connected to a 3-state buffer

→ o/p of flipflop fed back into a programmable AND i/ps

→ Functionalities include options to use or bypass the flipflop

select clock edge polarity etc.



*CPLD

→ Digital system design often requires the connection of several PLD devices to produce the complete specification

→ more economical

→ CPLD = Complex PLD

→ consists of a collection of individual PLDs on a single IC.

Construction

→ multiple PLDs are connected through a programmable switch matrix

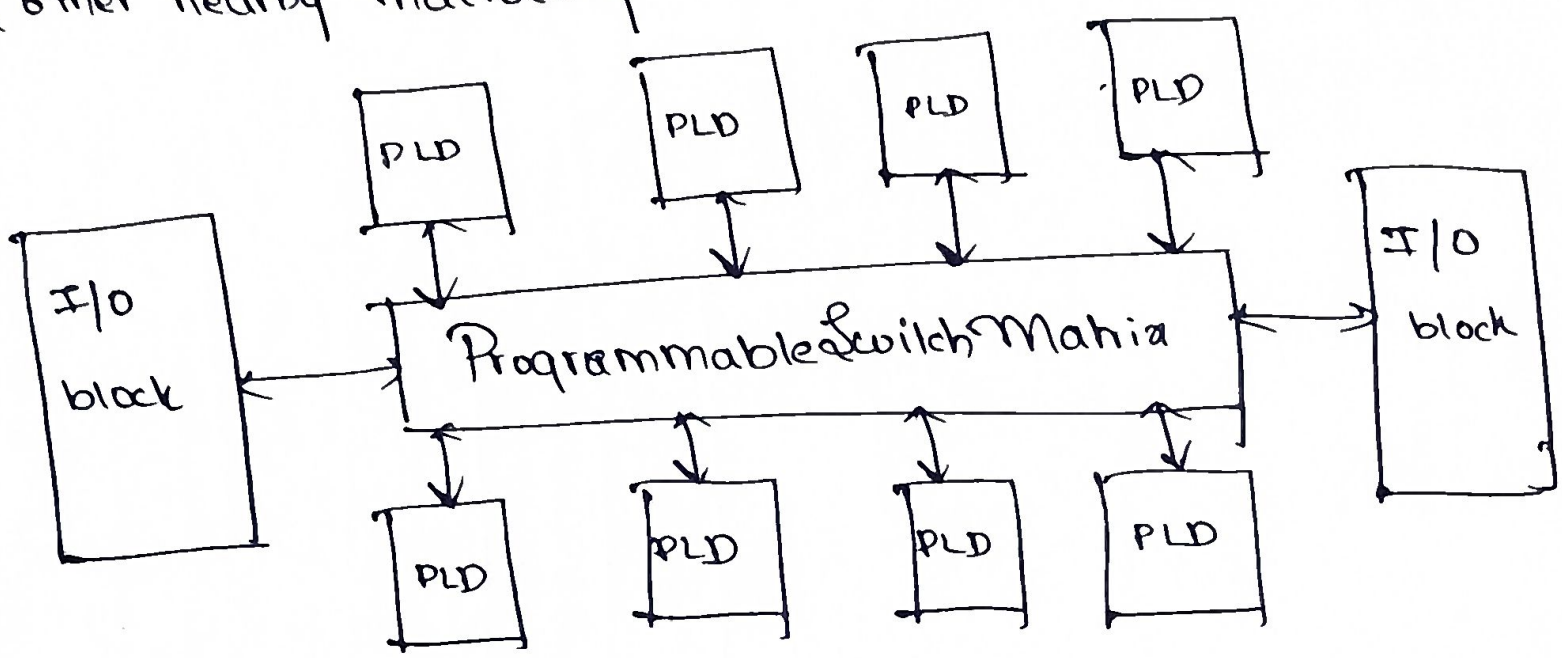
→ The I/O blocks provide connections to the IC pins.

→ The switch matrix receives inputs from the I/O block & directs them to individual macrocells.

Illy selected outputs from the macrocells are sent to the o/ps.

→ Each PLD has 8-16 macrocells

→ If a macrocell has unused product terms, they can be used by other nearby macrocells.



* FGPA - Field Programmable Gate Array

→ FGPA → a VLSI circuit that can be programmed at the user's location

→ has millions of logic blocks surrounded by programmable i/p and OP blocks, connected together via programmable interconnections

→ An FPGA has lookup table, mux, gates & flipflops

→ A lookup table is a truth table stored in SRAM, and provides combinational ckt fns. for the logic block of the FPGA.

→ FPGA can be reprogrammed every time the power is switched on / off.