컴퓨터 공학 기초 실험2 보고서

실험제목: Carry Look-ahead Adder(CLA)

실험일자: 2019년 09월 03일 (화)

제출일자: 2019년 09월 09일 (월)

학 과: 컴퓨터정보공학과

담당교수: 이준환 교수님

실습분반: 화요일 0, 1, 2

학 번: 2015722031

성 명: 박 태 성

1. 제목 및 목적

A. 제목

Carry Look-ahead Adder(CLA)

B. 목적

이전 실험에서 구현한 Ripple Carry Adder(RCA)와 같은 기능을 수행하지만 구현 방식이 다른 CLA를 직접 설계, 구현하여 두 가산기의 성능 비교 직접 해보는 데 목적을 둔다.

2. 원리(배경지식)

A. RCA의 단점

RCA는 이전 full adder에서 carry가 출력되어야 다음 full adder의 연산이 진행된다. 따라서 큰 용량의 계산을 진행하면 속도가 떨어진다. 이를 보완하기 위해 CLA에서는 Carry Look-ahead Block(CLB)이 입력 carry in으로 carry들을 미리 예측하여 full adder에 값을 전달한다.

B. 4-bits CLB

CLB는 입력 carry in으로 carry들을 미리 예측한다. 4-bits CLB에 적용되는 boolean equation식은 다음과 같다. G는 두 비트의 연산시 carry의 생성(generate), P는 carry의 전달(propogate)을 의미한다.

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

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

$= G_1 + G_0P_1 + P_0P_1C_0$

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

$= G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$

$C_4 = A_3B_3 + (A_3 + B_3)C_3 = G_3 + P_3C_3$

$= G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3$

C. 4-bits CLA

4-bits CLA는 4-bits CLB와 full adder 4개로 구성된다. CLA의 Full Adder(FA)와 RCA의 FA에는 차이점이 있다. RCA의 FA는 carry out을 다음 full adder의 carry in으로 전달해주지만 CLA의 carry는 CLB에서 계산하기 때문에 full adder의 출력 carry out이 없다.

D. RCA와 CLA의 성능 비교

Carry를 계산하는 방식의 차이 때문에 일반적으로 RCA가 CLA보다 속도가 느리다. 일반적이라고 표현한 이유는 16비트 이하의 계산은 RCA가 더 빠르다. 속도 측정은 Timing Quest Timing Anlayzer를 통해서 수행하여 준다. Slack의 값이 음수라면 clock 조정을 통하여 양수 값이 출력되게 해준다. Fmax Summary를 통하여 속도 비교를 할 수 있다. 일반적으로 CLA가 RCA보다 area가 더 크다. CLA가 CLB를 구성하는데 많은 gates를 사용하기 때문이다.

E. Clock

Timing analyzing을 하기 위해 clock을 input으로 준다. 사용자가 설정한 시간 단위마다 1과 0을 번갈아 output 값으로 반환한다.

3. 설계 세부사항

A. 4-bits CLA

i. Functional Description

RCA가 시간이 많이 걸리는 단점을 보완하기 위해 모든 carry를 동시해 구하여 계산 시간을 단축한 가산기이다. Carry만을 계산하는 CLB가 따로 존재한다.

ii. I/O Description

Port	Name	Bandwidth	Description
Input	a	4 bit	Input data A
	B	4 bit	Input data B
	ci	1 bit	Carry in
Output	co	1 bit	Carry out
	s	4 bit	Sum
Wire	c1	1 bit	Internal carry
	c2	1 bit	Internal carry
	c3	1 bit	Internal carry

iii. Module Description

Classification	Name	Description
Module	cla4	4-bits CLA
Instance	U0_fa_v2	Full adder
	U1_fa_v2	
	U2_fa_v2	
	U3_fa_v2	

	U4_clb4	Carry generation

B. 32-bits CLA

i. Functional Description

32-bits CLA는 4-bits CLA 8개를 직렬로 연결하여 구현한다.

ii. I/O Description

Port	Name	Bandwidth	Description
Input	a	32 bit	Input data A
	B	32 bit	Input data B
	ci	1 bit	Carry in
Output	co	1 bit	Carry out
	s	32 bit	Sum
Wire	c	8 bit	Internal carry

iii. Module Description

Classification	Name	Description
Module	cla32	32-bits CLA
Instance	U0_cla4	4-bits CLA
	U1_cla4	
	U2_cla4	
	U3_cla4	
	U4_cla4	
	U5_cla4	
	U6_cla4	
	U7_cla4	

C. 32-bits CLA with clock

i. Timing Analysis

CLA가 제대로 동작할 수 있는 조건을 찾기 위하여 maximum clock frequency를 분석한다. 32-bits CLA 앞, 뒤에 flip-flop을 추가하며, 해당 flip-flop에 clock을 연결하여, 32-bits CLA의 유효한 결과가 나오는데 필요한 maximum clock frequency를 찾는다.

ii. I/O Description

Port	Name	Bandwidth	Description
Input	clock	1 bit	clock
	a	32 bit	Input data A

	b	32 bit	Input data B
	ci	1 bit	Carry in
Output	co_cla	1 bit	Carry out
	s_cla	32 bit	Sum
Register	reg_a	32 bit	Register A
	reg_b	32 bit	Register B
	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

iii. Module Description

Classification	Name	Description
Top module	cla_clk	32-bits CLA with clock
Instance	U0_cla32	32-bits CLA

32-bits RCA와 성능 비교를 위해 RCA도 clock을 설정하였다.

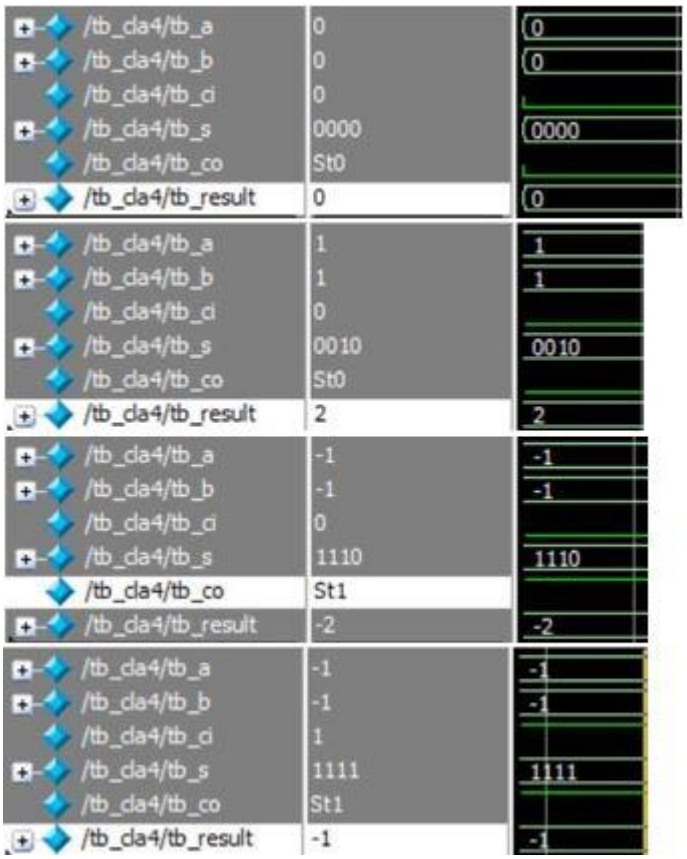
4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

i. 4-bits CLA

```
//4-bits CLA Testbench
`timescale 1ns/100ps

module tb_cla4;
    reg[3:0] tb_a, tb_b;
    reg tb_ci;
    wire[3:0] tb_s;
    wire tb_co;

    wire[4:0] tb_result;

    assign tb_result = {tb_co, tb_s};

    cla4 U0_cla4(.a(tb_a), .b(tb_b), .ci(tb_ci), .s(tb_s), .co(tb_co));

    initial
    begin
    tb_a = 4'b0; tb_b = 4'b0; tb_ci = 0; // tb_s = 4'b0000, tb_co = 0
    #10; tb_a = 4'b0001; tb_b = 4'b0001; // tb_s = 4'b0010, tb_co = 0
    #10; tb_a = 4'b0010; tb_b = 4'b0011; // tb_s = 4'b0101, tb_co = 0
    #10; tb_a = 4'b0111; tb_b = 4'b0111; // tb_s = 4'b1110, tb_co = 0
    #10; tb_a = 4'b1111; tb_b = 4'b1111; // tb_s = 4'b1110, tb_co = 1
    #10; tb_ci = 1; // tb_s = 4'b1111, tb_co = 1
    #10; $stop;
    end
endmodule
```

4-bits 2 inputs와 1-bit 1 input의 모든 경우의 수를 테스트하기는 많다. 따라서, exhaustive verification은 적합하지 않다. 대신, directed verification을 통하여 검증하였다. 4-bits CLA는 RCA와 같은 기능을 수행하는 가산기이기 때문에 RCA의 testbench와 동일하게 만들어 같은 결과가 출력되는지 보았다.



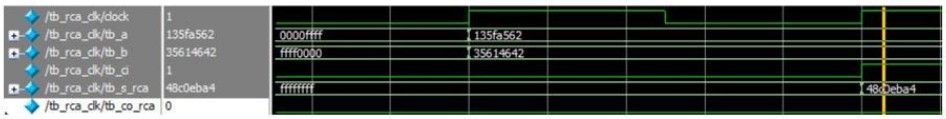
4-bits RCA와 같은 결과(Waveform)가 나옴을 확인하였다.

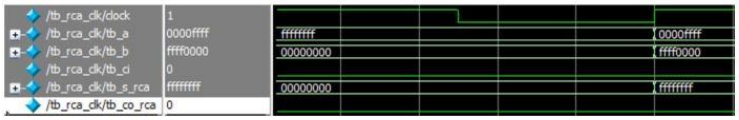
ii. RCA32 with Register

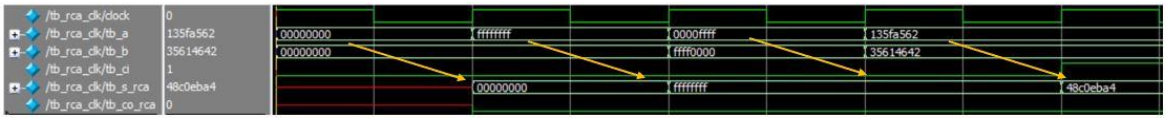
```
//Testbench of 32-bits RCA with Register
`timescale 1ns/100ps

module tb_rca_clk;
   reg clock;
   reg[31:0] tb_a, tb_b;
   reg tb_ci;
   wire[31:0] tb_s_rca;
   wire tb_co_rca;

   parameter STEP = 10;

   rca_clk U0_rca_clk(.clock(clock), .a(tb_a), .b(tb_b), .ci(tb_ci), .s_rca(tb_s_rca), .co_rca(tb_co_rca));

   always # (STEP/2) clock = ~clock;

   initial
   begin
   clock = 1; tb_a = 32'h00000000; tb_b = 32'h00000000; tb_ci = 0;
         #10; tb_a = 32'hFFFFFFFF;
         #10; tb_a = 32'h0000FFFF; tb_b = 32'hFFFF0000;
         #10; tb_a = 32'h135FA562; tb_b = 32'h35614642;
         #10; tb_ci = 1;
         #10; $stop;
   end
endmodule
```

마찬가지로 모든 경우의 수를 검증하기는 너무 많아 exhaustive verification은 적합하지 않다. 32-bits hexadecimal을 입력으로 넣어주었다. Hexadecimal 1 digit은 4-bits를 의미

한다. Clock은 5ns을 주기로 1과 0을 반복 출력한다.





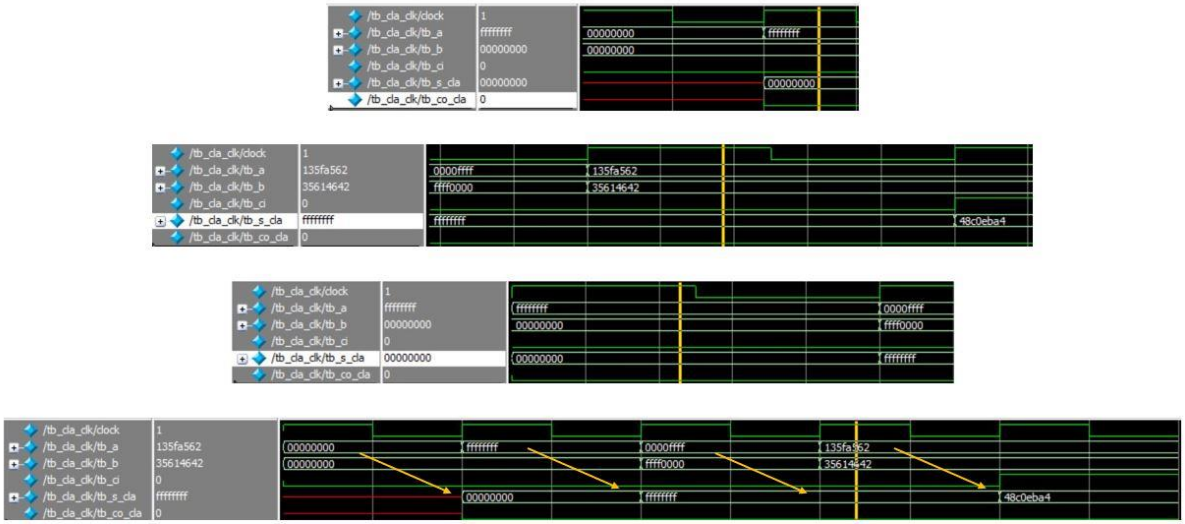


Waveform을 보면 즉시 결과값이 출력되지 않고 1 cycle후에 출력된다. 일반적으로 delay를 적용하면 즉시 결과값을 출력되는 것은 natural하지 않다. 가산기의 역할은 clock이 1이 될 때마다 정상적으로 수행함을 결과값을 통해 확인하였다.

iii. CLA32 with Register

```
//Testbench of 32-bits CLA with Register
`timescale 1ns/100ps

module tb_cla_clk;
   reg clock;
   reg[31:0] tb_a, tb_b;
   reg tb_ci;
   wire[31:0] tb_s_cla;
   wire tb_co_cla;

   parameter STEP = 10;

   cla_clk U0_cla_clk(.clock(clock), .a(tb_a), .b(tb_b), .ci(tb_ci), .s_cla(tb_s_cla), .co_cla(tb_co_cla));

   always # (STEP/2) clock = ~clock;

   initial
   begin
   clock = 1; tb_a = 32'h00000000; tb_b = 32'h00000000; tb_ci = 0;
         #10; tb_a = 32'hFFFFFFFF;
         #10; tb_a = 32'h0000FFFF; tb_b = 32'hFFFF0000;
         #10; tb_a = 32'h135FA562; tb_b = 32'h35614642;
         #10; tb_ci = 1;
         #10; $stop;
   end
endmodule
```

RCA와 같은 동작을 하는지 쉽게 비교하기 위해 RCA testbench의 instanciation 부분만 수정하였다. 같은 결과가 나와야 정상 작동하는 것이다.

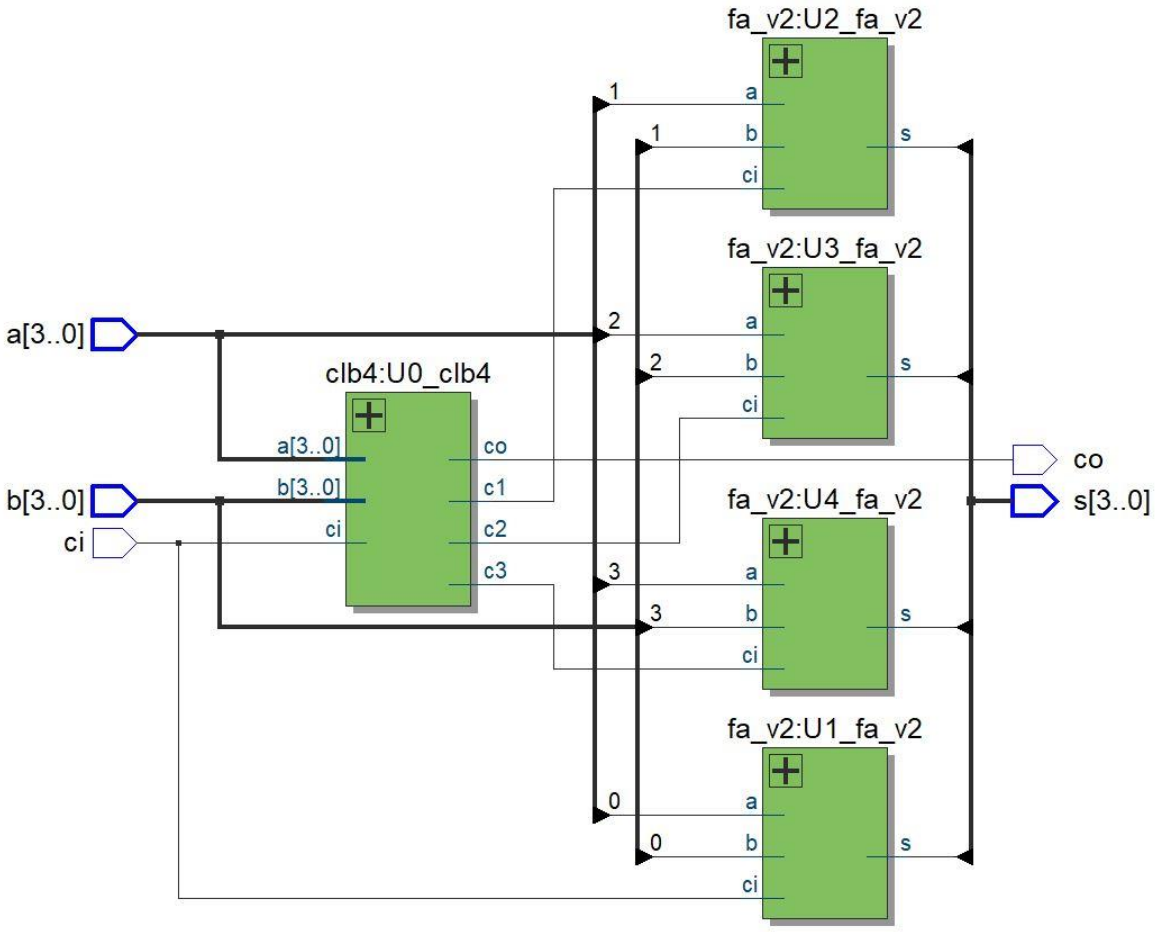


32-bits RCA with register와 같은 결과(waveform)가 출력됨을 확인하였다.

B. 합성(synthesis) 결과

i. 4-bits CLA

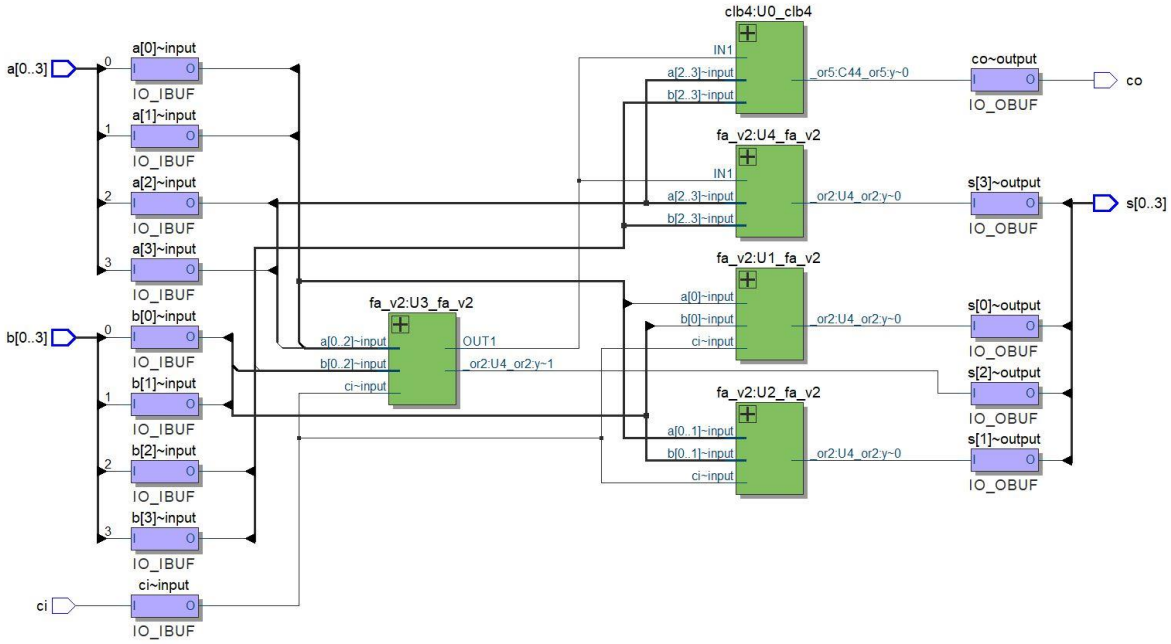
1. RTL map viewer



CLB4 1개와 fa_v2 3개가 instianciation 되었다. Full adder는 carry를 CLB로부터 input으로

받아 s만 출력한다.

2. Technology map viewer



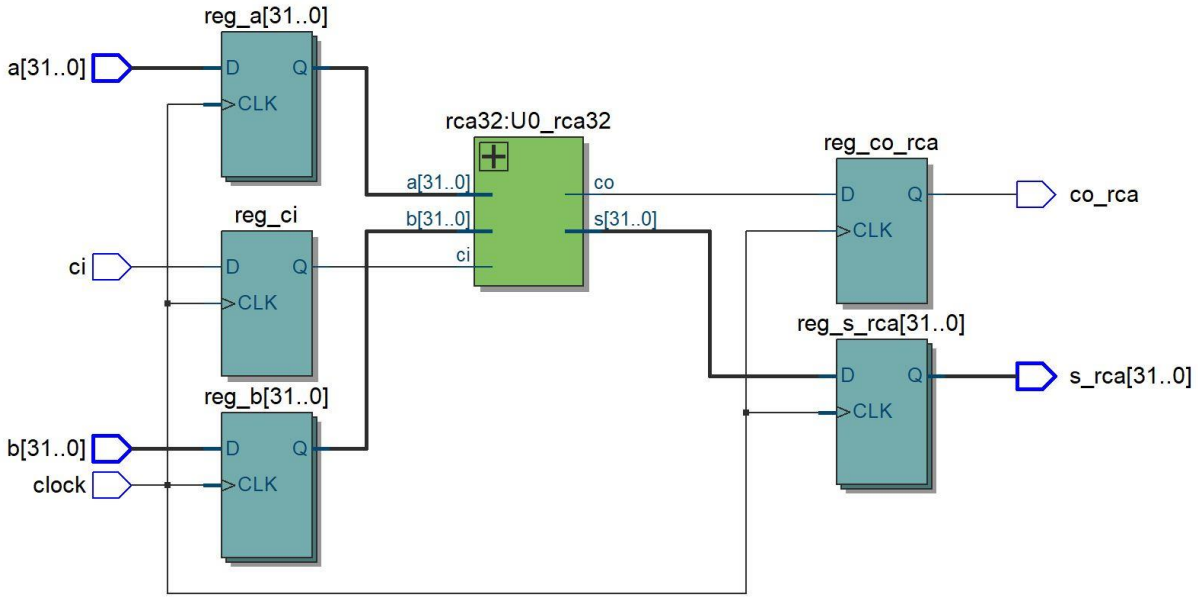
3. Flow Summary

Flow Summary	
Flow Status	Successful - Fri Sep 06 08:54:46 2019
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	seven_segment_cla
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

Total pins는 14개 임을 확인하였다. Input과 output port의 개수 합과 일치한다.

ii. RCA32 with Register

1. RTL map viewer



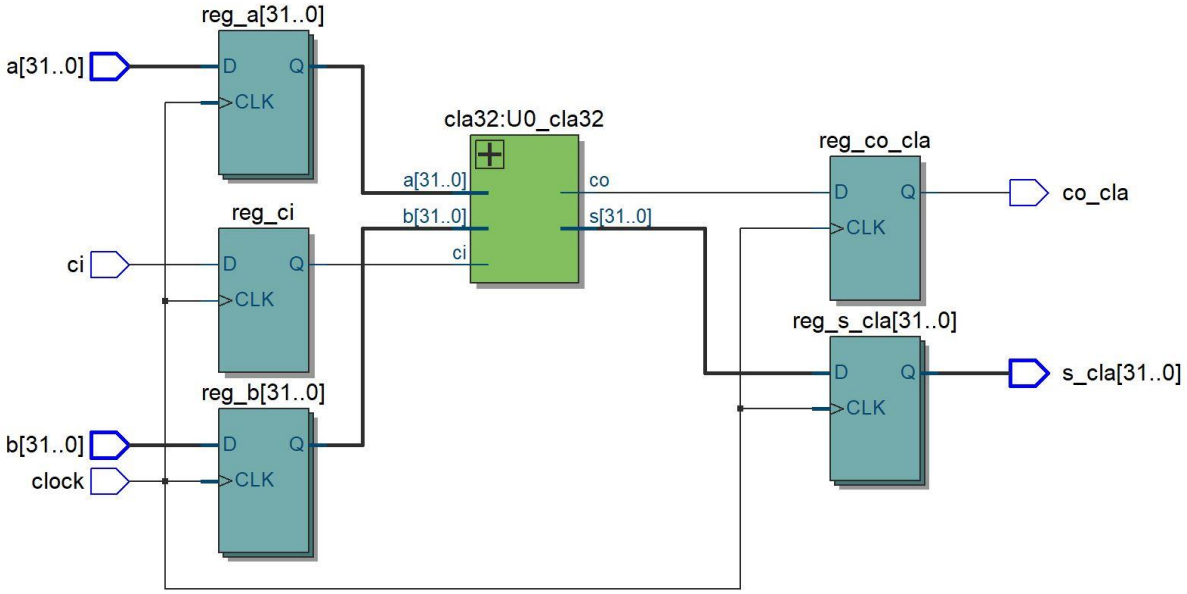
2. Flow Summary

Flow Summary	
Flow Status	Successful - Sat Sep 07 18:59:05 2019
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	seven_segment_cla
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 %)

Total pins는 input, output ports와 개수가 일치함을 확인하였다. Register이 추가로 total registers의 개수는 98개로 표시된다. 회로를 구성하는데 사용된 logic은 28개다.

iii. CLA32 with Register

1. RTL map viewer



2. Flow Summary

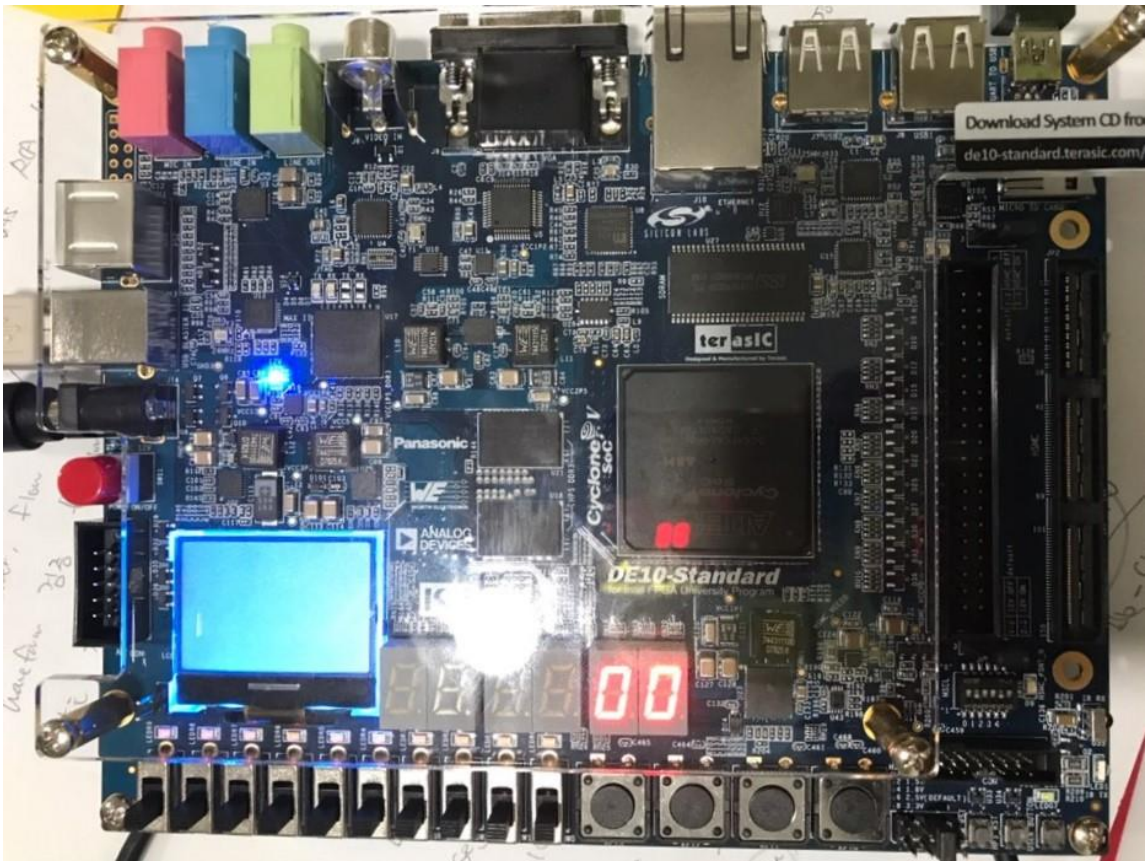
Flow Summary	
Flow Status	Successful - Sat Sep 07 18:54:46 2019
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	seven_segment_cla
Top-level Entity Name	cla_clk
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	43 / 41,910 (< 1 %)
Total 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 %)

마찬가지로 total pins는 input, output ports와 개수가 일치함을 확인하였다. Register이 추가로 total registers의 개수는 98개로 표시된다. 회로를 구성한 logic은 43개이다.

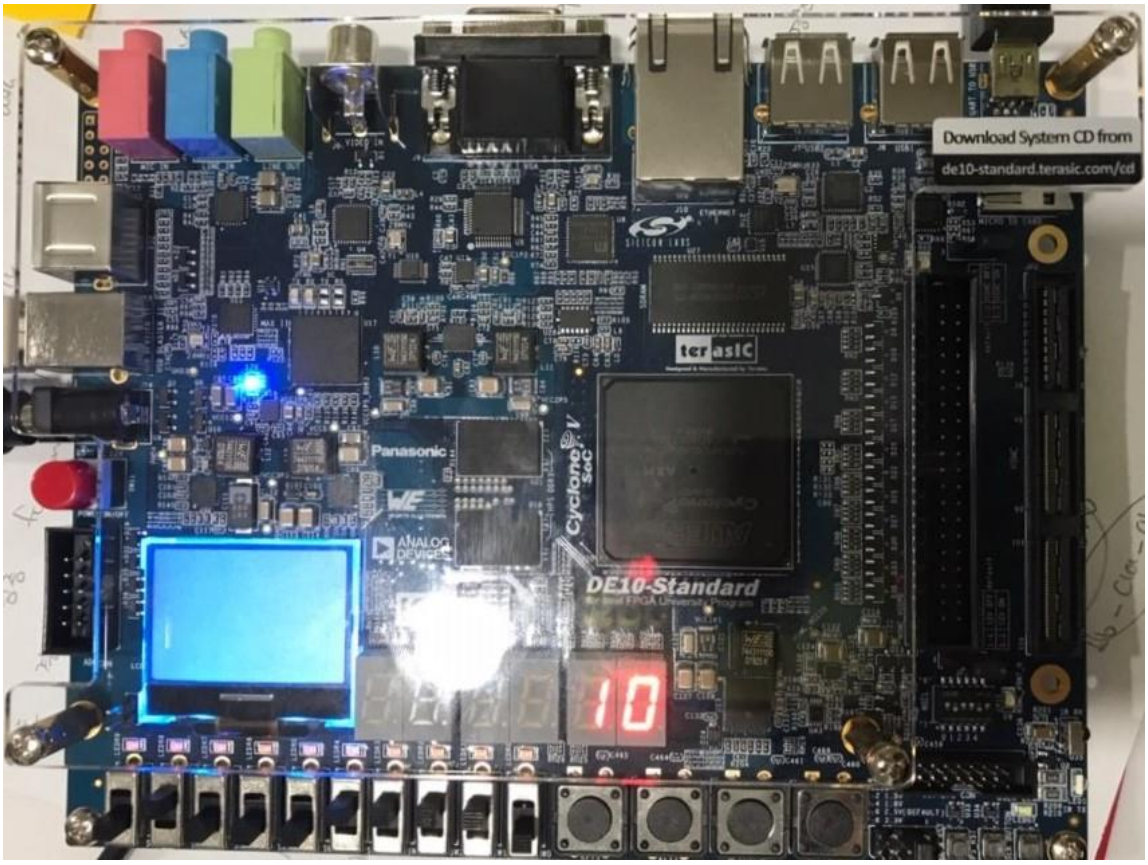
C. FPGA board targeting 결과

i. 초기 상태

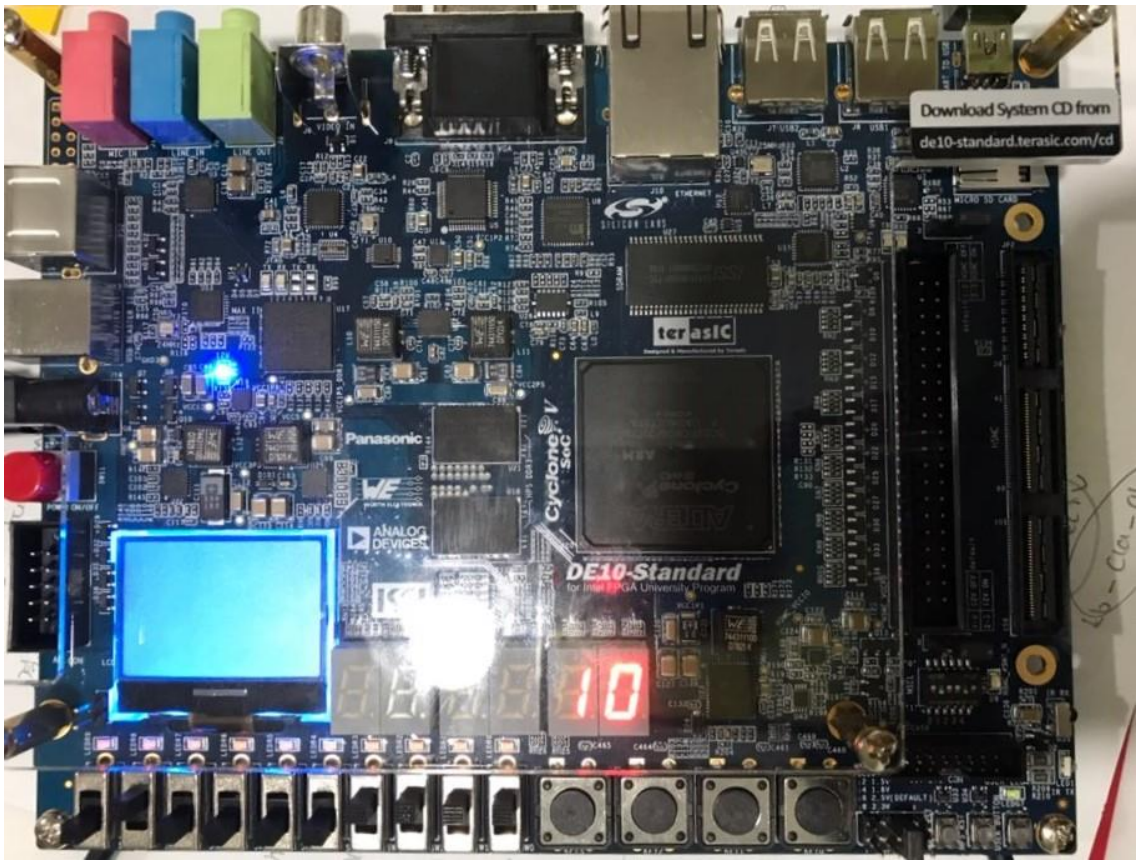
a = $0000_{2,}$ b = $0000_{2,}$ ci = 0_2



ii. a = $1110_{2,}$ b = $0001_{2,}$ ci = 1_2



iii. $a = 1111_{2,}\ b = 0000_{2,}\ ci = 1_2$



결과는 hexadecimal로 출력되기 때문에 알파벳 a~f까지 보드에 출력된다.

5. 고찰 및 결론

A. 고찰

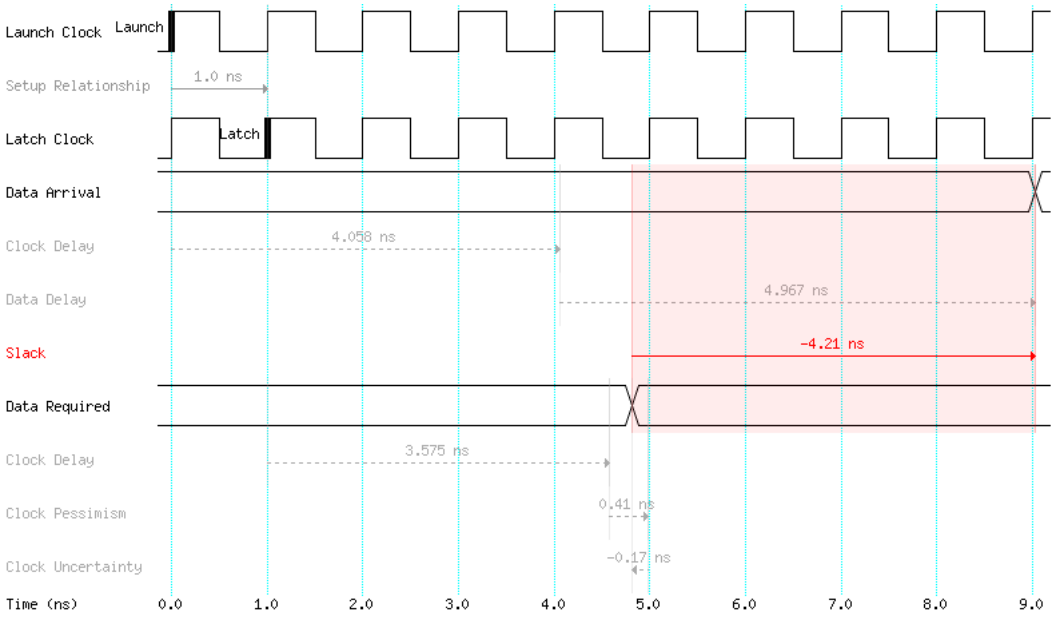
Board에 코드를 올렸을 때 seven segment에 00이 출력되지 않았다. 원인은 다양하다. Pin planner에 오류가 있을 수도 있고 seven_seg 코드나 CLB, CLA와 같은 베릴로그 파일 코드가 잘못 설계 됐을 수도 있다. Pin planner와 코드를 모두 살펴보았는데 오류를 찾지 못했다. 보드를 바꾸어 보니 값이 정상 출력되었다.

B. 결론

i. Timing Analysis of CLA32

Slow 1100mV 85C Model			
	Clock	Slack	End Point TNS
1	clock	-4.210	-94.010

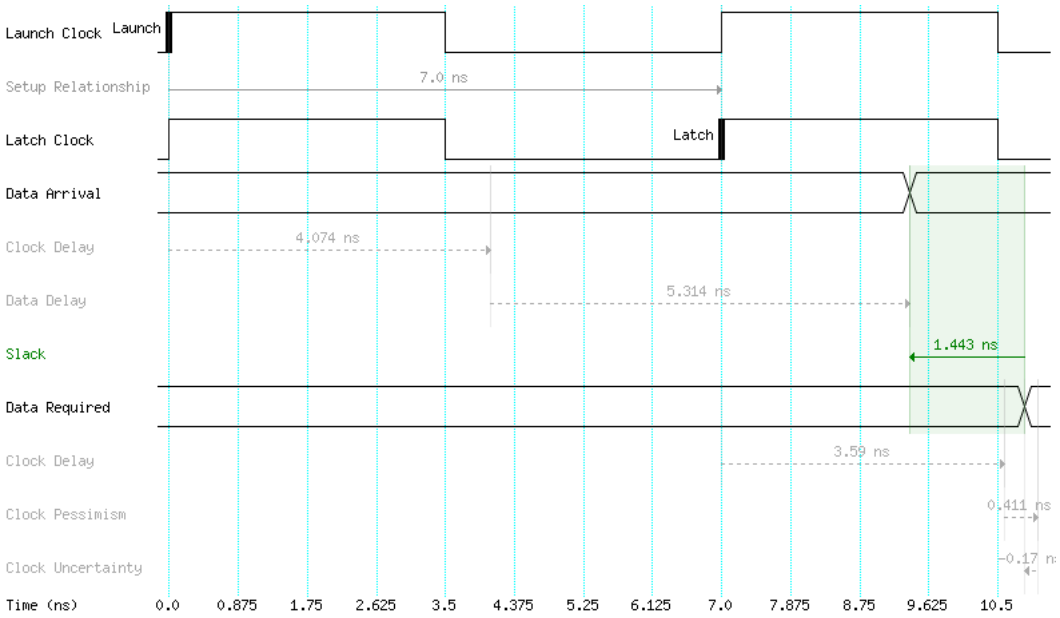
Slack은 신호가 flip flop으로 가는데 걸리는 시간을 의미한다. 회로가 연산을 수행하기 전에 clock 신호가 도착하면 slack이 음수가 된다.



Data arrival은 clock delay와 data delay의 합이다. Clock delay는 RCA와 CLA가 같기 때문에 data delay가 시간 차이를 발생한다. Data dealy를 보고 clock을 조절하면 된다. Slack의 절대값에 1을 더한 값으로 설정해도 된다.

Slow 1100mV 85C Model			
	Clock	Slack	End Point TNS
1	clock	1.443	0.000

clock을 7로 설정하여 slack을 양수로 만들었다.



Data arrival이 끝난 후에 data requried하는 것을 확인하였다.

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	191.94 MHz	191.94 MHz	clock	

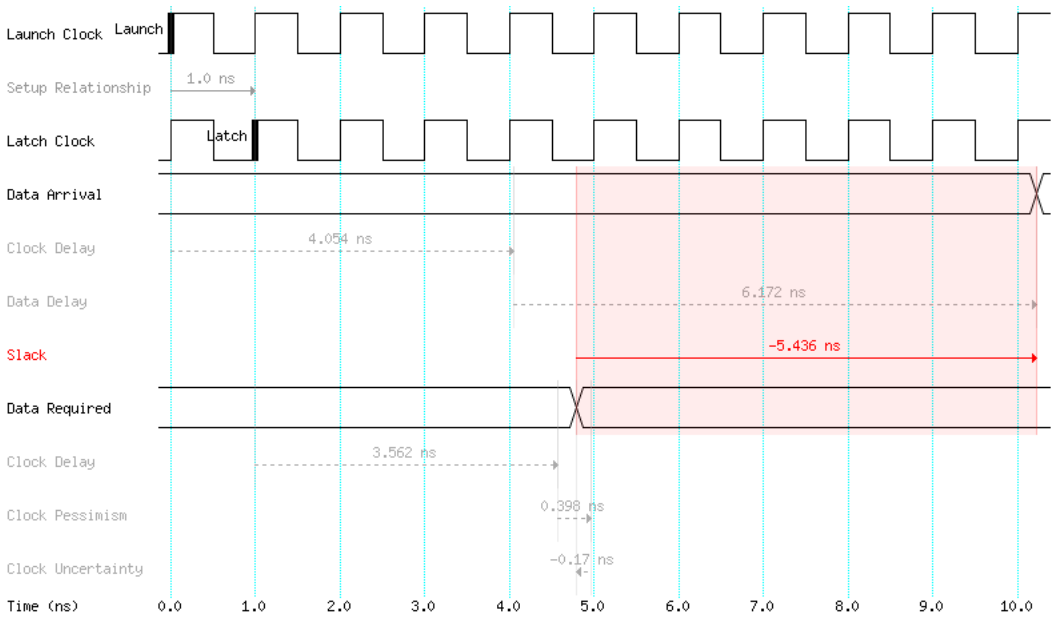
최대 클럭 주파수(Fmax)는 시간과 반비례 관계이다. 따라서 RCA와 CLA 중 Fmax의 값이 큰 가산기가 속도가 더 빠르다.

ii. Timing Analysis of RCA32

Slow 1100mV 85C Model

	Clock	Slack	End Point TNS
1	clock	-5.436	-111.732

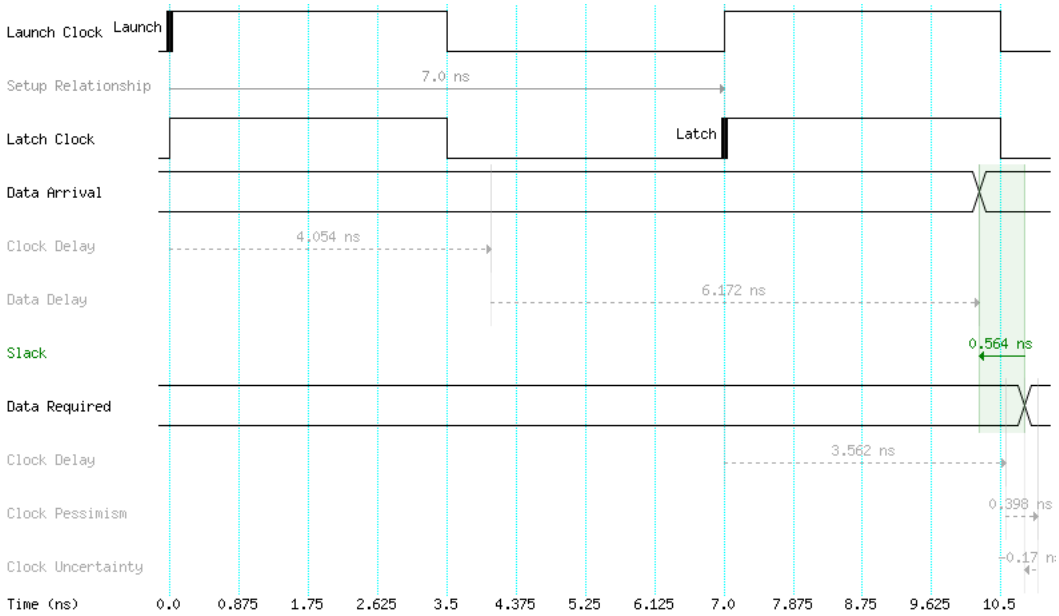
Clock 조정이 필요하다. Slack 값은 RCA가 더 크다. 차이가 크지는 않다.



CLA와 마찬가지로 clock 주기를 7로 설정하였다.

Slow 1100mV 85C Model

	Clock	Slack	End Point TNS
1	clock	0.564	0.000



Data arrival 전에 data requried가 발생하지 않는다.

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	155.38 MHz	155.38 MHz	clock	

32-bits CLA의 Fmax는 191MHz, 32-bits RCA의 Fmax는 155MHZ이다. 속도는 CLA가 더 빠르다. RCA를 구성하는 logic은 28개, CLA를 구성하는 logic은 43개로 크기는 CLA가 더

크다.

6. 참고문헌

David Money Harris & Sarah L. Harris / Digital Design and Computer Architecture / Elsevier Korea L.L.C / November 2013