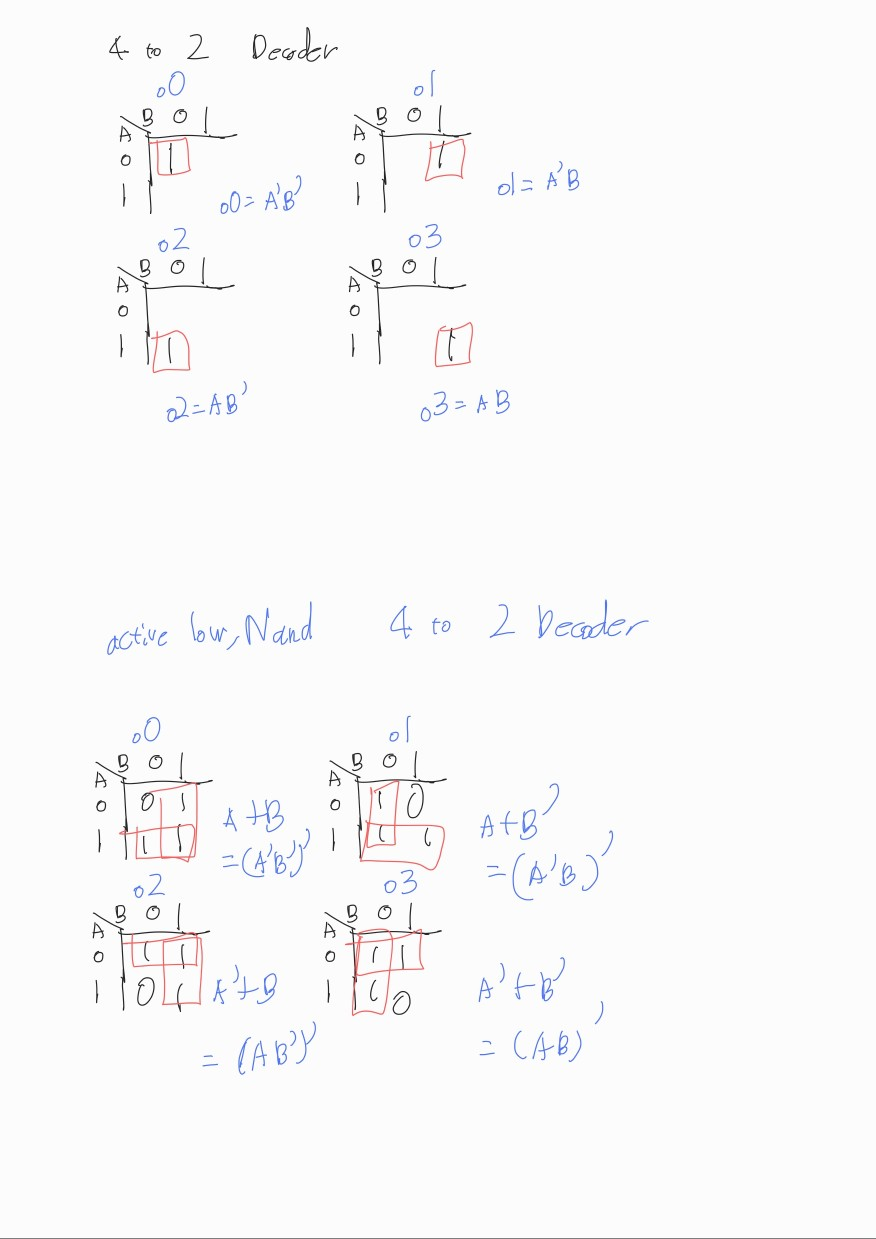
9주차 결과보고서

전공 : 아트앤테크놀로지 학년 : 3학년 학번 : 20191098 이름 : 백승주

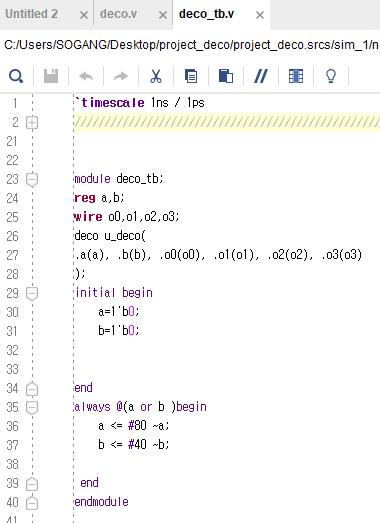
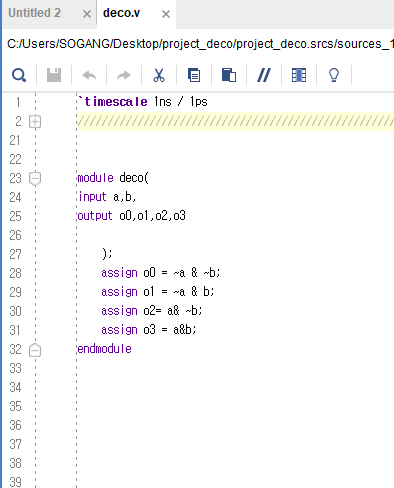
**1 -1.** 두 개의 입력값에 따라 4개의 출력값 중 하나의 값이 1이 되고 나머지 값들은 0이 되는 active high 디코더인 2 to 4 디코더의 진리표는 다음과 같다. 두 개의 입력값은 각각 a, b로 설정했고 4개의 output들은 각각 o0, o1, o2, o3, 으로 설정했다

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **a** | **b** | **o0** | **o1** | **o2** | **o3** |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

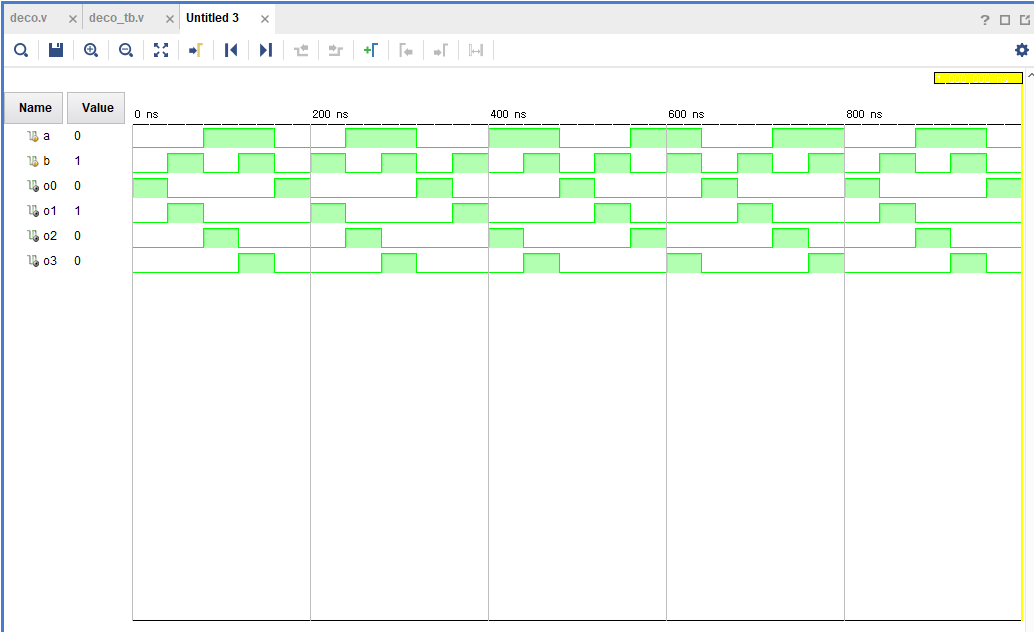
이 진리표에 따른 카르노맵은 다음과 같다.



위 카르노 맵에 따른 식들은 모두 AND 게이트로 이루어진 식들이다. 위 식들을 토대로Verilog 코드와 testbench 코드를 구현하면 다음과 같다.

l.

이에 따른 시뮬레이션 결과는 다음과 같다.



위 시뮬레이션을 보면 a = 0, b =1 일 때 이 input들을 이진수로 생각했을 때 나타나는 값이 1이므로 o1 output에만 1인 출력된 것을 알 수 있다. 이외에도 시뮬레이션에서 모든 output들이 서로 동시에 1이 되는 부분이 없는 것을 보면 하나의 출력만 1이 되는 디코더의 특징이 잘 나타난 것을 알 수 있다.

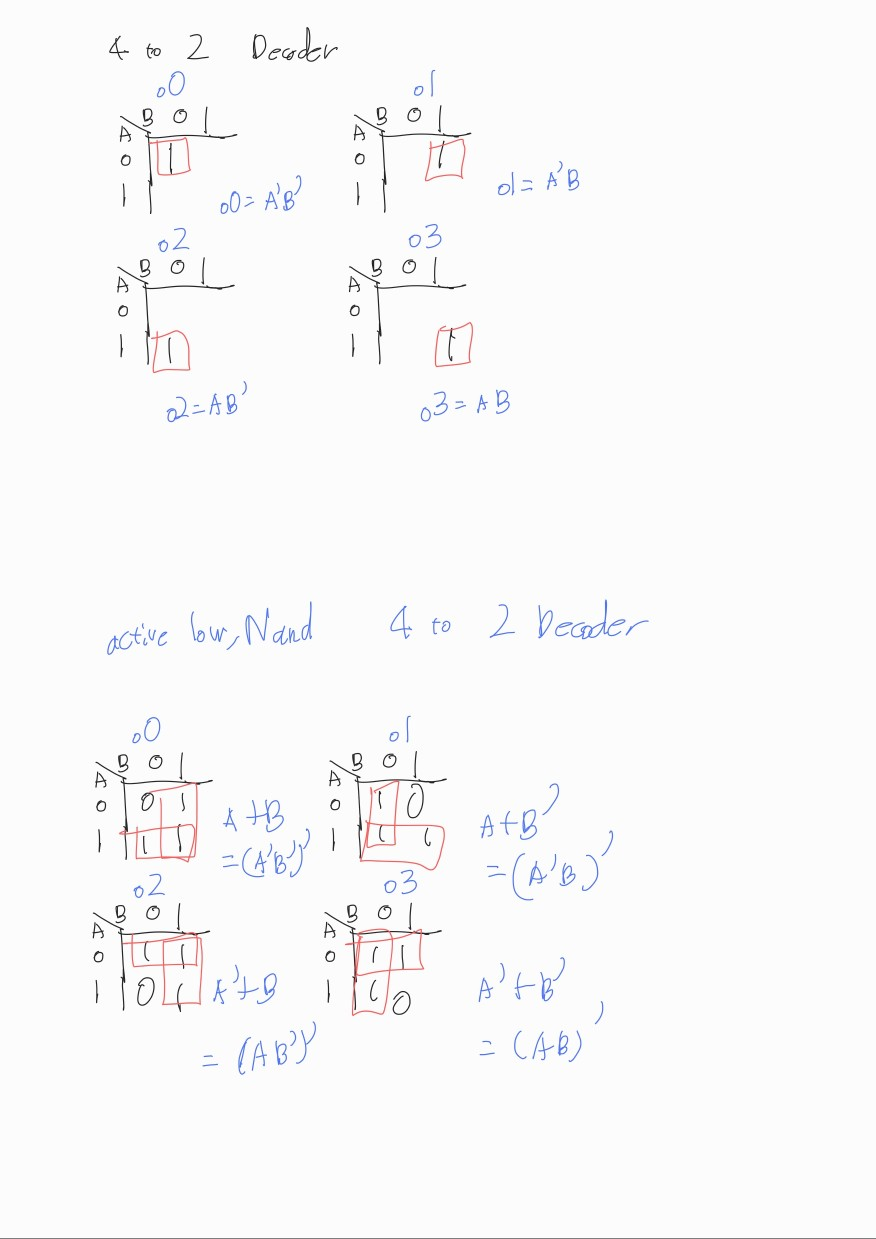
1-2. 2 to 4 decoder(NAND 게이트, active low)

또한 위에서 세운 식의 and 게이트를 nand 게이트로 바꾸면 input 값에 따라 특정 output만 0으로 출력하고 나머지 값들을 1로 출력하는 active high 디코더를 만들 수 있을 것이다.

Active low 디코더의 진리표와 카르노 맵은 다음과 같다.

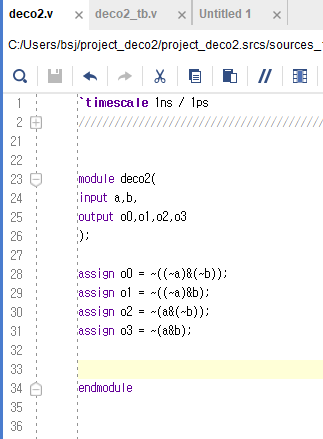
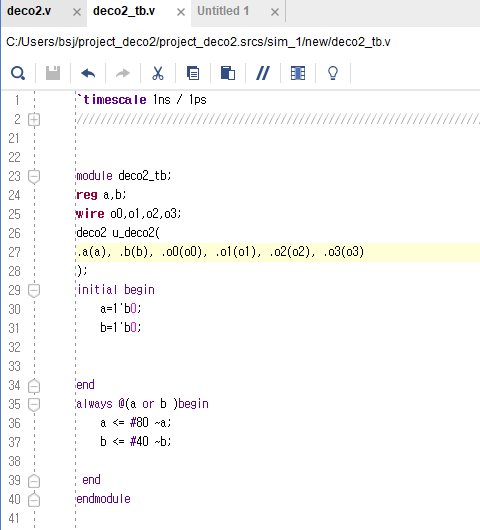
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **D0** | **D1** | **D2** | **D3** |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

위 진리표를 토대로 카르노 맵을 그리면 다음과 같다.

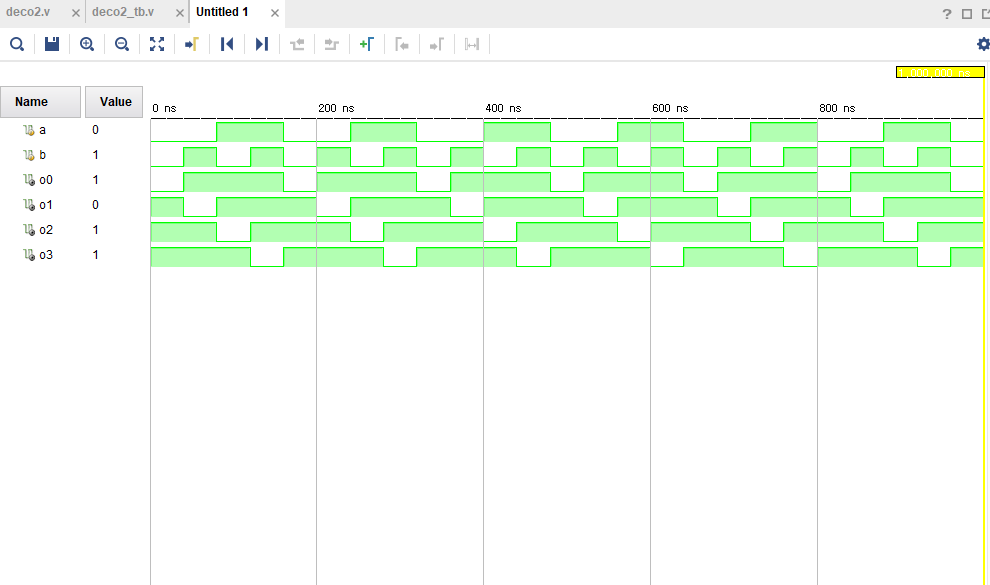


앞서 예상했던 대로 active high 디코더의 and 게이트를 nand 게이트로 바꾼 형태가 그대로 식으로 나타난 것을 알 수 있다.

위 카르노 맵을 이용하여 verilog 식을 세우면 다음과 같다.

위 식에 따른 시뮬레이션 결과는 다음과 같다.



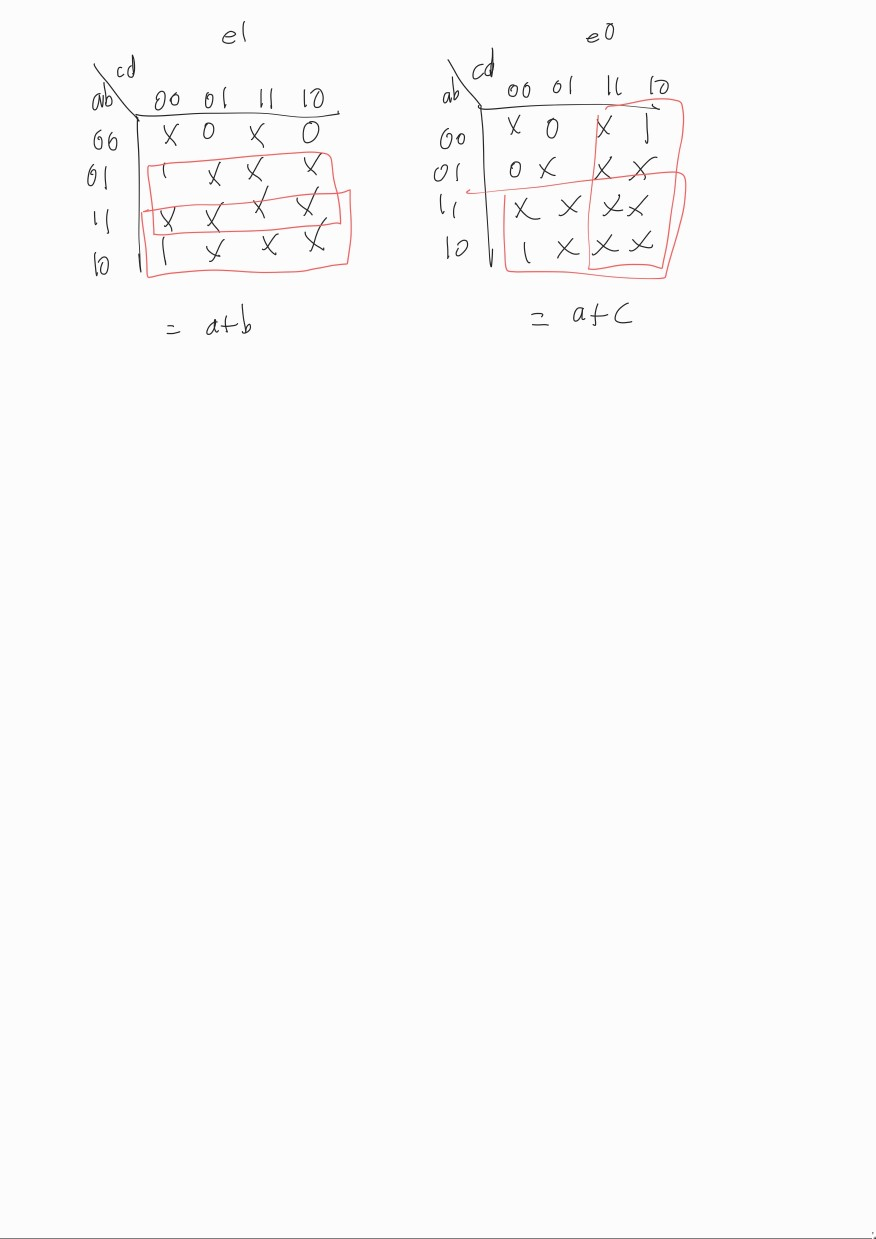
위 시뮬레이션의 결과를 보면 모든 output 값들에 대해 단 하나의 output만 0의 값을 가지는 것을 알 수 있다.

**2**. 4 to 2 인코더는 디코더와 반대 개념의 회로로 네 개의 input을 가지고 2개의 output을 가지는 회로다. 인코더의 진리표를 작성하면 다음과 같다.

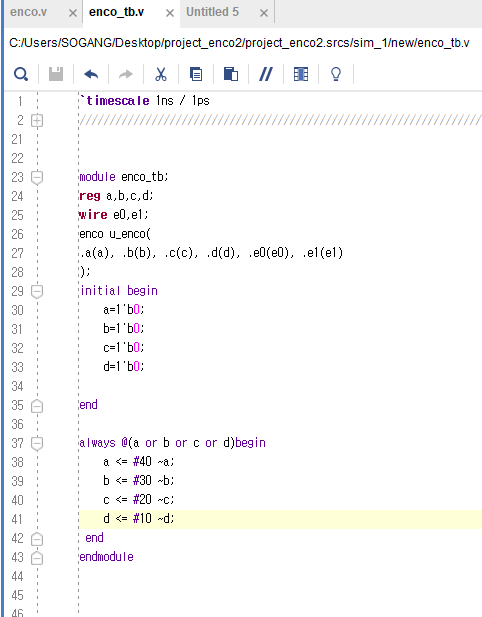
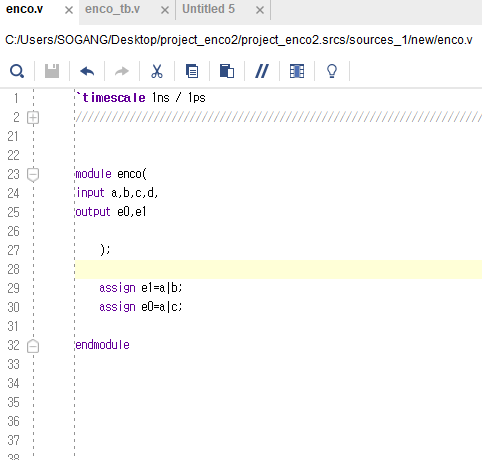
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | c | d | e1 | e0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

해당 진리표는 모든 input의 경우의 수를 다루지 않았다. 저 4가지의 input 값들의 경우를 제외하고는 다른 input들에 따른 output 값들은 어떤 값을 가져도 상관없기 때문이다.  
따라서 이 진리표를 토대로 만든 카르노맵은 don’t care 값들을 가지게 된다

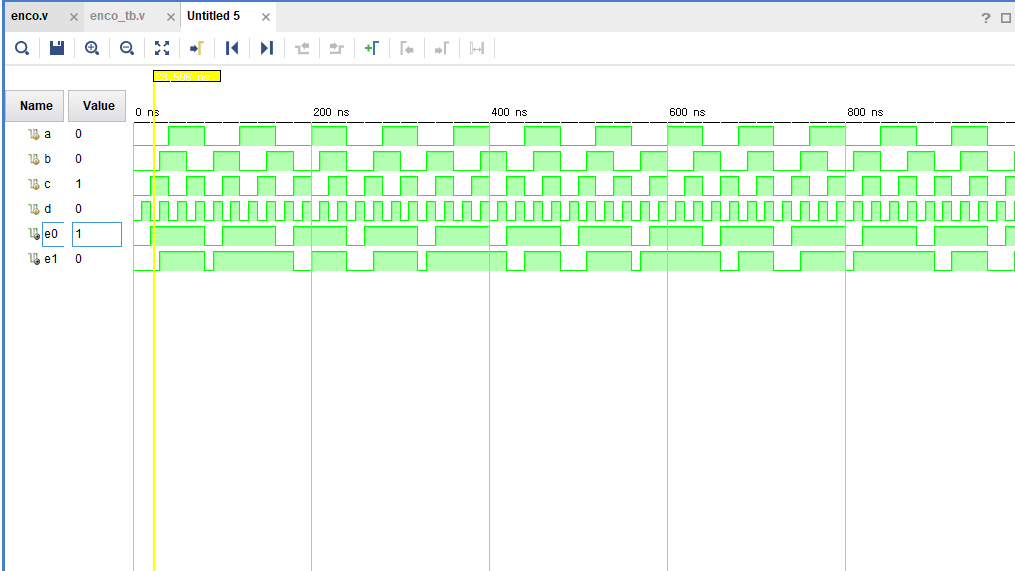
이 진리표를 토대로 다음과 같은 카르노맵이 작성된다.



위 진리표에 따라 Verilog 코드와 testbench 코드를 작성하면 다음과 같다.



위 코드에 따른 시뮬레이션 결과는 다음과 같다.



위 결과에서 abcd가 0010 일 때 e1 이 0 e0이 1값을 가지는 것을 확인할 수 있다. 마찬가지로 내가 원하는 input 값들인 1000, 0100, 0001 은 11, 10, 00 값을 가지는 것을 확인할 수 있다.

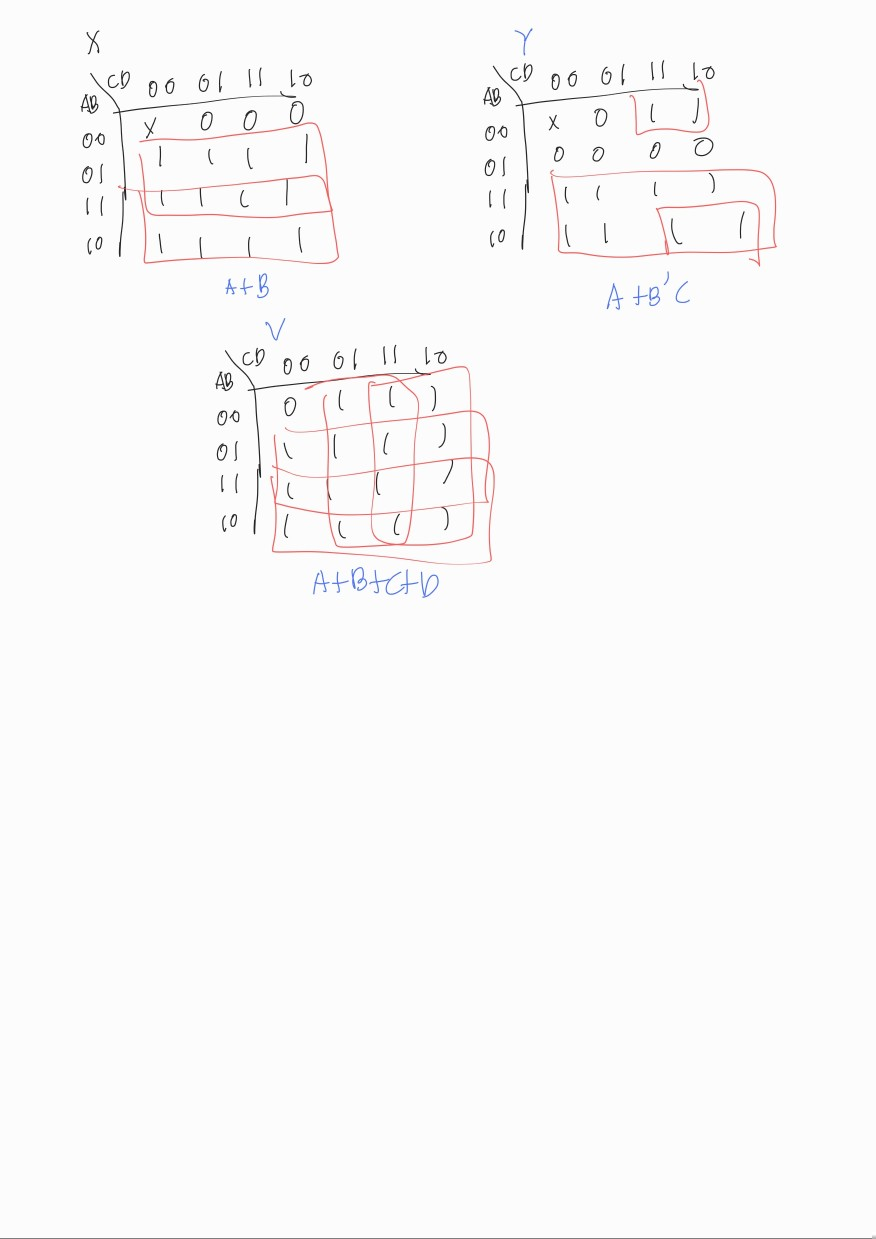
**4.** 4 to 2 인코더는 4가지 input에 대해서만 그 결과값을 정의하고 나머지 가능한 12개의 input들에 대해서는 정의를 하지 않은 회로다. 그러나 위 시뮬레이션 결과에서 볼 수 있듯이 12가지의 경우도 존재한다. 이러한 나머지 입력들의 경우가 나타날 경우 기기에서는 오류가 발생할 수 있다. 따라서 이런 오류 발생을 방지하기 위해 각 입력 비트들에 대해 우선순위를 설정해 개선한 인코더를 priority encoder 라 한다.

**5.** priority encoder의 진리표는 다음과 같다.

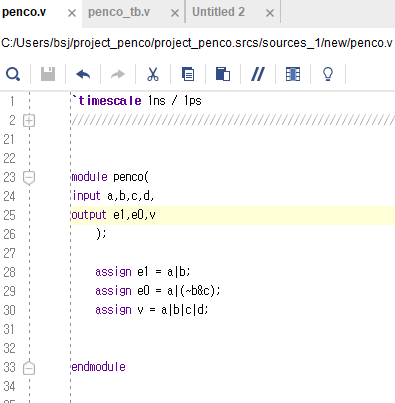
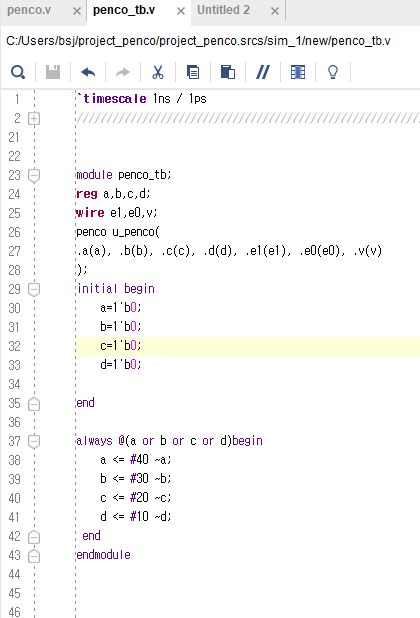
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | E1 | E0 | v |
| 0 | 0 | 0 | 0 | x | X | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

각 input들의 우선순위는 a가 가장 높고 b,c,d 순으로 낮아지게 했으며 v값은 입력 중 하나라도 1이 있으면 1을 나타내도록 했다. 또한 v값이 0이면 e1과 e0 값은 don’t care 하도록 했다.

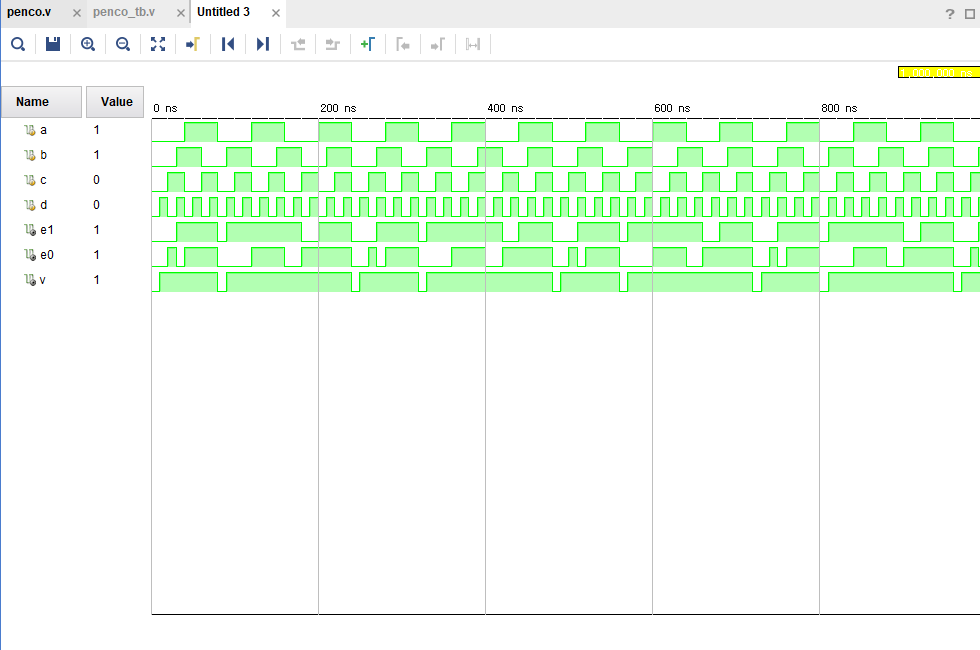
위 진리표에 따른 카르노 맵은 다음과 같다.



위 카르노 맵에 따른 verilog식은 다음과 같다.

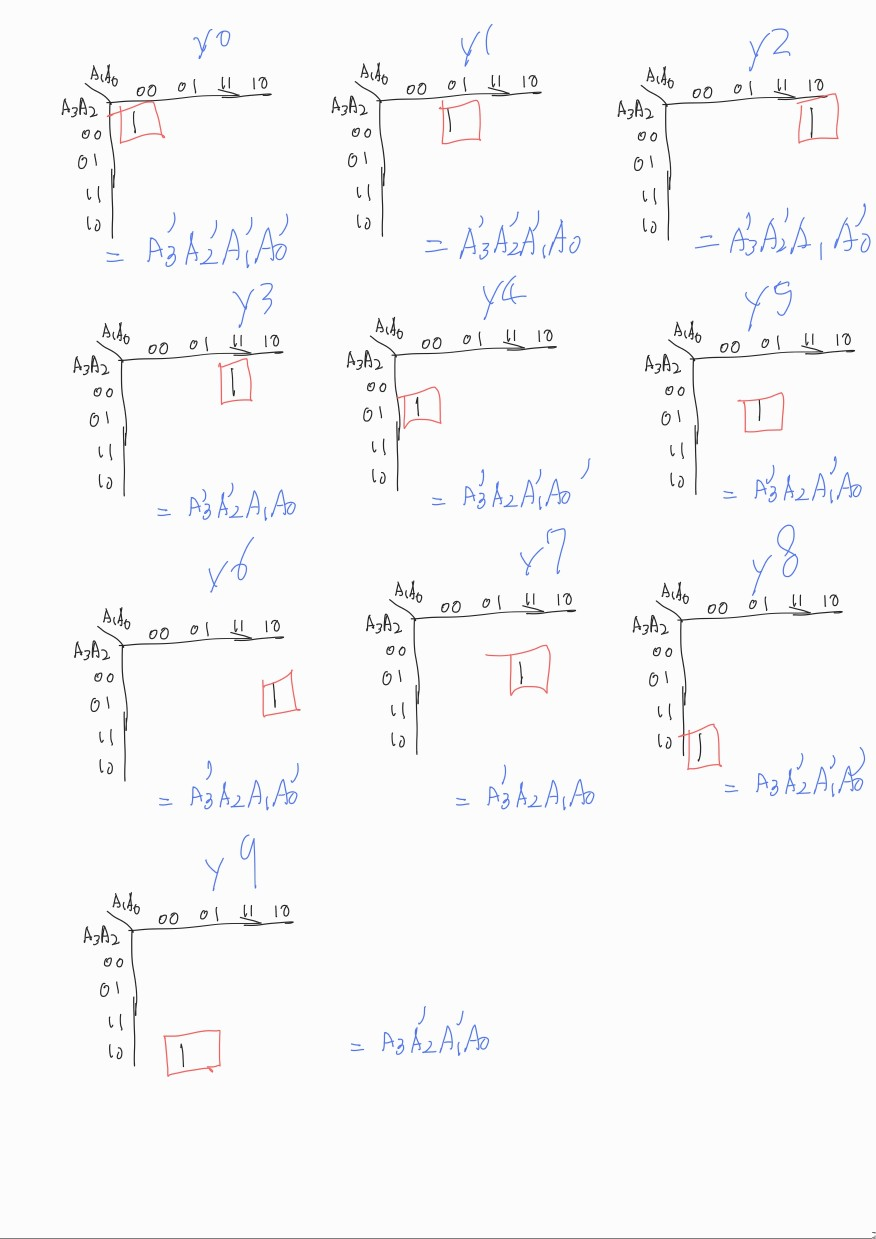
이에 따른 시뮬레이션 결과는 다음과 같다.



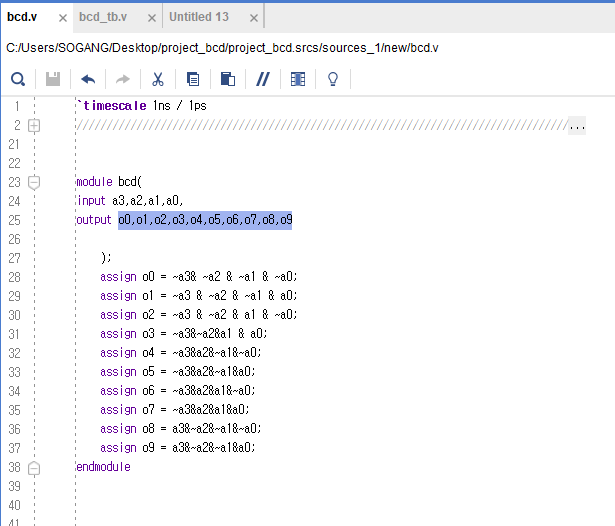
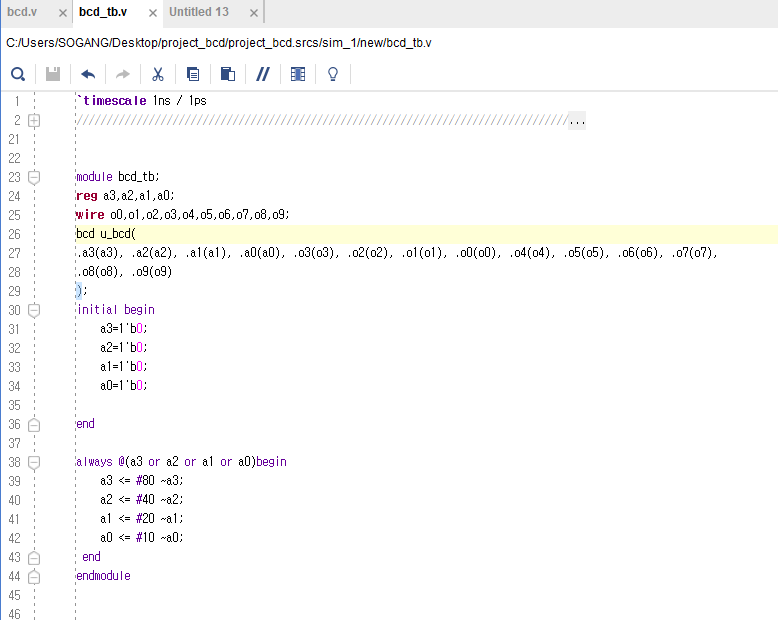
**5**. bcd to decimal 디코더는 4 비트 크기의 input들이 나타내는 숫자를 나타내는 output에 1을 출력하고 나머지에는 0을 출력하는 회로다. 이 회로의 진리표는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A3** | **A2** | **A1** | **A0** | **Y0** | **Y1** | **Y2** | **Y3** | **Y4** | **Y5** | **Y6** | **Y7** | **Y8** | **Y9** |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

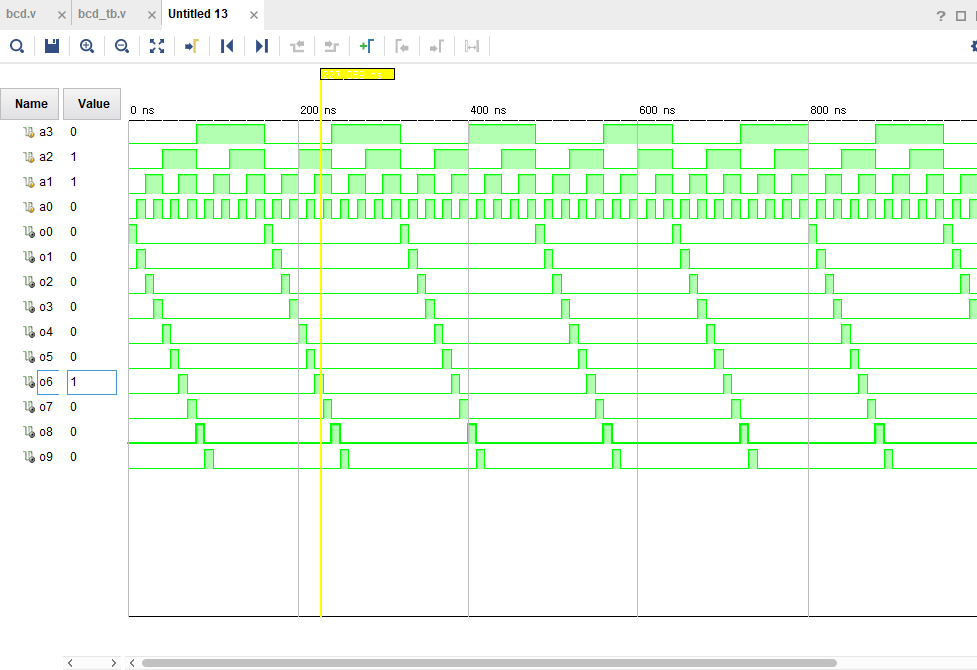
위 진리표에 따른 카르노맵은 다음과 같다.



위 카르노 맵을 토대로 소스 코드를 작성하면 다음과 같다.

위 코드들에 의한 시뮬이션 결과는 다음과 같다. input값이 1증가할 때마다 1이 들어간 결과값이 하나씩 옆으로 옮겨가는 것을 확인할 수 있다.



**6.** 디코더와 인코더는 실생활의 다양한 곳에서 쓰이고 있다. 인코더는 주로 데이터를 압축하거나 암호화할 때 사용할 수 있다. 먼저 데이터를 압축하는 용도로 인코더를 사용하는 사례로는 동영상 파일을 압축할 때, 또는 이미지나 음성 데이터를 압축할 때 등이 있다. 또한 암호화하기 위해 사용할 때는 통신과 같은 기술에서 사용이 가능하다. 또한 이외에도 디지털 tv를 송신할 때도 인코더 사용이 가능하다.

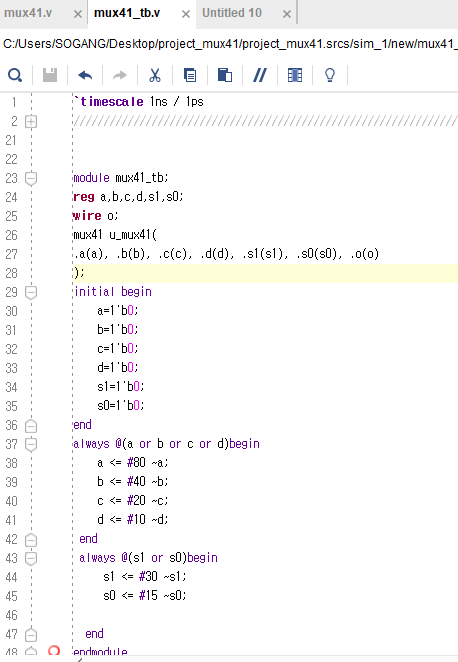
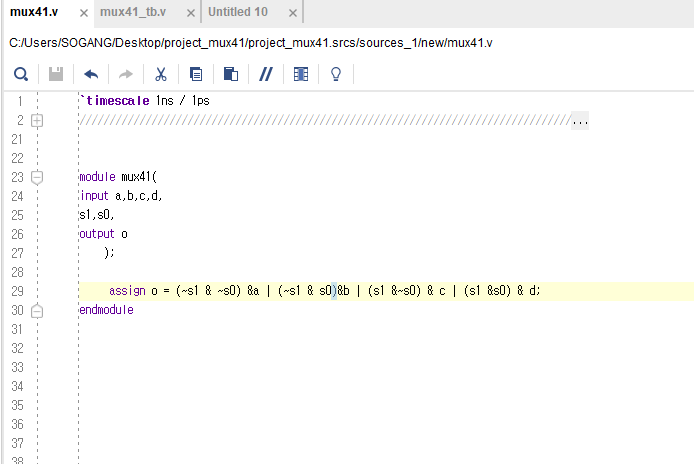
반대로 디코더는 이렇게 압축되거나 암호화된 데이터에 대해 원본으로 변환하거나 암호를 해독하는데 사용이 가능하다. 이 때문에 압축된 동영상이나 이미지를 원본 파일로 변환하는데 사용이 가능하며 송신자의 통신 신호를 받아을 때도 사용이 가능하다. 마찬가지로 디지털 tv 신호를 받아 텔레비전에 화면을 출력할 때도 사용이 가능하다.

즉 인코더와 디코더는 서로 한 세트처럼 같이 자주 쓰인다.

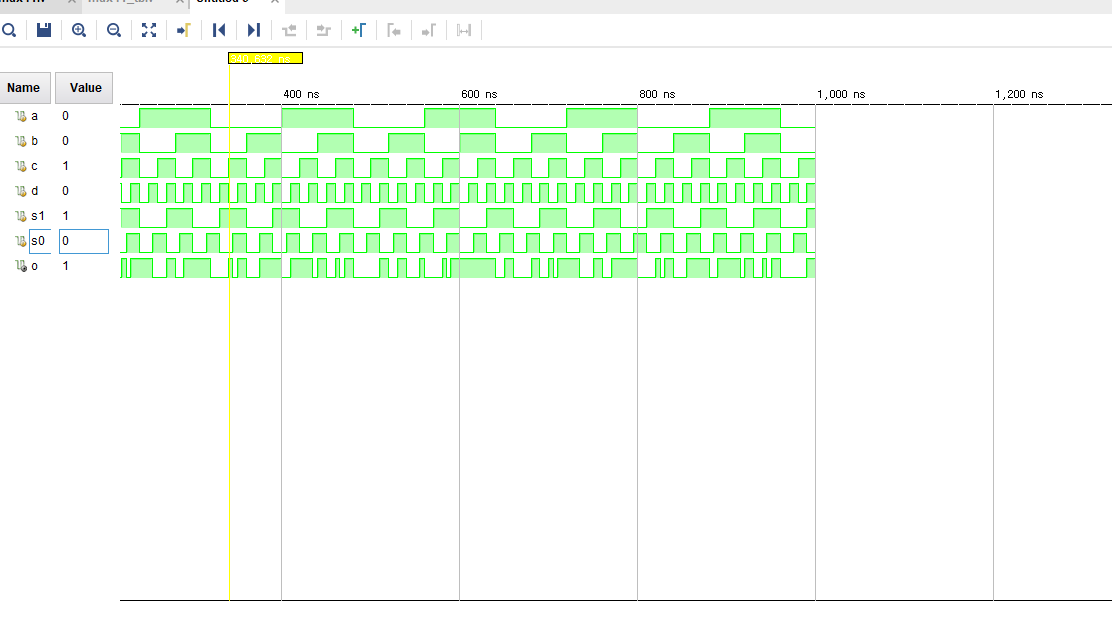
**7.** 4 to 1 line Multiplexer 는 네 개의 input 중 하나를 select bit에 따라 골라 output에 전달하는 회로기 때문에 6개의 input과 1개의 output을 가진다. 이러한 4 to 1 line mux의 진리표를 구성하면 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **s1** | **s0** | **Y** |
| 0 | X | X | X | 0 | 0 | 0 |
| 1 | X | X | X | 0 | 0 | 1 |
| X | 0 | X | X | 0 | 1 | 0 |
| X | 1 | X | X | 0 | 1 | 1 |
| X | X | 0 | X | 1 | 0 | 0 |
| X | X | 1 | X | 1 | 0 | 1 |
| X | X | X | 0 | 1 | 1 | 0 |
| X | X | X | 1 | 1 | 1 | 1 |

Select bit 가 선택하지 않은 input 은 상관이 없기 때문에 don’t care를 이용해 처리했다. 이에 따른 코드를 작성하면 다음과 같다.



위 소스 코드들에 의한 결과는 다음과 같다.

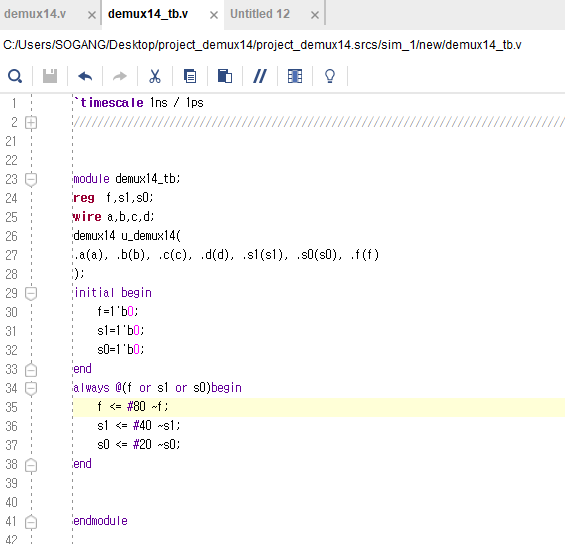
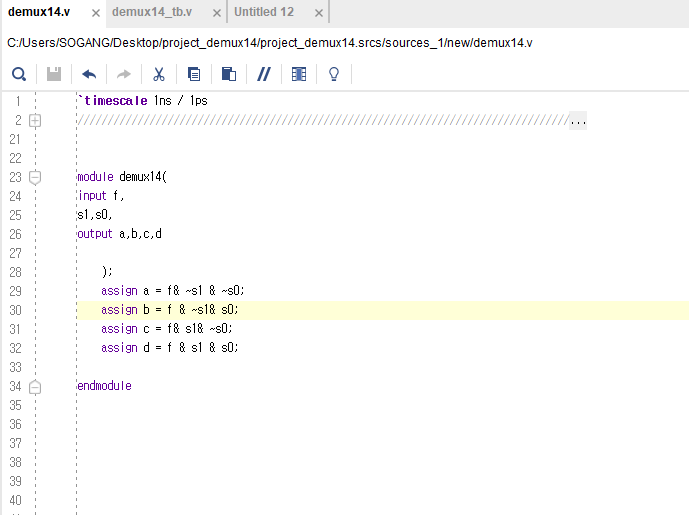


위 예시를 보면 s1,so이 10으로 input 값들 중에서 c를 선택했기 때문에 그에 대한 결과값인 o가 1을 출력하고 있는 것을 확인할 수 있다.

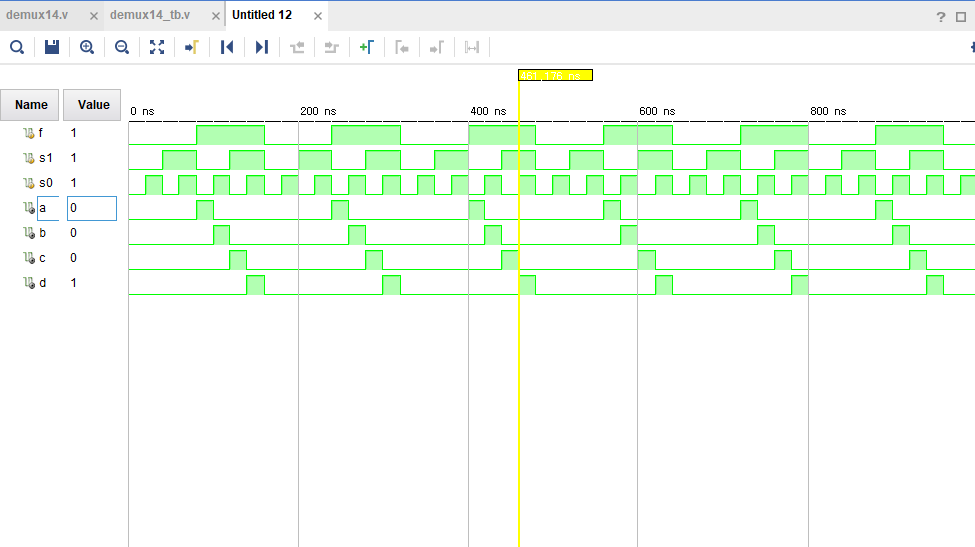
**8.** 1 to 4 line demux는 s1과 s0의 select bit가 결정하는 output에 input값인 f에 들어있는 값을 집어넣는 회로다. 이 회로의 진리표를 생성하면 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| s1 | s0 | f | a | b | c | d |
| 0 | 0 | 0 | 0 | x | x | X |
| 0 | 1 | 0 | X | 0 | x | x |
| 1 | 0 | 0 | x | x | 0 | X |
| 1 | 1 | 0 | X | x | x | 0 |
| 0 | 0 | 1 | 1 | x | X | X |
| 0 | 1 | 1 | X | 1 | x | X |
| 1 | 0 | 1 | X | X | 1 | x |
| 1 | 1 | 1 | x | X | 0 | 1 |

위 진리표에 따른 식을 작성하면 다음과 같다,



이에 따른 시뮬레이션 결과는 다음과 같다.



위의 예시에서 s1,s0이 11로 select bit가 output중 4번째 값인 d를 선택했음을 알 수 있고 이로 인해 이 output에 f값인 1이 그대로 출력되었음을 확인할 수 있다.

**9.** 2 to 4 디코더, 4 to 2 인토더, Priority Encoder, 4 to 1 line MUX, 1 to 4line demux의 회로들에 대해 진리표를 작성하고 카르노 맵을 활용해 직접 verilog 코드를 작성해보며 시뮬레이션 결과를 확인해봤다.   
2 to 4 디코더의 경우에는 AND 게이트와 NAND 게이트로 구성된 형태 주 가지 형태를 모두 확인했다.

또한 디코더와 인코더가 서로 반대되는 개념을 가진 회로라는 것을 시뮬레이션을 통해 한 번 더 확인할 수 있었으며 인코더에서 4가지의 input들에 대해서만 output을 정해 놓으면 그 나머지 입력들이 발생할 경우 오류가 발생할 수 있음을 알았다. 이런 단점을 개선하기 위해 우선순위를 정하는 priority encoder가 있음을 알았고 직접 구현해봤다. 이외에도 디코더와 인코더가 실생활에서 응용되는 방식을 살펴보고 멀티플렉서와 디멀티플렉서 서로 반대되는 개념이라는 것 또한 확인할 수 있었다.

**10.** 추가이론 – 플립플롭(flip-flop)

플립플롭은 디지털 시스템에서 주로 사용되는 논리 회로로 1비트의 정보를 저장하고 제어 신호에 따라 정보를 유지하거나 변경할 수 있다. 일반적인 플립플롭 유형은 D 플립플롭, JK 플립플롭, RS 플립플롭, T 플립플롭 등이 있다.

D 플립플롭은 입력 값과 출력값이 같은 플립플롭이다. 데이터 입력과 클럭(Clock) 입력을 가지고 있는데 클럭 신호가 positive edge에 도달할 때마다 D 입력 값이 플립플롭의 출력으로 전달된다.

RS 플립플롭: RS 플립플롭은 입력 세트(S)와 리셋(R) 입력을 가지고 있는 가장 기본적인 플립플롭이다. S와 r 값에 둘 다 1을 보내게 되면 보통 플립플롭이 불안정한 상태에 빠지게 되므로 이를 피해야 한다..

JK 플립플롭: JK 플립플롭은 앞선 rs 플립플롭의 문제점을 보완한 형태로 j와 k에 둘 다 1이 들어올 경우 상태를 반전시킨다. 널리 활용되는 플립 플롭이다.

T 플립플롭: T 플립플롭은 토글(toggle)을 사용하는 플립플롭으로 t가 0이면 그대로 표시하고 1이면 이전 상태의 보수 값을 표시한다. 주로 on, off가 교차되는 스위치에 많이 사용된다.