컴퓨터 공학 기초 실험2 보고서

실험제목: 32-bit carry look-ahead adder

실험일자: 2018년 09월 05일 (수)

제출일자: 2018년 09월 11일 (화)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 수요일 0, 1, 2

학 번: 2015722025

성 명: 정 용 훈

1. 제목 및 목적

A. 제목

32-bit carry look-ahead adder

B. 목적

1주차에 공부한 RCA의 문제점을 파악하고 계산 속도가 왜 오래 걸리는지, 왜 그런 문제가 생기는지 이해하며 이를 보완하기 위한 CLA의 기능과 어떤 방법으로 문제가 보완되는지 이해하며 설계하고 설계한 내용을 하드웨어에 직접 연결하여 확인한다.

2. 원리(배경지식)

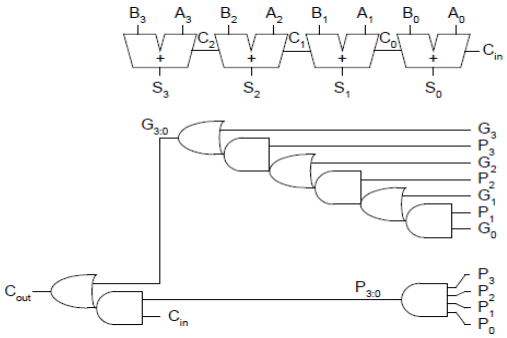
CLA를 설계하기 전에 CLA가 등장한 이유와 왜 필요한지 알아야한다. 우리가 처음 실험한 RCA의 문제점으로는 각 연결된 Full adder의 Co값이 결정되기 위해서는 순차적으로 계산이 진행해야 하며 이러한 이유로 마지막 Co값을 알기 위해서는 다소 시간이 많이 걸리는 단점이 있다. 이를 보완하기 위하여 등장한 하드웨어가 CLA이며 Propagation Signal(P)과 Generation Signal(G)을 사용하여 계산이 동시에 이루어지기 때문에 C의 값을 보다 빠르게 도출하여 동작시간을 감소 시킬 수 있다. 설계 순서로는 Co의 값을 따로 계산하는 CLB를 먼저 구현 후 Co값을 뺀 Full adder와 연결하면 설계를 할 수 있다. 앞서 말한 G, P는 bool식으로 나타내면 다음과 같이 나타낼 수 있다. $G_i=A_iB_i$, $P_i=A_i+B_i$ 두 식 모두 i 번째 Full adder의 carry out에 관련한 식이다. 이 식을 Full adder에 적용시키면 $C_{i+1}=C_i+A_iB_i+(A_i+B_i)C_i=G_i+P_iC_i$ 라는 식을 가질 수 있다. 각각의 Co값을 나타내면 아래와 같이 나타낼 수 있다.

$C_1=A_0B_0+(A_0+B_0)C_0=G_0+P_0C_0$

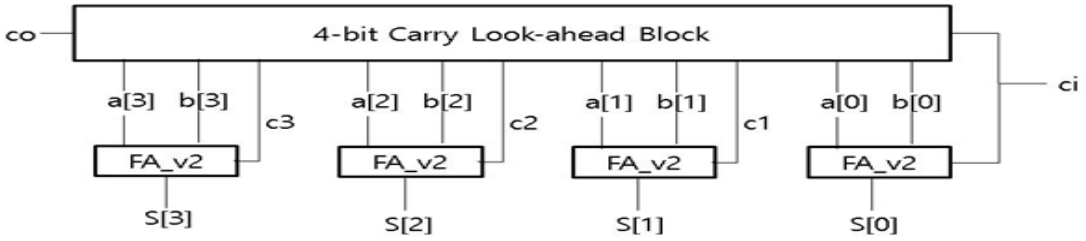
$C_2=A_1B_1+(A_1+B_1)C_1=G_1+P_1C_1$

$C_3=A_2B_2+(A_2+B_2)C_2=G_2+P_2C_2$

$C_{out}=A_3B_3+(A_3+B_3)C_3+G_3+P_3+C_3$



G식은 인풋 A, B가 모두 들어오게 되면 무조건 Co가 발생하는 것에서 식을 유도한 것이고 P식은 다른 Full adder에서 Ci으로 들어오는 신호와 A,B 인풋에 관련하여 Co을 결정하는 식이다. 이렇게 인풋 A, B를 가지고 Co의 결과 값을 따로 결정하기 때문에 계산하는 속도가 RCA보다 빠른 이유이다. 위에서 구한 불식을 가지고 CLB를 만들고 그 이후에 CLA를 만들면 된다. 아래는 CLB를 설계 하고 Carry가 분리된 Full adder를 합치면 아래와 같은 이미지가 나온다.

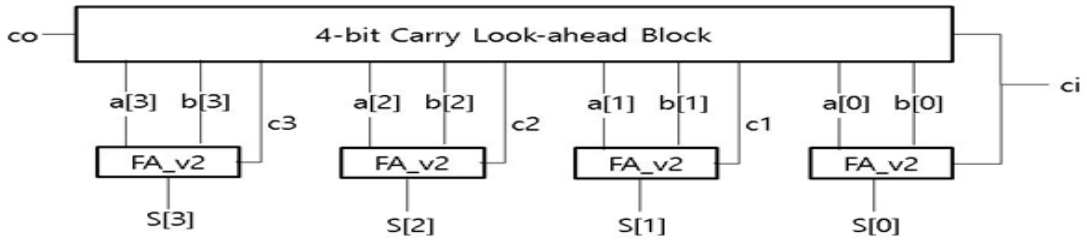


또한 이번 과제를 설계하고 검증하기 위하여 사용된 Verilog 문법은 clock과 관련된 always와 posedge, negedge 가 있다. Always는 기본적으로 모듈의 동작이 끝날 때까지 동작을 반복하는 것을 명령하며 뒤에 @ 표시와 조건이 따라올 경우 조건에 맞을 때 마다 동작을 수행한다. Posedge와 negedge는 조건 중 하나로 디지털논리회로 1에서 배웠던 clock의 rising edge와 falling edge를 뜻한다. 이번 설계에서는 플립플롭을 사용하여 검증하기 때문에 위와 같은 문법이 필요하다.

3. 설계 세부사항

이번 과제에서 설계한 내용으로는 크게 4-bit, 32-bit CLA, 32-bit CLA with CLK 로 나뉜다. 32-bit로 설계하는 이유는 16-bit 이하에서는 우리가 확인하고자 하는 delay의 차이를 볼 수 없기 때문이다. RCA는 비교를 위하여 저번주에 사용하였던 모듈을 그대로 가져와 사용 할 것이다. 새로 설계하는 것은 CLA이고 이제 CLA를 설계하고 검증하기 위한 각각 하드웨어의 세부적인 사항을 다뤄보겠다. (동작 자체는 첫 주에 실험 했던 RCA와 CLA 모두 동일하며 처리 속도에 대한 비교를 하기 위한 설계이다.)

a. 4-bit CLA



4-bit CLA는 위에 사항인 원리에서 언급 한 것처럼 CLB블록과 4개의 Co out이 없는 Full adder를 합쳐서 만든 하드웨어이다. 하드웨어를 설계하기 위하여 필요한 좀 더 낮은 단계의 모듈과 input과 output의 세부적인 사항은 아래 표와 같이 정리 할 수 있다.

I/O Description(Including Wire)

포트	이름	비트단위	설명
Input	a, b	4bit	Input data
	Ci	1bit	Carry in
Output	Co	4bit	Carry out
	S	1bit	Summation
wire	c1, c2, c3	1bit	Internal carry

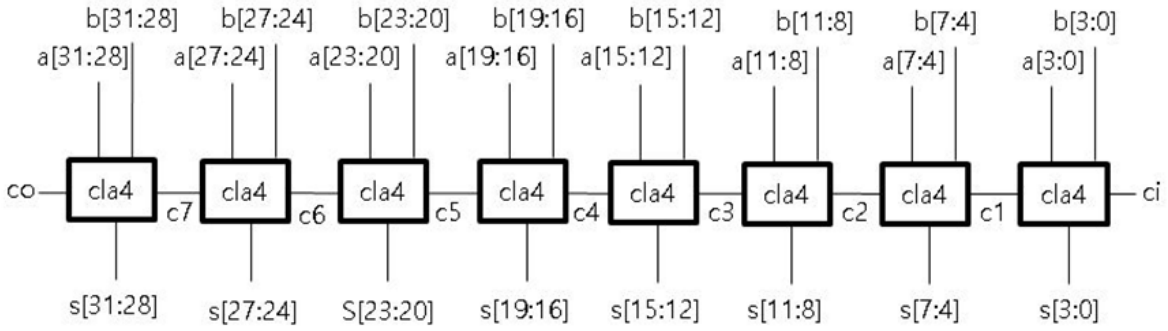
Module Description

구분	이름	설명
Module	Cla4	4-bit carry lokk-ahead adder(CLA)
Instance	U0_fa_v2	Instance
	U1_fa_v2	
	U2_fa_v2	
	U3_fa_v2	
	U0_clb4	Carry generator

위 표의 연결과 관련된 포트의 내용과 CLA의 이미지를 보면 쉽게 이해 할 수 있다. 아래 모듈과 관련된 표를 조금 설명하면 최종적인 모듈은 4-bit CLA이며 CLA를 구성하기 위해서는 새로 설계한 Full adder 4개와 원리에서 설명한 P, G 식을 이용한 CLB가 필요하며 이들을

포트와 관련시켜 연결하면 위와 같은 이미지를 가지는 하드웨어가 생성된다. 구현 한 모듈은 다음에 구현할 32bit-CLA를 구현하는 낮은 모듈로 사용 될 것이다.

b. 32-bit CLA



검증을 하기 위하여 구현하게 될 32-bit CLA이다. 실제 검증에서는 CLK이 같이 쓰인다. 검증에 대한 방법과 설명은 세부 설계를 보고 좀더 자세하게 알아보겠다.

I/O Description(Including Wire)

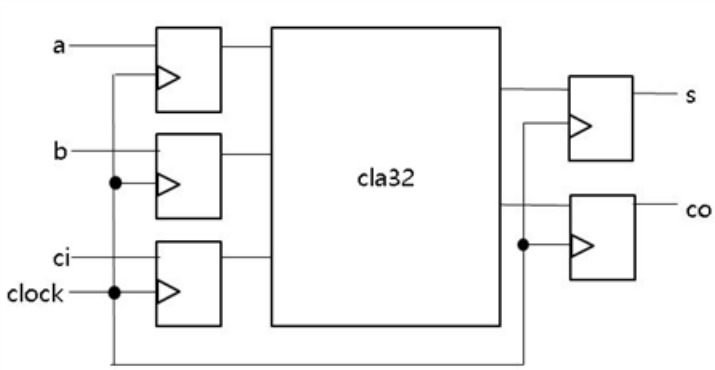
포트	이름	비트단위	설명
Input	a, b	32 bit	Input data
	ci	1 bit	Carry in
Output	co	1 bit	Carry out
	s	32 bit	Summation
wire	c	1 bit	Internal carry

Module Description

구분	이름	설명
Module	cla32	32-bit carry look-ahead adder(CLA)
instance	U0~U7_cla4	4-bit carry look-ahead adder(CLA)

위 하드웨어의 구성은 32-bit의 CLA를 구현하기 위한 세부적인 사항이다. 최종적으로 검증을 하기 위해서는 플립플롭을 사용하여 실험을 할 것이다. CLK이 들어가는 설계는 바로 다음 항목에서 진행한다.

c. 32-bit CLA(with CLK)



I/O Description(Including Wire and register)

포트	이름	비트단위	설명
Input	a, b	32 bit	Input data
	ci	1 bit	Carry in
	clock	1 bit	Clock
Output	co_cla	1 bit	Carry out
	s_cla	32 bit	Summation
Register	reg_a, reg_b	32 bit	Register
	reg_ci	1 bit	Register carry in
	reg_s_cla	32 bit	Register sum
	reg_co_cla	1 bit	Register carry out
wire	wire_s_cla	32 bit	Wire sum
	wire_co_cla	1 bit	Wire carry out

Module Description

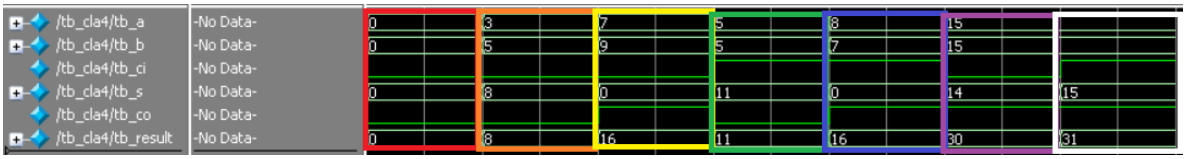
구분	이름	설명
Top module	cla_clk	32-bit carry look-ahead adder(CLA) with clock
Instance	U0_cla32	32-bit carry look-ahead adder(CLA)

위 그림은 표와 같이 보면 쉽게 이해 할 수 있다. 32-bit CLA와 많은 수의 플립플롭들을 간략하게 나타낸 이미지이며 앞서 말했듯이 CLK을 이용하여 Timing Analysis를 할 수 있으며 이를 통해 RCA와 CLA의 처리 속도를 비교 할 수 있다. 이에 대한 비교는 4번 항목인 설계 검증 및 실험 결과에서 비교 할 것이며 위 내용은 하드웨어 설계 자체에 의의를 둔다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

CLA4

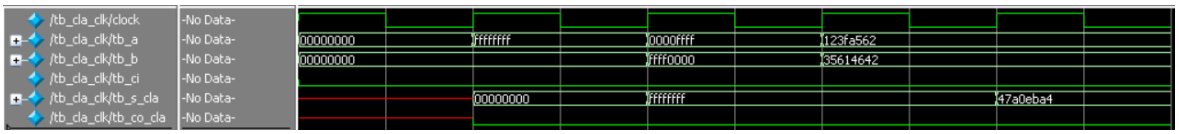


위 waveform은 CLA의 동작에 관련된 이미지이다 위의 동작을 표로 나타내면 아래와 같다.

Input			과정	result
a	b	Ci		
0(0)	0(0)	0	0+0+0	0
3(3)	5(5)	0	3+5+0	8
7(7)	9(9)	0	7+9+0	16
5(5)	5(5)	1	5+5+1	11
8(8)	7(7)	1	8+7+1	16
15(f)	15(f)	0	15+15+0	30
15(f)	15(f)	1	15+15+1	31

(괄호 안의 수는 hexadecimal 이다.)

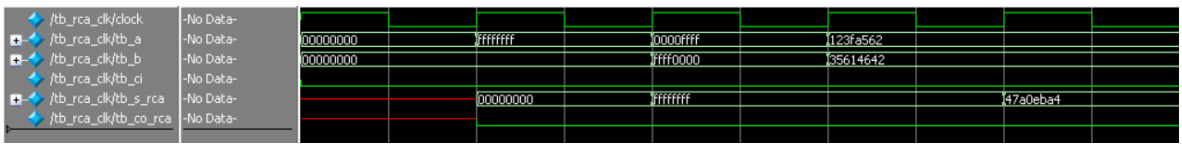
CLA_CLK



input		과정	result
tb_a	tb_b		
00000000	00000000	00000000+00000000	00000000
ffffffff	00000000	ffffffff+00000000	ffffffff
0000ffff	ffff0000	0000ffff+ffff0000	ffffffff
123fa562	35614642	123fa562+35614642	47a0eba4

위에 있는 input값을 test bench에 정보로 주어 waveform을 나타낸 결과이다 과정을 보면 clock이 rising edge일 때 마다 값이 변하는 것을 볼 수 있다.

RCA_CLK



input		과정	result
tb_a	tb_b		
00000000	00000000	00000000+00000000	00000000
Ffffffff	00000000	ffffffff+00000000	ffffffff
0000ffff	ffff0000	0000ffff+ffff0000	ffffffff
123fa562	35614642	123fa562+35614642	47a0eba4

미리 설계해둔 RCA의 정보를 가지고 32-bit RCA를 만든 후 인풋 값을 넣어 결과를 확인한 waveform과 그에 대한 과정을 정리한 표이다.

Timing Analyze

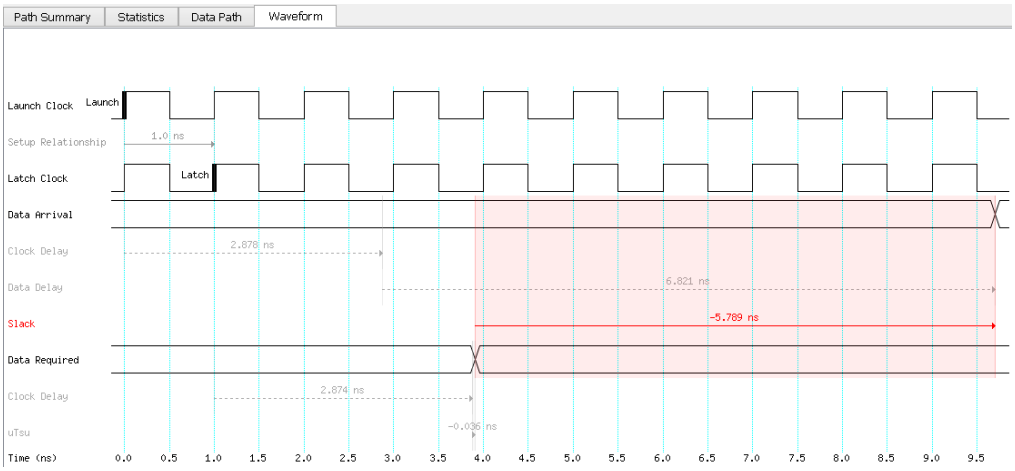
아래 그림과 설명은 Timing Analyzer를 사용하여 실험 하고자 하는 각 하드웨어의 처리속도를 비교하기 위하여 프로그램을 통하여 실험한 과정과 결과이다.

CLA_CLK

Setup: clock

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-5.789	reg_b[2]	reg_s_cla[31]	clock	clock	1.000	-0.004	6.821
2	-5.789	reg_b[2]	reg_co_cla	clock	clock	1.000	-0.004	6.821
3	-5.786	reg_b[2]	reg_s_cla[30]	clock	clock	1.000	-0.004	6.818
4	-5.743	reg_b[3]	reg_s_cla[31]	clock	clock	1.000	-0.004	6.775
5	-5.743	reg_b[3]	reg_co_cla	clock	clock	1.000	-0.004	6.775
6	-5.740	reg_b[3]	reg_s_cla[30]	clock	clock	1.000	-0.004	6.772
7	-5.671	reg_b[2]	reg_s_cla[26]	clock	clock	1.000	0.007	6.714
8	-5.670	reg_b[2]	reg_s_cla[27]	clock	clock	1.000	0.007	6.713
9	-5.638	reg_b[2]	reg_s_cla[23]	clock	clock	1.000	-0.006	6.668
10	-5.637	reg_b[0]	reg_s_cla[31]	clock	clock	1.000	-0.004	6.669



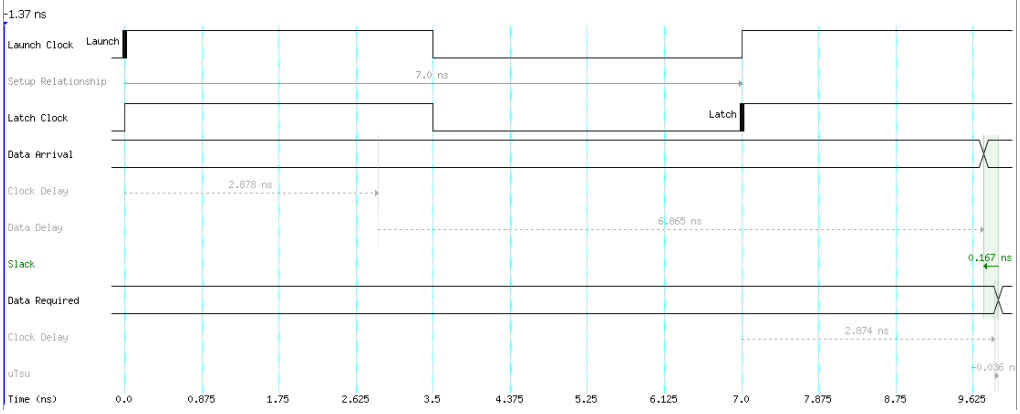
위 사진의 결과는 구현한 하드웨어의 input값에 대한 처리 시간이 부족하여 발생하는 현상이다. 처음 분석을 하게 되면 Clock의 값이 1ns로 초기화 되어있기 때문에 정상적으로 하드웨어가 작동하기 위해서는 Clock의 값을 사용자가 따로 정의 해주어야 한다. 정의 해주는 값은 따로 계산 하는 방법이 있지만 이번 CLA 하드웨어에서는 메뉴얼에 나와 있는 데로 시간을 7ns로 설정하여 다시 컴파일 하면 아래와 같은 결과가 나온다.

Setup: clock

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	0.167	reg_b[0]	reg_co_cla	clock	clock	7.000	-0.004	6.865
2	0.168	reg_b[0]	reg_s_cla[31]	clock	clock	7.000	-0.004	6.864
3	0.177	reg_b[0]	reg_s_cla[30]	clock	clock	7.000	-0.004	6.855
4	0.182	reg_b[0]	reg_s_cla[26]	clock	clock	7.000	0.007	6.861
5	0.219	reg_b[0]	reg_s_cla[29]	clock	clock	7.000	-0.001	6.816
6	0.322	reg_b[0]	reg_s_cla[27]	clock	clock	7.000	0.007	6.721
7	0.342	reg_b[0]	reg_s_cla[23]	clock	clock	7.000	-0.006	6.688
8	0.374	reg_b[2]	reg_co_cla	clock	clock	7.000	-0.004	6.658
9	0.375	reg_b[2]	reg_s_cla[31]	clock	clock	7.000	-0.004	6.657
10	0.382	reg_a[0]	reg_co_cla	clock	clock	7.000	-0.004	6.650

Path Summary | Statistics | Data Path | Waveform



두 결과를 비교해보면 slack의 값이 음수에서 양수로 변한 것을 볼 수 있다. Slack의 값은 Clock이 다시 rising하는 시간에서 하드웨어가 동작하기 위한 시간을 빼 준 값으로 Slack이 양수일 때 하드웨어가 정상적으로 작동하는 것을 볼 수 있다. 정상적인 값을 가지는 하드웨어의 그래프를 보면 0.167ns의 여유시간을 가지는 것을 확인 할 수 있다.

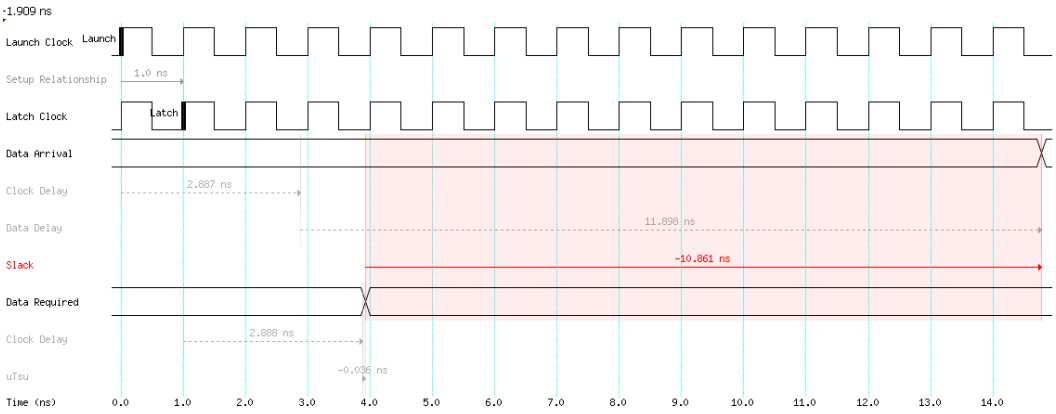
RCA_CLK

Setup: clock

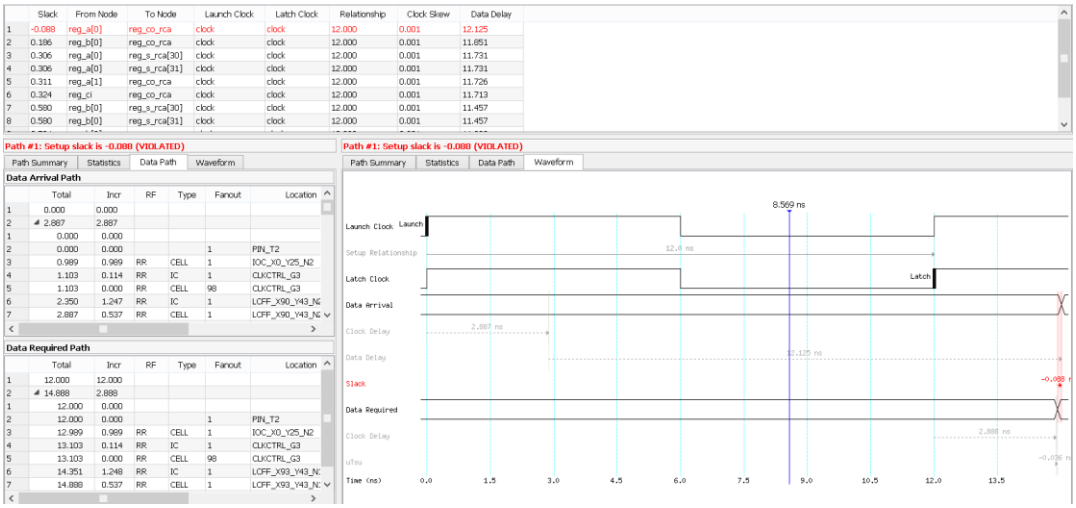
Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-10.861	reg_a[0]	reg_co_rca	clock	clock	1.000	0.001	11.898
2	-10.590	reg_b[0]	reg_co_rca	clock	clock	1.000	0.001	11.627
3	-10.478	reg_a[0]	reg_s_rca[30]	clock	clock	1.000	0.001	11.515
4	-10.478	reg_a[0]	reg_s_rca[31]	clock	clock	1.000	0.001	11.515
5	-10.455	reg_a[1]	reg_co_rca	clock	clock	1.000	0.001	11.492
6	-10.449	reg_ci	reg_co_rca	clock	clock	1.000	0.001	11.486
7	-10.207	reg_b[0]	reg_s_rca[30]	clock	clock	1.000	0.001	11.244
8	-10.207	reg_b[0]	reg_s_rca[31]	clock	clock	1.000	0.001	11.244
9	-10.106	reg_a[4]	reg_co_rca	clock	clock	1.000	0.001	11.143
10	-10.072	reg_a[1]	reg_s_rca[30]	clock	clock	1.000	0.001	11.109

Path Summary | Statistics | Data Path | Waveform



다음으로는 비교를 하기위한 32-bit RCA의 타이밍 분석이다. 표와 그래프를 보았을 때 기존에 설정 되어있던 시간을 12ns 로 바꾸어 주면 문제가 해결 될 것으로 보아 12ns로 설정 해준 후 다시 컴파일하여 아래와 같은 값을 가지게 되었다.



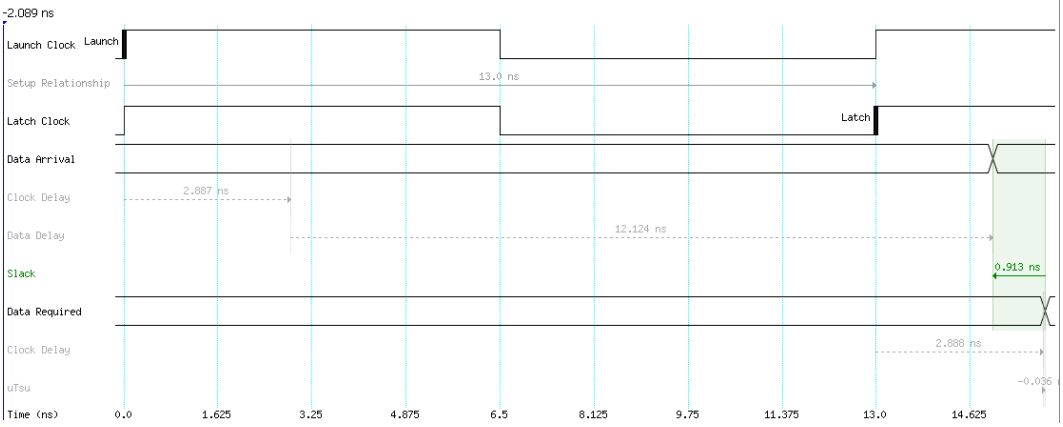
한부분에서 처리 시간에 비해 rising이 빨리 올라가 하드웨어가 정상적으로 처리되지 않는 것이 확인되었다. 이는 처음 1ns로 설정되어 있는 시간이 정확한 1ns의 값을 가지고 있지 않다는 것을 알 수 있으며 수업 중에 교수님이 한번 언급하신 내용이다.

Setup: clock

Command Info | Summary of Paths

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	0.913	reg_a[0]	reg_co_rca	clock	clock	13.000	0.001	12.124
2	1.180	reg_b[0]	reg_co_rca	clock	clock	13.000	0.001	11.857
3	1.306	reg_a[0]	reg_s_rca[30]	clock	clock	13.000	0.001	11.731
4	1.306	reg_a[0]	reg_s_rca[31]	clock	clock	13.000	0.001	11.731
5	1.325	reg_a[1]	reg_co_rca	clock	clock	13.000	0.001	11.712
6	1.328	reg_ci	reg_co_rca	clock	clock	13.000	0.001	11.709
7	1.482	reg_b[2]	reg_co_rca	clock	clock	13.000	0.001	11.555
8	1.573	reg_b[0]	reg_s_rca[30]	clock	clock	13.000	0.001	11.464
9	1.573	reg_b[0]	reg_s_rca[31]	clock	clock	13.000	0.001	11.464
10	1.718	reg_a[1]	reg_s_rca[30]	clock	clock	13.000	0.001	11.319

Path Summary | Statistics | Data Path | Waveform

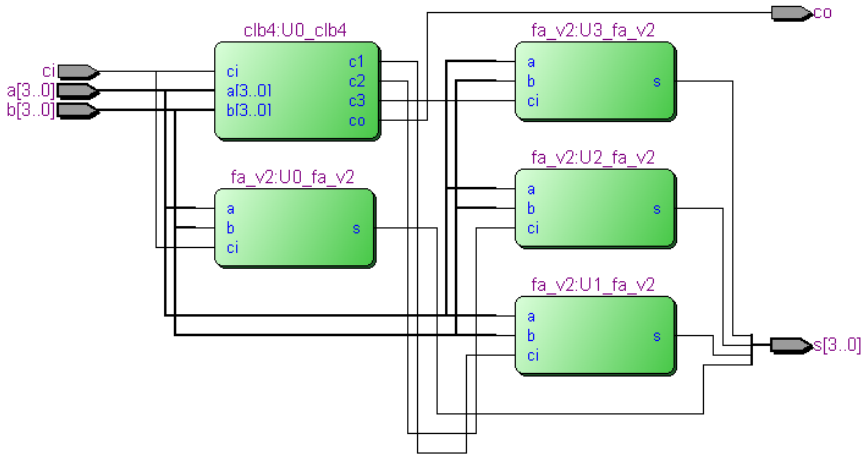


그리하여 시간 값을 13ns로 다시 설정하여 새로운 값을 얻어 낸 것이 위에 나오는 표의 값과 그래프의 값이다 여유 시간을 보면 0.913ns로 양수 값을 가지므로 하드웨어가 정상적으로 작동 할 수 있다는 것을 알 수 있다.

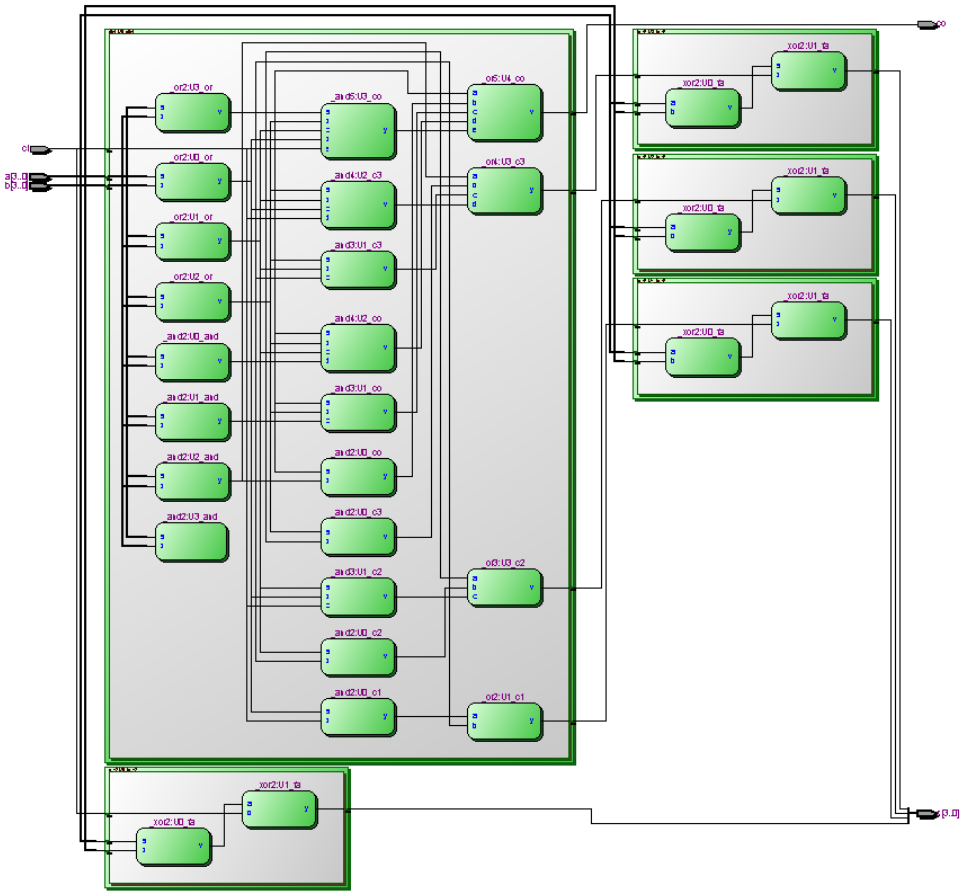
B. 합성(synthesis) 결과

CLA4

RTL Viewer

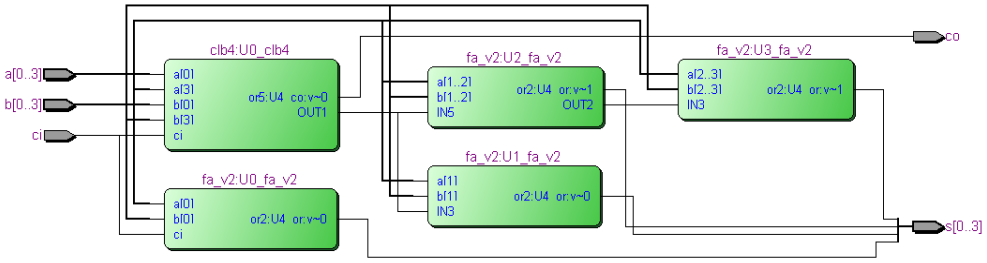


합성 결과를 보면 CLB에 Full adder 4개가 연결된 것을 확인 할 수 있다.



4-bit CLA의 RTL Viewer이다 프로그램을 통하여 설계한 것과 동일하게 나온 것을 확인 할 수 있다. 특히 아래 그림은 CLB의 세부 구조와 Full adder의 세부 구조를 같이 볼 수 있다.

Technology Map Viewer

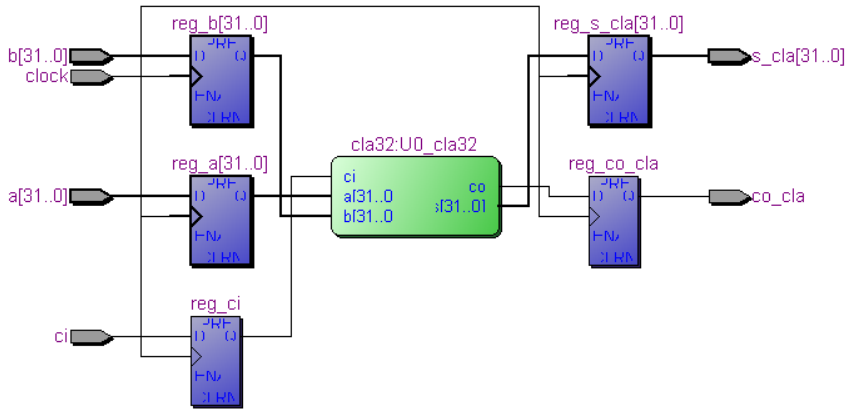


Flow summary

Flow Summary	
Flow Status	Successful - Sat Sep 08 17:27:46 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	cla4
Top-level Entity Name	cla4
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	8 / 68,416 (< 1 %)
Total combinational functions	8 / 68,416 (< 1 %)
Dedicated logic registers	0 / 68,416 (0 %)
Total registers	0
Total pins	14 / 622 (2 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

CLA_CLK

RTL Viewer



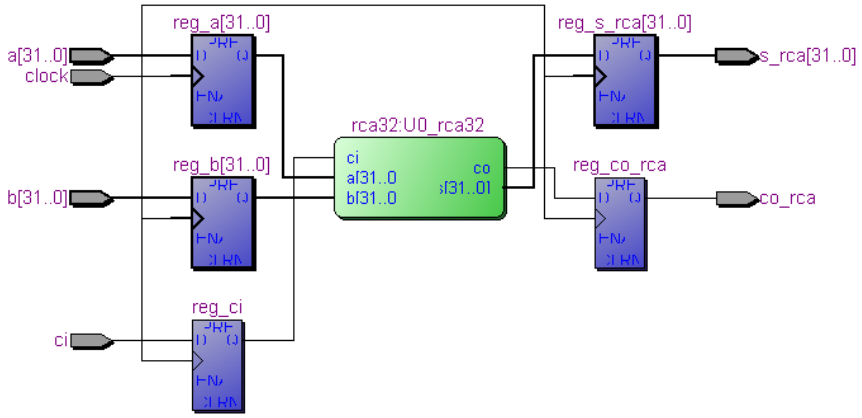
32-bit CLA에 플립플롭이 같이 연결된 것을 확인 할 수 있다. CLA는 미리 설계하였더 4-bit CLA를 8개를 연결 한 것으로 Viewer를 봤을 때 한 블록으로 간단하게 나타냈다.

Flow summary

Flow Summary	
Flow Status	Successful - Sat Sep 08 17:39:35 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	cla_clk
Top-level Entity Name	cla_clk
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	165 / 68,416 (< 1 %)
Total combinational functions	129 / 68,416 (< 1 %)
Dedicated logic registers	98 / 68,416 (< 1 %)
Total registers	98
Total pins	99 / 622 (16 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

RCA_CLK

RTL Viewer



지난번 설계한 RCA의 정보를 가져와 32-bit로 설계한 후 동일하게 플립플롭을 같이 연결한 모습을 볼 수 있다.

Flow summary

Flow Summary	
Flow Status	Successful - Sun Sep 09 11:27:22 2018
Quartus II 64-Bit Version	13.0.0 Build 156 04/24/2013 SJ Web Edition
Revision Name	rca_clk
Top-level Entity Name	rca_clk
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Total logic elements	114 / 68,416 (< 1 %)
Total combinational functions	79 / 68,416 (< 1 %)
Dedicated logic registers	98 / 68,416 (< 1 %)
Total registers	98
Total pins	99 / 622 (16 %)
Total virtual pins	0
Total memory bits	0 / 1,152,000 (0 %)
Embedded Multiplier 9-bit elements	0 / 300 (0 %)
Total PLLs	0 / 4 (0 %)

5. 고찰 및 결론

A. 고찰

이번 과제에서는 플립플롭을 사용하여 검증하는 방법을 추가적으로 사용하였습니다. 처음 타이밍 시뮬레이션을 돌렸을 경우 음수의 slack이 발생하며 하드웨어가 정상적으로 돌아가지 않는 문제가 발생 했습니다. Wave form을 확인하여 부족한 시간을 계산하여 반올림하고 그에 맞는 값으로 변경 후 다시 컴파일 하여 문제를 해결하여 정상적인 데이터를 받을 수 있게 고쳤습니다.

B. 결론

이번 과제에서는 CLA를 설계하고 전에 설계했던 RCA의 정보를 가져와 두 하드웨어의 처리 속도를 비교하고 CLA의 장점을 이해하는 실험이었습니다. 실제 결과 값으로도 플립플롭의 시간이 CLA가 7ns RCA가 13ns 로 설정을 해줘야 정상적으로 동작하는 것을 확인하였습니다. 설계 과정에서 프로그램을 통해 각 하드웨어의 CLK이 rising 하는 시간이 1ns로 짧게 초기화되어 있어 하드웨어의 처리 시간을 계산하여 시간을 맞춰주어 하드웨어가 정상적으로 돌아갈 수 있도록 설정을 해주어야 했습니다. 하드웨어의 delay와 처리 시간이 있다는 것을 이론적으로만 배우프로그램을 통해서 직접 눈으로 보게 되어 훨씬 이해가 잘되었습니다. 또한 이번 실험을 통해 아무리 같은 동작을 하는 하드웨어가 있더라도 설계를 다르게 함으로써 그에 대한 효율을 다르게 할 수 있다는 것을 배웠고 어떠한 일을 할 때 효율적인 설계가 굉장히 중요하다는 것을 알게 되었습니다.

6. 참고문헌

여러가지 adder의 종류와 출력 시간 단축// https://blog.naver.com/klp0712/221064682909

기본적인 CLA의 설계 구성과 이해 // https://blog.naver.com/lobdo777/220433382460

Carry Look Ahead의 사전적 의미 //
https://terms.naver.com/entry.nhn?docId=1585329&cid=50376&categoryId=50376

김영민 / 디지털 논리회로2 / 광운대학교(컴퓨터정보공학부) / 2018