컴퓨터 공학 기초 실험2 보고서

실험제목: Ripple-Carry Adder (RCA)

실험일자: 2022년 09월 27일 (화)

제출일자: 2022년 09월 29일 (목)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2021202058

성 명: 송채영

1. 제목 및 목적

A. 제목

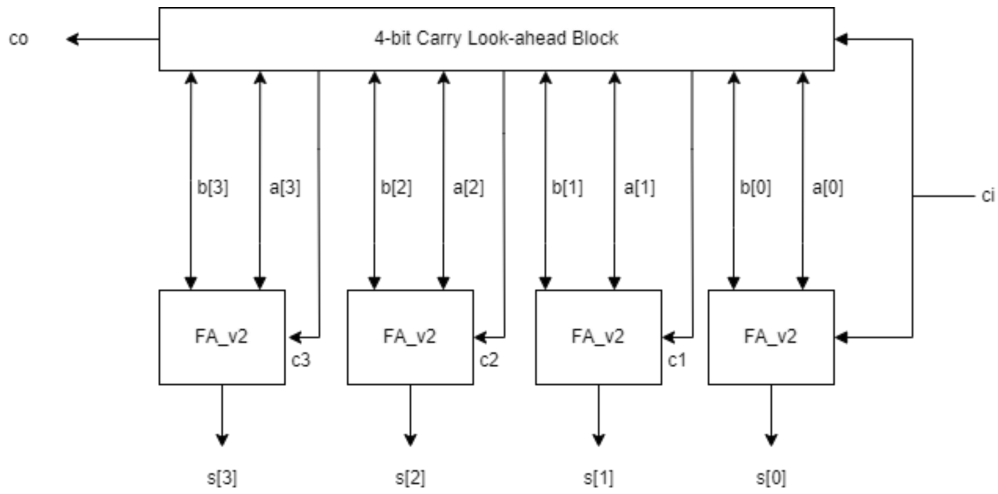
Carry Look-ahead Adder (CLA)

B. 목적

Ripple Carry Adder가 계산이 완료될 때 까지 시간이 많이 걸린다는 단점을 보완하고자 만든 Carry Look-ahead Adder의 동작원리를 이해하고, 이를 통해 CLA를 직접 설계해본다. CLA를 구현하고, TimingQuest Timing Analyzer을 사용해 delay를 확인하고, clock을 조절해 delay를 조절하며 RCA와의 차이를 비교한다. 또한 32 bit CLA와 32 bit RCA의 속도와 크기를 비교해본다.

2. 원리(배경지식)

Ripple Carry Adder가 계산이 완료될 때 까지 시간이 많이 걸린다는 단점이 있어 단점을 보완하고자 만들어진 것이 Carry Look-ahead Adder이다. CLA는 모든 올림 수가 동시에 구해지는 가산기로, 한 번 계산이 진행될 때마다 올림을 전달하는 RCA에 비해 계산 시간을 단축시키는 가산기이다. 간단한 계산을 할 때에는 RCA를 사용하는 것이 효율적이지만 비트가 길어질수록 올림을 판단하는 데 시간이 점점 더 길어지므로 CLA를 사용하는 것이 효율적이다. 아래의 그림은 4-bits CLA의 회로이다. 4개의 Full adder와 4-bit Carry Look-ahead Blcok을 연결해 만든 것을 볼 수 있다.



여기서 사용되는 4-bit Carry Look-ahead Bloack는 Carry out의 값을 미리 계산하기 위해 사용하는 Generation signal G_i와 Carry in이 발생하는 Propagation signal p_i는 다음을 의미한다. $G_i = A_iB_i$, $P_i = A_i + B_i$ 이 식을 Full adder의 carry out에 적용하면 $C_{i+1} = A_iB_{i,} + (A_i + B_i)C_i = P_iC_i$이며 이 식을 통해 carry out을 구할 수 있다.
이를 4-bit adder의 carry를 계산하는데 적용해보면

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0)$

$= G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = C_3 = G_2 + P_2(G_1 + P_1G_0 + P_1P_0C_0)$

$= G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

$C_4 = G_3 + P_3C_3 = G_3 + P_3(G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0)$

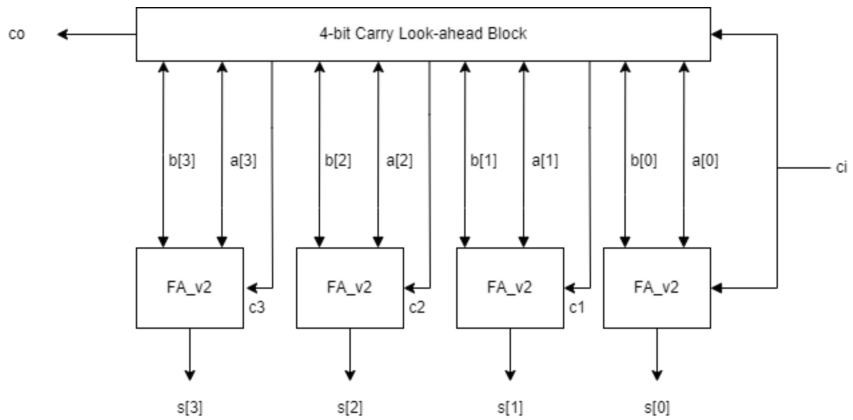
$= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1P_0G_0 + P_3P_2P_1P_0C_0$

임을 알 수 있다.

TimeQuest Timing Analyzer는 회로의 지연을 분석하는 과정으로, 최대 동작 주파수를 찾아주어 적절하게 동작할 수 있도록 한다. TimeQuest Timing Analyzerclock를 통해 clock을 바꾸고 slack이 음수가 되는 것을 방지 할 수 있다.

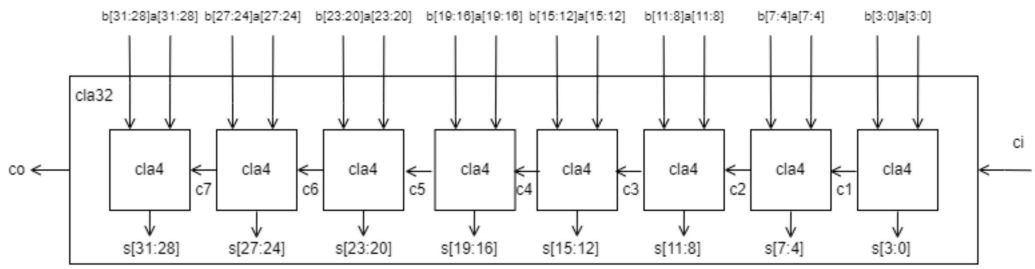
3. 설계 세부사항

- 4-bit Carry Look-ahead Adder



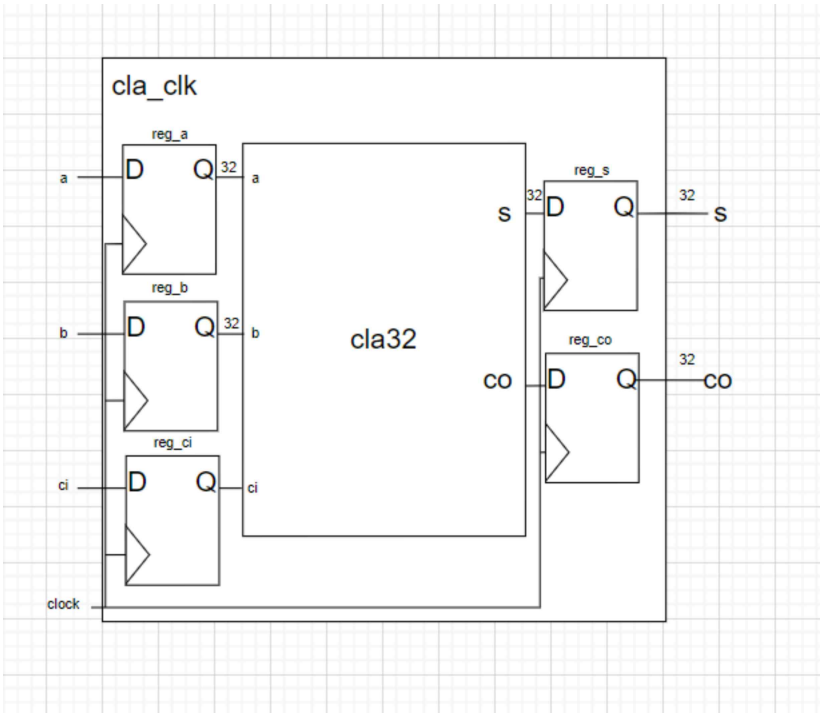
4-bits CLA는4개의 Full adder와 4-bit Carry Look-ahead Blcok을 연결해 설계한다. input은 a, b, ci로 나뉘는데 a, b는 각각 4-bit, ci는 1-bit의 크기를 가진다. output은 s, co로 나뉘는데 s는 4-bit, co는 1-bit의 크기를 가진다. CLB는 올림 수를 한번에 계산하기 위해 carry만을 계산하는 것이다. CLB는 위에서 구한 C_1, C_2, C_3, carryout의 식을 통해 코드를 구성하였다. input은 CLA와 마찬가지로 a, b, ci를 output은 c1, c2, c3, co를 가진다. 4-bit wire인 g, p에서 g는 generate, p는 propagate를 의미한다.

- 32-bit Carry Look-ahead Adder



32-bit CLA는 4-bit CLA 8개를 직렬로 연결하여 설계한다. input은 a, b, ci로 나뉘는데 a, b는 각각 4-bit의 크기를 가지며 ci는 1-bit의 크기를 가진다. output은 co, s로 나뉘는데 co는 1-bit의 크기를 가지고 s는 cla4에서 나온 것으로 32bit의 크기를 가진다. wire는 총 8개로 1-bit wire를 사용하였다.

- 32-bit CLA with clock

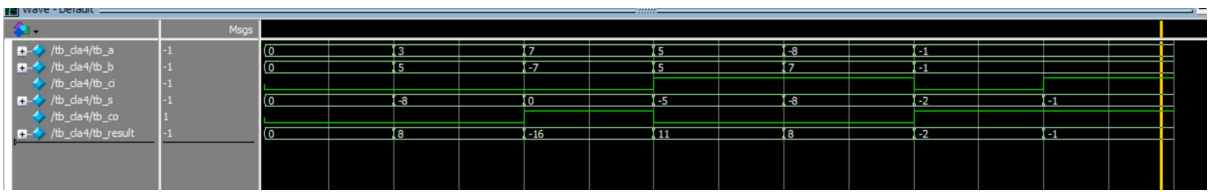


32-bit CLA에 앞 뒤로 D flip-flop을 추가해 clock과 연결하여 설계한다. input은 32bit의 크기를 가지는 a, b와 1-bit의 크기를 가지는 clock과 ci가 있다. output은 32-bit의 크기를 가지는 s와 1-bit의 크기를 가지는 co가 있다. register에는 reg_a와 reg_b, reg_s는 각각 32-bit의 크기를 가지며 1-bit의 크기를 가지는 reg_ci, reg_co가 있다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

- 4-bit CLA

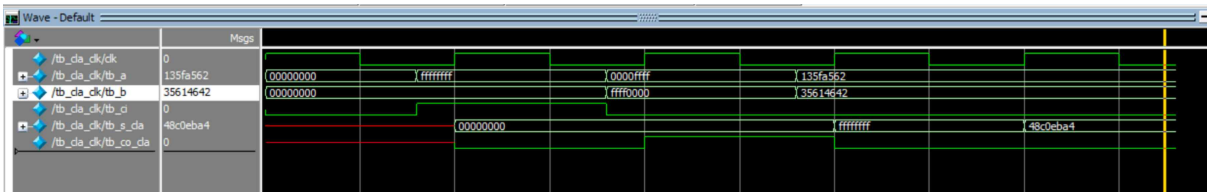


Input(10진수로 표현)			tb_result
tb_a	tb_b	tb_ci	
0	0	0	0

3	5	0	8
7	9	0	16
5	5	1	11
8	7	1	16
15	15	0	30
15	15	1	31

위의 표와 같이 testbench를 설정해주었다. tb_result의 값이 waveform에서도 같게 나온 것을 통해 설계가 잘 되었음을 확인할 수 있다.

- CLA_CLK



input(16진수)		tb_s_cla
tb_a	tb_b	
00000000	00000000	00000000
ffffffff	00000000	ffffffff
0000ffff	ffff0000	ffffffff
135fa562	35614642	48c0eba4

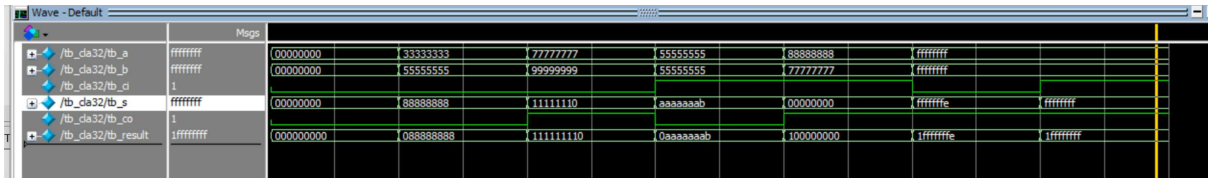
위의 표와 같이 testbench를 설정해주었다. tb_s_cla의 값이 waveform에서도 같게 나온것을 통해 설계가 잘 되었음을 확인할 수 있다.

- RCA_CLK

input(16진수)		tb_s_rca
tb_a	tb_b	
00000000	00000000	00000000
ffffffff	00000000	ffffffff
0000ffff	ffff0000	ffffffff
135fa562	35614642	48c0eba4

위의 표와 같이 testbench를 설정해주었다. tb_s_rca의 값이 waveform에서도 같게 나온 것을 통해 설계가 잘 되었음을 확인할 수 있다.

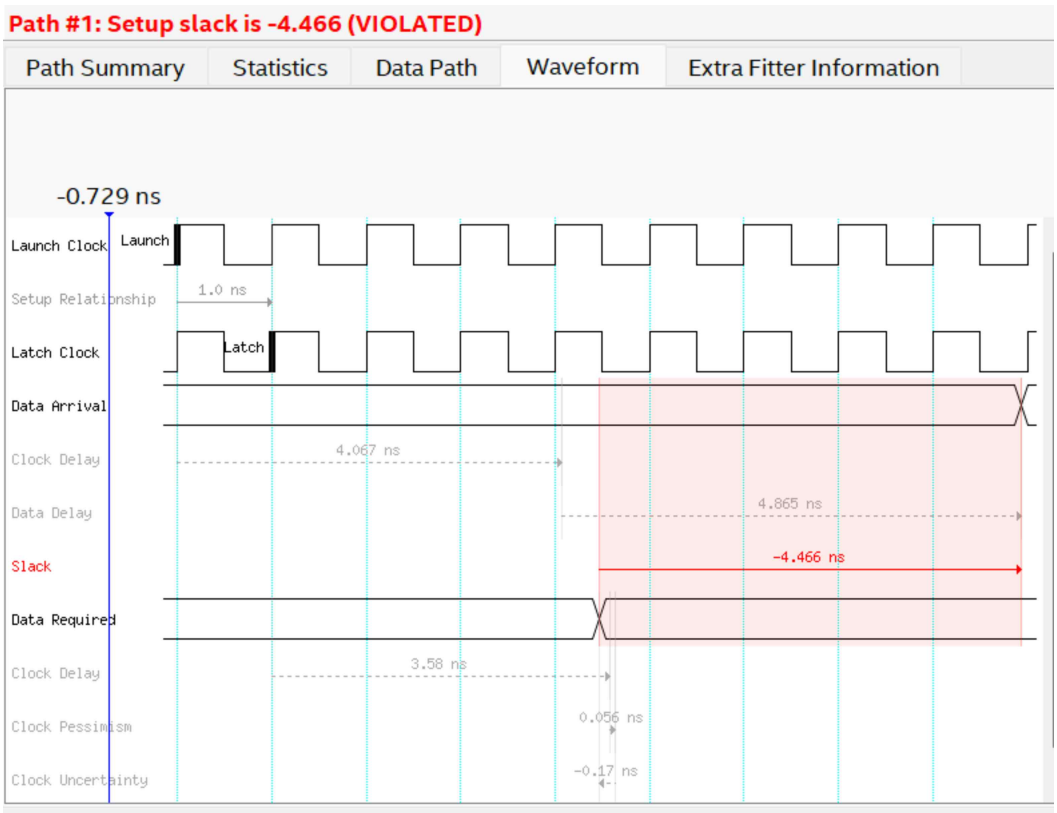
- cla32



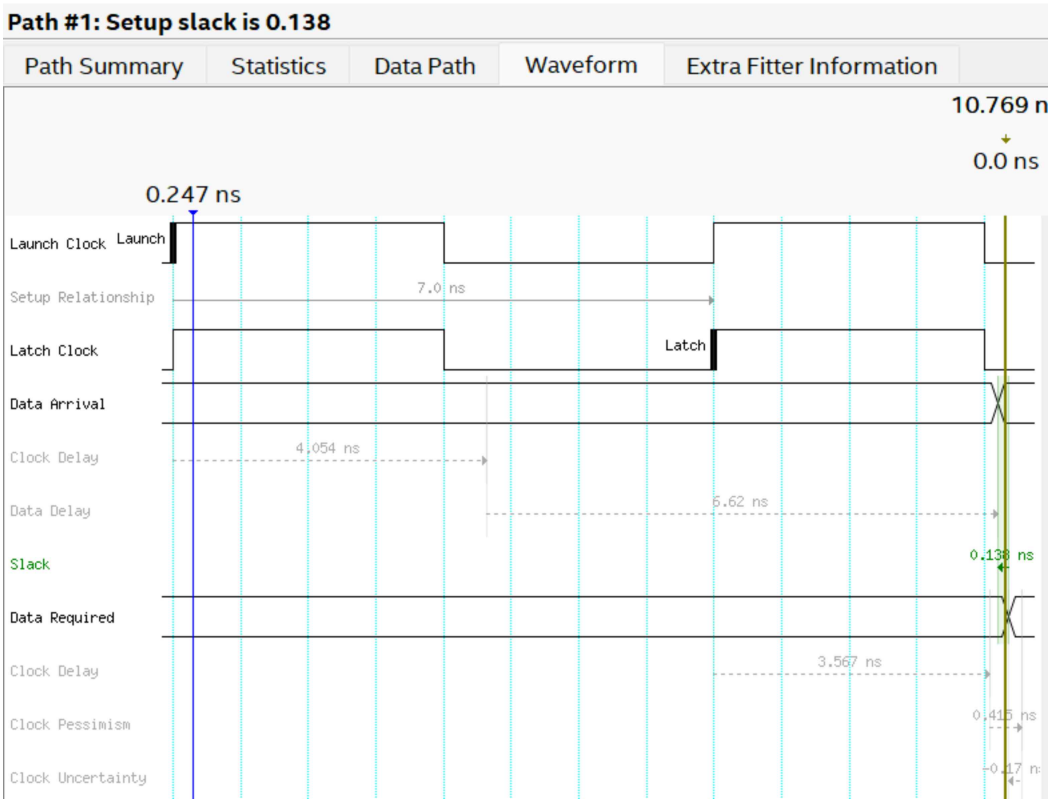
input(16진수)		tb_result
tb_a	tb_b	
00000000	00000000	000000000
33333333	55555555	08888888
77777777	99999999	111111110
55555555	55555555	0aaaaaaab
88888888	77777777	100000000
ffffffff	ffffffff	1fffffffe

위의 표와 같이 testbench를 설정해주었다. tb_result의 값이 waveform에서도 같게 나온 것을 통해 설계가 잘 되었음을 확인할 수 있다.

- Timing Analyze CLA_CLK



slack의 값이 -4.466ns로, 데이터가 필요한 시점보다 -4.466ns 만큼 늦게 도착한다는 것을 확인할 수 있다. clock의 주기를 7로 설정해 slack을 양수가 되게끔 바꾸었다.



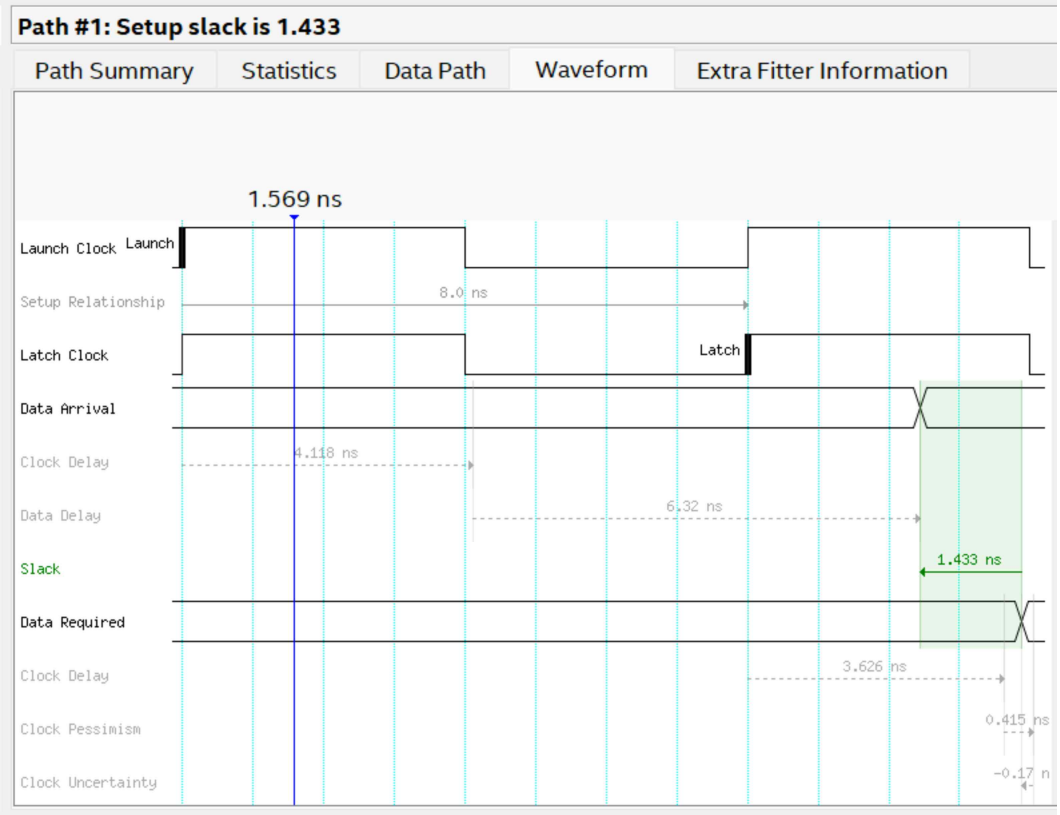
clock의 주기를 7로 설정하자 waveform에서 violation이 발생하지 않는 것을 확인할 수 있다.

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	145.73 MHz	145.73 MHz	clk	

Fmax는 145.73MHz인 것을 확인할 수 있다.

- Timing Analyze RCA_CLK

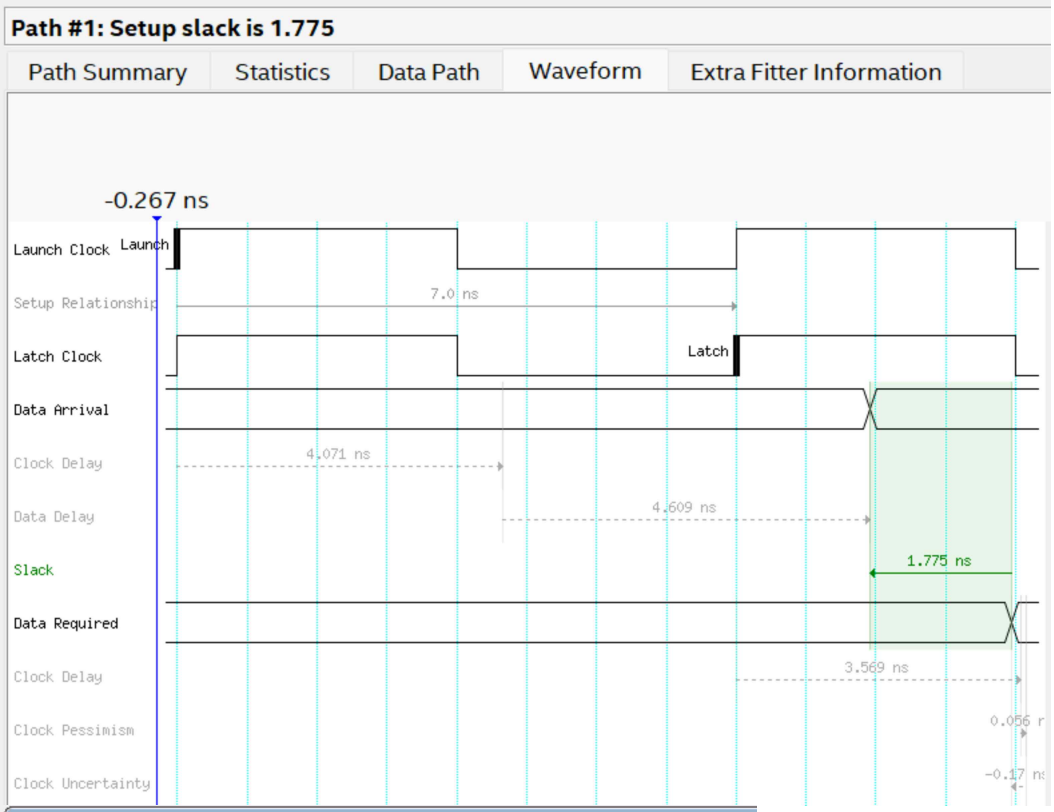


마찬가지로 clock의 주기를 8로 설정해 slack을 양수가 되게끔 바꾸었다. clock의 주기를 8로 설정하자 waveform에서 violation이 발생하지 않는 것을 확인할 수 있다.

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	152.28 MHz	152.28 MHz	clk	

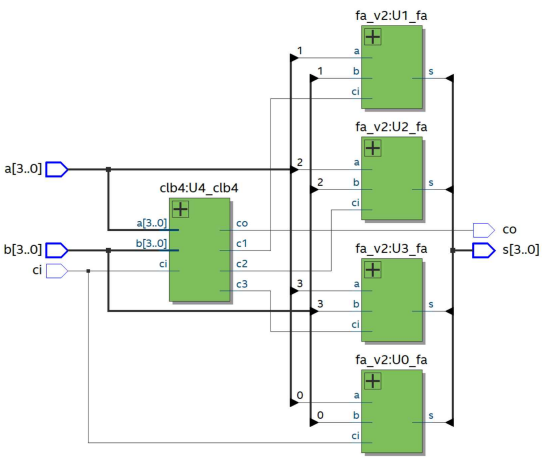
Fmax는 152.28MHz인 것을 확인할 수 있다.

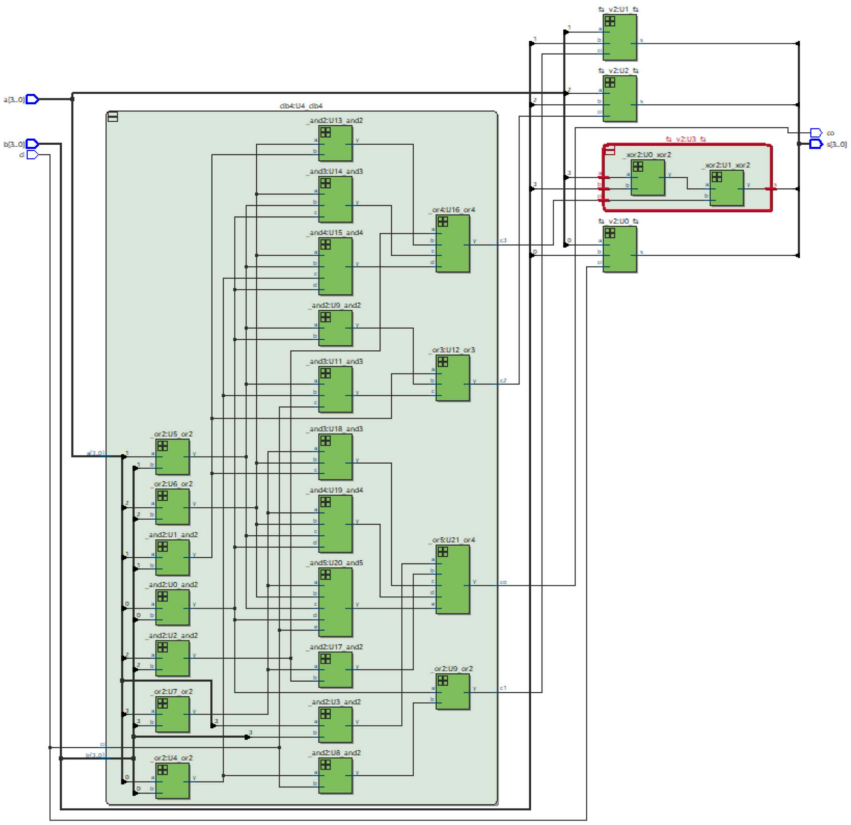


modified 32-bits CLA는 clock이 있으므로 operation frequency가 존재한다. 하지만 32-bits CLA는 clock이 존재하지 않아 operation frequency가 존재하지 않는다. size는 modified 32-bits CLA가 더 큰 것을 알 수 있다.

B. 합성(synthesis) 결과

- 4-bit CLA





위 사진은 4-bit CLA의 RTL map viewer이다. input에는 a[3:0}, b{3:0}, ci가 있고, output에는 s[3:0], co가있는 것을 알 수 있다. 한개의 CLB와 4개의 Full adder을 사용한 것을 확인할 수 있다.

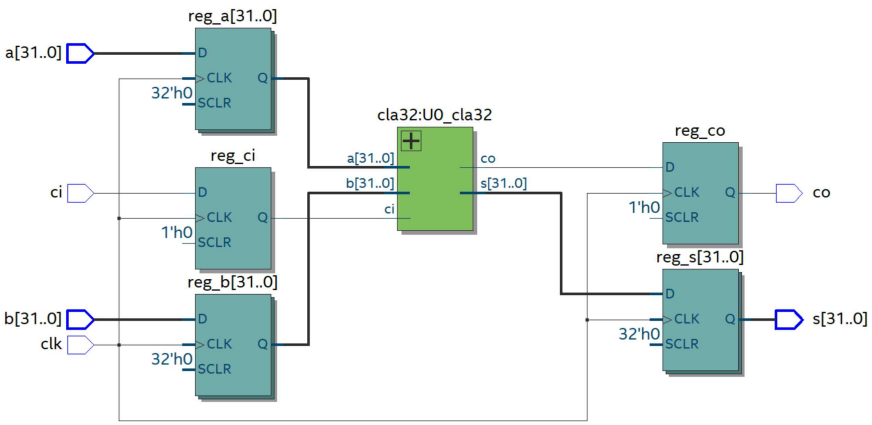
Flow Summary

<<Filter>>

Flow Status	Successful - Tue Sep 27 10:06:47 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	cla4
Top-level Entity Name	cla4
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	14
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

4-bit CLA의 flow summary이다. 총 14개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- CLA_CLK



위 사진은 CLA_CLK의 RTL map viewer이다. input에는 a[31:0}, b{31:0}, ci, clk(clock)가 있고, output에는 s[31:0], co가있는 것을 알 수 있다. cla32의 앞 뒤로 flip-flop이 연결된 것도 확인할 수 있었다.

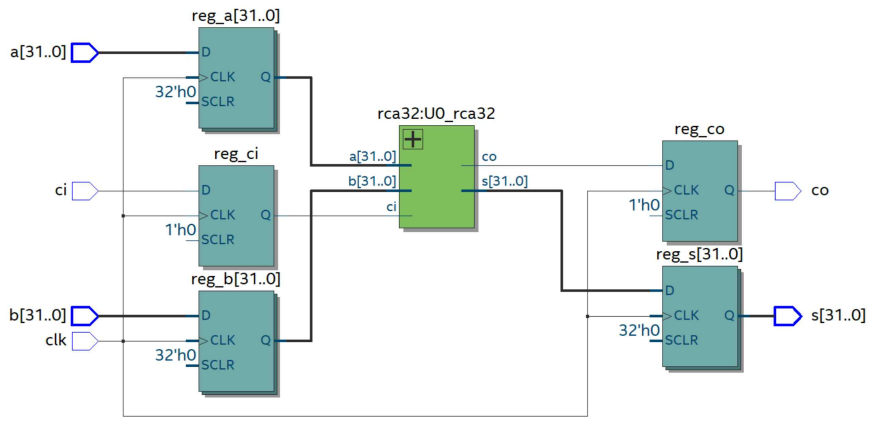
Flow Summary

<<Filter>>

Flow Status	Successful - Tue Sep 27 11:28:29 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	cla_clk
Top-level Entity Name	cla_clk
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	98
Total pins	99
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

CLA_CLK의 flow summary이다. 총 99개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- RCA_CLK



위 사진은 RCA_CLK의 RTL map viewer이다. input에는 a[31:0}, b{31:0}, ci, clk(clock)가 있고, output에는 s[31:0], co가있는 것을 알 수 있다. rca32의 앞 뒤로 flip-flop이 연결된 것도 확인할 수 있었다.

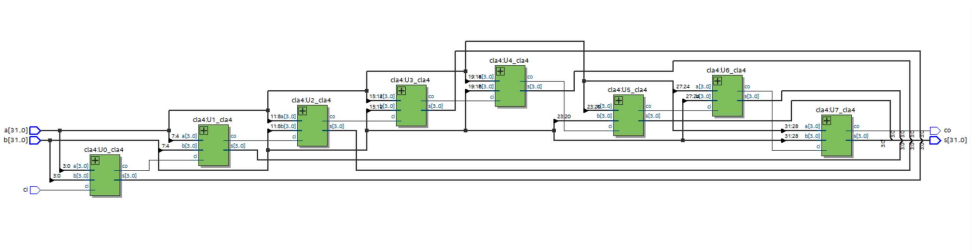
Flow Summary

<<Filter>>

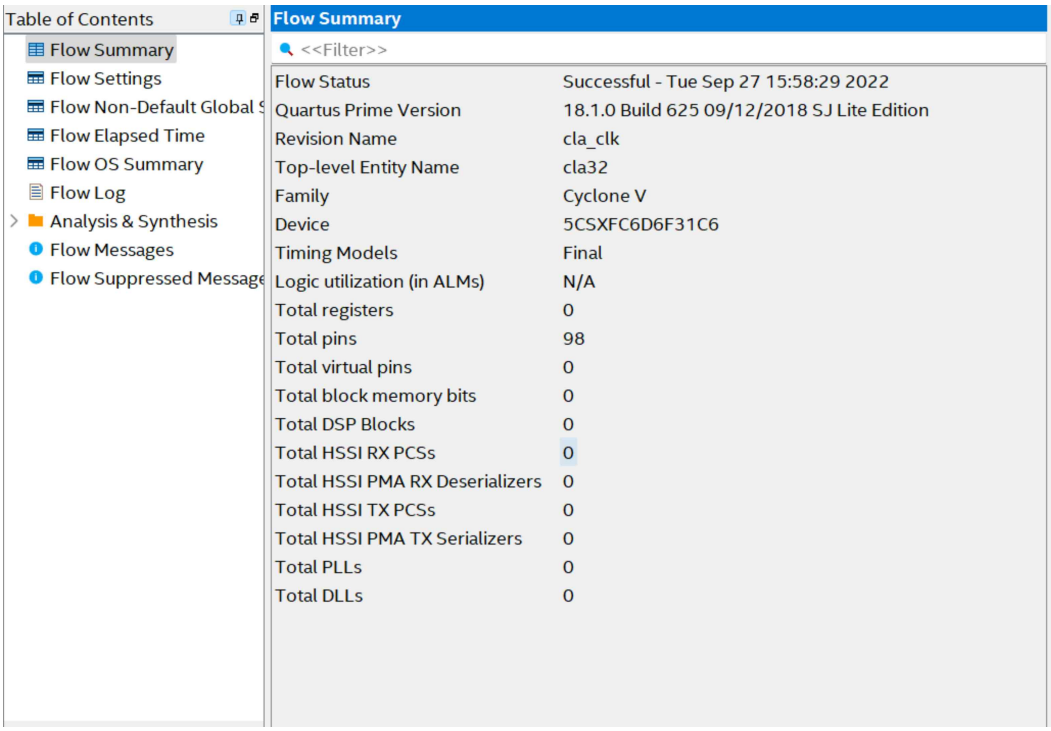
Flow Status	Successful - Tue Sep 27 16:52:36 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca_clk
Top-level Entity Name	rca_clk
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	98
Total pins	99
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

RCA_CLK의 flow summary이다. 총 99개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- cla32



위 사진은 CLA32의 RTL map viewer이다. input에는 a[31:0}, b{31:0}, ci가 있고, output에는 s[31:0], co가있는 것을 알 수 있다.



cla32의 flow summary이다. 총 98개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

5. 고찰 및 결론

A. 고찰

cla_clk의 testbench를 시행하던 중 waveform의 값이 틀리게 나오는 것을 확인했다. 코드를 다시 확인하던 중 clb의 p와 g를 바꿔써줬다는 것을 알았다. 이 점을 해결하니 waveform의 결과값이 올바르게 나오는 것을 확인할 수 있었다. 또한 TimeQuest Timing Analyzer을 처음 사용해 봐 report timing부분에서 잘못 설정해 많은 시간이 걸렸다. 한 번 실수를 통해 시간이 오래 걸리더라도 하나씩 차근차근 다시 수행해보는 과정을 겪으니 다음 실습에서는 좀 더 수월하게 진행할 수 있을 것 같다. 또한 `와 '차이나 '을 '라고 쓰는 차이에 의해서도 큰 차이가 발생하므로 작은 실수도 조심해야겠다는 생각이 들었다.

B. 결론

32-bit CLA는 32-bit RCA보다 크기가 크다. 하지만 n-bits RCA의 경우 Full adder를 n개 연결하여 설계하므로 n이 증가할수록 delay가 증가해 연산 속도가 느려지게 된다. 하지만 CLA는 이러한 RCA의 단점을 보완하고자 올림을 한꺼번에 계산하는 Carry Look-ahead Block, CLB를 활용하여 계산 속도를 줄일 수 있다.

실습 자료의 testbench에서 initial begin이 아닌 always문이 나와 이해하는데 다소 시간이 걸렸다. initial begin은 한 번 실행 후 end에 의해 종료한다. initial문은 언제 동작하는 것인지 정할 수 없어 always문을 사용하여 해결하는데, always문은 always @(sensitivity_list)의 형태로 사용할 수 있다는 것을 알았다.

6. 참고문헌

공영호 교수님/디지털논리회로2 강의자료/광운대학교 컴퓨터 공학과 2022 강의자료

공영호 교수님/컴퓨터공학기초실험2/광운대학교/2022 강의자료

공진흥 교수님/디지털논리회로1/광운대학교/2022 강의자료