6주차 예비보고서

전공: 기계공학과 학년: 3학년 학번: 20191820 이름: 김형준

**1.**

가산기(adder)는 2진수의 덧셈 연산을 수행하는 논리회로이며, 디지털 계산장치의 모든 수학연산에 사용되는 기초적인 소자이다. 가산기는 덧셈연산만을 수행하지만, 뺄셈, 곱셈, 나눗셈을 모두 덧셈연산으로 대체할 수 있어 가산기로 사칙연산을 모두 수행할 수 있다.

가산기의 종류에는 반가산기, 전가산기, 리플 캐리 가산기(Ripple Carry Adder), Carry Look-Ahead Adder 등이 있다.

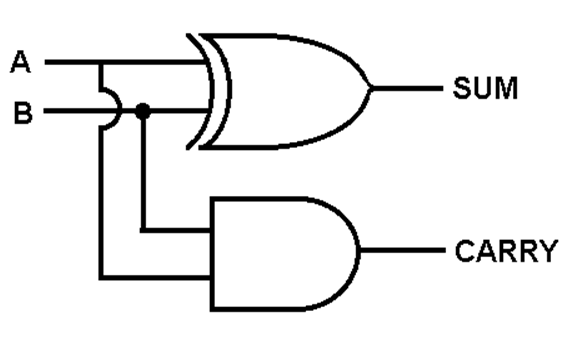
- 반가산기 (Half Adder)

반가산기는 가장 간단한 형태의 가산기로, 이전 자리의 캐리()가 없는 경우에만 사용된다. 반가산기는 2-input, 2-output 구조로, 1비트의 두 이진수(A, B)에 대해 덧셈연산을 수행하여 합(S)과 캐리(C)를 출력한다. 캐리는 자리올림이 발생할 때 발생하며, 합은 두 이진수를 더한 값에서 캐리를 제외한 나머지 값을 갖는다. ()

예를들어 A, B가 (0,1) 또는 (1,0)인 경우 자리올림이 발생하지 않으므로 캐리(C)는 0이되고, 합(S)는 1이 된다. (1,1)인 경우 이므로 자리올림이 발생하므로 캐리(C)가 1이되고, 합(S)는 0이 된다. 따라서 이를 논리식으로 작성하면 , 가 되고, 반가산기의 진리표와 회로도는 아래와 같다.



[반가산기의 진리표]



[반가산기의 회로도]

반가산기는 이전 자리의 캐리()가 없는 경우에만 사용되기 때문에 멀티 비트 가산기에서는 이전 비트가 없는 LSB에서만 사용된다.

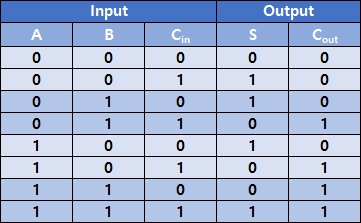
- 전가산기 (Full Adder)

전가산기는 3-input, 2-output 구조로 비트의 두 이진수(A, B)와 이전 자리의 캐리()를 포함해 덧셈연산을 수행하여 합(S)과 캐리()를 출력한다. ()

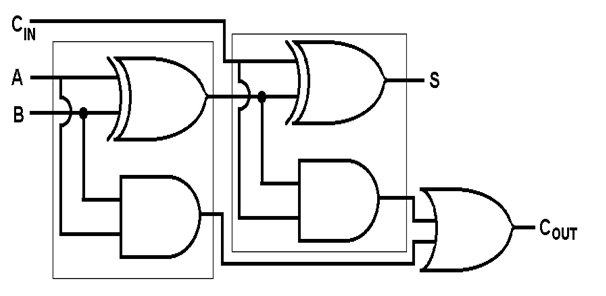
하나의 전가산기는 두 개의 반가산기와 하나의 OR 게이트로 구성된다. (아래의 회로도 참고)

전가산기를 여러 개 사용하여 이전자리의 캐리 출력(을 다음자리의 캐리 입력(으로 연결하면 임의의 자리수의 이진수 덧셈이 가능하다.

전가산기는 반가산기와 달리 멀티 비트 가산기에서 LSB를 제외한 비트에서 사용된다.



[전가산기의 진리표]



[전가산기의 회로도]

위 진리표를 살펴보면, 합(S)는 입력값들 중 1의 개수가 홀수일 때 1이고, 짝수일때 0인 것을 알 수 있다. 또한, 캐리()는 A, B의 값이 서로 다르고 이전 캐리()가 1일 때 1을 갖고, 또는 A, B의 값이 모두 1일 때 이전 캐리()의 값과 관계없이 1을 갖는 것을 알 수있다. 이를 논리식으로 작성하면 , 이다.

예를 들어, A, B, 가 (0,1,1)인 경우 , 이 되고, (1,1,1)인 경우 , 이 된다.

**2.**

이진수의 뺄셈은 빼는 수를 2의 보수 표현을 사용하여 덧셈 연산으로 변환해 수행한다.

이를 이용해 뺄셈을 가산기를 사용해 구현할 수 있으나, 직접 이진법끼리 뺄셈만 할 수 있는 감산기를 만들 수 있다. 감산기에서는 가산기와 달리, 두 이진수의 연산 결과가 합(Sum) 대신 차(Difference)가 되며, 올림수(Carry) 대신 빌림수(borrow)가 발생한다.

- 반감산기 (Half Subtractor)

반감산기는 가장 간단한 형태의 감산기로, 이전 자리의 빌림수()가 없는 경우에만 사용된다.

반감산기는 2-input, 2-output 구조로, 1비트의 두 이진수(A, B)에 대해 뺄셈연산을 수행하여 차(D)와 빌림수(b)를 출력한다.

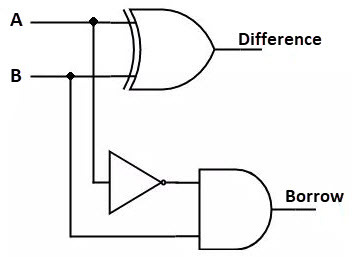
빌림수(b)는 두 이진수의 차가 음수가 되는 경우 발생하며, 차(D)는 두 이진수의 차에 빌림수(b)를 더한 값을 갖는다. ()

예를들어, A, B가 (1,0) 또는 (1,1)인 경우 빌림수가 발생하지 않으므로 빌림수(b)는 0이되고, 차(D)는 각각 1, 0이 된다. (0,1)인 경우 이므로 빌림수가 발생하므로 빌림수(b)가 1이되고, 차(S)는 1이 된다.

따라서 이를 논리식으로 작성하면 , 가 되고, 반감산기의 진리표와 회로도는 아래와 같다.



[반감산기의 진리표]



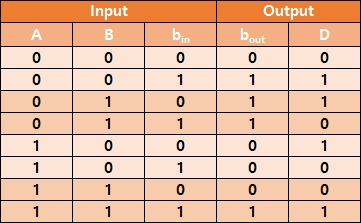
[반감산기의 회로도]

- 전감산기 (Full Subtractor)

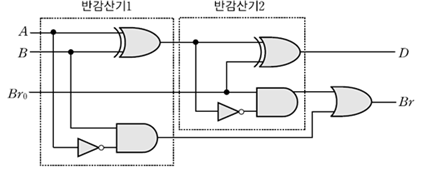
전감산기는 3-input, 2-output 구조로 비트의 두 이진수(A, B)와 아랫자리의 빌림수()를 포함해 뺄셈연산을 수행하여 차(D)와 빌림수()를 출력한다.

출력되는 빌림수()는 가 음수일 때 발생한다. ()

하나의 전감산기는 두 개의 반감산기와 하나의 OR 게이트로 구성된다. (아래의 회로도 참고)



[전감산기의 진리표]



[전감산기의 회로도]

위 진리표를 살펴보면, 차(D)는 입력값들 중 1의 개수가 홀수일 때 1이고, 짝수일때 0인 것을 알 수 있다. 또한, 빌림수()는 A, B의 값이 서로 같고 이전 빌림수()가 1일 때 1을 갖고, 또는 B가 A보다 클 때(, 와 동치) 이전 빌림수()의 값과 관계없이 1을 갖는 것을 알 수 있다. 이를 논리식으로 작성하면 , 이다.

예를 들어, A, B, 가 (1,1,1)인 경우 , 이 되고, (0,1,1)인 경우 , 이 된다.

**3.**

BCD(Binary Coded Decimal)는 이진화 십진수라고 불리며, 4비트의 2진수로 십진수 한자리를 나타내는 기수법을 의미한다.

일반적인 BCD 코드인 8421코드의 경우 10진수 0~9를 4비트의 2진수 0000 ~ 1001까지로 나타낸다. (1010 ~ 1111는 사용되지 않음)

두자리 이상의 10진수를 BCD 코드로 나타내면, 각 자릿수를 4비트의 2진법 코드로 각각

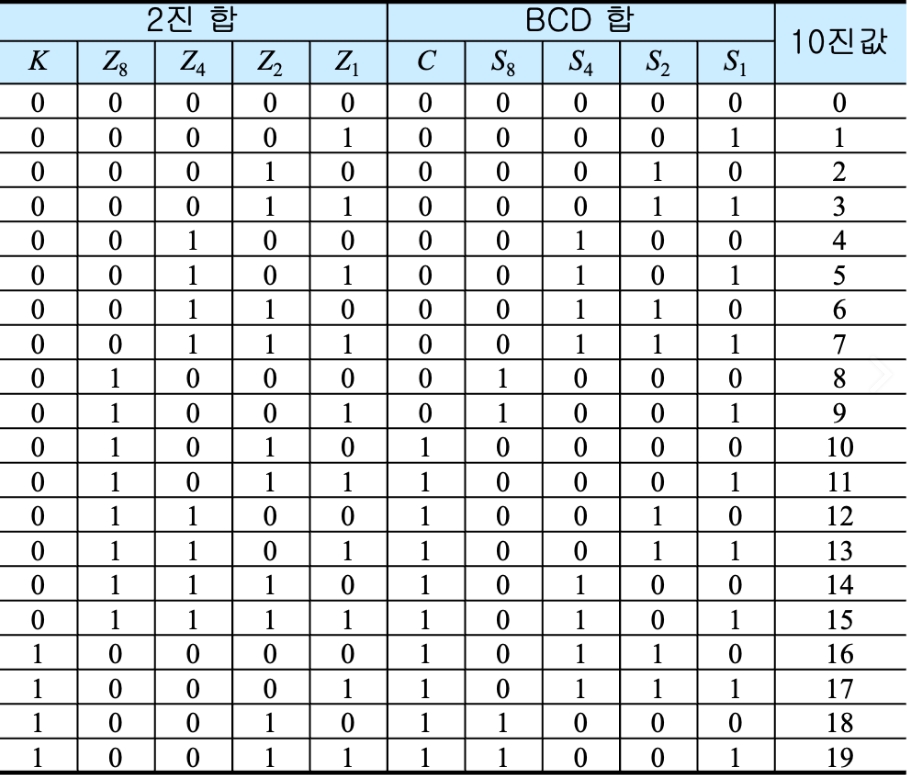
나타낸다. 예를들어 123를 8421코드로 표현한다면, 0001 0010 0011과 같이 표현된다.

BCD 가산기는 한자리 수의 BCD 코드의 덧셈을 수행하는 가산기이다. BCD가산기의 덧셈은

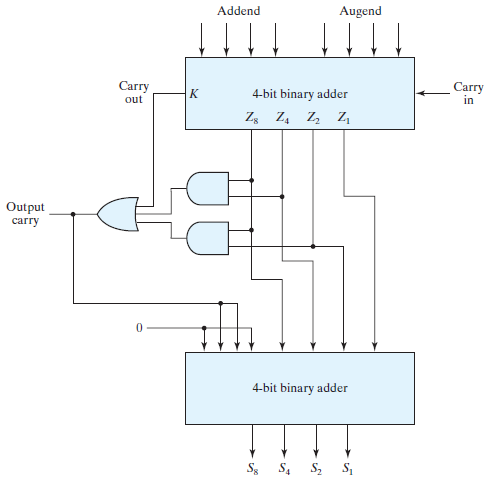
‘BCD값 + BCD값 + 아랫단으로부터의 캐리’의 형태로 2진수 덧셈 방법대로 더하는 방식으로 진행되며, 합의 최대값은 9+9+1=19이다.

합이 9이하인 경우는 별도의 변환없이 그대로 나타내고,

합이 10 이상인 경우 합을 한자리의 BCD 코드로 나타낼 수 없으므로 2진수의 합에 (6)을 더한 다음 발생한 캐리를 앞의 자릿수의 BCD코드의 최하위 비트에 캐리로 올려준다.



[BCD 가산기의 진리표]



[BCD 가산기의 회로도]

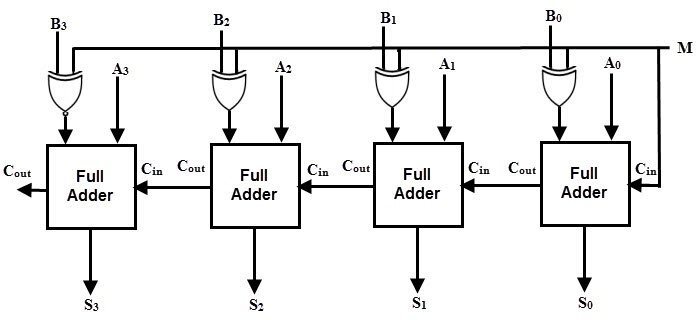
**4.**

병렬 가감산기는 여러 개의 전가산기를 병렬로 연결하여 멀티 비트 이진수의 덧셈과 뺄셈 연산을 수행하는 조합 논리 회로이다.

각각의 전가산기가 입력캐리()로 이전 자리의 출력캐리()를 받는 형태로 되어있어 LSB부터 연속적인 덧셈으로 멀티 비트의 덧셈 및 뺄셈 연산을 구현한다.

n개의 전가산기를 연결하면 n비트인 두 이진수의 덧셈과 뺄셈이 가능하다.

뺄셈(A-B)을 수행하기 위해 각 전가산기의 B에 M(Minus) bit값이 연결된 XOR게이트가 추가되었다.



[4비트 병렬 가감산기의 회로도]

M(minus) 비트는 수행하는 연산의 종류를 결정하며, 덧셈이면 0을, 뺄셈이면 1을 값으로 갖는다.

덧셈(A+B)을 수행하는 경우, M의 값이 0으로 설정되어 LSB의 입력캐리(가 0이 되고,

각 B의 입력이 XOR게이트를 그대로 통과해서( n비트 이진수에 대한 덧셈연산을 수행하게 된다.

뺄셈(A-B)를 수행하는 경우, M의 값이 1로 설정되어 LSB의 입력캐리()가 1이 되고,

각 B의 입력이 XOR 게이트를 통과하면서 반전되어( n비트 이진수에 대한 뺄셈연산을

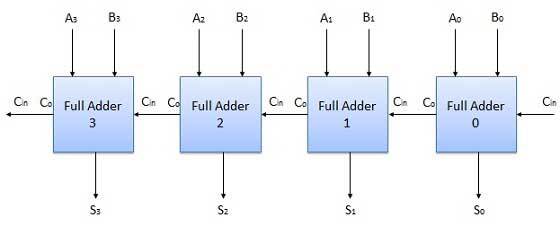
2의 보수 표현으로 바꿔 의 형태로 계산하게 된다.

**5.**

Ripple Carry Adder는 전가산기를 여러 개 연결하여 멀티 비트 덧셈연산을 수행하는 논리 회로로, 단순한 구조를 지닌다.

각각의 전가산기가 입력캐리()로 이전 자리의 출력캐리()를 받는 형태로 되어있고, LSB의 입력캐리()는 0으로 설정해, LSB부터 연속적인 덧셈으로 멀티 비트의 덧셈연산을 구현한다.

그러나 n번째 비트를 연산하기 위해서는 n-1번째의 출력캐리()가 계산되어있어야 하기 때문에 이전비트의 캐리가 모두 연산될 때까지 상위비트의 가산기는 대기해야 하므로 연산하는 이진수의 비트수가 클수록 회로의 지연시간이 커져 연산속도가 매우 느려진다.



[4bit Ripple Carry Adder 회로도]

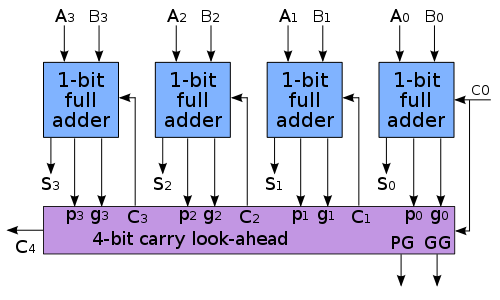
반면에, Carry Look-Ahead Adder는 위의 단점을 해결한 멀티 비트 가산기로서, 캐리를 별도의 로직으로 미리 계산해서 처리하는 방식으로 지연 문제를 해결한다.

각 비트에 대해 해당 자리의 입력값 A, B를 사용해 해당 자리의 입력캐리를 구한다.

식(1) 은 점화식이므로 i=0까지 전개하면,

이 된다. 위 식을 이용하면 이전비트의 캐리의 값을 사용하지 않아도 바로 현재 비트의 캐리를 구할 수 있다. 따라서 위와 같은 방식으로 Carry Look-Ahead Adder는 Ripple Carry Adder와 달리 이전비트의 연산을 기다리지 않고 덧셈 연산을 빠르게 처리할 수 있다.

그러나, 캐리를 미리 계산하는 부분 때문에 Ripple Carry Adder보다 더 많은 논리 게이트가 필요하다는 단점도 존재한다.



[4bit Carry Look-Ahead Adder 회로도]

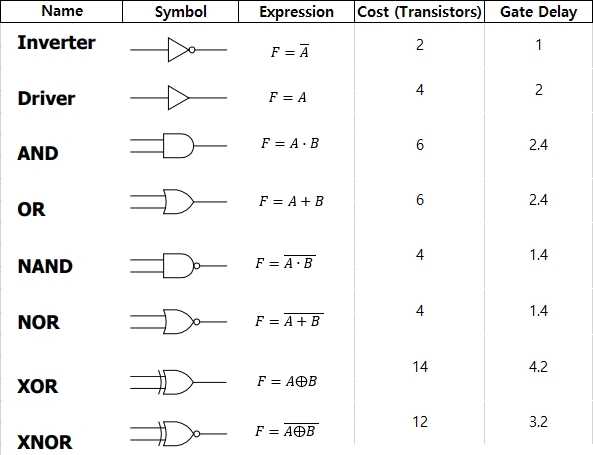
**6.**

기본 논리 게이트들의 비용과 전파 지연시간을 표로 나타내면 아래와 같다.

실제 회로에서 Driver는 Inverter 2개를 직렬로 연결해서 구현하므로

Cost와 Delay가 Inverter의 각각 2배인 것을 확인할 수 있다.

또한, AND와 OR도 NAND와 NOR보다 Delay가 1ns 더 긴 것을 확인 할 수 있는데, 이는 PMOS, NMOS 회로에서 AND와 OR은 각각 NAND와 NOR 뒤에 Inverter를 붙여서 구현하기 때문이다.



[Basic logic library (Gate Delay의 단위는 ns)]