7주차 결과보고서

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

1. even parity bit generator 및 checker의 simulation 결과 및 과정에 대해 설명하시오. (진리표 작성 및 k-map 포함)

a. even parity bit generator

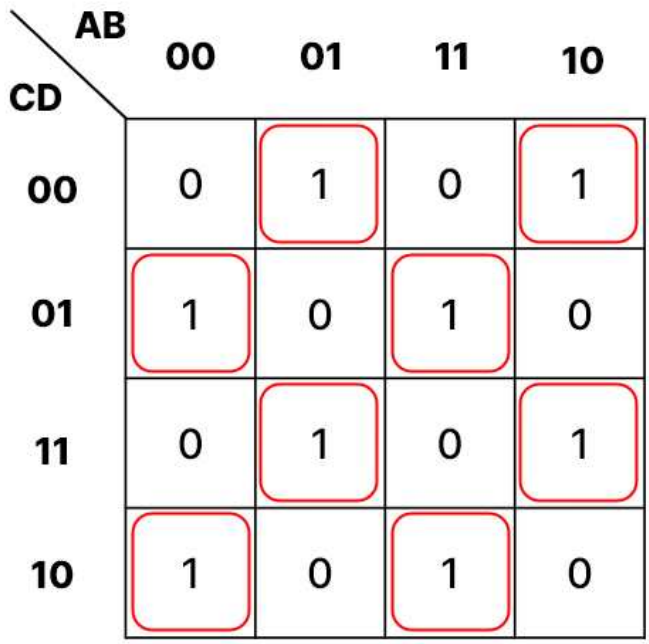
source code

```
`timescale 1ns / 1ps

module generator(
    input a, b, c, d,
    output out1
    );

assign out1 = a^b^c^d;

endmodule
```

testbench code

```
`timescale 1ns / 1ps

module generator_tb;

reg aa, bb, cc, dd;
wire out1;

generator u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.out1 (out1)
);

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
```

Karnaugh Map

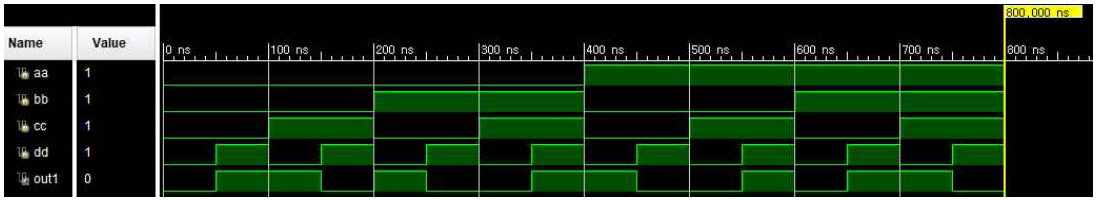


parity generator truth table

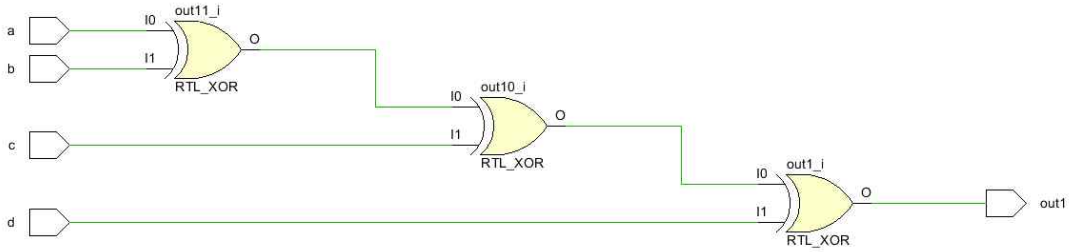
input				output
a	b	c	d	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

SOP form (by K-map)	
P	a^b^c^d

even parity bit generator simulation



even parity bit generator circuit design



even parity bit generator는 전송된 이진 데이터에서 '1'의 개수가 홀수 개이면 1을, '1'의 개수가 짝수 개이면 0을 출력한다. 따라서, 결과적으로 전송되는 이진 데이터에서 '1'의 개수는 항상 짝수 개가 된다. 따라서, binary data의 각 bit를 전부 XOR 연산하면 even parity bit를 생성할 수 있다. 이 경우 4bit data가 입력되므로 even parity bit는, 다음과 같다.

$$EPB = a \oplus b \oplus c \oplus d$$

실험 결과, EPB의 값이 해당 식을 만족하고 출력된 데이터에서 1이 홀수 개 출력된 경우가 없으므로 성공적으로 even parity bit가 출력되었다.

b. even parity bit checker

source code

```
`timescale 1ns / 1ps

module checker(
    input a, b, c, d, e,
    output out1 // PEC
    );

assign out1 = a^b^c^d^e;

endmodule
```

testbench code

```
`timescale 1ns / 1ps

module checker_tb;

reg aa, bb, cc, dd, ee;
wire out1;

checker u_test(
.a (aa), .b (bb), .c (cc), .d (dd), .e (ee),
.out1 (out1)
);

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

always@(aa or bb or cc or dd or ee)
begin
    aa <= #400 ~aa;
    bb <= #200 ~bb;
    cc <= #100 ~cc;
    dd <= #50 ~dd;
    ee <= #25 ~ee;
end

initial begin
    #800
    $finish;
end

endmodule
```

parity checker truth table

input					out
a	b	c	d	e	PEC
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	1
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	0	1	1
1	1	0	1	0	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	1

SOP form (by K-map)	
PEC	a^b^c^d^e

Karnaugh Map

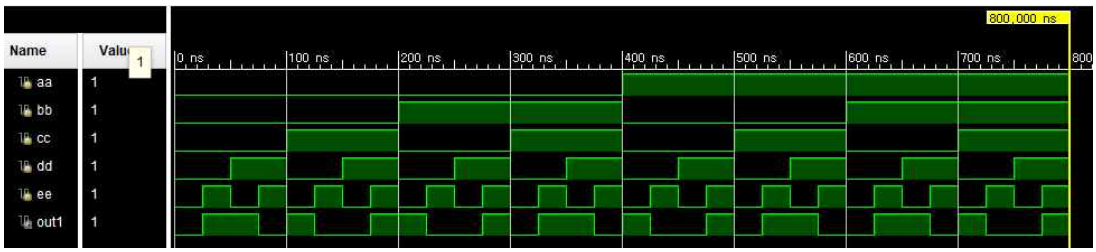
A = 0

DE \ BC	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

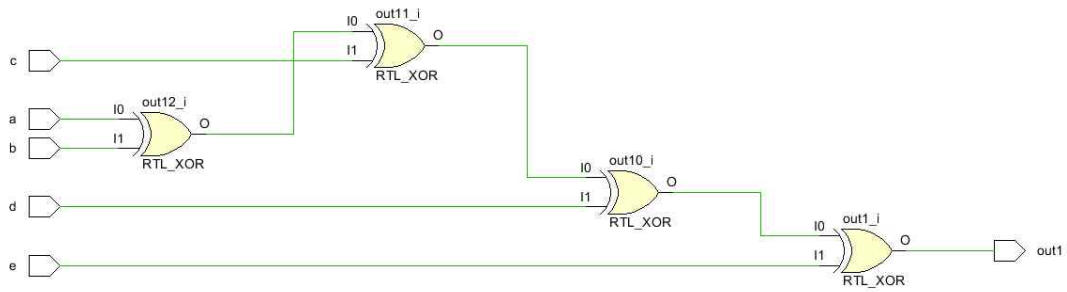
A = 1

DE \ BC	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

even parity bit checker simulation



even parity bit checker circuit design



even parity bit checker는 even parity bit를 포함한 이진 데이터에서 '1'의 개수가 홀수 개이면 1, 짝수 개이면 0을 출력한다. even parity bit가 추가되었을 뿐 동작 원리는 generator와 같으므로, 각 bit를 전부 XOR 연산해 even parity를 확인할 수 있다.

$$EPC = a \oplus b \oplus c \oplus d \oplus EPB$$

실험 결과, EPB의 값이 해당 식을 만족하고 5bit data에서 1이 홀수 개 출력된 경우에만 EPC가 1로 출력되었으므로 성공적으로 even parity checker가 구현되었다.

2. odd parity bit generator 및 checker의 simulation 결과 및 과정에 대해 설명하시오. (진리표 작성 및 k-map 포함)

a. odd parity bit generator

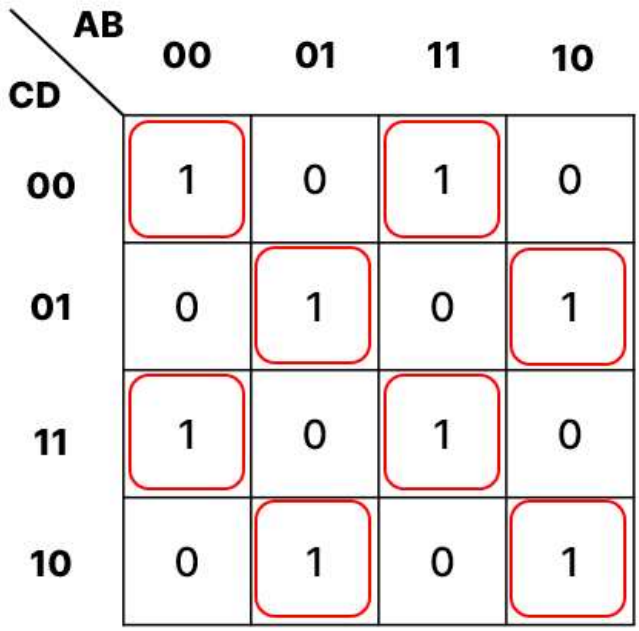
source code

```
`timescale 1ns / 1ps

module generator(
    input a, b, c, d,
    output out1
    );

assign out1 = ~(a^b^c^d);

endmodule
```

testbench code

```
`timescale 1ns / 1ps

module generator_tb;

reg aa, bb, cc, dd;
wire out1;

generator u_test(
.a (aa), .b (bb), .c (cc), .d (dd),
.out1 (out1)
);

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
```

Karnaugh Map

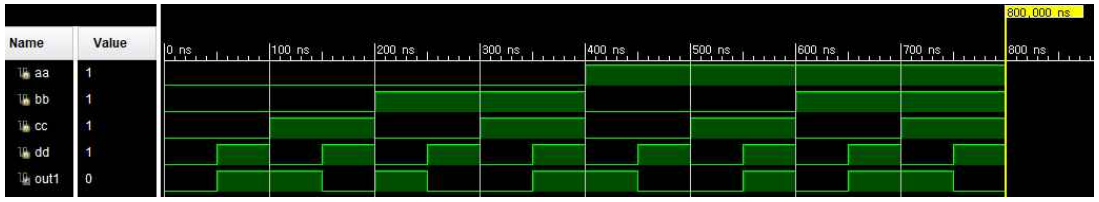


parity generator truth table

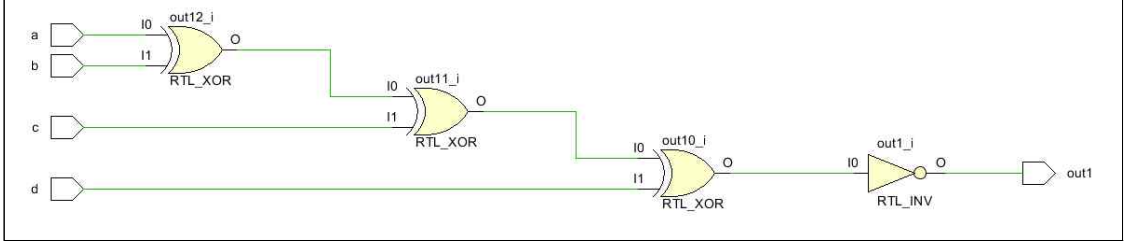
input				output
a	b	c	d	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

SOP form (by K-map)	
P	~(a^b^c^d)

odd parity bit generator simulation



odd parity bit generator circuit design



odd parity bit generator는 전송된 이진 데이터에서 '1'의 개수가 홀수 개이면 0을, '1'의 개수가 짝수 개이면 1을 출력한다. 따라서, 결과적으로 전송되는 이진 데이터에서 '1'의 개수는 항상 홀수 개가 된다. 따라서, binary data의 각 bit를 전부 XOR 연산한 뒤 NOT 연산하면 odd parity bit를 생성할 수 있다. 이 경우 4bit data가 입력되므로 odd parity bit는, 다음과 같다.

$$OPB = \overline{a \oplus b \oplus c \oplus d}$$

실험 결과, OPB의 값이 해당 식을 만족하고 출력된 데이터에서 1이 홀수 개 출력된 경우가 없으므로 성공적으로 odd parity bit가 출력되었다.

b. odd parity bit checker

source code

```
`timescale 1ns / 1ps

module checker(
    input a, b, c, d, e,
    output out1 // PEC
    );

assign out1 = ~(a^b^c^d^e);

endmodule
```

testbench code

```
`timescale 1ns / 1ps

module checker_tb;

reg aa, bb, cc, dd, ee;
wire out1;

checker u_test(
.a (aa), .b (bb), .c (cc), .d (dd), .e (ee),
.out1 (out1)
);

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

always@(aa or bb or cc or dd or ee)
begin
    aa <= #400 ~aa;
    bb <= #200 ~bb;
    cc <= #100 ~cc;
    dd <= #50 ~dd;
    ee <= #25 ~ee;
end

initial begin
    #800
    $finish;
end

endmodule
```

parity checker truth table

input					out
a	b	c	d	e	PEC
0	0	0	0	0	1
0	0	0	0	1	0
0	0	0	1	0	0
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	0
0	1	0	0	0	0
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	0	1	0
0	1	1	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	0	1	1
1	0	0	1	0	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	1
1	1	1	0	0	0
1	1	1	0	1	1
1	1	1	1	0	1
1	1	1	1	1	0

SOP form (by K-map)	
PEC	~(a^b^c^d^e)

Karnaugh Map

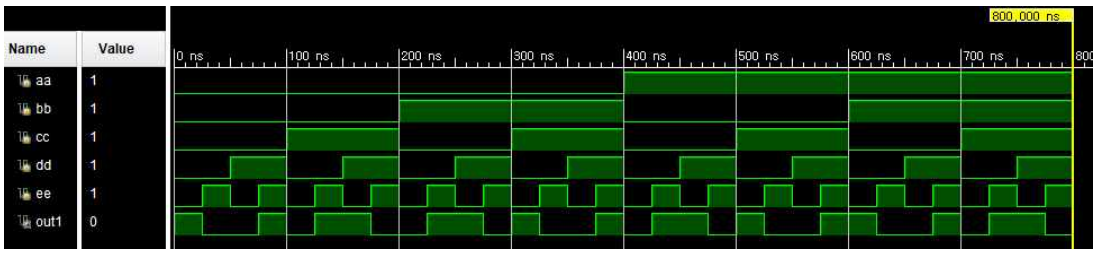
A = 0

DE \ BC	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

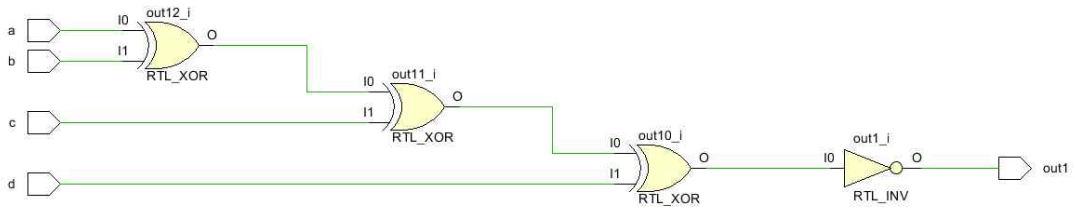
A = 1

DE \ BC	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

odd parity bit checker simulation



odd parity bit checker circuit design



odd parity bit checker는 odd parity bit를 포함한 이진 데이터에서 '1'의 개수가 홀수 개이면 0, 짝수 개이면 1을 출력한다. odd parity bit가 추가되었을 뿐 동작 원리는 generator와 같으므로, 각 bit를 전부 XOR 연산한 뒤 NOT 연산해 odd parity를 확인할 수 있다.

$$OPC = \overline{a \oplus b \oplus c \oplus d \oplus OPB}$$

실험 결과, OPB의 값이 해당 식을 만족하고 5bit data에서 1이 짝수 개 출력된 경우에만 OPC가 1로 출력되었으므로 성공적으로 odd parity checker가 구현되었다.

3. 2-bit binary comparator simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함)

source code

```
`timescale 1ns / 1ps

module comparator(
    input a1, a2, b1, b2,
    output G, Z, S // greater, smaller, equals
    );

assign G = (a1&~b1) | (a2&~b1&~b2) | (a1&a2&~b2);
assign Z = (~a1 & ~a2 & ~b1 & ~b2) | (~a1 & a2 & ~b1 & b2) | (a1 & ~a2 & b1 & ~b2) | (a1 & a2 & b1 & b2);
assign S = (~a1 & b1) | (~a2 & b1 & b2) | (~a1 & ~a2 & b2);

endmodule
```

testbench code

```
`timescale 1ns / 1ps

module comparator_tb;

reg a1, a2, b1, b2;
wire G, Z, S;

comparator u_test(
.a1 (a1), .a2 (a2), .b1 (b1), .b2 (b2),
.G (G), .Z (Z), .S (S)
);

initial begin
    a1 = 1'b0;
    a2 = 1'b0;
    b1 = 1'b0;
    b2 = 1'b0;
end

always@(a1 or a2 or b1 or b2) begin
    a1 <= #400 ~a1;
    a2 <= #200 ~a2;
    b1 <= #100 ~b1;
    b2 <= #50 ~b2;
end

initial begin
    #800
    $finish;
end

endmodule
```

2bit comparator truth table

input				output		
a_1	a_2	b_1	b_2	G	Z	S
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Karnaugh Map

G

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

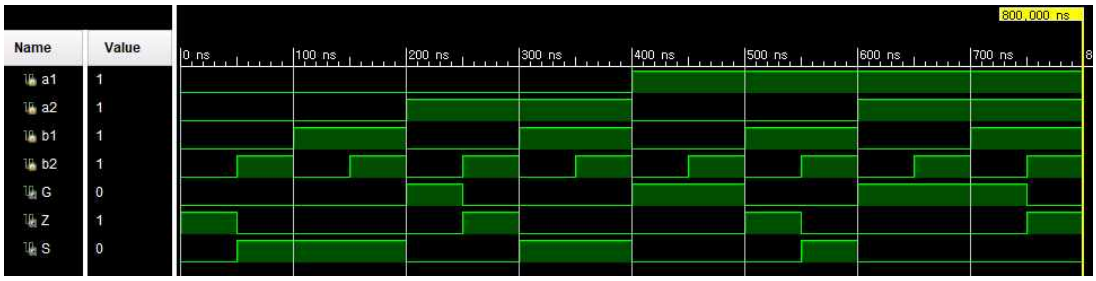
Z

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

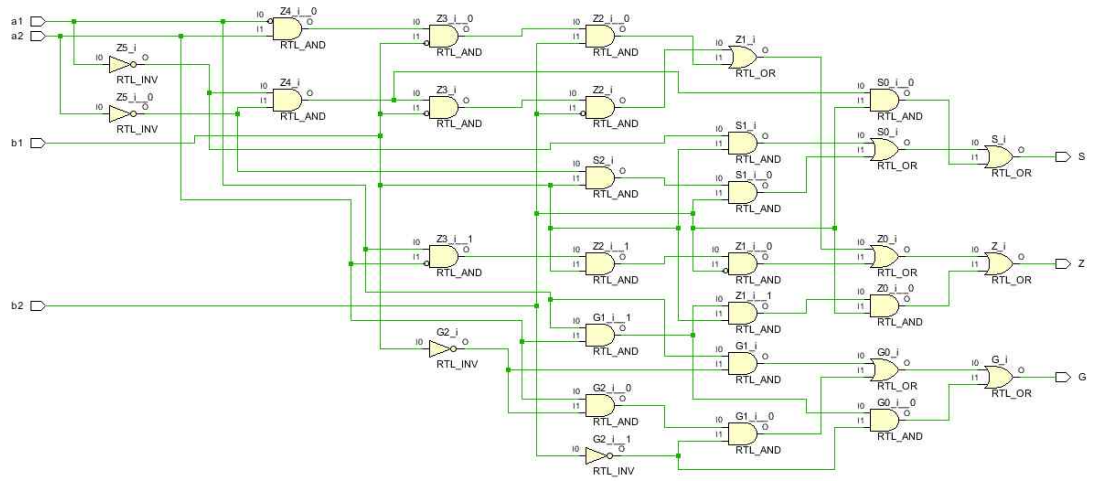
S

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

2bit compartator simulation



2bit compartator circuit design



2-bit 비교기는 2bit 이진수 A, B 둘을 비교하여 A>B, A=B, A<B의 결과를 출력하는 조합 논리 회로이다. MSB부터 비교해 먼저 대소를 비교하고, 그 결과가 같다면 다음 bit를 비교하는 방식으로 연산을 수행한다. 이 경우 A>B를 G, A=B를 Z, A<B를 S로 나타내면 다음과 같다.

SOP form (by K-map)	
G (A>B)	$a_1b_1' + a_2b_1'b_2' + a_1a_2b_2'$
Z (A=B)	$a_1a_2b_1b_2 + a_1'a_2'b_1'b_2' + a_1a_2'b_1b_2' + a_1'a_2b_1'b_2$
S (A<B)	$a_1'b_1 + a_2'b_1b_2 + a_1'a_2'b_2$

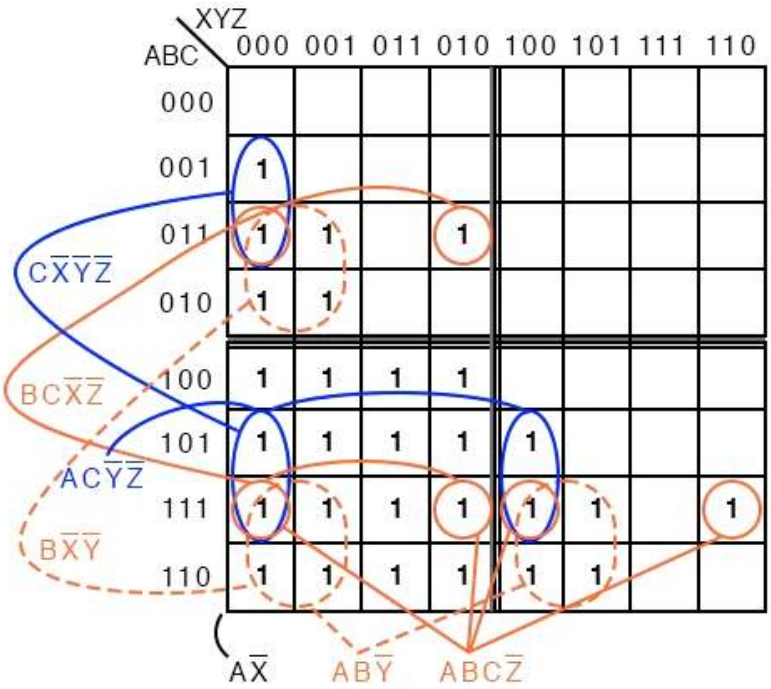
시뮬레이션 결과, G, Z, S의 논리적 동작과 실제 출력 결과가 동일하게 나타났으므로 정상적으로 2-bit 비교기가 구현된 것을 확인할 수 있었다.

4. 결과 검토 및 논의 사항

이번 실험에서는 even parity bit generator/checker, odd parity bit generator/checker, 2bit binary comparator를 각각 verilog 상에서 구현하여 시뮬레이션해 보았으며, FPGA 보드를 활용하여 실제 출력 결과를 확인해 보았다. 각 회로의 논리적 동작에 따라 진리표를 그리고, Karnaugh map을 그림으로써 SOP, POS 식을 도출했다. 실험 결과, 진리표를 바탕으로 기대되었던 동작이 정상적으로 이루어졌음을 확인했다.

5. 추가 이론 조사 및 작성

카르노 맵은 주로 6-input 입력에 대해서까지만 유용하다. 5-input 카르노맵까지는 4*4 grid를 두 개 그림으로써 작성했지만, 6-input 카르노 맵은 어떻게 그려야 할까?



$$Out = A\bar{X} + AB\bar{Y} + B\bar{X}\bar{Y} + ABC\bar{Z} + AC\bar{Y}\bar{Z} + BC\bar{X}\bar{Z} + C\bar{X}\bar{Y}\bar{Z}$$

6 - variable Karnaugh map (overlay)

다음과 같은 그림을 통해 그리는 방법을 살펴볼 수 있다. 한 줄마다 3변수를 할당하여 4개의 하위 맵으로 분할해 그린 뒤, 중복되는 표현식을 찾는다. 각 4개의 submap에서 공통되는 이진수에 따라 essential prime implicants를 찾는 논리는 동일하다.

위와 같은 방식이 어렵다면, 4변수 카르노 맵을 4개 그려 3차원으로 쌓아서 그릴 수도 있다. A, B, C, D, E, F 6변수에 대해 AB/CD 4변수 서브맵을 4개 그렸다면, EF = 00, 01, 10, 11에 대응하는 각 서브맵들 중에서 겹쳐져 있는 값을 찾아 각 EF 중 어떤 값이 공통되는지 확인하면 된다. 두 방법 모두 원리는 동일하므로 편한 방식으로 그리면 된다.