

5주차 결과보고서

전공: 경영학과

학년: 4학년

학번: 20190808

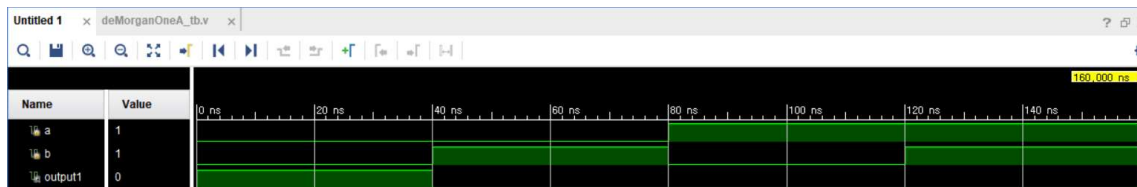
이름: 방지혁

1.

해당 실험의 목적은 드 모르간의 제1, 2 법칙과 Boolean 함수의 동작 및 1 bit 비교기를 Verilog를 사용하여 나타내고, 결과를 이해하는 것입니다. 더욱 자세히 설명하자면 변수에 보수를 취할 때, 논리합과 논리곱 간의 변환 관계를 이해하고, 1 bit 비교기를 verilog의 >, < 연산자가 아니라 xor, xnor gate 및 boolean 함수를 통해 구현하고자 합니다. simulation에서 그치지 않고, 최종적으로 설계를 FPGA 보드에 올려, 동작을 검증하고, 입력 신호에 따른 출력 결과를 분석하고자 합니다.

2.

우선 첫 번째로 설명할 것은 de-morgan 제 1 법칙 중 a 항목입니다.



Input a	Input b	Output a
0	0	1
0	1	0
1	0	0
1	1	0

```
`timescale 1ns / 1ps

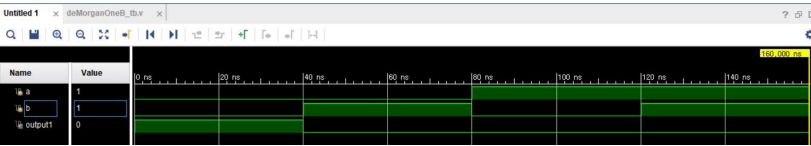
module deMorganOneA(
    input a, b,
    output output1
);

    assign output1 = ~(a|b);

endmodule
```

```
1 `timescale 1ns / 1ps
2
3 module deMorganOneA_tb;
4
5 reg a, b;
6 wire output1;
7
8 deMorganOneA u_test(
9     .a(a),
10    .b(b),
11
12    .output1(output1)
13 );
14
15 initial begin
16     a = 1'b0;
17     b = 1'b0;
18 end
19
20 always@(a or b) begin
21     a <= #80 ~a;
22     b <= #40 ~b;
23 end
24
25 initial begin
26     #160
27     $finish;
28 end
29 endmodule
```

다음은 de-morgan 제 1 법칙 중 b 항목입니다.



Input a	Input b	Output b
0	0	1
0	1	0
1	0	0
1	1	0

```
`timescale 1ns / 1ps

module deMorganOneB(
    input a, b,
    output output1
);

    assign output1 = (~a)&(~b);

endmodule
```

```
`timescale 1ns / 1ps

module deMorganOneB_tb;

    reg a, b;
    wire output1;

    deMorganOneB u_test(
        .a(a),
        .b(b),
        .output1(output1)
    );

    initial begin
        a = 1'b0;
        b = 1'b0;
    end

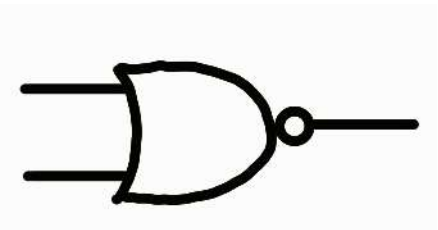
    always@(a or b) begin
        a <= #80 ~a;
        b <= #40 ~b;
    end

    initial begin
        #160
        $finish;
    end

endmodule
```

가장 처음의 simulation output은 (A) 항목의 (A+B)’ 결과이며, 두 번째 simulation의 output는 (B) 항목의 A’*B’ 결과입니다. 보다시피 <A>와 의 결과인 Output simulation은 두 개 다 같습니다. 이를 통해 드모르간의 제 1법칙에 의해 ~(A+B)와 (~A)·(~B)의 결과는 같은 것을 알 수 있습니다.

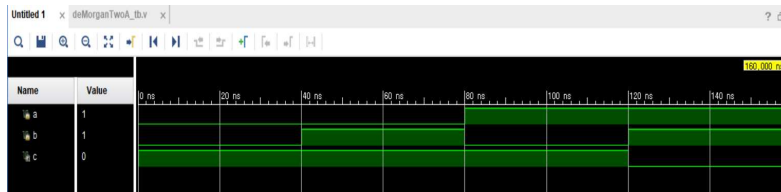
이를 NOR gate와 비교해보겠습니다. NOR 게이트는 OR 게이트에 NOT을 연결시킨 형태입니다.



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

진리표의 결과도 동일하며, (A)의 (A + B)’, (B)의 (A’*B’)와 nor 게이트 모두 동일하다는 것을 알 수 있습니다.

다음으로 설명할 것은 de-morgan 제 2 법칙 중 a 항목입니다.



Input a	Input b	Output c
0	0	1
0	1	1
1	0	1
1	1	0

```

`timescale 1ns / 1ps

module deMorganTwoA(
    input a, b,
    output c
);

assign c = ~(a&b);

endmodule

```

```

`timescale 1ns / 1ps

module deMorganTwoA_tb;

reg a, b;
wire c;

deMorganTwoA test(
    .a(a),
    .b(b),
    .c(c)
);

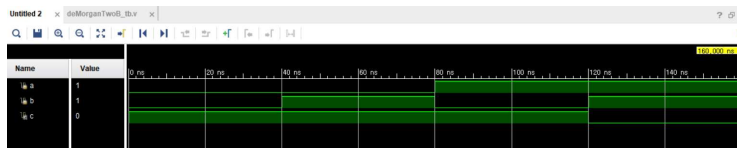
initial begin
    a = 1'b0;
    b = 1'b0;
    #160
    $finish;
end

always@(a or b) begin
    a <= #80 ~a;
    b <= #40 ~b;
end

endmodule

```

하단의 사진들은 de-morgan 제 2 법칙 중 b 항목입니다.



Input a	Input b	Output c
0	0	1
0	1	1
1	0	1
1	1	0

```

`timescale 1ns / 1ps

module deMorganTwoB(
    input a, b,
    output c
);

assign c = ~(a&b);

endmodule

```

```

`timescale 1ns / 1ps

module deMorganTwoB_tb;

reg a, b;
wire c;

deMorganTwoB test(
    .a(a),
    .b(b),
    .c(c)
);

initial begin
    a = 1'b0;
    b = 1'b0;
    #160
    $finish;
end

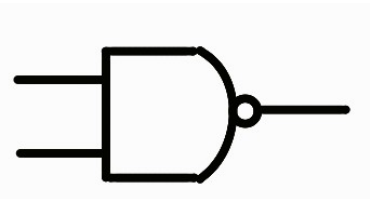
always@(a or b) begin
    a <= #80 ~a;
    b <= #40 ~b;
end

endmodule

```

가장 처음의 simulation output은 (A) 항목의 $(A*B)'$ 결과이며, 두 번째 simulation의 output는 (B) 항목의 $A'+B'$ 결과입니다. 보다시피 <A>와 의 결과인 Output simulation은 두 개 다 같습니다. 이를 통해 드모르간의 제 2법칙에 의해 $(A*B)'$ 와 $A'+B'$ 의 결과는 같은 것을 알 수 있습니다.

이를 NAND gate와 비교해보겠습니다. NAND 게이트는 AND 게이트에 NOT을 연결시킨 형태입니다.

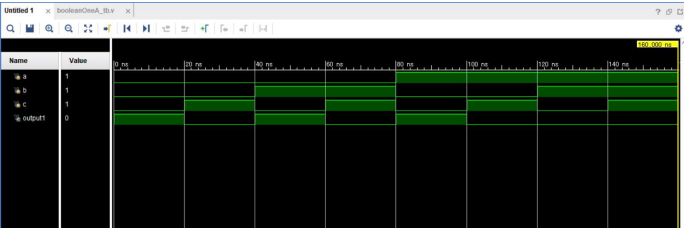


A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

진리표의 결과도 동일하며, (A)의 $(A \cdot B)'$, (B)의 $A' + B'$ 와 nand 게이트 모두 동일하다는 것을 알 수 있습니다.

3.

boolean 함수 $(A' + B') \cdot C'$ 의 결과입니다. (9 page - (A))



A	B	C	Output
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

`timescale 1ns / 1ps

```
module booleanOneA(
    input a, b, c,
    output output1
);

    assign output1 = (~a|~b)&~c;
endmodule
```

`timescale 1ns / 1ps

```
module booleanOneA_tb;
```

```
    reg a, b, c;
    wire output1;
```

```
    booleanOneA u_test(
        .a(a),
        .b(b),
        .c(c),
```

```
        .output1(output1)
    );
```

```
    initial begin
```

```
        a = 1'b0;
```

```
        b = 1'b0;
```

```
        c = 1'b0;
```

```
    end
```

```
    always@(a or b or c) begin
```

```
        a <= #80 ~a;
```

```
        b <= #40 ~b;
```

```
        c <= #20 ~c;
```

```
    end
```

```
    initial begin
```

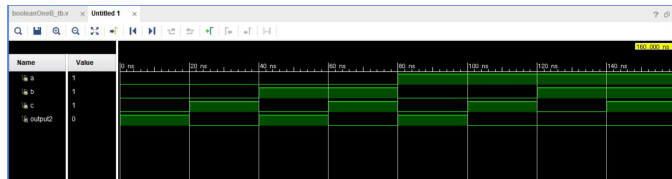
```
        #160
```

```
        $finish;
```

```
    end
```

```
endmodule
```

다음은 boolean 함수 $((A \cdot B) + C)'$ 의 결과입니다. (9 page - (B))



A	B	C	Output
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

```

1 `timescale 1ns / 1ps
2
3 module booleanOneB(
4     input a, b, c,
5     output output2
6 );
7
8     assign output2 = ~((a&b)|c);
9 endmodule

```

```

1 `timescale 1ns / 1ps
2
3 module booleanOneB_tb;
4
5     reg a, b, c;
6     wire output2;
7
8     booleanOneB u_test(
9         .a(a),
10        .b(b),
11        .c(c),
12
13        .output2(output2)
14    );
15
16    initial begin
17        a = 1'b0;
18        b = 1'b0;
19        c = 1'b0;
20        end
21
22    always@(a or b or c) begin
23        a <= #80 ~a;
24        b <= #40 ~b;
25        c <= #20 ~c;
26        end
27
28    initial begin
29        #160
30        $finish;
31        end
32
33 endmodule
34

```

결과인 Output이 동일한 것으로 보아 $(A' + B') \cdot C' = ((A \cdot B) + C)'$ 임을 알 수 있습니다.
이는 드모르간의 법칙을 통해 확인할 수도 있습니다.

다음은 boolean 함수 $(A' \cdot B') + C'$ 의 결과입니다. (11 page - (A))



A	B	C	Output
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

```

1 `timescale 1ns / 1ps
2
3 module booleanTwoA(
4     input a, b, c,
5     output output3
6 );
7
8     assign output3 = (~a&~b)|~c;
9 endmodule

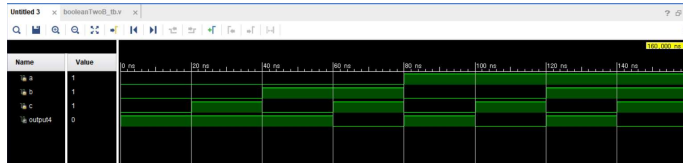
```

```

1 `timescale 1ns / 1ps
2
3 module booleanTwoA_tb;
4
5     reg a, b, c;
6     wire output3;
7
8     booleanTwoA u_test(
9         .a(a),
10        .b(b),
11        .c(c),
12
13        .output3(output3)
14    );
15
16    initial begin
17        a = 1'b0;
18        b = 1'b0;
19        c = 1'b0;
20        end
21
22    always@(a or b or c) begin
23        a <= #80 ~a;
24        b <= #40 ~b;
25        c <= #20 ~c;
26        end
27
28    initial begin
29        #160
30        $finish;
31        end
32
33 endmodule
34

```

다음은 boolean 함수 $((A + B) \cdot C)'$ 의 결과입니다. (11 page - (B))



A	B	C	Output
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

```

1 `timescale 1ns / 1ps
2
3 module booleanTwoB(
4     input a, b, c,
5     output output4
6 );
7
8     assign output4 = ~((a|b)&c);
9 endmodule

```


```

1 `timescale 1ns / 1ps
2
3 module booleanTwoB_tb;
4
5     reg a, b, c;
6     wire output4;
7
8     booleanTwoB u_test(
9         .a(a),
10        .b(b),
11        .c(c),
12
13        .output4(output4)
14    );
15
16    initial begin
17        a = 1'b0;
18        b = 1'b0;
19        c = 1'b0;
20        end
21
22    always@(a or b or c) begin
23        a <= #80 ~a;
24        b <= #40 ~b;
25        c <= #20 ~c;
26        end
27
28    initial begin
29        #160
30        $finish;
31        end
32
33 endmodule

```

결과인 Output이 동일한 것으로 보아 $(A' \cdot B') + C' = ((A + B) \cdot C)'$ 임을 알 수 있습니다. 이는 드모르간의 법칙을 통해 확인할 수도 있습니다.

4.



Input a	Input b	Output c (a=b)	Output d (a!=b)	Output e(a>b)	Output f(a<b)
0	0	1	0	0	0
0	1	0	1	0	1
1	0	0	1	1	0
1	1	1	0	0	0

```

1 `timescale 1ns / 1ps
2
3 module bit_tb;
4
5     reg a, b;
6     wire equal, notequal, leftbig, rightbig;
7
8     bit u_test(
9         .a(a),
10        .b(b),
11
12        .equal(equal),
13        .notequal(notequal),
14        .leftbig(leftbig),
15        .rightbig(rightbig)
16    );
17
18    initial begin
19        a = 1'b0;
20        b = 1'b0;
21        end
22
23    always@(a or b) begin
24        a <= #80 ~a;
25        b <= #40 ~b;
26        end
27
28    initial begin
29        #160
30        $finish;
31    end
32
33 endmodule
        
```

```

`timescale 1ns / 1ps

module bit(
    input a, b,
    output equal, notequal, leftbig, rightbig
);

assign equal = ~(a^b);
assign notequal = a^b;
assign leftbig = a&(~b);
assign rightbig = (~a)&b;

endmodule
        
```

동일 여부 비교는 $A \text{ XNOR } B$ 연산으로 구현했고, 차이 여부 비교는 $A \text{ XOR } B$ 연산으로 구현했습니다. $A > B$ 연산은 $A * B'$ 식으로, $A < B$ 연산은 $A' * B$ 로 구현할 수 있습니다. 대소 비교 연산은 입력값이 2개뿐이기 때문입니다. 이렇게 4가지 1bit 비교기를 Boolean 연산을 통해 구현할 수 있었습니다.

5.

실제 simulation 결과는 제가 생각했었던 결과와 동일하게 성공적으로 나왔습니다. 두 가지 논리식의 시뮬레이션과 값의 비교를 했었습니다. 동일한 시뮬레이션 결과가 나왔다는 것을 확인하는 것을 통해 복잡한 논리식을 변환할 때 발생하는 equivalence를 직접 하드웨어적으로 구현해 볼 수 있었습니다. 또한, 1 bit 연산기를 구현하기 위해 비교연산자 뿐만 아니라 단순한 논리 연산을 통한 함수 구현으로 할 수 있다는 것을 깨달았습니다.

6.

비교연산자에 대해 더 조사를 해보았습니다. verilog에서는 다음과 같은 비교연산자들을 지원합니다. $>$: 크다, $<$: 작다, $>=$: 크거나 같다, $<=$: 작거나 같다, $==$: 같다, $!=$: 다르다
비교연산자는 결과가 참일 경우 1을, 거짓일 경우 0을 조건문에 활용하거나 1비트 레지스터나 wire에 할당할 수 있습니다.

2 bit 비교기

입력		출력			
X	Y	X=Y	X≠Y	X>Y	X<Y
X ₁ X ₂	Y ₁ Y ₂	F ₁	F ₂	F ₃	F ₄
00	00	1	0	0	0
	01	0	1	0	1
	10	0	1	0	1
	11	0	1	0	1
01	00	0	1	1	0
	01	1	0	0	0
	10	0	1	0	1
	11	0	1	0	1
10	00	0	1	1	0
	01	0	1	1	0
	10	1	0	0	0
	11	0	1	0	1
11	00	0	1	1	0
	01	0	1	1	0
	10	0	1	1	0
	11	1	0	0	0

이를 이렇게 논리회로로 구현할 수 있습니다.

