



Ch.7 Memory And Programmable Logic - part A

ch7 intro

Memory

많은 양의 binary information을 저장할 수 있는 cell들의 collection

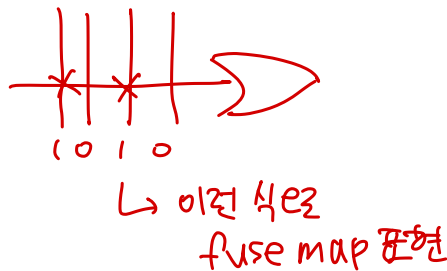
1. **RAM**(random access memory) : read + **write**
 - a. cpu가 원할 때 바로바로 전해줄 수 있는 memory
2. **ROM**(read only memory) : **only read**
 - a. 사용자가 값을 변경할 수는 없음 → 공정 과정에서 memory의 형태를 갖추

PLD(Programmable Logic Device, PLD)

electronic path로 연결되어 있는 내부 logic gate로 이루어진 integrated circuit

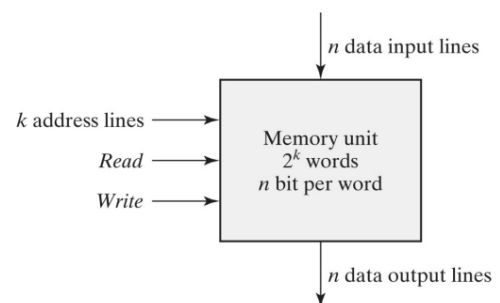
(like fuse)

1. **PLD**(programmable logic device)
2. **PLA**(programmable logic array)
3. **PAL**(programmable array logic)
4. **FPGA**(field programmable gate array)



RAM(Random Access Memory)

- **address** : read, write할 위치 → 특정 line
- **decoder**
 1. memory 내부에 접근하여 address 얻음
 2. address가 가리키는 word open



n data를 k line으로 write/read
↓
word 단위 (2바이트, 4바이트 ...)

⇒ 2^k 개 저장 가능

- example → 1024 X 16 memory

- address : 10bit
- word : 16bit

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

Handwritten notes:
 - Red box around address 0000000010 and its content 0000110101000110.
 - Red bracket on the left side of the address column labeled "k-bit (10bit) Address".
 - Red bracket on the right side of the content column labeled "n-bit (16bit) Data -> Word".
 - Red text "2^k개" next to the address bracket.
 - Red text "⇒ 2^k x n" at the bottom.

Write and Read Operations

• write

1. 원하는 word의 binary address를 address line에 전달
2. 메모리에 저장해야 하는 data bits를 data input line에 전달
3. activate write input

→ **cycle time of memory** : write operation을 끝내는데 필요한 시간

• read

1. 원하는 word의 binary address를 address line에 전달
2. activate read input

→ **access time of memory** : read operation을 끝내는데 필요한 시간

⇒ **memory time** = ^①cycle + ^②access(memory의 spec)

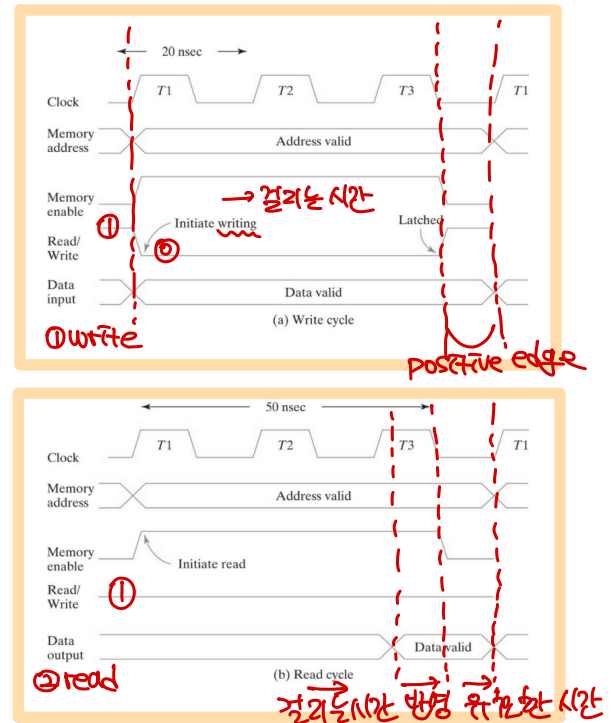
- **Timing Waveformz** (주의는 clock의 역전)

- CPU : 50MHz → 20nsec clock cycle

- Memory time : less than 50nsec

→ memory가 2.5개

즉, 3개의 clock cycle 필요로 함



Types of Memories

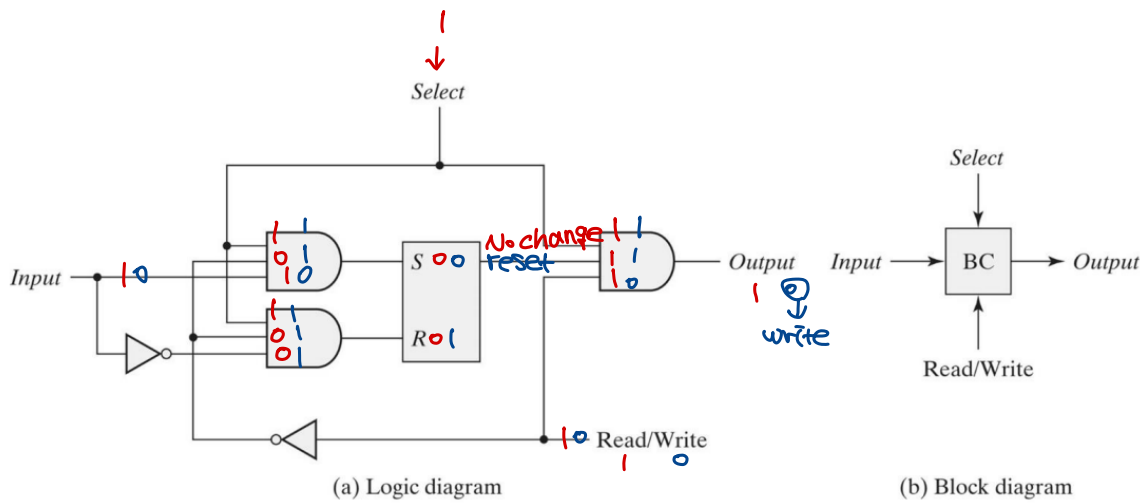
- **Random access Memory** : 순서 상관 없이 무작위로 저장
 - The access time is always the same
- **Sequential access Memory** : 순차적으로 데이터 읽음
 - 먼저 들어간 데이터가 가장 늦게 read The access time is variable.

- **Volatile** : pc에 전원이 들어오면 저장하고 꺼지면 저장하지 않음
- **Nonvolatile** : pc가 꺼져도 memory 유지
 - ex) magnetic disk ,ROM(Read Only Memory)

- **Static RAM(SRAM)** : 전원이 들어오는 동안 저장된 정보가 유지
 - 사용하기에 쉽고 더 빠르게 read/write
- **Dynamic RAM(DRAM)** : 공간을 동적할당 → 사용할 때 재할당
 - 전력 소비를 줄이고 많은 양의 저장 용량 가능

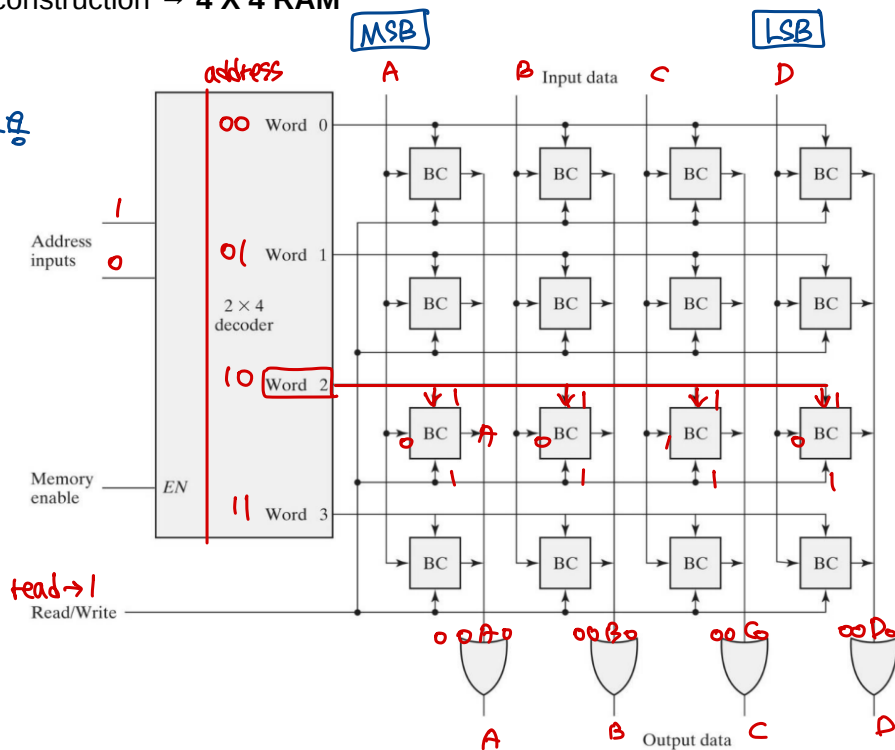
Memory decoding

- **binary cell**(memory cell) : one bit의 information을 담고 있음
 - 내부의 **SR Latch**로 one bit 저장
 - ① select = 0 ⇒ 동작 X
 - ② select = 1 ⇒ 동작 O



- Internal construction → 4 X 4 RAM

★ $2^k \times m$ RAM
 ⇒ k to 2^k DEC 사용



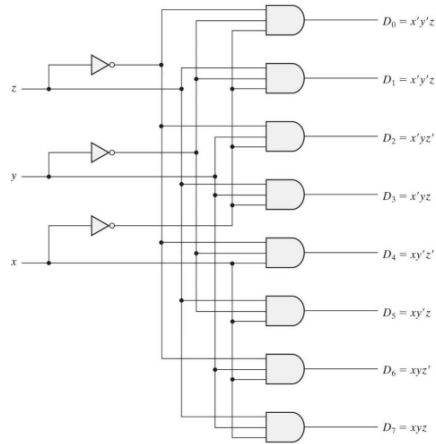
- $EN = 0$ → decoder의 모든 output이 0 + memory 중 아무것도 선택 x
- $EN = 1$ → 4개의 word 중 하나가 선택 → read/write input에 따라 수행
 - read operation : 선택된 4bit의 word가 OR gate를 통해서 output
 - write operation : 선택된 4bit의 word에 input line에 쓰인 data write

Coincident Decoding

- n-to- 2^n decoder : 2^n 개의 and gate 사용 → memory word 개수만큼 필요
 ⇒ address가 많아질수록 비효율적 ★

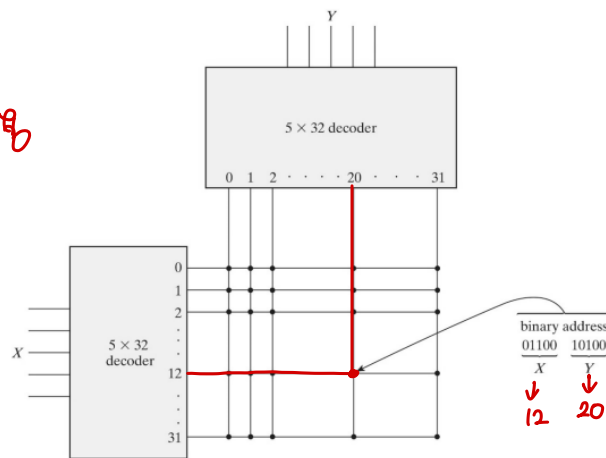
* 2^3 to 2^8 decoder

address : 2^{10}
word = 16 bit
⇒ 1024개의
and gate 사용



- solution : binary address를 반반씩 나눠서 decoder 2개 사용!

⇒ 64개의
and gate 사용



Error Detection and Correction : Hamming code

1의 개수를 짝수화 → 추가 bit

문제가 생겼을 때 check 가능

parity bit : 첫번째 1 빼고 XOR
check bit : parity bit 넣어서 XOR → 0이어야 함

XOR
: 같으면 0
다르면 1

8-bit data word : 11000100

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	

- $P_1 = \text{XOR of bits}(3,5,7,9,11) = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$
- $P_2 = \text{XOR of bits}(3,6,7,10,11) = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$
- $P_4 = \text{XOR of bits}(5,6,7,12) = 1 \oplus 0 \oplus 0 \oplus 0 = 1$
- $P_8 = \text{XOR of bits}(9,10,11,12) = 0 \oplus 1 \oplus 0 \oplus 0 = 1$

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

When the 12 bits are read from memory, the four check bits

- $C_1 = \text{XOR of bits}(1,3,5,7,9,11) = 0$
- $C_2 = \text{XOR of bits}(2,3,6,7,10,11) = 0$
- $C_4 = \text{XOR of bits}(4,5,6,7,12) = 0$
- $C_8 = \text{XOR of bits}(8,9,10,11,12) = 0$

bit가 8개 → P₁, P₂, P₄, P₈

	P ₈	P ₄	P ₂	P ₁
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

8+4 = 12개

- Error example

even parity $C=C_8C_4C_2C_1=0000$

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

No error
Error in bit 1
Error in bit 5

Check bits

$C_8 C_4 C_2 C_1$

With error in bit 1 : $C = 0 \ 0 \ 0 \ 1$

With error in bit 5 : $C = 0 \ 1 \ 0 \ 1$

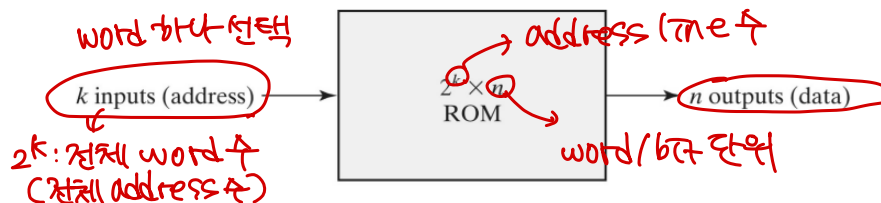
이 나쁜 C가 에러난 곳

$C_1 = \text{XOR of bits}(1,3,5,7,9,11)$
 $C_2 = \text{XOR of bits}(2,3,6,7,10,11)$
 $C_4 = \text{XOR of bits}(4,5,6,7,12)$
 $C_8 = \text{XOR of bits}(8,9,10,11,12)$

- hamming code : single error만 detect 가능
 - + double error → 추가적인 parity bit P13을 1로 추가
 - 모든 read한 P에 대해서
1. $C=0, P=0$: error x
 2. $C \neq 0, P=1$: single error → correct 가능(위치 알 수 있음)
 3. $C \neq 0, P=0$: double error → detect는 가능하나 correct 불가(위치는 모름)
 4. $C=0, P=1$: P13 bit에서 error 발생

ROM(read only memory)

ROM = AND gate의 Decoder + OR gates



- 영구적으로 data 저장 → programing 되어 있는 data ⇒ 전원 끄더라도 data 유지
- 사용 중에 write는 불가, read만 가능
- address line : k개의 input line(실제 input은 없다)
- n bit word : n개의 outputs

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	0	0	0	1	0	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

address line

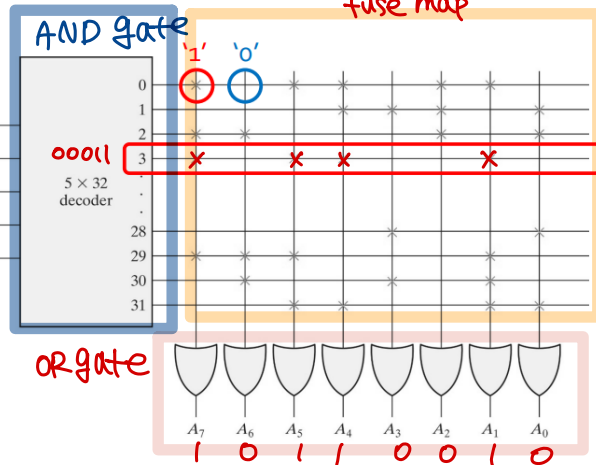
word

address line

* 32 x 8 ROM

programming으로 생성

fuse map



32H words

example 7-1

3bit의 input을 받아서 input의 제곱을 출력하는 example

0u7 input

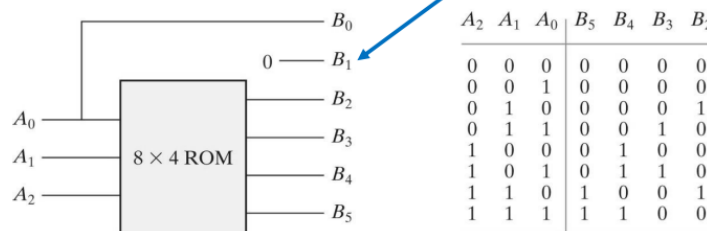
0u49 output

⇒ 8x6 ROM

하지만 B0, B1은 고정

8x4 ROM

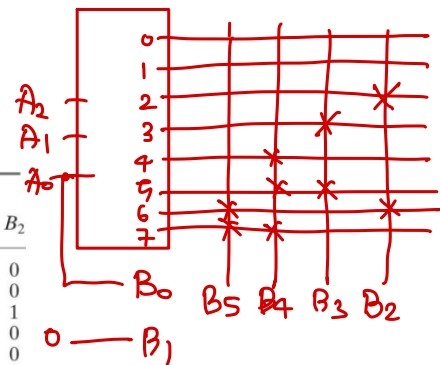
Inputs			Outputs						Decimal
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	1	1	49



(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

- **large** → mask programming : fuse map 위에 fuse 해야 하는 부분 제외하고 mask
- **small** → programmer : ROM writer 기능을 통해서 사용하도록 설정
 - PROM : 다시 programming 불가
 - EPROM : UV light으로 초기값을 다시 설정 가능
 - EEPROM : 전기 신호를 이용해 erasable