디지털시스템설계 실습 보고서

3. 디코더와 멀티플렉서

20210716 최대현

1. 실습 개요

이번 실습에서는 Multiple-output 회로를 대표하는 디코더와 Multiple-input 회로를 대표하는 멀티플렉서의 기능을 이해하고 이를 사용해 회로를 구성한다. active-low decoder expansion을 이해하고 구현하고, 소수 판별기, 배수 검출기와 같은 특수목적 디코더를 구현하며, 5bit Majority function을 additional gate를 이용한 multiplexer simplify로 구현한다.

2. 이론적 배경

1) 디코더(decoder)

디코더는 n개의 입력을 받아 최대 2^n개의 고유 출력을 가지는 회로다. n개의 이진 입력과 2^n개의 서로 다른 출력을 가지는 경우를 binary decoder라고 한다. decoder에는 data input 이외에 Enable input이 추가로 존재한다. Enable input은 디코더를 켜거나 끄기 위해서 사용된다.

X	Y	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

사진 1: 디코더의 진리표

2) 디코더 확장(decoder expansion)

EN(enable input)을 사용하여 디코더를 켜거나 끌 수 있다는 점을 이용해 여러 개의 디코더를 연결하여 더 많은 수의 입력을 처리하는 것.

3) 특수 목적 디코더

- **소수 판별기**

  주어진 입력이 소수일 때 참을 출력하는 디코더이다.

- **배수 검출기**

  입력이 각각 2, 3, 5, 7의 배수인지를 나타내는 디코더이다. 이 때, 입력에 따라 하나의 출력에 대해서 여러 출력이 동시에 참일 수도 있다.

4) 멀티플렉서 (Multiplexer, MUX)

선택 신호에 따라 여러 입력 신호 중 하나를 골라 출력하는 장치이다. 주로 2^n개의 입력 신호 중 한 개를 선택하여 출력한다. 2^n개의 data input에 대해 n bit control input으로 조절하여 사용한다.

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

사진 2: 4X1 멀티플렉서의 진리표

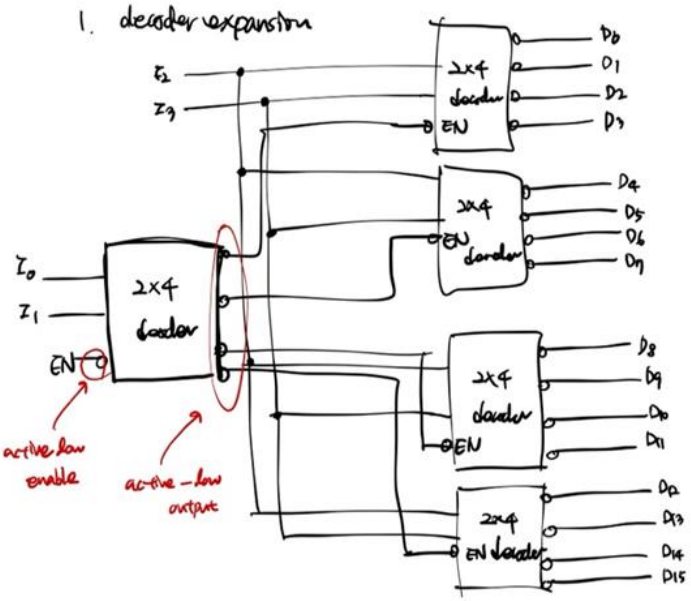
5) Majority / Minority function

홀수 개의 입력에 대해 다수 또는 소수를 차지하는 입력을 나타내는 함수.

3. 실험 전 준비

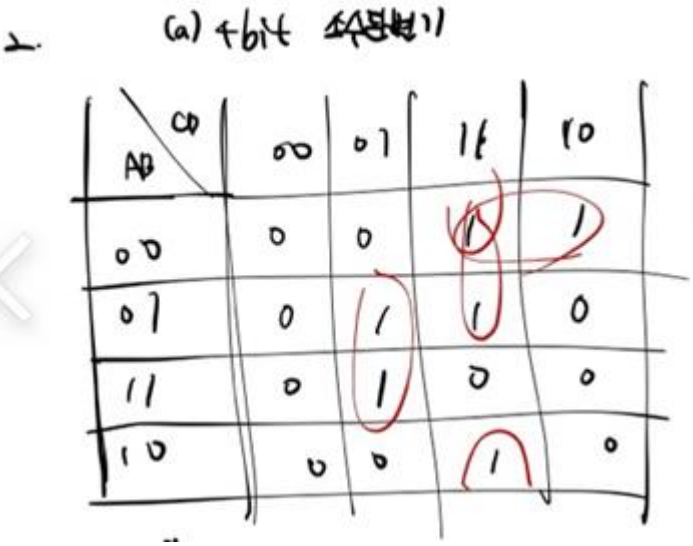
3-1. lab3_1

Decoder expansion



3-2. lab3_2

i) 4 bit 소수 판별기

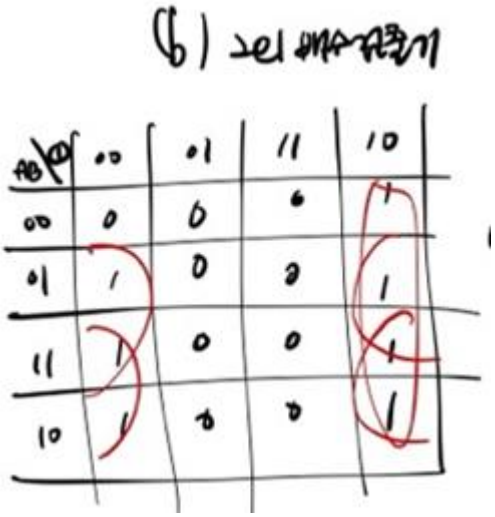


4 bit 소수 판별기를 K-map과 bool 연산법칙으로 simplify 하여 나타내면 다음과 같다.

F(A,B,C,D) = BC'D+B'CD+A'B'C+A'CD

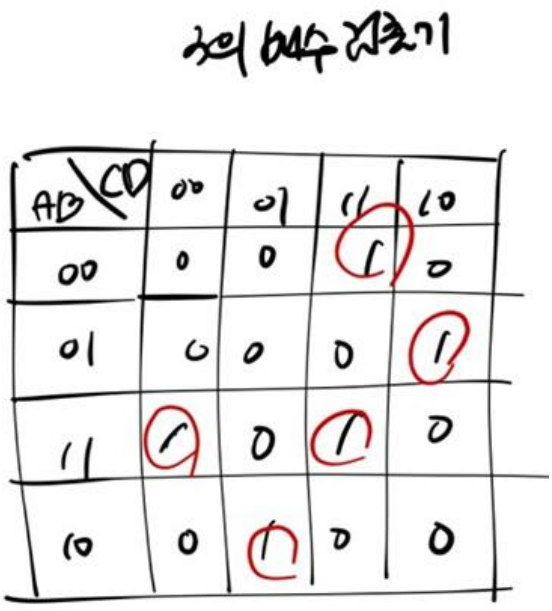
ii) 배수 검출기

2의 배수 검출기



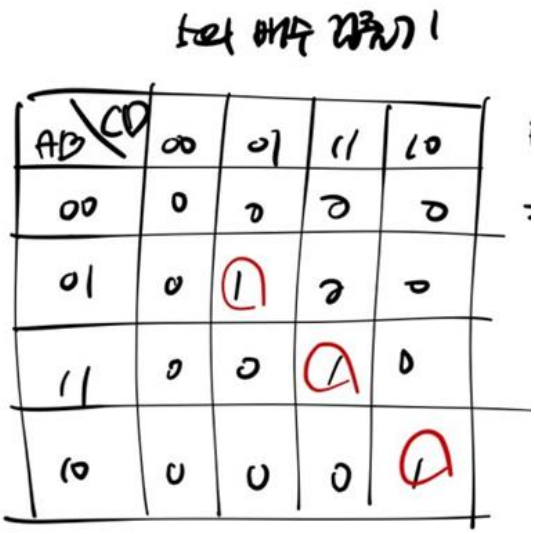
F(A,B,C,D) = AD'+BD'+CD' = D'(A+B+C)

3의 배수 검출기



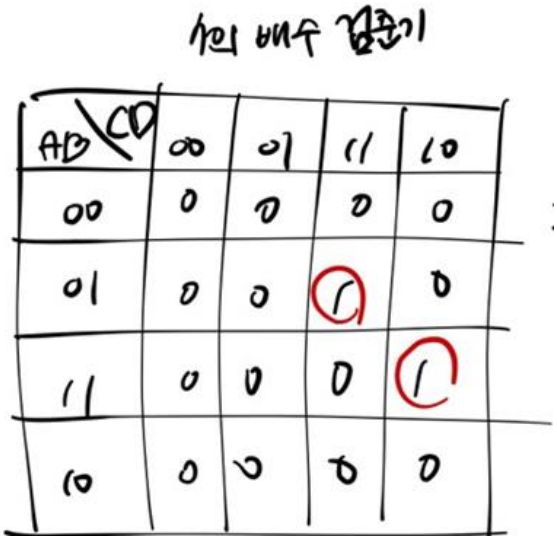
F(A,B,C,D) = A'B'CD+ A'BCD'+ABC'D'+ABCD+AB'C'D

5의 배수 검출기



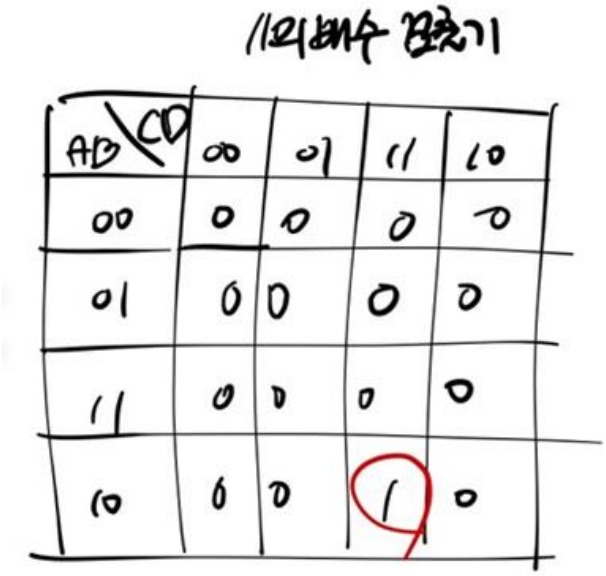
F(A,B,C,D) = A'BC'D+ABCD+AB'CD'

7의 배수 검출기



F(A,B,C,D) = A'BCD+ABCD'

11의 배수 검출기



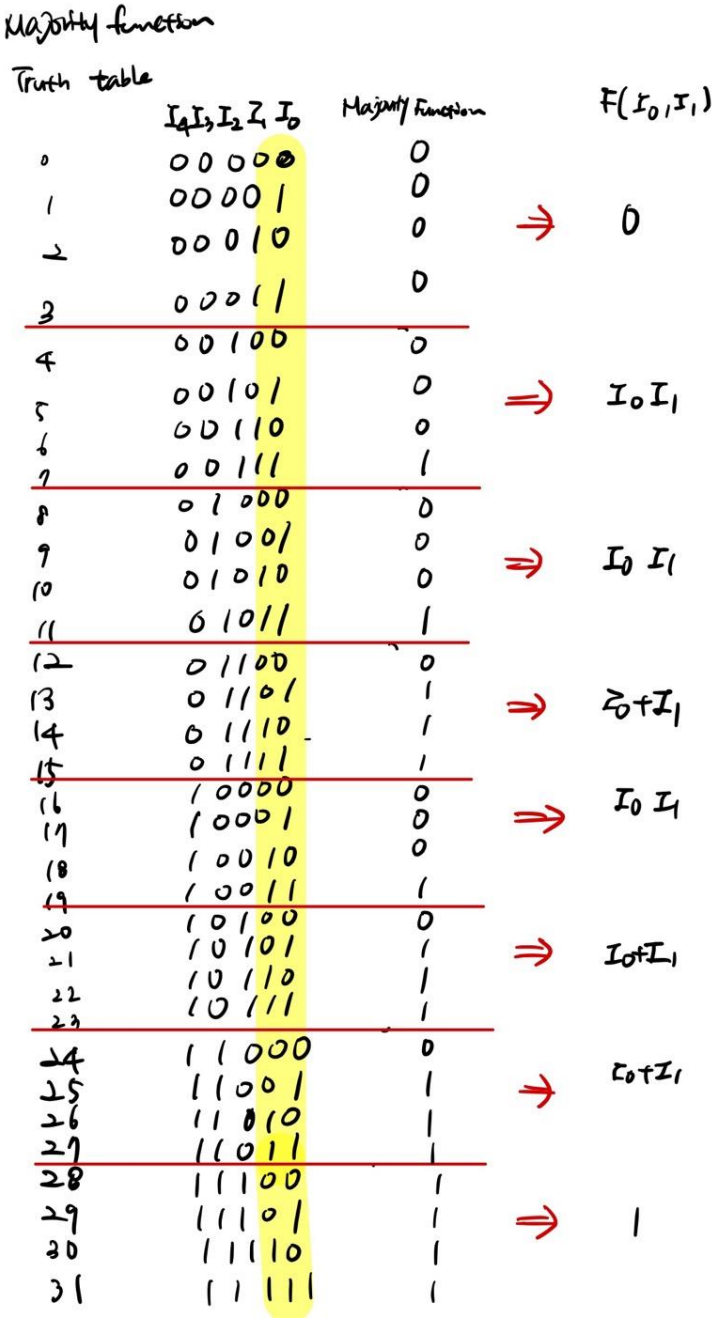
F(A,B,C,D) = AB'CD

3-3. lab3_3

5-bit majority function by 8X1 MUX

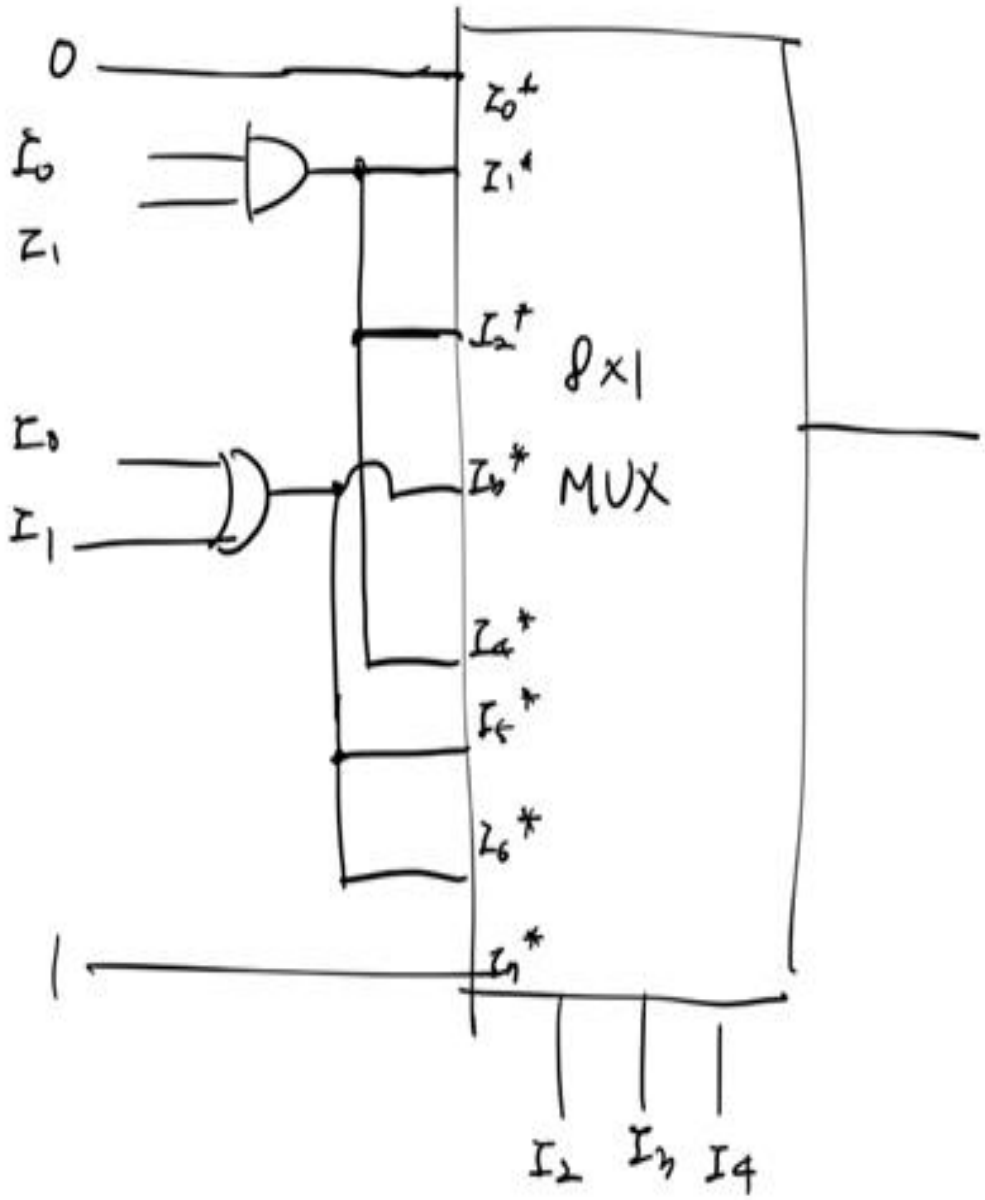
Majority function을 big endian 방식으로 input을 가져와서 구현하였고, 편의 상 하위 2 비트 (i0, i1)에 대해 additional gate를 구현하였다.

Truth table



8X1 MUX

3. 5-bit majority function by 8×1 MUX



4. 본 실험

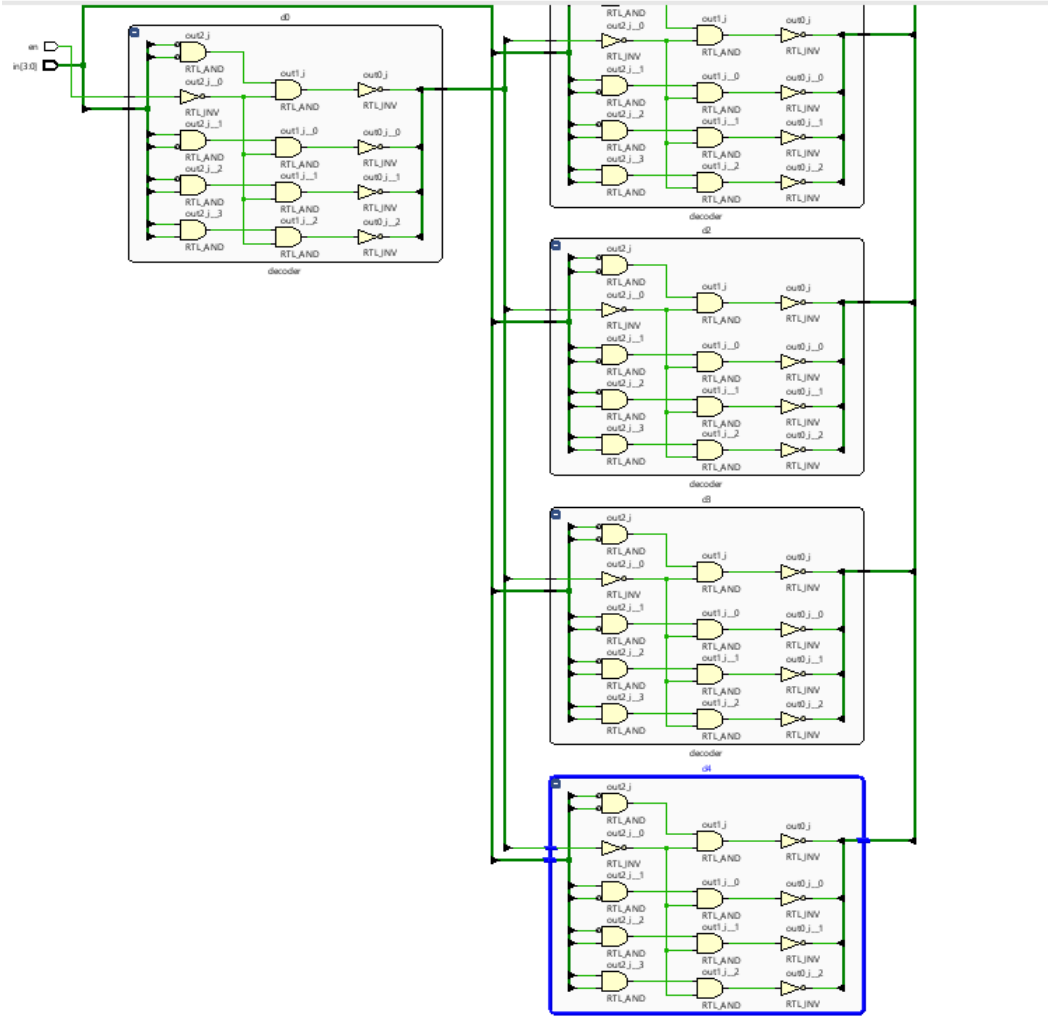
본 실험은 Verilog 프로그래밍으로 회로를 구현하며 진행되었다.

4-1. lab3_1.v

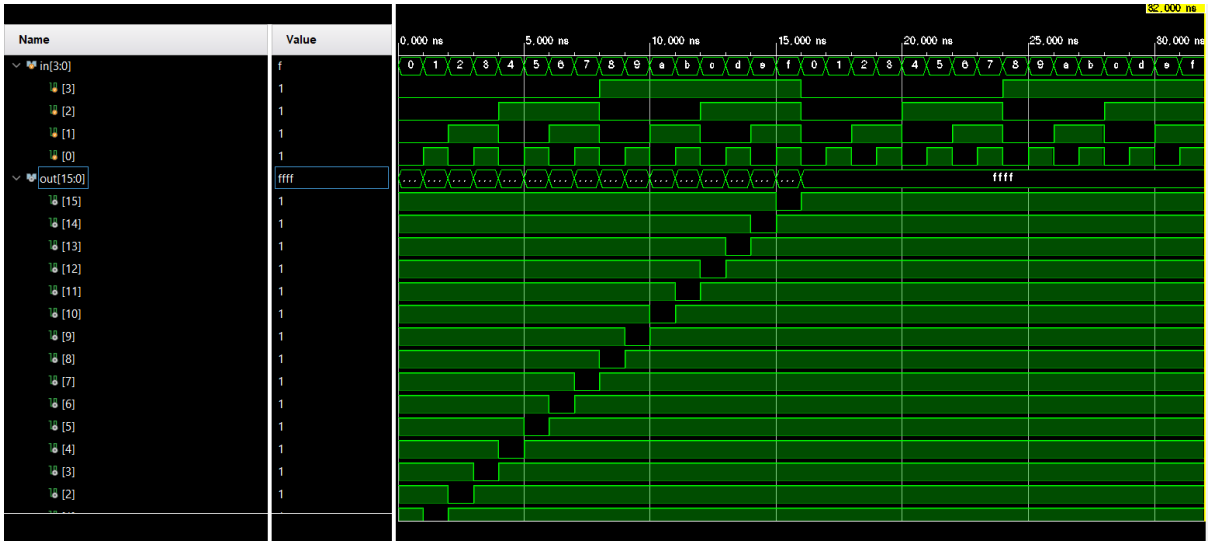
Decoder expansion을 구현한 Verilog 코드이다.

```
wire [3:0] a;
decoder d0(en, in[3:2], a); // en 물려주는 2X4 decoder , output은 [3:0] wire여야 함
decoder d1(a[0], in[1:0], out[3:0]); // output 0~3
decoder d2(a[1], in[1:0], out[7:4]); // output 4~7
decoder d3(a[2], in[1:0], out[11:8]); // output 8~11
decoder d4(a[3], in[1:0], out[15:12]); // output 12~15
```

<source code>



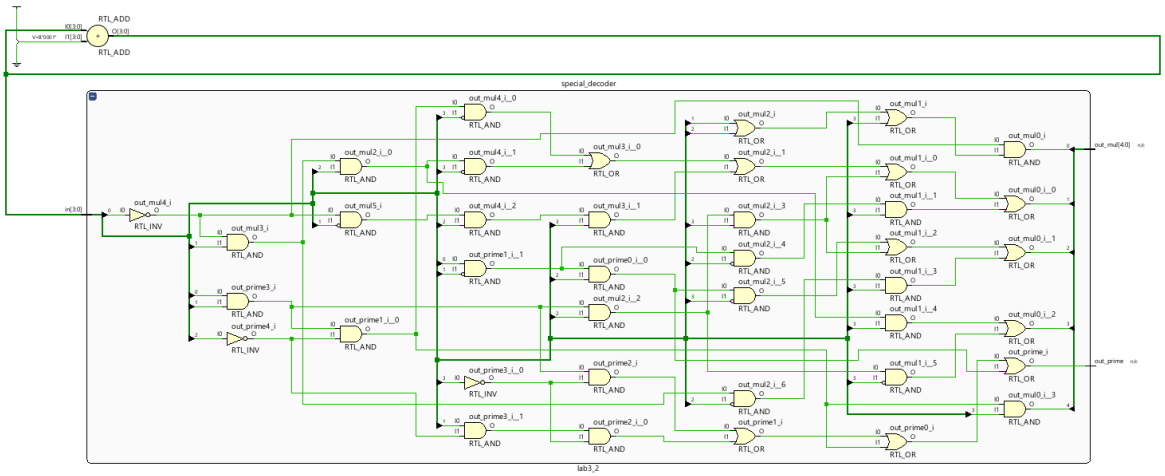
<Schematic 기능으로 출력한 회로도>



<simulation 결과>

4-2. lab3_2.v

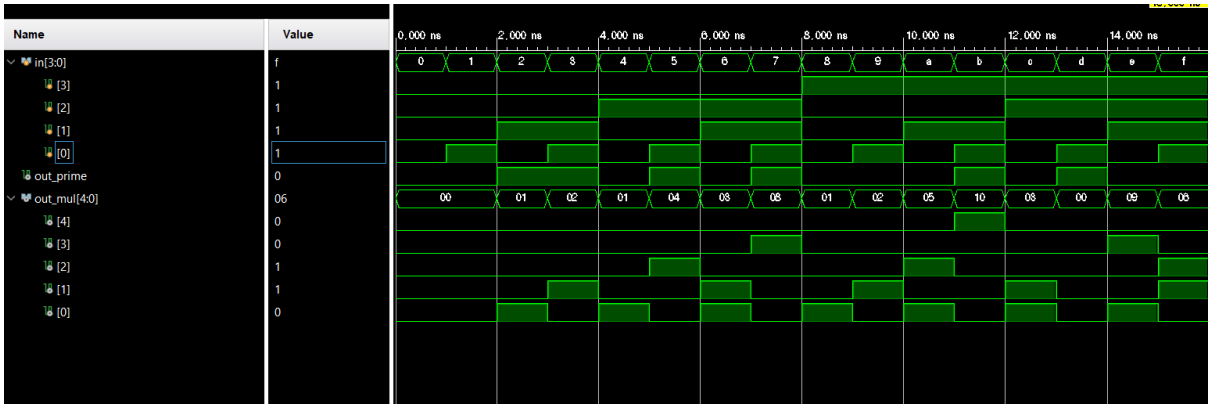
특수 목적 디코더를 구현한 코드이다. 이 때 0은 소수도, 배수도 아닌 것으로 처리했다. (각 decoder에 대해 output을 0으로 처리함)



<Schematic 기능으로 출력한 회로도>

```
//////////////////////////
/* Add your code here */
assign out_mul[0] = ~in[0]&(in[1]|in[2]|in[3]); // 2의 배수 검출기
assign out_mul[1] = (in[0]&in[1]&~in[2]&~in[3])|(~in[0]&in[1]&in[2]&~in[3])|(~in[0]&~in[1]&in[2]&in[3])|(in[0]&in[1]&in[2]&in[3])|(in[0]&~in[1]&~in[2]&in[3]);// 3의 배수 검출기
assign out_mul[2] = (in[0]&~in[1]&in[2]&~in[3])|(in[0]&in[1]&in[2]&in[3])|(~in[0]&in[1]&~in[2]&in[3]);// 5의 배수 검출기
assign out_mul[3] = (~in[0]&in[1]&in[2]&in[3])|(in[0]&in[1]&in[2]&~in[3]);// 7의 배수 검출기
assign out_mul[4] = (in[0]&in[1]&~in[2]&in[3]);// 11의 배수 검출기
assign out_prime = (in[0]&in[1]&~in[3])|(in[1]&~in[2]&~in[3])|(in[0]&in[1]&~in[2])|(in[0]&~in[1]&in[2]); // prime number indicator
//////////////////////////
```

<source code>



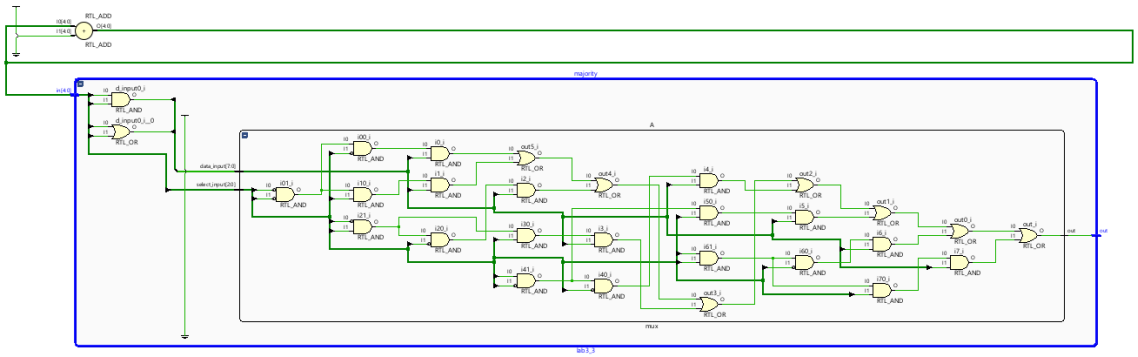
<simulation 결과>

4-3. lab3_3.v

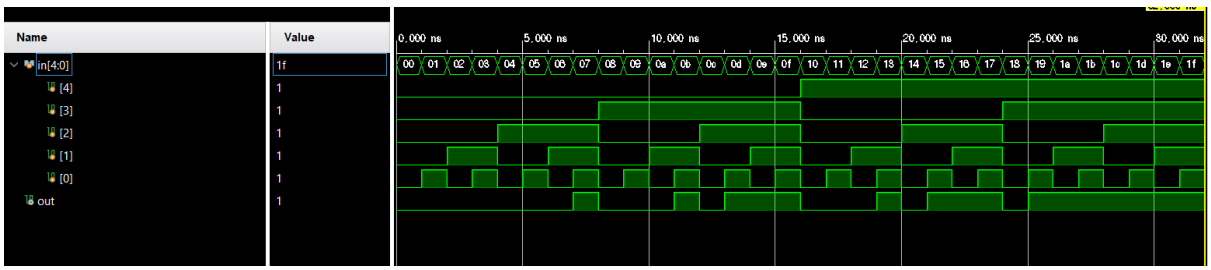
5 bit majority function을 multiplexer로 출력하는데, additional gate를 이용해서 8X1 MUX로 구현한 코드이다.

```
////////////////////////
wire [7:0] d_input;
assign c_input = in[4:2];
assign d_input[0] = 0;
assign d_input[1] = in[1]&in[0];
assign d_input[2] = in[1]&in[0];
assign d_input[3] = in[1]|in[0];
assign d_input[4] = in[1]&in[0];
assign d_input[5] = in[1]|in[0];
assign d_input[6] = in[1]|in[0];
assign d_input[7] = 1;
mux A(d_input, in[4:2], out);
////////////////////////
```

<source code>



<Schematic 기능으로 출력한 회로도>



<simulation 결과>

5. 결론

Lab 3_1에서는 지금까지 이론적으로만 해 봤던 decoder expansion을 직접 해 볼 수 있었고, lab 3_2에서는 디코더를 응용하여 소수 판독기와 배수 검출기라는 특수 디코더를 구현해볼 수 있었다. 또한, lab 3_3에서는 5비트, 즉 32개의 input이 필요한 multiplexer를 additional gate를 이용하여 8X1 multiplexer로 간단히 만들어볼 수 있었다.

6. 고찰

이번 실습에서는 이론적으로 학습했던 디코더와 멀티플렉서를 Verilog로 직접 구현해보는 시간을 가졌다. Schematic 기능과 testbench 코드를 이용하여 decoder expansion과 multiplexer simplify가 원활하게 되었음을 알 수 있었다. 또한, 기존에는 decoder라고 하면 여러 개의 output 중 하나씩만 출력하는 것들만 다뤘었는데, 배수 판별기에서는 여러 개의 output이 동시에 구현되는 것이 인상 깊었다. 유익한 수업이었다.