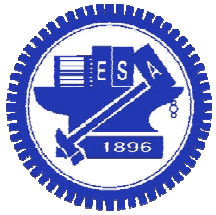
Memory and Programmable Logic



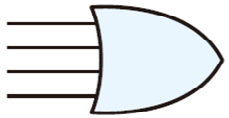
Chun-Jen Tsai
National Chiao Tung University
12/06/12

High Capacity Memory Device

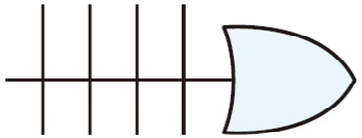
- Flip-flops are used only for small amount of data storage; different technology must be used for large amount of data storage
- RAM – Random-Access Memory
  - Allows both read and write operations at any selected storage cells
- ROM – Read-Only Memory
  - Allows only read operation; can be treated as a programmable logic device

Programmable Logic Devices

- A programmable logic device (PLD) is a device that has a generic gate level structure that can be used to implement different digital circuits



Conventional OR gate symbol

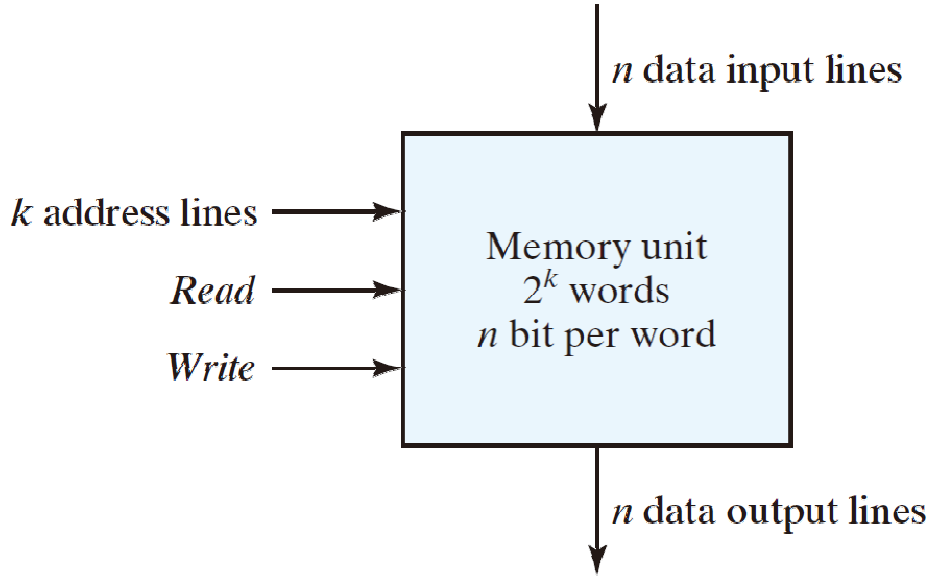


Programmable OR gate symbol

- Examples of PLD are:
  - ROM
  - Programmable Logic Array (PLA)
  - Programmable Array Logic (PAL)
  - Field-programmable Gate Array (FPGA)

Random-Access Memory

- A memory device stores binary information in groups of bits, typically called “words”
- Each word can be identified by an “address”
- Block diagram of a memory device



Example: an 1024×16 Memory

- A 1024×16 device has 10-bit address lines and 16-bit data lines
  - Data input lines and data output lines may be separated

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

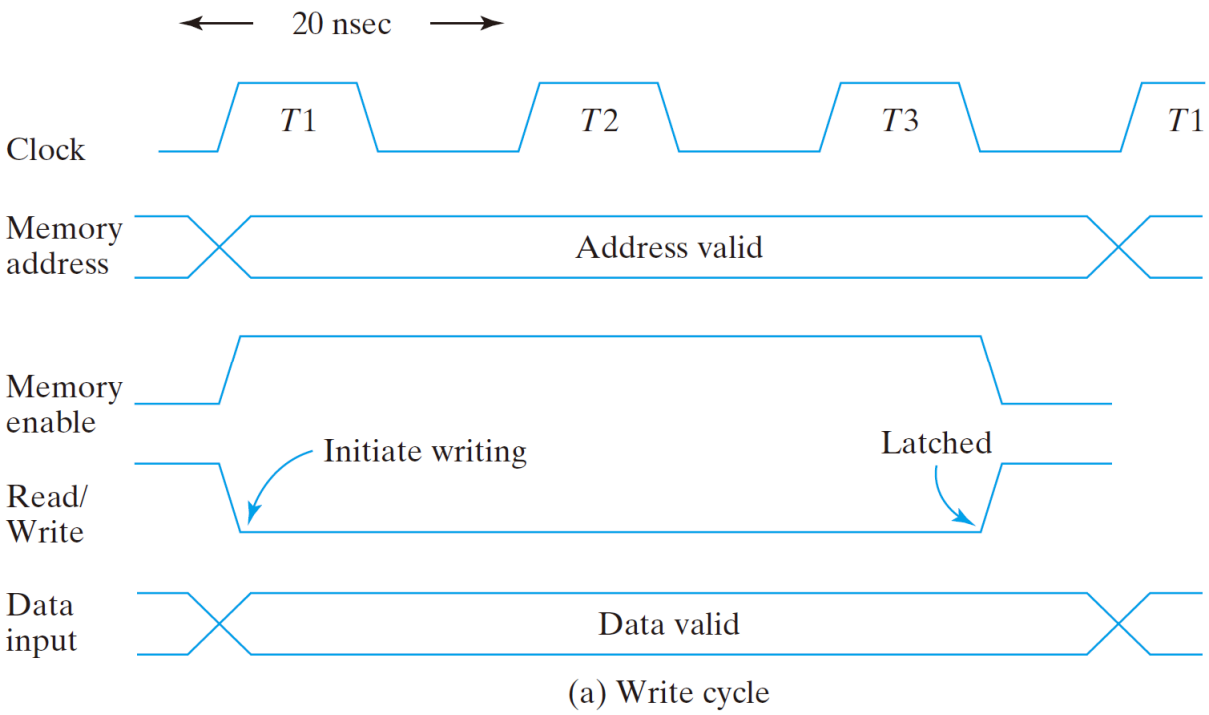
Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

Timing Waveforms

- The operation of the memory unit is controlled by an external device called memory controllers
- 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
- The read and write operations must be synchronized with an external clock

Timing Diagram of Write Operations

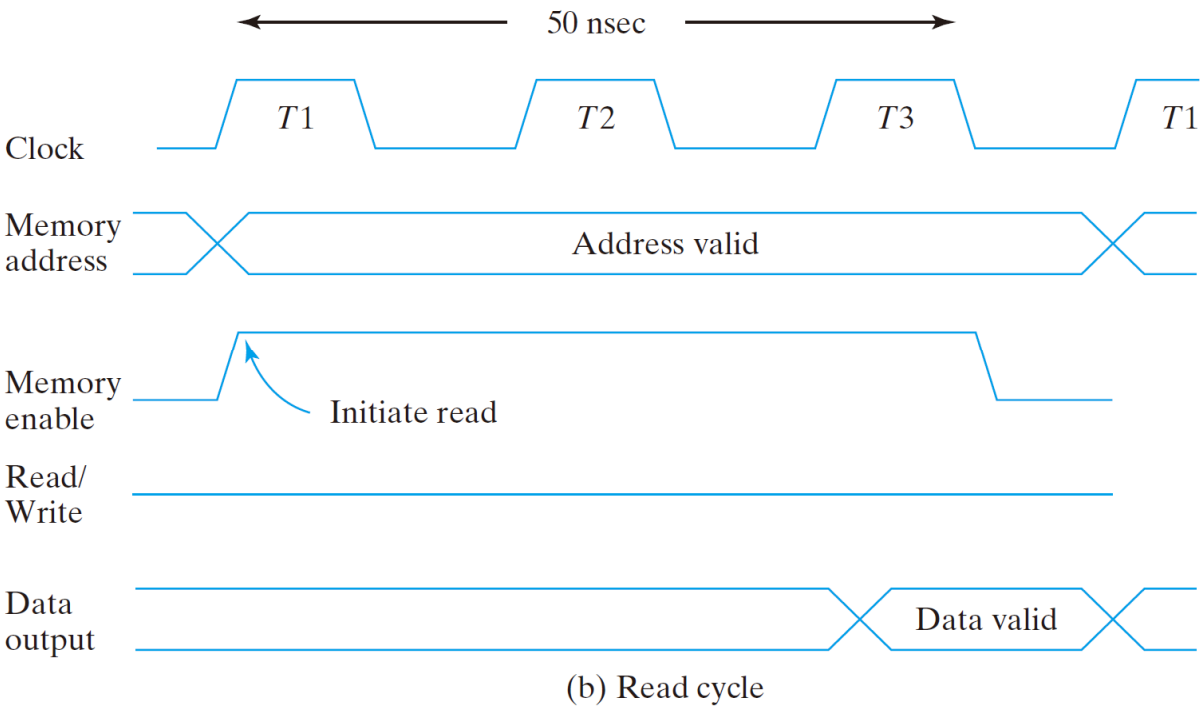
- CPU clock – 50 MHz
- The access/cycle time < 50 ns



(a) Write cycle

Timing Diagram of Read Operations

- In practice, due to the effect of the memory controller, a read cycle can be longer than a write cycle



(b) Read cycle

Types of Memories (1/2)

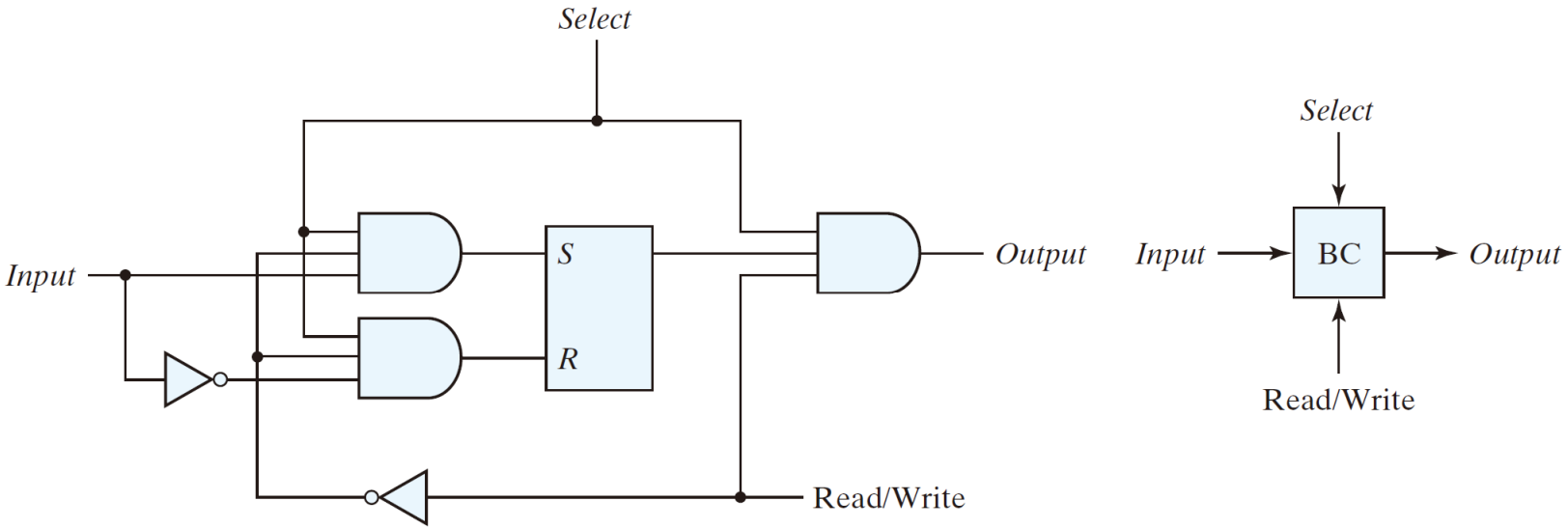
- Static memory devices (e.g. SRAM)
  - Information are stored in latches
  - Remains valid as long as power is applied
  - Short read/write cycle
- Dynamic memory devices (e.g. DRAM)
  - Information are stored in the form of charges on capacitors
  - The stored charge tends to discharge with time
  - Need to be refreshed (by read-and-write-back, similar to what a buffer gate does)
  - Reduced power consumption
  - Larger memory capacity

Types of Memories (2/2)

- Volatile memory
  - Lose stored information when power is turned off
  - SRAM, DRAM
- Non-volatile memory
  - Retains its stored information after the removal of power
  - ROM
  - EPROM, EEPROM
  - Flash memory

Memory Decoding

- A memory unit is a storage components that has a decoding circuits to select the memory word for read/write operation
- The logic diagram of a memory binary cell (BC):



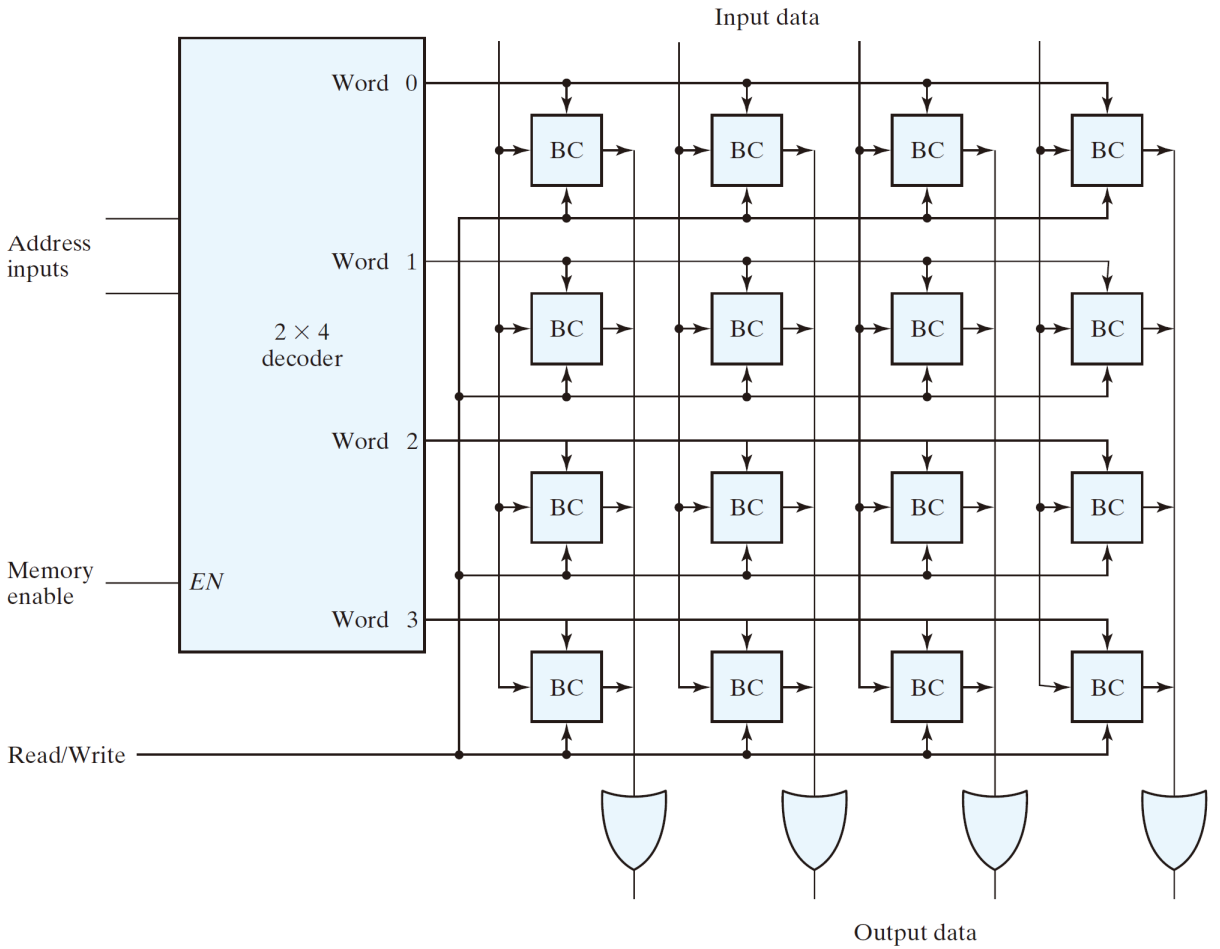
(a) Logic diagram (b) Block diagram

Internal Construction of RAM (1/2)

- A RAM of m words and n bits per word requires $m \times n$ binary storage cells
- The decoding circuits for selecting each individual word is a $k \times 2^k$ decoder, where $k = \lceil \log_2 m \rceil$
  - The decoder requires 2^k AND gates with k inputs per gate

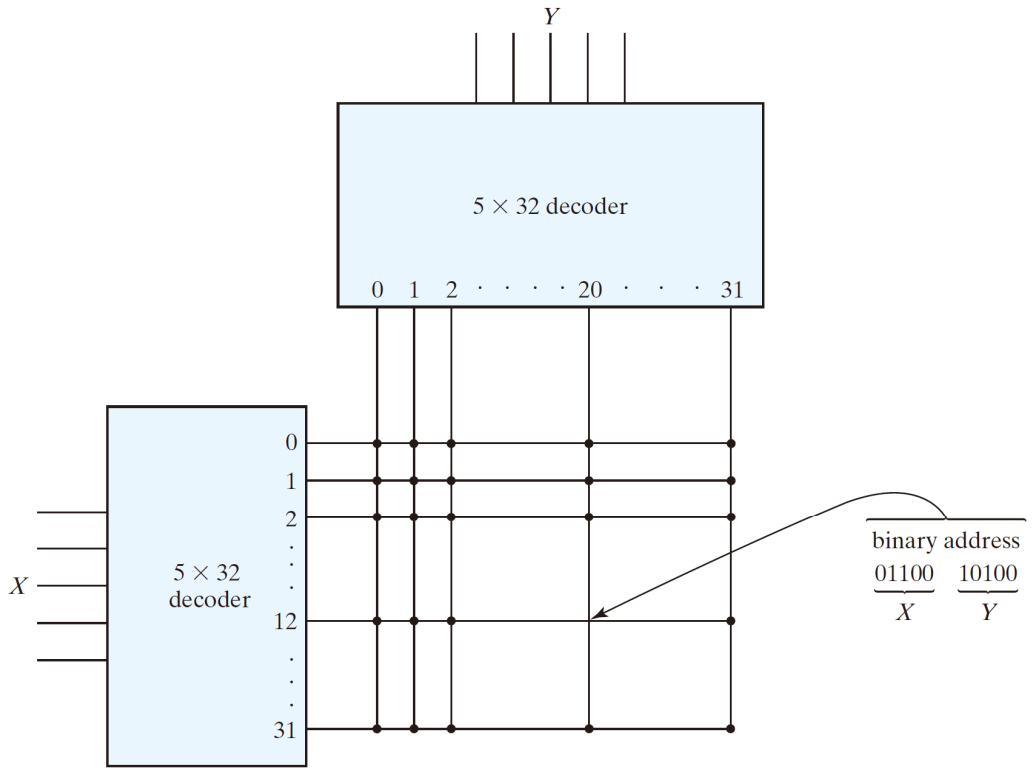
Internal Construction of RAM (2/2)

- Example: a 4×4 RAM (a four 4-bit words RAM)



Coincident Decoding

- To reduce the complexity of the address decoder, we can split the address into row and column addresses
  - A 10×1024 decoder needs 1024 10-input AND gates
  - Two 5×32 decoders need 2 × (32 5-input AND gates)

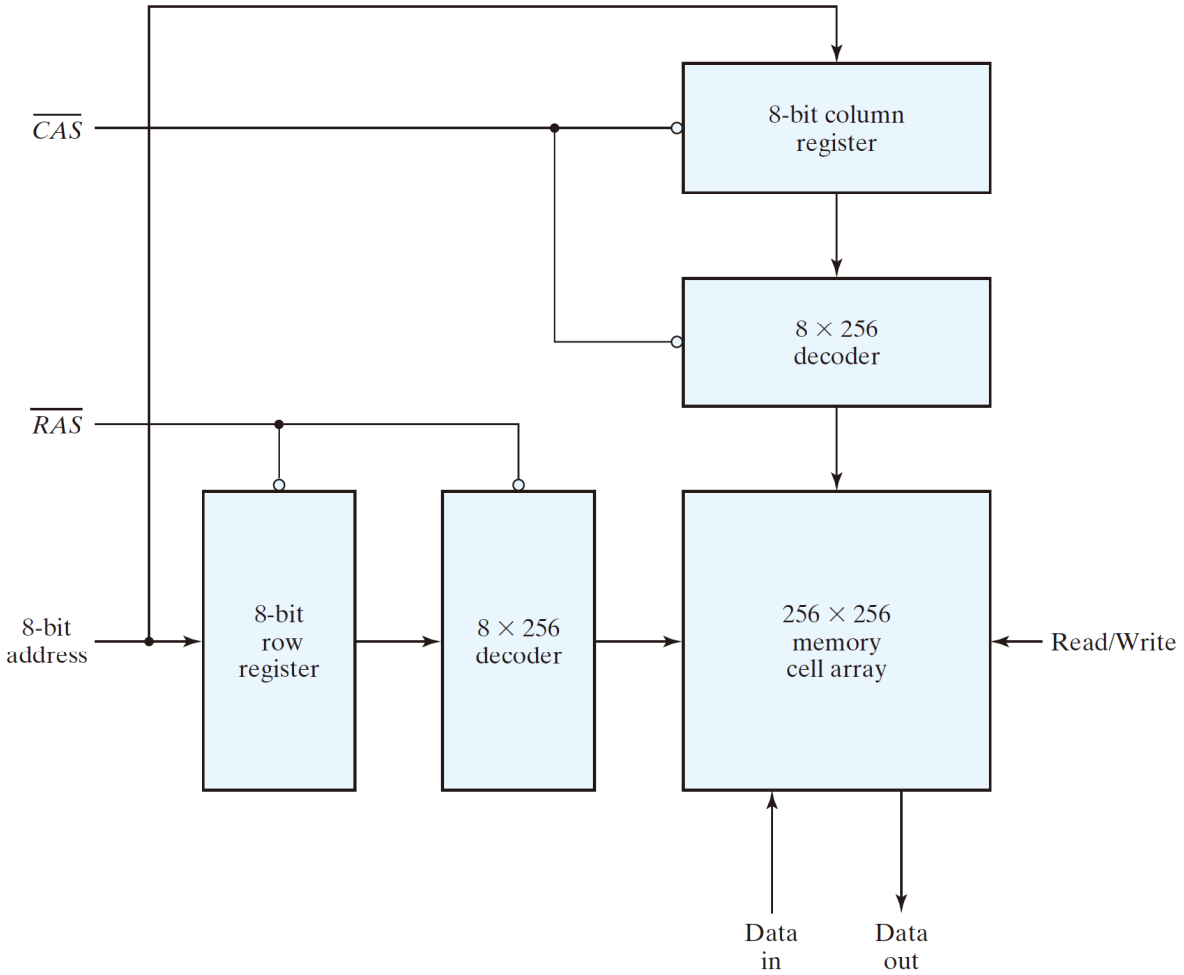


Address Multiplexing (1/2)

- Address multiplexing is often used to reduce the number of pins in the IC package
  - A 64M×1 DRAM requires 26-bit address lines, which can be multiplexed into one set of, say 10, address input pins
- As in coincident decoding, an address is composed of a column address and a row address
- The signals CAS and RAS are used to indicate the current address type on the address lines
  - RAS – row address strobe
  - CAS – column address strobe

Address Multiplexing (2/2)

- Address multiplexing of a 64K DRAM:



Error Detection And Correction

- Error detection and correction can improve the reliability of a memory unit
- A single parity bit can be used to detect errors in a word, but cannot correct errors
- An error-correction code must be used to correct errors:
  - Generates multiple parity check bits
  - The check bits generate a unique pattern, called a syndrome, which identify the specific bit in error

Hamming Code

- In Hamming code, k parity bits are added to an n-bit data word
  - The bit positions are numbered in sequence from 1 to $n + k$, those positions numbered as a power of 2 are for the parity bits and the remaining bits are the data bits
  - Example: for $n = 8$, $k = 4$:

Bit Position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	D_1	P_4	D_2	D_3	D_4	P_8	D_5	D_6	D_7	D_8

  - Note that k must fulfill the following constraint: $2^k - 1 \geq n + k$

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

ECC Example (1/3)

- For $n = 8$ bits data word 11000100, we can design the following even parity check rule:
  - P1 = XOR of bits (3, 5, 7, 9, 11) = 1 ⊕ 1 ⊕ 0 ⊕ 0 ⊕ 0 = 0
  - P2 = XOR of bits (3, 6, 7, 10, 11) = 1 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ 0 = 0
  - P4 = XOR of bits (5, 6, 7, 12) = 1 ⊕ 0 ⊕ 0 ⊕ 0 = 1
  - P8 = XOR of bits (9, 10, 11, 12) = 0 ⊕ 1 ⊕ 0 ⊕ 0 = 1
- Therefore, the stored 12-bit word becomes: 001110010100

ECC Example (2/3)

- When the 12 bits are read from the memory
  - Check bits are calculated

    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)

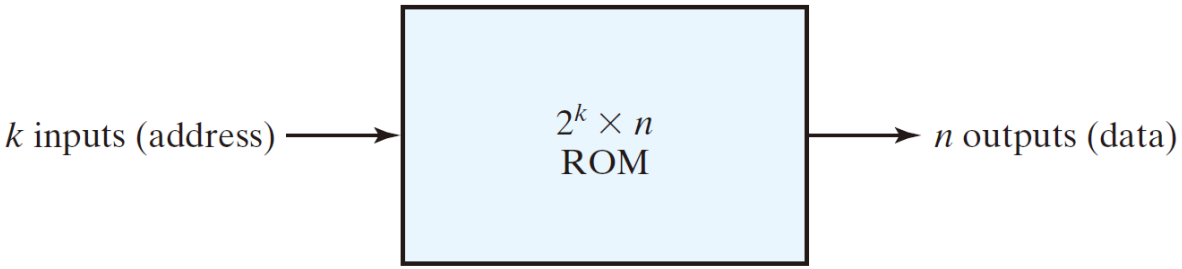
    C_8 = XOR of bits (8, 9, 10, 11, 12)
- If no error, the syndrome $C = C_8C_4C_2C_1 = 0000$
- If only a single error occurred in bit position j, then syndrome is the binary code of j. That is: $j = 8\times C_8 + 4\times C_4 + 2\times C_2 + C_1$
- If more than one bits are in error, they cannot be corrected

ECC Example (3/3)

- Previous Hamming code can detect and correct only a single error, multiple errors can not be detected.
- An extra even parity bit P of the $n+k$ coded bits can be used to further detect double errors:
  - If $C = 0$, $P = 0$, no error
  - If $C \neq 0$, $P = 1$, detects a single error that can be corrected
  - If $C \neq 0$, $P = 0$, detects an uncorrectable double-error
  - If $C = 0$, $P = 1$, an error occurred in the P bit

Read-Only Memory

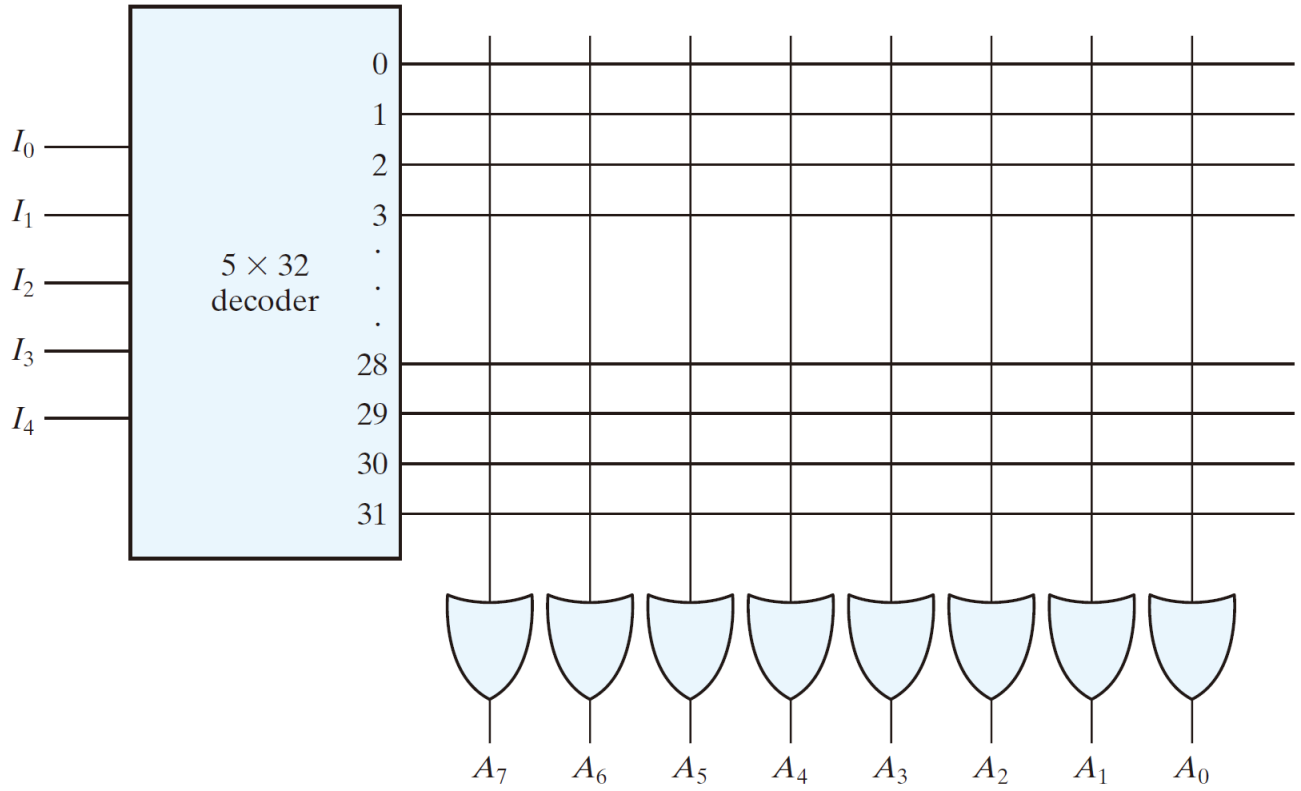
- ROM is used to store permanent binary information
- A $2^k \times n$ ROM has:
  - k address input lines
  - n data outputs



- A ROM used for circuit implementation also has
  - An enable inputs
  - Three-state outputs

Example: 32 × 8 ROM (1/3)

- Requires a 5×32 decoder
  - 8 OR gates (each has 32 inputs)
  - 32×8 internal programmable connections



Example: 32 × 8 ROM (2/3)

- The ROM contents can be specified by a truth table:

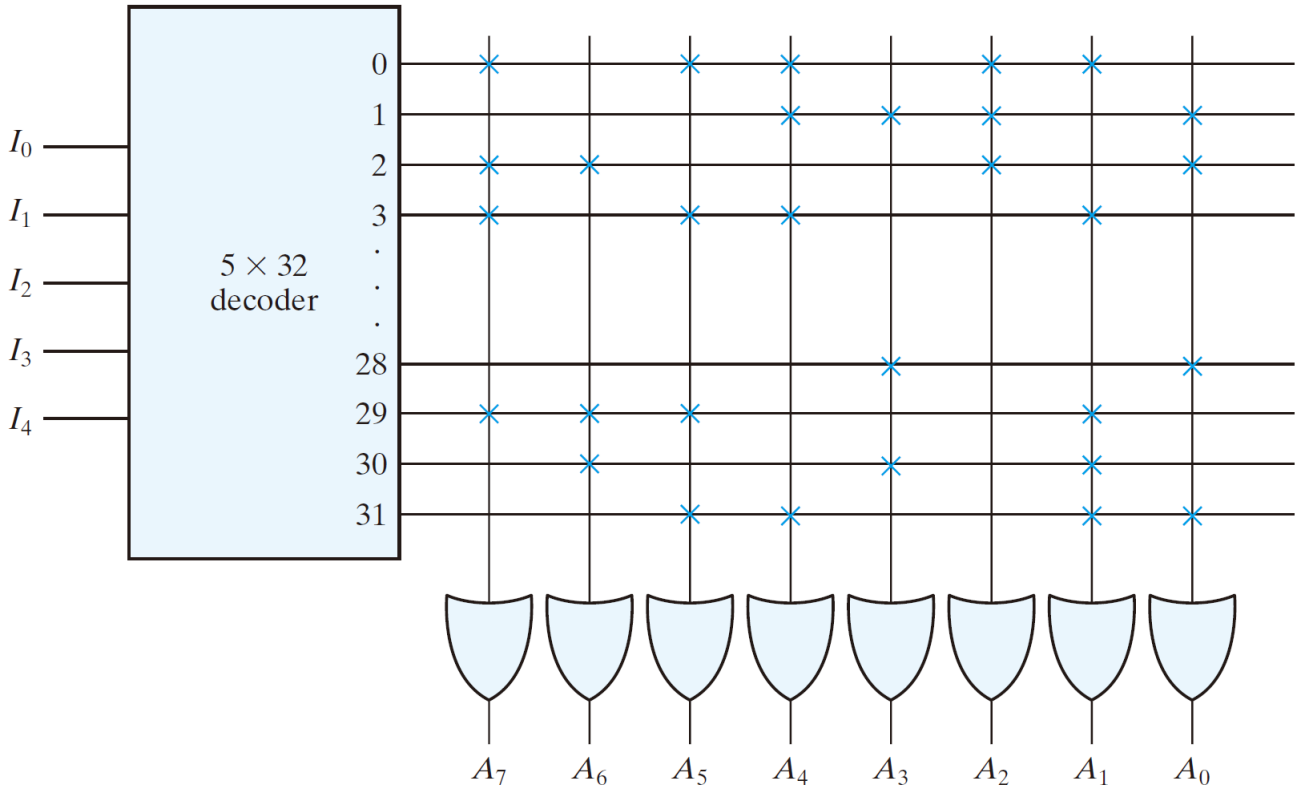
ROM Truth Table (Partial)

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 are used to program the ROM

Example: 32 × 8 ROM (3/3)

- Programmable interconnections can close or open the joint of two crossing lines by a fuse or anti-fuse



Combinational Circuit Using ROM

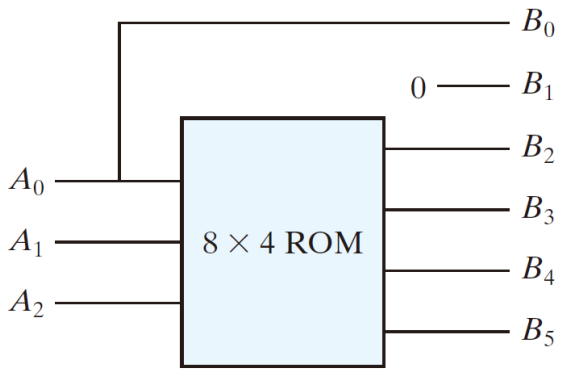
- A ROM is a decoder plus OR gates
  - Can express sum of minterms
  - For an n-input, m-output combinational circuit, we need a $2^n \times m$ ROM
- Design procedure:
  1. Determine the size of the ROM to be used
  2. Obtain the truth table to be implemented
  3. The truth table will be used as the fuse pattern

ROM of a 3-input, 6-ouput Function

- The truth table of the function:

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

- Simplified function:



(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

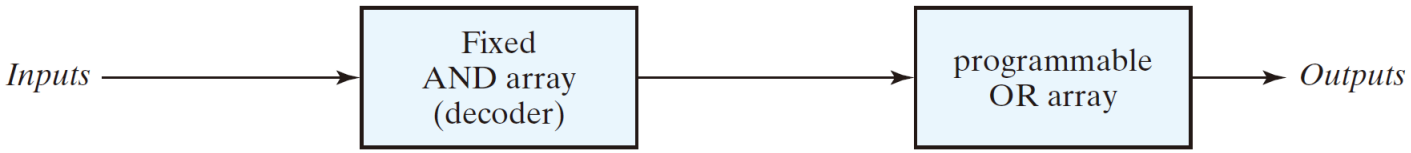
(b) ROM truth table

Types of ROM

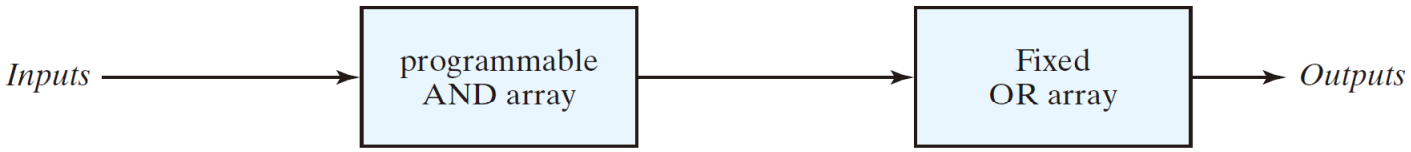
- Mask programming ROM
  - Through IC fabrication
  - Economical only if large quantities
- Programmable ROM (PROM)
  - Programming using fuses with a programmer
- Erasable PROM (EPROM)
  - Programming using floating gate transistors, using ultraviolet light for erasing
- Electrically erasable PROM (EEPROM)
  - Electronically erasable floating gate transistors
  - Popular today, known as flash storage
  - Limited times of write operations

Combinational PLDs

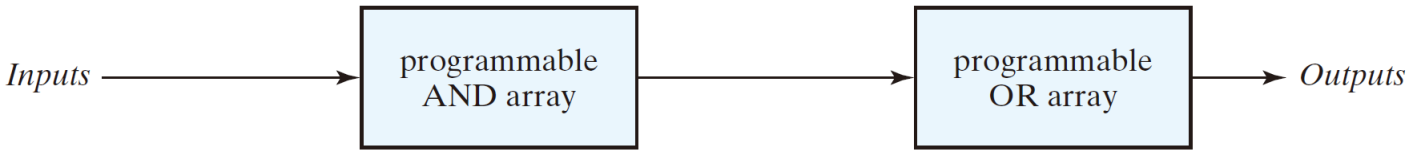
- Programmable two-level logic is composed of an AND array and an OR array



(a) Programmable read-only memory (PROM)



(b) Programmable array logic (PAL)



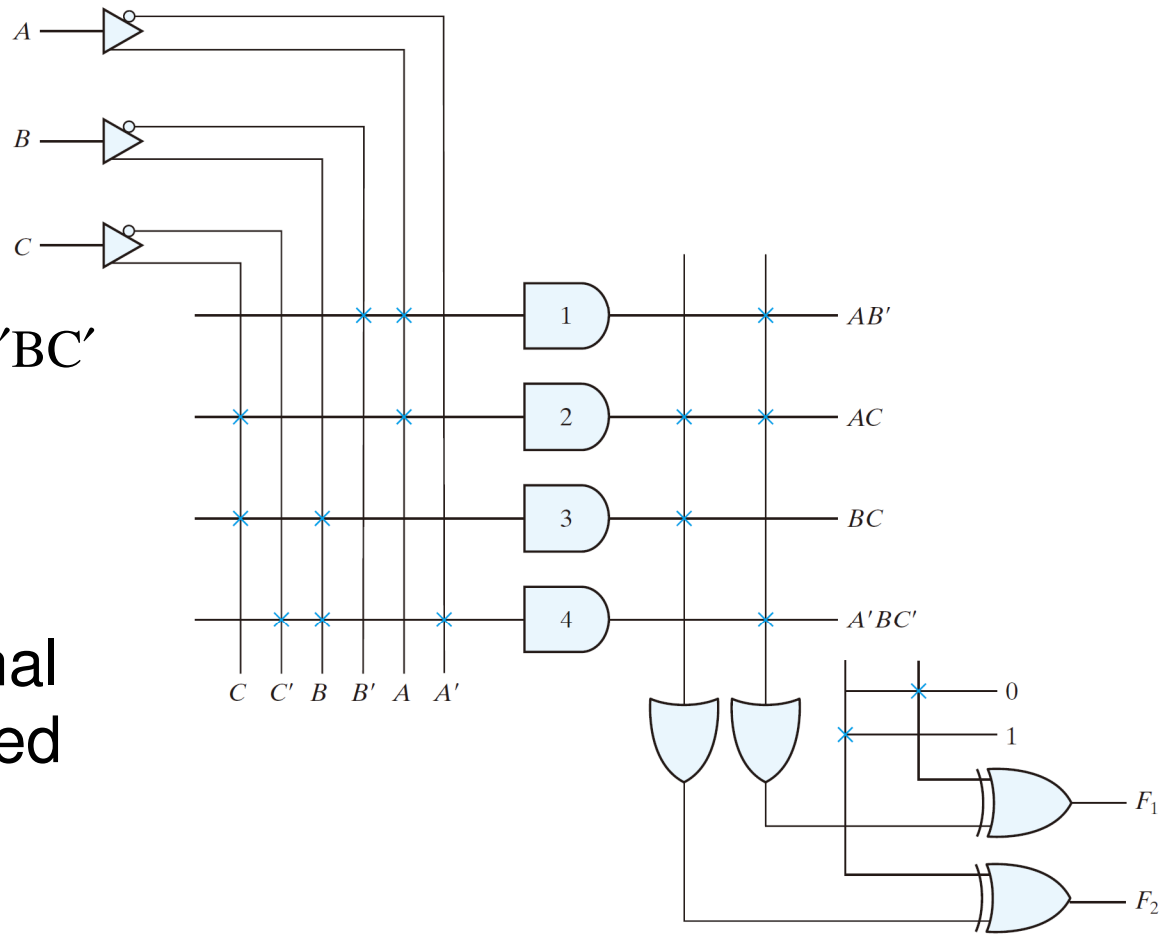
(c) Programmable logic array (PLA)

Programmable Logic Array

- A PLA has
  - 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
  - use less circuits than ROM
    - only the needed product terms are generated

PLA Implementation of a Function

- Example: for a 3-input, 2-output function:
  - $F_1 = AB' + AC + A'BC'$
  - $F_2 = (AC + BC)'$
- XOR gates are used to take either the original or complemented outputs



PLA Programming Table

- We use a PLA programming table to specify the fuse map of the device
- Example:

PLA Programming Table

		Inputs			Outputs (T)	(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	—

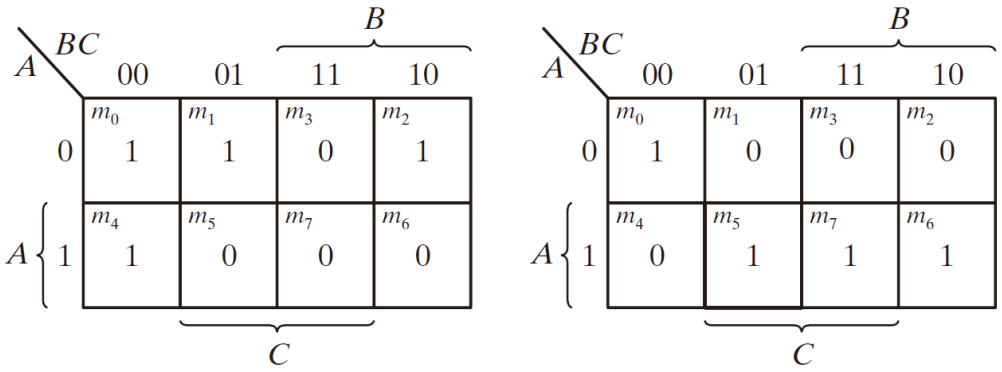
Note that:
1. T stands for “true” and C stands for “complement”
2. “–” means the literal (or product term) is absent

PLA Implementation Issues

- 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 we must reduce the number of distinct product terms; the number of terms in a product is not important.

Example of PLA Design

- $F_1(A, B, C) = \Sigma\,(0, 1, 2, 4)$; $F_2(A, B, C) = \Sigma\,(0, 5, 6, 7)$
  - Both the original and the complement of the function should be checked for simplification



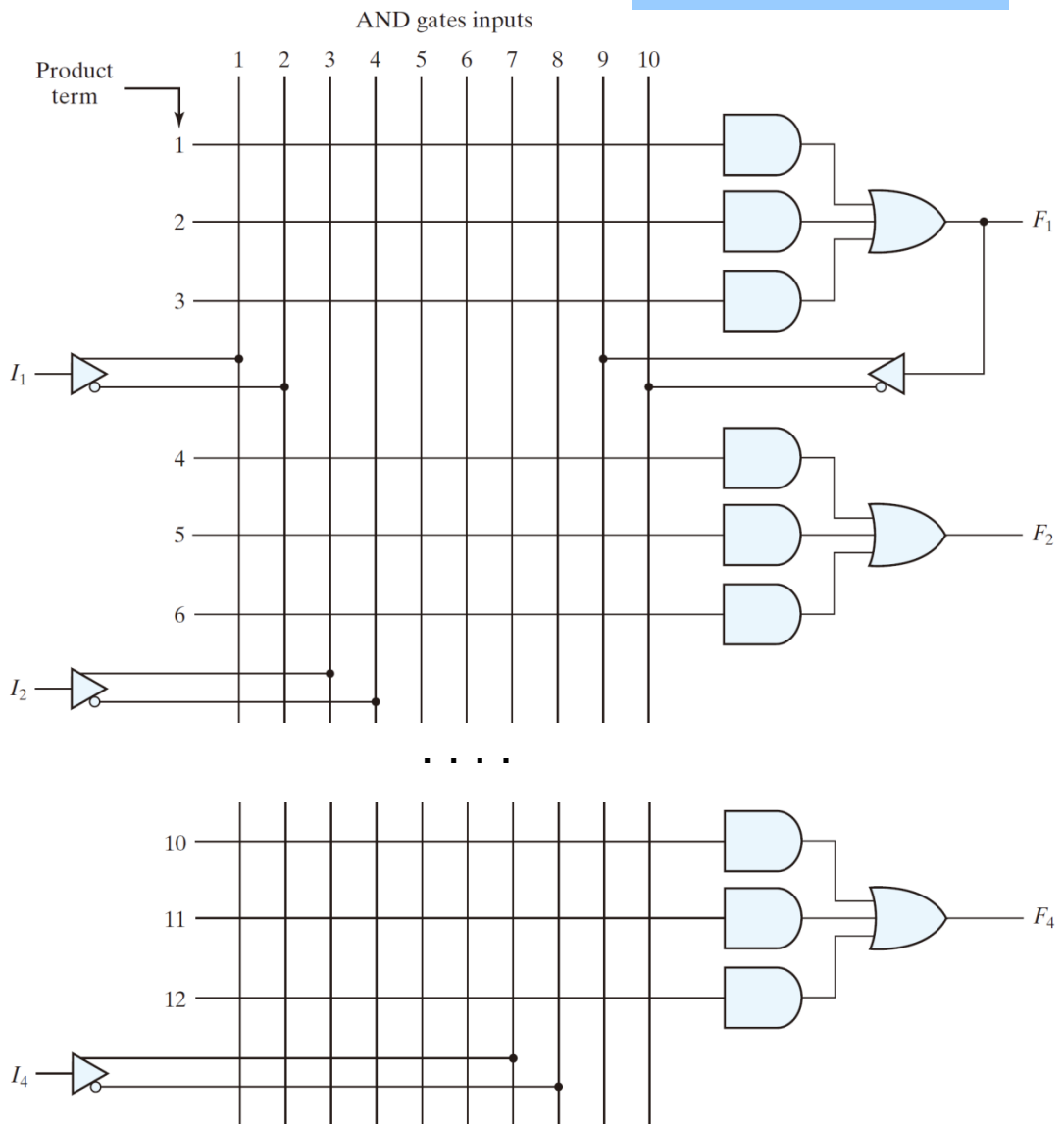
- Function form with minimal # product terms
  - $F_1 = (AB + AC + BC)'$
  - $F_2 = AB + AC + A'B'C'$

PLA programming table

	Product term	Inputs			Outputs (C)	Outputs (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

Programmable Array Logic

- A PAL is contains a programmable AND array and a fixed OR array
  - A PAL is easier to program, but is not as flexible as a PLA
  - For PAL, product terms cannot be shared



Example of PAL Design (1/3)

- Original functions:
  - $w(A, B, C, D) = \Sigma(2,12,13)$
  - $x(A, B, C, D) = \Sigma(7,8,9,10,11,12,13,14)$
  - $y(A, B, C, D) = \Sigma(0,2,3,4,5,6,7,8,10,11,15)$
  - $z(A, B, C, D) = \Sigma(1,2,8,12,13)$
- Simplified functions:
  - $w = ABC' + A'B'CD'$
  - $x = A + BCD$
  - $y = A'B + CD + B'D'$
  - $z = ABC' + A'B'CD' + AC'D' + A'B'C'D$
    $= w + AC'D' + A'B'C'D$

Example of PAL Design (2/3)

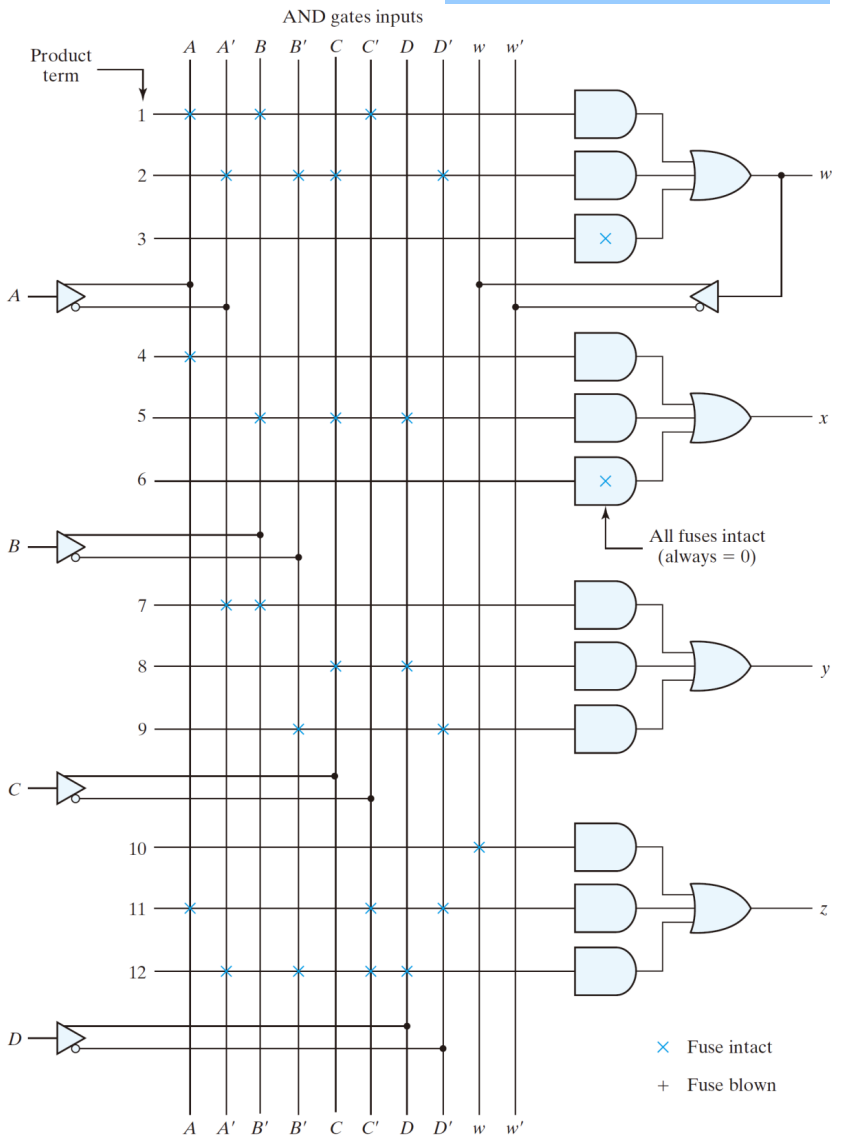
- Again, a programming table is used for PAL fuse map for logic implementation

PAL Programming Table

Product Term	AND Inputs 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	—	

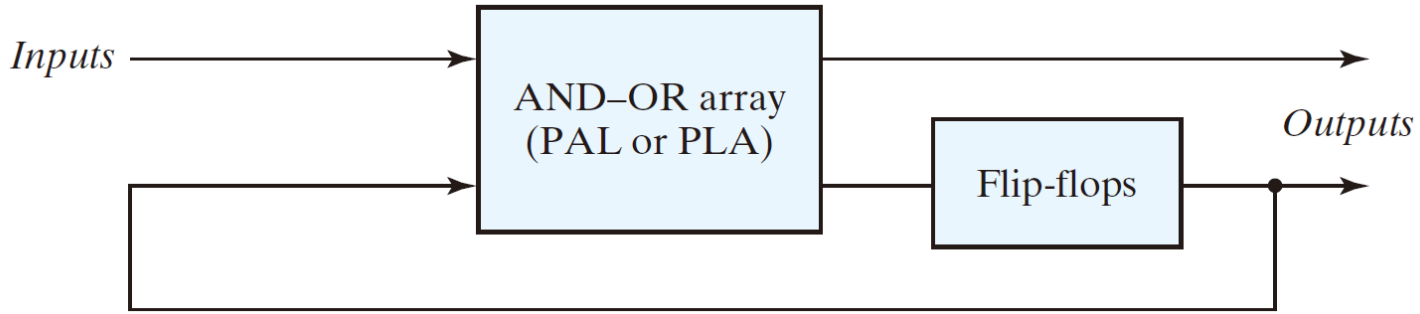
Example of PAL Design (3/3)

- The final implementation of the functions:
  - $w = ABC' + A'B'CD'$
  - $x = A + BCD$
  - $y = A'B + CD + B'D'$
  - $z = w + AC'D' + A'B'C'D$



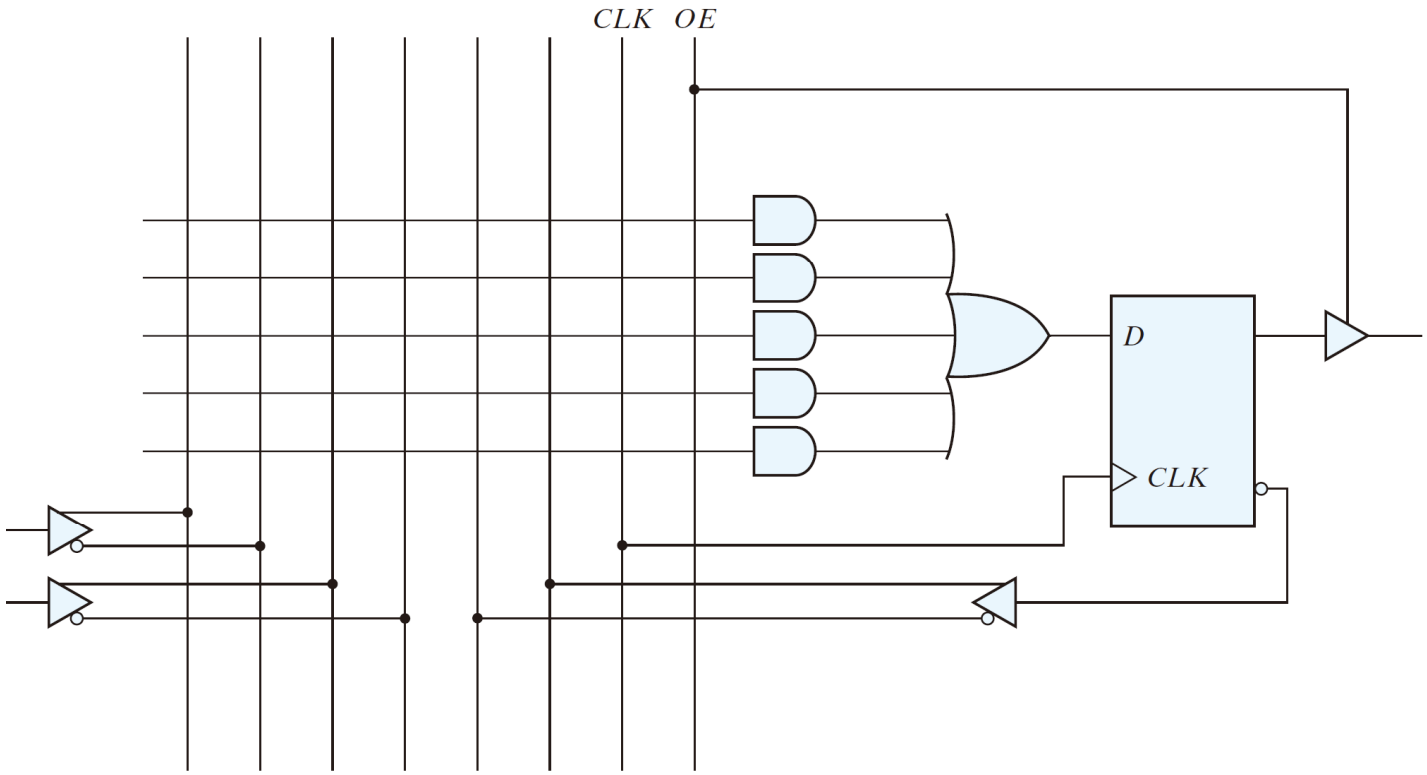
Sequential Programmable Devices

- Sequential programmable logic device is composed of gate level logics and flip-flops
- There are several types of sequential programmable devices
  - Simple programmable logic device (SPLD)
  - Complex programmable logic device (CPLD)
  - Field programmable gate array (FPGA)



Macrocell of SPLD

- A typical SPLD contains 8-10 macrocells, each macrocell has the following architecture:

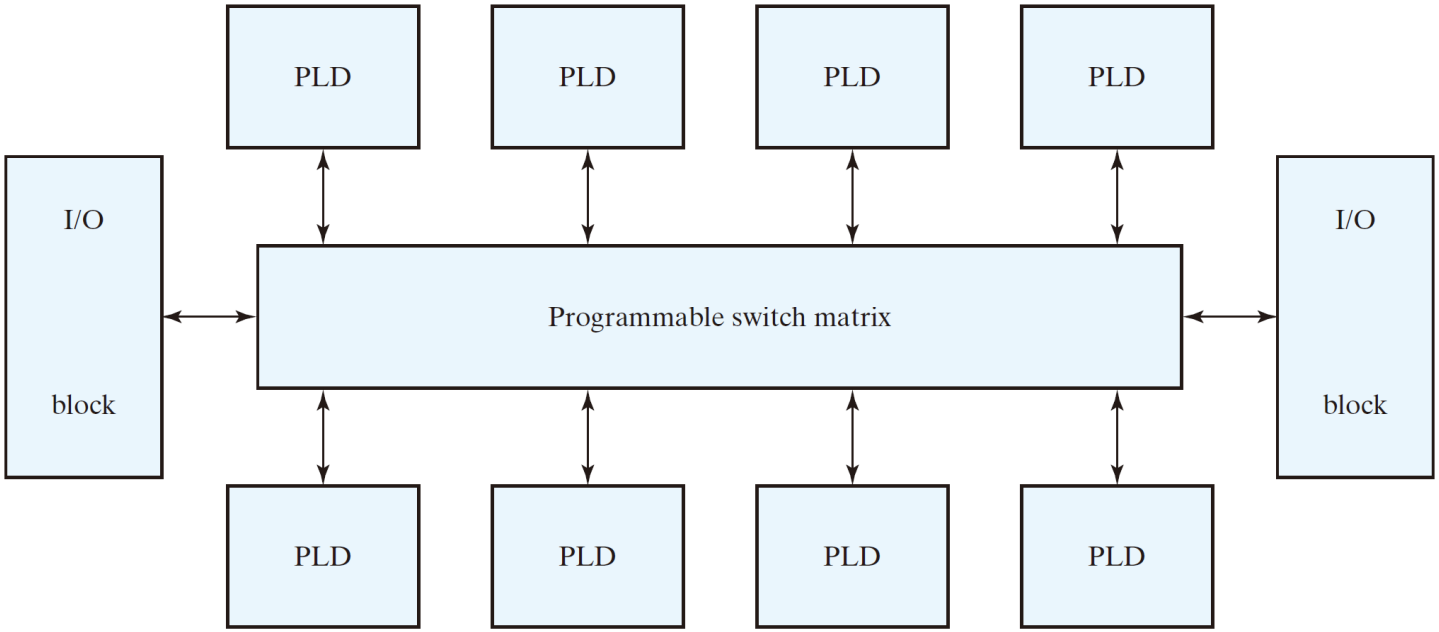


SPLD Programmable Components

- The following features of a SPLD can be programmed:
  - AND array
  - Use or bypass the flip-flop
  - Select clock edge polarity
  - Preset or clear for the register
  - Complement or straight output
  - Programmable input/output pins

Complex PLD

- A CPLD is a device that
  - Puts a lot of PLDS on a chip
  - Add wires between them whose connections can be programmed
  - Use fuse/EEPROM technology

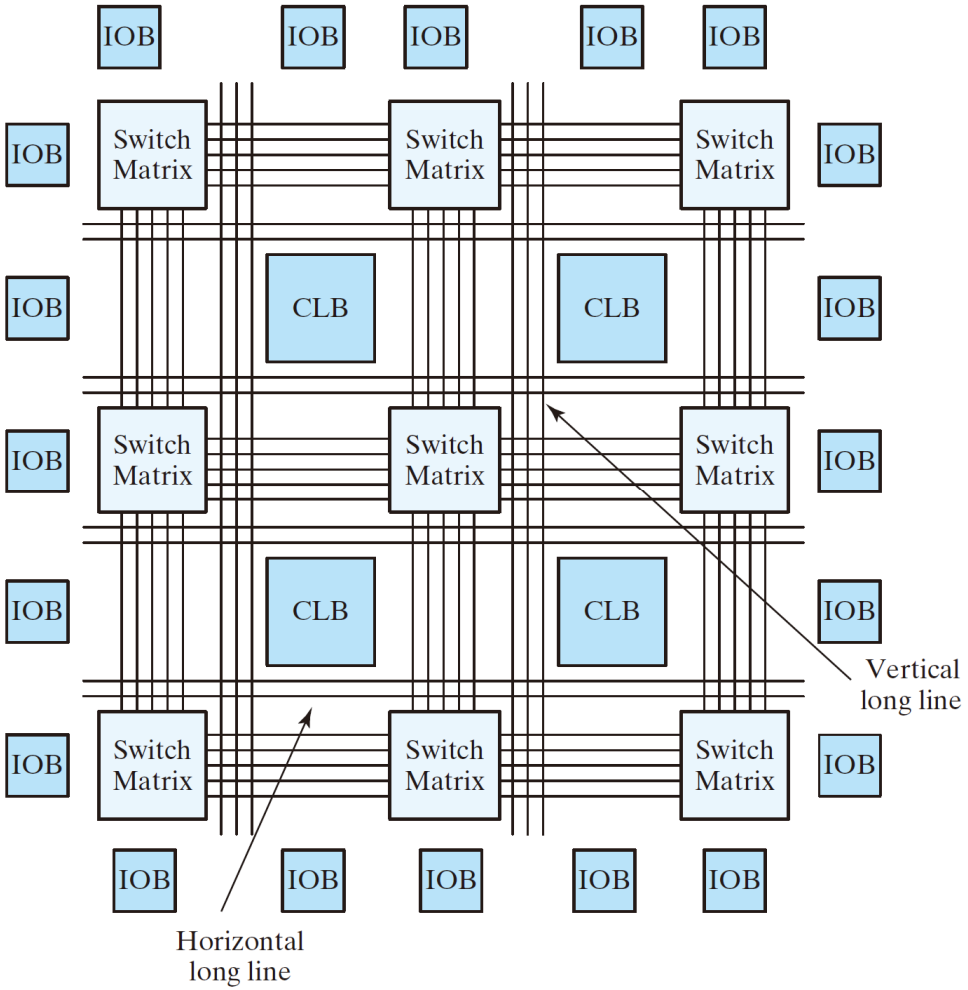


Field-Programmable Gate Array

- An FPGA emulate gate array technology by lookup tables (LUTs)
- FPGA can be programmed as easy as programming a flash memory electronically
- PALs and PLAs typically can implement digital circuits with complexity of 10 ~ 100 gates
- FPGAs can implement circuits with complexity anywhere from 100 to several millions of gates

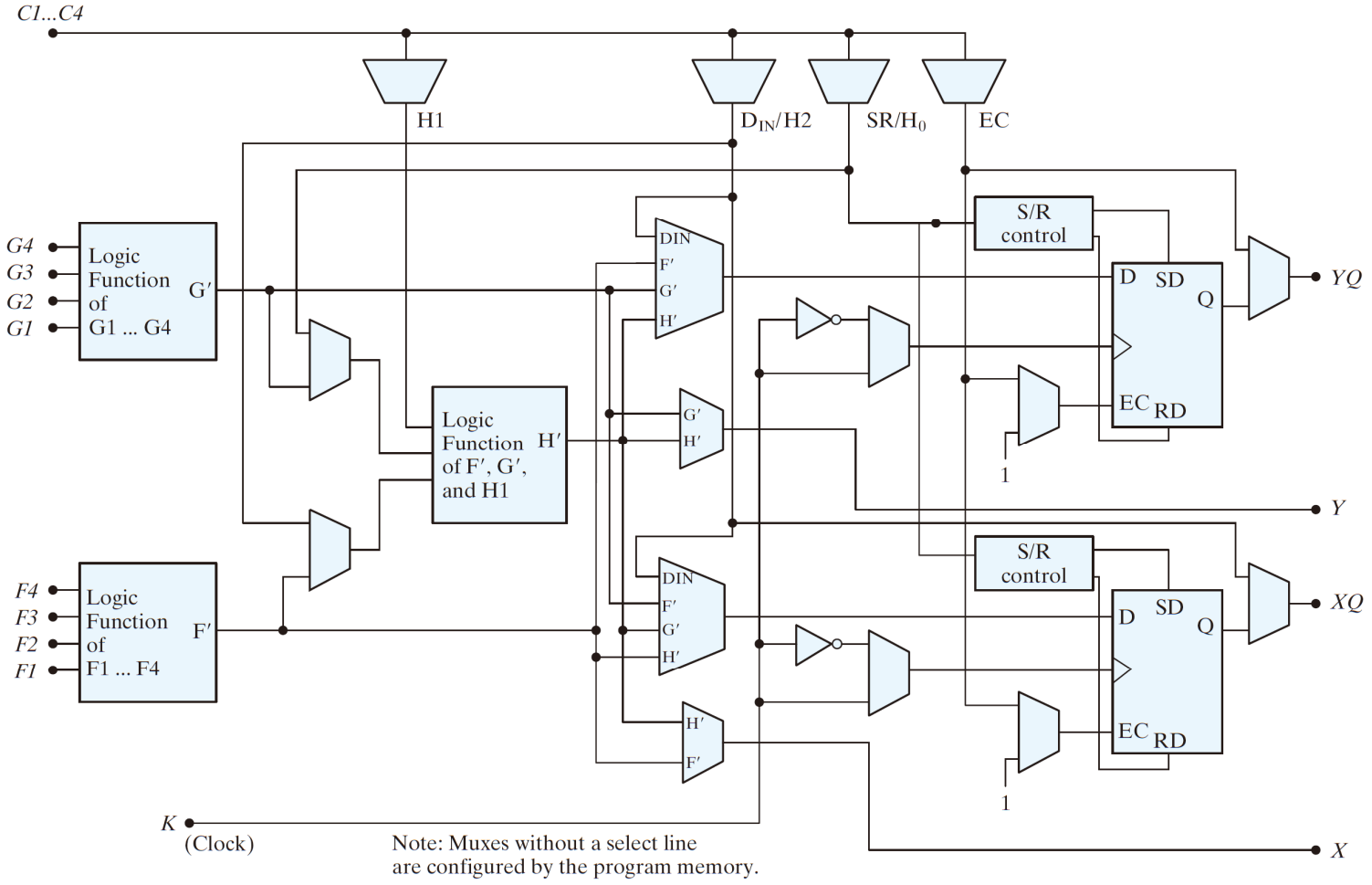
Basic Xilinx Architecture

- Xilinx Spartan Architecture
  - A CLB contains several logic slices
  - A logic slice contains several logic cells
  - A logic cell contains a lookup tables (LUTs) a flip-flop, and a few gates



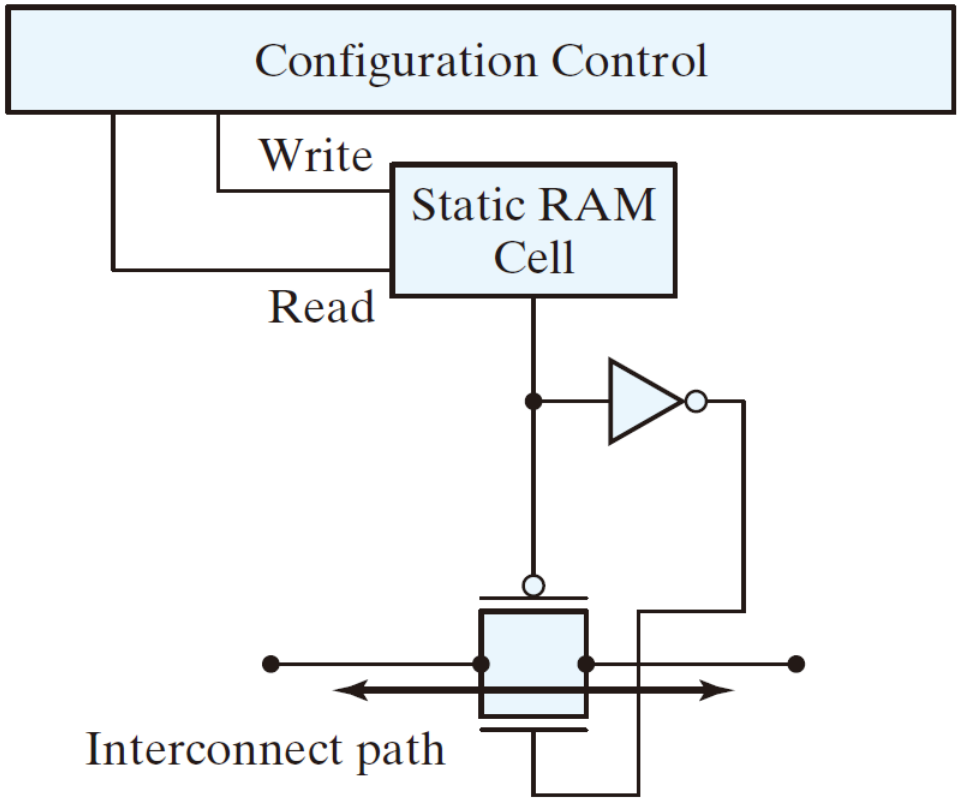
Xilinx FPGA Slice

- A slice is a fine-grain logic unit in Xilinx FPGAs



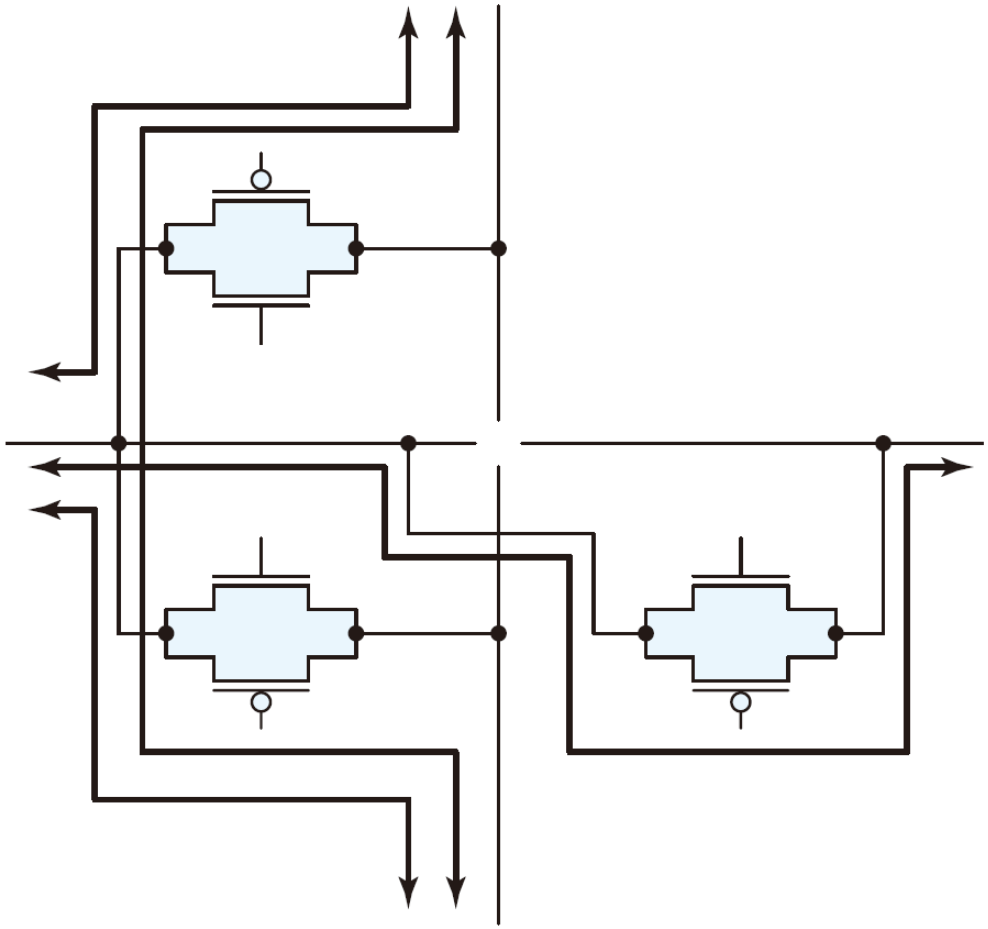
Interconnect Resources

- RAM cell controlling a PIP transmission gate:



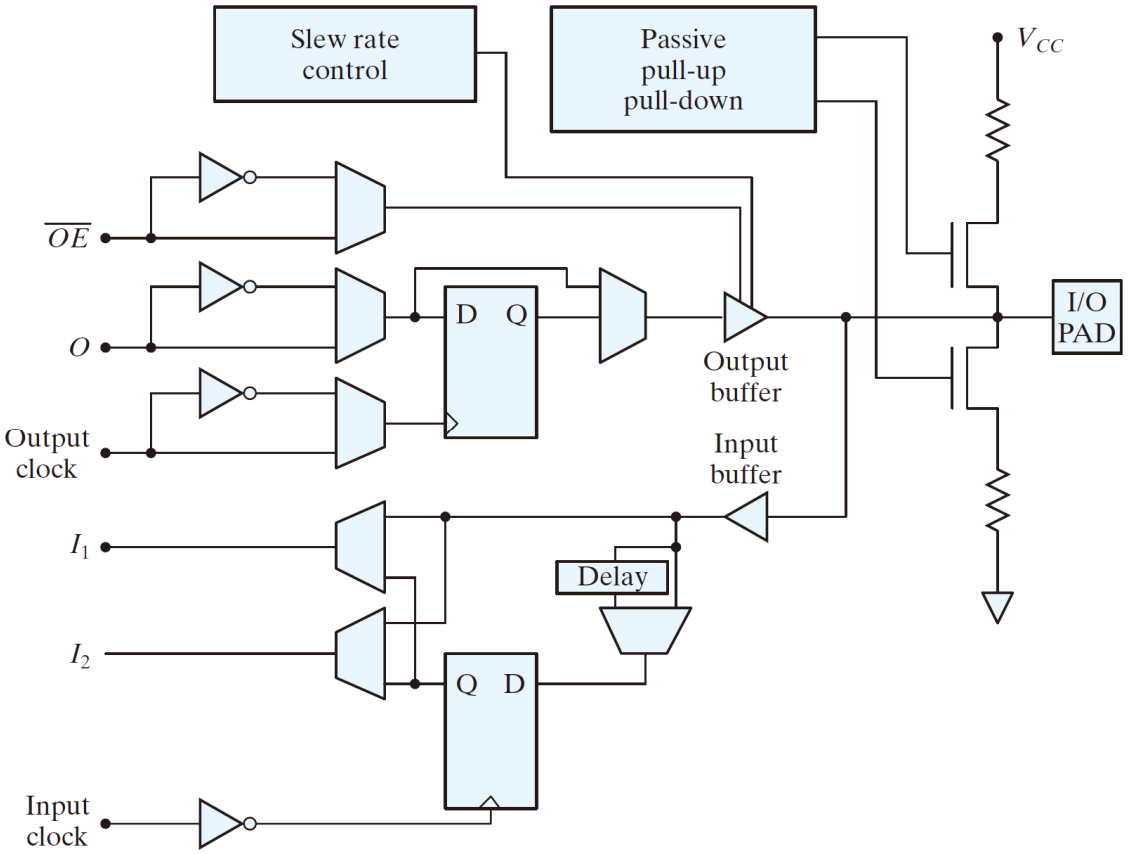
Programmable Interconnect Point

- Circuit for a programmable interconnect point (PIP)



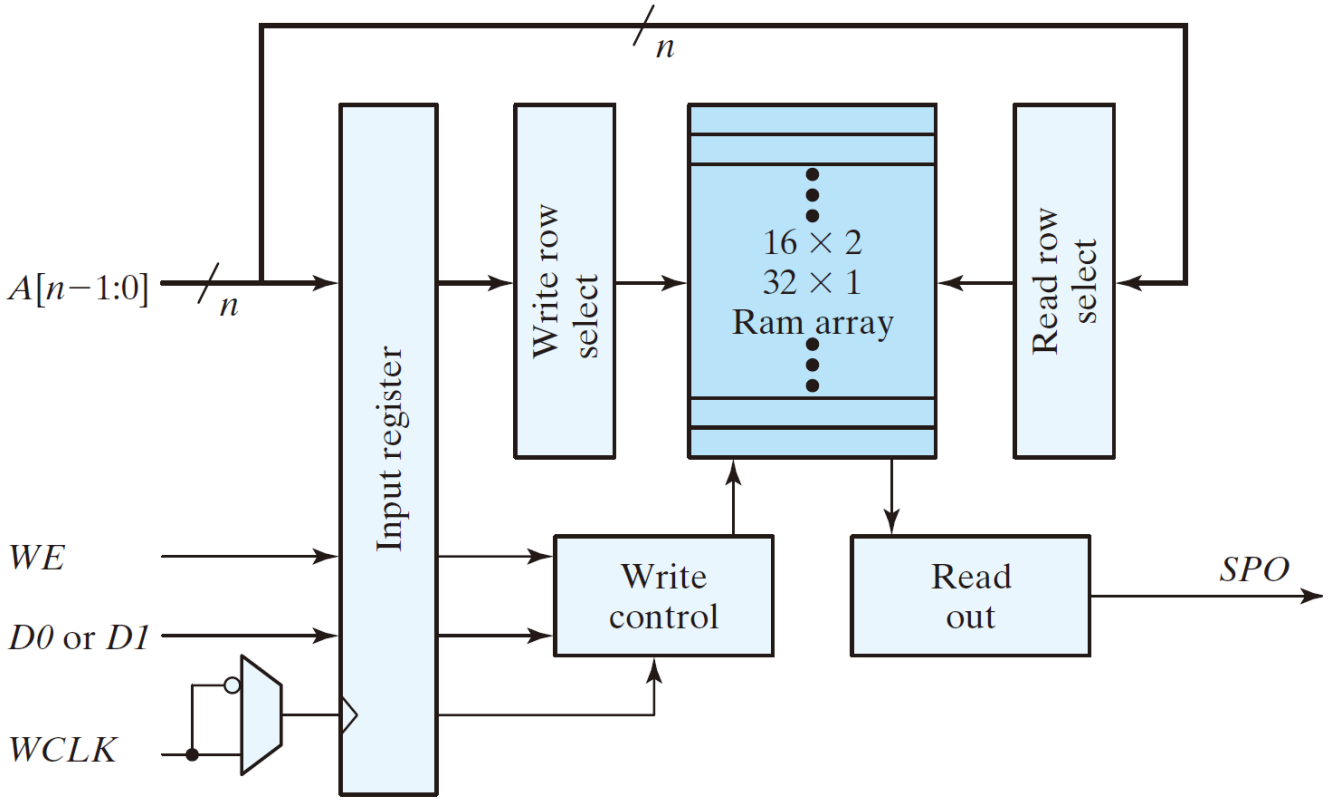
IO Block

- An FPGA IO block (IOB) is similar to the IO pad in an custom IC



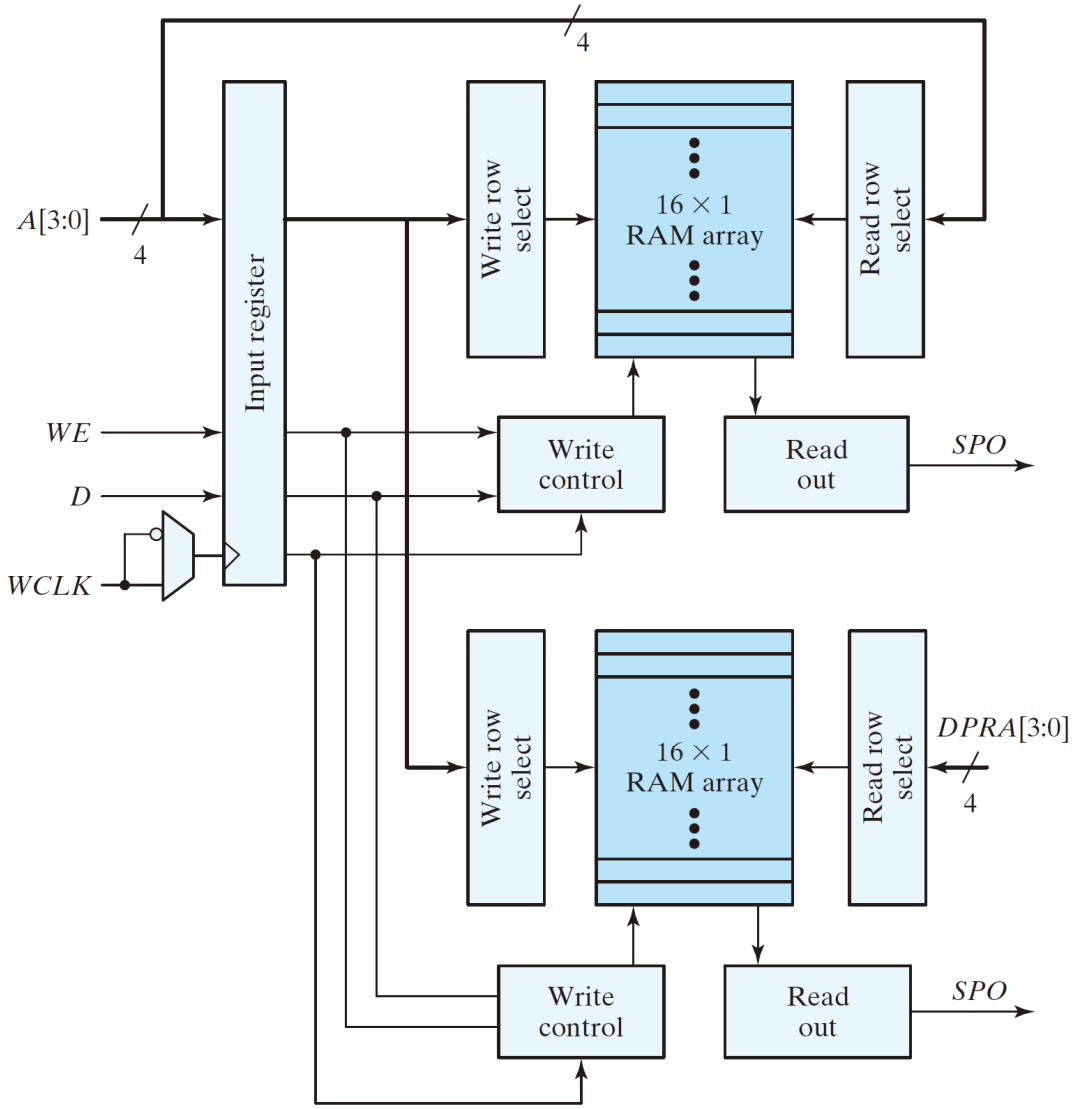
Distributed RAM Cells

- Distributed RAM cell formed from a lookup table:



Block RAMs

- Xilinx FPGAs contains larger memory devices called BRAMs
  - Dual-port
  - Word-length configurable



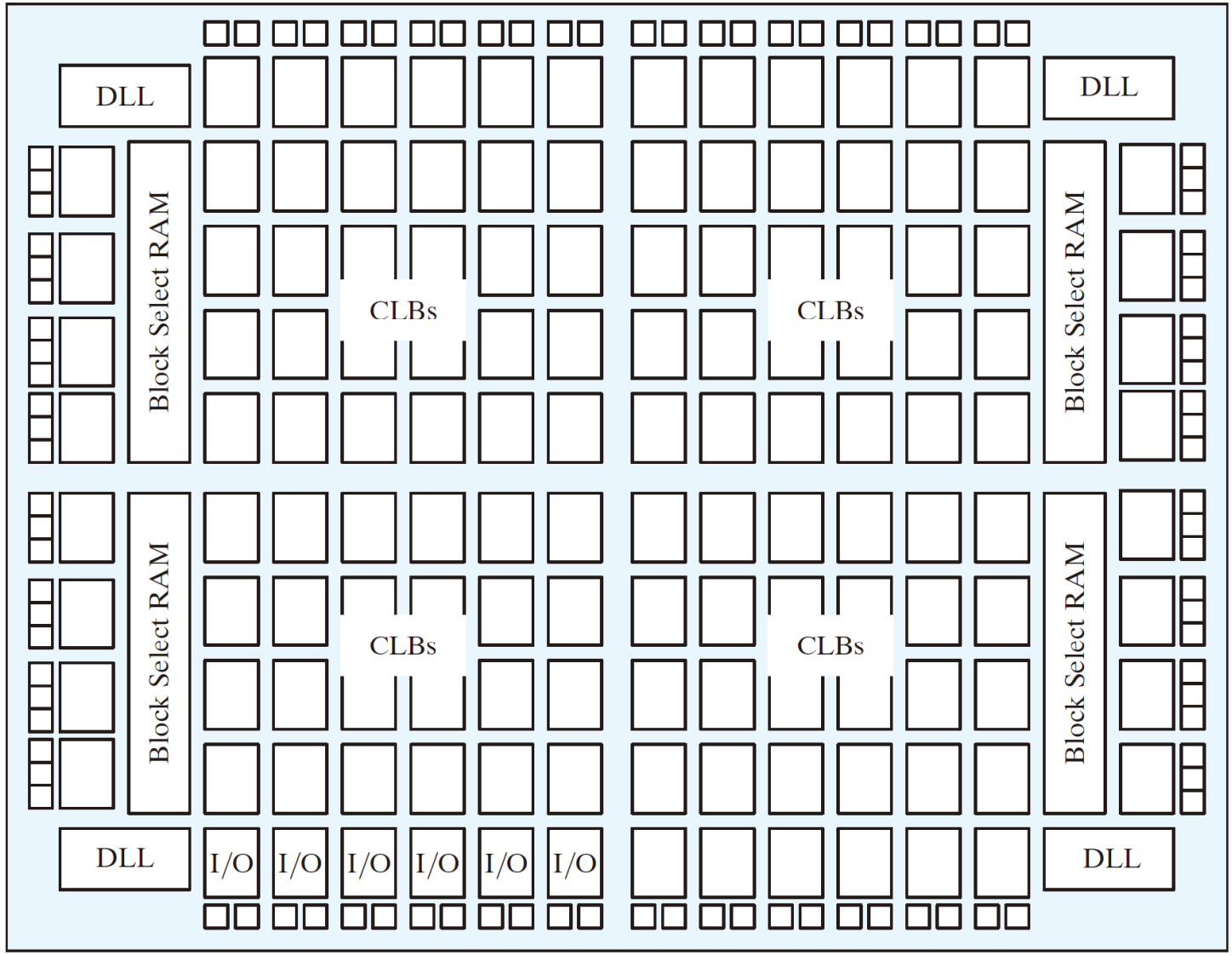
Xilinx Spartan II FPGAs

- Attribute of the Xilinx Spartan II FPGAs

Spartan II FPGAs	XC2S15	XC2S30	XC2S50	XC2S100	XC2S150	XC2S200
System Gates[1]	6K–15K	13K–30K	23K–50K	37K–100K	52K–150K	71K–200K
Logic Cells[2]	432	972	1,728	2,700	3,888	5,292
Block RAM Bits	16,384	24,576	32,768	40,960	49,152	57,344
Max Avail I/O	86	132	176	196	260	284

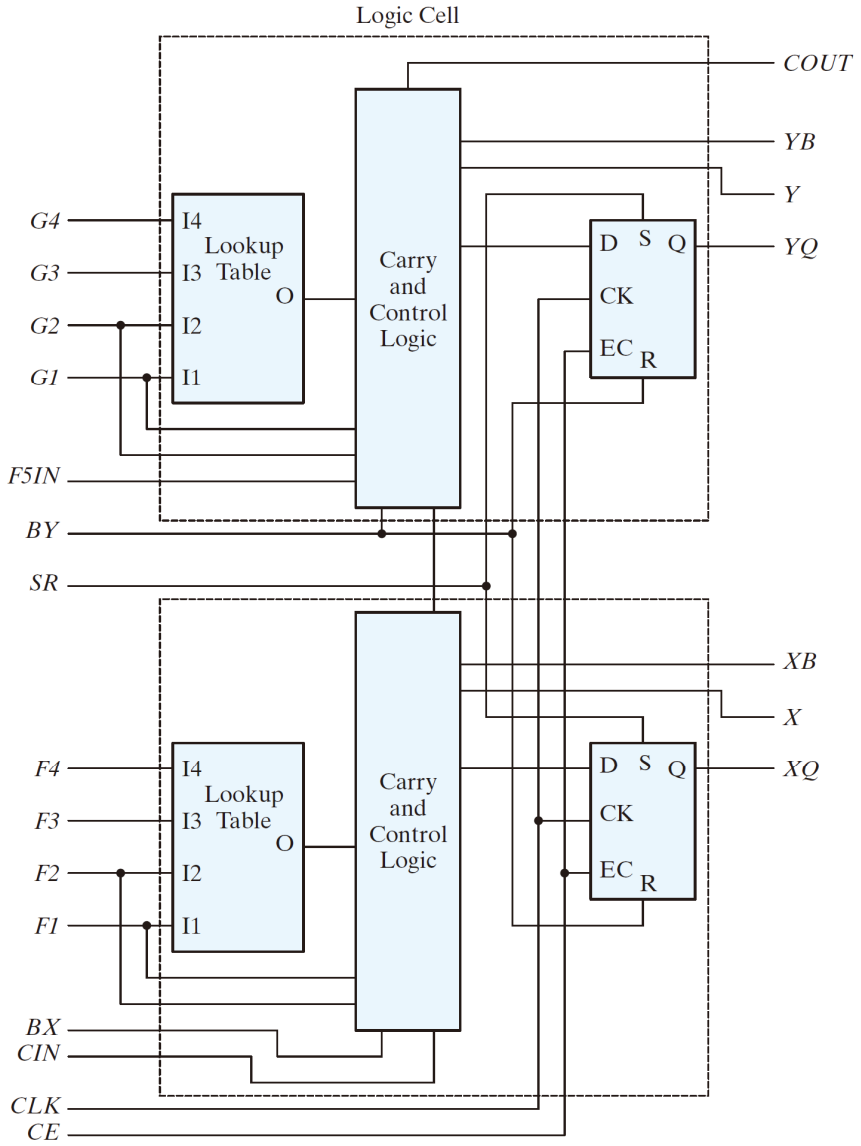
[1] 20–30% of CLBs as RAM.
[2] 1 Logic cell = four-input lookup table + flip-flop.

Xilinx Spartan II FPGAs

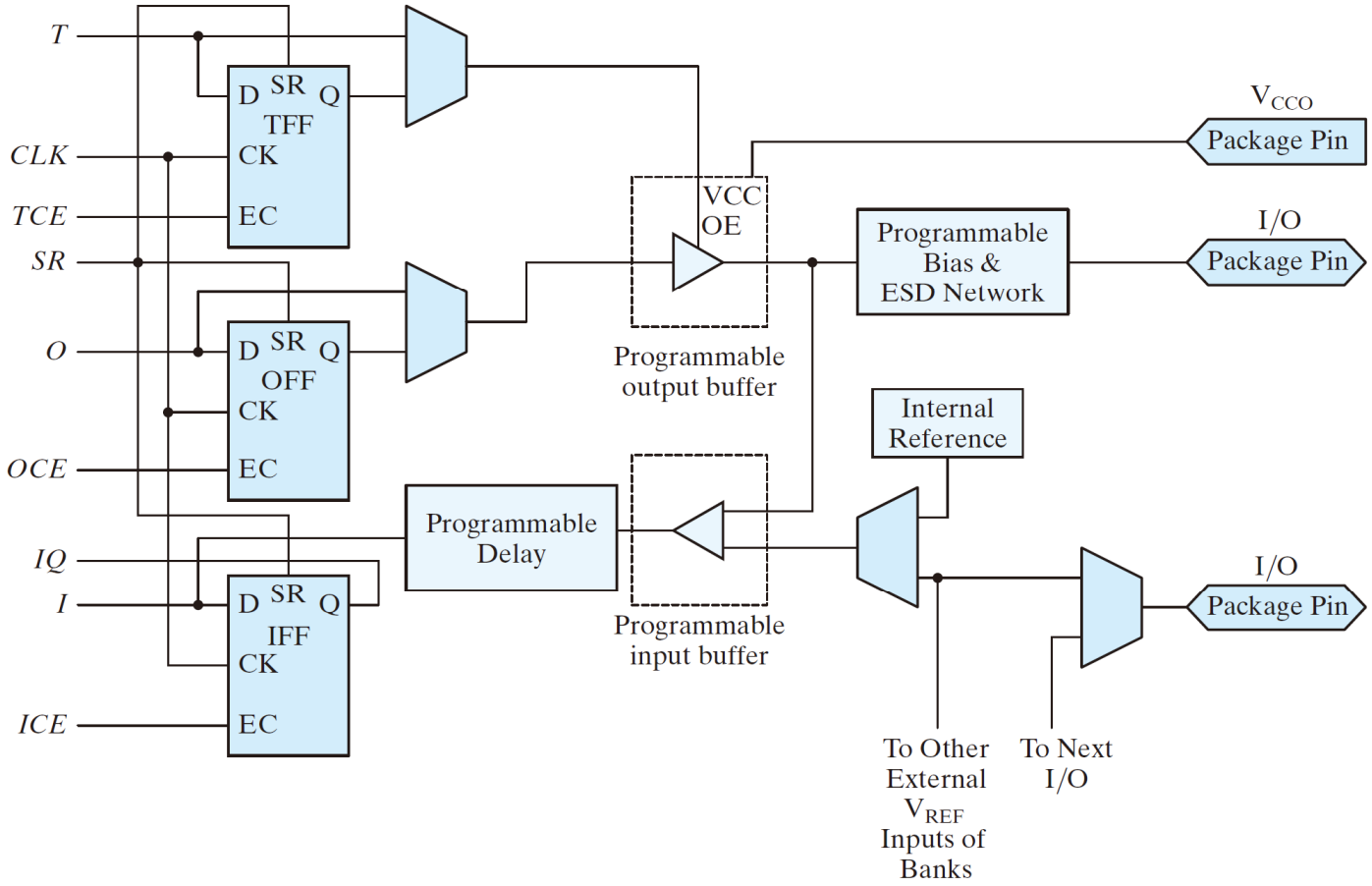


Spartan II CLB Slice

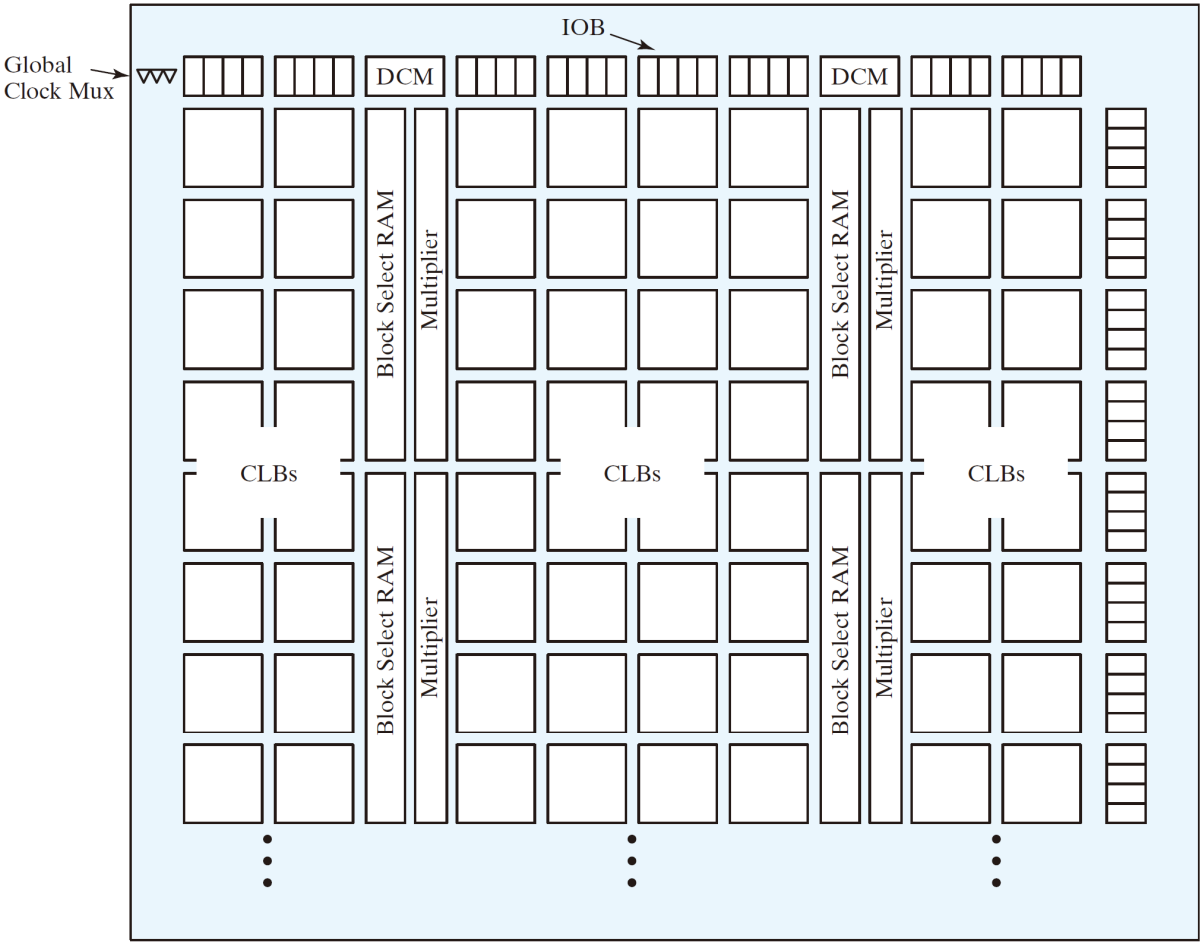
- Each slice has two logic cells



Spartan II IOB



Xilinx Virtex FPGAs



DCM: Clock Manager

Xilinx Virtex IOBs

- Virtex IOBs support more IO standards at physical layer