# Digital Logic Design

Sung-Soo Lim

# Combinational Logic(조합 논리 회로) vs. Sequential Logic(순차 논리 회로)

- 조합 논리 회로
  - 출력은 현재 입력에 의해서만 결정됨
  - No memory
    - 이전 입력/출력을 기억할 방법이 없음
- 순차 논리 회로
  - 출력은 현재 입력과 현재 상태(이전의 입력/출력)에 의해 결정됨
  - 이전 입력/출력을 기억해야 함

**Door Bell**

push button (input), bell rings (output)

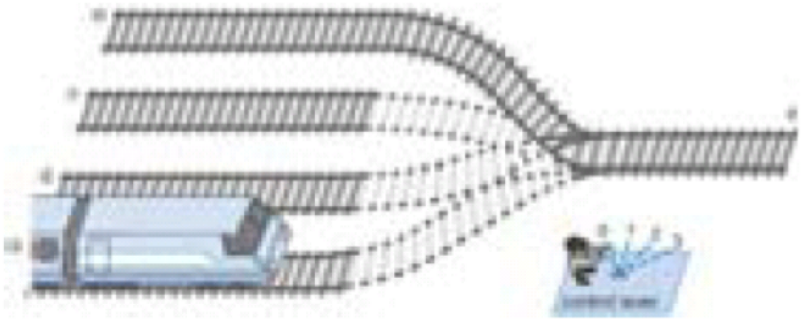
**Garage Door**

push button (input), if door open (state), close door (output)

push button (input), if door closed (state), open door (output)

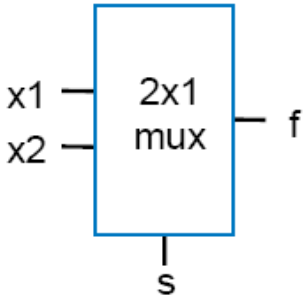
# 조합 논리 회로의 예: MUX

- Multiplexer (MUX)
  - **N 개의 입력 중에서 하나의 출력으로 나갈 입력을 선택 (선택을 위해 별도의 이진 입력 추가)**
    - **2 input mux**
      - **1 비트의 선택(select) 입력 필요**
    - **4 input mux**
      - **needs 2 select inputs**
    - **8 input mux**
      - **3 select inputs**
    - **N inputs**
      - **log2(N) selects**
  - **예) 기찻길 선로 선택 시스템**

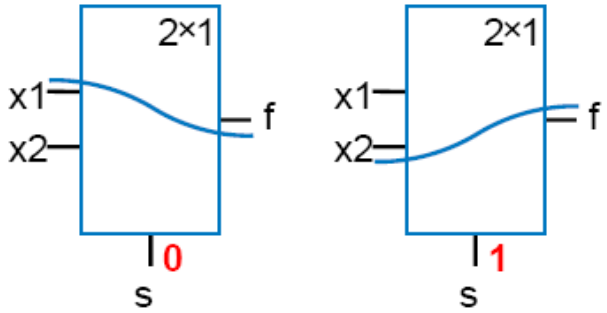


# 조합 논리 회로의 예: MUX

definition




example



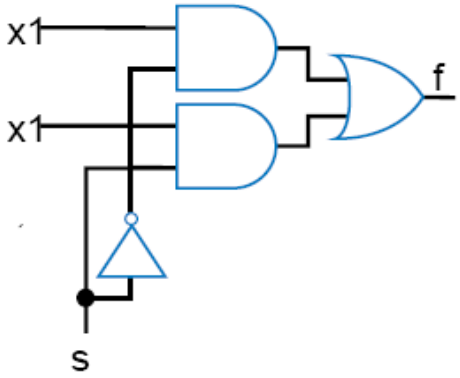

truth table

| s | x1 | x2 | f |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

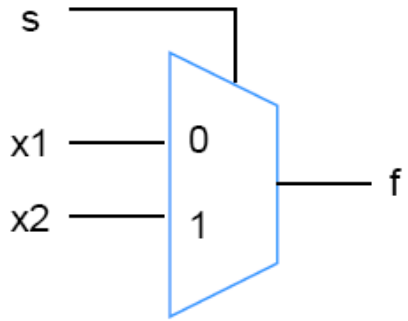
logic expression
(optimized)

f = s'x1 + sx2

circuit implementation

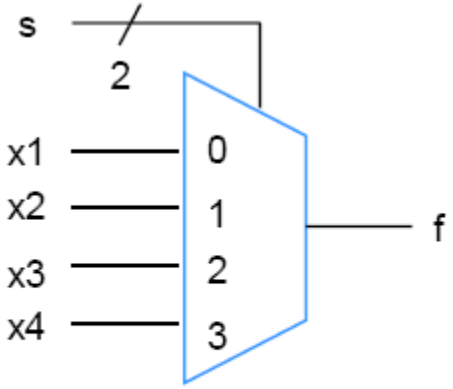



graphical symbol

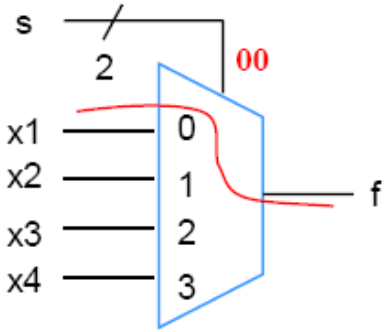


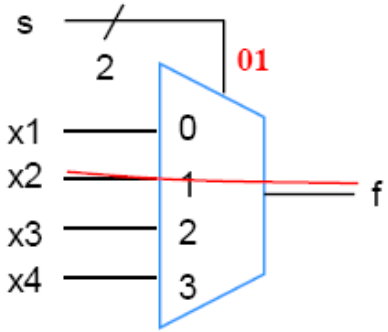
# Larger MUX

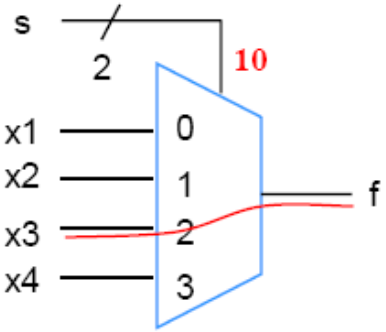
- 기존 MUX를 확장하여 더 큰 MUX 구성
  - 4x1 mux : 2 select lines
  - 8x1 mux : 3 select lines

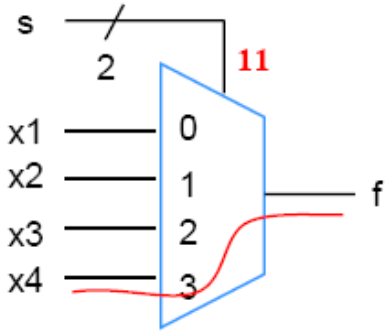



4x1 mux



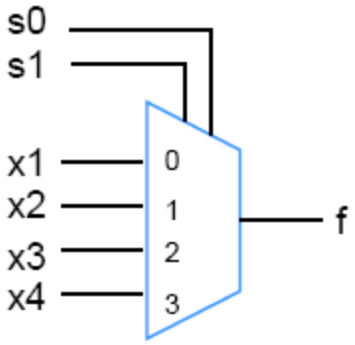



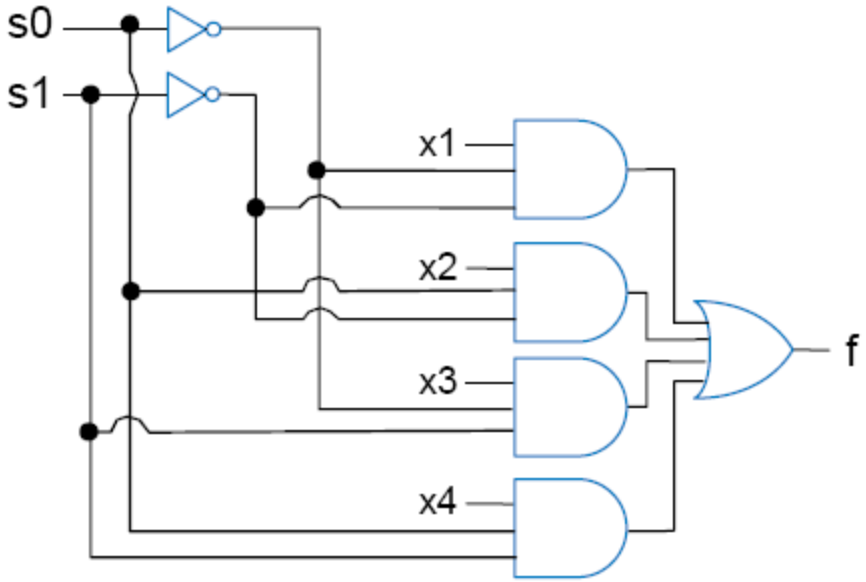
# 로직 게이트로 4-to-1 MUX 구현하기

- 4-to-1 MUX 구현 방법?
  - 진리표 이용하기
    - If S1 = 0 and S0 = 0, output x1
    - If S1 = 0 and S0 = 1, output x2
    - If S1 = 1 and S0 = 0, output x3
    - If S1 = 1 and S0 = 1, output x4



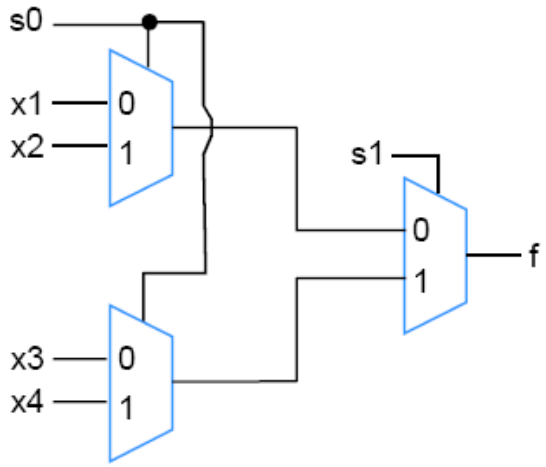

4x1 mux

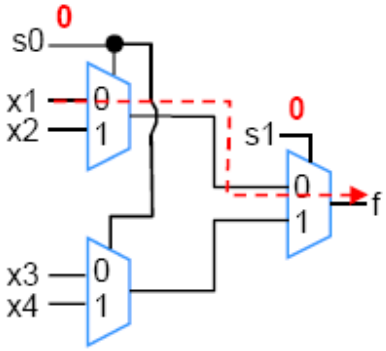
| s1 | s0 | f |
|---|---|---|
| 0 | 0 | x1 |
| 0 | 1 | x2 |
| 1 | 0 | x3 |
| 1 | 1 | x4 |

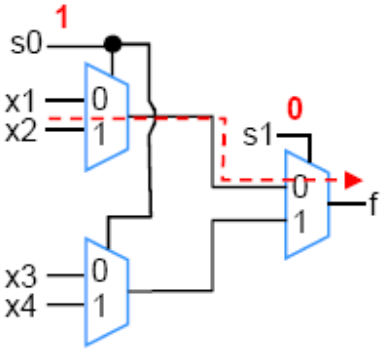


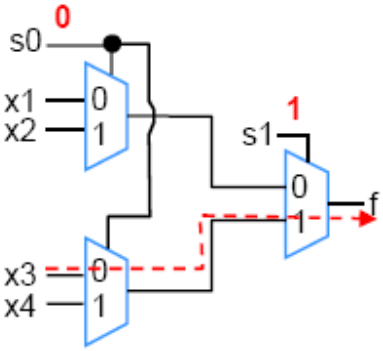
# 소규모 MUX로 4-to-1 MUX 구현하기

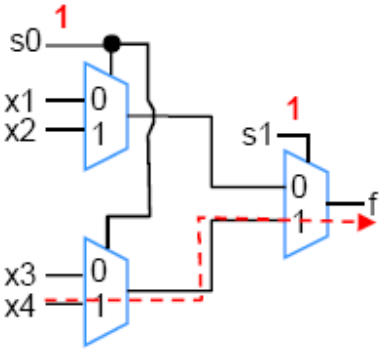
- 2-to-1 MUX로  4-to-1 MUX 구성하기



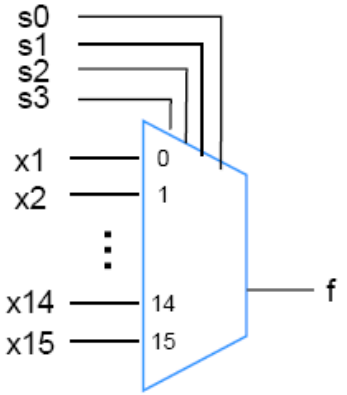



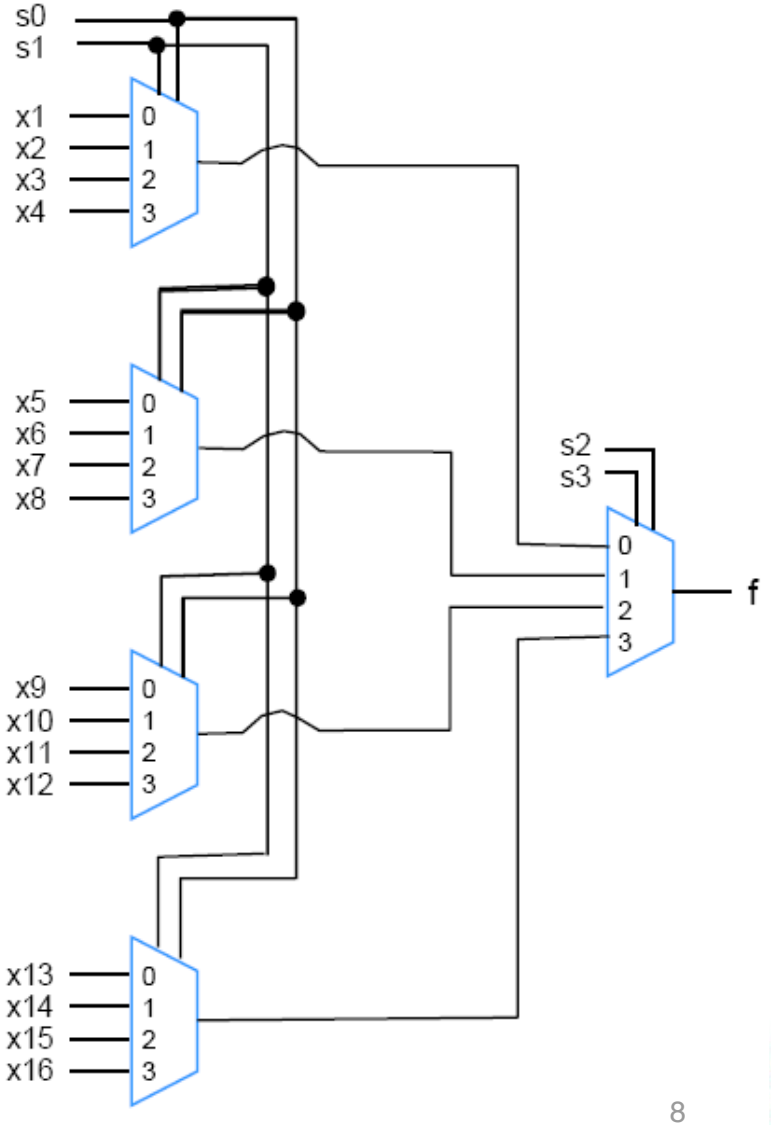

KESL
Kookmin Univ. Embedded Systems Lab

# 소규모 MUX로 16-to-1 MUX 구성하고

- 16-to-1 input MUX
  - 16 inputs
  - Need log2(16) = 4 select lines
- 구현 방법
  - Truth table
  - 소규모 MUX 이용하기
    - 2-to-1 or 4-to-1 MUXes




16x1 mux

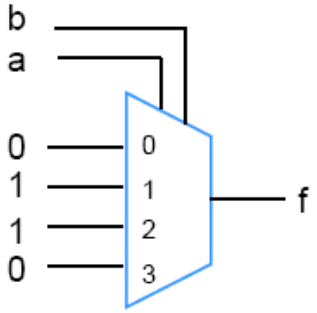


# 만능 MUX??: XOR Gate 를 MUX로

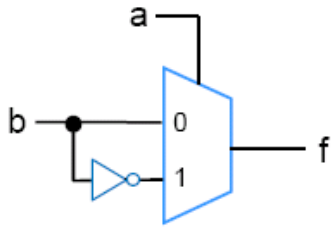
- MUX가 꽤 만능이다!!
  - **Consider XOR gate**

| a | b | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| a | f |
|---|---|
| 0 | b |
| 1 | b' |




Direct translation of truth table to 4-to-1 MUX



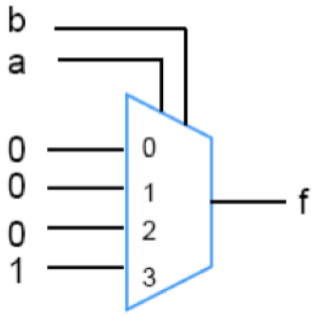

XOR implementation using modified truth table

# AND 게이트도 MUX로 구현하기

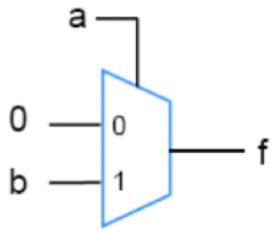
- AND Gate
  - **Direct translation**
  - **Modified Translation**

| a | b | f |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

| a | f |
|---|---|
| 0 | 0 |
| 1 | b |




Direct translation of truth table to 4-to-1 MUX




AND implementation using modified truth table

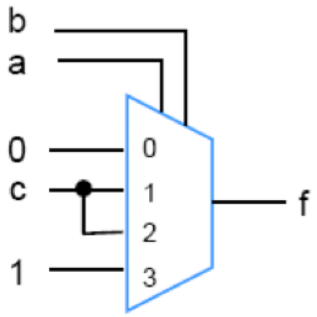
# 3개 입력 중 더 많은 입력 찾기(0과 1 중에서)

- Three-input majority circuit
  - **Interface**
    - **입력: a, b, c**
    - **출력: f**
  - **Function**
    - **만약 0 입력이 더 많은 경우, f = 0**
    - **만약 1 입력이 더 많은 경우, f = 1**

truth table

| a | b | c | f |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

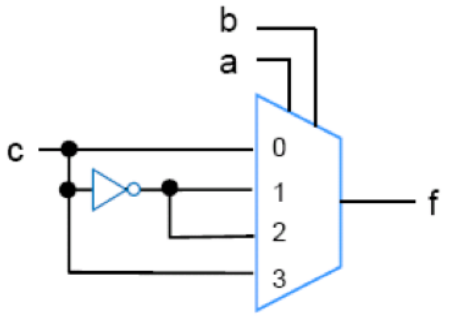
| a | b | f |
|---|---|---|
| 0 | 0 | '0' |
| 0 | 1 | c |
| 1 | 0 | c |
| 1 | 1 | 1 |




Three-input majority circuit

# 3-Input XOR Circuit: 4-to-1 MUX로 구현

- 3-input XOR gate
  - **4-to-1 MUX로 구현하기**
  - **2 select lines (a and b)**

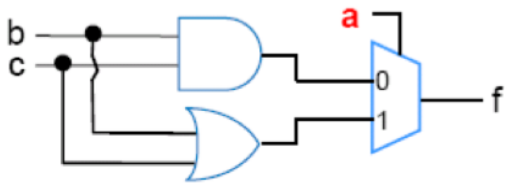



| a | b | c | f | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | c |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 1 | c' |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | c' |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 0 | c |
| 1 | 1 | 1 | 1 | |

# 샤논 확장 이론(Shannon Expansion Theorem)

- MUX를 쓸 수 있는 회로로 바꾸기 위해 selection 신호 분리해 내기
  - **복잡한 회로를 간단하게 구현하는 방법**
- Shannon Expansion Theorem
  - **Boolean logic function F를 Boolean variable X에 대해 확장하기 (in terms of (or with respect to) a Boolean variable X)**

F = a'bc + ab'c + abc' + abc
= a'(bc) + a(c + b)




$$F(w_1, w_2, w_3, \ldots, w_n) = w_1' \cdot f(w_1=0, w_1, w_2, w_3, \ldots, w_n) + w_1 \cdot f(w_1=1, w_2, w_3, \ldots, w_n)$$

# Shannon Expansion Theorem 예

- Three-input majority function을 변수 a에 대해 확장하기
  - **F(a, b, c) = a'bc + ab'c + abc' + abc**

$$F(w1, w2, \ldots, wn) = w1' \cdot f(w1=0, w1, w2, \ldots, wn) + w1 \cdot f(w1=1, w2, \ldots, wn)$$

What happens when a=0?

$$F(a=0, b, c) = a'bc + ab'c + abc' + abc$$
$$= (1)bc + (0)b'c + (0)bc' + (0)bc$$
$$F_{a'} = bc$$

This is the cofactor of F with respect to a'

What happens when a=1?

$$F(a=1, b, c) = a'bc + ab'c + abc' + abc$$
$$= (0)bc + (1)b'c + (1)bc' + (1)bc$$
$$= b'c + bc' + bc$$
$$F_a = c + b$$

This is the cofactor of F with respect to a

$$F = a'F_{a'} + aF_a$$
$$= a'(bc) + a(c+b)$$

Rewrite the function using Shannon Expansion Theorem

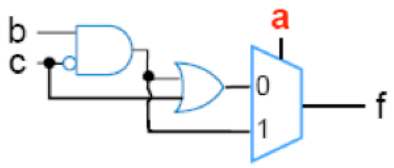
# Shannon Expansion Theorem 예

- 어떤 변수에 대해서도 확장이 가능함
  - **F(a, b, c) = a'c + bc'**

Expand for variable a

F(a=0, b, c) = (1)c + bc'
= c + bc'
F(a=1, b, c) = (0)c + bc'
= bc'

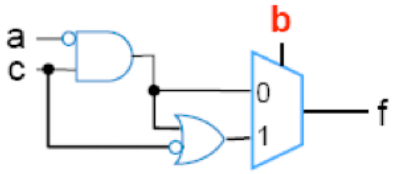
F = a'Fa' + aFa
= a'(c+bc') + a(bc')




Expand for variable b

F(a, b=0, c) = a'c + (0)c'
= a'c
F(a, b=1, c) = a'c + (1)c'
= a'c + c'

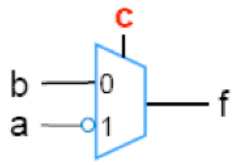
F = b'Fb' + bFb
= b'(a'c) + b(a'c + c')




Expand for variable c

F(a, b, c=0) = a'(0) + b(1)
= b
F(a, b, c=1) = a'(1) + b(0)
= a'

F = c'Fc' + cFc
= c'(b) + c(a')



# Shannon Expansion Theorem 예

- 복수의 변수에 대해서도 확장이 가능함
  - **Expand F(a, b, c) = a'c' + ab + ac를 a와 b에 대해 확장하기**

**Set a=0, b=0**

F(a=0, b=0, c) = a'c' + ab + ac
= (1)c' + (0)(0) + (0)c
$F_{a'b'}$ = c'

**Set a=0, b=1**

F(a=0, b=1, c) = a'c' + ab + ac
= (1)c' + (0)(1) + (0)c
$F_{a'b}$ = c'

**Set a=1, b=0**

F(a=1, b=0, c) = a'c' + ab + ac
= (0)c' + (1)(0) + (1)c
$F_{ab'}$ = c

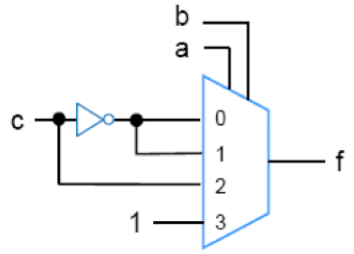
**Set a=1, b=1**

F(a=1, b=1, c) = a'c' + ab + ac
= (0)c' + (1)(1) + (1)c
$F_{ab}$ = 1

**Rewrite the function using Shannon Expansion Theorem**

$F = a'b'F_{a'b'} + a'bF_{a'b} + ab'F_{ab'} + abF_{ab}$

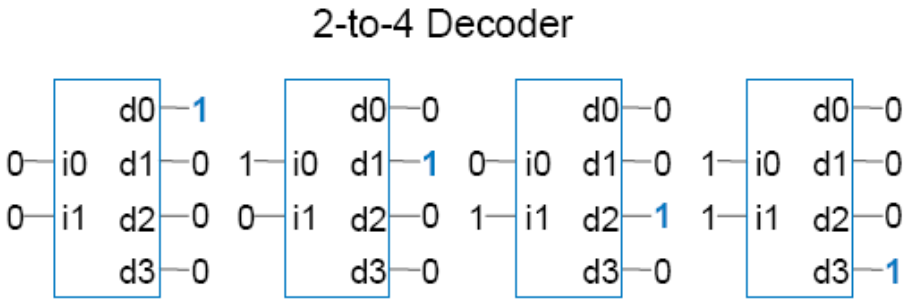
$= a'b'(c') + a'b(c') + ab'(c) + ab(1)$

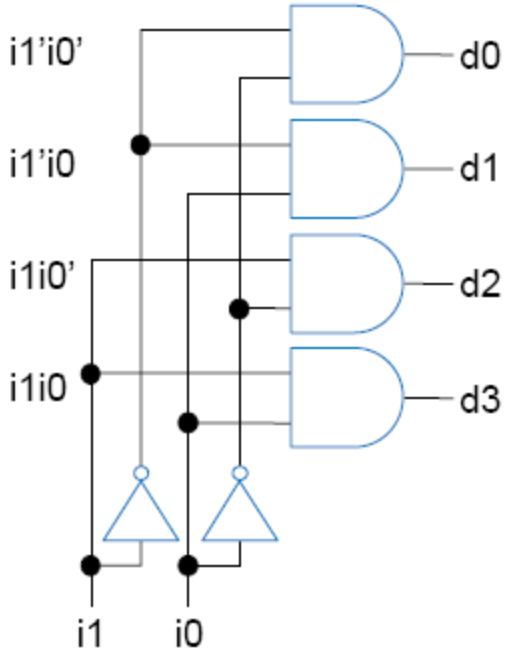



KESL
Kookmin Univ. Embedded Systems Lab

# Decoder

- Decoder
  - 입력 이진수를 하나의 출력으로 연결함
    - “One-hot encoded”
- 2-to-4 Decoder
  - 2개 입력, 4 개 출력(하나의 출력만 1)
- 내부 설계
  - AND 게이트를 활용



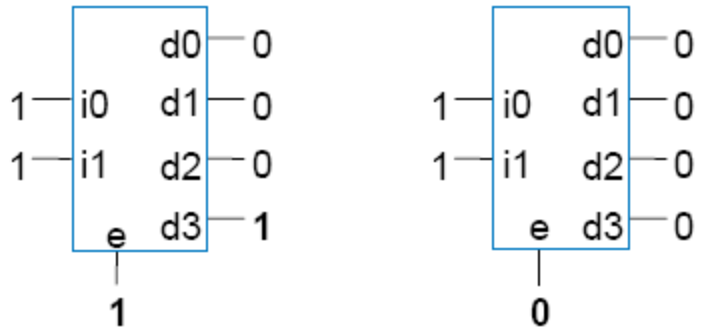




KESL
Kookmin Univ. Embedded Systems Lab

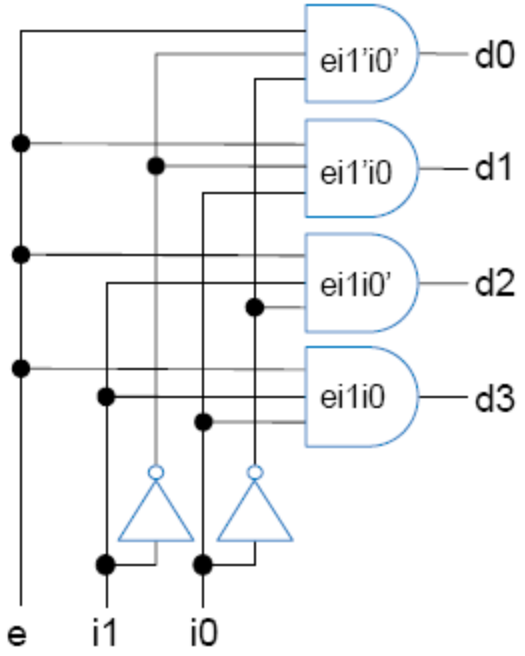
# Decoder with Enable

- Decoder with enable e
  - Outputs all 0 if e=0
  - Regular behavior if e=1

2-to-4 Decoder with enable

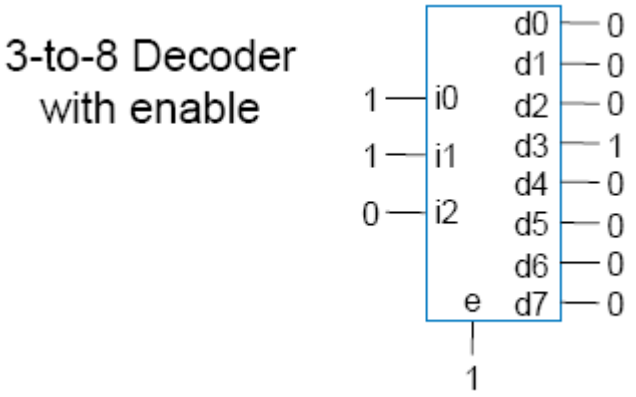


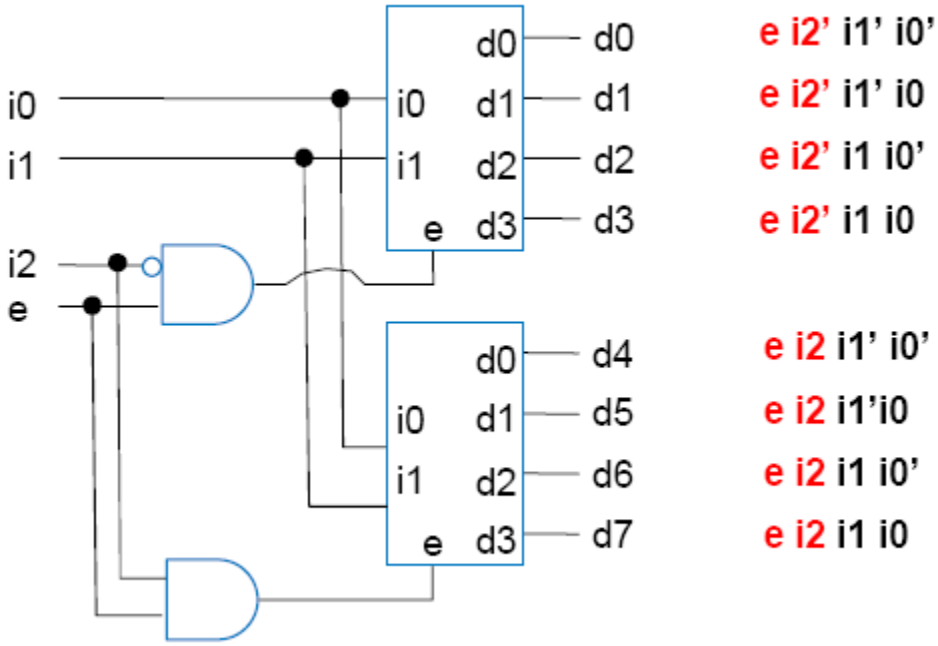

| e | i1 | 10 | d0 | d1 | d2 | d3 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |



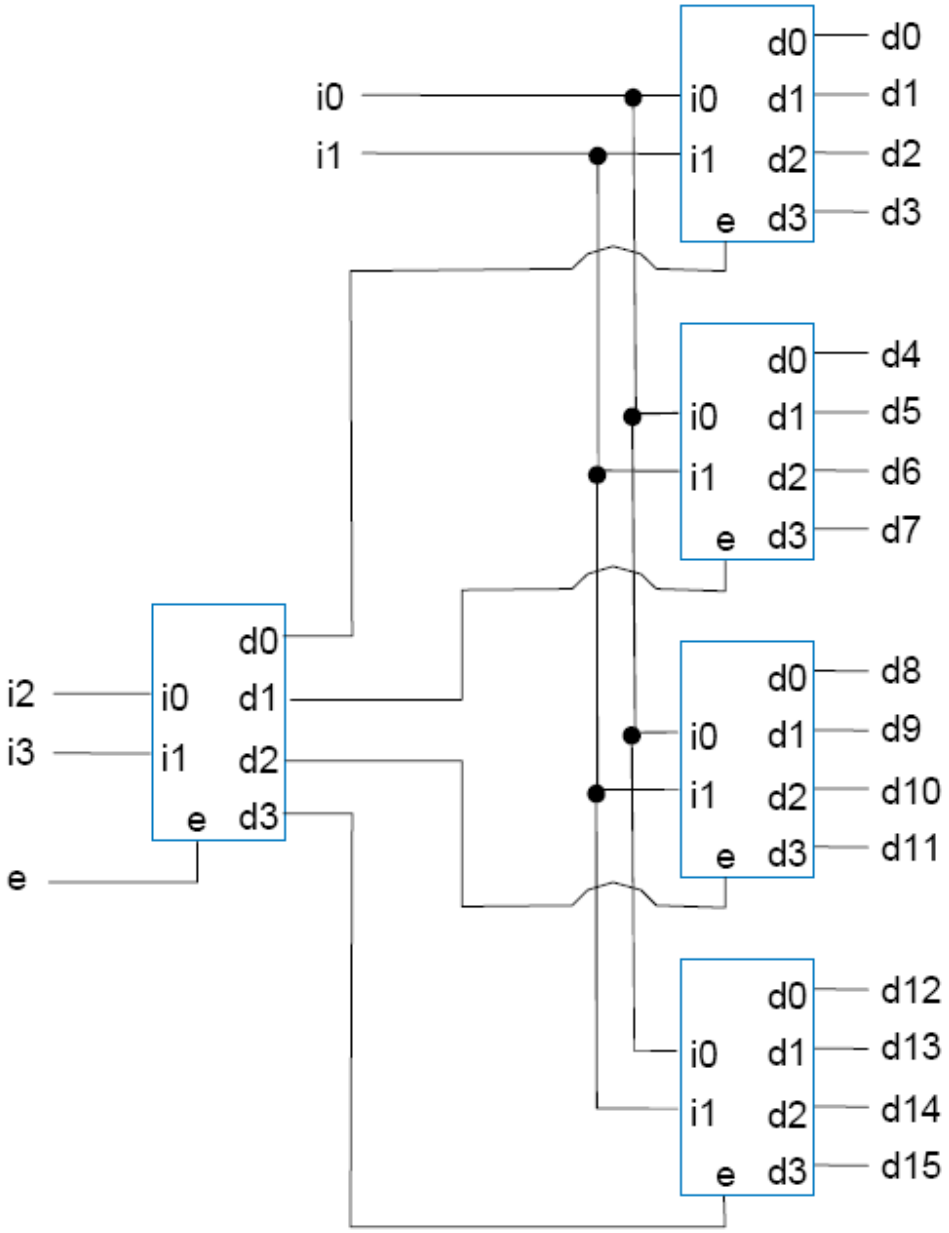
# 소규모 Decoder로 큰 Decoder 구성하기

- N-input decoder: 2N outputs
- 어떤 규모의 Decoder도 구현 가능함
  - two-level logic
  - 소규모 decoders 활용
- 3-to-8 decoder
  - 3 inputs: i2, i1, i0
  - 1 control input: e
  - 8 (23) outputs: d0, …, d7
- 3-to-8 decoder를 2-to-4 decoder로 만들기



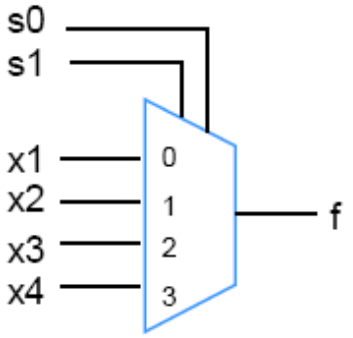



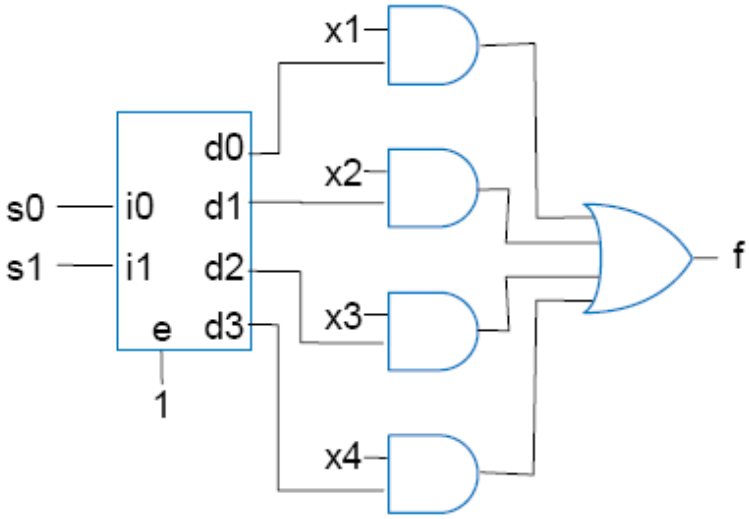
# Even Larger Decoder



# Decoder와 논리 게이트로 4-to-1 MUX 구현하기

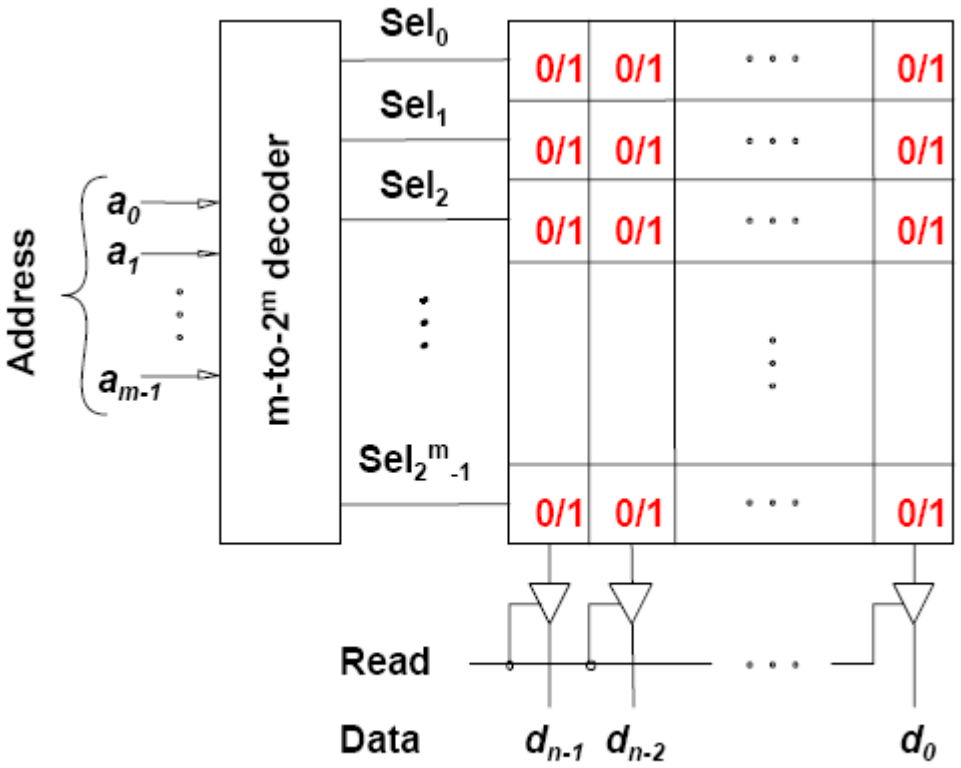
- Decoder를 써서 4-to-1 MUX 구현하기
  - s0, s1 선택 시그널
  - x1, …, x4 ANDed with decoder output
    - 오직 1개의 출력만 1
  - OR gate 가 x 값 출력 생성
    - selected data appears on output (x OR 0 = x)



# ROM Decoder

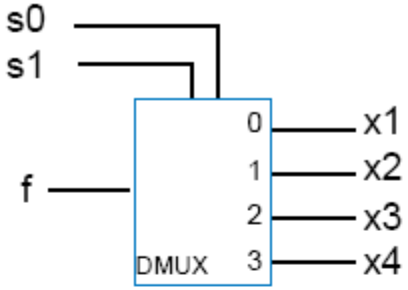
- 메모리에서 주소에 따라 데이터를 선택하는 회로
- ROM (read-only memory)
  - 각 셀은 1/0 중 하나의 데이터 기록
  - $2^m$ 행, n cell/행으로 구성
    - 각 행은 n bits 데이터 저장
    - 주소란 원하는 행을 선택하는 값
  - 선택(selection) 라인에 의해 원하는 행의 데이터 접근
  - Decoder는 원하는 행을 접근하기 위한 시그널 생성
    - 원하는 데이터는 data line에 출력

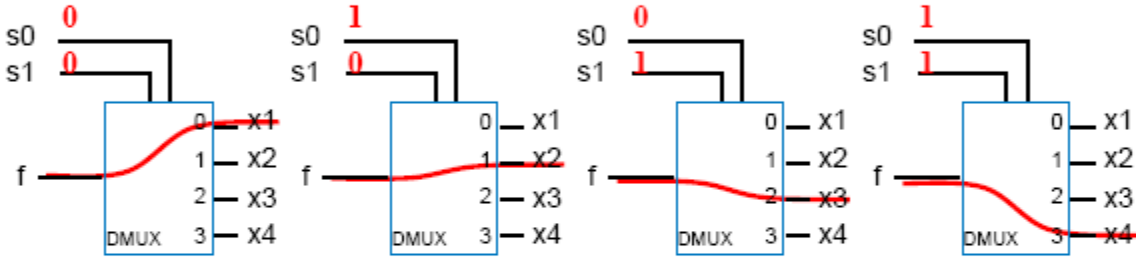


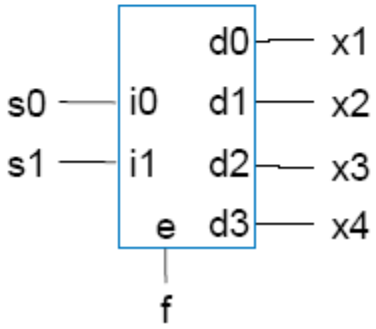

A $2^m$ x $n$ read-only memory (ROM) block

KESL
Kookmin Univ. Embedded Systems Lab

# 1-to-4 DEMUX Using Decoder and Logic Gates



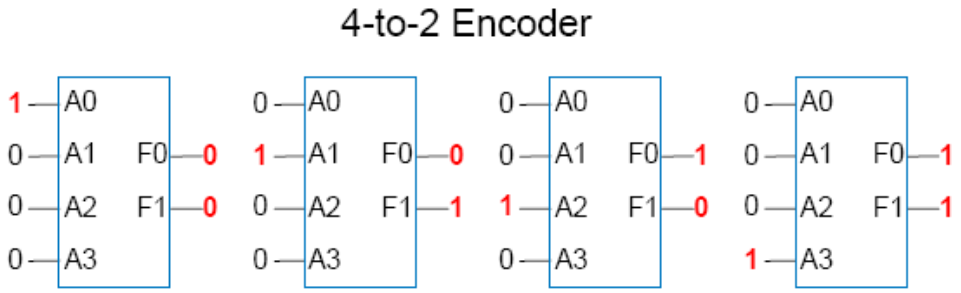




1-to-4 Demultiplexer implemented with decoder

# Encoder

- Encoder
  - Decoder의 반대
  - 하나의 ‘One-Hot Input’을 이진 출력으로 변환
- 4-to-2 Encoder
  - 4 inputs, 2 outputs
- 내부 설계
  - F0 = A1 + A3
  - F1 = A2 + A3
  - 나머지 입력 조합은 Don’t Care




| A3 | A2 | A1 | A0 | F1 | F0 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |