컴퓨터 공학 기초 실험2 보고서

실험제목: Carry Look-ahead Adder

실험일자: 2021년 09월 27일 (월)

제출일자: 2021년 10월 2일 (토)

학 과: 컴퓨터정보공학부

담당교수: 공진흥 교수님

실습분반: 월요일 0, 1, 2

학 번: 2018202046

성 명: 이준휘

1. 제목 및 목적

A. 제목

Carry Look-ahead Adder

B. 목적

해당 수업을 통해 Carry Look-ahead Adder의 동작 방식을 이해한다. 이를 바탕으로 CLA를 제작할 수 있는 능력을 기른다. RCA와 CLA의 속도 차이를 비교해보며 어떠한 차이가 있는지 알 수 있다.

2. 원리(배경지식)

i. Carry Look-ahead Adder

CLA(Carry Look-ahead Adder)란 일반적인 Adder 블록에서 마지막 위치의 co을 따로 계산하여 산출한다. 이를 통해 기존 RCA(Ripple Carry Adder)의 단점이었던 시간을 개선하였다.

Carry를 별도로 계산하는 과정은 다음과 같다.
우선 각 자리에서 나타나는 연산과정을 확인한다. 기존 A와 B가 둘 다 1일 경우 Carry는 무조건 발생한다. 이를 우리는 $G = A * B$로 둔다. 그리고 A 또는 B가 적어도 1이 하나라도 있어야 Cin이 있을 경우 Carry가 발생할 수 있다. 이를 우리는 $P = A + B$로 둔다. 이 값들을 통해 Carry를 확인하면 $Cout = G + P * Cin$으로 정의할 수 있다. 이는 다음 덧셈의 Cin임으로 우리는 이 식을 확장시켜 4bit 블록으로 만들 경우 $Cout = G3 + P3 * (G2 + P2 * (G1 + P1 * (G0 + P0 * Cin)))$ 와 같다. 또한 Cin을 괄호 밖으로 빼기 위하여 정리하면 우리는 $Cout = G3 + P3 * \big(G2 + P2 * (G1 + P1 * G0)\big) + P3 * P2 * P1 * P0 * Cin$과 같은 식으로 정리할 수 있다.

CLA는 두 가지 방법을 통해 구현할 수 있다.

첫 번째로 해당 블록 내의 모든 C를 계산해주는 회로를 구현하는 것이다.
우선 각각의 Full Adder에서는 S만 계산하고 Cout는 계산하지 않도록 한다. 그 후 Carry Look-ahead Blook을 만든다. 이 블록은 각 자리의 Carry를 위에서 설명하였던 식을 토대로 구성하여 Full Adder에 쓰일 Cin값을 만들어준다. 그 후 해당 회로를 Full Adder와 와이어로 연결하면 회로가 완성된다.

두 번째 방법으로는 기존의 RCA를 활용하되 해당 블록의 Cout만을 Carry Look-ahead 블록을 통해 계산하는 것이다.

우선 RCA와 같이 Full Adder를 통해 회로를 구성한다. 하지만 여기서 마지막 가산기의 Carry는 Cout에 연결하지 않는다. 그리고 위에서 정리하였던 식인 $G3 + P3 * \big(G2 + P2 * (G1 + P1 * G0)\big) + P3 * P2 * P1 * P0 * Cin$ 를 통해 Cout을 구한다. 그 후 이를 연결하면 Modified 4-Bit CLA Block을 구성할 수 있다.

3. 설계 세부사항

i. fa_v2

해당 모듈은 저번 주차에서 산출할 값을 이용하면 $s = a \oplus b \oplus c$로 표현할 수 있다.

wire w0를 output으로 받고, a, b를 입력으로 받는 _XOR2 모듈과 w0, ci를 입력으로 받고 s를 output으로 받는 _XOR2 모듈을 연결하여 해당 모듈을 구성한다.

ii. clb4

해당 모듈은 c1~c3와 co를 output으로 받고 a[3:0], b[3:0]과 ci을 input으로 받는 회로다.

해당 회로에서는 우선 wire [3:0] g, p를 만들고 각각의 와이어에 위의 설명에서 나온 값을 토대로 _and2 모듈과 _or2 모듈을 통해 값을 할당한다.

그 후 아래의 식에 맞도록 모듈들을 구성한다.

c1 = g[0] | (p[0] & ci)

c2 = g[1] | (p[1] & g[0]) | (p[1] & p[0] & ci)

c3 = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & ci)

co = 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] & ci)

iii. cla4

해당 모듈은 fa_v2와 clb4 모듈을 통해 구성하며, a[3:0], b[3:0], ci를 입력으로 가지고 s[3:0]과 co을 출력으로 가진다.

각 자리의 fa_v2 모듈에 해당 자리에 입력 a, b, wire(첫 자리일 경우 ci)을 가지고 출력 s를 가지도록 구성한다.

그 후 clb4 모듈에는 각 자리의 wire와 co를 output으로 연결하고, a, b, ci를 입력으로 가지도록 구성한다.

iv. cla32 & rca32

해당 모듈은 8개의 cla4 모듈 또는 rca4 모듈을 사용하여 구성한다.

각 자리에 4bit씩 끊어서 a, b, 그리고 이전 자리의 Carry를 넣고 출력으로 다음 자리의 Carry와 해당 자리의 c를 구성한다.

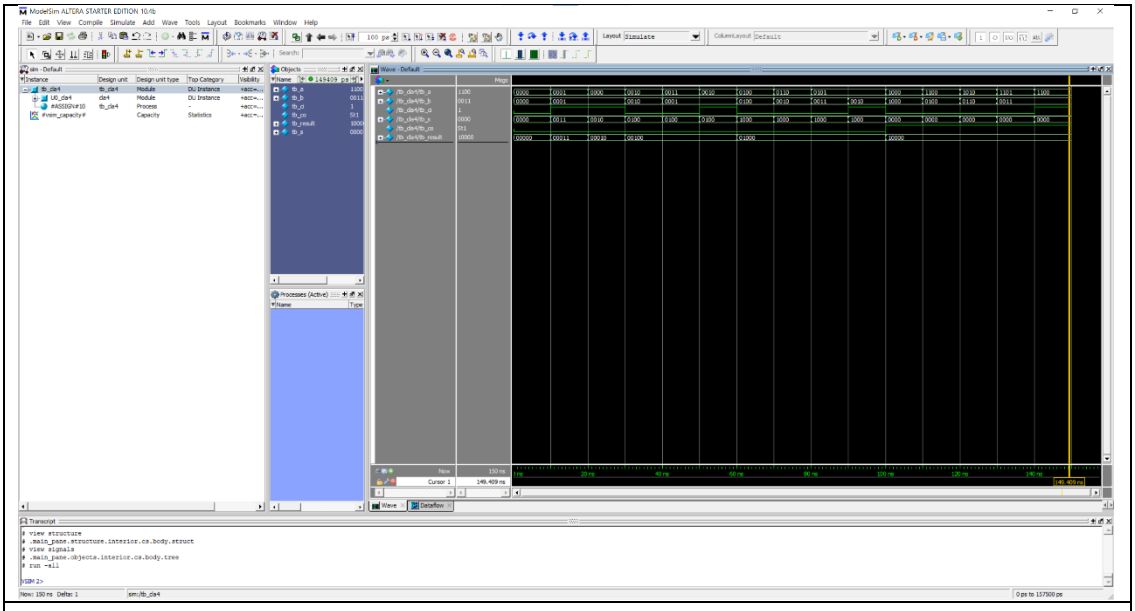
v. cla_clk & rca_clk

해당 모듈은 각각 cla32와 rca32를 이용하여 구성한다.

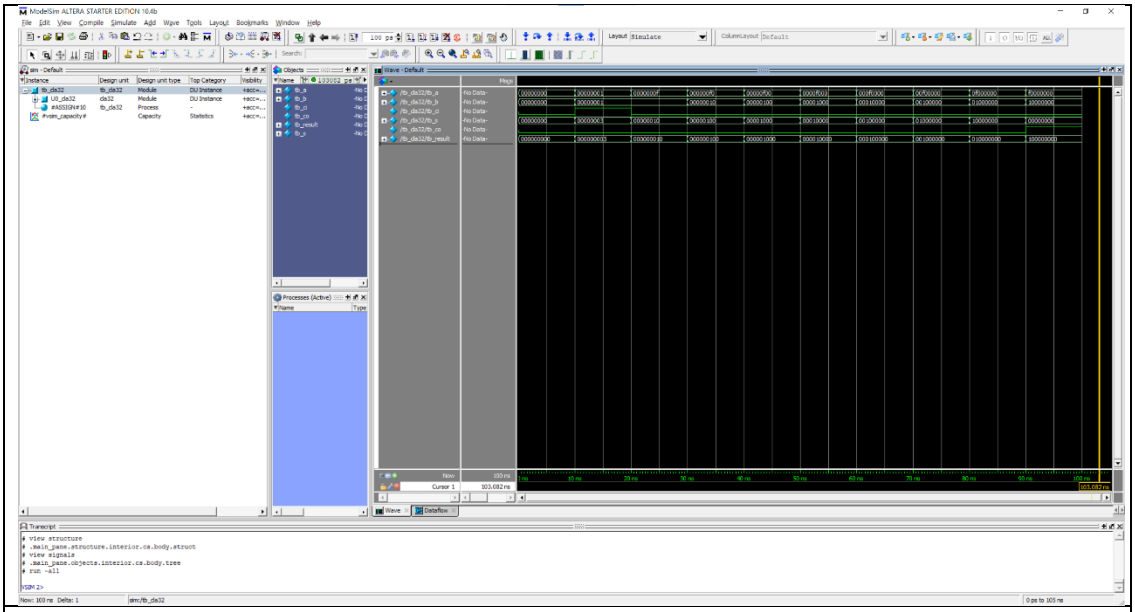
입력 a[31:0], b[31:0], ci와 reg reg_ci reg_a[31:0], reg_b[31:0]를 만들어 연결하고, 해당 reg를 모듈과 연결한다. 그리고 해당 모듈의 출력을 wire_co과 wire_s[31:0]에 연결하고, 이를 reg_co과 reg_s[31:0]에 연결한 후 이를 s와 co에 연결한다.

4. 설계 검증 및 실험 결과

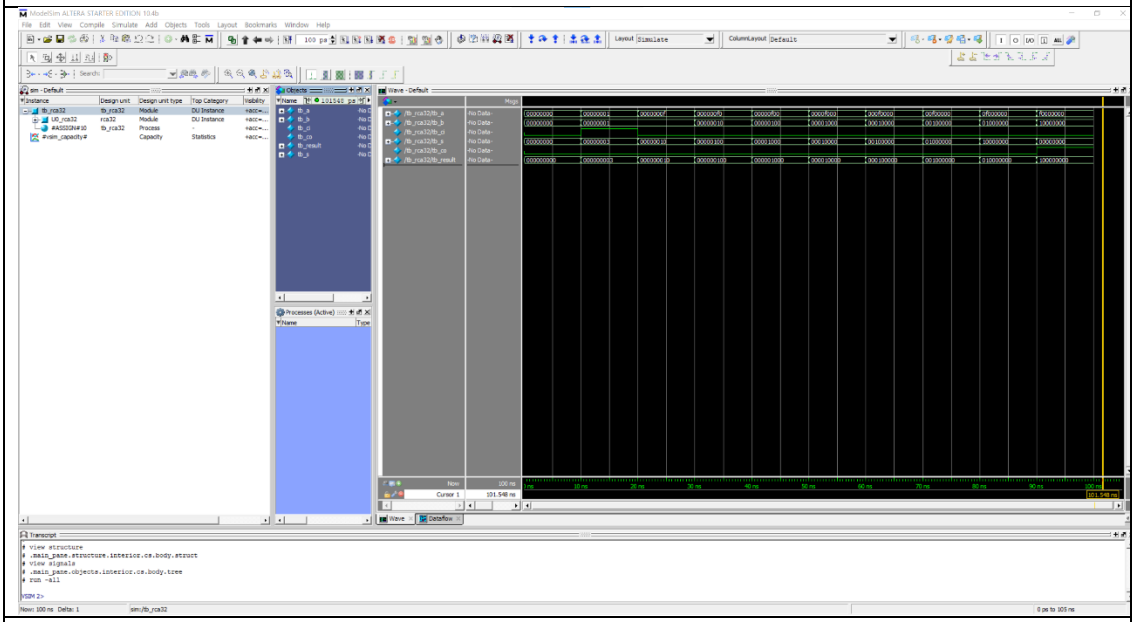
A. 시뮬레이션 결과



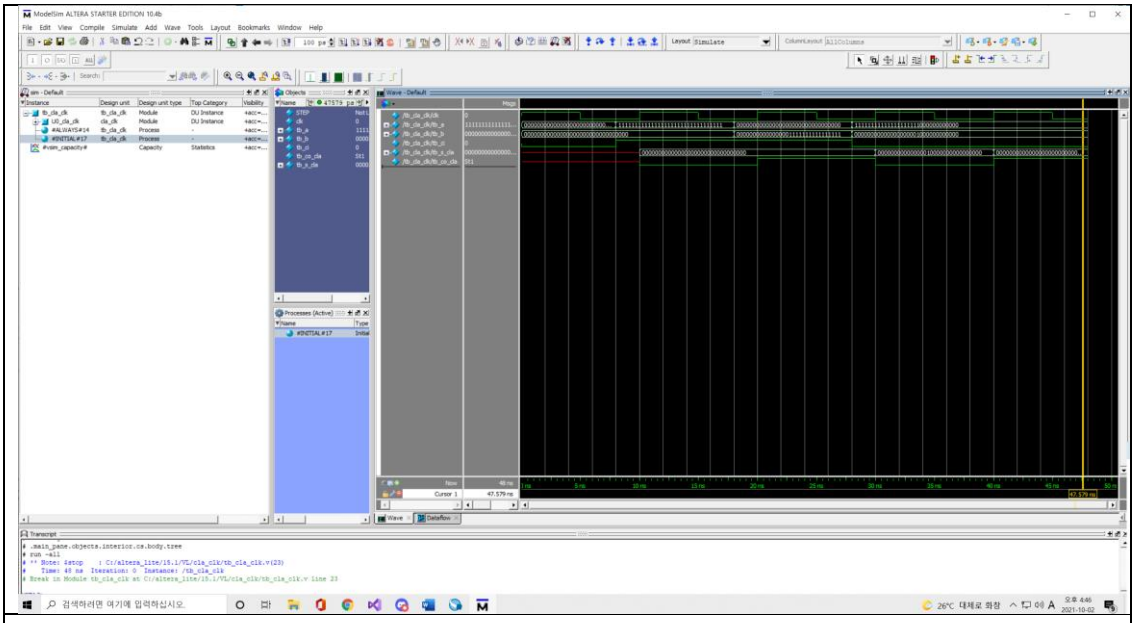
해당 testbench는 cla4를 검증하는 것이다. 각 자리의 입력이 정상적으로 되어있는지 확인하고, 각 자리에서 자리수가 정상적으로 넘어가는지 만 확인하여 회로가 정상적으로 구현되었는지 확인하는 Directed Verification을 사용하여 정상적으로 가동하는 것을 확인하였다.



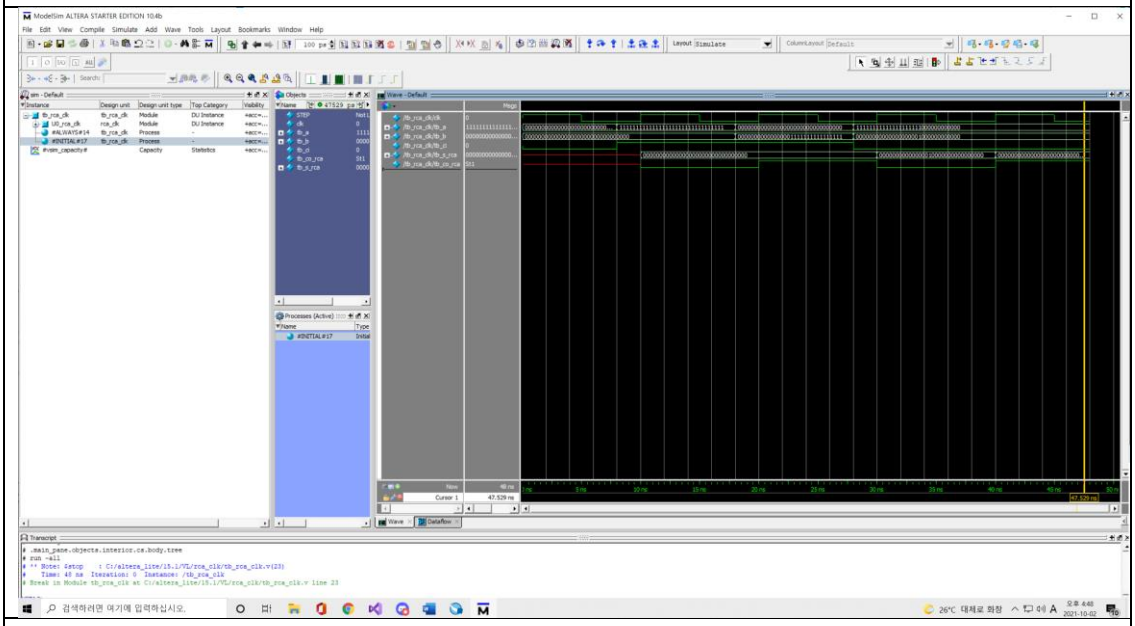
해당 testbench에서 또한 각 4bit 블록 단위로 값이 정상적으로 넘어가는지를 확인하여 해당 회로가 정상 작동하는지를 확인하는 Directed Verification을 사용하였다.



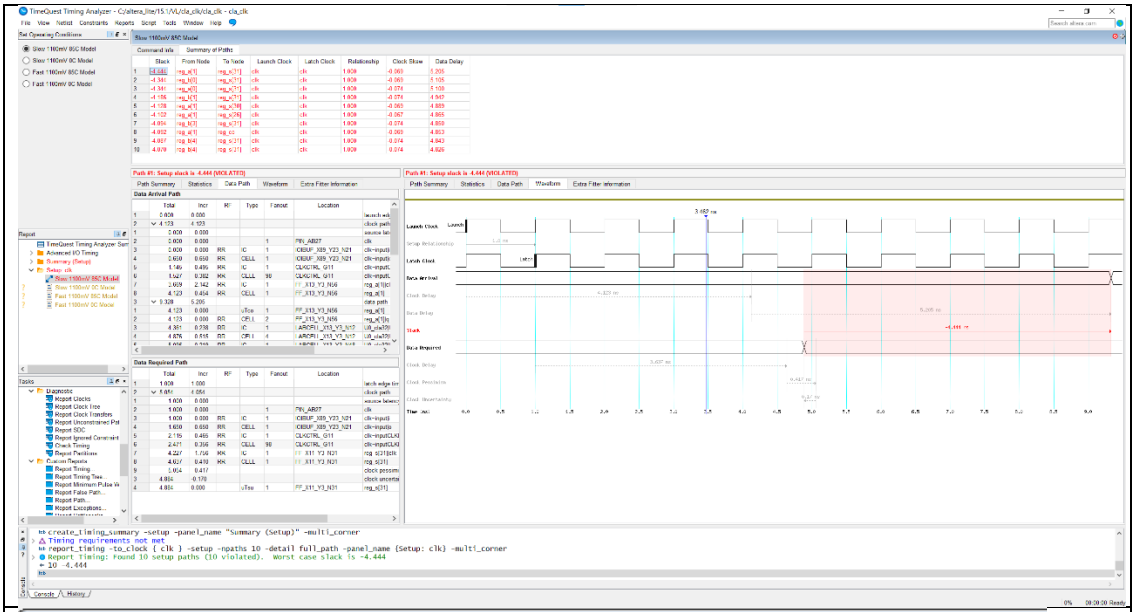
해당 testbench는 위에 사용되었던 tb_cla32와 같은 테스트벤치를 사용하여 동일한 값이 나오는지 확인함으로써 해당 회로를 검증하였다.



해당 testbench는 cla_clk 모듈이 클럭에 따라 정상적으로 값을 읽고 쓰는지 확인하는 모듈이다. 해당 모듈에서 정해진 위치에 따라 정상적으로 값이 들어가는 것을 확인할 수 있다.



해당 testbench는 rca_clk 모듈이 클럭에 따라 정상적으로 값을 읽고 쓰는지 확인하는 모듈이다. 해당 모듈에서 정해진 위치에 따라 정상적으로 값이 들어가는 것을 확인할 수 있다.

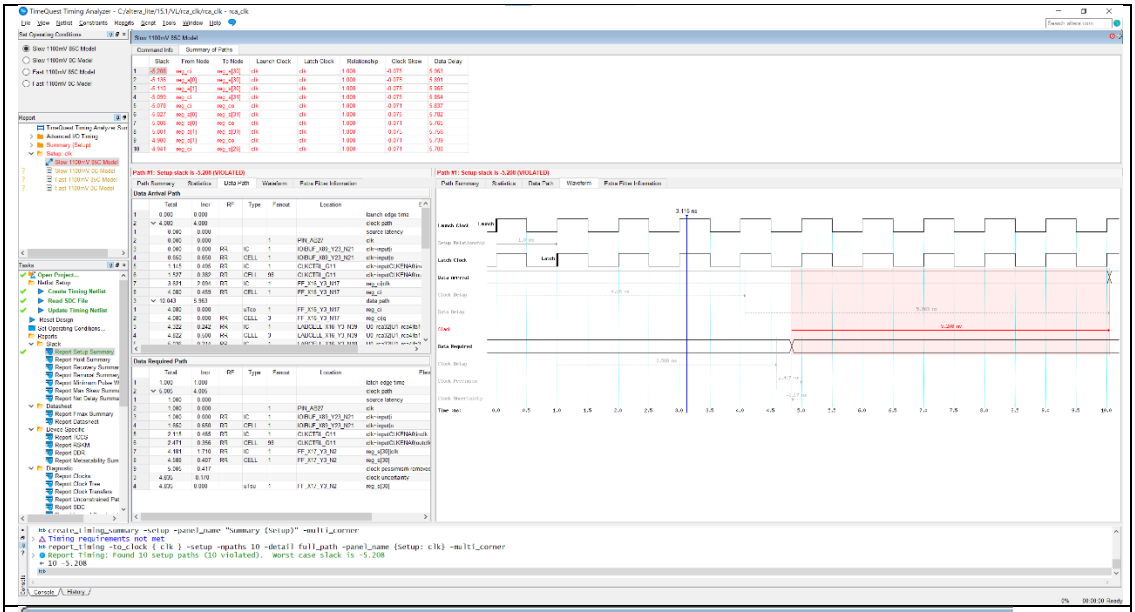


Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	183.69 MHz	183.69 MHz	clk	

해당 wavefrom은 cla_clk의 값이다. 해당 회로는 기본값 1ns에서 4.444만큼을 더한 5.444ns를 최소 clk으로 가지게 되며 이는 Frequency로 바꾸면 약 183.69MHz가 나오게 된다.

goekd



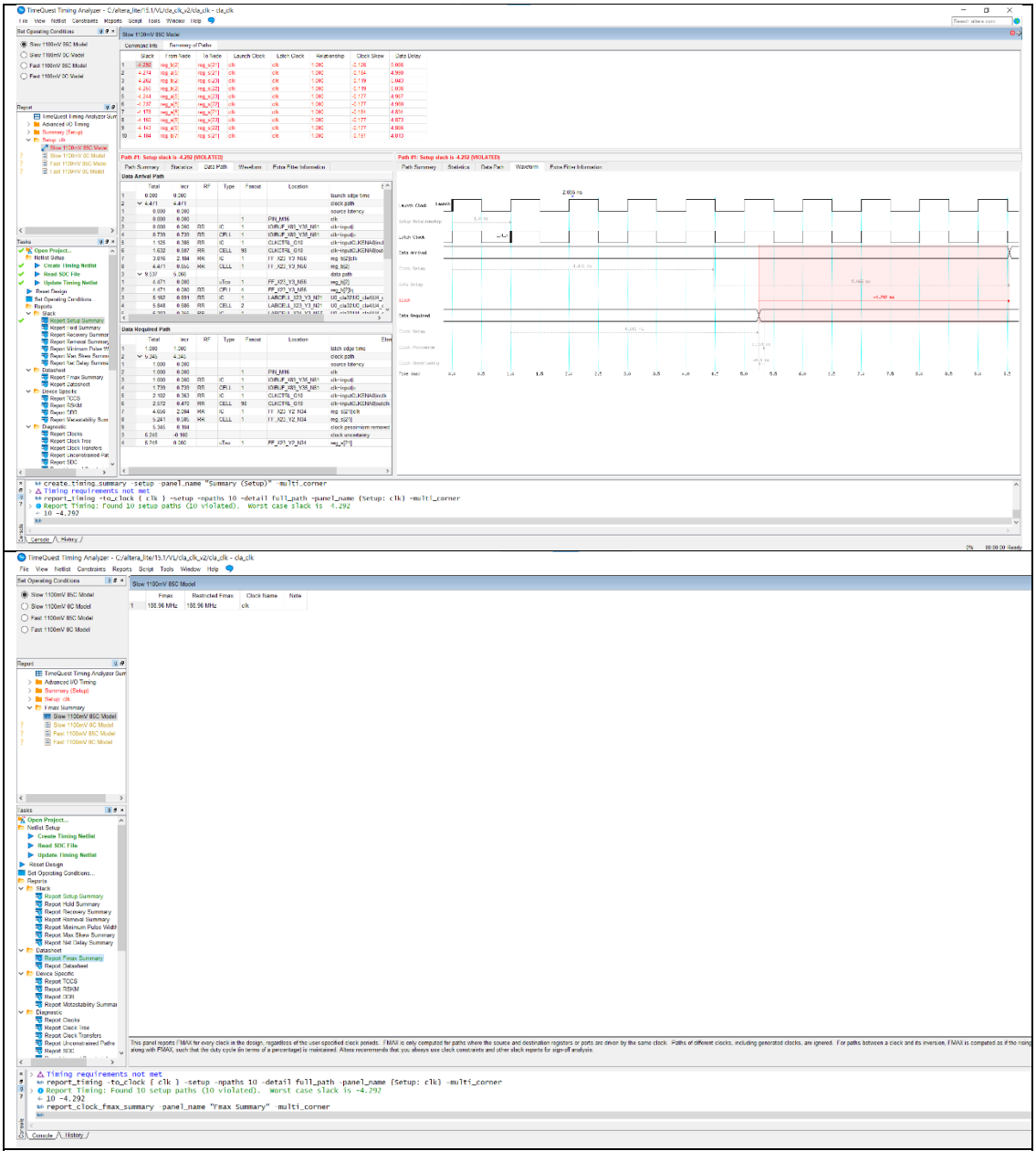
Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	161.08 MHz	161.08 MHz	clk	

해당 Waveform은 rca_clk의 값을 나타낸다. 해당 회로에서는 최소한으로 요구되는 clk의 값이 1ns + 5.26ns인 6.26ns보다 큰 값을 요구하며 이는 Frequency로 환산 시 161.08MHz의 값으로 바뀐다.

```verilog
module cla4(s, co, a, b, ci);
   input [3:0] a, b;
   input ci;
   output [3:0] s;
   output co;

   wire [3:0] c;

   fa U0_fa (s[0], c[0], a[0], b[0], ci);       //LSB ADD
   fa U1_fa (s[1], c[1], a[1], b[1], c[0]);     //Second bit ADD
   fa U2_fa (s[2], c[2], a[2], b[2], c[1]);     //Third bit ADD
   fa U3_fa (s[3], c[3], a[3], b[3], c[2]);     //MSB ADD
   clb4 U4_clb (co, a, b, ci);   //Carry Look-ahead Block
endmodule
```

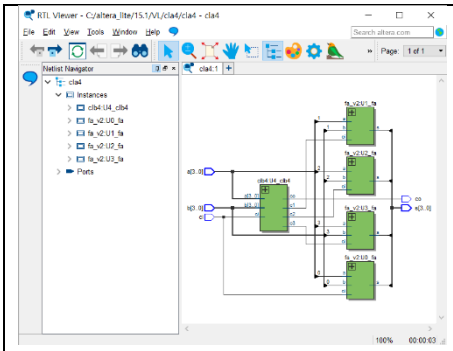
```verilog
module clb4(co, a, b, ci);
   input [3:0] a, b;
   input ci;
   output co;
   wire [3:0] g, p;
   wire w0_c1;
   wire w0_c2, w1_c2;
   wire w0_c3, w1_c3, w2_c3;
   wire w0_co, w1_co, w2_co, w3_co;

   // Generate
   _and2 G0(g[0], a[0], b[0]);
   _and2 G1(g[1], a[1], b[1]);
   _and2 G2(g[2], a[2], b[2]);
   _and2 G3(g[3], a[3], b[3]);


   // Propagate
   _or2 P0(p[0], a[0], b[0]);
   _or2 P1(p[1], a[1], b[1]);
   _or2 P2(p[2], a[2], b[2]);
   _or2 P3(p[3], a[3], b[3]);


   //a : P[3:0], ci 5input AND Gate
   _and5 U0and5(w0, p[0], p[1], p[2], p[3], ci);

   //b : G[3] + P[3] * (G[2] + P[2] * (G[1] + G[0] * P[1]))
   _and2 U0and2(w0c0, g[0], p[1]);
   _or2 U0or2(w0c1, w0c0, g[1]);
   _and2 U1and2(w1c0, w0c1, p[2]);
   _or2 U1or2(w1c1, w1c0, g[2]);
   _and2 U2and2(w2c0, w1c1, p[3]);
   _or2 U2or2(w1, w2c0, g[3]);

   //a OR b
   _or2 U3or2(co, w0, w1);

endmodule
```



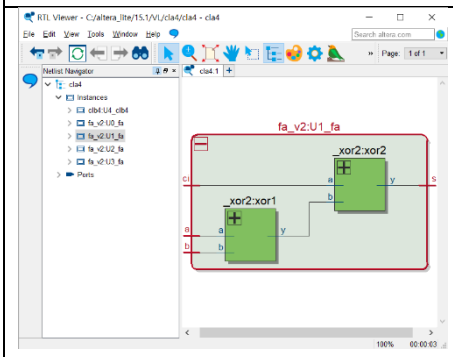
해당 회로는 modified cla32를 만들어 이 회로의 delay 시간을 측정한 것이다. 해당 회로의 clk은 5.292ns 이상이어야 하며 이는 188.96MHz로 바꿀 수 있다.

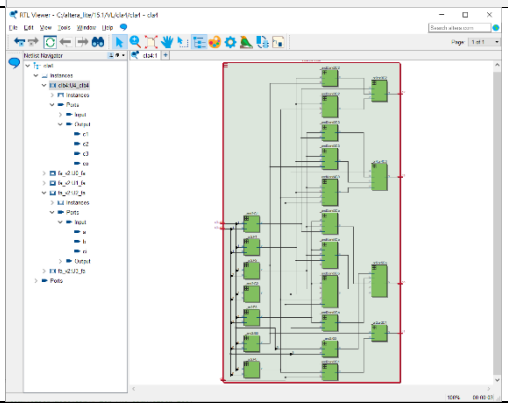
B. 합성(synthesis) 결과

cla4

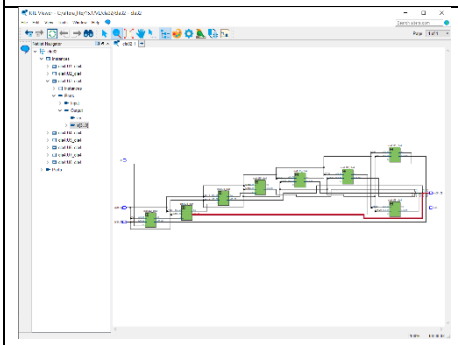


Flow Summary	
Flow Status	Successful - Wed Sep 29 20:13:15 2021
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	cla4
Top-level Entity Name	cla4
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	4 / 41,910 (< 1 %)
Total registers	0
Total 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 %)



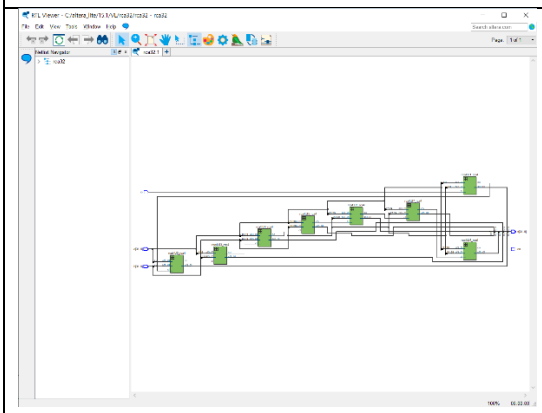


cla32



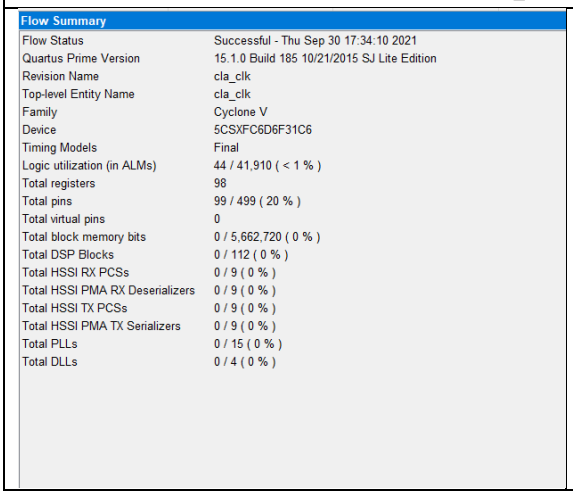
Flow Summary	
Flow Status	Successful - Wed Sep 29 21:46:46 2021
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	cla32
Top-level Entity Name	cla32
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	43 / 41,910 (< 1 %)
Total registers	0
Total pins	98 / 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 %)

rca32

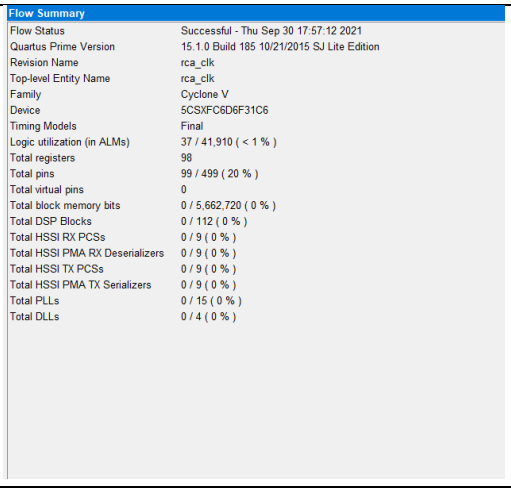


Flow Summary	
Flow Status	Successful - Wed Sep 29 22:36:18 2021
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	rca32
Top-level Entity Name	rca32
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	25 / 41,910 (< 1 %)
Total registers	0
Total pins	98 / 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 %)

cla_clk, rca_clk



Flow Summary	
Flow Status	Successful - Thu Sep 30 17:34:10 2021
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)	44 / 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 %)



Flow Summary	
Flow Status	Successful - Thu Sep 30 17:57:12 2021
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 %)

해당 모듈의 설계를 확인하면 각각의 모듈이 정상적으로 설계되어 연결되었음을 알 수 있다.

C. FPGA board targeting 결과

5. 고찰 및 결론

A. 고찰

이번 주차의 과제는 양이 많고 기존에 설계한 회로보다 좀 더 복잡한 회로였기 때문에 차근차근 해당 과제를 진행하였다. 해당 과제에서 또한 처음으로 clk을 넣어서 회로를 구성해 보는 시간이었기에 익숙하지 않아서 오는 어려움 또한 있었다. 다행히도 이러한 문제들이 있었지만 친구와 상의하며 서로의 문제를 이해하고 해결하여 과제를 완수할 수 있었다.

B. 결론

해당 실험을 통하여 RCA와 CLA의 구조의 차이를 알 수 있었다. 그리고 CLA가 RCA보다 딜레이가 더욱 작은 이유가 Carry만을 빠르게 보내기 때문이라는 것을 이해했다. CLA와 modified CLA 중 modified CLA의 속도가 빠른데, 이는 Carry Look-ahead 블록에서 ci가 co까지 도달하는데 지나는 모듈의 수가 CLA에 비해 적기 때문이라는 것도 알았다.

6. 참고문헌

이준환 교수님/디지털논리회로1/광운대학교(컴퓨터정보공학부)/2021

이준환 교수님/디지털논리회로2/광운대학교(컴퓨터정보공학부)/2021