실험 3. 디코더와 멀티플렉서

20210774 김주은

1. 개요

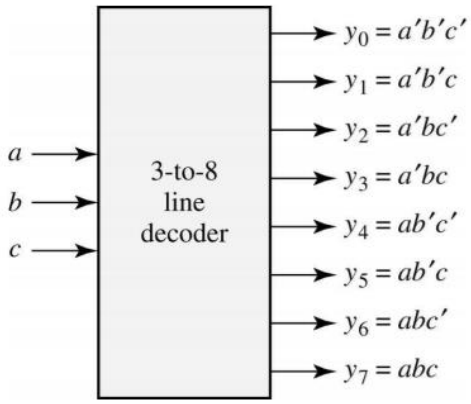
이번 실습의 목표는 multiple-output 회로를 대표하는 'decoder'와 multiple-input 회로를 대표하는 'multiplexer'의 기능을 이해하고 이를 사용해 회로를 구성하는 것이다.

첫번째 실습은 Active-low decoder을 확장하는 이론에 대해 이해한 후 이를 구현하는 것이다. 이번 실습에서는 2 to 4 decoder들을 가지고 4 to 16 decoder을 만든다. 그리고 두번째 실습은 특수 목적의 decoder을 구현한다. 이번 실습에서는 두 개의 decoder을 구현하게 되는데 하나는 소수 판별을 해주는 기능, 다른 하나는 배수 검출 기능이다. 마지막 실습은 멀티플렉서의 데이터 선택 기능을 이해하고 이를 활용하여 majority function을 구현하는 것이다.

2. 이론적 배경

1) decoder

디코더는 n개의 입력을 받고 2^n개의 출력을 하는 회로라고 할 수 있다. 여기서 2^n개의 출력 중 단 한 개만 1이다. 디코더는 'minterm generator'이라고 불리는데 이는 각 출력이 minterm에 따라 출력되기 때문이다. 디코더의 형태는 다음과 같다.



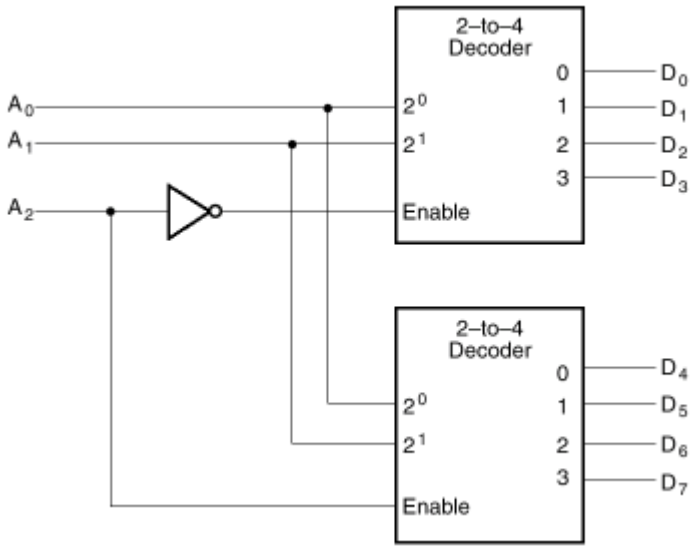
a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

디코더는 Enable input을 가지는데 이 EN의 경우 1일 때는 decoder가 활성화되고, EN이 0일 때는 decoder가 비활성화된다. 즉, EN은 decoder을 켜고 끄는 기능을 가진다. Decoder의 경우 다양한 목적으로 다양하게 활용되는 회로이다. 이번 실습에서도 두 가지의 목적을 위해 사용되는데 소수 판별과 배수 검출의 목적이다. EN을 가지는 decoder는 demultiplexer로 기능하기도 한다.

2) decoder 확장

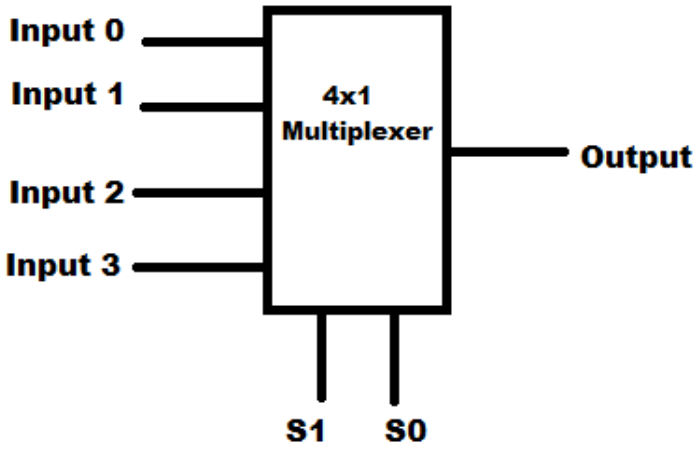
디코더의 확장은 디코더들을 연결하고, 그런 디코더들의 EN을 다른 디코더로 조절함으로써 더 많은 수의 입력과 출력을 가지는 디코더로 확장할 수 있다는 개념이다. 예를 들어, 2 to 4 디코더

를 두 개 사용하여 3 to 8 디코더를 구성해야 한다면, 아래와 같이 될 것이다.



3) multiplexer

멀티플렉서는 입력 신호들이 여러 개가 있을 때 그 중 하나를 선택하여 출력하는 회로라고 할 수 있다. 2^n개의 입력 신호가 들어온다면 n개의 selection 신호에 따라 어떤 출력 신호가 나올지 결정된다. 회로의 형태는 다음과 같다.

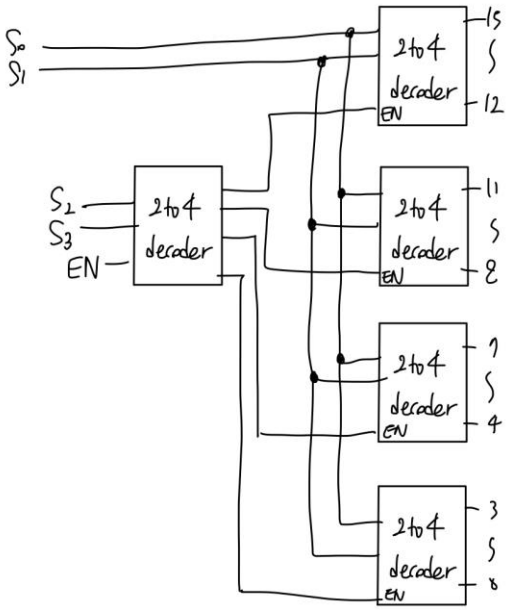


멀티플렉서를 활용하면 다양한 함수 표현도 가능하다. Minterm과 입력들이 곱해진 것의 SUM의 형태로 표현할 수 있으므로 이를 활용하여 다양한 함수 표현을 할 수 있다.

$$\text{Output} = \sum_{k=0}^{2^n-1} m_k I_k$$

3. 실험 준비

Lab 3-1)

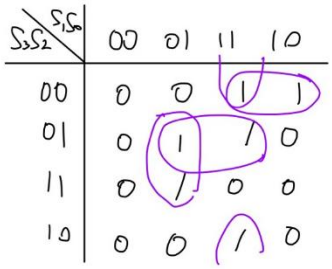


Decoder을 확장한 형태는 다음과 같다.

Lab 3-2)

1) 소수 판별기

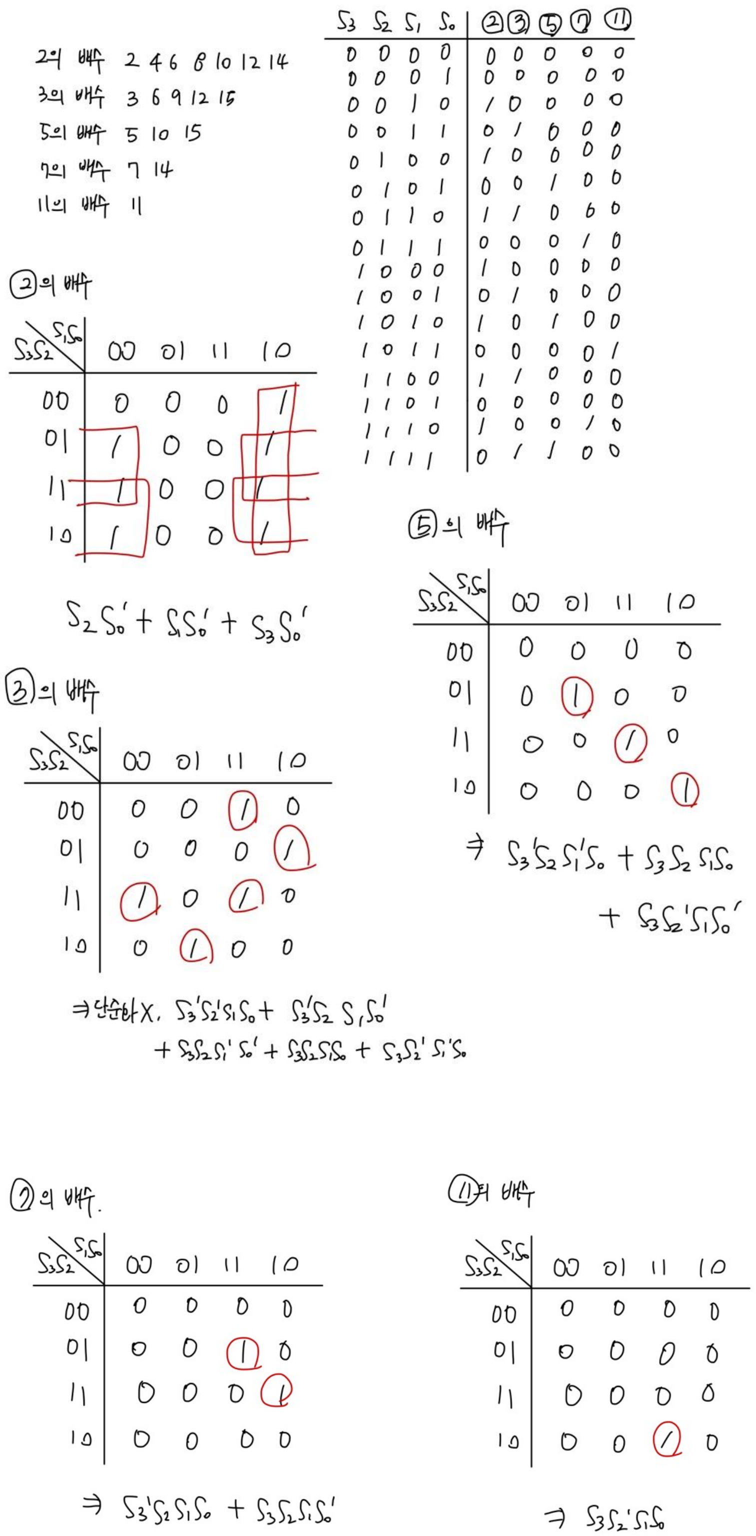
S_3	S_2	S_1	S_0	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0



$S_2S_1'S_0 + S_3'S_2S_0 + S_2'S_1S_0 + S_3'S_2'S_1$

먼저 truth table을 그리고 이를 바탕으로 k-map을 통해 단순화를 진행한다.

2) 배수 검출기



2의 배수 2 4 6 8 10 12 14
3의 배수 3 6 9 12 15
5의 배수 5 10 15
7의 배수 7 14
11의 배수 11

S_3	S_2	S_1	S_0	②	③	⑤	⑦	⑪
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	0	1	0	0	0
0	1	0	0	1	0	0	0	0
0	1	0	1	0	0	1	0	0
0	1	1	0	1	1	0	0	0
0	1	1	1	0	0	0	1	0
1	0	0	0	1	0	0	0	0
1	0	0	1	0	1	0	0	0
1	0	1	0	1	0	1	0	0
1	0	1	1	0	0	0	0	1
1	1	0	0	1	1	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	0	1	0	0	1	0
1	1	1	1	0	1	1	0	0

②의 배수

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

$S_2S_0' + S_1S_0' + S_3S_0'$

③의 배수

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	1	0
01	0	0	0	1
11	1	0	1	0
10	0	1	0	0

⇒ 단순화X. $S_3'S_2'S_1S_0 + S_3'S_2S_1S_0' + S_3S_2S_1'S_0' + S_3S_2S_1S_0 + S_3S_2'S_1'S_0$

⑤의 배수

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

⇒ $S_3'S_2S_1'S_0 + S_3S_2S_1S_0 + S_3S_2'S_1S_0'$

⑦의 배수.

S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	0	1
10	0	0	0	0

⇒ $S_3'S_2S_1S_0 + S_3S_2S_1S_0'$

⑪의 배수

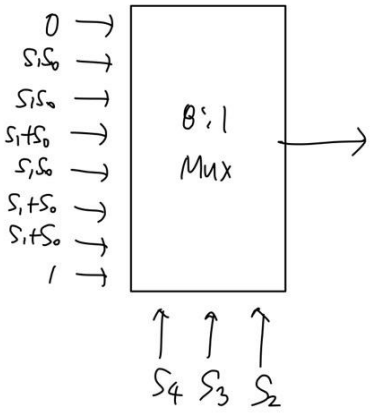
S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	1	0

⇒ $S_3S_2'S_1S_0$

먼저 truth table을 그리고 이를 바탕으로 k-map을 통해 단순화를 진행한다.

Lab 3-3)

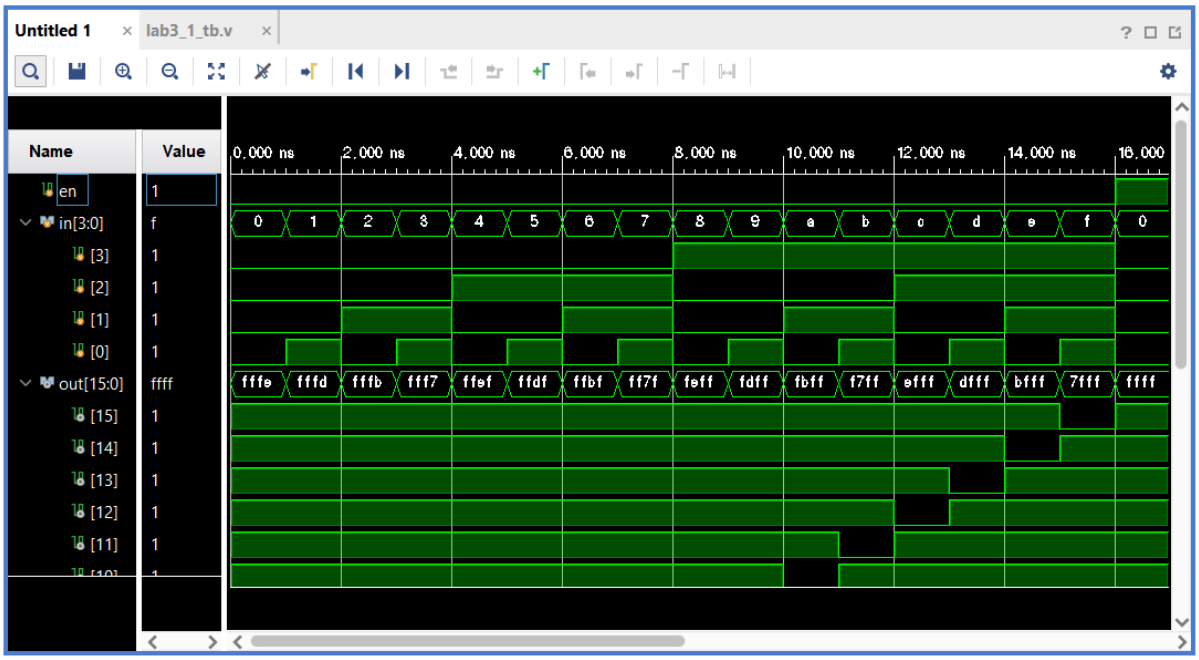
S_4	S_3	S_2	S_1	S_0	F	
0	0	0	0	0	0	0
0	0	0	0	1	0	
0	0	0	1	0	0	
0	0	0	1	1	0	
0	0	1	0	0	0	S_1S_0
0	0	1	0	1	0	
0	0	1	1	0	0	
0	0	1	1	1	1	
0	1	0	0	0	0	S_1S_0
0	1	0	0	1	0	
0	1	0	1	0	0	
0	1	0	1	1	1	
0	1	1	0	0	0	S_1+S_0
0	1	1	0	1	1	
0	1	1	1	0	1	
0	1	1	1	1	1	
1	0	0	0	0	0	S_1S_0
1	0	0	0	1	0	
1	0	0	1	0	0	
1	0	0	1	1	1	
1	0	1	0	0	0	S_1+S_0
1	0	1	0	1	1	
1	0	1	1	0	1	
1	0	1	1	1	1	
1	1	0	0	0	0	S_1+S_0
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	

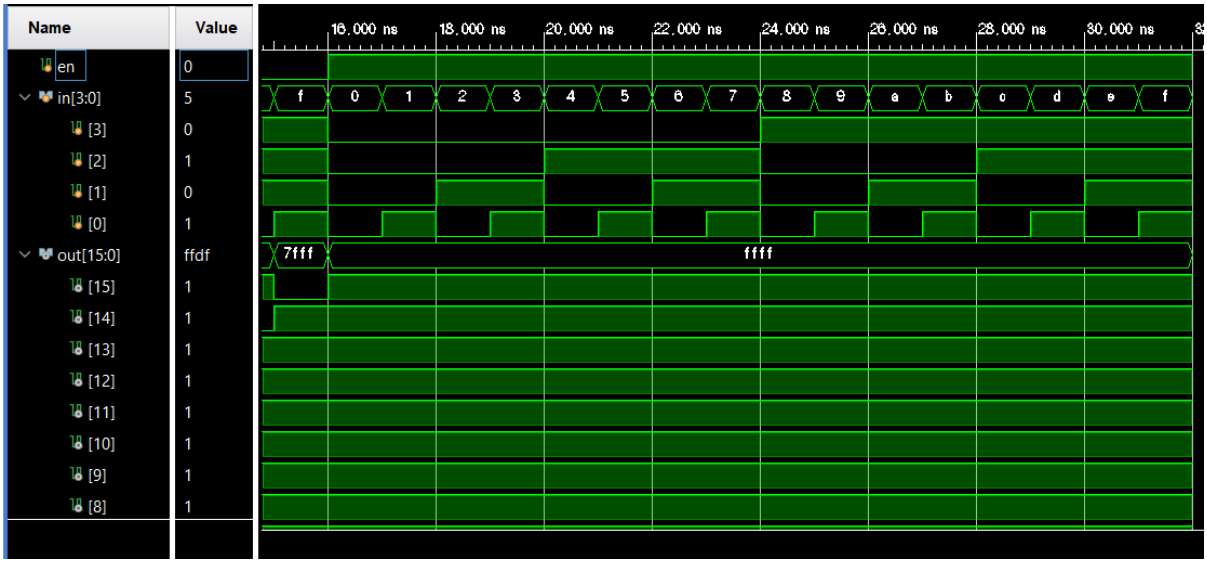


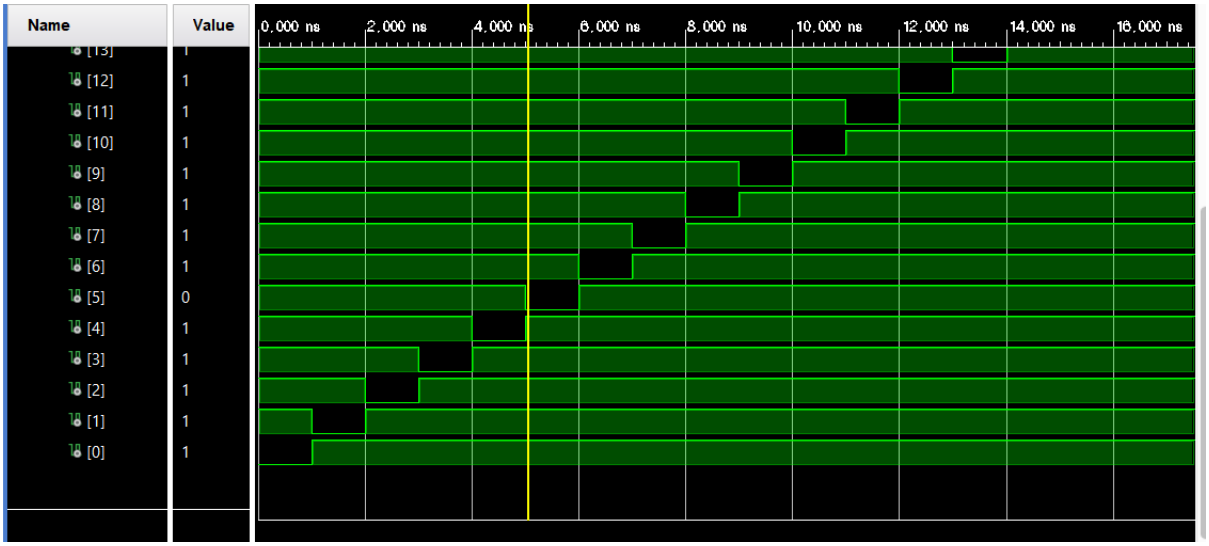
먼저 truth table을 그리고 단순화를 진행한다.

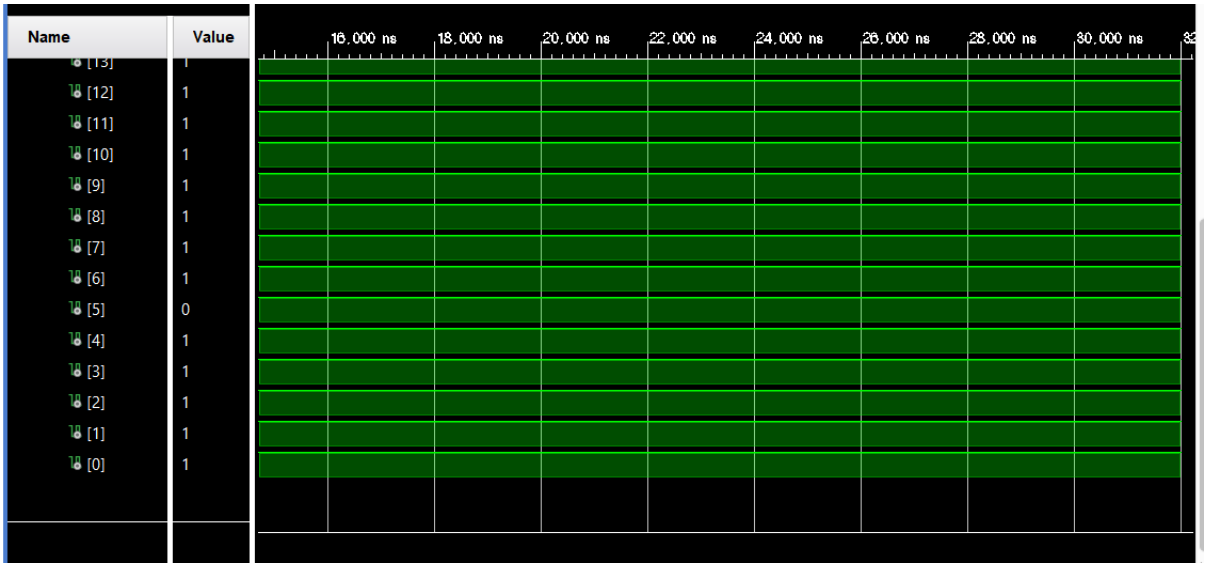
4. 결과

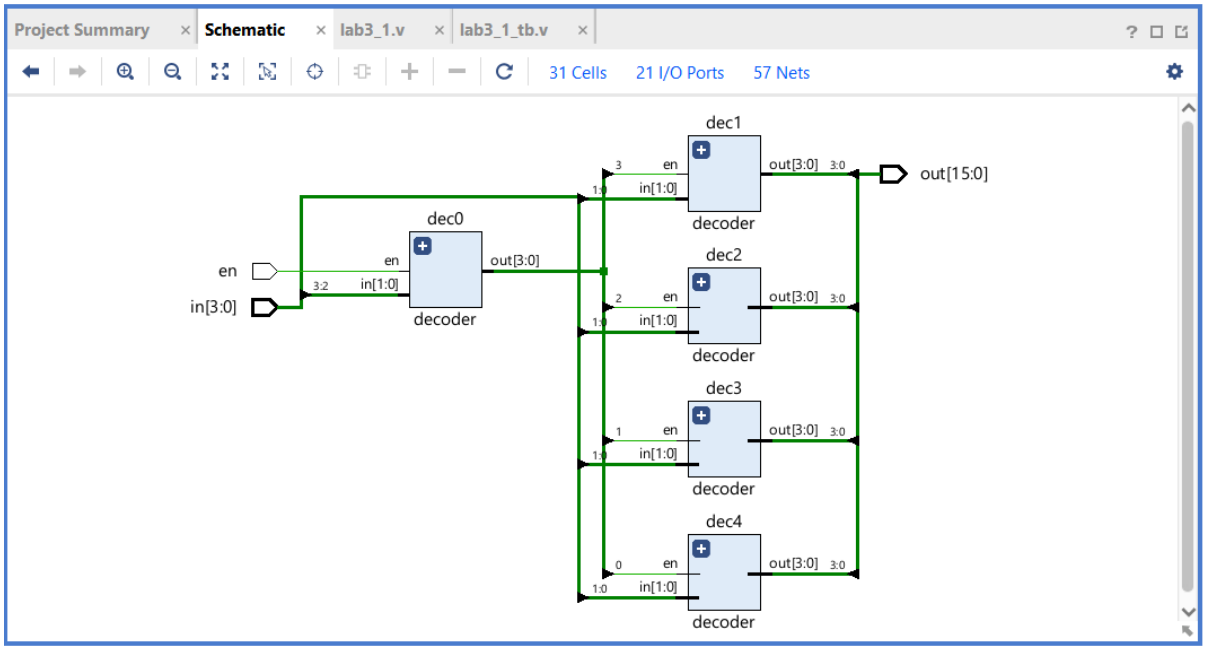
Lab 3-1)

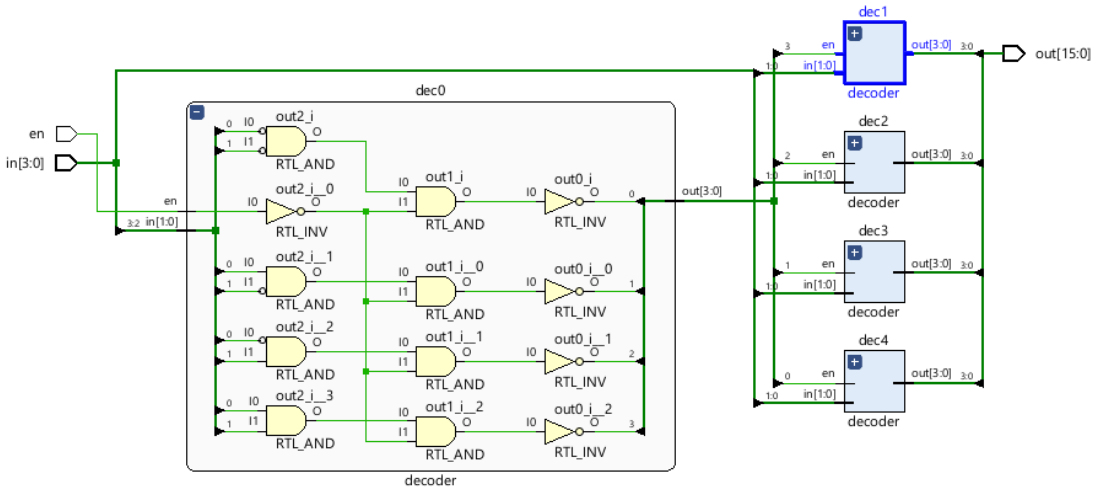

Untitled 1
lab3_1_tb.v
Name
Value
en
in[3:0]
[3]
[2]
[1]
[0]
out[15:0]
[15]
[14]
[13]
[12]
[11]
0.000 ns
2.000 ns
4.000 ns
6.000 ns
8.000 ns
10.000 ns
12.000 ns
14.000 ns
16.000

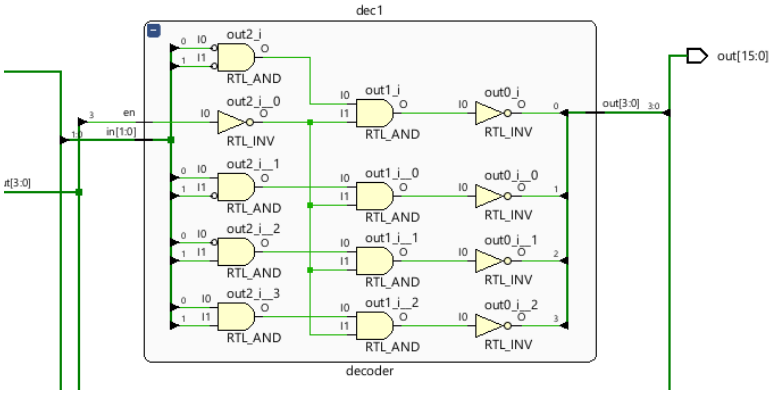

Name
Value
en
in[3:0]
[3]
[2]
[1]
[0]
out[15:0]
[15]
[14]
[13]
[12]
[11]
[10]
[9]
[8]
16.000 ns
18.000 ns
20.000 ns
22.000 ns
24.000 ns
26.000 ns
28.000 ns
30.000 ns
7fff
ffff

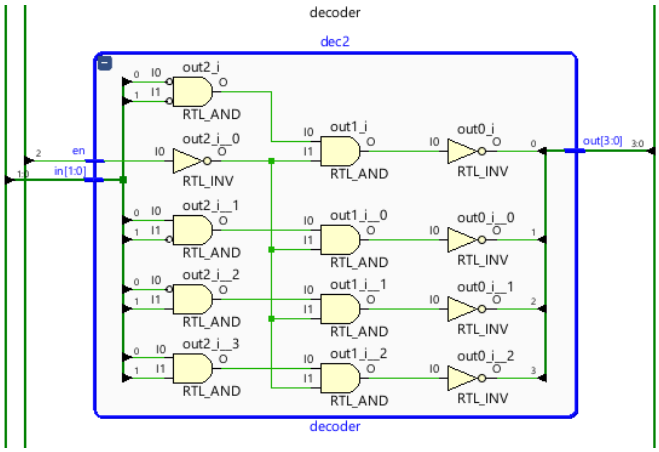

Name
Value
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
0.000 ns
2.000 ns
4.000 ns
6.000 ns
8.000 ns
10.000 ns
12.000 ns
14.000 ns
16.000 ns

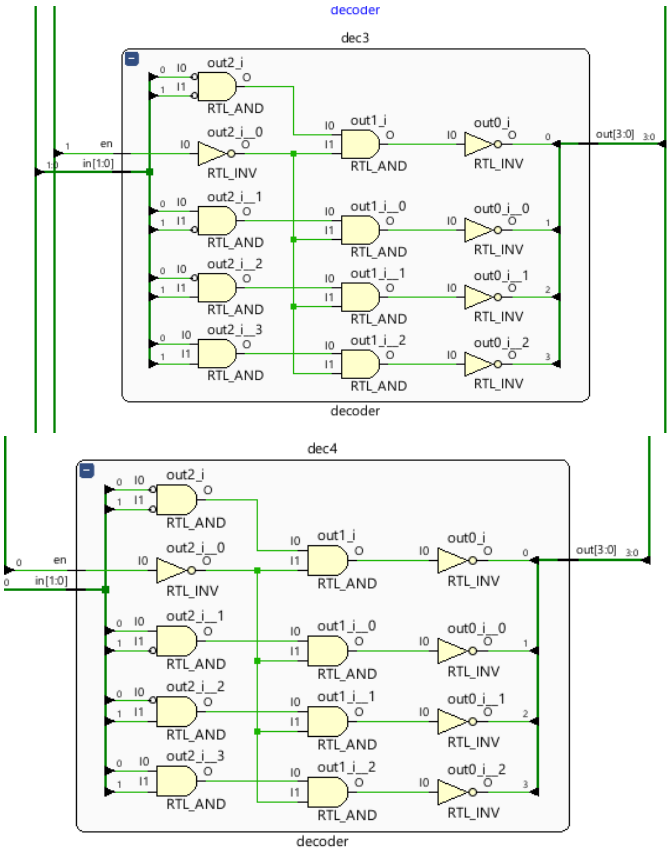

Name
Value
16,000 ns
18,000 ns
20,000 ns
22,000 ns
24,000 ns
26,000 ns
28,000 ns
30,000 ns
[12] 1
[11] 1
[10] 1
[9] 1
[8] 1
[7] 1
[6] 1
[5] 0
[4] 1
[3] 1
[2] 1
[1] 1
[0] 1


Project Summary
Schematic
lab3_1.v
lab3_1_tb.v
31 Cells
21 I/O Ports
57 Nets
dec1
dec0
dec2
dec3
dec4
decoder
en
in[3:0]
in[1:0]
out[3:0]
out[15:0]

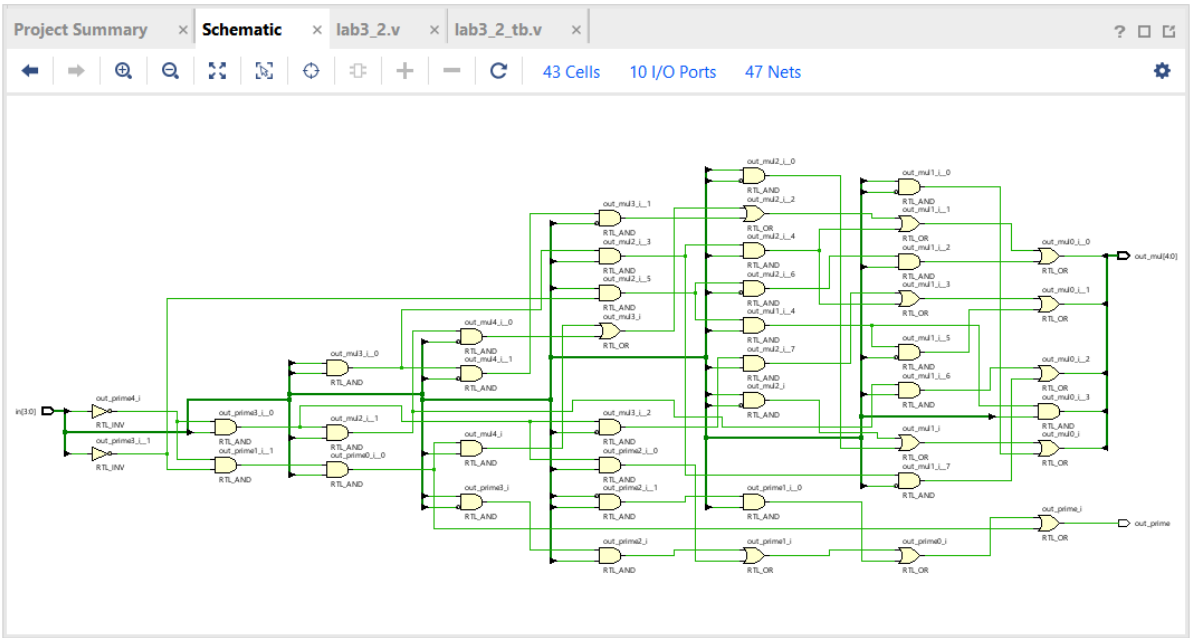

dec1
dec0
dec2
dec3
dec4
decoder
en
in[3:0]
in[1:0]
out[3:0]
out[15:0]
RTL_AND
RTL_INV
out2_i
out2_i_0
out2_i_1
out2_i_2
out2_i_3
out1_i
out1_i_0
out1_i_1
out1_i_2
out0_i
out0_i_0
out0_i_1
out0_i_2

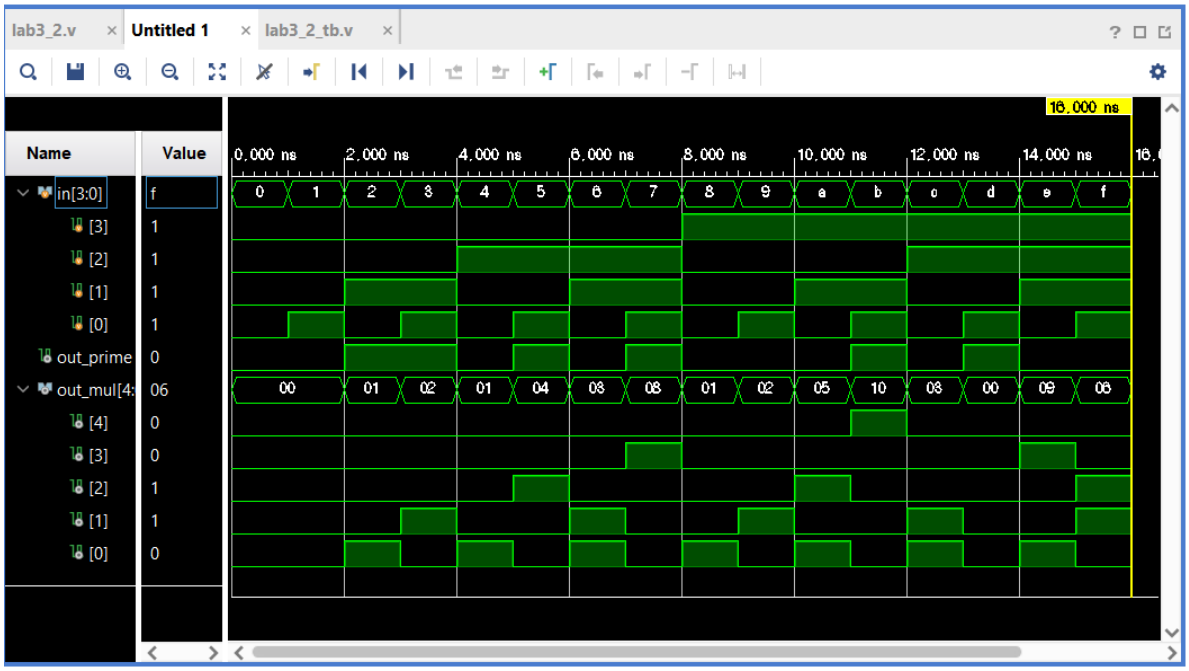

dec1
decoder
en
in[1:0]
out[3:0]
out[15:0]
RTL_AND
RTL_INV
out2_i
out2_i_0
out2_i_1
out2_i_2
out2_i_3
out1_i
out1_i_0
out1_i_1
out1_i_2
out0_i
out0_i_0
out0_i_1
out0_i_2


decoder
dec2
decoder
en
in[1:0]
out[3:0]
RTL_AND
RTL_INV
out2_i
out2_i_0
out2_i_1
out2_i_2
out2_i_3
out1_i
out1_i_0
out1_i_1
out1_i_2
out0_i
out0_i_0
out0_i_1
out0_i_2

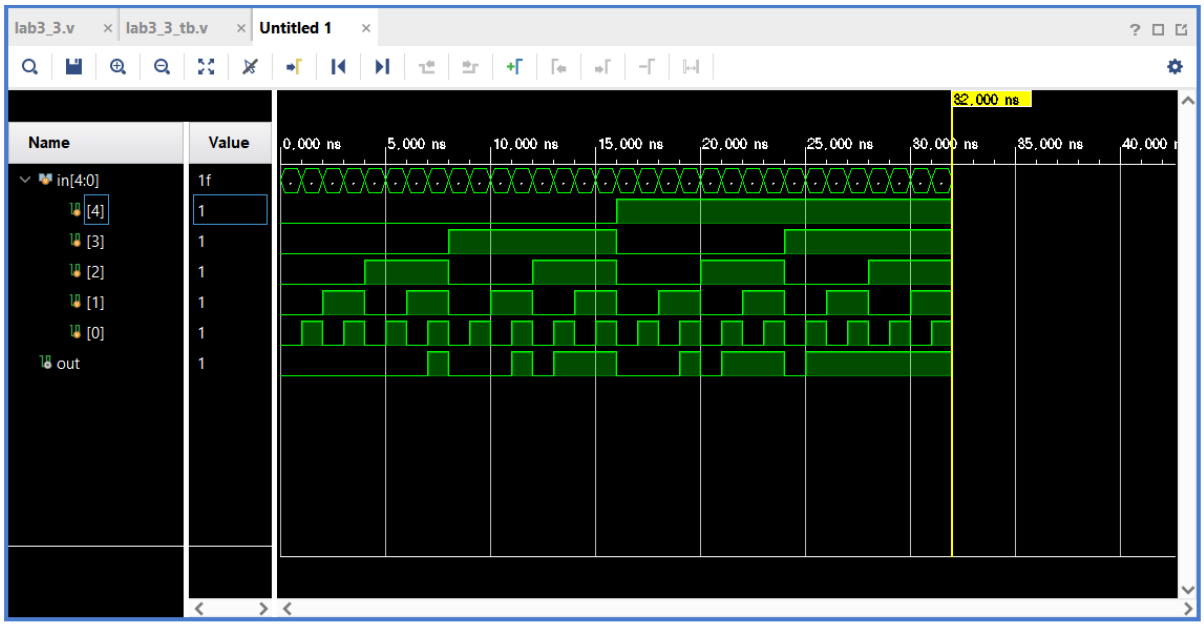

decoder
dec3
out2_i
RTL_AND
out2_i_0
RTL_INV
out1_i
RTL_AND
out0_i
RTL_INV
out2_i_1
out1_i_0
out0_i_0
out2_i_2
out1_i_1
out0_i_1
out2_i_3
out1_i_2
out0_i_2
en
in[1:0]
out[3:0]
decoder
dec4
out2_i
out2_i_0
out1_i
out0_j
out2_i_1
out1_i_0
out0_j_0
out2_i_2
out1_i_1
out0_j_1
out2_i_3
out1_i_2
out0_j_2
en
in[1:0]
out[3:0]
decoder

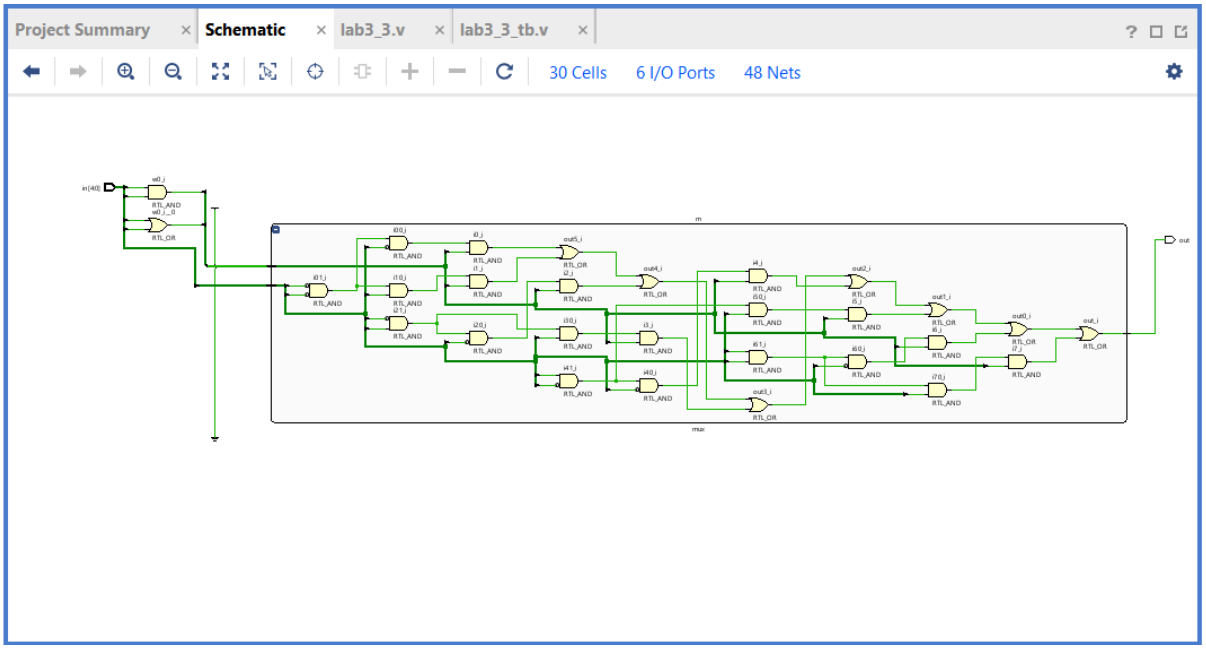
Lab 3-2)

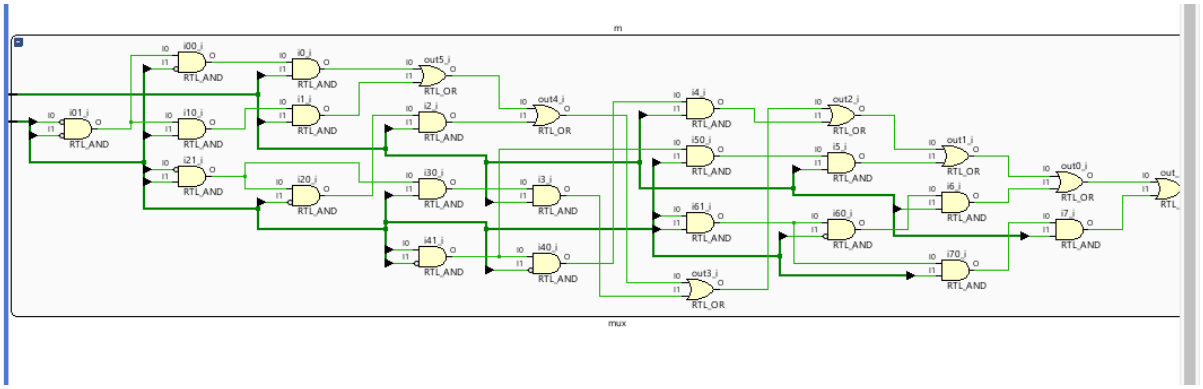

Project Summary
Schematic
lab3_2.v
lab3_2_tb.v
43 Cells
10 I/O Ports
47 Nets


lab3_2.v
Untitled 1
lab3_2_tb.v
Name
Value
in[3:0]
f
[3]
1
[2]
1
[1]
1
[0]
1
out_prime
0
out_mul[4:
06
[4]
0
[3]
0
[2]
1
[1]
1
[0]
0
16.000 ns
0.000 ns
2.000 ns
4.000 ns
6.000 ns
8.000 ns
10.000 ns
12.000 ns
14.000 ns
0 1 2 3 4 5 6 7 8 9 a b c d e f
00 01 02 01 04 08 08 01 02 05 10 08 00 09 06

Lab 3-3)


lab3_3.v
lab3_3_tb.v
Untitled 1
Name
Value
in[4:0]
1f
[4]
1
[3]
1
[2]
1
[1]
1
[0]
1
out
1
32.000 ns
0.000 ns
5.000 ns
10.000 ns
15.000 ns
20.000 ns
25.000 ns
30.000 ns
35.000 ns
40.000





5. 논의

3-1)

2 to 4 decoder을 4 to 16 decoder을 만드는 과정은 이미 시험 기간에 많은 연습을 했기에 덜 어려웠다. 이미 만들어진 decoder 모듈을 활용하여 4 to 16 decoder을 이론적인 게 아닌 직접 베릴로그 코딩을 해보면서 더 와 닿을 수 있었다.

하지만 decoder 모듈을 직접 어떻게 활용해야 하는지에 대한 베릴로그 문법에 익숙하지 않아 조금 고생하였던 것 같다. 그 중 가장 시간을 투자했던 부분은 decoder의 경우 input과 output이 모두 벡터형이었는데 이를 매개변수로 어떻게 넘겨줘야 하는지에 대한 부분이었다. 처음에는 예를 들어 out[15:0] 중 15~12 만 쓰기 위하여 out[15:12] 라는 식의 코드로 구현을 해봤지만 오류가 뜨고 실패하였고 다시 열심히 찾아보아 모색한 방법이 중괄호를 쓰는 방법이었다. 그래서 {out[15],out[14],out[13],out[12]} 으로 구현하여 구현에 성공하였다.

3-2)

소수 판별기와 배수 검출기는 단순히 진리표를 그린 다음 단순화를 한 다음에 회로를 구현하였는데 이전 decoder의 실습보다는 난이도가 높지 않았다. 하지만 진리표가 꽤나 복잡했는데 이는 output이 총 5개였던 것이 원인이었던 것 같다. 그래서 각 output에 대해 구현하느라 많이 힘들기는 하였지만 이보다 더 효율적이고 최적화된 방법을 모색하여 찾는다면 그 방법을 선택하는 것이 좋을 것 같다고 생각하였다.

3-3)

Multiplexer은 decoder보다 훨씬 난이도가 낮았고 덜 어려웠다. 그리고 2^5개에 대한 truth table을 그리는 과정이 조금 번거롭고 귀찮긴 했지만 난이도는 훨씬 낮았던 것 같다.