디지털논리회로
(Digital Logic Circuit)

- Chapter 5

Chapters 1, 2 & 3에서 공부한 것

Chapter 1 Introduction
컴퓨터에서의 수 체계 (이진수 + alpha)

Chapter 2 Combinational Systems
진리표로 구현하는 combinational systems
스위칭 대수로 표현되는 진리표
게이트로 표현되는 진리표
입력→Combinational systems→출력

Chapter 3 The Karnaugh Map (K-map)
진리표를 map으로 표시 → 단순화, 최소화 → 최적의 시스템 만들기

조합회로 시스템 설계 과정

1 단계: 각 **입력과 출력**을 2진으로 표현하라.

1.5 단계: 필요하면, 문제를 더 작은 부(sub) 문제로 나누어라.

2 단계: 설계 사양을 **진리표 혹은 대수 식**으로 형식화(formalize)해라.

3 단계: 서술을 **간단히** 하라.

4 단계: 설계 **목표와 제약에 근거**하여,

사용 가능한 부품으로 시스템을 **구현**한다.

Chapter 5 Designing Combinational Systems

5.1 Iterative (반복) 시스템 ex) ALUs

5.2 Binary Decoders

5.3 Encoders and Priority Encoders

5.4 Multiplexer (Mux) & Demultiplexers (Demux)

5.5 Three-State Gates

5.6 Gate Arrays – ROM, PLA, PALs..

Chapter 5 Designing Combinational Systems

5.1 Iterative (반복) 시스템 ex) ALUs

5.2 Binary Decoders

5.3 Encoders and Priority Encoders

5.4 Multiplexer (Mux) & Demultiplexers (Demux)

5.5 Tree-State Gates

5.6 Gate Arrays – ROM, PLA, PALs..

Iterative (반복) 시스템이란?

1 bit 덧셈 회로 표현:

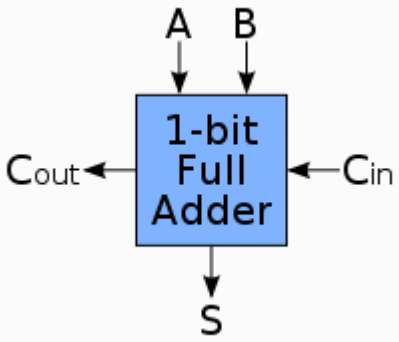
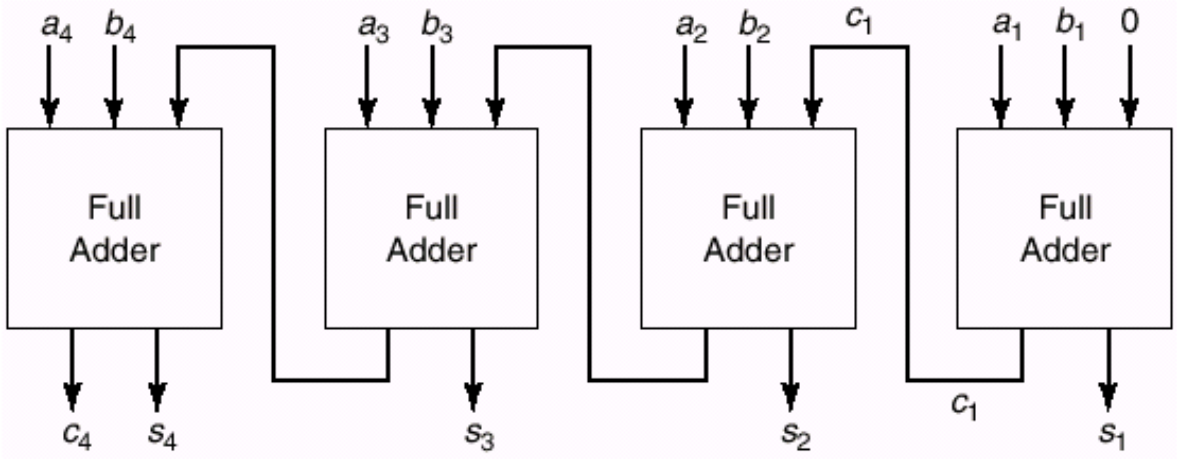


Table 1.5 One-bit adder.

a	*b*	c_{in}	c_{out}	*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

- **작은 블록의 회로를 여러 번 복사하여 시스템 구성**
  - 1비트 full adder 4개의 복사본을 직렬로 연결하여 4비트 adder 구성

Figure 1.2 A 4-bit adder.



Combinational system에서의 *지연*

1 bit 덧셈 회로 표현:

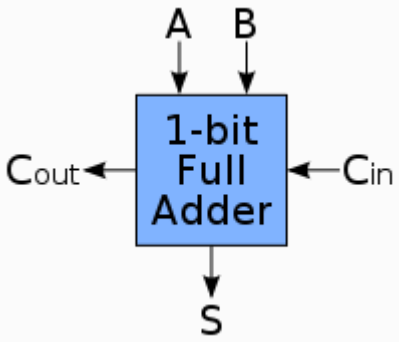
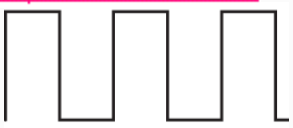
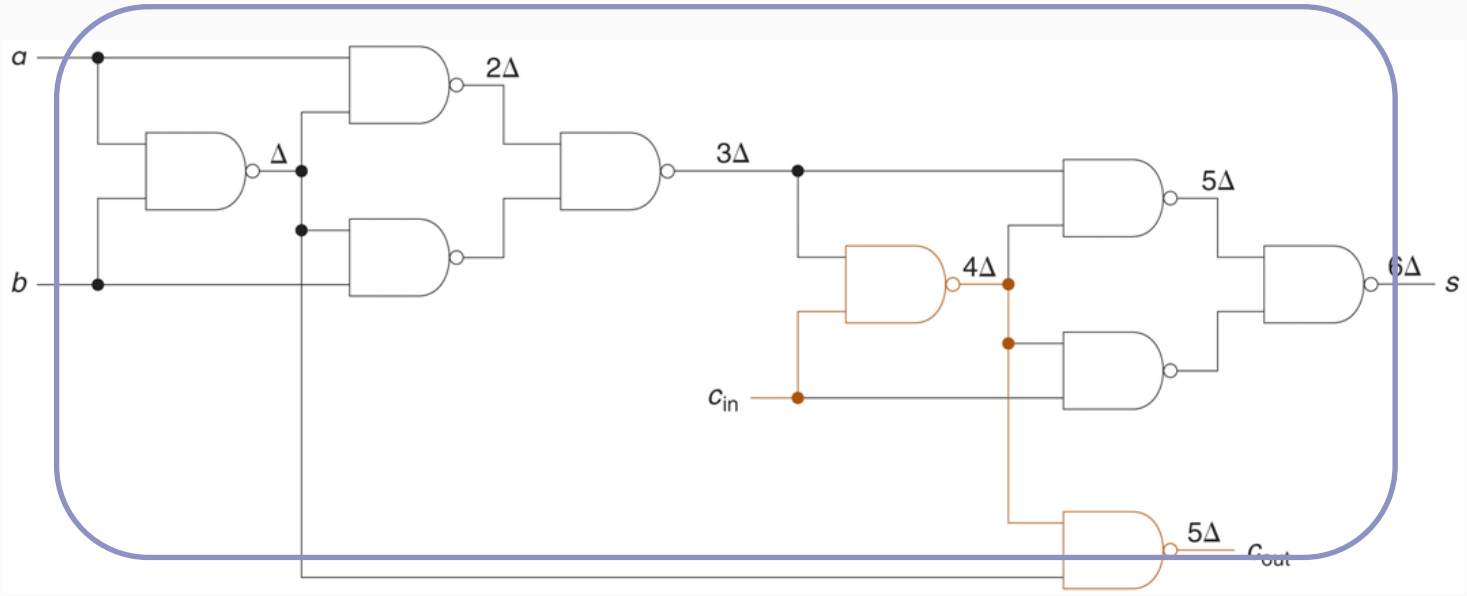


Table 1.5 One-bit adder.

a	*b*	c_{in}	c_{out}	*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

시간의 흐름

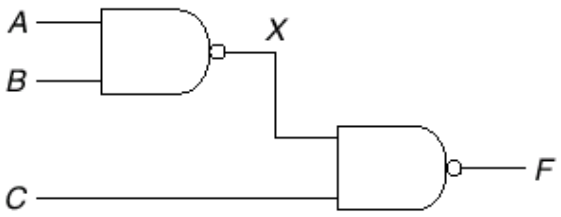




Combinational system에서의 *지연*

- 게이트의 입력 변화시 게이트의 출력은 동시에 변하지 않음

→ 작은 지연 (Δ) 발생

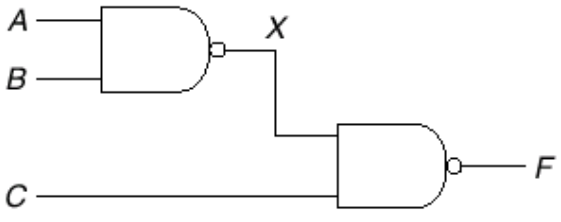


A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

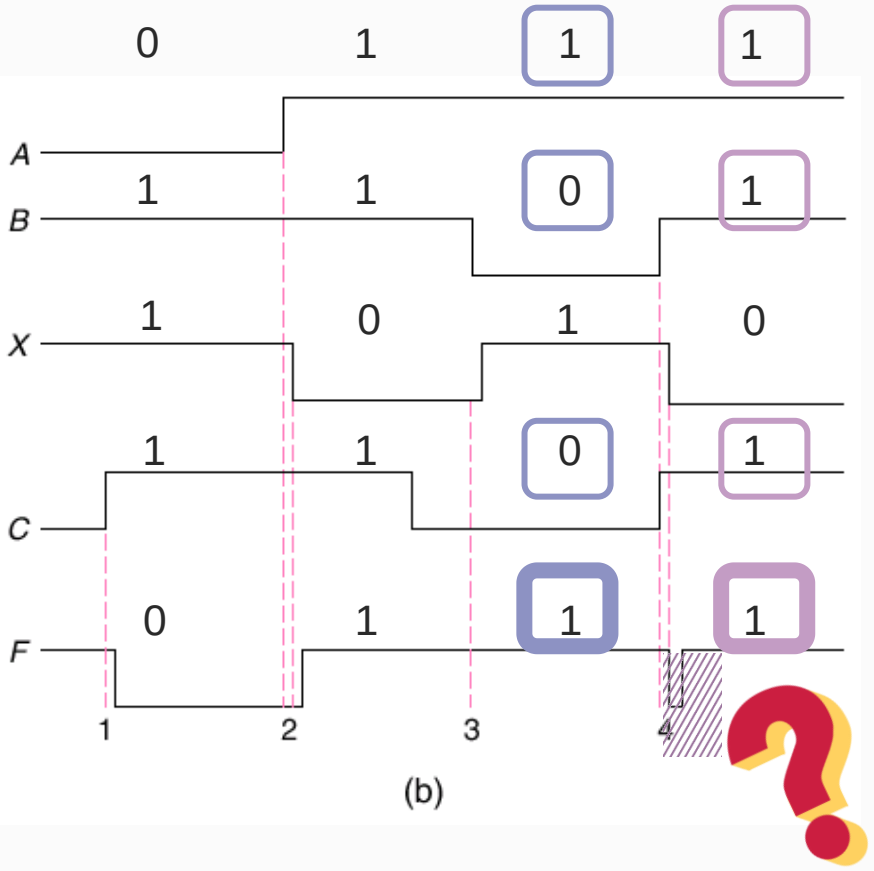
Combinational system에서의 *지연*

- 게이트의 입력 변화시 게이트의 출력은 동시에 변하지 않음

→ 작은 지연 (Δ) 발생



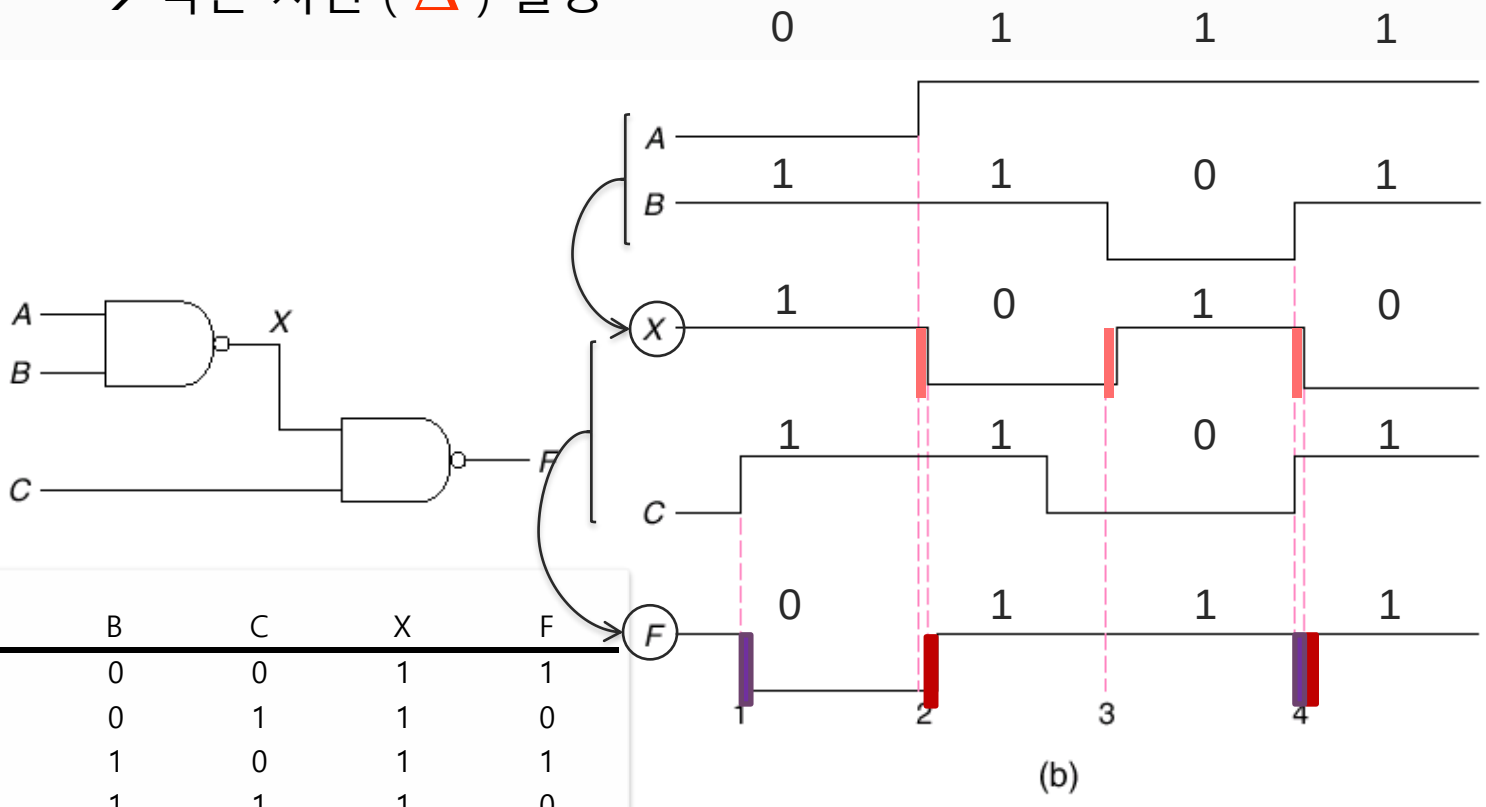
A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



Combinational system에서의 *지연*

- 게이트의 입력 변화시 게이트의 출력은 동시에 변하지 않음

→ 작은 지연 (Δ) 발생



A	B	C	X	F
0	0	0	1	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

A,B 변화 후 X 에 반영되는 지연

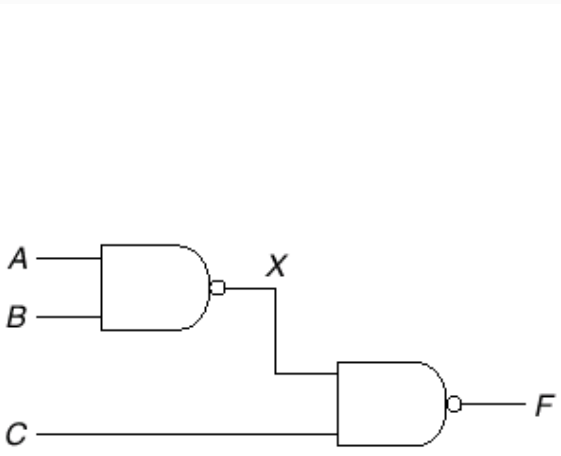
C 변화 후 F 에 반영되는 지연

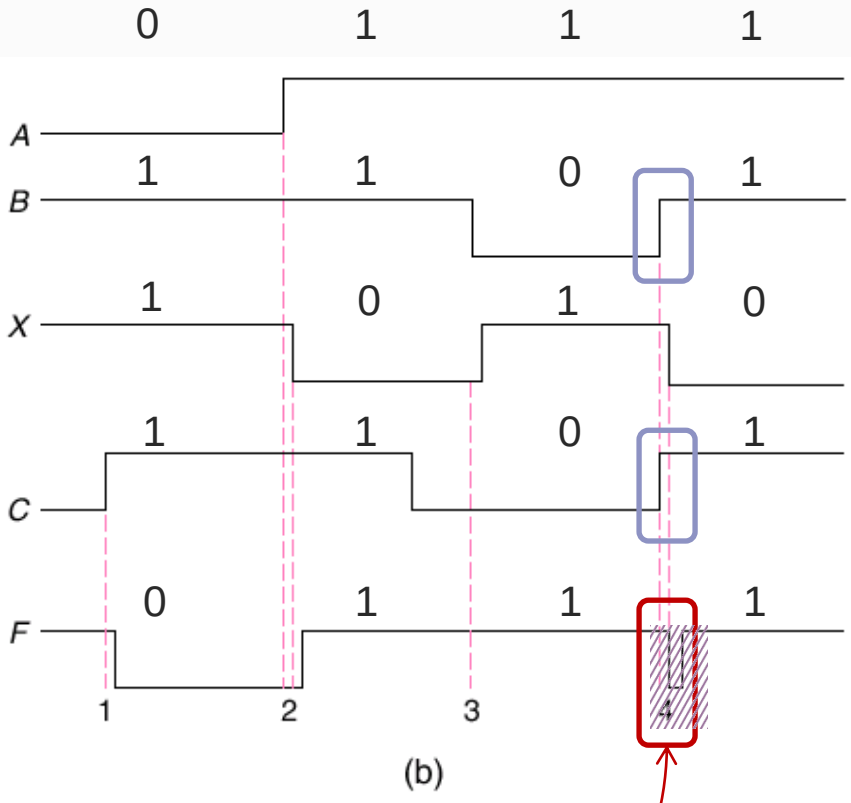
X 변화 후 F 에 반영되는 지연

Combinational system에서의 *지연*

- 게이트의 입력 변화시 게이트의 출력은 동시에 변하지 않음

→ 작은 지연 (Δ) 발생





- 시간 4에서, **B와 C가 동시에 변해서**

→ 해저드(hazard) 또는 글리치(glitch) 상황 발생

1-bit Adder에서의 지연

- 가산기의 덧셈에 걸리는 시간 계산(모든 입력이 동시에 들어간다고 가정)
- 입력 *a*와 *b*가 변할 때 회로의 다양한 지점에서 생기는 지연을 표시

<예제 2.34 참고>

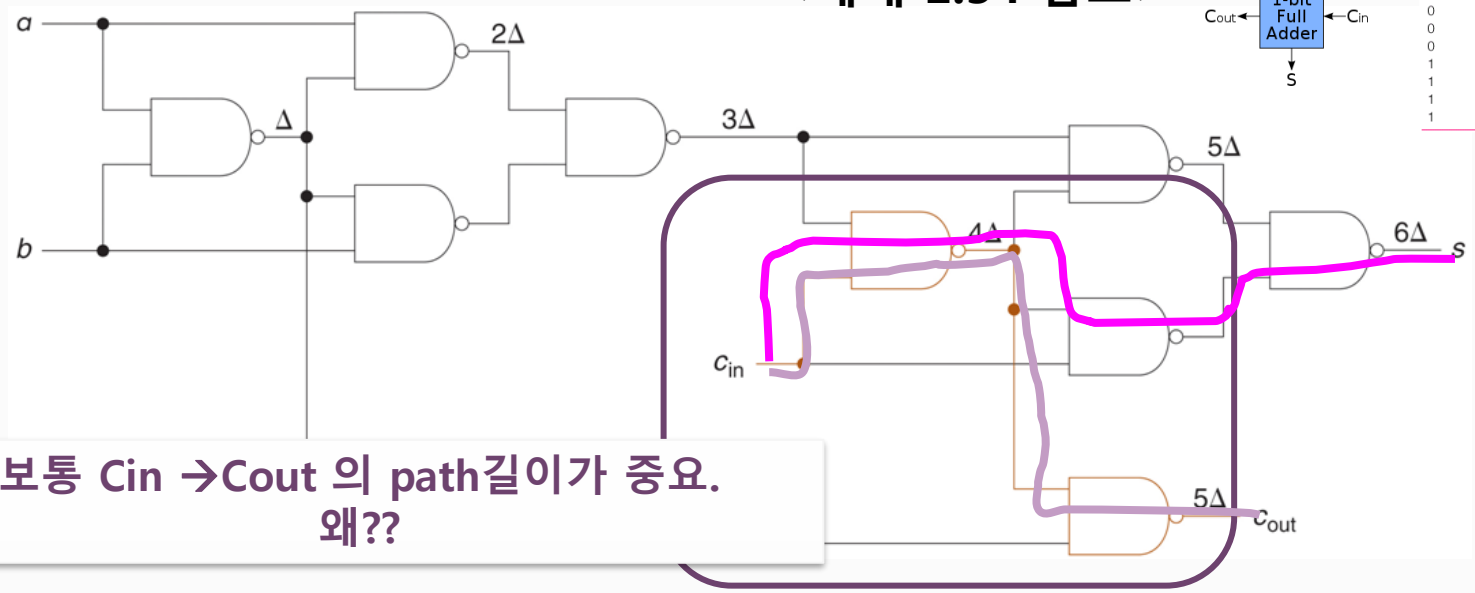


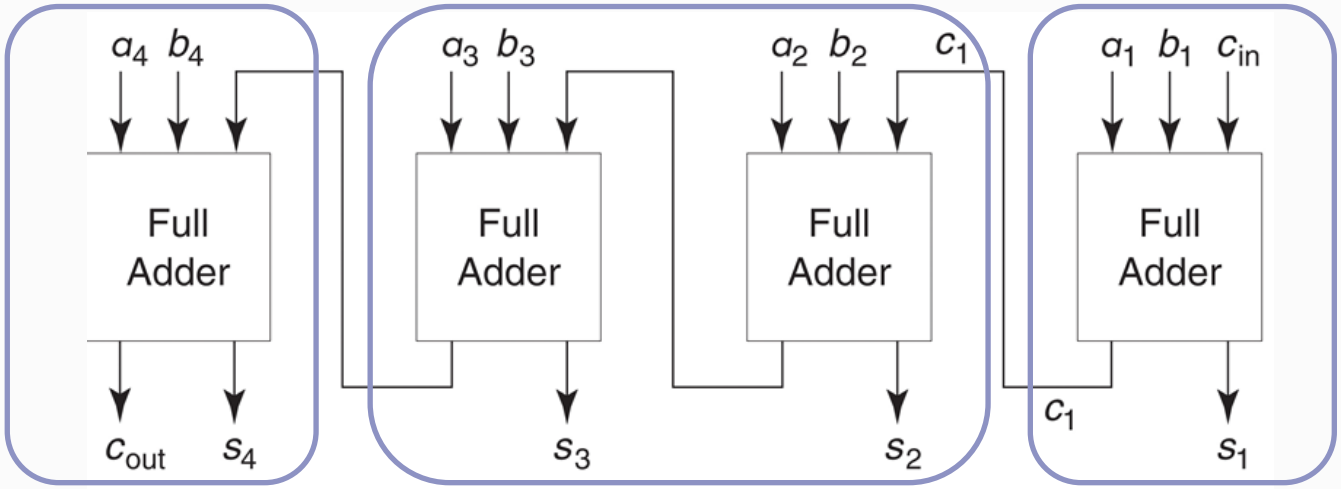
Table 1.5 One-bit adder.

a	*b*	c_{in}	c_{out}	*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

보통 Cin →Cout 의 path길이가 중요.
왜??

- 입력 a 와 b가 변하는 시점에서 **합(s)이 나오는 시점까지 지연 시간은 6△, 캐리 출력 (Cout)이 나오는 시점까지 지연 시간은 5△**
- a, b의 값이 고정되면, Cin 에서 **캐리 출력 (Cout)까지 지연 시간은 단지 2△,** Cin 에서 **합(S)까지 지연은 3△**

N-bit Adder에서의 지연



- n개의 1비트 Adder를 가지고 n비트 Adder 구현시 총 지연 시간 계산

  = (1) + (2) + (3)

주의: **다음 bit Adder 영향을 미치는 Cout 중심**으로 path 계산

(1) LSB의 입력(a1, b1)에서 C_{out}까지의 지연 시간 — (a1, b1 → C_1)

(2) 중간 Adder들의 C_{in} 에서 C_{out}까지의 지연 시간×(n-2) — (C_{in} → C_{out})

(3) MSB의 C_{in} 에서 C_{out} 혹은 C_{in} 에서 s까지의 지연시간 중 긴 것 — (C_{in} → C_{out} or C_{in} → S)

N-bit Adder에서의 지연

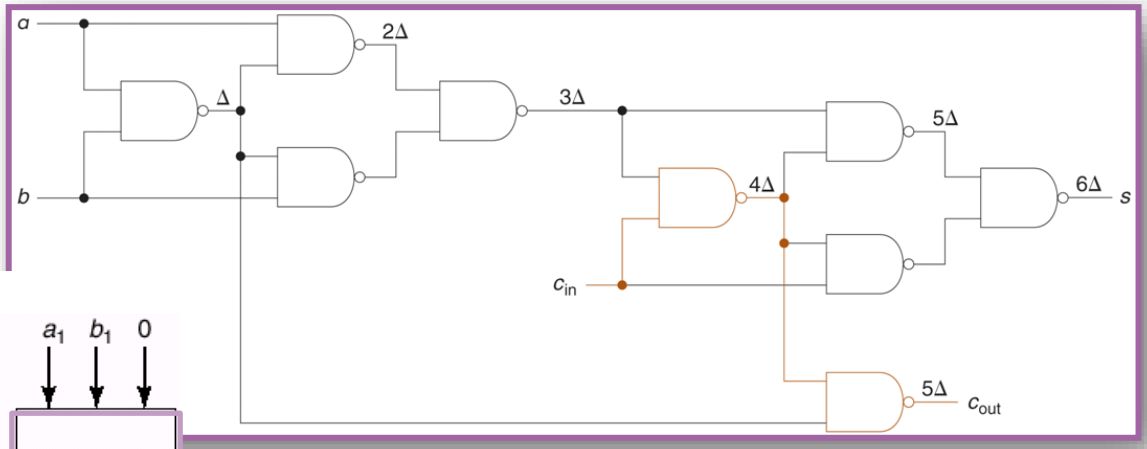
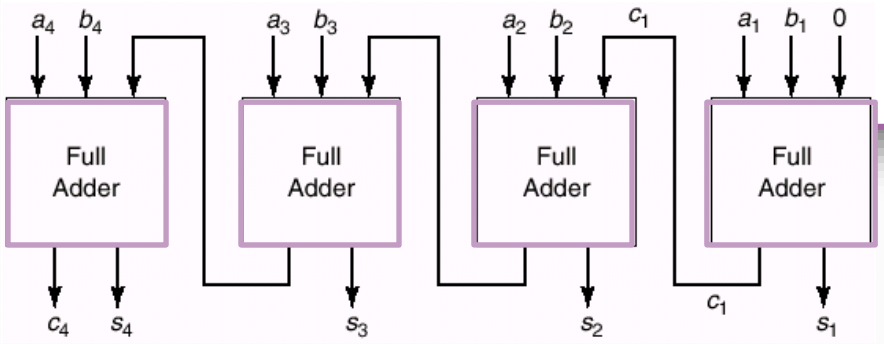


Figure 1.2 A 4-bit adder.



- 예제 2.34의 회로에서 n-bit adder의 지연 시간을 구하면

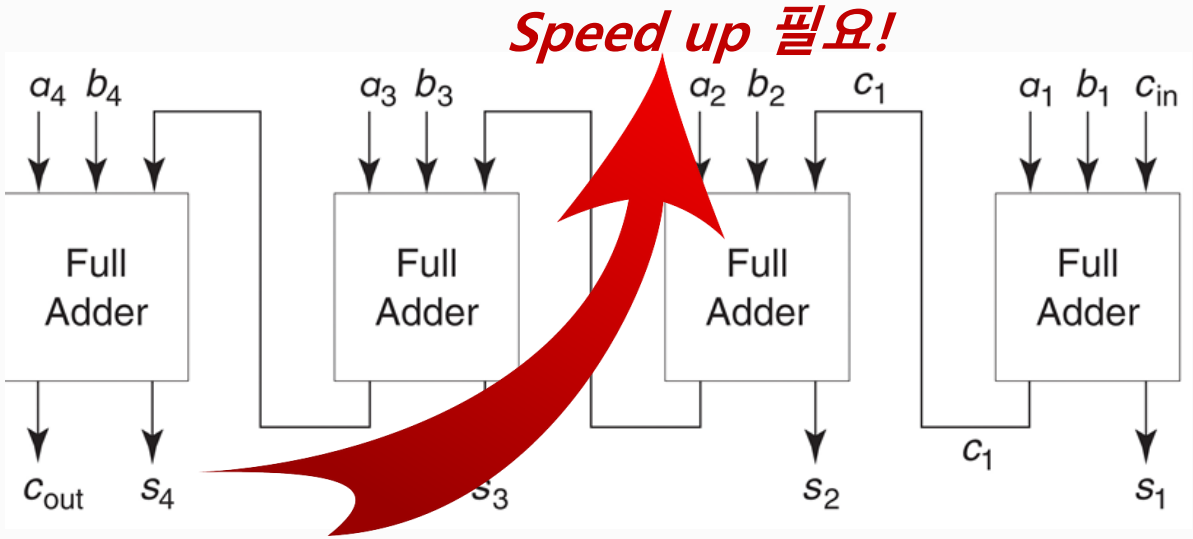
  (1) LSB의 입력에서 Cout까지의 지연 시간 : **5△**

  (2) 중간 가산기들의 Cin 에서 Cout까지의 지연 시간×(n-2) : **2△ ×(n-2)**

  (3) MSB의 Cin 에서 Cout 혹은 Cin 에서 s까지의 지연시간 중 긴 것: **3△**

(1) + (2) + (3) = 5△ + 2(n-2)△ + 3△ = **(2n + 4)△**

N-bit Adder에서의 지연



64bit adder라면?? : 132△! 너무 크다!!

- 예제 2.34의 회로를 사용하면..

  (1) LSB의 입력에서 Cout까지의 지연 시간 : **5△**

  (2) 중간 가산기들의 Cin 에서 Cout까지의 지연 시간×(n-2) : **2△ ×(n-2)**

  (3) MSB의 Cin 에서 Cout 혹은 Cin 에서 s까지의 지연시간 중 긴 것: **3△**

(1) + (2) + (3) = 5△ + 2(n-2)△ + 3△ = **(2n + 4)△**

Chapter 5 Designing Combinational Systems

5.1 Iterative (반복) 시스템 ex) ALUs

- Adder 설계

5.2 Binary Decoders

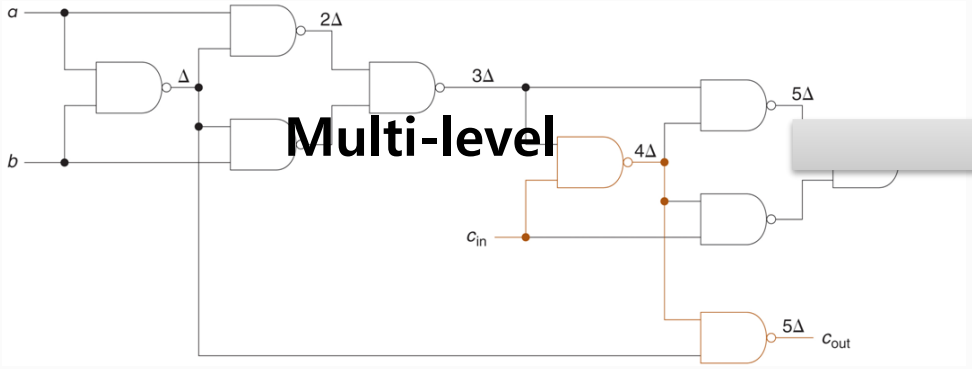
5.3 Encoders and Priority Encoders

5.4 Multiplexer (Mux) & Demultiplexers (Demux)

5.5 Tree-State Gates

5.6 Gate Arrays – ROM, PLA, PALs..

저지연 Adder 설계 (1): *two-level adder*



SOP 활용

Table 1.5 One-bit adder.

a	*b*	c_{in}	c_{out}	*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

- **SOP 형식으로 구현한 1-bit adder의 2-level 회로**

$$C_{out} = aC_{in} + bC_{in} + ab$$

$$s = a'b'C_{in} + a'bC'_{in} + ab'C'_{in} + abC_{in}$$

- 캐리출력 (Cout): 각 입력(a,b,C_{in})이 주어졌을 때 C_{out}까지의 지연은 **2△**
- 합(s): C_{in}이 주어졌을 때 NOT 게이트를 필요로 하므로 s까지의 지연은 **3△**

저지연 Adder 설계 (1): *two-level adder*

- **n개의 1-bit Adder 회로를 이용한 n-bit Adder 구현 시 총 지연 시간 비교**

  (1) LSB의 입력에서 C_{out}까지의 지연 시간

  (2) 중간 가산기들의 C_{in} 에서 C_{out}까지의 지연 시간×(n-2)

  (3) MSB의 C_{in} 에서 C_{out} 혹은 C_{in} 에서 s까지의 지연시간중 긴 것

  => multilevel 가산기 : 5△ + 2(n-2)△ + 3△ = (2n + 4)△

  => 2-level 가산기 : 2△ + 2(n-2)△ + 3△ = (2n + 1)△

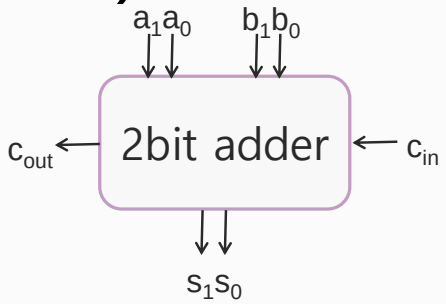
  Ex. 64비트 가산기: 132△(multilevel 가산기)와 129△(2-level 가산기)

- △가 아무리 작다 해도 총 지연 시간이 길어짐.

저지연 Adder 설계 (2): *two-level multi-bit adder*

- **2-bit adder 에 대한 minimum SOP 표현 (2-level 구현)**

$$
\begin{aligned}
c_{out} &= a_1b_1 + a_0b_0a_1 + a_0b_0b_1 + c_{in}b_0b_1 + c_{in}b_0a_1 + c_{in}a_0b_1 \\
&\quad + c_{in}a_0a_1 \\
s_1 &= a_0b_0a_1'b_1' + a_0b_0a_1b_1 + c_{in}'a_0'a_1'b_1 + c_{in}'a_0'a_1b_1' + c_{in}'b_0'a_1'b_1 \\
&\quad + c_{in}'b_0'a_1b_1' + a_0'b_0'a_1b_1' + a_0'b_0'a_1'b_1 + c_{in}b_0a_1'b_1' \\
&\quad + c_{in}b_0a_1b_1 + c_{in}a_0a_1'b_1' + c_{in}a_0a_1b_1 \\
s_0 &= c_{in}'a_0'b_0 + c_{in}'a_0b_0' + c_{in}a_0'b_0' + c_{in}a_0b_0
\end{aligned}
$$



a_1	a_0	b_1	b_0	c_{in}	s_1	s_0	c_{out}

- 캐리출력 (Cout): 각 입력(*a0, b0, a1, b1, C_{in}*)에서 C_{out}까지의 지연은 **2△**

- 합(s0, s1): C_{in}에 대한 NOT 게이트를 필요로 함. 따라서, **3△**

- 총 지연 시간은 $2\triangle + 2(n/2-2)\triangle + 3\triangle = (n+1)\triangle$

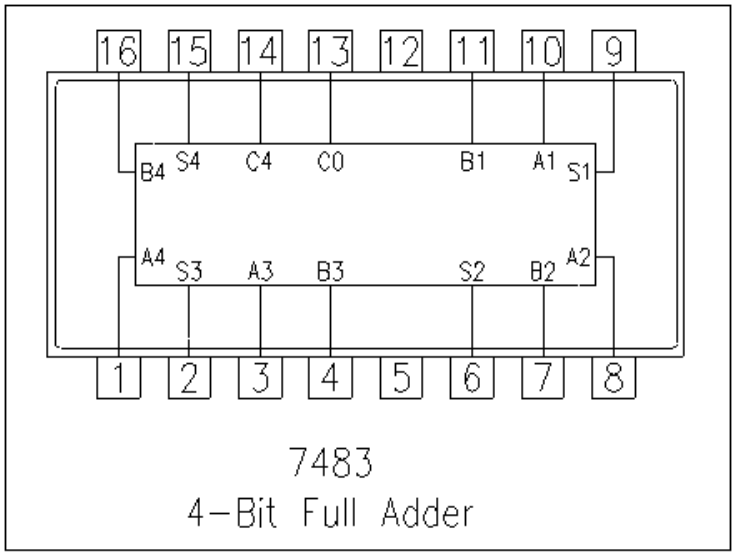
=> 2-level 가산기 : $2\triangle + 2(n-2)\triangle + 3\triangle = (2n+1)\triangle$

=> multilevel 가산기 : $5\triangle + 2(n-2)\triangle + 3\triangle = (2n+4)\triangle$

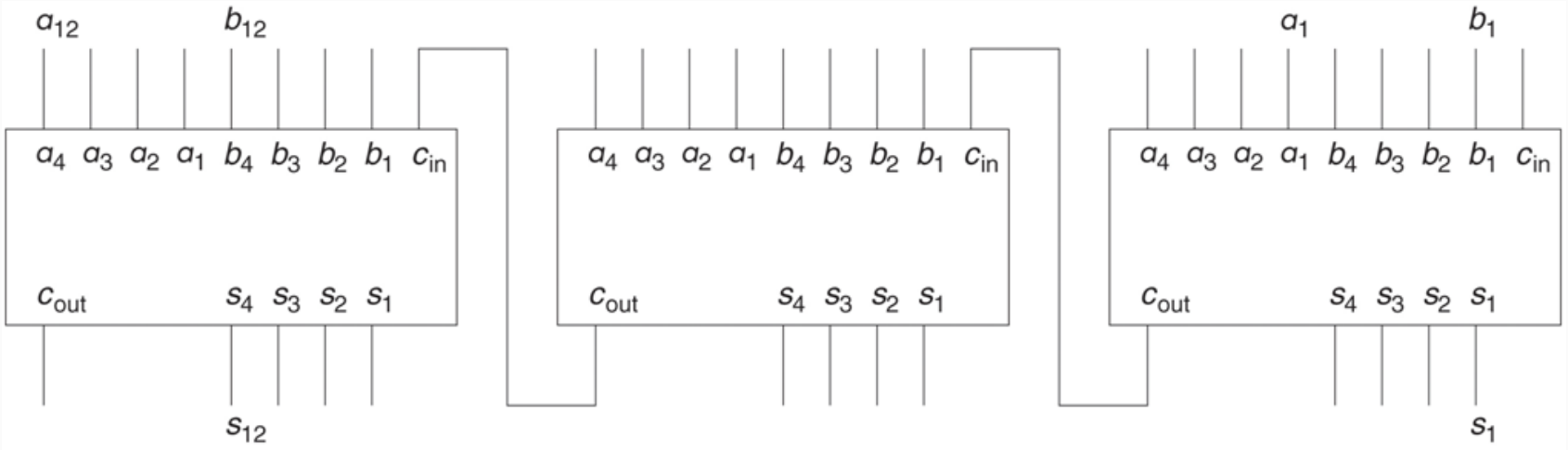
거의 절반!

저지연 Adder 설계 (2): *two-level multi-bit adder*

- 4-bit adder : 7483, 74283

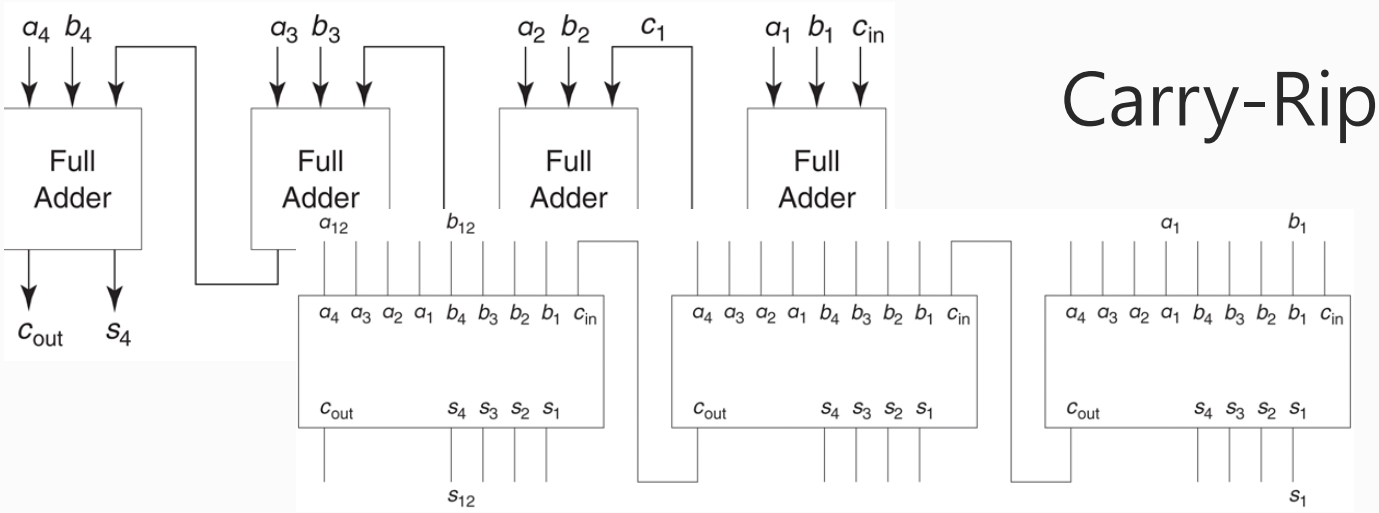


- 12-bit adder by cascading 4-bit adders



저지연 Adder 설계 (3): *carry-look-ahead adder*

문제는 Carry!



Carry-Ripple Adder

➔ **Carry-Look-Ahead (CLA) Adder**

저지연 Adder 설계 (3): *carry-look-ahead adder*

- **좋은 설계를 위한 관찰 – Carry에 초점을 맞추어서 살펴 보니..**

1) 입력 a와 b가 모두 1일 때는 Cin과 상관 없이 Cout이 항상 1: ab
2) 입력 a와 b가 모두 0일 때는 Cin과 상관 없이 Cout이 항상 0
3) 입력 a, b 중 둘 중 하나가 1일 때는 Cin 값으로 Cout 결정 $C_{in}(a + b)$

Table 1.5 One-bit adder.

a	b	c_{in}	c_{out}	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

$$C_{out} = aC_{in} + bC_{in} + ab$$

저지연 Adder 설계 (3): *carry-look-ahead adder*

- **좋은 설계를 위한 관찰 – Carry에 초점을 맞추어서 살펴 보니..**

1) 입력 a와 b가 모두 1일 때는 Cin과 상관 없이 Cout이 항상 1: ab
2) 입력 a와 b가 모두 0일 때는 Cin과 상관 없이 Cout이 항상 0
3) 입력 a, b 중 둘 중 하나가 1일 때는 Cin 값으로 Cout 결정 $C_{in}(a + b)$

- g (generate) = ab //Cout이 1로 **'생성'**되는 경우
- p (propagate) = a + b //Cin이 1일 때 Cout이 1이 되어 **'전파'**되는 경우

C_{out} 을 g와 p로 표현하면,

$C_{out} = \underbrace{aC_{in} + bC_{in}}_{\text{전파되거나}} + \underbrace{ab}_{\text{생성되면 } C_{out}=1} \rightarrow g + pC_{in}$

저지연 Adder 설계 (3): *carry-look-ahead adder*

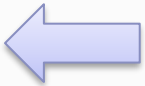
Table 1.5 One-bit adder.

a	b	c_{in}	c_{out}	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

- multi-bit adder의 각 자리수(i)에서 두 개의 신호를 만들어 냄
  - G_i(generate) = $a_i b_i$
  - P_i(propagate) = $a_i + b_i$

$C_{out} = g + pC_{in}$

- $c_{i+1} = a_i b_i + a_i c_i + b_i c_i$
  $= a_i b_i + (a_i + b_i) c_i$
  $= G_i + P_i c_i$



Carry-ripple

- $c_1 = G_0 + P_0 c_0$
- $c_2 = G_1 + P_1 c_1$
  $= G_1 + P_1 (G_0 + P_0 c_0)$
  $= G_1 + P_1 G_0 + P_1 P_0 c_0$

Carry-look-ahead

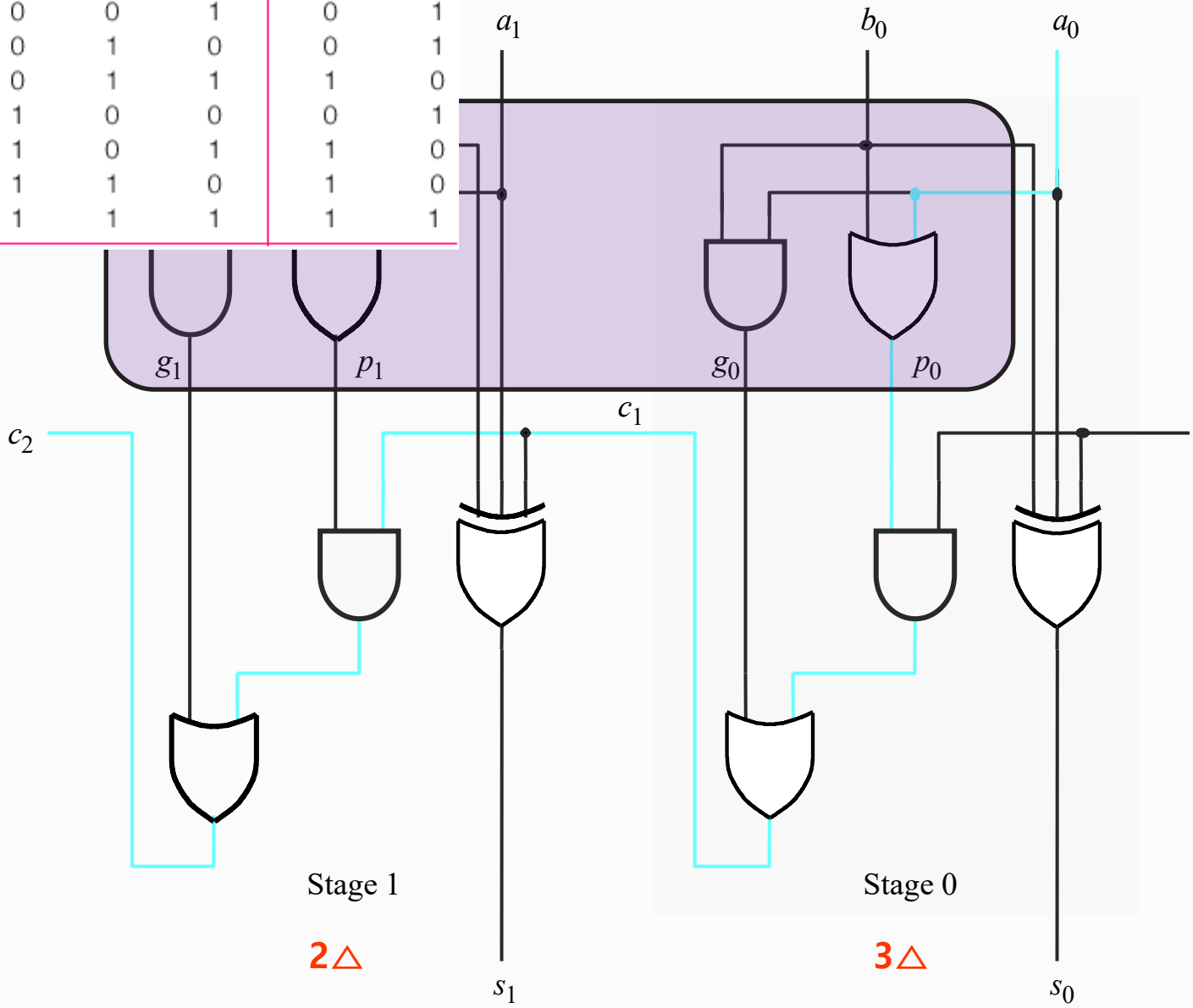
- $c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + P_i P_{i-1} P_{i-2} G_{i-3} + \ldots + P_i P_{i-1} P_{i-2} \ldots P_2 P_1 c_0$

C_{in}이 아니라 a, b 입력으로 C_{out}을 계산할 수 있다!
즉, 더 이상 아래 자리 bit의 덧셈이 끝나길 기다리지 않아도 된다.

CR Adder 회로 분석

Table 1.5 One-bit adder.

a	b	c_{in}	c_{out}	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

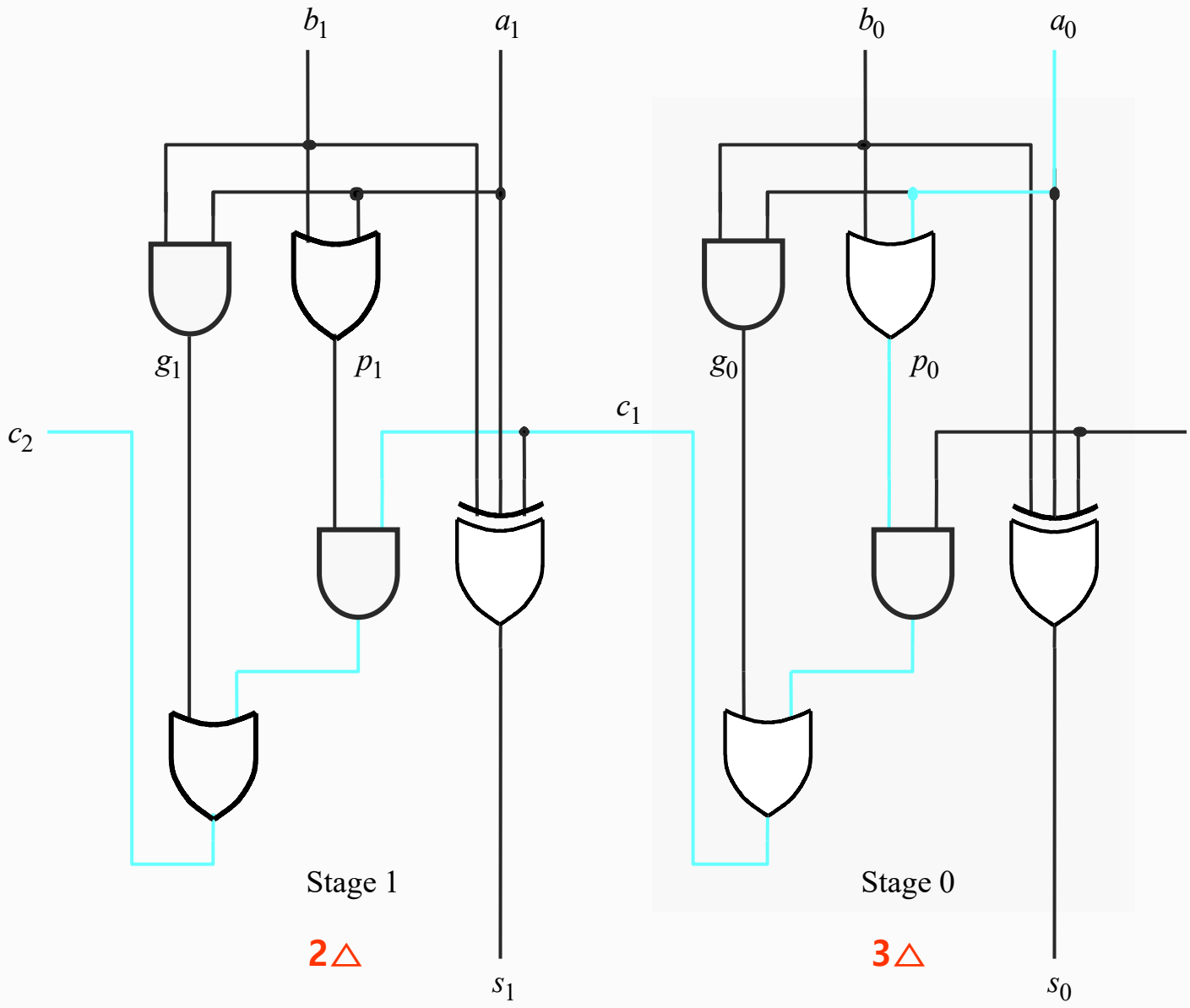


- g = ab
- p = a + b

$c_2 = G_1 + P_1 c_1$

Q1. Sum 생성 로직 설명해 보기

<Carry-ripple>



- g = ab
- p = a + b

$c_2 = G_1 + P_1 c_1$

Q1. Sum 생성 로직 설명해 보기

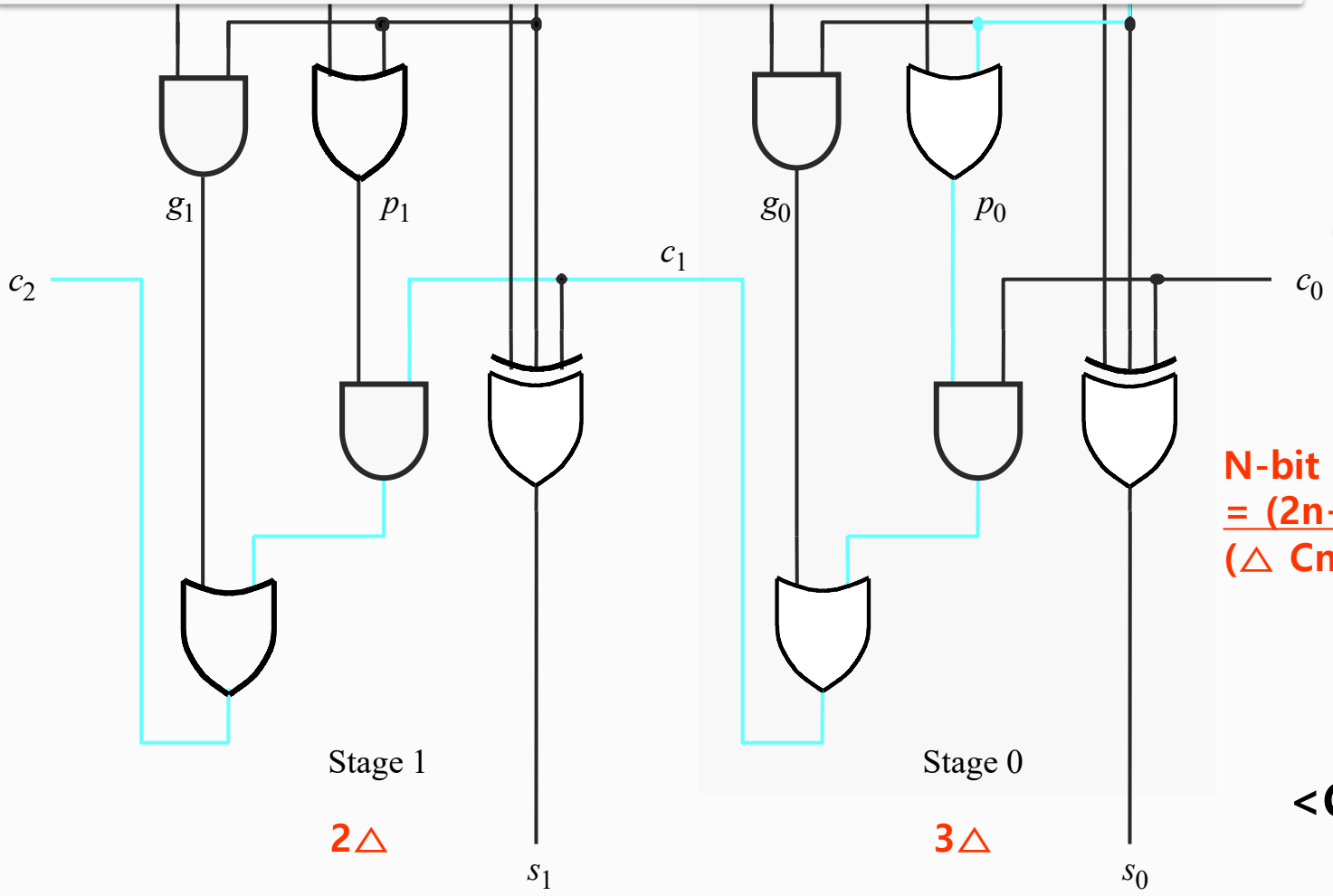
Q2. Cout Delay 가 Stage 0과 1이 다른 이유?

<Carry-ripple>

(1) LSB의 입력(a1, b1)에서 C_{out}까지의 지연 시간

(2) (2) 중간 Adder들의 C_{in} 에서 C_{out}까지의 지연 시간×(n-2)

(3) (3) MSB의 C_{in} 에서 C_{out} 혹은 C_{in} 에서 s까지의 지연시간 중 긴 것



- g = ab
- p = a + b

$c_2 = G_1 + P_1 c_1$

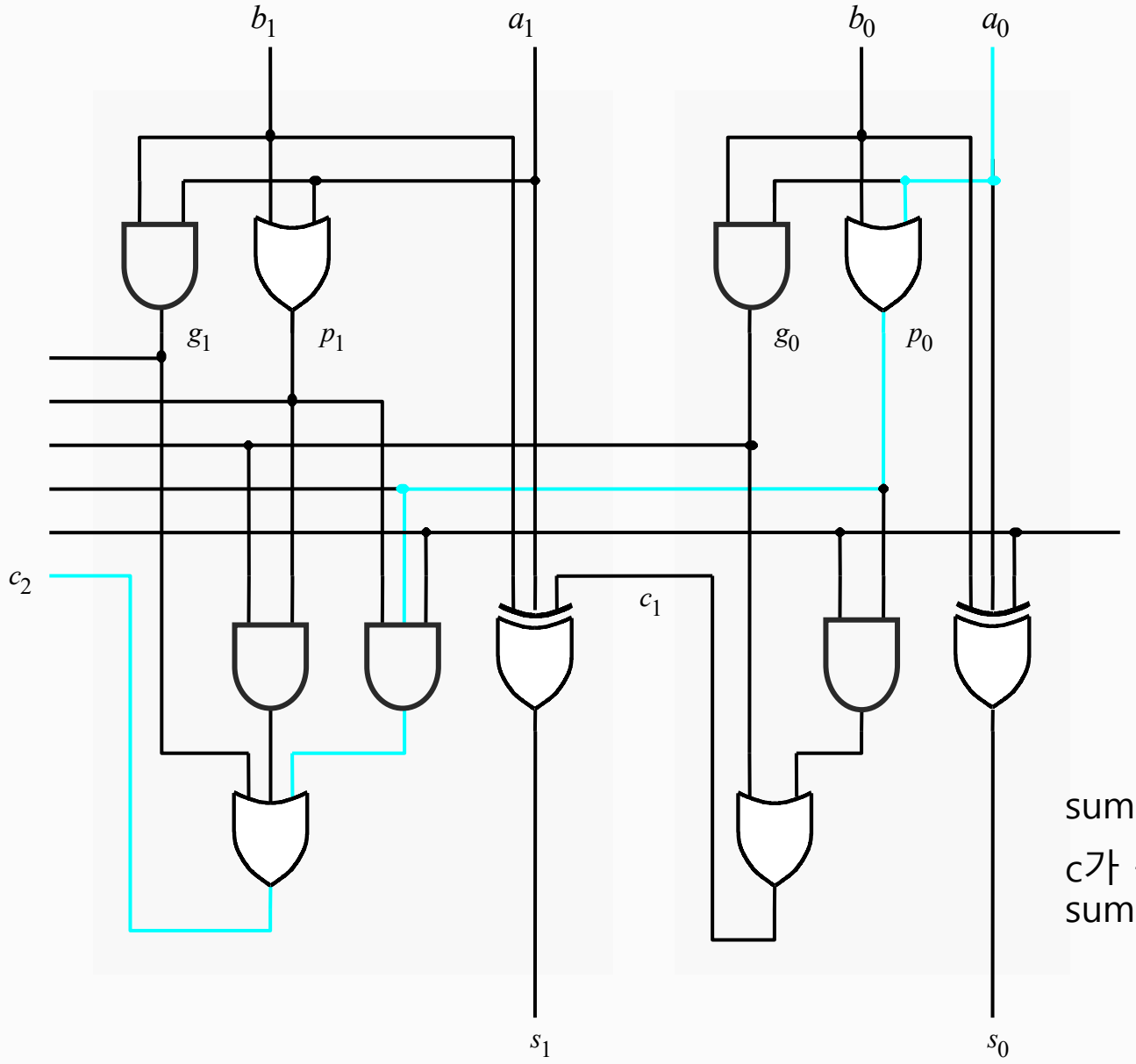
C_n = (2n+1)△

설명해 볼 사람?

N-bit adder를 위한 총 지연
= (2n+1)△
(△ Cn > △ Sn 이므로)

<Carry-ripple>

CLA Adder 회로 분석



- g = ab
- p = a + b

c_2 =

$G_1 + P_1 G_0 + P_1 P_0 c_0$

C_n = 3△

설명해 볼 사람?

N-bit adder를 위한 총 지연
= 3△ + 1△
= 4△ //always!!

sum을 위한 추가 XOR

c가 준비되면 XOR 로 바로 모든 sum값을 알아낼 수 있음

<Carry-look-ahead>

Chapter 5 Designing Combinational Systems

5.1 Iterative (반복) 시스템 ex) ALUs

- Adder 설계

- Subtractor/Adder 설계

5.2 Binary Decoders

5.3 Encoders and Priority Encoders

5.4 Multiplexer (Mux) & Demultiplexers (Demux)

5.5 Tree-State Gates

5.6 Gate Arrays – ROM, PLA, PALs..

2진수 뺄셈

- 뺄셈: 2의 보수를 취하여 더한다.

  a-b = a+(-b)

예제 1.18 : 7 - 5

5:	0101	7:	0111
	1010	-5:	+1011
+	1	2	(1) 0010
-5:	1011		

2의 보수 (2's complement)이용 음수 만드는 법

음수 만드는 법

1. 크기에 해당하는 2 진수 값을 찾는다.
2. 각 비트에 대한 보수를 취한다.
3. 1을 더한다.

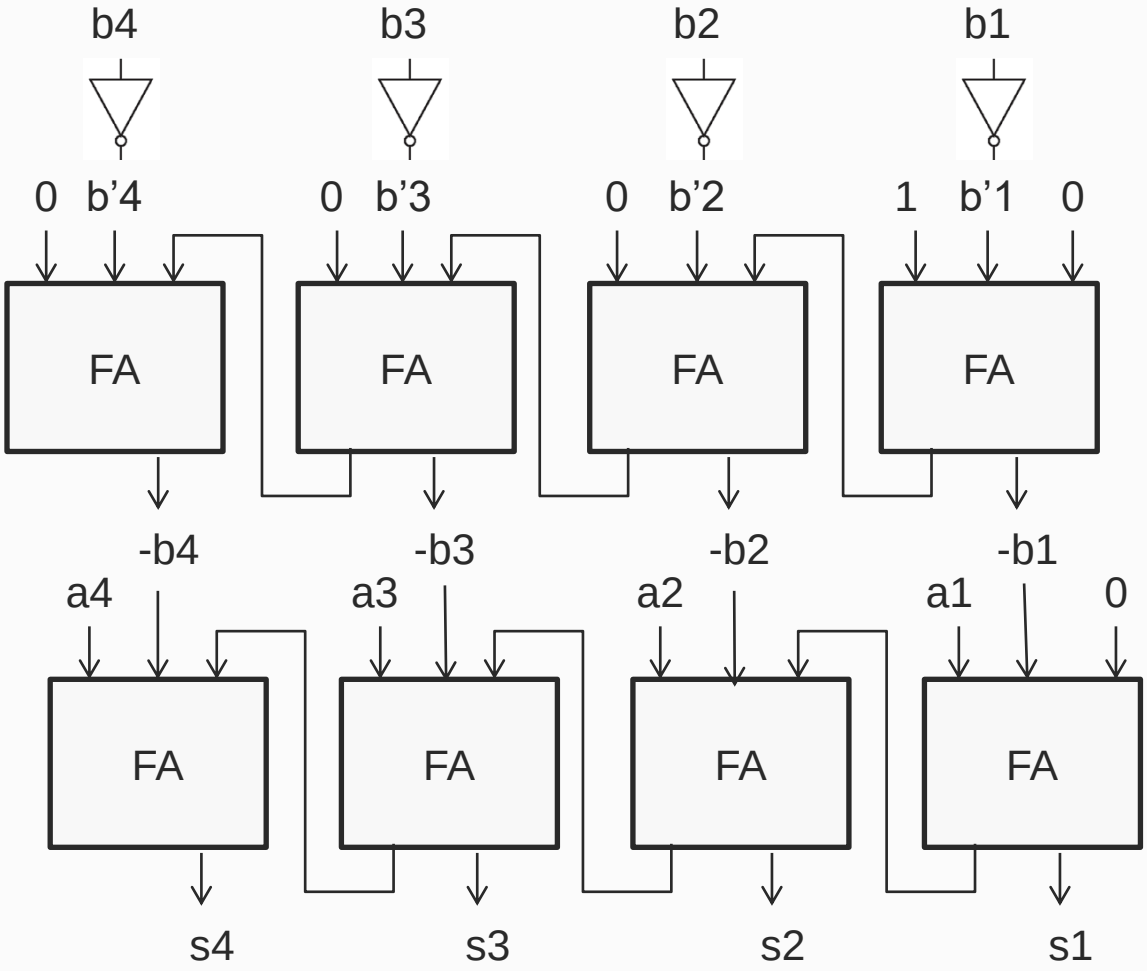
예제 1.13

	-5?	**-1?**	**-0?**
1.	5: 0101	1: 0001	0: 0000
2.	1010	1110	1111
3.	+1	+1	+1
	-5: 1011	-1: 1111	0000

Subtractor 설계 1

음수 만드는 법
1. 크기에 해당하는 2 진수 값을 찾는다.
2. 각 비트에 대한 보수를 취한다.
3. 1을 더한다.

4bit subtractor :
a+(-b)



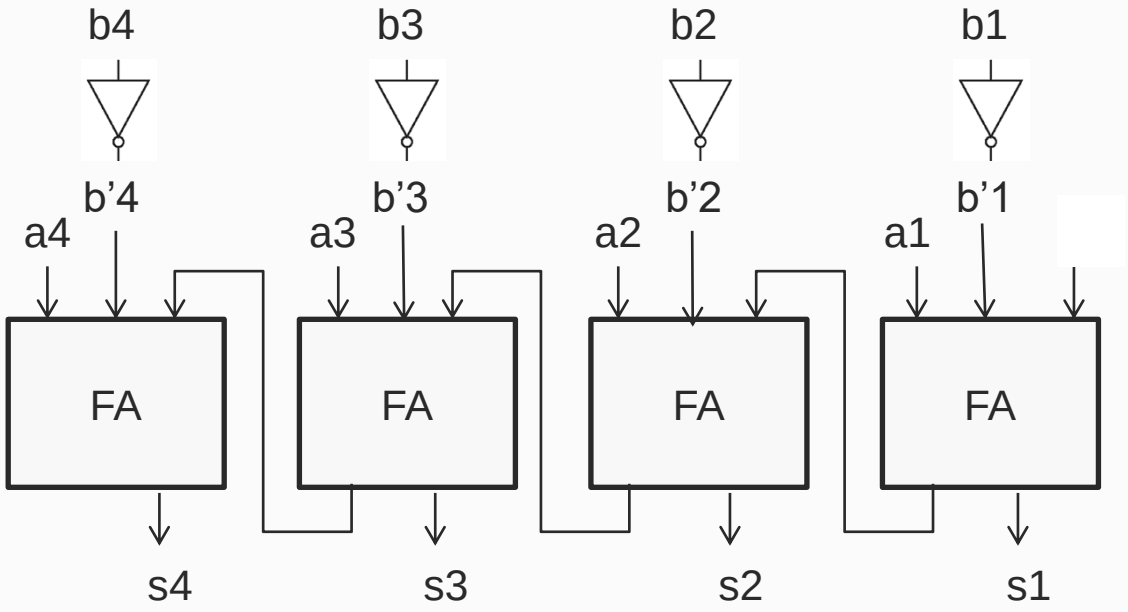
더 좋은 솔루션 찾아보자.

+1을 하기 위해서
복잡한 4개의 FA를
꼭 써야만 하나?

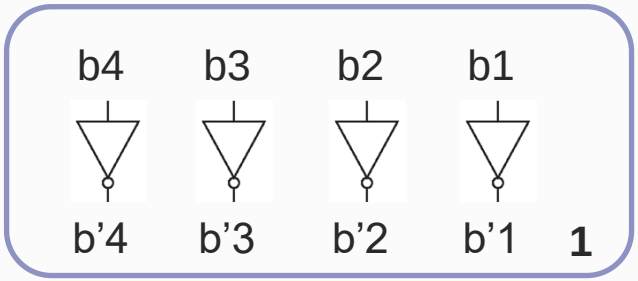
Subtractor 설계 2

4bit subtractor :
a+(-b)

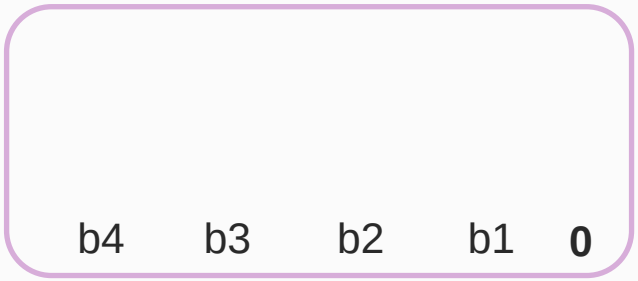
음수 만들기 위한
Adder를 삭제하고,
1은 어디서 더하지?



Adder/Subtractor 설계



Subtractor 입력

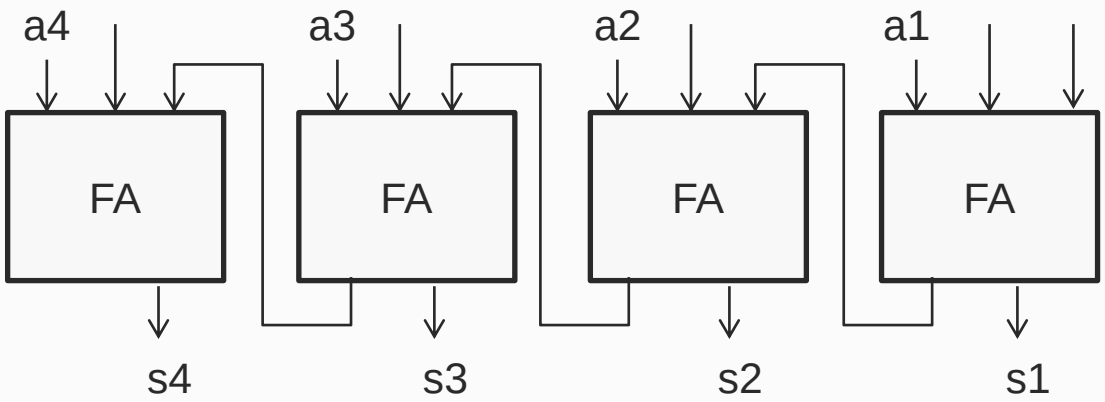


Adder 입력

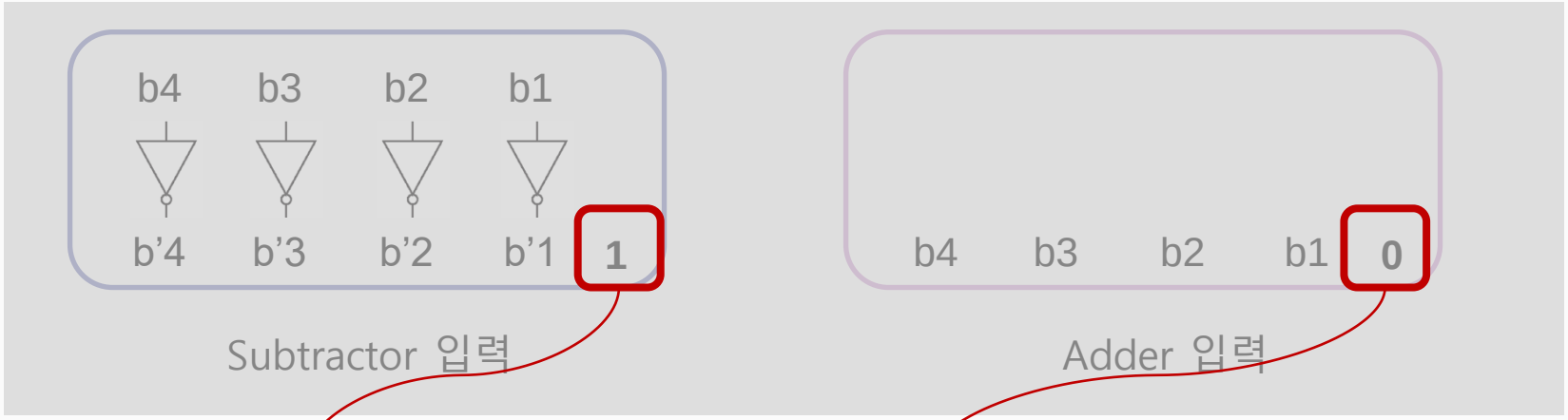
두 가지 로직을 손쉽게 컨트롤 할 수 있는 방법?

1) 동일한 로직

2) 컨트롤 신호만 다를 것



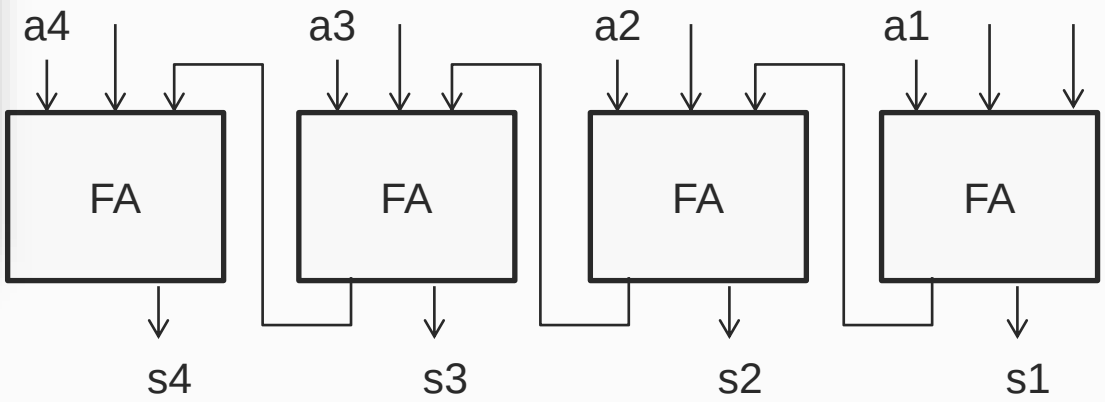
Adder/Subtractor 설계



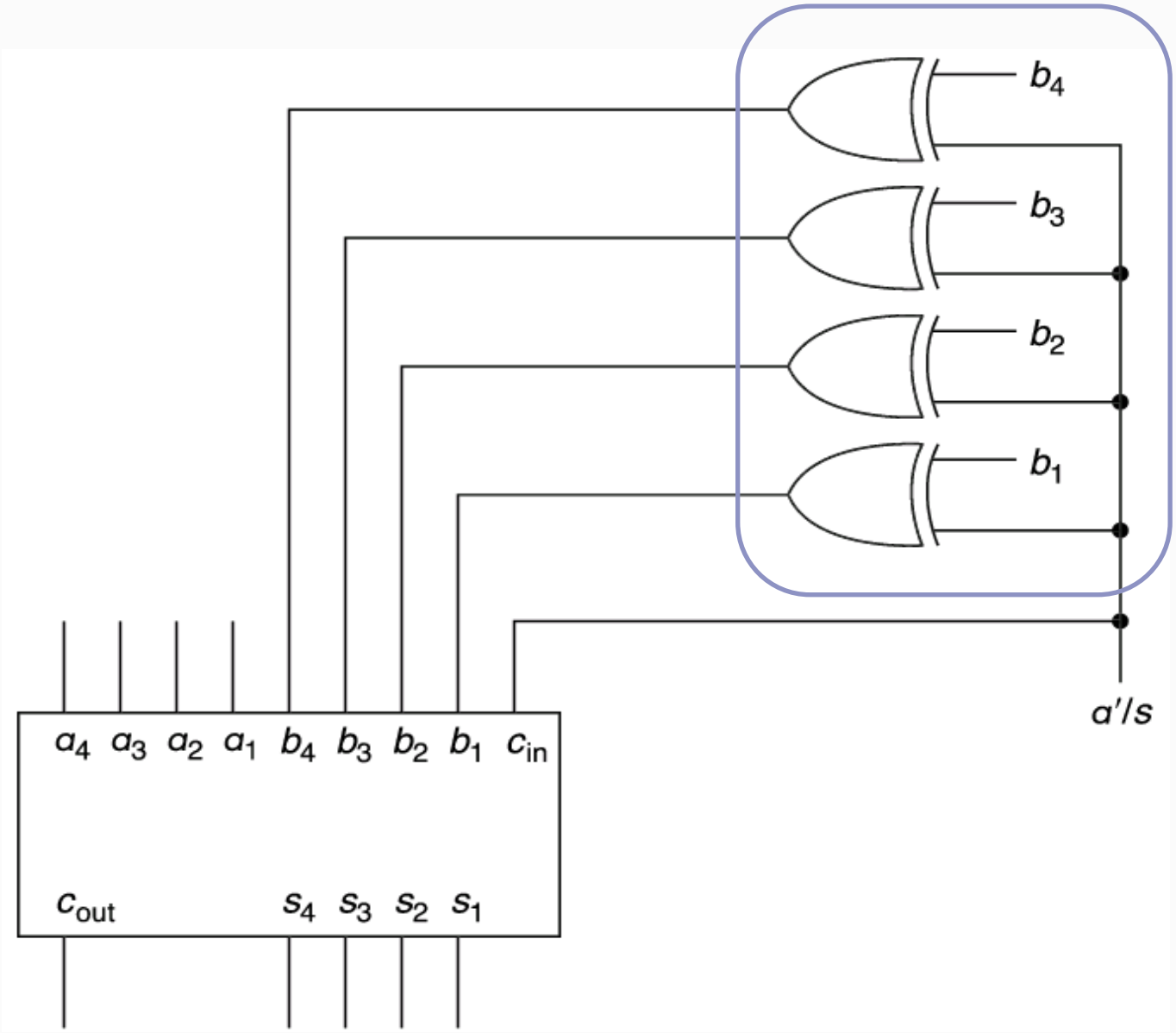
$1 \oplus X = X'$ $0 \oplus X = X$

Exclusive-OR gate

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



Adder/Subtractor 설계



Adder/Subtractor
를 위한
공용로직

하나의 컨트롤 신호
Adder : 0
Subtractor: 1