CSED273 Lab 4 보고서

2022.04.28

20210643 김현준

1. 개요

반가산기(Half adder)와 전가산기(Full adder)의 기능을 이해하여 회로를 구성한다. 특히, 반가산기와 전가산기의 이진수 덧셈 기능에 대해 이해하여, 이를 바탕으로 다양한 회로를 구성한다. 우선, 반가산기와 전가산기를 직접 구현하며, 이를 바탕으로 5-bit 리플 가산기 및 감산기를 구현한다. 또한, 5x3 이진 곱셈기를 구현하여 작동하여 본다. 이때, 테스트벤치를 이용하여 각 회로가 제대로 작동하는지 확인하여, 회로 구성이 알맞게 이루어졌는지 확인한다. 직접 회로를 구성함으로써 수업시간에 배운 각 회로에 대한 이해도를 높인다.

2. 이론적 배경

1) 반가산기 (Half adder)

반가산기는 1-bit 이진수 두 개를 입력받아 합과 Carry out을 출력한다. 즉, 덧셈의 기능을 수행할 수 있도록 올림값과 합을 각각 계산할 수 있는 회로이다.

2) 전가산기 (Full adder)

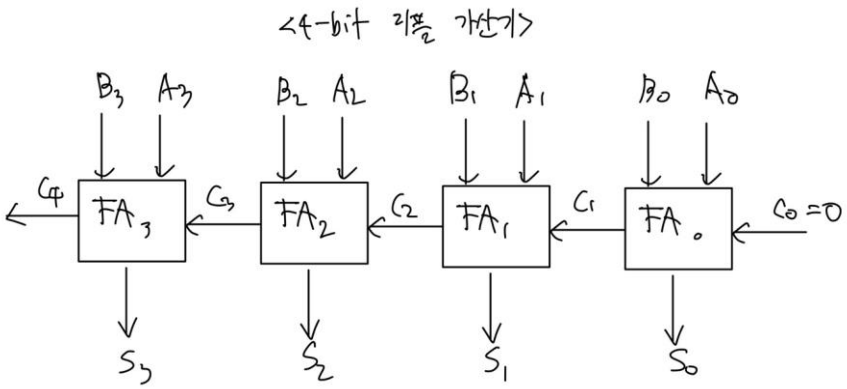
전가산기는 1-bit 이진수 두 개와 이전 가산기의 Carry in을 입력받아 합과 Carry out을 출력할 수 있는 회로이다. 2개의 반가산기를 이용하여 만들 수 있으며, 전가산기를 일렬로 연결하여 덧셈기를 구현할 수 있다.

3) N-bit 가산기

N-bit 가산기는 N-bit 이진수 두 개를 더하여 덧셈의 결과와 Carry out을 출력하는 기능을 수행한다. 다양한 방법으로 N-bit 가산기를 구현할 수 있는데, 앞서 나온 전가산기를 연결하여 간단하게 N-bit 리플 가산기를 구현할 수 있다.

3-1) N-bit 리플 가산기

N-bit 리플 가산기는 전가산기를 일렬로 연결하여 구성한 가산기이다. 첫 전가산기의 Carry in으로는 0이 들어오며, 각 전가산기는 다음 전가산기에게 Carry out을 보내준다. 따라서 각 자릿수에서 계산된 올림값이 다음 전가산기의 덧셈에 더해지는 것으로 덧셈을 구현할 수 있다. 즉, N-bit 리플 가산기는 k번째 자릿수를 계산하는 전가산기의 carry out이 k+1 자릿수의 전가산기의 carry in으로 들어가며 순차적으로 연산이 진행되는 것이다. 하지만, 이 리플 가산기에는 단점이 있는데, 각 전가산기 마다 순차적인 연산이 불가피하여 연산 자릿수가 늘어남에 따라 연산에 걸리는 시간도 증가하게 된다. 이를 방지하기 위해 Carry lookahead adder를 사용할 수 있다.

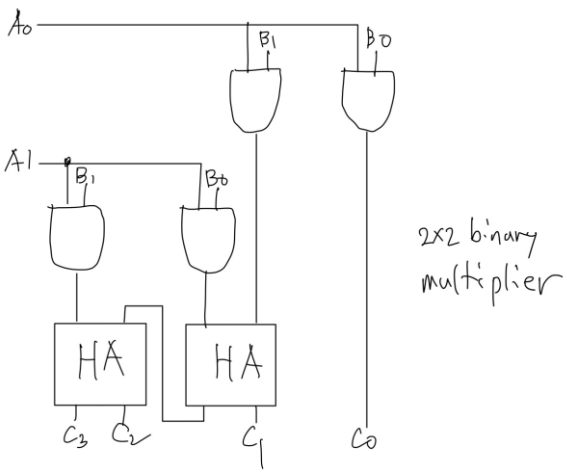


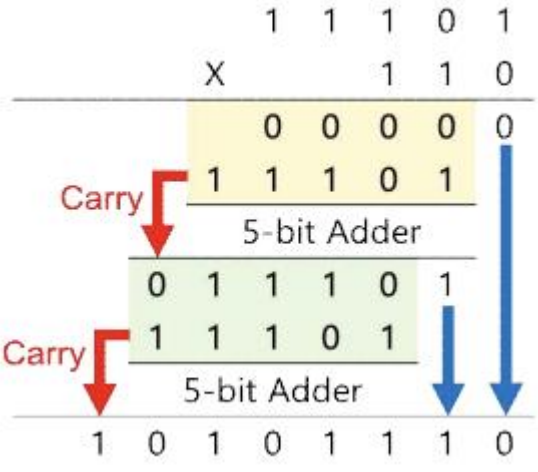
3-2) N-bit 리플 감산기

N-bit 리플 가산기를 이용하여 감산기 또한 간단하게 만들 수 있다. 2의 보수의 성질을 활용하여, 간단하게 N-bit 리플 가산기의 첫 carry out에 0이 아닌 1을 넣고, 뺄 수의 각 입력에 not 게이트를 추가하면 감산기를 구현할 수가 있다. 이때, XOR 게이트를 각 입력에 넣으면 덧셈과 뺄셈을 모두 한번에 수행할 수 있는 Binary parallel adder/subtractor를 만들 수 있다.

4) MxN 이진 곱셈기 (MxN Binary Multiplier)

인간이 곱셈을 하는 방법과 정확히 똑 같은 방법으로, MxN 이진 곱셈은 M-bit의 Multiplicand와 N-bit Multiplier의 각 자릿수의 부분곱을 더하는 방식으로 곱셉기를 구현할 수가 있다. 이때, 각 부분곱은 AND 연산을 통해 구현할 수가 있고, 계산된 각 부분 곱을 모두 더하여 최종 답을 구할 수 있다. 각 부분곱은 M-bit의 크기이기 때문에, 가산기는 M-bit 가산기 N-1개가 필요하게 된다.

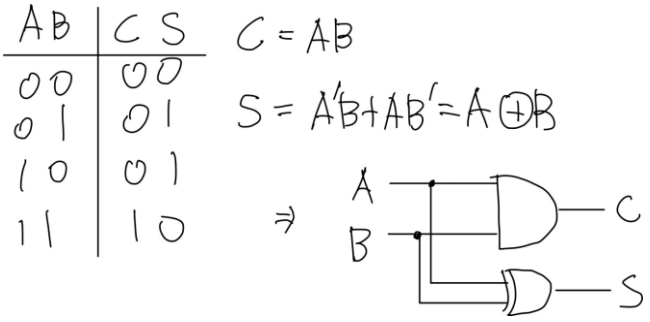




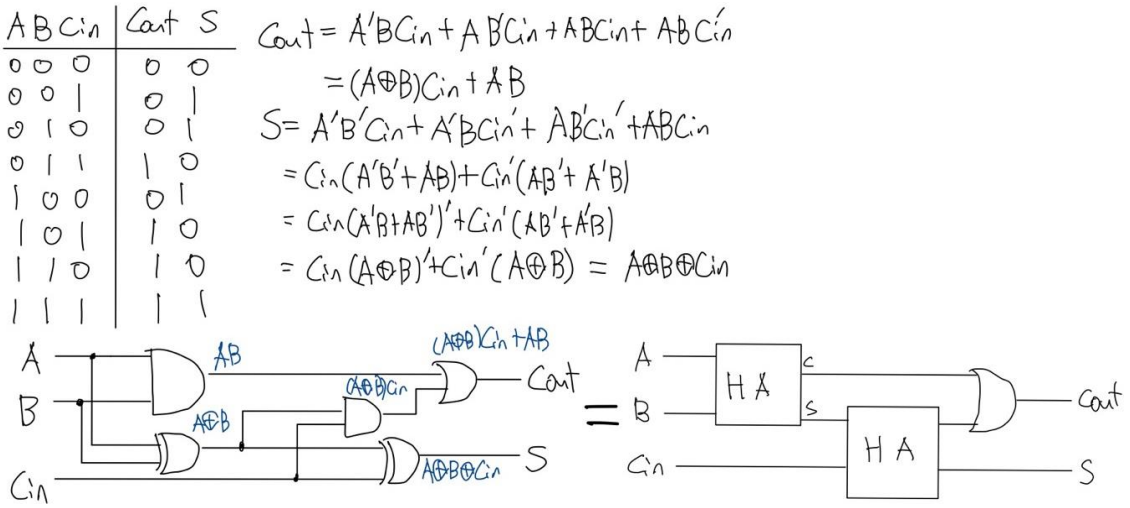
3. 실험 준비

1) 반가산기 및 전가산기 구현

우선 반가산기와 전가산기로 다양한 가산기, 곱셈기를 만들 것이기 때문에 반가산기와 전가산기를 먼저 구현하도록 한다. 반가산기의 진리표는 아래와 같고, 회로도를 그려보면 아래와 같다.

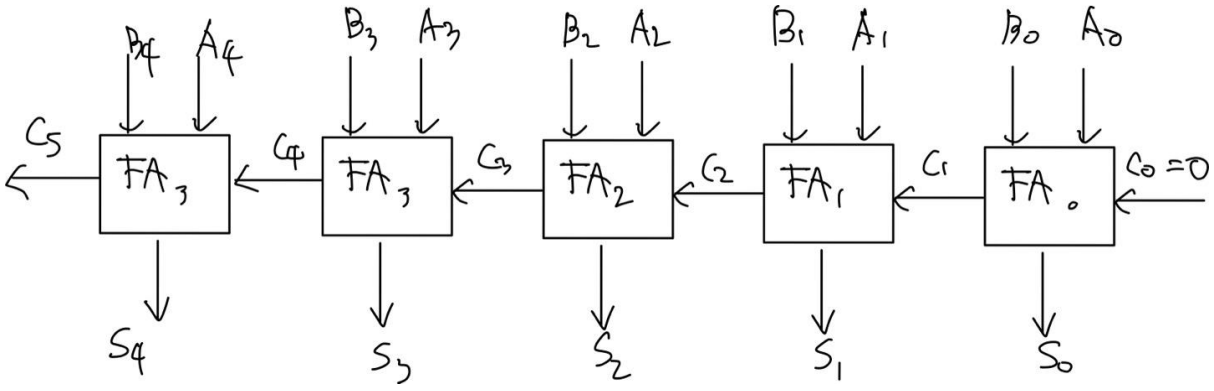


전가산기의 경우 진리표와 회로도는 아래와 같이 나타나는 것을 알 수 있다. 즉, 전가산기는 반가산기 두개와 OR 게이트를 이용해 구성할 수가 있다.



2) 5-bit 리플 가산기

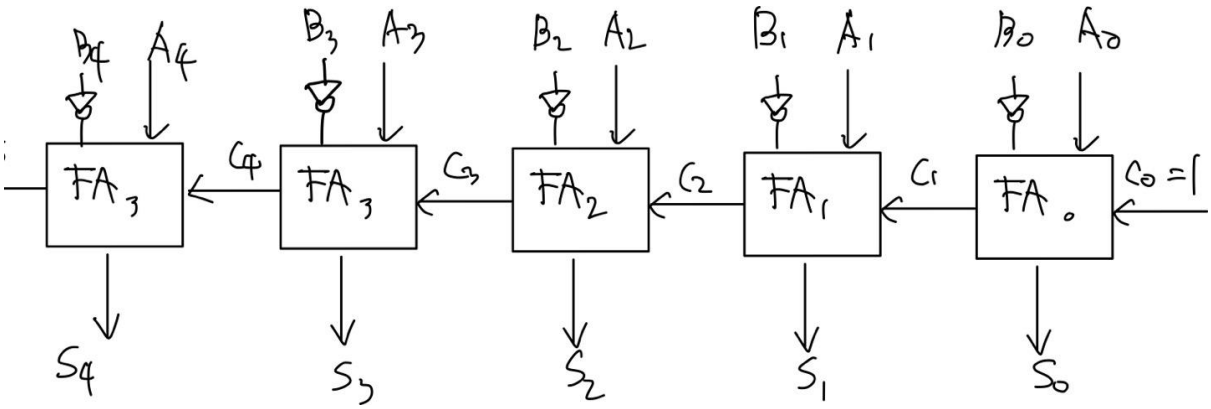
5비트 리플 가산기는 전가산기를 이용하여 간단하게 구현할 수 있다. 전가산기 하나 당, 각 자릿수의 계산을 하고, Carry out을 다음 전가산기의 Carry in으로 입력하면 된다. 5비트 리플 가산기는 이렇게 5개의 전가산기를 연결하면 될 것이다. 회로도는 아래와 같다.



3) 5-bit 리플 감산기

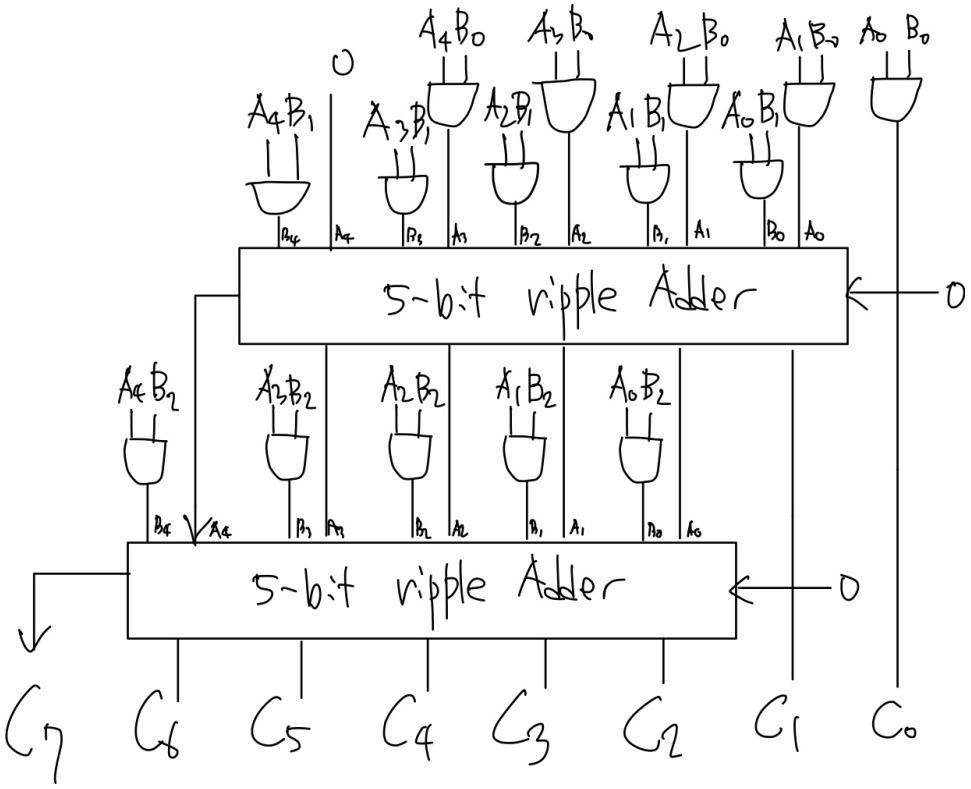
5비트 리플 감산기의 경우, 5비트 리플 가산기의 첫 carry in을 1로 바꾸고, 입력 B에 NOT 게이트를 붙여 구현할 수 있다. 이는 2's complement의 원리를 이용한 것이다. 2's complement에서,

A-B=A+(-B)=A+B'+1 이므로, 첫 carry in에 1을 넣어주고, 입력 B에는 not 게이트를 연결하는 것이다. 회로도는 아래와 같다.



4) 5x3 이진 곱셈기

5x3 이진 곱셈기의 경우, 5비트 리플 가산기 2개를 이용해 만들 수 있다. AND 연산을 통해 3개의 부분곱을 구하고, 이것을 가산기로 더하는 방식으로 곱셈기를 구성한다. 5x3 이진 곱셈기의 회로는 아래와 같이 구성할 수 있다.

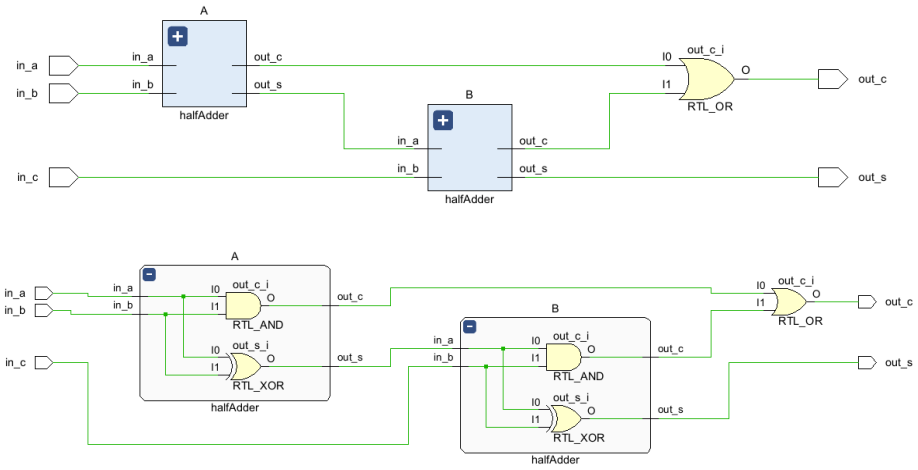


첫 번째 부분곱과 두 번째 부분곱을 자릿수를 맞추어 더해주는 것이 첫 번째 가산기의 역할이고, 그 합과 세 번째 부분곱을 자릿수를 맞추어 더해주는 것이 두 번째 가산기의 역할이다. 부분곱을 구하는 것은 AND게이트들이 수행한다. 이렇게 하면 5비트 이진수와 3비트 이진수의 곱을 출력하는 5x3 이진 곱셈기를 구현할 수 있을 것이다.

4. 결과

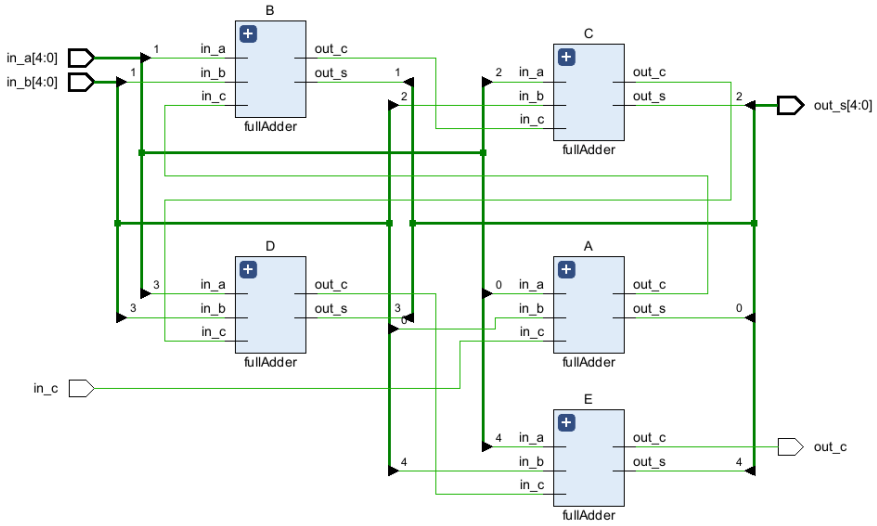
1) 반가산기, 전가산기

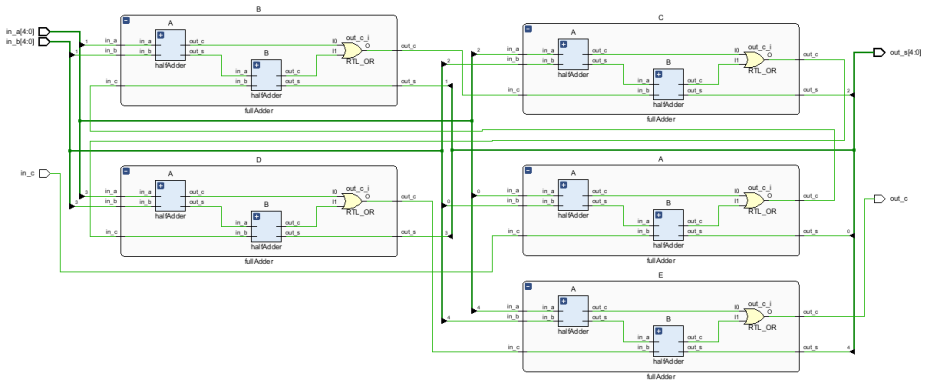
실험 준비 과정에서 구상한 대로, 우선 입력 A, B에 대해 XOR 게이트와 AND 게이트를 통과시켜 각각 S와 C를 출력하는 반가산기(Half Adder)를 구현하였다. 이후, 반가산기 두 개와 OR 게이트를 이용해 실험 준비 과정에서 구상한 대로 전가산기를 구현하였다. 구현한 결과 회로가 아래와 같이 나왔고, 반가산기 각각의 회로도 구상한 것과 같았으며 전체 전가산기도 구상한 것과 동일한 형태로 나타난 것을 확인할 수 있었다. 실험 준비 단계에서 그린 회로도와 정확히 똑 같은 모양의 회로가 구성된 것을 확인할 수 있다.



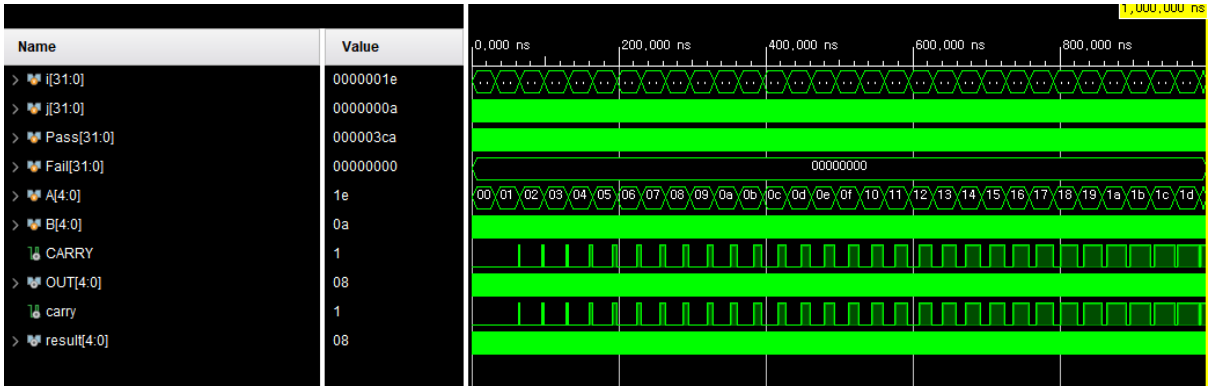
2) 5비트 리플 가산기

실험 준비 과정에서 구상했던 대로, 각 전가산기에 각 자릿수의 이진 입력을 넣고, k번째 자릿수의 전가산기로부터 나오는 carry out을 k+1번째 자릿수의 전가산기의 carry in으로 연결하는 방식으로 5개의 전가산기를 연결하여 5비트 리플 가산기를 구성하였다. 그 결과 실험 준비 과정에서 그린 회로도와 똑같은 회로를 Schematic 기능으로 얻을 수 있었다.



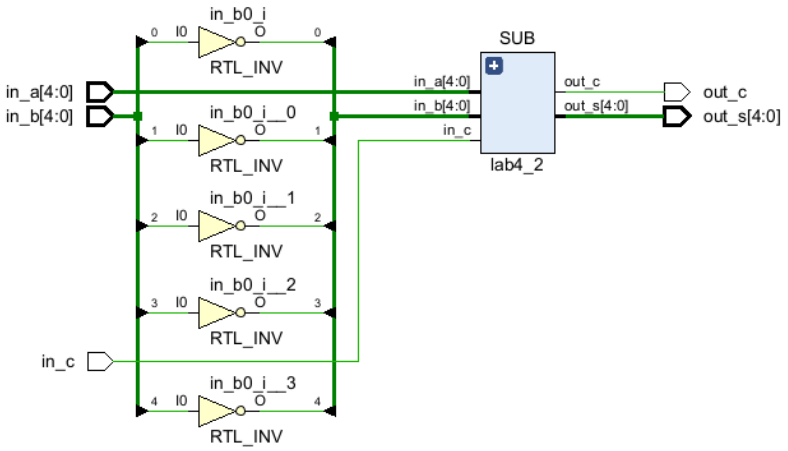


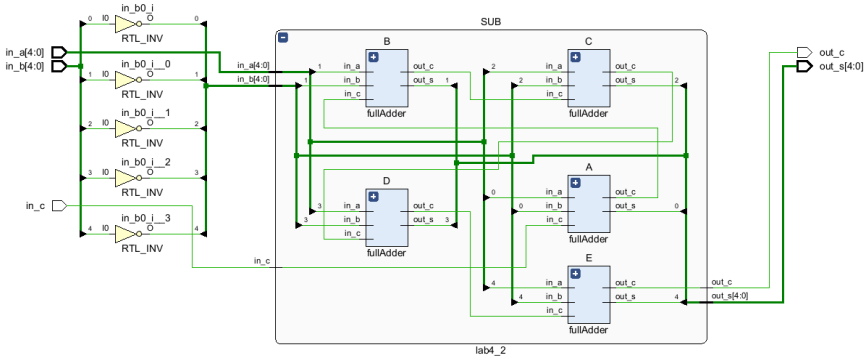
또한 주어진 testbench를 이용하여 simulation을 돌린 결과, 아래와 같은 결과를 얻을 수 있었다. Testbench의 경우 A와 B를 각각 0부터 31까지 증가시키며 5비트 리플 가산기를 이용한 합을 구하는 방식으로 진행되었고, 아래에서 확인할 수 있듯이 Fail 없이 제대로 된 결과가 나온 것을 확인할 수 있다. 예를 들어 A가 1이고 B가 0부터 31까지 증가할 때의 경우 합이 1부터 carry 1, S가 0이 나오기까지 1씩 증가하는 모습을 볼 수 있고, 이는 가산기가 제대로 작동한다는 것을 의미한다. 두 번째 사진에서도 A 01과 B 0b를 더하여 결과 carry 0과 result 0c가 나오는 것을 확인할 수가 있다.



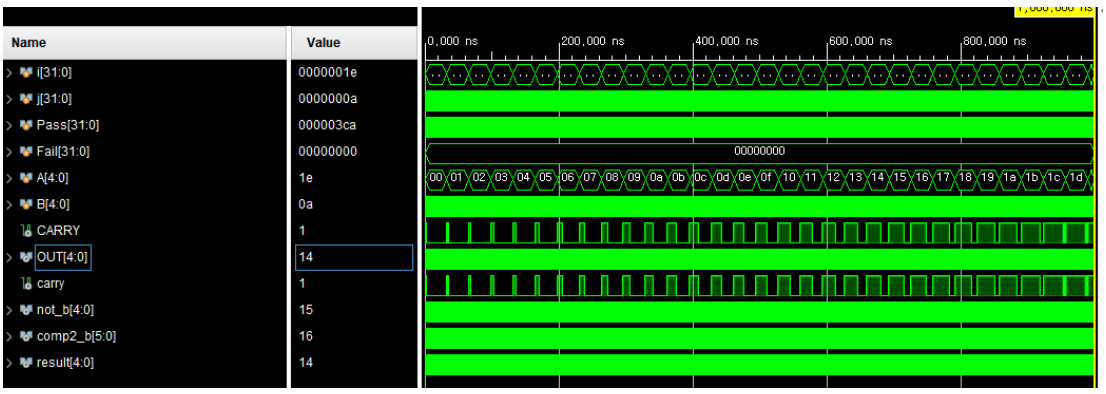
3) 5비트 리플 감산기

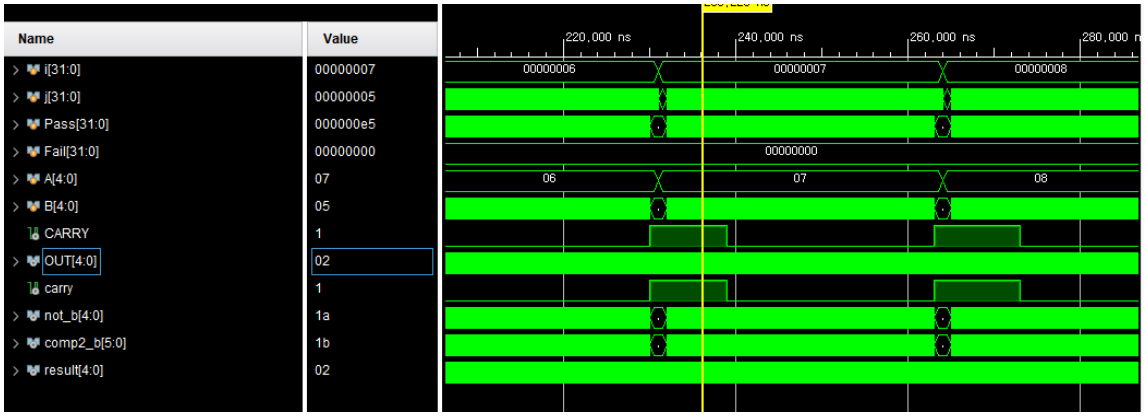
감산기의 경우 가산기를 이용하여 구현할 수 있었다. 실험 준비 과정에서 구성한 것처럼, 빼고자 하는 B의 입력값들을 모두 NOT 게이트로 통과시키고, 가산기로 들어가는 첫 carry in을 1로 함으로써 감산기를 구현하였다. 그 결과, 아래와 같이 실험 준비 과정에서 구상한 것과 동일한 형태의 회로도를 얻을 수 있었다.





또한, 이를 testbench에서 돌려본 결과 아래와 같은 결과를 얻을 수 있었다. 위에서 가산기 구현의 testbench와 비슷한 방식으로, 감산기의 경우에도 A와 B가 0부터 31까지 증가하면서 각각의 상황에 대해 값을 얻는 방식으로 진행되었다. 그 결과, A가 B보다 큰 경우 OUT 값이 뺄셈의 값으로 잘 출력되는 것을 확인할 수 있었고, B가 A보다 큰 경우 OUT 값이 뺄셈 값의 2의 보수로 잘 나타나는 것을 확인할 수 있었다.

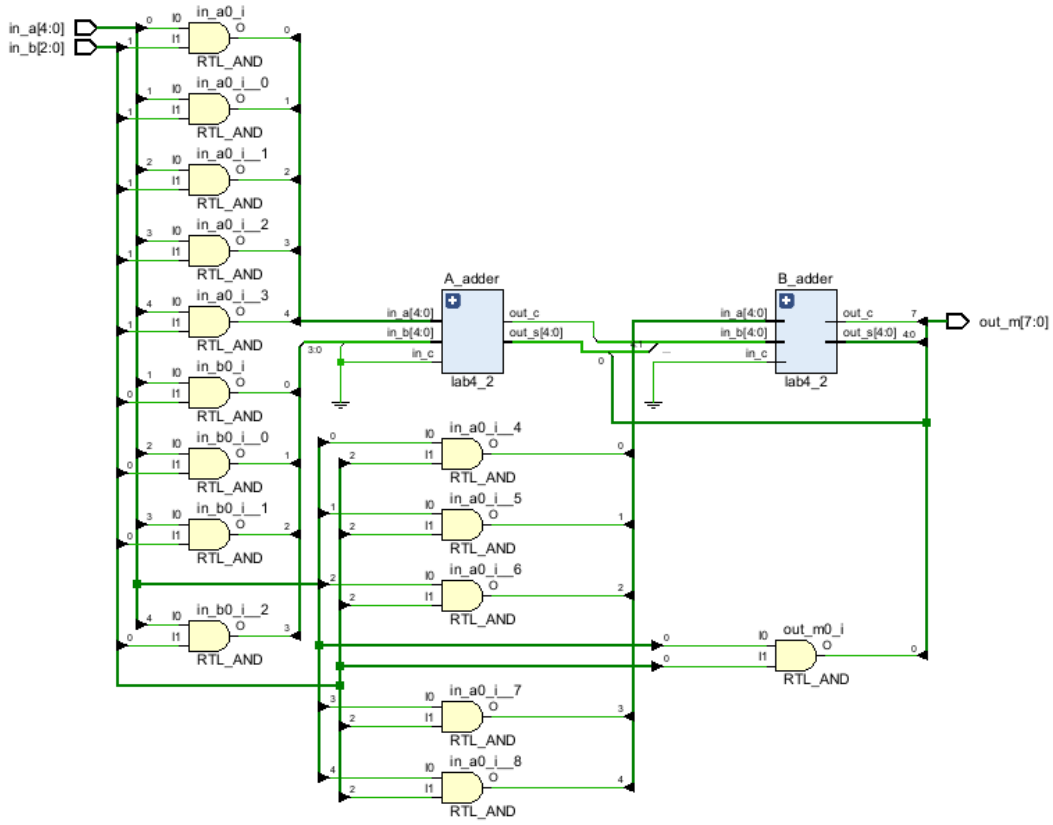


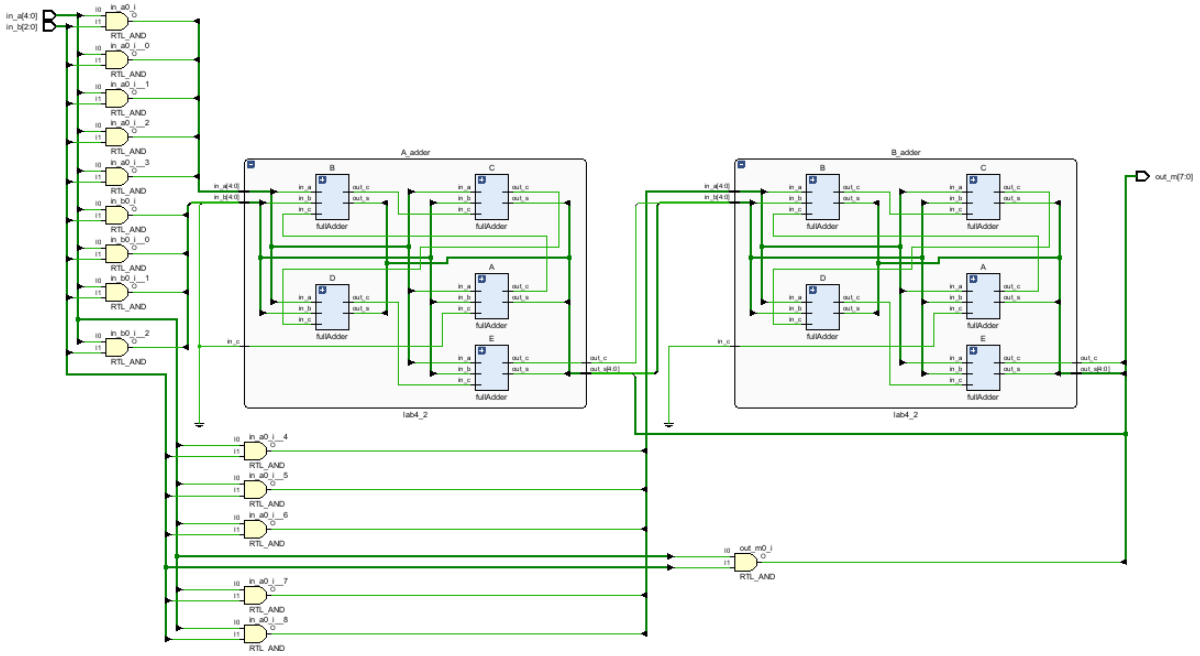


위의 예시 사진에서 볼 수 있듯, A가 7이고 B가 2일 때 OUT이 2로 잘 뺄셈이 구해지는 것을 알 수가 있다. 또한 Fail값도 0으로, 결과적으로 실험 준비 과정에서 구상한 대로 5비트 리플 감산기가 제대로 구현되었음을 알 수 있다.

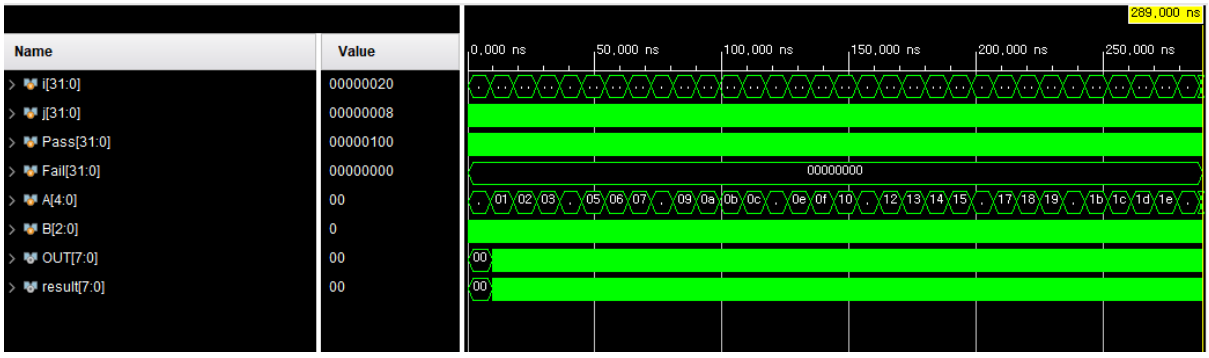
4) 5x3 이진 곱셈기

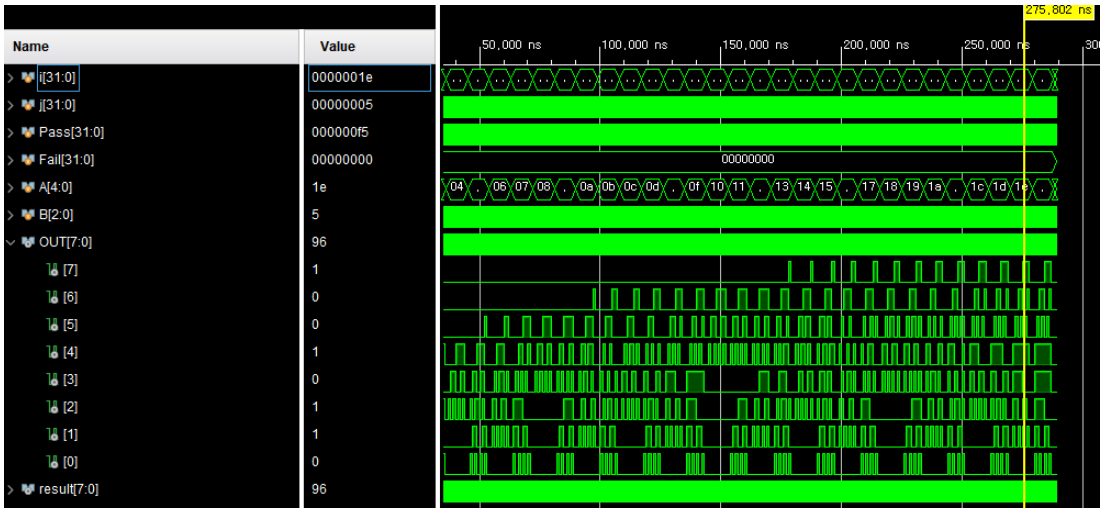
5x3 이진 곱셈기는 실험 준비 과정에서 구상한 것처럼, AND 연산자를 이용해 부분곱을 구하고 이를 두 개의 5비트 리플 가산기로 연결하여 더하는 방식으로 곱을 구할 수 있었다. 준비한 대로 verilog 코드를 짜고, schematic 기능으로 회로를 살펴보니 실험 준비 과정에서 구상한 것과 같음을 확인할 수 있었다. 각 부분곱은 자릿수에 따라 적절히 5비트 가산기로 입력되고, 첫 번째 가산기의 출력과 나머지 부분곱들도 다음 5비트 가산기로 입력되어 최종 7비트의 출력 값이 나오는 형태의 회로를 확인할 수 있었다.





다음으로 주어진 testbench에 따라 해당 회로를 돌려보았다. 그 결과 아래와 같이 나왔는데, A와 B의 곱이 제대로 OUT으로 출력되는 것을 확인할 수 있었다. Fail 또한 0으로, 값이 다른 오류는 발생하지 않았음을 알 수 있었다. 예를 들어, A가 1e, B가 5일 때 그 곱이 95로 제대로 나타나는 것을 확인할 수 있었다. (10진수로는 A가 30, B가 5, 곱이 150) 따라서, testbench의 결과에 따르면 5x3 이진 곱셈기는 제대로 작동하는 것을 알 수 있다.





5. 논의

본 lab4를 통해서 2진수의 덧셈, 뺄셈, 곱셈을 회로로 어떻게 구현하는지 이해할 수 있었다. 더불어 수업 시간에 배운 내용을 복습하며 개념을 확고히 할 수 있었다. 특히, 5x3 이진 곱셈기를 구현하고 회로를 작성하면서, ppt에는 없는 내용에 대해 직접 회로를 짜 볼 수 있었고, 이를 통해 이해도를 더욱 높일 수 있었다. 또한, 처음에 testbench를 확인하고는 온통 초록색으로 칠해져 있어서 오류로 착각하고 헤맸던 적이 있는데, 이내 확대하는 법을 알게 되어 제대로 작동했다는 것을 확인할 수 있었다. 추가로 본 lab4에서는 감산기의 경우 c_in을 1로 그냥 설정해서 testbench를 돌렸는데, c_in의 값에 따라 감산기, 가산기의 기능을 모두 수행할 수 있는, XOR 게이트를 이용한 감/가산기도 한 번 만들어보고 싶다.