디지털 시스템 설계 (CSED273) Lab 1. 논리회로 기초 보고서

2023.03.18.

20220923 이윤혁

1. 개요

Lab2에서는 불 대수식을 K-map 알고리즘을 통해 단순화시키고, 단순화 했을 때 gate와 wire의 개수가 얼마나 줄어들었는지 비교한다. Lab2_1은 단순화 전, Lab2_2는 단순화 후의 불 대수식을 논리회로로 구현한다.

2. 이론적 배경

2.1. 불 대수식의 단순화

불 대수식의 복잡도는 불 대수식을 구현한 논리회로에 사용된 gate와 wire의 개수에 의존한다. Gate와 wire의 개수가 많아지면 불 대수식의 복잡도도 높아진다. 불 대수식을 단순화함으로써 논리회로에 사용하는 gate와 wire의 개수를 줄일 수 있다. 즉 불 대수식의 복잡도를 낮출 수 있다. 또한 신호가 거치는 gate와 wire의 개수가 줄어듦으로 소비전력과 신호의 딜레이가 줄어든다.

불 대수식은 Karnaugh-Map이나 Quine-McCluskey 알고리즘을 이용해 단순화할 수 있다. Lab2에서는 K-map을 사용했다. K-map은 Graphical하게 회로를 simplify하는 방법으로, 인접한 min term이 직사각형으로 2의 거듭제곱 꼴을 형성하는 경우, PI인지, EPI인지 확인한다. PI는 다른 implicant들과 그것의 형태 때문에 더 이상 같이 묶일 수 없는 Implicant들의 집합을 의미한다. Implicant는 K-map에 표기된 1 또는 1의 집합을 의미한다. 다른 PI들이 가질 수 없는 implicant가 있는 경우 그 PI는 EPI이다. 이 과정을 거친 후 K-map위의 모든 implicant를 cover할 수 있는 EPI, PI만 골라 or 연산으로 이어주면 그 불 대수식은 원래 불 대수식을 simplify한 식이다. 해당하는 literal을 제거함으로써 simplification을 수행하는 알고리즘이다. Quine-McCluskey는 컴퓨터에게 적합한 방법으로, min term의 1의 개수로 그룹을 나누고, 인접한 그룹끼리 겹치는 literal을 제거해나가며 EPI와 PI를 찾는 방법이다.

2.2. 2-bit Magnitude Comparator

2-bit Magnitude Comparator는 2bit의 수 2개를 입력 받아 두 수의 크기를 비교해 대소관계를 신호로 출력하는 회로이다. 총 4개의 input(2bit 수 2개)을 입력 받고 3개의 output(A > B, A = B, A < B)를 출력한다. A와 B의 대소관계를 출력하기 때문에 Output 중 1 wire만 1로 출력한다.

3. 실험 준비

3.1. Lab2_1

2-bit Magnitude Comparator의 A > B, A = B, A < B 회로를 설계하기 위해 K-map을 그렸다. Gray code의 2bit 수를 비교하기 편하게 10진수로 변환한 값도 K-map 상단과 좌측에 기록했다

[Table 1] K-Map의 A > B

		0	1	3	2
	A_1A_0₩B_1B_0	00	01	11	10
0	00	0	0	0	0
1	01	1	0	0	0
3	11	1	1	0	1
2	10	1	1	0	0

[Table 2] K-map of A = B

		0	1	3	2
	A_1A_0₩B_1B_0	00	01	11	10
0	00	1	0	0	0
1	01	0	1	0	0
	11	0	0	1	0
2	10	0	0	0	1

[Table 3] K-map of A < B

		0	1	3	2
	A_1A_0₩B_1B_0	00	01	11	10
0	00	0	1	1	1
1	01	0	0	1	1
3	11	0	0	0	0
2	10	0	0	1	0

단순화하지 않은 A > B, A = B, A < B의 불 대수식을 써보면 [Table 4]이다.

[Table 4] Boolean algebra expression before simplification

A > B	$A_1'A_0B_1'B_0' + A_1A_0'B_1'B_0' + A_1A_0'B_1'B_0 + A_1A_0B_1'B_0' + A_1A_0B_1'B_0 + A_1A_0B_1B_0'$
A = B	$A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0'B_1B_0' + A_1A_0B_1B_0$
A < B	$A_1'A_0'B_1'B_0 + A_1'A_0'B_1B_0' + A_1'A_0'B_1'B_0' + A_1'A_0B_1B_0' + A_1'A_0B_1B_0 + A_1A_0'B_1B_0$

3.2. Lab2_2

Karnaugh-Map으로 simplify하기 위해 인접한 1들을 그룹화하고, simplify했다.

[Figure 1] K-Map simplification of A > B

		0	1	3	2
	A_1A_0₩B_1B_0	00	01	11	10
0	00	0	0	0	0
1	01	1	0	0	0
3	11	1	1	0	1
2	10	1	1	0	0

[Figure 2] K-Map simplification of A = B

		0	1	3	2
	A_1A_0₩B_1B_0	00	01	11	10
0	00	1	0	0	0
1	01	0	1	0	0
3	11	0	0	1	0
2	10	0	0	0	1

[Figure 3] K-Map simplification of A > B

		0	1	3	2
	A_1A_0₩B_1B_0	00	01	11	10
0	00	0	1	1	1
1	01	0	0	1	1
3	11	0	0	0	0
2	10	0	0	1	0

A = B는 K-Map을 통해서 모든 min term들이 인접하지 않아 Simplify할 수 없었다. Simplify한 결과는 [Table 5]이다.

[Table 5] Boolean algebra expression after simplification

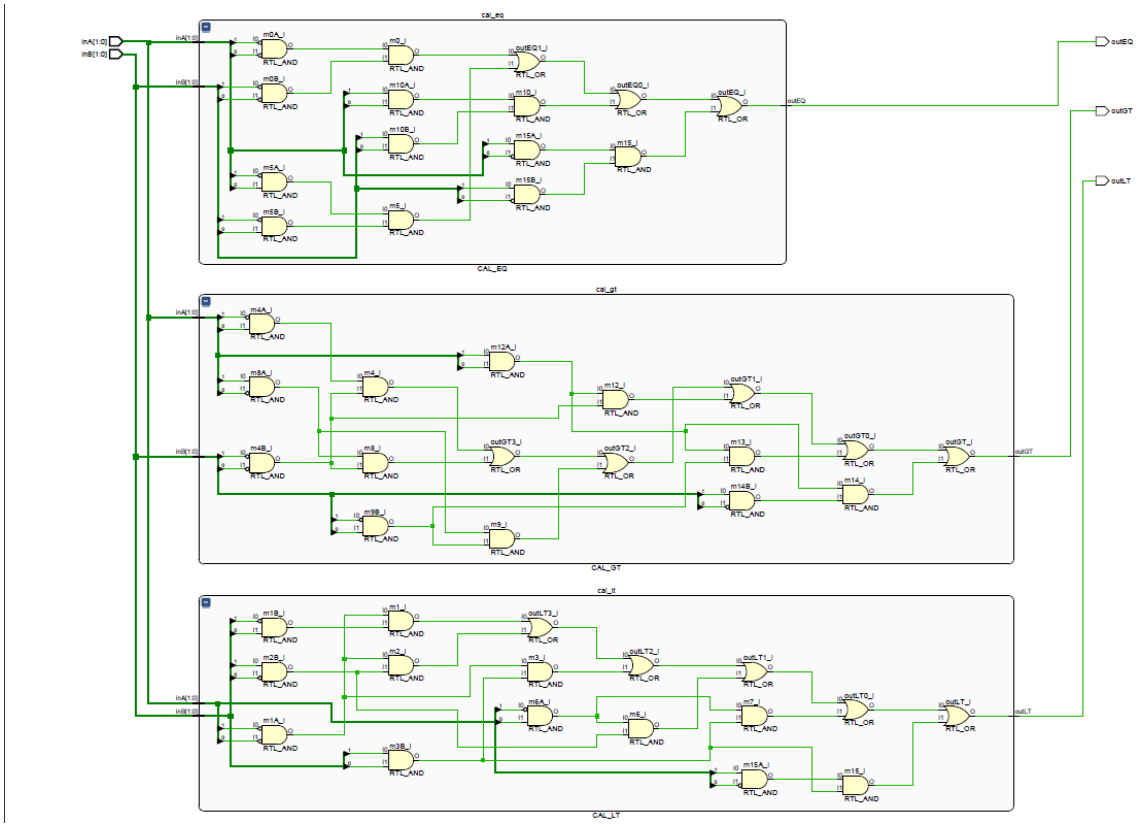
A > B	$A_1'B_1' + A_0B_1'B_0' + A_1A_0B_0'$
A = B	$A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0'B_1B_0' + A_1A_0B_1B_0$
A < B	$A_1'B_1 + A_1'A_0'B_0 + A_0'B_1B_0$

4. 결과

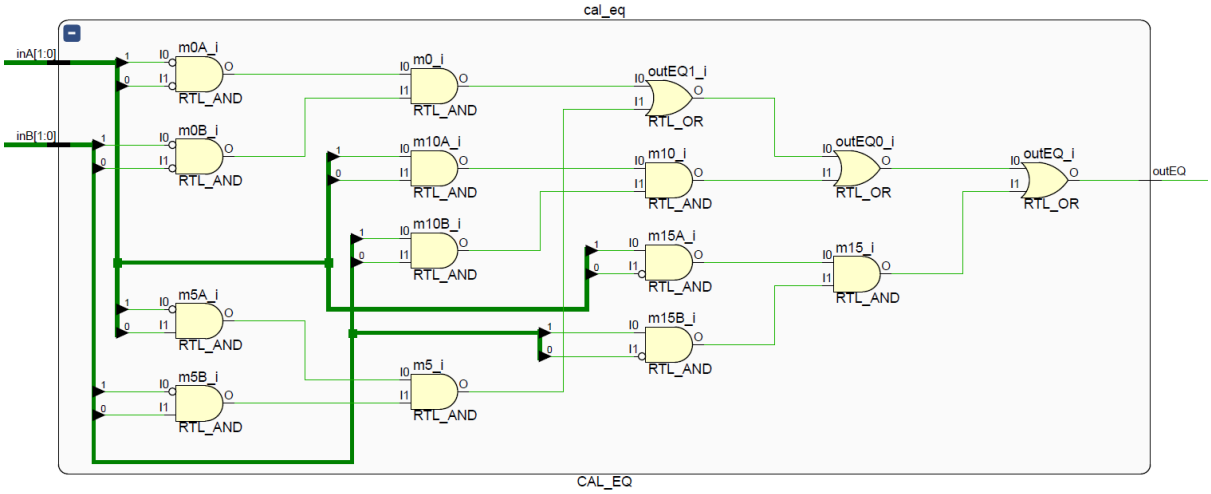
4.1. Lab2_1

전체 Schematic은 [Figure 4]이고, A=B, A > B, A < B의 회로는 각각 [Figure 5], [Figure 6], [Figure 7]이다.

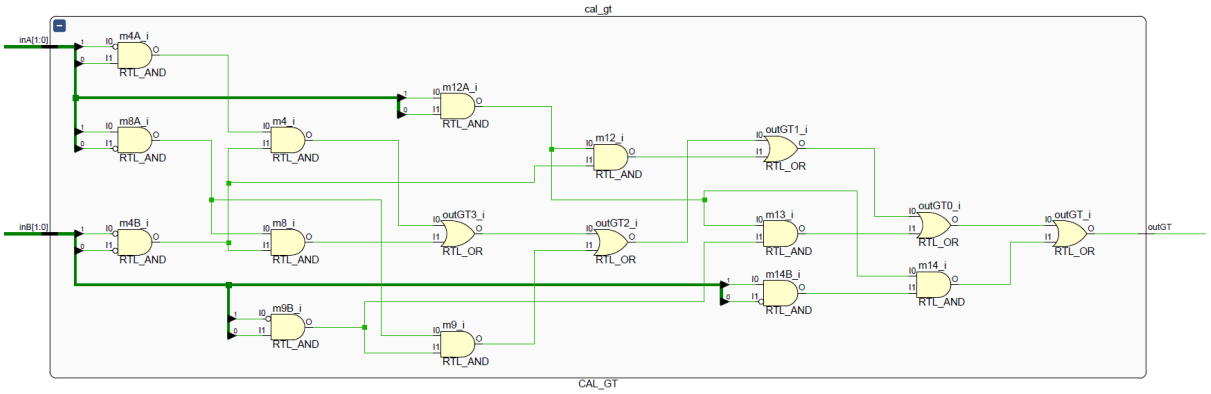
[Figure 4] Schmatic of Unsimplified 2-bit Magnitude Comparator



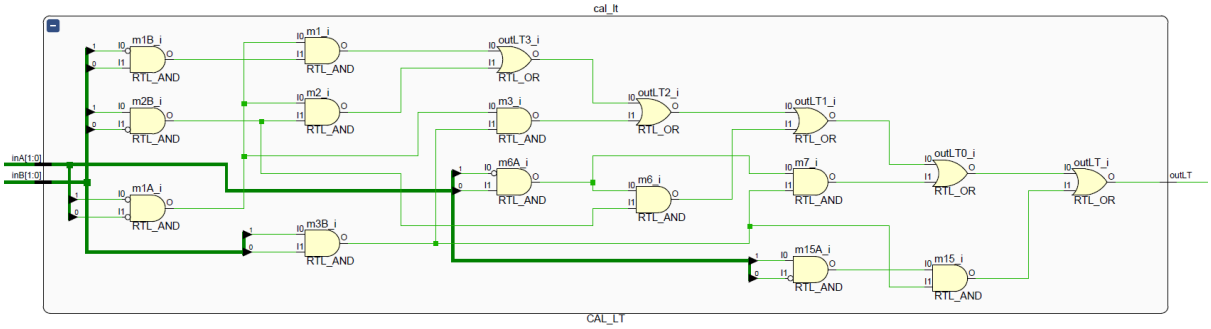
[Figure 5] Unsimplified A = B



[Figure 6] Unsimplified A > B

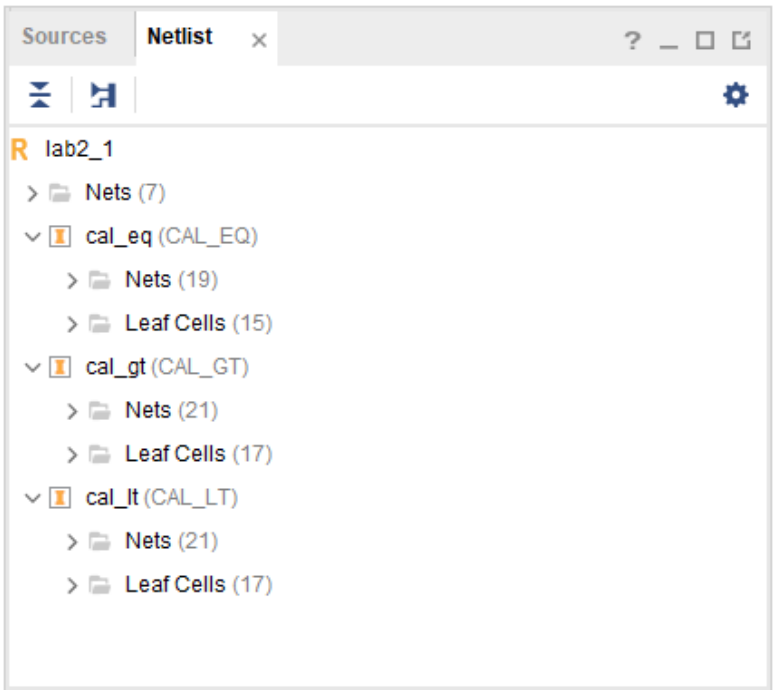


[Figure 6] Unsimplified A < B



Vivado의 Netlist 기능을 이용해 각 회로에 사용된 gate와 wire의 개수를 세었다.

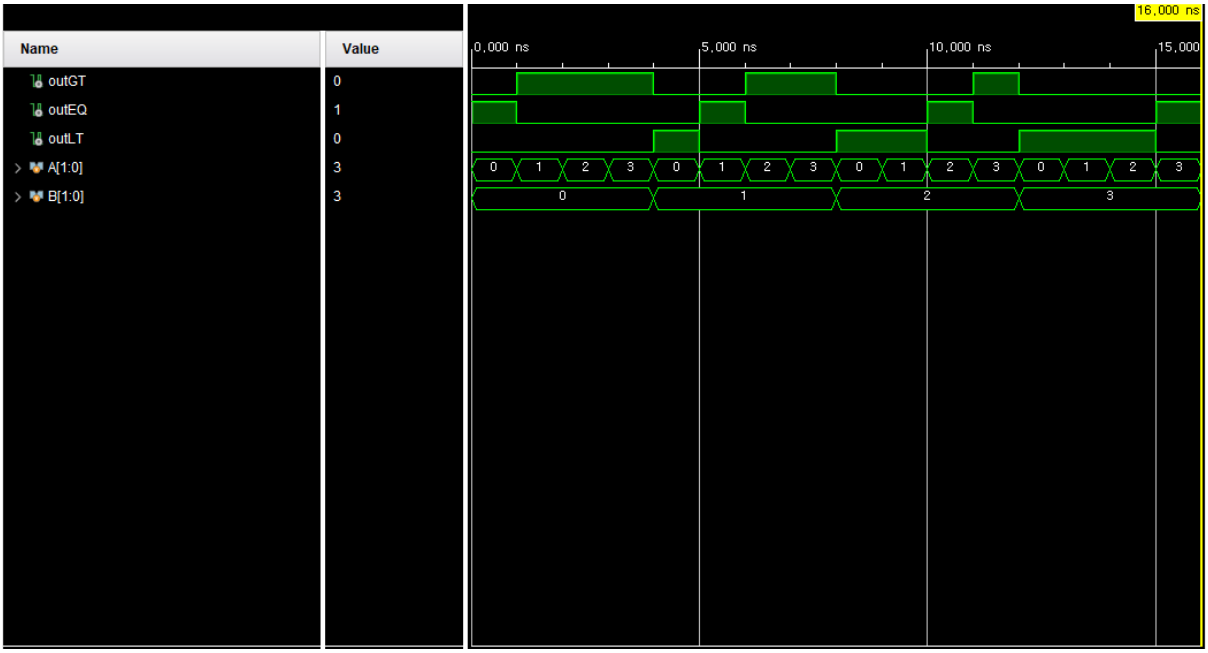
[Figure 10] Number of wires and gates



2-bit Magnitude Comparator가 제대로 구현되었는지 확인하기 위해 Testbatch로 simulation을

진행했다. A[1]을 2초, A[0]를 1초, B[1]를 8초, B[0]를 4초마다 반전시켜 A와 B를 모두 0 부터 3까지 매칭하여 값을 비교했다. 시뮬레이션한 결과 정상 작동했다.

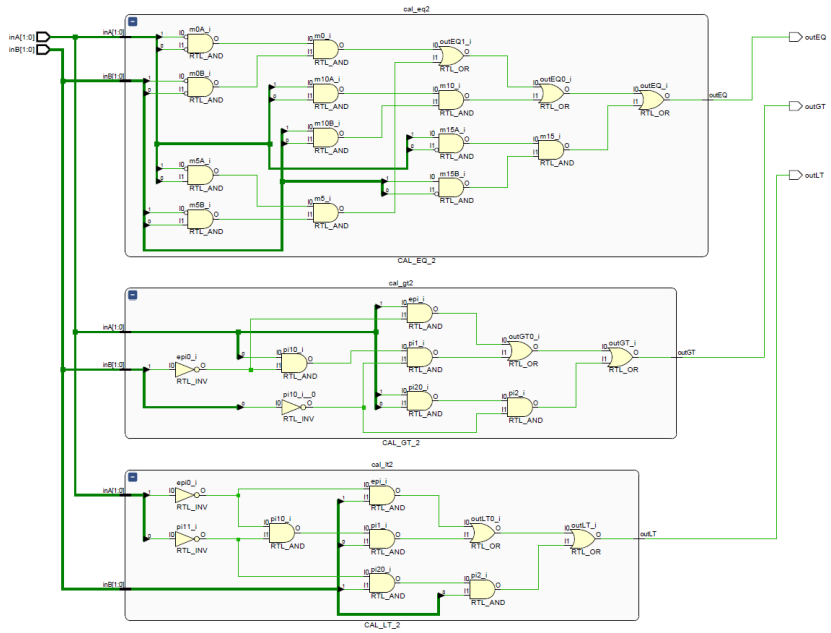
[Figure 11] Testbatch simulation



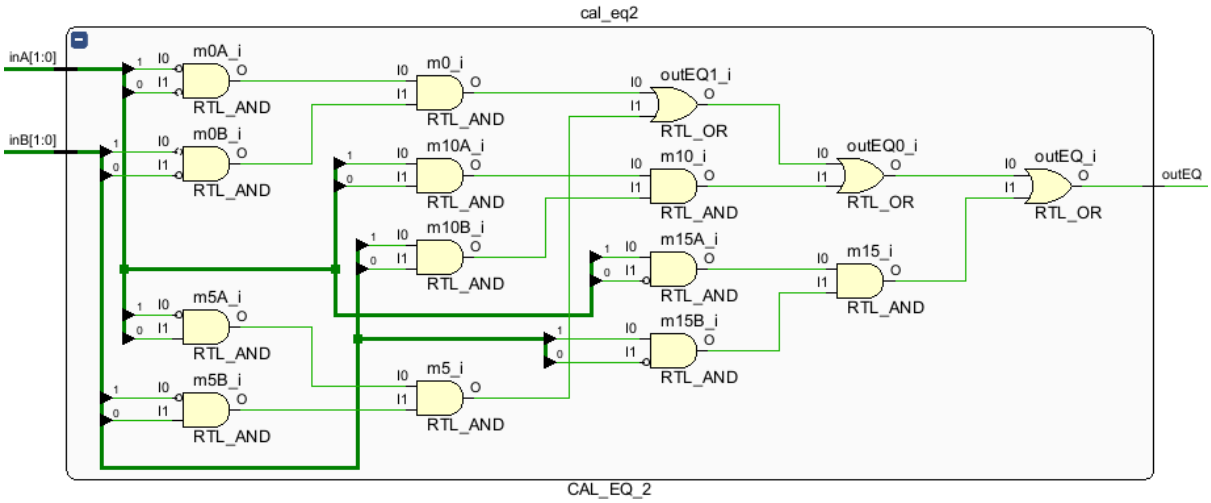
4.2. Lab2_2

전체 Schematic은 [Figure 12]이고, A=B, A > B, A < B의 회로는 각각 [Figure 13], [Figure 14], [Figure 15]이다.

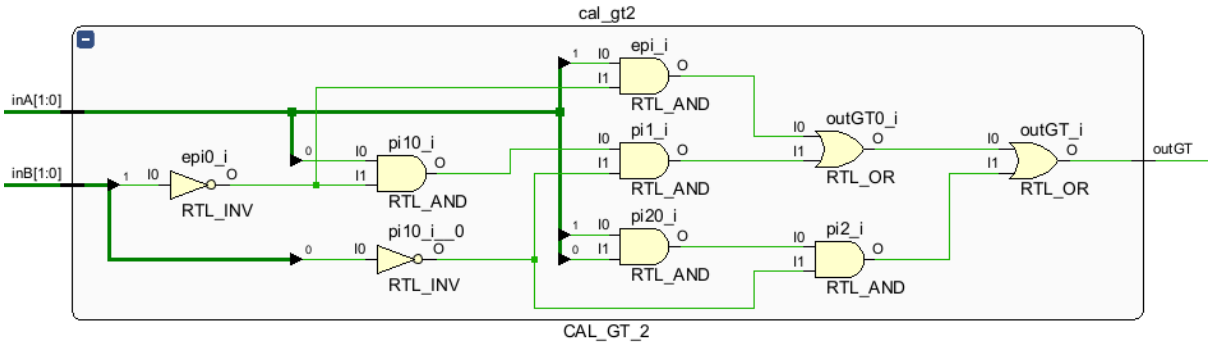
[Figure 12] Schematic of Simplified 2-bit Magnitude Comparator



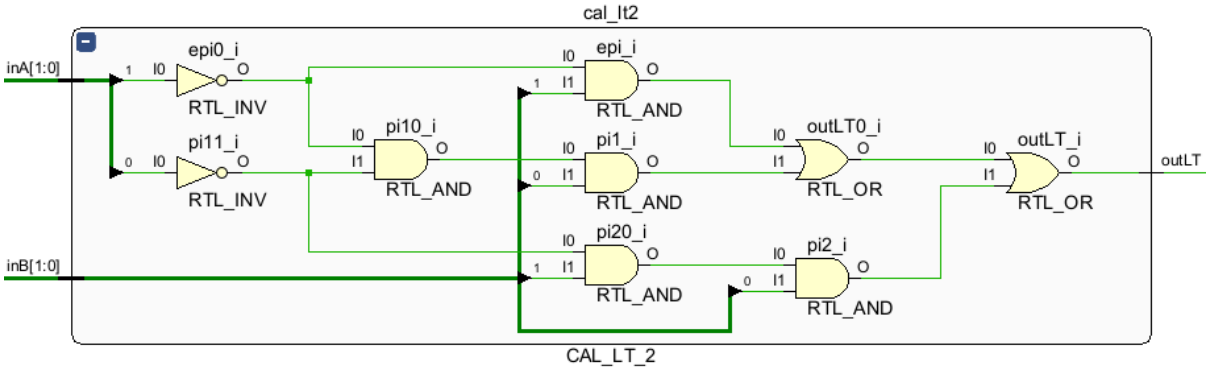
[Figure 13] Simplified A = B



[Figure 14] Simplified A > B

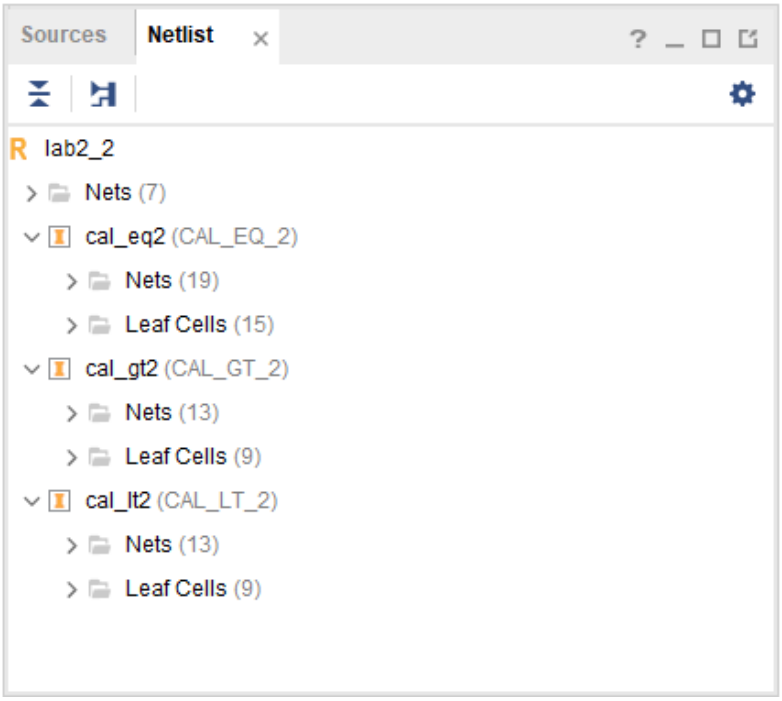


[Figure 15] Simplified A < B



Vivado의 Netlist 기능을 이용해 각 회로에 사용된 gate와 wire의 개수를 세었다.

[Figure 16] Number of wires and gates



Simplify한 2-bit Magnitude Comparator가 제대로 구현되었는지 확인하기 위해 Testbatch로 simulation을 진행했다. A[1]을 2초, A[0]를 1초, B[1]를 8초, B[0]를 4초마다 반전시켜 A와 B를 모두 0 부터 3까지 매칭하여 값을 비교했다. 시뮬레이션한 결과 정상 작동했다.

[Figure 17] Testbatch simulation

Unsimplified Boolean algebra expression과 Simplified Boolean algebra expression의 wire와 gate의 개수를 비교했다.

[Figure 18] Comparison of Number of gates and wires

Unsimplified	wire	gate	Simplified	wire	gate
A = B	19	15	A = B	19	15
A > B	21	17	A > B	13	9
A < B	21	17	A < B	13	9

그 결과 Simplified한 회로가 Unsimplified한 회로보다 wire와 gate의 개수가 작거나 같음을 알 수 있다.

5. 논의

2 bit-Magnitude Comparator를 직접 구현하고, K-map을 이용해 Simplify하면서 Simplification이 복잡한 회로에서 필요한 wire와 gate를 굉장히 많이 줄여 줄 수 있음을 깨달았다. 그리고 simulation을 진행하면서 처음 구현한 회로가 잘못 된 것을 관찰했는데, 약간의 실수로 논리회로가 아주 다른 답을 내놓을 수 있음을 깨달았다.