실험 3. 디코더와 멀티플렉서

무은재학부 20210661 오승준

1. 개요

이번 실험은 디코더와 멀티플렉서를 활용한 회로 구현을 진행하는 실험이다. Multiple-output 회로를 대표하는 디코더와, Multiput-input 회로를 대표하는 멀티플렉서의 기능을 이해하는 것을 이 실험의 주 목표로 한다. 이 실험의 세부 목표는 Active low 디코더의 확장을 직접 구현하는 것, 특수 목적의 디코더를 구현하는 것. 멀티플렉서의 데이터 선택 기능을 통한 Majority function 구현이다.

2. 이론적 배경

1) 디코더 (Decoder)

디코더는 n개의 이진 입력을 받아, 최대 2^n 개의 고유 출력을 내보내는 회로이다. 이때 n개의 이진 입력과 2^n개의 서로 다른 출력을 가지는 경우 각 출력이 minterm으로 나타나, minterm generator 로 부르기도 한다. 디코더에서는 n개의 이진 입력 외에도 en이라는 하나의 추가 입력을 받는데, 이는 디코더의 on/off를 위한 것이다.
디코더의 구성이 후술할 멀티플렉서를 뒤집은 것 과 유사하여, 디코더를 De-multiplexer로 부르기도 한다.
디코더의 입력-출력 데이터 수의 크기를 표현하기 위해, 디코더를 표현할 때, n to 2^n, 혹은 k of 2^n 이라는 표현을 사용한다. 각 표현의 의미는 아래와 같다.
n to 2^n : n개의 입력을 받아 2^n개의 출력을 내보냄.
k of 2^n : 2^n 개의 입력 중 k개의 입력이 동시에 참이 됨.

2) 디코더의 확장

각 디코더에 있는 en을 활용하면, 여러개의 디코더를 병렬적으로 연결하여 더 큰 규모의 디코더와 동일한 역할을 하도록 회로를 구성할 수 있다. 이를 디코더 확장(Decoder expansion) 이라 한다. 예를 들어, 2-to-4 디코더 2개를 이용하면 3-to-8 디코더를 , 2-to-4 디코더 5개를 이용하면 4-to-16 디코더를 구성할 수 있다.

3)특수 목적 디코더

말 그대로, 특수 목적에서 이용하기 위해 회로를 구성한 디코더이다. 이 실험에서는 소수 판별기 디코더와, 배수 검출기 디코더를 이용한다.
소수 판별기란, 주어진 입력이 소수일 때 참을 입력하는 회로이며, 배수 검출기는 주어진 입력이 지정된 수의 배수인 경우 참을 출력하는 회로이다.

4)멀티플렉서(Multiplexer)

멀티플렉서는 디코더와 반대로, 선택 신호에 따라 여러 입력 신호 중 하나를 골라서 출력을 하는 회로 이다. 멀티플렉서는 줄여서 MUX로 표현을 많이 하며, 2^nX1 MUX 현태로 보통 크기를 표현한다.

5) Majority/Minority Function

‘홀수 개의 입력’에 대하여, 다수, 소수를 차치하는 입력을 나타내는 함수이다. 다수인 값을 출력하면 Majority, 소수인 값을 출력하면 Minority Function이다. 예를 들어, 1이 5개, 0이 2개인 입력이 있다면, Majority function의 결과는 1, Minority function의 결과는 0이다.

6) MUX를 통한 함수 표현

멀티플렉서를 적절히 활용하면, 어떤 값을 대입하느냐에 따라 원하는 함수를 구현할 수 있다. 예를 들어 n=2인 규모의 MUX의 경우, S0, S1을 통하여 I0~I3 의 입력 중 하나를 선택한다. 이를 함수로 표현 시, O = S0’S1’I0 + S0S1’I1 + S0’S1I2 + S0S1I3로 표현할 수 있다. 각 변수에 적절한 값을 대입함으로써, 원하는 함수를 구현할 수 있다.

3. 실험 준비

1)2-to-4 Active-low enable, Active-low output, Active-high input 디코더로 4-to-16 Active-low enable, Active-low output, Active-high input 디코더를 구현한다.

- 하나의 2-to-4 Decoder의 output을 나머지 4개의 2-to-4 Decoder의 Enable에 연결하고, 상위 비트 2개를 첫 2-to-4 Decoder에, 하위 비트 2개를 4개의 2-to-4 Decoder에 동일하게 input함으로써 4-to-16 Decoder를 구현할 수 있다.

2)4비트 소수 판별기와 배수 검출기(2, 3, 5, 7, 11)의 진리표를 구하고, 식을 단순화한다.

input (ABCD)	prime	out[0]	out[1]	out[2]	out[3]	out[4]
0000	0	0	0	0	0	0
0001	0	0	0	0	0	0
0010	1	1	0	0	0	0
0011	1	0	1	0	0	0
0100	0	1	0	0	0	0
0101	1	0	0	1	0	0
0110	0	1	1	0	0	0
0111	1	0	0	0	1	0
1000	0	1	0	0	0	0
1001	0	0	1	0	0	0
1010	0	1	0	1	0	0
1011	1	0	0	0	0	1
1100	0	1	1	0	0	0
1101	1	0	0	0	0	0
1110	0	1	0	0	1	0
1111	0	0	1	1	0	0

f(prime) = A’B’C + A’CD + B’CD + BC’D

f(out[0]) = CD’ + BD’ + AD’

f(out[1]) = A’B’CD + A’BCD’ + ABC’D’ + ABCD

f(out[2]) = A'BC'D + AB'CD' + ABCD

f(out[3]) = A'BCD + ABCD’

f(out[4]) = AB’CD

3) Majority function의 진리표를 구하고, 식을 SOP꼴로 바꿔 8:1 멀티플렉서로 표현한다.

Input (A,B,C,D,E)	Output	Input (A,B,C,D,E)	Output
00000	0	10000	0
00001	0	10001	0
00010	0	10010	0
00011	0	10011	1
00100	0	10100	0
00101	0	10101	1
00110	0	10110	1
00111	1	10111	1
01000	0	11000	0
01001	0	11001	1
01010	0	11010	1
01011	1	11011	1
01100	0	11100	1
01101	1	11101	1
01110	1	11110	1
01111	1	11111	1

y = CDE + BDE + BCE + BCD + ADE + ACE + ACD + ABE + ABD + ABC

D, E를 통하여 8:1 MUX의 input을 표현해낼 시,

I0 = 0

I1 = DE

I2 = DE

I3 = D + E

I4 = DE

I5 = D + E

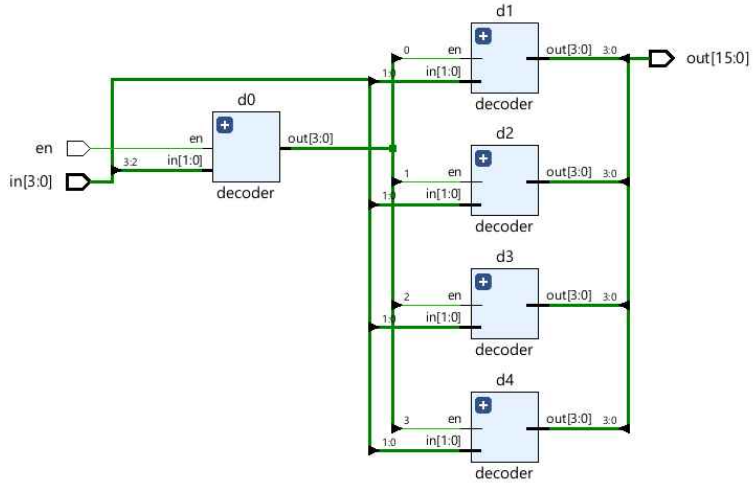
I6 = D + E

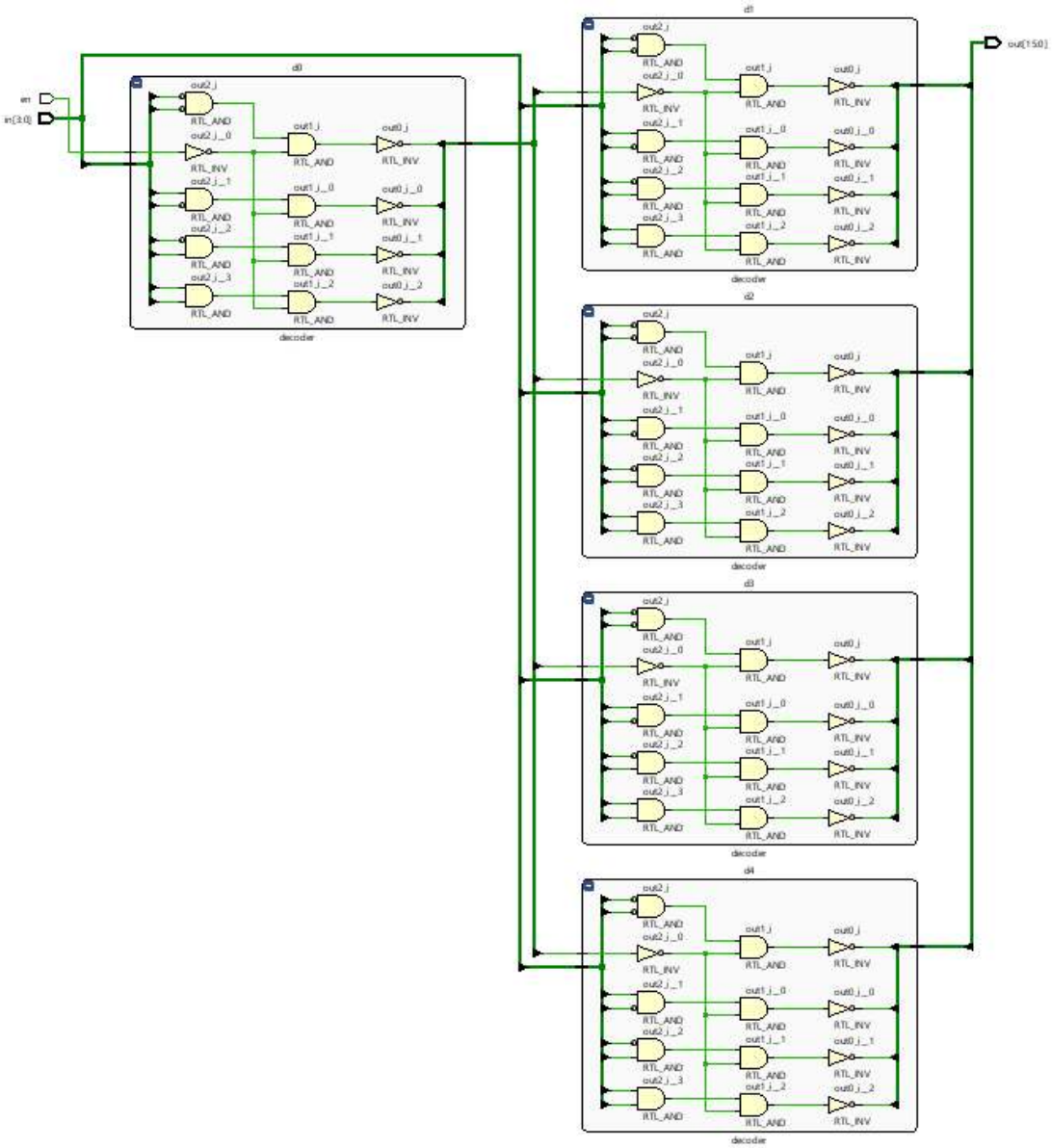
I7 = 1

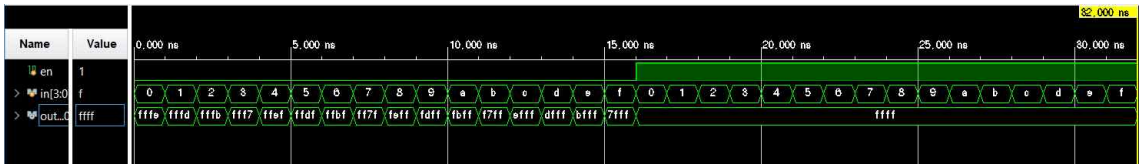
4. 결과

1) 4-to-16 Active-low 디코더

실험 결과 회로도와 시뮬레이션 결과는 아래와 같다.


d1
d0
d2
d3
d4
en
in[3:0]
out[15:0]
out[3:0]
in[1:0]
decoder

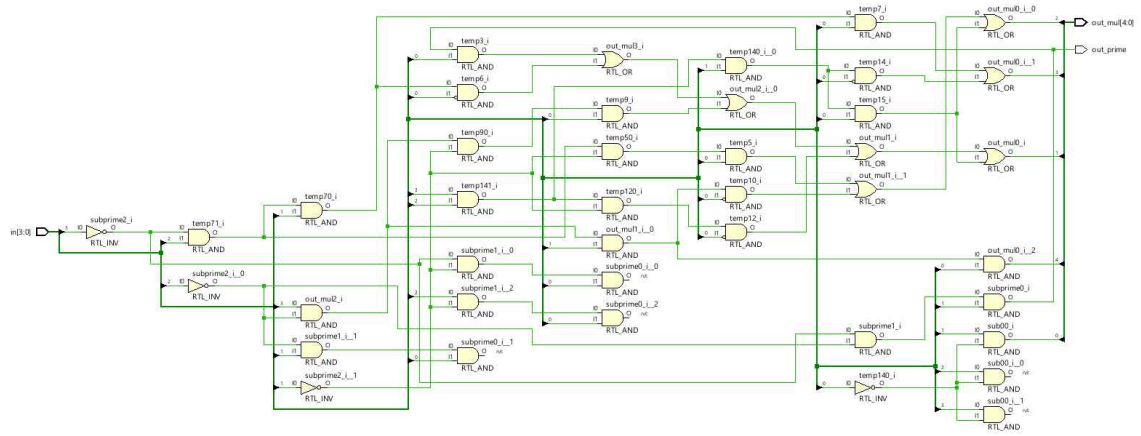

d0
d1
d2
d3
d4
en
in[3:0]
out[15:0]
decoder

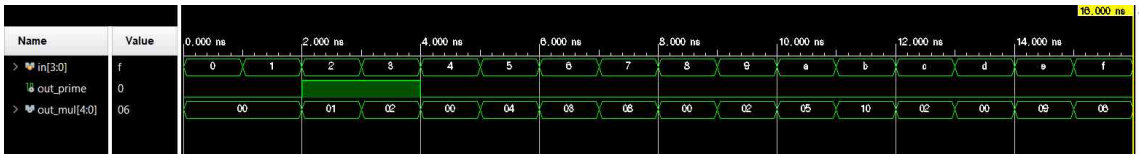


시뮬레이션 상 오류가 없음으로부터 구현이 올바름을 확인할 수 있다.

2)4비트 소수 판별기 및 배수 검출기

실험 결과 회로도와 시뮬레이션 결과는 아래와 같다.

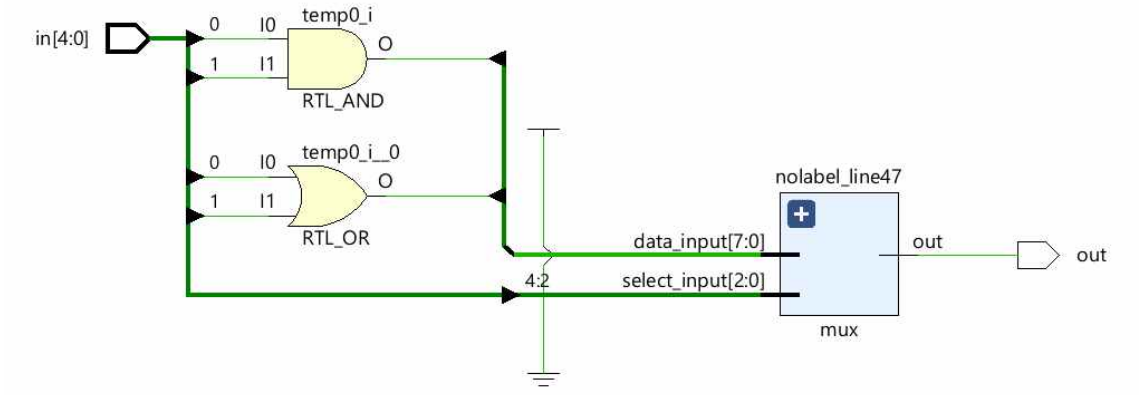


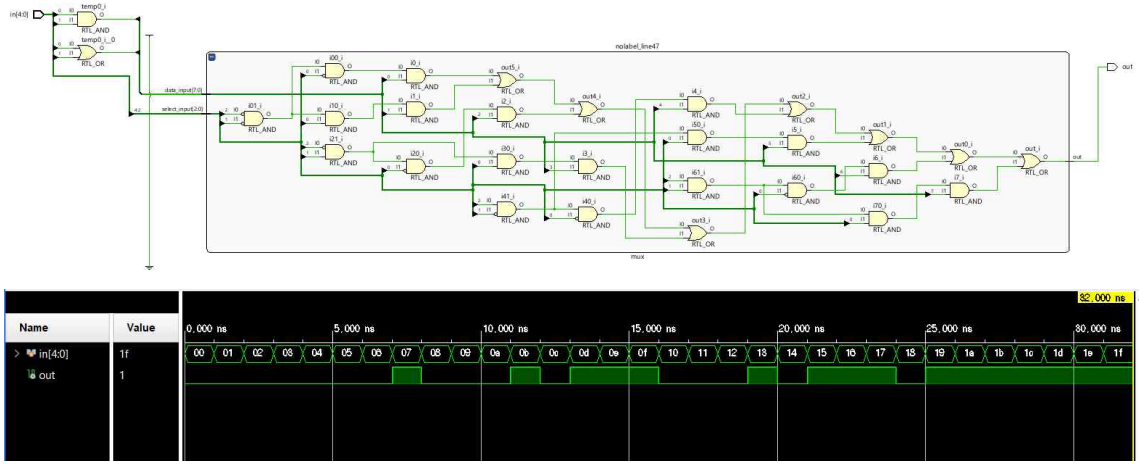


시뮬레이션 상 결과가 실험 준비 중 작성한 TT와 동일한 것으로부터 구현이 올바르게 됨을 확인할 수 있다.

3) 5비트 Majority function

실험 결과와 시뮬레이션 결과는 아래와 같다.





시뮬레이션 상 결과가 실험 준비 중 작성한 TT와 동일한 것으로부터 구현이 올바르게 됨을 확인할 수 있다.

5. 논의

- 이번 실험을 진행하며 Decoder와 MUX를 통한 회로 구성을 경험해볼 수 있었다. 회로를 직접 구현해보며 왜 교수님께서 Decoder를 설명하실 때 Active-low 형태의 Decoder가 주로 쓰인다고 말씀해주셨는지 이해할 수 있었다.

코드 작성 중에서 dimension mismatch라는 에러를 만났다. 처음에는 배열을 잘못 써 나타난 오류라고 생각하여 한참을 헤맸으나, , 대신 .를 입력한 곳이 한 곳 있어 이로 인한 에러였다. 그러나 이 오입력된 곳에는 코드 아래에 붉게 에러 표시가 생겨나지 않았었다. 코드를 작성함에 있어 신중하게 작성을 해야 하고 작성 과정 중에서 오류를 최대한 줄여야함을 느낄 수 있었다. 에러 코드에서 파일명 옆에 적히는 숫자가 에러 코드가 발생한 라인을 나타낸다는 사실도 알게 되었다.