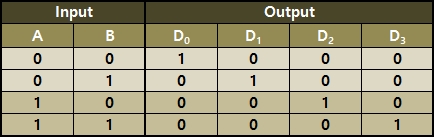
9주차 결과보고서

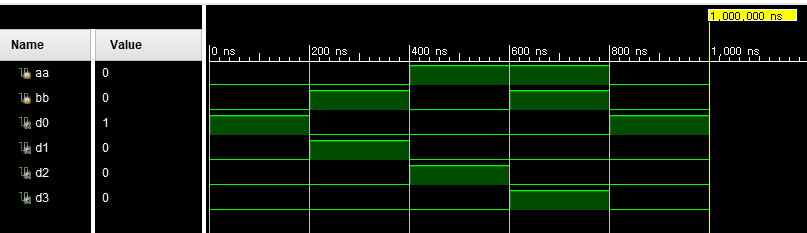
전공: 기계공학과 학년: 3학년 학번: 20191820 이름: 김형준

**1.**

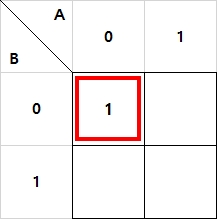
- 2 to 4 active high decoder



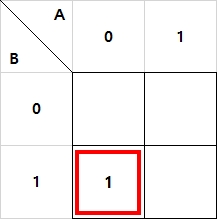
[2 to 4 active high decoder의 진리표]



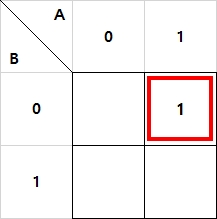
[2 to 4 active high decoder의 Simulation 결과]



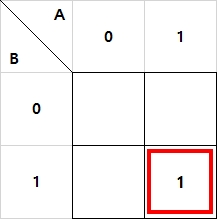
[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]

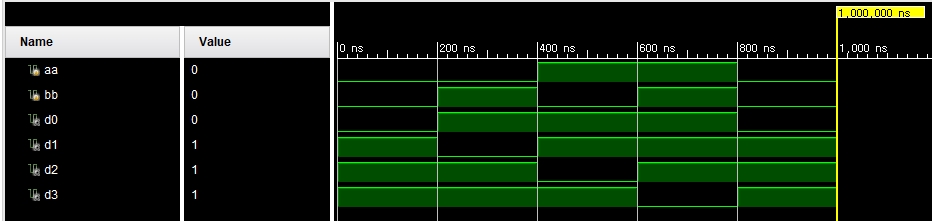
2 to 4 decoder는 2 input, 4 output을 가지는 decoder로, 입력신호의 조합에 따라 1개의 output만이 선택되어 활성화 된다. 2 to 4 active high decoder는 AND 게이트를 사용해 만든 2 to 4 decoder로, output이 1일때 해당 Output이 활성화 된다.

위 진리표에서 입력신호 A, B의 조합(개)에 따라 각각의 Output()이 선택되는 것을 확인 할 수 있다. 또한, 선택된 Output만이 1을 갖고, 선택되지 않은 Output들은 모두 0을 가지는 것을 확인 할 수 있다.

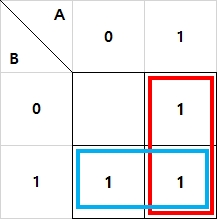
- 2 to 4 active low decoder



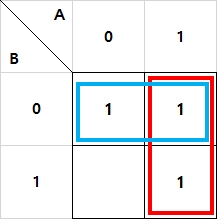
[2 to 4 active low decoder의 진리표]



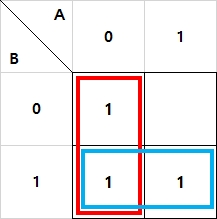
[2 to 4 active low decoder의 Simulation 결과]



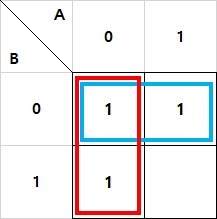
[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]

2 to 4 active low decoder는 NAND 게이트를 사용해 만든 2 to 4 decoder로, output이 0일때 해당 Output이 활성화 된다.

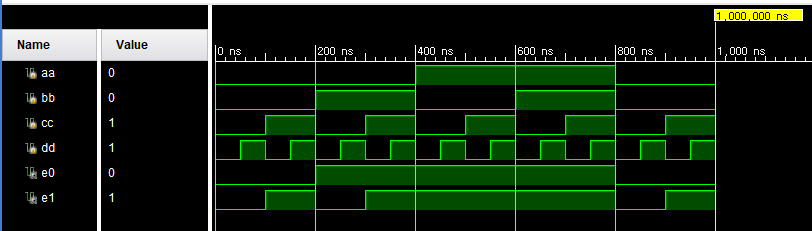
위 진리표에서 입력신호 A, B의 조합(개)에 따라 각각의 Output()이 선택되는 것을 확인 할 수 있다. 또한, 선택된 Output만이 0을 갖고, 선택되지 않은 Output들은 모두 1을 가지는 것을 확인 할 수 있다.

2 to 4 active high decoder와의 차이는 A와 B를 엮을 때 AND 게이트를 사용하는지, 또는 NAND 게이트를 사용하는지의 여부밖에 없다.

**2.**

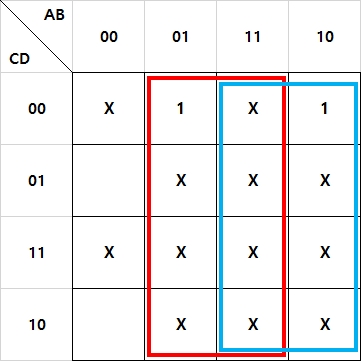


[4 to 2 encoder의 진리표]

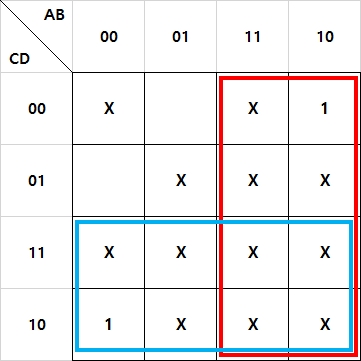


[4 to 2 encoder의 Simulation 결과]

※ 위 진리표에서 invalid한 input의 조합(1의 개수가 1개가 아님)은 모두 생략되었음



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]

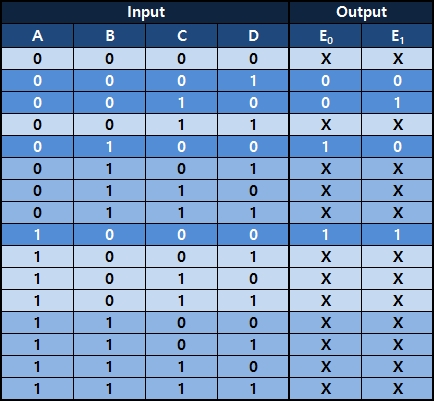
4 to 2 encoder는 4 input, 2 output을 가지는 encoder로, 2 to 4 decoder와 반대되는 동작을 한다. 위의 진리표를 보면 4 to 2 encoder의 input은 2 to 4 active high decoder의 output과 같고, encoder의 output은 decoder의 input과 같은 것을 확인 할 수 있다.

4 to 2 encoder는 각각의 유효한 input(A, B, C, D)의 조합에 대해 서로 다른 Output()의 조합을 출력한다. 이때, 유효한 input은 모든 input들 중 1의 개수가 1개인 input 조합을 말하며, 나머지 input 조합은 모두 유효하지 않은 것으로 간주되어 출력값이 don’t care로 처리된다.

**3.**

앞서 간략하게 설명했듯이 4 to 2 encoder에 들어오는 input들 중 1의 개수가 1개인 input만이 유효한 입력형태이고, 나머지 모든 input의 조합은 (1의 개수가 0개이거나 2개 이상) 유효하지 않은 입력형태로 처리되어 해당 경우에 대한 출력을 고려하지 않는다.

(don’t care로 처리됨)



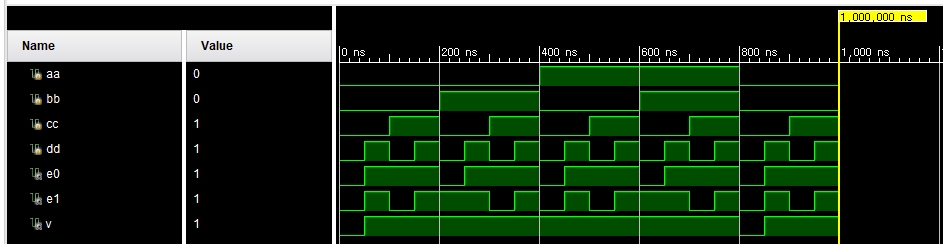
[4 to 2 encoder의 진리표 (invalid input을 생략하지 않음)]

위의 진리표에서 4 to 2 encoder에서 유효하지 않은 입력의 경우 모두 don’t care가 출력되는 것을 확인 할 수 있다.

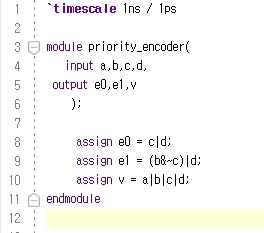
**4.**

****

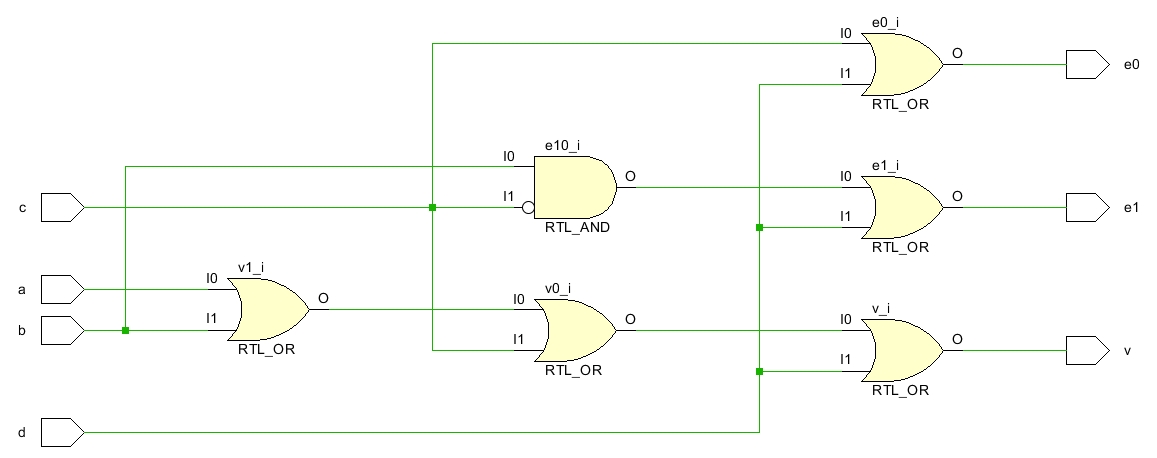
[4 to 2 priority encoder의 진리표]

****

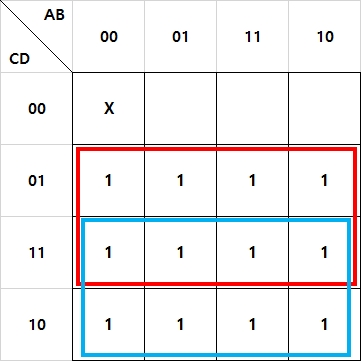
[4 to 2 priority encoder의 Simulation 결과]



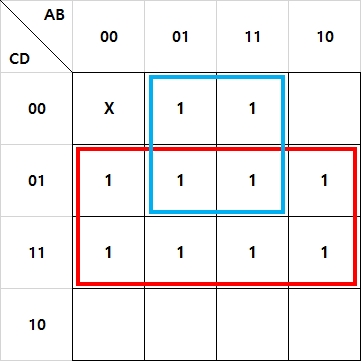
[4 to 2 priority encoder의 Verilog Code]



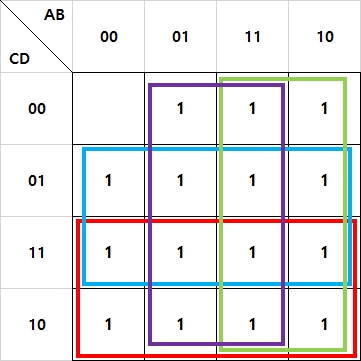
[4 to 2 priority encoder의 Schematic]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]

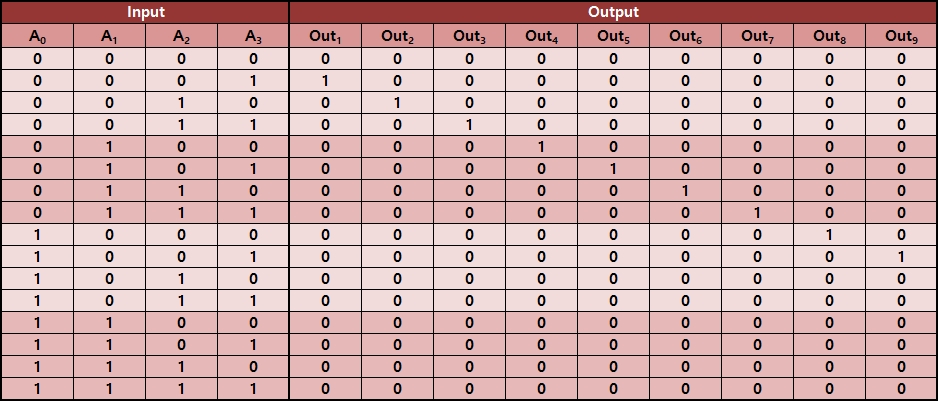
priority encoder는 input간의 우선순위(Priority)를 추가해 유효하지 않은 입력이 들어와도 정해진 우선순위에 따라 출력을 결정하는 encoder를 의미한다.

priority encoder에 input에 1이 여러 개 인 경우, 가장 높은 우선순위를 가진 1만을 사용하여 output을 결정하고, 나머지 input들을 모두 무시한다.

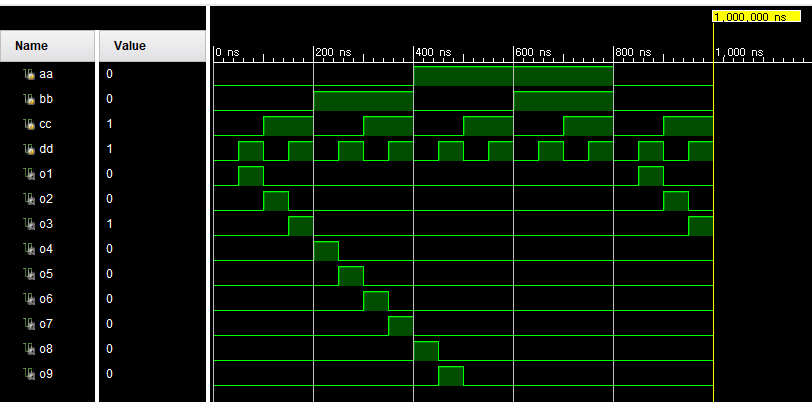
출력값 중 V는 어떤 입력도 활성화 되지 않았다는 것을 표시하며, input이 모두 0인 경우 V의 값을 0으로 설정하고, 나머지 경우들은 모두 V의 값을 1로 설정한다.

위는 오름차순으로 input(A, B, C, D)에 우선순위(A<B<C<D)를 배정한 4 to 2 priority encoder이다. 위 진리표에서 우선순위가 높은 input의 값이 1인 경우 해당 input의 값만을 사용하여 output을 결정하고, 우선순위가 낮은 input들의 값은 모두 무시되는 것을 확인 할 수 있다. (D>C>B>A 순으로 우선순위가 높은 것을 확인할 수 있음)

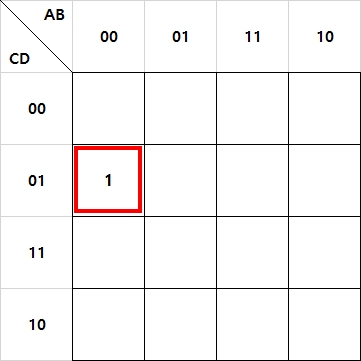
**5.**



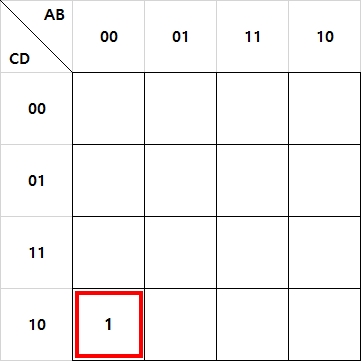
[BCD to Decimal decoder의 진리표 (각각 A, B, C, D에 대응됨)]



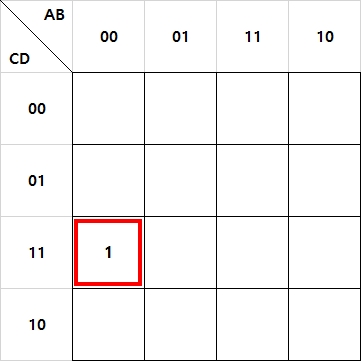
[BCD to Decimal decoder의 Simulation 결과]



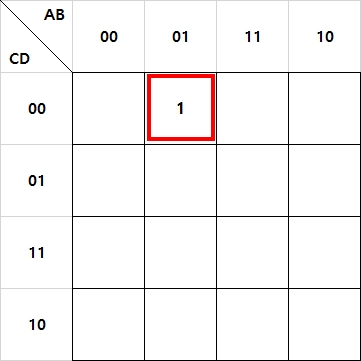
[Output의 의 논리 함수 카르노맵]



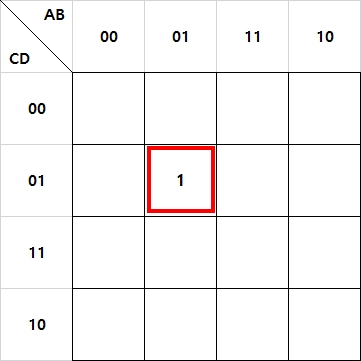
[Output의 의 논리 함수 카르노맵]



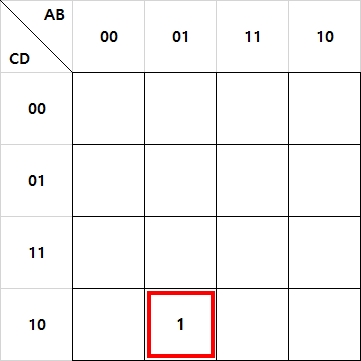
[Output의 의 논리 함수 카르노맵]



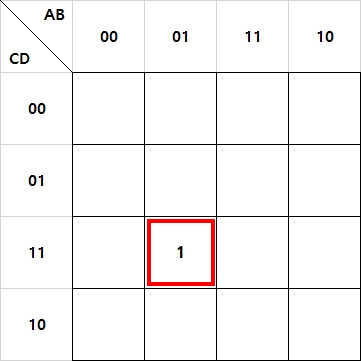
[Output의 의 논리 함수 카르노맵]



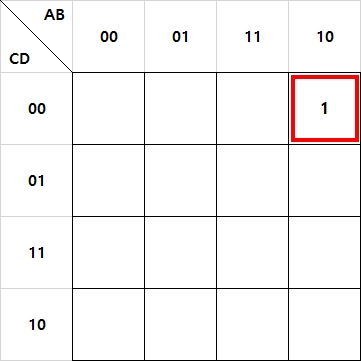
[Output의 의 논리 함수 카르노맵]



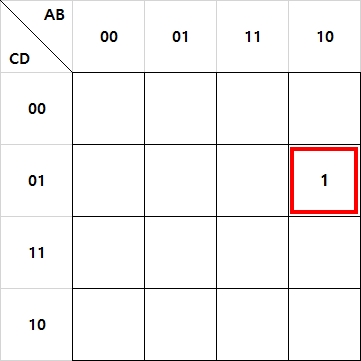
[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]



[Output의 의 논리 함수 카르노맵]

BCD to decimal decoder는 4개의 input(A, B, C, D)으로 0~9까지의 8421 BCD 코드 입력에 대해서 대응되는 10진수 output(O1~O9)을 출력하는 decoder이다.

1~9까지의 입력값에 대해서는 Output중 해당되는 1개의 Output만이 1의 값을 가지게 되며, 나머지 Output들은 모두 0이 된다.

모든 input이 0인 경우 모든 Output의 값이 0이 되고, 나머지 10~15의 입력에 대해서는 유효하지 않은 입력으로 간주하여 모든 Output의 값을 0으로 처리하였다.

**6.**

Encoder는 입력받은 데이터를 다른 형태로 변환하면서 크기를 줄이는 역할(부호화)을 하고, Decoder는 Encoder에 의해 변환된 데이터를 다시 원래대로 되돌리는 역할(복호화) 한다.

Encoder와 Decoder가 실생활에 적용되는 사례들은 동영상 파일 압축, 통신보안, 문자 집합 등이 있다.

동영상 파일 압축에서 인코더는 영상 파일의 크기를 부호화를 통해 압축시키고, 압축된 영상을 디코더를 통해 복호화하여 원래대로 복구한다.

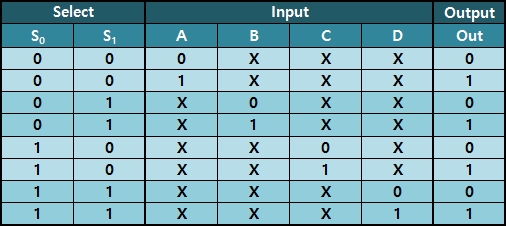
통신 보안에서도 보안을 위해 전송할 데이터를 인코더를 사용해 암호화하고, 수신된 데이터를 디코더를 통해 복호화하여 해독한다.

또한, 문자 집합(Character Set)에도 사용되는데, 각 문자를 인코더를 사용해 컴퓨터에 저장하거나 통신에 사용할 목적으로 부호화하고, 디코더를 사용해 변환된 문자를 다시 부호화 되기 전의 문자로 복호화 한다.

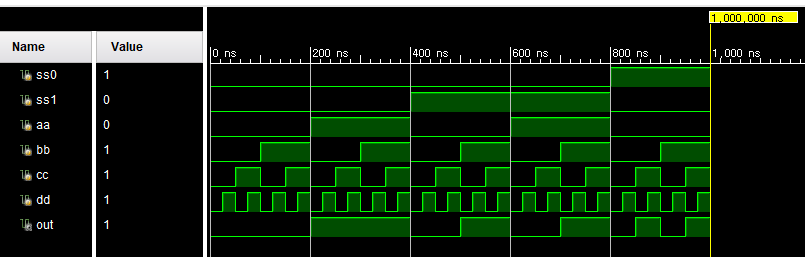
이 뿐만이 아니라 디코더를 사용해 논리함수를 구현하는 데에도 사용할 수 있다.

Active high decoder에서 활성화된 1 출력이 minterm에 해당하며, 구현할 논리함수의 minterm에 해당하는 1 출력들을 하나의 OR 게이트로 연결하면 된다.

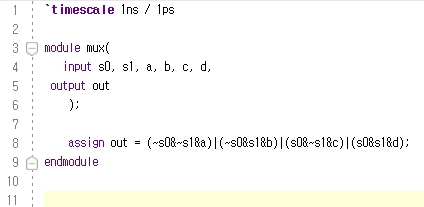
**7.**



[4 to 1 line MUX의 진리표]



[4 to 1 line MUX의 Simulation 결과]



[4 to 1 line MUX의 Verilog Code]

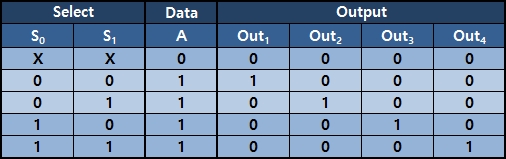
4 to 1 Mux는 4개의 input(A, B, C, D)중 하나를 2개의 선택 입력()의 값에 따라 출력에 연결하는 회로이다.

위 진리표에서 선택입력의 값이 각각 00, 01, 10, 11이면 출력으로 각각 A, B, C, D가 선택되는 것을 확인 할 수 있다.

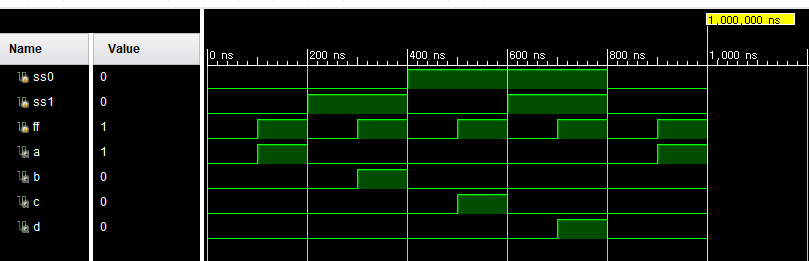
또한, 선택 입력에 의해 선택되지 않은 입력값은 출력값에 영향을 미치지 않으므로 모두 don’t care로 처리된다.

**8.**

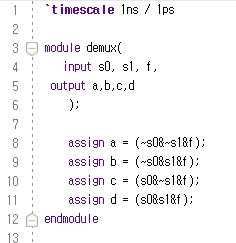
- 1 to 4 line deMUX



[1 to 4 line deMUX의 진리표]



[1 to 4 line deMUX의 Simulation 결과]



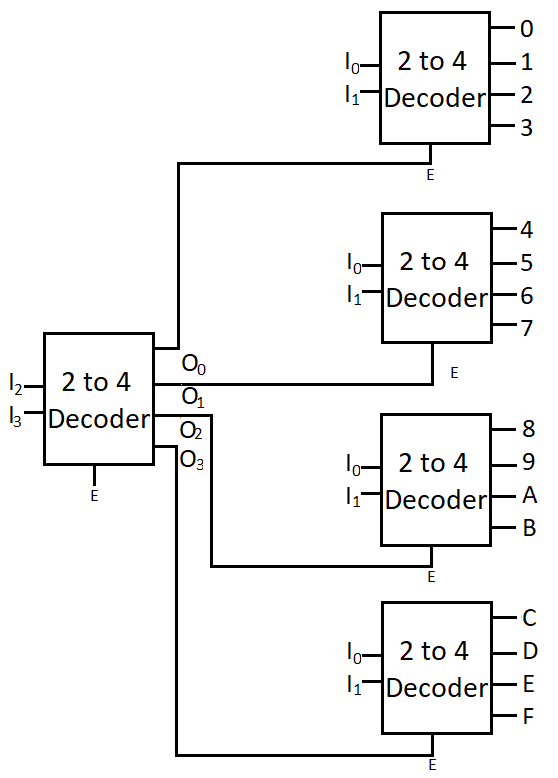
[1 to 4 line deMUX의 Verilog Code]

1 to 4 line deMUX는 1개의 input(A)을 2개의 선택 입력()의 값에 따라 Output

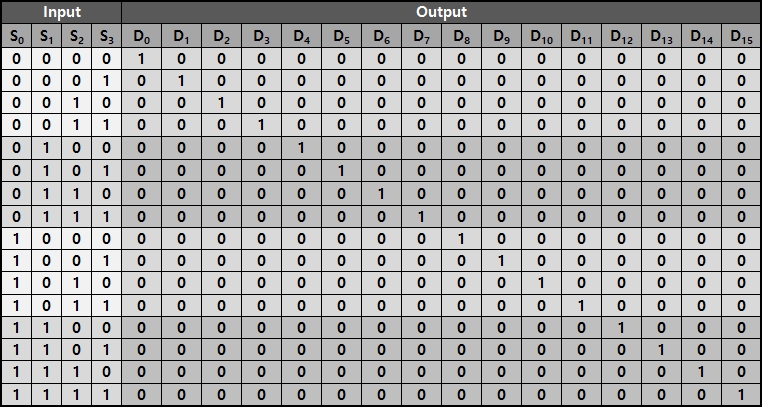
()중 하나를 선택해 연결하는 회로이다.

Input의 값이 0일때는 선택 입력의 값과 관계 없이 모든 Output이 0이 되고, Input의 값이 1일때는 선택 입력에 의해 선택된 Output만이 1을 가지고 나머지 Output들은 모두 0을 가지게 된다.

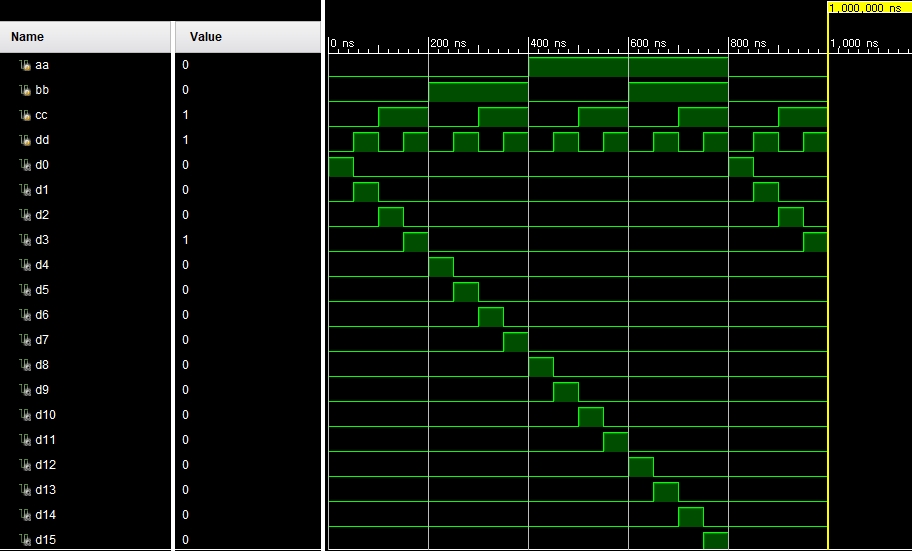
이때, 위의 1 to 4 line deMUX의 Input(Data)를 decoder의 enable로, deMUX의 선택 입력들은 decoder의 input으로 간주하면, enable 신호가 있는 2 to 4 decoder처럼 사용할 수 있다. 아래와 같이 5개의 2 to 4 decoder를 사용하면 4 to 16 decoder를 설계할 수 있다.

****

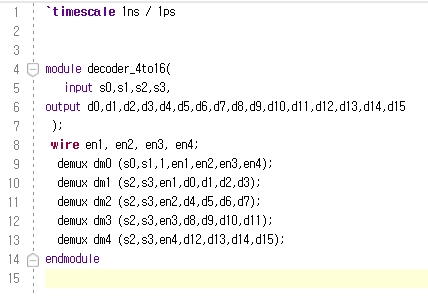
[4 to 16 decoder의 회로도]



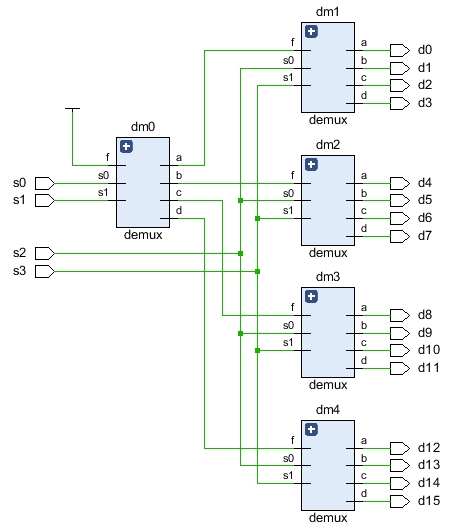
[4 to 16 decoder의 진리표]



[4 to 16 decoder의 Simulation 결과]



[4 to 16 decoder의 Verilog Code]



[4 to 16 decoder의 Schematic]

위의 코드에서 4 to 16 decoder의 입력은 demux의 선택선 으로 하였으며,

출력값은 에 해당한다. 코드에서 demux 모듈은 위의 1 to 4 line deMUX의 Verilog Code를 그대로 사용하였다.

위 진리표와 시뮬레이션에서 1 to 4 deMUX로 4 to 16 decoder를 만들 수 있고, 각각의 선택선 의 값의 조합에 따라 출력값() 중 선택된 하나의 출력값만 1이 되고, 나머지는 0이 되는 것을 확인할 수 있다.

**9.**

1~8까지 각각의 시뮬레이션 결과와 진리표, 카르노맵을 비교했을 때 차이가 없으므로 의도한 대로 코드가 작성되었음을 확인 할 수 있다.

이번 실험을 통해 (active High/Low) Decoder, (Priority) Encoder, MUX, deMUX의 구조와 작동방식에 대해서 알 수 있었다.

2 to 4 decoder는 입력신호(A, B)의 조합에 따라 4개의 output 중 1개의 output만이 선택되어 활성화 되는 것을 알게 되었고, decoder를 구현할 때 AND 게이트를 사용하면 active high, NAND 게이트를 사용하면 active low가 되는 것을 알게 되었다.

4 to 2 encoder는 2 to 4 decoder의 반대 기능을 한다는 것을 알게 되었고,

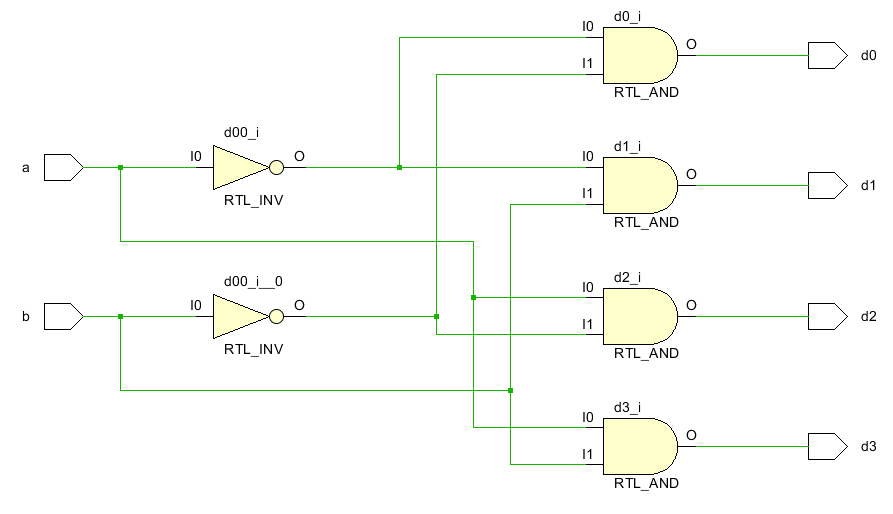
4개의 input 중 1의 개수가 1개인 유효한 입력이 들어왔을 때 서로 다른 Output(A, B)값의 조합을 출력하고, 유효하지 않은 입력이 들어오면 모든 Output의 값이 don’t care가 되는 것을 알게 되었다.

4 to 1 line MUX는 4개의 input 중 선택선의 조합에 따라 input 중 하나를 선택하여 Output으로 출력하는 것을 알게 되었고, 1 to 4 line deMUX는 4 to 1 line MUX의 반대 기능을 한다는 것을 알게 되었다.

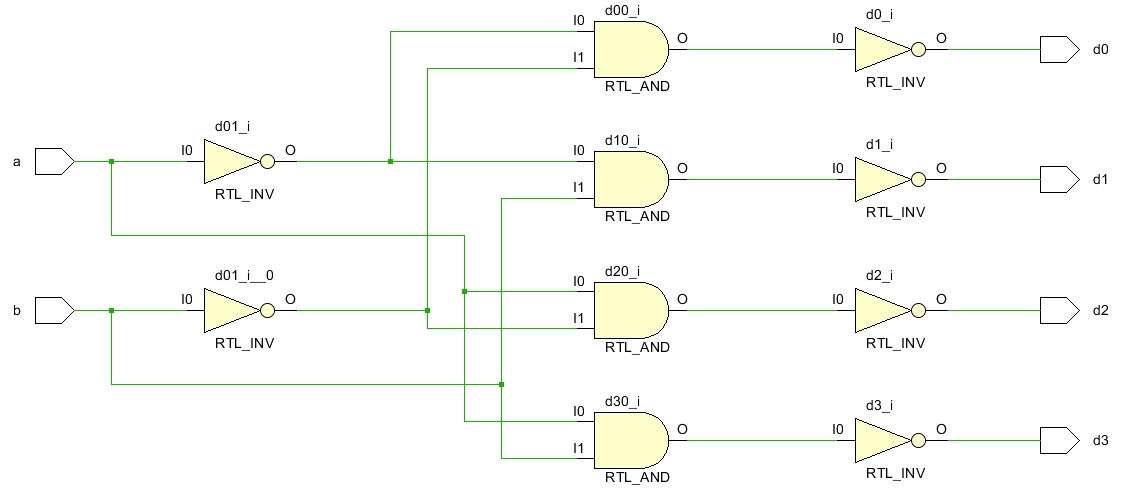
또한, deMUX와 enable 신호가 있는 decoder의 구조가 같아 deMUX의 input을 decoder의 enable 신호로, deMUX의 선택 입력을 decoder의 input으로 간주하면 deMUX를 decoder처럼 사용할 수 있다는 것을 알게 되었다.

마지막으로, 2 to 4 encoder를 조합해 4 to 16 encoder를 만들 수 있음을 알게 되었다.

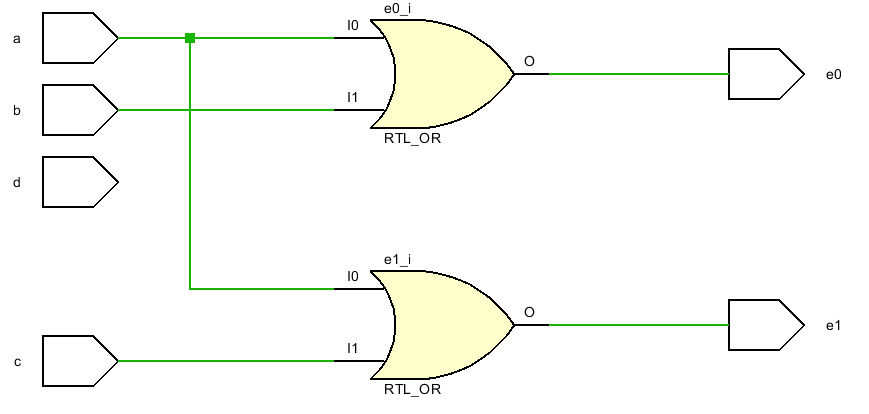
아래의 사진들은 1~8까지 각각의 Schematic이다.



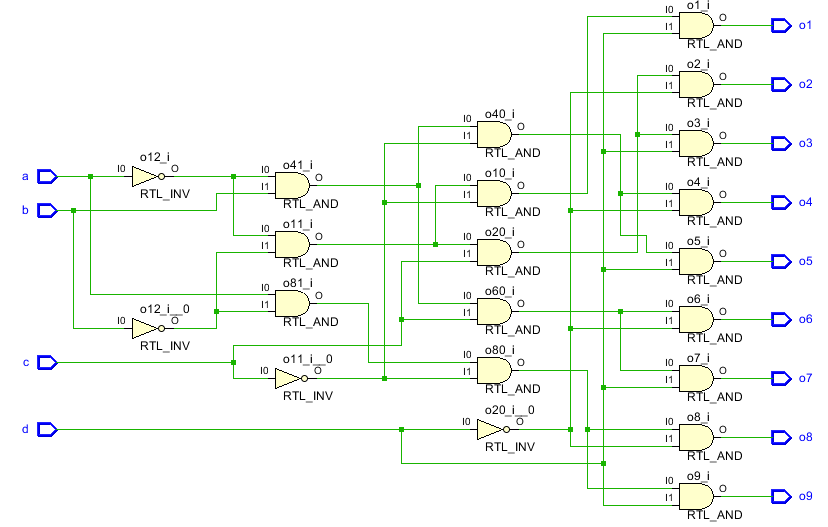
[2 to 4 active high decoder의 Schematic (AND gate 사용)]



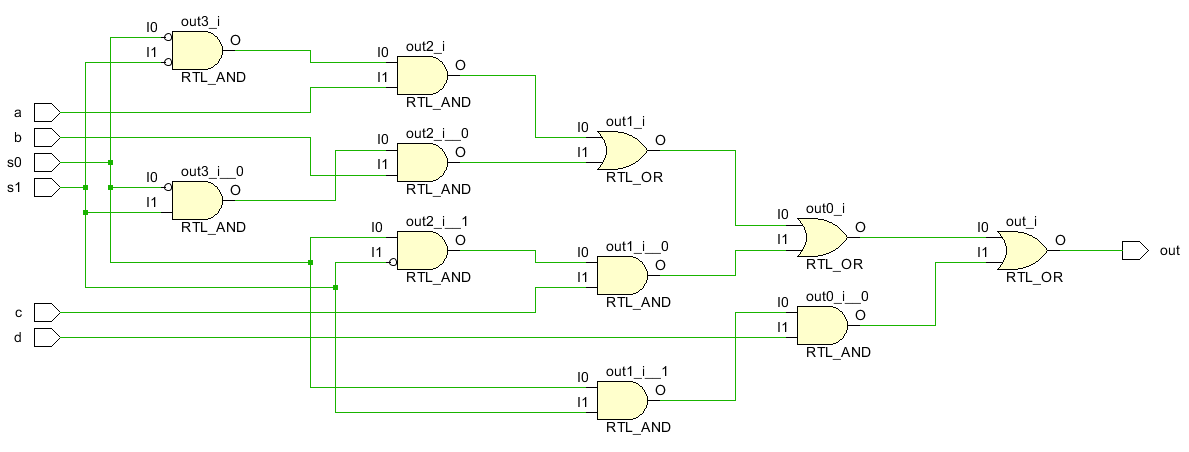
[2 to 4 active low decoder의 Schematic (NAND gate 사용)]



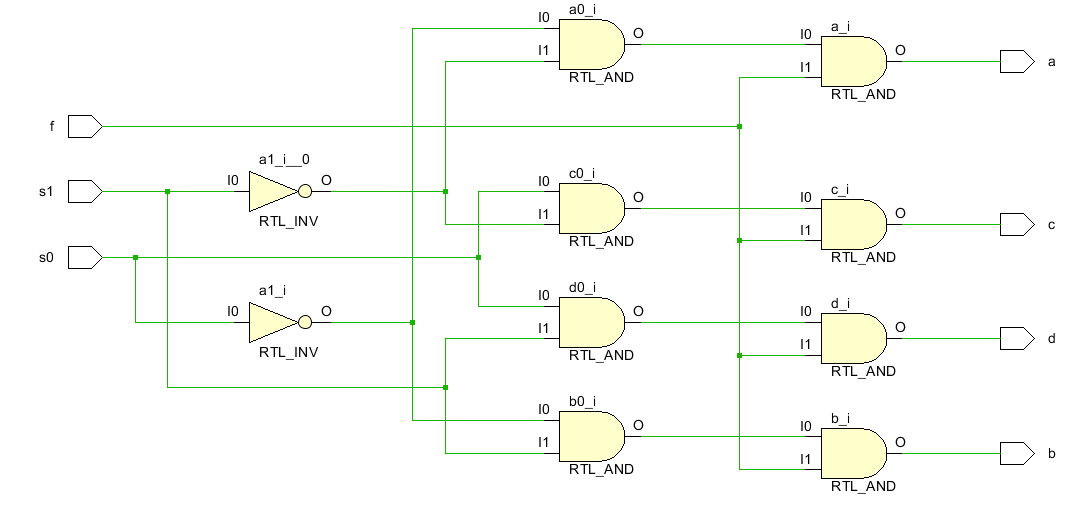
[4 to 2 encoder의 Schematic]



[BCD to Decimal decoder의 Schematic]



[4 to 1 line MUX의 Schematic]



[1 to 4 line deMUX의 Schematic]

**10.**

- ROM (Read Only Memory)

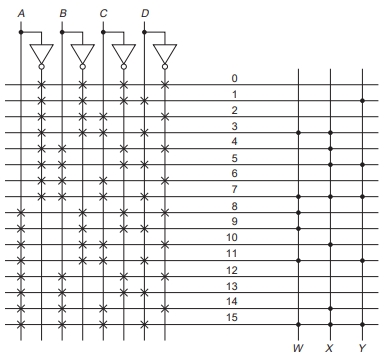
ROM은 읽기 전용 메모리로, ROM에 저장된 데이터는 언제든지 읽을 수 있으나, 저장된 데이터를 정상적인 동작 상태에서는 변경할 수 없다.

ROM은 n 비트 입력에 대해서 AND 게이트로 구성된 디코더와 메모리 어레이로 구성되어 있는데, 사용자는 메모리 어레이의 OR 게이트에 대한 연결만을 표현할 수 있어, 최소항의 합의 형태로 논리함수를 구현할 수 있다.

ROM을 사용하여 시스템을 설계하기 위해서는 각 함수에 대한 최소항(minterm)들을 알아야 한다.

아래 그림은 , ,

를 각각 ROM으로 구현한 회로를 나타낸다.



그림에서, 왼쪽 부분은 4 to 16 decoder의 기능을 하고, 오른쪽 부분은 각각의 함수의 최소항들을 메모리 어레이 위에서 OR 게이트에 연결함으로써 함수를 구현한다.