2024 가을학기
Team 5
20190741 김지수, 20170281 이동욱

컴퓨터공학실험2 발표

: 10주차 MSI/LSI 연산회로

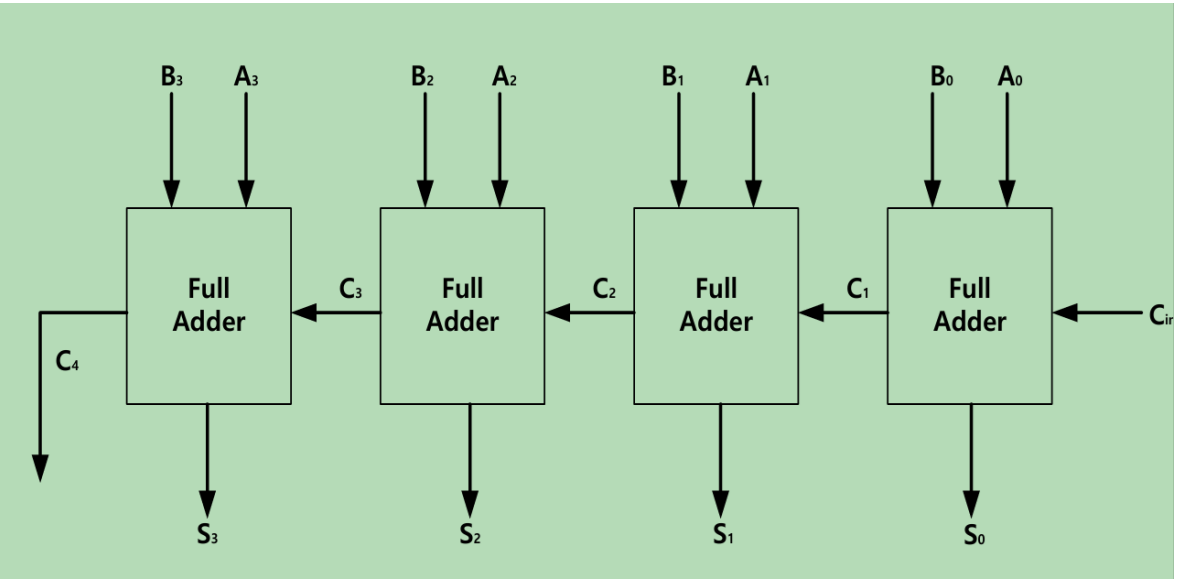
4-bit binary Parallel Adder (4bit 이진 병렬 가산기)

1) 2개의 4-bit 2진수를 가산하기 위한 논리 회로
2) 4개의 1-bit Full Adder(전가산기)가 병렬로 연결되어 있는 구조
3) 두 개의 4비트 이진수와 초기 Carry(C_0)는 가산기에서 0으로 설정

INPUT			OUTPUT	
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

<1비트 전가산기의 진리표>

4-bit binary Parallel Adder (4bit 이진 병렬 가산기)



$$S = (A \oplus B) \oplus C_{in}$$

$$C_{out} = C_{in}(A \oplus B) + AB$$

1. 각 자리의 계산

첫 번째 자리(LSB) = S0 = A0 ⊕ B0 ⊕ Cin

두 번째 자리 = S1 = A1 ⊕ B1 ⊕ C1

세 번째 자리 = S2 = A2 ⊕ B2 ⊕ C2

네 번째 자리(MSB) = S3 = A3 ⊕ B3 ⊕ C3

2. 입력과 출력

입력: 2개의 4비트 이진수 A와 B, 그리고 C0

출력: 4개의 비트 S3, S2, S1, S0로 표현되는 합(sum),
최종 자리 올림 C4 (Overflow)

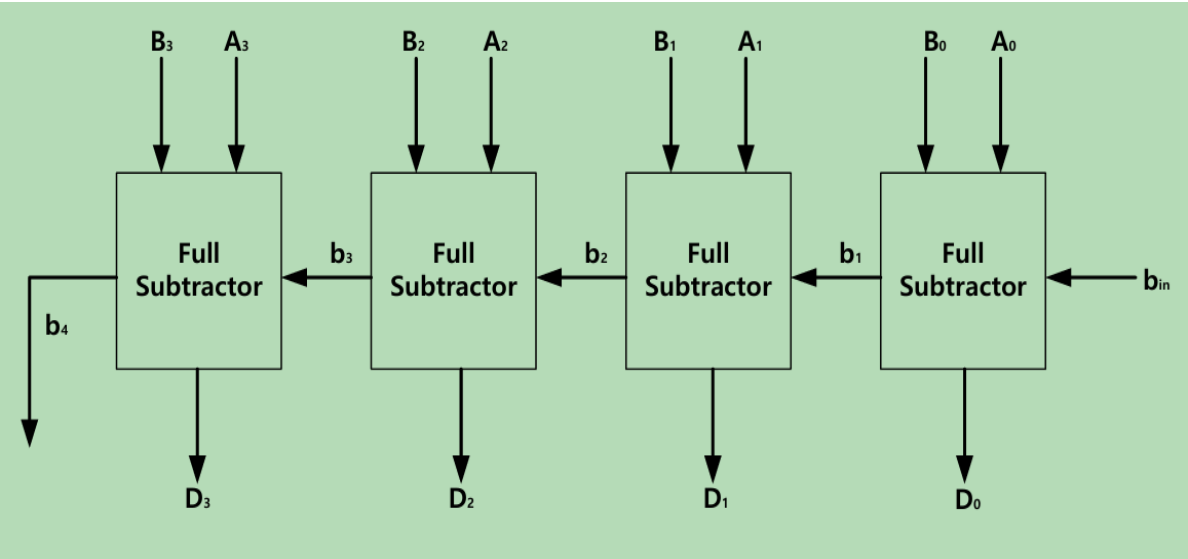
4-bit binary Parallel Subtractor (4bit 이진 병렬 감산기)

1) 2개의 4-bit 2진수를 감산하기 위한 논리 회로
2) 4개의 1-bit Full Subtractor(전감산기)가 병렬로 연결되어 있는 구조
3) 두 개의 4비트 이진수와 초기 Borrow(B_0)는 감산기에서 0으로 설정

INPUT			OUTPUT	
A	B	b_{n-1}	D	b_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

<1비트 전감산기의 진리표>

4-bit binary Parallel Subtractor (4bit 이진 병렬 감산기)



$$A_n \oplus B_n \oplus b_{n-1}$$

$$b_n = (A_n \oplus B_n)' \cdot b_{n-1} + A_n' \cdot B_n$$

1. 각 자리의 계산

첫 번째 자리(LSB) = D0 = A0 ⊕ B0 ⊕ bin

두 번째 자리 = D1 = A1 ⊕ B1 ⊕ b1

세 번째 자리 = D2 = A2 ⊕ B2 ⊕ b2

네 번째 자리(MSB) = D3 = A3 ⊕ B3 ⊕ b3

2. 입력과 출력

입력: 2개의 4비트 이진수 A와 B, 그리고 bin

출력: 4개의 비트 D3, D2, D1, D0로 표현되는 차(diff),
최종 자리 내림 b4 (Underflow)

4-bit binary Parallel Adder / Subtractor 특징

✓ **연산에서 발생한 Carry나 Borrow를 다음 연산으로 전달해 순차적으로 계산하는 Ripple Carry 방식**

- 현재 자리의 계산은 이전 자리의 계산 결과로 도출된 Carry/Borrow를 입력으로 전달 받아야 연산이 진행되어 Propagation Delay(전파 지연)가 발생한다.
- 따라서 계산의 속도가 연산 대상의 비트 수에 비례하여 느려지는 구조적인 한계가 존재함
- 추후 설명할 Carry Look-Ahead 방식이 해당 방식의 단점을 해결하고자 사용됨
- 전력 소비의 경우에도 비트 수에 비례하여 증가함
- 입력 비트 수의 확장이 용이하고 오버플로우/언더플로우를 확인할 수 있음

BCD Adder

Concept

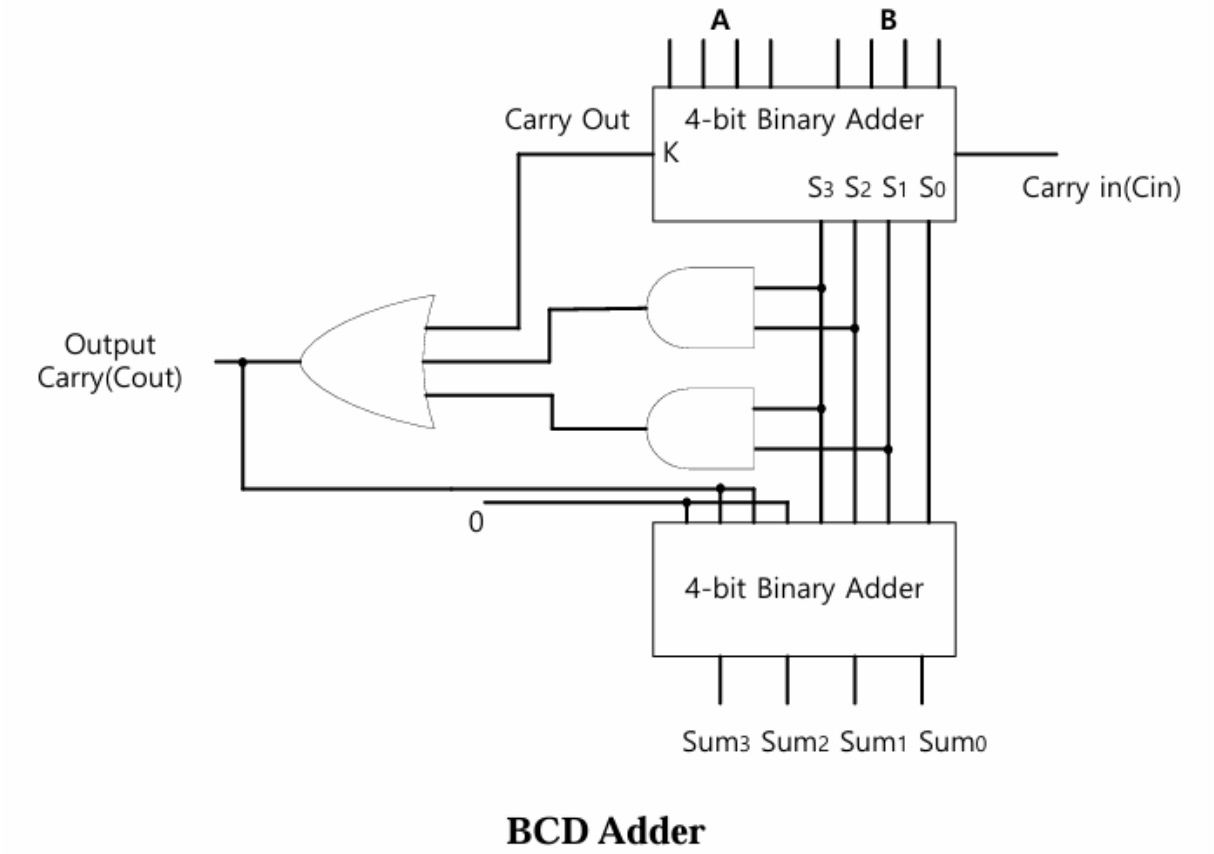
- BCD 코드의 합을 계산하는 가산기
- 두 수의 합 결과가 0~9일 경우 이진수와 동일하게 표기하지만 10~19인 경우에는 carry 비트를 1로 표현하고, 실제 4비트 2진수에 6 (0110)을 더해 보정한다.

ex)
11 | 2진수 0 1011 | BCD 코드 1 0001

```
          1 0 1 1
+         0 1 1 0
_________________
          0 0 0 1
carry 1
```

BCD Adder

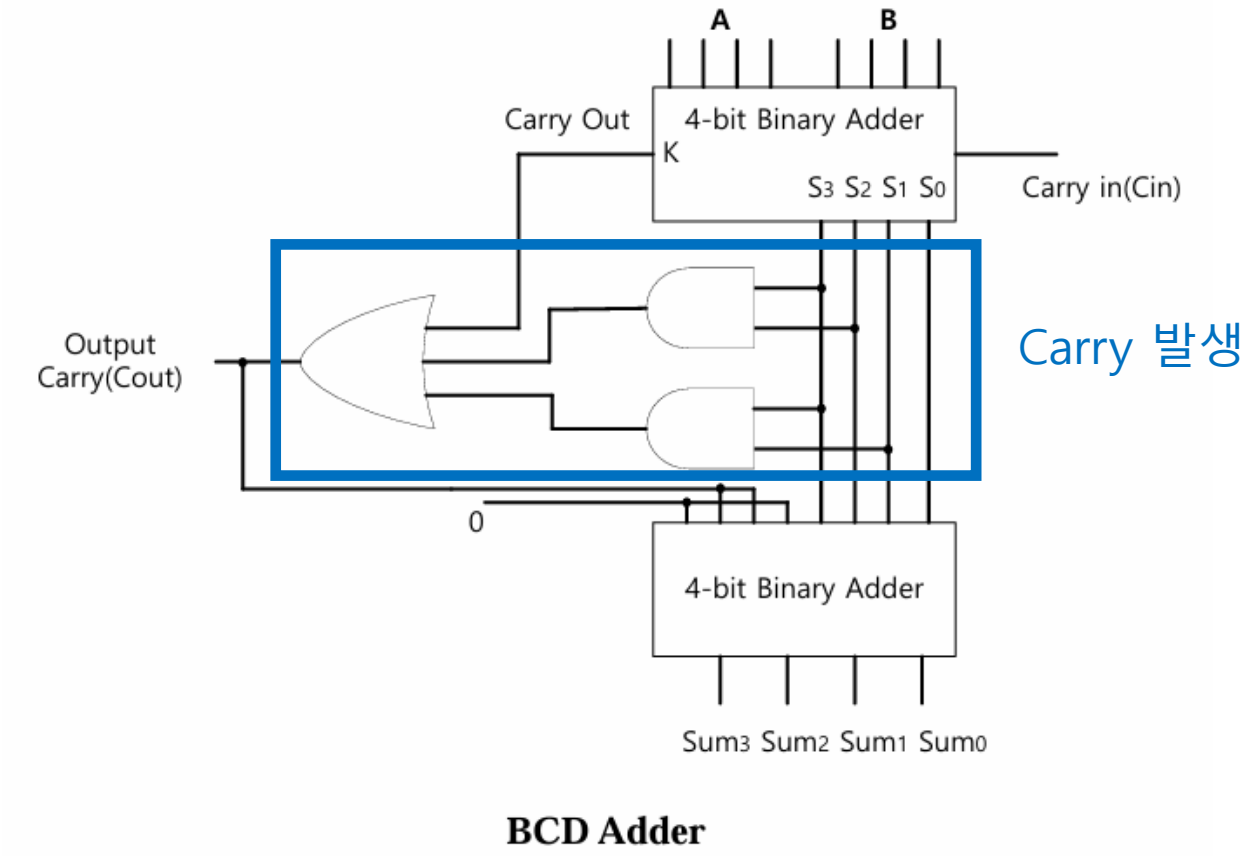
Circuit Diagram



BCD Adder

BCD Adder

Circuit Diagram

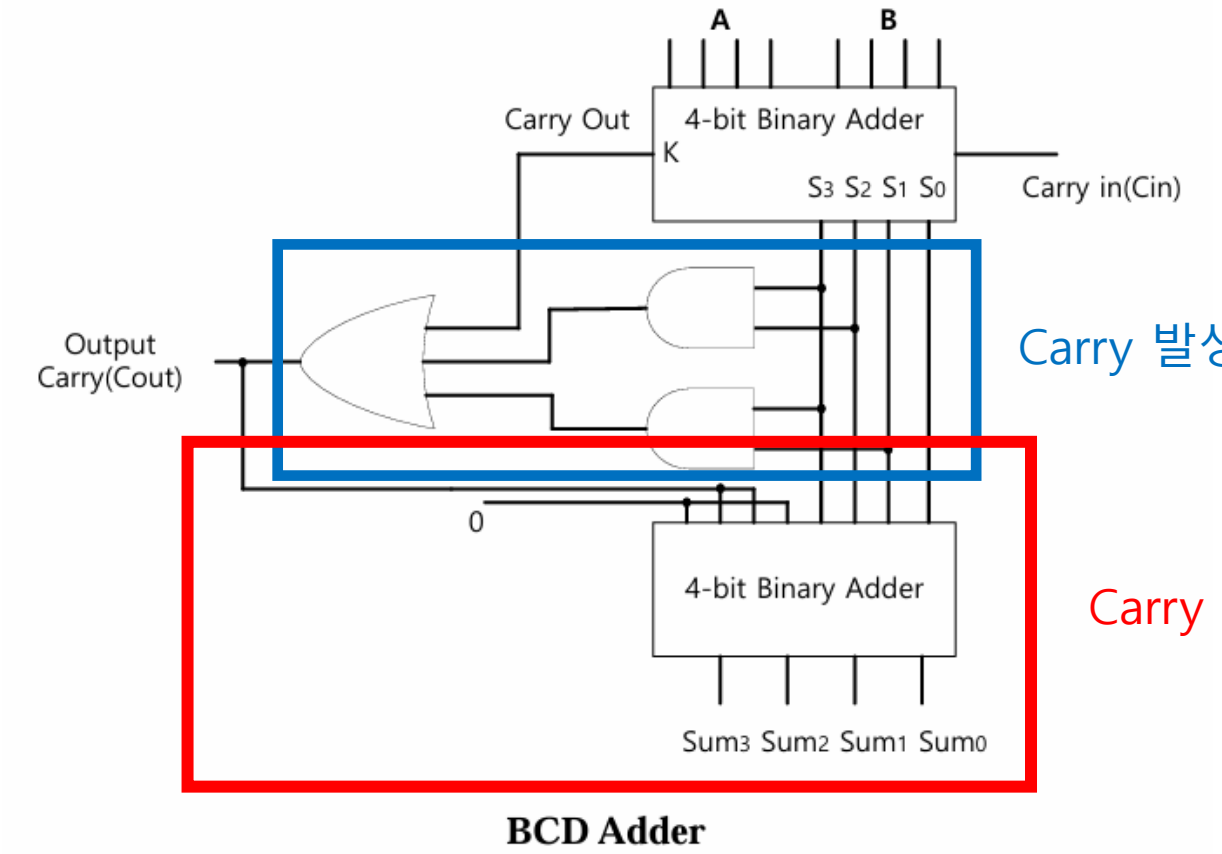


Carry 발생 여부 확인

BCD Adder

BCD Adder

Circuit Diagram



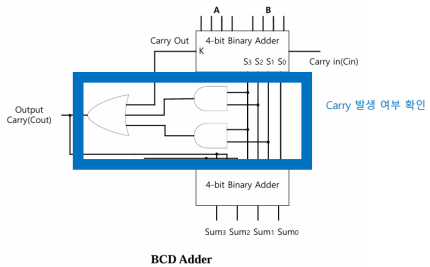
Carry 발생 여부 확인

Carry 발생 여부에 따라 보정

BCD Adder

Materialization

1) Carry 발생 여부 확인



BCD Adder

2진 합					BCD 합					10진값
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

BCD 덧셈표

Z_8Z_4 \ Z_2Z_1	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

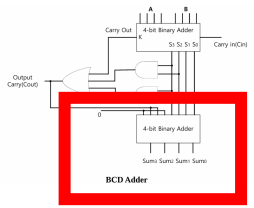
$$C = K + Z_8Z_4 + Z_8Z_2$$

- 기존 2진 합의 진리표를 바탕으로 k map 작성 및 논리식 간소화
- 1,2 번째 비트, 1, 3번째 비트 AND게이트로 연결
- 기존 2진합의 Carry, AND 게이트를 OR 게이트로 연결

BCD Adder

Materialization

2) Carry 발생 시 보정



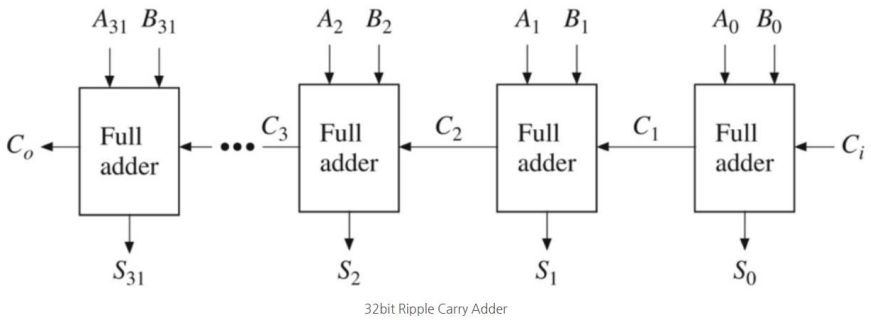
Carry 발생 여부에 따라 보정

- 기존 2진 합에 추가로 4-bit-Adder를 연결
- 1, 4 번째 비트는 0으로 고정, 2, 3번째 비트는 carry 발생 여부에 따라 0 혹은 1
- 결과값이 0~9, 즉 carry가 발생하지 않을 때는 0을 더한다. (보정 없음)
- 반면 결과값이 10~19일 때, 즉 carry가 발생할 때는 0110을 더해 보정한다.

추가 이론 – Carry Look ahead Adder

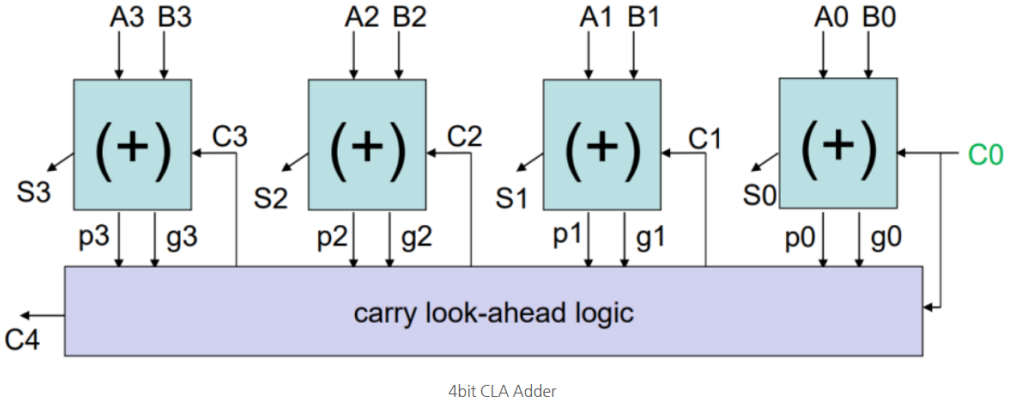
Concept

- look ahead : 내다보다
- 이전 adder의 carry를 기다리고 전달받는 대신 예측하여 계산하는 회로
- 이전 adder에서 carry를 전달받는 시간, 즉 carry propagation을 줄여 계산 속도를 향상시키기 위한 개념



32bit Ripple Carry Adder

기존 adder : 계산하는 비트 수에 비례하여 propagation delay 발생

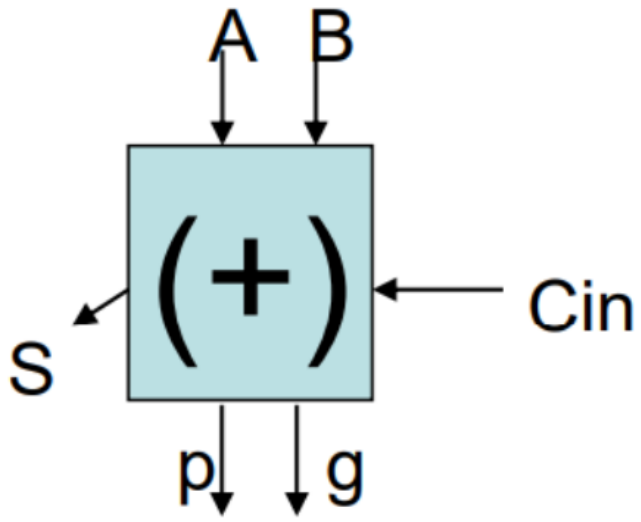


4bit CLA Adder

4-bit-CLA :기존 Adder보다 회로 복잡하지만 빠른 계산 가능

추가 이론 – Carry Look ahead Adder

Concept



다음 Adder에 직접 carry를 전달하는 대신 lookahead 부분에 p와 G 비트 전달

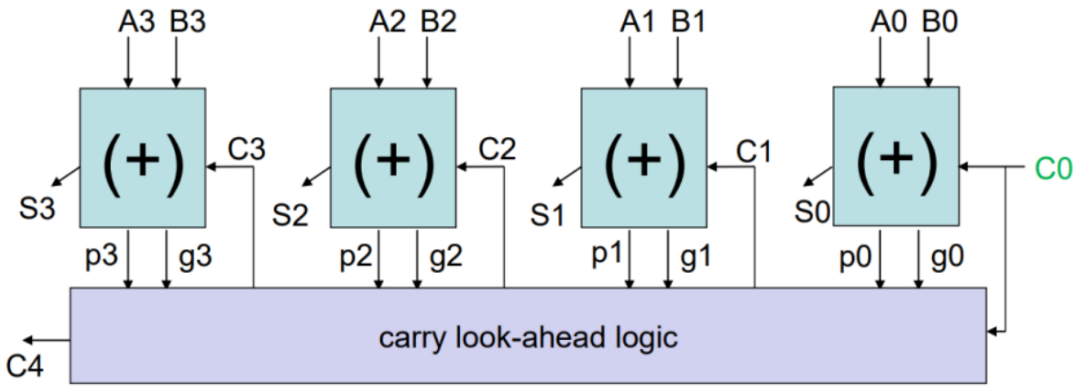
p: carry propagation (carry가 전달되었다)
P = A+B (A OR(XOR) B)

G: carry generation (carry가 생성되었다)
g = AB (A AND B)

Cout = g + pCin

추가 이론 – Carry Look ahead Adder

Concept

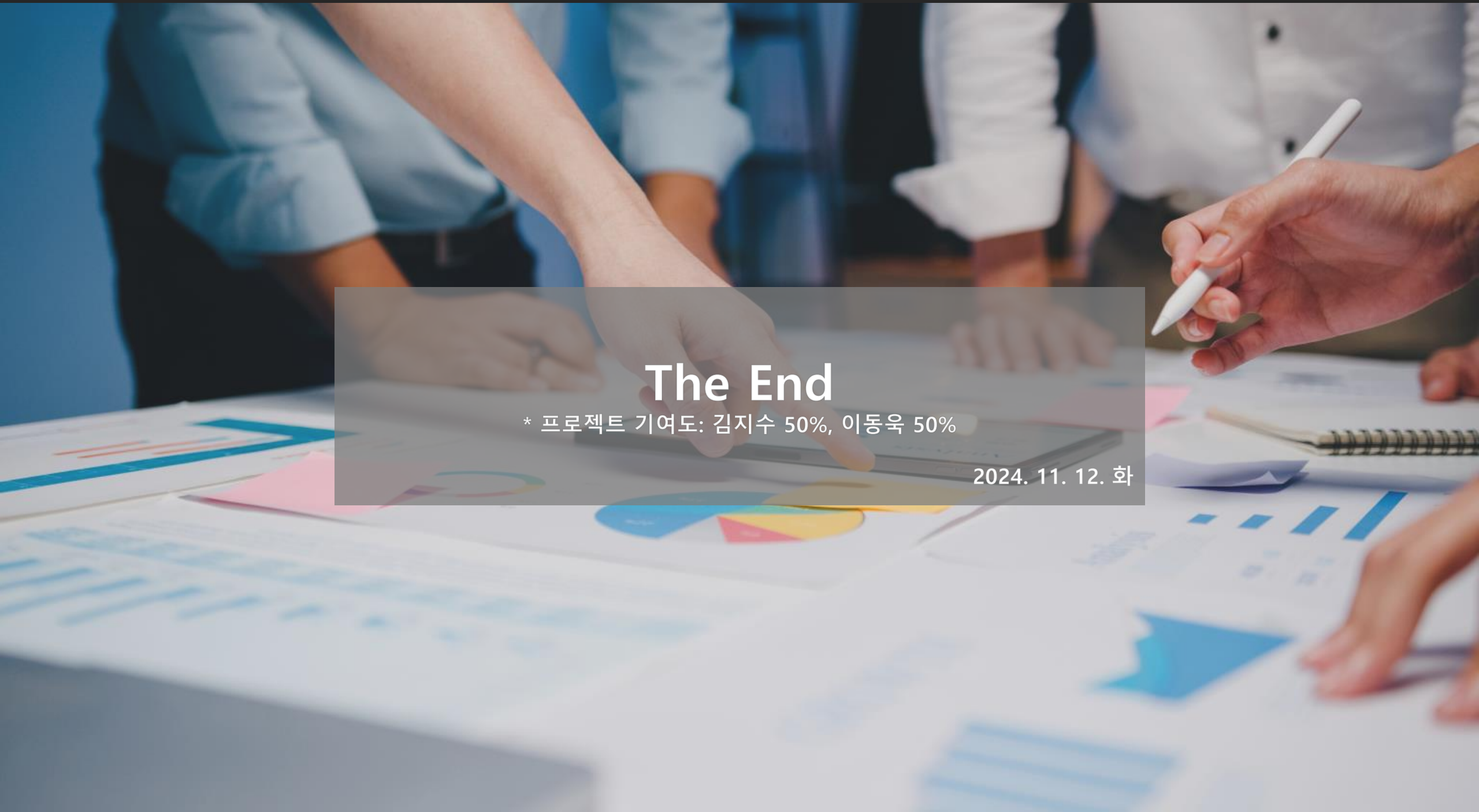


4bit CLA Adder

C1 = G0+P0*C0
C2 = G1+P1*C1
C3 = G2+P2*C2
C4 = G3+P3*C3

C1 = G0+P0*C0
C2 = G1+P1*G0+P1*P0*C0
C3 = G2+P2*G1+P2*P1*G0+P2*P1*P0*C0
C4 = G3+P3*G2+P3*P2*G1+P3*P2*P1*G0+P3*P2*P1*P0*C0

- 모든 adder의 carry가 처음 입력된 C0와 각 adder에 입력되는 A, B값 만으로 계산이 가능
- 4-bit-adder의 경우 delay가 절반 가까이 감소



The End

* 프로젝트 기여도: 김지수 50%, 이동욱 50%

2024. 11. 12. 화