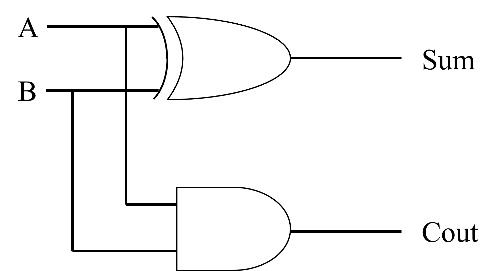
6주차 예비보고서

전공 : 아트앤테크놀로지 학년 : 3학년 학번 : 20191098 이름 : 백승주

* 1. **반 가산기**



반 가산기란 2개의 input 값을 더해 합을 Sum, 자리올림 값을 Carry로 나누어 출력하는 회로이다.

예를 들어 input으로 1,1을 입력했다면 출력값 Sum은 값을 0을 가지고 출력값 Carry는 값 1을 출력한다. 이진수로 표현하면 1+1 = 10 이기 때문에 sum이 0 carry가 1의 값을 가진다. 반대로 입력 값이 0과 1이면 Sum이 1이 되며 Carry 값은 0이 될 것이다.

출력값 Sum은 두 input 값이 동일하면 0을 출력하고 다를 때 1을 출력하므로 XOR 게이트를 통해 표현이 가능하며 출력값 Carry는 input이 모두 1일 때에만 1을 출력하므로 AND 게이트를 통해 나타낼 수 있다.

**1-2. 전 가산기**

텍스트, 바퀴, 교통, 클립아트이(가) 표시된 사진

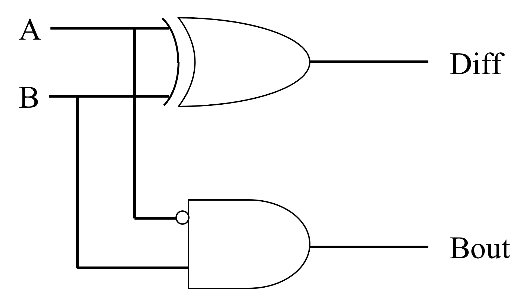
자동 생성된 설명

전 가산기는 반 가산기처럼 덧셈을 하며 Sum과 Carry를 output으로 출력하는 점에서는 같지만 이전 단계에서 발생한 자리올림 값까지 input으로 받아 결과값에 반영하기 때문에 3개의 input을 받는 회로다. 따라서 여러 자리의 이진수를 더할 때 사용하며 전 가산기를 n개 이으면 n bit의 연산을 할 수 있게 된다. 전 가산기의 출력값 Sum은 두 input a,b 의 값이 모두 같고 Carry값이 1인 경우에만 1이 되기 때문에 수식으로 표현하면 (a ⊕ b) ⊕ cin 이다. 또한 출력값 Carry는 a,b가 서로 다르며 Carry 값이 1인 경우에 발생해 수식으로 표현하면 cin(a⊕b)+ab 이다.

예를 들어 2+1을 전가산기로 덧셈을 한다 생각해보자. 이를 2진수 변환하면 11과 01을 더하는 것이다. 2진수 11은 첫 번째 자리가 1이고 2진수 01도 1이다. 따라서 입력 값은 A = 1, B = 1, Cin= 0이 된다. (Cin는 이전 연산에서의 자리올림 값이므로, 0이다.)A와 B를 더한 값에 이전 연산에서의 자리올림 값을 더한 값을 합산합니다. 즉, A + B + C의 값은 1 + 1 + 0 = 2가 되고. 이진수에서는 2진수 1과 같은 단위를 넘어가면, 그 다음 자리에서 1을 올린다. 따라서, 2를 2진수로 표현하면 10이 되므로, Sum 값은 0, Carry out 값은 1이 된다.

다음으로 두 번째 자리에서 전 가산기를 사용하면 carry in 값이 1이므로 A+ B + Cin = 1+0+ 1 = 2로 sum = 0 carry out은 1이 된다. 따라서, 2진수 11과 2진수 01의 합은 2진수 100(4)이며, 전 가산기는 이 결과를 출력한다.

**2-1. 반 감산기**



반 감산기는 2개의 1-bit 입력 값을 빼서 그 차를 출력값 Difference로 빌려 온 수를 출력값 Borrow로 나타내는 회로다. Borrow는 a-b 연산 시 b가 a보다 클 때 즉 a가 0,b가 1인 경우에만 발생하기 때문에 수식으로 나타내면 a’b이다. 출력값 Difference는 a와 b가 서로 다른 값일 때에만 1의 값을 가지기 때문에 수식으로 나타내면 a⊕b이다. 예를 들어 첫째 자릿수의 계싼이 0-1일 경우 앞 자리 bit 자릿수에서 borrow 값을 가져와 borrow 값은 1이 되고 diff는 0이 될 것이다.

**2-2. 전 감산기**

텍스트, 바퀴, 마스크, 클립아트이(가) 표시된 사진

자동 생성된 설명

전 감산기는 반 감산기와 뺄셈 연산을 하고 Difference과 Borrow out을 출력한다는 점에서는 같지만 이전 자릿수에서 중에 bit를 빌려주었는지를 나타내는 입력값 Borrow in을 추가해 3개의 input을 가지는 회로다.

10(2) – 01(1)을 예로 들면 식의 가장 오른쪽 비트부터 빼는데 0에서 1을 빼려면 왼쪽 비트로부터 Borrow를 해야 된다. 따라서 오른쪽 비트 연산의 결과로 Borrow out은 1이 되어 2-0-1 = 1이 되고, 왼쪽 비트를 연산할 때에는 아까 가장 오른쪽 비트에게 1을 빌려주었기 때문에 Borrow in 1이 되어 1-0-1 = 0이기 때문에 최종 결과는 01(1)이 될 것이다. 전 감산기의 Difference는 수식으로 나타내면 a⊕b⊕Bin 이고 Borrow out은 수식으로 나타내면 (a⊕b)’Bin + a’b이다.

**3.** Binary Coded Decimal의 약자인 BCD는 이진수로 표현된 10진수라는 뜻으로 BCD 가산기는 BCD 형태의 두 개의 10진수를 더하는 데 사용되는 디지털 논리 회로이다. BCD는 단순히 10진수를 2진수로 표현하는 것과는 그 방식이 다른데 각 자리수를 4비트로 부호화한다, 각 자리수에 대한 값을 0000(0)부터 1001(9)까지의 이진수로 표현하는 BCD 가산기는 각 자리수마다 BCD 형태의 덧셈을 수행하여 두 수의 합을 구하고 빌림 연산을 이용해 결과를 출력한다.

BCD 가산기는 보통 4비트 BCD 숫자를 처리하지만, 더 큰 자릿수의 BCD 숫자를 처리할 수 있는 BCD 가산기도 존재한다.

예시로, 25-18을 BCD 감산기를 사용하여 계산해보자. 이때, 25를 BCD 형태로 표현하면 0010 0101이고, 18을 BCD 형태로 표현하면 0001 1000이다. 첫 번째 자리에서는 0001 0101에서 1000을 뺀 값인 0000 1101이므로 borrow-out 값은 0이다. 두 번째 자리에서는 0010 0000에서 0001 0000을 뺀 값인 0001 0000이므로 borrow-out 값은 0이다. 따라서, 25-18의 값은 BCD 형태로 0001 0000(16)이다.

**4.** 병렬 가산기와 병렬 감산기는 앞서 조사한 전 가산기나 전 감산기를 여러 개 병렬 연결한 형태의 회로다.

병렬 가산기는 어떤 전 가산기의 출력값인 Carry out이 다음 가산기의 입력값인 Carry in으로 나타나는 형식으로 구성되고 병렬 감산기는 하나의 감산기에서 나온 출력 값인 Borrow out이 다음 전 가산기의 입력 값인 Borrow in 값으 사용된다. 즉 전 가산기나 전 감산기를 n개 사용하면 병렬 가산기나 감산기는 총 n 비트의 덧셈 또는 뺄셈이 가능하다.

텍스트이(가) 표시된 사진

자동 생성된 설명

병렬 가감산기는 병렬 가산기의 입력을 sign 비트와 Xor연산을 해서 가산과 감산 모두 가능하게 만들어주는 회로다. 이를 위해 2의 보수 성질을 이용하는데 이는 A – B가 A + B’ +1 형태로, 즉 덧셈의 형태로 계산할 수 있다고 알려주는 성질이다.

병렬 감가산기는 sign비트가 0이면 덧셈, 1이면 뺄셈을 수행한다. Sign 값을 받고 입력 값 A는 값은 가산기에 집어넣고, 입력값 B는 sign 비트와 XOR 연산하여 가산기에 입력한다. XOR 연산은 1일 때는 b’ 0일 때는 b를 그대로 출력해 sign 비트가 1이면면 b의 보수가 출력되어 가산기에 입력 값이 된다. 이후 sign 비트를 가산기의 입력 값 carry in 으로 대입하면 sign 비트가 1이면 B 값의 보수가 더해지고 sign 비트가 0이면 b값 그대로 더해지는 것이기 때문에 뺄셈 덧셈 모두 가능하게 된다.

**5. Carry Look-Ahead Adder 와 Ripple Carry Adder 비교**

Ripple Carry Adder는 n개의 전 가산기가 병렬 형태로 이어져 하나의 전 가산기의 carry out 비트가 다음 전 가산기의 carry in 값으로 입력되어 총 n비트의 덧셈이 가능한회로다. 각 전 가산기는 이전 가산기의 연산이 끝나야 Carry 값이 전달외어 입력되기 때문에 그 전까지는 연산을 완료할 수 없다. 따라서 연산이 순차적으로 수행되기 때문에 Ripple Carry Adder는 딜레이가 높다.

텍스트이(가) 표시된 사진

자동 생성된 설명

<4 bit Carry Look-Ahead Adder>

이런 Ripple carry Adder가 순차적으로 수행해 delay가 길어지는 점을 보완하기 위해 계산을 동시에 수행하는 회로가 Carry Look-Ahead Adder(CLA)다.

CLA는 각 가산기에 입력되는 a,b 비트의 ab값과 a⊕b 값, 그리고 첫 carry input 값이 존재하면 각 가산기의 carry out 값을 모두 계산 가능하다는 사실을 이용한다. 먼저 CLA는 ab연산과 a⊕b 연산 값을 한꺼번에 받은 뒤 각 자리의 carry out 비트를 연산하고, 그 결과를 다시 각 가산기로 보내어 각 자리의 최종 sum을 한꺼번에 연산할 수 있게 해 딜레이를 줄인다.

**6.**

도표이(가) 표시된 사진

자동 생성된 설명

**1-level CLA**

도표이(가) 표시된 사진

자동 생성된 설명

**2-Level CLA**

4비트가 아닌 16 비트의 연산을 수행하게 되면 4 비트 CLA를 병렬적으로 연결해 결국 앞선 ripple carry adder처럼 이전 CLA의 연산 결과를 기다려야 되기 때문에 불필요한 딜레이가 생길 수 있다. 따라서 이럴 경우 4개의 4 비트 CLA를 다시 감싸는 2-Level CLA를 만들어 딜레이를 줄일 수 있다. 각각의 1-Level CLA를 계산하고 2-Level CLA가 1-level CLA에 필요한 carry를 계산해주기 때문에 딜레이를 줄일 수 있다. 모든 게이트의 딜레이가 x라 가정하면 16 비트 1-level CLA는 10x, 16 비트 2-Level CLA는 8x의 딜레이를 가진다. 큰 차이가 안 나지만, n-Level CLA는 n이 증가할 때마다 딜레이가 logarithm식으로 증가하기 때문에 더 많은 수의 비트를 연산 할 때에 효율적이다.