컴퓨터 공학 기초 실험2 보고서

실험제목: Carry Look-ahead Adder(CLA)

실험일자: 2021년 09월 27일 (화)

제출일자: 2021년 10월 05일 (수)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2020202040

성 명: 박민형

1. 제목 및 목적

A. 제목

Carry Look-ahead Adder (CLA)

B. 목적

지난 시간에는 입력 값을 받아서 한단계 한단계 더해나가는 RCA를 공부했다. 그리고 또 직접 설계해서 정확한 값을 출력하는 지를 확인하였다. 그러나 이런 일일이 더하는 RCA는 4bits에서는 그럭저럭 돌아가지만 bits 수가 커질수록 엄청나게 느린 계산을 할 것이란 것은 알고리즘 수업의 시간 복잡도를 공부했기 때문에 알 수 있었다. 아니나 다를까, 공학자들은 CLA라는 Input만으로 최종적인 Carry-out을 예측할 수 있는 방법을 만들어냈다. 이번 실험에서는 이 CLA에 대해서 탐구할 것이다.

2. 원리(배경지식)

컴퓨터의 디지털 논리에서 쓰이는 CLA(Carry-Lookahead adder, 자리올림수 예측 가산기)는 가산기의 한 종류이다. 이전 실험에서 사용했던 RCA보다 더 빠르고 더 효율적인 특별한 논리에 따라 한번에 계산할 수 있다는 것이 특징이다. 그 이유는 다음과 같다.

RCA는 많은 수의 논리 게이트가 요구된다. 이론적으로 RCA는 구현하기 위해서 Big-O notation으로 따져보았을 때, $O(n^2)$개의 논리 게이트가 필요하다. 더욱이 현실에서는 구현 효율이 더 나빠지는데 만약 n이 커지게 되면, 더 많은 입력과 논리 게이트 사용이 필요하게 되고 더 많은 트랜지스터가 요구되기 된다. 논리 게이트의 수가 $O(n^2)$라고 할지라도, 트랜지스터의 수는 $O(n^2)$보다 커진다. 따라서 RCA의 크기는 현실적으로 다루기가 매우 어렵다.

반면에 CLA의 논리에서, 각 비트는 딱 일정한 수의 논리 게이트만 요구된다. 앞서 말한 CLA만의 특별한 논리를 수식으로 정리하면 다음과 같다.

$$G_i = A_i B_i$$
$$P_i = A_i + B_i$$
$$C_{i+1} = A_i B_i + A_i + B_i C_i = G_i + P_i C_i$$

이를 4bits carry-lookahead adder의 자리올림수인 $C_0 \dots C_3, C_{out}$까지 구해본다면 다음과 같다.

$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$
$$C_{out} = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

즉, 우리는 최종적인 Carryout을 모든 자리 올림수를 알 필요 없이, 오로지 처음 Input인 A와 B의 연산만으로 구할 수 있는 것이다!

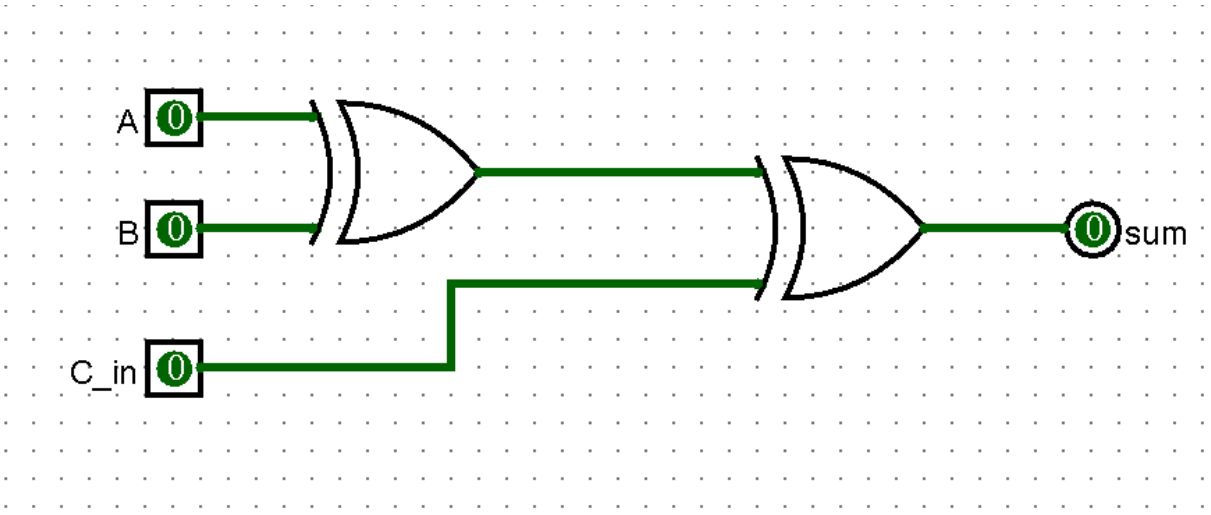
이렇게 CLA에서는 몇 개의 값이 정해져 있다면 RCA처럼 모든 과정을 거치지 않고서도 가산된 값을 예측할 수 있다. 따라서 만약 n이 가산기의 비트 수라면, 논리 게이트의 수는 $O(n)$이 되고 RCA보다 훨씬 효율적이게 된다.

3. 설계 세부사항

1. fa_v2 (full adder without carry-out)

fa_v2는 full adder에서 carry out을 제거한 회로이다. CLA에서는 full adder의 carry out 포트가 필요하지 않기 때문이다.

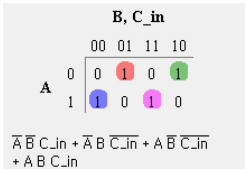
Schematic은 다음과 같다.



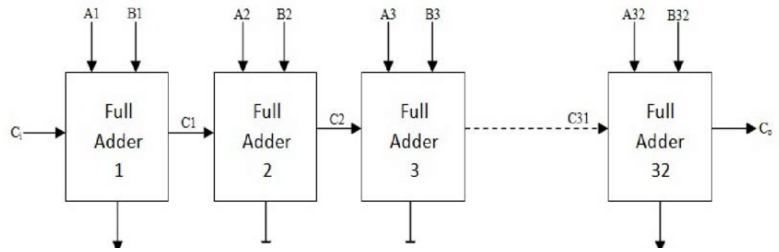
진리표는 다음과 같다.

A	B	C_in	sum
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Karnaugh-Map은 다음과 같다.



2. RCA 32bits

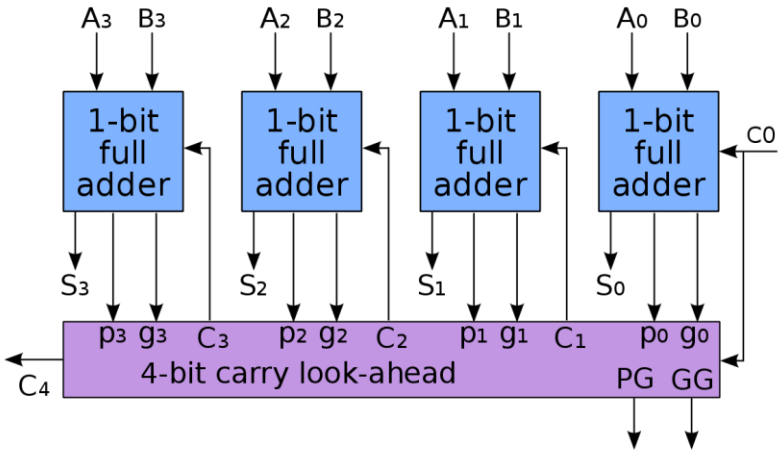


RCA의 Schematic은 위의 그림과 같다.

따라서, 그림의 Full Adder 4개를 묶은 것을 RCA 4bits로 본다고 하면, RCA 4bits 9개를 나란히 연결하면 RCA 32bits가 된다.

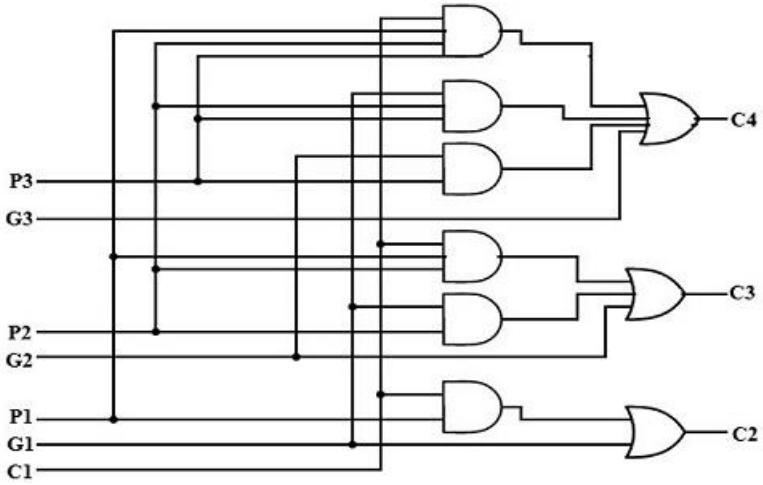
3. CLA

CLA 4bits의 Schematic은 다음과 같다.



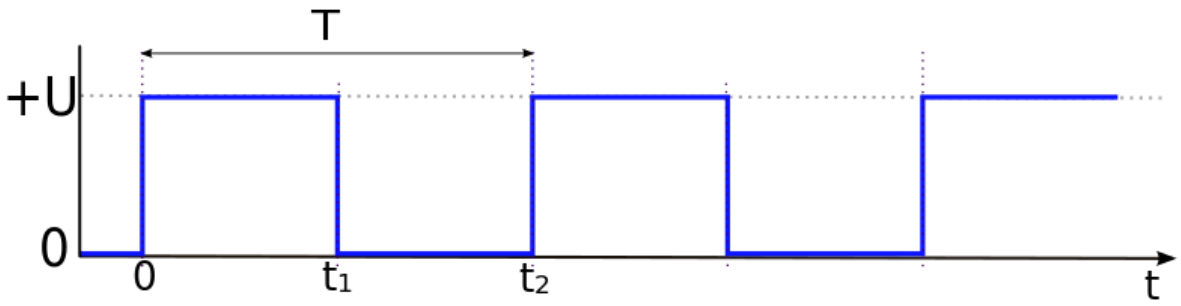
여기서 우리는 CLA 32bits를 만들 때에는 RCA 32bits와 마찬가지로 CLA 4bits의 C_4를 다음 CLA 4bits의 C_0에 넣는 방식을 이용해서 CLA 4bits 8개를 이어 붙이는 것으로 만들 수 있다.

참고로 CLA 4bits의 게이트 수준에서의 회로도는 다음과 같이 작성할 수 있다.



4. CLK

Clock Signal(CLK)는 주기적으로 High와 Low를 출력하는 방형파 신호이다. 주기적인 신호를 보낼 수 있다는 특성 때문에 컴퓨터공학을 비롯한 전자공학에서는 디지털 회로에서 클럭 신호에 맞추어 신호 처리를 하는 동기 처리를 많이 쓰고 이때 CLK 신호를 사용한다. 지금은 조합 논리 회로를 집중적으로 설계하지만 후에 나오는 순차회로(Sequential Circuit)에서는 요긴하게 쓰인다.



Verilog 관련)

Initial Block

지금까지 주로 쓴 Block인데 Simulation이 시작할 때 한번만 실행되는 Block이다.

Always Block

Always Block은 계속 실행된다. Always Block에는 괄호가 따라붙는데, @ 뒤에 있는 괄호 안에 들어간 port의 변화가 있을 때마다 동작하게 된다. 이러한 변화에는 두가지 정도의 변화를 꼽을 수 있다.

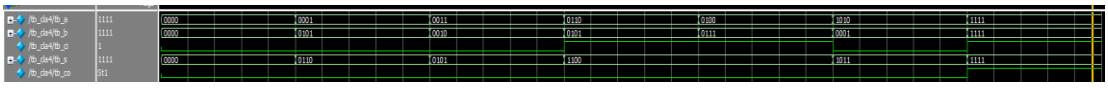
1. level sensitive: @(a) 이런 경우에 a가 바뀌면 always 문이 수행된다.
2. edge sensitive: @(posedge something) 이런 경우는 edge에 따라 always 문이 수행된다.

CLK를 다룰 때에는 Always Block을 이용해서 구현할 수 있다.

4. 설계 검증 및 실험 결과

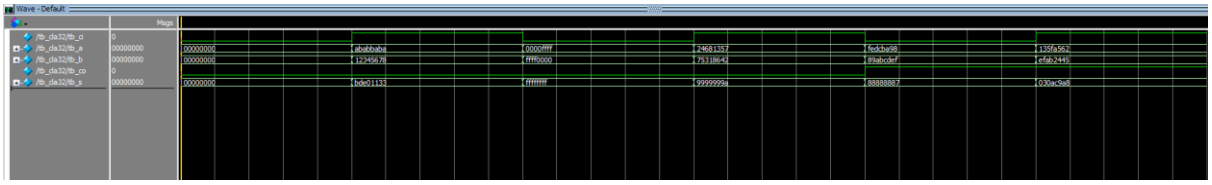
A. 시뮬레이션 결과

1. tb_cla4



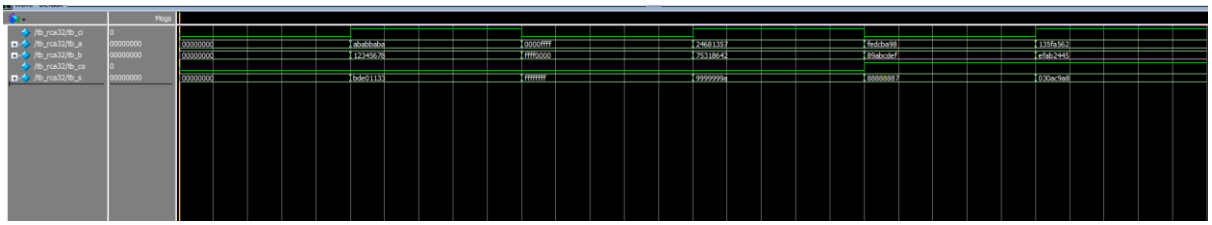
CLA 4bits의 testbench 결과이다.

2. tb_cla32

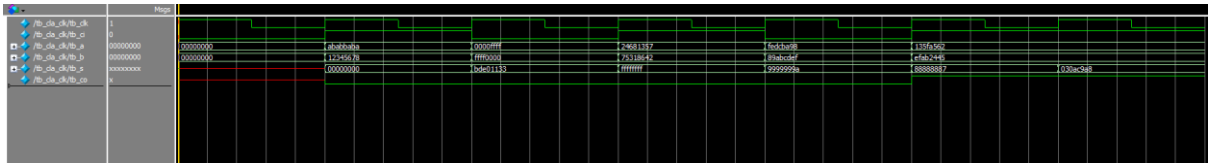


CLA 32bits의 testbench 결과이다.

밑의 RCA 32bits의 testbench 결과와 정확하게 같게 나오는 것을 확인할 수 있다.

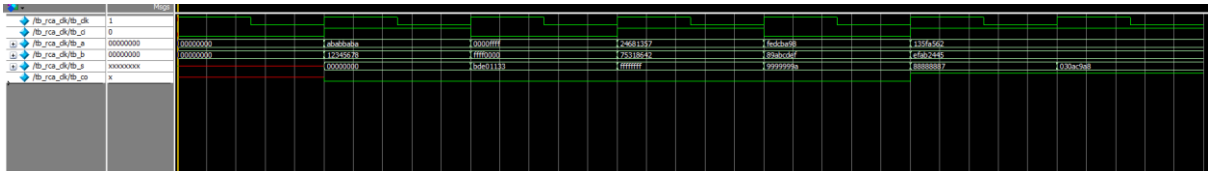


3. tb_cla_clk



Clock과 연결한 CLA 32bits의 testbench 결과이다.

4. tb_rca_clk

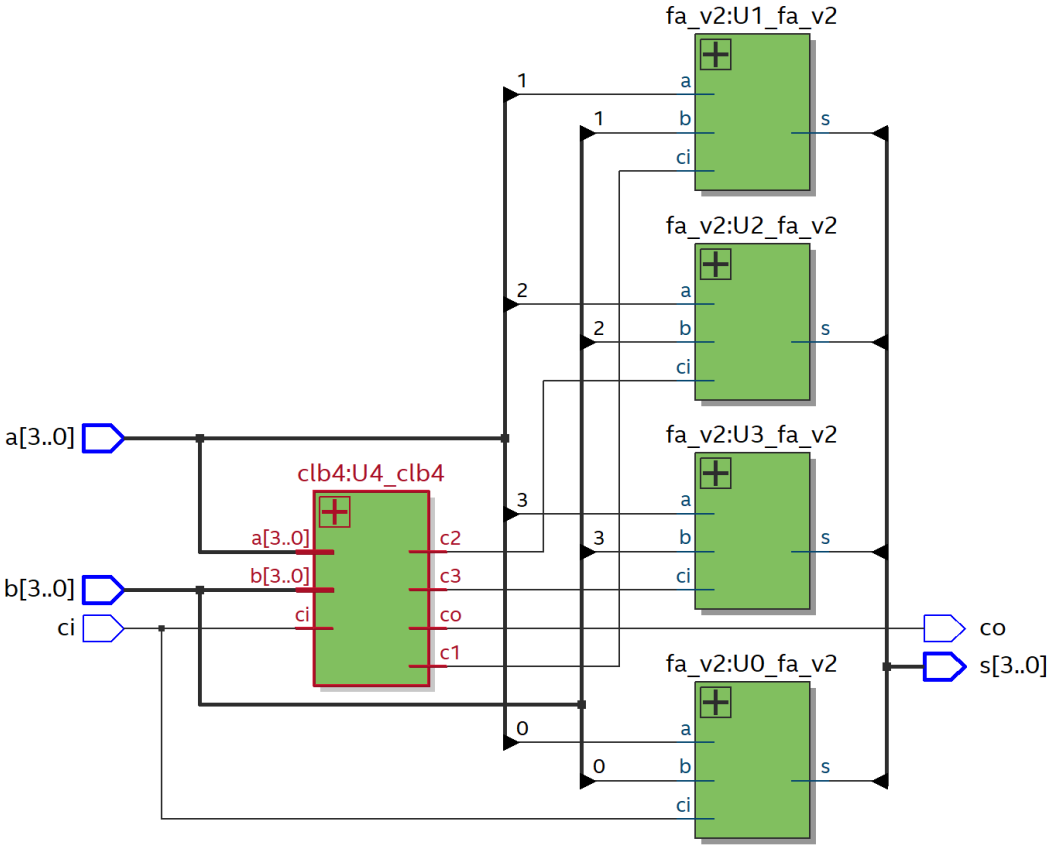


Clock 과 연결한 RCA 32bits 의 testbench 결과이다.

위의 tb_cla_clk 의 결과와 정확히 일치하게 나온다

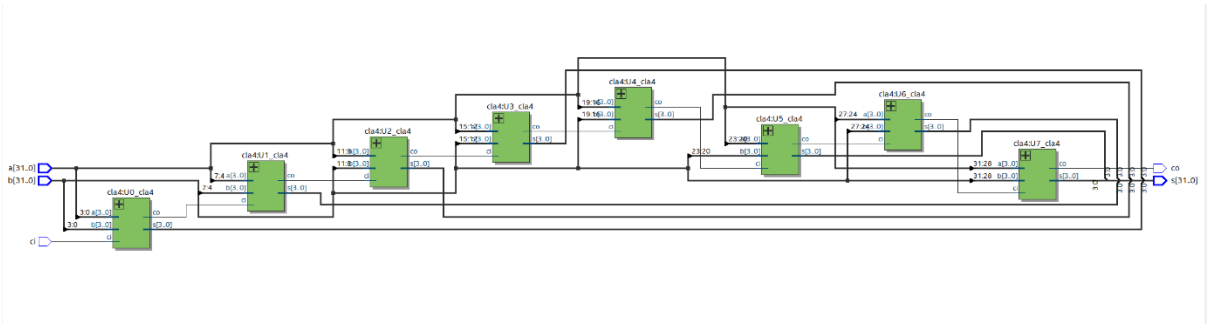
B. 합성(synthesis) 결과

5. tb_cla4



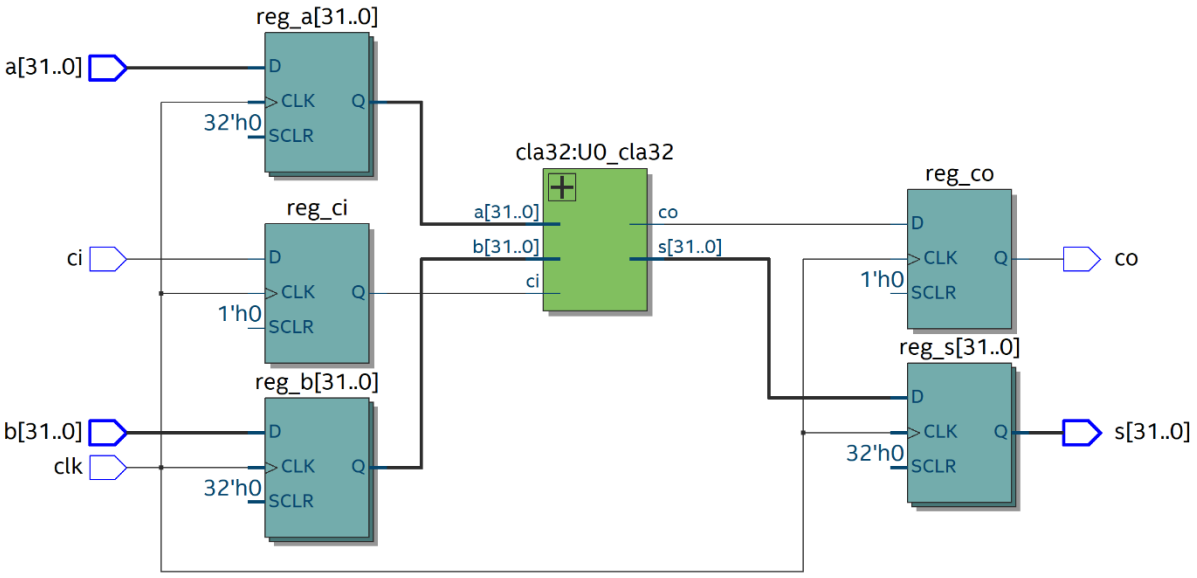
tb_cla4.v의 RTL Viewer이다.

6. tb_cla32



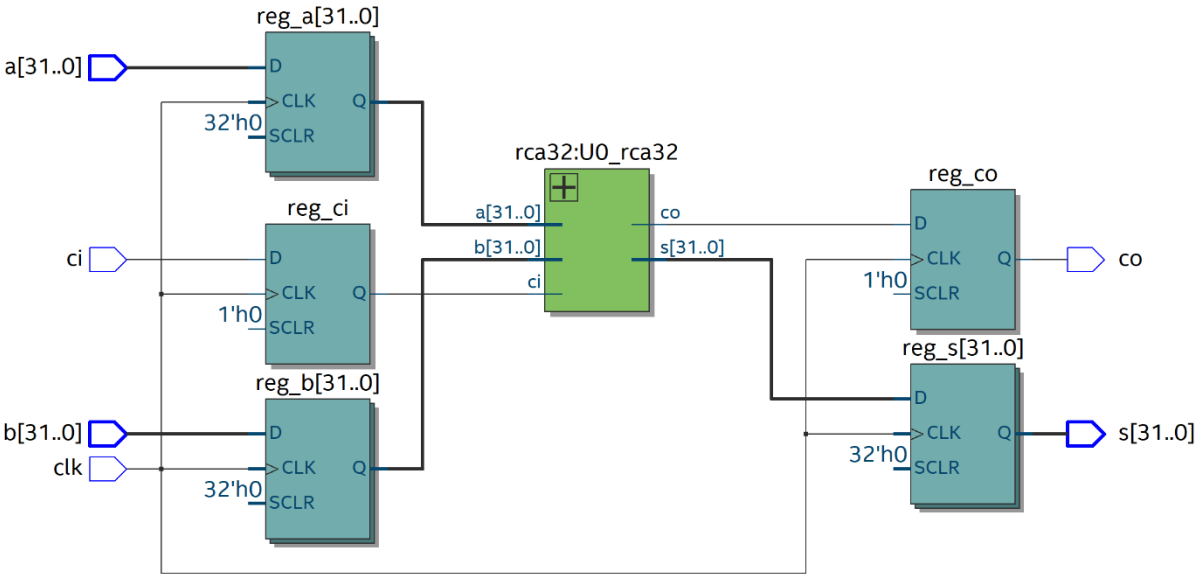
tb_cla32.v의 RTL Viewer이다.

7. tb_cla_clk



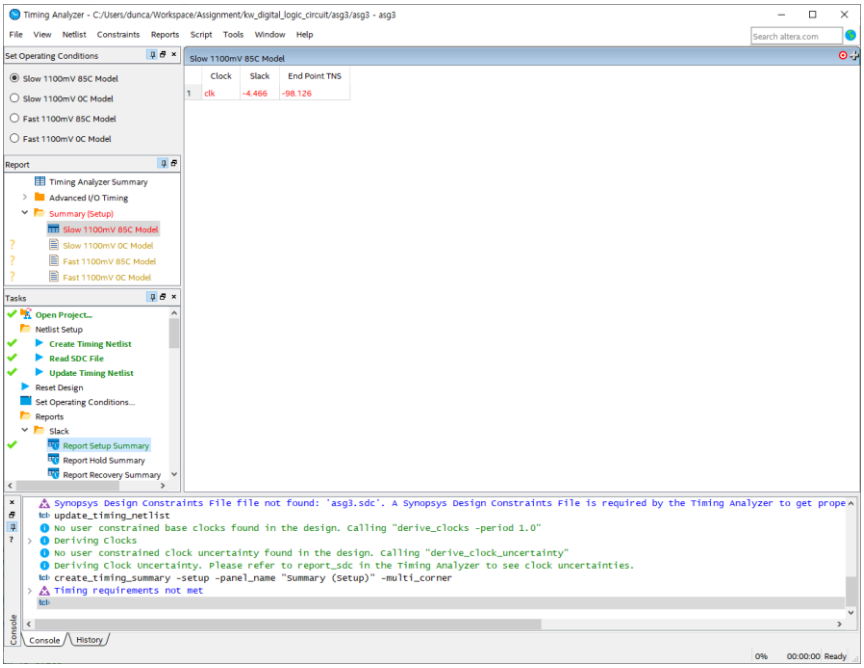
tb_cla_clk.v의 RTL Viewer이다.

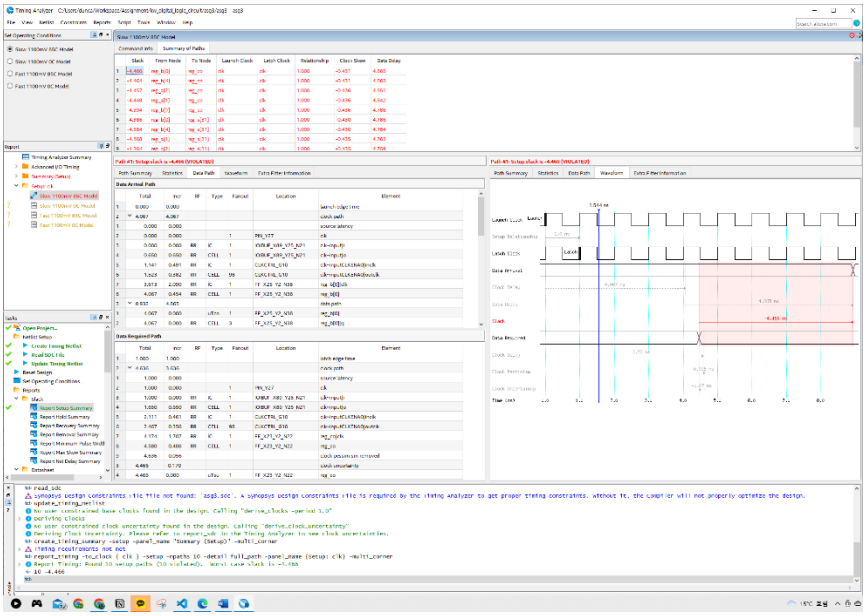
8. tb_rca_clk

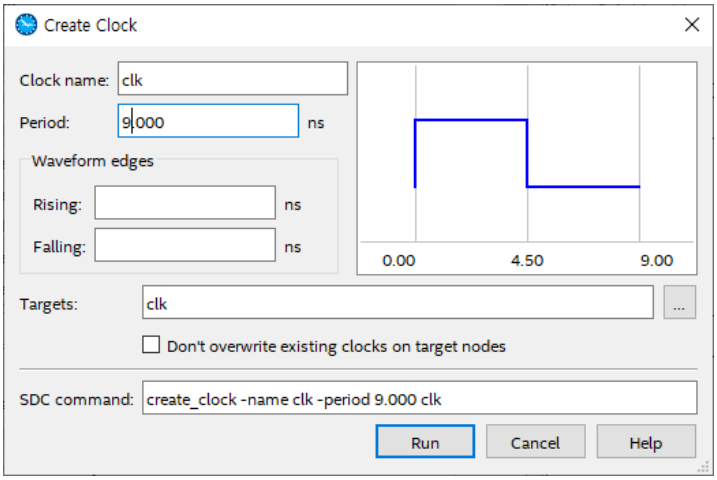


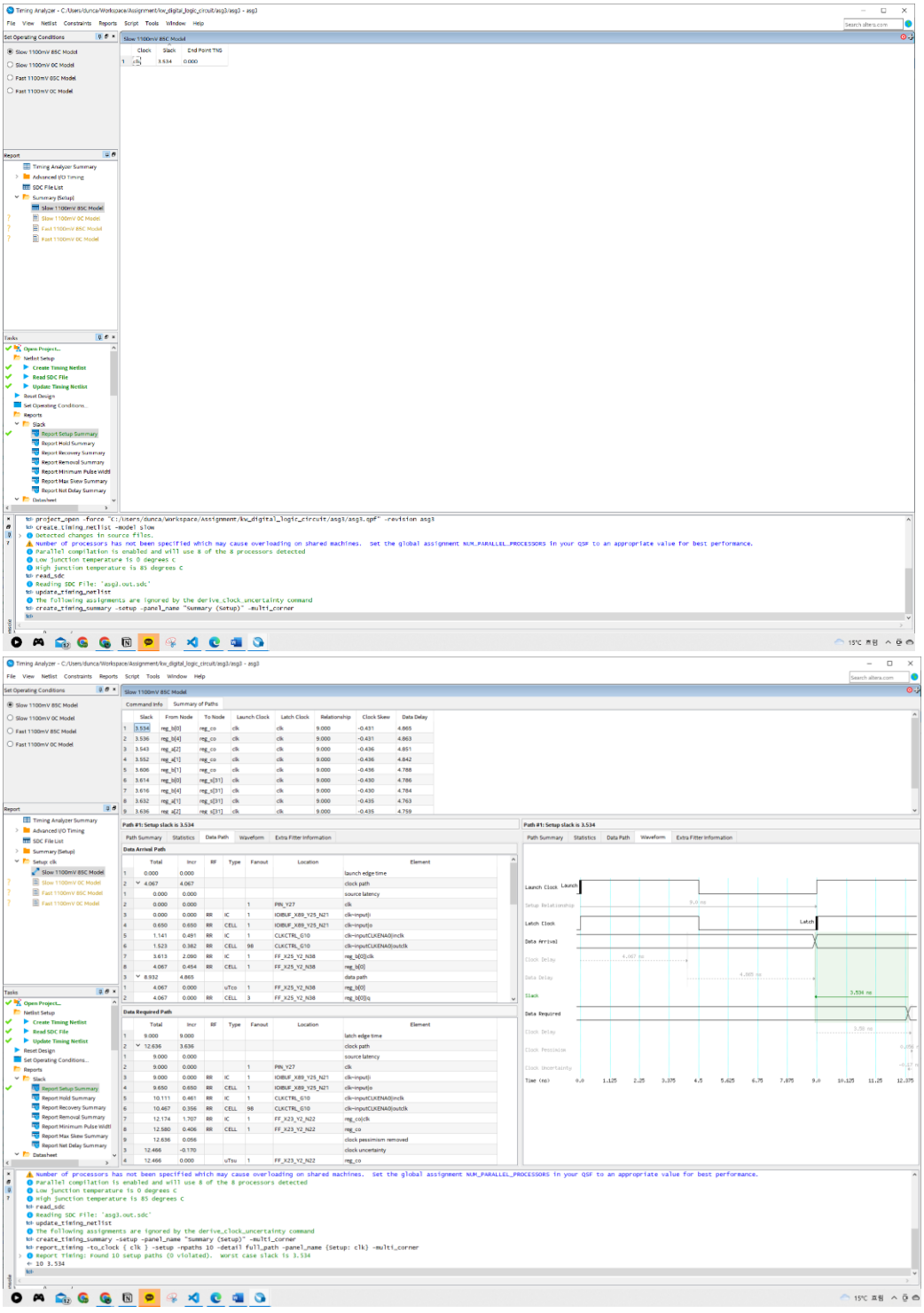
tb_rca_clk.v의 RTL Viewer이다.

C. Time Analysis 실습









순서대로 수행했더니 빨간색 Slack 값이 없어졌다.

5. 고찰 및 결론

A. 고찰

Full Adder와 CLB를 잘못 짠 상태로 진행하니까 어디서부터 고쳐야 할지를 갈팡질팡하여 시간을 날리고 제출도 늦게 하게 되었다. 낮은 단위에서부터 주의해서 설계를 하는 것이 좋겠다. 그리고 Time Analysis는 일단 따라하는 수준에서만 그쳤지만, 구체적으로 이걸로 어떤 파라미터를 조작할 수 있는지도 궁금하다.

CLA에 대해서 그래도 꽤 많은 사실을 알고 완전히 이해했다고 생각하지만, 늦게 제출하게 된 것이 유감스럽다. 다음 실험에서는 ALU를 다루는데, 어떤 오류가 기다리고 있을지 모르니 미리미리 회로도 설계 해놓고 낮은 단위에서 오류를 잡아서 지금처럼 기껏 해놓고 감점 당하는 일이 없도록 해야겠다.

B. 결론

CLA는 Input 값만 가지고서 모든 carry 값을 알 필요 없이 정확히 가산이 가능하다는 점이 놀라웠다. 설계에 있어서 어떤 알고리즘을 쓰느냐가 얼마나 효율적인 작업을 만들 수 있는지를 알게 해주는 시간이었다.

6. 참고문헌

신경욱/Verilog HDL을 이용한 디지털 시스템 설계 및 실습/카오스북/2013

자리올림수예측가산기/

https://ko.wikipedia.org/wiki/%EC%9E%90%EB%A6%AC%EC%98%AC%EB%A6%BC%EC%88%98_%EC%98%88%EC%B8%A1_%EA%B0%80%EC%82%B0%EA%B8%B0