컴퓨터 공학 기초 실험2 보고서

실험제목: Ripple Carry Adder

실험일자: 2023년 09월 20일 (수)

제출일자: 2023년 09월 20일 (수)

학 과: 컴퓨터정보공학부

담당교수: 이혁준 교수님

실습분반: 수 0, 1, 2

학 번: 2022202075

성 명: 우나륜

1. 제목 및 목적

A. 제목

Ripple Carry Adder

B. 목적

Half adder와 full adder의 진리표와 카르노맵을 통해 각 회로의 작동 원리를 이해하고 이를 사용하여 Ripple carry adder를 설계할 수 있다. 위에서 얻은 각 회로의 식을 통해 Verilog로 half adder, full adder, 4 bits ripple carry adder를 구현할 수 있다.

2. 원리(배경지식)

A. Half adder

Half adder(반가산기)는 2개의 1 bit input을 받아 각 입력들의 합인 sum(s)과 자리올림수인 carry out(co)을 출력하는 가산기이다. 아래의 표1은 half adder의 진리표이며, 표2와 표3은 진리표를 바탕으로 계산한 co와 s의 Karnaugh map이다.

Input		**Output**	
a	**b**	**co**	**s**
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

표1. Half Adder truth table

a / **b**	**0**	**1**
0	0	0
1	0	1

표2. Carry out Karnaugh map

위 표2를 통해 얻은 carry out의 Boolean equation은 다음과 같다. $Carry\ out\ co = ab$

a / **b**	**0**	**1**
0	0	1
1	1	0

표3. Sum Karnaugh map

위 표3을 통해 얻은 s의 Boolean equation은 다음과 같다. $Sum\ s = a\bar{b} + \bar{a}b = a \oplus b$

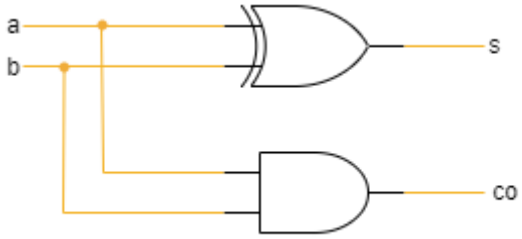

그림1. Half Adder

위에서 구한 s와 co의 Boolean equation을 바탕으로 회로를 나타내면 그림1과 같다. S는 a와 b의 XOR연산 결과이며, co는 a와 b의 AND연산을 통해 구할 수 있다.

B. Full adder

Full adder(전가산기)는 3개의 1 bit input을 받아 각 입력들의 합인 sum과 carry out을 출력하는 가산기이다. Half adder와 다르게, full adder는 1 bit 입력 a와 b 뿐만 아니라 전 단계의 자리 올림수인 carry in을 추가로 입력받는다. 따라서 입력 a, b와 carry in(ci)을 더하여 sum(s)와 carry out(co)을 계산한다. 아래의 표4는 Full adder의 진리표이고, 이를 바탕으로 full adder의 carry out과 sum의 Karnaugh map을 표5와 표6으로 정리하였다.

Input			Output	
a	**b**	**ci**	**co**	**s**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

표4. Full Adder truth table

ci \ ab	**00**	**01**	**11**	**10**
0	0	0	1	0
1	0	1	1	1

표5. Carry out Karnaugh map

위 표5를 통해 얻은 carry out의 Boolean equation은 다음과 같다.

$$Carry\ out\ co = ab + bci + aci$$

ci \ ab	00	01	11	10
0	0	1	0	1
1	1	0	1	0

표6. Sum Karnaugh map

위 표3을 통해 얻은 s의 Boolean equation은 다음과 같다.

$$Sum\ s = \bar{a}b\overline{ci} + a\overline{bci} + \overline{ab}ci + abci$$
$$= \overline{ci}(\bar{a}b + a\bar{b}) + ci(\overline{ab} + ab) = \overline{ci}(a \oplus b) + ci(\overline{a \oplus b})$$
$$= ci \oplus a \oplus b$$

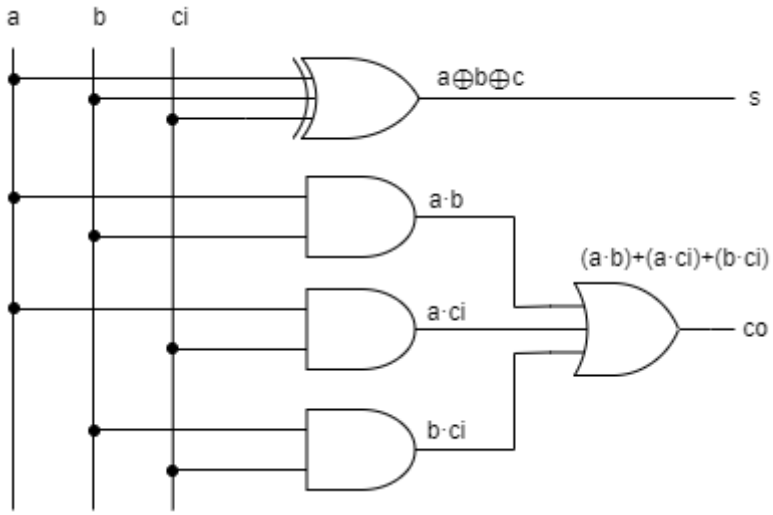

그림2. Full Adder

위에서 구한 s와 co의 Boolean equation을 통해 full adder의 회로를 나타내면 그림 2와 같다. 합 s의 값은 a와 b와 ci를 XOR연산한 값이며, co의 값은 ab와 aci와 bci를 OR연산한 값과 같다.

C. Ripple Carry Adder (RCA)

N-bit Ripple Carry Adder는 n개의 비트 수를 가지는 두 개의 입력을 더할 수 있는 가산기이다. Ripple Carry Adder는 더하려는 두 입력의 비트의 개수만큼 full adder를 연결하여 구현할 수 있다.

RCA는 전가산기를 사용하여 간단하게 구현할 수 있지만, 이전 비트에서 캐리가 계산되어 입력으로 입력될 때까지 상위 비트 가산기는 아무런 수행없이 기다려야 하므로 비트 수가 늘어날수록 연산 속도가 느려진다는 단점을 가진다.

D. 2's Complement

2's complement(2의 보수)란 어떠한 2진수의 모든 0과 1을 0은 1로, 1은 0으로 반

전하고 1을 더해 음수를 얻고 어떠한 수와 어떠한 수의 2의 보수를 더하면 결과값은 0이 되어야 한다. 그리고 어떠한 수의 2의 보수의 MSB는 1이 되어 음수를 나타낸다. 예를 들어, 4-bit 0011_2(+3)의 보수를 구하자면, 먼저 0과 1의 값을 모두 반전시킨다. 그러면 1100_2를 얻을 수 있고 이에 +1을 더하면 1101_2를 얻을 수 있고, 그 값은 -3이 된다. 그리고 0011_2와 그에 대한 보수 1101_2를 더하면 10000_2을 얻을 수 있고, 이에 따라 0011_2의 보수는 1101_2이 된다는 것을 알 수 있다.

3. 설계 세부사항

A. Half Adder 설계

Half Adder는 위 표2와 표3의 Karnaugh map에서 구한 Boolean equation을 바탕으로 그림1과 같이 1개의 AND 게이트와 1개의 XOR 게이트를 사용하여 간단하게 구현할 수 있다. 따라서 half adder는 Verilog에서 input a와 b를 입력받고 2-input AND 게이트 모듈과 2-input XOR 게이트 모듈을 사용하며 output으로 s와 co를 가지도록 구현하였다.

B. Full Adder 설계

Full adder는 그림2와 같이 1개의 XOR 게이트, 3개의 AND 게이트, 1개의 OR 게이트로 구현할 수 있다. Full adder의 진리표인 표4에서 SOP를 사용하여 co의 Boolean equation을 구하면 다음과 같이 나타낼 수 있다.

$$co = \bar{a}bci + a\bar{b}ci + ab\bar{ci} + abci = ab(ci + \bar{ci}) + ci(\bar{a}b + a\bar{b}) = ab + ci(a \oplus b)$$

Full adder의 carry out인 co의 식을 위의 식으로 사용하면, 2개의 XOR 게이트와 2개의 AND 게이트, 1개의 OR 게이트를 사용하여 full adder를 아래의 그림3과 같이 설계할 수 있다.

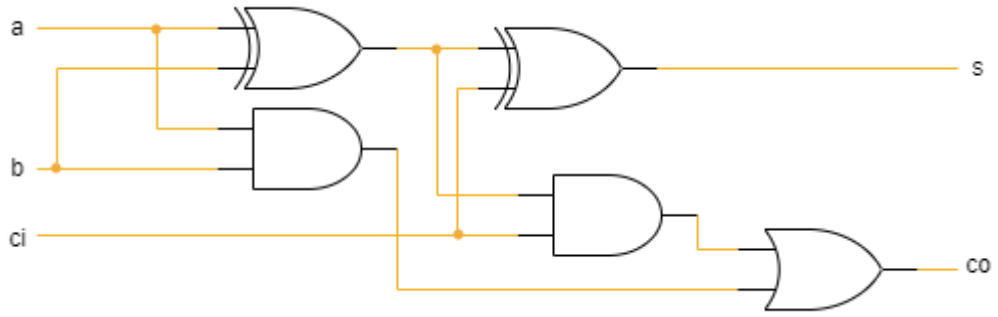

그림3. Full Adder design

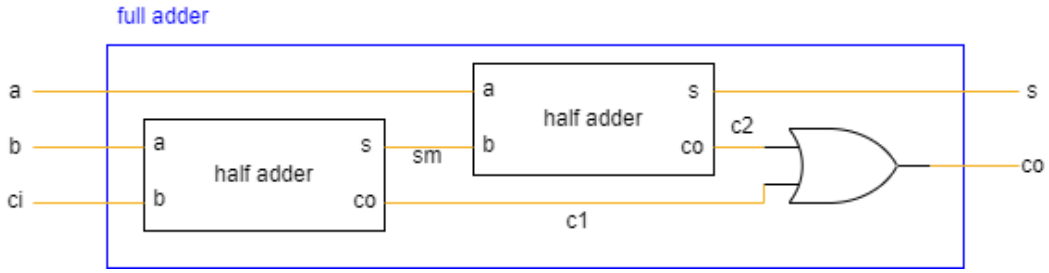

그림4. Full Adder design using half adder

또한, 각 XOR 게이트 1개와 AND게이트 1개는 1개의 half adder로 나타낼 수 있고, 이를 그림으로 표현하면 위의 그림4와 같다. 따라서 본 프로젝트에서 full adder는 input으로 a, b, ci를 가지고 2개의 half adder와 1개의 OR 게이트를 사용하여 연산하고, output의 값으로 s와 co를 가지도록 구현하였다.

C. 4-bit Ripple Carry Adder 설계

본 프로젝트에서 구현하고자 하는 4-bit Ripple Carry Adder는 4-bit의 a와 b, 1-bit의 ci를 input으로 입력받고 4개의 full adder를 사용하여 연산하며 그 결과값인 output으로 4-bit s와 1-bit co를 가진다. 또한, 3-bit wire로 선언된 c를 사용하여 full adder의 연산 결과로 얻은 carry out의 값을 저장하고 다음 full adder의 carry in으로 사용할 수 있도록 설계하였다. 설계한 4-bit Ripple Carry Adder를 간단히 나타내면 아래의 그림 5와 같다.

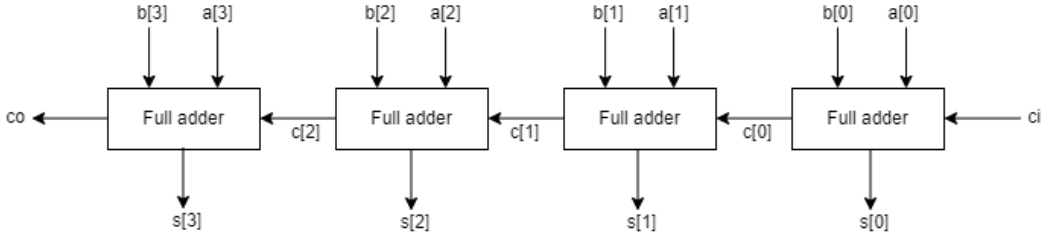

그림5. 4-bit Ripple Carry Adder

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

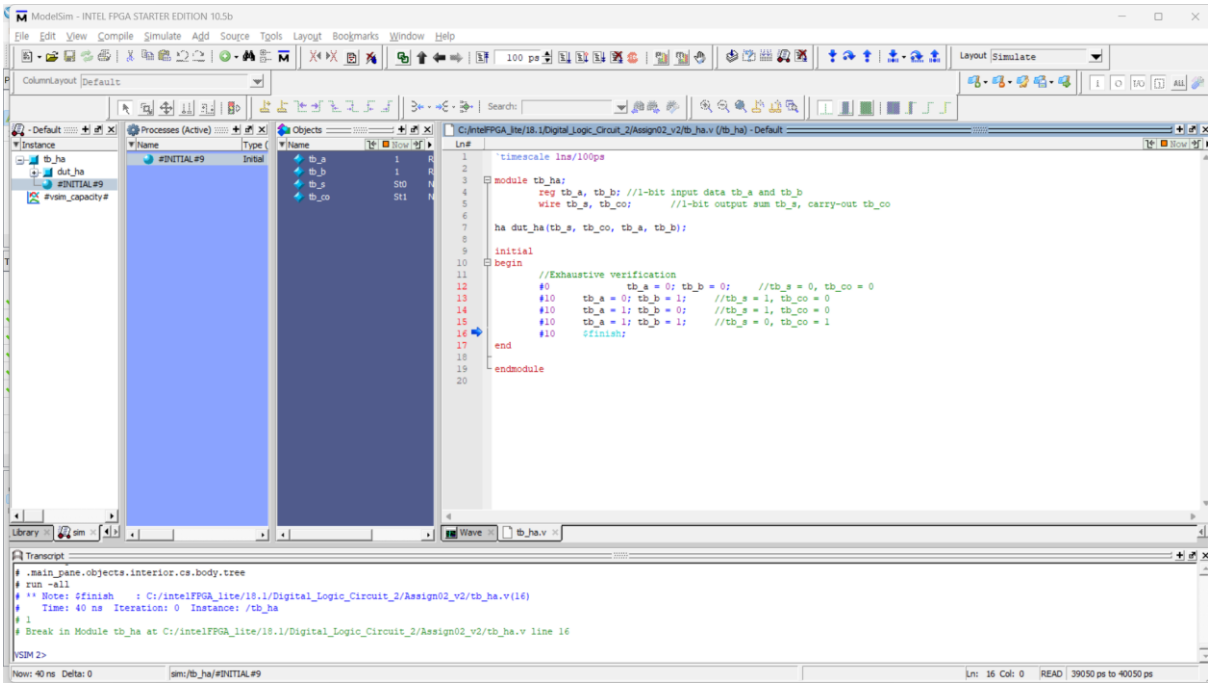

그림6. Half adder testbench

그림6은 half adder의 testbench Verilog 코드이다. Half adder는 입력값이 2개이므로 exhaustive verification을 사용하여 검증하였다.

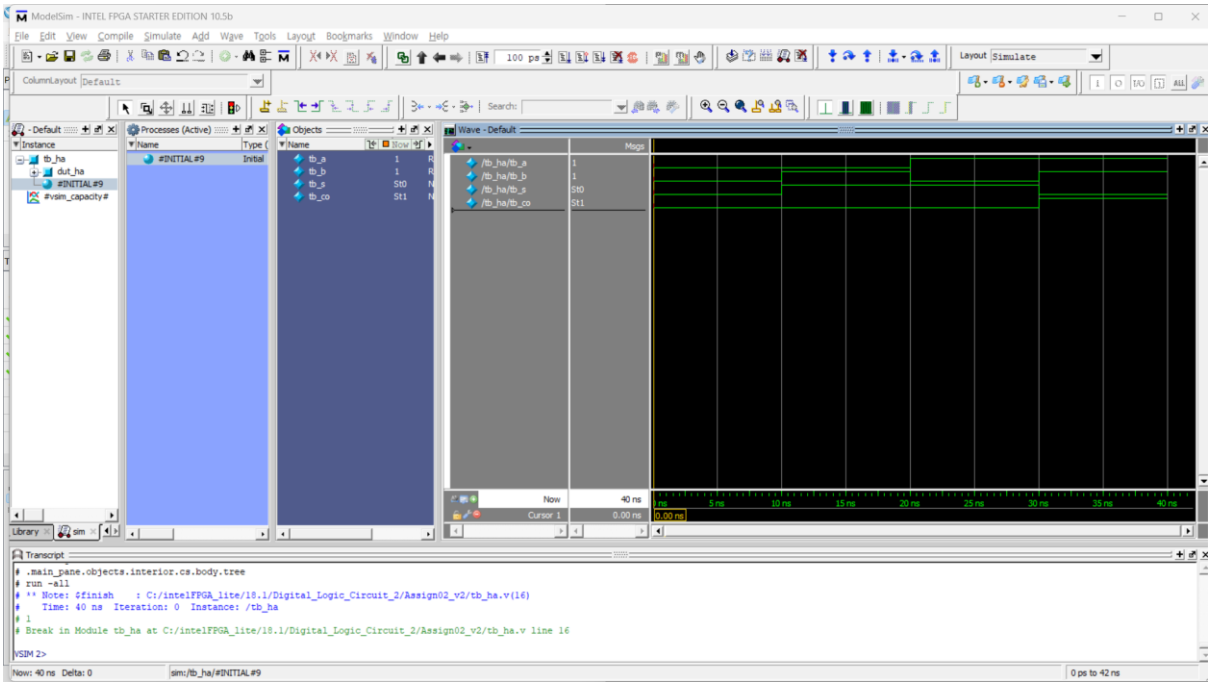

그림7. Half adder waveform

위 half adder의 waveform을 통해 tb_s와 tb_co의 결과값이 표1의 값과 동일하다는 것을 확인할 수 있다.

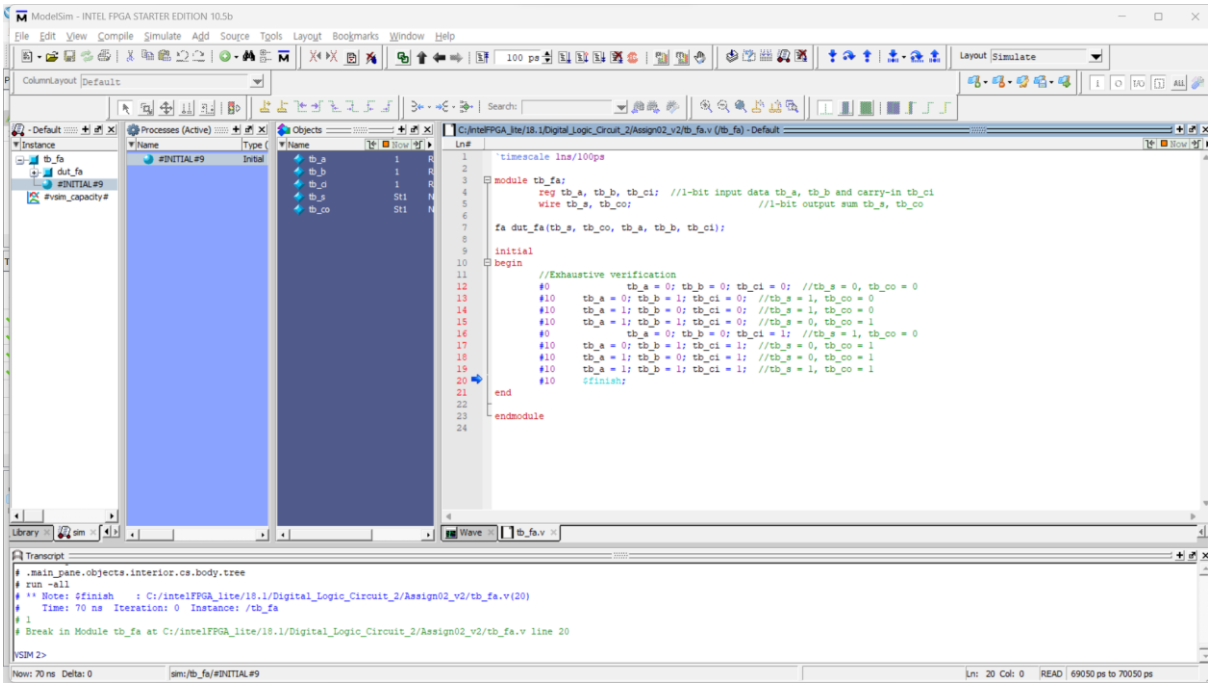

그림8. Full adder testbench

그림8은 full adder의 testbench Verilog 코드이다. Full adder는 입력값이 3개이므로 exhaustive verification을 사용하여 검증하였다.

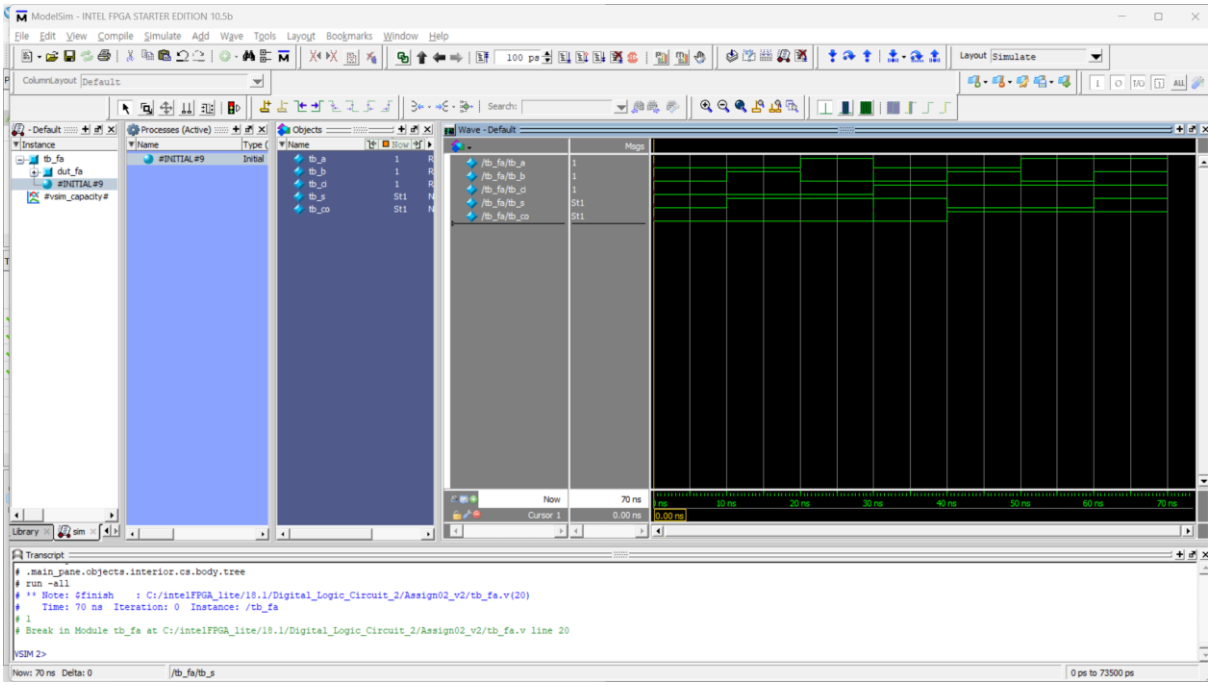

그림9. Full adder waveform

위 full adder의 waveform을 통해 tb_ci의 값에 따라 tb_s와 tb_co의 결과값이 표4의 값과 동일하다는 것을 확인할 수 있다.

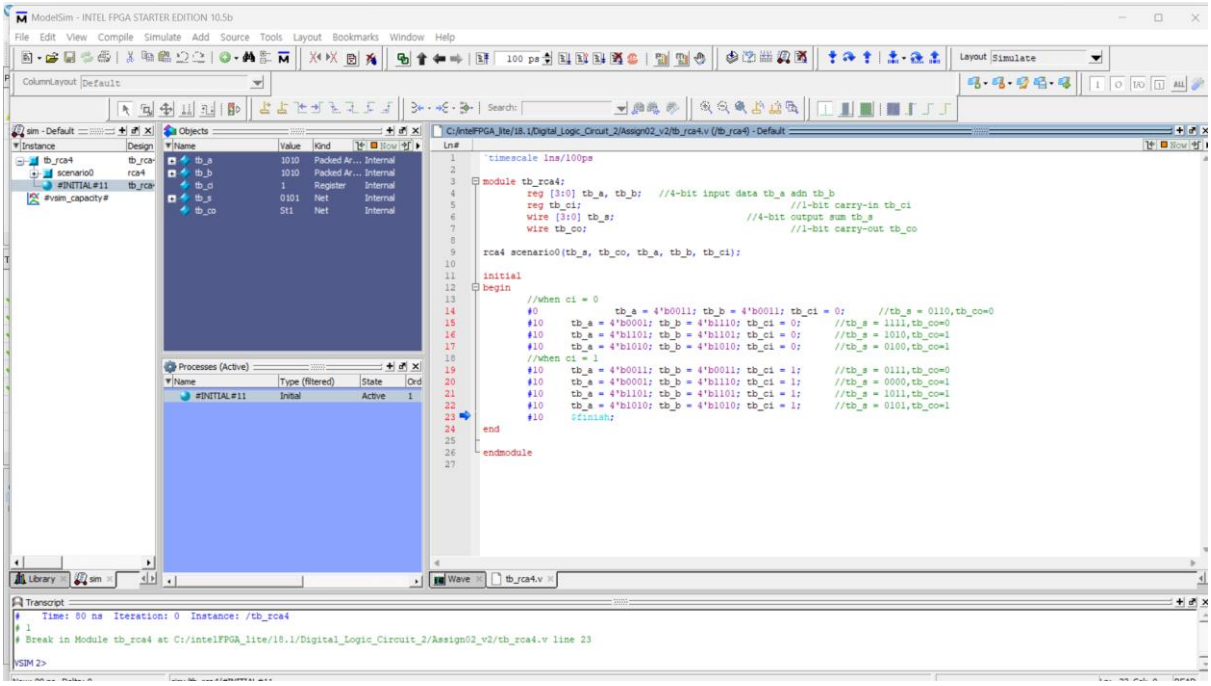

그림10. 4-bit Ripple Carry Adder testbench

4-bit RCA는 9개의 입력을 가지므로 모든 경우의 수를 대입할 수 없어 directed verification 방식을 사용하였다. Testbentch는 reg로 선언된 [3:0] bus tb_a, tb_b와 carry in인 tb_ci가 input이며, wire로 선언된 [3:0] bus tb_s와 carry out인 tb_co가 선언되었다. 그리고 rca4 module의 scenario0을 실행하였다. 위의 testbench는 총 8개의 case가 존재하는데, 그 중 상위 4개의 case는 ci=0일 때이며, carry in이 정상적으로 작동하는 지 확인하기 위해 하위 4개의 case는 tb_a와 tb_b의 값은 같지만, ci=1로 바꾸어 연산을 수행하였다. 그리고 각 연산의 결과값을 waveform으로 확인하기 위해 다음의 연산이 수행될 때마다 10ns의 delay를 주었다.

tb_a	tb_b	tb_ci=0		tb_ci=1	
		tb_co	tb_s	tb_co	tb_s
0011	0011	0	0110	0	0111
0001	1110	0	1111	1	0000
1101	1101	1	1010	1	1011
1010	1010	1	0100	1	0101

표7. 4-bit RCA testbench expected result

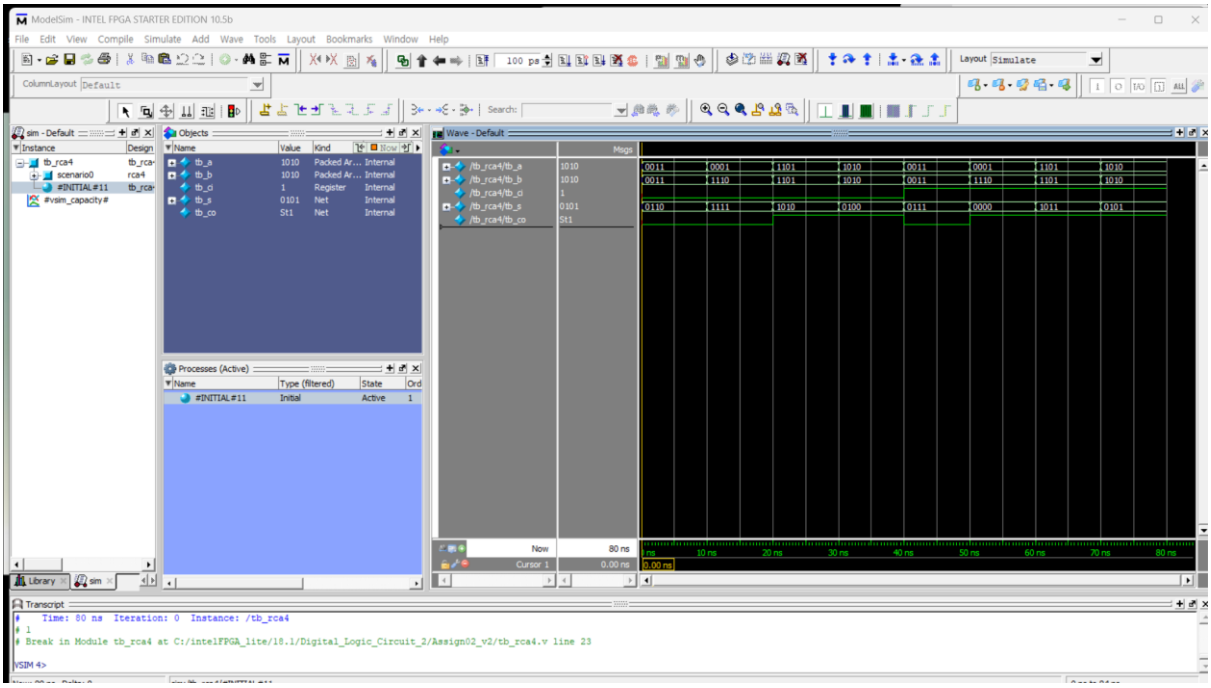

그림11. 4-bit Ripple Carry Adder waveform

Testbench는 크게 ci=0일 때와 ci=1일 때로 나눌 수 있다. 각 ci의 값에 대한 데이터 tb_a와 tb_b의 합 s는 위의 표7과 같다.

위의 expected result와 그림11의 결과값 tb_co, tb_s의 값이 같은 것을 확인할 수 있다. 따라서 rca4 module 내의 자리 올림수와 각 비트들의 연산이 제대로 이루어졌음을 알 수 있다. 또한, tb_ci=0에서 tb_ci=1이 되었을 때, tb_s의 LSB가 1씩 증가하거나 tb_co의 값이 1이 되는 것을 통해 ci=1가 제대로 입력되었으며 그에 따른 각 비트 연산과 올림수가 제대로 수행되었음을 알 수 있다.

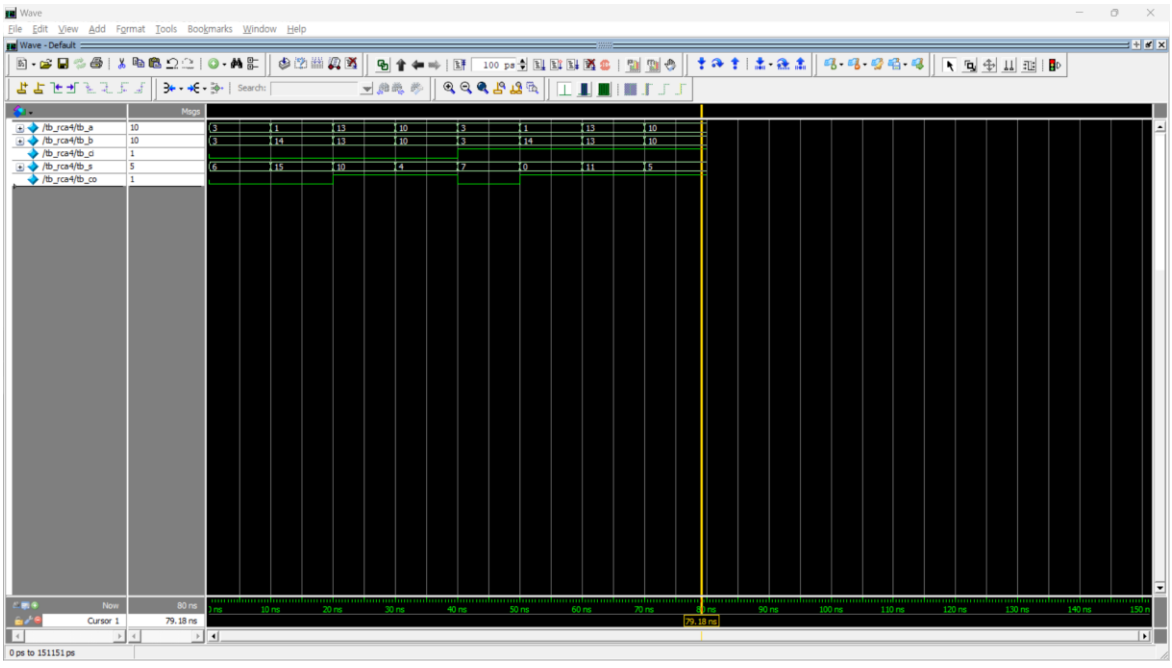

그림12. 4-bit Ripple Carry Adder waveform – unsigned

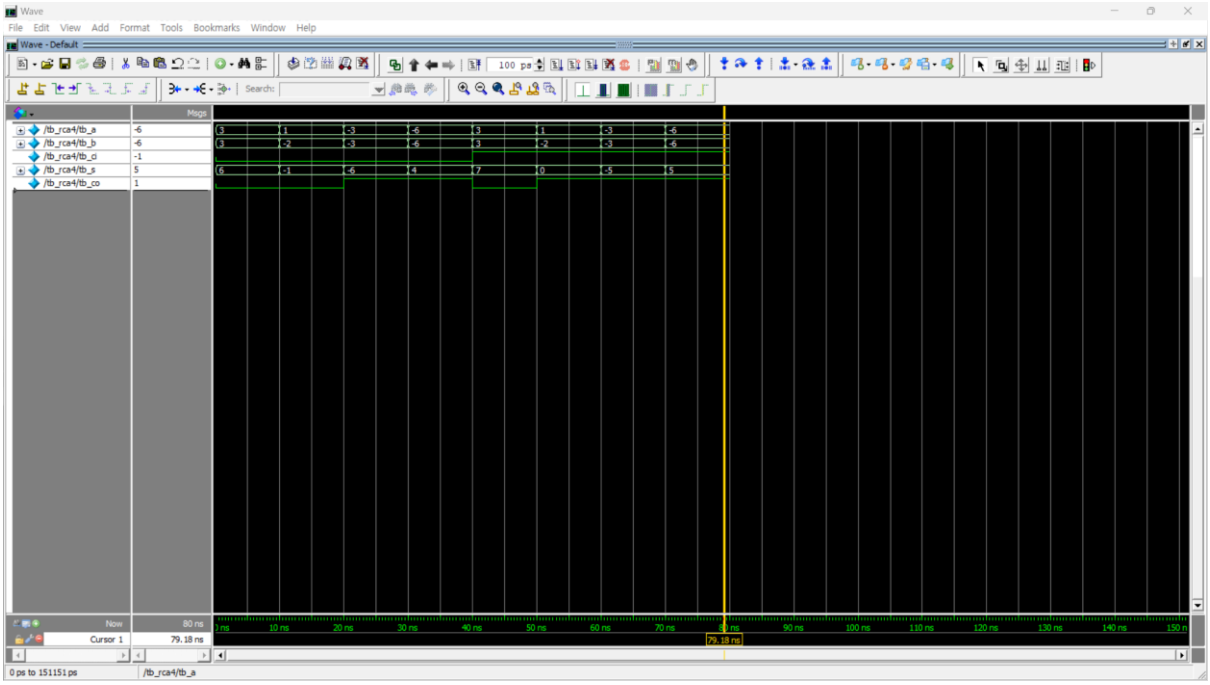

그림13. 4-bit Ripple Carry Adder waveform - decimal

위의 그림12와 그림13은 rca4 testbench에서 result의 radix를 각각 unsigned와 decimal로 변경한 결과 화면이다. 이때 unsigned는 부호를 표시하지 않는 것으로, 모든 값이 양수가 되고, decimal은 음수와 양수를 구분하며 MSB=0일 때 양수이고 MSB=1일 때 음수로 여긴다. 예를 들어, tb_ci=0, tb_a=0001, tb_b=1110일 때를 비교해보면, unsigned일 때, 10진수로 tb_ci=0, tb_a=1, tb_b=14의 값을 가진다. 그리고 세가지 값을 더한 값 tb_s=15가 되고 자리올림수가 발생하지 않으므로 tb_co=0이 된다. 다음으로 radix가 decimal일 때, 10진수로 tb_ci=0, tb_a=1이고 tb_b의 MSB=1이므로 음수의 값을 가지고, 1110에 2의 보수를 사용하면 0010이 되고 이는 10진수로 2가된다. 따라서 tb_b=-2의 값을 가진다. 그리고 결과값 tb_s=-1, co=0이된다. 이를 2진수로 표현한 값은 tb_s=1111, tb_co=0으로 2의 보수를 사용하여 10진수로 변환하면 각각 1, 0이 되고 tb_s의 MSB=1이므로 음수가 되어 -1인 것을 알 수 있다. 따라서 radix가 unsigned와 decimal일 때 그에 대한 결과값도 올바르다.

B. 합성(synthesis) 결과

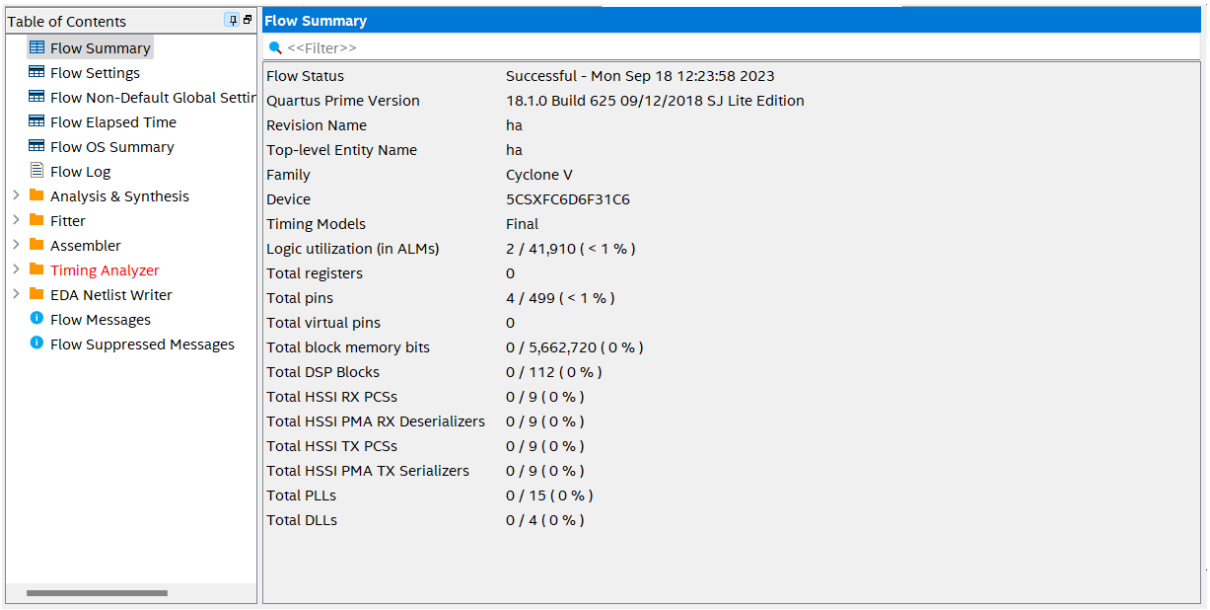

그림12. Half adder Flow Summary

위 그림은 half adder의 flow summary이다. 그림에서 확인할 수 있듯이, compilation은 성공적(successful)으로 수행되었고 logic utilization 은 2로 1% 미만이며, 사용된 total pins는 4로 1%미만 사용된 것을 확인할 수 있다.

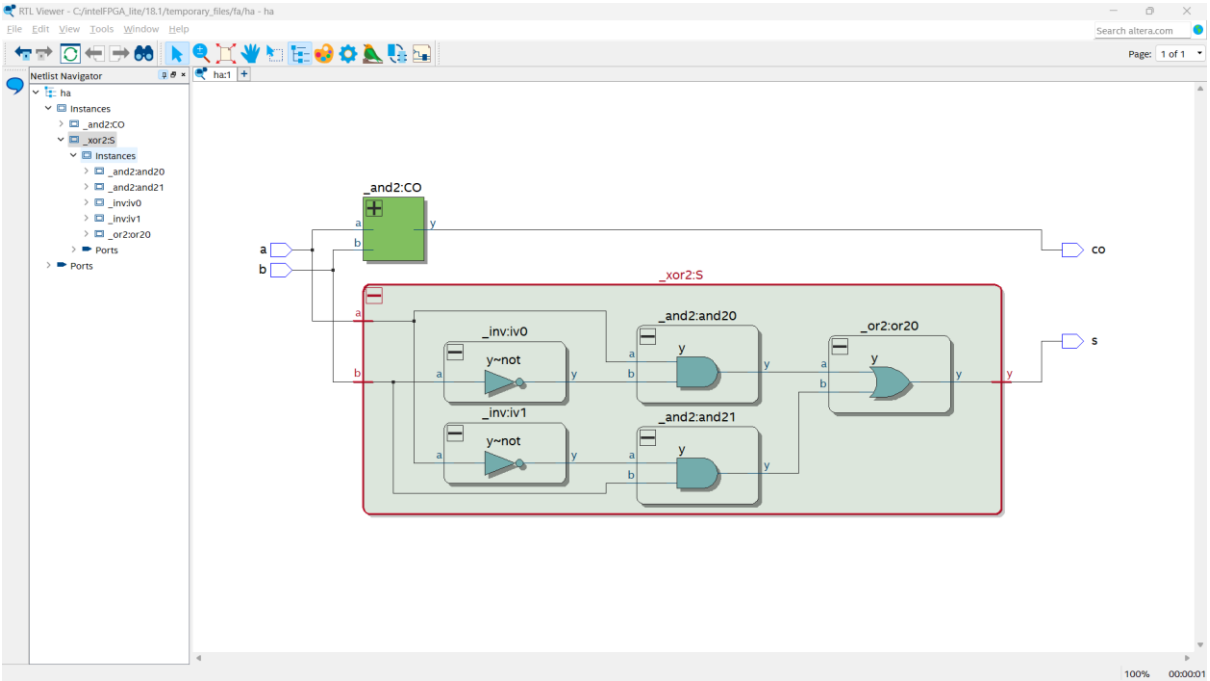

그림13. Half adder RTL Viewer

Compilation 이후에 확인할 수 있는 half adder의 RTL viewer는 위와 같다. Half adder는 1개의 2-input AND 게이트와 1개의 2-input XOR 게이트로 구현할 수 있으며, XOR 게이트는 2개의 NOT 게이트, 2개의 2-input AND 게이트, 1개의 2-input OR 게이트를 사용하

여 구조적으로 설계하였다. Output에서 carry out인 co는 a와 b의 AND 연산한 값이고, sum인 s는 a와 b의 XOR 연산한 값이라는 것을 확인할 수 있다.

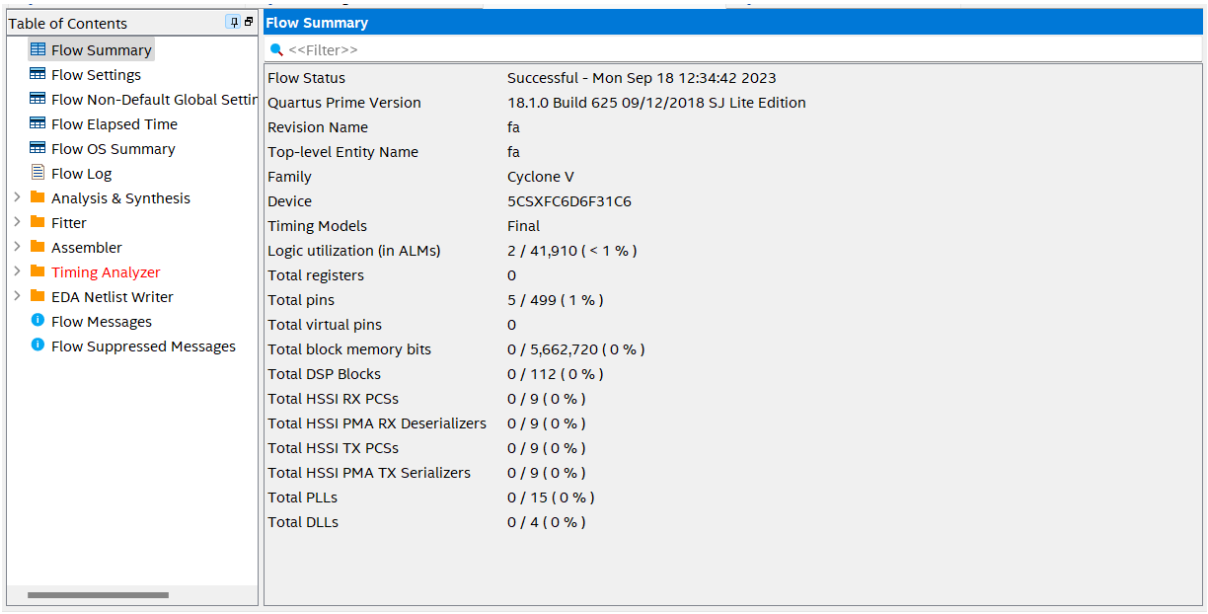

그림14. Full adder Flow Summary

위 그림은 full adder의 flow summary이다. 결과를 요약하면, compilation결과는 성공적(successful)이며, fa module에서 logic utilization은 2로 1%미만이고 total pins는 5개로 1% 정도 사용된 것을 확인할 수 있다.

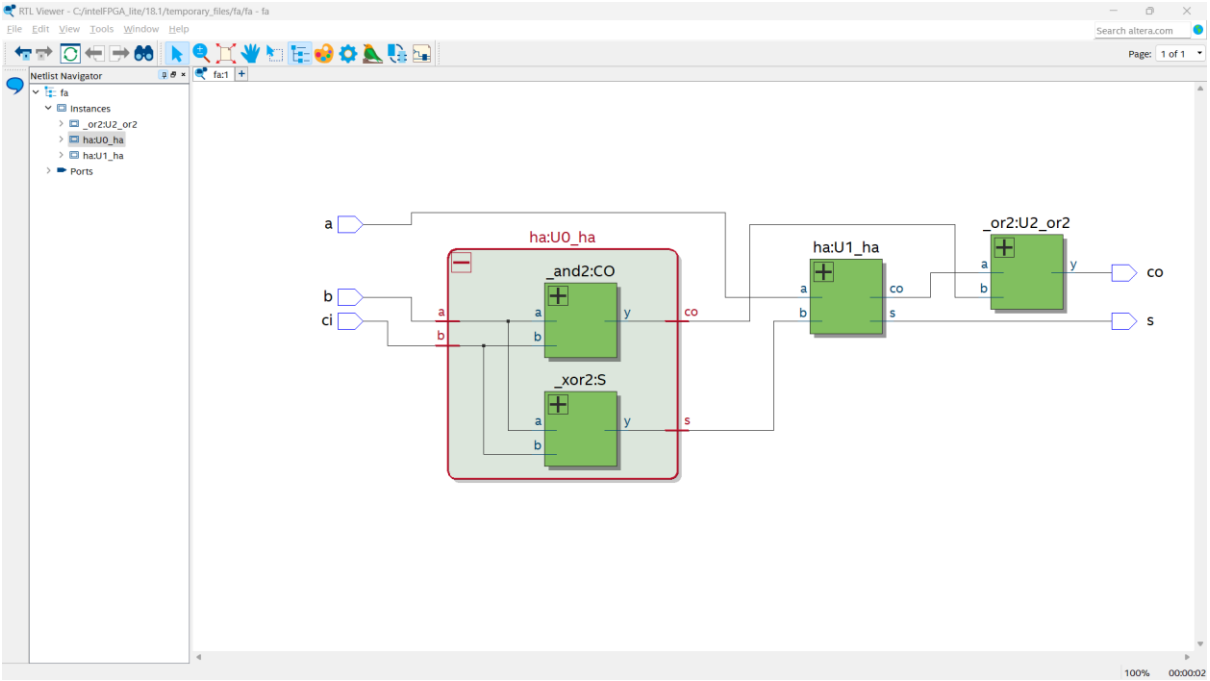

그림15. Full adder RTL Viewer

본 과제에서 full adder는 2개의 half adder와 1개의 2-input OR 게이트로 설계하였다. Full adder의 output에서 carry out인 co는 U0_ha의 instance의 co값인 b·ci값과 U0_ha의 s값인 b⊕ci와 a를 U1_ha에 입력하여 얻은 co값인 a(b⊕ci)를 OR 연산한 b·ci+a(b⊕ci)를

갖는다. 그리고 sum인 s는 U0_ha의 s의 값인 b⊕ci와 a를 U1_ha에 입력하여 얻은 s인 a⊕b⊕ci를 갖는다.

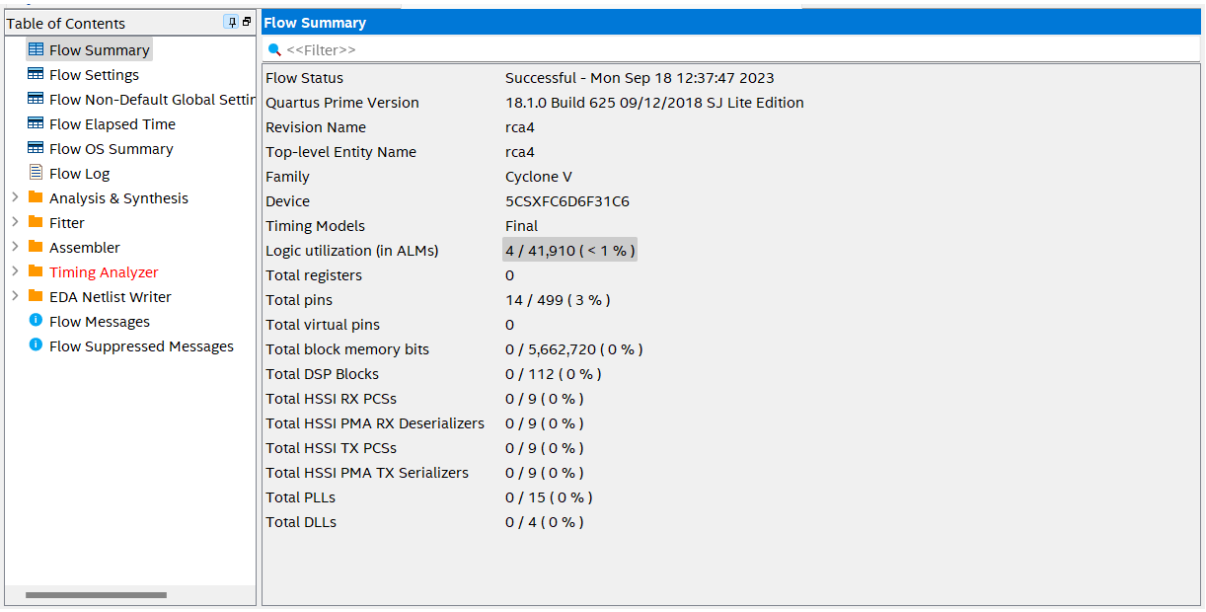

그림16. 4-bit Ripple Carry Adder Flow Summary

위 그림은 4-bit Ripple Carry Adder의 flow summary이다. 위에서 확인할 수 있듯이, rca4의 compilation 결과는 성공적(successful)이며, logic utilization은 4로 1%미만이고 total pins는 14로 3%정도 사용된 것을 확인할 수 있다.

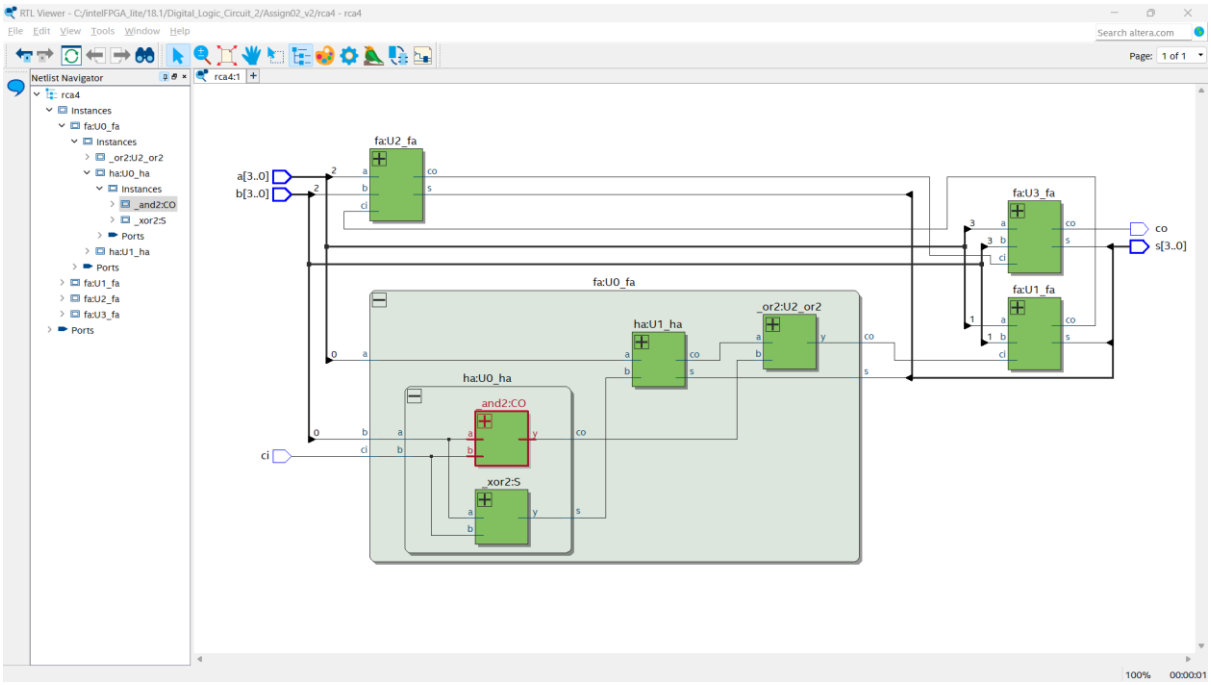

그림17. 4-bit Ripple Carry Adder RTL Viewer

본 과제에서 4-bit RCA는 4개의 full adder로 설계하였다. 또한 각 full adder는 2개의 half adder와 1개의 2-input OR 게이트로 구성되었고, half adder는 1개의 2-input AND 게이트와 1개의 2-input XOR 게이트로 구성된 것을 확인할 수 있다.

5. 고찰 및 결론

A. 고찰

4-bit RCA의 testbench를 생성하는 데 어려움을 겪었다. RCA의 testbench는 이전의 full adder에서 다음의 full adder에 입력되는 carry-in이 제대로 수행이 되는지 확인할 수 있어야 하고, 또한 carry-in의 값이 0일 때와 1일 때 두 가지 case를 증명할 수 있는 조합을 찾아야 했기 때문이다. 따라서 제일 큰 분류로 carry-in의 값이 0 또는 1일 때로 나눈 다음에, 입력 data tb_a와 tb_b의 값을 고정하고 그 내에서 carry-in의 값에 따라 ouput이 달라지는 지 확인하여 full adder 간의 carry-in과 carry-out이 제대로 수행되는 지 확인할 수 있었고, 최종적으로 4-bit RCA의 carry-in이 입력되었을 때도 연산이 제대로 수행되는 것을 확인할 수 있었다.

B. 결론

Half adder는 1-bit 2개의 data a와 b를 입력받고 a와 b에 대한 합 s와 올림수 co를 계산하여 output으로 갖는다. 또한, full adder는 1개의 XOR 게이트, 1개의 OR 게이트, 3개의 AND 게이트로 구현할 수 있고, 또는 2개의 half adder와 1개의 OR 게이트를 사용하는 두 가지 방법으로 설계할 수 있다. 본 과제에서 구현한 4-bit Ripple Carry Adder는 4개의 full adder를 연결하여 구현할 수 있다. 따라서 이를 통해 Ripple Carry Adder는 비트 수의 개수만큼 full adder를 연결하여 구현할 수 있다는 것을 깨달았다.

위의 결론을 통해 32-bit RCA를 설계하면, 32개의 full adder를 직렬로 연결하여 구현할 수 있고 따라서 4-bit RCA를 인스턴스화하여 8개를 사용해 구현할 수 있다. 하나의 4-bit RCA가 4개의 비트를 연산하고, 그에 대한 sum와 carry out을 그 다음 RCA의 input으로 입력되어 32비트를 더하는 가산기를 생성할 수 있다.

6. 참고문헌

[1] 전가산기, 반가산기 / https://m.blog.naver.com/deepb1ue/221235465857

[2] 가산기 / https://ko.wikipedia.org/wiki/%EA%B0%80%EC%82%B0%EA%B8%B0

[3] 2의 보수 / https://life-with-coding.tistory.com/298

[4] [Verilog] signed, unsigned 사칙연산 / https://trts1004.tistory.com/12109182