

# 컴퓨터 공학 기초 실험2 보고서

실험제목: 4bit RCA

실험일자: 2020년 09월 18일 (금)

제출일자: 2020년 09월 21일 (월)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 금요일 5,6,7

학 번: 2019202009

성 명: 서여지

## 1. 제목 및 목적

### A. 제목

4bit Ripple Carry Adder

### B. 목적

Ripple Carry Adder의 동작을 이해한다. 베릴로그를 이용하여 기본 게이트 모듈을 작성하고, 이를 이용하여 half adder, full adder, 4-bit ripple carry adder를 구현한다. 적당한 TestBench를 작성하고, 실행결과를 바탕으로 결과를 해석한다.

## 2. 원리(배경지식)

### 1. XOR

XOR(exclusive OR) gate는 두 개의 입력이 있을 때 두 값이 일치하지 않는 경우, 즉 하나의 input이 1이고 나머지는 0인 경우에 참 값을 output으로 출력하는 동작을 한다.

2 input XOR gate의 진리표와 k-map은 다음과 같다.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

A \ B	0	1
0	0	1
1	1	0

그리고 기호  $\oplus$ 로 나타낸다.

$$Y = A\bar{B} + \bar{A}B = A \oplus B$$

### 2. Adder

Adder는 더하기 계산을 하는 회로이다. input으로 들어가는 올림수의 여부에 따라 half adder와 full adder가 나뉜다. Half adder와 full adder는 한 bit에 대한 더하기 계산을 할 수 있고, ripple carry adder는 여러 bit의 더하기 계산을 할 수 있다. Half adder는 input으로 더해질 두 수를 받고, output으로 더한 결과와 올림수인 carry out을 반환한다. 이것에 올림수 하나를 입력으로(carry in) 더 받는 것이 full adder이고, 하나의 full adder의 carry out을 다음 full adder의 carry in으로 연결하여 여러 개의 full adder를 연결하여 여러 bit의 계산을 가능하게 한 것이 ripple carry adder이다.

### 3. 모듈, 인스턴스

모듈은 베릴로그의 basic building block, 즉 프로그램을 구성하는 기본 단위가 된다. 모듈 내부에는 사용되는 레지스터와 와이어 등을 선언하고, 해당 모듈이 실행할 명령이 있다. 모듈 내부에서 다른 모듈을 불러와 사용할 수 있다. 모듈을 불러올 때 생성되는 것이 인스턴스이다. 하나의 모듈로 여러 개의 인스턴스를 생성하여 사용할 수도 있다. 모듈과 인스턴스의 관계는 얼음틀과 얼음의 관계에 대입하여 생각할 수 있다.

### 4. exhaustive/directed verification

Verification은 작성한 프로그램을 확인하는 과정이다. 완성한 회로에 적절한 input값을 지정하여 정상적인 결과가 반환되는지 여부를 확인한다. 검증법은 모든 경우에 대해 직접 확인해보는 exhaustive verification과 일부 경우에 대해서만 확인해보는 directed verification이 있다. 이번 과제의 Ripple Carry Adder의 경우 input의 수가 많기 때문에 directed verification을 이용한다.

### 5. 2의 보수 원리

2의 보수 원리는 2진수로 음수를 표현하는 방법 중 하나이다. 가장 앞의 비트인 MSB를 수의 부호를 나타내는 sign bit로 사용한다. Sign bit가 1인 경우 음수이고, 0인 경우 해당 수가 양수임을 의미한다. 2의 보수를 구하는 방법은 2진수로 표현된 수의 각 비트를 1은 0으로, 0은 1로 모두 반전시킨 뒤 1을 더하는 것이다. -6의 경우 6을 4비트의 2진수로 표현하면 0110이고, 비트를 반전시켜 1의 보수를 구하면 1001, 이것에 1을 더하면 1010이 된다. 2의 보수로 나타난 수에서 MSB에 해당하는 비트는 음수가 되고, 나머지 자리에 나타난 수는 양수 값으로 이해할 수 있다. 앞에서 예를 든 1010의 경우, 1000은 -8이고, 10은 +2 이므로, 결국 -6이 된다.

## 3. 설계 세부사항

### 1. Half adder

Half adder의 진리표와 k-map, Boolean Equation은 다음과 같다.

CO				S			
a	b	co	s	a \ b		a \ b	
0	0	0	0	0	0	0	1
0	1	0	1	0	0	0	1
1	0	0	1	1	0	1	0
1	1	1	0	1	1	1	0

$$co = ab$$

$$s = a\bar{b} + \bar{a}b = a \oplus b$$

따라서 half adder를 구현한 ha모듈은 s를 \_xor2 모듈을 이용해 구하고, co는 \_and2 모듈을 이용해 계산하도록 했다.

## 2. Full adder

Full adder의 진리표와 k-map, Boolean equation은 다음과 같다.

co					s					
ci	a	b	co	s	ab	ci	00	01	11	10
0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	1	0	0	0	1	0	1
0	1	0	0	1	1	0	0	1	1	1
0	1	1	1	0	1	0	1	0	1	0
1	0	0	0	1	1	1	1	1	0	1
1	0	1	1	0	1	1	1	1	1	0
1	1	0	1	0	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1

$$co = ab + aci + bci = a(b \oplus ci) + bci$$

$$s = \bar{a}\bar{b}ci + \bar{a}b\bar{c}i + abci + a\bar{b}\bar{c}i = a \oplus b \oplus ci$$

Full adder를 구현한 모듈인 fa는 a,b,ci를 input으로 받고, s,co를 output으로 내보낸다. Co의 연산은 b와 ci를 input으로 하는 ha모듈의 co(=bci)와, 해당 모듈의 s(=b⊕ci)와 a를 input으로 하는 ha모듈의 co(=a(b⊕ci))를 or게이트를 이용해 연산한 값(a(b⊕ci)+ bci)으로 한다. S는 b와 ci을 input으로 하는 ha모듈의 s(=b⊕ci)와, a를 다시 input으로 받는 ha모듈의 s값(=a⊕b⊕ci) 이다.

## 3. Ripple carry adder

Ripple carry adder는 full adder의 co를 다른 full adder의 ci으로 연결하여 제작한다. 4 bit RCA에서 full adder는 모두 4개 필요하다. Ripple carry adder를 구현한 모듈인 rca에서는 4개의 fa모듈을 인스턴스 하여 각각의 co와 ci을 c와이어로 연결해서 RCA를 구현하였다.

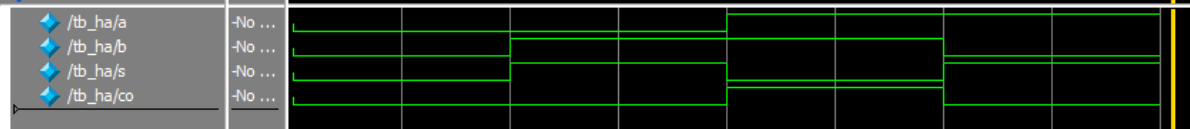
모듈은 a와 b, ci의 값을 모두 더한 값을 s에, 올림수를 co에 반환해야 한다

#### 4. 설계 검증 및 실험 결과

##### A. 시뮬레이션 결과

###### 1. Half adder

testbench는 a와 b값을 10ns 주기로 변경하며, 모든 경우의 수에 대하여 작성하였다.

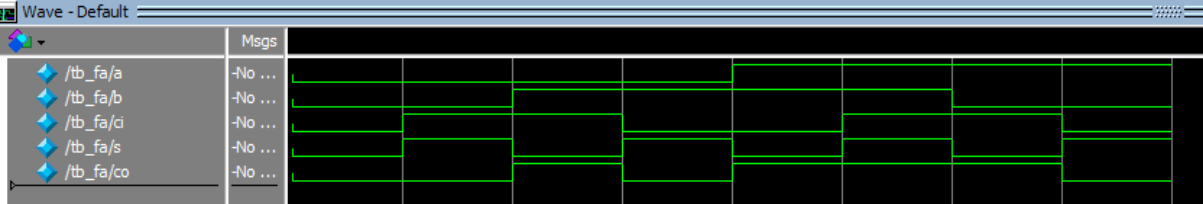


	0	~10ns	~20ns	~30ns	~40ns
a	0	0	0	1	1
b	0	0	1	1	0
s	0	0	1	0	1
co	0	0	0	1	0

a, b의 모든 경우에서 s와 co의 결과가 모두 정상적으로 나온 것을 확인할 수 있다. a, b의 값이 일치하지 않을 때 s값이 1, 다른 경우 0이 출력되었고, a, b 값이 모두 1일 때 co의 값이 1이 출력되었다.

###### 2. Full adder

testbench는 a와 b, ci의 값을 10ns 주기로 변경하며, 모든 경우에 대하여 작성하였다.



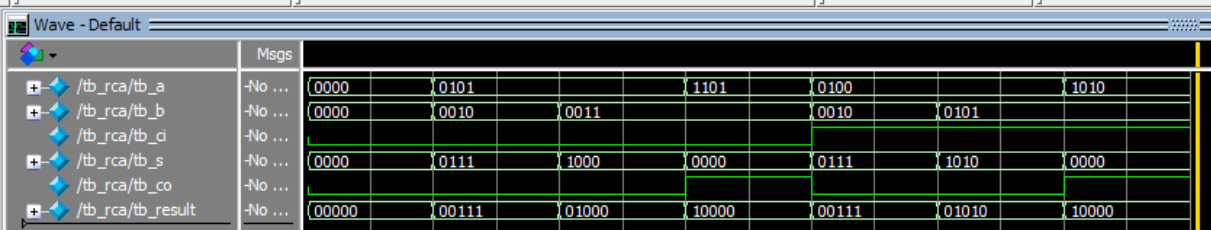
	0	~10ns	~20ns	~30ns	~40ns	~50ns	~60ns	~70ns	~80ns
a	0	0	0	0	1	1	1	1	1
b	0	0	0	1	1	1	1	0	0
ci	0	1	1	0	0	1	1	1	0
s	0	1	0	1	0	1	0	0	1
co	0	0	1	0	1	1	1	1	0

a, b, ci의 조합인 8가지 경우 모두에서 s와 co의 결과가 모든 경우에서 정상적으로 나왔다. a,b,ci 중 값이 1인 것이 하나, 혹은 셋 일 때 s의 값이 1이고, 둘 이상일 때 co의 값이 1로 출력되었다.

###### 3. 4 bits RCA

testbench는 a와 b, ci의 값을 10ns 주기로 변경하며, 임의의 경우에 대하여 작성하였다. carry in이 있는 경우와 없는 경우로 나누고, 각각의 경우를 다시 carry가 발생하지 않는 경우, full adder사이에서만 carry가 발생하고 carry out은 없는 경우, carry out이 발생하는 3가지의 경우로 나누었고, 0을 더하는 경우를 포함한 총 7가지 이진수 덧셈을 시행하였다.

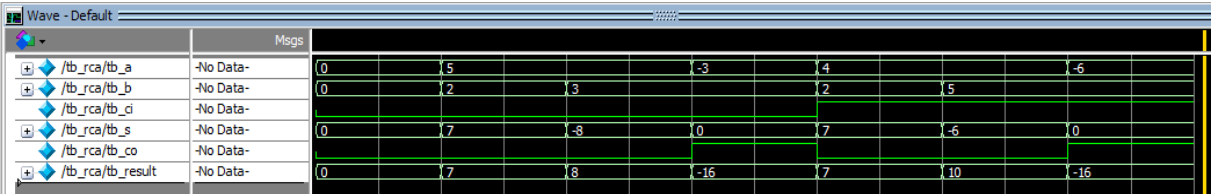
### ● Binary



	0	~10ns	~20ns	~30ns	~40ns	~50ns	~60ns	~70ns
tb_a	0000	0101	0101	1101	0100	0100	1010	
tb_b	0000	0010	0011	0011	0010	0101	0101	
tb_ci	0	0	0	0	1	1	1	
tb_s	0000	0111	1000	0000	0111	1010	0000	
tb_co	0	0	0	1	0	0	1	
tb_result	00000	00111	01000	10000	00111	01010	10000	

결과를 이진수로 표현했을 때 tb\_s, tb\_co, tb\_result의 결과는 쉽게 이해할 수 있다. 각 자리의 수가 더해져 s에 출력되었고, 올림수는 다음자리에서 정상적으로 계산되었다. 그러나 carry out이 있는 경우 tb\_s의 비트 수를 초과하므로 값이 잘리는 것을 확인할 수 있다.

### ● Decimal



	0	5	5	-3	4	4	-6
tb_a	0	5	5	-3	4	4	-6
tb_b	0	2	3	3	2	5	5
tb_ci	0	0	0	0	1	1	1
tb_s	0	7	-8	0	7	-6	0
tb_co	0	0	0	1	0	0	1
tb_result	0	7	8	-16	7	10	-16

수를 2의 보수를 사용한 방식의 2진수에서 10진수로 표현하였다. 앞에서 분류한 6가지 경우 중, carry가 발생한 모든 경우에서 tb\_s 혹은 tb\_result에서 특이한 점을 찾을 수 있다. 먼저 carry out

이 발생하지 않은 경우(20~30ns, 50~60ns) tb\_result 값은 정상적이지만 tb\_s의 값이 의미와 맞지 않는다. 이것은 부호를 나타내는 MSB가 채워졌기 때문에 나타나는 문제이다. 그렇기 때문에 비트 수가 하나 더 많은 tb\_result에서는 수가 제대로 출력된 것을 볼 수 있다. 반면 carry out이 발생한 경우(30~40ns, 60~70ns)는 tb\_s값이 정상적이지만 tb\_result값이 음수 값으로 나타난 것을 볼 수 있다. 이것은 결과를 binary로 확인했을 때와 같은 이유로, tb\_s와 tb\_result의 비트 수가 다르기 때문에 나타난 결과이다. tb\_s의 경우 carry out이 버려져서 정상적으로 0이 출력되지만, tb\_result에서는 MSB에 1이 저장되었기 때문에 음수가 출력된다.

- Unsigned

Wave - Default								
Mega								
/tb_rca/tb_a	-No Data-	0	5	5	13	4	4	10
/tb_rca/tb_b	-No Data-	0	2	3	3	2	5	5
/tb_rca/tb_ci	-No Data-	0	0	0	0	1	1	1
/tb_rca/tb_s	-No Data-	0	7	8	0	7	10	0
/tb_rca/tb_co	-No Data-	0	0	0	1	0	0	1
/tb_rca/tb_result	-No Data-	0	7	8	16	7	10	16

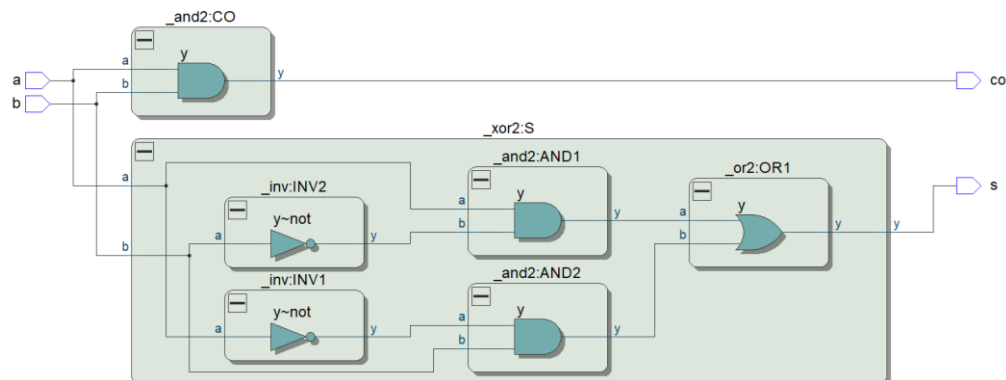
tb_a	0	5	5	13	4	4	10
tb_b	0	2	3	3	2	5	5
tb_ci	0	0	0	0	1	1	1
tb_s	0	7	8	0	7	10	0
tb_co	0	0	0	1	0	0	1
tb_result	0	7	8	16	7	10	16

결과를 Unsigned로 표시하는 경우 carry out이 발생한 경우(30~40ns, 60~70ns)의 tb\_s가 기대한 값과 다른 것을 확인할 수 있다. 결과를 Binary로 표현했을 때 살펴본 것과 같이, tb\_s는 carry out에 의한 비트가 한 자리 잘렸지만, tb\_result에서는 MSB에 1이 저장되었기 때문에 나타난 결과임을 알 수 있다.

## B. 합성(synthesis) 결과.

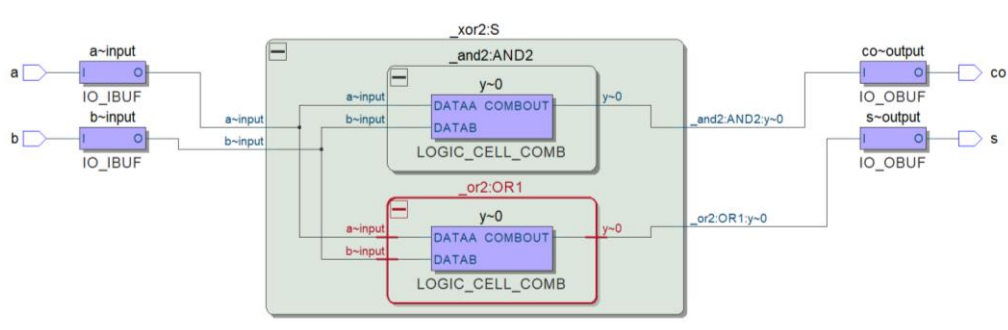
### 4. Half adder

#### (1) RTL viewer



co는 a와 b의 값을 and 한 결과이고, s는 a와 b', a'와 b를 and 연산한 뒤 or 계산을 한 결과임이 나타난다.

## (2) Technology map viewer



Technology map viewer는 위와 같이 나타났다. AND2의 결과가 co로 출력되고, OR1의 결과가 s로 출력된다.

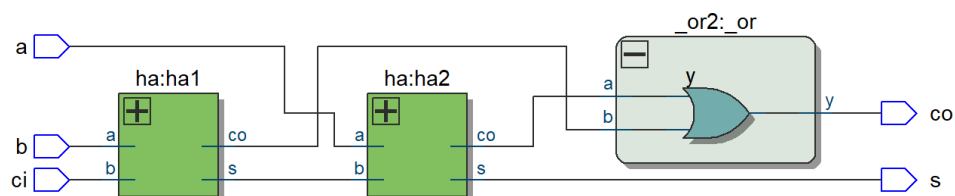
## (3) Flow summary

Flow Summary	
Flow Status	Successful - Sun Sep 20 07:33:31 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	RCA
Top-level Entity Name	ha
Family	Cyclone V
Device	5CSXFC6D6F3106
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 %)

Top-level Entity Name은 half adder 모듈의 이름인 ha이다.

## 5. Full adder

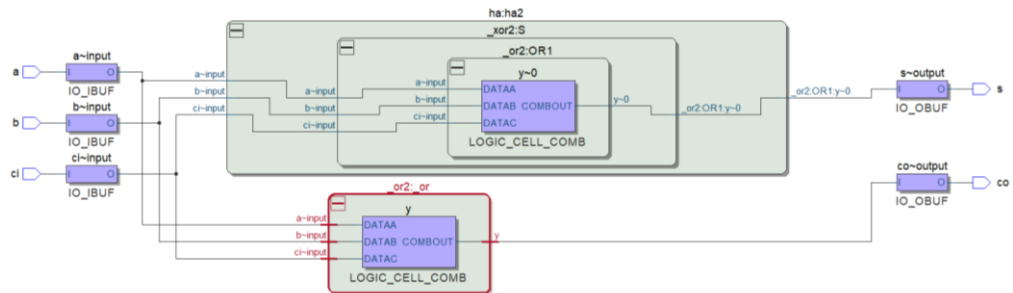
### (1) RTL viewer



Half adder 모듈 ha의 인스턴스인 ha1, h2를 이용하여 full adder를 구성한 모습이다. 설계과정에서 k-map을 이용해 살펴본 것처럼, b와 ci 입력을 ha1에서  $b \cdot ci$ 와  $b \oplus ci$ 를 구한 후, a와  $b \oplus ci$ 를 ha2의 입력으로 넣어  $a(b \oplus ci)$ 와  $a \oplus b \oplus ci$ 를 얻는다. 마지막으로  $bci + ab + aci$ 를 구해 co를 출력한다.



## (2) Technology map viewer



Technology map viewer는 위와 같이 나타났다.

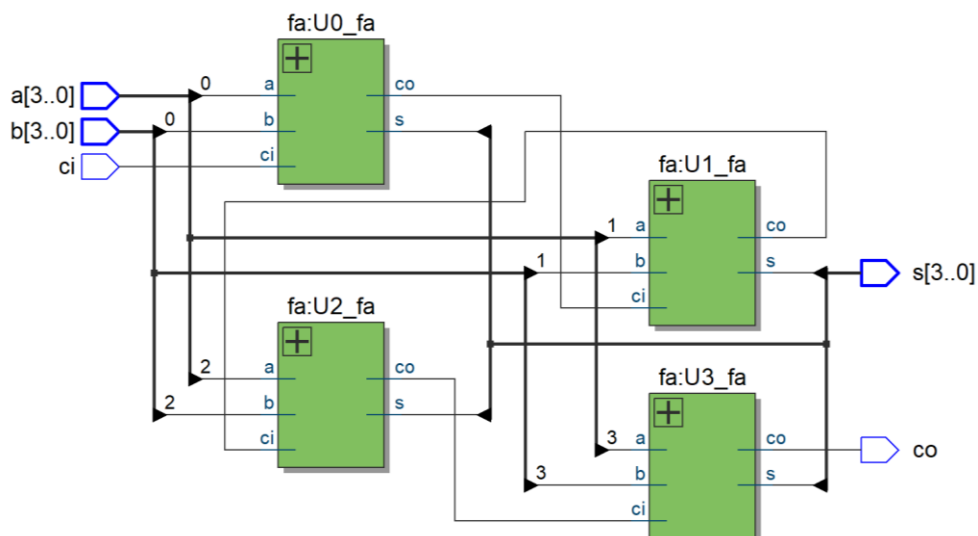
## (3) Flow summary

Flow Summary	
Flow Status	Successful - Sun Sep 20 07:39:01 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	RCA
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 % )

Top-level Entity Name은 full adder 모듈의 이름인 fa이다. Half adder의 경우와 비교해서 ci 입력이 생겼으므로 Total pins의 수가 1 증가하였다.

## 6. 4 bits RCA

### (1) RTL viewer



4개의 full adder를 이용해서 4bits RCA를 구성하였다. 한 비트를 계산했던 half adder나 full adder와는 다르게 여러 개의 비트가 굵은 선으로 표시된 것을 확인할 수 있다. 한 모듈의 co출력이 다음 full adder의 ci로 입력되고 있다. 마지막으로 계산되는 U3\_fa모듈의 co출력은 RCA의 co출력이 된다.

## (2) Flow summary

Flow Summary	
Flow Status	Successful - Sun Sep 20 07:43:17 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	RCA
Top-level Entity Name	rca
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 % )

Top-Level Entity Name은 ripple carry adder 모듈의 이름인 rca이다. a와 b, s가 각각 4개의 핀을 사용하고, ci와 co가 각각 하나의 핀을 가지기 때문에 Total pins가 14이다.

## 5. 고찰 및 결론

### A. 고찰

이번 과제는 이전에 생성한 모듈을 인스턴스 해서 사용하는 구조가 반복되었다. 처음에 모듈명을 작성하고 인스턴스의 이름을 따로 지정해주지 않아서 컴파일은 제대로 실행되었으나 시뮬레이션이 동작하지 않았다. 각각의 인스턴스명을 작성해서 문제를 해결했다. 완성된 rca의 testbench를 작성하는 과정에서 0101과 같이 입력했다가 기대하는 결과와 다른 값이 출력되었다. 문제는 입력 값이 10진수로 입력된 것이었다. 이 경우 4'b0101과 같이 비트 수와 2진수 표현임을 명시하여 문제를 해결했다.

### B. 결론

저번 주에 제작한 MUX에 이어서 간단한 베릴로그 프로그래밍을 수행하였다. 하나의 프로젝트에서 여러 개의 베릴로그 코드 파일과 testbench를 생성하여 원하는 testbench를 실행하는 과정을 연습할 수 있었다. 아직까지 프로그래밍 언어와 프로그램에 익숙하지 않은 것 같아서 디지털논리회로1에서 학습했던 다른 모듈도 구현해보며 베릴로그 프로그래밍을 연습을 하는 것이 좋을 것 같다.

### 32-bit RCA 설계하기

4-bit ripple carry adder는 4개의 full adder를 이용해 4자리 덧셈을 수행할 수 있었다. 32자리의 덧셈을 수행하는 32-bit RCA를 4-bit RCA를 이용해 설계하려면 총 8개의 4-bit RCA가 필요하다. 각 4-bit rca의 s는 그대로 32-bit rca의 s값으로 출력되고, co는 다음 비트를 계산하는 RCA의 ci으로 연결해야 한다. 마지막으로 작동하는 rca의 co는 32-bit RCA의 co 출력이 된다. 한 비트를 계산하는 full-adder를 4개 연결하여 4비트 계산을 한 것과 마찬가지로, 네 자리의 비트를 계산하는 rca를 8개 연결하여 32 자리 비트를 연산하는 rca를 만들 수 있다.

## 6. 참고문헌

이준환교수님, 디지털논리로2 강의자료, 광운대학교 컴퓨터정보공학과,2020

David Money Harris 외 1인, Digital Design and Computer Architecture, Elsevier