컴퓨터 공학 기초 실험2 보고서

실험제목: Carry Look-ahead Adder (CLA)

실험일자: 2023년 09월 23일 (월)

제출일자: 2023년 10월 04일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학 번: 20222020264

성 명: 최봉규

1. 제목 및 목적

A. 제목

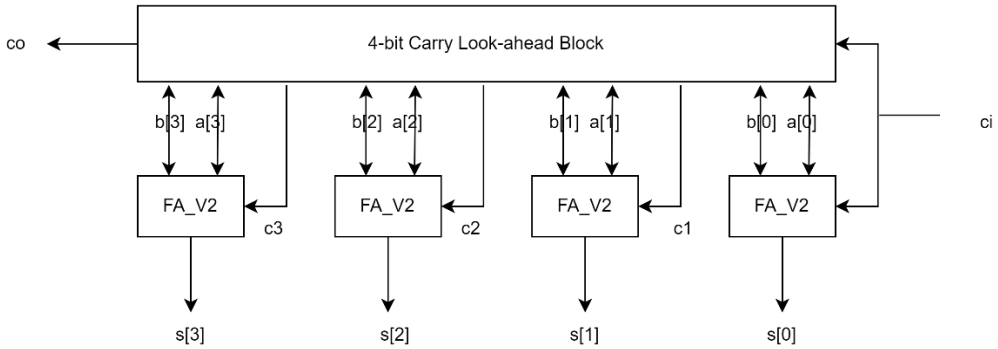
Carry Look-ahead Adder(CLA)

B. 목적

이번 실습은 carry look-ahead adder(CLA)의 원리를 이해하고 이를 디자인한다. Flip-flop을 이용하여 CLA와 ripple carry adder(RCA)의 input bit 수에 따른 delay를 관찰하고 두 가산기의 delay를 비교한다. Carry block의 형태를 바꿔 두 가지 형태의 CLA를 비교한다.

2. 원리(배경지식)

Carry Look-ahead Adder는 Ripple Carry Adder가 계산이 완료될 때까지의 시간이 많이 걸리는 단점을 보완하기 위해 입력 a, b 그리고 carry 가 주어질 때, 모든 올림수가 동시에 구해져 계산시간을 단축시킨 가산기이다. Carry 만을 계산해주는 별도의 carry look-ahead block이 존재한다. 이번 실습에서는 두 가지의 형태로 carry look-ahead block을 구현한다. 다음은 carry look-ahead adder의 형태이다.



Carry Look-ahead Block은 CLA로 carry out 값을 미리 계산해주기 위해서 generation signal, propagation signal을 정의한다.

$$G_i = A_iB_i$$

$$P_i = Ai + B_i$$

위 식을 full adder의 carry out에 적용하면 다음과 같다.

$$C_{i+1} = A_iB_i + (A_i + B_i)C_i = G_i + P_iC_i$$

이를 적용하여 4-bit adder의 carry를 미리 계산하면 다음과 같다. 다음의 연산은 이미 값이 결정된 carry의 연산 값을 활용하지 않고 한 번에 Instance하게끔 만든 것이다.

$$C_1 = G_0 + P_0Ci$$

$$C_2 = G_1 + P_1(G_0 + P_0Ci) = G_1 + P_1G_0 + P_1P_0Ci$$

$$C_3 = G_2 + P_2(G_1 + P_1(G_0 + P_0Ci)) = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0Ci$$

$$Co = G_3 + P_3(G_2 + P_2(G_1 + P_1(G_0 + P_0Ci))) = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0Ci$$

계산하는 bit수가 작을 때에는 RCA가 더 유리하다. 하지만 bit 수가 클 때에는 CLA가 더 유리하다. Delay를 계산할 때 RCA는 선형적으로 증가하는 것을 보여준다. Full adder 하나의 delay를 각 bit 마다 가지기 때문에 n-bit일 때 다음과 같은 딜레이 계산식을 갖는다.

$$t_{Delay} = n * t_{FA}$$

하지만 CLA와 같은 경우에는 올림수를 미리 계산하여 병렬적으로 처리하므로 큰 bit 수에 대해 Full adder보다 더 빠르다. CLA가 n-bit 가산할 때 k-bit로 carry look-ahead block이 있다고 가정할 때 다음은 CLA의 delay를 계산하는 식이다.

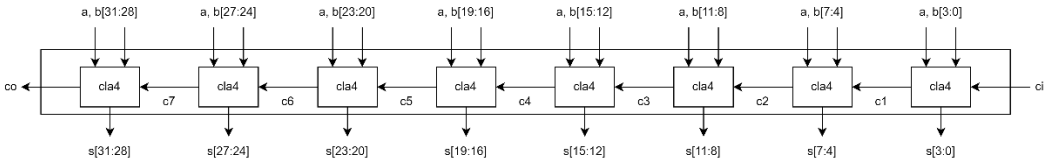
$$t_{Delay} = t_{pg} + t_{pg\_block} * (n/k - 1) + t_{AND\_OR} + k * t_{FA}$$

timing analysis tool을 이용하여 회로의 delay를 분석한다.

3. 설계 세부사항

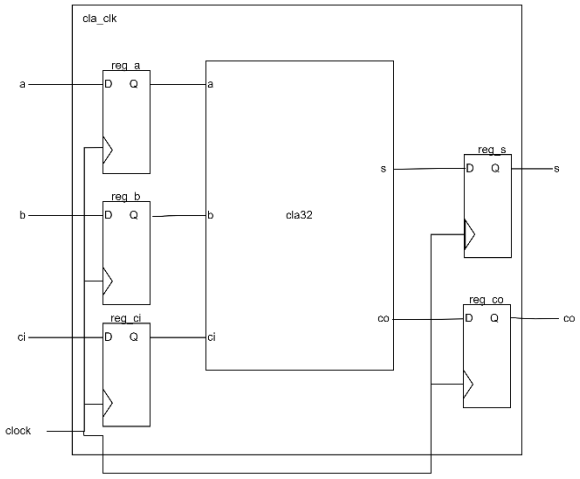
이번 실습에서의 input은 32bits a, b 그리고 1bit인 ci, output으로는 32bits s와 1bit co가 있다. a와 b는 서로 더할 값이며 ci는 LSB의 올림수 역할 output인 s는 a와 b를 가산한 값이 저장되며 co는 a와 b의 MSB에서의 덧셈 연산으로 인한 carry 가 발생한 것을 저장하게 된다.

Cla32는 cla4를 8개를 이어 붙어 만든다. Cla4를 설계할 때에는 full adder 4개를 이용해 sum을 계산하고 carry look-ahead block을 이용하여 각 bit에서의 올림수를 계산한다. 기존 full adder는 sum과 carry 까지만 계산하지만 이번 실습에서의 carry는 따로 연산하니 full adder에서의 co연산은 없애고 sum만 계산할 수 있도록 한다. 이 full adder를 fa_v2로 칭하겠다. Clb 에서는 앞서 설명했던 방식대로 propagation signal과 generation signal를 계산하여 각 bit에 대한 carry 값을 연산한다.



Cla_clk는 cla32를 토대로 각 input port와 output port에 flip flop을 달아서 사용한다. 이를 이용해 delay를 계산한다. 테벤을 이용하여 값이 입력되면 clk이 상승 엣지일 때 reg_a와 reg_b, reg_ci에 저장된다. 첫 번째 상승엣지에서는 input 값이 세팅만 되면서 output으

로 나오는 것은 z값이 된다. 초기의 값이 아직 output에 전달되었으나 전달되기 전 이미 업데이트를 하여 output port의 flip flop에 결과값이 저장되지 않았기 때문이다. 두번째 상승 엣지부터는 이전의 input이 circuit logic을 거쳐 연산이 완료됐던 값들이 output port의 flip flop인 reg_s와 reg_co에 업데이트 된다.

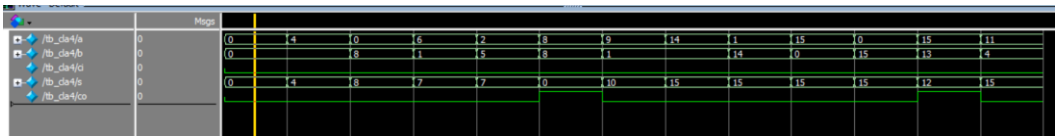


그림에서 확인할 수 있듯이 cla_clk module의 입력과 출력이 D-flip flop과 연결되어 있으며 모든 D-flip flop은 입력 clk에 연결된다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

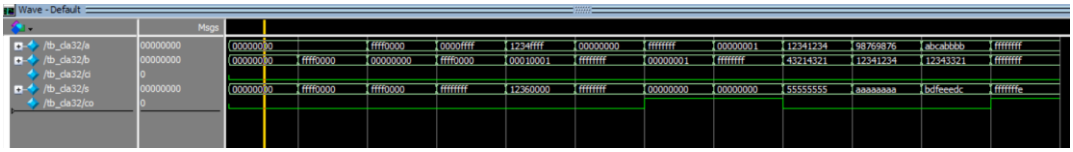
- tb_cla4



ID	Description	Input condition	Expected value	Measured value	comparison
1	0 + 0	a = 4'd0 b = 4'd0 ci = 1'b0	s = 4'd0 co = 1b'0	s = 4'd0 co = 1b'0	corrected
2	작은 수 + 0	a = 4'd4 b = 4'd0 ci = 1'b0	s = 4'd4 co = 1b'0	s = 4'd4 co = 1b'0	Corrected
3	0 + 중간 수	a = 4'd0 b = 4'd8 ci = 1'b0	s = 4'd8 co = 1b'0	s = 4'd8 co = 1b'0	Corrected
4	중간 수 + 작은 수	a = 4'd6 b = 4'd1 ci = 1'b0	s = 4'd7 co = 1b'0	s = 4'd7 co = 1b'0	Corrected

5	작은 수 + 중간 수	a = 4'd2 b = 4'd5 ci = 1'b0	s = 4'd7 co = 1b'0	s = 4'd7 co = 1b'0	Corrected
6	중간 수 + 중간 수 (overflow 발생)	a = 4'd8 b = 4'd8 ci = 1'b0	s = 4'd0 co = 1b'1	s = 4'd0 co = 1b'1	Corrected
7	중간 수 + 작은 수	a = 4'd9 b = 4'd1 ci = 1'b0	s = 4'd10 co = 1b'0	s = 4'd10 co = 1b'0	Corrected
8	큰 수 + 작은 수	a = 4'd14 b = 4'd1 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd15 co = 1b'0	Corrected
9	작은 수 + 큰 수	a = 4'd1 b = 4'd14 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd15 co = 1b'0	Corrected
10	큰 수 + 0	a = 4'd15 b = 4'd0 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd15 co = 1b'0	corrected
11	0 + 큰 수	a = 4'd0 b = 4'd15 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd15 co = 1b'0	Corrected
12	큰 수 + 큰 수 (overflow 발생)	a = 4'd15 b = 4'd13 ci = 1'b0	s = 4'd12 co = 1b'0	s = 4'd12 co = 1b'0	Corrected
13	큰 수 + 중간 수	a = 4'd11 b = 4'd4 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd15 co = 1b'0	Corrected

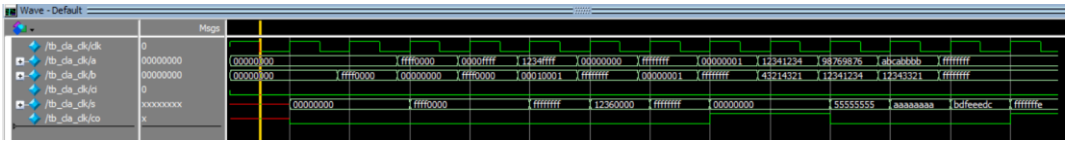
- tb_cla32



ID	Description	Input condition	Expected	Measured	comparison

1	0 + 0	a ='h0000_0000 b ='h0000_0000 ci = 1'b0	s ='h0000_0000 co = 1b'0	s ='h0000_0000 co = 1b'0	corrected
2	0 + 중간 수	a ='h0000_0000 b ='hffff_0000 ci = 1'b0	s ='hffff_0000 co = 1b'0	s ='hffff_0000 co = 1b'0	corrected
3	중간 수 + 0	a ='hffff_0000 b ='h0000_0000 ci = 1'b0	s ='hffff_0000 co = 1b'0	s ='hffff_0000 co = 1b'0	corrected
4	작은 수+ 중간 수	a ='h0000_ffff b ='hffff_0000 ci = 1'b0	s ='hffff_ffff co = 1b'0	s ='hffff_ffff co = 1b'0	corrected
5	중간 + 작은	a ='h1234_ffff b ='h0001_0001 ci = 1'b0	s ='h1236_0000 co = 1b'0	s ='h1236_0000 co = 1b'0	corrected
6	0 + 큰 수	a ='h0000_0000 b ='hffff_ffff ci = 1'b0	s ='hffff_ffff co = 1b'0	s ='hffff_ffff co = 1b'0	corrected
7	큰 수 + 작은 수	a ='hffff_ffff b ='h0000_0001 ci = 1'b0	s ='h0000_0000 co = 1b'1	s ='h0000_0000 co = 1b'1	corrected
8	작은 수 + 큰 수	a ='h0000_0001 b ='hffff_ffff ci = 1'b0	s ='h0000_0000 co = 1b'1	s ='h0000_0000 co = 1b'1	corrected
9	중간 수 + 중간 수	a ='h1234_1234 b ='h4321_4321 ci = 1'b0	s ='h5555_5555 co = 1b'0	s ='h5555_5555 co = 1b'0	corrected
10	중간 수 + 중간 수	a ='h9876_9876 b ='h1234_1234 ci = 1'b0	s ='haaaa_aaaa co = 1b'0	s ='haaaa_aaaa co = 1b'0	corrected
11	중간 수 + 중간 수	a ='habca_bbbb b ='h1234_3321 ci = 1'b0	s ='hbdfe_eedc co = 1b'0	s ='hbdfe_eedc co = 1b'0	corrected
12	큰 수 + 큰 수	a ='hffff_ffff b ='hffff_ffff ci = 1'b0	s ='hffff_fffe co = 1b'1	s ='hffff_fffe co = 1b'1	corrected

- tb_cla_clk

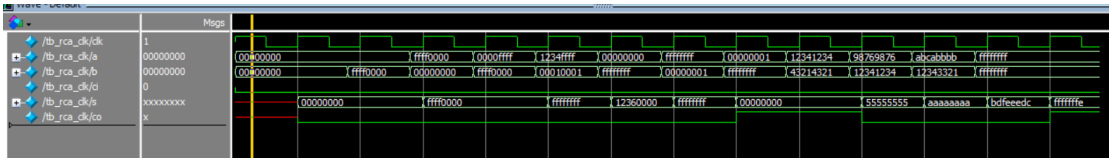


ID	Description	Input condition	Expected	Measured	comparison
1	0 + 0	a ='h0000_0000 b ='h0000_0000 ci = 1'b0	s ='hxxxx_xxxx co = 1b'x	s ='hxxxx_xxxx co = 1b'x	corrected
2	0 + 중간 수	a ='h0000_0000 b ='hffff_0000 ci = 1'b0	s ='h0000_0000 co = 1b'0	s ='h0000_0000 co = 1b'0	corrected
3	중간 수 + 0	a ='hffff_0000 b ='h0000_0000 ci = 1'b0	s ='hffff_0000 co = 1b'0	s ='hffff_0000 co = 1b'0	corrected
4	작은 수+ 중간 수	a ='h0000_ffff b ='hffff_0000 ci = 1'b0	s ='hffff_0000 co = 1b'0	s ='hffff_0000 co = 1b'0	corrected
5	중간 + 작은	a ='h1234_ffff b ='h0001_0001 ci = 1'b0	s ='hffff_ffff co = 1b'0	s ='hffff_ffff co = 1b'0	corrected
6	0 + 큰 수	a ='h0000_0000 b ='hffff_ffff ci = 1'b0	s ='h1236_0000 co = 1b'0	s ='h1236_0000 co = 1b'0	corrected
7	큰 수 + 작은 수	a ='hffff_ffff b ='h0000_0001 ci = 1'b0	s ='hffff_ffff co = 1b'0	s ='hffff_ffff co = 1b'0	corrected
8	작은 수 + 큰 수	a ='h0000_0001 b ='hffff_ffff ci = 1'b0	s ='h0000_0000 co = 1b'1	s ='h0000_0000 co = 1b'1	corrected
9	중간 수 + 중간 수	a ='h1234_1234 b ='h4321_4321 ci = 1'b0	s ='h0000_0000 co = 1b'1	s ='h0000_0000 co = 1b'1	corrected
10	중간 수 + 중간 수	a ='h9876_9876 b ='h1234_1234 ci = 1'b0	s ='h5555_5555 co = 1b'0	s ='h5555_5555 co = 1b'0	corrected
11	중간 수 + 중간 수	a ='habca_bbbb b ='h1234_3321 ci = 1'b0	s ='haaaa_aaaa co = 1b'0	s ='haaaa_aaaa co = 1b'0	corrected

12	큰 수 + 큰 수	a ='hffff_ffff b ='hffff_ffff ci = 1'b0	s ='hbdfe_eedc co = 1b'0	s ='hbdfe_eedc co = 1b'0	corrected
13			s ='hffff_fffe co = 1b'1	s ='hffff_fffe co = 1b'1	corrected

Cla_clk는 flip flop을 활용하기 때문에 해당 ID의 입력 수는 다음 ID에 적용되어 연산 결과가 나온다. 그렇기에 ID1의 결과값은 Instance가 안 된 것처럼 보이지만 ID2부터의 결과값을 보면 이전 입력 값들의 합산이 결과로 나오는 것을 확인할 수 있다.

- tb_rca_clk



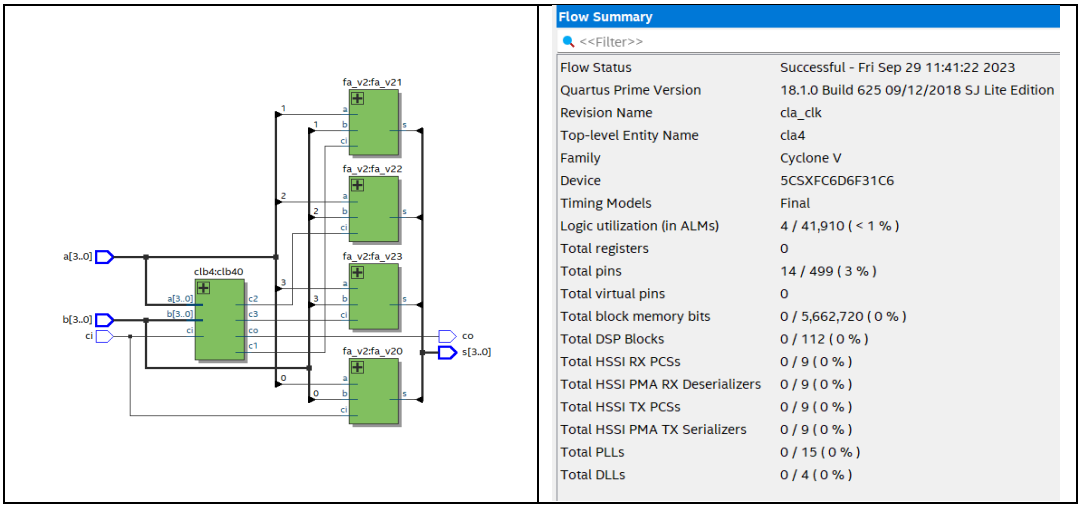
ID	Description	Input condition	Expected	Measured	comparison
1	0 + 0	a ='h0000_0000 b ='h0000_0000 ci = 1'b0	s ='hxxxx_xxxx co = 1b'x	s ='hxxxx_xxxx co = 1b'x	corrected
2	0 + 중간 수	a ='h0000_0000 b ='hffff_0000 ci = 1'b0	s ='h0000_0000 co = 1b'0	s ='h0000_0000 co = 1b'0	corrected
3	중간 수 + 0	a ='hffff_0000 b ='h0000_0000 ci = 1'b0	s ='hffff_0000 co = 1b'0	s ='hffff_0000 co = 1b'0	corrected
4	작은 수+ 중간 수	a ='h0000_ffff b ='hffff_0000 ci = 1'b0	s ='hffff_0000 co = 1b'0	s ='hffff_0000 co = 1b'0	corrected
5	중간 + 작은	a ='h1234_ffff b ='h0001_0001 ci = 1'b0	s ='hffff_ffff co = 1b'0	s ='hffff_ffff co = 1b'0	corrected
6	0 + 큰 수	a ='h0000_0000 b ='hffff_ffff ci = 1'b0	s ='h1236_0000 co = 1b'0	s ='h1236_0000 co = 1b'0	corrected
7	큰 수 + 작은 수	a ='hffff_ffff b ='h0000_0001 ci = 1'b0	s ='hffff_ffff co = 1b'0	s ='hffff_ffff co = 1b'0	corrected

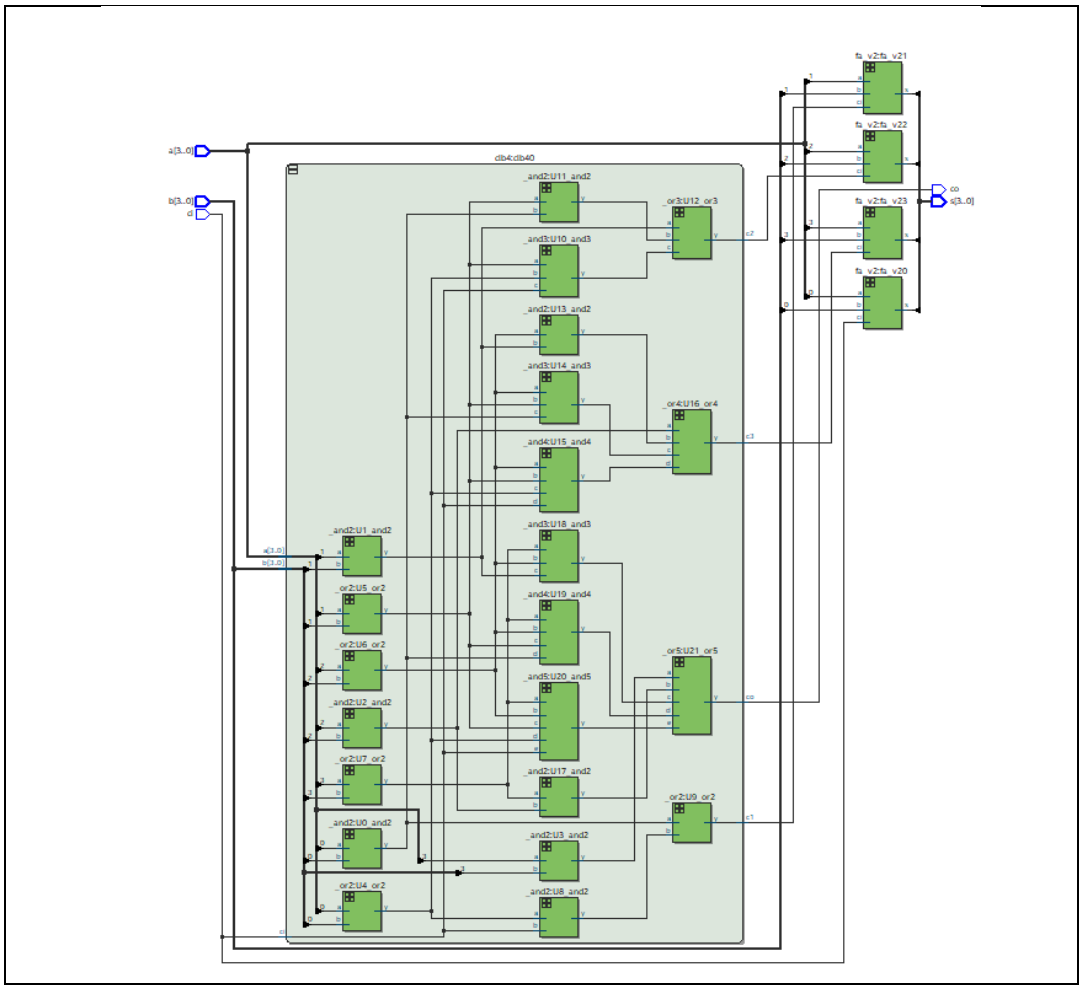
8	작은 수 + 큰 수	a ='h0000_0001 b ='hffff_ffff ci = 1'b0	s ='h0000_0000 co = 1b'1	s ='h0000_0000 co = 1b'1	corrected
9	중간 수 + 중간 수	a ='h1234_1234 b ='h4321_4321 ci = 1'b0	s ='h0000_0000 co = 1b'1	s ='h0000_0000 co = 1b'1	corrected
10	중간 수 + 중간 수	a ='h9876_9876 b ='h1234_1234 ci = 1'b0	s ='h5555_5555 co = 1b'0	s ='h5555_5555 co = 1b'0	corrected
11	중간 수 + 중간 수	a ='habca_bbbb b ='h1234_3321 ci = 1'b0	s ='haaaa_aaaa co = 1b'0	s ='haaaa_aaaa co = 1b'0	corrected
12	큰 수 + 큰 수	a ='hffff_ffff b ='hffff_ffff ci = 1'b0	s ='hbdfe_eedc co = 1b'0	s ='hbdfe_eedc co = 1b'0	corrected
13			s ='hffff_fffe co = 1b'1	s ='hffff_fffe co = 1b'1	corrected

rca_clk 역시 cla_clk와 같이 flip flop을 활용하기 때문에 해당 ID의 입력 수는 다음 ID에 적용되어 연산 결과가 나온다. 그렇기에 ID1의 결과값은 Instance가 안 된 것처럼 보이지만 ID2부터의 결과값을 보면 이전 입력 값들의 합산이 결과로 나오는 것을 확인할 수 있다.

B. 합성(synthesis) 결과

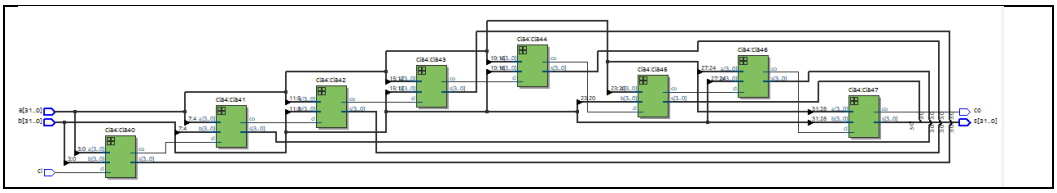
- tb_cla4





RTL viewer에서 볼 수 있듯이 Cla4는 carry look-ahead block과 sum값을 계산하는 full-adder를 수정한 fa_v2 4개로 이루어져 있다. Flow summary를 보면 successful로 나온 것을 보아 제대로 합성되었음을 알 수 있다.

- tb_cla32

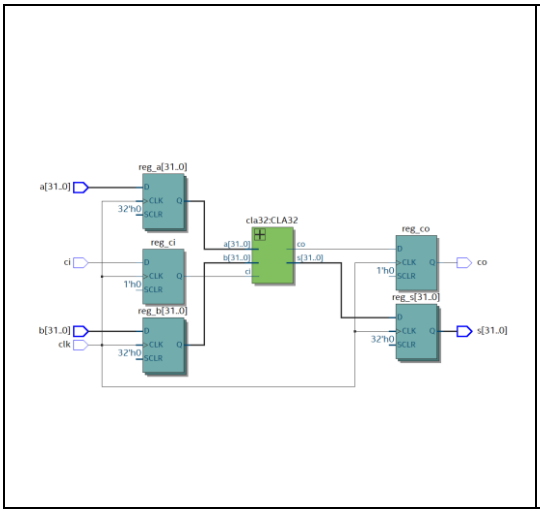


Flow Summary

<<Filter>>

Flow Status	Successful - Fri Sep 29 11:47:25 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	cla_clk
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 %)

- RTL viewer를 보면 알 수 있듯이 cla32는 cla4를 8개로 이어 붙여 만든 것을 확인할 수 있다. Flow summary를 보면 successful로 나온 것을 보아 제대로 합성되었음을 알 수 있다.
- tb_cla_clk



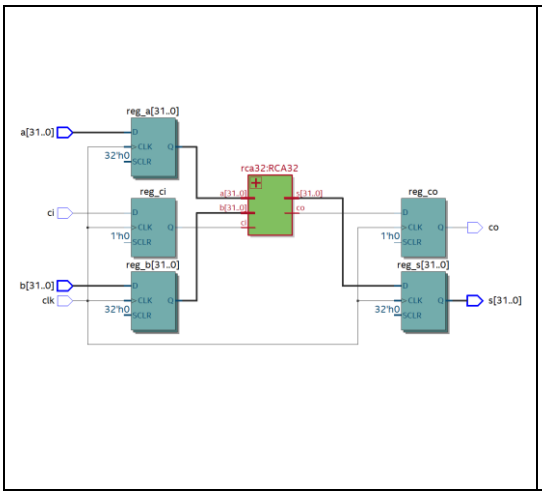
Flow Summary

<<Filter>>

Flow Status	Successful - Fri Sep 29 11:52:10 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 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)	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 %)

RTL viewer를 보면 알 수 있듯이 cla_clk라는 logic circuit 과 그 바깥에 input과 output을 저장하는 register역할 flip flop이 있는 것을 확인할 수 있다. LC는 cla 32로 carry look-ahead adder 가산기로 32bit를 연산할 수 있는 회로가 안에 있다. Flow summary에서 보면 Total registers가 98개 인 것을 확인할 수 있는데, 이는 flip flop에 연결되어 있는 bit수로 알 수 있다. input에서 a, b이고 output 에서는 s가 32bit를 가지고 있다. 그리고 1bit짜리인 ci와 co가 있다. 32* 3 + 2 = 98개이며 레지스터의 개수가 맞게 연결됐음과 status에서 successful로 잘 합성됐음을 알 수 있다.

- tb_rca_clk



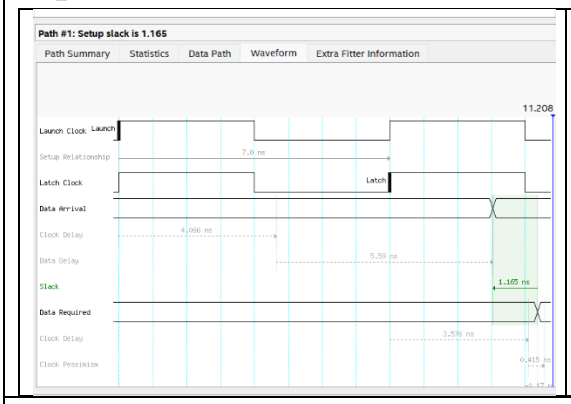
Flow Summary	
<<Filter>>	
Flow Status	Successful - Fri Sep 29 13:32:35 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	cla_clk
Top-level Entity Name	rca_clk
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	31 / 41,910 (< 1 %)
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 %)

앞서 설명한 cla_clk와 마찬 가지로 LC에 rca 32가 들어가 있다. Total registers역시 32bit 3개와 1bit 2개인 것을 확인할 수 있고, flow status에서 successful로 잘 합성됐음을 확인할 수 있다.

5. 고찰 및 결론

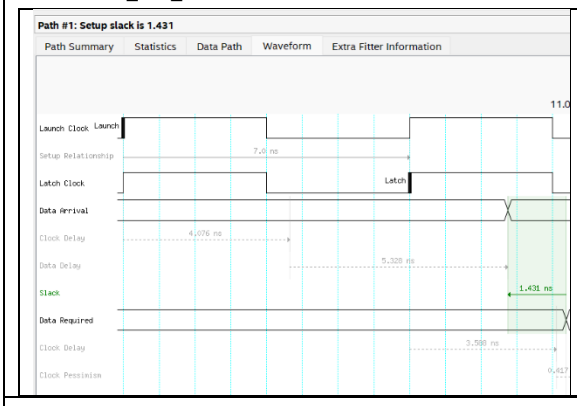
A. 고찰

Cla_clk



	Fmax	Restricted Fmax	Clock Name
1	171.38 MHz	171.38 MHz	clk

Modified_cla_clk

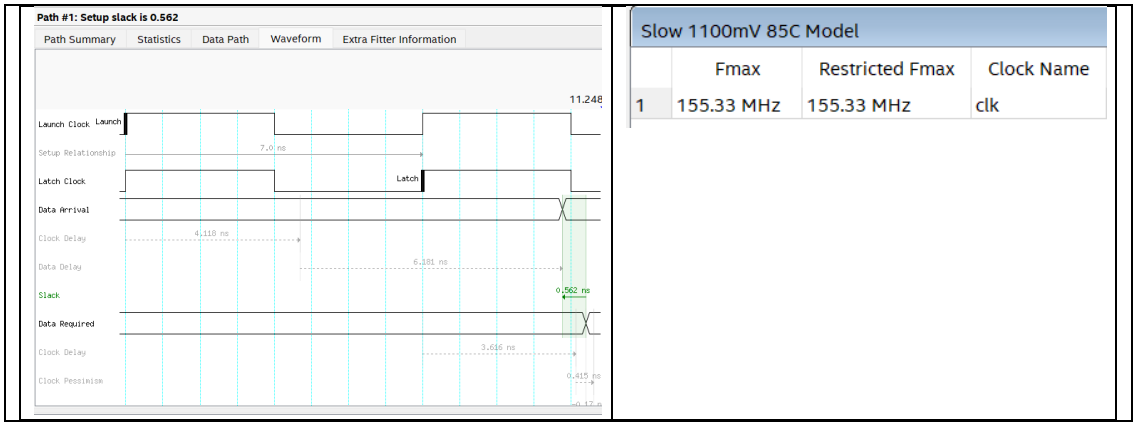


Reports Script Tools Window Help

Slow 1100mV 85C Model

	Fmax	Restricted Fmax	Clock Name
1	179.57 MHz	179.57 MHz	clk

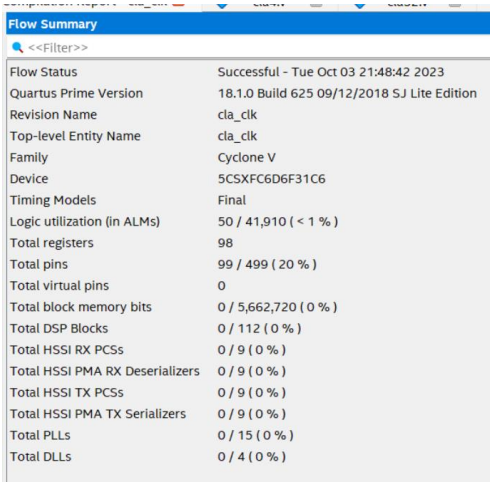
Rca_clk



클럭의 주기를 7ns로 세팅하고나서 32bit 수를 가산할 때 최대 클럭 주파수가 modified_cla > cla > rca 순으로 modified_cla일 때 연산이 좀 더 빠르다는 것을 확인할 수 있다. 세 개의 Fmax의 값은 위에 적은 순서대로 179.57MHz, 171.38MHz, 155.33MHz 이다. 세 slack은 1.431ns, 1.165ns, 0.562ns이다. setup timing violation이 발생하지 않도록 slack을 양수로 만들고 나서 fmax를 비교를 하는데, 클럭 주파수 1ns일 때는 violation이 발생했는데 7ns로 바꾸니 세 경우 모두 slack이 양수가 되는 것을 확인할 수 있었다.

Flow summary에서 Cla와 size를 비교했을 때 cla는 Logic utilization이 43이 었으며, rca가 31인 것을 보아 rca의 size가 더 작다. Modified cla의 경우에는 50이 나와 셋 중 제일 컸다.

Modified_cla의 경우에 기존 full adder 역할을 하는 modified_fa 세 개와 cla용 full adder인 fa_v2 한 개를 이용해서 4-bits sum와 c1, c2, c3를 구하고 co의 경우에는 modified_clb4에서 cin과 propagation signal과 generation signal의 조합만으로 구하였다. modified_fa는 fa_v2.v 파일에 있으며, rca4의 fa는 half adder를 gates.v에 있는 모듈이 아닌 assign을 이용하기 때문에 이와 구분되는 fa의 필요로 따로 구현하게 되었다. 아래는 modified_cla에서의 flow summary이다.



B. 결론

Quartus 프로그램에서 Timing analysis tool을 이용하여 slack을 조절하고 클럭 최대 주파수(Fmax)라는 것을 관찰하는 방법에 대해 알게 되었다.

32bit cla는 32bit rca보다 크기가 크다. 이는 cla4와 rca4를 비교해 보면 알 수 있다. Cla4는 clb에 의해 rca4와 비교했을 때 더욱 복잡한 회로를 보이는 것을 확인할 수 있었다. 하지만 n-bits rca의 경우 Full adder를 n개 연결하여 설계하므로 n이 증가할 수록 delay가 선형적으로 증가해 연산 속도가 느려지는 것을 앞서 확인했다. 하지만 cla는 bit 수가 커져도 올림을 한꺼번에 계산하여 delay가 증가는 하지만 rca의 증가율 보다 적은 증가율을 보인다. Size가 커지면 대체로 Fmax가 커지는 것을 파악했다. 무조건 비례한다는 것은 아니지만 이 세가지의 경우로 분석했을 때 어느 정도의 상관관계가 있다고 볼 수 있다.

6. 참고문헌

이준환 교수님/ 디지털논리회로2/ 광운대학교(컴퓨터정보공학부)/ 2023