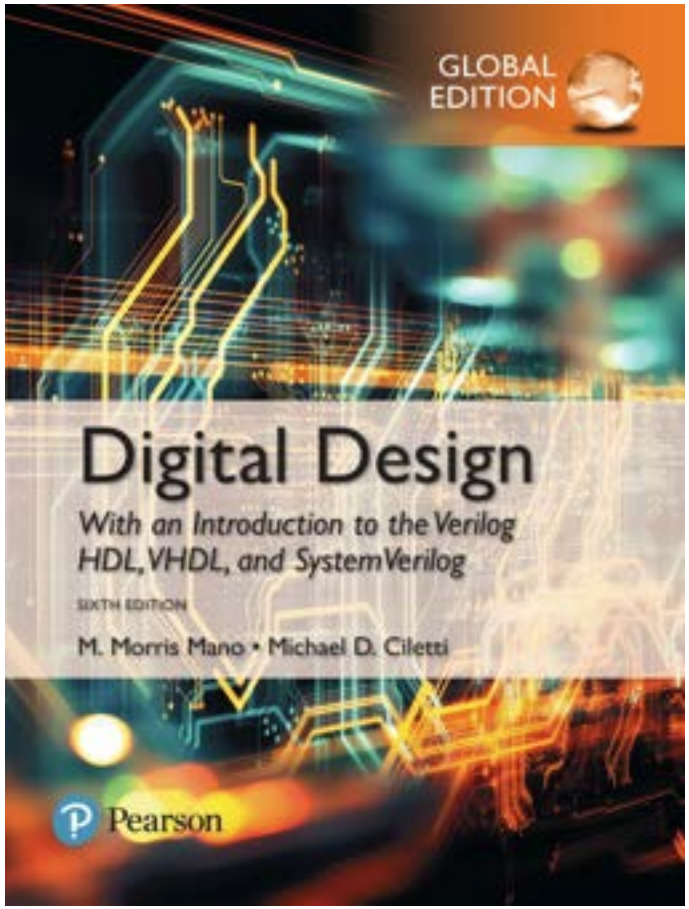
Digital Design

With an Introduction to the Verilog HDL, VHDL, and SystemVerilog

6th Edition, Global Edition



Chapter 07

Memory and Programmable Logic

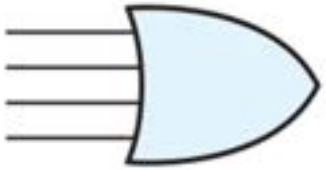
Pearson

7.1 개요

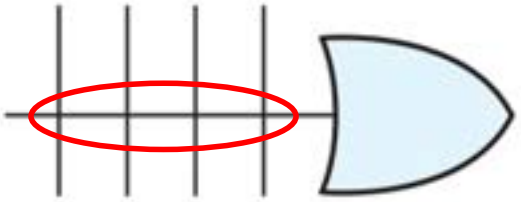
- 메모리 유닛: 2진 정보를 저장하고 처리에 필요한 정보를 꺼내 오는 데 사용하는 장
  - 데이터 처리하는 동안 정보는 메모리로부터 프로세싱 유닛 안에 있는 선택된 레지스터로 이동
  - 프로세싱 유닛에서 얻어진 계산의 결과는 다시 메모리로 이동하여 저장
  - 입력 장치에서 받아들인 2진 정보는 메모리에 저장되고, 출력 장치로 보내질 정보는 메모리에서 가져옴
- 메모리의 종류
  - RAM(Random Access Memory) : 쓰기와 읽기 동작 가능
  - ROM(Read-Only Memory) : 읽기 동작만 가능

- ROM
  - 일종의 프로그래머블 논리 장치(PLD, programmable logic device)
  - 프로그래밍(programming): 2진 정보를 하드웨어에 저장하는 작업
- PLD
  - 내부의 논리 게이트들이 퓨즈와 비슷한 동작을 하는 전기적인 패스로 연결된 집적회로. 원래 모든 퓨즈가 연결된 상태이며, 프로그래밍을 통해 원하는 논리적인 기능을 수행하도록 하드웨어를 설정하고 나머지 필요없는 퓨즈를 제거함
  - 종류
    - 프로그래머블 논리 어레이(PLA, programmable logic array)
    - 프로그래머블 어레이 논리(PAL, programmable array logic)
    - 필드 프로그래매블 게이트 어레이 (FPGA, field programmable gate array)
    - ROM

Figure 7.1
Conventional and array logic diagrams for OR gate.



(a) Conventional symbol



(b) Array logic symbol

7.2 RAM(Random Access Memory)

- 임의의 장소에 데이터를 저장하거나 꺼내 오는데 필요한 시간이 언제나 같기 때문에 ramdom access memory 라고 함
- 메모리 유닛은 word라는 여러 개 bit의 그룹(보통 8의 배수)으로 구성된 2진 정보를 저장. 1과 0으로 이루어지며, 숫자, 명령어, 기호 등의 정보를 나타냄. 8개 bit의 모임을 byte라고 부름. 메모리 유닛의 용량은 저장할 수 있는 전체 byte수임 (32비트 워드 = 4바이트)
- 메모리와 주변 장치들과의 통신은 데이터 입출력 라인, 어드레스 선택 라인, 제어라인을 통해 이루어짐

Figure 7.2
Block diagram of a memory unit.

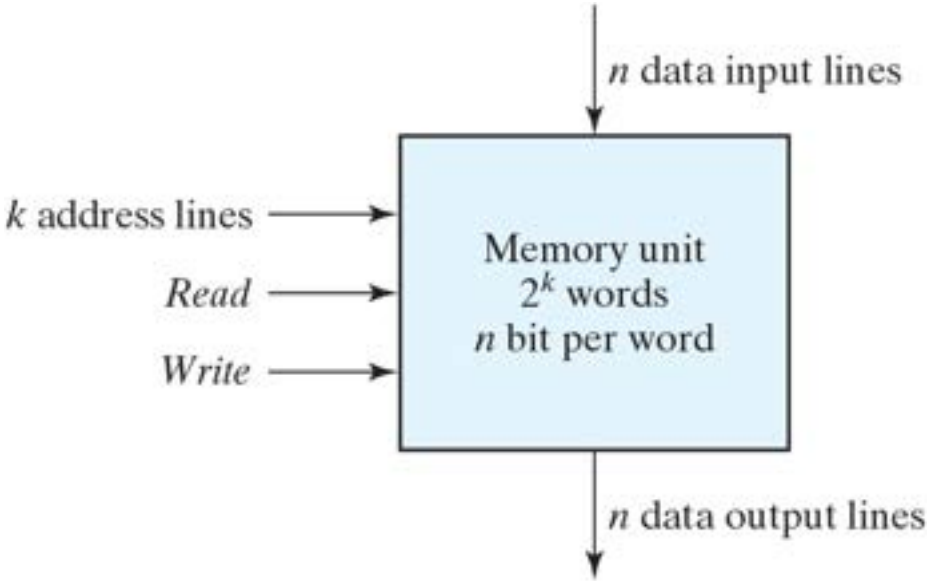


Figure 7.3
Contents of a 1024 × 16 memory.

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

- n개의 데이터 입력 라인은 메모리에 저장될 정보를 제공하고, n개의 데이터 출력 라인은 메모리로부터 출려고디는 정보를 제공.
- k개의 어드레스 라인은 많은 워드 가운데 하나를 선택하는데 사용
- 2개의 제어 라인은 쓰기, 읽기 선택

- 어드레스(address)
  - 특정한 하나의 워드를 선택
  - k개의 어드레스 라인을 가진 경우에 0부터 2^k-1까지의 값을 가짐
  - 어드레스 라인에 k비트 어드레서를 입력함으로써 메모리 내부의 특정한 워드를 선택
- 메모리의 크기는 10비트 어드레스가 필요한 1024워드(1킬로)에서 32비트 어드레스가 필요한 2^{32}워드(4기가)
  - K = 2^{10}, M = 2^{20}, G = 2^{30}
  - 예를 들어, 워드의 크기가 16비트이고 1K개의 워드를 저장할 수 있는 메모리 유닛 → 1K는 1024 즉, 2^{10}이고 16비트는 2바이트이므로 메모리는 2048 즉 2K 바이트의 용량을 가짐 = 1K x 16
  - 예를 들어, 64K x 10 메모리는 16비트 어드레스(왜냐하면 64K는 2^{16}이므로)와 10비트로 구성된 워드를 가짐

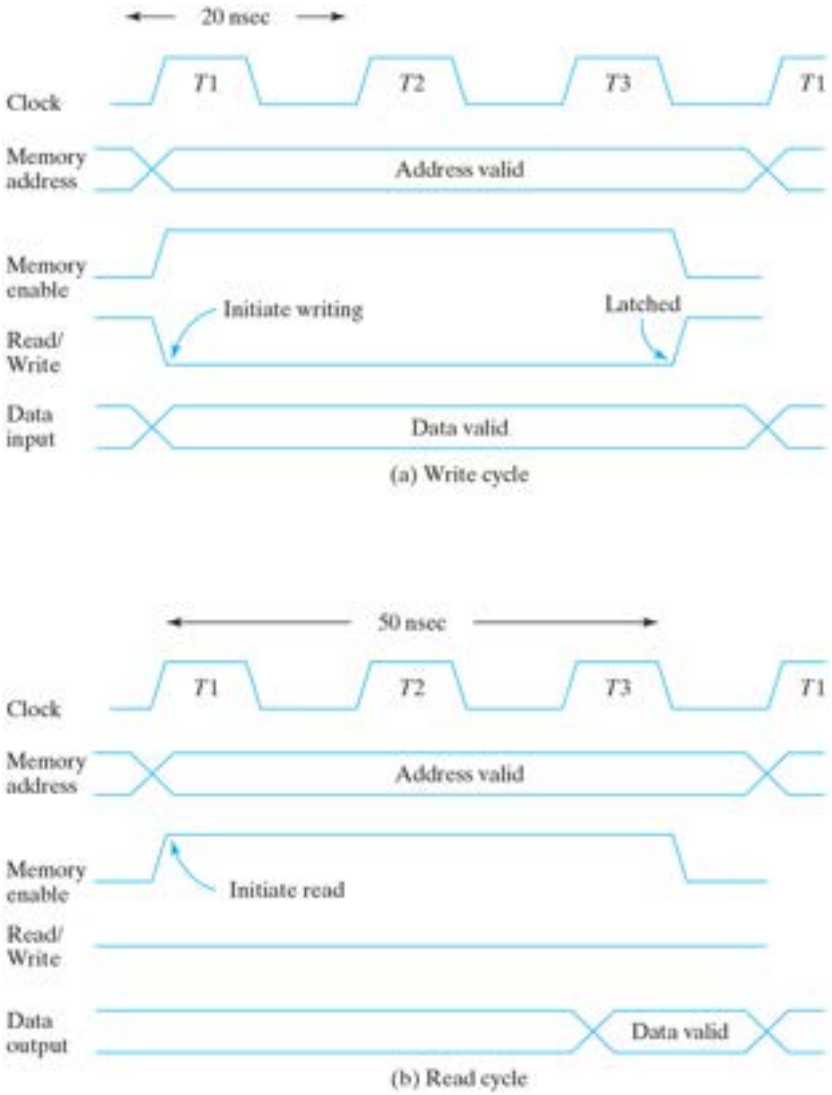
- 쓰기와 읽기 동작
  - 새로운 워드를 메모리에 전송하여 저장하는 절차
    1. 어드레스 라인에 원하는 워드의 2진 어드레스를 제공
    2. 데이터 입력 라인에 메모리에 저장하고자 하는 데이터 비트를 제공
    3. 쓰기 입력을 활성화
  - 저장된 워드를 메모리에서 꺼내는 절차
    1. 어드레스 라인에 원하는 워드의 2진 어드레스를 제공
    2. 읽기 입력을 활성화
  - 상용 메모리 소자에서 사용하는 약간 다른 방법
    - 읽기/쓰기 입력을 분리하는 제어 입력 대신 enable과 동작선택(읽기/쓰기)의 제어 입력을 사용

표 7.1
메모리 칩의 제어 입력

메모리 인에이블	읽기/쓰기	메모리동작
0	X	동작 없음
1	0	선택된 워드에 쓰기
1	1	선택된 워드로부터 읽기

- 타이밍 파형
  - 메모리 유닛의 동작은 CPU와 같은 외부 장치에 의해 제어됨. CPU는 자신의 클럭에 동기화되지만, 메모리는 내부 클럭을 가지고 있지 않음. 대신에 메모리의 읽기, 쓰기 동작은 제어 입력에 의해 동작 시간이 결정됨.
  - 액세스 타임(access time): 워드를 선택하고 저장된 정보를 읽는 데 필요한 시간
  - 사이클 타임(cycle time): 쓰기 동작을 완료하는 데 필요한 시간
  - CPU는 내부 클럭에 따라 실행하는 자신의 동작과 메모리의 쓰기/읽기 동작이 동기화되도록 메모리에 제어 신호를 보내야 하므로 메모리의 액세스 타임이나 사이클 타임이 CPU 클럭 주기의 고정된 배수와 같아야 함
    - 예를 들어, 클럭 주파수가 50MHz, 즉 클럭 주기가 20ns로 동작하는 CPU가 액세스 타임과 사이클 타임이 50ns를 넘지 않는 메모리와 통신 한다고 생각해보면, 각 메모리의 동작에 적어도 2개 반, 혹은 3개의 클럭 사이클이 필요

Figure 7.4
Memory cycle timing waveforms.



- 쓰기 사이클에는 T1, T2, T3 사이클이 필요함. T1의 시작점에 어드레스와 입력 데이터를 제공. 이후 어드레스 라인이 안정화된 다음에 인에이블과 읽기/쓰기 신호를 활성화하고 이를 50ns 유지. 어드레스와 데이터 신호는 제어 신호가 비활성화된 후에도 짧은 시간 동안 유지. T3가 종료될 때 쓰기 동작 완료.
- 읽기 사이클에서는 CPU가 어드레스를 제공하고, 인에이블과 읽기/쓰기 신호를 고레벨로 유지. 이후 50ns 안에 메모리는 어드레스에 의해 선택된 워드를 출력 데이터 라인에 옮김. T3 클럭이 하강할 때 CPU는 내부 레지스터로 데이터 전송가능

Pearson

- 메모리의 종류
  - 순차 액세스 메모리: 정보가 즉시 액세스될 수 없고, 특정 시간이 지난 후 가능 (자기 디스크, 테이프)
  - 랜덤 액세스 메모리: 워드의 저장 위치에 상관 없이 액세스 타임이 동일

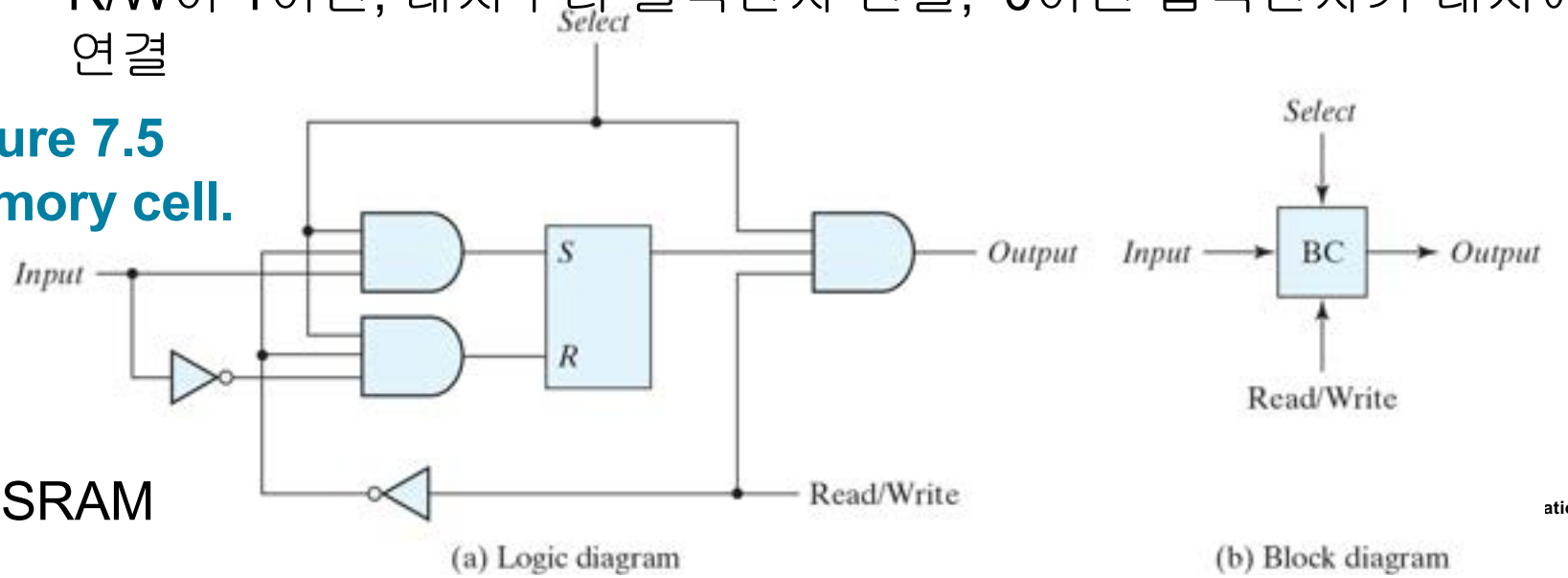
  - SRAM(static RAM): 2진 정보를 저장할 수 있는 래치들로 구성. 전기가 공급되는 동안 정보유지. 사용이 쉽고 읽고/쓰기 사이클이 짧음
  - DRAM(dynamic RAM): MOS 트랜지스터 형태로 제공되는 캐패시터에 전기 충전하는 방식. 시간이 지나면서 방전되므로 수밀리초마다 리프레싱해야 함. 전력소모를 줄일 수 있고, 단일 칩에 많은 용량을 저장가능

  - 휘발성(volatile): 전원이 공급되지 않으면 지워지는 메모리 (SRAM, DRAM)
  - 비휘발성(non-volatile): 자기 디스크, ROM, 플래시 메모리

7.3 메모리 디코딩

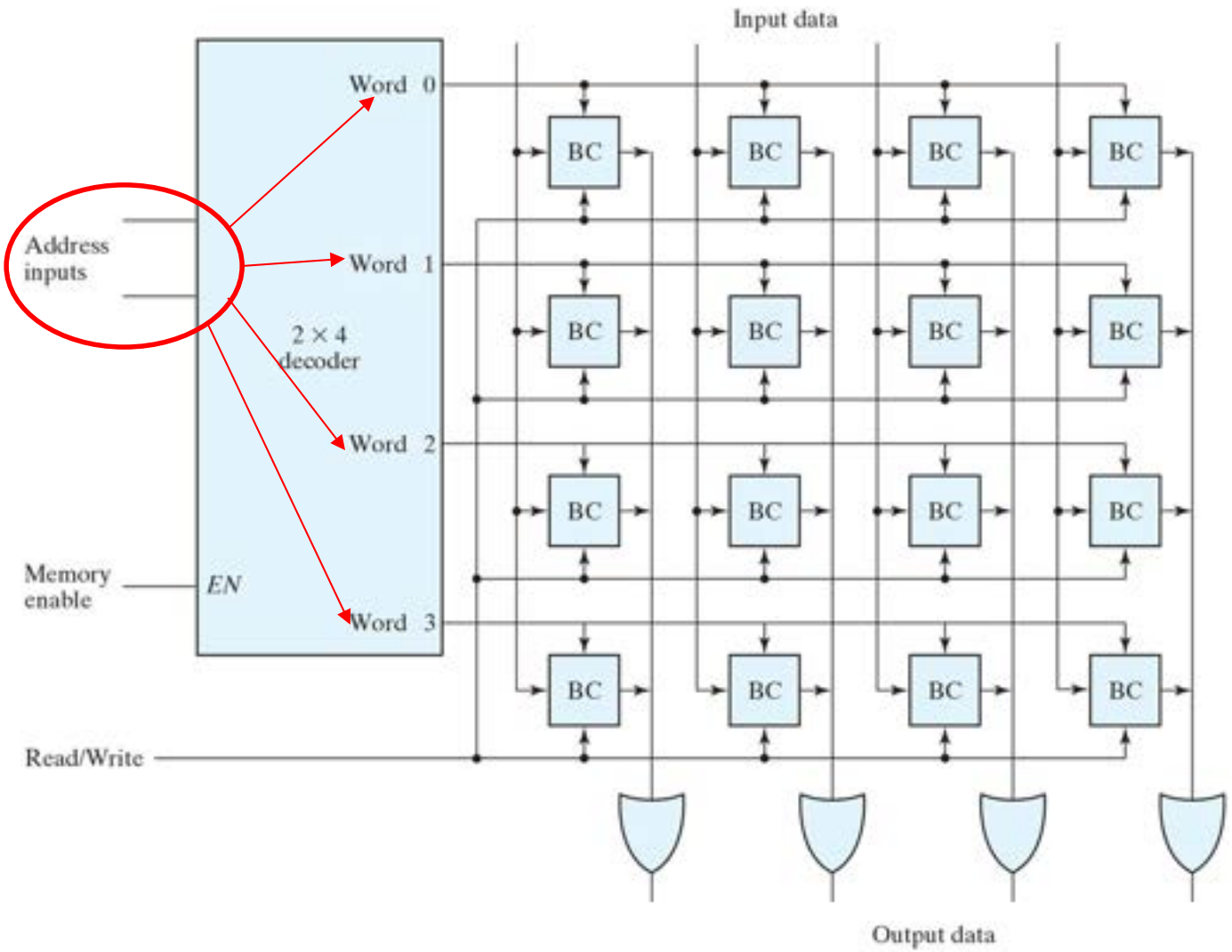
- 메모리 유닛에는 정보를 저장하는 소자뿐만 아니라 입력되는 어드레스에 의해 지정되는 메모리 워드를 선택할 수 있는 디코딩 회로가 필요함
- 여기서는 4비트로 구성된 4개의 워드를 갖는 16비트 메모리 유닛 설명
- 내부 구조
  - SR래치를 이용한 D래치 형태로 1개의 셀 구성
  - 선택(select), 읽기/쓰기(Read/Write)로 동작선택
  - R/W이 1이면, 래치부터 출력단자 연결, 0이면 입력단자가 래치에 연결

Figure 7.5
Memory cell.



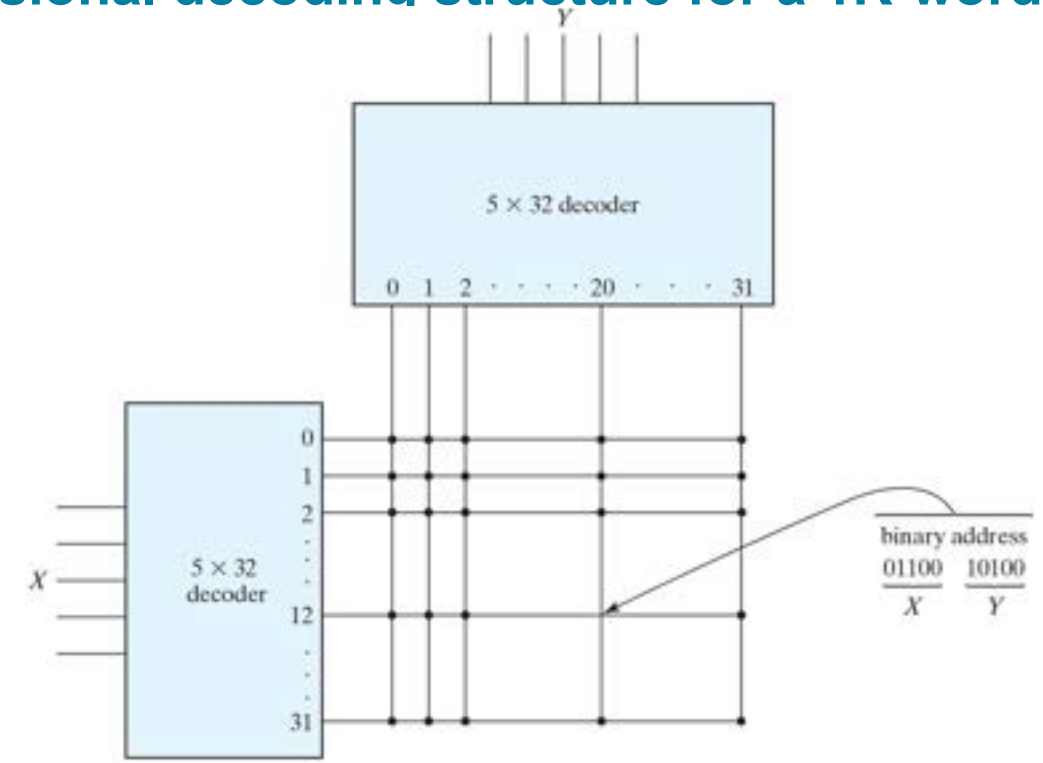
SRAM

Figure 7.6
Diagram of a 4 × 4 RAM.



- 동시 디코딩
  - k입력과 2k 출력을 가진 디코더에는 k개의 입력을 가진 2k개의 AND 게이트가 필요 → 2차원 선택 구조를 가진 2개의 디코더를 사용함으로써 필요한 게이트의 수와 각 게이트의 입력 수를 줄임
  - 즉, k입력 디코더 대신, 2개의 k/2 입력 디코더 사용
  - 1k 워드 메모리의 2차원 선택방법은 10x1024 디코더(10개의 입력을 가진 AND 게이트 1024개) 하나 대신 2개의 5x32 디코더(5개의 입력을 가진 AND 게이트 64개)를 사용

Figure 7.7
Two-dimensional decoding structure for a 1K-word memory.



- 어드레스 멀티플렉싱
  - 그림 7.5의 SRAM은 6개의 트랜지스터로 구성. DRAM은 하나의 MOS 트랜지스터와 커패시터로 구성. 따라서 DRAM은 SRAM보다 4배 정도 높은 밀도로 제작 가능하고, 비용은 3-4배 저렴, 저전력
  - DRAM은 용량이 크기 때문에 어드레스 디코딩을 2차원 배열로 나열하고 다차원 배열을 사용하기도 함
  - IC 패키지의 핀 수를 줄이기 위해 입력 어드레스 핀 집합을 어드레스 소자에 적용하여 어드레스 멀티플렉싱 사용함

Figure 7.8
Address multiplexing for a 64K DRAM.

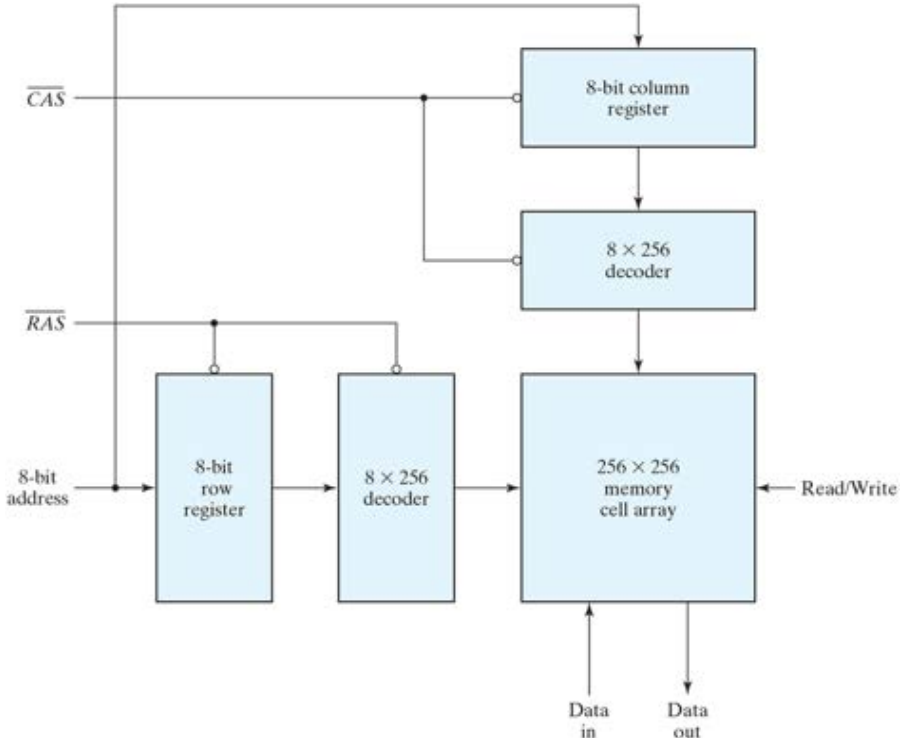
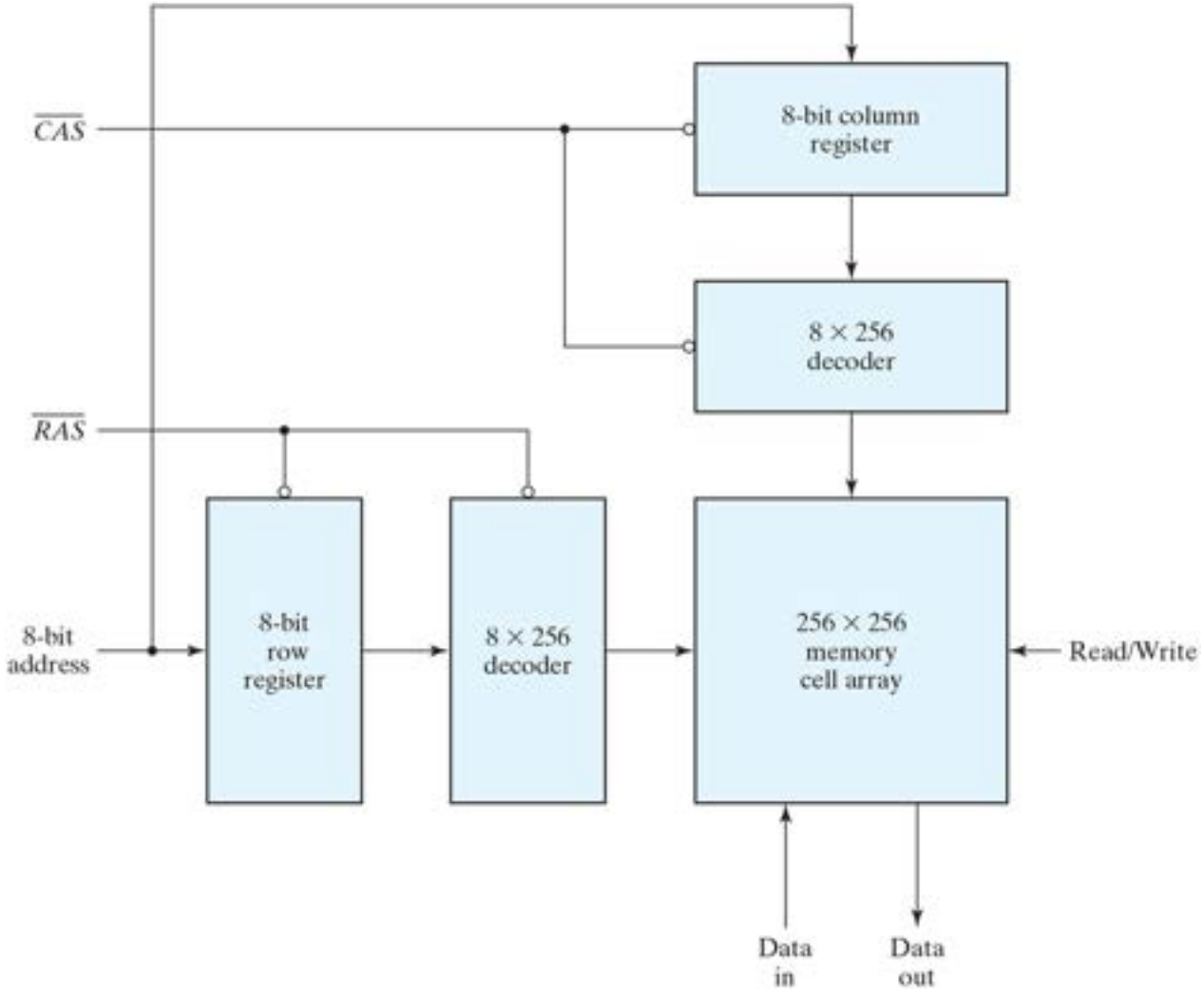


Figure 7.8
Address multiplexing for a 64K DRAM.



7.4 에러 검출 및 수정

- 메모리 유닛도 당연히 에러가 발생할 수 있음. 가장 일반적인 에러 검출 방법은 3.8절의 패리티 비트 사용임. 하지만 패리티 비트로는 에러 검출만 가능
- 에러 수정 코드는 여러 개의 패리티 검출 비트들을 생성시켜 데이터 워드와 함께 메모리 안제 저장. 각각의 검출 비트는 데이터 워드의 일부분에 대한 패리티임.
- 메모리로부터 워드가 읽혀졌을 때 관련된 패리티 비트들도 함께 읽혀져서 데이터로부터 생성된 검출 비트와 비교됨.
- 검출 비트들이 일치하면 에러가 발생하지 않은 것이고, 일치하지 않으면 어떤 비트에서 에러가 발생했는가를 판단하는 신드롬(syndrome)이라는 고유한 패턴을 만듬. 만약 단일 에러가 발생한 거라면 그 비트를 찾아서 보수 취하면 에러 수정됨

- 동시 디코딩
  - k입력과 2k 출력을 가진 디코더에는 k개의 입력을 가진 2k개의 AND 게이트가 필요 → 2차원 선택 구조를 가진 2개의 디코더를 사용함으로써 필요한 게이트의 수와 각 게이트의 입력 수를 줄임
  - 즉, k입력 디코더 대신, 2개의 k/2 입력 디코더 사용
  - 1k 워드 메모리의 2차원 선택방법은 10x1024 디코더(10개의 입력을 가진 AND 게이트 1024개) 하나 대신 2개의 5x32 디코더(5개의 입력을 가진 AND 게이트 64개)를 사용
- 해밍 코드 (by R.W. Hamming)
  - n비트의 데이터 워드에 k비트의 패리티 추가
  - 2의 제곱수인 위치에 패리티 비트를 추가하고 나머지 위치에 데이터 추가
  - 예를 들어, 8비트 데이터 워드 11000100과 4개의 패리티 비트 추가하면
    - 비트위치 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
    - 4개의 패리티 비트: P_1, P_2, P_4, P_8

- 각 패리티 비트는 다음과 같이 계산함
  - P_1 = XOR of bits (3, 5, 7, 9, 11) = 1⊕1 ⊕ 0 ⊕ 0 ⊕ 0 = 0
  - P_2 = XOR of bits (3, 6, 7, 19, 11) = 1⊕0 ⊕ 0 ⊕ 1 ⊕ 0 = 0
  - P_4 = XOR of bits (5, 6, 7, 12) = 1 ⊕ 0 ⊕ 0 ⊕ 0 = 1
  - P_8 = XOR of bits (9, 10, 11, 12) = 0 ⊕ 1 ⊕ 0 ⊕ 0 = 1
- 따라서 8비트의 데이터 워드와 4비트의 패리티를 합치면 12비트의 복합 워드(composite word)로 저장됨
  - 비트위치 1 2 3 4 5 6 7 8 9 10 11 12
  - 0 0 1 1 1 0 0 1 0 1 0 0
- 이러한 12비트 데이터가 메모리에서 읽혀질때, 에러검출을 위해 다시 검사함
  - C_1 = XOR of bits (1, 3, 5, 7, 9, 11)
  - C_2 = XOR of bits (2, 3, 6, 7, 19, 11)
  - C_4 = XOR of bits (4, 5, 6, 7, 12)
  - C_8 = XOR of bits (8, 9, 10, 11, 12)
  - '0' 검사 비트는 짝수 패리티, '1' 검사 비트는 홀수 패리티

- 검출 결과 C = $C_8C_4C_2C_1$ = 0000이면 에러가 발생하지 않은 것임. C가 0이 아니라면, 검출 비트로 생성된 4비트 2진수 C는 에러가 발생한 위치를 나타냄
- 예를 들어, 다음의 3 가지 경우
  - 비트위치 1 2 3 4 5 6 7 8 9 10 11 12
  - 0 0 1 1 1 0 0 1 0 1 0 0 → 오류 없음
  - 1 0 1 1 1 0 0 1 0 1 0 0 → 1번 비트 오류
  - 0 0 1 1 0 0 0 1 0 1 0 0 → 5번 비트 오류
- C를 구해보면,

	C_8	C_4	C_2	C_1
오류 없음	0	0	0	0
1번 비트 오류	0	0	0	1
5번 비트 오류	0	1	0	1

Table 7.2
Range of Data Bits for k Check Bits.

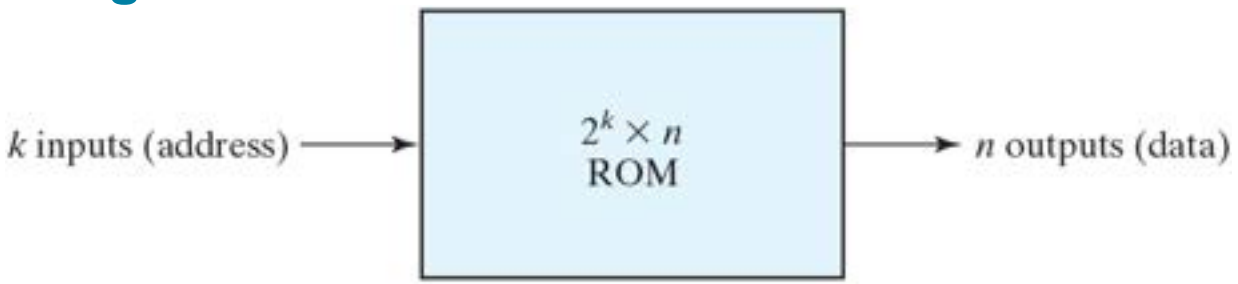
Number of Check Bits, *k*	Range of Data Bits, *n*
3	2–4
4	5–11
5	12–26
6	27–57
7	58–120

- 단일 에러 수정, 이중 에러 검출
  - 해밍 코드는 하나의 에러를 검출하고 수정 가능
  - 해밍 코드가 적용된 워드에 추가로 패리티 비트를 추가하면 이중 에러를 검출할 수 있음
  - 예를 들어, 앞 예제의 12비트 데이터(8비트 워드에 해밍 코드 4비트 추가) 001110010100에 13번째 비트인 P_{13} 을 추가 패리티 비트로 적용. 만약 짝수 패리티라면 0011100101001을 생성
  - 따라서 13비트 워드를 읽으면 검출 비트 평가하고, 동시에 패리티 P도 평가함.
    - C = 0, P = 0: 에러 없음
    - C != 0, P = 1: 수정할 수 있는 단일 에러 발생
    - C != 0, P = 0: 이중 에러 발생, 수정할 수 없음
    - C = 0, P = 1: 비트 P_{13} 에서 에러 발생

7.5 ROM (Read-Only Memory)

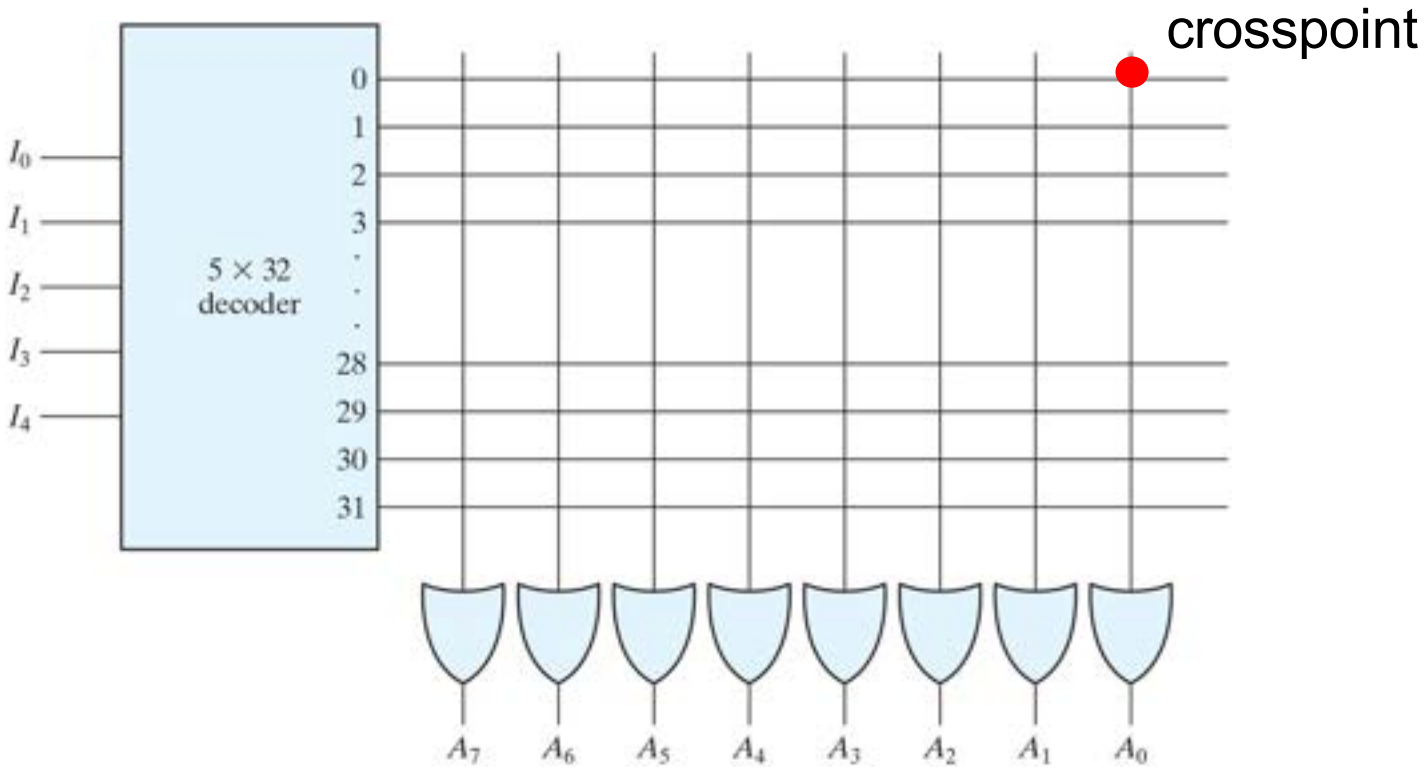
- 영구적인 2진 정보가 저장되는 메모리 장치

Figure 7.9
ROM block diagram.



- k개의 입력과 n개의 출력으로 구성된 ROM을 보여줌. 입력은 메모리의 어드레스, 출력은 어드레스에 의해 선택된 위치에 저장된 워드의 데이터 비트들

Figure 7.10
Internal logic of a 32 × 8 ROM.

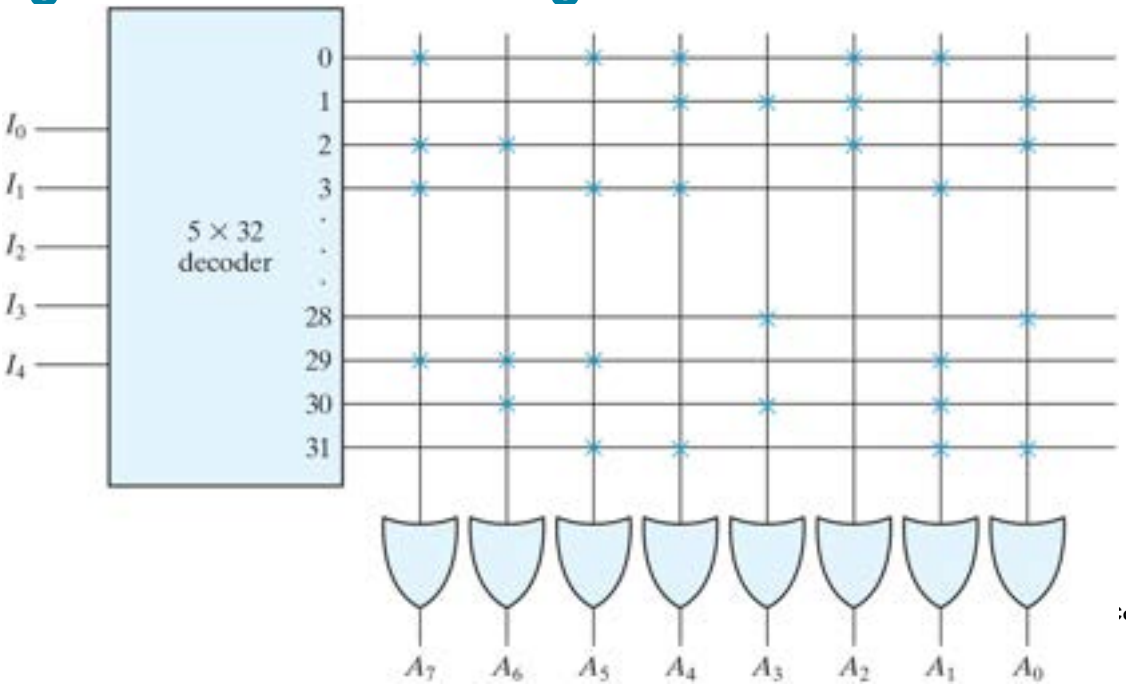


- 8비트 크기의 32워드로 구성. 5개 입력은 5 x 32 디코더를 통해 32개의 출력으로 디코딩됨. 디코더의 출력이 어드레스임. 디코더의 32개 출력은 8개의 OR게이트에 연결됨.
- 2^k x n ROM은 k x 2^k 내부 디코더와 n개의 OR 게이트로 구성

Table 7.3
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

Figure 7.11
Programming the ROM according to Table 7.3.



- 조합회로 구현
  - ROM은 고정된 패턴의 워드를 저장하는 메모리 유닛으로 보거나 조합회로를 구현한 유닛으로 생각할 수 있음
  - 4.9절의 디코더와 비교

Figure 4.18
Three-to-eight-line decoder.

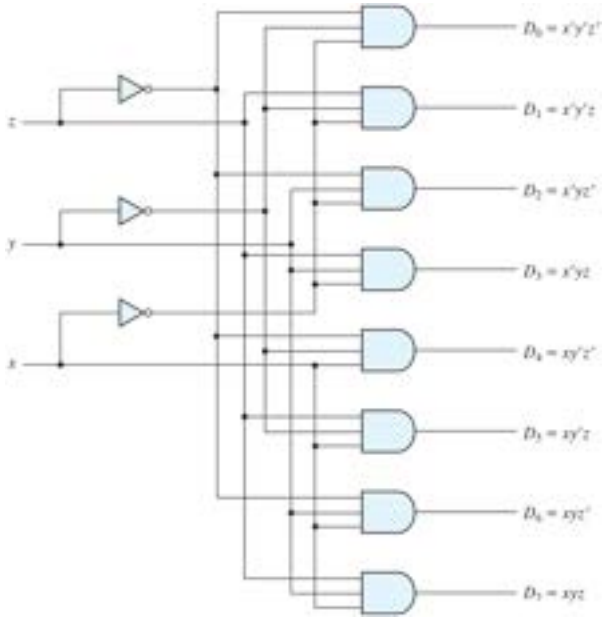


Table 4.6
Truth Table of a Three-to-Eight-Line Decoder.

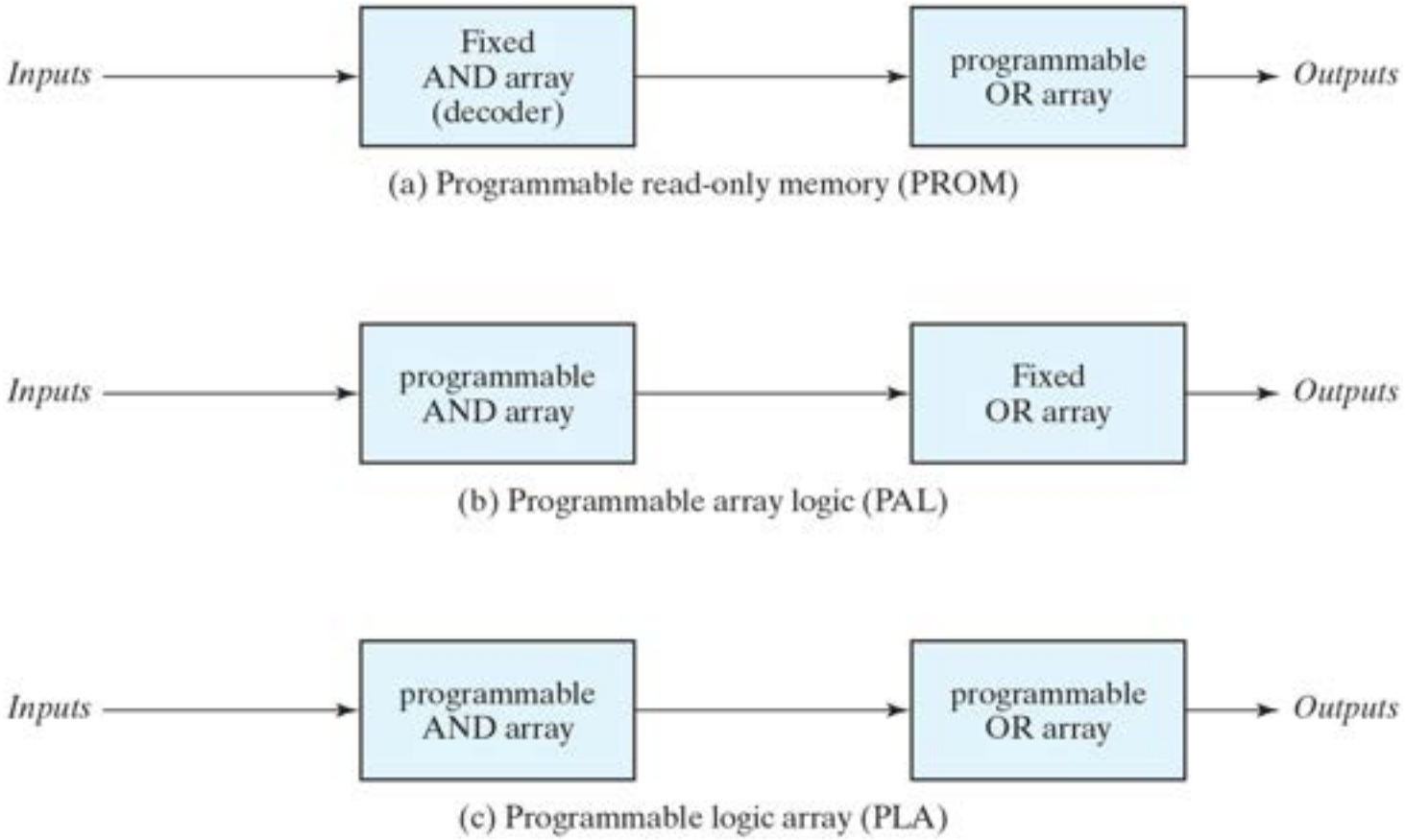
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

- ROM의 종류
  - 마스크 프로그래밍으로 반도체 회사에서 프로그래밍된 ROM
  - PROM(programmable ROM): 퓨즈를 이용해서 1번 프로그래밍 가능
  - EPROM(erasable PROM): 자외선으로 초기화 가능한 PROM
  - EEPROM(electrically EPROM): 전기적으로 초기화 가능한 PROM
  - Flash memory: On-board 상태에서 사용자가 내용을 Byte 단위로 자유로이 Read 할 수는 있지만, Write는 Page 또는 Sector 라고 불리는 Block 단위로만 수행 할 수 있는 변형된 EEPROM이다. 블록의 크기는 Memory 소자나 Maker에 따라 다르지만 대체로 64Byte, 128 Byte, 246Byte 등에서 부터 128KB까지도 사용한다. 이렇게 Flash Memory는 EEPROM과 매우 유사하지만, Byte단위로 Write하는 것이 불가능하므로 Page mode write 기능만 가지는 EEPROM이라고 생각하면 되며, 따라서 역시 SRAM처럼 실시간 Data memory로 사용하는 것은 불가능하다. 그러나, Flash memory는 EEPROM보다 Memory Cell 구조가 간단하여 훨씬 대용량의 메모리 소자를 만드는데 적합하며, 1개의 Block 전체를 Write하는데 수 ms 정도가 걸리므로 대용량 Data를 Write할 때는 EEPROM보다 훨씬 빠르다는 장점을 가진다.

    출처: https://treeroad.tistory.com/entry/Flash-Memory와-EEPROM-차이점

- PLD (프로그래머블 논리 장치)

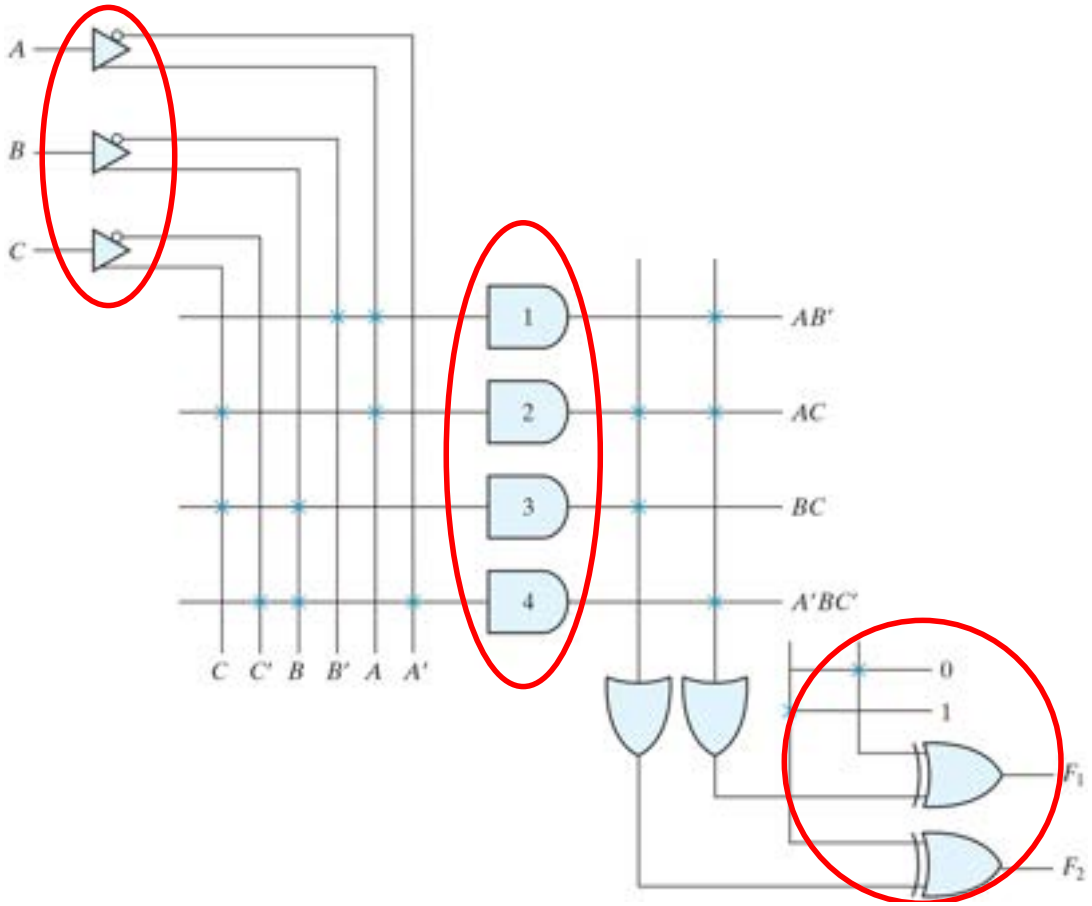
Figure 7.13
Basic configuration of three PLDs.



7.6 PLA (Programmable Logic Array)

- 모든 변수에 대해 완전히 디코딩을 지원하지 않고 최소항을 모두 생성하지 않는다는 점을 제외하면 개념적으로 PROM과 유사
- 디코더는 프로그래밍이 가능한 AND 게이트 어레이로 대체

Figure 7.14
PLA with three inputs, four product terms, and two outputs.



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

Table 7.5
PLA Programming Table.

	Product Term	Inputs			Outputs (T)	(C)
		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	–

-- 는 연결하지 않는다는 뜻

- 위의 표에서 첫번째 부분은 곱의 항을 숫자로 나열, 두번째 부분은 입력과 AND 게이트 사이에 요구되는 패스를 표현, 세번째 부분은 AND 게이트와 OR 게이트 사이의 패스를 표현하고 XOR 게이트를 프로그래밍해서 T(true)나 C(complement)를 지정할 수 있음

7.7 PAL (Programmable Array Logic)

- 고정된 OR 어레이와 프로그래머블 AND 어레이를 가지고 있는 프로그래머블 논리 소자임. AND 게이트만 프로그래밍 가능하기 때문에 PAL은 PLA보다 프로그래밍하기 쉽지만 유연성이 부족함
- 예를 들어,
  - 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) = $\Sigma(1, 2, 8, 12, 13)$
- 위의 예를 최소수의 항으로 간소화하면
  - 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

Figure 7.16
PAL with four inputs, four outputs, and a three-wide AND–OR structure.

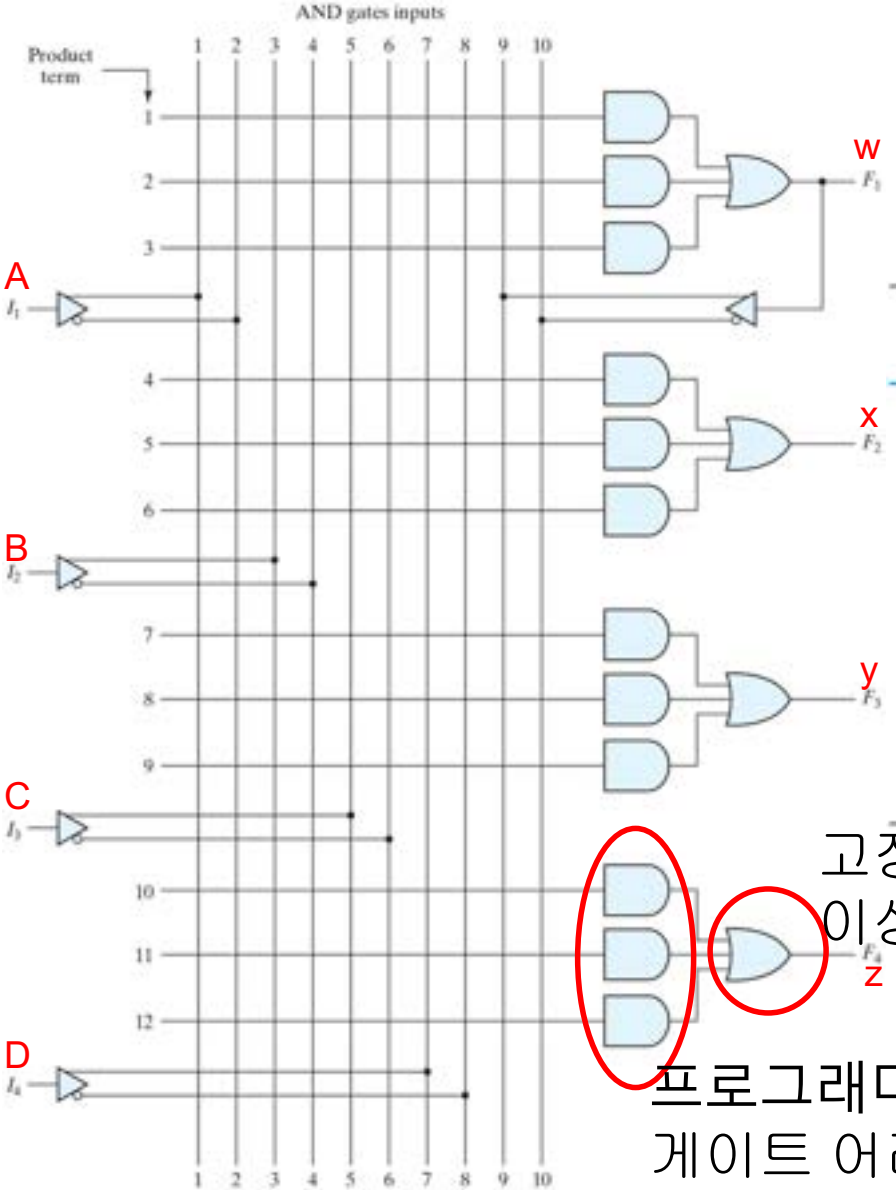


Table 7.6
PAL Programming Table.

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	—	

고정된 OR 게이트: 곱의 항이 3개 이상이면 불가능 → w 피드백 사용

프로그래머블 AND 게이트 어레이

Figure 7.17
Fuse map for PAL as specified in Table 7.6.

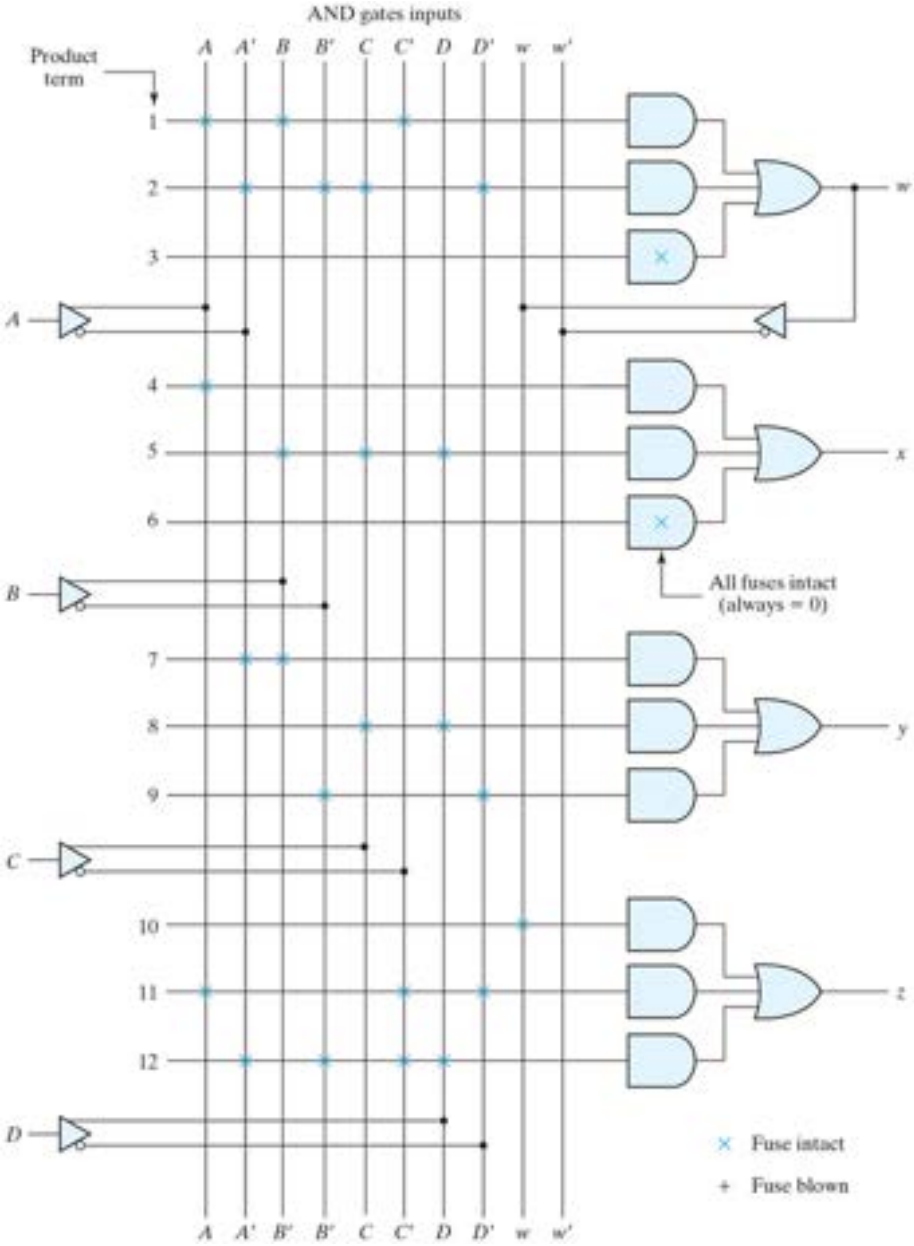


Table 7.6
PAL Programming Table.

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	—	

7.8 순차적 프로그래머블 논리 소자

- 조합 PLD는 게이트들로만 구성되기 때문에 외부 플립플롭을 포함시켜야 함. 상용으로 다양한 종류의 순차적 프로그래머블 소자가 판매중이며 제조사마다 규격 차이가 있음
  - 프로그래머블 순차 논리 소자 (SPLD, sequential/simple PLD): AND-OR 어레이와 D 플립플롭으로 구성되는 매크로셀 8-10개 사용
  - 프로그래머블 복합 논리 소자 (CPLD, complex PLD): 여러 PLD의 조합. 8-16개의 매크로셀을 포함하는 PLD를 프로그래머블 스위치 매트릭스를 통해 연결
  - 필드 프로그래머블 게이트 어레이 (FPGA, field programmable gate array): 수백만개의 논리 블록의 어레이로 구성

Figure 7.18
Sequential programmable logic device.

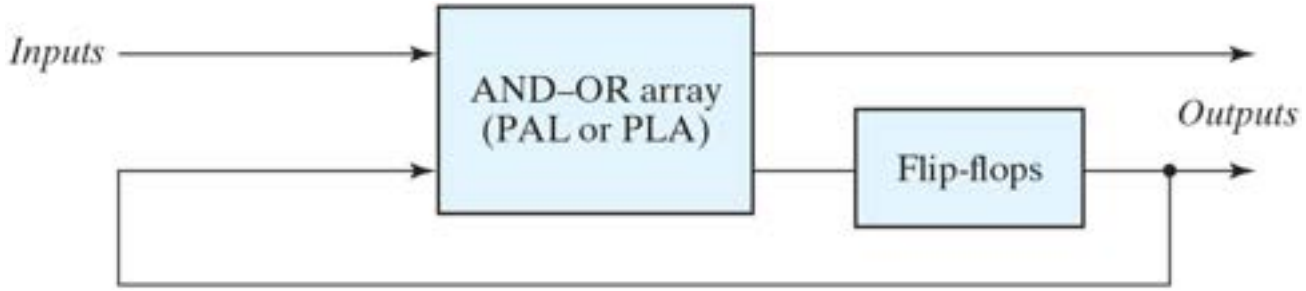


Figure 7.19
Basic macrocell logic.

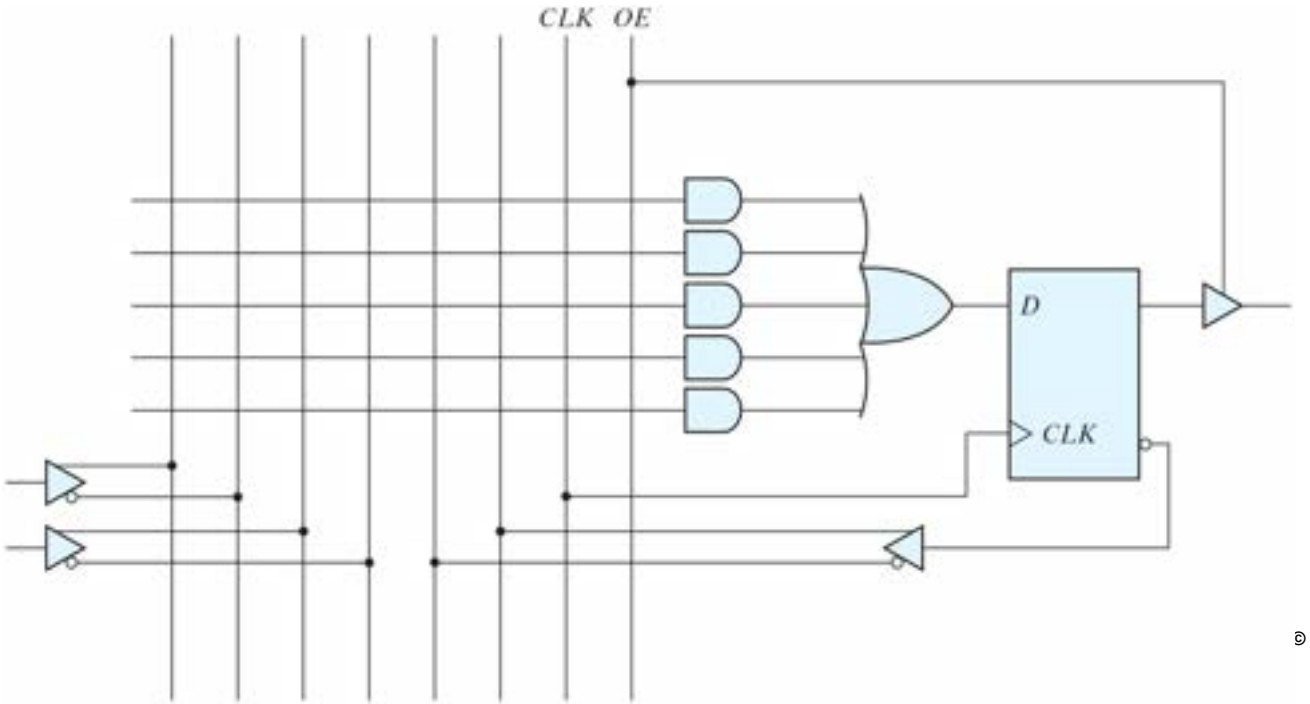


Figure 7.20
General CPLD configuration.

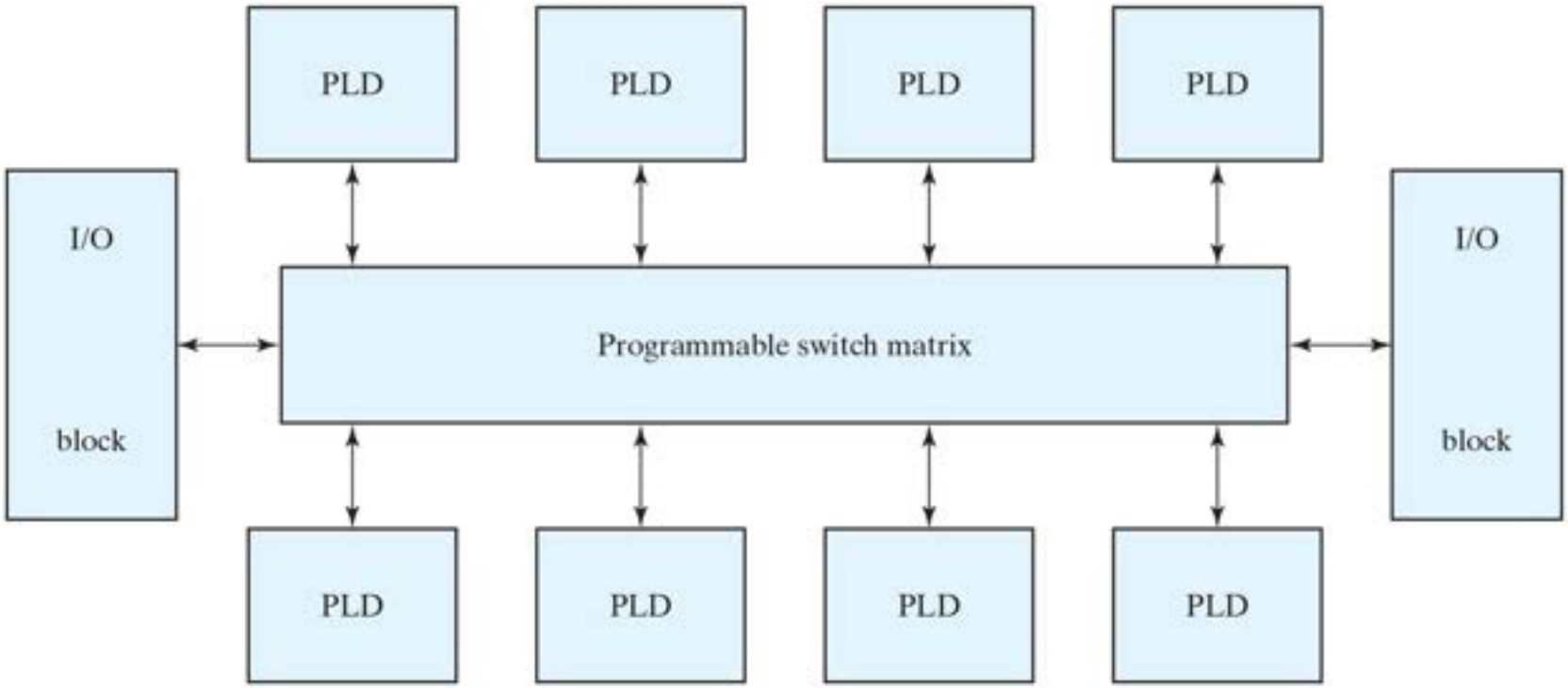
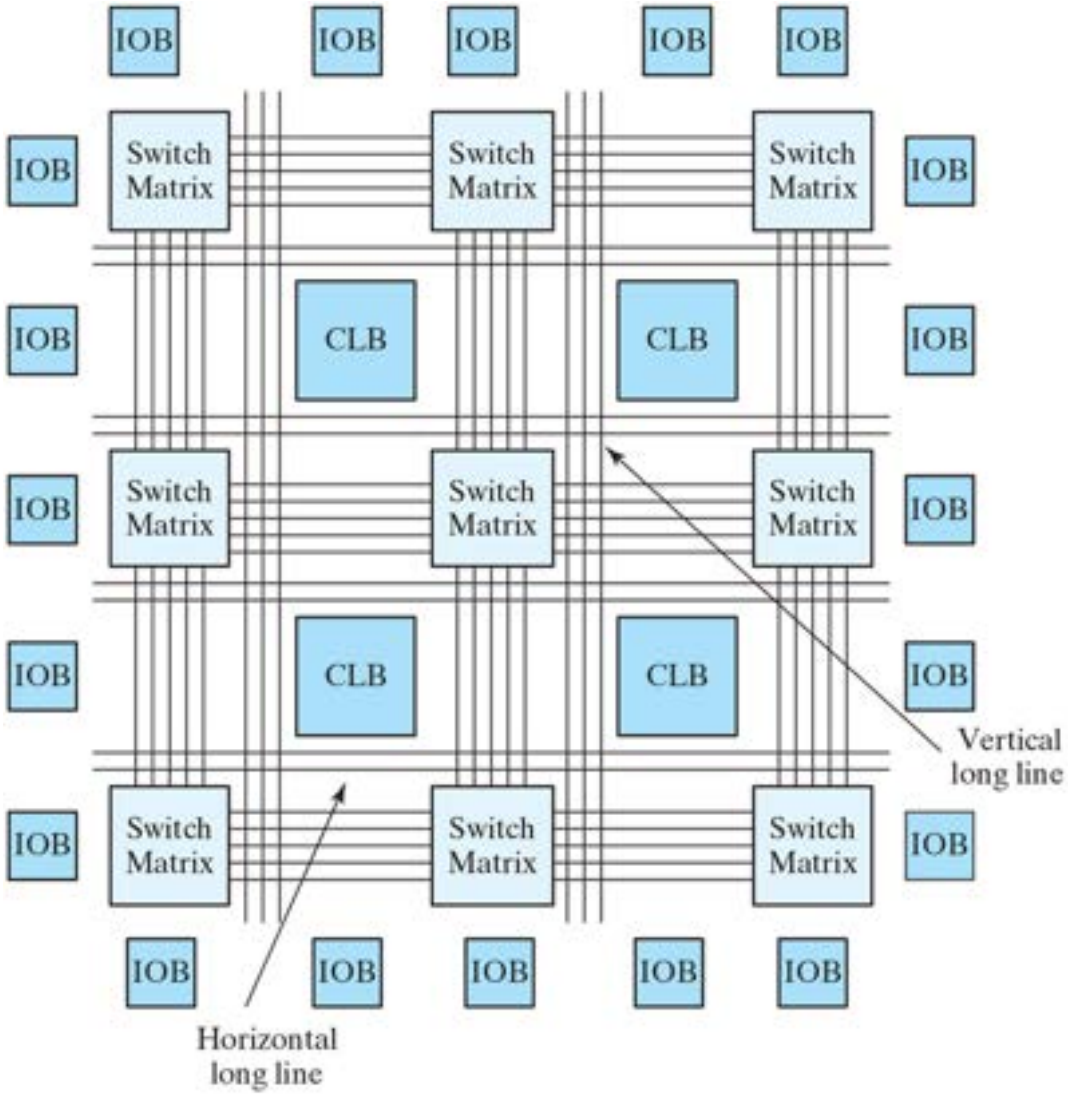


Figure 7.21
Basic architecture of Xilinx Spartan and predecessor devices.



앞으로 수강할 과목 추천

- 회로 및 시스템 트랙

디지털회로설계 및 언어	EE361

  - 복잡한 디지털회로를 효율적으로 모델링하여 빠른 시간 내에 회로의 기능을 검증하고, 이를 재사용할 수 있도록 하는 하드웨어 설계언어에 대한 기술을 습득한다. 논리회로의 지식을 바탕으로 디지털 시스템의 설계에 필요한 상태머신의 설계, 프로그램 로직 어레이, ROM, FPGA(Field Programmable Gate Array)에 대한 요소기술을 습득한 후, 이를 설계하는데 필요한 하드웨어 설계언어에 대한 지식 및 응용기술을 배운다.

마이크로프로세서	EE364
임베디드시스템설계	EE367
로봇제어공학	EE461