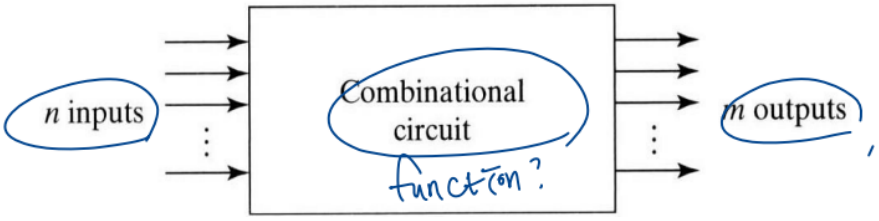
Ch.4 Combinational Logic - part A

Combinational Logics(조합 로직)

① Logic circuits(논리회로)

- Combinational circuit(조합회로) → f(n) = m
  - Output depends on the present inputs. →함수와 유사하다고 생각하기
  - No feedback line and no storage element.



② Sequential circuit(순차회로) → ch.5

- Combinational circuit + Storage elements

- Output depends on present inputs as well as past inputs
  - 출력 값을 저장하고 있다가 입력 값으로 사용
  - the time sequence of inputs and internal states.

⇒ Basic circuit analysis(회로분석) and design(회로도)에 대해서 배울 것

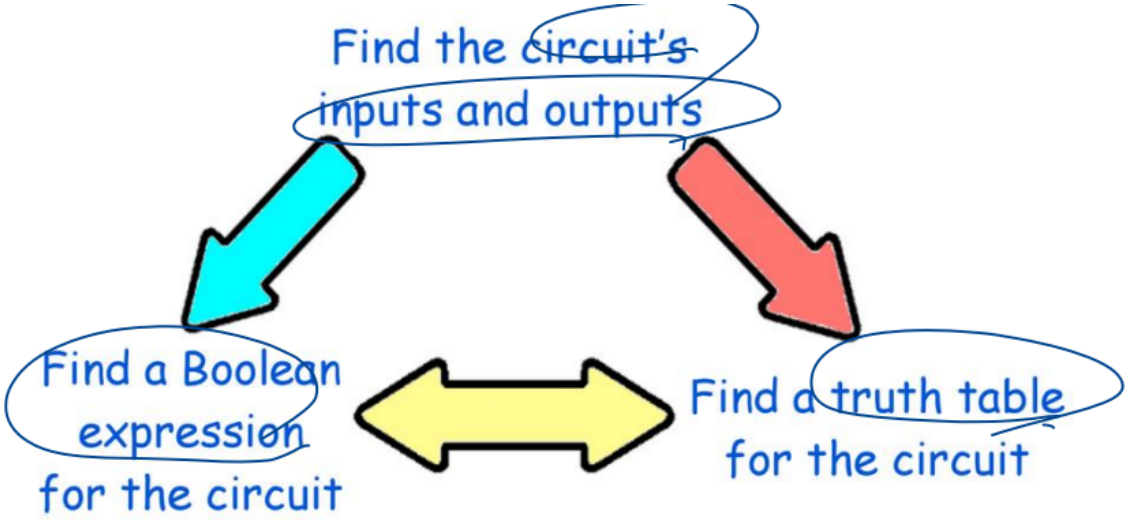
- truth tables , boolean expressions describe functions.
  ① ② ③

- expressions can be converted into h/w circuits
- boolean algebra, K-maps help simplify expressions and circuits.

회로 분석

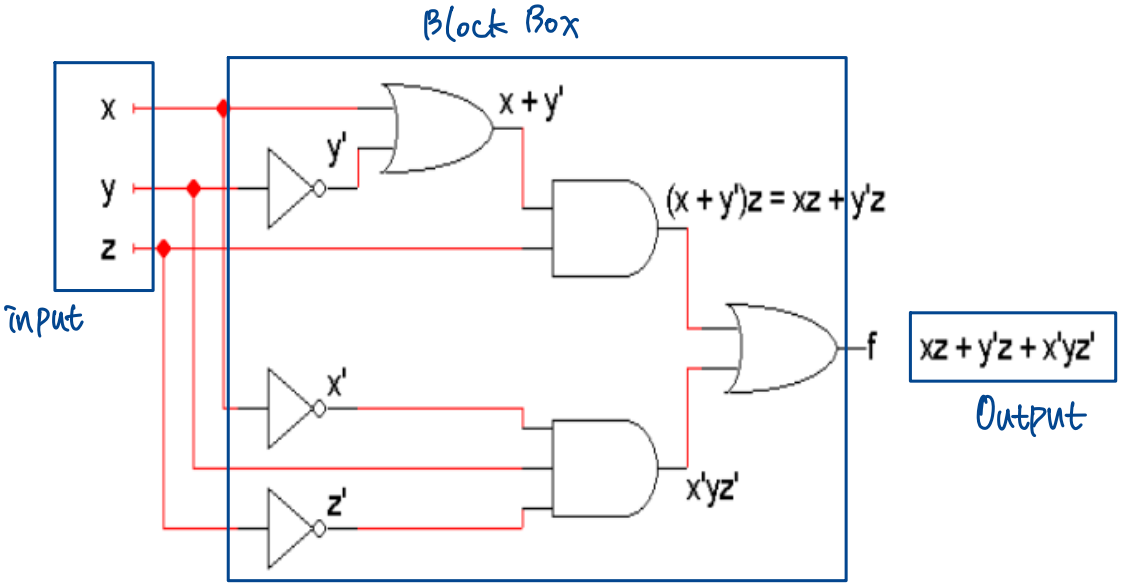
Circuit analysis

involves figuring out what some cirtcuit does.



- 모든 회로는 function으로 compute되고 이 function은 boolean expression, 진리표로 표현 가능
  - 이 boolean expression, or truth table은 무엇임 ?? = circuit analysis

  - **boolean expression**
    1. 가장 먼저 전체 회로의 input, output 파악하기
    2. gate의 input에 따른 각 individual gate의 ouput에 대한 expression 작성하기
       a. input → ouput ⇒ simplication



② **truth table**

1. input, ouput의 개수 찾기
2. 가능한 모든 input 조합들을 나열하기 → 진리표 작성
   a. n개의 input → 2^n개의 rows
3. 회로로 simulating 해서 input들에 대한 output 찾기(by, hand..program..)

+) ▪ boolean expression을 안다면 truth table 쉽게 만들 수 있음

function $f(x, y, z) = xz + y'z + x'yz'$, we can use that to fill in a table:

- We show intermediate columns for the terms xz, $y'z$ and $x'yz'$.
- Then, f is obtained by just OR'ing the intermediate columns.

x	y	z	xz	y'z	x'yz'	f
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	1	1
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	1	1	0	1
1	1	0	0	0	0	0
1	1	1	1	0	0	1

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

→ sum of minterms.

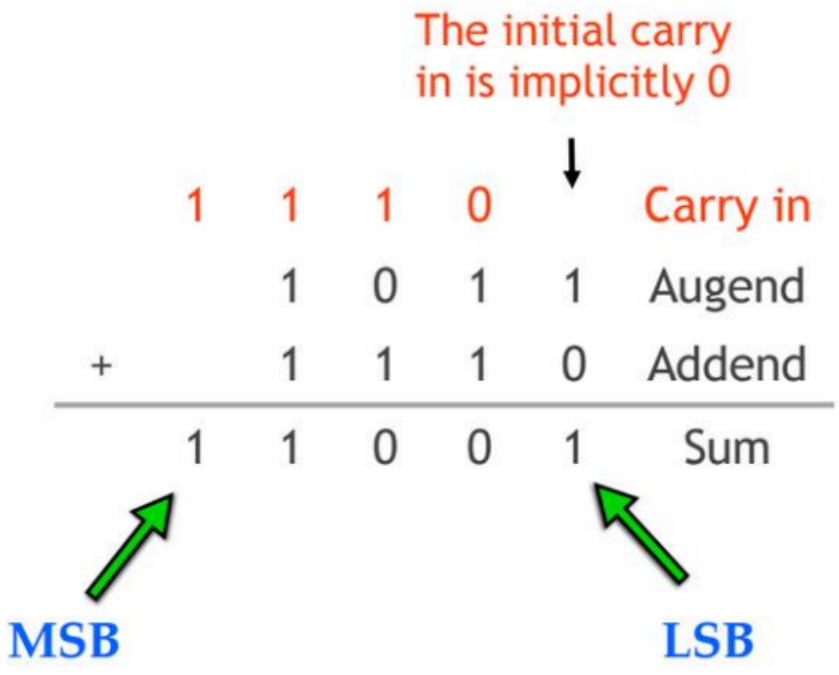
$$f(x,y,z) = \underbrace{x'y'z + x'yz' + xy'z + xyz}$$
$$= m_1 + m_2 + m_5 + m_7$$

- k-map으로 더 단순하게 만들기도 가능

Binary addition by hand

Half adder(반가산기)

- adding two bits
  - two binary numbers → carry 발생 가능



사람의 방식으로 hardware adder 제작

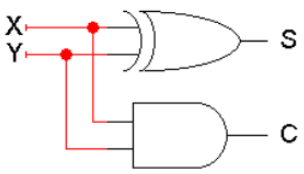
- **half adder** : adds two bits and produces a two bit result ⇒ sum(right) , carry out(left)

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

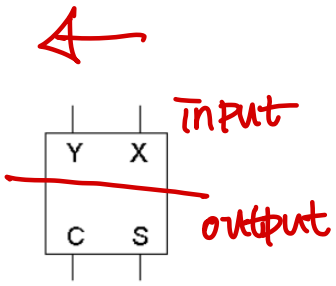
진리표

$$C = XY$$
$$S = X'Y + XY' = X \oplus Y$$

equation



circuit



block symbol

Full adder(전가산기)

- adding three bits
  - three binary numbers → 더하는 수, 더해지는 수, and carry in (right)

    addend augend

    오른쪽, 하위 bit에서 온 carry
- full adder equations
  - full adder circuit takes three bits of input, and produces a two-bit result(sum, carry out)
  - equations
    - XOR operations → simplify

X	Y	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

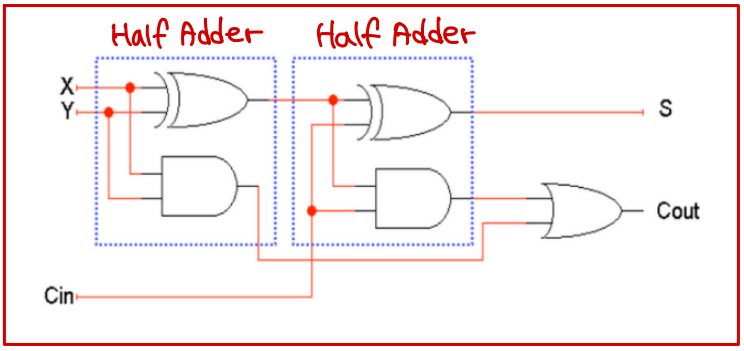
$$\begin{aligned} S &= \Sigma m(1,2,4,7) \\ &= X'Y'Cin + X'YCin' + XY'Cin' + XYC_{in} \\ &= X'(Y'Cin + YC_{in}') + X(Y'Cin' + YC_{in}) \\ &= X'(Y \oplus Cin) + X(Y \oplus Cin)' \\ &= X \oplus Y \oplus Cin \end{aligned}$$

$$\begin{aligned} C_{out} &= \Sigma m(3,5,6,7) \\ &= X'YCin + XY'Cin + XYC_{in}' + XYC_{in} \\ &= (X'Y + XY')C_{in} + XY(C_{in}' + C_{in}) \\ &= (X \oplus Y)C_{in} + XY \end{aligned}$$

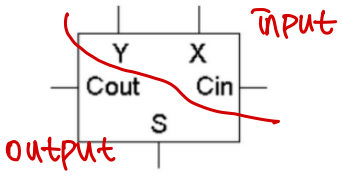
    - used algebra → k-maps 으로부터 XOR 하기는 쉽지 않음

- full adder circuit
  - 두 개의 half adder로 구성

$$S = X \oplus Y \oplus C_{in}$$
$$C_{out} = (X \oplus Y)\, C_{in} + XY$$



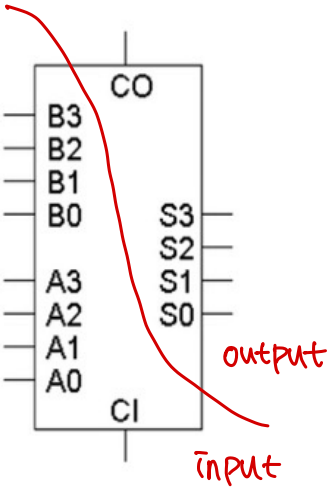
Full Adder

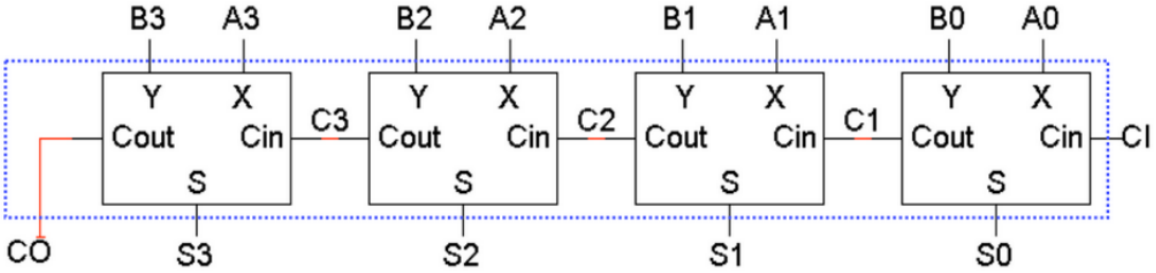


A 4-bit adder

4개의 full adder로 구현

- input : 9개
  - A3, A2, A1, A0 / B3, B2, B1, B0 / CI(carry in)
- ouput : 5개
  - S3, S2, S1, S0 / CO(carry out)

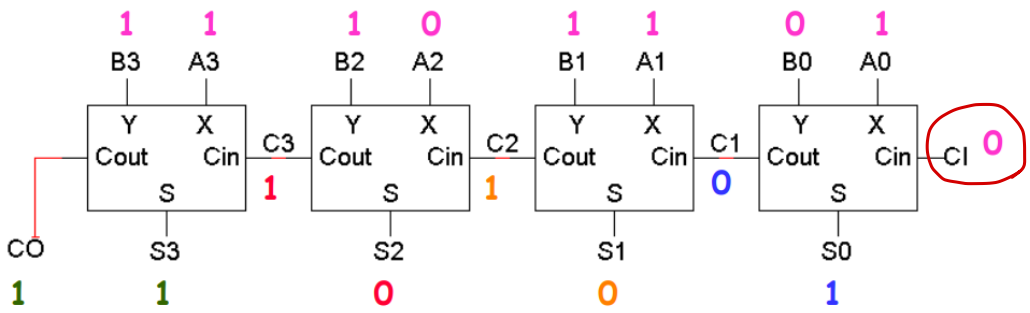




→ hierarchical structure, 5개의 input에 대한 512개의 row를 가진 진리표

- example of 4-bit adder

Let's try our initial example: A=1011 (eleven), B=1110 (fourteen).



1. Fill in all the inputs, including CI=0
2. The circuit produces C1 and S0 (1 + 0 + 0 = 01)
3. Use C1 to find C2 and S1 (1 + 1 + 0 = 10)
4. Use C2 to compute C3 and S2 (0 + 1 + 1 = 10)
5. Use C3 to compute CO and S3 (1 + 1 + 1 = 11)

The final answer is 11001 (twenty-five).

⇒ 4bit인데 결과값은 5bit → **overflow** !!

- **Overflow**

부호가 없는 덧셈에서 overflow ; carry out이 1이 될 때 일어남

즉, input bit 수와 최종 output bit 수가 다름 → 사용 x

- **Hierarchical adder design**

Q. 4bit 두 개를 더하면 항상 carry in이 0인데, CI input이 있는 이유는?

- A.

ex) 8 bit adder → 4 bit 2개

1. 우리가 4bit adder를 여러 개 합쳐서 더 큰 adders를 만들 수 있기 때문이다.

   (like, 4bit adder를 만들 때 full adder를 여러 개 합친 것처럼)

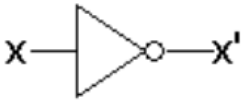
⇒ 이때, ① symbol끼리 잘 같게 써줘야 함

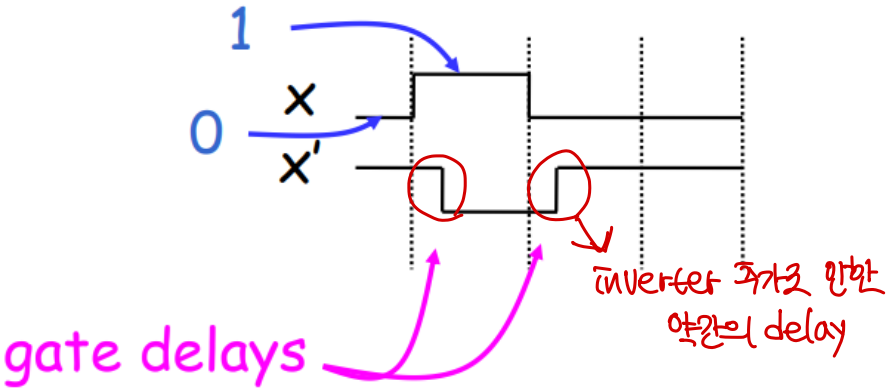
② 주어진 회로의 역할에 맞게 연결할 것

2. plus, subtraction에 carry in이 쓰임

Gate delays

gate가 갖는 시간적인 지연





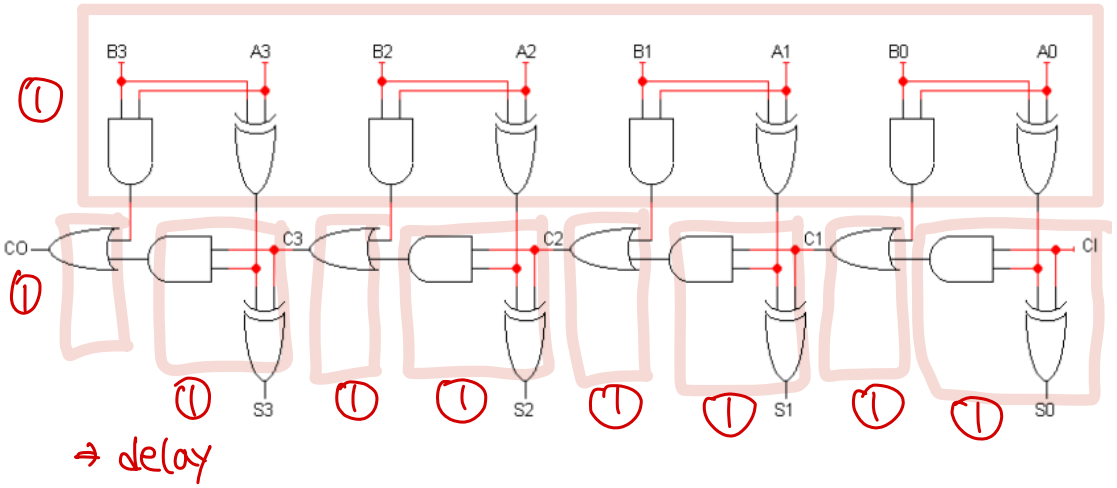
- 모든 gate는 input과 output 사이에 아주 짧은 fraction이 존재 ⇒ gate delay
  - θ delay or θ cylcle 표현
- 실제로 gate delay를 계산하는 복잡한 방법이 있지만 우리 수업에서는 모두 동일한 지연 가정
- timing diagram으로 graphically하게 표현 가능

Ripple carry adder(RCA) : delay

- ripple carry adder
  - input A0, B0, CI 이 S3가 생성될 때까지 왼쪽으로 ripple 됨.
  - ripple carry adders are slow
    - nbit ripple carry adder → longes path has 2n + 1 gates

ex) 4 bit RCA → 2x4+1 = 9 gates

ex) 64 bit → 129 gates



↔ **Algebraic carry out : a faster way to compute carry outs**

Carry lookahead adders

- instead of waiting for the carry out from all the previous stage.
  - we could compute it directly with a two-level circuit ⇒ minimizing

- 가장 하위 carry out만 알면 다 구할 수 있음 → 모든 carry out은 SOP
  1. define two functons
     a. generate function
     : produces 1 → position i에서 Ai and Bi가 모두 1일 때

CARY는 둘 중 하나면 생성됨 ⇒ a+b

$$g_i = A_i B_i$$

A_i	B_i	C_i	C_{i+1}
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

     b. propagate function
     : position i의 incoming carry가 있을 때, Ai or Bi가 1일 때

$$p_i = Ai \oplus Bi$$

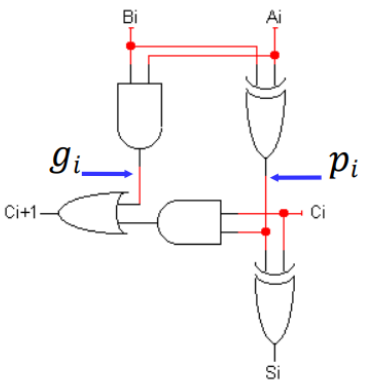
→ 둘이 다르면 1
⇒ CARY가 생김

2. carry out function

$$c_{i+1} = g_i + p_i c_i$$

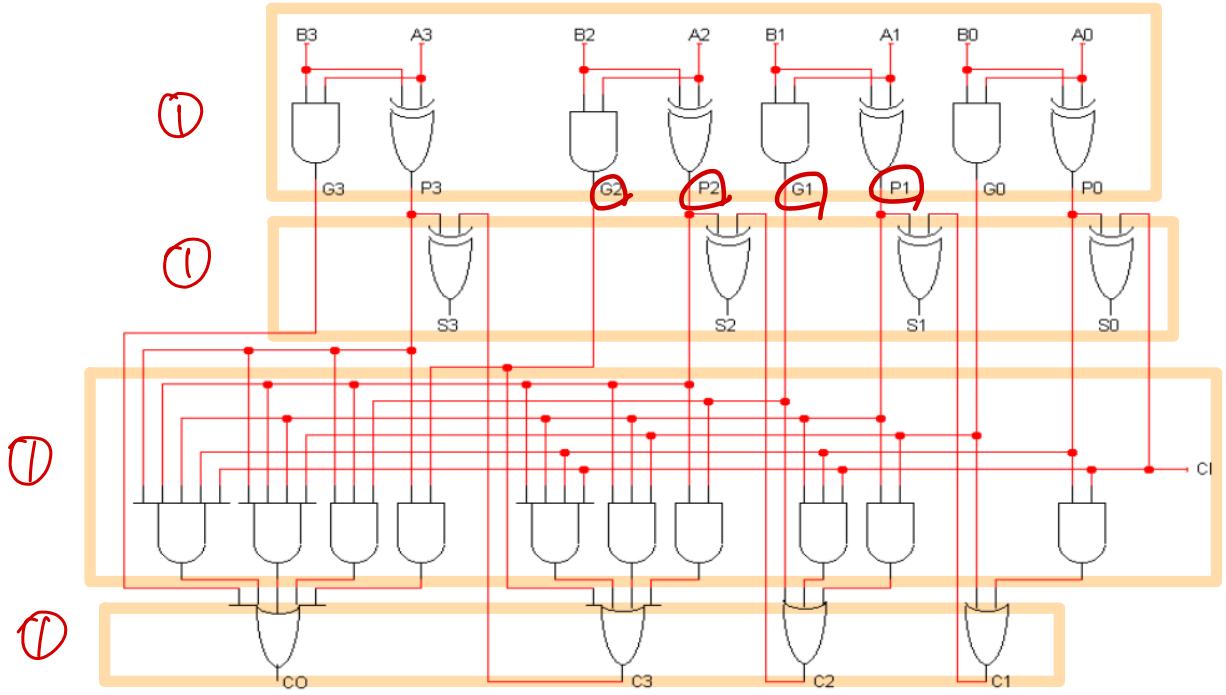
이전 것

$$
\begin{aligned}
c_1 &= g_0 + p_0 c_0 \\
c_2 &= g_1 + p_1 c_1 \\
&= g_1 + p_1(g_0 + p_0 c_0) \\
&= g_1 + p_1 g_0 + p_1 p_0 c_0 \\
c_3 &= g_2 + p_2 c_2 \\
&= g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 c_0) \\
&= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \\
c_4 &= g_3 + p_3 c_3 \\
&= g_3 + p_3(g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0) \\
&= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0
\end{aligned}
$$



- pros and cons
  - by adding more h/w
    - 회로의 level을 줄일 수 있음
    - speed up !
      - ex) RCA : delay 9개 / CLA : delay 4개 (4bit에서)
  - CLA와 RCA
    - CLA :
      - delay가 logarithmically 하게 증가
      - 빠르지만 복잡함
    - RCA :
      - delay가 linearly하게 증가
      - 느리지만 간단함

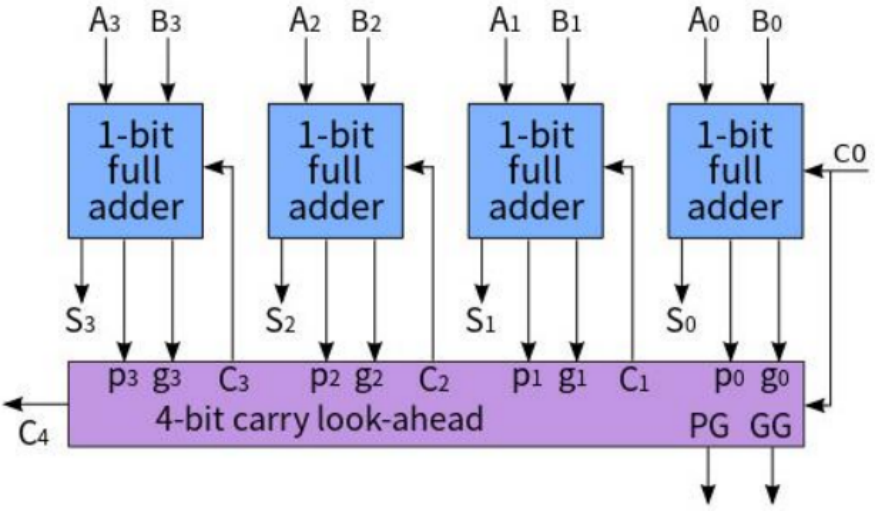
A 4-bit carry lookahead adder



⇒ 4bit CLA는 4 gate delay

Group Propagation : $PG = P_0P_1P_2P_3$

Group Generation : $GG = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3$



+) 16 bit CLA는 8 gate delay가 maximal time

16bit RCA는 33 gate delay

64bit : CLA 12개 / RCA 129개