컴퓨터 공학 기초 실험2 보고서

실험제목: Carry Look-ahead Adder

실험일자: 2020년 09월 25일 (금)

제출일자: 2020년 09월 29일 (화)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 금요일 5,6,7

학 번: 2019202009

성 명: 서여지

1. 제목 및 목적

A. 제목

Carry Look-ahead Adder

B. 목적

Carry Look-ahead Adder의 동작원리를 이해한다. 4-bits CLA를 설계하고 이를 바탕으로 32-bit CLA를 제작한다. 레지스터를 이용한 모듈을 제작하고, Timing Analysis를 진행한다. 저번 주 실습 내용을 바탕으로 레지스터를 이용하는 32bit RCA를 제작하여 32bit CLA와 동작을 비교한다.

2. 원리(배경지식)

1) CLA

Carry look-ahead adder는 모든 자리의 carry값을 Carry look ahead block에서 먼저 계산하는 방법을 통해 ripple carry adder보다 빠르게 계산을 할 수 있게 한다. Ripple carry adder의 경우, 다음 bit의 carry in은 이전 bit의 계산 결과가 나와야 알 수 있기 때문에 마지막으로 계산되는 bit는 이전의 모든 bit의 계산이 끝난 뒤에 연산을 시작할 수 있다. 이러한 방법은 delay가 크다. 반면 CLA의 경우 clb에서 각 bit의 carry값을 구하는 과정은 각 자리의 bit가 순서대로 계산되지만, 한 번 carry를 구한 이후 더하기 연산을 하는 과정은 delay없이 진행할 수 있다.

CLA에서 carry를 구하는 방법은 다음과 같다. 우선 carry가 발생하는 경우는 두 가지이다. a,b,ci이 있을 때 (1) ci의 값과 관계 없이 a와 b가 모두 1인 경우와, (2) ci가 1일 때 a와 b중 하나 이상의 값이 1인 경우이다. (1)의 경우 and gate를 이용하여 Generate를 구해 확인하고, (2)의 경우에는 or gate를 이용하여 구한 Propagate신호에 각 자리의 ci을 and 연산하여 확인한다.

$$(1)\quad G_i = A_i B_i$$
$$(2)\quad P_i = A_i + B_i$$
$$C_{i+1} = G_i + P_i C_i = A_i B_i + (A_i + B_i) C_i$$

4bit CLA에서 이용되는 C_1, C_2, C_3, C_{out}을 구하면 다음과 같이 나타난다.

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

2) Flip flop

Flip flop은 synchronous logic circuit이고, sequential circuit에 이용되는 메모리이다. 하나의 flip flop에는 한 bit를 저장할 수 있다. Clock에 동기되어 있으므로 clock의 값이 0에서 1로 올라가는 rising edge에서 입력 값에 따라 출력이 바뀔 수 있고, 다른 경우에는 입력 값이 바뀌어도 출력이 변하지 않는다. 반면 flip flop과 유사하게 한 bit의 값을 저장할 수 있는 latch의 경우, clock입력이 없는 비동기 회로이다. latch는 Enable신호에 따라 출력을 바꾼다.

3) 레지스터

레지스터는 여러 개의 flip flop으로 구성되어 있다. 따라서 여러 bit를 저장할 수 있다. CPU내부의 레지스터는 데이터를 불러올 메모리의 정보, 연산에 필요한 데이터 등을 저장한다. Program counter(pc)와 Instruction resigter(ir)과 같이 명령어와 관련된 정보를 저장하는 레지스터도 존재한다.

4) Timing analysis

Timing analysis는 synchronous circuit이 정상적으로 동작하는 조건을 찾는 것을 목표로, 회로의 delay를 분석하는 것이다. Timing analysis를 통해 maximum clock frequency를 찾을 수 있다. Clock의 주기보다 data가 변하는 주기가 더 빠른 경우 clock의 rising edge이전에 다시 data값이 변할 수 있으므로 회로는 정상적으로 동작할 수 없다. 반면 Clock의 주파수가 너무 높은 경우에는 회로의 delay에 의해 정상적으로 결과가 출력되지 않는다. 따라서 clock의 주파수는 적절하게 설정되어야한다.

3. 설계 세부사항

1. 4-bits CLA

(1) clb

CLA에서 carry값을 계산하는 Carry Look ahead Block은 위에서 Generation과 Propagation을 구한 것을 이용하여 작성한다. clb는 각 bit마다 G와 P를 구하는 부분과, C를 구하는 부분으로 나눌 수 있다. G_i는 A_i와 B_i를 AND를 이용하여 구하고, P_i는 OR을 이용하여 구한다. C_{i+1}는 G_i OR $(P_i\ AND\ C_i)$로 작성한다. input port인 a, b, ci의 값을 이용하여 c1, c2, c3, co를 계산하여 출력한다.

(2) full adder

CLA에서 이용하는 full adder는 RCA에서와 달리 각 자리의 carry out이 필요하지 않다. 저번 실습에서 작성한 full adder의 carry out을 삭제하여 이용한다.

(3) CLA

4bit계산을 하는 cla는 4bit clb 하나와 4개의 full adder로 구성된다. 가장 먼저 각 자리의 carry값을 계산해야 하므로 clb4 module에 4bit인 a,b와 1 bit의 ci을 input으로 넣고, 각 자리의 carry값을 wire와 output인 co를 이용하여 얻는다. clb를 이용해 얻은 carry를 각각의 full adder module에 ci 입력으로 연결하고, 각 자리의 a,b, 값을 연결하여 s[i]를 얻는다.

2. 32-bits CLA, 32-bits RCA

32bit CLA는 4bit CLA를 8개 사용한다. 32bit의 input을 4bit씩 나누어 각각의 cla4 module에 입력 값으로 연결한다. 각 모듈 사이는 wire를 이용하여 co와 다음 모듈의 ci을 연결한다. 완성된 32bit CLA module은 a,b,ci의 값을 모두 더한 값을 s에, 올림수를 co에 반환해야 한다.

32-bits CLA를 만든 방법과 동일한 방법으로 4bit RCA를 8개 이용하여 rca32 module을 만든다.

3. 32-bits CLA with register, 32-bits RCA with register

32-bits CLA의 input, output에 총 5개의 레지스터를 추가하여 설계한다. sensitivity list에 posedge clock을 입력하여 clock이 0에서 1이될 때 always문이 실행되어 레지스터가 동작하도록 한다. cla32모듈을 instance하여 이용하고, s와 co의 값은 reg_s_cla와 reg_co_cla 레지스터의 값을 assign하여 얻는다.

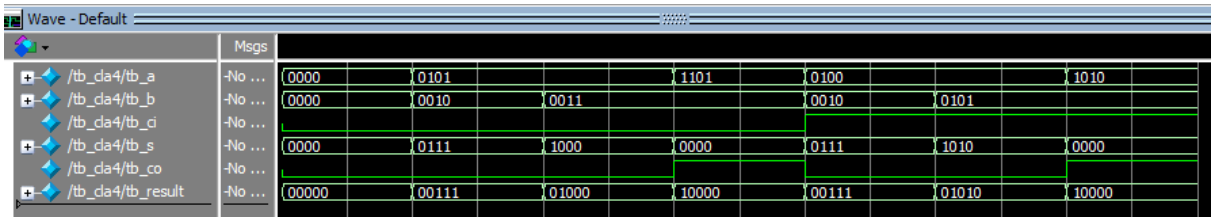
32-bits RCA with register는 위의 방법과 동일한 방법으로 32-bits RCA module에 레지스터를 추가하여 설계한다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

1. 4 bits CLA

testbench는 지난 4 bits RCA 실험에서 모듈을 실험했을 때 사용한 값을 이용하였다. 계산에서 carry가 발생하지 않는 경우, full adder사이에서만 carry가 발생하고 carry out은 없는 경우, carry out이 발생하는 3가지의 경우로 나누고, 각 경우에 대해 carry in이 있는 경우와 없는 경우를 실험했다.

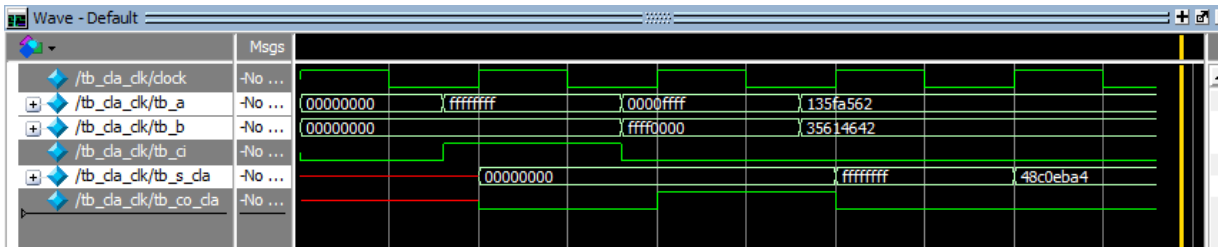


0	~10ns	~20ns	~30ns	~40ns	~50ns	~60ns	~70ns
tb_a	0000	0101	0101	1101	0100	0100	1010
tb_b	0000	0010	0011	0011	0010	0101	0101
tb_ci	0	0	0	0	1	1	1
tb_s	0000	0111	1000	0000	0111	1010	0000
tb_co	0	0	0	1	0	0	1
tb_result	00000	00111	01000	10000	00111	01010	10000

결과는 4bit RCA의 결과와 동일하게 나타났다. a,b,ci가 더해져 s에 출력되었고, 올림수는 다음자리에서 정상적으로 계산된 것을 확인할 수 있다.

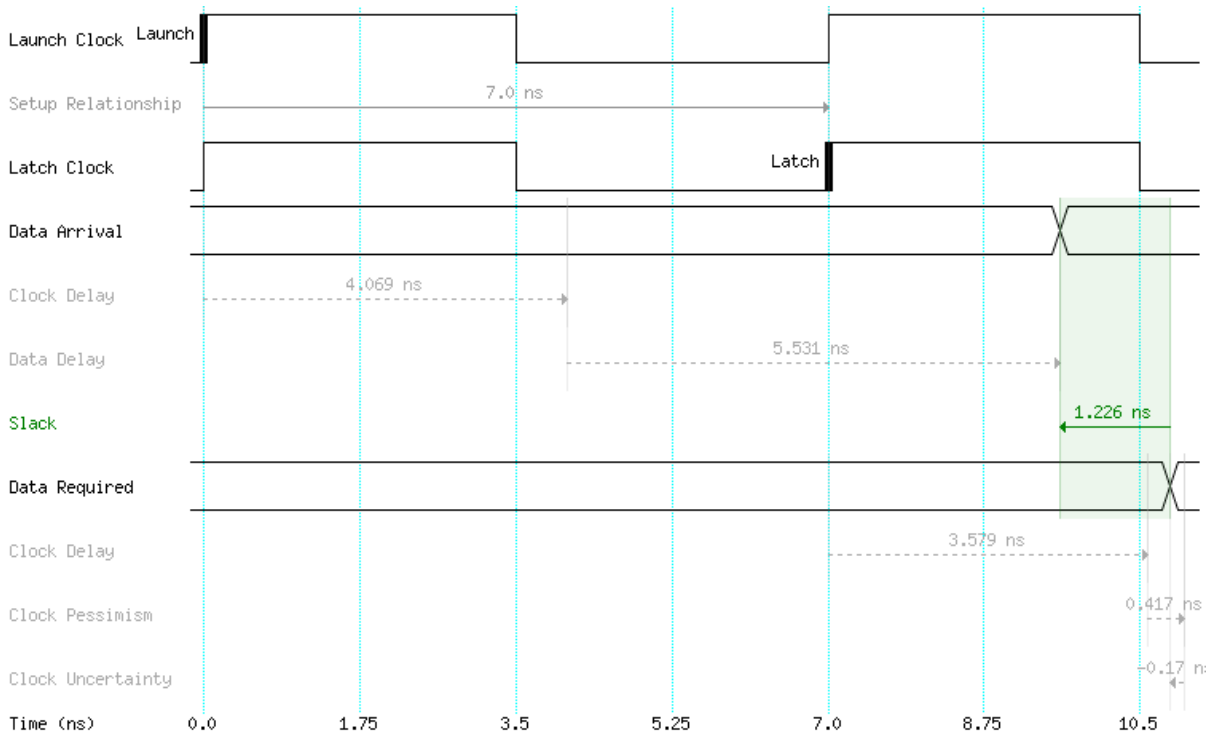
2. 32 bits CLA with register

testbench는 실습 강의자료에서 제시된 case로 진행하였다.



clock	0	1	2	3	4	5
tb_a	00000000	ffffffff	0000ffff	135fa562		
tb_b	00000000	00000000	ffff0000	35614642		
tb_ci	0	1	0	0		
tb_s_cla		00000000	00000000	ffffffff	48c0eba4	
tb_co_cla		0	1	0	0	

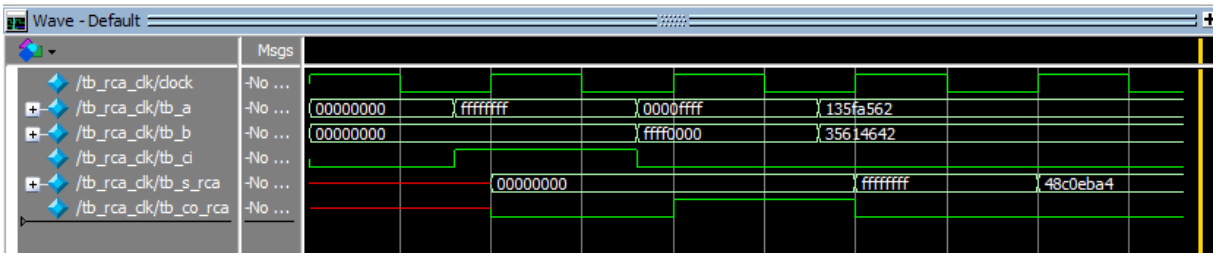
flip flop이 clock의 rising edge에서 동작하므로 clock의 주기 시작 전에 a,b,ci의 값을 변경하는 것을 볼 수 있다. 또한 a,b에 대한 덧셈연산의 결과가 다음 clock 주기에 표시된다. 따라서 0번째 주기의 출력은 의도한 값이 들어있지 않다.



clock이 rising edge일 때 clock delay가 지난 후, 나타나는 data delay에 출력이 계산되는 시간이 포함되어 나타난다. 32bit CLA에서 data delay는 5.531ns로 나타났고, Slack은 1.226ns이다.

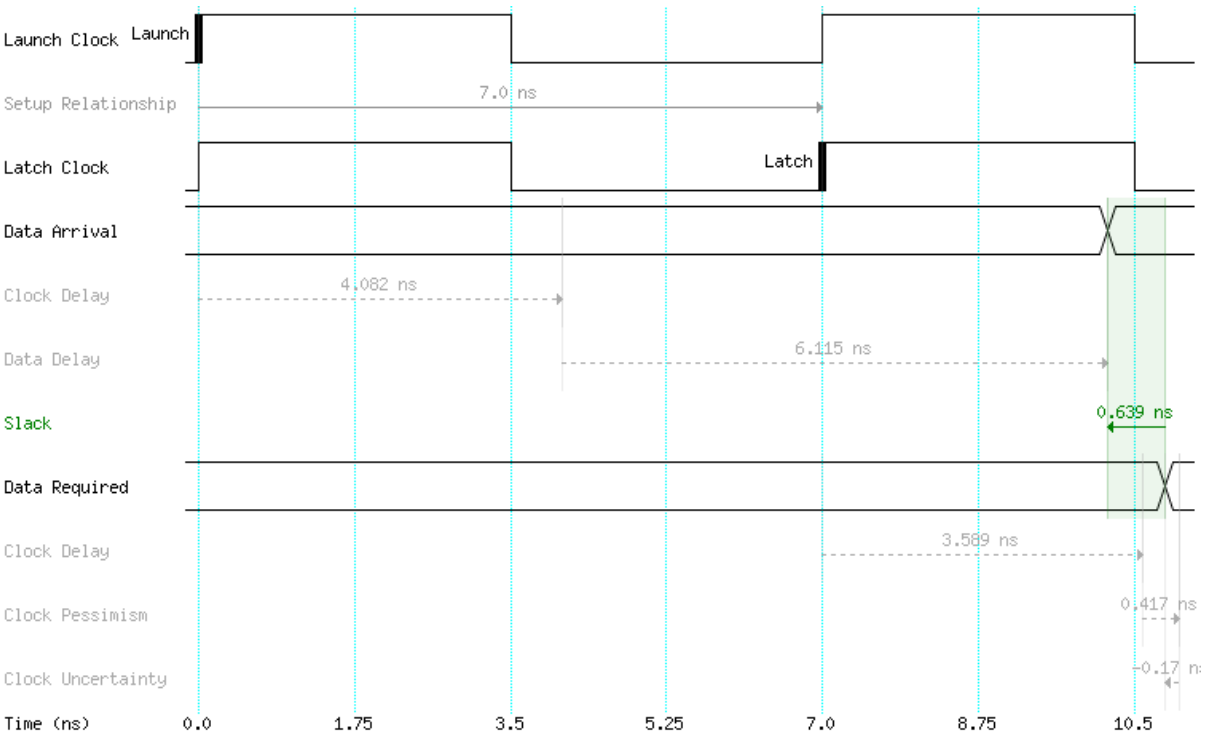
3. 32 bits RCA with register

32 bits RCA의 testbench는 위에서 실험한 32 bits CLA의 testcase와 같은 case를 이용하여 결과를 구하였다.



clock	0	1	2	3	4	5
tb_a	00000000	ffffffff	0000ffff	135fa562		
tb_b	00000000	00000000	ffff0000	35614642		
tb_ci	0	1	0	0		
tb_s_cla		00000000	00000000	ffffffff	48c0eba4	
tb_co_cla		0	1	0	0	

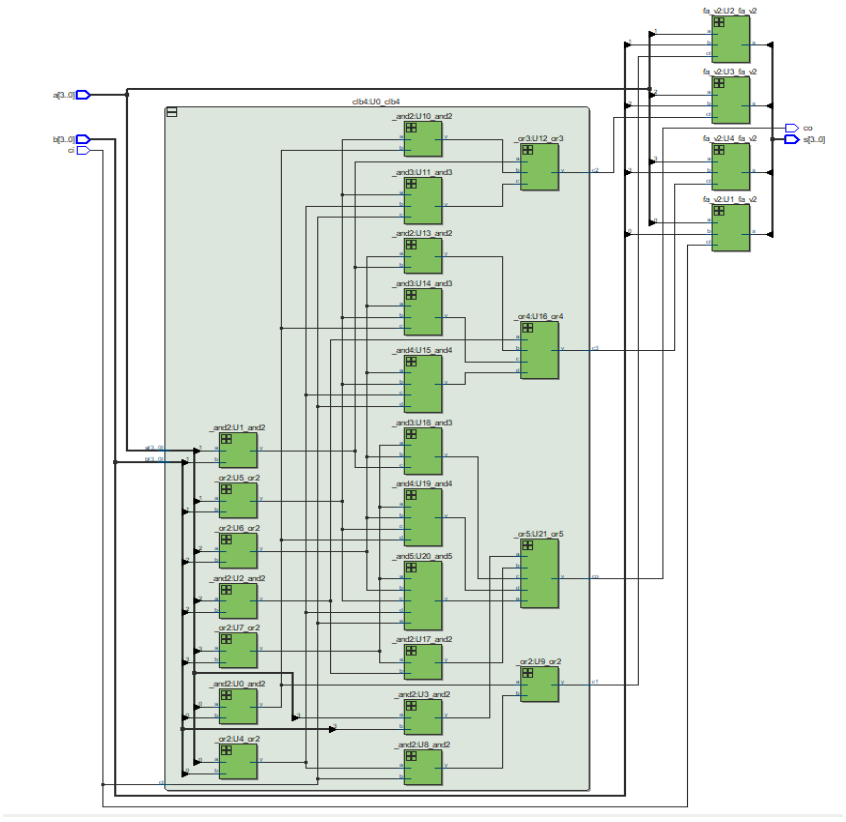
실험결과는 CLA의 결과와 동일하게 출력되었다.



32bit RCA에서 data delay는 6.115ns로 나타났고, Slack은 0.639ns이다.

B. 합성(synthesis) 결과.

1. 4bit CLA

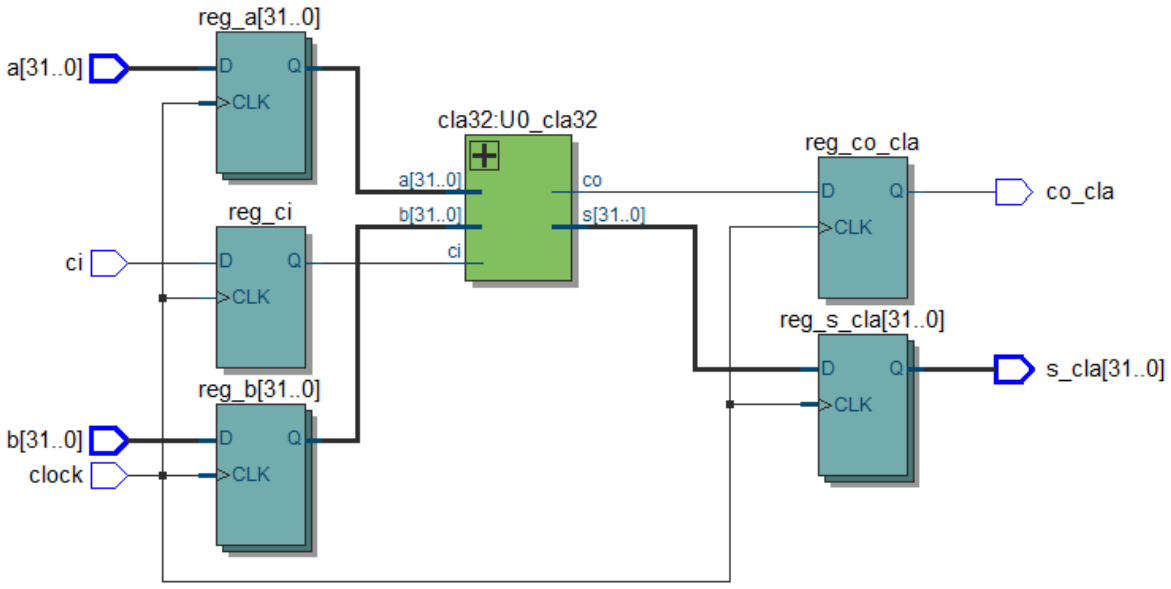


4bit CLA의 RTL viewer는 위와 같이 나타난다. clb module에서 각 자리의 carry를 구한 뒤, full adder에서 그 값을 이용해 계산하는 것을 확인할 수 있다.

Flow Summary	
Flow Status	Successful - Sun Sep 27 16:16:21 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	cla4
Top-level Entity Name	cla4
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	4 / 41,910 (< 1 %)
Total registers	0
Total pins	14 / 499 (3 %)
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 %)

사용한 register의 수는 없고, input a와 b가 각각 4개, ci가 하나의 핀을 사용하며 output s가 4개, co가 하나의 핀을 사용하여 총 14개의 핀이 사용되었다.

2. 32bit CLA

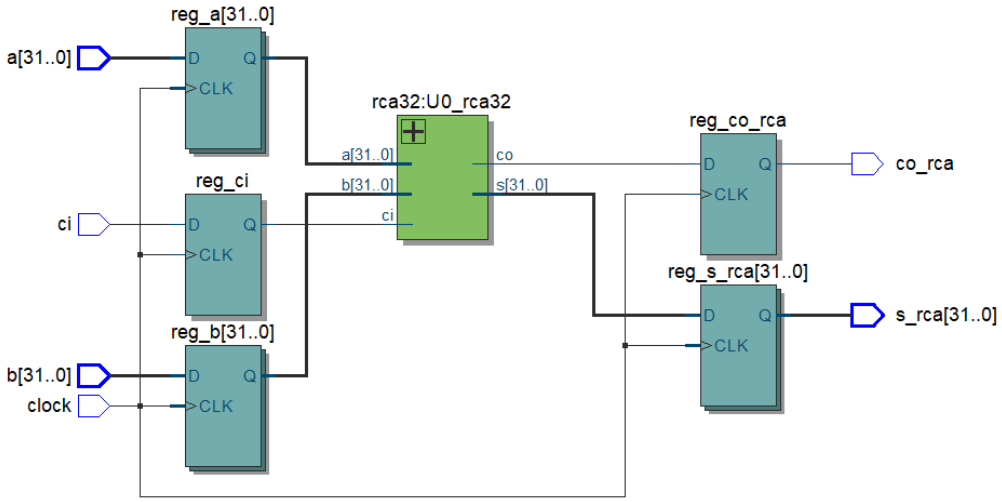


32bit cla에 5개의 register가 추가된 것을 확인할 수 있다. 모든 레지스터는 clock을 공통으로 사용하여 동기화 되어있다.

Flow Summary	
Flow Status	Successful - Sun Sep 27 16:05:48 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	cla4
Top-level Entity Name	cla_clk
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	43 / 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 %)

사용된 register의 수는 98개이다. input의 a,b가 각각 32개씩, ci가 하나의 레지스터를 이용하고, output에 연결되는 s가 32개, co가 하나의 레지스터를 이용한다. 사용된 핀의 개수는 register의 수보다 하나 많은 99개이다. register에 포함되지 않은 clock이 하나의 핀을 더 사용한다. Logic utilization은 43/41,910으로, 1%미만으로 나타났다.

3. 32bit RCA



32bit 계산을 하는 rca의 input과 output에 각각 register를 이용한 것을 확인할 수 있다. cla와 동일하게 공통적인 clock으로 동기화 되어있다.

Flow Summary	
Flow Status	Successful - Sun Sep 27 16:12:16 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	cla4
Top-level Entity Name	rca_clk
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	38 / 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 %)

사용된 register의 수와 사용된 핀의 개수는 CLA의 경우와 일치한다. Logic utilization은 38/41,910으로, cla보다 5작은 것을 확인할 수 있다.

5. 고찰 및 결론

A. 고찰

베릴로그 코드를 작성하는 과정에서 포트의 이름을 혼동하거나 오타가 발생하여 컴파일이 되지 않거나, 결과가 제대로 출력되지 않았었다. 동일한 module을 instance해서 사용하는 과정이 여러 번 반복되다 보니 실수가 많아졌던 것 같다. register를 이용한 module의 testbench를 작성할 때 &stop명령어를 붙이지 않아서 시뮬레이션이 계속 진행되기도 하고, clock의 주기를 잘못 설정하여 마지막 case의 결과가 잘리는 경우도 있었다.

B. 결론

계산과정에서 발생하는 carry의 값을 모두 계산한 뒤 a+b 계산을 하는 Carry Look-ahead Adder를 설계하는 실험이었다. 연산결과는 Ripple Carry Adder와 동일하게 출력되는 것을 확인할 수 있었다. 그러나 RCA의 긴 delay를 줄일 수 있다는 이점이 있다. 이것을 확인하기 위해 register를 이용하였으며, timing analysis 결과는 다음과 같다.

	CLA	RCA
data delay	5.531ns	6.115ns
Slack	1.226ns	0.639ns

CLA의 data delay가 RCA보다 짧았으며, 그 결과 CLA의 Slack이 RCA보다 약 1.9배 길게 나타났다. CLA와 RCA모두 사용한 register와 pin의 수는 같았지만, RCA의 Logic utilization이 더 작은 것으로 나타나 회로의 크기는 RCA가 더 작다는 것을 알 수 있다.

CLA에 register를 추가하면 CLA의 input과 output이 clock에 동기화되어 동작한다. 따라서 CLA의 출력이 입력 값이 변하는 순간과 관계없이 clock의 rising edge에 맞추어 변하게 된다. 이것은 input값이 흔들리는 등 불안정 할 때에도 output값을 안정적으로 나타낼 수 있다. 반면 clock의 주기가 과도하게 낮거나 높다면, 원하는 결과를 얻기 힘들 수 있다.

6. 참고문헌

이준환교수님, 디지털논리회로2 강의자료, 광운대학교 컴퓨터정보공학과,2020

이준환교수님, 컴퓨터공학기초실험2 강의자료, 광운대학교 컴퓨터정보공학과,2020

David Money Harris 외 1인, Digital Design and Computer Architecture, Elesvier