컴퓨터 공학 기초 실험2 보고서

실험제목: Ripple-Carry Adder (RCA)

실험일자: 2022년 09월 20일 (화)

제출일자: 2022년 09월 21일 (수)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2021202058

성 명: 송채영

1. 제목 및 목적

A. 제목

Ripple-Carry Adder (RCA)

B. 목적

가산기 half-adder와 full-adder의 동작원리에 대해 알아보고, 직접 설계해본다. 또한 이를 이용하여 두 개의 수를 더하는 4-bit ripple-carry-adder을 구현해본다. 또한 2의 보수의 원리에 대해 알아본다.

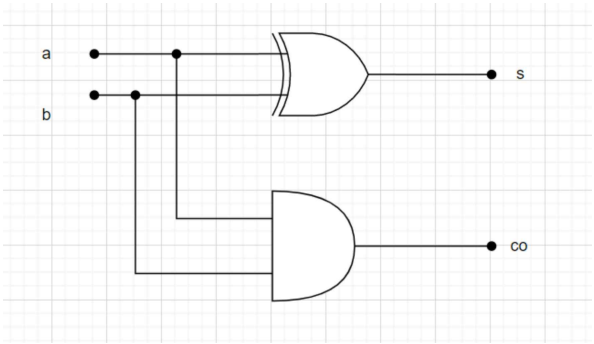
2. 원리(배경지식)

2의 보수 원리

컴퓨터가 음수와 양수를 구별하기 위한 방법에는 부호 절댓값 방식, 1의 보수 방식, 2의 보수 방식 3가지가 있다. 먼저 첫 번째, 부호 절댓값 방식으로 최상위 비트 MSB(Most Significiant Bit)를 사용하는 것인데, 1이면 음수, 0이면 양수로 표현한다. 이 방법은 0이 양수로도, 음수로도 표현 될 수 있다는 점에서 단점이 있다. 두 번째, 1의 보수 방식은 0을 1로, 1을 0으로 바꾼 수이다. 하지만 부호 절댓값 방식과 마찬가지로 0이 음수로도 양수로도 표현될 수 있어 캐리가 발생하면 최하위 비트 LSB(Least significiant Bit)에 1을 더해주어야 하므로 마지막 방법인 2의 보수 방식이 생겼다. 컴퓨터에서 음수를 표현하기 위해서는 주어진 비트에서 양수, 음수를 모두 표현할 수 있으며 두 합이 0이 되어야 하므로 2의 보수의 방식을 사용한다. 2의 보수 방식은 1의 보수에 1을 더해주면서 위의 방식들의 문제를 해결하였으며, carry가 발생했을 때 최하위 비트인 LSB에 1을 더해주지 않아도 계산이 성립된다. 2의 보수는 0을 1로, 1을 0으로 바꾸고 1을 더해주며 사용하면 된다.

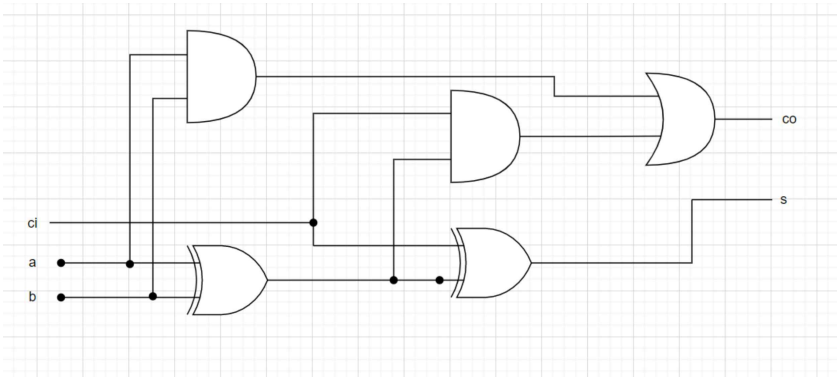
가산기(Adder)는 더하기 연산을 수행하는 논리회로로, 반 가산기(Half Adder)와 전 가산기(Full Adder)로 나뉜다.

아래의 그림은 Half Adder의 논리 회로이다.



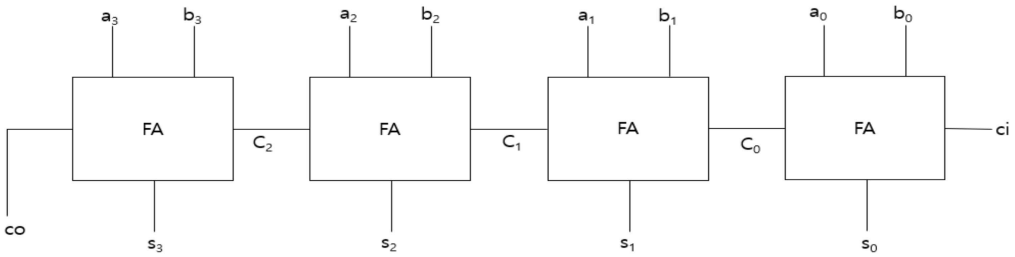
Half Adder는 2개의 1 bit input 입력을 받아 Sum과 Carry out을 구하는 가산기이며, XOR 게이트와 AND 게이트로 구현할 수 있다. 또한 Half Adder는 2진수 한 자리 덧셈을 하므로 아랫자리에서 발생한 carry를 고려하지 않는다. 따라서 2bit이상의 2진수 덧셈을 할 수 없다. 이러한 캐리를 고려해서 만든 회로가 full adder 회로이다.

아래의 그림은 Full adder의 논리 회로이다.



Full adder는 2개의 1-bit 입력과 1-bit carry in을 입력받아 sum과 carry out을 출력하는 가산기이며, 2개의 반가산기와 OR 게이트로 구현할 수 있다.

아래의 그림은 4개의 full adder을 연결해서 만든 4-bit ripple-carry-adder 회로이다.



여러개의 전 가산기를 일렬로 연결해 구성한 논리회로로, 물결치듯이 다음 가산기로 넘어간다고 해서 붙여진 이름이다. N-bit ripple carry adder 는 n-bit 를 가지는 두 개의 수를 더하기 위한 가장 간단한 형태의 가산기이다. Ripple carry adder는 더하려는 수의 bit 개수만큼 full adder를 연결하여 구현한다.

3. 설계 세부사항

half adder의 truth table을 살펴보면 다음과 같다.

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

진리표를 바탕으로 Carry out의 카르노맵을 살펴보면 다음과 같다.

a \ b	0	1
0	0	0
1	0	1

Carry out의 부울식은, Carry out co = AB 이다.

진리표를 바탕으로 Sum의 카르노맵을 살펴보면 다음과 같다.

a \ b	0	1
0	0	1
1	1	0

Sum의 부울식은, Sum s =A'B+AB'=A⊕B 이다.

Half Adder은 2개의 1비트 입력을 받아 sum과 carry out을 구해야 하므로 input으로 a, b를 output 으로 s, co를 받았다. a,b를 gates.v의 _xor2에서 instance해 s(sum)값을 얻게 했고, _and2 역시 gates.v에서 instance해, co(carry out)값을 얻게 구현했다.

full adder의 truth table을 살펴보면 다음과 같다.

Input			Output	
ci	a	b	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

진리표를 바탕으로 Carry out의 카르노맵을 살펴보면 다음과 같다.

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

Carry out의 부울식은,

Carry out c0 = $\overline{c_{in}}ab + c_{in}ab + c_{in}\bar{a}b + c_{in}ab + c_{in}a\bar{b} + c_{in}ab$

$= ab(c_{in} + \overline{c_{in}}) + c_{in}b(a + \bar{a}) + c_{in}(b + \bar{b})$

$= ab + c_{in}a + c_{in}b$ 이다.

진리표를 바탕으로 Sum의 카르노맵을 살펴보면 다음과 같다.

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

1	1	0	1	0

Sum의 부울식은, $\text{Sum s} = c_{in}\bar{a}\bar{b} + \overline{c_{in}}a\bar{b} + c_{in}ab + \overline{c_{in}}\bar{a}b$

$= \overline{c_{in}}(a\bar{b} + \bar{a}b) + c_{in}(\bar{a}\bar{b} + ab)$

$= \overline{c_{in}}(a \oplus b) + c_{in}(a \odot b)$

$= \overline{c_{in}}(a \oplus b) + c_{in}\overline{(a \oplus b)}$

$= c_{in} \oplus (a \oplus b)$

$= c_{in} \oplus a \oplus b$ 이다.

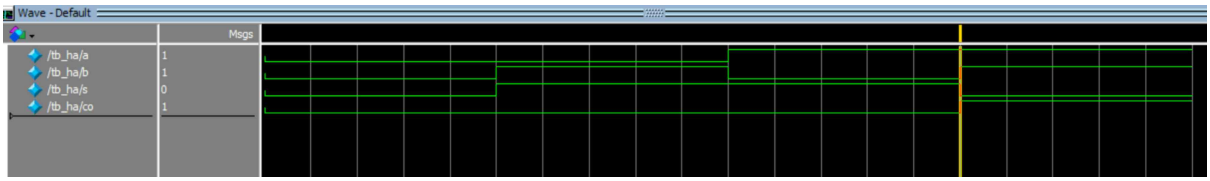
Full Adder은 2개의 1비트 입력과 1비트 carry in 입력을 받아 sum과 carry out을 구해야 하므로 input에 a, b와 carry in인 ci를 output에 s, co를 받았다. ha.v로 부터 half adder을 instance해 sm, c1, c2값을 얻도록 했다. gates.v에서 _or2를 instance하여 맨 처음 Half Adder로 얻은 c1과 c2를 집어넣고 carry out값인 co를 출력하도록 구현했다.

Ripple-Carry Adder은 더하려는 수의 bit 개수만큼 Full Adder을 연결해야 하므로, input으로 4bit를 받았기 때문에, 4개의 Full Adder을 instance하고 RCA에 넣어 구현했다. a와 b, s는 각각 4-bit의 값을 가지며 carry in을 의미하는 ci와 carry out을 의미하는 co는 1-bit의 데이터를 저장하므로 총 9개의 input을 가진다. 또한 총 4개의 full adder가 필요하므로 U0_fa ~ U3_fa로 구분했다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

- Half Adder

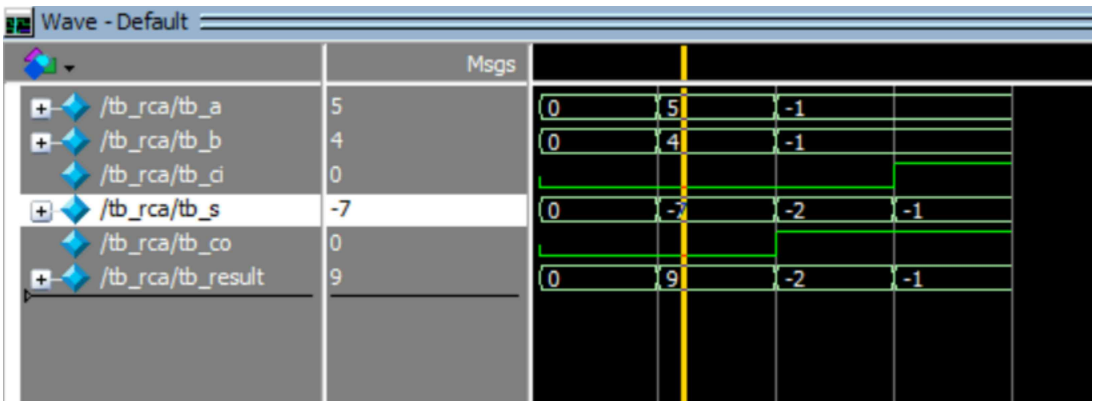


Half Adder의 timescale을 1ns/100ps로, 모듈의 이름은 tb_ha로 설정하였다. waveform을 통해 위의 진리표와 동일한 결과임을 알 수 있다. s는 a가 0, b가 1일때, a가 1, b가 0일 때 1이 나오고 a와 b가 모두 1인 경우 carry가 발생해 co의 값이 1이 되는 것을 확인 확인했다.

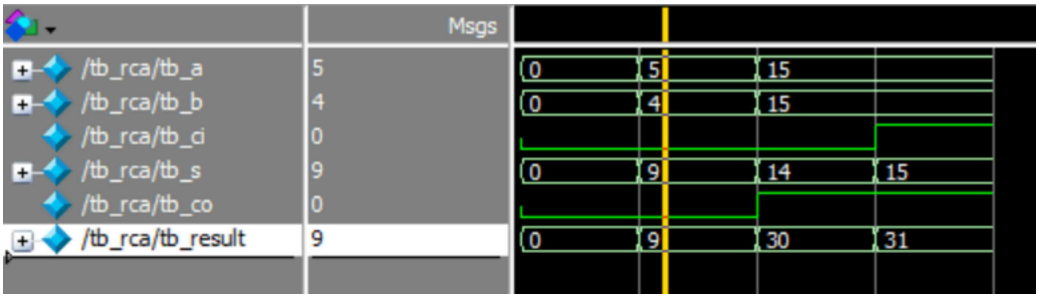
- Full Adder

Full Adder의 timescale을 1ns/100ps로, 모듈의 이름은 tb_fa로 설정하였다. waveform을 통해 위의 진리표와 동일한 결과임을 알 수 있다. a, b, ci의 값 중 1이 홀수개 일 때 s가 1이 되고 co는 a, b ,ci의 값 중 1의 개수가 2개 이상일 때 1이 된다.

- 4-bit RCA



decimal results



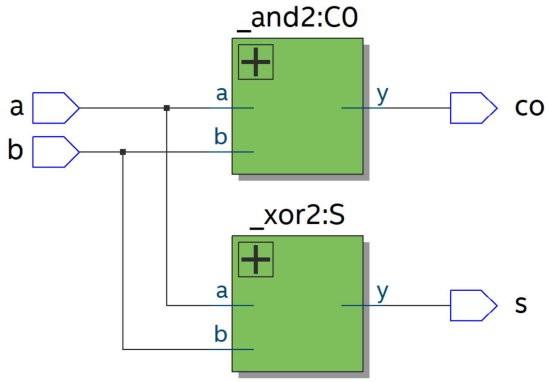
unsigned results

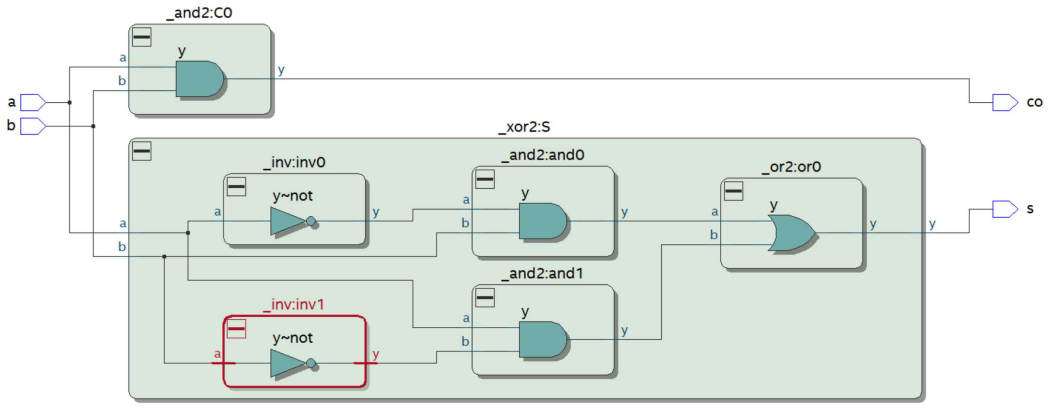
timescale을 1ns/100ps로, 모듈의 이름은 tb_rca로 설정하였다. a와 b, s는 각각 4-bit의 값을 가지며 carry in을 의미하는 ci와 carry out을 의미하는 co는 1-bit의 데이터를 저장하므로 총 9개의 input을 가진다. input이 9개로, 모든 경우의 수를 test해야 하므로 몇 개의 경우만 작성하였다. 자료의 rca testbench를 참고하여 같은 값이 나오게끔 하였다. 예를 들어 이진수 0000을 입력시 4'b0000라고 쓰는데, 4비트를 사용하여 binary로 표현한다는 뜻이다.

4-bits RCA에서 radix를 decimal과 unsigned로 바꾸어 결과를 비교해 보았다. decimal의 경우 2의 보수와 똑같이 계산되고, unsigned의 경우 부호 bit가 없는 무조건 양수의 값이 나왔다. decimal과 unsigned일 때의 값을 비교하면 설계가 잘 되었음을 확인했다.

B. 합성(synthesis) 결과

- Half Adder





위의 사진은 Half Adder의 RTL viewer이다. AND gate에서 계산된 값은 carry out co로, XOR gate에서 계산된 값은 sum s로 나간 것을 확인했다.

Flow Summary

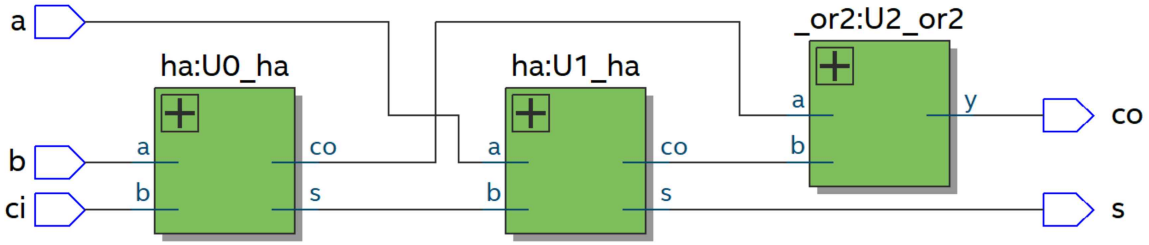
<<Filter>>

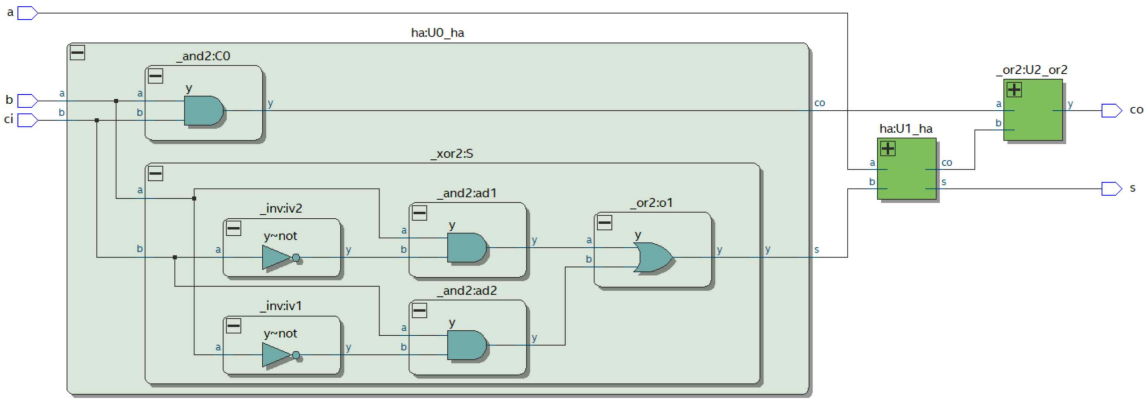
Flow Status	Successful - Tue Sep 20 00:20:13 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	ha
Top-level Entity Name	ha
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	4
Total virtual pins	0
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

위의 사진은 Half Adder의 flow summary이다. Input 2개와 output 2개로 총 4개의 pin을

사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- Full Adder





위의 사진은 Full Adder의 RTL viewer이다. b와 ci가 첫 번째 Half Adder에서 sm값을 얻었고, sm값과 a가 두 번째 Half Adder를 통해 s와 c2를 출력했다. c1값과 c2값을 OR gate를 통해 carry out으로 출력하는 것을 확인했다.

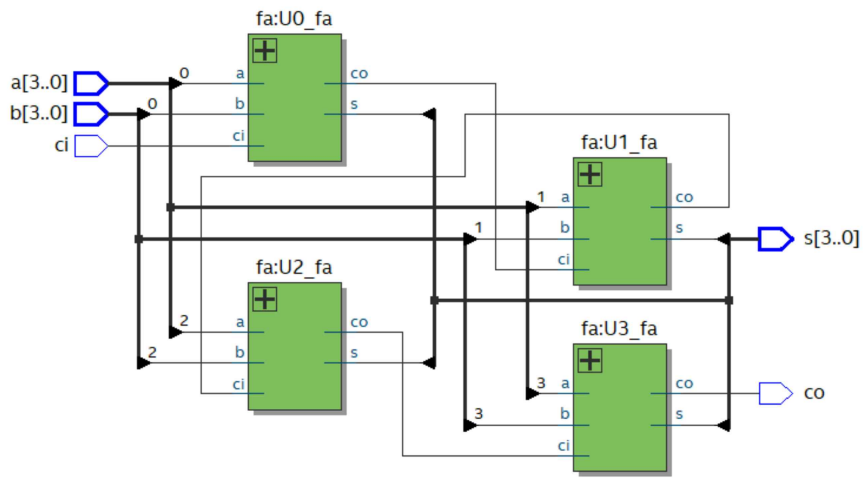
Flow Summary

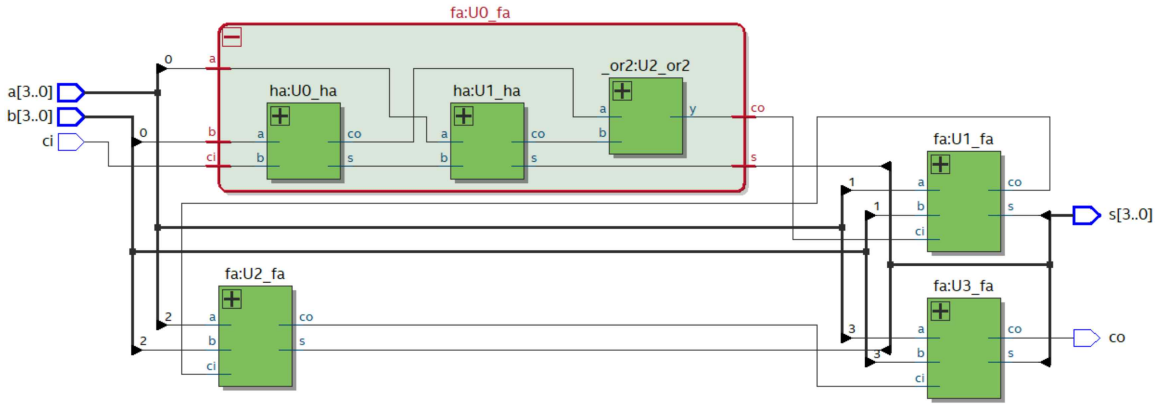
<<Filter>>

Flow Status	Successful - Tue Sep 20 00:53:30 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	fa
Top-level Entity Name	fa
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	5
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

위의 사진은 Full Adder의 flow summary이다. Input 3개와 output 2개로 총 5개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

- 4-bit RCA





위의 사진은 4-bit RCA의 RTL viewer이다. 4 bit의 a, b를 계산해야 하므로 Full Adder를 4개를 넣어 구현하였다. (첫 번째 그림) 처음 a[0]과 b[0]와 ci을 입력해 s[0]와 c[0]를 출력, 다음 연산에서 c[0]과 a[1], b[1]을 입력해 s[1]과 c[1]을 출력, 다음 연산에서 a[2]와 b[2], c[1]를 입력해 s[2]와 c[2]를 출력, 다음 연산에서 a[3], b[3], c[2]를 입력해 s[3]와 co을 출력하는 것을 확인했다.

Flow Summary

<<Filter>>

Flow Status	Successful - Tue Sep 20 09:30:37 2022
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca4
Top-level Entity Name	rca4
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

위의 사진은 4-bit RCA의 flow summary이다. 4 bit a, b, s와 1-bit ci, co로 총 14개의 pin을 사용한 것을 알 수 있으며 성공적으로 컴파일이 완료된 것을 알 수 있다.

5. 고찰 및 결론

A. 고찰

4-bit RCA의 testbench를 짜고 waveform을 확인하기 위해 실행하였을 때 계속된 오류가 떴었다. rca4로 모듈했기 때문에 testbench에서도 rca4 tb_rca4라고 해줬어야 했는데 rca로 해둬 당연한 결과였다. 이번 오류를 통해 사소한 실수로도 오류로 이어지기 때문에, 확인을 잘 해야겠다는 생각이 들었다. 또한 프로젝트의 이름과 파일의 이름을 설정하는 것이 아직 헷갈리는 것 같다. 이 부분을 수정하기 위해 여러번 삭제하고 다시 만드는 과정을 거친 것 같다. 특히 RCA부분이 가장 헷갈렸다. 두 번째 실습임에도 불구하고 많이 버벅거리고 많은 시간이 들었던 것 같다. 이번 실험을 통해 더욱 verilog를 만져보고 익숙해졌던 것 같다.

B. 결론

저번 실험에서는 000부터 111까지의 값을 다 넣어주어 확인하는 방법을 썼었는데 이번 실험에서는 4'b0000과 같은 방법을 알게 되었다. 외에도 4'hf로도 표현할 수 있는데 h는 16진수인 hexadecimal number을 뜻하며, F는 십진수 15를 의미한다.

4-bit RCA를 이용하여 32-bit RCA 설계하기 위해, 4-bit RCA에서 Full Adder을 4개 사용했던 것 처럼 32-bit RCA의 경우 Full Adder을 8개 사용하여 Full Adder를 32개 사용하는 것과 같게 만든다. 이를 통해 32 bit 연산을 가능하게 만든다.

6. 참고문헌

공영호 교수님/디지털논리회로2 강의자료/광운대학교컴퓨터 공학과 2022 강의자료

2의 보수

/https://terms.naver.com/entry.nhn?docId=2835898&cid=40942&categoryId=32828

half adder, full adder / https://woodforest.tistory.com/122