컴퓨터 공학 기초 실험2 보고서

실험제목: Assignment_03 CLA

실험일자: 2020년 09월 21일 (월)

제출일자: 2020년 09월 28일 (월)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2019202052

성 명: 김 호 성

1. 제목 및 목적

A. 제목

4bit carry look ahead, 32bit carry look ahead, 32bit Ripple carry adder

B. 목적

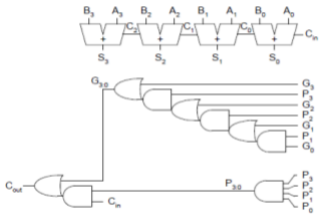
- 4bit CLA에 대해 이해하고, 프로그래밍한다.
- 32bit CLA에 대해 이해하고, 프로그래밍한다.
- 32bit RCA에 대해 이해하고, 프로그래밍한다.

2. 원리(배경지식)

CLA의 설계 이전에 CLA가 등장한 배경을 알아야한다. 우리가 이전에 실험하여 만들어 낸 RCA의 문제점으로 지적된 부분으로는, RCA 내부의 Full Adder의 캐리 값이 결정되기 위해서 순서대로 계산이 진행되어야 하며, 이 때문에 마지막 캐리아웃 값을 알기 위해 시간이 오래 걸리는 문제점이 있다. 이를 충족시키기 위하여 나온 것이 CLA: Carry Look Ahead이며, Propagation Signal과 Generation Signal을 사용하여 계산이 동시에 이루어지므로 캐리 값을 보다 빠르게 도출해 내어 계산하는 시간을 단축시킬 수 있다. 설계 순서로는 캐리 아웃 값을 따로 계산하는 Carry Look Ahead block을 먼저 만들어서 carry값을 뺀 Full Adder 와 연결하면 설계가 가능하다. 미리 언급한 Propagation Signal 과 Generation Signal 은 bool식으로 나타내면 Gi=Ai*Bi, Pi=Ai+Bi 로 나타낼 수 있고 두 식 모두 (i) 번째 Full Adder의 캐리아웃 값에 관련된 식이다. 준식을 Full Adder에 적용시키면 다음과 같은 식을 갖는다.

Ci+1=Ci+AiBi+(Ai+Bi)Ci=Gi+PiCi

이때, 각각의 Co값을 나타내면 아래와 같이 나타낼 수 있다.



C1=A0B0+(A0+B0)C0=G0+P0C0

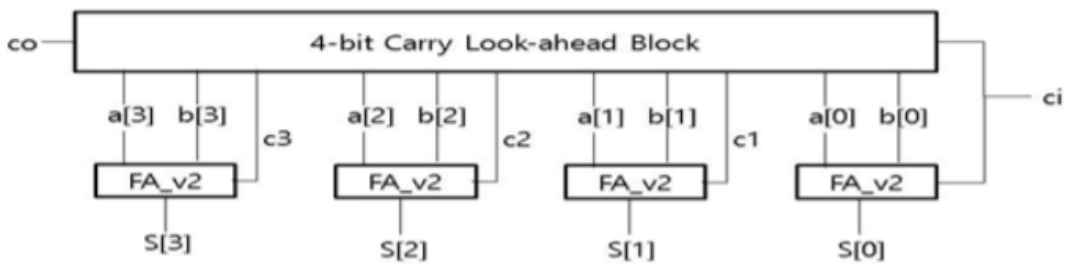
C2=A1B1+(A1+B1)C1=G1+P1C1

C3=A2B2+(A2+B2)C2=G2+P2C2

Cout=A3B3+(A3+B3)C3+G3+P3+C3

G 관련 식은 input A, B 가 모두 들어온다면 무조건 Carry Out 이 발생하는 것에 기초하여 식을 만들어낸 것이고, P 관련 식은 Full Adder에 Carry 가 들어오는 신호와 A, B input 에 관련하여 carry Out을 선별하는 식이다. 이렇게 input A, B 를 갖고 Carry Out 의 결과를 따로 결정하기 때문에 앞서 실습한 Ripple Carry Adder 보다 빠른 이유이다, 위에 서구한 bool식을 갖고 Carry Look Ahead block 을 만들고 그 이후 Carry Look Ahead 를 본격적으로 만들면 된다, 아래의 이미지는 Carry Look Ahead block 을설계하고 Carry 가 분

리된 Full Adder를 합친 것이다.

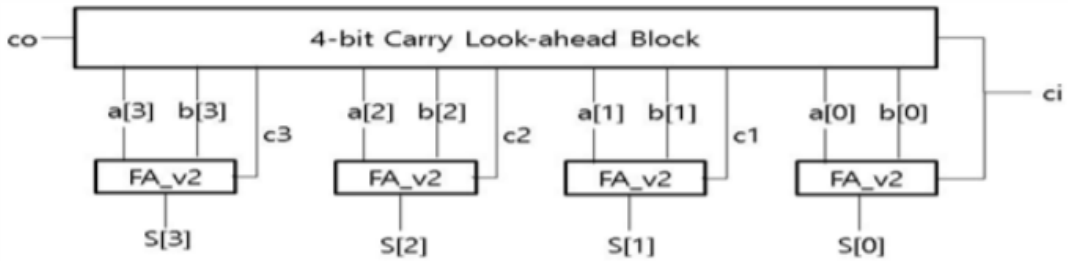


이번 실습에서의 Verilog 에서는 Always 와 clock 그리고 timing analysis, pin planner 등 여러가지 이전 RCA에서 했던 것과 다른 내용들을 다루었다.

3. 설계 세부사항

이번주의 실습에서 실험한 내용으로는 cla4, cla32, 32-cla-clk 로 나뉜다. 왜 32-bit 로 설계하여 실험하였는지 하면, 16-bit 아래로는 우리가 timing analysis를 통하여 delay를 비교해 주어야 하는데 그 delay 가 잘 확인할 수 없기 때문이다. 앞서 말한 것처럼 delay 비교는 rca와 cla를 비교할 것이고 rca는 지난 실습시간에 작성한 내용을 활용할 것이다.

4-bit CLA



4-bit CLA는 앞서 언급한 바와 같이 CLB 와 4개의 Carry 가 없는 Full adder를 합성하여 만든 하드웨어이다, 이 하드웨어를 만들기 위해서 필요한 정보는 아래의 표와 같다.

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

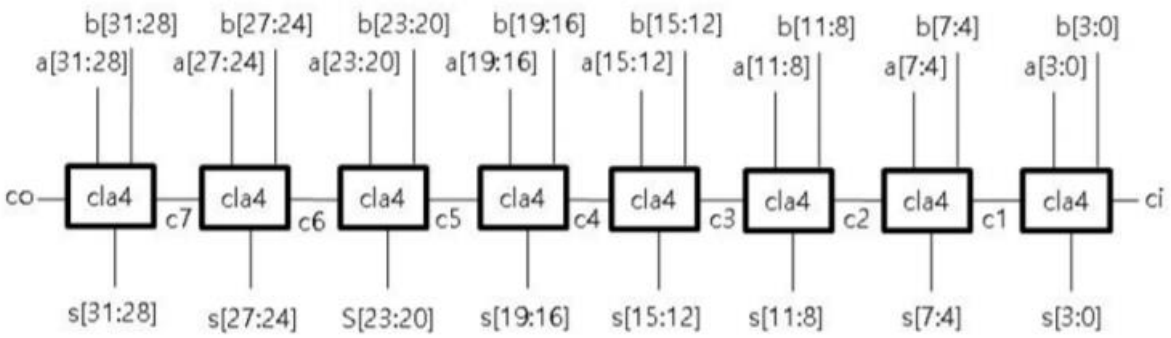
Module Description

Classification	Name	Description
Module	cla4	4-bit carry look-ahead adder

Instance	U0_fa_v2	Full adder
	U1_fa_v2	
	U2_fa_v2	
	U3_fa_v2	
	U4_clb4	Carry generation

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. 구현한 모듈은 다음에 구현할 32bit-CLA를 구현하는 낮은 모듈로 사용될 것이다.

32-bit CLA



검증을 목적으로 구현하게 될 32 bit CLA이다. 실검증에서 CLK 가 같이 쓰이며, 검증에 대한 방법과 설명은 세부 설계 이후에 자세하게 기술하겠다.

I/O Description(Including Wire)

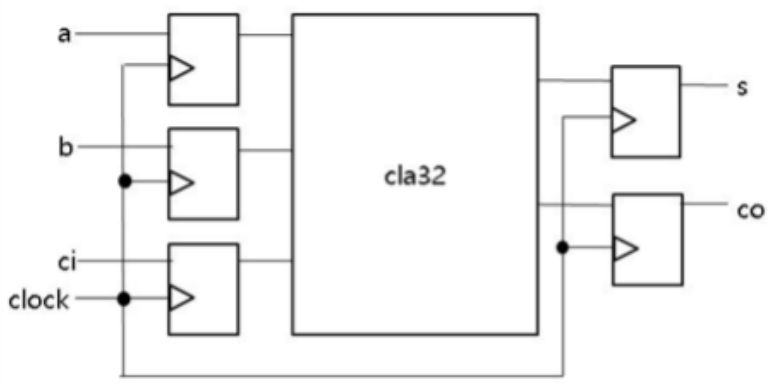
포트	이름	비트단위	설명
Input	a, b	32 bit	Input data
	ci	1 bit	Carry in
Output	co	1 bit	Carry out
	s	32 bit	Summation
wire	c	1 bit	Internal carry

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 flip-flop을 사용하여 실험을 할 것이다. CLK이 들어가는 설계는 바로 다음 항목에서 진행한다.

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)

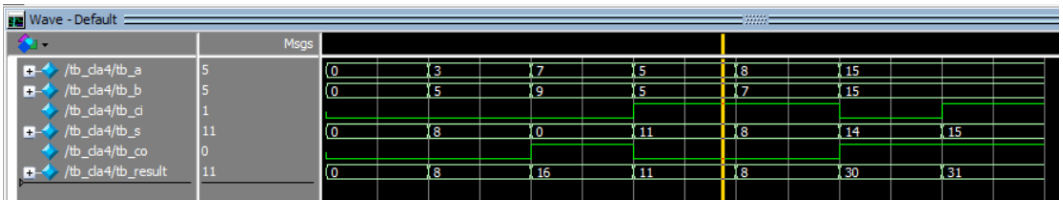
위 그림은 표와 같이 보면 쉽게 이해할 수 있다. 32-bit CLA를 간략하게 나타낸 이미지이며 앞서 말했듯이 CLK을 이용하여 Timing Analysis를 할 수 있으며 이를 통해 RCA와 CLA의 처리 속도를 비교할 수 있다. 이에 대한 비교내용은 4번 항목인 설계 검증 및 실험 결과에서 비교할 것이며 위 내용은 하드웨어 설계 자체에 의미를 둔다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

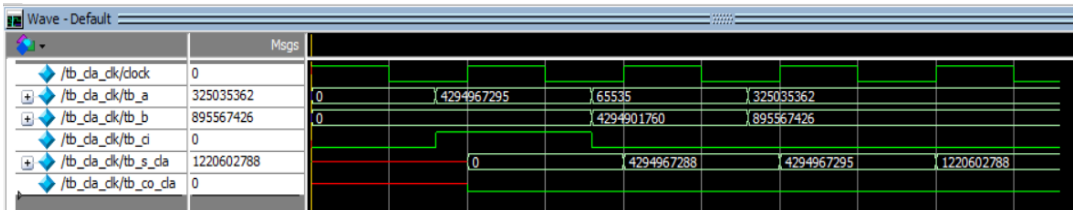
CLA4

waveform



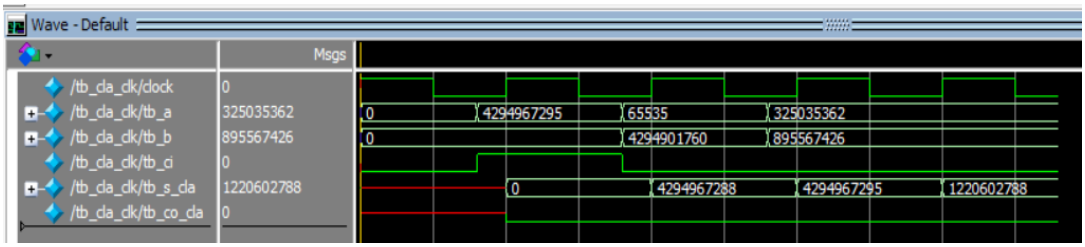
CLA_CLK

waveform



RCA_CLK

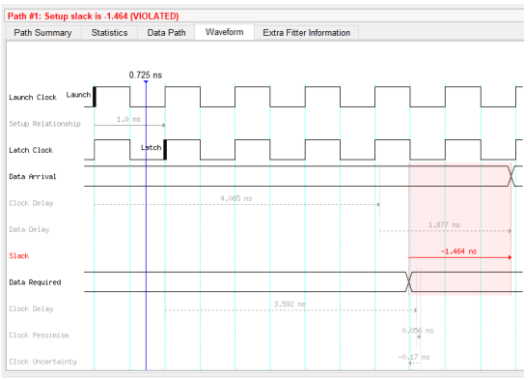
waveform



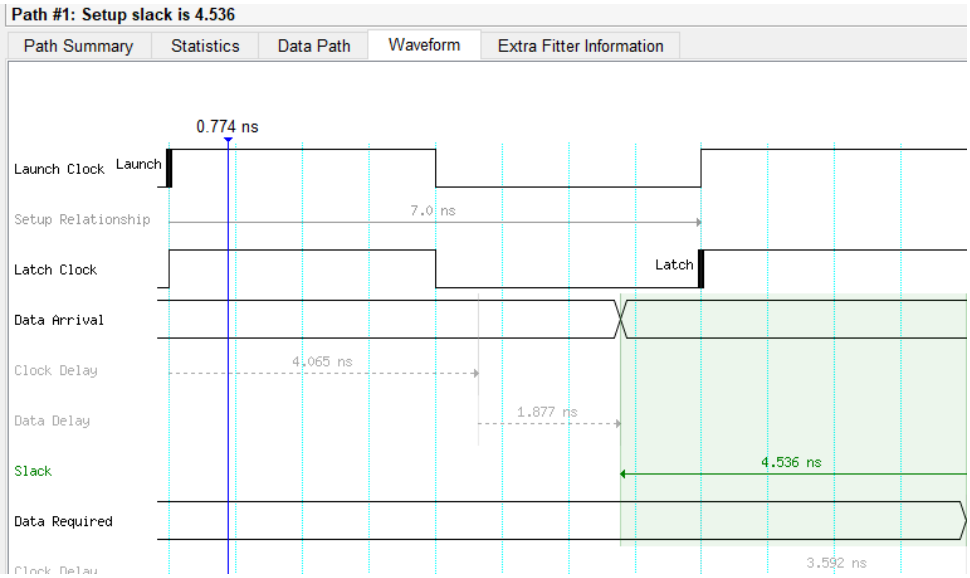
Timing analysis

아래의 그림과 설명은 Timing analyzer를 활용하여 실험하고자 하는 각 하드웨어의 처리속도를 비교하기 위해 실험한 결과이다.

CLA_CLK

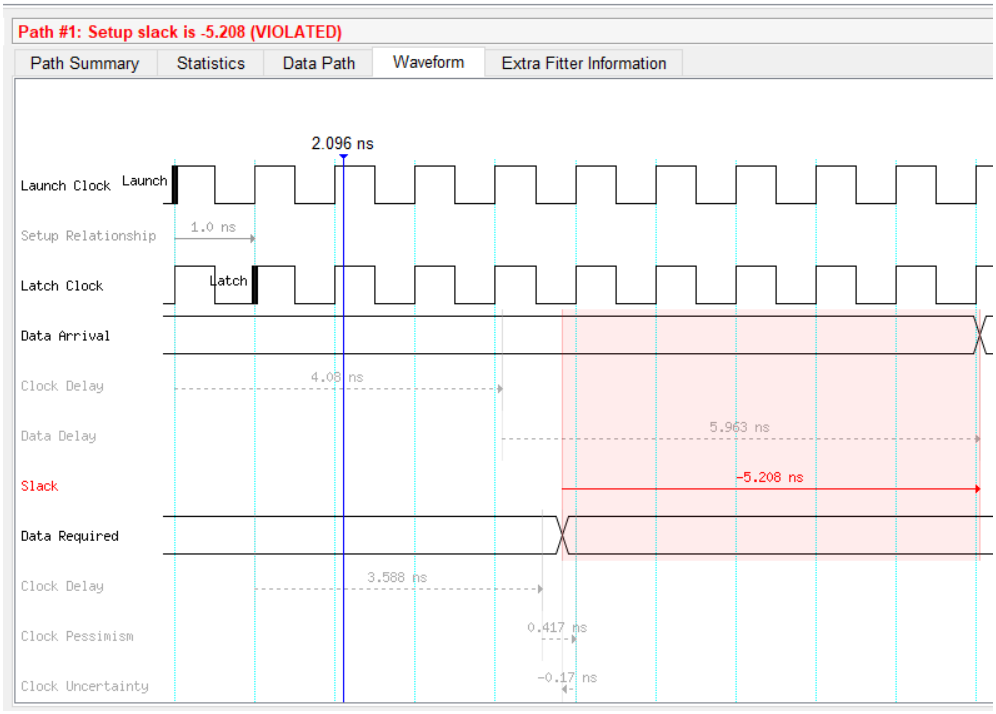


위 사진의 결과는 구현한 하드웨어의 input값에 대한 처리 시간이 부족하여 발생하는 현상이다. 처음 분석을 하게 되면 Clock의 값이 1ns로 초기화 되어있기 때문에 정상적으로 하드웨어가 작동하기 위해서는 Clock의 값을 사용자가 따로 정의해주어야 한다. 아래는 -1.464보다 절대값이 큰 양수를 더한 후의 상태이다.

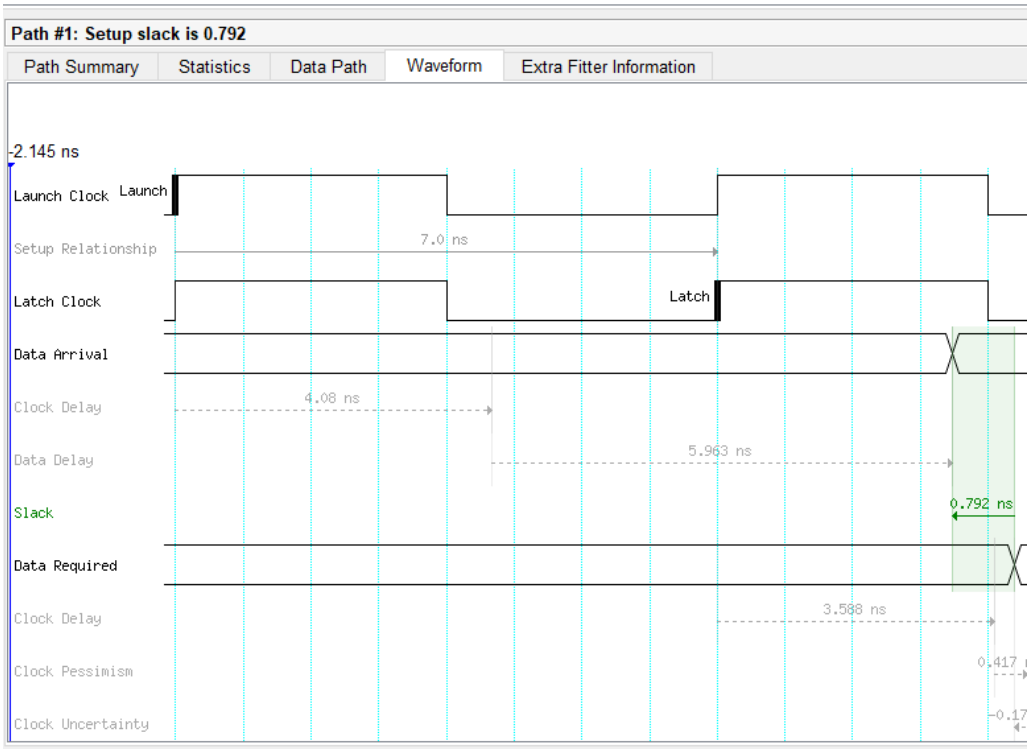


두 결과를 비교해보면 slack의 값이 음수에서 양수로 변한 것을 볼 수 있다. Slack의 값은 Clock이 다시 rising하는 시간에서 하드웨어가 동작하기 위한 시간을 빼 준 값으로 Slack 이 양수일 때 하드웨어가 정상적으로 작동하는 것을 볼 수 있다. 정상적인 값을 가지는 하드웨어의 그래프를 보면 0.792ns의 여유시간을 가지는 것을 확인할 수 있다.

RCA_CLK



이번에는 비교를 위하여 RCA의 타이밍분석이다. 5.208이상의 양수를 넣어주면 해결될 것이다.



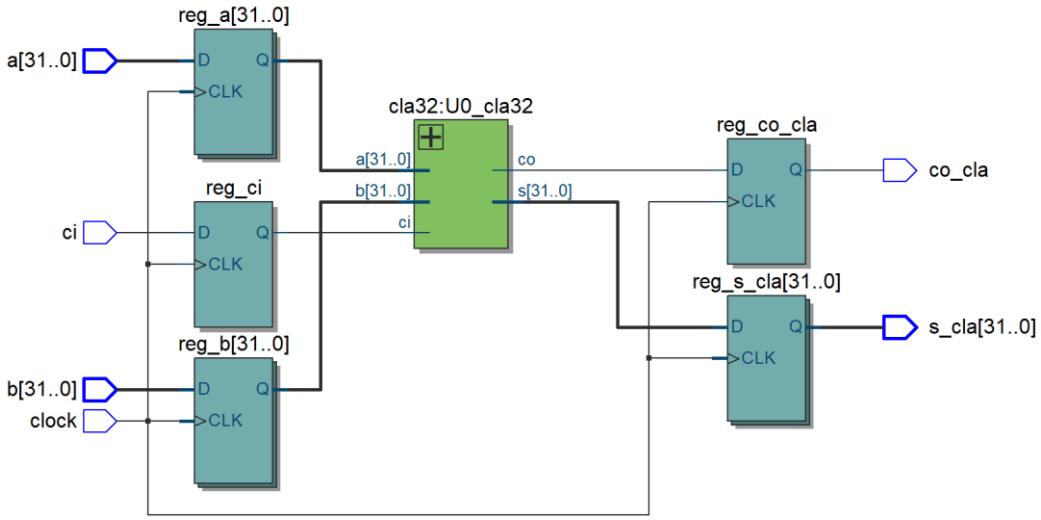
7ns의 delay를 주어 하드웨어가 정상적으로 작동하게끔 해주었다.

여기서 확인할 수 있는 점은 RCA의 경우 CLA보다 3.744ns만큼 오래 걸리는 것을 확인할 수 있었으며, 즉 CLA가 RCA보다 연산속도가 빠르다는 것을 증명할 수 있다.

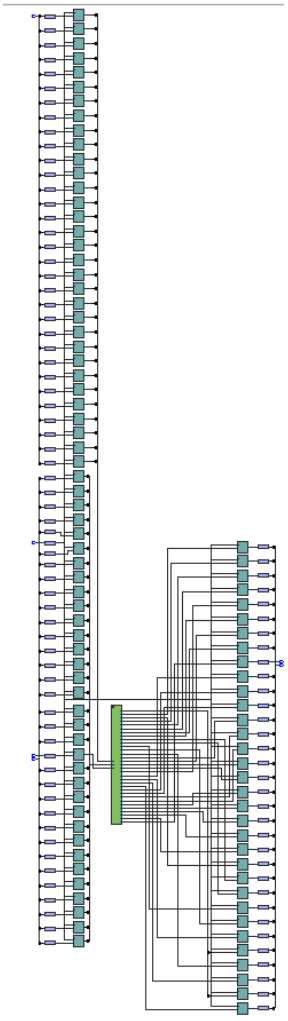
B. 합성(synthesis) 결과

CLA_CLK

- RTL viewer



- Technology viewer



- Flow Summary

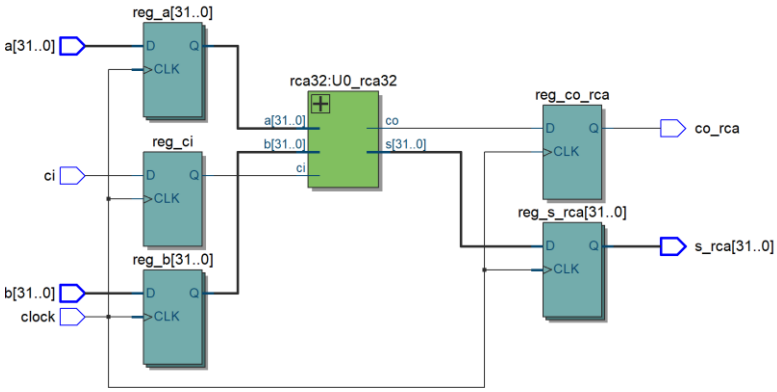
Compilation Report - cla_clk

Flow Summary	
Flow Status	Successful - Mon Sep 28 21:48:18 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 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)	40 / 41,910 (< 1 %)
Total registers	98
Total pins	99 / 499 (20 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

32-bit CLA의 경우 module 4bit CLA 8개를 불러와 구현했다. 또한 bit의 (4n, 4n +3)인 값에 CLA를 연결하여 직렬연결 해주었다. (RCA도 4-bit RCA 8개를 직렬 연결함.)

RCA_CLK

- RTL viewer

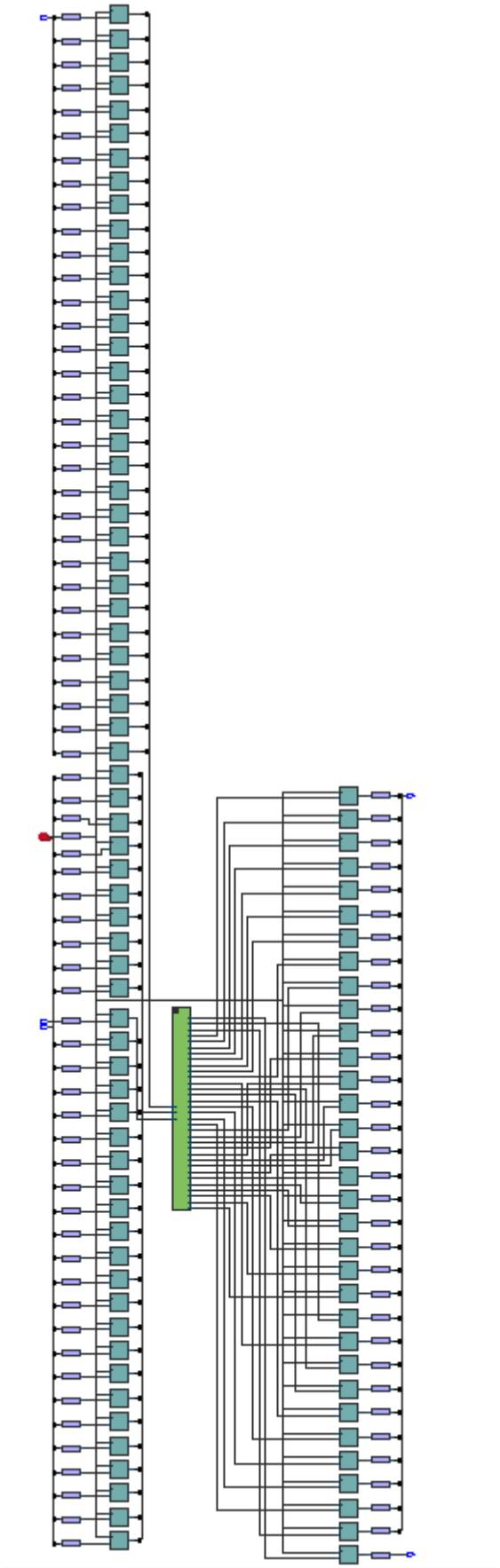


- flow summary

Compilation Report - rca_clk

Flow Summary	
Flow Status	Successful - Mon Sep 28 21:23:44 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 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)	37 / 41,910 (< 1 %)
Total registers	98
Total pins	99 / 499 (20 %)
Total virtual pins	0
Total block memory bits	0 / 5,662,720 (0 %)
Total DSP Blocks	0 / 112 (0 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total PLLs	0 / 15 (0 %)
Total DLLs	0 / 4 (0 %)

- Technology viewer



5. 고찰 및 결론

A. 고찰

- CLA의 경우 RCA보다 연산이 빠르다는 점을 확인했다. 그러나 계산식이 RCA보다 복잡해 다소 어려웠다.
- Timing 분석을 하는 중 매뉴얼을 봐도 어떤 방법으로 해야 하는지 감을 잡지 못해 몇 시간 동안 진전이 없었다. 후에 CLA(2/11)에서 빨간 박스만 누르면 되는 것을 일일이 똑같이 하겠다고 값을 친 게 화근이었다.
- 기본적인 모듈을 만들었고, 그게 하위 모듈로 다시 재사용되면서 점점 bit수가 큰 연산이 가능해짐을 느꼈다.

B. 결론

- RCA와 CLA를 비교했을 때 CLA의 연산속도가 32비트일 때 더 빠르다는 것을 확인했다. 그러나 device공간 차지를 더 적게 하는 것은 RCA다. (flow summary의 Logic utilization을 보면 알 수 있다.) 즉, 우리는 CLA와 RCA를 적절히 혼합하여 사용해야할 것이다. 속도가 빨라질수록 부품 값은 비싸지는 딜레마가 존재하기 때문이다. (시뮬레이션 결과 파트와 flow summary에서 모두 확인할 수 있다.)

CLA의 Logic utilization: 40

RCA의 Logic utilization: 37

CLA에서 모자랐던 delay: 1.464

RCA에서 모자랐던 delay: 5.208

6. 참고문헌

이준환/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2020

공영호/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2020