Lab 4. 이진수 연산

Digital System Design (CSED273)

20220100 박기현

1. 개요

이번 실험은 이진수 덧셈에 사용하는 반가산기(Half adder)와 전가산기(Full adder)의 기능을 이해하고, verilog 를 이용하여 회로를 구성한다. 또한, 앞서 구현한 전가산기를 이용하여 5-bit 리플 가산기를, 5-bit 리플 가산기를 이용하여 5-bit 리플 감산기를 구현한다. 마지막으로 5-bit 리플 가산기로 5x3 이진 곱셈기를 구현함으로써 이진수의 덧셈, 뺄셈, 곱셈의 연산에 대해 학습하는 것이 최종 목표이다. 자세한 학습 목표는 다음과 같다.

- 반가산기와 전가산기 구현
- 5-bit 리플 가산기/감산기 구현
- 5x3 이진 곱셈기 구현
- 테스트 벤치를 이용한 회로 검증

2. 이론적 배경

1) Carry out/Carry in

Carry out 은 연산 이후 발생한 자리 올림을 의미하고, Carry in 은 이전 자리에서 발생한 자리 올림을 의미한다.

2) 반가산기 (Half adder)

반가산기는 1-bit 이진수 두 개를 입력으로 받아 합과 Carry out 을 출력한다.

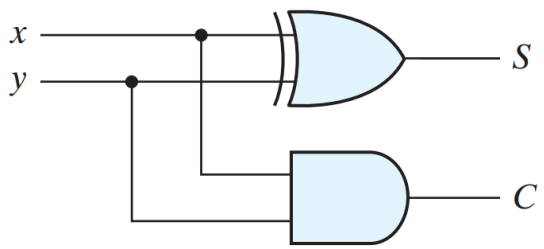


그림 1. 반가산기

3) 전가산기 (Full adder)

전가산기는 1-bit 이진수 두 개와 이전 가산기의 Carry in 을 입력으로 받아 합과 Carry out 을 출력한다. 두 개의 반가산기를 사용하여 구현할 수 있다.

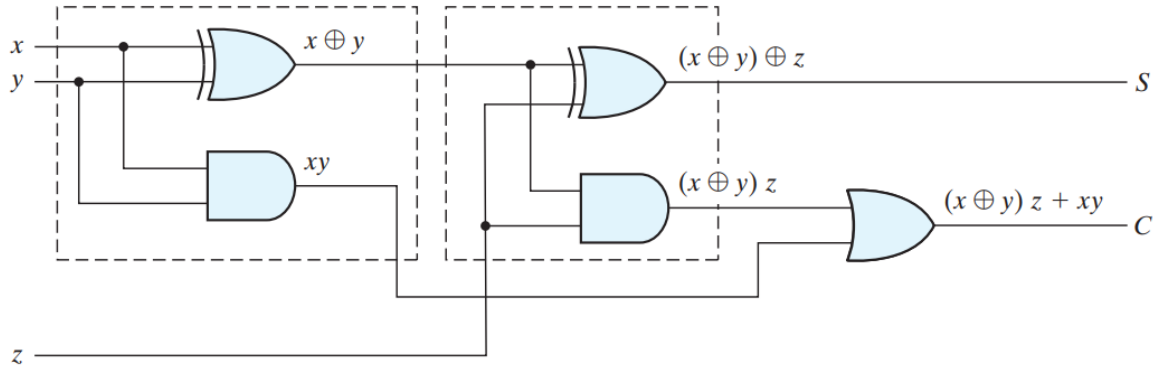


그림 2. 전가산기

4) 2 의 보수 (2's complement)

2 의 보수는 어떤 수를 커다란 2 의 제곱수에서 빼서 얻은 이진수를 의미한다. 2 의 보수를 구하는 방법은 먼저 1 의 보수를 구한 후, 1 을 더해주면 된다.

본 실험에서는 5-bit 를 다루기 때문에 5-bit 의 2 의 보수를 나타내면 다음 표와 같다.

Binary number	Decimal
00000	0
00001	+1
00010	+2
00011	+3
00100	+4
00101	+5
00110	+6
00111	+7
01000	+8

01001	+9
01010	+10
01011	+11
01100	+12
01101	+13
01110	+14
01111	+15
10000	-16
10001	-15
10010	-14
10011	-13
10100	-12
10101	-11
10110	-10
10111	-9
11000	-8
11001	-7
11010	-6
11011	-5
11100	-4
11101	-3
11110	-2
11111	-1

5) N-bit 리플 가산기/감산기 (N-bit Ripple Adder/Subtractor)

N-bit 가산기는 N-bit 이진수 두 개를 입력으로 받아 N-bit 덧셈 결과와 Carry out 을 출력한다. N-bit 가산기의 구현 방법에는 여러 가지가 존재하는데, 본 실험에서는 N-bit 리플 가산기를 구현한다.

N-bit 리플 가산기는 N 개의 전가산기를 순차적으로 연결하여 구현한다. N 개의 전가산기는 각각 N-bit 중 한 자릿수의 연산을 수행한다. 이때 k 번째 자릿수를 계산하는 전가산기의 Carry out 은 K+1 번째 자릿수를 담당하는 전가산기의 Carry in 으로 입력되어 가장 낮은 자릿수부터 가장 높은 자릿수까지 Carry 가 순차적으로 전파(propagation)된다. 이러한 순차적인 연결 구조로 인해 낮은 자릿수의 연산이 수행되어야만 Carry out 이 결정되어 높은 자릿수의 연산을 수행할 수 있다. 즉, 연산하고자 하는 자릿수가 늘어날수록 연산을 수행하는 시간도 이에 비례하여 늘어난다.

N-bit 리플 감산기를 구현하기에 앞서 A-B 꼴의 뺄셈은 A+(-B)의 꼴의 덧셈으로 표현할 수 있다. 이러한 성질과 2 의 보수(2's complement)를 활용하면 N-bit 리플 가산기를 사용하여 N-bit 리플 감산기를 쉽게 구현할 수 있다.

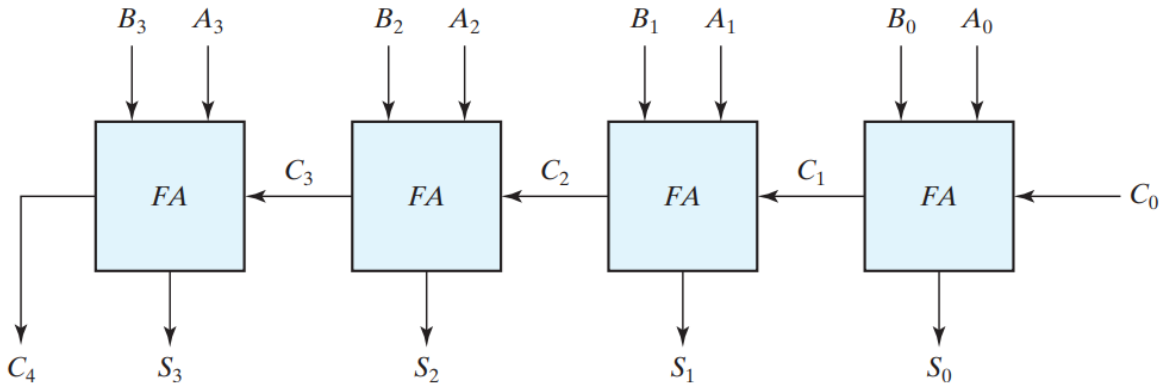


그림 4. 4-bit 리플 가산기

6) MxN 이진 곱셈기 (MxN Binary Multiplier)

MxN 이진 곱셈은 M-bit Multiplicand 와 N-bit Multiplier 의 각 자릿수의 부분 곱을 통해 얻은 이진수 N 개를 합하여 계산 가능하다. 부분 곱은 bitwise AND 연산을 통해 구현 가능하며, 가산기로 부분 곱을 모두 합하여 답을 구한다. 이때, 다음과 같이 각 부분 곱을 더하는 과정을 M-bit 리플 가산기 N-1 개로 구할 수 있다.

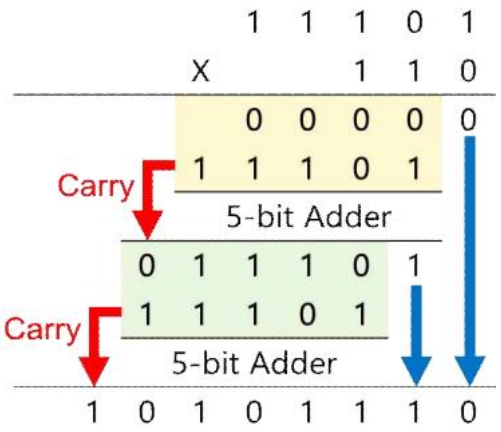


그림 5. 5x3 이진 곱셈

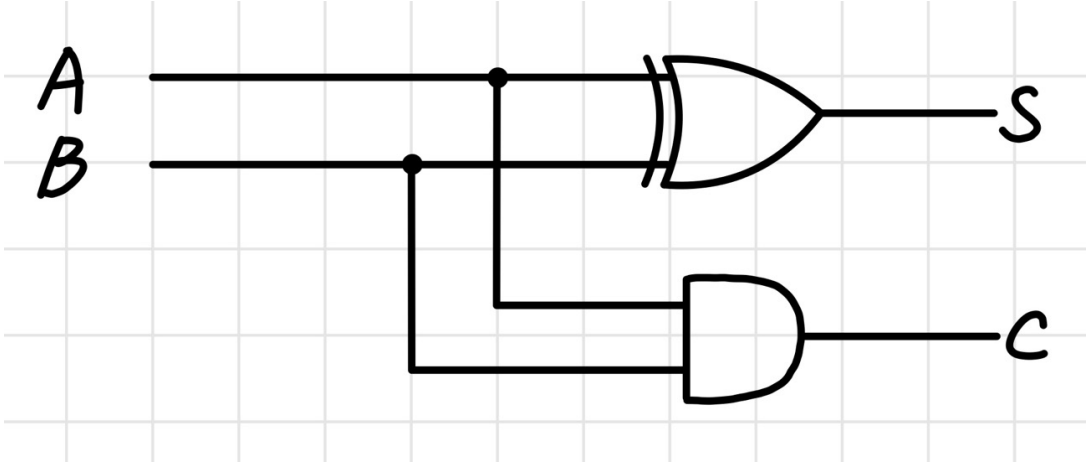
3. 실험 준비

1) 반가산기의 진리표와 식을 구하고 회로를 그린다.

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S = A'B+AB' = A⊕B

C = AB

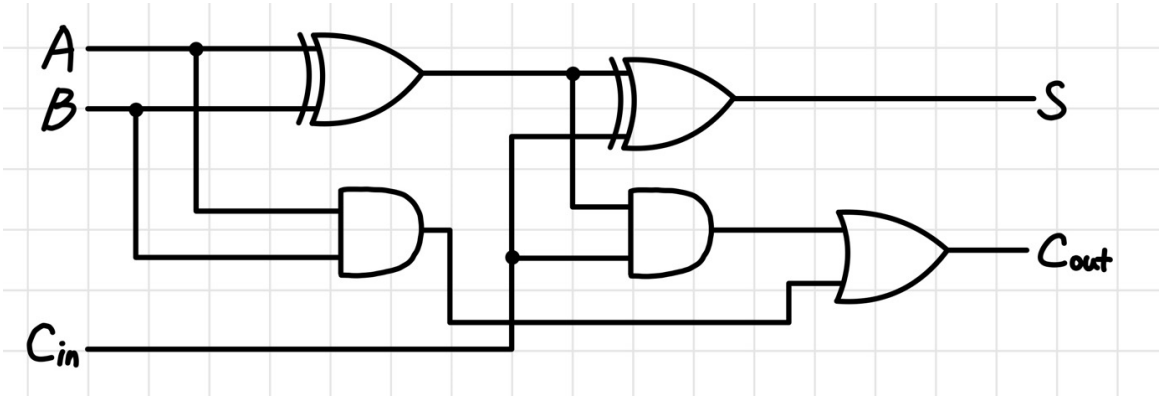


2) 전가산기의 진리표와 식을 구하고 반가산기를 사용해 전가산기 회로를 그린다.

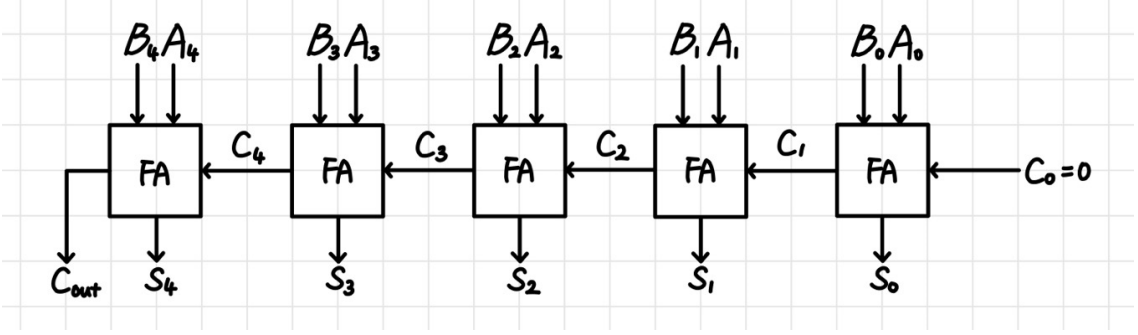
A	B	C_in	C_out	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

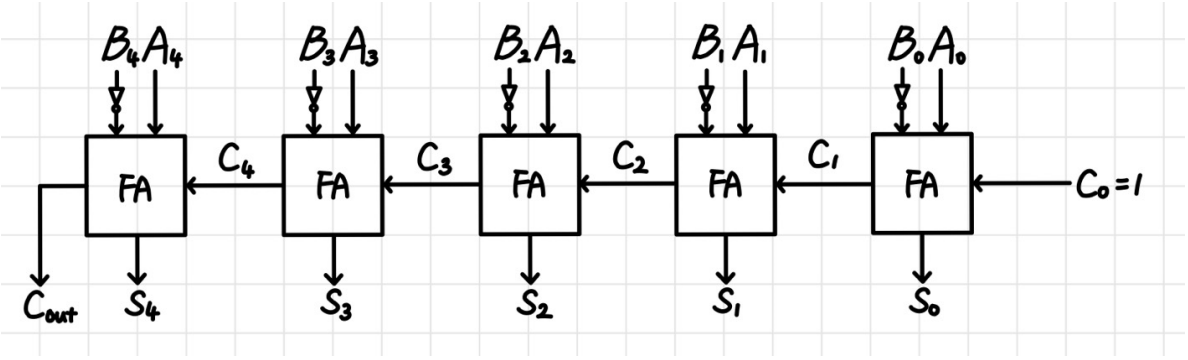
$S = A'B'C_{in}+A'BC_{in}'+AB'C_{in}'+ABC_{in} = (A'B'+AB)C_{in}+(A'B+AB')C_{in}' = (A'B+AB')'C_{in}+(A'B0+AB')C_{in}' = A\oplus B\oplus C_{in}$

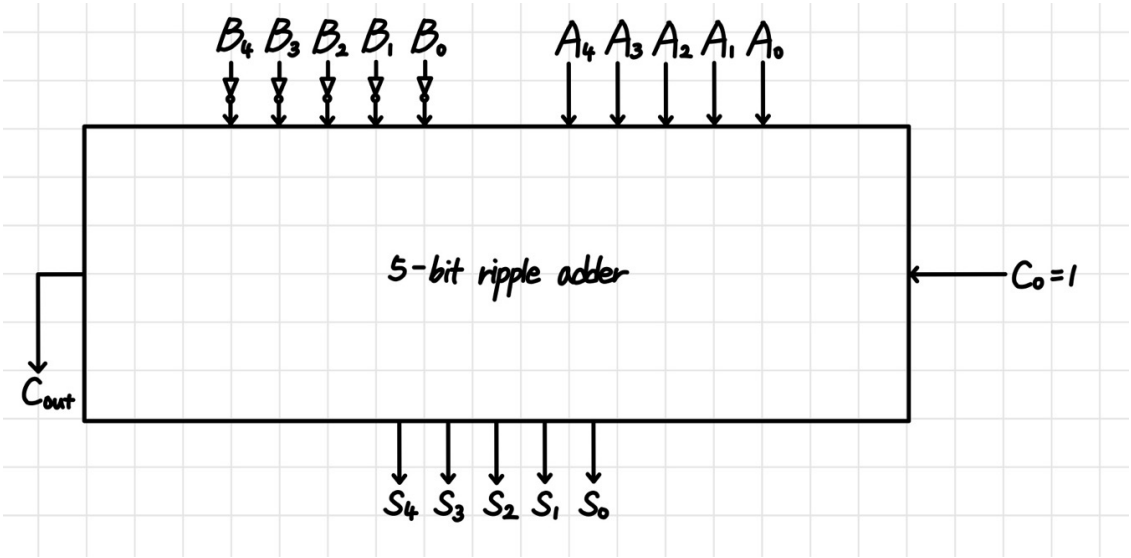
$C_{out} = AB+AC+BC$ 또는 $AB+A'BC+AB'C = AB+(A\oplus B)C$



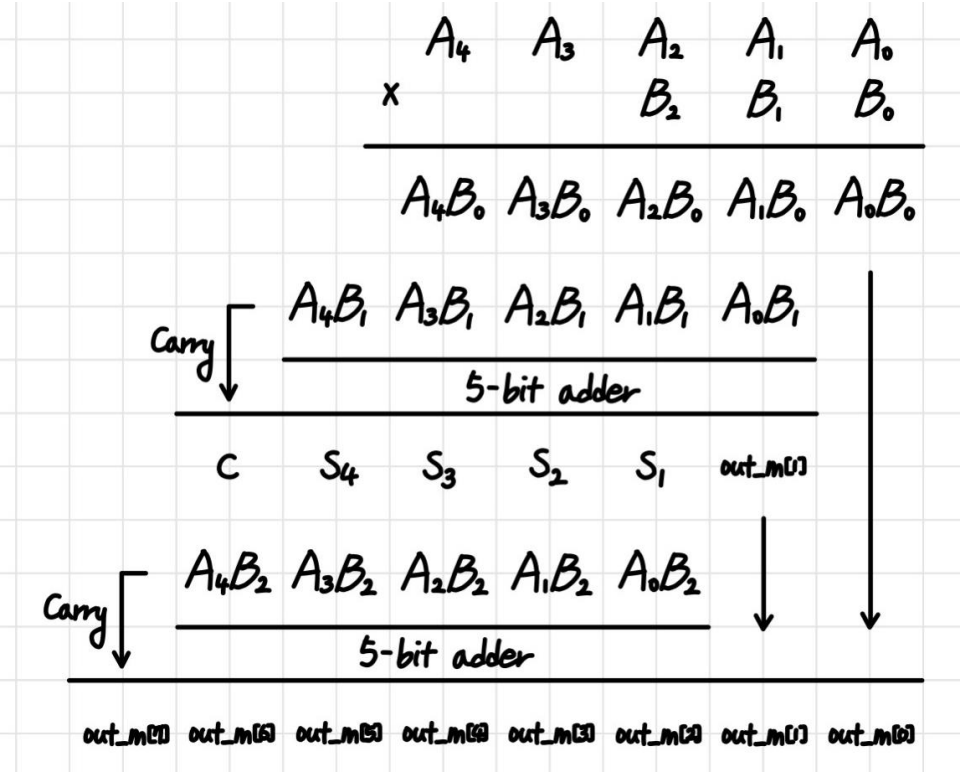
3) 전가산기를 사용해 5 비트 리플 가산기와 감산기 회로를 그린다.

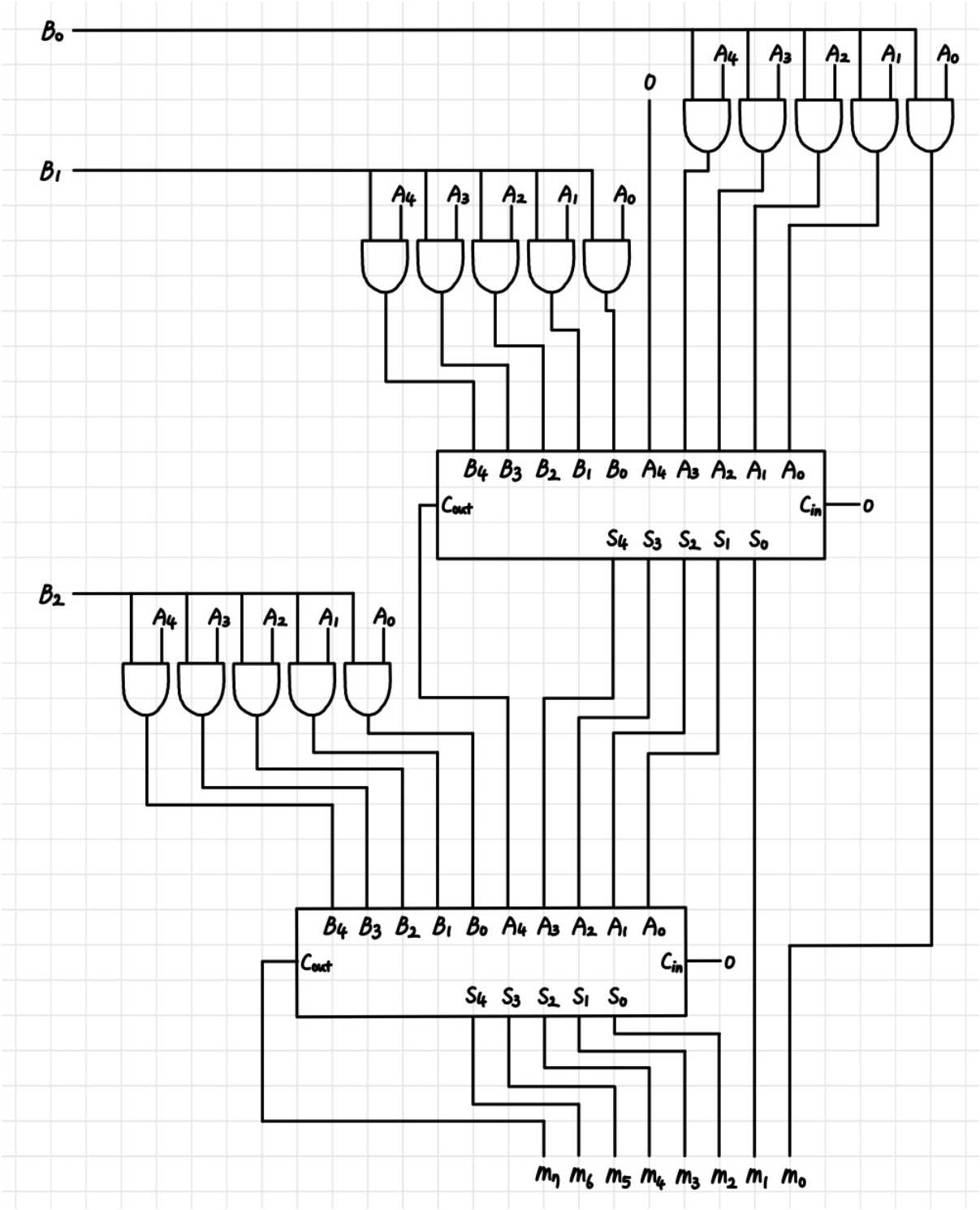






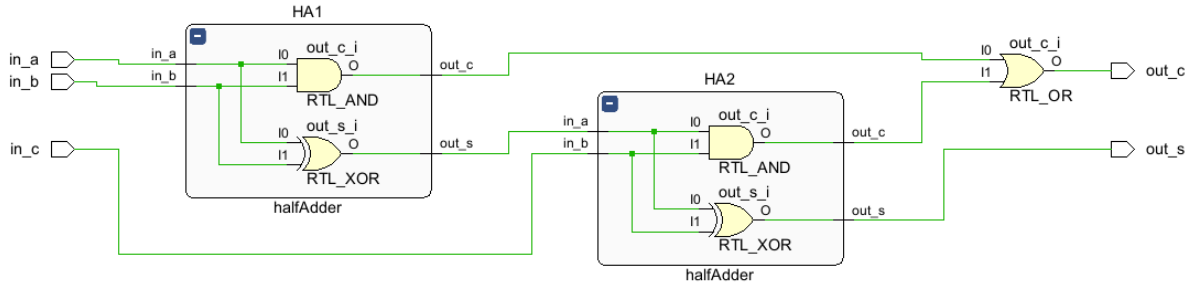
4) 5 비트 리플 가산기를 사용해 5x3 이진 곱셈기 회로를 그린다.


A4 A3 A2 A1 A0
x B2 B1 B0
A4B0 A3B0 A2B0 A1B0 A0B0
Carry
A4B1 A3B1 A2B1 A1B1 A0B1
5-bit adder
C S4 S3 S2 S1 out_m[1]
Carry
A4B2 A3B2 A2B2 A1B2 A0B2
5-bit adder
out_m[7] out_m[6] out_m[5] out_m[4] out_m[3] out_m[2] out_m[1] out_m[0]



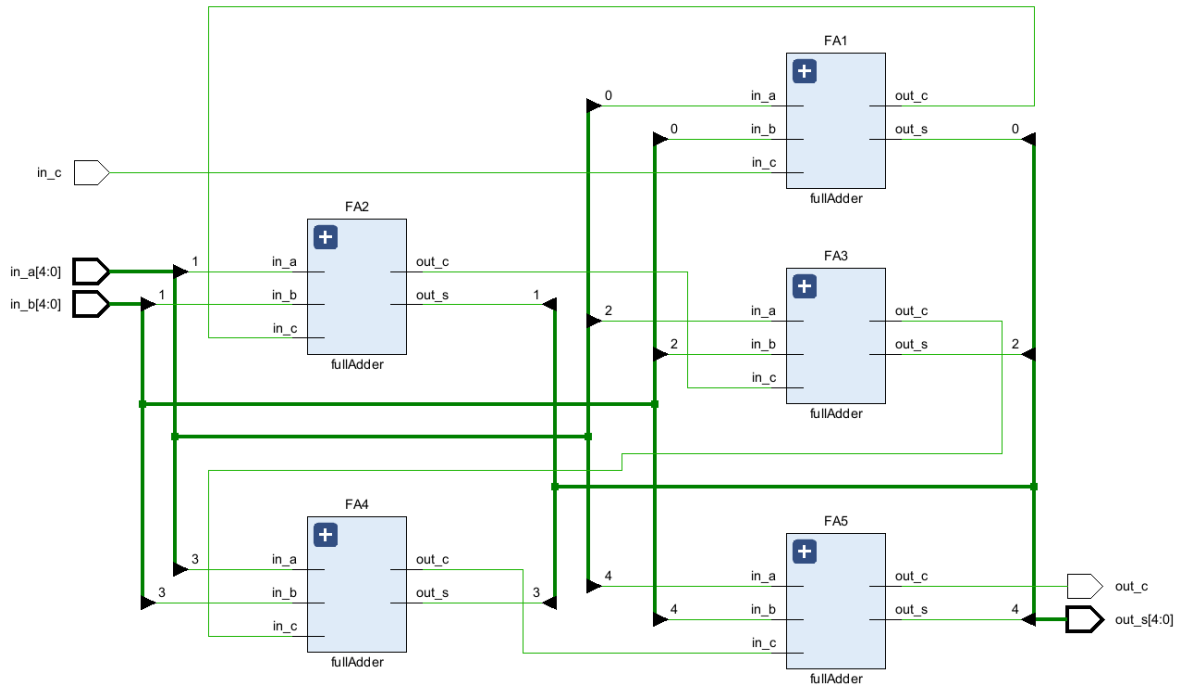
4. 결과

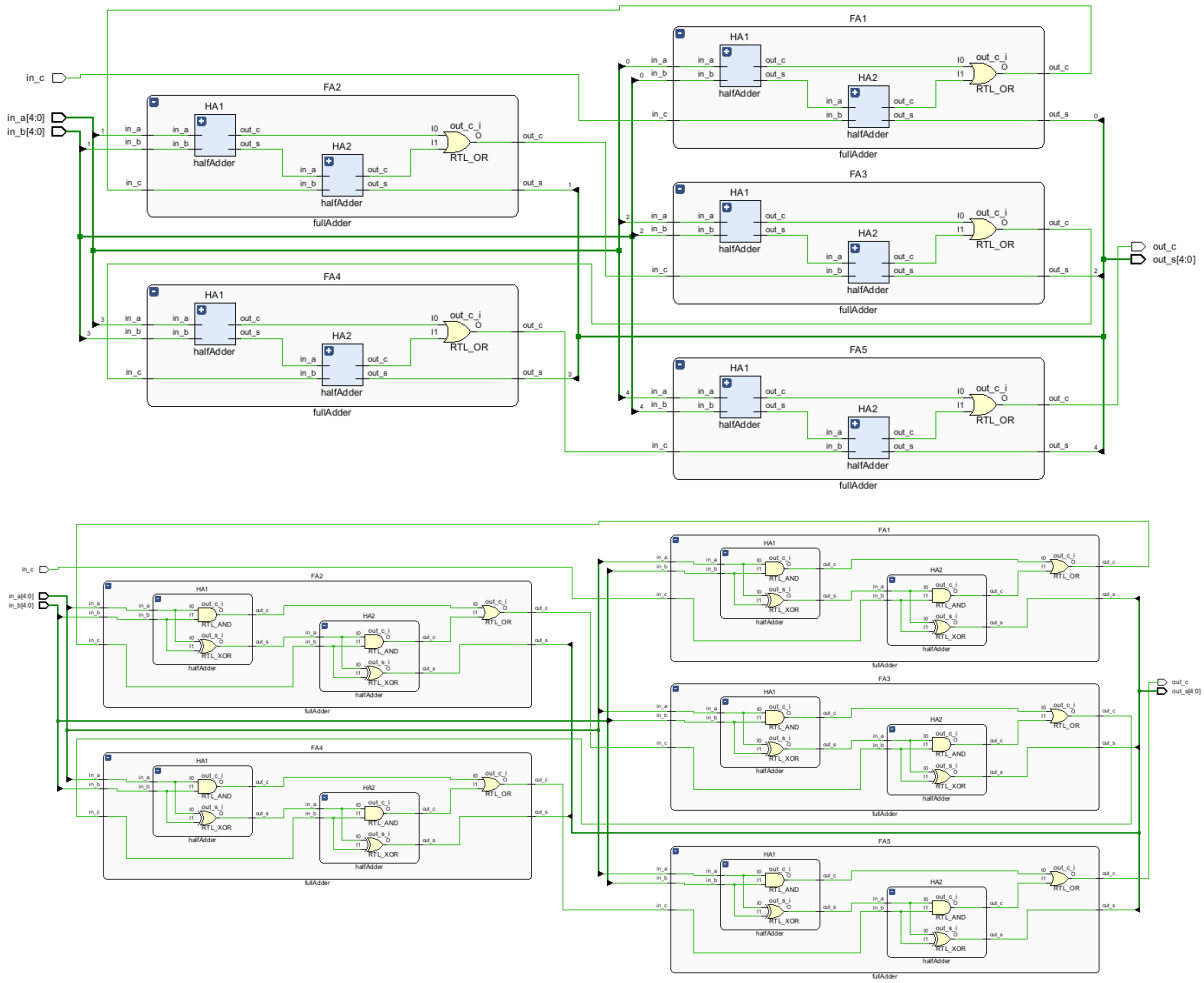
1) 반가산기, 전가산기



반가산기를 이용하여 구현한 전가산기 회로

2) 5 비트 리플 가산기

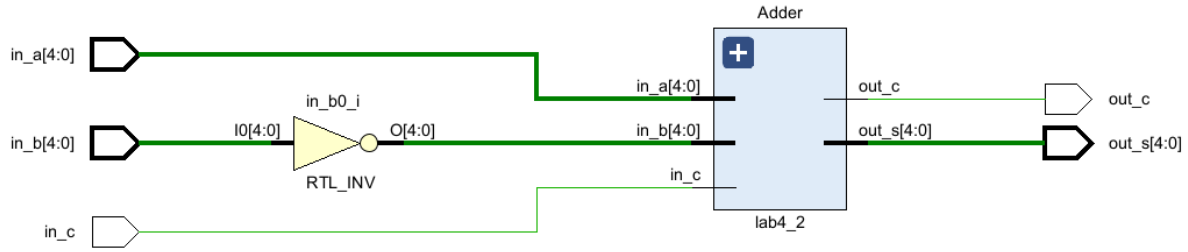


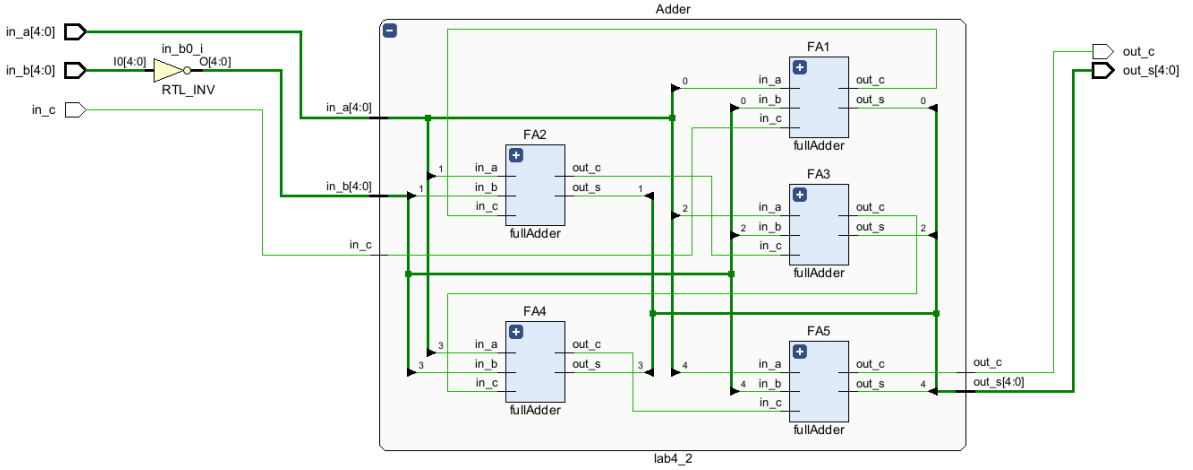


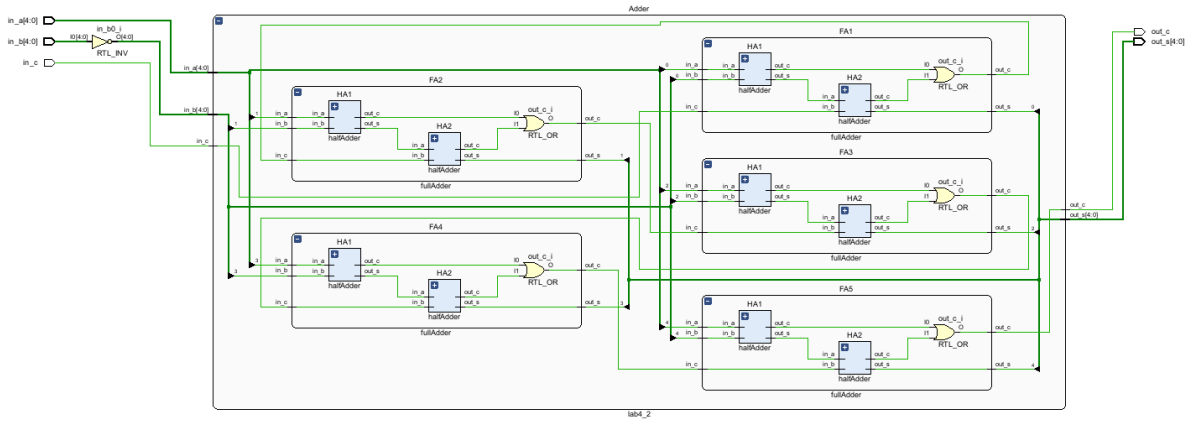
1)을 이용하여 구현한 5 비트 리플 가산기

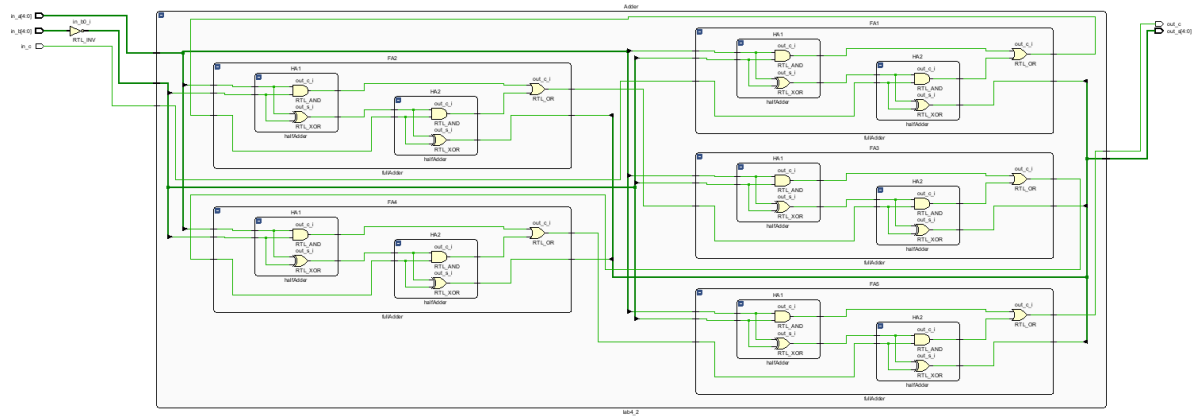
시뮬레이션 결과

3) 5 비트 리플 감산기





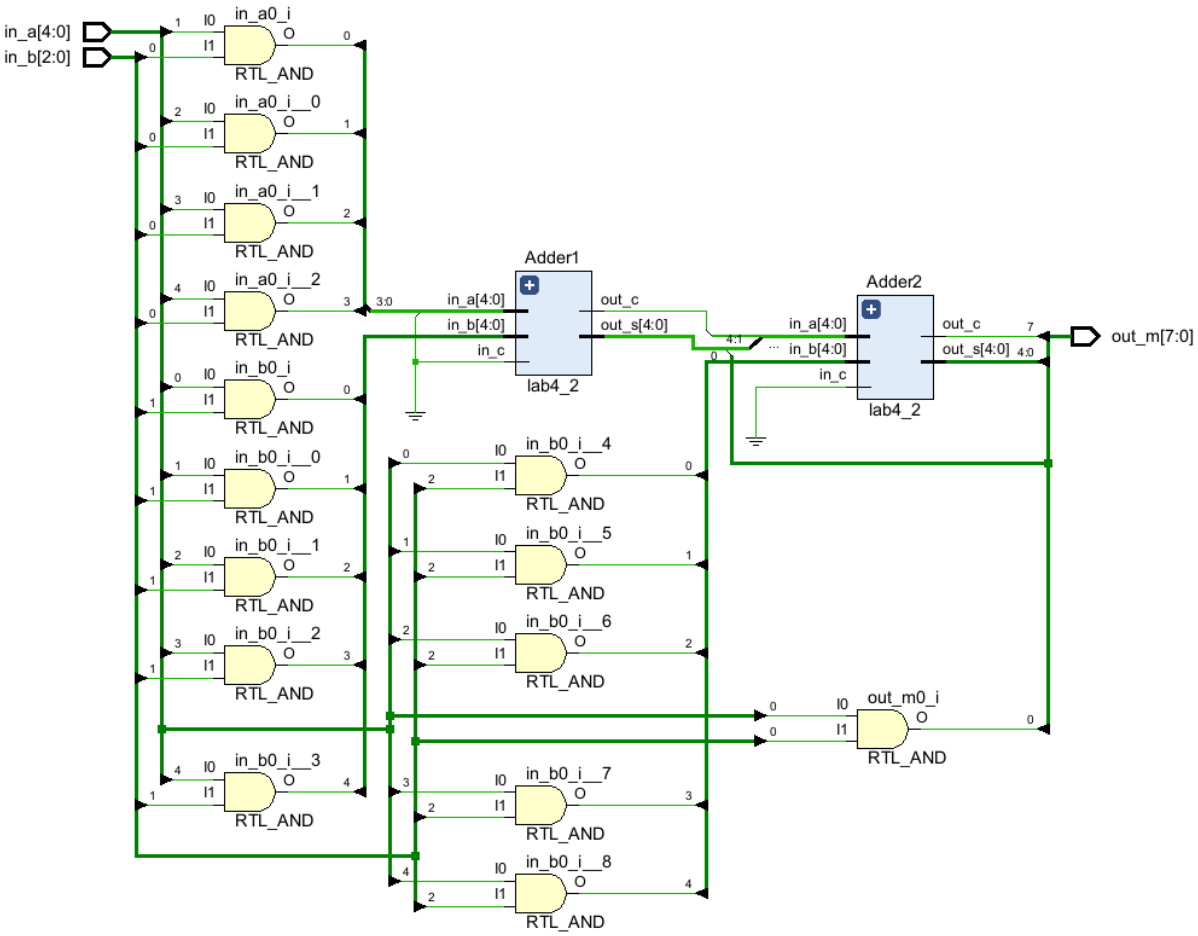


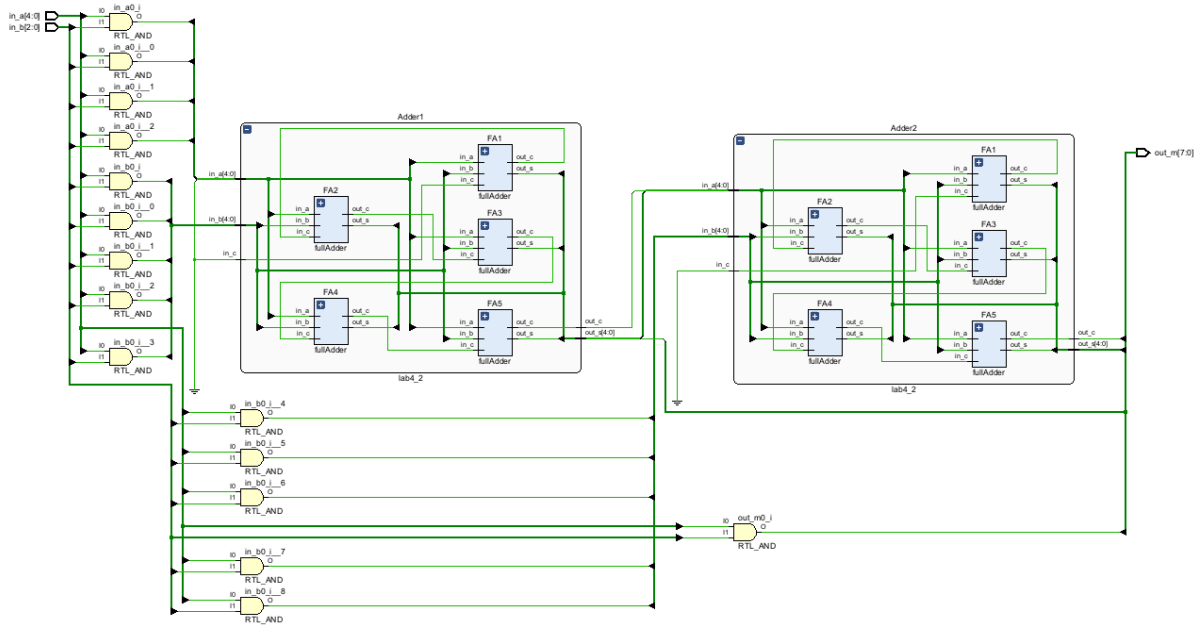


2)를 이용하여 구현한 5 비트 리플 감산기

시뮬레이션 결과

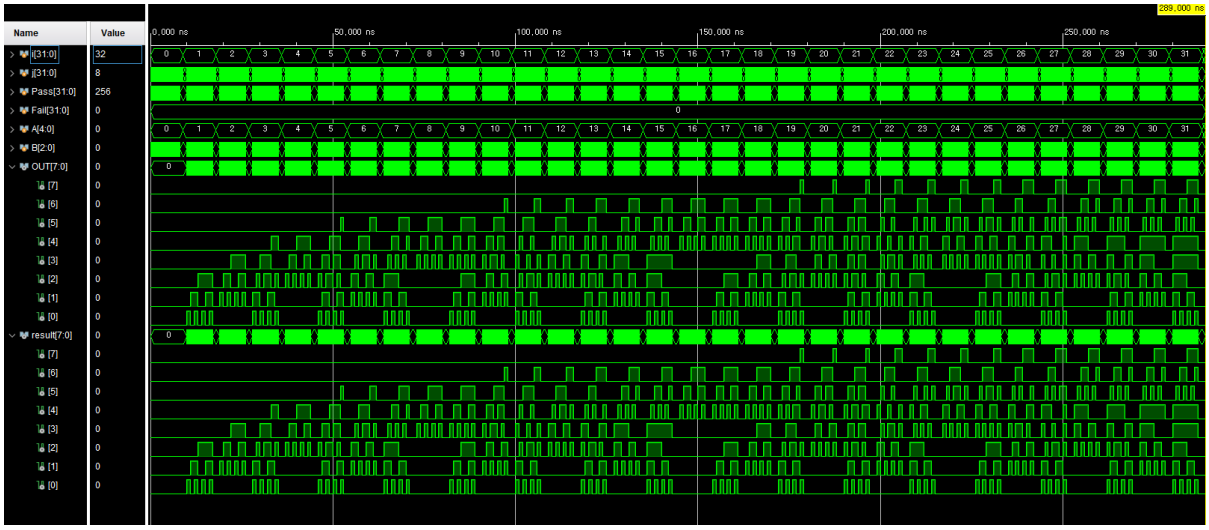
4) 5x3 이진 곱셈기





2)를 이용하여 구현한 5x3 이진 곱셈기

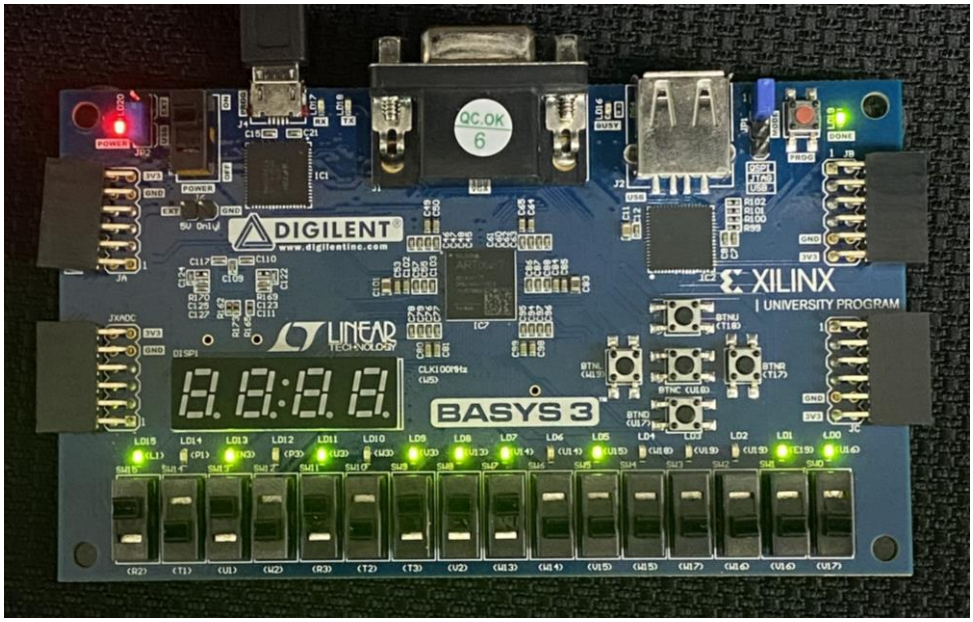
(전가산기 확대 생략)



시뮬레이션 결과

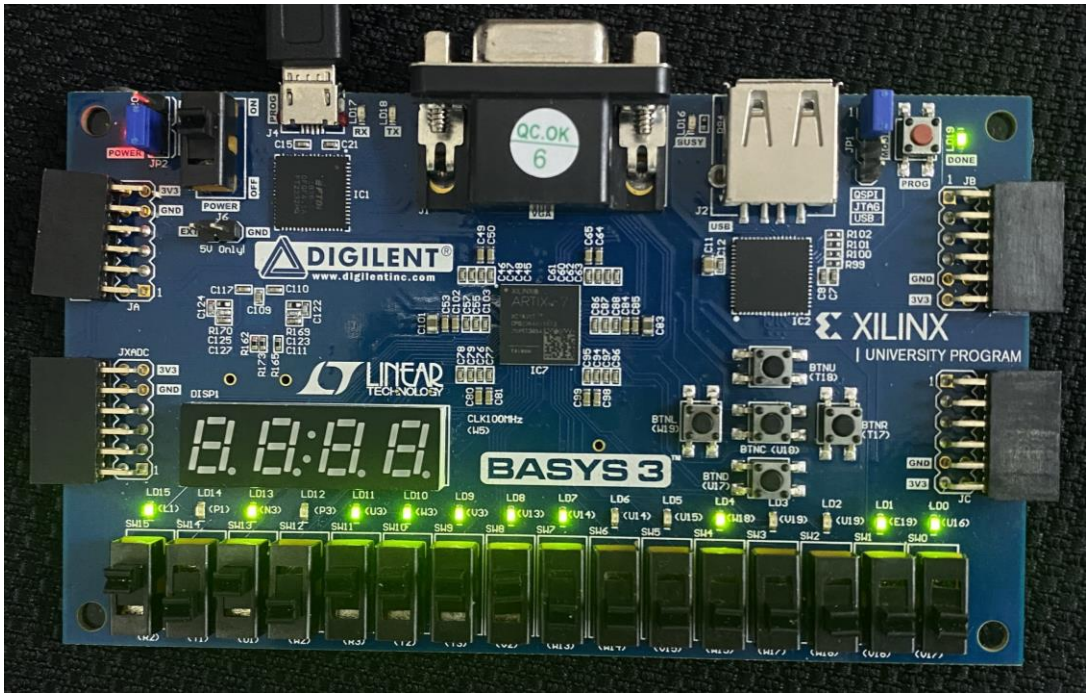
5) 주어진 코드(lab4_fpga.v)를 BASYS3 에 업로드하여 아래 연산을 수행하라.

10101+01110=100011



6) 주어진 코드(lab4_fpga.v)를 수정하여 5x3 binary multiplier 를 아래 사진과 같이 작동하도록 BASYS3 에 업로드하여 아래 연산을 수행하라.

10101X111=10010011



5. 논의

1) 느낀 점

이번 실험을 통해 이진수 연산에 사용되는 반가산기, 전가산기를 이해할 수 있었고, 이를 이용하여 덧셈, 뺄셈 연산을 수행하는 N-비트 리플 가산기/감산기, 나아가 곱셈 연산을 수행하는 MxN 이진 곱셈기까지 직접 구현해봄으로써 수업 시간에 배웠던 내용을 직접 적용해볼 수 있었다. 반가산기라는 간단한 회로로부터, 반가산기 2 개를 이용하여 전가산기를 구현하고, 전가산기를 이용하여, N-비트 리플 가산기, 감산기, MxN 이진 곱셈기 등 더욱 구체적이고 복잡한 회로를 구현할 수 있다는 점이 인상 깊었다.

기존에는 verilog 에 대해서만 다루었다면, 이번 실습에서는 직접 FPGA 를 다루면서 하드웨어 측면에서도 배운 내용들을 직접 적용시켜볼 수 있어 인상깊었다.

2) 어려웠던 점 및 해결 방법

이번 실험에서 헷갈렸던 점은 2 의 보수였다. 보수라는 개념이 그저 보충하는 수라고만 의미를 해석했을 때, 잘 와닿지 않았지만, 이번 실험을 통해 컴퓨터가 음수를 표현하기 위해서 보수를 사용함을 알고, 그중 2 의 보수에 대해 스스로 추가 학습하면서 수업 시간에 배웠던 내용에 대해 완전히 이해할 수 있었다.

또 어려웠던 점으로는 시뮬레이션을 진행한 이후, 연산이 잘 출력되고 있는지 결과를 해석하기 어려웠다는 것이다. 어떻게 하면 직관적으로 확인할 수 있을까 고민한 결과, 표현되어 있던 방식이 16 진수였음을 알게 되었고, 설정에서 2 진수, 양의 정수, 정수 등의 다양한 표현 방식으로 변환할 수 있음을 알게 되어 각 회로에 맞는 방식을 선택함에 따라 직관적으로 덧셈, 뺄셈, 곱셈 연산이 잘 수행되고 있음을 확인할 수 있었다.

저번과 같이 vivado 와 관련한 문제점이 있었다. 이 부분에 대해서는 해결책을 찾지 못해 조금은 아쉬웠던 것 같다.