디지털 시스템 설계 (CSED273) Lab 3. 디코더와 멀티플렉서 보고서

2023.04.05.

20220923 이윤혁

1. 개요

Lab3에서는 디코더와 멀티플렉서를 다룬다. Lab3_1에서는 2-to-4 Active low decoder를 확장해 4-to-16 Active low decoder를 구현한다. Lab3_2에서는 4비트 소수 판별기와 배수 검출기를 구현한다. Lab3_3에서는 5비트 Majority function을 8:1 Multiplexer와 external gate를 이용해 구현한다.

2. 이론적 배경

2.1. 디코더 (Decoder/De-Multiplexer)

디코더는 N개의 입력을 받아 2^N개 신호를 출력하는 회로이다. 이 때, Active High 디코더인 경우 2^N개의 출력신호 중 단 한 신호만 1이고, Active Low 디코더인 경우 2^N개의 출력신호 중 단 한 신호만 0이다. 서로 다른 N개의 신호를 입력했을 때 출력되는 2^N개의 출력 신호가 모두 다른 경우, 2^N개의 신호는 입력신호로 이루어진 minterm에 대응할 수 있다.

2.2. 디코더 확장

디코더에는 입력신호 말고도 Enable(i.e. EN) 입력이 존재한다. EN은 디코더를 키거나 끄는 역할을 한다. EN을 이용해 디코더와 디코더끼리 연결해서 더 많은 신호를 받을 수 있는 디코더를 만들 수 있다. 이를 디코더 확장이라고 한다.

2.3. 멀티플렉서 (Multiplexer; MUX)

멀티플렉서는 입력신호를 선택하여 입력 받을 수 있는 회로이다. 주로 2^N개의 입력신호를 n개의 선택신호로 선택하여, 선택 결과를 출력한다.

2.4. Majority / Minority function

Majority / Minority function은 입력 받은 홀수 개의 신호 중 어떤 신호가 더 많은/더 적은 신호를 출력하는 함수이다. 예를 들어 7개의 입력신호를 받고, 신호 중 3개가 1, 4개가 0이면 Majority function의 값은 0, Minority function의 값은 1이다.

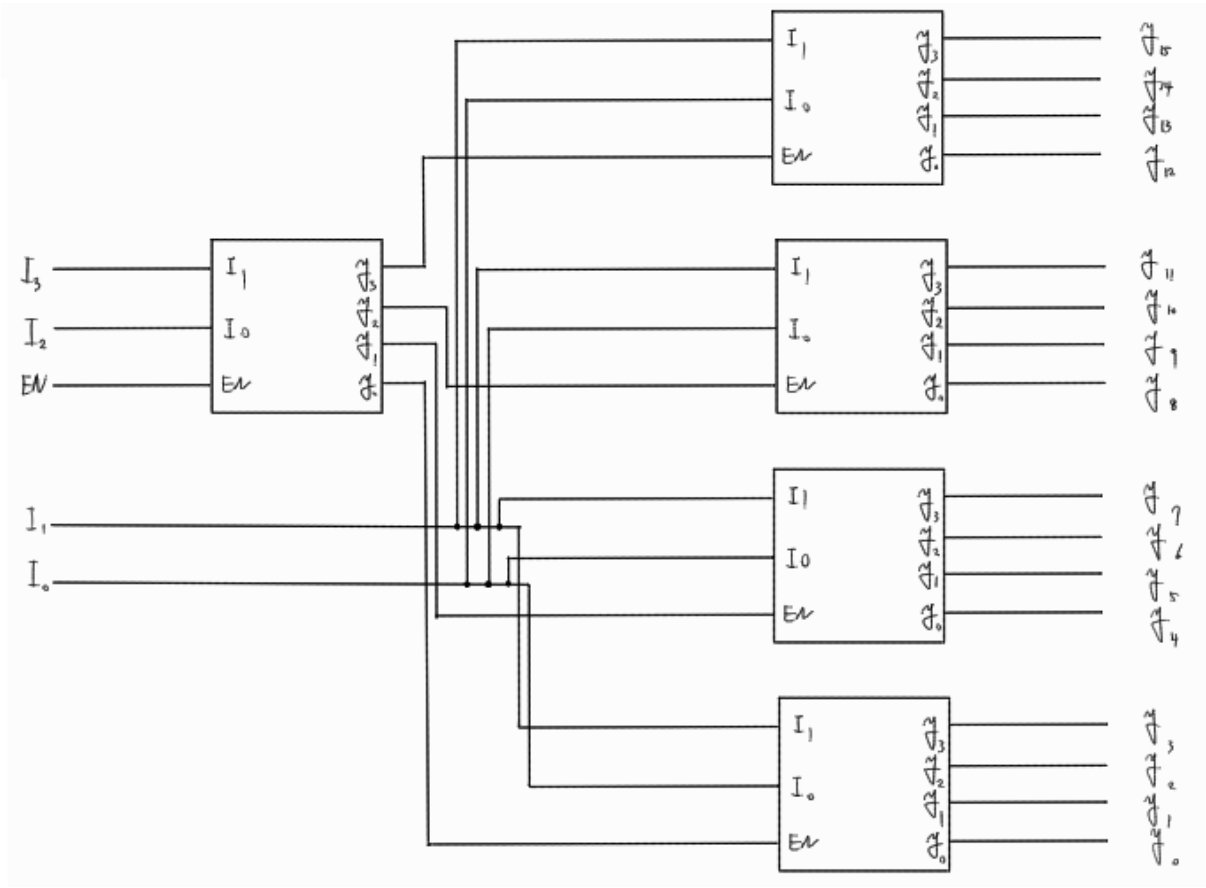
3. 실험 준비

3.1. Lab3_1

모든 입출력이 Little-endian이고, 사용할 2-to-4 디코더는 Active low enable, Active-low output,

Active-high input이다. 구현하고자 하는 4-to-16 디코더 또한 Active low enable, Active-low output, Active-high input이므로, External gate 없이 주어진 2-to-4 디코더를 연결하면 된다. 그 이유는 다음과 같다. level2의 디코더들의 Enable에 level1의 output이 각각 입력되는데, Active-low output이 Active-low enable에 입력되는 것이므로 level2의 디코더 중 오직 한 개만 활성화된다. 활성화된 디코더는 Active-low output, Active-high input임으로 Active-high input이 입력되면 4-to-16 Active low enable, Active-low output, Active-high input 디코더의 일부분으로 작동할 것이다. [그림 1]은 디코더를 확장한 회로도이다

[그림 1] 4-to-16 Decoder



3.2. Lab3_2

모든 입출력은 Little endian 방식이다. 4비트 소수 판별기의 Truth Table은 [표 1]이다. 4비트 배수 판별기의 Truth Table은 [표 2]이다. [표 2]에서 out_mul[4], out_mul[3], out_mul[2], out_mul[1], out_mul[0]은 각각 11, 7, 5, 3, 2의 배수 판별 출력이다.

[표 1] 4비트 소수 판별기의 Truth table

4비트 10진수	in[3]	in[2]	in[1]	in[0]	out_prime
0	0	0	0	0	0
1	0	0	0	1	0

2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

[표 2] 배수 판별기의 Truth table

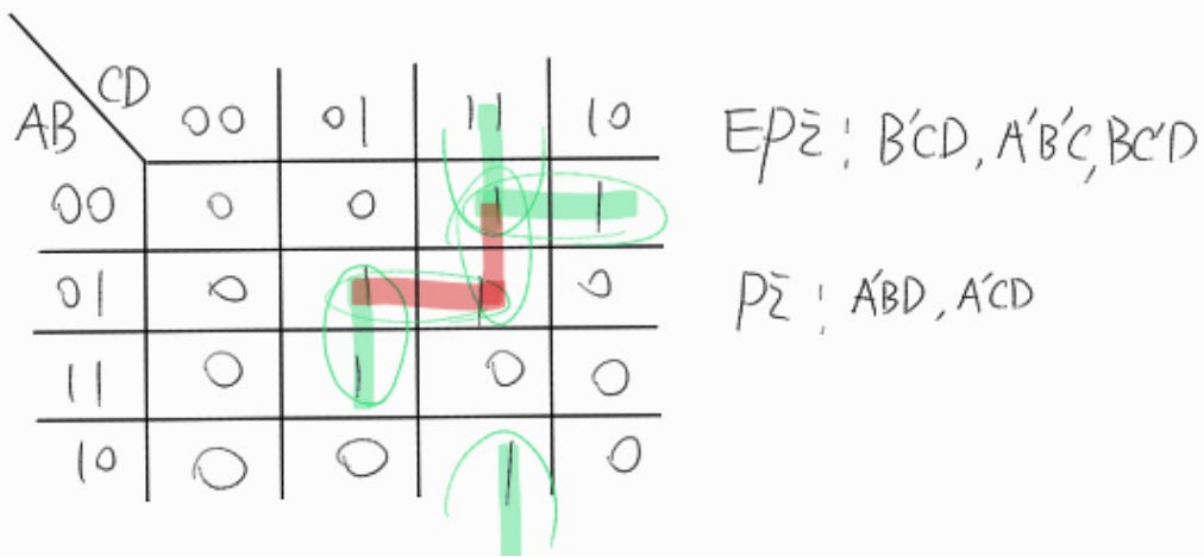
	in[3]	in[2]	in[1]	in[0]	out_mul[4]	out_mul[3]	out_mul[2]	out_mul[1]	out_mul[0]
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0
2	0	0	1	0	0	0	0	0	1
3	0	0	1	1	0	0	0	1	0
4	0	1	0	0	0	0	0	0	1
5	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	0	0	1	1
7	0	1	1	1	0	1	0	0	0
8	1	0	0	0	0	0	0	0	1
9	1	0	0	1	0	0	0	1	0
10	1	0	1	0	0	0	1	0	1
11	1	0	1	1	1	0	0	0	0
12	1	1	0	0	0	0	0	1	1
13	1	1	0	1	0	0	0	0	0
14	1	1	1	0	0	1	0	0	1
15	1	1	1	1	0	0	1	1	0

[표 1], [표 2]로부터 단순화하지 않은 SOP form을 계산하겠다. 편의성을 위해 in[3], in[2], in[1], in[0]는 각각 A, B, C, D로 치환한다.

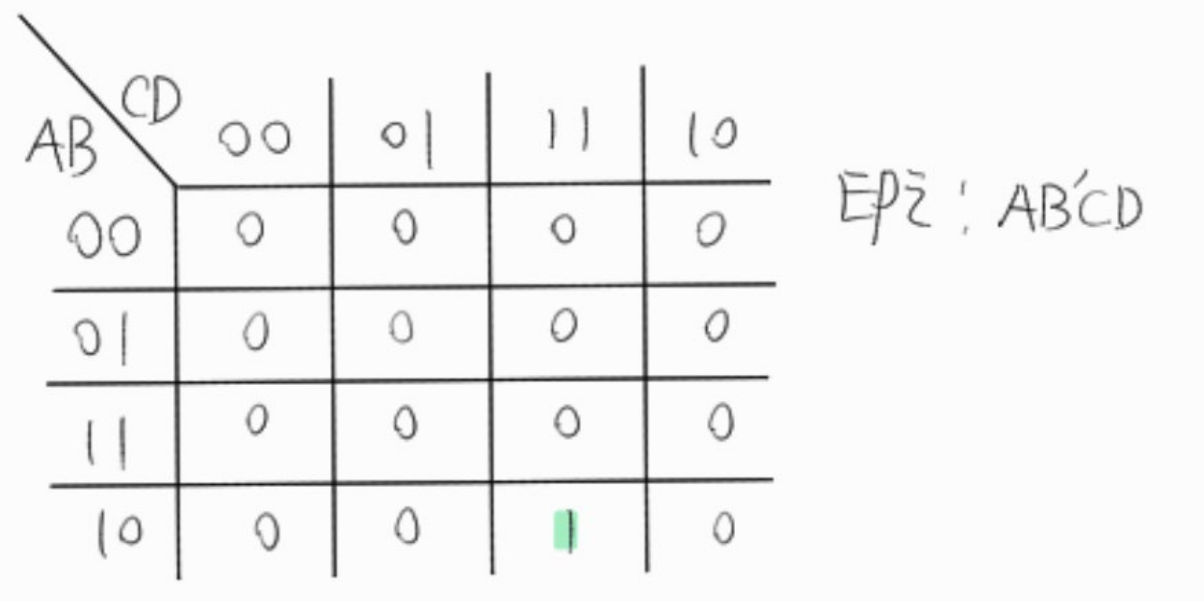
out_prime = A′B′CD + A′B′CD′ + A′BC′D + A′BCD + ABC′D + AB′CD
out_mul[4] = AB′CD
out_mul[3] = A′BCD + ABCD′
out_mul[2] = A′BC′D + AB′CD′ + ABCD
out_mul[1] = A′B′CD + A′BCD′ + AB′C′D + ABC′D′ + ABCD
out_mul[0] = $ABCD' + AB'CD' + A'BCD' + A'B'CD' + ABC'D' + AB'C'D' + A'BC'D'$

out_prime과 out_mul에 K-map을 적용시킨 그림은 각각 [그림 2], [그림 3], [그림 4], [그림 5], [그림 6], [그림 7]이다.

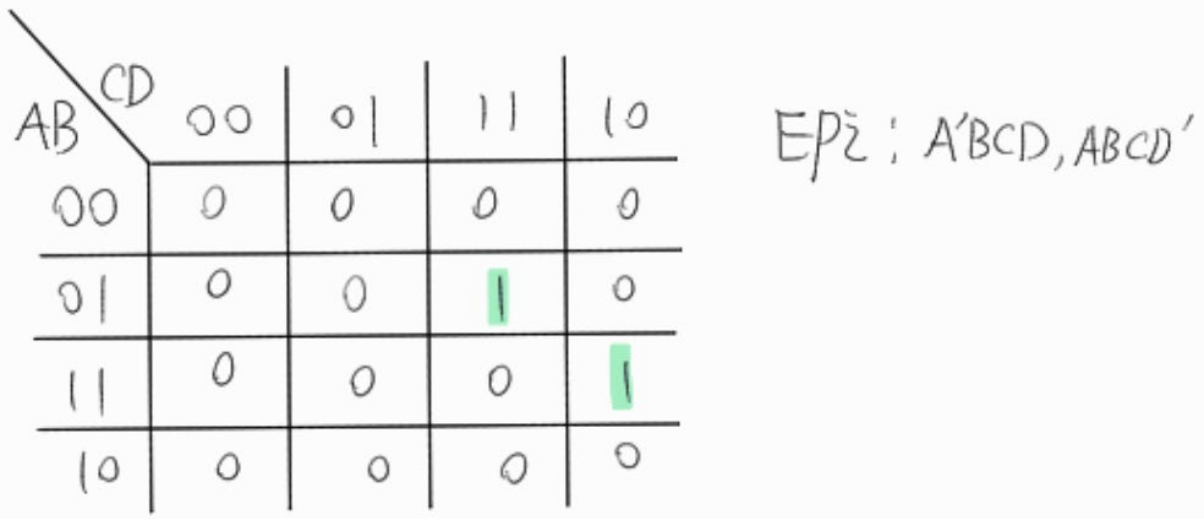
[그림 2] out_prime의 K-map



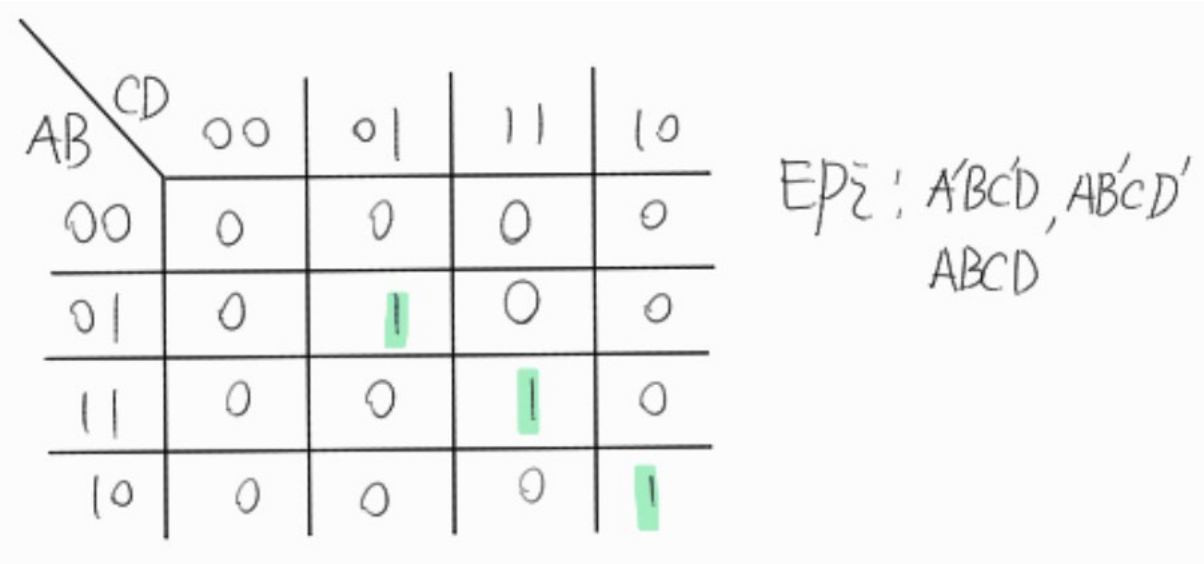
[그림 3] out_mul[4]의 K-map



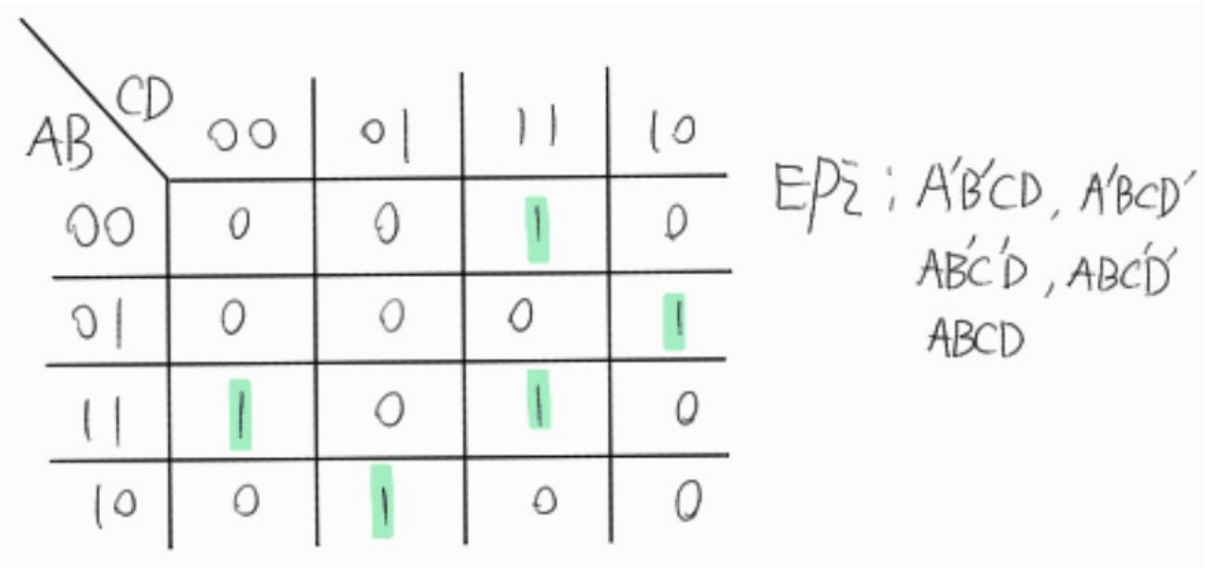
[그림 4] out_mul[3]의 K-map



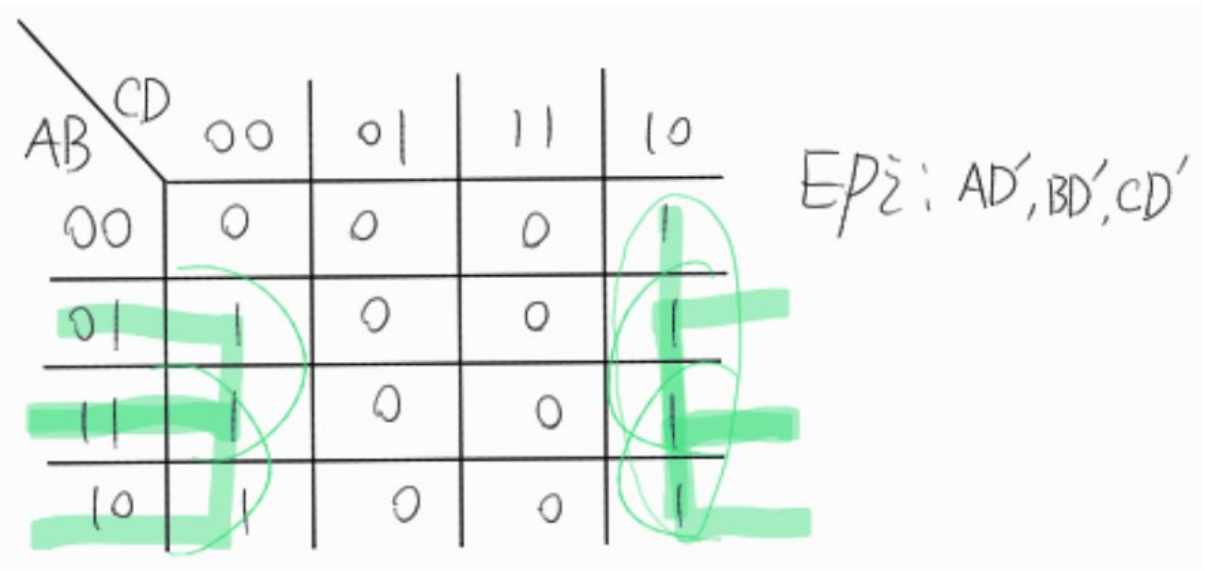
[그림 5] out_mul[2]의 K-map



[그림 6] out_mul[1]의 K-map



[그림 7] out_mul[0]의 K-map



K-map으로 단순화하면 다음과 같이 계산된다.

$out\_prime = A'B'C + A'BD + BC'D + B'CD$
$out\_mul[4] = AB'CD$
$out\_mul[3] = A'BCD + ABCD'$
$out\_mul[2] = A'BC'D + AB'CD' + ABCD$
$out\_mul[1] = A'B'CD + A'BCD' + AB'C'D + ABC'D' + ABCD$
$out\_mul[0] = AD' + BD' + CD'$

3.3. Lab3_3

모든 입출력 방식은 Little Endian이다. 5비트 Majority function의 Truth Table은 [표 3]이다. In[4], In[3], In[2], In[1], In[0]은 각각 A, B, C, D, E로 치환했다.

[표 3] 5비트 Majority function의 Truth Table

A	B	C	D	E	Majority	
0	0	0	0	0	0	0
0	0	0	0	1	0	
0	0	0	1	0	0	
0	0	0	1	1	0	
0	0	1	0	0	0	D and E
0	0	1	0	1	0	
0	0	1	1	0	0	
0	0	1	1	1	1	
0	1	0	0	0	0	D and E
0	1	0	0	1	0	
0	1	0	1	0	0	
0	1	0	1	1	1	
0	1	1	0	0	0	D or E
0	1	1	0	1	1	
0	1	1	1	0	1	
0	1	1	1	1	1	
1	0	0	0	0	0	D and E
1	0	0	0	1	0	
1	0	0	1	0	0	
1	0	0	1	1	1	
1	0	1	0	0	0	D or E
1	0	1	0	1	1	
1	0	1	1	0	1	
1	0	1	1	1	1	
1	1	0	0	0	0	D or E
1	1	0	0	1	1	
1	1	0	1	0	1	
1	1	0	1	1	1	
1	1	1	0	0	1	1
1	1	1	0	1	1	
1	1	1	1	0	1	

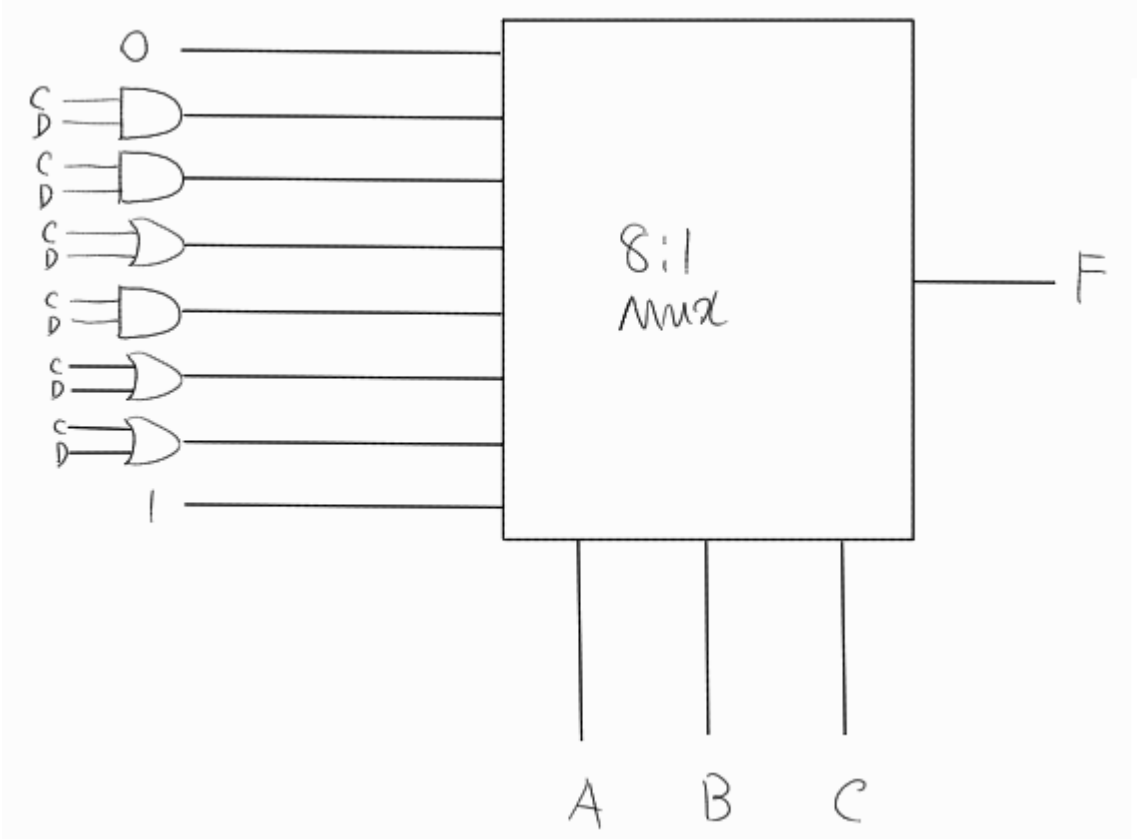
1	1	1	1	1	1	

Truth Table의 값을 토대로 SOP form을 계산하면 다음과 같다.

$$F = A'B'CDE + A'BC'DE + A'BCD'E + A'BCDE' + A'BCDE + AB'C'DE + AB'CD'E + AB'CDE' + AB'CDE + ABC'D'E + ABC'DE' + ABC'DE + ABCD'E' + ABCD'E + ABCDE' + ABCDE$$

8:1 멀티플렉서에 신호 입력으로 A, B, C를 사용한다면, 멀티플렉서의 데이터 입력은 [그림 8]과 같다.

[그림 8] 8:1 멀티플렉서 구성

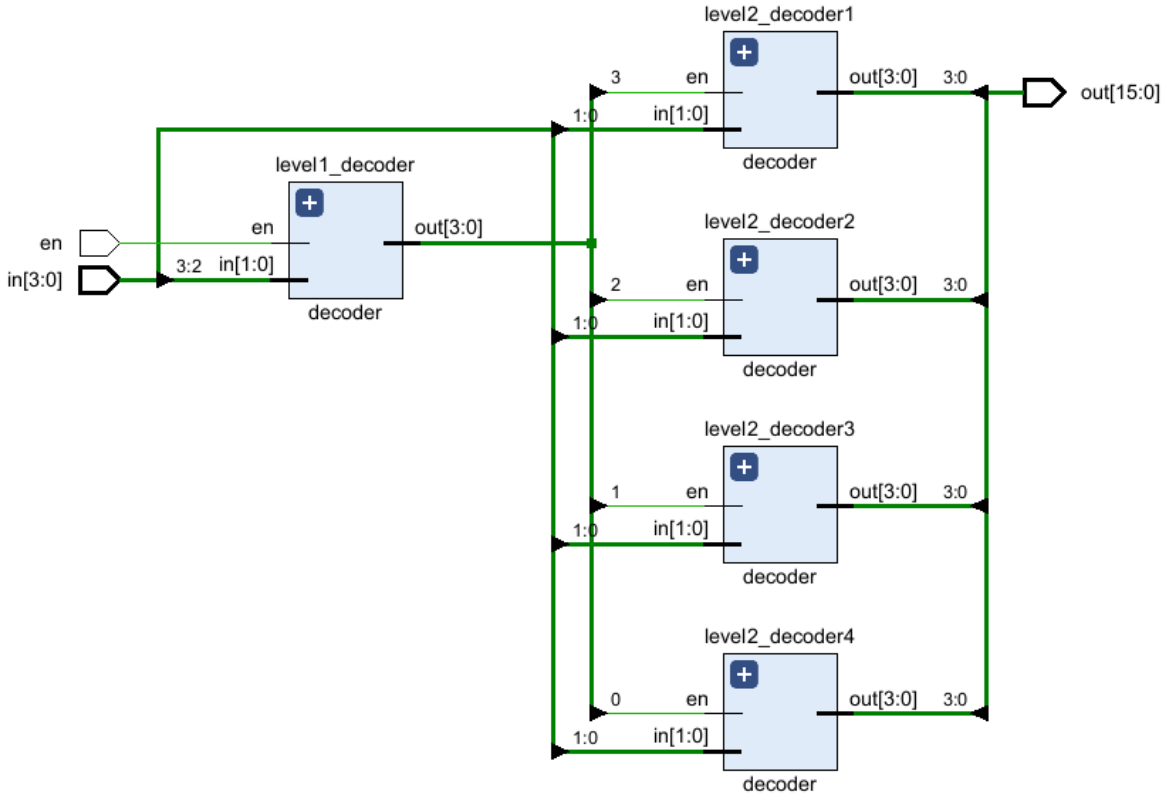


4. 결과

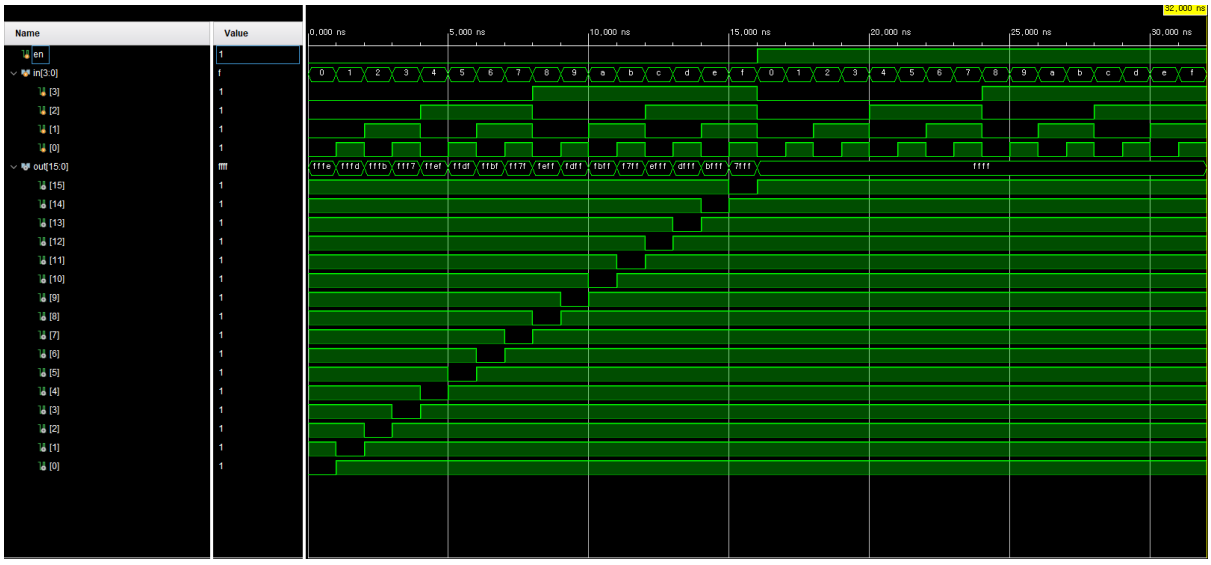
4.1. Lab3_1

[그림 9]는 4-to-16 decoder의 schematic이다. Decoder가 미리 주어져서, schematic에서 decoder의 내부는 숨김 처리했다. [그림 10]은 4-to-16 decoder의 simulation 파형이다.

[그림 9] Schematic of 4-to-16 decoder



[그림 10] 4-to-16 decoder의 simulation

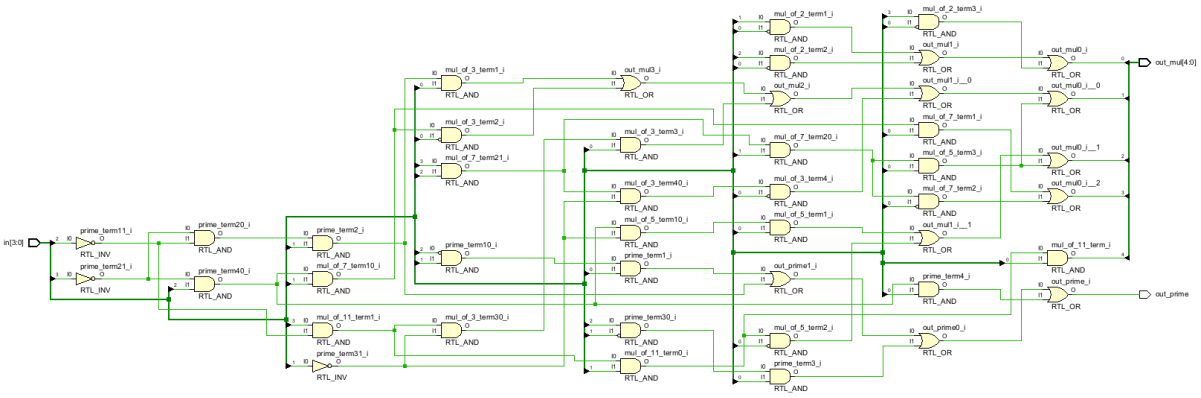


Active-low output이기 때문에 디코더의 출력 중 오직 하나만 0이 되는데, [그림 10]에서 f 이외의 16진수 값인 e, d, b, 7은 각각 BCD코드로 1110, 1101, 1011, 0111이므로 decoder가 0 출력을 오직 하나만 하는 것을 알 수 있다. Active-low enable이기 때문에 enable에 0이 입력되어야 decoder가 작동한다. Simulation 결과 디코더가 제대로 작동했음을 알 수 있다.

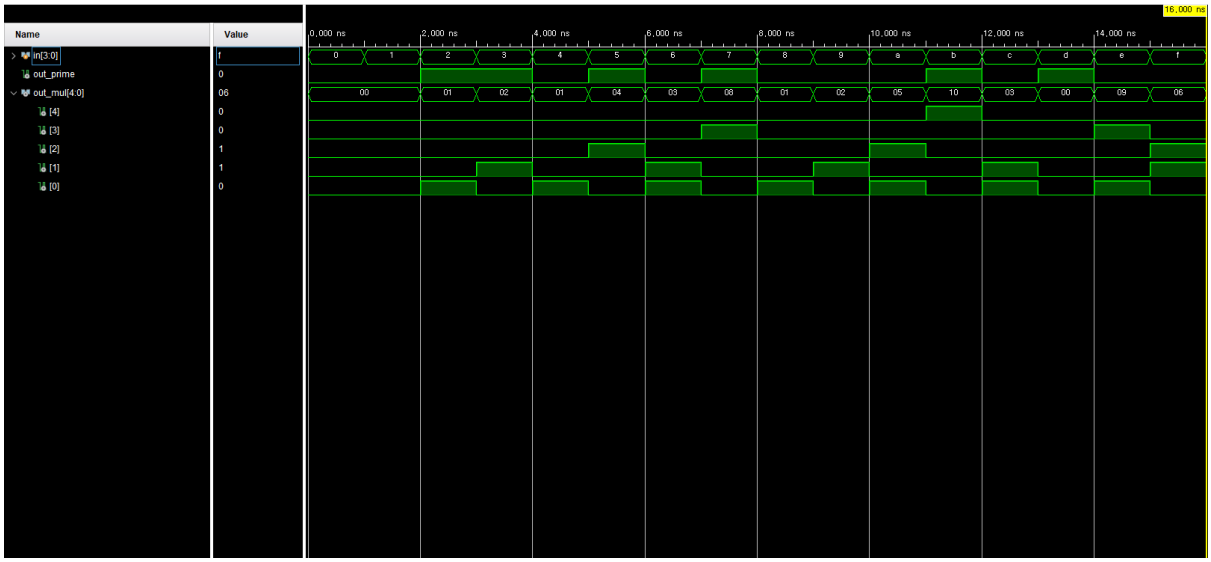
4.2. Lab3_2

소수판별기와 배수검출기의 schematic은 [그림 11]이고, 소수판별기와 배수검출기의 simulation은 [그림 12]이다.

[그림 11] 소수판별기와 배수검출기의 Schematic



[그림 12] 소수판별기와 배수검출기의 simulation

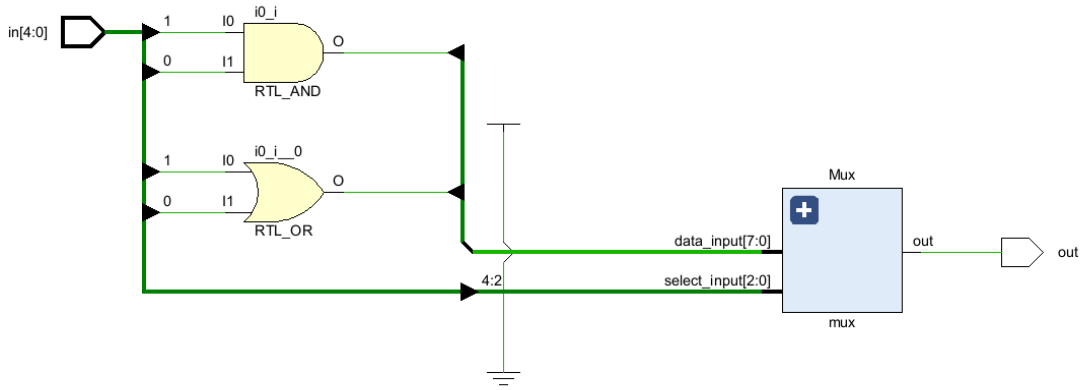


simulation 결과 소수 검출기인 out_prime은 소수인 2, 3, 5, 7, 11, 13에서만 1을 출력하고, 배수판별기는 각각 입력이 11, 7, 5, 3 ,2의 배수일 때 1을 출력한다. 소수판별기와 배수검출기가 정상작동했다.

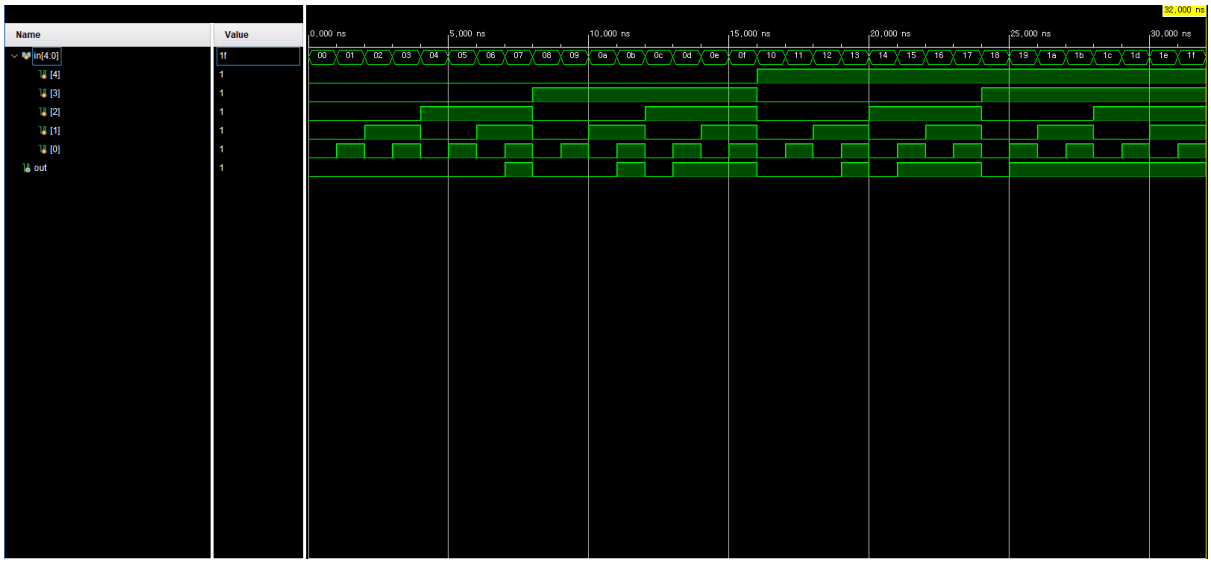
4.3. Lab3_3

5bit-majority function using 8:1mux의 schematic은 [그림 11]이고, 5bit-majority function using 8:1mux의 simulation은 [그림 12]이다.

[그림 13] 5bit majority function using 8:1 Mux의 schematic



[그림 14] Simulation of 5bit majority function using 8:1 Mux



Simulation 결과 입력 중 1이 3개 이상 들어왔을 때만 출력이 1이 되었고, 입력 중 1이 2개 이하인 경우 출력이 0이 되었다. 5bit majority function이 제대로 구현되었음을 알 수 있다.

5. 논의

Lab3_1에서는 작은 디코더를 이용해 더 큰 디코더로 확장하는 방법을 익힐 수 있었다. Lab3_2에서는 소수판별기과 배수검출기를 구현하면서 직접 디코더를 설계하는 방법을 익힐 수 있었다. Lab3_3에서는 멀티플렉서를 이용해 단순히 data seleting을 하는게 아닌 특정 기능(5bit majority function)을 하는 회로를 만드는 법을 익힐 수 있었다. Lab3_1에서 회로를 구현할 때, Active-high

와 Active-low가 섞여 있어 입출력이 헷갈렸다. 이는 Truth Table와 회로를 같이 두고 직접 입출력을 시뮬레이팅해 해결할 수 있었다.