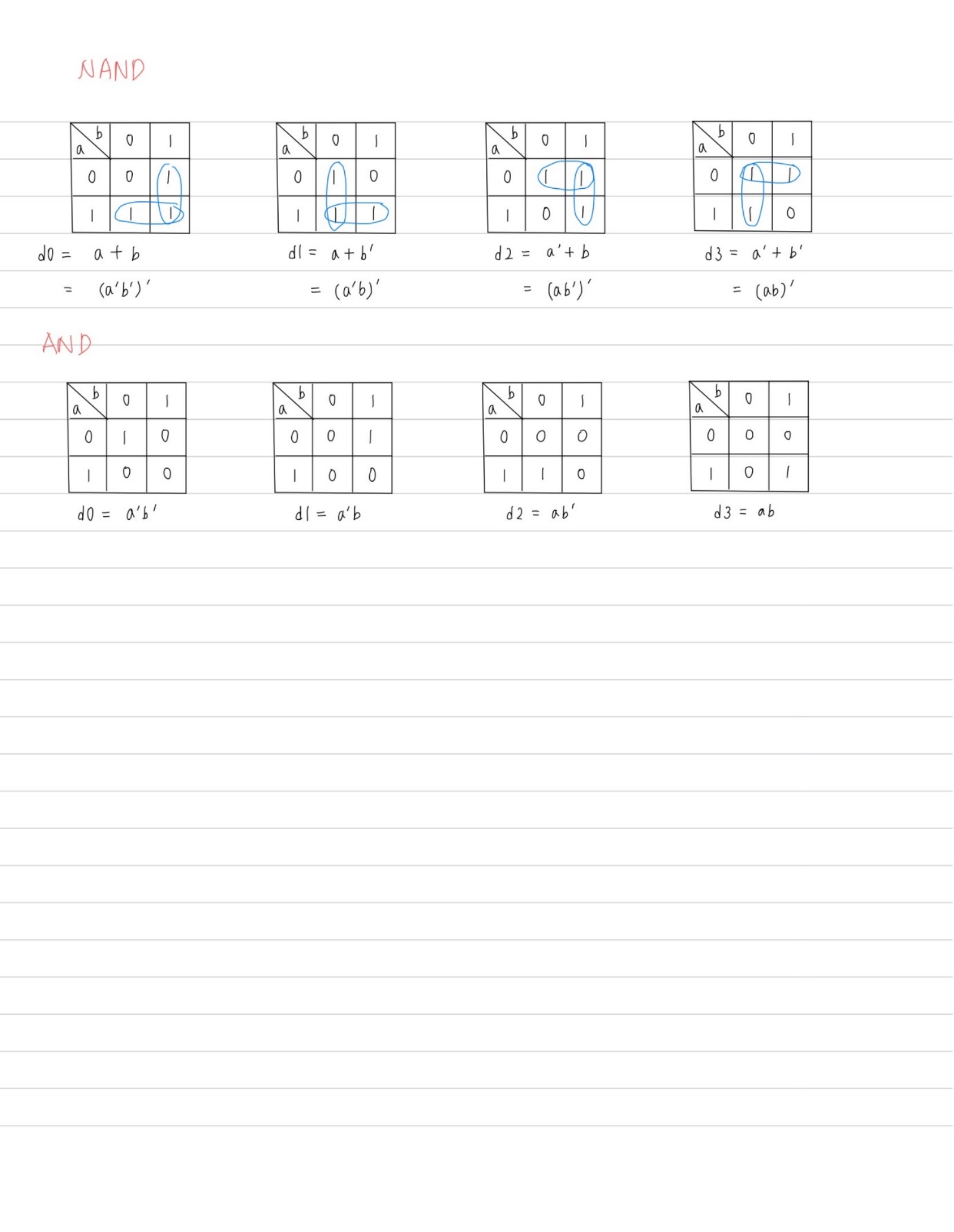
9주차 결과보고서

전공: 영미어문/컴퓨터공학과 학년: 3학년 학번: 20200185 이름: 박정주

**1.**

AND gate를 사용하는 2 to 4 Decoder의 카르노맵을 그려 논리식을 구하였다.



AND gate를 사용하는 2 to 4 Decoder의 Design source를 다음과 같이 코딩하였다. 2개의 입력 a, b는 암호화된 데이터를 의미하고, 출력 d0, d1, d2, d3은 입력 데이터를 복호화한 결과를 의미한다. a, b의 값에 따라 d0, d1, d2, d3의 값이 달라지게 된다.

테이블이(가) 표시된 사진

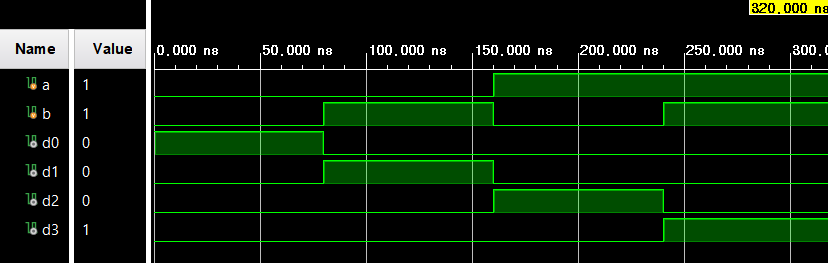
자동 생성된 설명

2x4 Decoder의 Simulation source를 다음과 같이 코딩하였다. 160 시간단위가 지날 때마다 a가 부정되고, 80 시간단위가 지날 때마다 b가 부정된다. 320 시간단위가 지나면 시뮬레이션이 종료되도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

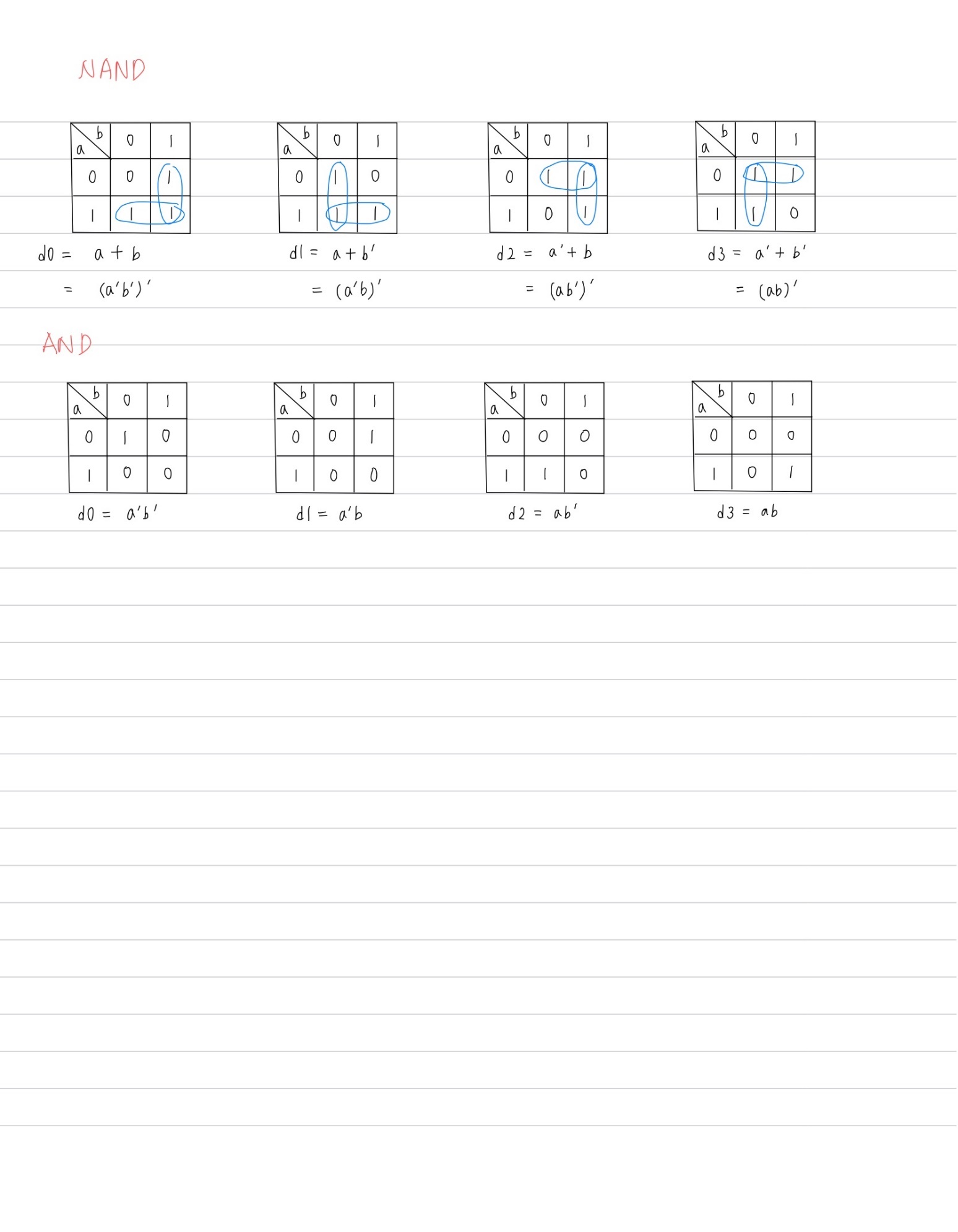
시뮬레이션 결과는 다음과 같다.



진리표는 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a | b | d0 | d1 | d2 | d3 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

NAND gate를 사용하는 2 to 4 Decoder의 카르노맵을 그리고 드모르간의 법칙을 사용하여 nand 연산을 사용하는 논리식을 구하였다.

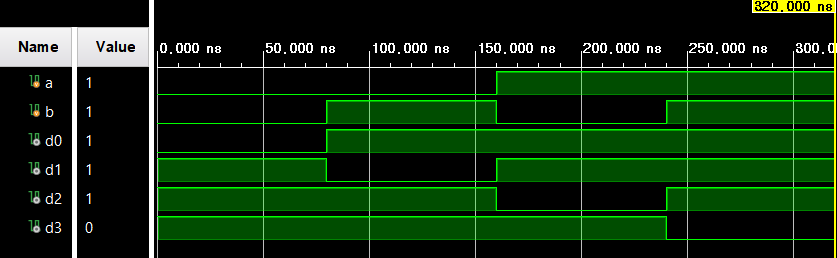


NAND gate를 사용하는 2 to 4 Decoder의 Design source를 다음과 같이 코딩하였다. 2개의 입력 a, b는 암호화된 데이터를 의미하고, 출력 d0, d1, d2, d3은 입력 데이터를 복호화한 결과를 의미한다.

테이블이(가) 표시된 사진

자동 생성된 설명

AND gate를 사용하는 2 to 4 Decoder와 동일한 시뮬레이션 소스를 사용하였다. 시뮬레이션 결과는 다음과 같다.

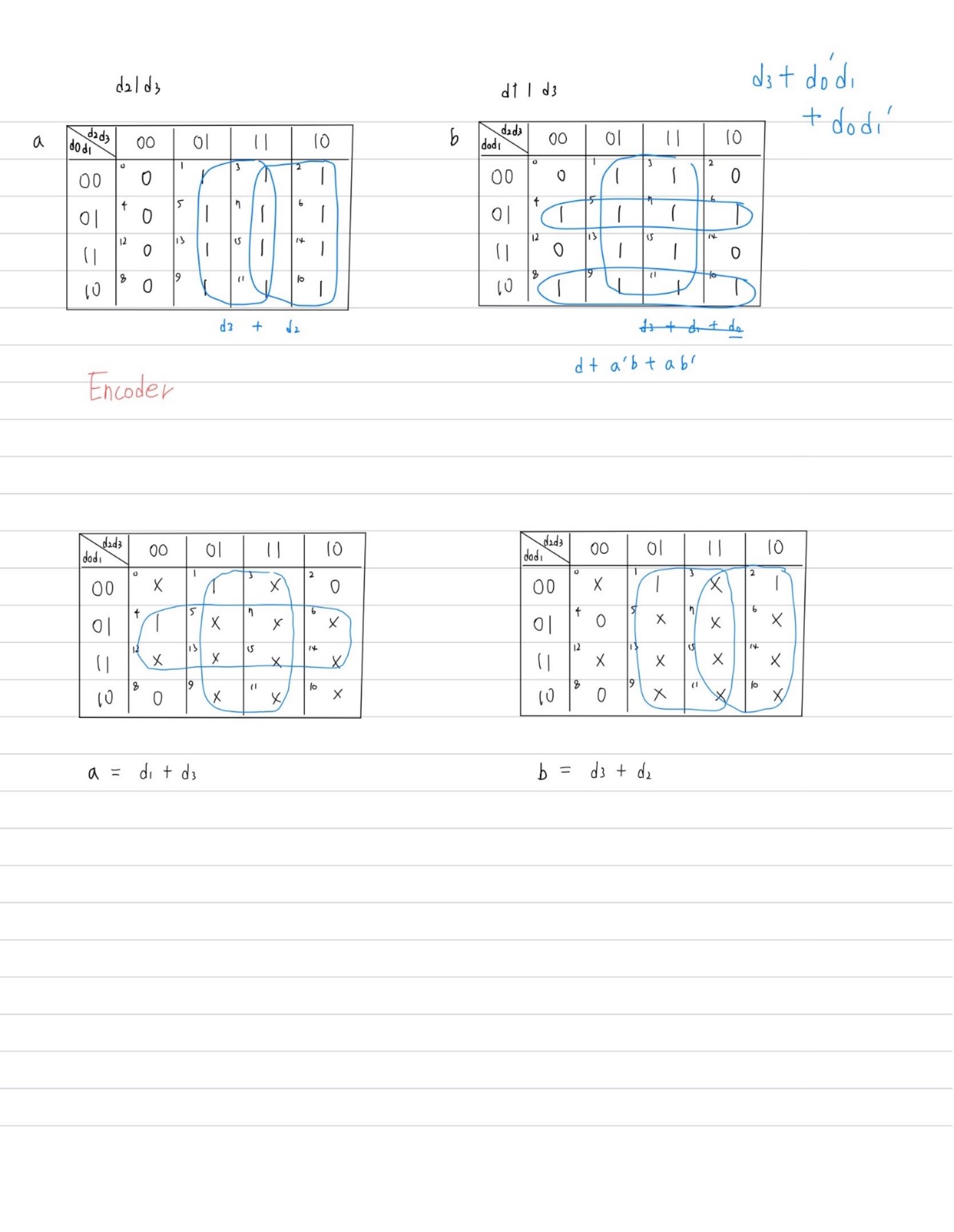


진리표는 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

**2.**

4 to 2 Encoder의 카르노맵을 그려 논리식을 구하였다.



4 to 2 Encoder의 design source를 다음과 같이 코딩하였다. 입력 d0, d1, d2, d3는 암호화할 데이터를 의미하며, 출력 a, b는 입력 데이터를 암호화한 결과이다. OR gate를 사용하였다.

테이블이(가) 표시된 사진

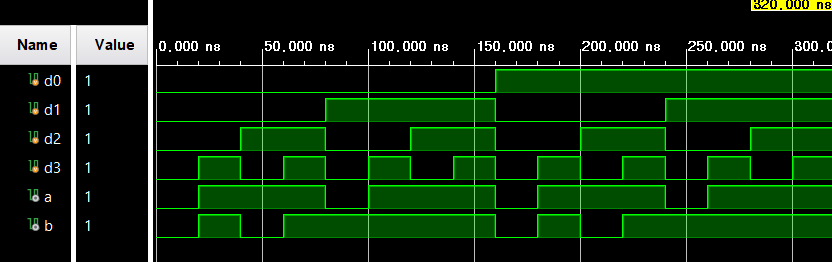
자동 생성된 설명

4x2 Encoder의 Simulation source를 다음과 같이 코딩하였다. 각각 160, 80, 40, 20 시간단위가 지날 때마다 d0, d1, d2, d3이 부정되도록 하였다. 320 시간단위가 지나면 시뮬레이션이 종료되도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

시뮬레이션 결과는 다음과 같다.



4to2 Encoder의 입력 형태 4가지에 대해 작성한 진리표는 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **d0** | **d1** | **d2** | **d3** | **a** | **b** |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |

**3.**

4to2 Encoder에서 2개 이상의 입력 값이 1이거나, 모든 입력이 0일 경우에는 잘못된 결과값이 출력된다. 예를 들어, d0, d1, d2, d3이 0, 1, 1, 0인 경우, d3이 0인데도 불구하고 a, b의 값으로 1, 1이 출력된다. 4가지 입력 형태를 제외한 다른 입력 형태들은 어떤 결과값을 출력해야 하는지 정확히 알 수 없는 모호한 입력 값들이다.

모든 입력 형태에 대해 작성한 진리표는 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **d0** | **d1** | **d2** | **d3** | **a** | **b** |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

**4.**

16가지 입력 형태에 대해 동작하는 priority encoder의 design source를 다음과 같이 코딩하였다. 우선순위는 d3 > d2 > d1 > d0이다. 우선순위가 가장 높은 d3이 1이면 나머지 입력들의 값에 상관없이 결과값은 1, 1이다. d1이 0일 때 d2이 1이면 나머지 입력들의 값에 상관없이 결과값은 1, 0이다. d3과 d2이 0일 때 d1이 1이면 d0의 값에 상관없이 결과값은 0, 1이다. d3, d2, d1이 모두 0이고 d0만 1일 때 결과값은 0, 0이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

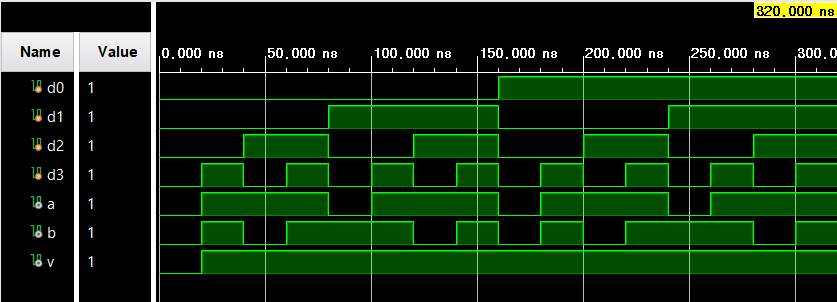
출력 v는 입력이 유효한지 판단하기 위한 변수이다. 어떤 입력도 1이 아닌 경우, 즉 모든 입력이 0일 때는 어떠한 결과값에도 해당하지 않으므로, 입력이 유효하지 않다. 따라서 모든 입력이 0일 때만 v가 0이 되고, 나머지 경우에는 1이 되도록 v에 d0|d1|d2|d3을 할당하였다.

Priority encoder의 simulation source를 다음과 같이 코딩하였다. 각각 160, 80, 40, 20 시간단위가 지날 때마다 d0, d1, d2, d3이 부정되도록 하였다. 320 시간단위가 지나면 시뮬레이션이 종료되도록 하였다.

텍스트이(가) 표시된 사진

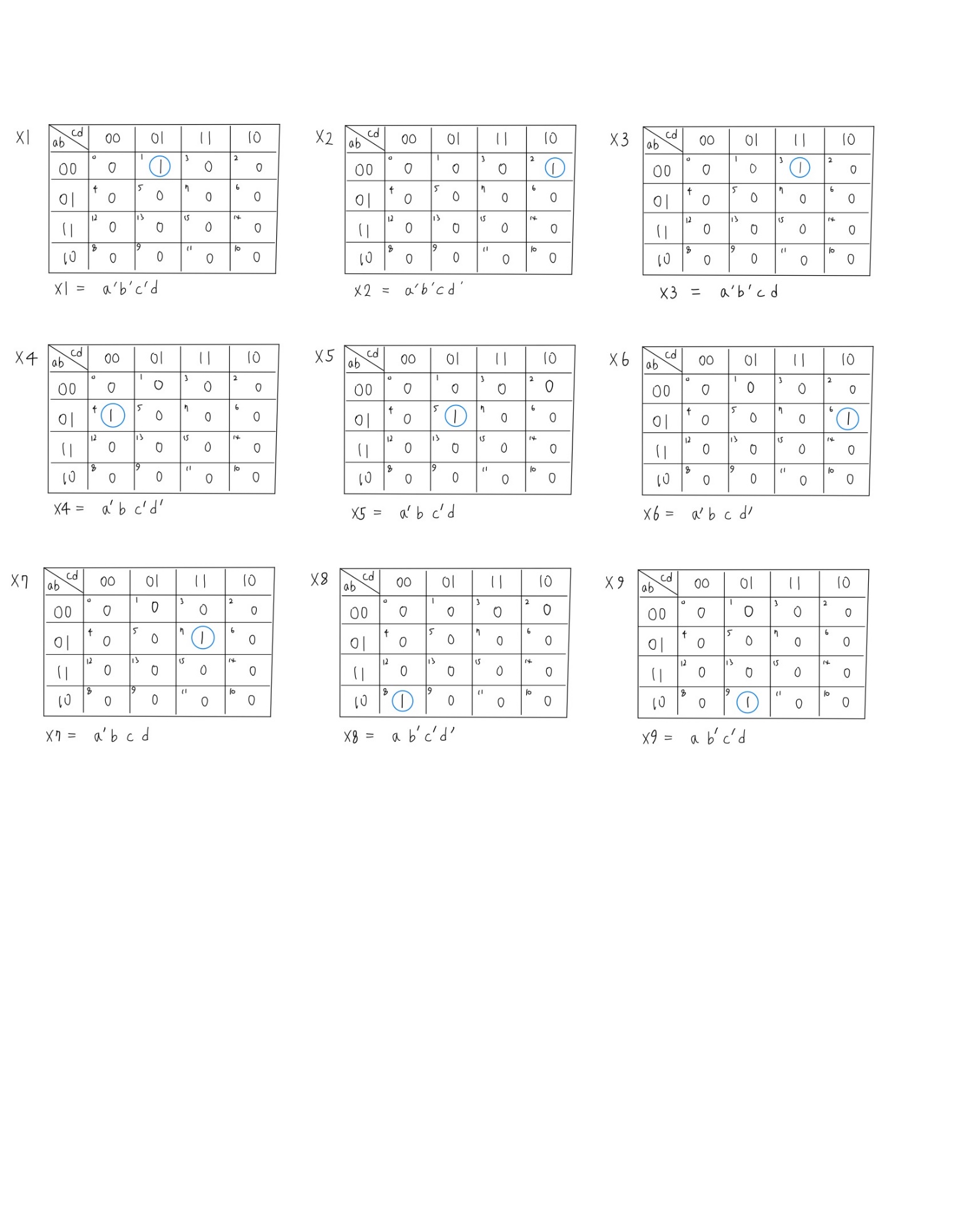
자동 생성된 설명

Priority encoder의 시뮬레이션 결과는 다음과 같다. 의도한 바와 같이 우선순위가 높은 입력이 1이면 우선순위가 더 낮은 입력은 고려하지 않고 결과가 정상적으로 출력되는 것을 확인할 수 있다. 또한, 입력이 모두 0일 경우 v가 0으로 출력되는 것을 확인할 수 있다.



**5.**

BCD to Decimal Decoder의 각 출력들의 카르노 맵은 다음과 같다.



BCD to Decimal Decoder의 design source를 다음과 같이 코딩하였다. 입력 a, b, c, d는 BCD 수의 각 자리를 의미한다. 출력 x0부터 x9는 BCD수가 Decimal로 변환되었을 때 어떤 수가 되는지를 의미한다. 예를 들어, x1의 값이 1이고 나머지 출력들의 값이 0이면, 입력된 BCD수를 Decimal로 바꾸면 1이라는 사실을 의미한다. x5의 값이 1이고 나머지 출력들의 값이 0이면, 입력된 BCD를 Decimal로 바꾸면 5가 된다는 것을 의미한다.

테이블이(가) 표시된 사진

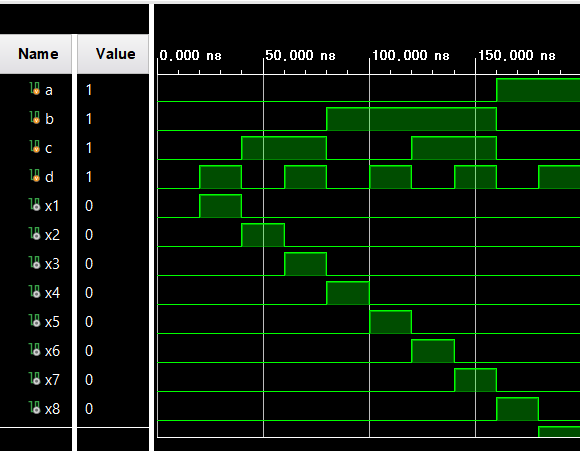
자동 생성된 설명

BCD to Decimal decoder의 simulation source를 다음과 같이 코딩하였다. 1bit 수 a, b, c, d의 값이 0으로 시작하도록 하고, 각각 160, 80, 40, 20 시간단위마다 a, b, c, d의 값을 부정하도록 하였다. 320 시간단위가 지나면 시뮬레이션을 끝내도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

시뮬레이션 결과는 다음과 같다.



진리표는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **a** | **b** | **c** | **d** | **x1** | **x2** | **x3** | **x4** | **x5** | **x6** | **x7** | **x8** | **x9** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**6.**

컴퓨터 내부에 2진수로 저장되어 있는 데이터를 해독하여 인간이 읽을 수 있는 문자와 같은 형태로 복호화 하여 출력할 때 Decoder가 사용된다. 반대로 인간이 키보드에 입력한 것을 컴퓨터가 해석 가능한 2진수로 부호화 하여 전달하거나 저장할 때 Encoder가 사용된다. 또한 텍스트, 음성, 사진, 영상 파일 등을 보안이나 최적화 등을 목적으로 복호화할 때도 Encoder가 사용된다.

**7.**

4 to 1 line mux의 design source를 다음과 같이 코딩하였다. 입력 x0, x1, x2, x3은 mux의 입력 데이터, 입력 c0, c1은 선택자를 의미한다. c0과 c1의 값에 따라 x0, x1, x2, x3 중 어떤 입력이 선택될지 결정된다. 출력 y는 입력 데이터 중 선택된 것을 의미한다.

텍스트이(가) 표시된 사진

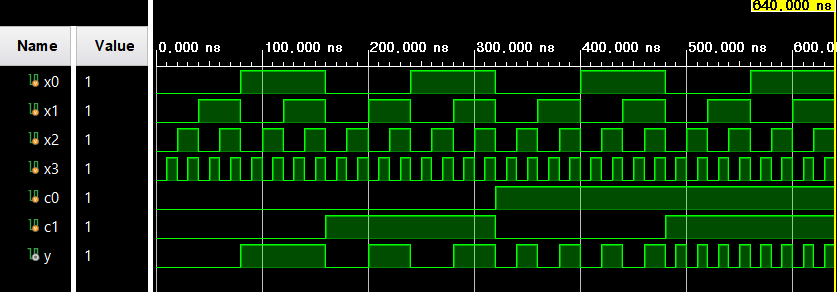
자동 생성된 설명

MUX의 simulation source를 다음과 같이 코딩하였다. 모든 입력의 값이 0으로 시작하도록 하였다. 각각 320, 160, 80, 40, 20, 10 시간단위마다 c0, c1, x0, x1, x2, x3의 값이 부정되도록 하였다. 640 시간단위가 지나면 모든 경우의 수를 확인할 수 있으므로 시뮬레이션이 끝나도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

시뮬레이션 결과는 다음과 같다.



진리표는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x0 | x1 | x2 | x3 | c0 | c1 | 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 |

**8.**

1x4 line DEMUX의 진리표는 다음과 같다. 선택자 c1, c0이 0, 0이면 y0에 x의 값을 출력하고, 나머지 y1, y2, y3의 값으로는 0을 출력한다. c1, c0이 0, 1이면 y1의 값으로 x의 값을 출력한다. c1, c0이 1, 0이면 y2에 x의 값을 출력한다. c1, c0이 1, 1이면 y3에 x의 값을 출력한다. 그러므로 x가 0인 경우에는 선택자에 상관없이 출력이 모두 0이 된다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | c1 | c0 | y0 | y1 | y2 | y3 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

4x16 Decoder를 만들기 위해서는 4개의 입력과 16개의 출력이 필요하다. 4개의 1 to 4 line Demux를 이용하여 4to16 Decoder를 만들 수 있다.

1bit수 4개의 입력을 조합해 가능한 16가지의 입력 형태를 나열한 결과, 다음과 같은 규칙을 찾을 수 있었다.

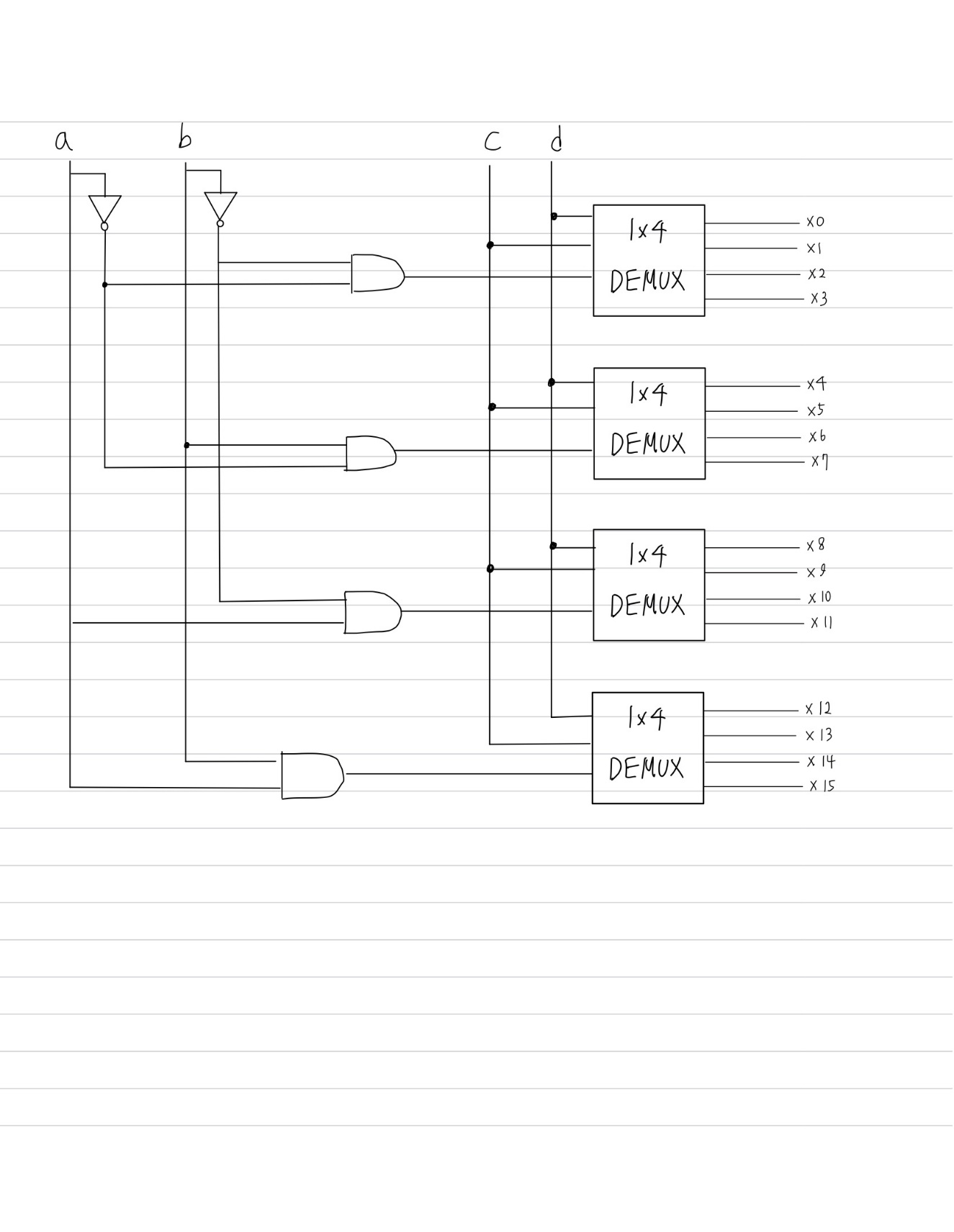
0000 0001 0010 0011

0100 0101 0110 0111

1000 1001 1010 1011

1100 1101 1110 1111

입력 형태를 4개씩 나누어서 볼 때, 맨 뒤의 두 자리는 항상 00, 01, 10, 11 순서로 반복된다. 앞의 두자리는 00 4개, 01 4개, 10 4개, 11 4개가 순서대로 등장한다. 이 사실을 이용하여 4 to 16 Decoder를 다음과 같이 설계하였다.



4개의 입력을 a, b, c, d라고 할 때, c와 d를 Demux의 선택자로 입력하면, 각 입력이 들어왔을 때 알맞은 출력이 1이 되도록 할 수 있다. cd가 00일때는 각 Demux의 첫번째 출력이 활성화되고(1이 되고), 01일때는 두번째, 10일때는 세번째, 11일때는 네번째 출력이 활성화된다.

또한, 1to4 Demux는 입력이 1일 경우 선택자에 따라 출력이 선택되고, 입력이 0이면 출력이 모두 0이 된다. 따라서, not gate와 and gate를 이용하여 각각의 Demux에 입력값 1개를 넣어서, ab가 00일 때는 첫번째 Demux, 01일 때는 두번째 Demux, 10일 때는 세번째 Demux, 11일 때는 네번째 Demux에서 하나의 출력이 1이 되도록 만들었다. 예를 들어, ab가 00일 때는 첫 번째의 Demux의 입력값만 1이 되고 나머지 Demux의 입력값은 0이 된다. 따라서 선택자에 의해 선택된 첫번째의 Demux의 출력 중 하나가 1이 된다. 나머지 세 개의 Demux에서 입력은 0이므로 선택자에 관계없이 4개의 출력이 모두 0이 된다.

1x4 line DEMUX를 이용하여 4x16 Decoder를 만든 design source는 다음과 같다. c, d를 Demux의 선택자로 사용하였고, 출력을 4개씩 묶어 순서대로 ~a&~b, ~a&b, a&~b, a&b를 Demux의 입력으로 사용하였다. x0, x1, x2, x3은 ~a&~b를 입력으로 받는 첫 번째 demux의 출력에 해당한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

simulation source는 다음과 같다. 모든 입력의 값이 0으로 시작해, 각각 160, 80, 40, 20 시간단위마다 a, b, c, d의 값을 부정하여 모든 입력 형태를 확인할 수 있도록 하였다. 320 시간단위가 지나면 시뮬레이션이 종료되도록 하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

시뮬레이션 결과는 다음과 같다. 16개의 입력 형태에 따라 x0부터 x15 중 하나의 출력만이 정상적으로 활성화되는 것을 확인할 수 있다.



**9.**

AND gate로 구성한 2x4 Decoder에서는 2개의 1bit수 a, b를 입력 받아, 입력 형태에 따라 d0, d1, d2, d3 중 하나의 출력만이 1이 되고 나머지 출력들은 0이 되는 것을 확인할 수 있었다. NAND gate로 구성한 2x4 Decoder에서는 마찬가지로 a, b를 입력 받아, 입력 형태에 따라 d0, d1, d2, d3 중 하나의 출력만이 0이 되고 나머지 출력들은 1이 되는 것을 확인할 수 있었다.

4x2 Encoder에서는 d0, d1, d2, d3을 입력 받아, 네 가지 입력 중 어떤 입력이 1인가에 따라 출력 a, b 값이 다르게 나타났다. Decoder와 반대되는 기능을 하였다. Encoder에 d0, d1, d2, d3을 입력해 나온 출력 a, b를 Decoder의 입력으로 나오면, 본래의 d0, d1, d2, d3을 구할 수 있다.

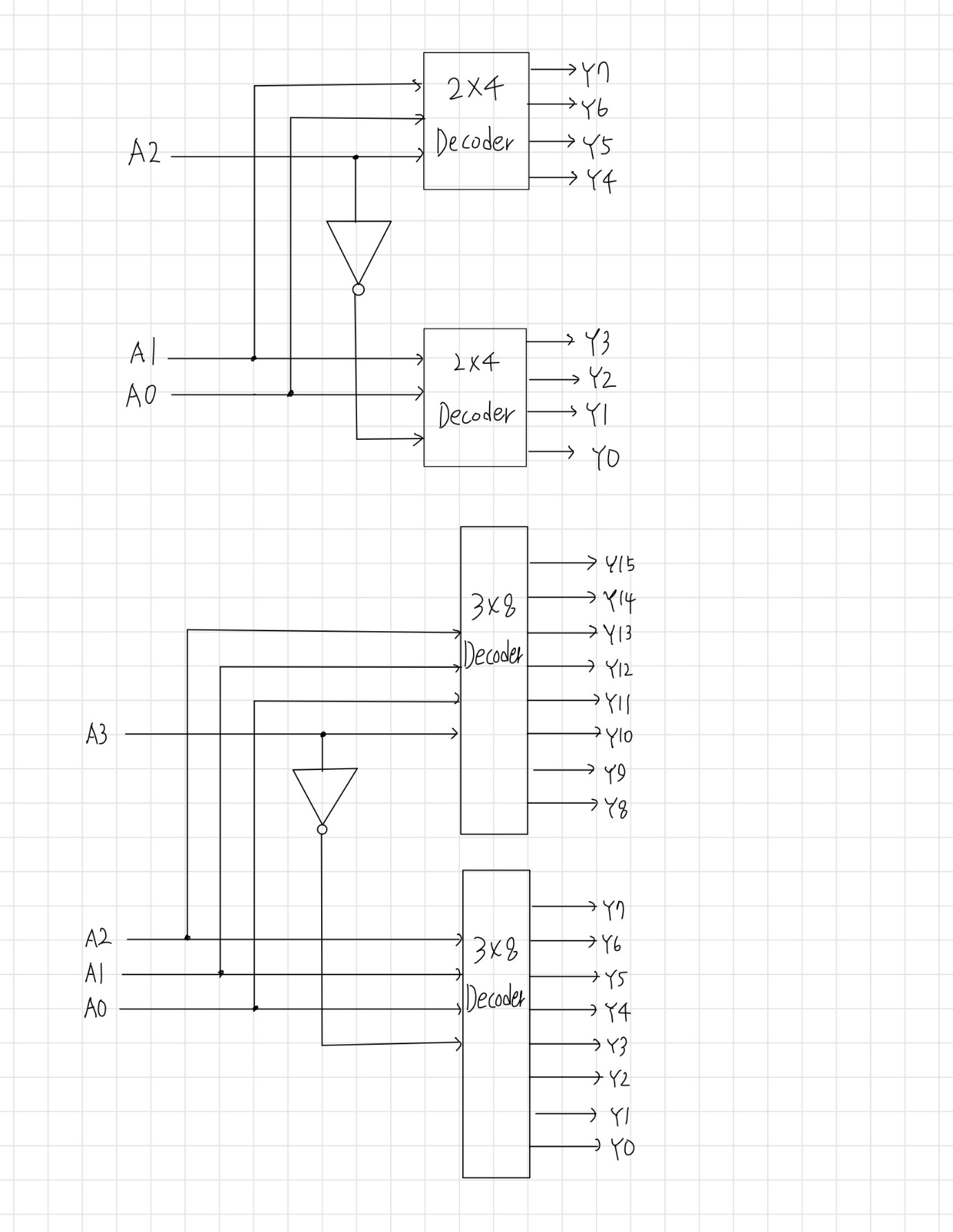
BCD to Decimal Decoder에서는 BCD의 각 자리를 의미하는 a, b, c, d를 입력으로 받아, 1부터 9까지의 Decimal number를 의미하는 x1, x2, x3, x4, x5, x6, x7, x8, x9를 출력하였다. BCD로 0001을 입력하면, 십진수로 1을 의미하는 x1이 활성화되었다. 0111을 입력하면, 8을 의미하는 x8이 활성화되었다.

4x1 line MUX는 선택자를 의미하는 c1, c0의 값에 따라, 입력 x0, x1, x2, x3 중 하나를 선택해 출력 y의 값으로 출력하였다. 예를 들어, c1, c0이 0, 1이고 x2가 0이면, y의 값을 0으로 출력하였다. c1, c0이 1, 1이고 x3의 값이 1이면 y값으로 1을 출력하였다.

1x4 line DEMUX에서는 선택자를 의미하는 c1, c0의 값에 따라, 입력 x의 값이 출력 y0, y1, y2, y3 중 하나에서 출력되었다. 즉, x가 0인 경우에는 선택자에 상관없이 모든 출력이 0이 되었다. 예를 들어, c1, c0이 0, 0이고 x가 0이면 y0의 값으로 1이 출력되고, 나머지 출력들은 0이 되었다. c1, c0이 0, 0이고 x가 0이면, y0의 값으로 x의 값인 0이 출력되고, 나머지 출력들도 0이 되므로 결과적으로 모든 출력이 0이 되었다.

**10.**

다음과 같이 2 to 4 Decoder 2개를 사용해 3 to 8 Decoder를 만들 수 있다.



다음과 같이 3 to 8 Decoder 2개를 사용해 4 to 16 Decoder를 만들 수 있다.

