Lab 2. 불 대수식의 단순화

Digital System Design (CSED273)

20220100 박기현

1. 개요

불 대수식을 단순화하는 방법에 대해 이해하고, 단순화하기 전과 후를 비교하여 그 효과를 확인한다. 세부적은 학습 목표는 아래와 같다.

1) K-map 알고리즘 이해

K-map, Karnaugh map 은 불 대수식을 단순화하는 방법 중 하나이다. 입력변수와 출력을 도식화하고, 1 이 나오는 경우, 즉 최소항만을 찾아 그 때의 입력 값을 식으로 표현하는 방법이다.

이때 주의해야 할 점은 2 의 제곱의 개수로 묶어주어야 하며, 사각형의 형태로 묶어주어야 한다. 또한, 묶을 수 있는 경우 중 가장 많은 1 을 포함하는 경우로 단순화해야 한다.

2) 와이어와 논리 게이트 개수를 확인하여 단순화 효과 확인

Netlist 를 참조하여 와이어 개수와 논리 게이트 개수를 확인한다. 이때 와이어는 Nets, 논리 게이트는 Leaf Cells 수로 확인한다. Invertor(NOT)의 개수는 구현에 따라 포함되지 않을 수 있다.

2. 이론적 배경

1) 불 대수식의 단순화

불 대수식의 복잡도는 그 식을 실제로 구현하는데 사용된 와이어 개수와 논리 게이트 개수로 평가한다. 따라서 불 대수식을 단순화하면 자연스럽게 회로에 들어가는 와이어 개수와 논리 게이트 개수를 줄일 수 있다. 또한, 회로를 단순화시킴으로써 소비 전력을 줄이고, 작동 속도를 높일 수 있다.

단순화하는 방법 중 대표적인 것으로 카르노 맵(Karnaugh-Map, K-map)과 퀸 매클러스키(Quine-McClskey, QM) 알고리즘이 있다. 이번 실험에서는 K-map 알고리즘을 이용하여 불 대수식을 단순화한다. K-map 알고리즘에 대한 설명은 [1. 개요 – 1) K-map 알고리즘 이해]에 적어두었다.

K-map 알고리즘을 이용하면 SOP 형태로 단순화된 불 대수식을 구할 수 있다. K-map 은 2^(변수의 개수)에 해당하는 테이블을 그려 구한다. 따라서 주로 변수의 개수가 작을 때 사용된다. 또한, 입력변수의 출력 경우의 수를 gray code 를 통해 표현한다. 따라서 묶어줄 때 오른쪽 끝 박스는 왼쪽 끝 박스와, 위쪽 끝 박스는 아래쪽 끝 박스와 연결되므로, 이를 잘 고려하여 가장 많은 1 이 묶일 수 있도록 고려해야 한다.

이번 실험에서 다루지는 않지만, 5 개 이상과 같이 많은 수의 변수가 주어졌을 때는 QM 알고리즘을 주로 이용한다.

2) 2-Bit Magnitude Comparator

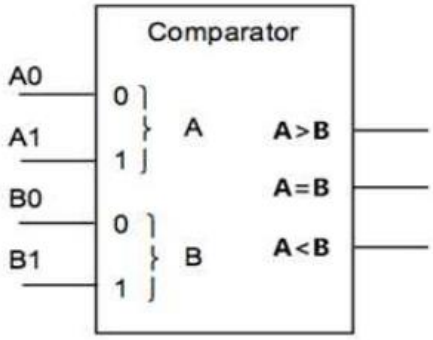


그림 1 2-Bit Magnitude Comparator

2-Bit Magnitude Comparator 는 서로 다른 두 개의 2-Bit 수가 입력으로 주어졌을 때, 둘의 대소 관계를 판별하여 알맞은 출력을 1 로, 나머지 출력을 0 으로 설정하여 출력해 주는 회로이다. 그림 1 과 같이 입력이 주어지고, 입력 A1 과 B1 이 더 높은 자릿수라 가정할 경우, 출력 'A>B'는 0, 출력 'A=B'는 1, 출력 'A>B'는 0 이 된다.

A>B 인 경우를 K-map 을 이용하여 나타내면 다음과 같다.

B1B0 \ A1A0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

A=B 인 경우를 K-map 을 이용하여 나타내면 다음과 같다.

A1A0 / B1B0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

A<B 인 경우를 K-map 을 이용하여 나타내면 다음과 같다.

A1A0 / B1B0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

3. 실험 준비

1) 2-Bit Magnitude Comparator 의 세 출력 각각에 대한 식을 단순화하지 않고 작성한다.

A>B 인 경우는 다음과 같다.

F = A1'A0B1'B0' + A1A0'B1'B0' + A1A0B1'B0' + A1A0'B1'B0 + A1A0B1'B0 + A1A0B1B0'

A=B 인 경우는 다음과 같다.

F = A1'A0'B1'B0' + A1'A0B1'B0 + A1A0'B1B0' + A1A0B1B0

A>B 인 경우는 다음과 같다.

F = A1'A0'B1'B0 + A1'A0'B1B0 + A1'A0B1B0 + A1'A0'B1B0' + A1'A0B1B0' + A1A0'B1B0

2) K-map 을 활용하여 세 식을 단순화한다.

(1) A>B

A1A0 / B1B0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

위와 같이 총 세 번 묶어 단순화할 수 있다. 이를 단순화한 불 대수식으로 표현하면 다음과 같다.

F = A1B1' + A0B1'B0' + A1A0B0'

(2) A=B

A1A0 / B1B0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

위와 같이 총 네 번 묶어 불 대수식으로 표현하면 다음과 같다.

F = A1'A0'B1'B0' + A1'A0B1'B0 + A1A0'B1B0' + A1A0B1B0

이를 단순화하면

F = A1′B1′(A0′B0′+A0B0)+A1B1(A0′B0′+A0B0)

= (A1′B1′+A1B1)(A0′B0′+A0B0)

= (A1′+B1)(A1+B1′)(A0′+B0)(A0+B0′)

complement 를 이용해서도 위 식을 구할 수 있다.

B1B0 \ A1A0	00	01	11	10
00	0	1	1	1
01	1	0	1	1
11	1	1	0	1
10	1	1	1	0

위와 같이 총 여섯 번 묶어 불 대수식으로 표현하면 다음과 같다.

F′ = A1′B1+A1B1′+A1′A0′B0+A0B1B0′+A0B1′B0′+A1A0′B0

이를 다시 complement 하여 F 를 구할 수 있다.

F = (F′)′ = (A1B1′+A1′B1+A1′A0′B0+A0B1B0′+A0B1′B0′+A1A0′B0)′

= (A1B1′+A1′B1+A0B0′+A0′B0)′

= (A1′+B1)(A1+B1′)(A0′+B0)(A0+B0′)

(3) A<B

B1B0 \ A1A0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

위와 같이 총 세 번 묶어 단순화할 수 있다. 이를 단순화한 불 대수식으로 표현하면 다음과 같다.

F = A1'B1 + A0'B1B0 + A1'A0'B0

4. 결과

1) 단순화 이전

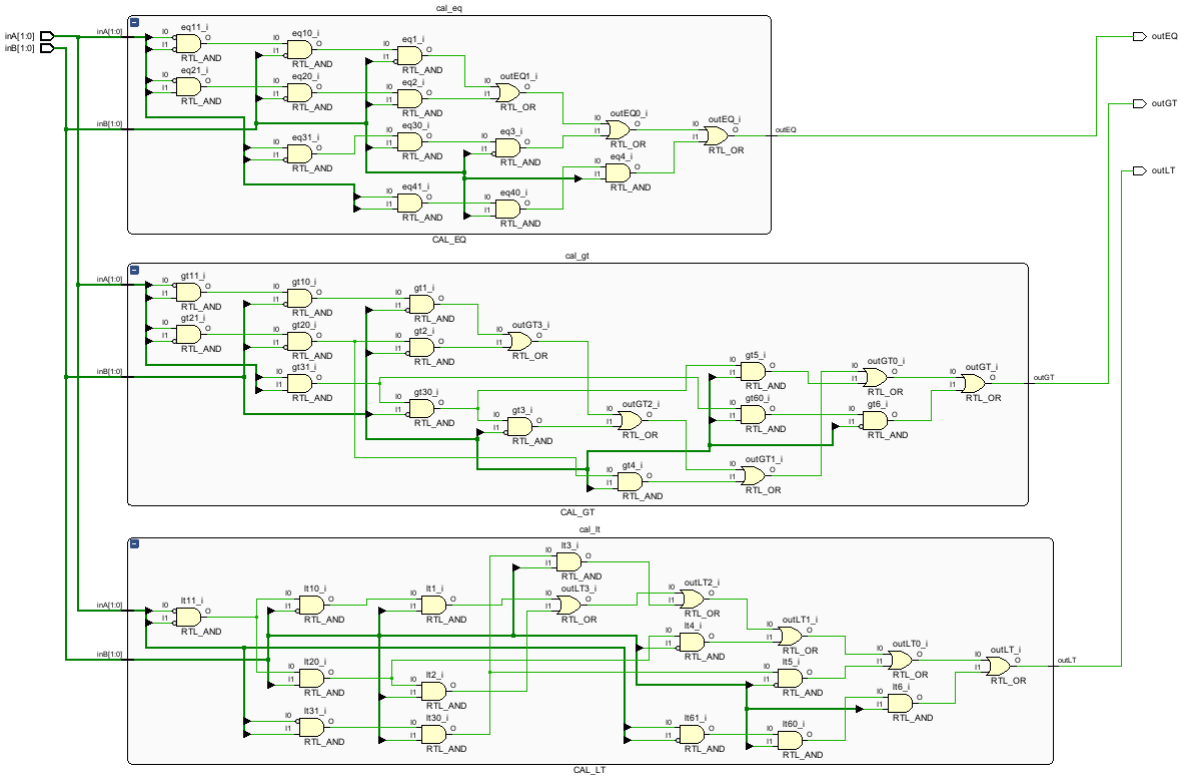
ㄱ. 2-Bit Magnitude Comparator 를 단순화하기 전 식을 구현한다.

F(A>B) = A1'A0B1'B0' + A1A0'B1'B0' + A1A0B1'B0' + A1A0'B1'B0 + A1A0B1'B0 + A1A0B1B0'

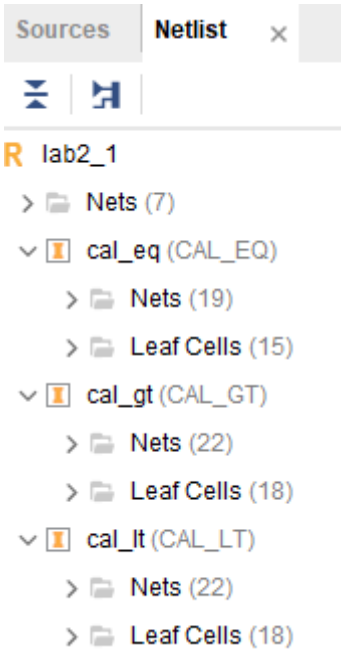
F(A=B) = A1'A0'B1'B0' + A1'A0B1'B0 + A1A0'B1B0' + A1A0B1B0

F(A<B) = A1'A0'B1'B0 + A1'A0'B1B0 + A1'A0B1B0 + A1'A0'B1B0' + A1'A0B1B0' + A1A0'B1B0

ㄴ. Schematic 기능으로 회로가 잘 구현되었는지 확인한다.



ㄷ. RTL ANALYSIS > Netlist 를 참조하여 와이어와 논리 게이트 개수를 확인한다.



2) 단순화 이후

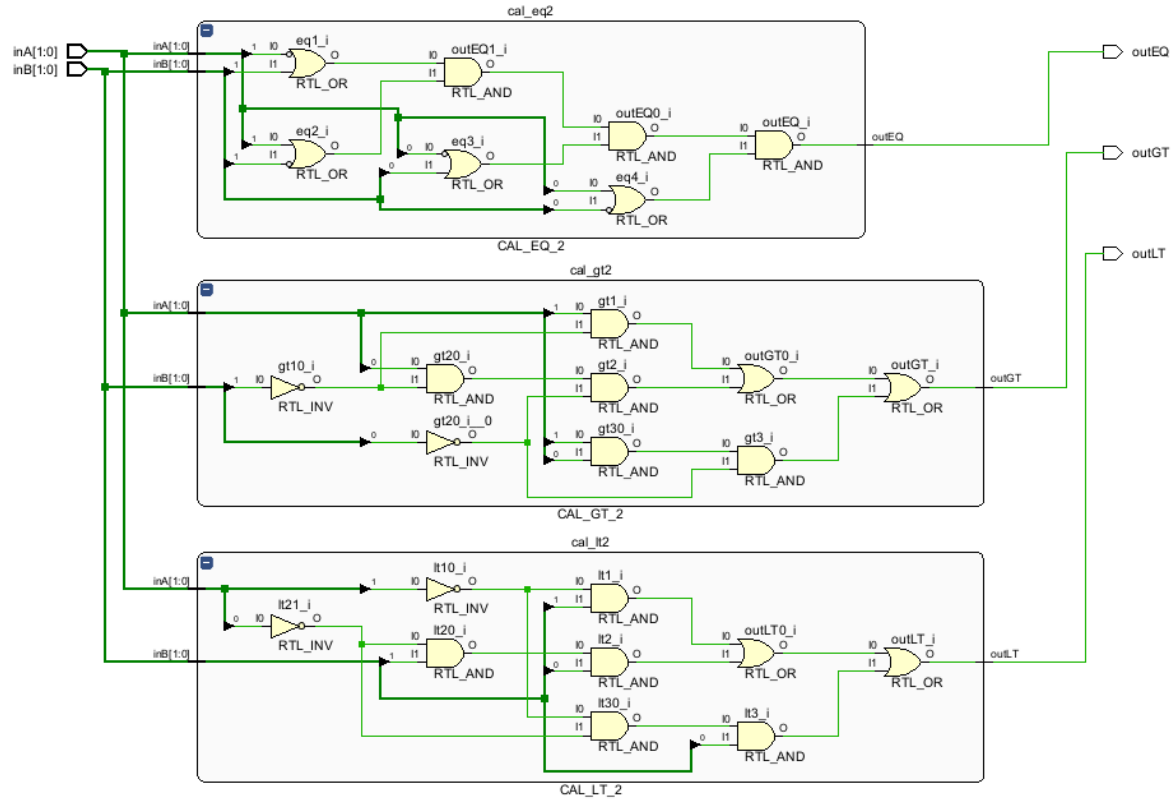
ㄱ. K-map 으로 단순화한 세 가지 출력을 구현한다.

F(A>B) = A1B_1' + A0B1'B0' + A1A0B0'

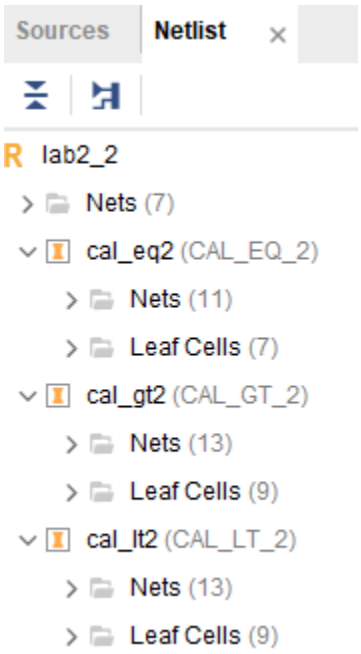
F(A=B) = (A1′+B1)(A1+B1′)(A0′+B0)(A0+B0′)

F(A<B) = A1'B1 + A0'B1B0 + A1'A0'B0

ㄴ. Schematic 기능으로 회로가 잘 구현되었는지 확인한다.



ㄷ. RTL ANALYSIS > Netlist 를 참조하여 와이어와 논리 게이트 개수를 확인한다.



ㄹ. 실험 1 과 와이어 및 논리 게이트 개수를 비교한다.

실험 1 에서 와이어 및 논리 게이트 개수는

(1) A>B – 와이어 : 22 / 논리 게이트 : 18

(2) A=B – 와이어 : 19 / 논리 게이트 : 15

(3) A<B – 와이어 : 22 / 논리 게이트 : 18

실험 2 에서 와이어 및 논리 게이트 개수는

(1) A>B – 와이어 : 13 / 논리 게이트 : 9

(2) A=B – 와이어 : 11 / 논리 게이트 : 7

(3) A<B – 와이어 : 13 / 논리 게이트 : 9

5. 논의

1) 느낀 점

HW 를 하면서 K-map 에 대한 이해도를 높일 수 있었는데, 이번 실습을 통해 직접 와이어와 논리 게이트 개수가 줄어드는 것을 보니 단순화의 중요성에 대해서도 잘 알 수 있었다. 이번 실습에서는 변수가 4 개였기에 그나마 수월했지만, 만약 그 이상의 변수가 나온다면 K-map 을 활용한 단순화는 효율적이지 못할 것이라 생각했다.

이번 실습에서 Gate-Level Modeling 이외에도 '~' 연산자가 허용되어 사용하였는데, 기존의 Gate-Level Modeling 으로 구현했던 NOT 보다 훨씬 효율적이고, 쉽게 코드를 짤 수 있다는 것을 알게 되었다. 하지만 OR 과 AND 의 경우에는 기존의 방식이 나에게는 더 직관적이었기에 NOT 경우에만 '~' 연산자를 사용하여 구현하였다.

2) 어려웠던 점 및 해결 방법

K-map 알고리즘을 이용하여 단순화를 하는 과정에 있어서 더 좋은 경우의 수가 있다는 것을 자꾸 놓치는 경우가 많았다. 이 점에 대해서 2 의 제곱의 수로 묶인다는 성질을 이용하여 가장 많은 1 이 묶이는 경우의 수를 하나씩 찾아가면 놓치지 않고 잘 찾을 수 있을 것이라 생각하였다.

또한, 처음에 가장 크게 실수했던 점이 위의 행과 열은 gray code 로 나타내야 하는데, 이를 놓쳐 잘못 K-map 을 그리고 있었다는 것이었다. 실수를 바로잡으며 왜 gray code 로 행과 열을

구성하였는지에 대한 궁금증에 K-map 의 끝과 끝이 이어지게 함으로써 더 많은 경우의 수를 표현할 수 있었다는 것을 알게 되었다.

또한, A=B 경우에서 모든 1 이 각각 한 개씩만 묶여 단순화가 되지 않아, 처음에는 단순화가 불가능하다고 생각했었지만, complement 를 이용하여 K-map 으로 단순화를 시도해보니 POS 형태로 더 단순화된 식을 얻을 수 있었다. 자칫 하면 놓칠 수 있는 부분을 다시 점검하며 단순화에 대한 개념을 다시 한번 익힐 수 있었던 것 같다.