EE 200: Digital Logic Circuit Design

Dr Radwan E Abdel-Aal, COE

Unit 6

1.Random Access Memory (RAM)

(Sections 7.2- 7.3)

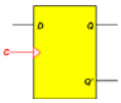
2.Programmable Logic

(Sections 7.5-7.7)

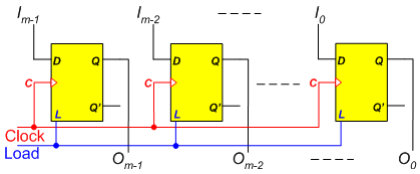
Charles Kime & Thomas Kaminski

1. Random Access Memory (Read/Write)

- **Storage element → Memory device for data storage**
- **A latch or a flip flop stores one bit**



- **A register of m such flip flops stores m bits**



m = 8: Byte, m = 16: Word,
m = 32: double words, m = 64: Quad words, etc.

Over the years, Processors have used larger and larger registers to store/ process data: m = 4, 8, 16, 32, 64, 128 bits

Computer Storage
The computer needs to store programs and data

Primary Storage

Solid State (Semiconductor) Memory (**Random** Access)

- Faster Speed
- More Costly (per bit)
- Smaller Capacity
- Closer to processor

Secondary **(Mass)** Storage

Discs, tapes, CDs (**Sequential** Access)

Now new mass storage devices are Solid State (semiconductor)!

Volatile

Read/Write (RAM)*

Non Volatile (e.g. for booting)

Read Only (ROM)

Static
- SRAM

ROM
PROM
EPROM
EEPROM?

Dynamic
- DRAM:
  - Larger Capacity
  - Cheaper per bit
  - Lower power
  - But needs continuous refreshing to maintain data

* Misnomer – Why?
The Read/Write becoming a misnomer as well!

Random Access (Read/Write) Memory (RAM)

Example: 1024 (1 k) locations × 16-bit each

- A RAM memory Chip effectively contains a number of registers

- Why "**random access**"?

Time taken to transfer data to or from **any** address (storage location) **is the same** regardless of the address

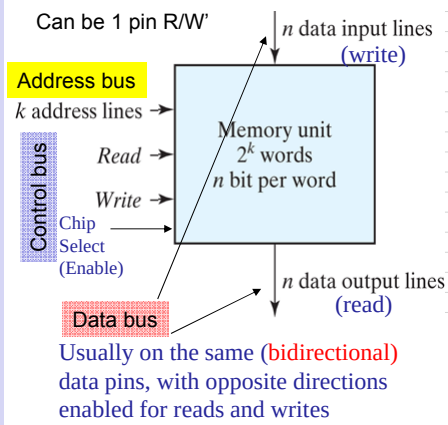
- This is different from **sequential storage**, e.g. Disc, CD, or tape

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

Size or Storage Capacity of a Memory Device

of storage locations x width of each location (bits)

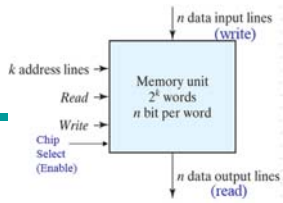
- Address = k bits, Data width = n
  Number of registers (storage locations) = 2^k locations
- Each location is m-bit wide
- Memory size is
  2^k locations x n bits each
- Total storage Capacity in bits:
  Number of storage locations x width of data in each location
- Example: k = 10, n = 8
  → Size: 2^{10} locations x 8 bits each
  → 1K Bytes of storage
  = 1 K Bytes = 8 K bits



What is k, n for a 64 K x Byte memory?

RAM Memory

Read and Write Operations



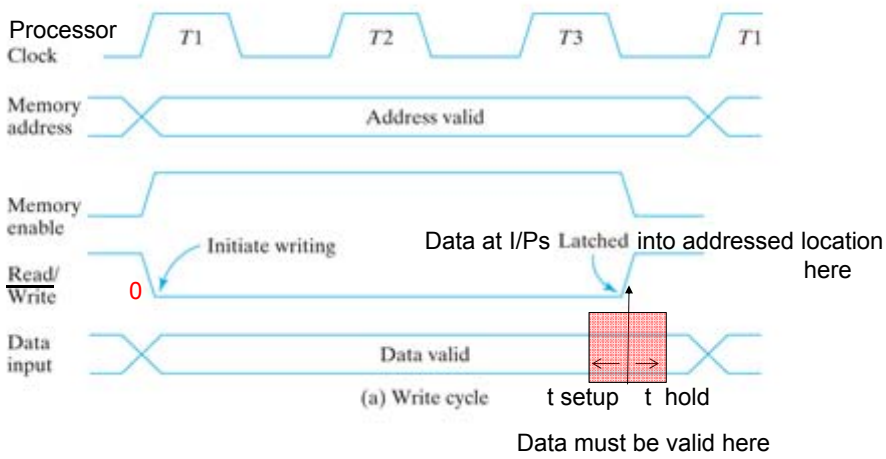
- Write:
  - Enable the memory device (Chip Select)
  - Apply the address of the location you want to write to
  - Apply the data to be written to data I/P pins
  - Activate the WRITE Control input → Data on the input pins are written into addressed location

Chip Select

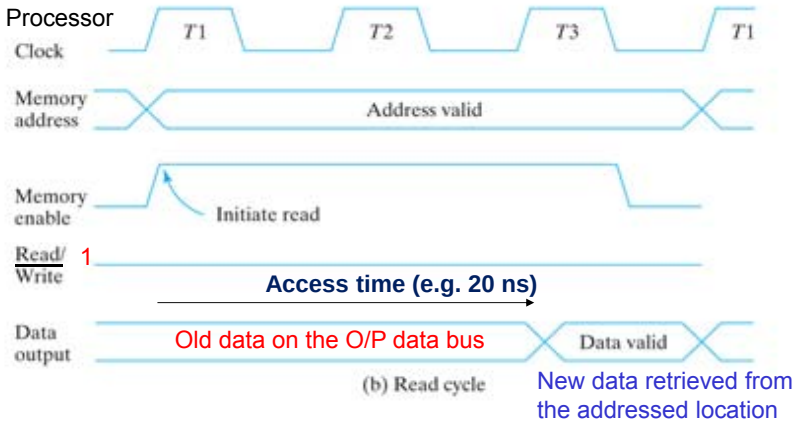
Memory Enable	Read/$\overline{\text{Write}}$	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

- READ:
  - Enable the memory device (Chip Select)
  - Apply the address of the location you want to write to
  - Activate the READ Control input
    → Data stored at addressed location appears at the Data O/P pins (are read) after some propagation delay (access time)

Memory
Write and Read Cycles



Memory
Write and Read Cycles

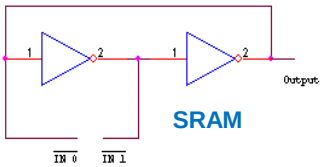


Types of RAM (Read/Write) Memories

Two Basic Types:

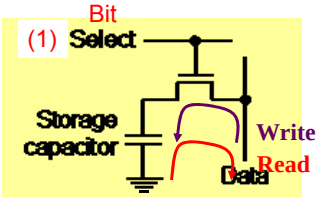
- **Static RAM (SRAM): Data stored in latch (1 latch per bit) and remains as long as power remains ON**

1-bit Cell ≈ 4 Transistors



SRAM

- **Dynamic (DRAM): Data stored as a charge on a capacitor – can be lost due to leakage, needs to be periodically refreshed, otherwise data will be lost even with power ON**



Only 1 Transistor Per 1-bit cell

DRAM

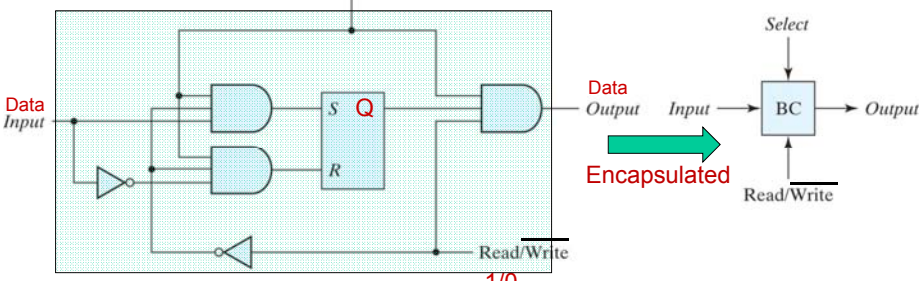
Comparison: Speed, Cost, Density, Applications

Bare Circuits (Storage Only)

Modeling a Complete bit circuit (BC) with controls

=1 to select bit *Select*

Here, a cell that is not selected Produces a _____(0/1) O/P?



Encapsulated

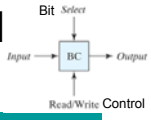
1/0

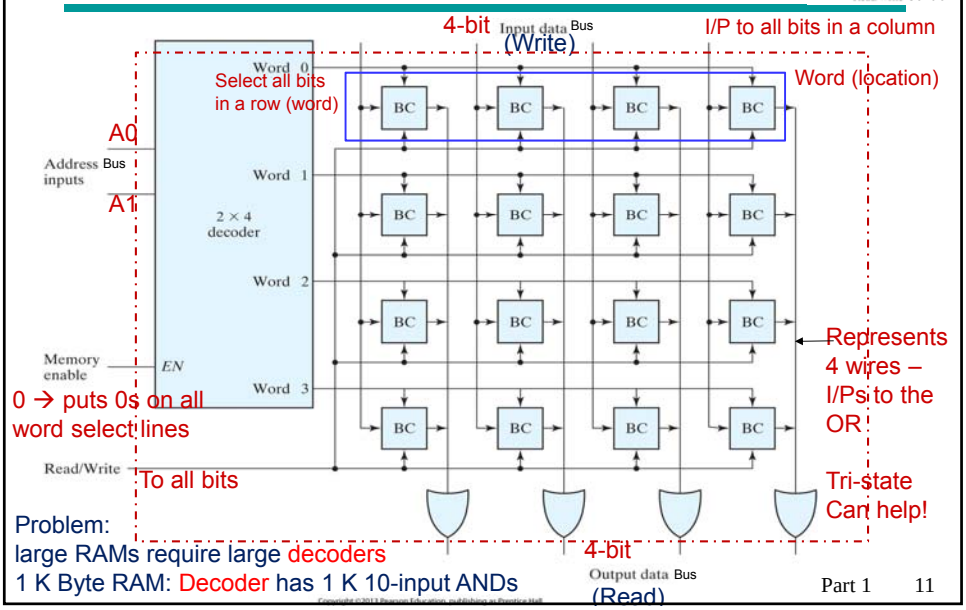
(a) Logic diagram (b) Block diagram

SR=00: No Change (Read)

Memory Enable	Read/$\overline{\text{Write}}$	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

4 words (locations) x 4 bits (16-bit) RAM With Direct Decoding (1 Enable for each "word")



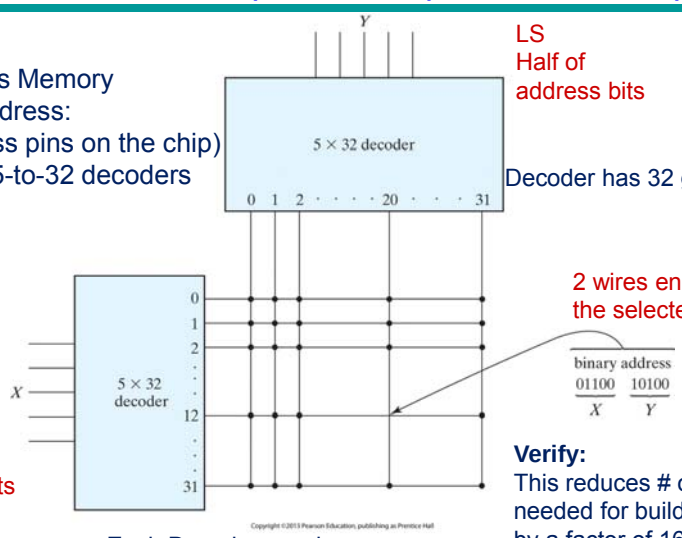


Problem:
large RAMs require large decoders
1 K Byte RAM: Decoder has 1 K 10-input ANDs

Solution: 2-D Decoding (2 Enables for each word)

Uses 2 smaller (1/2 size) decoders (X, Y)

1 K Words Memory
(10-bit address:
10 address pins on the chip)
Use two 5-to-32 decoders



Each Decoder now has
32 gates (of 5 I/Ps each)

Verify:
This reduces # of AND gates needed for building decoders by a factor of 16

Further Improvements for DRAM … Address Multiplexing (reduced the # of address pins

DRAMs are large in capacity

Their large address needs a
Large number of pins on the chip

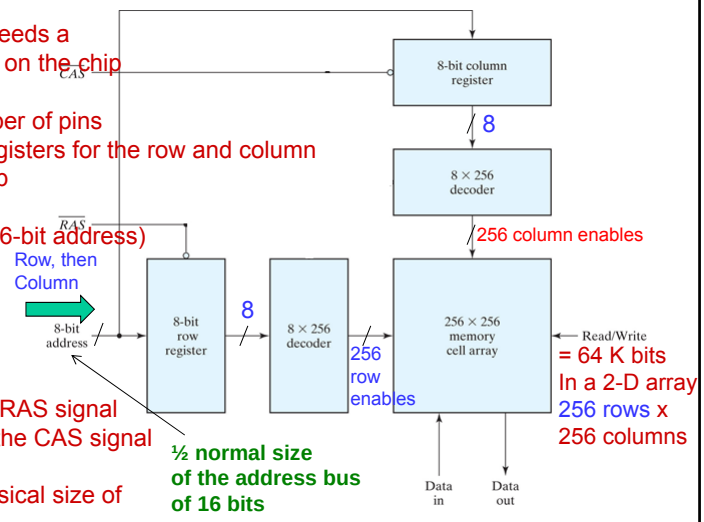
We can half the number of pins
If we use separate registers for the row and column
Addresses on the chip

64K = 2^{16} locations (16-bit address)

The 16 bit address is
Supplied to the chip
on only 8-pins:

1. Row part with the RAS signal
2. Column part with the CAS signal

Helps reduce the physical size of
DRAM chips



2. Programmable Logic Implementation Technologies Overview

- **Why programmable logic?**
- **Programmable logic: Technologies and Configurations**
- **Examples of Programmable Logic Devices:**
  - **Read-Only Memory (ROM)**
  - **Programmable Array Logic (PAL)**
  - **Programmable Logic Array (PLA)**
  - **VLSI Programmable Logic Devices (Field Programmable Gate Arrays- FPGA)**

The Rationale: Why Programmable Logic?

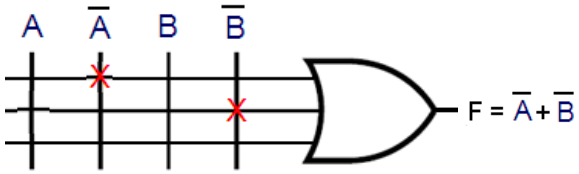
- Facts: (Economy of Scale)
  - It is most economical to produce ICs in large volumes
  - But:
  - → In many situations users require:
    - → ICs in smaller volumes
    - → Frequent changes to be done in the field, e.g. on the Firmware of a product under development

Integrated Circuits

- A **programmable logic device** can be **a good compromise**:
  - **Produced** in large quantities
  - But also allows users to **program it** many times* (in the field) to accommodate changes on small volumes

(*nowadays: erasable, reprogrammable, etc.)

Programmable Logic Concept



Locations of connections inserted/removed by user determine the logic function implemented

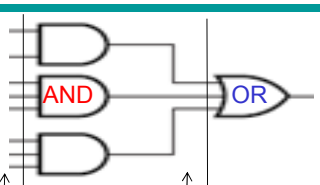
Hardware Programming Technologies
How connections are made: The ROM case

- **ROM: In the Factory (not user-programmable)**
  Cannot be erased or reprogrammed by user
  - **"Programmed" permanently through the VLSI mask during manufacturing of the chip**

- **PROM: Programmable only once,**
  - **e.g. using fusible links (metal connections)**

- **Programmable many times (Erase & then Re-program)**
  - **Ultra-Violet (UV) Erasable, e.g. EPROMs (off situ)**
  - **Electrically Erasable, (in situ) e.g.**
    - **EEPROMs**
    - **Flash Memory**

Programmable Logic Configurations: (SOm or SOP)
All use AND-OR structure- differ in which is programmable

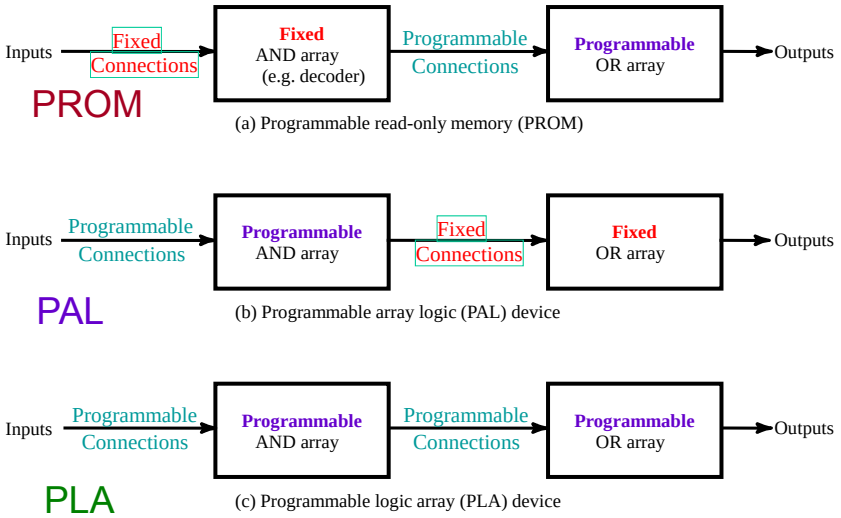


How many Possibilities?

Connections to the AND	Connections to the OR	
Fixed	Programmable	→ Read Only (non-volatile) Memory (PROM)
Programmable	Fixed	→ Programmable Array Logic (PAL)
Programmable	Programmable	→ Programmable Logic Array (PLA) (Maximum flexibility)

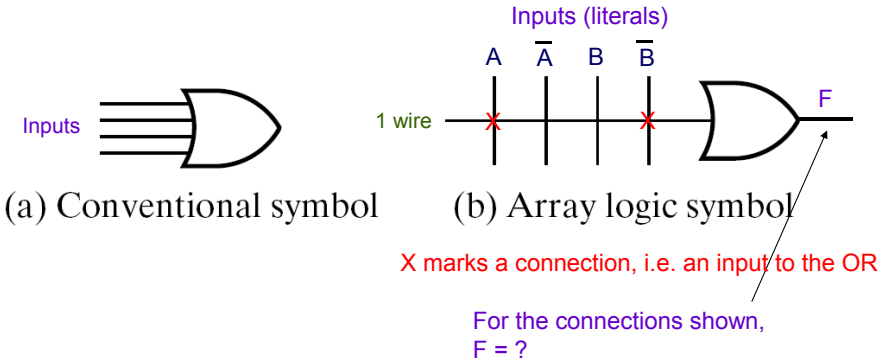
18

ROM, PAL and PLA Configurations



Wiring Conventions for Programmable Logic

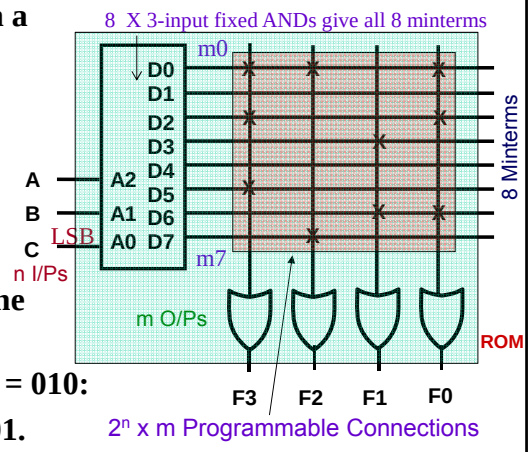
- We deal with a large number of gates and gate inputs
- Need a more **concise** way of expressing gate circuits graphically



a. Read Only Memory (ROM): &: Fixed- OR: Prog

Programmable sum of fixed minterms

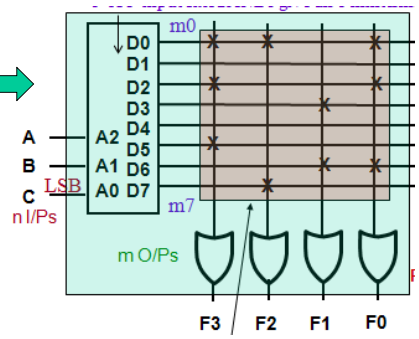
- Example: 8 X 4 PROM (n = 3 i/p (address) lines, m = 4 o/p lines)
- The fixed "AND" array is in a 3-to-8 "decoder" giving all 8 minterms
- Programmable "OR"s. We use a single line to represent all inputs to an OR gate. An "X" in the array indicates connecting the minterm to the OR
- Example: For input (A,B,C) = 010: output is $(F_3,F_2,F_1,F_0) = 1001$.
- Exercise: Express the F_1 as - $\Sigma m()$ - an algebraic expression in A,B,C



3-input 4-output CL cct **seen differently**

This ROM implements this truth table

A	B	C	F3	F2	F1	F0
0	0	0	1	1	0	1
0	0	1	0			
0	1	0	1			
0	1	1	0			
1	0	0	0			
1	0	1	1			
1	1	0	0			
1	1	1	0			



Can look at it in several ways:

- As an 8 words x 4-bit memory: at address 000 we permanently stored data 1101 (non volatile)
- As a look-up table: Enter I/P 000 you get corresponding O/P 1101
- As an implementation of 3-I/P 4 O/P Combinational Logic (Using a decoder and 4 OR gate- Unit 3)

Read Only Memory (ROM): n i/ps to m o/ps

2^n permanent storage locations x m bits each

- A Read Only Memory (ROM) has:
  - n input (address) lines → 2^n storage locations, 2^n minterms
  - m output lines (word width for each storage location)
- A Fixed array of 2^n AND gates implements all the n-input minterms
  Obtained from an **n-to-2^n decoder**
- Programmable Array of m OR gates to form m Som outputs.
- The program for a PROM is simply the **multiple-output truth table** to be implemented
  - For a 1 at an output in the table, a connection is made from the corresponding minterm to the corresponding OR gate
  - For a 0 entry, no connection is made
- Can be viewed as a *memory* with the m inputs as *addresses* of *data* (output values), hence ROM or PROM names!
  Device on previous slide is an 8 x 4 memory (8 locations x 4-bit wide)
- Truth table is a listing of the memory contents at each input address

Read Only Memory (ROM) Advantages/Limitations

Q. For 1 M (Mega) Byte PROM Device
(1M locations x 8 bits)

# of Input (address) lines n = ?

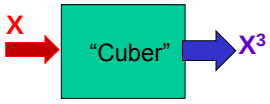
# of Output lines m = ?

- Advantages of ROMs: (= advantages of the canonical form)
  - Can implement any function (since all minterms of the input variables are available!)
  - PROM Program is derived **directly** from the truth table specifying the information to be stored
- Disadvantages:
  (disadvantages of the canonical form- No SOP optimization)
  As n increases → Large Decoders needed (2^n AND gates, each having n inputs)
  Example 1 G byte memory? Sizes

ROM-based Designs (Canonical Form), Som for Combinational & Sequential circuits

- **For Combinational Circuits:**
  ROMs can be used to implement a combinational circuit given its truth tables

- **Example: Look up table**
  example: X → X^3 ; X is a 4-bit unsigned integer



  You should be able to determine the minimum ROM size (# of locations x "word" size) required for a given problem

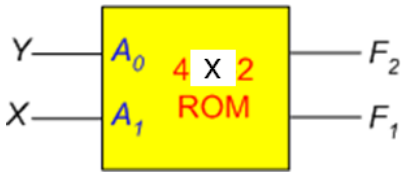
- **For Sequential Circuits:**
  Use ROMs to implement the combinational part of the sequential circuit

ROM-based Designs: Combinational Circuits: Individual O/Ps

Example 1: Implement the following two combinational functions using a ROM

F1 (X,Y) = ∑ m (1,2,3)

F2 (X,Y) = ∑ m (0,2)



Solution should:

- Specify the smallest ROM required:
  ROM has n = 2 inputs (→ 2^2 = 4 locations)
  and m = 2 outputs (→ Each location has 2 bits) …i.e. a 4 x 2 bit ROM
- Specifying the ROM data content (to be programmed into the ROM):
  Directly from the truth table of the two functions

Show the connection diagram for this ROM (Decoder + ORs)

Index	ROM Address A_1	ROM Address A_0	Stored Information F_1	Stored Information F_2
0	0	0	0	1
1	0	1	1	0
2	1	0	1	1
3	1	1	1	0

Give Logic Diagram

ROM Programming Table

ROM-based Design Examples:
Combinational Circuits: Look-up Tables

Example 2: X^2 look-up table, X is 3-bit binary number

→ Specification: Use a ROM to implement a combinational circuit that accepts a 3-bit unsigned binary number at the input and generates its **squared value** at the output.

→ Formulation:

8 x 6 bits ROM, Truth Table →

→ Observations on the truth table:

1. Output B_0 = Input A_0
2. Output B_1 = Always 0

X X^2

8 Locations

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

6 bits

No need to 'store' data for B0 and B1

This reduces the size of the ROM required from 8 x 6 bits to 8 x 4 bits

ROM-based Designs:
Combinational Circuits : Look-up Tables

Example 2, Continued

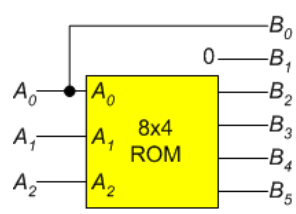
Truth Table for Reduced ROM

8 Locations

ROM Address			Stored Information			
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

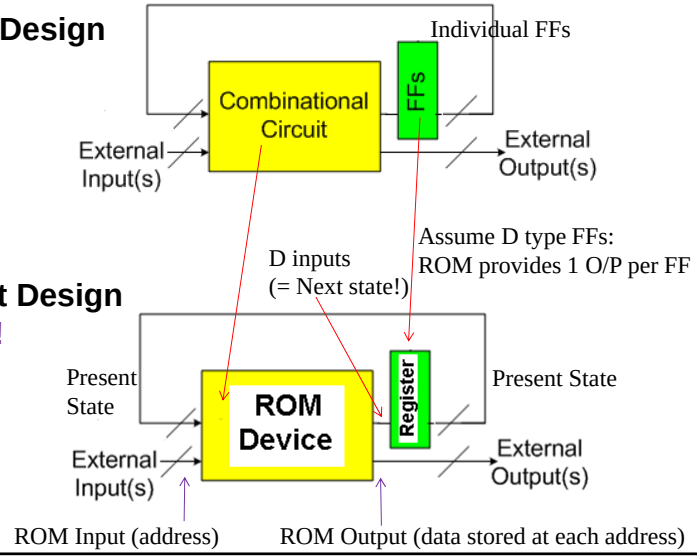
4 bits

Implementations of the X^2 Look-up Table:



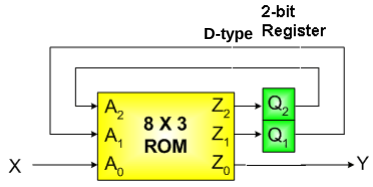
ROM-based Designs:
Sequential Circuits

Conventional Design



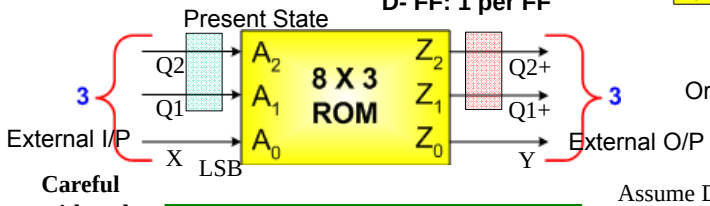
ROM-Register
Sequential Cct Design
Very compact!

ROM-based Designs:
Sequential Circuit- D FF



The ROM Required

Inputs to FFs
D- FF: 1 per FF



Organization

Careful with order of inputs

Assume D type FFs:
ROM provides 1 O/P per FF

ROM Address			Stored Information		
A_2	A_1	A_0	Z_2	Z_1	Z_0
0	0	0	0	0	0
0	0	1	0	1	0
1	0	0	0	1	0
1	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1
Present State		I/P	Next State		O/P

ROM Truth Table

Derive the state diagram
For this sequential circuit

Programmable Array Logic (PAL)

- PAL is the opposite of the ROM, having a set of programmable ANDs combined with a set of fixed ORs-
  (Programmable = has selectable I/Ps)
- PAL has some outputs from OR terms that can be fed back (as internal inputs) to all other AND terms, allowing implementation as multi-level (>2) logic circuits
- Some PALs have outputs that can be **complemented**, allowing F implementation as a POS:

$F = \overline{F}$ SOP

- Advantages over ROM
  - Allows optimized implementation as Sum of a few Products (usually not all minterms would be available)
  - For a given total # of gates, a PAL can support larger n and m than a ROM

→Limitation: 1. Functions with many products: may not be possible
2. Sharing of products between sections is not possible!
→ multiple generation!

PAL Example

sum of > 3 products

$$F1 = A\overline{B}\,\overline{C} + \overline{A}\,B\overline{C} + \boxed{\overline{A}\,\overline{B}\,C + ABC} = W$$

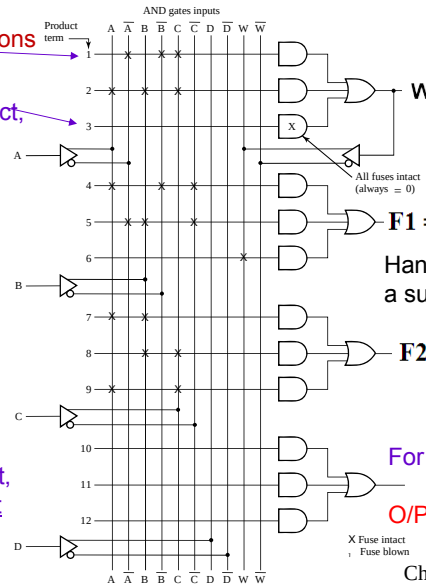
$$F2 = AB + BC + AC$$

Remove all unused connections

For this totally unused product, Leave all connections intact Why?

Opposite of ROM:
AND I/Ps: programmable
OR I/Ps: Fixed

For a totally unused product, Leave all connections intact



$W = \overline{A}\,\overline{B}\,C + ABC$

F1: Deceptively 2-level! but its is not!

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

Handling a sum of > 3 products

$F2 = AB + BC + AC$

For Unused Section:

O/P = 0 or 1?

PAL Programming Table

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

  $F2 = AB + BC + AC$
- F1 was **factored** - has four terms (> 3)
- Factor out last two terms as W

PAL Programming Table

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

Keep Complement. & remove other

Keep True & remove other.

Remove both connections

Full gate not used. Keep both

Section

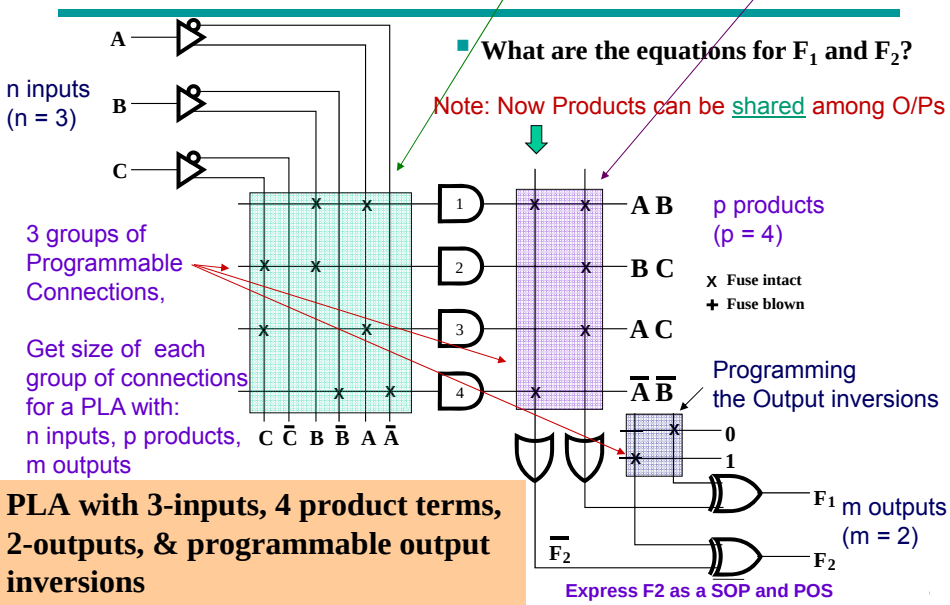
Unused Section
Keep All Connections → 0 O/P

PAL comes with all Connections made. Connections that are not needed **should be removed**

How many connections are removed for product 1?, for product 3?

c. Programmable Logic Array (PLA)

Programming at both the product and the sum levels



PLA with 3-inputs, 4 product terms, 2-outputs, & programmable output inversions

Programmable Logic Array (PLA)

- **Compared to ROMs and PALs, PLA is the most flexible & economical programmable device: having programmable ANDs, programmable ORs, and even programmable output inversions!**
- **Advantages**
  - **More concise implementations than ROM – Uses K-map optimized products**
  - **Allows larger 2-level sums than with PALs: All product terms are available for sharing by all summing Ors**
  - **You do not have to generate a product more than once as in PAL**
  - **PLAs have outputs that can be complemented, so a SOP can be that of either F or F', whichever proves more optimal - globally**
- **But still**
  - **The limited # of product terms can limit the application of a PLA. Solution: → Use global multiple-output optimization to reduce the number of product terms required to fit it into the PLA.**

PLA Global Optimization Example

F1(A,B,C), F2(A,B,C), PLA: (3 inputs, 4 products, 2 outputs with programmable o/p inversion)

- **K-map specifications**
- **How can this be implemented with only four products?**
- **Complete the programming table**
- **Choose implementations (F or $\overline{F}$) that has the largest # of shared products!**
- **Is this PLA enough if we choose to implement F1 & F2 directly?**

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	0	0

F1 map

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

F2 map

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

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

$F_2 = AB + AC + BC$

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

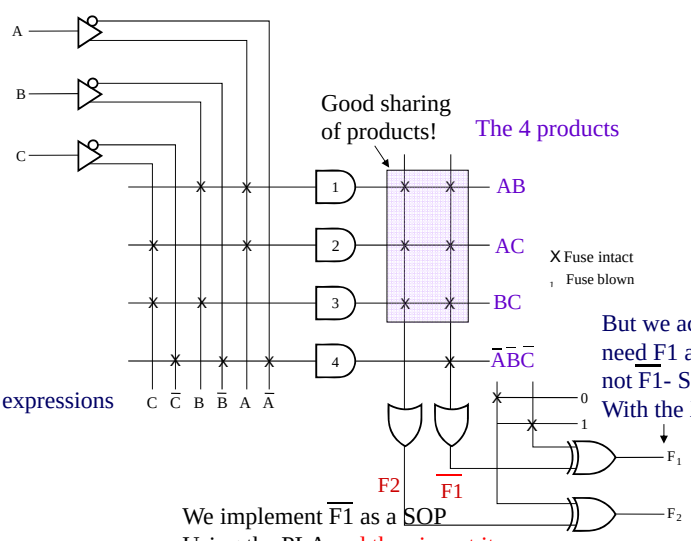
PLA programming table

	Product term	Inputs A	B	C	Outputs (C) 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	–

SUM (OR) Programming (Vertically)

Product (AND) Programming (horizontally)

Programmable Logic Array (PLA) Example, Contd.



Give algebraic expressions of F1 and F2

But we actually need F1 as an O/P, not $\overline{F1}$- So invert $\overline{F1}$ With the XOR

We implement $\overline{F1}$ as a SOP Using the PLA and then invert it , as this is more economical