Digital System Design Lab 2

김지현 (20220302)

1. 개요

2-Bit magnitude Comparator 의 결과를 이용해 단순화 하기 전 식과 Karnaugh-Map 을 이용해 단순화 시킨 식을 비교하는 실습을 진행한다. 이번 lab2 에서는 gate-level modeling 외에 Behavioral modeling 을 사용해 assign, |, &, ~ 등을 이용해 식을 구현할 것이다. 그리고 schematic 과 nets 와 leaf cells 개수를 이용해 단순화 하기 전 후를 비교할 것이다.

2. 이론적 배경

불 대수 (Boolean Algebra) 란 true 와 false 로 표현되는 수식을 말한다. 기존의 Truth table 만을 이용해서 식을 작성하면 식이 길어질 뿐만 아니라 회로도를 작성할 때 사용되는 gate 와 와이어 갯수가 늘어나게 된다. 그렇기 때문에 Karnaugh map 과 Quine McCluskey 를 사용해 불 대수를 단순화 하는 것이다. Karnaugh map 같은 경우엔 실제로 적어서 단순화하기 쉽지만 만약에 variable 의 개수가 5 개 이상으로 늘어나게 된다면 3d 차트를 만들어야 하기 때문에 직접 적어서 단순화 시키기엔 어려움이 있다. 그렇기 때문에 컴퓨터가 확인 하는 방식인 Quine McCluskey 를 사용하는 것이다. Quine McCluskey 는 각 variable 을 0 과 1 의 조합으로 만든 뒤 1 의 갯수에 따라 나열하고 옆에 위치하는 값과 각 자릿수를 비교하여 결과적으로는 EPI 를 찾고 가장 단순화 된 식을 찾을 수 있는 방법 중 하나이다.

개요에서 설명한 2-Bit Magnitude Comparator 는 두 개의 2 비트 이진수를 비교하여 상대적인 크기를 결정하는 디지털 회로이다. 두 개의 2 비트 A0, A1, B0, B1 이 입력으로 들어오게 되면 A>B, A<B, 와 A=B 라는 세 개의 출력을 나타낸다. 밑의 표는 2 bit magnitude comparator 의 출력값을 truth table 로 나타내고 각 출력에 대해 단순화 하기 전의 식을 나타낸 것이다. (GT 는 greater than, LT 는 less than, 그리고 EQ 는 equal 을 나타낸다)

					A<B		A=B		A>B	
A_1	A_0	B_1	B_0		LT		EQ		GT	
0	0	0	0		0		1		0	m0
0	0	0	1		1		0		0	m1
0	0	1	0		1		0		0	m2
0	0	1	1		1		0		0	m3
0	1	0	0		0		0		1	m4
0	1	0	1		0		1		0	m5
0	1	1	0		1		0		0	m6
0	1	1	1		1		0		0	m7
1	0	0	0		0		0		1	m8
1	0	0	1		0		0		1	m9
1	0	1	0		0		1		0	m10
1	0	1	1		1		0		0	m11
1	1	0	0		0		0		1	m12
1	1	0	1		0		0		1	m13
1	1	1	0		0		0		1	m14
1	1	1	1		0		1		0	m15

LT: $A_1'A_0'B_1'B_0 + A_1'A_0'B_1B_0' + A_1'A_0B_1B_0 + A_1'A_0B_1B_0' + A_1A_0'B_1B_0 + A_1A_0'B_1B_0'$

$\Sigma m(1,2,3,6,7,11)$

EQ: $A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0'B_1B_0' + A_1A_0B_1B_0$

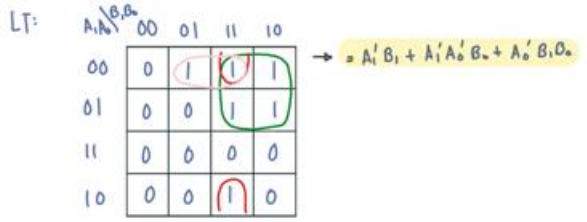
$\Sigma m(0,5,10,15)$

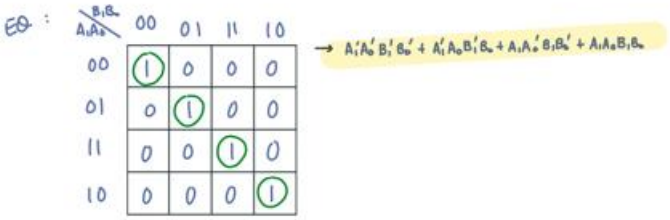
GT: $A_1'A_0B_1'B_0' + A_1A_0'B_1'B_0' + A_1A_0'B_1'B_0 + A_1A_0B_1'B_0' + A_1A_0B_1'B_0 + A_1A_0B_1B_0'$

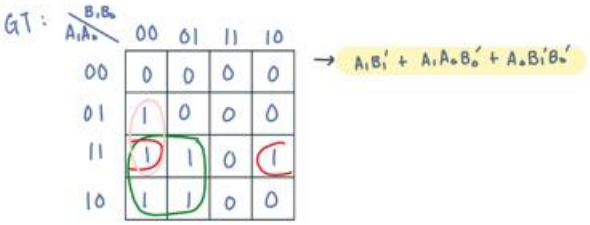
$\Sigma m(4,8,9,12,13,14)$

3. 실험준비

위와 같은 truth table 에서 1 (truth) 가 나온 수식끼리 묶어서 식을 만들게 되면 단순화 되지 않은 식이다. 단순화 하기 위해서는 밑에와 같은 Karnaugh map 을 사용하면 표를 만들어서 PI 와 EPI 들을 찾고 가장 단순화 되는 식을 찾을 수 있다.



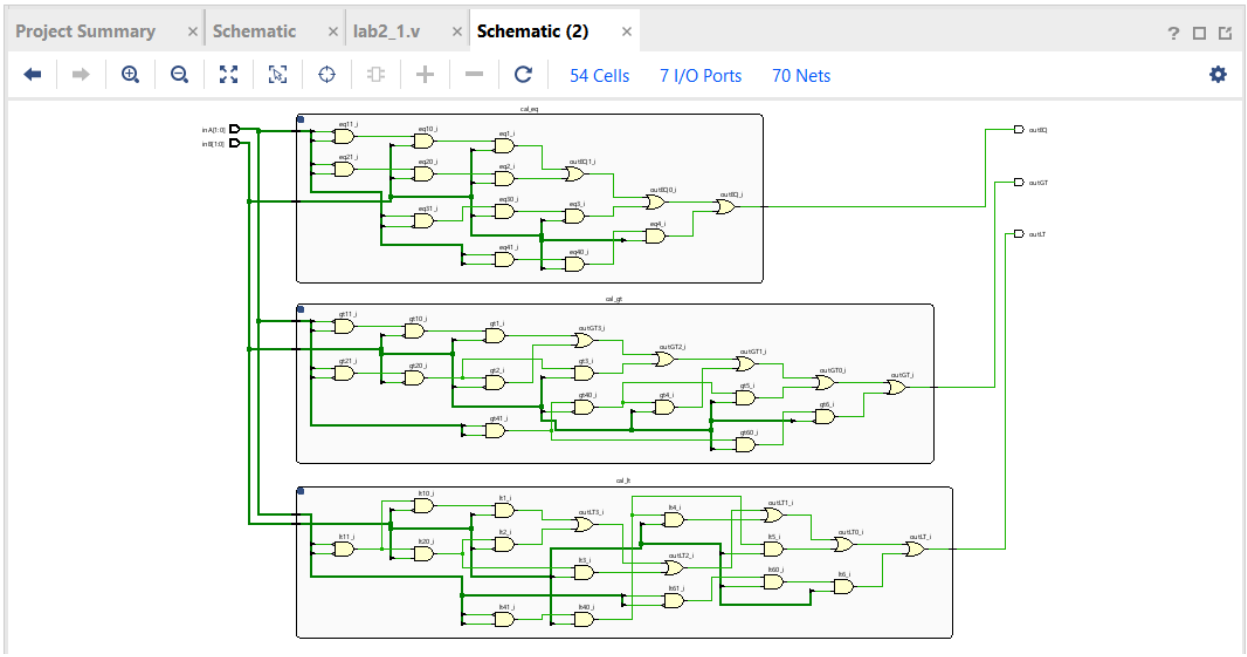


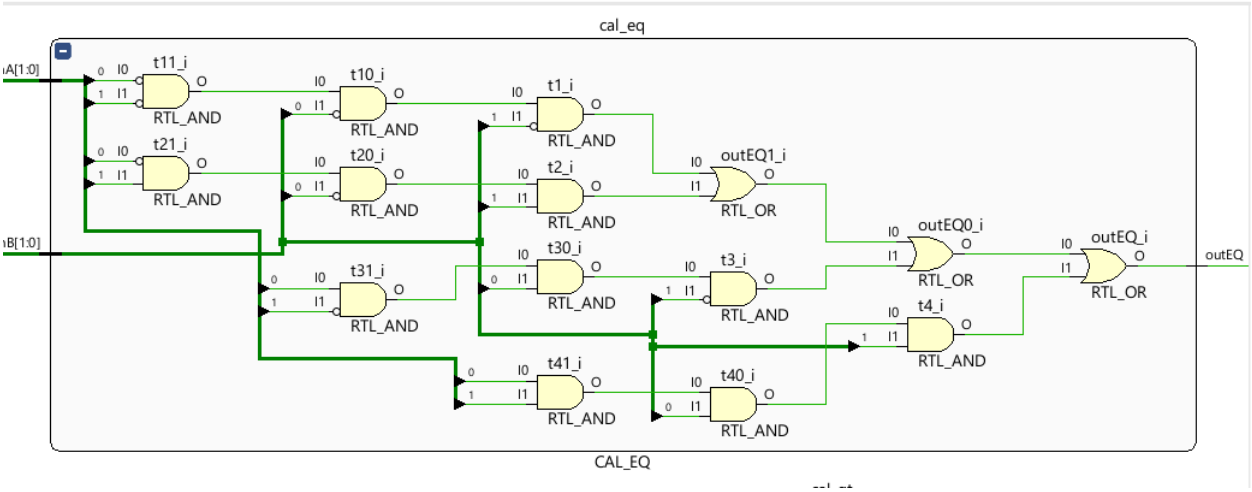


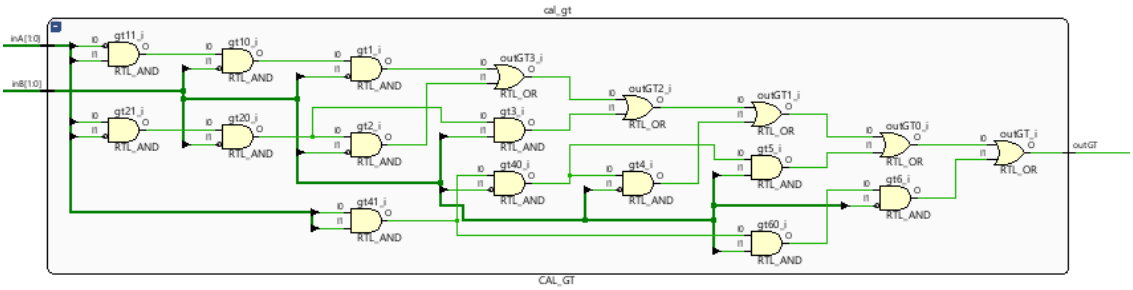
4. 결과

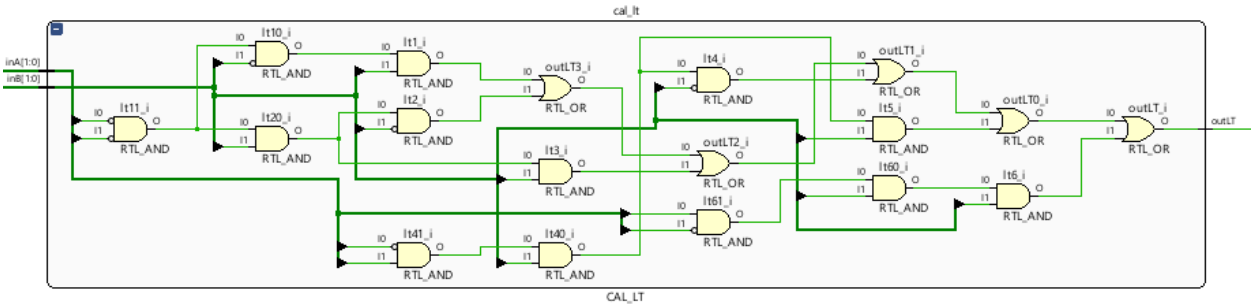
(1) Lab 2_1 (단순화 시키기 전)

밑의 회로도는 단순화 되기 전인 lab2_1 의 결과이고, 잇따른 3 개의 사진은 EQ, GT, LT 회로도를 확대한 것이다.


Project Summary
Schematic
lab2_1.v
Schematic (2)
54 Cells
7 I/O Ports
70 Nets


cal_eq
t11_i
t10_i
t1_i
t21_i
t20_i
t2_i
outEQ1_i
t30_i
t31_i
t3_i
outEQ0_i
outEQ_i
t4_i
t41_i
t40_i
RTL_AND
RTL_OR
outEQ
CAL_EQ


cal_gt
gt11_i
gt10_i
gt1_i
outGT3_i
outGT2_i
outGT1_i
gt21_i
gt20_i
gt2_i
gt3_i
gt5_i
outGT0_i
outGT_i
gt40_i
gt4_i
gt6_i
gt41_i
gt60_i
RTL_AND
RTL_OR
outGT
CAL_GT



Lab 2_1 에서는 실험 준비에서 첨부한 truth table 에서 얻은 밑과 같은 식을 세웠다.

LT : $A_1'A_0'B_1'B_0 + A_1'A_0'B_1B_0' + A_1'A_0B_1B_0 + A_1'A_0B_1B_0' + A_1'A_0B_1B_0 + A_1A_0'B_1B_0$

$\Sigma m(1, 2, 3, 6, 7, 11)$

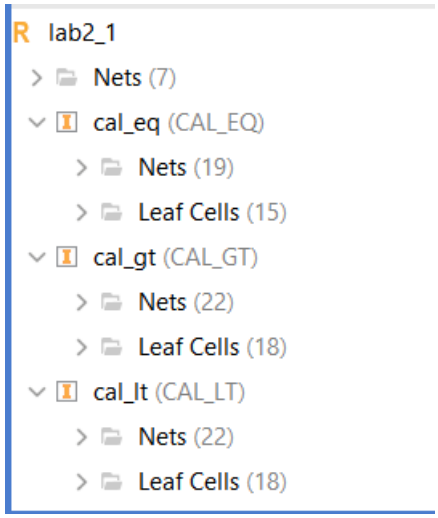
EQ : $A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0'B_1B_0' + A_1A_0B_1B_0$

$\Sigma m(0, 5, 10, 15)$

GT : $A_1'A_0B_1'B_0' + A_1A_0'B_1'B_0' + A_1A_0'B_1'B_0 + A_1A_0B_1'B_0' + A_1A_0B_1'B_0 + A_1A_0B_1B_0'$

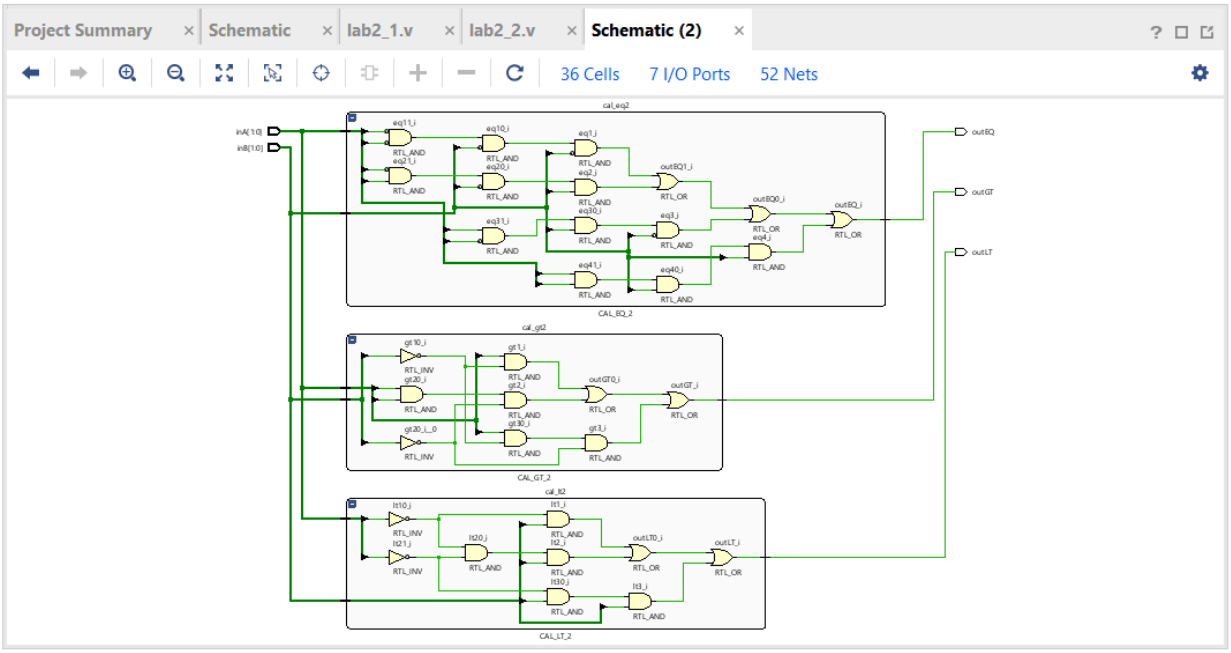
$\Sigma m(4, 8, 9, 12, 13, 14)$

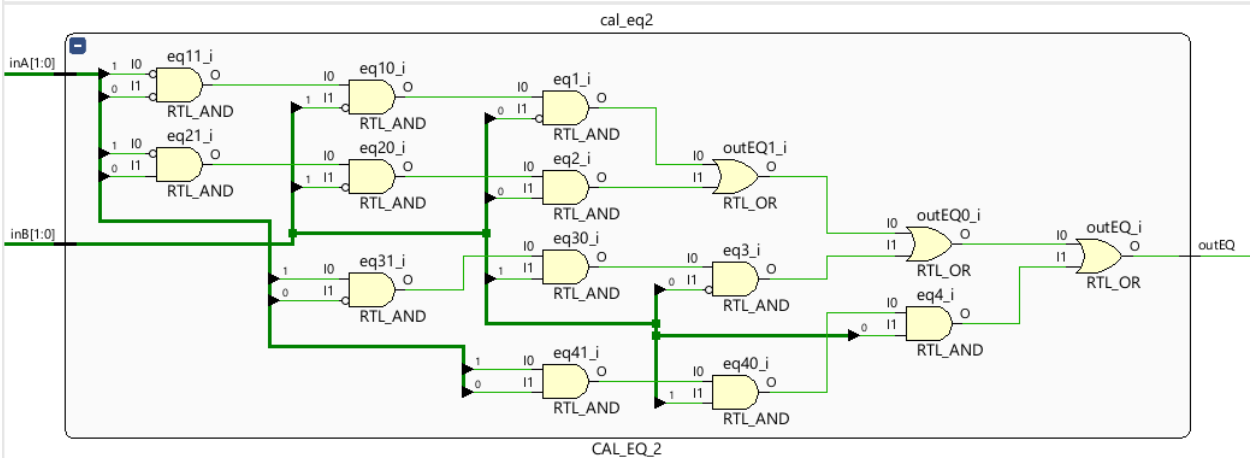
개요에서 언급했듯이 behavioral modeling 을 이용하면 assign 을 이용해 할당하고 * 는 & 을 이용하고 + 는 | 을 이용해 값을 저장했다. 그 결과 밑과 같은 nets 와 leaf cells 가 나왔다. 아직 단순화 되기 전이기 때문에 와이어와 논리게이트 모두 많이 사용되었다는 것을 알 수 있다.

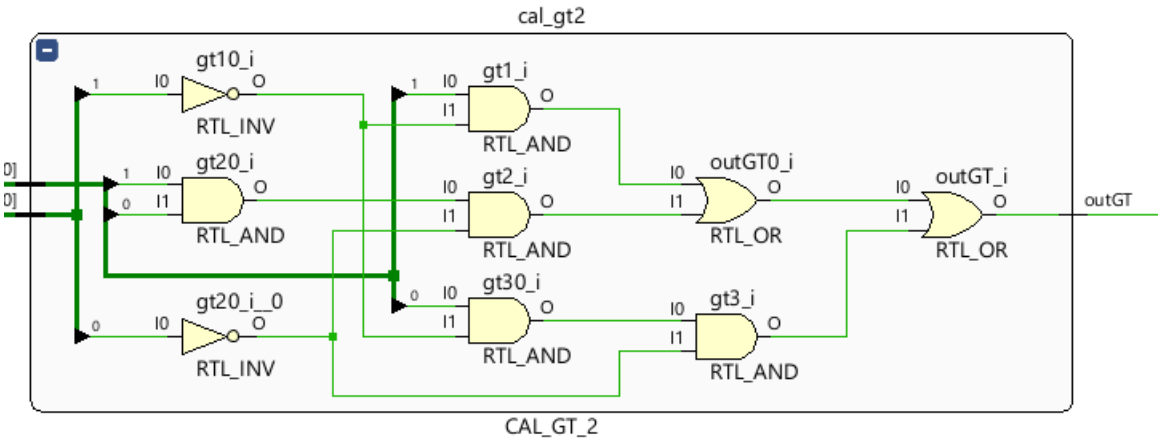


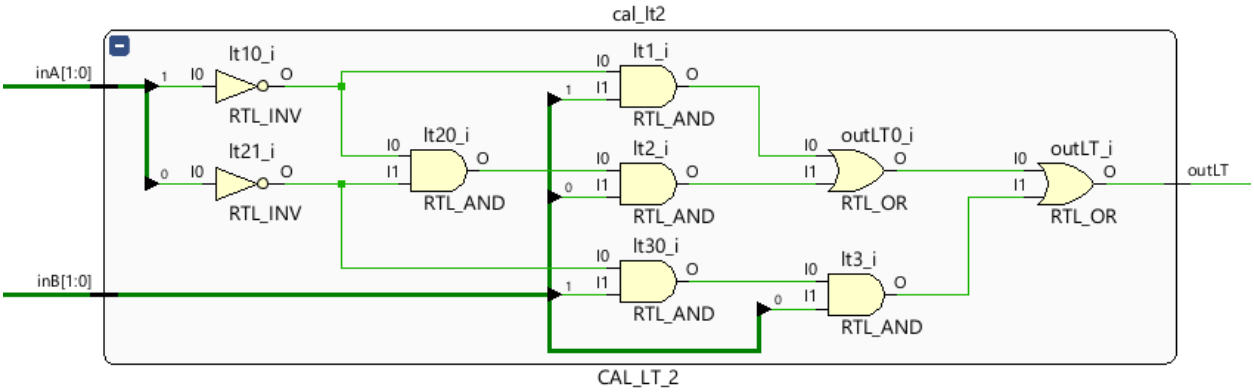
(2) lab 2_2 (단순화 시킨 후)

밑의 회로도는 K-map 을 이용해 단순화 시킨 lab2_2 의 결과이고, 잇따른 3 개의 사진은 EQ, GT, LT 회로도를 확대한 것이다.

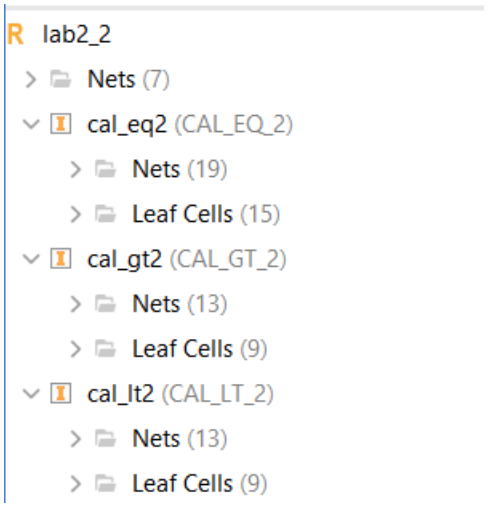








Lab 2_2 에서는 실험 준비에서 첨부한 4*4 Karnaugh map 에서 나온 단순화된 식을 이용해서 위와 같은 방법으로 회로를 작성했다. 이번 식 역시 위의 실험 준비에 첨부한 Karnaugh map 에 작성되어있다. 밑의 netlist 를 보면 lab2_1 에 비해서 현저히 적은 수의 와이어와 논리 게이트가 사용되었다는 것을 알 수 있다.



5. 논의

Nets 와 leaf cells 개수를 비교하면 단순화 하기 전 후의 차이를 확인할 수 있다. Nets 의 개수는 와이어를 나타내고 leaf cells 은 논리 게이트의 수를 나타낸다. GT, LT 의 경우를 살펴보면 K-map 을 사용해 단순화 시키면 거의 절반으로 줄어들었다는 것을 알 수 있다. EQ 는 K-map 을 사용하더라도 단순화가 더이상 되지 않아서 lab 2_1 과 2_2 모두 19 개의 와이어, 그리고 15 개의 논리 게이트가 사용되었다. 반면에 GT 와 LT 는 단순화 되기 전엔 각각 22 개의 와이어와 18 개의 논리게이트가 사용되었는데 K-map 을 이용해 단순화 시킨 이후엔 13 개의 와이어와 9 개의 논리게이트가 사용되었다. 더불어 EQ 역시 식 자체를 단순화 시키면 11 개의 와이어와 7 개의 논리게이트만을 사용해서 구현해낼 수 있다. 만약 XNOR gate 를 사용하면 7 개의 와이어와 3 개의 논리게이트만을 사용해서 구현하는 것도 가능하다.

이번 랩을 통해서 복잡한 Boolean equation 을 단순화 하는 것이 회로에 직접적인 영향을 미치며 전력, 속도, 그리고 효율성과 연관되어 있다는 것을 알 수 있었다. 이런 Simplification 의 중요도에 대해 알게 되었으며 같은 netlist 결과를 가진 EQ 회로도를 simplification 하는 방법에 대해 생각해 볼 수 있었다.

EQ 를 단순화 한 과정은 밑과 같다.

$$A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0'B_1B_0' + A_1A_0B_1B_0$$

$$= A_1'B_1'(A_0'B_0' + A_0B_0) + A_1B_1(A_0'B_0' + A_0B_0)$$

$$= (A_1'B_1' + A_1B_1)(A_0'B_0' + A_0B_0)$$

$$= (A_1A_1' + A_1B_1 + A_1'B_1' + B_1B_1')(A_0A_0' + A_0B_0 + A_0'B_0' + B_0B_0')$$

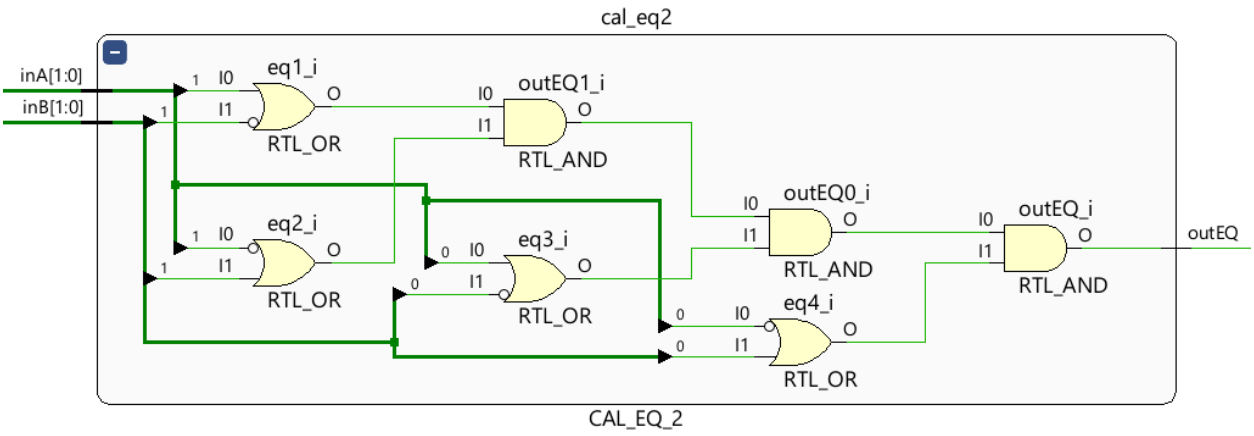
$$= \{A_1(A_1' + B_1) + B_1'(A_1' + B_1)\}\{A_0(A_0' + B_0) + B_0'(A_0' + B_0)\}$$

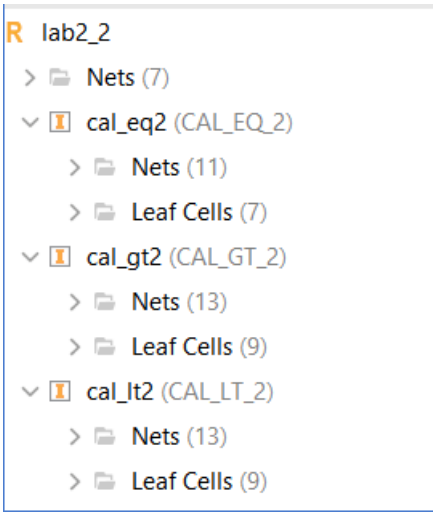
$$= (A_1 + B_1')(A_1' + B_1)(A_0 + B_0')(A_0' + B_0)$$

$$= (A_1 \oplus B_1)'(A_0 \oplus B_0)'$$

밑은 식을 단순화 시켰을 때의 Verilog 소스코드, EQ 회로도와 netlist 이다.

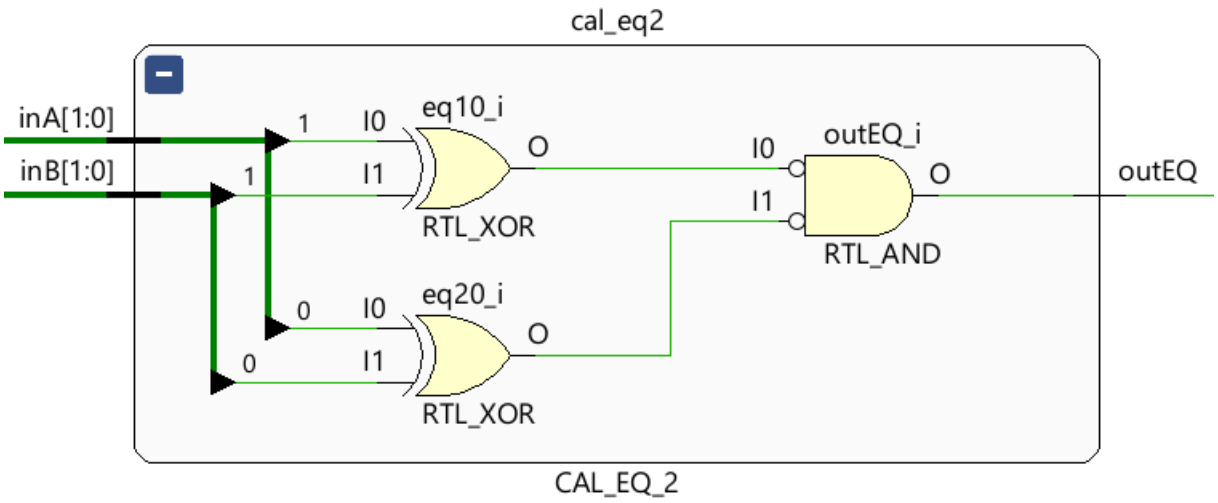
```
/* <when simplify above boolean equation>
wire eq1, eq2, eq3, eq4;
assign eq1 = inA[1]|~inB[1];
assign eq2 = ~inA[1]|inB[1];
assign eq3 = inA[0]|~inB[0];
assign eq4 = ~inA[0]|inB[0];
assign outEQ = eq1&eq2&eq3&eq4;
*/
```

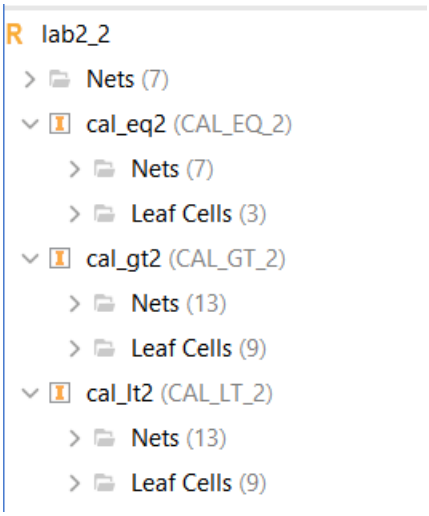




밑은 XNOR gate 를 사용했을 때의 Verilog 소스코드, EQ 회로도와 netlist 이다.

```
/* <when xnor gate is used>
wire eq1, eq2;
xnor(eq1, inA[1], inB[1]);
xnor(eq2, inA[0], inB[0]);
and(outEQ, eq1, eq2);
*/
```


cal_eq2
inA[1:0]
inB[1:0]
1
1
0
0
I0
I1
eq10_i
O
RTL_XOR
eq20_i
O
RTL_XOR
I0
I1
outEQ_i
O
RTL_AND
outEQ
CAL_EQ_2


lab2_2
Nets (7)
cal_eq2 (CAL_EQ_2)
Nets (7)
Leaf Cells (3)
cal_gt2 (CAL_GT_2)
Nets (13)
Leaf Cells (9)
cal_lt2 (CAL_LT_2)
Nets (13)
Leaf Cells (9)