Ch.4 Combinational Logic - part C

Decoders

범용 회로 → “detection” 하는 아이

- circuit analysis, design techniques
- can be used to implement arbitrary functions.
- introduced to abstraction and modularity as h/w design principles

⇒ 더 복잡한 h/w를 designing 하는데 사용할 예정

→ ex . 8bit로 encoding 된 글자가 computer에 들어오면 detect 해서 상응하는 출력 라인 활성화

- n-bit의 input에서 2의 n승 개의 output을 만들어 냄. 그 중 하나만 “ 1 “

- 작은 decoder에서 큰 decoder 만들 수 있음(like adder)

input output

n-to-2^n decoder

n-bit input → 2^n output

n개의 input은 2^n개의 output 중 uniquely true인 binary number를 나타냄

2-to-4 decoder

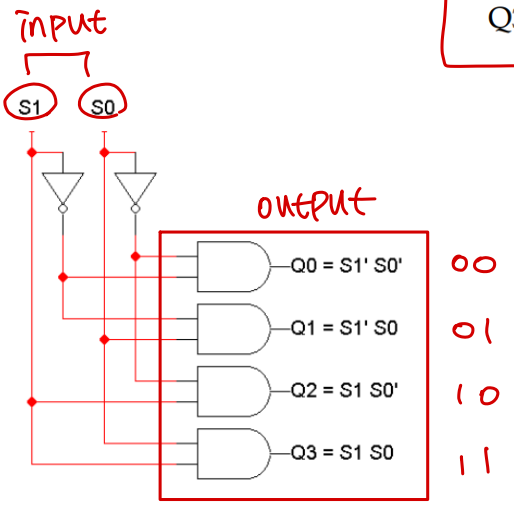
- 2개 input : S1, S0
- 4개 output : Q0, Q1, Q2, Q3
- i번째 input이 true ⇒ Qi가 true

$Q0 = S1' S0'$
$Q1 = S1' S0$
$Q2 = S1 S0'$
$Q3 = S1 S0$

00
01
10
11

S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

0
1
2
3



1. 어떤 output이 1인지 detect 가능
2. 4개 중의 하나를 decode 가능

3-to-8 decoder

Enable inputs

decoder의 variation.

회로를 필요한 경우에만 작동시키고 싶을 때 사용(activate / deactivate)

EN	S1	S0	Q0	Q1	Q2	Q3
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

deactive
active

input이 상관없이 모든 output이 0일 때 ⇒ disable
enable

- EN=1 ⇒ activate the decoder → output 중 하나만 1
- EN=0 ⇒ deactivate the decoder → 모든 output이 0

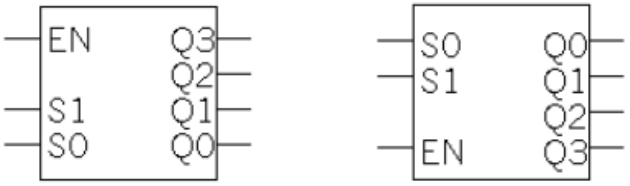
- abbrebviated truth tables
  - EN=0일 때의 S0, S1에 상관없이 다 output이 0
  - truth table을 축약해서 표현 가능
    - ↳ don't care!

EN	S1	S0	Q0	Q1	Q2	Q3
0	x	x	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

Blocks and abstraction

decoder가 block symbol 형태라면 내가 필요할 때마다 가지고 와서 사용 가능(함수처럼!)

(decoder는 그만큼 많이 쓰이는 회로이며 특성이 강함)



$Q0 = S1' S0'$
$Q1 = S1' S0$
$Q2 = S1 S0'$
$Q3 = S1 S0$

- boolean func에 맞게 순서를 잘 정해서 작성해야 함. 그림대로 쓸 것

- decoder block provides abstraction
  - decoder 내부적으로 어떤지 정확히 몰라도 진리표나 방정식 사용 가능
  - diagram 간단하게 만들어줌
  - 재사용이 쉬움

A 3-to-8 decoder (DEC)

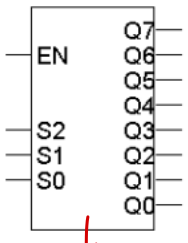
	S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	1	0	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	0
5	1	0	1	0	0	0	0	0	1	0	0
6	1	1	0	0	0	0	0	0	0	1	0
7	1	1	1	0	0	0	0	0	0	0	1

[truth table]

[equation]

$Q0 = S2' S1' S0'$
$Q1 = S2' S1' S0$
$Q2 = S2' S1 S0'$
$Q3 = S2' S1 S0$
$Q4 = S2 S1' S0'$
$Q5 = S2 S1' S0$
$Q6 = S2 S1 S0'$
$Q7 = S2 S1 S0$

[block symbol]



내부에 있는 신호 이름은 다 같아야 함.

decoder → arbitrary functions

minterm generator

decoder는 minterm generator라고도 불림

모든 input 조합에 대해 각각 하나의 output만 true

각 output equation은 모든 input 변수를 포함

⇒ 이런 특성이 decoder의 모든 크기에 적용됨

⇒ arbitrary functions에도 적용 가능

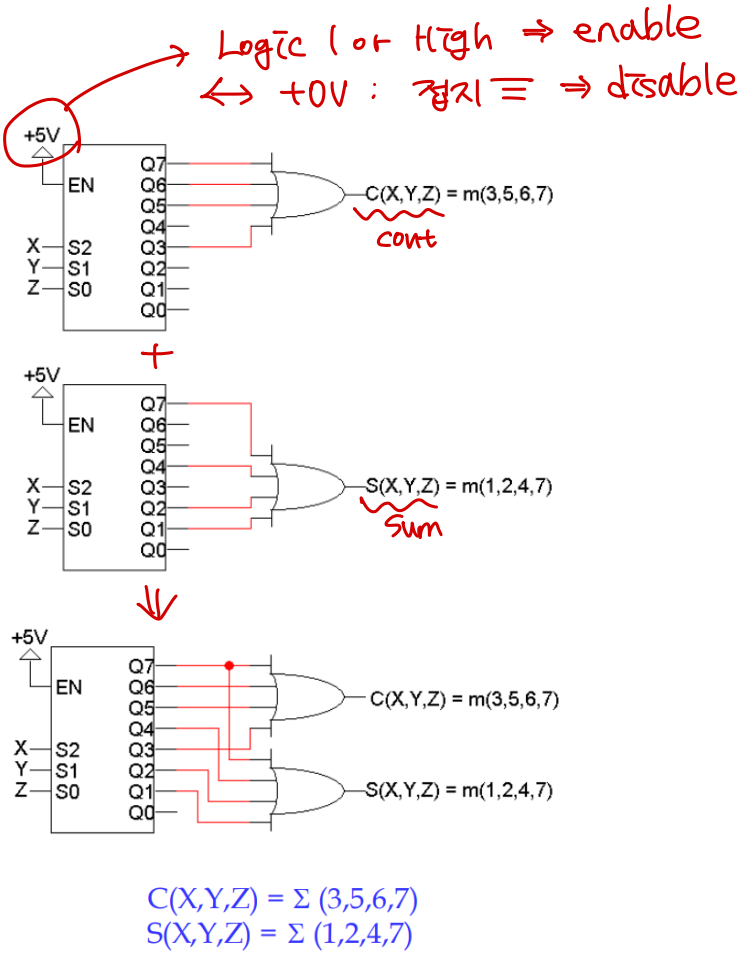
addition

- example
  - 1bit인 X, Y, Z adder

    → output : sum, carry가 존재
  - Z ⇒ carry in이라고 해보자
  - C와 S에 대한 sum of minterms
  - 하나의 decoder로도 표현이 가능함

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

0 + 1 + 1 = 10

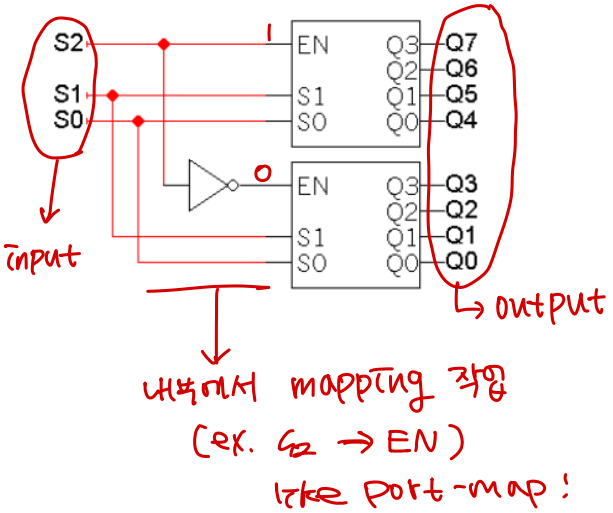


$C(X,Y,Z) = \Sigma\,(3,5,6,7)$
$S(X,Y,Z) = \Sigma\,(1,2,4,7)$

building a 3-to-8 decoder

- 2- to-4 decoder로 표현이 가능 → (2 to 4) x2
- S2 = 0, outputs Q0-Q3 / S2 = 1, outputs Q4-Q7

S2	S1	S0	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
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



Modularity

- 내부 input, output과 외부 input, output을 헷갈리면 xx
  - 내부 : 2 to 4 decoders
  - 외부:3 to 8 decoders
- 2 to 4 decoder equation을 활용하여 이 회로가 3 to 8 decoder인지 알아낼 수 있음

Encoder (ENC) → 범용X, 특이한 경우에 사용

↔ Decoder

a digital circuit performs the "inverse" operation of decoder

- 하나만 "1"인 2의 n승 개의 input에서 n개의 output을 만들어 냄.

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

하나만 1

$z = D_1 + D_3 + D_5 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

$x = D_4 + D_5 + D_6 + D_7$

Ambiguity of Encoder

1. 2개 이상의 input이 1로 들어오면?
   - ↳ solution: 1인 input 중 하나만 encode 하도록 input priority 매기기 ⇒ Priority Encoder
2. 모든 input이 0이면?
   - ↳ solution1 : this output == D0이 1일 때
   - ↳ solution2 : output을 하나 추가하여 1인 input이 하나라도 있는지 indicate

     ex) v=0 → 하나도 안 들어옴 / v=1 → 하나라도 들어옴

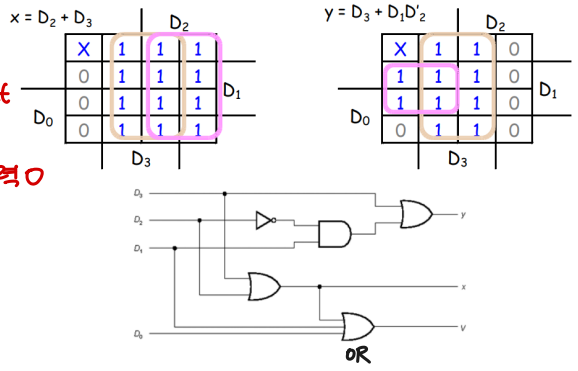
Priority Encoder

만약 여러 개의 1 input이 동시에 들어온다면 높은 우선순위를 가진 input 고르는 decoder

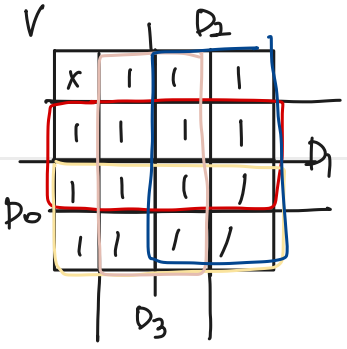
Inputs				Outputs		
LSB D_0	D_1	D_2	MSB D_3	x	y	V detect
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

don't care 하나에 경우의 수 2개씩

모든 경우 0



⇒ 더 상위만 간주하기 → 하위인 애는 don't care



$V = D_0 + D_1 + D_2 + D_3$

Multiplexers

decoder처럼 잘 쓰이는 범용회로

→ 신호를 받아들여서 하나만 output ⇒ (switch, GTB 처럼)

- These serve as examples for circuit analysis and modular design.
- Multiplexers can implement arbitrary functions.

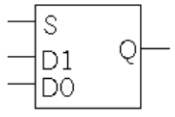
2^n-to-1 multiplexer

- 2^n개의 input 중 하나만 output으로
  - 2^n input
  - n select lines → to pick one of the 2^n data inputs
    ↳ single bit

2-to-1 MUX

- S=0, then D0 is the output (Q=D0)
- S=1, then D1 is the output (Q=D1)
- i번째 input이 true ⇒ Qi가 true

$Q = S' D0 + S D1$



[Block]

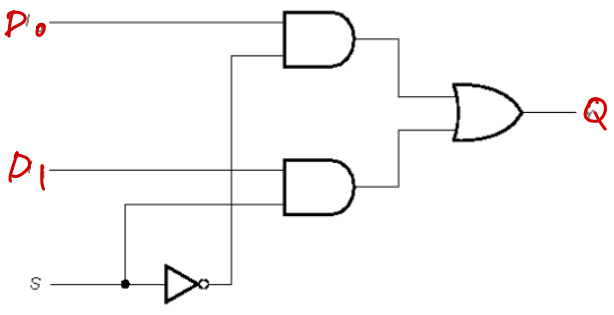
S	D1	D0	Q	
0	0	0	0	
0	0	1	1	D_0
0	1	0	0	
0	1	1	1	D_0
1	0	0	0	
1	0	1	0	
1	1	0	1	D_1
1	1	1	1	D_1

[Truth Table]

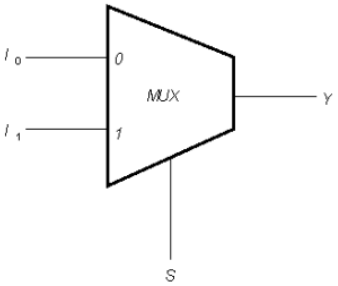
abbrebiated [Truth Table]

S	Q
0	D0
1	D1

- abbrebiated truth table

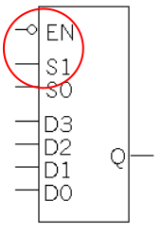


(a) Logic diagram



(b) Block diagram

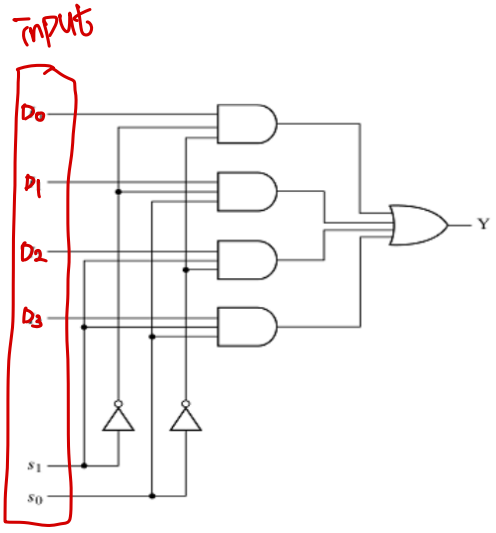
4-to-1 MUX



EN'	S1	S0	Q
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	X	X	0

↳ don't care

$Q = S1' S0' D0 + S1' S0 D1 + S1 S0' D2 + S1 S0 D3$



MUX → Implementing arbitrary functions

each minterm **mi** of the func → connect 1 to MUX data input **Di**

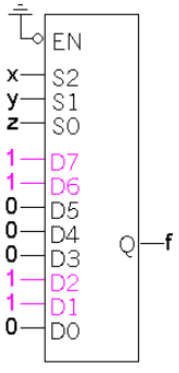
⇒ 즉, function의 input variable과 MUX의 선택된 input 연결

→ particular input combination을 나타낼 때 사용

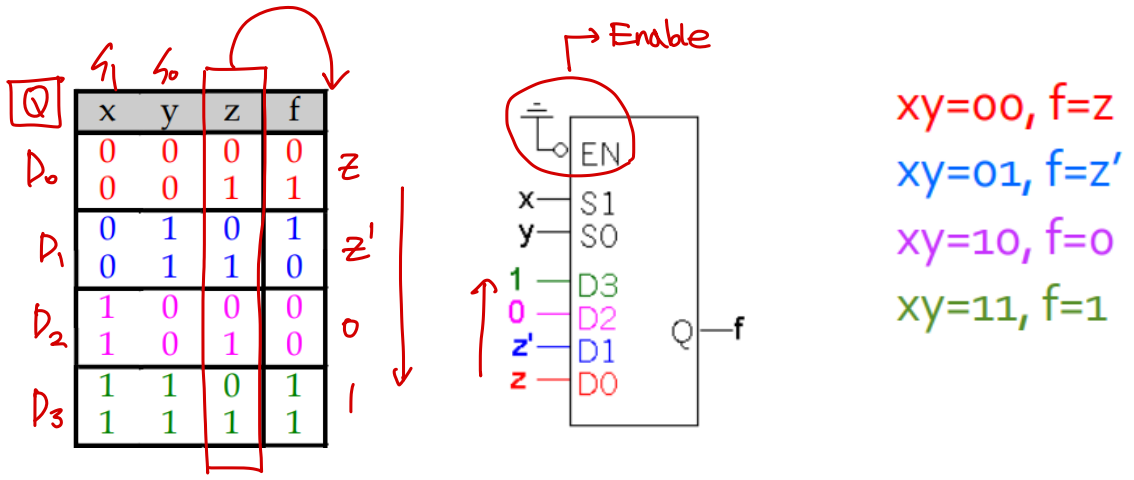
example : 8 to 1 MUX

For example, let's look at $f(x,y,z) = m(1,2,6,7)$.

	x	y	z	f	Q
0	0	0	0	0	D0
1	0	0	1	1	D1
2	0	1	0	1	D2
3	0	1	1	0	D3
4	1	0	0	0	D4
5	1	0	1	0	D5
6	1	1	0	1	D6
7	1	1	1	1	D7



→ **more efficient way : 4 to 1 MUX**

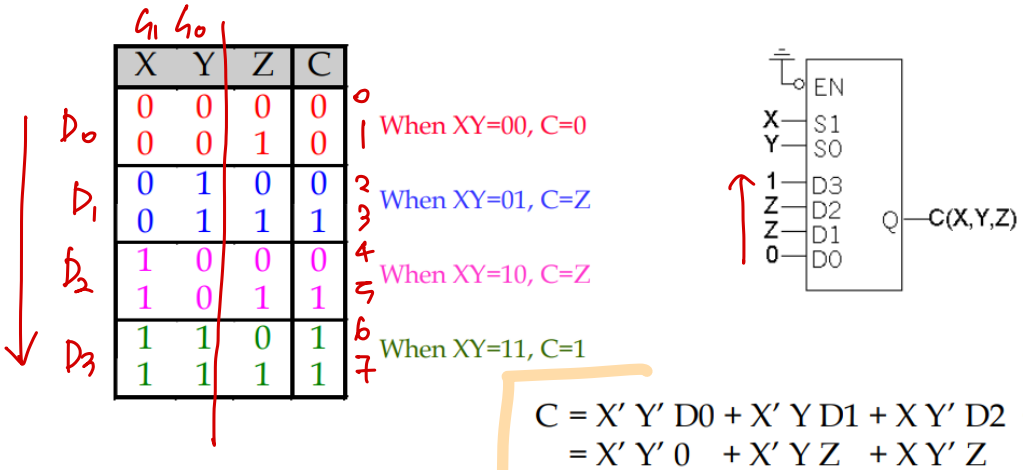


1. truth table for the function and 2개씩 group → 하나의 input으로 표현하기
2. S1, S0에 select한 2개의 input 연결(1번에서 표현한 input 제외한 2개)
3. f(z)에 대한 equations 하나씩 D0~D3 연결

example : MUX-based adder (4 to 1 MUX 2개로 8 to 1 MUX)

carry가 있는 adder function도 가능

- **MUX-based carry(4 to 1 MUX)**



$$C = X'\,Y'\,D0 + X'\,Y\,D1 + X\,Y'\,D2 + X\,Y\,D3$$
$$= X'\,Y'\,0 + X'\,Y\,Z + X\,Y'\,Z + X\,Y\,1$$
$$= X'\,Y\,Z + X\,Y'\,Z + XY$$
$$= \Sigma\,(3,5,6,7)$$

- **MUX-based sum(4 to 1 MUX)**

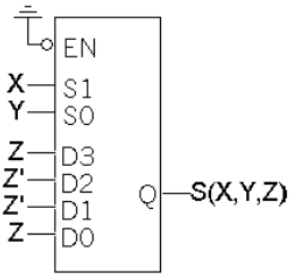
X	Y	Z	S		
0	0	0	0	0	D_0
0	0	1	1	1	
0	1	0	1	2	D_1
0	1	1	0	3	
1	0	0	1	4	D_2
1	0	1	0	5	
1	1	0	0	6	D_3
1	1	1	1	7	

When XY=00, S=Z

When XY=01, S=Z'

When XY=10, S=Z'

When XY=11, S=Z

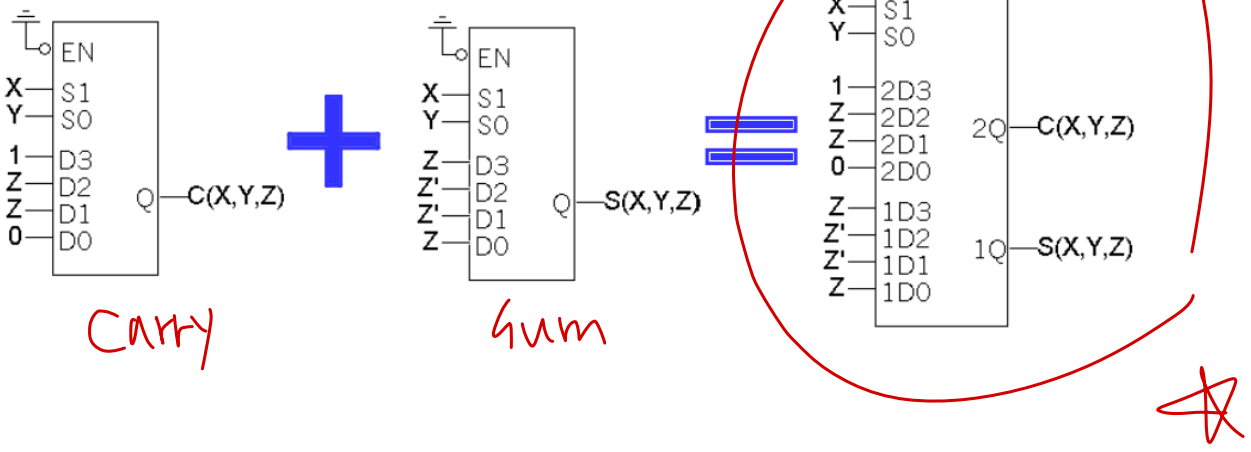


$$S = X'\,Y'\,D0 + X'\,Y\,D1 + X\,Y'\,D2 + X\,Y\,D3$$
$$= X'\,Y'\,Z + X'\,Y\,Z' + X\,Y'\,Z' + X\,Y\,Z$$
$$= \Sigma\,(1,2,4,7)$$

⇒ **dual MUX-based full adder(8 to 1 MUX)**

: MUX-based Carry + MUX-based Sum

two 4-to-1 MUXes



Carry

Sum

2D3 1D3, when S1 So = 11 3

2D2 1D2, when S1 So = 10 2

2D1 1D1, when S1 So = 01 1

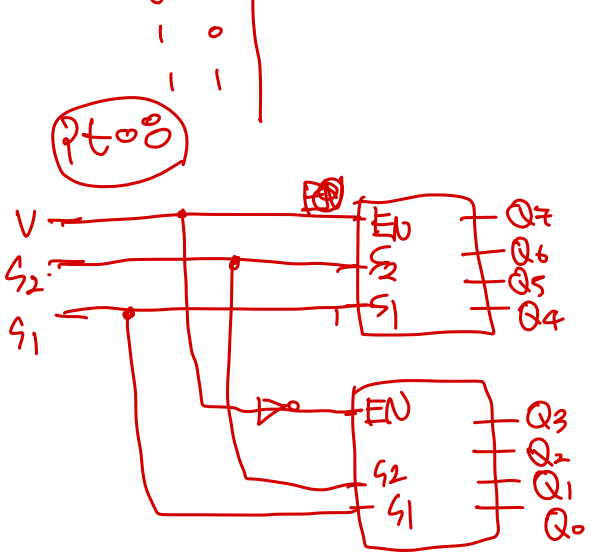
2Do 1Do, when S1 So = 00 0

⇒ k to 1 MUX는 k bit number 3 사용 가능

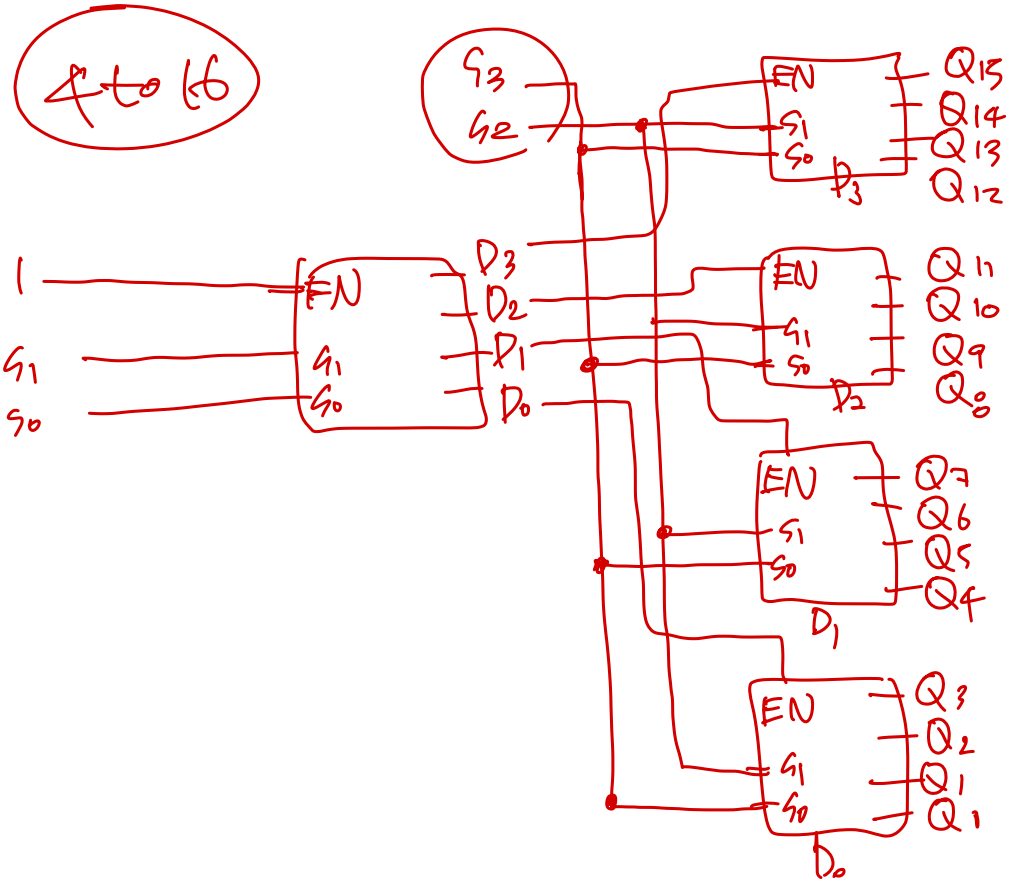
시험 기출

1. 2 to 4 decoder → 4 to 16 decoder

S_1	S_2	D_0	D_1	D_2	D_3
0	0				
0	1				
1	0				
1	1				

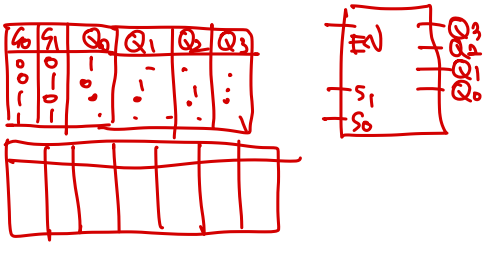


11



Decoder

- n개의 input → 2^n개의 output 중 하나만 1
  - 2 to 4
  - 3 to 8
  - 4 to 16

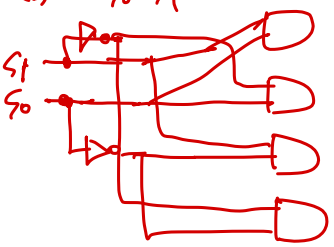


$Q_0 = S_0'S_1'$

$Q_1 = S_0'S_1$

$Q_2 = S_0S_1'$

$Q_3 = S_0S_1$



⇒ adder : OR GATE 필요!

Encoder

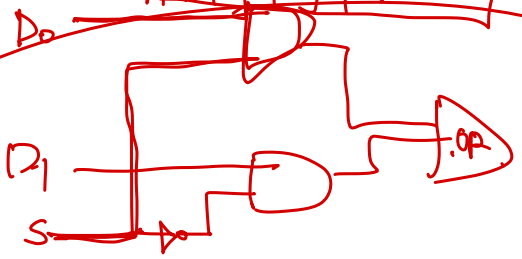
- 2^n개의 input 중 하나만 1 → n개의 output
- Priority Encoder
- output 하나로 = detect.

Multiplexer

- 2^n개의 input → input 중 하나가 output

2 to 1 MUX

S	D_1	D_0	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$Q = S'D_0 + SD_1$

$Q = S_1'S_0'D_0 + S_1'S_0D_1 + S_1S_0'D_2 + S_1S_0D_3$

4 to 1 MUX

EN	S_1	S_0	D_3	D_2	D_1	D_0	Q
0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0
0	0	0	0	0	1	1	1
0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	1	0	1
0	0	1	0	0	1	1	1
0	1	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	0	0	1	1	1
0	1	1	0	0	0	0	0
0	1	1	0	0	0	1	0
0	1	1	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	0	0	0	0	0
1	0	0	0	0	0	1	D_0
1	0	0	0	0	1	0	D_1
1	0	0	0	0	1	1	D_0
1	0	1	0	0	0	0	0
1	0	1	0	0	0	1	D_2
1	0	1	0	0	1	0	D_3
1	0	1	0	0	1	1	D_2
1	1	0	0	0	0	0	0
1	1	0	0	0	0	1	D_0
1	1	0	0	0	1	0	D_1
1	1	0	0	0	1	1	D_0
1	1	1	0	0	0	0	0
1	1	1	0	0	0	1	D_3
1	1	1	0	0	1	0	D_3
1	1	1	0	0	1	1	D_3

간략X.

8 to 1 MUX

8 to 1 MUX

↳ Adder 활용.

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1