컴퓨터 공학 기초 실험2 보고서

실험제목: Ripple Carry Adder

실험일자: 2023년 09월 18일 (월)

제출일자: 2023년 09월 27일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 월요일 0, 1, 2

학 번: 20222020264

성 명: 최봉규

1. 제목 및 목적

A. 제목

Ripple Carry Adder

B. 목적

이번 실습은 Half Adder, Full Adder에 대해 학습한다. Half Adder와 Full Adder를 바탕으로 Ripple Carry Adder를 구현한다. Ripple Carry Adder를 테스트하면서 0과 1이 아닌 2bit가 넘는 수를 저장하는 방법에 대해 학습한다.

2. 원리(배경지식)

2의 보수는 이진수를 표기하는 데 MSB를 sign표시로 빼고 나머지를 이용한다. 부호를 바꾸게 될 시 각 bit를 반대로 만들고 LSB에 1을 더하게 되면 그 수의 음수를 만들 수 있게 된다. 아니면 그 수보다 1bit가 크고 LSB = 1이며 LSB를 제외한 나머지 bit가 0일 때 해당 수를 이 수에서 빼게 되면 그 수의 반대 부호의 수를 이진수로 표현이 가능하다.

1의 보수는 각 bit 수를 반대로 하는 데에 그쳐 그 bit수 –1의 크기만큼 양수와 음수를 표현할 수 있는데, 이렇게 되면 0또한 양수인 경우와 음수인 경우가 나오게 되면 컴퓨터의 계산에 지장이 가게 된다. 먼저 앞에서 부호를 판별하지만 그 수가 -0인지 +0인지를 또 구분하게 되니 비효율적으로 된다. 2의 보수는 이런 상황을 방지하고자 0을 양수의 0만 남겨두고 LSM에 1을 더하게 되면서 n bit일 때 수의 범위가 [- 2^(n-1), 2^(n-1) –1] 1의 보수보다 음수 쪽에서 하나의 수를 더 표현이 가능하다는 이점이 생긴다.

이번 실습은 half adder와 full adder를 학습하고 ripple carry adder를 만드는 것이었다. Half adder의 경우 올림이 없어 full adder와 달리 Cin(Carry in)이 없다. 이전 bit에서 받을 수는 없으나 두 비트를 더하여 결과값을 내는 S와 다음 비트로 올릴 수 있는 Cout(Carry out)은 있다. 이제 다수의 bit에서의 덧셈은 rca를 이용하여 표현이 가능해진다. rca에서는 LSB 쪽을 연산할 때는 half adder로 하고 나머지 bit에 대한 덧셈을 full adder를 하거나 full adder n bit로 연속해서 만들되 Cin의 값을 0으로 넘겨 가능하다. 두 수에 대한 덧셈이 unsigned면 2^n - 1까지 표현이 가능하고 decimal이면 2^(n-1) - 1까지 표현이 가능해진다.

1bit가 아닌 여러비트의 값을 입력할 때 multi-bits 또는 bus를 이용한다. 예시는 다음과 같다. Input [3:0] a, b; 와 wire[2:0] c; 같이 사용한다. 만약 전자의 예시처럼 bus를 이용해서 사용했다면 b는 별도의 표시 없이 a처럼 4bit의 bus를 가지게 된다. c/c++ 처럼 배

열을 사용할 때처럼 변수마다 따로 표시하는 수고스러움을 덜할 수 있다.

2 input xor gate를 구문할 때에는 '^' 식을 이용하여 assign을 할 수 있지만, Boolean equation을 이용해서 다른 방식으로도 구할 수 있다. 이를 구하기 위 2 input AND gate 2개와 invertor 2개 2 input OR gate 1개가 필요하다. A⊕B = A!B + !AB 로 이번 실습 시간에 이 관계를 이용하여 xor을 구현이 가능하다.

3. 설계 세부사항

Half adder의 input output

Input		**Output**	
A	B	Cout	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half adder의 Truth Table

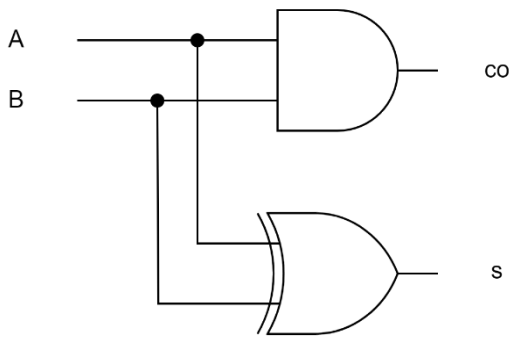
A \ B	0	1
0	0	0
1	0	1

Carry out Cout = AB

A \ B	0	1
0	0	1
1	1	1

Sum s = A!B + !AB = A⊕B

Logic gate



Full adder의 input output

Input			Output	
Cin	A	B	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full adder의 Truth Table

Cin \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

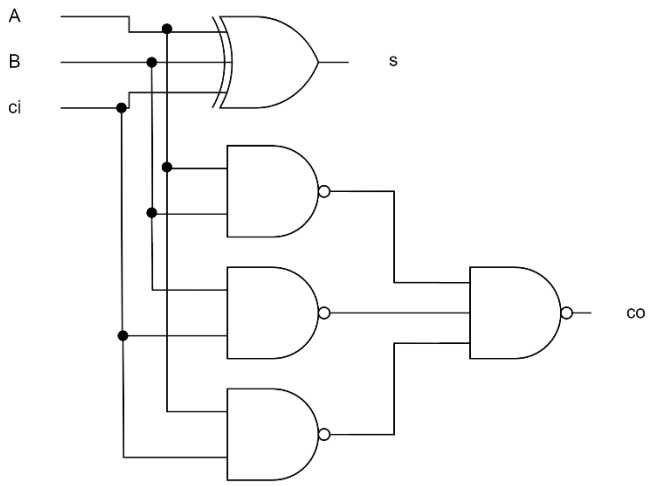
Carry out Cout = !ABCin + A!BCin + AB!Cin +ABCin = AB + BCin + CinA

Cin \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

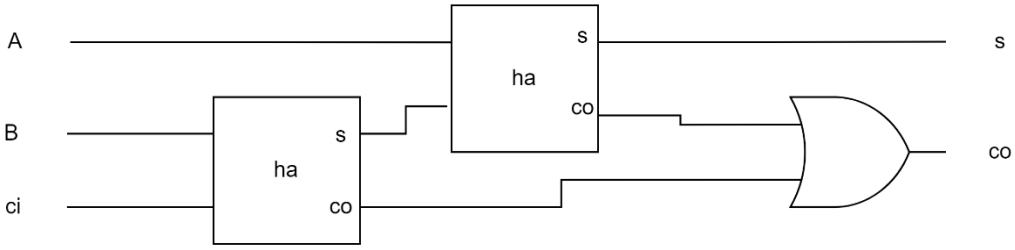
Sum s = A!B!Cin + !AB!Cin + !A!BCin + ABCin = A(!B!Cin + BCin) + !A(B!Cin + !BCin)

= A!(B⊕C) + !A(B⊕C) = A⊕B⊕C

Logic gate

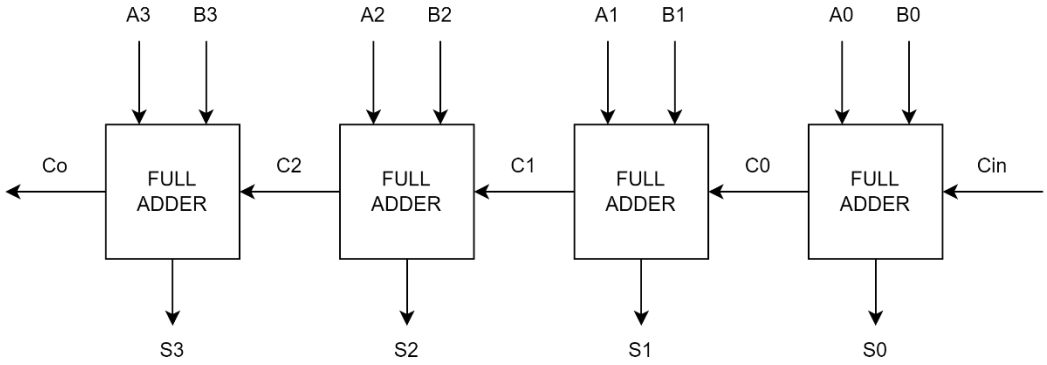


3 input xor gate 1개, 2 input nand gate 3개, 3 input nand gate 1개를 이용한 full adder



Half adder 2개와 2 input or gate 1개를 이용한 full adder

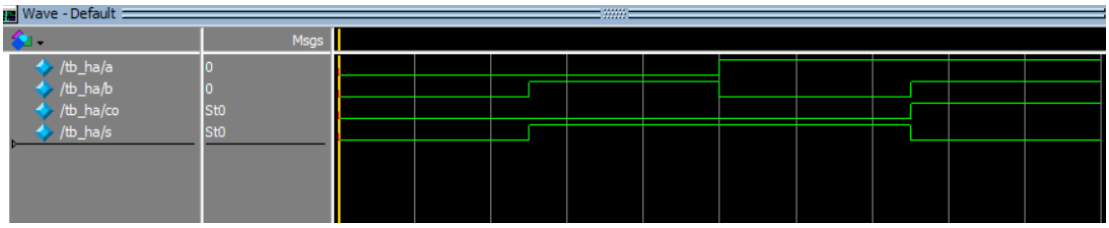
Rca4의 회로도



4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

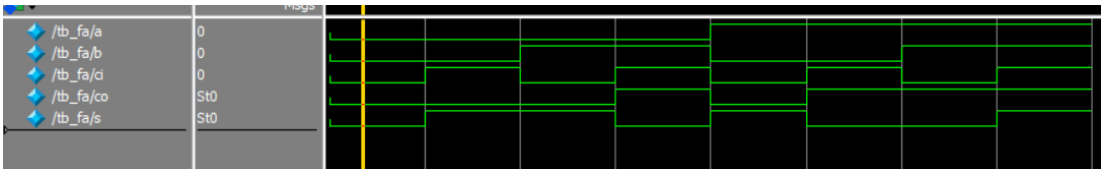
Half adder의 testbench



ID	Input condition	Expected value	Measured value	comparison
1	a = 1'b0 b = 1'b0	co = 1'b0 s = 1'b0	co = 1'b0 s = 1'b0	corrected
2	a = 1'b0 b = 1'b1	co = 1'b0 s = 1'b1	co = 1'b0 s = 1'b1	corrected
3	a = 1'b1 b = 1'b0	co = 1'b0 s = 1'b1	co = 1'b0 s = 1'b1	corrected
4	a = 1'b1	co = 1'b1	co = 1'b1	corrected

	b = 1'b1	s = 1'b0	s = 1'b0	

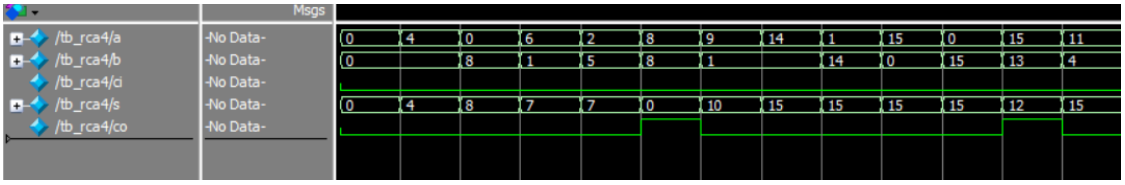
Full adder의 testbench



ID	Input condition	Expected value	Measured value	comparison
1	a = 1'b0 b = 1'b0 ci = 1b'0	co = 1'b0 s = 1'b0	co = 1'b0 s = 1'b0	corrected
2	a = 1'b0 b = 1'b0 ci = 1b'1	co = 1'b0 s = 1'b1	co = 1'b0 s = 1'b1	corrected
3	a = 1'b0 b = 1'b1 ci = 1b'0	co = 1'b0 s = 1'b1	co = 1'b0 s = 1'b1	corrected
4	a = 1'b0 b = 1'b1 ci = 1b'1	co = 1'b1 s = 1'b0	co = 1'b1 s = 1'b0	corrected
5	a = 1'b1 b = 1'b0 ci = 1b'0	co = 1'b0 s = 1'b1	co = 1'b0 s = 1'b1	corrected
6	a = 1'b1 b = 1'b0 ci = 1b'1	co = 1'b1 s = 1'b0	co = 1'b1 s = 1'b0	corrected
7	a = 1'b1 b = 1'b1 ci = 1b'0	co = 1'b1 s = 1'b0	co = 1'b1 s = 1'b0	corrected
8	a = 1'b1 b = 1'b1 ci = 1b'1	co = 1'b1 s = 1'b1	co = 1'b1 s = 1'b1	corrected

rca4의 testbench

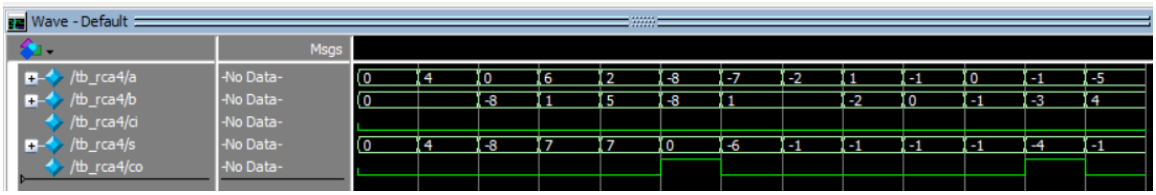
- decimal



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

- unsigned

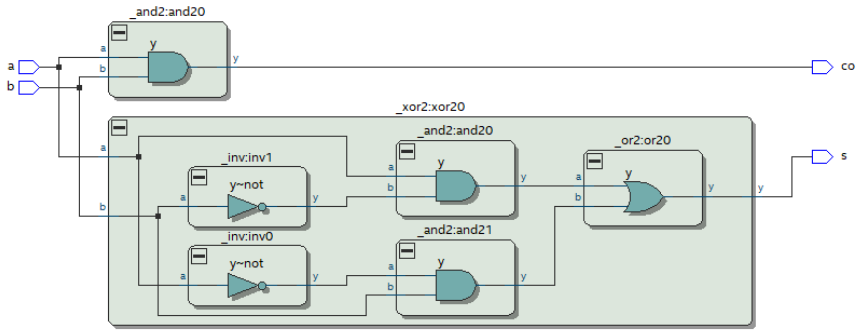


ID	Input condition	Decimal value	Expected unsigned value	Measured unsigned value	correspond
1	a = 4'd0 b = 4'd0 ci = 1'b0	s = 4'd0 co = 1b'0	s = 4'd0 co = 1b'0	s = 4'd0 co = 1b'0	Yes
2	a = 4'd4 b = 4'd0 ci = 1'b0	s = 4'd4 co = 1b'0	s = 4'd4 co = 1b'0	s = 4'd4 co = 1b'0	Yes
3	a = 4'd0 b = 4'd8 ci = 1'b0	s = 4'd8 co = 1b'0	s = 4'd-8 co = 1b'0	s = 4'd-8 co = 1b'0	Yes
4	a = 4'd6 b = 4'd1 ci = 1'b0	s = 4'd7 co = 1b'0	s = 4'd7 co = 1b'0	s = 4'd7 co = 1b'0	Yes
5	a = 4'd2 b = 4'd5 ci = 1'b0	s = 4'd7 co = 1b'0	s = 4'd7 co = 1b'0	s = 4'd7 co = 1b'0	Yes
6	a = 4'd8 b = 4'd8 ci = 1'b0	s = 4'd0 co = 1b'1	s = 4'd0 co = 1b'1	s = 4'd0 co = 1b'1	Yes
7	a = 4'd9 b = 4'd1 ci = 1'b0	s = 4'd10 co = 1b'0	s = 4'd-6 co = 1b'0	s = 4'd-6 co = 1b'0	Yes
8	a = 4'd14 b = 4'd1	s = 4'd15 co = 1b'0	s = 4'd-1 co = 1b'0	s = 4'd-1 co = 1b'0	Yes

	ci = 1'b0				
9	a = 4'd1 b = 4'd14 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd-1 co = 1b'0	s = 4'd-1 co = 1b'0	Yes
10	a = 4'd15 b = 4'd0 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd-1 co = 1b'0	s = 4'd-1 co = 1b'0	yes
11	a = 4'd0 b = 4'd15 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd-1 co = 1b'0	s = 4'd-1 co = 1b'0	Yes
12	a = 4'd15 b = 4'd13 ci = 1'b0	s = 4'd12 co = 1b'0	s = 4'd-4 co = 1b'0	s = 4'd-4 co = 1b'0	Yes
13	a = 4'd11 b = 4'd4 ci = 1'b0	s = 4'd15 co = 1b'0	s = 4'd-1 co = 1b'0	s = 4'd-1 co = 1b'0	Yes

B. 합성(synthesis) 결과

Half adder의 RTL map viewer & flow summary



Half adder의 boolean equation을 살펴보면 다음과 같다.

Carry out Cout = AB, Sum s = A!B + !AB = A⊕B

RTL map viewer를 통해 해당 boolean equation 대로 잘 연결되었음을 확인할 수 있다.

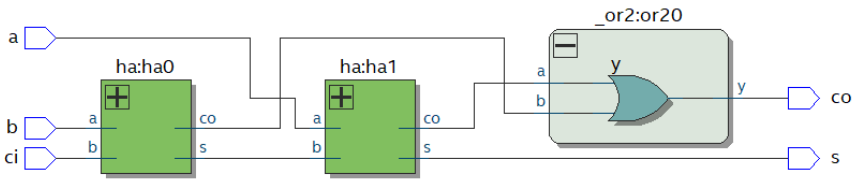
Flow Summary

<<Filter>>

Flow Status	In progress - Mon Sep 18 15:31:31 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca4
Top-level Entity Name	ha
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	2 / 41,910 (< 1 %)
Total registers	0
Total pins	4 / 499 (< 1 %)
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 %)

Project를 어차피 rca4에서 해결하는 거라서 half adder를 여기서 같이 설계했다. Top-level 에서 보면 해당 flow summary는 half adder를 돌린 결과임을 확인할 수 있다.

Full adder의 RTL map viewer & flow summary



Full adder의 boolean식을 보면 다음과 같다.

Carry out Cout = AB + BCin + CinA, Sum s = A!(B⊕C) + !A(B⊕C) = A⊕B⊕C

그러나 Full adder는 두 개의 half adder와 2 input or gate를 이용해서도 구현 가능하다. 위의 RTL map viewer 위와 같은 방식을 이용하고 있다.

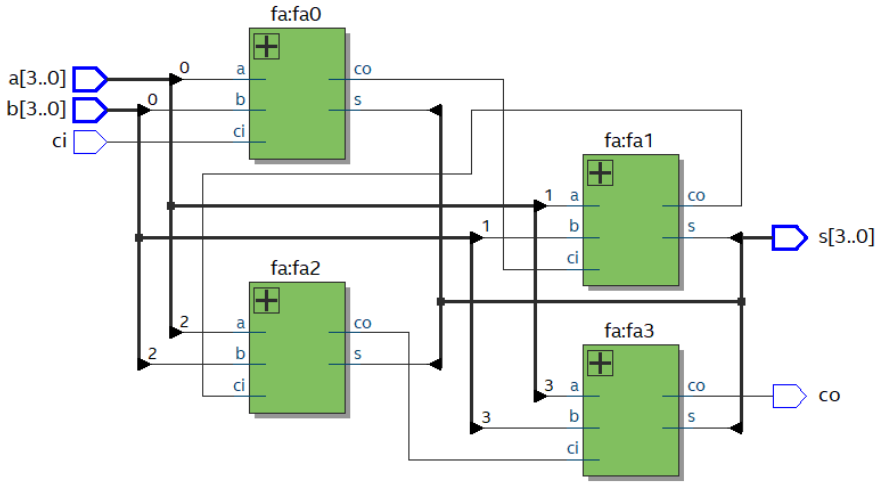
Flow Summary

<<Filter>>

Flow Status	Successful - Mon Sep 18 15:29:51 2023
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca4
Top-level Entity Name	fa
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	2 / 41,910 (< 1 %)
Total registers	0
Total pins	5 / 499 (1 %)
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 %)

Full Adder또한 half adder와 같이 rca4의 Project를 사용했다. Top-level에서 보면 해당 flow summary는 full adder를 돌린 결과임을 확인할 수 있다.

rca4의 RTL map viewer & flow summary



Rca4는 full adder를 4개를 연속으로 연결한 가산기이다. 각 비트별로 덧셈을 수행하여, cout이 다음 full adder의 cin으로 들어가는 것을 확인할 수 있다. 마지막 full adder의 co은 다음 비트의 덧셈을 수행할 수 없어 overflow로 값을 무시하면 된다.

Flow Summary	
<<Filter>>	
Flow Status	In progress - Mon Sep 18 15:28:18 2023
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)	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 %)

위 flow summary를 살펴보면 rca4가 top module인걸 확인할 수 있으며 flow status에서 successful를 나와 제대로 합성된 사실을 알 수 있다.

5. 고찰 및 결론

A. 고찰

bus를 베릴로그 코드 내에서 처음 사용하게 되어 조금 헷갈렸으나, bus의 특성이 c/c++의 배열 느낌으로 적용하면서 적응했다. 그러나 nets나 ports에서의 bus를 선언할 때 [MSB : LSB] 형식이라는 점이 배열과 달라 bus 쓸 때마다 이 개념을 되뇌이면서 사용했다. 후에 베릴로그로 [LSB : MSB] 형식으로 모듈을 돌려보았다. 나머지는 위의 형식을 따르고 S를 [LSB : MSB] 형식으로 rca를 돌렸을 때 이진수로 S가 반전된 것처럼 나왔다. 아마 모든 bus의 형식만 맞추게 된다면 이 형식으로 사용해도 모듈은 잘 돌아갈 것 같다.

B. 결론

4-bit RCA를 이용하여 32-bit RCA를 구현하기 위해서 rca4모듈을 8개를 이어 붙인다. 이는 그냥 full adder를 32개를 이어 붙이는 역할과 같이 수행하니 이번 실습으로 구현한 rca4를 이어 붙임으로써 좀 더 효율적으로 수행할 수 있다. 더 나아가 이런 원리를 이용하면 32bit를 초과하는 rca를 구현이 가능하다.

6. 참고문헌

이준환 교수/ 디지털논리회로2/ 광운대학교(컴퓨터정보공학부)/ 2023