CSED273 Lab 3 보고서

2022.04.21

20210643 김현준

1. 개요

수업에서 다루었던 디코더와 멀티플렉서의 기능을 이해하고 회로를 구성한다. 특히 대표적인 Multiple-output 회로인 디코더와 대표적인 Multiple-input 회로인 멀치플렉서를 이해하여 회로 이해도를 높인다. 세부적으로는 수업시간에 배운 Active-high 디코더에서 나아가 Active-low 디코더의 확장을 이해 및 구현하고, 소수 판별 등의 특수 목적을 가진 디코더를 구현하고, Majority function을 멀티플렉서를 이용하여 구현한다. 수업시간에 배운 내용을 실습으로 구현함으로써 수업 이해도 또한 높인다.

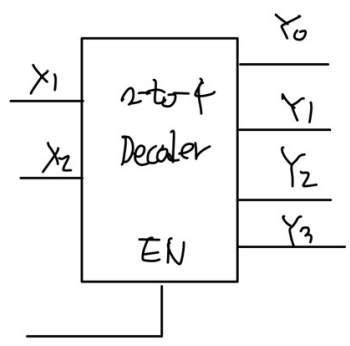
2. 이론적 배경

1) 디코더 (Decoder)

디코더는 n개의 입력을 받아 2^n개의 출력을 가지는 Multiple-output 회로이다. n개의 이진 입력과 2^n 개의 서로 다른 출력을 가지는 경우에는 각 출력이 minterm이 되기 때문에 minterm generator라고 부르기도 한다. 또한, 디코더에는 n개의 입력 외에도 Enable 입력이 존재하는데, 이것은 디코더를 켜고 끄는 스위치 역할을 하게 된다. 추가로 디코더는 인코더와 정 반대의 기능을 수행한다.

디코더의 입력과 출력을 표현할 때는 n-to-2^n 또는 k-of-2^n이라는 표현을 사용한다. 전자의 경우 n개의 입력을 받아 2^n개의 출력을 가진다는 것을 표현하는 것으로, 입력과 출력의 관계에 초점을 맞춘 것이고, 후자의 경우 2^n개의 입력 중에서 k개의 입력이 동시에 참이 된다는 출력의 특성을 나타내는 것이다.

디코더는 아래와 같이 그림으로 표현되며, Active-low와 Active-high 종류가 있고, 각각의 진리표는 아래와 같다.



X_1	X_0	EN	Y_3	Y_2	Y_1	Y_0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0
×	×	0	0	0	0	0

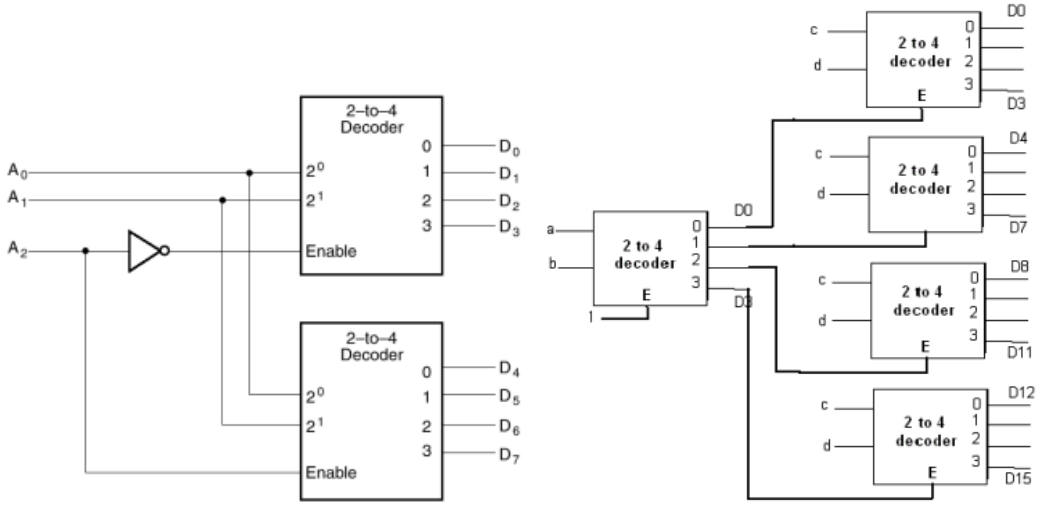
Active high decoder

X_1	X_0	EN	Y_3	Y_2	Y_1	Y_0
0	0	0	1	1	1	0
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	0	0	1	1	1
×	×	1	1	1	1	1

Active low decoder

2) 디코더 확장

디코더를 켜고 끄는 Enable 입력을 이용해 우리는 여러 개의 디코더를 연결하여 더 큰 입력을 처리하는 방법을 고안할 수 있고, 이를 디코더 확장이라고 한다. 수업시간에 다룬 바와 같이, 2-to-4 디코더 두 개를 연결하여 3-to-8 디코더의 기능을 하는 장치를 구성할 수 있는 것이 그 예이다. 아래는 예시 그림으로, 각각 2-to-4 => 3-to-8 확장과 2-to-4 => 4-to-16 확장이다.



3) 특수 목적 디코더

디코더와 같이 Multiple-output 기능의 회로이면서, 특수한 목적을 가진 것이 특수 목적 디코더이다. 본 lab3에서 다루는 특수 목적 디코더는 소수 판별기와 배수 검출기이다.

3-1) 소수 판별기

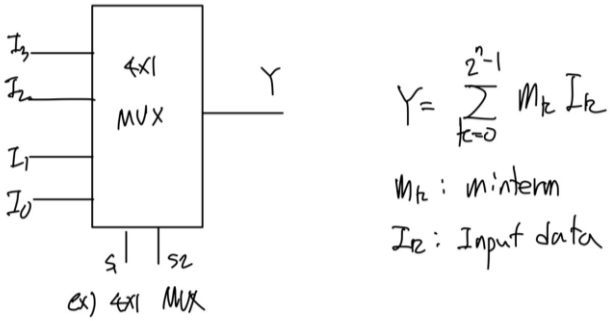
소수 판별기는 특정 이진 입력을 받아, 주어진 입력이 소수인 경우에 참을 출력하는 회로이다. 예를 들어 4-bit 소수 판별기에 1111을 입력하면 거짓을 출력하고, 0010을 입력하면 참을 출력하게 된다. 소수 판별기를 이용하면 쉽게 소수 여부를 알 수 있다.

3-2) 배수 검출기

배수 검출기는 특정 개수의 이진 입력을 받아 주어진 입력이 특정 수의 배수인 경우에는 각 출력이 참으로 출력되는 특수 목적 디코더이다. 따라서 배수 검출기의 각 출력은 단 하나의 출력도 참이 아닐 수도 있고, 여러 출력이 동시에 참일 경우도 존재할 수 있다. 본 lab3에서 다루는 것은 입력이 각각 2, 3, 5, 7, 11의 배수인지를 판별하는 배수 검출기를 구성한다.

4) 멀티플렉서 (Multiplexer)

멀티플렉서는 여러 신호 중 하나를 골라 출력하는 회로로, 선택 신호를 이용하여 원하는 신호를 고르는 방식으로 구성된다. 주로 n개의 선택 신호에 대해 2^n개의 입력 신호가 들어가서, 그 중 하나의 신호가 선택된다. 멀티플렉서의 출력을 수식으로 나타내면 아래와 같다.



4-1) 멀티플렉서를 사용한 함수의 표현

위에서 멀티플렉서의 출력을 수식으로 표현한 것에서 나아가, 멀티플렉서를 사용하여 함수의 표현을 할 수가 있다. 왜냐하면 minterm과 입력의 곱의 합, 즉 SOP 형태로 식이 나타나기 때문이다. 따라서, 예를 들어 n=2인 멀티플렉서의 경우, S_0와 S_1으로 4개의 입력 중에서 하나를 선택하는데, 이는 아래와 같이 나타낼 수가 있다. 이 식에 값을 대입하여 원하는 함수를 구현할 수가 있는 것이다.

$$O = \overline{S_0}\,\overline{S_1}\,I_0 + S_0\,\overline{S_1}\,I_1 + \overline{S_0}\,S_1\,I_2 + S_0\,S_1\,I_3$$

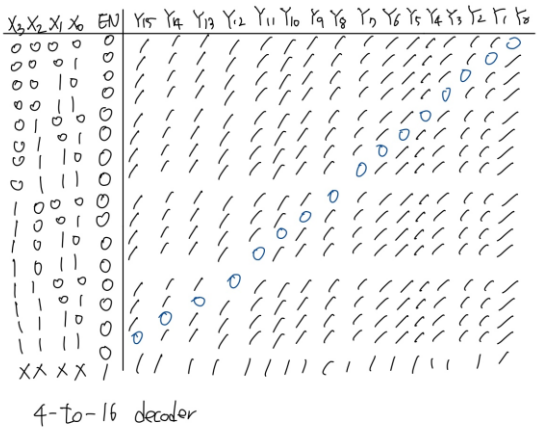
5) Majority / Minority function

홀수 개의 입력에 대해서 다수의 입력이 무엇인지, 소수의 입력이 무엇인지를 출력하는 함수이다. 즉, 예를 들어 5개의 입력이 11100으로 들어온다면, 다수의 입력을 출력하는 Majority function은 1을, 소수의 입력을 출력하는 Minority function은 0을 출력하게 된다.

3. 실험 준비

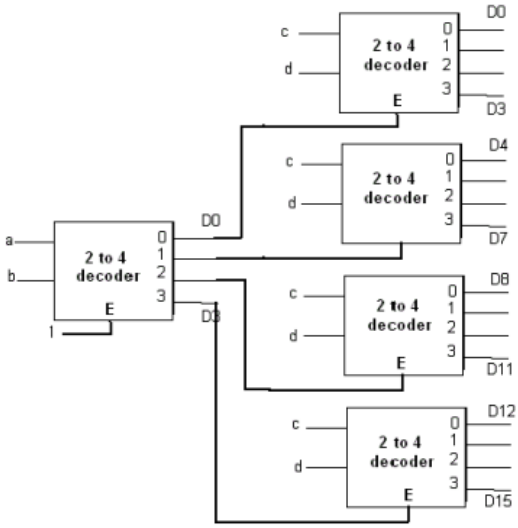
1) 4-to-16 Active-low 디코더

우선 첫 번째 실험은 수업시간에 배운 2-to-4 => 4-to-16 Active-high 디코더 확장과 반대로, 2-to-4 => 4-to-16 Active-low 디코더 확장을 구현해본다. 우선, 각 Active-low 디코더의 진리표는 아래와 같다.



X_1	X_0	EN	Y_3	Y_2	Y_1	Y_0
0	0	0	1	1	1	0
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	0	0	1	1	1
x	x	1	1	1	1	1

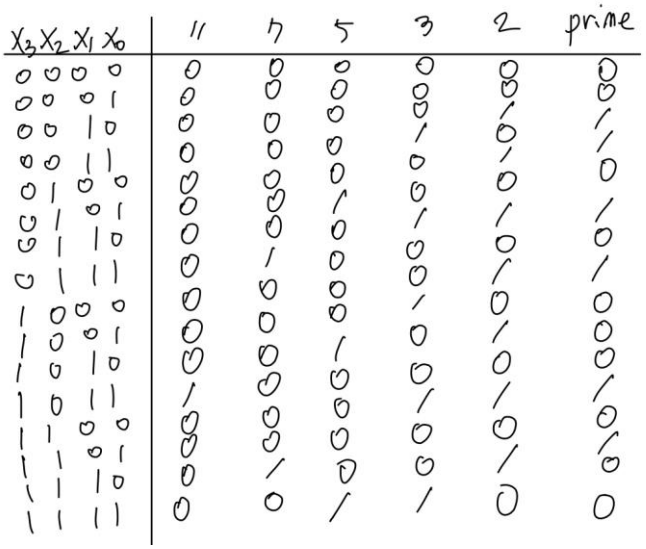
Active-high 와 마찬가지로, 아래와 같이 구성하면 2-to-4 디코더로 4-to-16 디코더를 구현할 수 있을 것으로 판단하였다. 이때, 왼쪽에 위치한 디코더는 X_3과 X_2와 EN을 받아 4개의 출력을 내보내고, 나머지 4개의 디코더가 각각 X_1과 X_0에 해당하는 입력을 처리하는 방식으로 구현한다.



실험 준비 과정에서, 우선 주어진 Verilog 코드에 2-to-4 디코더가 주어져 있으므로 해당 모듈을 활용하여 4-to-16 디코더를 만드는 것을 중점으로 생각하였다. 따라서 와이어 4개를 선언하여, 각 입력에 대해 디코더 모듈 5개를 구현하는 방식으로 코드를 구성한다면, 위와 같은 형태의 회로를 완성할 수 있을 것이라 생각한다.

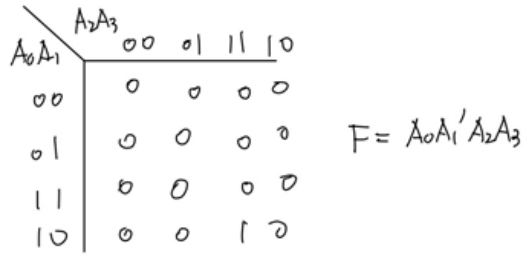
2) 4비트 소수 판별기 및 배수 검출기

두 번째 lab은 4비트 소수 판별기와 배수 검출기의 기능을 동시에 수행하는 특수 목적 디코더를 구성하는 것으로, 11, 7, 5, 3, 2의 배수를 판별하고, 소수를 판별하는 출력 각각에 대한 진리표는 아래와 같이 나타난다.



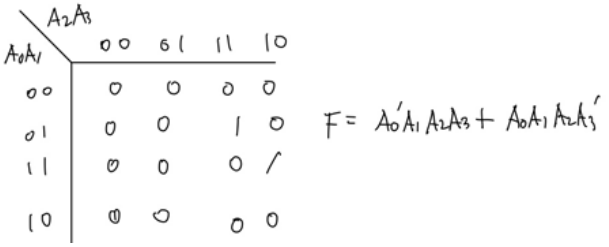
따라서 각각의 출력에 대해 카노 맵을 그려서 함수를 찾는 방식으로 진행하면 될 것이고, 각각의 함수의 경우 아래와 같이 구성할 수 있었다.

2-1) 11의 배수를 검출하는 함수



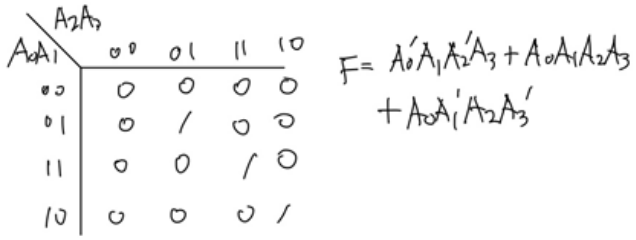
$F = A_0A_1'A_2A_3$

2-2) 7의 배수를 검출하는 함수



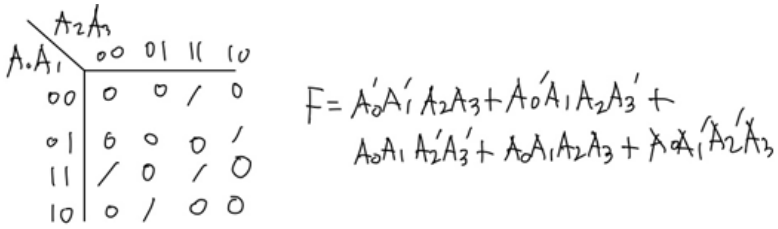
$F = A_0'A_1A_2A_3 + A_0A_1A_2A_3'$

2-3) 5의 배수를 검출하는 함수



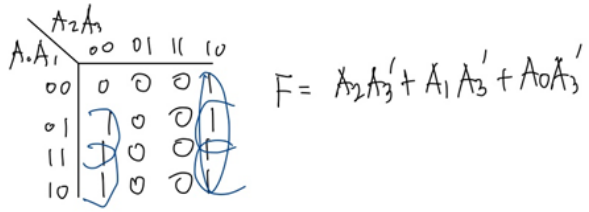
$F = A_0'A_1A_2'A_3 + A_0A_1A_2A_3 + A_0A_1'A_2A_3'$

2-4) 3의 배수를 검출하는 함수



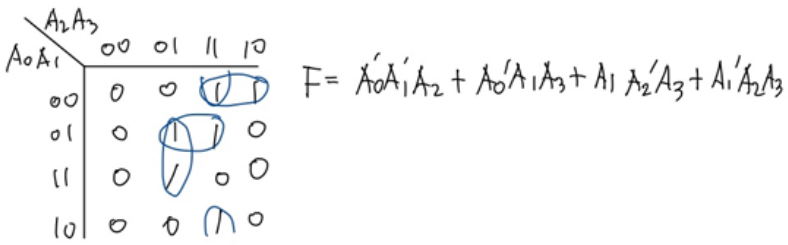
$F = A_0'A_1'A_2A_3 + A_0'A_1A_2A_3' + A_0A_1A_2'A_3' + A_0A_1A_2A_3 + A_0A_1'A_2'A_3$

2-5) 2의 배수를 검출하는 함수



$F = A_2A_3' + A_1A_3' + A_0A_3'$

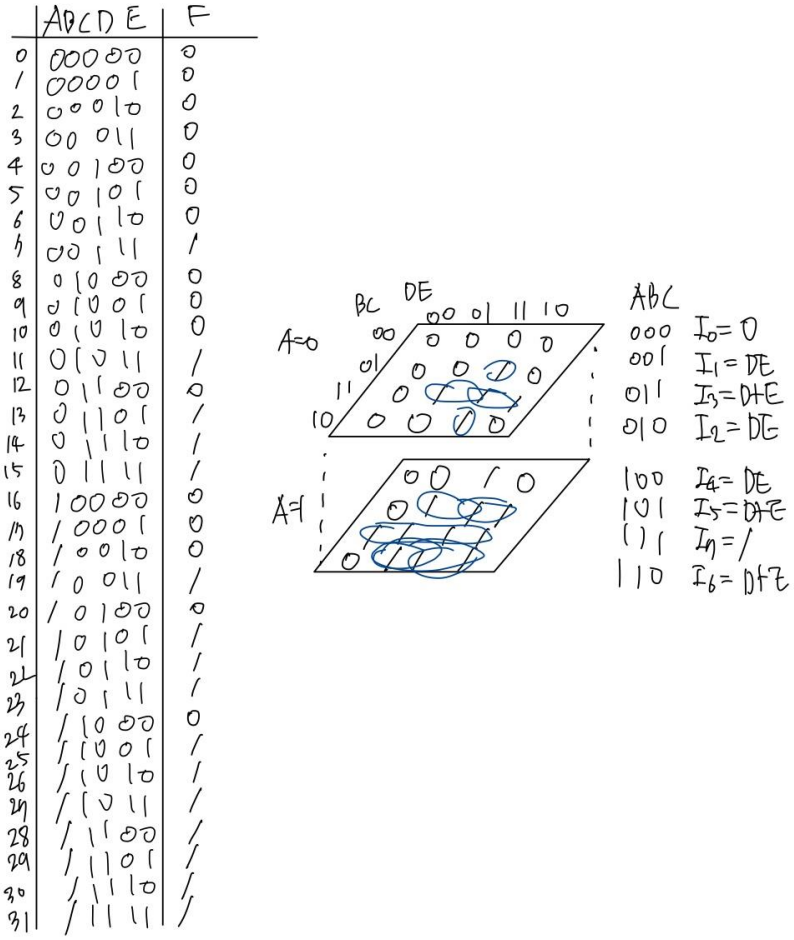
2-6) 소수를 검출하는 함수



위와 같이 구성된 함수들을 각각 verilog로 구현하면, 목표했던 특수 목적 디코더를 구성할 수 있을 것으로 생각한다. 또한, 코드를 구성하면서 ~, &, | 연산자를 활용하여, 와이어를 새로 선언하지 않고도 간단하게 함수를 구현하는 식으로 구성하고자 한다.

3) 5비트 Majority function

5-bit Majority function을 구성하기 위해서는 5비트, 즉 32:1 멀티플렉서를 사용하는 방법도 있겠지만, SOP 꼴을 사용하여 8:1 멀티플렉서로 표현하는 방식을 사용한다. 따라서, I_1과 I_0의 입력을 (아래 그림에서는 D, E에 해당) 각 게이트를 통과시켜 8:1 MUX에 연결하는 방식으로 구성하였다. 5비트 Majority function의 진리표 및 SOP 형태로 구성한 것은 아래와 같이 나타난다.

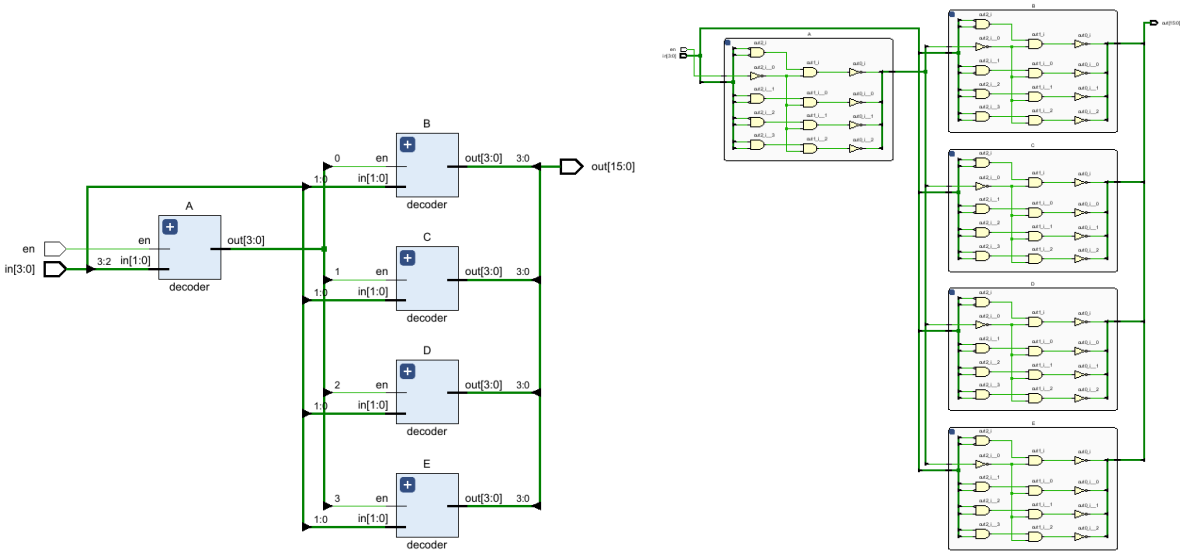


이제, 구성한 SOP 형태를 and, or 게이트를 이용하여 만들고, 이를 8:1 MUX의 input에 연결하는 것을 코드로 구현하면 될 것이다.

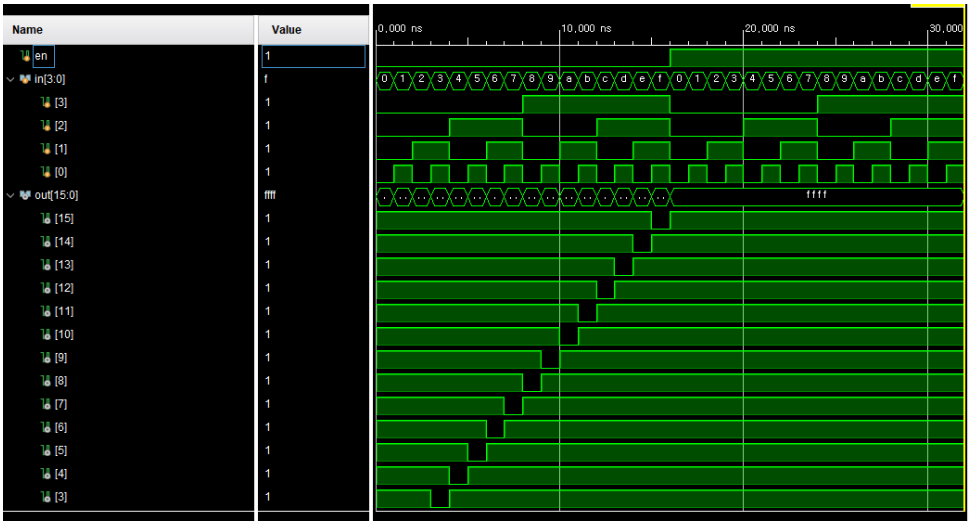
4. 결과

1) 4-to-16 Active-low 디코더

주어진 2-to-4 Active-low 디코더 모듈을 사용하여 실험 준비 과정에서 구상했던 대로, 4-to-16 Active-low 디코더를 구현할 수 있었다. 실험 준비 과정에서 구성했던 회로와 똑같은 회로를 Schematic 기능을 통해 확인할 수 있었다.

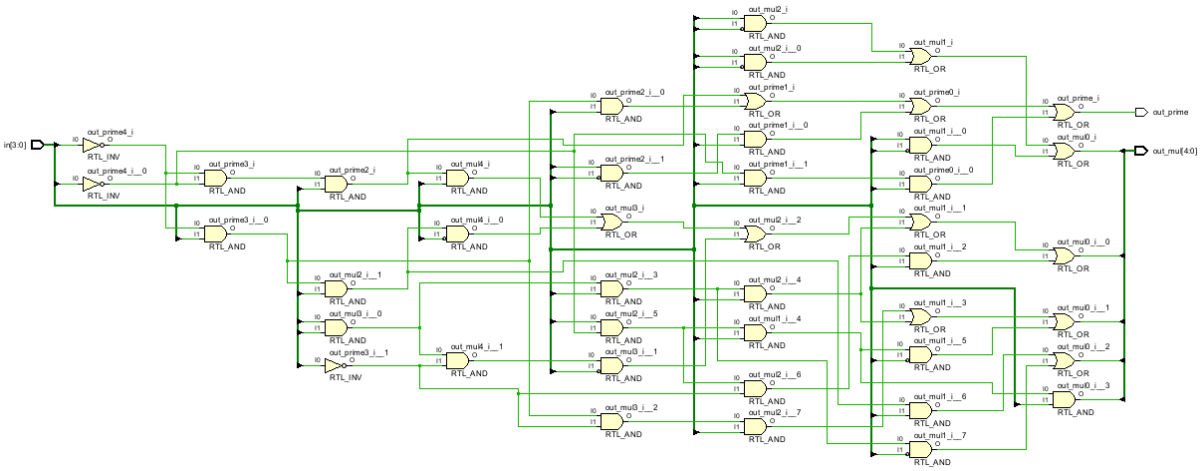


또한 testbench를 돌려본 결과, 4-to-16 Active-low 디코더의 기능을 수행하는 것을 알 수 있었다. Testbench 코드의 경우, 4비트의 입력이 0부터 15까지 1ns 단위로 끊어져 디코더에 들어오는데, 입력에 따라, 예를 들어 0001의 입력이면 1번 output이 0으로 출력되는 것을 확인할 수 있었다. 또한 EN 입력에 대해서도 EN이 0일 때만 디코더의 기능을 하는 것을 확인할 수 있었다. 이러한 것들은 실험 준비 단계에서 구상했던 대로 디코더가 제대로 완성되었다는 것을 의미한다.

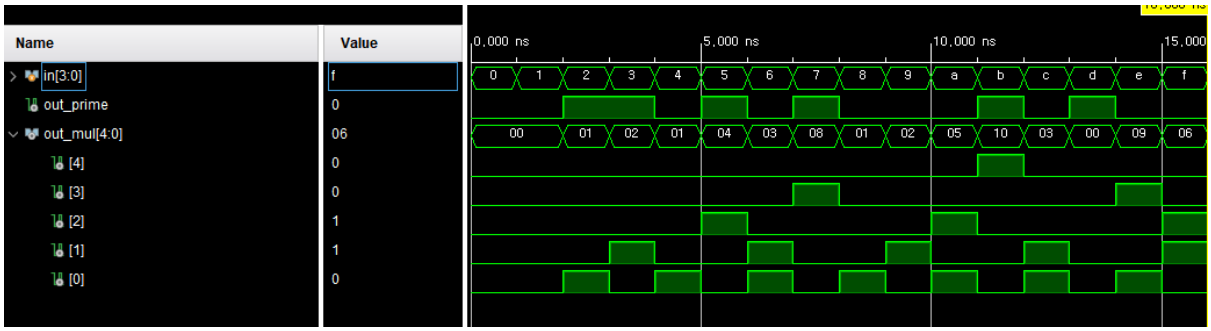


2) 4비트 소수 판별기 및 배수 검출기

이번 lab에서 구성하고자 하였던 특수 목적 디코더의 경우에도 실험 준비 단계에서 구상했던 것과 일치하는 결과를 얻을 수 있었다. 우선, Schematic 기능으로 얻은 회로의 모습은 아래와 같았다.



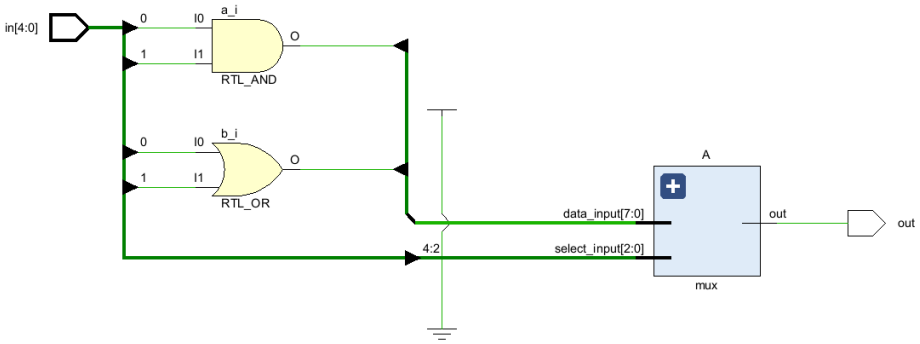
자세히 살펴보면, 구생했던 각 함수가 회로로 제대로 구현됐다는 것을 확인할 수 있었다. 또한, 이 특수 목적 디코더를 testbench로 돌려본 결과, 진리표에서 확인할 수 있는 값과 일치하는 것을 확인할 수 있었다. 이번 testbench의 코드 구성도 1ns마다 4비트 수가 1씩 증가되는 방식으로 구성되었고, 따라서 진리표와 똑같은 출력을 확인할 수 있었다. 이는 각 함수 및 전체 디코더의 구성이 제대로 이루어졌다는 것을 의미한다.

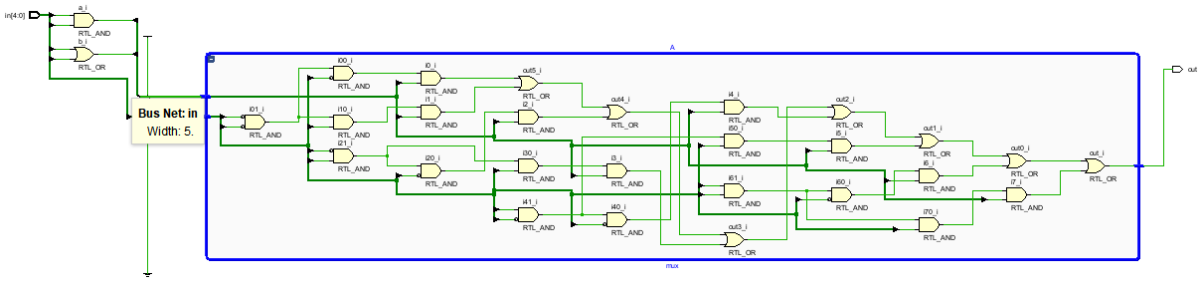


예를 들어 out_mul[4], 즉 11의 배수를 판별하는 출력의 경우 십진수 11, 즉 이진수 1011이 입력으로 들어왔을 때에만 1이 출력되는 것을 확인할 수 있다.

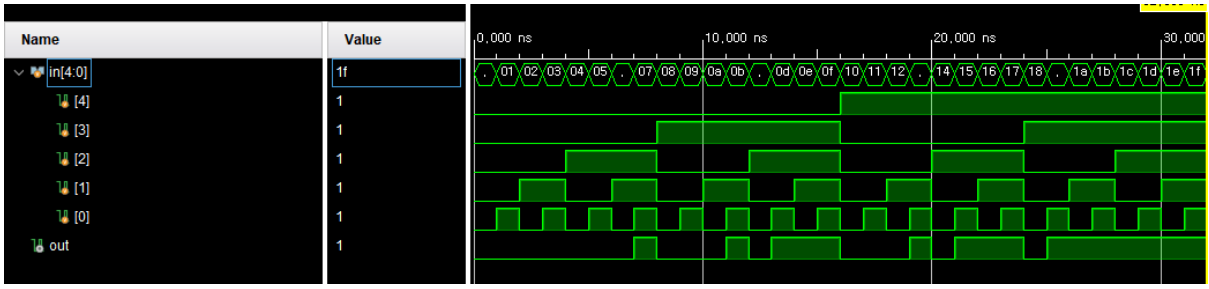
3) 5비트 Majority function

5비트 Majority function의 경우에도 실험 준비 단계에서 구성한 대로 결과가 나오는 것을 확인할 수 있었다. 우선 Schematic 기능으로 회로도를 확인한 결과, SOP 형태의 게이트를 통과한 입력 값 8개가 8:1 멀티플렉서로 들어가고, 3개의 입력은 선택 신호로 들어가 멀티플렉서를 제어하는 형태로 잘 나타났다. 회로도는 아래와 같다.





또한, 입력 중에 0과 1도 존재하기 때문에 해당 부분도 존재하는 것을 회로도에서 확인할 수 있었다. 그리고, 이를 testbench로 돌려본 결과도 구상했던 바와 일치하는 것을 확인할 수 있었다. 이번 testbench도 4비트 수가 0부터 31까지 1ns마다 1씩 증가하는 방식으로 구성된 것을 확인할 수 있었는데, 그러므로 testbench 실행 결과는 진리표와 같이 나타나야 했다. 실행 결과 진리표와 같은 형태로 나타나는 것을 확인할 수 있었고, 이는 제대로 회로가 구성되었다는 것을 의미한다.



5. 논의

본 lab3을 통해서 수업시간에 배웠던 디코더와 멀티플렉서에 대해 더 잘 이해할 수 있게 되었다. 또한, Verilog 문법에 대해 잘 몰라서 중간 중간에 코드 구성에 어려움을 겪었을 때, 여러 자료를 찾아보면서 해결하는 과정을 통해 Verilog 문법에 대해서도 이해도를 높일 수 있었다. 세부적으로는, Active-high가 아닌 Active-low 디코더도 동일한 방식으로 확장할 수 있다는 것을 확인할 수 있었고, 특수 목적 디코더를 구현하면서, 카노 맵 및 불 대수식의 단순화 내용을 복습함과 동시에 multiple-ouput 회로에 대해 이해할 수 있었다. 또한 Majority function을 구현하면서, Majority function에 대해 이해할 수 있었다. 추가로, lab을 진행하면서 한 가지 궁금증이 생겼는데, Majority function 여러 개를 이용하여 더 큰 입력을 받는 Majority function을 만들 수 있겠다는 생각이 들

었다. 기회가 된다면 그것도 한 번 만들어보면서 가능한지 확인해보고 싶다.