**컴퓨터공학실험 2 6 주차 예비보고서**

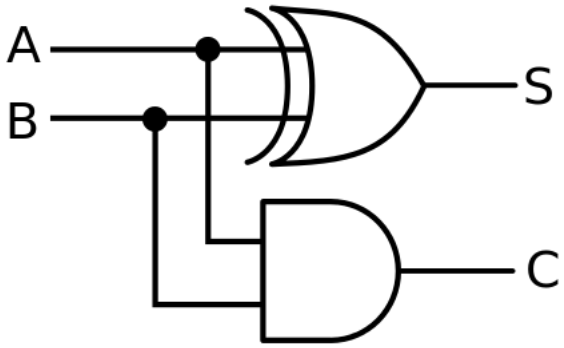
**전공: 컴퓨터공학과 학년: 2 학번: 20191559 이름: 강상원**

**1. 전 가산기 및 반 가산기에 대해 조사하시오(예시 포함).**

가산기는 수의 덧셈을 수행하는 회로이다. 가산기는 크게 전가산기(Full Adder), 반가산기(Half Adder)로 나뉜다. 먼저 반가산기에 대해 다뤄보겠다.

1) 반가산기

반가산기는 1비트 두 입력으로 합과 carry(자리 올림)을 계산하는 회로이다. 아래 대표적인 형태의 반가산기는 하나의 XOR gate와 AND gate로 구성되어 있다. $S = A'B + AB'$, $C = AB$로 나타낼 수 있다. 반가산기의 회로도와 진리표는 다음과 같다.

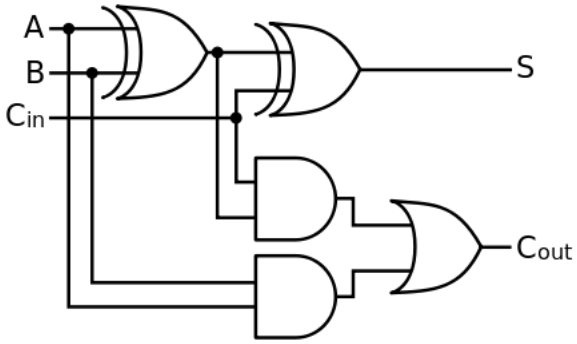



| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

총 합의 결과는 $2C + S$이다.

2) 전가산기

전가산기는 입력 A와 B, 그리고 하위 비트의 carry(자리 올림)을 포함해 2진수 입력 3개를 덧셈 연산하는 회로이다. 아래 대표적인 형태의 전가산기는 두 개의 반가산기와 하나의 OR gate로 이루어져 있다. $S = A \oplus B \oplus C_{in}$, $C_{out} = (A \cdot B) + \left(C_{in} \cdot (A \oplus B)\right)$으로 표현된다. 전가산기의 회로도와 진리표는 다음과 같다.




| A | B | $C_{in}$ | S | $C_{out}$ |
|---|---|---|---|---|
| 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 |

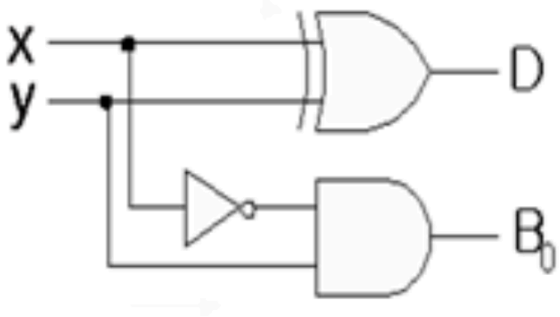
총 합의 결과는 $2C_{out} + \mathrm{S}$이다.

**2. 전 감산기 및 반 감산기에 대해 조사하시오(예시 포함).**

감산기는 수의 뺄셈을 수행하는 회로이다. 가산기는 크게 전감산기(Full Subtractor), 반감산기(Half Subtractor)로 나뉜다. 먼저 반감산기에 대해 다뤄보겠다.

1) 반감산기

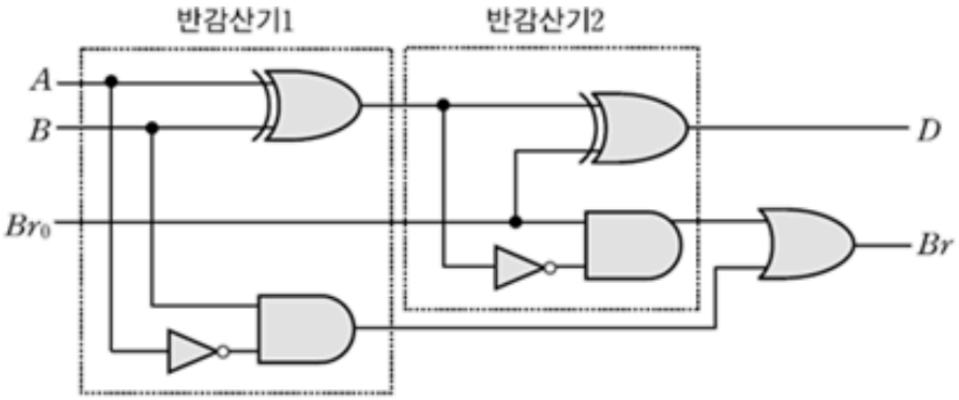
반감산기는 1비트 두 입력의 차를 계산하는 회로이다. X, Y 두 입력이 주어지며 차 D와 빌림수 $B_{out}$이 출력된다. 논리식 $\mathrm{D} = \mathrm{X} \oplus \mathrm{Y}$, $B_{\mathrm{out}} = \overline{X} \cdot Y$으로 표현된다. 반감산기의 회로도와 진리표는 다음과 같다.



| X | Y | D | $B_{\text{out}}$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

2) 전감산기

전감산기는 반감산기와 마찬가지로 1비트 입력의 차를 계산하는 회로이지만, 아랫자리에서요하는 빌림수에 대한 뺄셈까지 계산하는 회로이다. $A, B, \mathrm{B}r_0 (= B_{in})$ 세 입력이 주어지며 차 D와 빌림수 $B_{out}$이 출력된다. 아래 대표적인 형태의 전감산기는 두 개의 반감산기와 하나의 OR gate로 이루어져 있다. 논리식 $D = \mathrm{A} \oplus \mathrm{B} \oplus \mathrm{B}r_0$, $B_{\text{out}} = \overline{\mathrm{A}} \cdot \mathrm{B}r_0 + \overline{\mathrm{A}} \cdot \mathrm{B} + \mathrm{B} \cdot \mathrm{B}r_0$으로 표현된다. 전감산기의 회로도와 진리표는 다음과 같다.




| A | B | $\mathrm{B}r_0$ | D | $B_{out}$ |
|---|---|---|---|---|
| 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 |

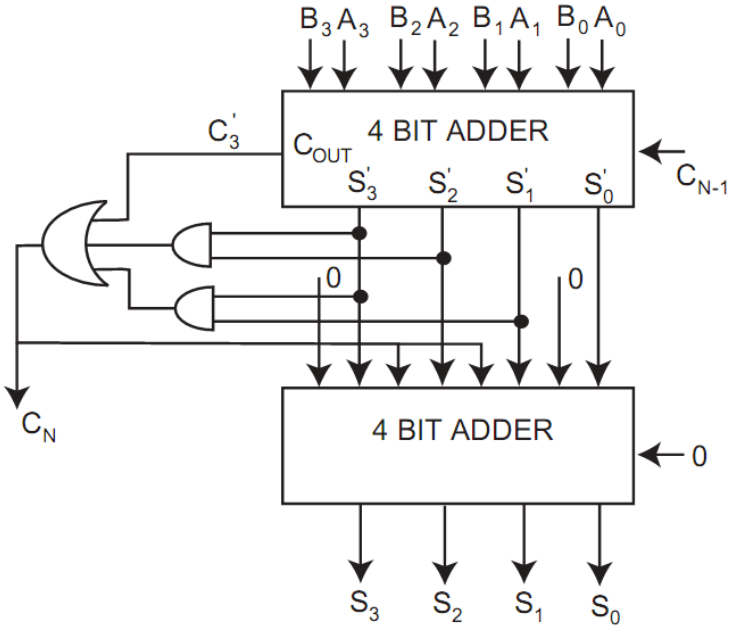
**3. BCD 가산기에 대해 조사하시오.**

BCD 가산기는 두 BCD(Binary Coded Decimal, 이진화된 십진수) 수를 더해 BCD로 결괏값을 출력하는 회로이다. 합의 결과가 9 이하이면 그대로 출력, 9보타 크면 보정을 수행한다. (BCD가 한 자리씩, 0~9까지의 숫자만 변환 가능하기 때문) 9+9+carry => 최대 19의 결과값이 나오는데, 10~19의 값일 때 보정이 필요한 것이다.

아래의 표는 BCD 가산기를 설계할 때 사용하는 표이다.

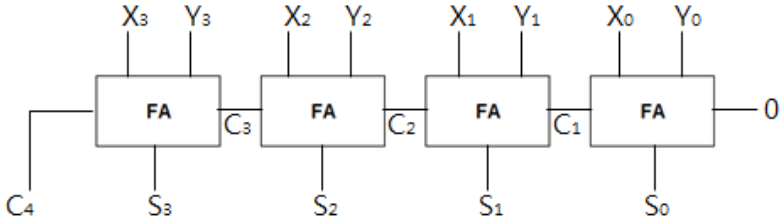
| Binary Sum | | | | | | BCD Sum | | | | | | Decimal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| K | $Z_8$ | $Z_4$ | $Z_2$ | $Z_1$ | | C | $S_8$ | $S_4$ | $S_2$ | $S_1$ | | |
| 0 | 0 | 0 | 0 | 0 | SAME CODE | 0 | 0 | 0 | 0 | 0 | | 0 |
| 0 | 0 | 0 | 0 | 1 | | 0 | 0 | 0 | 0 | 1 | | 1 |
| 0 | 0 | 0 | 1 | 0 | | 0 | 0 | 0 | 1 | 0 | | 2 |
| . | . | . | . | . | | . | . | . | . | . | | . |
| . | . | . | . | . | | . | . | . | . | . | | . |
| . | . | . | . | . | | . | . | . | . | . | | . |
| . | . | . | . | . | | . | . | . | . | . | | . |
| 0 | 1 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | 0 | | 8 |
| 0 | 1 | 0 | 0 | 1 | | 0 | 1 | 0 | 0 | 1 | | 9 |
| **10 to 19 Binary and BCD codes are not the same** | | | | | | | | | | | | |
| 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 |

➔ 다음과 같은 회로 개형을 가진다.

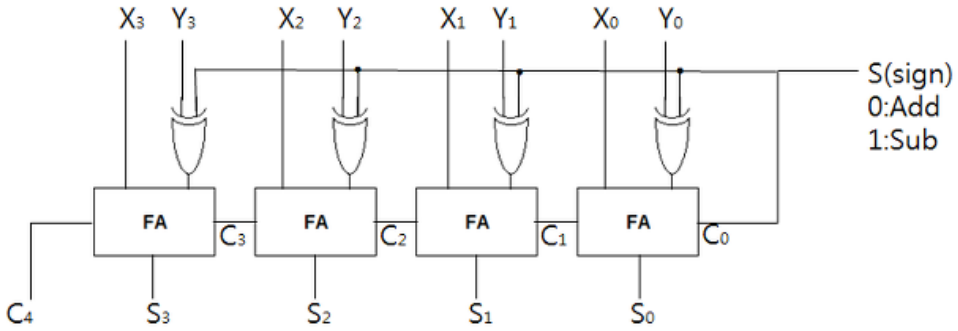


**4. 병렬 가감산기에 대해 조사하시오.**

병렬 가감산기는 가산기와 감산기 기능을 모두 수행하는 회로이다. 전가산기 여러 개를 병렬로 연결하면 병렬가산기를 만들 수 있는데, (아래 회로도 참고)




이렇게 만들어진 병렬가산기의 입력 Y 대신 Y XOR S(부호)를 입력하면 가산기, 감산기 기능을 하는 병렬 가감산기를 만들 수 있다.



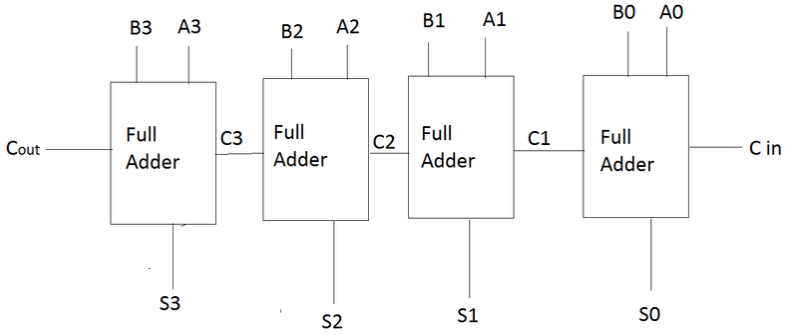

위 그림에서 알 수 있듯이, 덧셈의 경우에는 S로 0이 입력되어 0과 B XOR => B가 다음으로 전달된다. ∴ 전가산기를 지나며 덧셈을 한다.

뺄셈의 경우에는 S로 1이 입력되어 B XOR 1 => B의 1의 보수가 다음으로 전달된다. $C_0$에는 1이 입력되어 +1을 수행한다.

➔ B의 2의보수 형태 만듦. ∴ 결과적으로 뺄셈의 역할을 수행하는 회로가 된다.

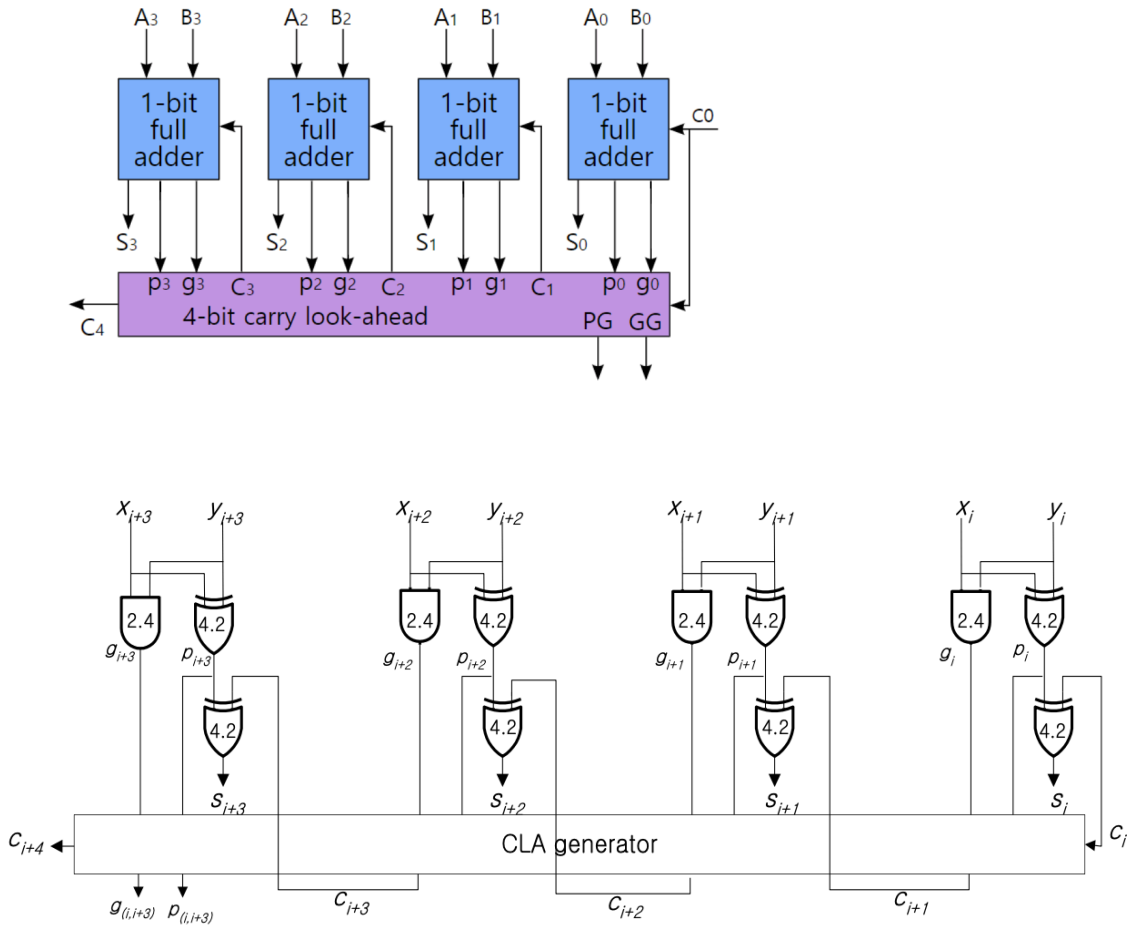
**5. Carry Look-Ahead Adder 을 Ripple Carry Adder 와 비교하여 설명하시오.**

두 개 이상의 Bit 덧셈을 하기 위해서는 전가산기 여러 개가 필요하다. 가장 단순한 형태로 구현하는 방법은 Ripple Carry Adder이다. Ripple Carry Adder는 이름과 같이, 하나의 가산기에서 나온 출력을 다음 가산기로 넘겨주고, 다시 가산기로 들어가 출력하여 넘겨주고… 를 반복하는 가산기이다.



그러나 이러한 방식은 회로에 지연이 길어진다는 단점이 있다. (n-비트 가산기의 출력이 안정화되는 시간은 $(2n+4)\Delta$ 이다. 따라서 가산기의 속도를 향상시키기 위하여 제안된 방법 중 하나가 Carry Look-Ahead Adder이다.

Carry Look-Ahead Adder는 Carry의 계산 결과값을 기다리며 발생하는 지연을 줄이기 위해 Carry를 따로 계산해 준다. 미리 계산된 Carry를 전가산기에 넘겨 기존 Ripple Carry Adder보다 빠르게 주어진 계산을 수행할 수 있다.




↑ 4-bit Carry Look-Ahead Adder

## 6. 기타이론.

Carry Generate $g_i = x_i y_i$. Carry Propagate $p_i = x_i \oplus y_i$라 부른다.

이를 이용해 정리하면

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

$c_{i+2} = g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i$

$c_{i+3} = g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i$

$c_{i+4} = g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i + p_{i+3} p_{i+2} p_{i+1} p_i c_i$

와 같이 표현된다. 간략화된 버전은 아래와 같다.

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

$c_{i+2} = g_{i+1} + p_{i+1} c_{i+1}$

$c_{i+3} = g_{i+2} + p_{i+2} c_{i+2}$

$c_{i+4} = g_{i+3} + p_{i+3} c_{i+3}$

∴ 직접 입력 bit 와 carry $c_i$로부터 계산 수행이 가능하다.

➔ 다음 carry를 예측하여, 그 이전 carry가 계산되어 있는 상태가 아닐지라도 두 입력값 A와 B만으로도 carry를 구할 수 있게 된다.

이러한 방식으로 Carry Look-Ahead Logic 이 구성된다.

- Manchester Carry Chain

  사용되는 트랜지스터의 개수를 줄이기 위해 나온 방식이다. 가장 위의 carry를 계산하는 gate에서 노드를 꺼내서 중간 carry를 생성한다.