Chapter 7
Memory and Programmable Logic

Outline

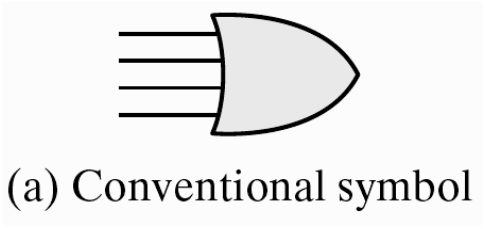
- Random-Access Memory
- Memory Decoding
- Error Detecting and Correction
- Read-Only Memory
- Programmable Logic Array
- Programmable Array Logic
- Sequential Programmable Devices

Introduction

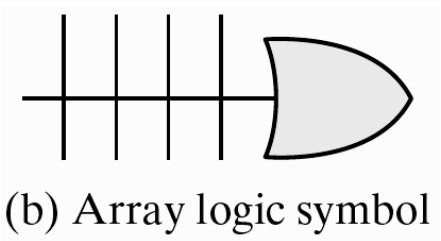
- A memory unit
  - is a storage to provide and to receive binary information.
  - is a collection of cells stores binary information
- RAM – Random-Access Memory
  - To accept new information for storage to be available later for use
  - Memory *read* operation
  - Memory *write* operation
- ROM – Read-Only Memory
  - The information inside can not be altered by writing.
  - A *programmable logic device* specified by some hardware procedure.

Introduction

- Programmable Logic Device (PLD)
  - ROM– data is written in when memory is fabricated.
  - PLA– programmable logic array: data is written in through burning machine.
  - PAL– programmable array logic: data is written in through burning machine.
  - FPGA– field-programmable gate array: data is written in through download port from external system.
    - programmable logic blocks
    - programmable interconnects



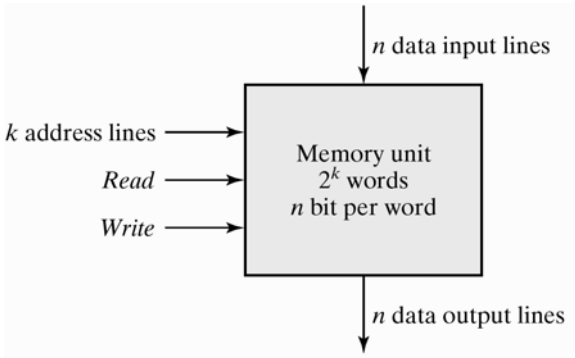
(a) Conventional symbol



(b) Array logic symbol

Random-Access Memory

- Characteristics
  - The time it takes to transfer data to or from any desired location is always the same.
  - The communication between a memory and its environment is achieved through
    - Data input lines,
    - Data output lines,
    - Address selection lines, and
    - Control lines (read and write).
- A memory unit (width)
  - stores binary information in groups of bits (words)
  - 8 bits (a byte), 2 bytes, 4 bytes



Memory Content Array

- A memory with k-bit address has 2^k data. (depth)
- (depth) x (width) memory
  - Ex. An 1k*16 Memory

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

Write and Read Operations

- Write operation
  - Apply the binary address to the address lines
  - Apply the data bits to the data input lines
  - Activate the *write* input
- Read operation
  - Apply the binary address to the address lines
  - Activate the *read* input

Control Inputs to Memory Chip

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

Memory Description in HDL

- Declaration: **reg** [width-:0] memword [0:depth-1]

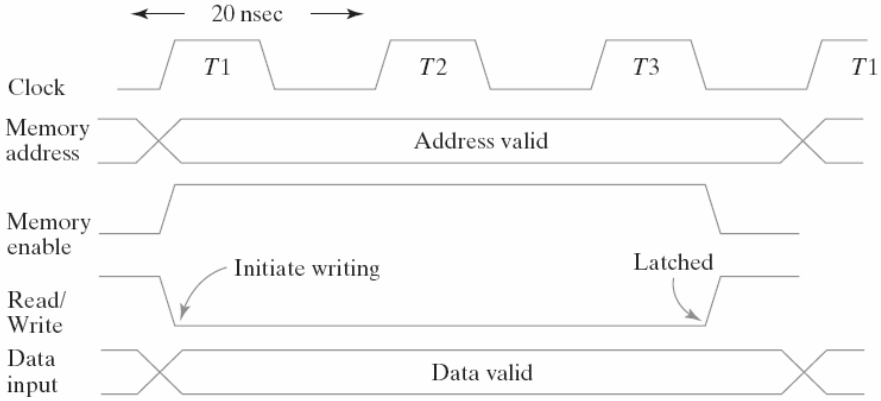
```
// Read and write operations of memory
// Memory size is 64 words of 4 bits each
module memory(Enable, ReadWrite, Address, DataIn, DataOut);
  input   Enable, ReadWrite;
  input   [3:0] DataIn;
  input   [5:0] Address;
  output [3:0] DataOut;
  reg      [3:0] DataOut;
  reg      [3:0] Mem [0:63];
  always @ (Enable or ReadWrite)
    if (Enable)
      if (ReadWrite) DataOut = Mem[Address];   // Read
      else Mem[Address] = DataIn;              // Write
    else DataOut = 4'bz;                       // High impedance state
endmodule
```

Timing Waveforms

- The operation of the memory unit is controlled by an external device such as CPU
- The *access time* is the time required to select a word and read it
- The *cycle time* is the time required to complete a write operation
- Read and write operations must be controlled by CPU and be synchronized with an external clock.

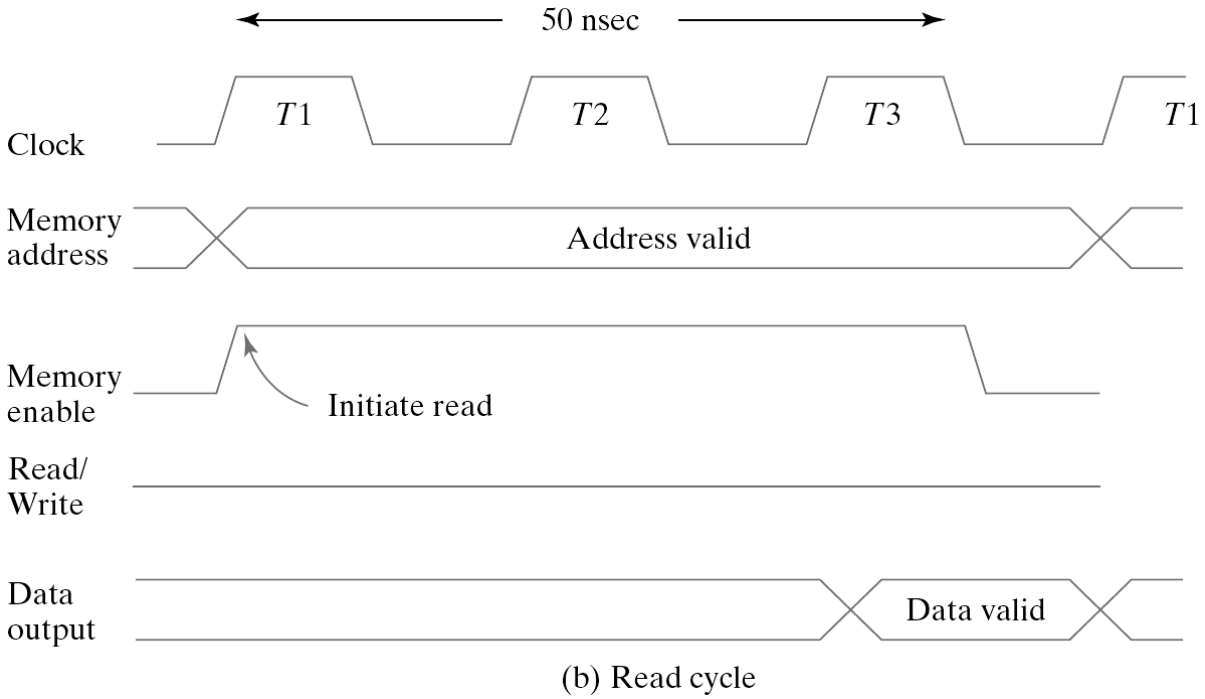
Memory Cycle Timing Waveforms

- CPU clock – 50 MHz (20ns period)
- A write cycle
  - T1: Providing the address and input data to the memory, and the read/write control signal
  - Address bus and read/write control must stay active for 50ns
  - The address and data signal must remain stable for a short time after control signal is deactivated.



Memory Cycle Timing Waveforms

- A Read Cycle



(b) Read cycle

Types of Memories

- Sequential access memory
  - The information stored in the medium is available only at certain interval of time.
  - The access time depends on the location of the accessed word. The access time is variable.
  - A magnetic disk or tape is this type.
- Random access memory
  - The access time is always the same regardless of the particular location of the word.
  - Integrated circuit memory is this type

Static and Dynamic Memory

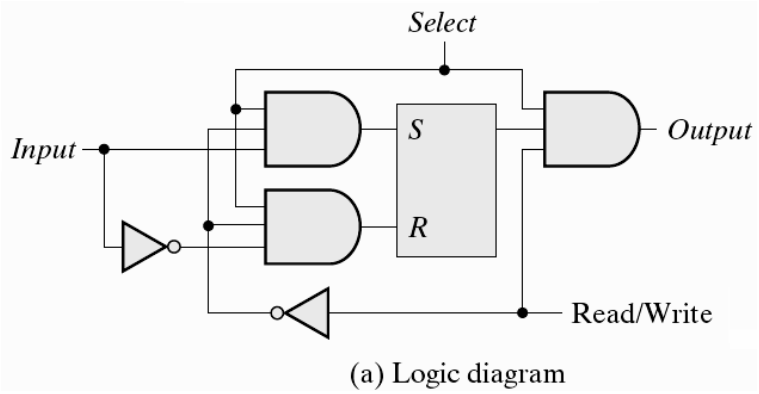
- Static memory
  - Information are stored in latches.
  - Data remains valid as long as power is applied.
  - Read/write cycle is shorter.
  - Control is easier
- Dynamic memory
  - Information are stored in the form of charges on capacitors.
  - Refreshing: The stored charge tends to discharge with time and need be refreshed (read and write back) due to leakage current.
  - Power consumption is reduced.
  - Memory capacity is larger.

Volatile and Non-Volatile Memory

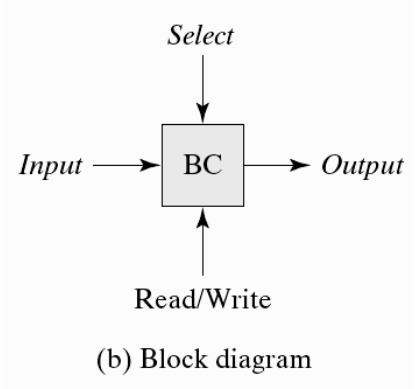
- *Volatile* Memory
  - Stored information is lost when power is turned off.
  - SRAM, DRAM
- *Non-volatile* Memory
  - Stored information is retained after the removal of power.
    - Magnetic disk or tape
    - ROM
    - EPROM, EEPROM
    - Flash memory
  - Ex. The startup of the computer must use the program in the *non-volatile* memory (ROM).

Memory Decoding

- A memory unit has two parts
  - the storage components
  - the decoding circuits to select the memory word
- A basic memory cell model (Virtual, not practical)
  - Actually, the cell is an electronic circuit with four to six transistors.



(a) Logic diagram

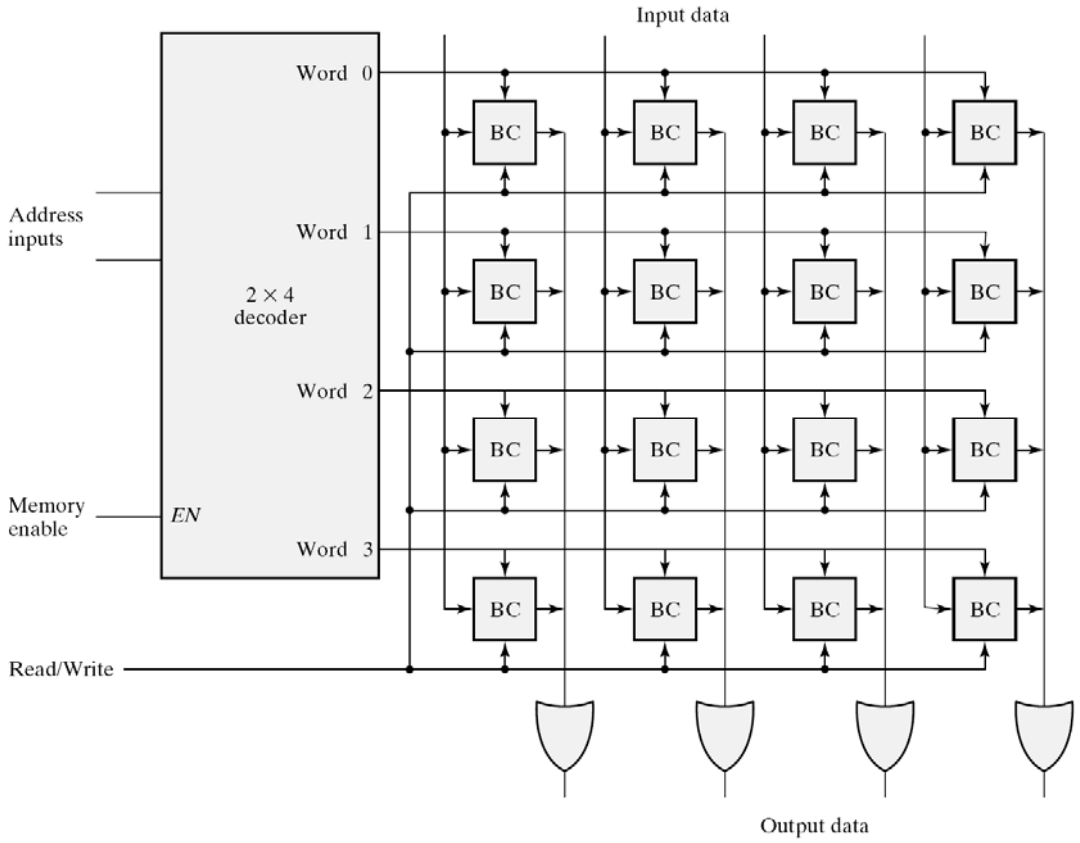


(b) Block diagram

Internal Construction

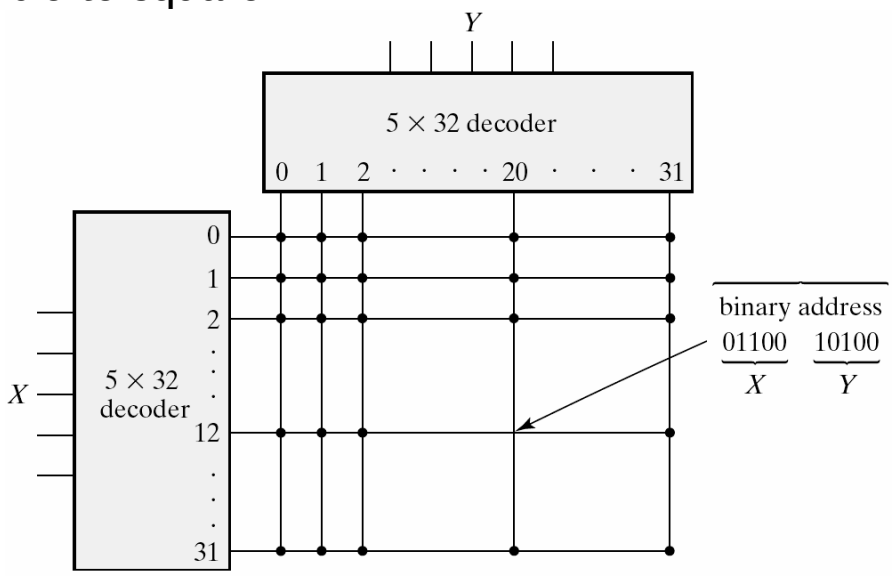
- A RAM of *m (depth)* words and *n (width)* bits per word
  - *m* n* binary storage cells
  - Decoding circuits to select individual words
    - k-to-2k decoder : address input to word line
    - Read/Write control
    - Input data/Output data

A 4 × 4 RAM



Coincident Decoding

- A two-dimensional selection scheme
  - To reduce the complexity of the decoding circuits
  - To arrange the memory cells in an array that is close as possible to square



Example

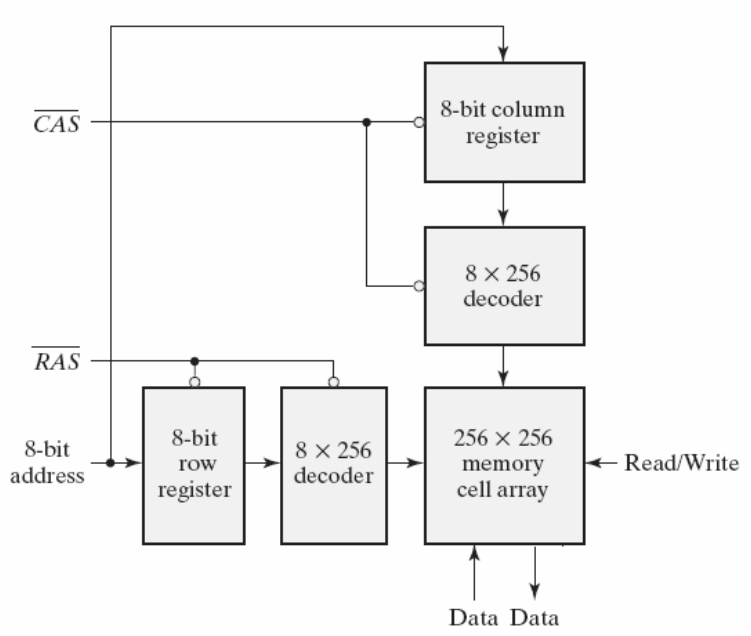
- A 10-to-1024 decoder
  - 1024 AND gates with 10 inputs per gates
- Two 5-to-32 decoders
  - 2 * (32 AND gates with 5 inputs per gates)
- Each word in the memory is selected by the coincidence between 1 of 32 rows and 1 of 32 columns for a total 1,024 words.
- Two dimensional decoding structure reduces the circuit complexity and the cycle time of the memory.

Address Multiplexing

- Address multiplexing reduces the number of pins in the IC package (especially DRAM)
  - DRAM is about four times the density of SRAM, thus has larger capacity than SRAM and more address bits.
  - Ex. a 64M*1 DRAM
    - 26-bit address lines
  - In order to reduce the pins in IC package, multiplexing the address lines (espcially in DRAM) is utilized in one set of address input pins.

Address Multiplexing

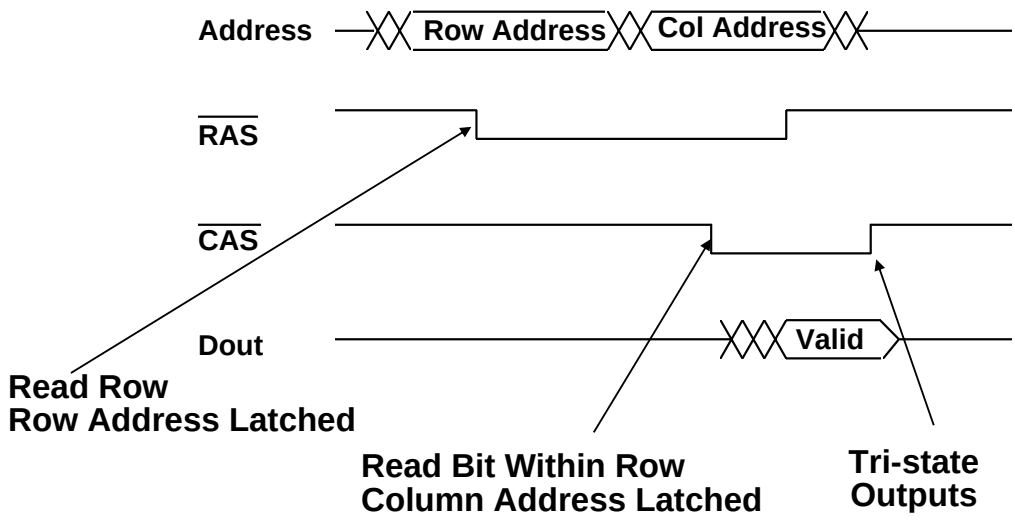
- An examples
  - RAS – row address strobe
  - CAS – column address strobe
- 8-bit MSB and 8-bit LSB addresses are multiplexed into column register through 8 input pins



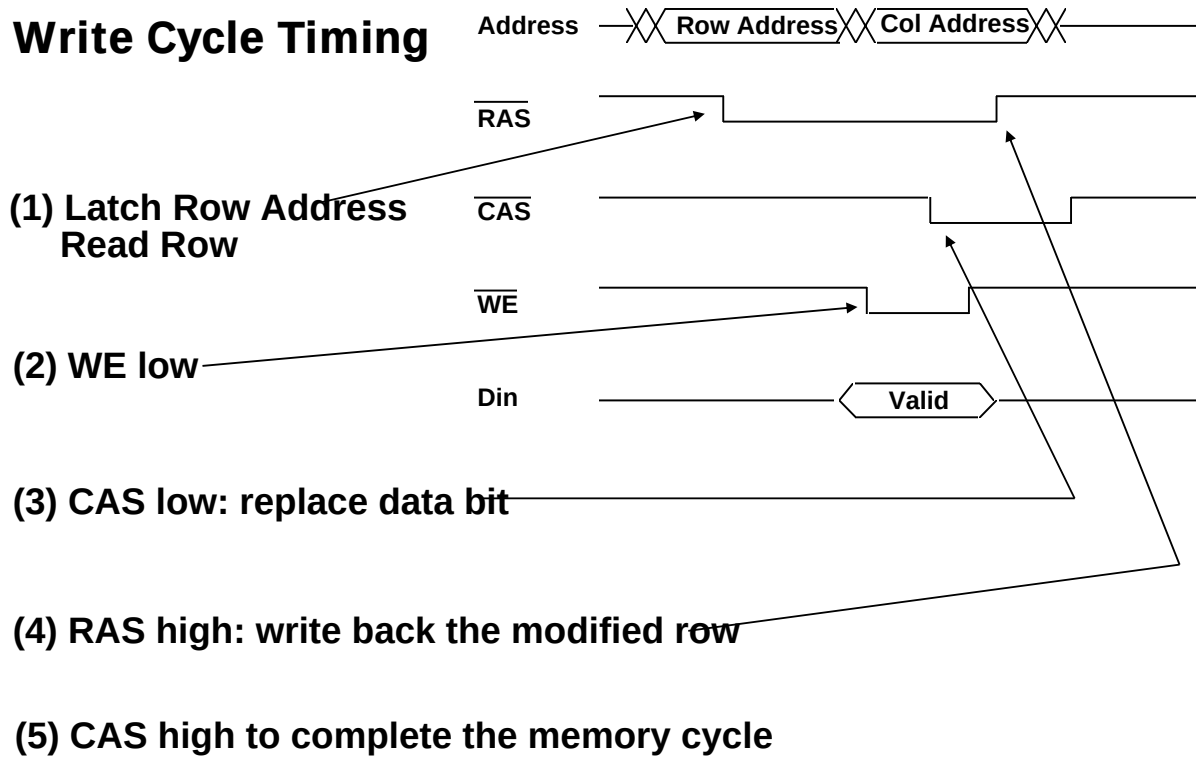
RAS, CAS Addressing - Read

- Even to read 1 bit, an entire 64-bit row is read!
- Separate addressing into two cycles: Row Address, Column Address
- Saves on package pins, speeds RAM access for sequential bits!

Read Cycle



RAS, CAS Addressing - Write



Error Detection And Correction

- Error detection and correction improve the reliability of a memory unit.
- A simple error detection scheme
  - A parity bit (Sec. 3-8) is stored with the data and checked after reading out from the memory.
  - A single bit error can be detected, but cannot be corrected.
- An error-correction code
  - To generates multiple parity check bits and be stored with the memory
  - To read out and check a unique pattern, called a syndrome.
  - The specific bit in error can be identified and corrected by implementing erroneous bits.

Hamming Code

- k parity bits are added to an n-bit data word
  - $(2^k - 1 \geqslant n + k)$
  - The bit positions are numbered in sequence from 1 to $n + k$
  - Those positions numbered as a power of 2 are reserved for the parity bits
  - The remaining bits are the data bits

Hamming Code – Encoding

- Example: 8-bit data word 11000100
  - Include 4 parity bits and the 8-bit word ⇒ 12 bits

    $2^k - 1 \geq n + k$, $n = 8 \Rightarrow k = 4$

    Bit position: 1 2 3 4 5 6 7 8 9 10 11 12

    P_1 P_2 1 P_4 1 0 0 P_8 0 1 0 0
  - Calculate the parity bits: even parity – assumption

    P_1 = XOR of bits (3, 5, 7, 9, 11) = $1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$

    P_2 = XOR of bits (3, 6, 7, 10, 11) = $1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$

    P_4 = XOR of bits (5, 6, 7, 12) = $1 \oplus 0 \oplus 0 \oplus 0 = 1$

    P_8 = XOR of bits (9, 10, 11, 12) = $0 \oplus 1 \oplus 0 \oplus 0 = 1$
  - Store the 12-bit composite word in memory.

    Bit position: 1 2 3 4 5 6 7 8 9 10 11 12

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

Hamming Code – Decoding

- When the 12 bits are read from the memory
  - Check bits are calculated using XOR operation for oddparity

    C_1 = XOR of bits (1, 3, 5, 7, 9, 11)

    C_2 = XOR of bits (2, 3, 6, 7, 10, 11)

    C_4 = XOR of bits (4, 5, 6, 7, 12)

    C_8 = XOR of bits (8, 9, 10, 11, 12)

  - If no error has occurred

    Bit position: 1 2 3 4 5 6 7 8 9 10 11 12

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

    $\Rightarrow \boldsymbol{C} = C_8C_4C_2C_1 = 0000$

Hamming Code – Decoding

- One-bit error
  - error in bit 1
    - C_1 = XOR of bits (1, 3, 5, 7, 9, 11) = 1
    - C_2 = XOR of bits (2, 3, 6, 7, 10, 11) = 0
    - C_4 = XOR of bits (4, 5, 6, 7, 12) = 0
    - C_8 = XOR of bits (8, 9, 10, 11, 12) = 0
    - $C_8C_4C_2C_1 = 0001$
  - error in bit 5
    - $C_8C_4C_2C_1 = 0101$
- Two-bit error can not be identified.
  - errors in bits 1 and 5
  - errors in bit 4
    - $C_8C_4C_2C_1 = 0100$

Hamming Code

- The Hamming code can be used for data of nay length
  - k check bits
  - $2^k - 1 \geq n + k$

Table 7.2
Range of Data Bits for k Check Bits

Number of Check Bits, *k*	Range of Data Bits, *n*
3	2–4
4	5–11
5	12–26
6	27–57
7	58–120

Single-Error Correction, Double-Error Detection

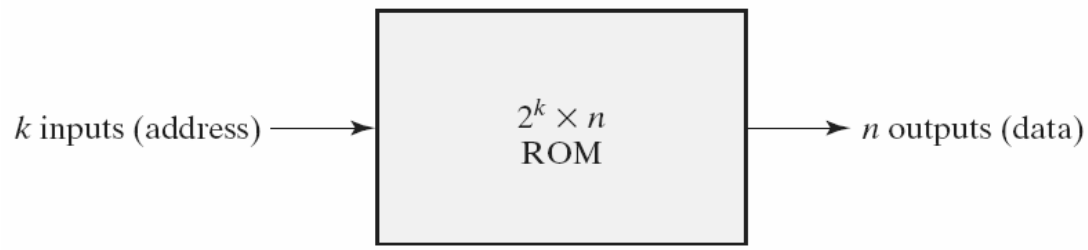
- Hamming code
  - It can detect and correct only a single error
  - Multiple errors may not be detected.
- Hamming code + a parity bit
  - It can detect double errors and correct a single error.
  - The additional parity bit is the XOR of all the other bits.
    - E.g.: the previous 12-bit coded word
      0 0 1 1 1 0 0 1 0 1 0 0 P_{13} $\Rightarrow$ 0 0 1 1 1 0 0 1 0 1 0 0 1 (even parity).

Single-Error Correction, Double-Error Detection

- When the word is read from memory
  - If P = 0, the parity is correct; P = 1, incorrect
  - Four cases
    - 1. If C = 0, P = 0, no error
    - 2. If C $\neq$ 0, P = 1, a single error that can be corrected
    - 3. If C $\neq$ 0, P = 0, a double error that is detected but cannot be corrected
    - 4. If C = 0, P = 1, an error occurred in the P_{13} bit

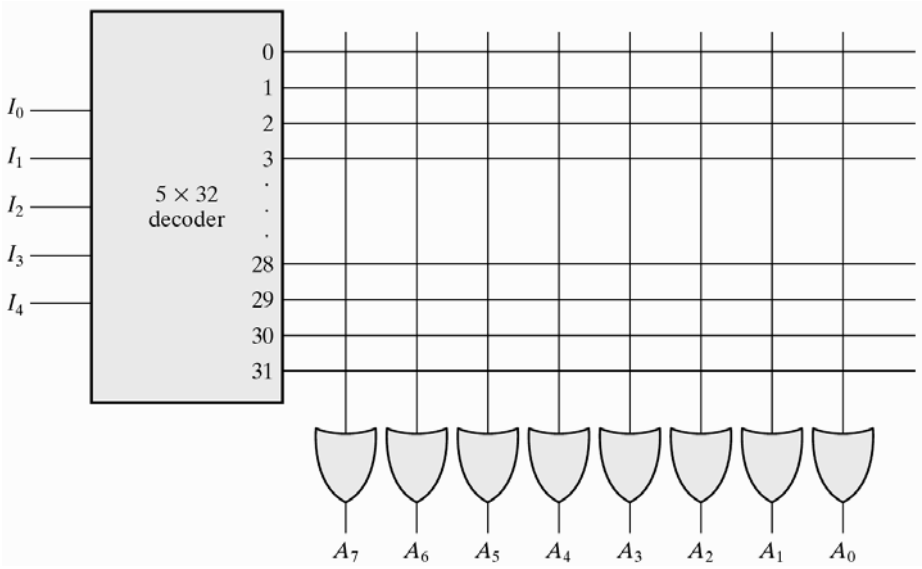
Read-Only Memory

- ROM stores permanent binary information
  - Once the pattern is established in the ROM, it stays within the unit when power is on or off.
- 2^k x n ROM
  - k address input lines
  - enable input(s)
  - three-state outputs



Example

- 32 x 8 ROM
  - 5 (k) – to – 32 (2^k) decoder
  - 32 (2^k) outputs of the decoder are connected to each of the eight OR gates, which have 32(2^k) inputs
  - 32 x 8 internal programmable connections



ROM Programming

- Programmable interconnections
  - close [1]: connected to high voltage
  - open [0]: ground left
  - A fuse that can be blown by applying a high voltage pulse

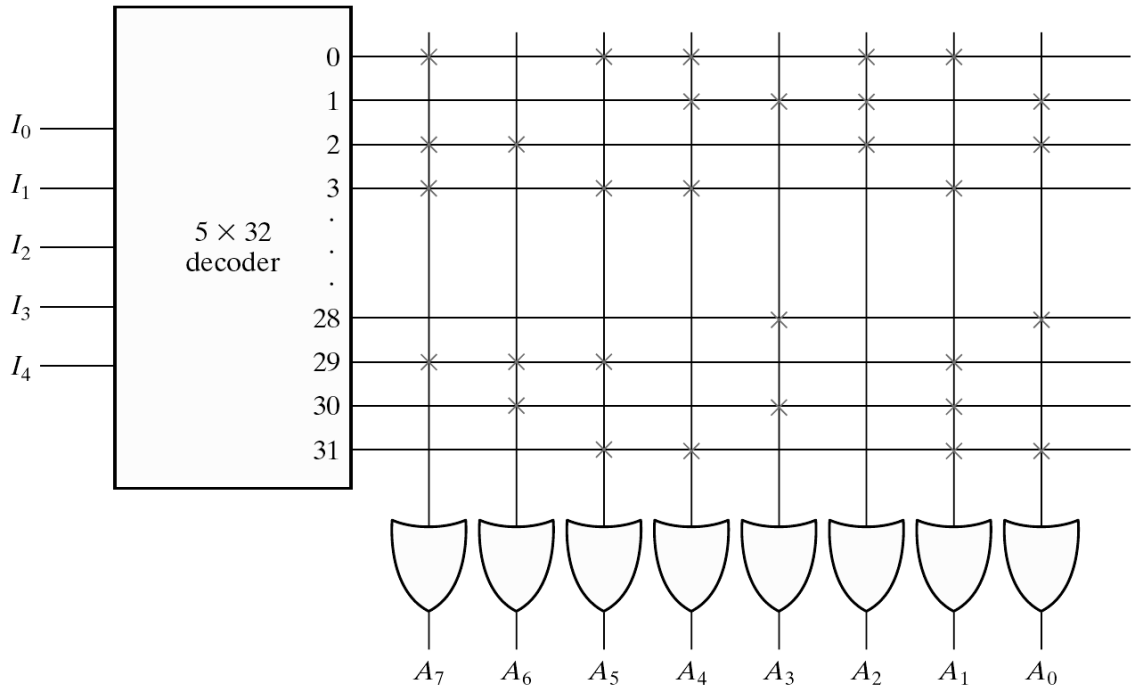
Example

ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
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	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	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

Programmable Interconnections

- ROM programming according to ROM table using fusing.



Combinational Circuit Implementation

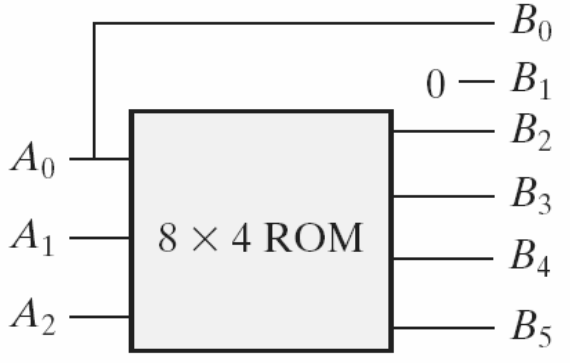
- The internal operation of ROM can be interpreted in two ways.
  - A memory contains a fixed pattern of stored words.
  - A unit that implements a combination circuit as sum of minterns in Section 4-8.
- ROM: a decoder + OR gates
  - a Boolean function = sum of minterms
  - Example: $A_7 = (I_4, I_3, I_2, I_1, I_0) = \Sigma(0, 2, 3, \ldots, 29)$
  - For an n-input, m-output combinational circuit
    $\Rightarrow 2^n \times m$ ROM
- Design procedure:
  1. Determine the size of ROM
  2. Obtain the programming truth table of the ROM
  3. The truth table = the fuse pattern

Example

- A combination circuit has 3 inputs, 6 outputs
- $B_1 = 0$, $B_0 = A_0$
- 8 x 4 ROM is used.

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

ROM Implementation



Truth Table

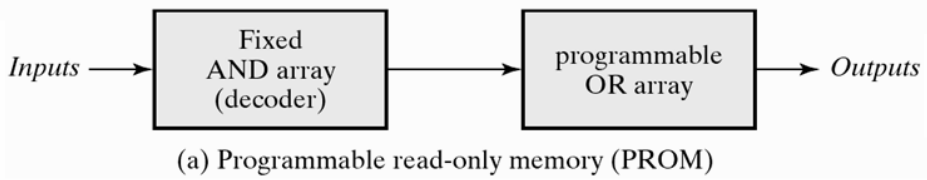
A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

Types of ROM

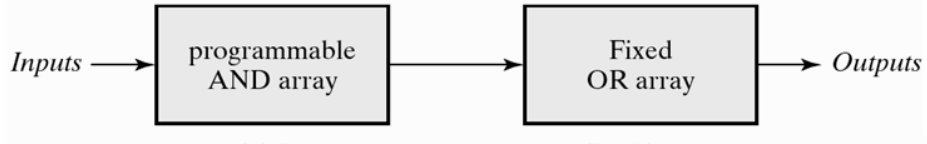
- Mask programming ROM
  - Data is programmed by IC manufacturers
  - It is economical if a large guantity of the same ROM
- PROM: Programmable ROM
  - Fuses in the ROM are blown by high-voltage pulse
  - Universal programmer can program PROM one time.
- EPROM: erasable PROM
  - Floating gate
  - The data in the ROM can be erased by ultraviolet light.
- EEPROM: electrically erasable PROM
  - Longer time is needed to write
  - Flash ROM
  - Limited times of write operations

Combinational PLDs

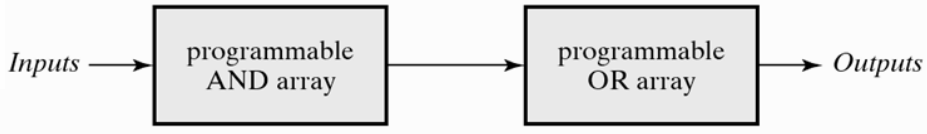
- PLD : Programmable logic device
- Programmable two-level logic
  - Sum of products
  - an AND array and an OR array



(a) Programmable read-only memory (PROM)



(b) Programmable array logic (PAL)



(c) Programmable logic array (PLA)

Programmable Logic Array

- PLA has two main parts
  - An array of programmable AND gates
    - can generate any product terms of the inputs
  - An array of programmable OR gates
    - can generate the sums of the products
- PLA is more flexible than programmable ROM
  - PLA uses any combination of products while programmable ROM uses sum of minterms
- PLA uses less circuits than programmable ROM
  - Only the needed product terms are generated in PLA while all the minterms must be generated in programmable ROM.

Programmable Logic Array

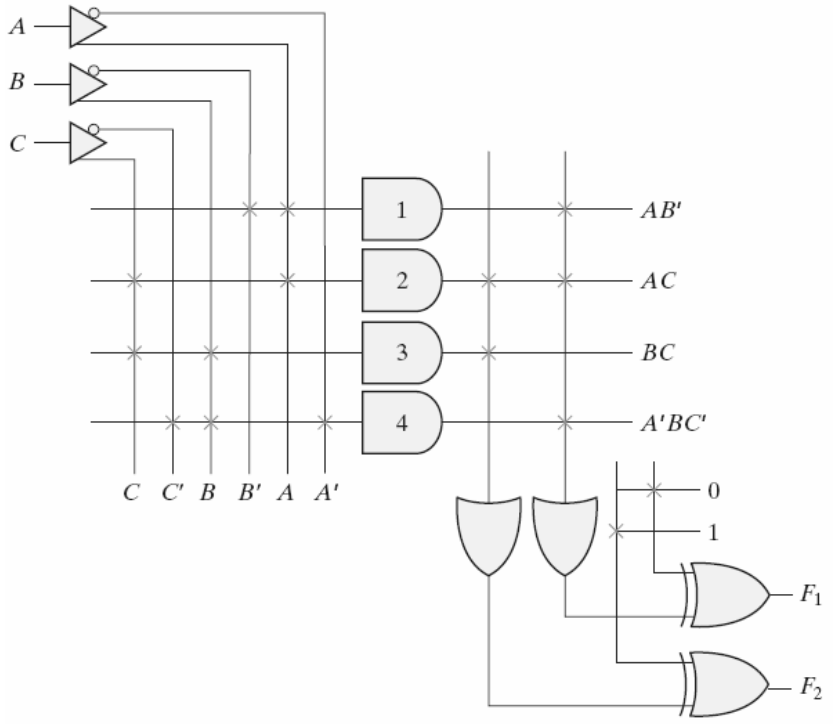
- An example:

➢ F_1 = AB' + AC + A'BC'

➢ F_2 = (AC + BC)'

➢ XOR gates can invert

the outputs



PLA Programming Table

- PLA programming table
  - specify the fuse map
  - 1st column: List the product terms numerically
  - 2nd column: Specify the required paths between inputs and AND gates.
  - 3rd column: Specify the required paths between the AND gates and OR gates.
  - (T)(C) stands for true or complement for programming XOR gate.

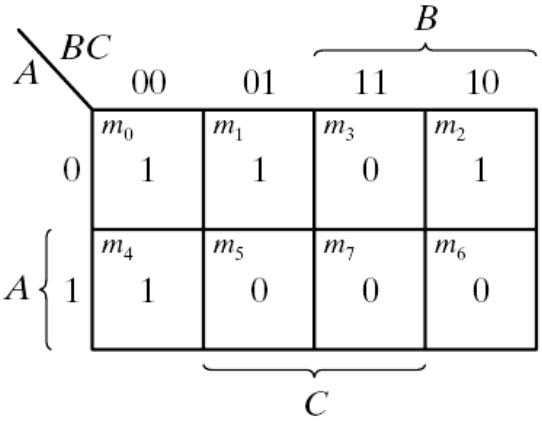
		Inputs			Outputs (T)	Outputs (C)
	Product Term	*A*	*B*	C	F_1	F_2
AB'	1	1	0	—	1	—
AC	2	1	—	1	1	1
BC	3	—	1	1	—	1
$A'BC'$	4	0	1	0	1	—

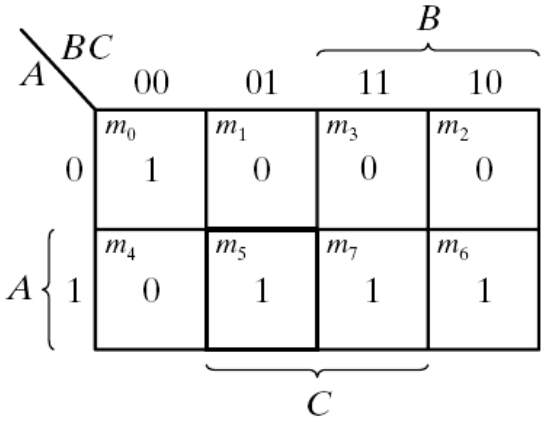
Characteristics of PLA

- The size of a PLA is determined by
  - The number of inputs
  - The number of product terms (AND gates)
  - The number of outputs (OR gates)
- When implementing with a PLA
  - One must reduce the number of distinct product terms before implementation.
  - The number of terms in a product is not important.
- Mask programmable and field programmable

Example

- Example
  - $F_1(A, B, C) = \sum(0, 1, 2, 4)$; $F_2(A, B, C) = \sum(0, 5, 6, 7)$
  - both the true value and the complement of the function should be simplified to check





$F_1 = A'B' + A'C' + B'C'$
$= (AB + AC + BC)'$

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

PLA Programming Table

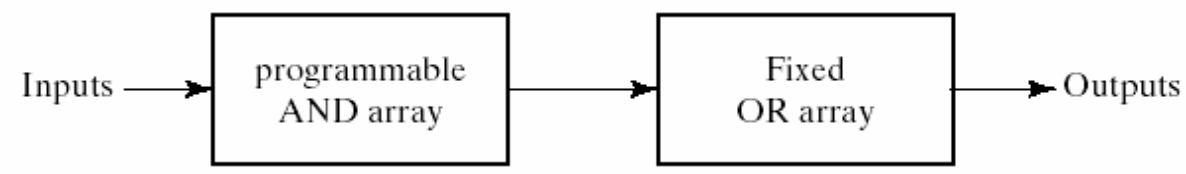
- $F_1 = (AB + AC + BC)'$
- $F_2 = AB + AC + A'B'C'$

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	–
$A'B'C'$	4	0	0	0	–	1

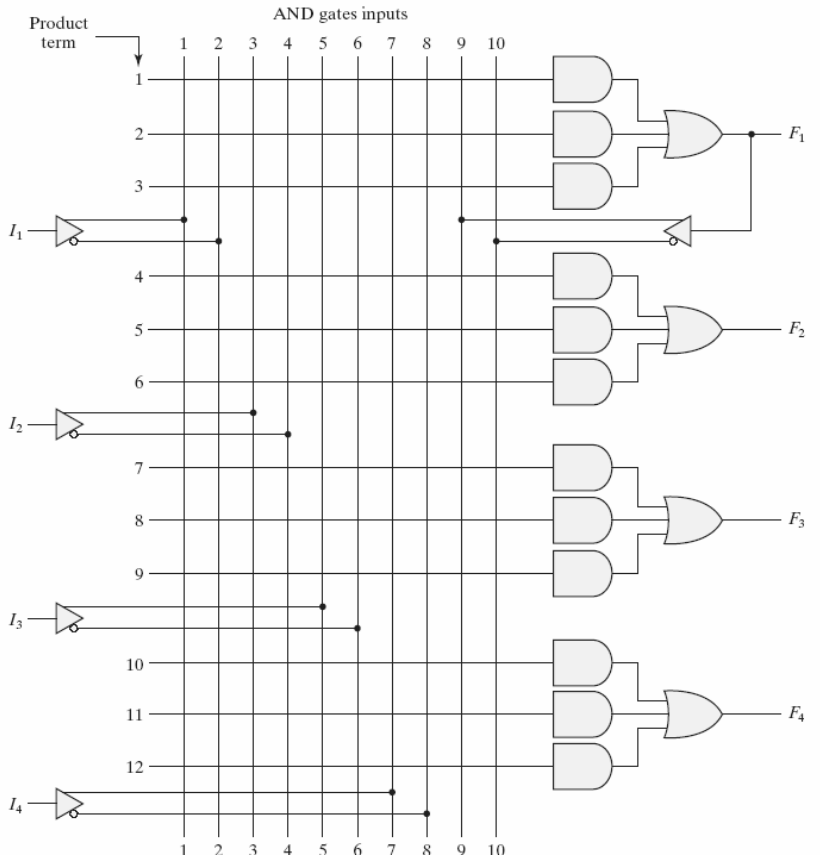
Programmable Array Logic

- A programmable AND array and a fixed OR array
  - The PAL is easier to program, but is not as flexible as the PLA



PAL Structure

An example PAL
  - product terms cannot be shared.



An Example Implementation

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

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

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

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

- Simplify the functions

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

$x = A + BCD$

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

$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$

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

PAL Programming Table

	AND Inputs					
Product Term	A	B	C	D	w	Outputs
1	1	1	0	—	—	$w = ABC' + A'B'CD'$
2	0	0	1	0	—	
3	—	—	—	—	—	
4	1	—	—	—	—	$x = A + BCD$
5	—	1	1	1	—	
6	—	—	—	—	—	
7	0	1	—	—	—	$y = A'B + CD + B'D'$
8	—	—	1	1	—	
9	—	0	—	0	—	
10	—	—	—	—	1	$z = w + AC'D' + A'B'C'D$
11	1	—	0	0	—	
12	0	0	0	1	—	

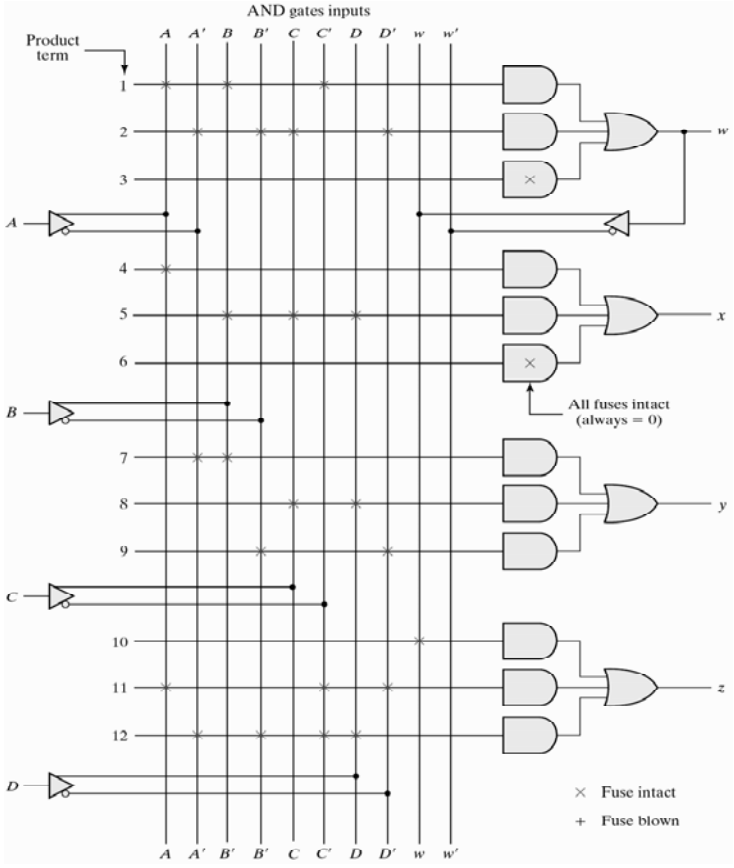
Programmed PLA

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

$x = A + BCD$

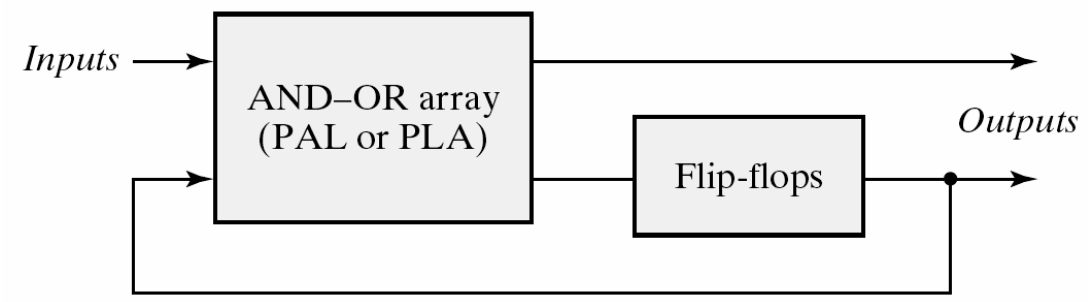
$y = A'B + CD + B'D'$

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



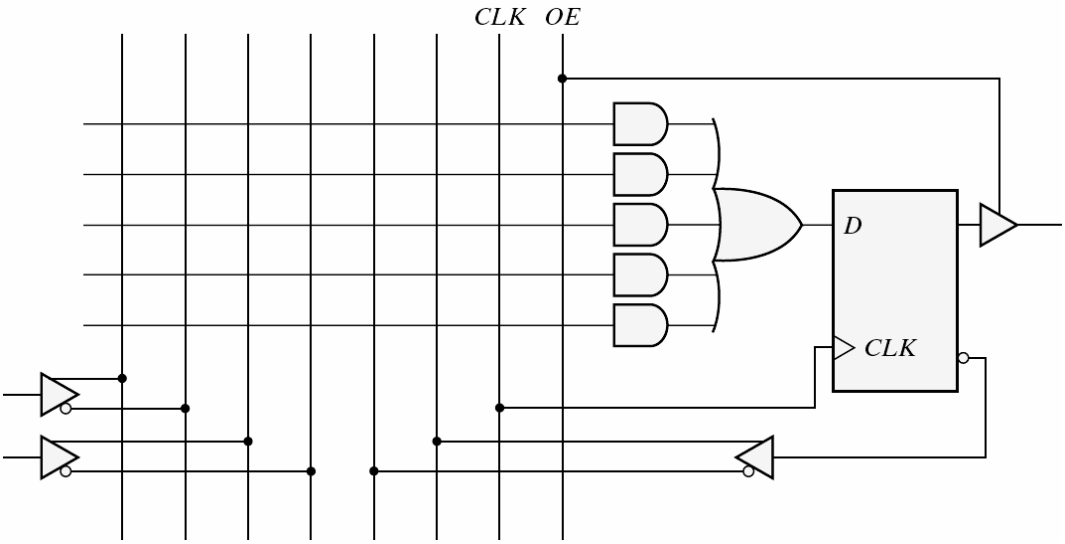
Sequential Programmable Devices

- Sequential programmable devices include both programmable gates and flip-flop.
- Three major types of sequential programmable devices
  - SPLD : Sequential programmable logic device
  - CPLD : Complex programmable logic device
  - FPGA : Field programmable gate array



Sequential PLD

- A PAL or PLA is modified by including a number of flip-flops
  - Sequential PLD macrocell
  - A typical SPLD contains 8-10 macrocells

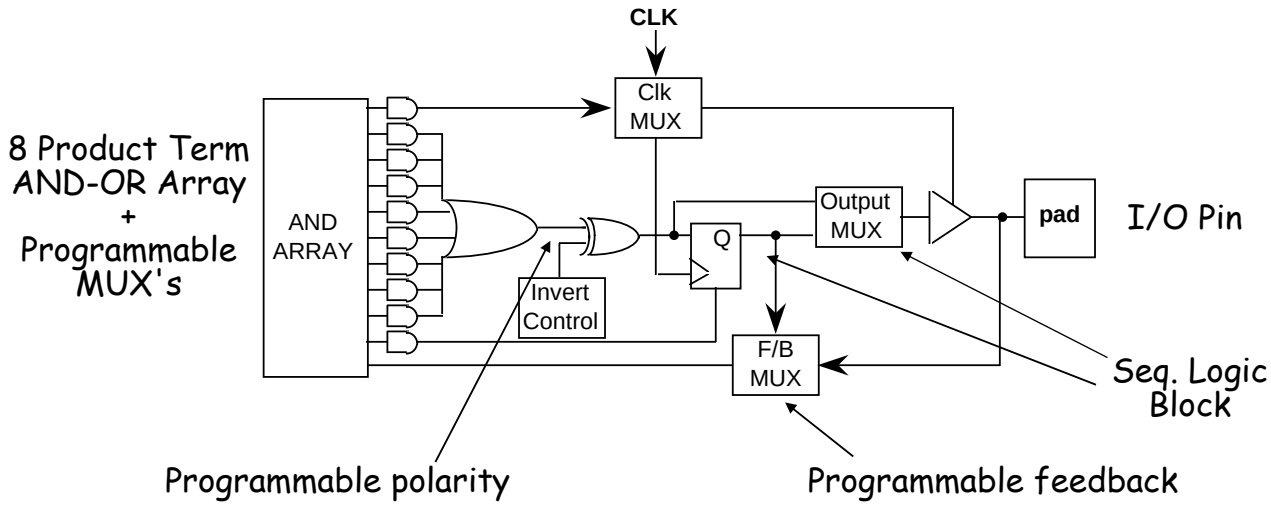


Sequential PLD

- Programming features:
  - AND array
  - use or bypass the flip-flop
  - select clock edge polarity
  - preset or clear for the register
  - complement an output
  - programmable input/output pins

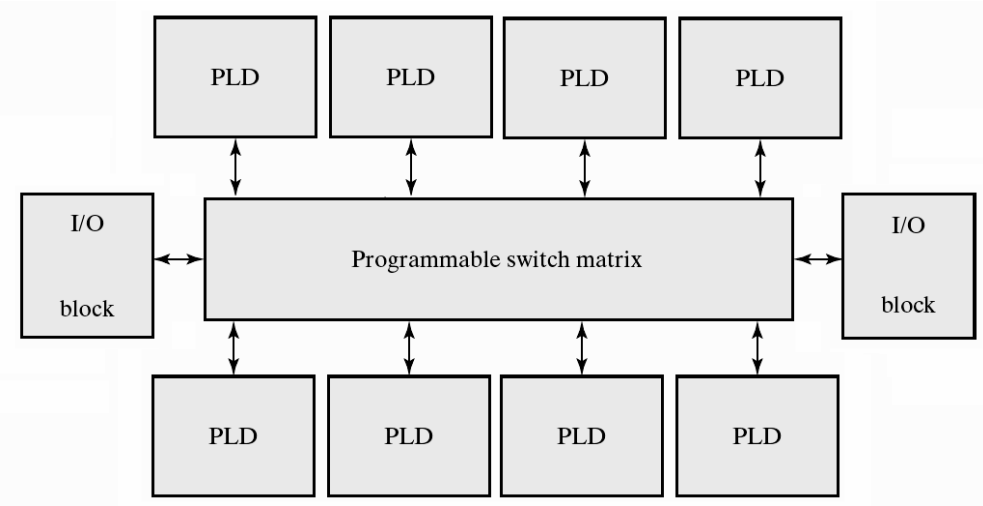
Example

- Altera macrocell



CPLD

- Complex PLD
  - Puts a lot of PLDs on a chip
  - Adds wires between them whose connections can be programmed
  - Uses fuse/EEPROM technology

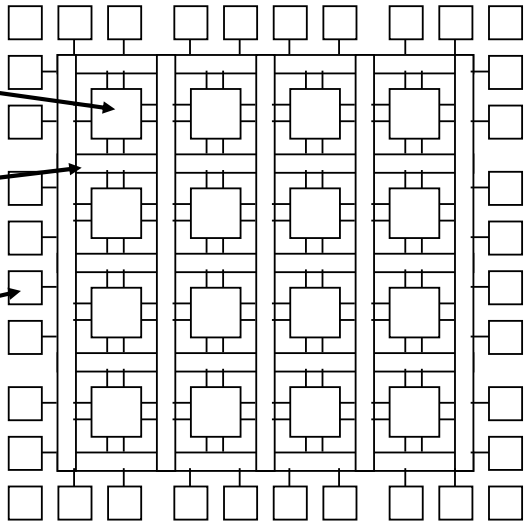


FPGA

- Field-Programmable Gate Array
  - Emulate gate array technology
  - Hence *Field Programmable Gate Array*
  - You need:
    - A way to implement logic gates
    - A way to connect them together
  - SRAM is often used to configure the combinational circuit function. (Volatile)
- PALs, PLAs = 10 - 100 Gate Equivalents
- Field Programmable Gate Arrays = FPGAs
  - 100 - 1000(s) of Gate Equivalents

Field-Programmable Gate Arrays

- Logic blocks
  - To implement combinational and sequential logic
- Interconnect
  - Wires to connect inputs and outputs to logic blocks
- I/O blocks
  - Special logic blocks at periphery of device for external connections



- Key questions:
  - How to make logic blocks programmable?
  - How to connect the wires?

Basic Xilinx Architecture