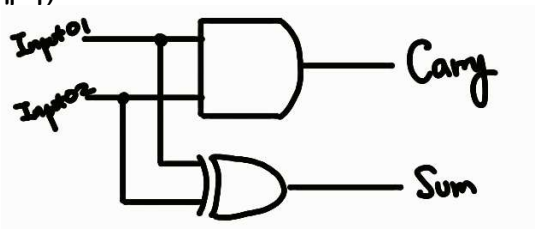
6주차 예비보고서

전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.

반가산기 (Half Adder)는 입력 2개 및 출력 2개가 존재합니다. 각 출력으로는 입력 2개 값의 합(sum)과 올림 수(carry)가 출력됩니다. carry가 발생하는 경우에 대해 서술하자면, 두 개의 입력 값에 대한 계산이 이루어지고 난 뒤 이 결과값이 1 bit를 초과하면 발생하게 됩니다. 다시 말하자면, Carry는 두 입력값이 모두 1일 경우 1이 되며, 그렇지 않을 경우에는 0입니다. Sum은 두 입력 값이 서로 다를 경우 1이 됩니다. 즉, carry 연산을 위해서 두 입력이 모두 1인지 확인해야 하기 때문에 AND를 사용하고, 두 입력값이 서로 다른지의 여부는 XOR을 이용해 확인합니다.

예시)



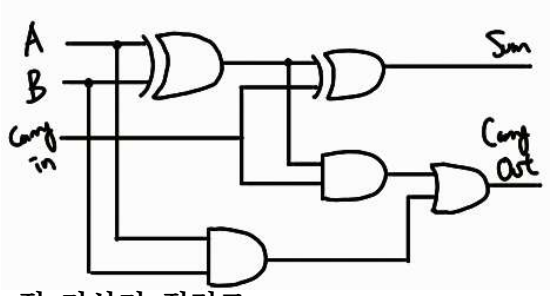
반가산기 회로를 함수로 나타내면, sum = a ^ b, carry = a · b입니다,

<반 가산기 진리표>

입력		출력	
a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

예를 들어, 1과 1이 input의 값으로 들어가면 sum은 0이 나오고 carry로는 1이 출력됩니다. 그러나 이 반가산기로는 한자릿수 계산밖에 할 수 없습니다.

이에 대한 보완책으로 전가산기 (Full Adder)가 있습니다. 전가산기는 반가산기에 자리올림 수(carry in) 입력이 추가된 것을 말합니다. 그렇기에 이전 자릿수의 연산에서 나온 carry out을 입력으로 받아 사용할 수 있습니다. 각 출력으로는 입력 3개 값의 합(sum)과 올림 수(carry out)가 출력됩니다. carry out은 세 개의 입력값 중 2개 이상이 1일 때 발생하게 됩니다. sum은 입력 값 중 1이 홀수 개이면 1이 됩니다. 즉, sum 연산을 위해서는 xor 연산을 사용합니다. 전가산기 회로를 함수로 나타내면, sum = a ^ b ^ carry in, carry out = carry in · (a ^ b) + a · b입니다.



<전 가산기 진리표>

input			output	
a	b	carry in	sum	carry 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

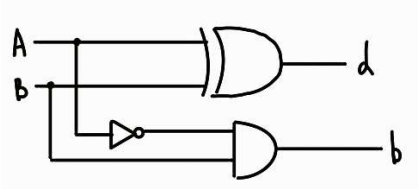
예를 들어, 11 + 11을 계산할 때, 1과 1이 입력으로 들어오면, sum으로 0이 carry out으로 1이 출력이 됩니다. 이때 carry out이 2번째 자릿수의 bit 1, 1과 함께 다음 자리의 입력으로 들어오게 되고, sum이 1, carry는 1이 출력됩니다. 그래서 결과가 110이 나오게 됩니다.

2.

반감산기 (Half Subtractor)는 입력 2개 및 출력 2개가 존재합니다. 각 출력으로는 입력 2개 값의 차(difference)과 빌림 수(borrow)가 출력됩니다. borrow가 발생하는 경우에 대해 서술하자면, 두 개의 입력 값에 대한 계산(A - B)이 이루어질 때, B가 A보다 크면 발생하게 됩니다. Difference은 두 입력 값이 서로 다를 경우 1이 됩니다.

<반감산기 진리표>

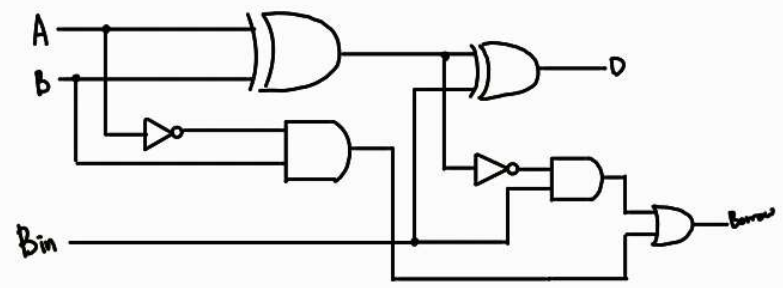
입력		출력	
a	b	Difference	borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



반감산기 회로를 함수로 나타내면, Difference = a ^ b, borrow = (~a) · b입니다. 예를 들어, 1과 0이 input의 값으로 들어가면 difference은 1이 나오고 borrow로는 0이 출력됩니다. 그러나 이 반감산기로는 한 자릿수 계산밖에 할 수 없습니다.

이에 대한 보완책으로 전감산기 (Full Subtractor)가 있습니다. 전감산기는 반감산기에 자리빌림 수(borrow) 입력이 추가된 것을 말합니다. 각 출력으로는 입력 3개 값의 차이(difference)과 빌림 수(borrow)가 출력됩니다. difference은 입력 값 중 1이 홀수 개이면 1이 됩니다. 즉, difference 연산을 위해서는 xor 연산을 사용합니다. borrow는 입력 값 중 2개 이상이 1일 때 발생합니다. 전감산기 회로를 함수로 나타내면, difference=a^b^c, borrow=~(a^b)·c+(~a)·b입니다.
<전감산기 진리표>

입력			출력	
a	b	c	difference	borrow
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



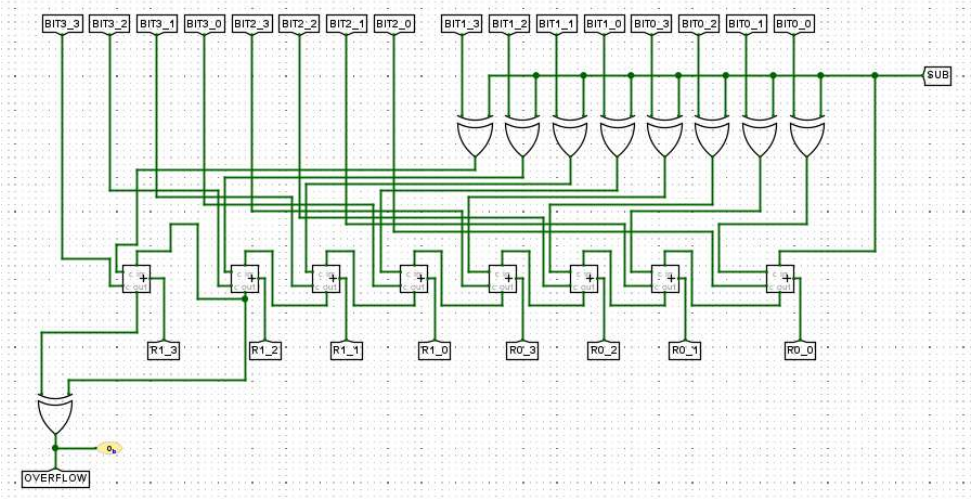
예를 들어, 10 - 01을 계산할 때, 첫 자리수의 비트는 반감산기로, 그 이외에는 전감산기로 계산을 하는데, 0과 1이 반감산기에 입력으로 들어오면, difference는 1이 borrow는 1이 출력됩니다. 이때 borrow가 1, 0과 함께 다음 자리 입력으로 들어오게 되고, difference는 0, borrow는 0이 출력됩니다. 따라서 결과가 01이 나오게 됩니다.

3.

BCD 가산기란 Binary-Coded Decimal를 더하는 회로를 의미합니다. 기존에는 각 자리를 1비트의 2진수로 표현하여 덧셈과 뺄셈을 하기 위한 회로였다면, BCD란 각 자리를 4비트의 이진수로 표현하는 방식입니다. 각 자리가 4비트의 이진수로 표현되기 때문에 0에서 9까지의 값을 표현할 수 있으며 10 이상일 경우 윗 자리에 carry를 보냅니다. 그렇기에 10 이상인 경우라면 기존의 덧셈 방식과는 다른 결과가 초래됩니

다. 과정에 대해 설명하자면, 기존의 결과에 0110을 더해주고, carry는 상위 자릿수로 보냅니다. 예를 들어 9 + 7의 결과인 16이 나왔을 때 기존에는 0001 0000으로 표현했겠지만 이에 0110을 더해주어 0001 0110으로 표현합니다.
이러한 BCD 가산기는 계산 시스템, 디지털 시계 같이 십지수 연산을 정확하게 처리하여 계산의 오류를 방지해야 하는 시스템에서 활용됩니다.

4.
병렬 가산기란 여러 개의 전가산기를 병렬로 연결한 것을 말합니다. 이런 구조상 여러 비트를 동시에 더할 수 있습니다. 주로 Ripple Carry Adder로 구현하며, 각 자리를 sum하고 Carry Out을 다음 자리수의 비트로 전달합니다.
반면, 병렬 가감산기는 병렬 가산기의 b 입력(a - b 중 b)을 부호와 xor함으로써 덧셈과 뺄셈 모두 할 수 있도록 한 회로를 말합니다. 더하기는 0, 빼기는 1로 Sign 비트가 주어집니다. 즉, 더하기일 때 B 입력이 그대로 주어지지만, 빼기일 때에는 B 입력이 보수화됩니다.



5.
앞서 4번 항목에서 첨부한 사진이 Ripple Carry Adder입니다. 가장 낮은 자릿수의 비트인 LSB에서 생성된 캐리가 가장 높은 자릿수의 MSB까지 전파된 후 sum을 serial 하게 계산합니다. 따라서 propagation delay는 (2n + 1) gate delay만큼 걸립니다.(n은 비트 자리수) 왜냐하면, 회로의 속도는 critical path delay(가장 긴 path)에 의해 결정되기 때문입니다.

이를 위해 Carry Look-Ahead Adder는 Ripple Carry Adder과 달리 c_{i+1} 계산을 하기 위해 c_0부터 c_1, c_2, c_3, …, c_i 전달되도록 기다릴 필요 없이 한 번에 계산하여, 계산 시간이 짧아진다는 장점이 있습니다. 단점으로는 회로가 좀 복잡해집니다. 이를

위해 carry out function을 풀어 미리 계산하여 parallel 하게 계산합니다.
Generate function: $g_i = x_i y_i$, Propagate function: $p_i = x_i \oplus y_i$일 때,
carryout은 $c_{i+1} = x_i y_i + x_i c_i + y_i c_i = x_i y_i + (x_i + y_i) c_i = g_i + p_i c_i$ 이렇게 표현할 수 있습니다. 이렇게 carry out들을 바꾸어 c_i로 표현할 수 있게 됩니다.

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

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

6.

Carry Save Adder

기존의 2 inputs와 carry-in을 함께 계산하는 방식과 달리, 부분합과 부분 캐리를 병렬로 각각 계산한뒤, 최종 단계에서 결합합니다. 주로 고속 곱셈기 같은 경우 여러 덧셈 연산을 동시에 빠르게 처리할 수 있어 많이 사용됩니다.

1의 보수 & 2의 보수

두 방식 모두 이진수에서 음수를 표현하기 위한 방식입니다.
1의 보수는 각 비트를 보수화하여 표현하는 방식입니다. 간편하지만, 문제점이라 한다면 1의 보수는 0을 00000000과 11111111 두 가지로 표현하게 되어 중복이 발생합니다.
반면, 2의 보수는 음수를 표현할 때 1의 보수를 구한 후 1을 더합니다. 예를 들자면, 5(00000101)는 1의 보수(00111010)를 구한 후 1을 더하면 11111011이 됩니다. 0을 유일하게 하나로만 표현할 수 있기에 장점입니다. 또한, 덧셈과 뺄셈 연산이 동일한 방법으로 처리되므로 별도의 캐리 처리가 필요 없어 간단합니다.