2018년 2학기 컴퓨터공학실험Ⅱ
CSE3016-05반 3주차 결과 보고서

학번: 20171665
이름: 이 선 호

2018. 09. 28

목 차

Ⅰ FPGA 동작법

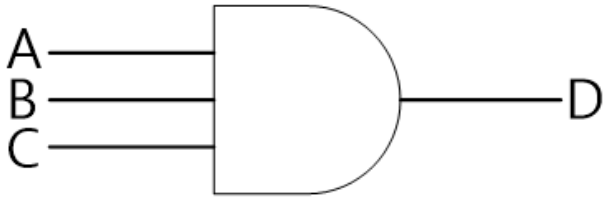
1. FPGA 동작법

FPGA를 PC와 연결하여 컴퓨터의 Verilog로 작성한 소스 코드를 가지고 실행한다. 우선 FPGA의 전원 케이블을 콘센트에 연결하고 전원 스위치를 켠 후에 마이크로 5핀 단자 케이블과 USB 케이블을 PC와 연결한다. 그리고 PC에서 Vivado 프로그램을 실행해서 프로젝트를 생성한다. 그 다음에 디자인 소스를 작성하고 나서 시뮬레이션 코드를 작성한다. 필요한 코드를 다 작성하고 나면 설정을 통해서 Device Assignment에서 사용할 FPGA Part를 선택하고 적용한다. 또한 FPGA에서 작동할 버튼이나 Pin이 필요하면 create constraints를 실행하여 Pin Assignment를 설정한 확장자 xdc 파일을 생성한다. Pin Assignment에서는 할당하고 싶은 Pin과 Verilog 소스의 port를 링크시키는 작업을 진행한다.

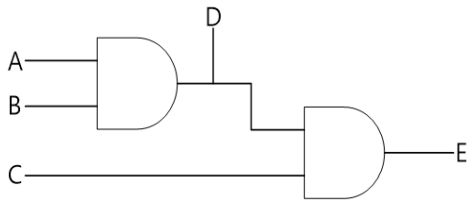
FPGA 동작을 검증하려면 첫 번째로 Run Synthesis를 실행한다. 그리고 Run Implementation을 실행하고 Run simulation을 실행한다. 마지막으로 Pin Assignment에서 설정한 Pin에 따라서 output이 정상적으로 나오는지 확인한다.

Ⅱ 다중 입력 Gate Simulation 결과 및 과정

1. 3-input AND Gate



(A)의 논리 회로도



(B)의 논리 회로도

1) (A)와 (B)의 Boolean 식 비교

(A)	(B)
$A \bullet B \bullet C = D$	$A \bullet B = D$ $(A \bullet B) \bullet C = E$

[표 Ⅱ-1-(1)] (A)와 (B)의 Boolean 식 비교

2) (A)와 (B)의 Verilog 코딩

(A)	(B)

`timescale 1ns / 1ps
 	
module inv(ina, inb, inc, outd);
 	
input ina, inb, inc;
output outd;
 	
and(outd, ina, inb, inc);
 	
endmodule
	`timescale 1ns / 1ps
	
	module inv(ina, inb, inc, outd, oute);
	
	input ina, inb, inc;
	output outd, oute;
	
	and(outd, ina, inb);
	and(oute, outd, inc);
	
	endmodule

[표 Ⅱ-1-(2)] (A)와 (B)의 Verilog Design Source 코드

(A)	(B)

`timescale 1ns / 1ps
 	
module inv_tb;
 	
wire dout;
reg ain, bin, cin;
 	
inv and_sim(.ina(ain),
.inb(bin),
.inc(cin),
.outd(dout));
 	
initial
begin
ain = 1'b0;
bin = 1'b0;
cin = 1'b0;
	`timescale 1ns / 1ps
	
	module inv_tb;
	
	wire dout, eout;
	reg ain, bin, cin;
	
	inv and_sim(.ina(ain),
	.inb(bin),
	.inc(cin),
	.outd(dout),
	.oute(eout));
	
	initial
	begin
	ain = 1'b0;
	bin = 1'b0;

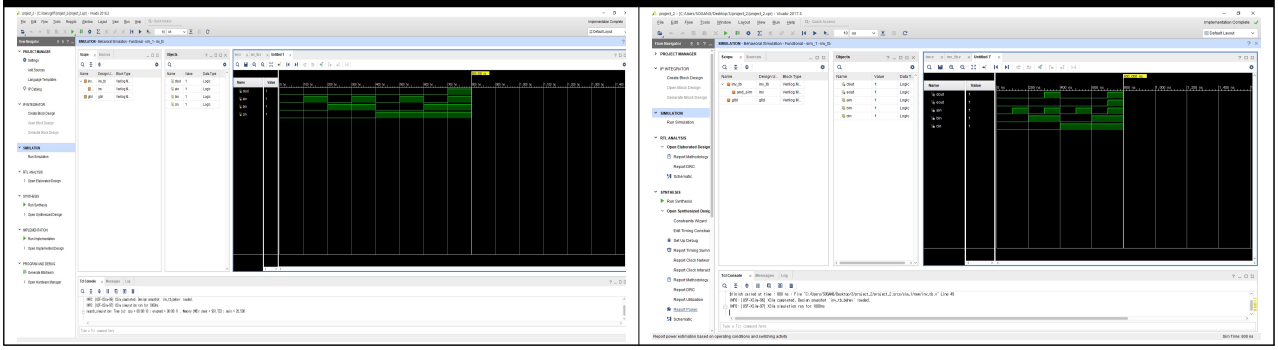
```
end

always@(ain or bin or cin) begin
ain <= #100 ~ain;
bin <= #200 ~bin;
cin <= #400 ~cin;
end

initial begin
      #800
      $finish;
end

endmodule
```

```
cin = 1'b0;
end

always@(ain or bin or cin) begin
ain <= #100 ~ain;
bin <= #200 ~bin;
cin <= #400 ~cin;
end

initial begin
      #800
      $finish;
end

endmodule
```

[표 Ⅱ-1-(3)] (A)와 (B)의 Verilog Simulation 코드

3) (A)와 (B)의 Simulation 결과

(A) (B)



[표 Ⅱ-1-(4)] (A)와 (B)의 Simulation 결과

4) (A)와 (B)의 진리표

(A)

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0

(B)

A	B	C	D	E
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0

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

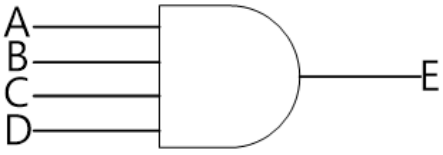
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

[표 Ⅱ-1-(5)] (A)와 (B)의 진리표

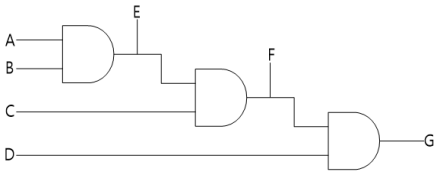
5) 설명

(A)에서는 A와 B와 C의 input 값이 모두 1일 때 D가 output 값으로 1이 출력되고, 그 외에는 모두 0이 출력된다. (B)에서는 A와 B의 값이 1일 때 D가 output 값으로 1이 출력되고, 그 외에는 모두 0이 출력된다. C의 값과 D의 값이 모두 1일 때 E가 output 값으로 1이 출력되고, 그 외에는 모두 0이 출력된다. 따라서 A, B, C가 모두 1일 때 E가 output 값으로 1이 출력된다.

2. 4-input AND Gate



(A)의 논리 회로도



(B)의 논리 회로도

1) (A)와 (B)의 Boolean 식 비교

(A)	(B)
$A \cdot B \cdot C \cdot D = E$	$A \cdot B = E$ $(A \cdot B) \cdot C = F$ $((A \cdot B) \cdot C) \cdot D = G$

[표 Ⅱ-2-(1)] (A)와 (B)의 Boolean 식 비교

2) (A)와 (B)의 Verilog 코딩

(A)	(B)
`timescale 1ns / 1ps	`timescale 1ns / 1ps

module inv(ina, inb, inc, ind, oute);
 	
input ina, inb, inc, ind;
output oute;
 	
and(oute, ina, inb, inc, ind);
 	
endmodule
	module inv(ina, inb, inc, ind, oute, outf, outg);
	
	input ina, inb, inc, ind;
	output oute, outf, outg;
	
	and(oute, ina, inb);
	and(outf, oute, inc);
	and(outg, outf, ind);
	
	endmodule

[표 Ⅱ-2-(2)] (A)와 (B)의 Verilog Design Source 코드

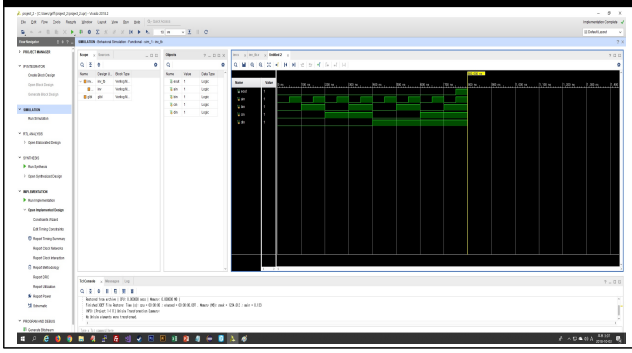
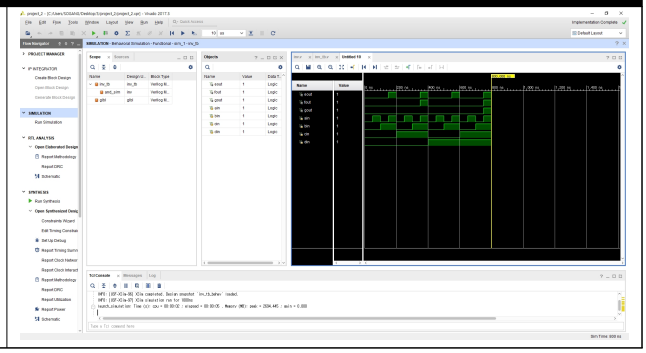
(A)	(B)

`timescale 1ns / 1ps
 	
module inv_tb;
 	
wire eout;
reg ain, bin, cin, din;
 	
inv and_sim(.ina(ain),
.inb(bin),
.inc(cin),
.ind(din),
.oute(eout));
 	
initial
begin
ain = 1'b0;
bin = 1'b0;
cin = 1'b0;
din = 1'b0;
end
 	
always@(ain or bin or cin or din) begin
ain <= #50 ~ain;
bin <= #100 ~bin;
cin <= #200 ~cin;
din <= #400 ~din;
end
	`timescale 1ns / 1ps
	
	module inv_tb;
	
	wire eout, fout, gout;
	reg ain, bin, cin, din;
	
	inv and_sim(.ina(ain),
	.inb(bin),
	.inc(cin),
	.ind(din),
	.oute(eout),
	.outf(fout)
	.outg(gout));
	
	initial
	begin
	ain = 1'b0;
	bin = 1'b0;
	cin = 1'b0;
	din = 1'b0;
	end
	
	always@(ain or bin or cin or din) begin
	ain <= #50 ~ain;
	bin <= #100 ~bin;
	cin <= #200 ~cin;

initial begin #800 $finish; end endmodule	din <= #400 ~din; end initial begin #800 $finish; end endmodule

[표 Ⅱ-2-(3)] (A)와 (B)의 Verilog Simulation 코드

3) (A)와 (B)의 Simulation 결과

(A)	(B)
	

[표 Ⅱ-2-(4)] (A)와 (B)의 Simulation 결과

4) (A)와 (B)의 진리표

(A)

A	B	C	D	E
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0

(B)

A	B	C	D	E	F	G
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0

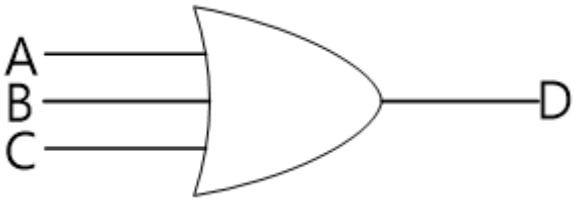
1	1	0	0	0		1	1	0	0	1	0	0
1	1	0	1	0		1	1	0	1	1	0	0
1	1	1	0	0		1	1	1	0	1	1	0
1	1	1	1	1		1	1	1	1	1	1	1

[표 Ⅱ-2-(5)] (A)와 (B)의 진리표

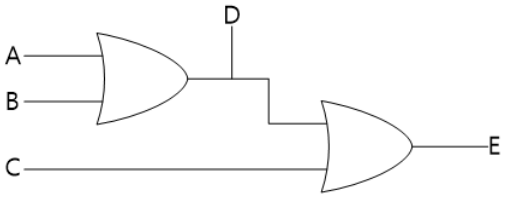
5) 설명

(A)에서는 A, B, C, 그리고 D input 값이 모두 1일 때 E가 output 값으로 1이 출력되고, 그 외에는 모두 0이 출력된다. (B)에서는 A와 B의 값이 1일 때 E가 output 값으로 1이 출력되고, 그 외에는 모두 0이 출력된다. C의 값과 E의 값이 모두 1일 때 F가 output 값으로 1이 출력되고, 그 외에는 모두 0이 출력된다. 마지막으로 D의 값과 F의 값이 모두 1일 때 G가 output 값으로 1이 출력된다. 따라서 A, B, C, D가 모두 1일 때 G가 output 값으로 1이 출력된다.

3. 3-input OR Gate



(A)의 논리 회로도



(B)의 논리 회로도

1) (A)와 (B)의 Boolean 식 비교

(A)	(B)
$A+B+C=D$	$A+B=D$ $(A+B)+C=E$

[표 Ⅱ-3-(1)] (A)와 (B)의 Boolean 식 비교

2) (A)와 (B)의 Verilog 코딩

(A)	(B)

```
`timescale 1ns / 1ps

module inv(ina, inb, inc, outd);

input ina, inb, inc;
output outd;

or(outd, ina, inb, inc);

endmodule
```

```
`timescale 1ns / 1ps

module inv(ina, inb, inc, outd, oute);

input ina, inb, inc;
output outd, oute;

or(outd, ina, inb);
or(oute, outd, inc);

endmodule
```

[표 Ⅱ-3-(2)] (A)와 (B)의 Verilog Design Source 코드

(A)

```
`timescale 1ns / 1ps

module inv_tb;

wire dout;
reg ain, bin, cin;

inv and_sim(.ina(ain),
.inb(bin),
.inc(cin),
.outd(dout));

initial
begin
ain = 1'b0;
bin = 1'b0;
cin = 1'b0;
end

always@(ain or bin or cin) begin
ain <= #100 ~ain;
bin <= #200 ~bin;
cin <= #400 ~cin;
end

initial begin
    #800
```

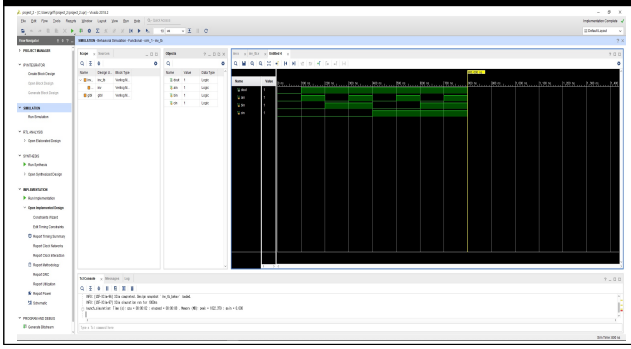
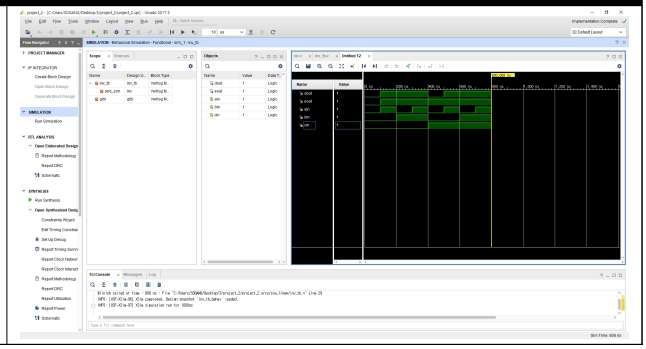
(B)

```
`timescale 1ns / 1ps

module inv_tb;

wire dout, eout;
reg ain, bin, cin;

inv and_sim(.ina(ain),
.inb(bin),
.inc(cin),
.outd(dout),
.oute(eout));

initial
begin
ain = 1'b0;
bin = 1'b0;
cin = 1'b0;
end

always@(ain or bin or cin) begin
ain <= #100 ~ain;
bin <= #200 ~bin;
cin <= #400 ~cin;
end

initial begin
```

$finish; end endmodule	#800 $finish; end endmodule

[표 Ⅱ-3-(3)] (A)와 (B)의 Verilog Simulation 코드

3) (A)와 (B)의 Simulation 결과

(A)	(B)
	

[표 Ⅱ-3-(4)] (A)와 (B)의 Simulation 결과

4) (A)와 (B)의 진리표

(A)

A	B	C	D
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(B)

A	B	C	D	E
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

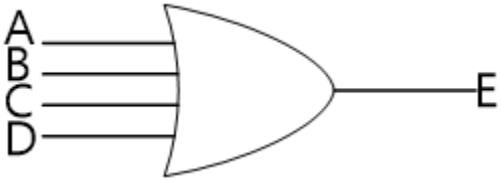
[표 Ⅱ-3-(5)] (A)와 (B)의 진리표

5) 설명

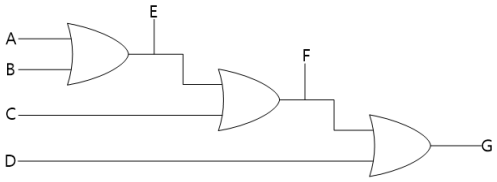
(A)에서는 A와 B와 C의 input 값이 모두 0일 때 D가 output 값으로 0이 출력되고, 그 외에는 모두 1이 출력된다. (B)에서는 A와 B의 값이 0일 때 D가 output 값으로 0이 출력되고, 그 외에는 모두 1이 출력된다. C의 값과 D의 값이 모두 0일 때 E가

output 값으로 0이 출력되고, 그 외에는 모두 1이 출력된다. 따라서 A, B, C가 모두 0일 때 E가 output 값으로 0이 출력된다.

4. 4-input OR Gate



(A)의 논리 회로도



(B)의 논리 회로도

1) (A)와 (B)의 Boolean 식 비교

(A)	(B)
$A+B+C+D=E$	$A+B=E$ $(A+B)+C=F$ $((A+B)+C)+D=G$

[표 Ⅱ-4-(1)] (A)와 (B)의 Boolean 식 비교

2) (A)와 (B)의 Verilog 코딩

(A)	(B)

`timescale 1ns / 1ps
 	
module inv(ina, inb, inc, ind, oute);
 	
input ina, inb, inc, ind;
output oute;
 	
or(oute, ina, inb, inc, ind);
 	
endmodule
	`timescale 1ns / 1ps
	
	module inv(ina, inb, inc, ind, oute, outf, outg);
	
	input ina, inb, inc, ind;
	output oute, outf, outg;
	
	or(oute, ina, inb);
	or(outf, oute, inc);
	or(outg, outf, ind);
	
	endmodule

[표 Ⅱ-4-(2)] (A)와 (B)의 Verilog Design Source 코드

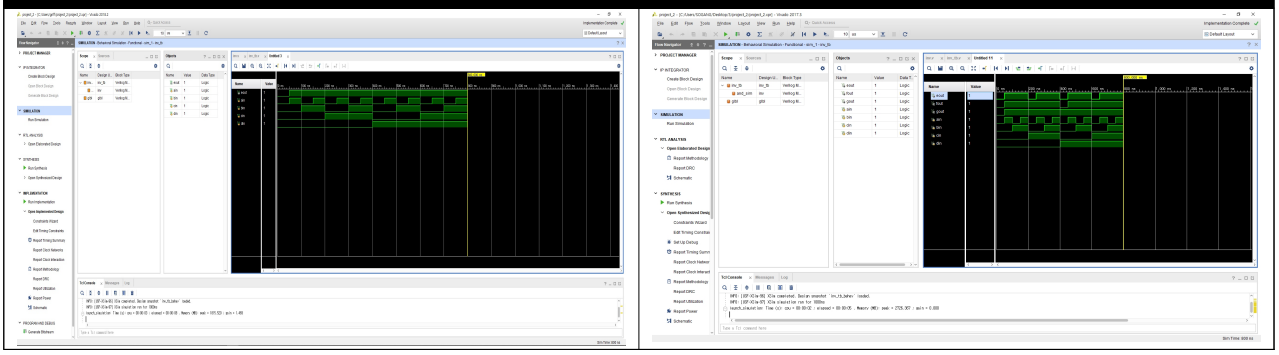
(A)	(B)

`timescale 1ns / 1ps
 	
module inv_tb;
 	
wire eout;
reg ain, bin, cin, din;
 	
inv and_sim(.ina(ain),
.inb(bin),
.inc(cin),
.ind(din),
.oute(eout));
 	
initial
begin
ain = 1'b0;
bin = 1'b0;
cin = 1'b0;
din = 1'b0;
end
 	
always@(ain or bin or cin or din) begin
ain <= #50 ~ain;
bin <= #100 ~bin;
cin <= #200 ~cin;
din <= #400 ~din;
end
 	
initial begin
#800
$finish;
end
 	
endmodule
	`timescale 1ns / 1ps
	
	module inv_tb;
	
	wire eout, fout, gout;
	reg ain, bin, cin, din;
	
	inv and_sim(.ina(ain),
	.inb(bin),
	.inc(cin),
	.ind(din),
	.oute(eout),
	.outf(fout)
	.outg(gout));
	
	initial
	begin
	ain = 1'b0;
	bin = 1'b0;
	cin = 1'b0;
	din = 1'b0;
	end
	
	always@(ain or bin or cin or din) begin
	ain <= #50 ~ain;
	bin <= #100 ~bin;
	cin <= #200 ~cin;
	din <= #400 ~din;
	end
	
	initial begin
	#800
	$finish;
	end
	
	endmodule

[표 Ⅱ-4-(3)] (A)와 (B)의 Verilog Simulation 코드

3) (A)와 (B)의 Simulation 결과

(A) (B)



[표 Ⅱ-4-(4)] (A)와 (B)의 Simulation 결과

4) (A)와 (B)의 진리표

(A)

A	B	C	D	E
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

(B)

A	B	C	D	E	F	G
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	1	1
0	1	0	1	1	1	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

[표 Ⅱ-4-(5)] (A)와 (B)의 진리표

5) 설명

(A)에서는 A, B, C, 그리고 D input 값이 모두 0일 때 E가 output 값으로 0이 출력되고, 그 외에는 모두 1이 출력된다. (B)에서는 A와 B의 값이 0일 때 E가 output 값으로 0이 출력되고, 그 외에는 모두 1이 출력된다. C의 값과 E의 값이 모두 0일 때 F가 output 값으로 0이 출력되고, 그 외에는 모두 1이 출력된다. 마지막으로 D의 값과 F의 값이 모두 0일 때 G가 output 값으로 0이 출력된다. 따라서 A, B, C, D가 모두 0일 때 G가 output 값으로 0이 출력된다.

Ⅲ 논의

1. 결과 검토 및 논의사항

다중 AND와 OR 게이트 회로 동작을 실험하면서 두 게이트의 input에 따른 output 값이 서로 달라짐을 알 수 있다. AND 게이트는 input 값들이 모두 1일 때 output 값이 1로 출력되고, 그 외의 나머지 경우에는 출력 값이 모두 0임을 알 수 있다. 다시 말해서, input 값들에 하나라도 0인 값이 존재하면 output 값은 0이 된다. 반면에 OR 게이트는 input 값들이 모두 0일 때 output 값이 0으로 출력되고, 그 외의 나머지 경우에는 출력 값이 모두 1임을 알 수 있다. 다시 말해서, input 값들에 하나라도 1인 값이 존재하면 output 값은 1이 된다.

input 값들이 어떤 논리식의 형태로 들어오는가에 따라서도 실험을 해보았다. (A) 회로도와 같이 input 값들에 모두 한 번에 논리연산을 적용할 때. AND 게이트에서는 모든 input 값이 1이어야 output 값이 1로 출력되고 OR 게이트에서는 모든 input 값이 0이어야 output 값이 0으로 출력된다. (B) 회로도에서도 마지막에 나오는 output 출력 값은 (A) 회로도와 같지만, (B) 회로도에서는 중간 과정에서 연산되어 나오는 output 값들이 존재한다. 이를 통해 같은 논리 연산자 사이에서는 결합법칙이 성립한다는 것을 알 수 있다. 예컨대 $((A \cdot B) \cdot C) \cdot D = E$의 output 값은 $A \cdot B \cdot C \cdot D = E$의 output 값과 같으므로 두 논리식은 동치관계임을 확인할 수 있다. AND 논리연산 외에 OR 연산도 마찬가지로 성립한다.

2. 추가 이론 조사 및 작성

AND, OR, 그리고 NOT 논리 연산을 응용한 Implication 연산도 존재한다. 논리식은 $x \subset y$ 또는 $x \supset y$이며, $x \subset y$는 $x + y'$와 동치이고 $x \supset y$는 $x' + y$와 동치이다.

AND와 OR 논리 연산을 사용한 Canonical Forms가 존재하는데, AND 연산을 사용하여 input들로 간단히 나타낸 최소항(Midterm)이 존재하고 OR 연산을 사용하여 input들로 나타낸 최대항(Maxterm)이 존재한다.

AND와 OR 게이트는 NAND 게이트와 NOR 게이트를 사용하여 구현할 수도 있다. 전자보다 후자를 사용하는 것이 상대적으로 더 효율적이기 때문인데, 이에 관한 내용은 4주차 예비보고서에 서술하였다. 아래의 표는 AND와 OR 게이트를 NAND와 NOR 게

이트로 구현한 것을 보여준다.

AND 게이트	OR 게이트