5주차 결과보고서

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

1. 실험 목적

- De Morgan 의 정리/Boolean 함수의 동작 이해 및 확인
- Verilog를 사용하여 De Morgan 의 정리 및 Boolean 함수의 동작 구현
- 입력 신호 생성 후 Simulation을 통하여 구현된 결과 확인
- FPGA를 통해서 Verilog로 구현된 회로의 동작 확인

2. De-Morgan의 제 1,2 법칙의 simulation 결과 및 과정에 대해서 설명하시오. (NAND, NOR과 비교 포함)

source code

```
`timescale 1ns / 1ps

module deMorgan(
    input a, b,
    output c, d, e, f
);

assign c = ~(a|b);
assign d = ~a&~b;
assign e = ~(a&b);
assign f = ~a|~b;

endmodule
```

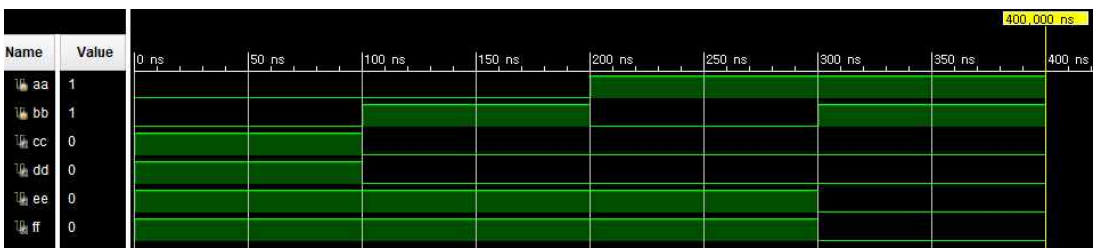
testbench code

```
`timescale 1ns / 1ps
module deMorgan_tb;

reg aa, bb;
wire cc, dd, ee, ff;

deMorgan u_test(
.a (aa), .b (bb),
.c (cc), .d (dd), .e (ee), .f (ff)
);

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
```

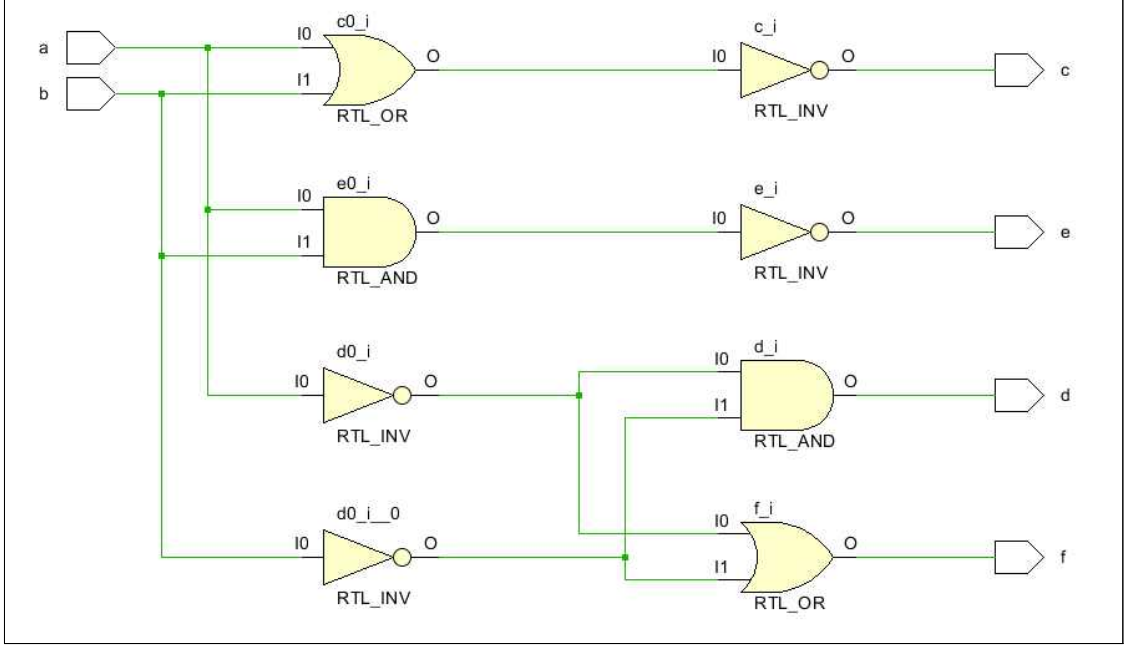
De Morgan's Law truth table

input		output			
a	b	~(a|b)	~a&~b	~(a&b)	~a|~b
0	0	1	0	0	0
0	1	1	0	0	0
1	0	1	1	1	0
1	1	1	1	1	0

De Morgan's Law simulation



De Morgan's Law circuit design



시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다. 또한, 각각 진리표, 회로, 소스 코드 및 테스트벤치 코드를 따로 작성했을 때 보기 불편하고 장황해지는 문제가 있어 드모르간의 법칙에 대한 4가지 연산을 모두 한꺼번에 작성했다.

a) De Morgan의 제 1법칙과 NOR gate

De Morgan의 제 1법칙은 $\overline{A + B} = \overline{A} \cdot \overline{B}$의 연산 결과가 동일하다는 것을 의미한다. 이 때, 첫 번째 $\overline{A + B}$의 식은 식의 형태가 **NOR gate와 동일**하다는 것을 회로도로 확인할 수 있다. 시뮬레이션 결과 역시 De Morgan의 제 1법칙에 의해 출력값 c, d가 동일하게 출력되는 것을 볼 수 있다.

두 식이 같은 결과를 반환하기 때문에, 상황에 따라 서로 다른 식을 선택함으로써 논리식 및 회로를 간소화할 수 있다.

b) De Morgan의 제 2법칙과 NAND gate

De Morgan의 제 2법칙은 $\overline{A \cdot B} = \overline{A} + \overline{B}$의 연산 결과가 동일하다는 것을 의미한다. 이 때, 첫 번째 $\overline{A \cdot B}$의 식은 식의 형태가 **NAND gate와 동일**하다는 것을 회로도로 확인할 수 있다. 시뮬레이션 결과 역시 De Morgan의 제 2

법칙에 의해 출력값 e, f가 동일하게 출력되는 것을 볼 수 있다.

역시 두 식이 같은 결과를 반환하기 때문에, 상황에 따라 서로 다른 식을 선택함으로써 논리식 및 회로를 간소화할 수 있다.

3. (A'+B')*C' = ((A*B)+C)'와 (A'*B')+C' = ((A+B)*C)'의 simulation 결과 및 과정에 대해서 설명하시오.

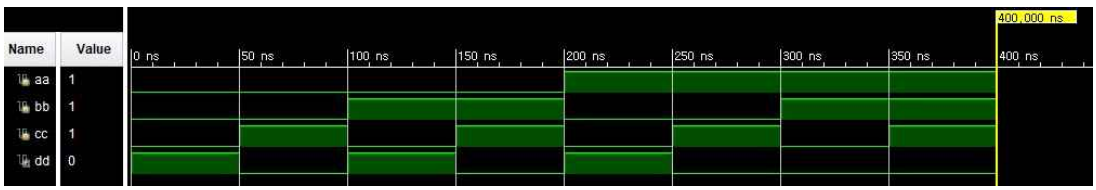
a. (A'+B')*C' = ((A*B)+C)'

source code	testbench code	(a'+b') * c' truth table			
		input			output
		a	b	c	d

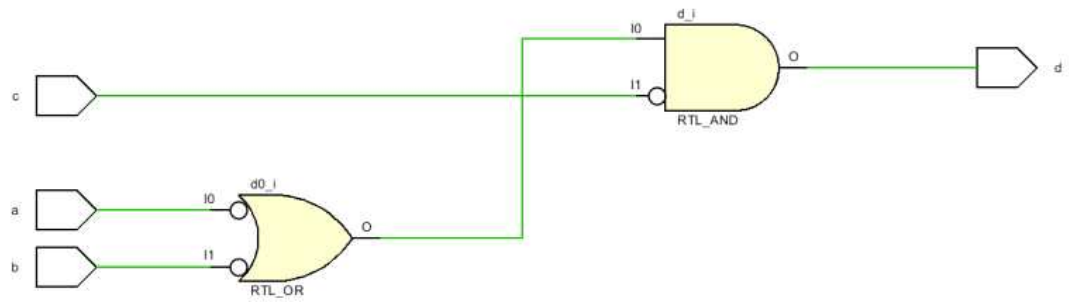
`timescale 1ns / 1ps
 	
module bool_1a(
input a, b, c,
output d
);
 	
assign d = (~a|~b) & ~c;
 	
endmodule
	`timescale 1ns / 1ps
	
	module bool_1a_tb;
	
	reg aa, bb, cc;
	wire dd;
	
	bool_1a u_test(
	.a (aa), .b (bb), .c (cc), .d (dd)
	);
	
	initial begin
	aa = 1'b0;
	bb = 1'b0;
	cc = 1'b0;
	end
	
	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	1

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

(a'+b') * c' simulation



(a'+b') * c' circuit design

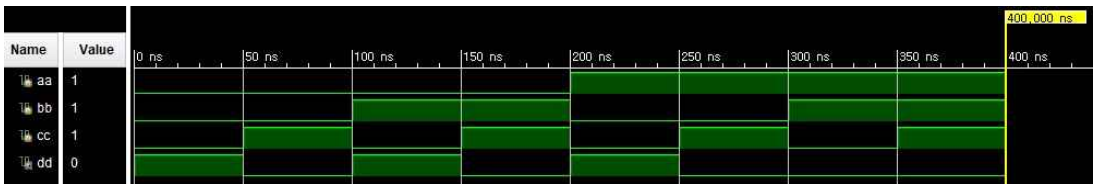


source code	testbench code	((a*b)+c)' truth table			
		input			output
		a	b	c	d

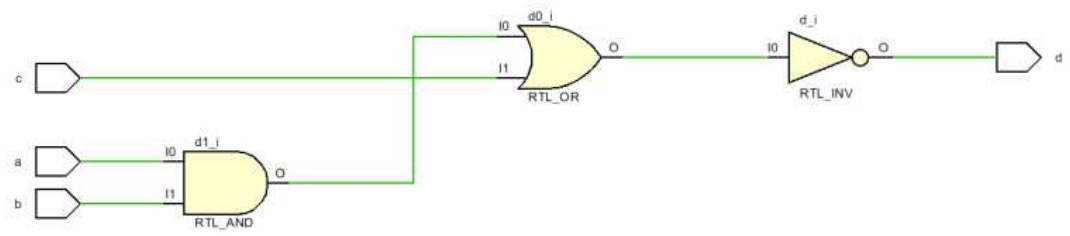
`timescale 1ns / 1ps
 	
module bool_1b(
input a, b, c,
output d
);
 	
assign d = ~((a&b)|c);
 	
endmodule
	`timescale 1ns / 1ps
	
	module bool_1b_tb;
	
	reg aa, bb, cc;
	wire dd;
	
	bool_1b u_test(
	.a (aa), .b (bb), .c (cc), .d (dd)
	);
	
	initial begin
	aa = 1'b0;
	bb = 1'b0;
	cc = 1'b0;
	end
	
	always@(aa or bb or cc) begin
	aa <= #200 ~aa;
	bb <= #100 ~bb;
	cc <= #50 ~cc;
	end
	
	initial begin
	#400
	$finish;
	end
	
	endmodule
		0	0	0	1

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

((a*b)+c)' simulation



((a*b)+c)' circuit design



드 모르간의 법칙을 활용하여 (A'+B')*C' = ((A*B)+C)'임을 확인할 수 있다. 좌변의 식은 우변 ((A*B)+C)에 대한 부정이므로, A, B, C 각 리터럴들을 부정하고 AND 연산과 OR 연산을 서로 치환해 주면 된다. 시뮬레이션을 통해서 실제로 드 모르간의 법칙에 의해 결과값이 동일하게 출력되는 것을 확인할 수 있다.

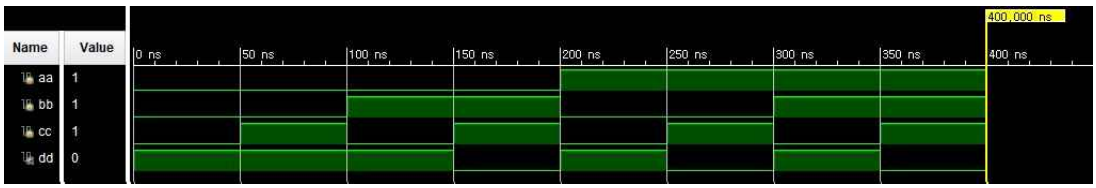
b. (A'*B')+C' = ((A+B)*C)'

source code	testbench code	(a'*b') + c' truth table			
		input			output
		a	b	c	d

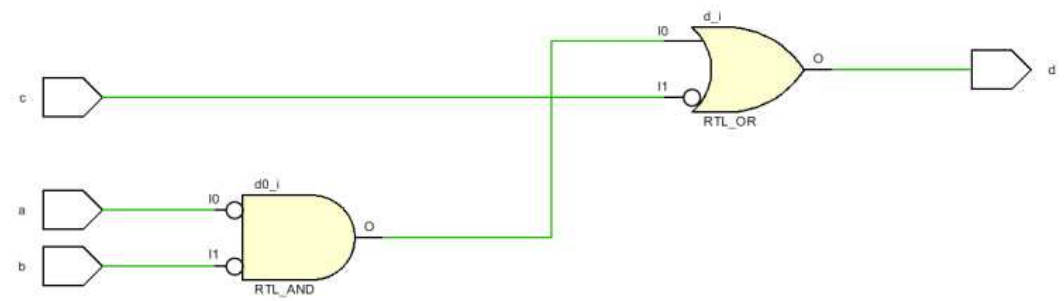
`timescale 1ns / 1ps
 	
module bool_2a(
input a, b, c,
output d
);
 	
assign d = (~a&~b)|~c;
 	
endmodule
	`timescale 1ns / 1ps
	
	module bool_2a_tb;
	
	reg aa, bb, cc;
	wire dd;
	
	bool_2a u_test(
	.a (aa), .b (bb), .c (cc), .d (dd)
	);
	
	initial begin
	aa = 1'b0;
	bb = 1'b0;
	cc = 1'b0;
	end
	
	always@(aa or bb or cc) begin
	aa <= #200 ~aa;
	bb <= #100 ~bb;
	cc <= #50 ~cc;
	end
	
	initial begin
	#400
	$finish;
	end
	
	endmodule
		0	0	0	1

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

(a'*b') + c' simulation



(a'*b') + c' circuit design

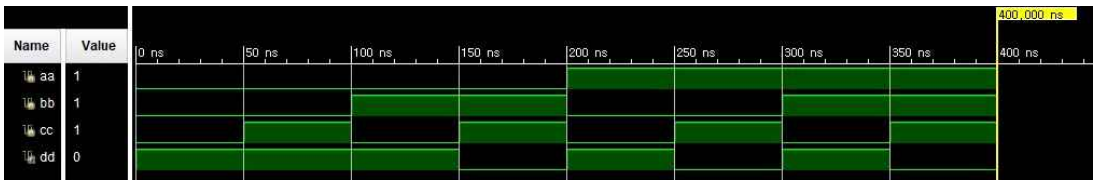


source code	testbench code	((a+b)*c)' truth table			
		input			output
		a	b	c	d

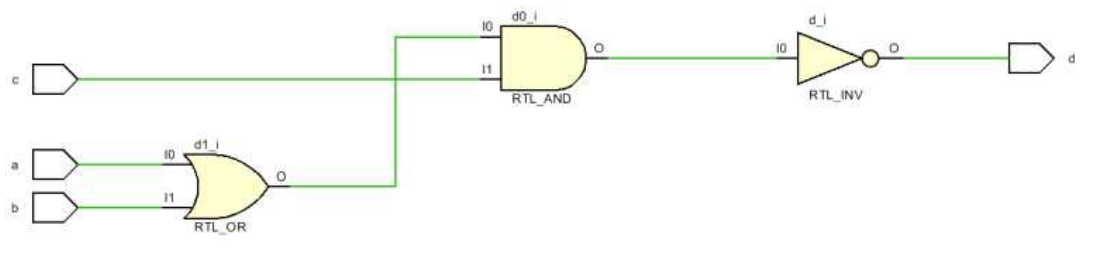
`timescale 1ns / 1ps
 	
module bool_2b(
input a, b, c,
output d
);
 	
assign d = ~((a|b)&c);
 	
endmodule
	`timescale 1ns / 1ps
	
	module bool_2b_tb;
	
	reg aa, bb, cc;
	wire dd;
	
	bool_2b u_test(
	.a (aa), .b (bb), .c (cc), .d (dd)
	);
	
	initial begin
	aa = 1'b0;
	bb = 1'b0;
	cc = 1'b0;
	end
	
	always@(aa or bb or cc) begin
	aa <= #200 ~aa;
	bb <= #100 ~bb;
	cc <= #50 ~cc;
	end
	
	initial begin
	#400
	$finish;
	end
	
	endmodule
		0	0	0	1

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

((a+b)*c)' simulation



((a+b)*c)' circuit design



드 모르간의 법칙을 활용하여 (A'*B')+C' = ((A+B)*C)'임을 확인할 수 있다. 좌변의 식은 우변 ((A+B)*C)'에 대한 부정이므로, A, B, C 각 리터럴들을 부정하고 AND 연산과 OR 연산을 서로 치환해 주면 된다. 시뮬레이션을 통해서 실제로 드 모르간의 법칙에 의해 결과값이 동일하게 출력되는 것을 확인할 수 있다.

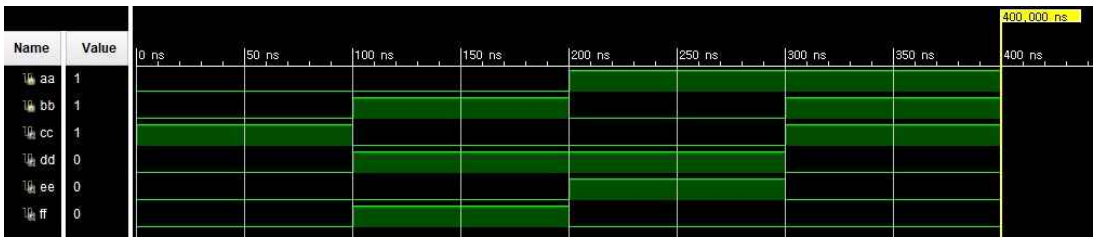
4. 1bit 비교기의 simulation 결과 및 과정에 대해서 설명하시오. (2 input, 4 output) [진리표 작성]

source code	testbench code	1bit comparator truth table					
		input		output			
		a	b	a==b	a!=b	a>b	a<b

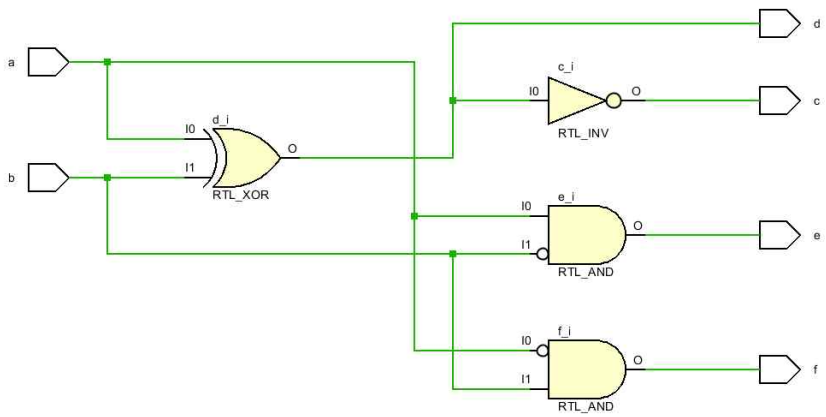
`timescale 1ns / 1ps
 	
module onebit(
input a, b,
output c, d, e, f
);
 	
assign c = ~(a^b);
assign d = a^b;
assign e = a&(~b);
assign f = (~a)&b;
 	
endmodule
	`timescale 1ns / 1ps
	
	module onebit_tb;
	
	reg aa, bb;
	wire cc, dd, ee, ff;
	
	onebit u_test(
	.a (aa), .b (bb), .c (cc),
	.d (dd), .e (ee), .f (ff)
	);
	
	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
		0	0	1	0	0	0

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

1bit comparator simulation



1bit comparator circuit design



시뮬레이션의 편의성을 위해, 진리표와 동일한 형식대로 값이 변화하도록 테스트벤치 코드를 작성했다.

1bit 비교기는 두 1bit 입력 A, B에 대해 A **== B, A != B, A > B, A < B** 4가지 값을 구현한다. 각각 다음과 같이

$$A == B \rightarrow A \text{ XOR } B \qquad A != B \rightarrow A \text{ XNOR } B$$

$$A > B \rightarrow A \cdot (\sim B) \qquad A < B \rightarrow (\sim A) \cdot B$$

연산으로 치환할 수 있다. 입력값이 1 또는 0으로만 주어지기 때문에 inverter와 AND gate만을 이용해 비교를 수행할 수 있다.

또한 A == B, A != B 연산은 A XOR B 연산과 동일하므로 XOR gate로 연산을 수행할 수 있고, A == B는 A != B의 역이므로 A XNOR B 연산으로 치환할 수 있다.

5. 결과 검토 및 논의 사항

이번 실험에서는 **De Morgan의 법칙이 실제로 성립하는지 특정한 이진함수 식을 드 모르간의 법칙으로 치환해 시뮬레이션**해 보았고, **1bit 비교기를 gate level에서 구현해본 뒤 시뮬레이션**해 보았다. FPGA를 활용해서 보드 상에서도 결과값을 출력해 보았다.

실험 결과, 1) De Morgan의 제 1법칙, 제 2법칙이 시뮬레이션에서도 그대로 나타났다. 2) 특정한 이진함수 식을 De Morgan의 법칙으로 치환했을 때에도 동일한 결과가 나타나는 것을 확인했다. 3) 1bit 비교기를 gate level로 치환했을 때 기대했던 동작이 나타나는 것을 확인했다.

뿐만 아니라, NAND gate와 NOR gate가 De Morgan의 법칙에 의해 각각 치환될 수 있으며, 이를 통해 회로를 간소화할 수 있다는 사실을 알 수 있었다.

6. 추가 이론 조사 및 작성

술어 논리에서도 드 모르간의 법칙이 사용되는데, 논리 회로에서 다루는 드 모르간의 법칙에 대해 조금 더 일반화된 내용이다. A(x)를 변수 x에 대한 서술자(=술어)라고 할 때, 다음과 같은 드 모르간 법칙이 성립한다.

NOT (모든 x에 대해 A(x)이다.) = (어떤 x에 대해 A(x)가 아니다.)

NOT (어떤 x에 대해 A(x)이다.) = (모든 x에 대해 A(x)가 아니다.)

확인한 것처럼 술어를 부정하고, '모든', '어떤'이라는 양화사를 서로 바꿔주면 드 모르간의 법칙에 따라 명제를 부정할 수 있다.