6주차 결과보고서

전공: 신문방송학과 학년: 4학년 학번: 20191150 이름: 전현길

1. 실험 목적

- Adder(가산기)/ Subtractor(감산기)의 개념 이해
- Code converter(부호 변환기)의 개념 이해
- Verilog를 사용하여 다양한 Adder 및 Subtractor 구현
- Verilog를 사용하여 다양한 Code converter 구현
- FPGA 통해서 Verilog로 구현된 회로의 동작 확인

2. Full Adder 및 Half Adder 의 simulation 결과 및 과정에 대해서 설명하시오. (진리표 포함)

source code

```
`timescale 1ns / 1ps

module halfAdd(
    input a, b,
    output out1, out2
);

assign out1 = a^b;
// sum
assign out2 = a&b;
// carry out

endmodule
```

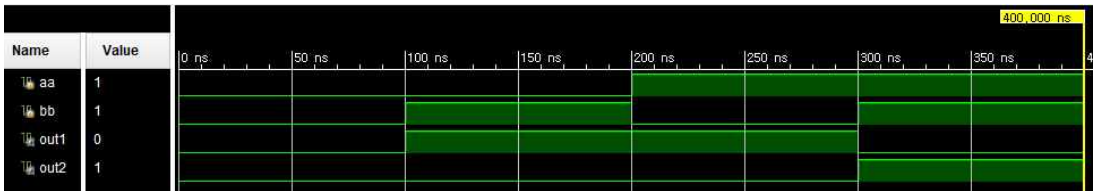
testbench code

```
`timescale 1ns / 1ps

module halfAdd_tb;

reg aa, bb;
wire out1, out2;

halfAdd u_test(
    .a (aa), .b (bb),
    .out1 (out1), .out2 (out2)
);

initial begin
    aa = 1'b0;
    bb = 1'b0;
end

always@(aa or bb) begin
    aa <= #200 ~aa;
    bb <= #100 ~bb;
end

initial begin
    #400
    $finish;
end

endmodule
```

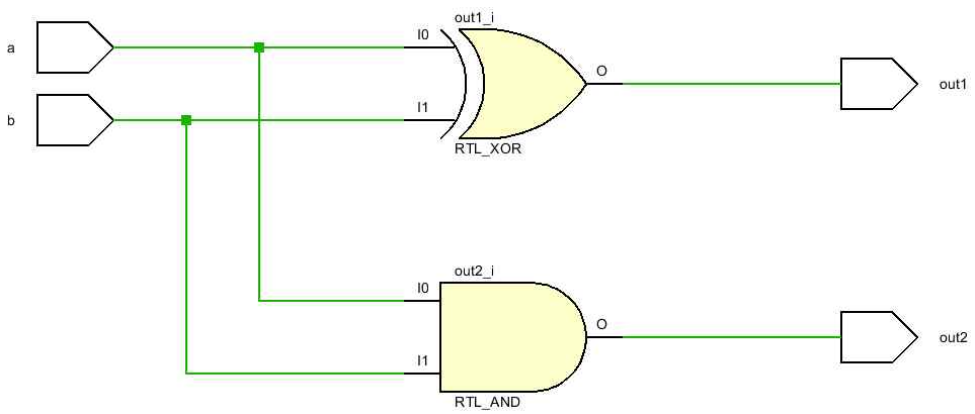
half adder truth table

input		output	
a	b	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

half adder simulation



half adder circuit design

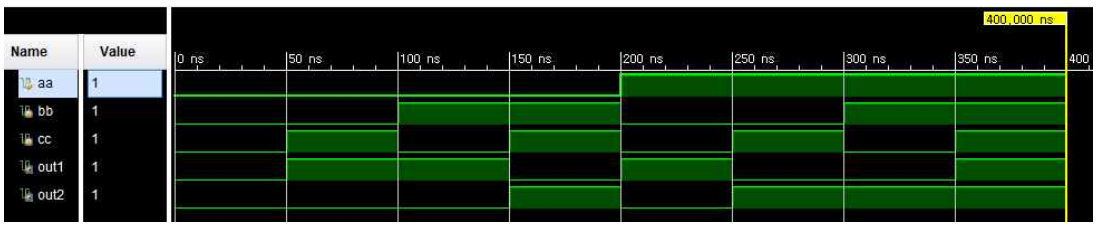


source code	testbench code	full adder truth table				
		input			output	
		a	b	c	sum	carry

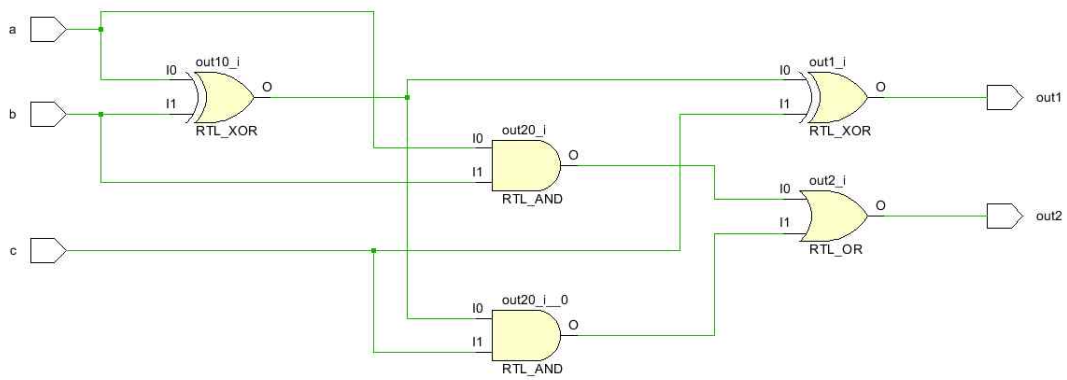
`timescale 1ns / 1ps
 	
module fullAdd(
input a, b, c, // c: carry in
output out1, out2
);
 	
assign out1 = (a^b)^c; // sum
assign out2 = (a&b) | ((a^b) & c);
// carry out
 	
endmodule
	`timescale 1ns / 1ps
	
	module fullAdd_tb;
	
	reg aa, bb, cc;
	wire out1, out2;
	
	fullAdd u_test(
	.a (aa), .b (bb), .c (cc),
	.out1 (out1), .out2 (out2)
	);
	
	initial begin
	aa = 1'b0;
	bb = 1'b0;
	cc = 1'b0;
	end
	
	always@(aa or bb or cc) begin
	aa <= #200 ~aa;
	bb <= #100 ~bb;
	cc <= #50 ~cc;
	end
	
	initial begin
	#400
	$finish;
	end
	
	endmodule
		0	0	0	0	0

		0	0	1	1	0
		0	1	0	1	0
		0	1	1	0	1
		1	0	0	1	0
		1	0	1	0	1
		1	1	0	0	1
		1	1	1	1	1

full adder simulation



full adder circuit design



반가산기는 A와 B 두 입력을 받아서 **합(sum)과 자리올림수(carry)**를 출력한다. 동작하는 방식은 이진수 덧셈과 같다. 1과 1을 더할 때에는 Sum 1bit만

으론 표현할 수 없다는 점이 문제가 되는데, 이 경우 sum은 0을 출력하고 carry는 1을 출력한다. 구현식은 다음과 같다.

$$Sum = A \oplus B \mid Carry = A \cdot B$$

전가산기는 A와 B, C_{in} 세 입력을 받아서 **합(sum)과 자리올림수(carry)**를 출력한다. 따라서 1bit 입력 A + B + C_{in}을 모두 더한 뒤 적절한 합과 자리올림수를 출력해야 한다. 모든 입력이 0이라면 Sum/Carry 0/0을 출력하고, 한 개의 입력이 1이라면 1/0, 두 개의 입력이 1이라면 0/1, 세 개의 입력이 1이라면 1/1을 출력한다. 구현식은 다음과 같다.

$$Sum = A \oplus B \oplus C_{in} \mid C_{out} = C_{in} \cdot (A \oplus B) + A \cdot B$$

시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. half adder, full adder 양쪽 모두 진리표에서 작성한 바와 동일하게 시뮬레이션 결과가 나타난 것을 확인할 수 있었다.

3. Full Subtractor 및 Half Subtractorr 의 simulation 결과 및 과정에 대해서 설명하시오. (진리표 포함)

source code

```
`timescale 1ns / 1ps

module halfSub(
    input a, b,
    output out1, out2
);

assign out1 = a^b;
// difference
assign out2 = (~a)&b;
// borrow out

endmodule
```

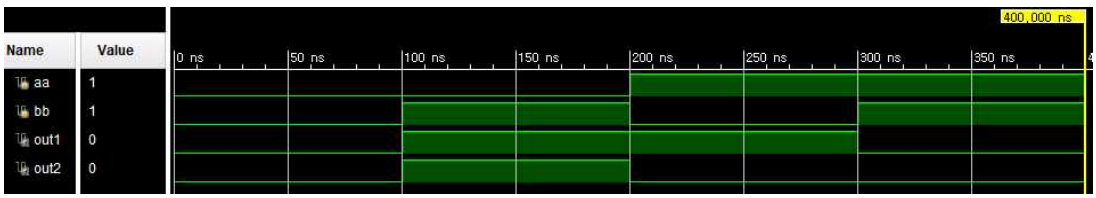
testbench code

```
`timescale 1ns / 1ps

module halfSub_tb;

reg aa, bb;
wire out1, out2;

halfSub u_test(
    .a (aa), .b (bb),
    .out1 (out1), .out2 (out2)
);

initial begin
    aa = 1'b0;
    bb = 1'b0;
end

always@(aa or bb) begin
    aa <= #200 ~aa;
    bb <= #100 ~bb;
end

initial begin
    #400
    $finish;
end

endmodule
```

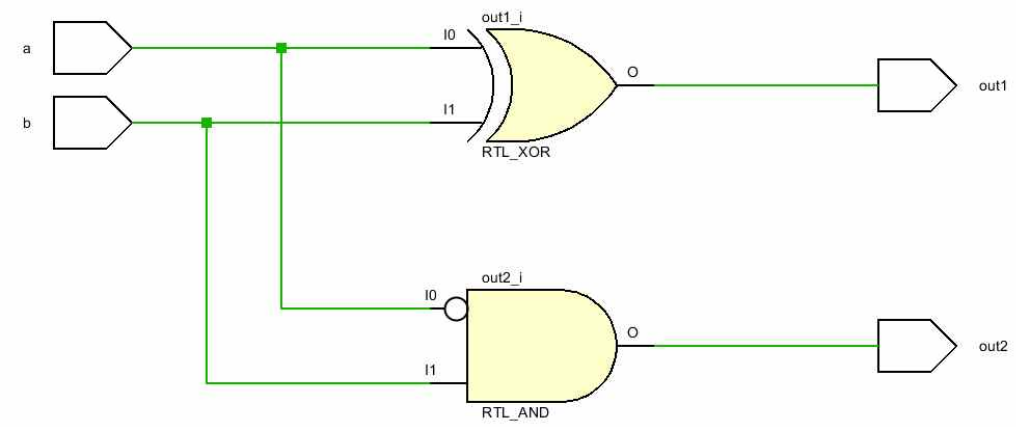
half subtractor truth table

input		output	
a	b	diff	borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

half subtractor simulation



half subtractor circuit design

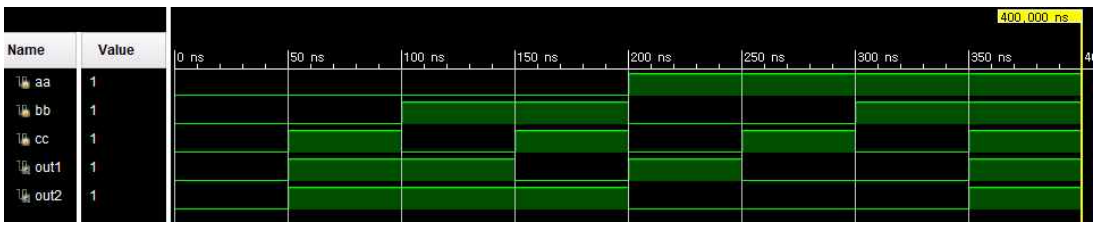


source code	testbench code	full subtractor truth table				
		input			output	
		a	b	c	diff	borrow

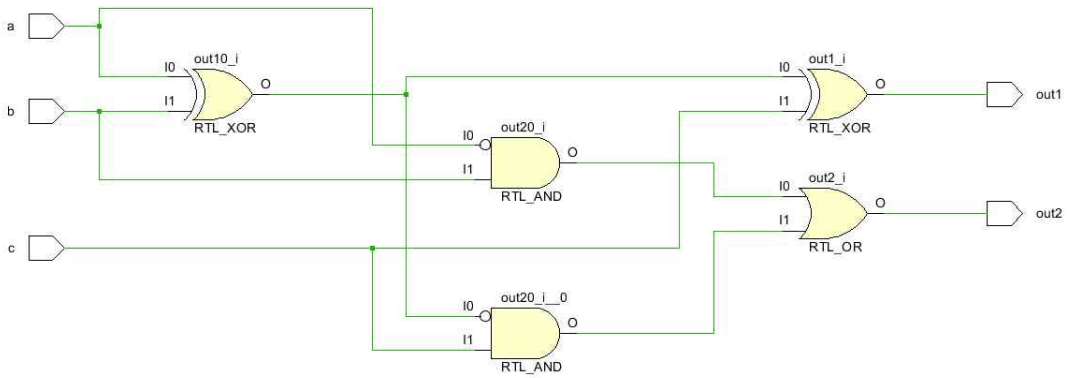
`timescale 1ns / 1ps
 	
module fullSub(
input a, b, c, // c: borrow in
output out1, out2
);
 	
assign out1 = (a^b)^c;
// difference
assign out2 = (~a&b) | (~(a^b) & c);
// borrow out
 	
endmodule
	`timescale 1ns / 1ps
	
	module fullSub_tb;
	
	reg aa, bb, cc;
	wire out1, out2;
	
	fullSub u_test(
	.a (aa), .b (bb), .c (cc),
	.out1 (out1), .out2 (out2)
	);
	
	initial begin
	aa = 1'b0;
	bb = 1'b0;
	cc = 1'b0;
	end
	
	always@(aa or bb or cc)
	begin
	aa <= #200 ~aa;
	bb <= #100 ~bb;
	cc <= #50 ~cc;
	end
	
	initial begin
	#400
	$finish;
	end
	
	endmodule
		0	0	0	0	0

		0	0	1	1	1
		0	1	0	1	1
		0	1	1	0	1
		1	0	0	1	0
		1	0	1	0	0
		1	1	0	0	0
		1	1	1	1	1

full subtractor simulation



full subtractor circuit design



반감산기는 A와 B 두 입력을 받아서 차(difference; D)와 **빌림(borrow; B_{out})**를 출력한다. **동작하는 방식은 2의 보수 뺄셈 A-B와 같다.** A, B가 각각 0/

0, 1/1이면 빌리지 않고, 차도 0이므로 Diff/B_{out} 각각 0/0을 출력한다. A가 1, B가 0일 때는 차는 1이고 빌리지도 않으므로 1/0을 출력하며, A가 0, B가 1일 때는 값을 빌려 뺄셈을 수행하므로 1/1을 출력한다.

$$\text{Difference(D)} = A \oplus B \mid \text{Borrow}(B_{out}) = A \cdot B$$

전감산기는 A와 B, B_{in} 세 입력을 받아서 차**(difference; D)와 빌림수(borrow; B_{out})**를 출력한다.. 따라서 1bit 입력 A – B - B_{in}의 식과 동일하게 동작하고, 이에 따라 적절한 차와 빌림수를 계산한다. 반가산기처럼 식의 결과에 따라 값을 빌려와야 한다면 B_{out}이 1이 되고, 그렇지 않으면 0이 된다.결과식은 다음과 같다.

$$\text{Diff} = A \oplus B \oplus B_{in} \mid \text{Carry} = C \cdot (\overline{A \oplus B}) + \overline{A} \cdot B$$

시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. half subtractor, full subtractor 양쪽 모두 진리표에서 작성한 바와 동일하게 시뮬레이션 결과가 나타난 것을 확인할 수 있었다.

4. 8421(BCD)-2421 Code converter simulation 결과 및 과정에 대해서 설명하시오. (진리표 작성 및 카르노맵 SOP form, POS form 포함)

source code

```
`timescale 1ns / 1ps

module convert(
    input a, b, c, d,
    output out1, out2, out3, out4
    );

assign out1 = a | (b&d) | (b&c);
assign out2 = a | (b&~d) | (b&c);
assign out3 = a | (~b&c) | (b&~c&d);
assign out4 = d;

endmodule
```

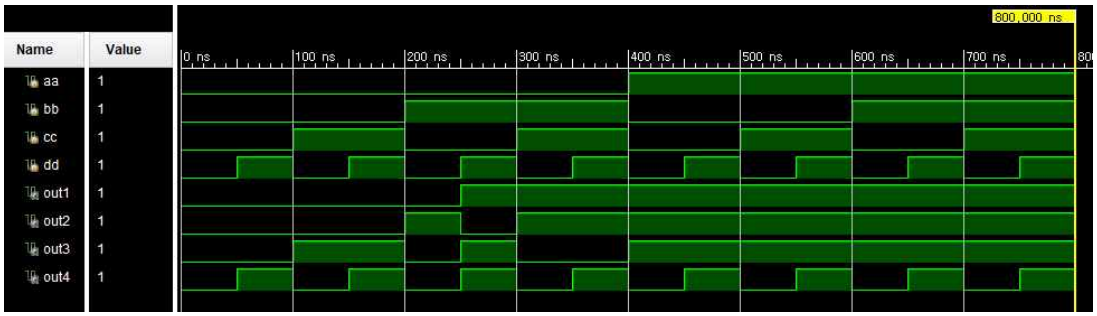
testbench code

```
`timescale 1ns / 1ps

module convert_tb;

reg aa, bb, cc, dd;
wire out1, out2, out3, out4;

convert u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.out1 (out1), .out2 (out2), .out3 (out3),
.out4 (out4)
);

initial begin
    aa = 1'b0;
    bb = 1'b0;
    cc = 1'b0;
    dd = 1'b0;
end

always@(aa or bb or cc or dd) begin
    aa <= #400 ~aa;
    bb <= #200 ~bb;
    cc <= #100 ~cc;
    dd <= #50 ~dd;
end

initial begin
    #800
    $finish;
end

endmodule
```

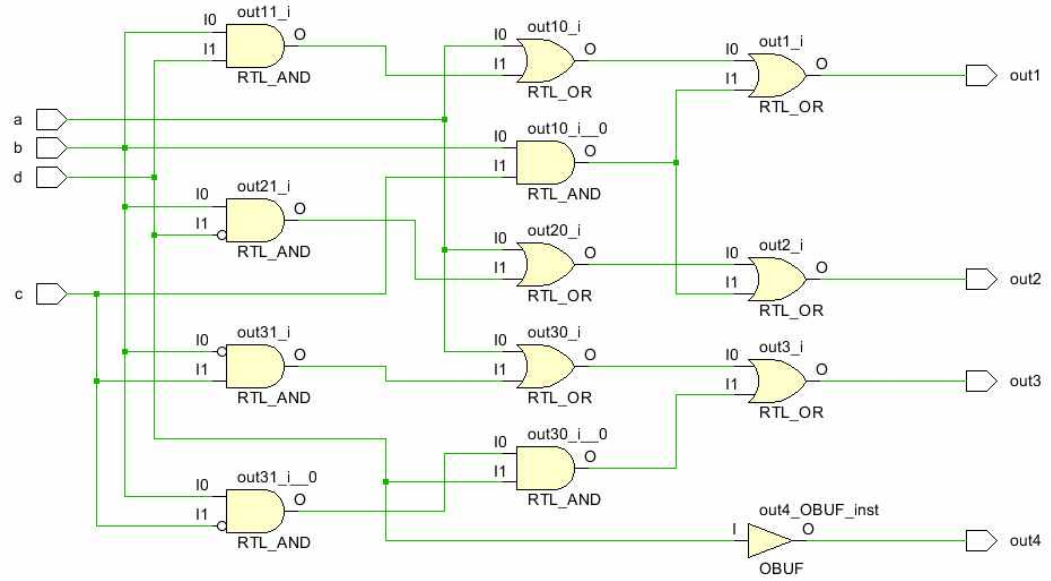
converter truth table

input				output			
a	b	c	d	o1	o2	o3	o4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	1	0	0
0	1	1	1	1	1	0	1
1	0	0	0	1	1	1	0
1	0	0	1	1	1	1	1
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

converter simulation



converter circuit design



out1 (cd\ab)	00	01	11	10
00	0	0	x	1
01	0	1	x	1
11	0	1	x	x
10	0	1	x	x

out2 (cd\ab)	00	01	11	10
00	0	1	x	1
01	0	0	x	1
11	0	1	x	x
10	0	1	x	x

out3 (cd\ab)	00	01	11	10
00	0	0	x	1
01	0	1	x	1
11	1	0	x	x
10	1	0	x	x

out4 (cd\ab)	00	01	11	10
00	0	0	x	0
01	1	1	x	1
11	1	1	x	x
10	0	0	x	x

SOP form		POS form	
out1	a + bd + bc	out1	(a+b)(a+c+d)
out2	a + bd' + bc	out2	(a+b)(a+c+d')
out3	a + b'c + bc'd	out3	(a+c+d)(a+b+c)(b'+c')
out4	d	out4	d

SOP form은 Karnaugh map의 1로 나타난 값들을 묶어서 구할 수 있고, POS form은 Karnaugh map의 0으로 나타난 값들을 묶어서 함수 F의 inverse F'를 구한 뒤, 드 모르간의 법칙을 사용해 구할 수 있다.

simulation 결과 don't care항을 제외한 모든 값(0000 ~ 1011)이 진리표에 작성한 바와 동일하게 결과값이 나타난 것을 확인할 수 있었다.

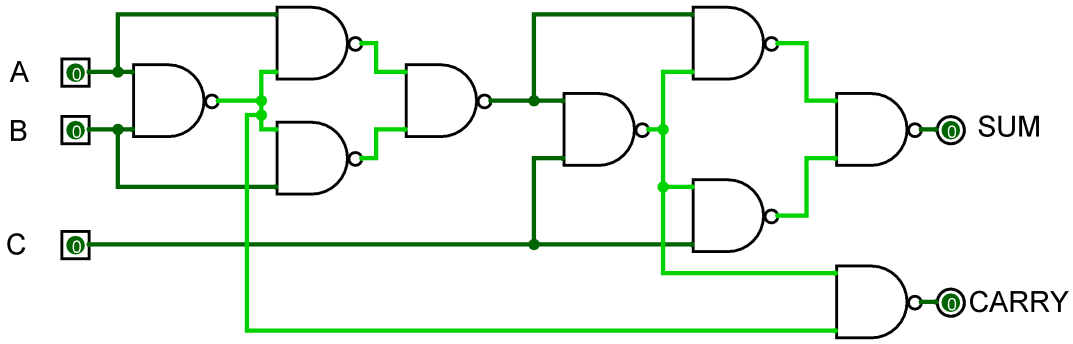
5. 결과 검토 및 논의 사항

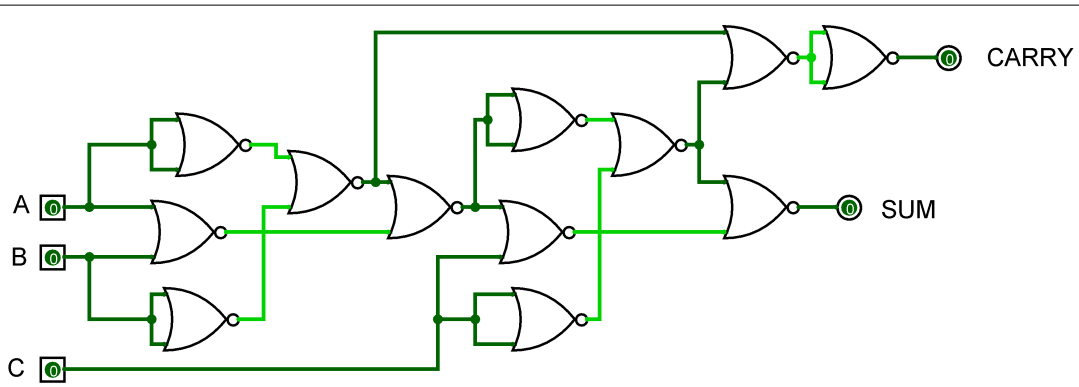
이번 실험에서는 full adder/subtractor, half adder/subtractor, 8421-2421 code converter를 각각 verilog 상에서 구현하여 시뮬레이션해 보았으며, FPGA 보드를 활용하여 실제 출력 결과를 확인해 보았다. 특히 8421-2421 code converter의 경우, 진리표를 작성하여 Karnaugh map을 그림으로써 SOP, POS 식을 도출했다. 실험 결과, 진리표를 바탕으로 기대되었던 동작이 정상적으로 이루어졌음을 확인했다.

6. 추가 이론 조사 및 작성

full adder/subtractor를 NAND/NOR gate로 구현한 회로는 다음과 같다.

Full Adder





Full Subtractor