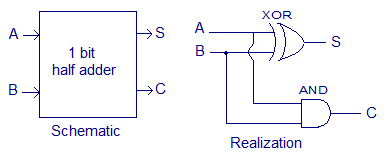
6주차 예비보고서

|  |
| --- |
| - 반 가산기 및 전 가산기  - 반 감산기 및 전 감산기  - BCD 가산기와 병렬 감/가산기  - Carry Look-Ahead Adder와 Ripple Carry Adder |

20141196 김성희

1. 반 가산기 및 전 가산기

**# 반 가산기** (1bit 계산에 사용)

1bit 이진수 두 개(A, B)의 SUM(S)와 CARRY(C)를 구하는 회로

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

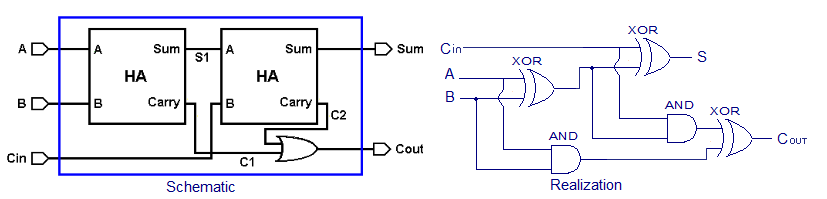
\*논리식: SUM = AⓧB, CARRY = AB

\*Carry는 자릿수 올림을 뜻한다.

**# 전 가산기** (1bit 포함 2bit이상의 자릿수의 계산에 사용)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INPUT | | | OUTPUT | |
| A | B | Cin | S | Cout |
| 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 |

1bit 이진수 두 개(A, B)와 CARRY IN(Cin)의 SUM(S)와 CARRY OUT(Cout)을 구하는 회로.

****

카르노맵을 사용하면 Cout은 (A+B)Cin+AB로 줄일 수 있지만 S에서 재사용해야 하므로 (A+B)대신 (AⓧB)를 사용한다.

\*논리식: SUM = (AⓧB)ⓧCin, CARRY OUT = (AⓧB)Cin + AB

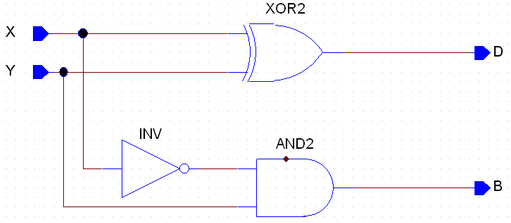
\*Cin은 이전 자릿수 계산의 자릿수 올림을, Cout은 현재 올림을 뜻한다.

2. 반 감산기 및 전 감산기

**#반 감산기** (X-Y)

1bit 이진수 두 개(X, Y)의 Difference(D, X-Y)와 Borrow(B)여부를 구하는 회로

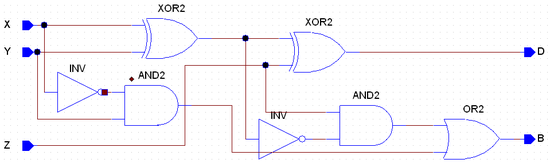
|  |  |  |  |
| --- | --- | --- | --- |
| INPUTS | | OUTPUTS | |
| X | Y | D | B |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |



\*논리식 D = XⓧY, B = X`Y

\*X=0,Y=1일 때, X-Y는 음수다. 따라서 이 때, 앞의 자리수로부터 bit를 빌려와야 하며 이를 B로 나타낸다. 여기서의 출력 B는 밑의 전 감산기의 (X-Y-B)의 입력 B처럼 활용된다.

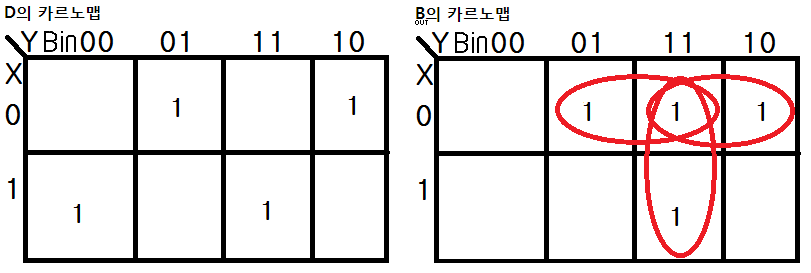
**#전 감산기 (X-Y-B)**

1bit 이진수 3개 X, Y, Borrow in(Bin)을 통해 Difference(D, X-Y-Bin)와 Borrow out(Bout)을 구하는 회로

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INPUTS | | | OUTPUTS | |
| X | Y | Bin | D | Bout |
| 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 |

\*Bin은 이전 자리수 계산에서 현재 자릿수의 bit를 빌렸는지의 여부다. 따라서 D = X-Y-Bin으로 계산한다. 이 때, D가 양수 또는 0이면 Bout=0이고 D가 음수라면 Bout=1이고 D= (b10+X-Y-Bin)으로 계산한다.

\*카르노맵을 통해 논리식을 도출해보자. 카르노맵은 다음과 같다.



여기서 D의 경우 묶을 수 있는 게 한 칸짜리 정사각형밖에 없으므로 SOP는 D=X`Y`Bin+X`YBin`+XY`Bin`+XYBin이다. 간소화를 하기 위해 논리법칙들을 사용하면, D=X`(Y`Bin+YBin`)+X(Y`Bin`+YBin)=X`(YⓧBin)+X(Y`ⓧBin)=X`(YⓧBin)+X(YⓧBin)`=XⓧYⓧBin이다.

Bout의 경우 빨간색 묶음에 따라 SOP는 Bout=X`Bin+X`Y+YBin이다.

3. BCD 가산기와 병렬 감/가산기 (BCD: Binary Coded Decimal)

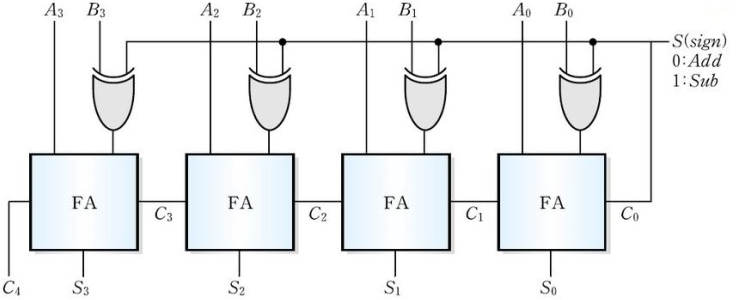
\*BCD가산기는 병렬가산기 2개를 이용한다. 십진수(0~9) 2개를 입력 받아서 십진수(0~19) 1개를 출력한다. 4bit입력 2개로 K Z4 Z3 Z2 Z1 5bit 값을 출력해준다. 그러나 이 때 K=1은 십진수 10을 의미한다. 따라서 병렬가산기에서 나온 C S4 S3 S2 S1을 K Z4 Z3 Z2 Z1으로 변환해 줄 필요가 있고 여기서 병렬가산기를 하나 더 사용한다.(C=1이면 십진수2를, K=1이면 십진수 10을 뜻한다.) 먼저 병렬 가산기에 대해 알아보자.

**#병렬 감/가산기**

-보통 4비트 계산기를 사용한다.

-전 가산기 4개를 선형으로 이어서 사용한다.

<그림>



A = A3 A2 A1 A0, B = B3 B2 B1 B0

A-B = A+(B`+1)를 이용해서 병렬 감산기도 전 가산기를 이용한다. 위 그림에서 S(sign)이 0이면 A+B가 되고, 1이면 A+(B`+1)이 된다.

**#BCD가산기**

BCD가산기는 앞서 말했듯이 입력 값도 출력 값도 K A3 A2 A1 A0의 형태다.(K=1이면 십진수10을 의미한다. A0~3은 0또는 1이다.) 4비트 병렬가산기를 베이스로 사용하는데 병렬 가산기를 쓴다면 carry가 나왔을 때, 그 carry는 십진수로 2를 의미한다. 그러나 BCD는 carry가 십진수 10을 의미하므로 우리는 병렬 가산기의 결과를 BCD(Binary Coded Decimal)로 바꿔 줄 필요가 있다. 이를 위해 다음 표를 보자.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 병렬가산기출력 | | | | | BCD가산기출력 | | | | | 십진수 |
| C5 | S4 | S3 | S2 | S1 | K5 | Z4 | Z3 | Z2 | Z1 |
| 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 |
| 0 | 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 |

표의 왼쪽 부분과 오른부분에서 십진수 9까지는 같은 패턴의 bit를 가진다. 그러나 10이상부터 K5 Z4 Z3 Z2 Z1 패턴은 C5 S4 S3 S2 S1 패턴 + 6이라는 규칙으로 바뀌기 시작한다. 이 때, +6에 해당하는 부분을 병렬 가산기 한 개를 추가해서 처리한다. 따라서 병렬 가산기를 2개 써야 한다.

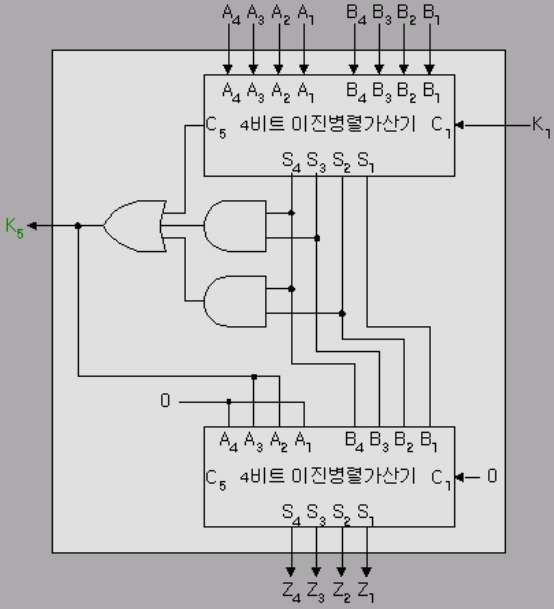
K5 = C5+(S4)(S3)+(S4)(S2)임을 알 수 있다. (잘 모르겠으면 카르노맵 사용)

즉 K5 Z4 Z3 Z2 Z1의 규칙은 다음과 같다.

K5 = 0, K5 Z4 Z3 Z2 Z1 = C5 S4 S3 S2 S1

K5 = 1, K5 Z4 Z3 Z2 Z1 = C5 S4 S3 S2 S1 + 6

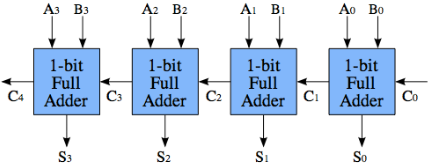
이제 그림을 통해 BCD가산기가 어떻게 구현되는지 알아보자.

왼쪽 그림에서 K=1일 때 +6을 해주는 부분이 밑의 병렬 가산기에서 A4 A3 A2 A1에 0 1 1 0을 입력하는 부분이다.

**4. Carry Look-Ahead Adder와 Ripple Carry Adder**

**# Ripple Carry Adder**

전 가산기를 선형구조로 이어 붙인 가산기이다.

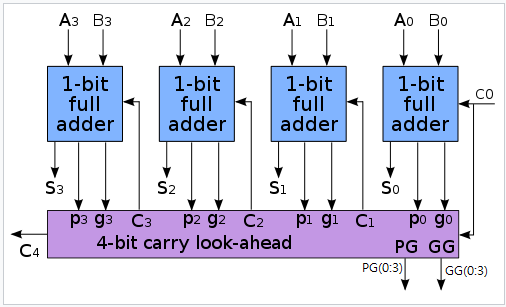


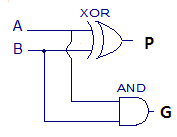
1-bit full adder내부 구조는 1”. 반 가산기 및 전 가산기”를 참조.

따라서 4bit Ripple Carry Adder는 C1을 구하는 데 4 gate delay(이하 gd) C2를 구하기까지 4\*2 = 8gd가 걸리고 C4를 구하려면 16gd가 걸린다. (S3을 구하는 데에는 12+2=14gd가 걸린다.)

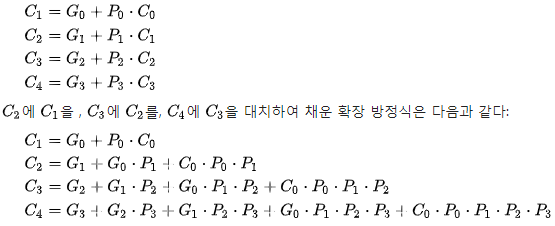
**# Carry Look-Ahead Adder**

Carry Look-Ahead Adder는 이름에 나와 있는 것처럼 앞의 carry를 미리 보겠다는 의미다. 먼저 그림을 살펴 보자.

여기서 1-bit full adder의 회로는 다음과 같다.



Cout = AB+(AⓧB)Cin였음을 기억하자. 여기서 P=AⓧB, G=AB를 뜻한다. 그리고 PG(0:3)은 G3+G2\*P3+G1\*P3\*P2+G0\*P3\*P2\*P1을, GG(0:3)은 P0\*P1\*P2\*P3을 뜻한다. 또한 C1, C2, C3, C4를 다음 방식으로 계산하면,



이다.

이 식을 살펴 보면 1bit full adder(전가산기)를 동시에 4개를 돌려서 P,G값을 1gate delay(이하 gd)안에 뽑아내고 이어서 AND와 OR로 2gd, 총 3gd 안에 모든 carry를 모두 출력할 수 있다. 이어서 S3~S0를 구하는 데에 4gd가 걸린다. (S3 = P3ⓧC3이므로 S3은 1+3=4gd가 걸린다. S2 S1 S0도 4gd 걸린다. ) 즉 Ripple Carry Adder에 비해 월등히 빠르다. 또한 PP와 PG를 16bit carry look-ahead adder에 사용해 월등히 빨라진다. Big-O notaion을 쓴다면 시간이 O(logN)으로 줄어든다.