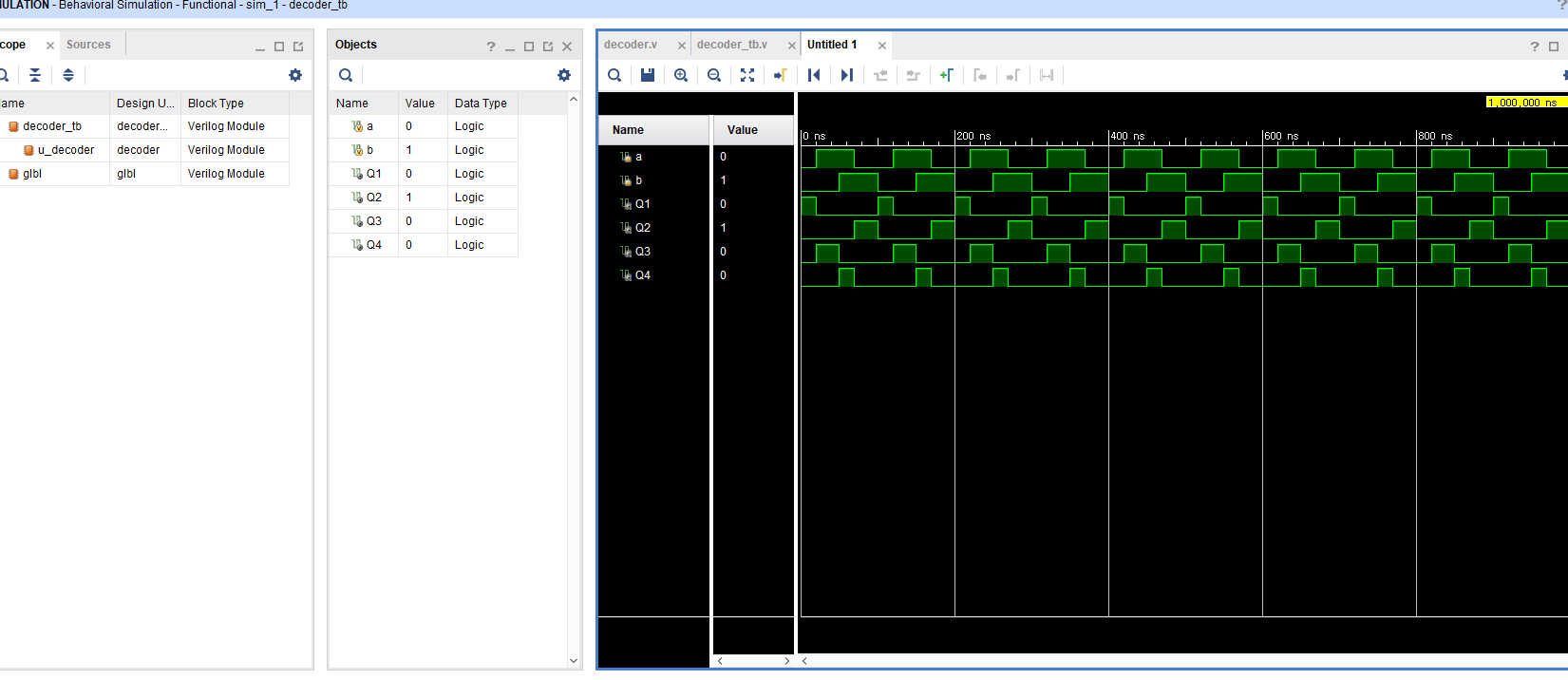
9주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20221549 이름: 김효림

**1.**

.................



2 to 4 Decoder의 simulation 결과(active high)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input A | Input B | Output Q1 | Output Q2 | Output Q3 | Output Q4 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

2 to 4 Decoder의 진리표 결과(active high)

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

2 to 4 Decoder의 카르노맵 결과(active high)

스크린샷, 소프트웨어, 텍스트, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

2 to 4 Decoder의 simulation(active low)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input A | Input B | Output Q1 | Output Q2 | Output Q3 | Output Q4 |
| 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 to 4 Decoder의 진리표(active low)

텍스트, 폰트, 친필, 라인이(가) 표시된 사진

자동 생성된 설명

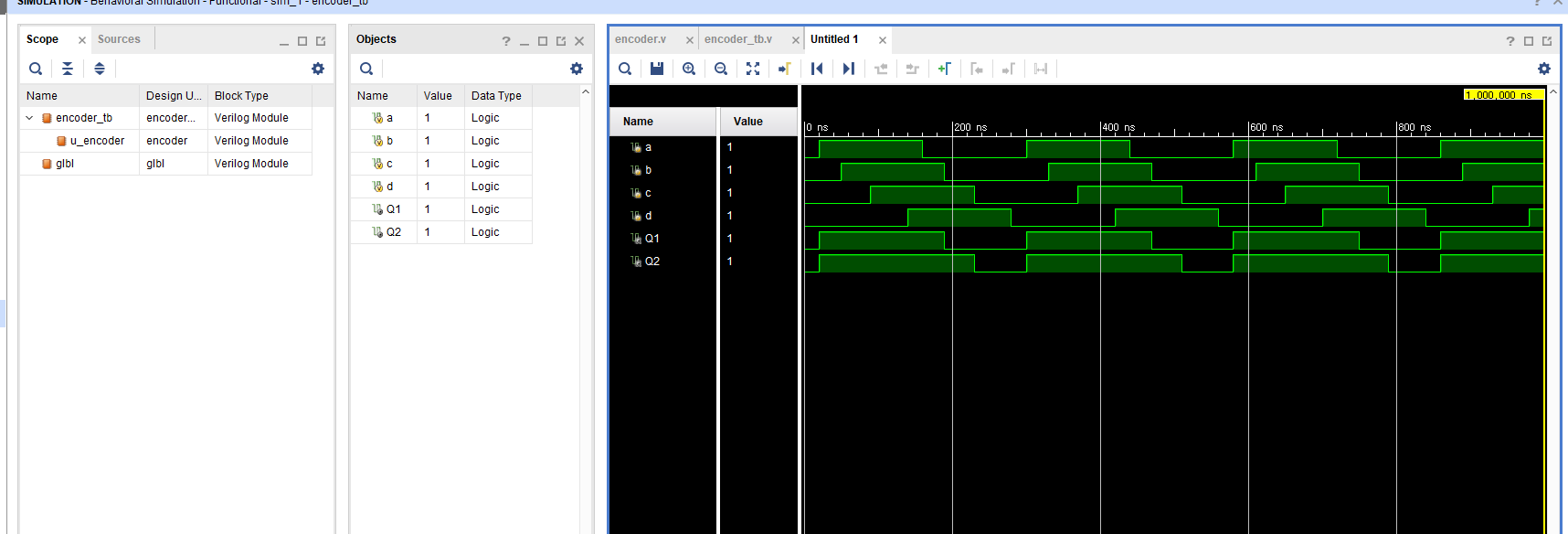
2 to 4 decoder의 카르노맵 결과(active low)

Active high의 경우와 active low는 진리표의 값이 반대인 것을 알 수 있다. Active low의 output 식은 Active high의 output 식에 not을 붙인 NAND 형태임을 확인할 수 있다.

................

**2.**

.......................



4 to 2 encoder의 simulation 결과

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output Q1 | Output Q2 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

4 to 2 Encoder의 진리표

텍스트, 번호, 폰트, 친필이(가) 표시된 사진

자동 생성된 설명

4 to 2 Encoder의 카르노맵 결과

0001, 0010, 0100, 1000을 제외한 나머지 입력에 대해서는 don’t care로 처리됨을 확인할 수 있다. 각 입력에 따라 차례로 00, 01, 10, 11을 결과값으로 갖는 것을 확인할 수 있다.

.........................

**3.**

.......................

Encoder의 입력은 여러 개의 입력 중 단 한 개의 bit만 1이나 0의 값을 가지는 배타성을 가진다. 이러한 형태를 가지지 않는 나머지 입력의 경우 처리는 don’t care로 된다. 즉, 적절한 입력 형식이 아닌 오류로 받아들여지게 되는 것이다.

.........................

**4.**

.......................

3에서 발생한 문제는, 입력에 우선순위를 줄 수 있다면 오류가 아닌 정상적인 입력값으로 처리할 수 있다. 이는 priority encoder를 통해 구현 가능하다.

진리표는 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input A | Input B | Input C | Input D | Output a | Output b | d |
| 0 | 0 | 0 | 0 | X | X | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | X | 0 | 1 | 1 |
| 0 | 1 | X | X | 1 | 0 | 1 |
| 1 | X | X | X | 1 | 1 | 1 |

이를 통해 output식은 다음과 같이 구할 수 있다.

|  |
| --- |
| module priority(  input a,b,c,d,  output A,B,D  );    assign A = a|b;  assign B = a | (~b&c);  assign D = a|b|c|d;    endmodule |

이를 코드로 구현하면 이와 같으며, simulation을 돌리면 다음과 같은 결과를 얻을 수 있다.

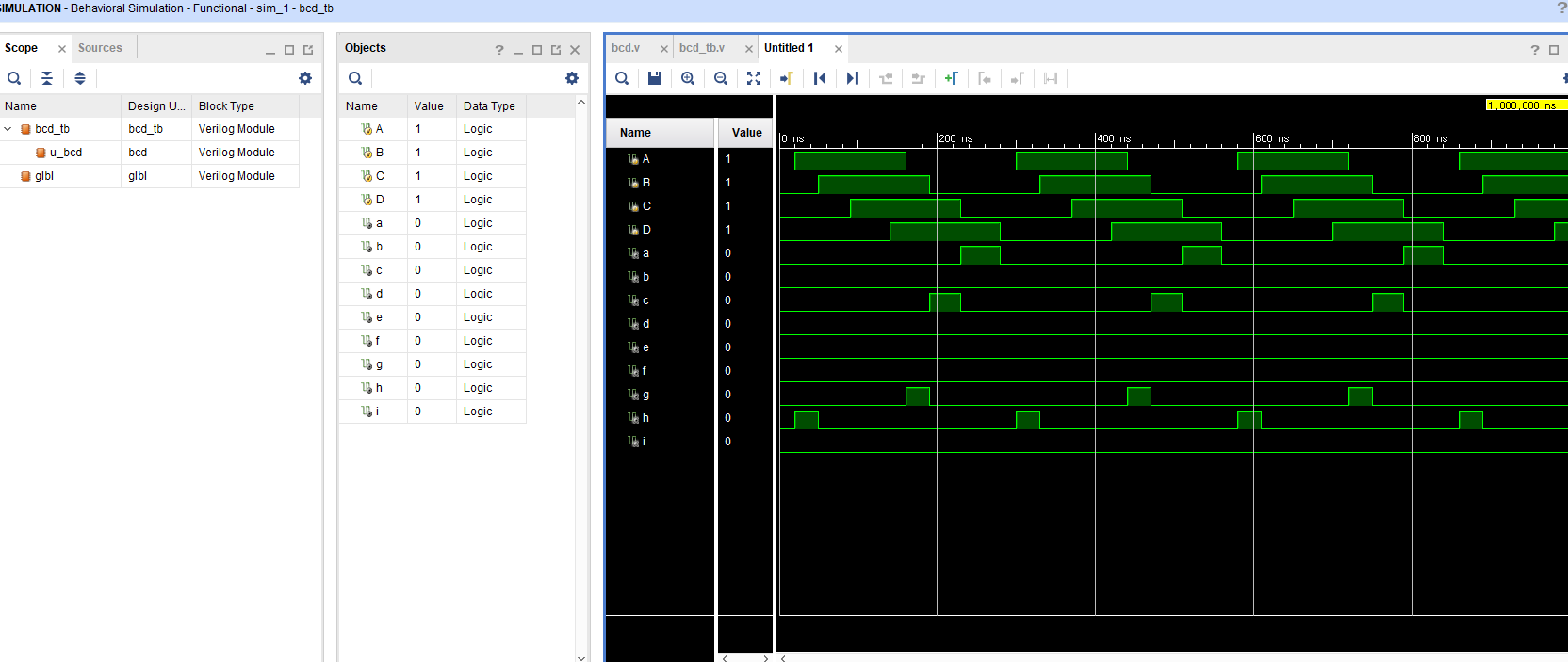
소프트웨어, 멀티미디어 소프트웨어, 컴퓨터 아이콘, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

.........................

**5.**

.......................



BCD to Decimal simulation 결과

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out a | Out b | Out c | Out d | Out e | Out f | Out g | Out h | Out i | Out j |
| 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 |

BCD to decimal의 진리표

텍스트, 번호, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 번호, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

BCD to Decimal의 카르노맵 결과

BCD 코드를 입력 값으로 받아 그에 대응되는 십진수 값으로 변환하여 출력하는 것을 확인할 수 있다. Output 비트 중 하나의 bit만 1로 set되는 것을 알 수 있다.

.........................

**6.**

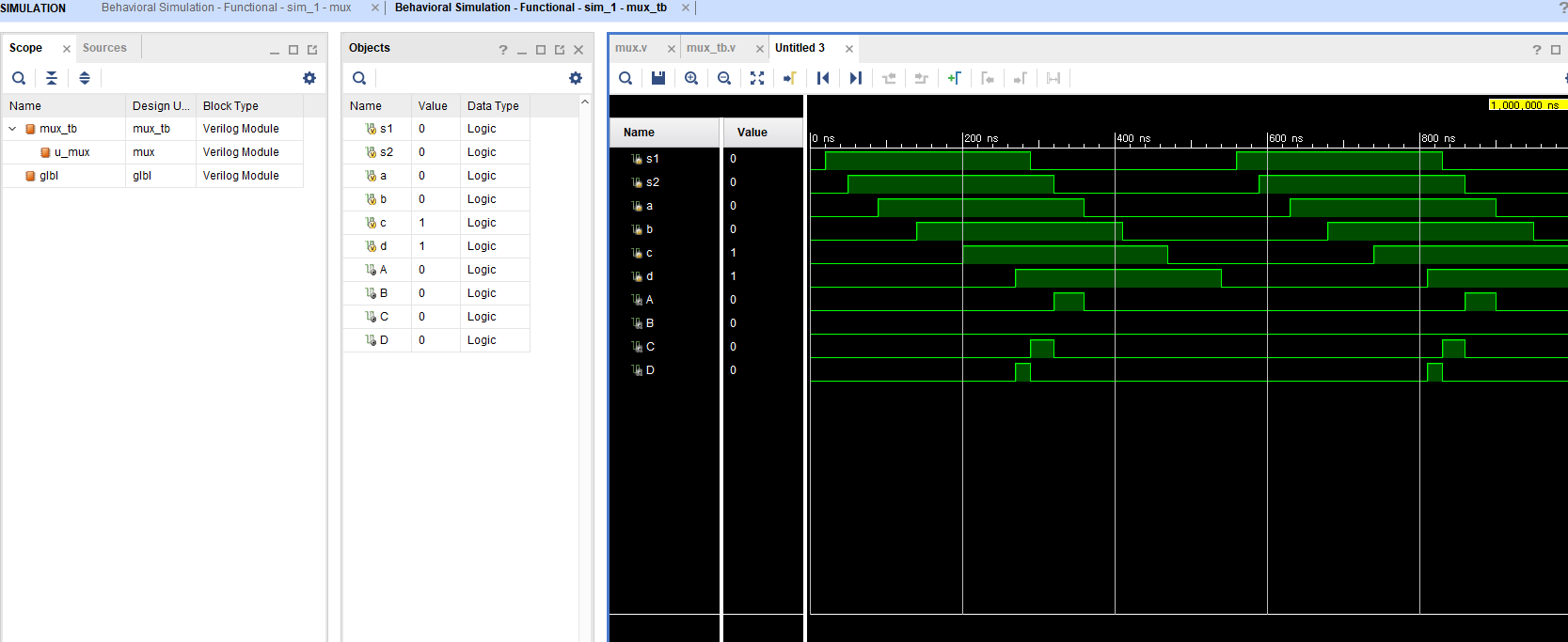
.......................

인코더와 디코더는 입력 형태와 출력 형태가 다른 경우 사용된다. 디코더는 컴퓨터 내부에서 2진수로 코드화된 데이터를 해독하는 역할을 하고, 인코더는 여러 개의 입력 단자 중 어느 하나의 단자에 나타난 정보를 여러 자리의 2진수로 코드화하여 전달한다. 따라서 인코더와 디코더는 보안이나 데이터 전달 및 전송 분야에 주로 활용된다.

.........................

**7.**

.......................



MUX simulation 결과

|  |  |
| --- | --- |
| module mux(  input s1, s2, a, b, c, d,  output A, B, C, D  );    assign A = (~s1 & ~s2 & a);  assign B = (s1 & ~s2 & b);  assign C = (~s1 & s2 & c);  assign D = (s1 & s2 & d);  endmodule | module mux\_tb;  reg s1,s2,a,b,c,d;  wire A,B,C,D;  mux u\_mux (  .s1(s1),  .s2(s2),  .a(a),  .b(b),  .c(c),  .d(d),  .A(A),  .B(B),  .C(C),  .D(D)  );  initial begin  s1 = 1'b0;  s2 = 1'b0;  a = 1'b0;  b = 1'b0;  c = 1'b0;  d = 1'b0;  end  always begin  s1 = #20 ~s1;  s2 = #30 ~s2;  a = #40 ~a;  b = #50 ~b;  c = #60 ~c;  d = #70 ~d;  end  initial begin  #1000  $finish;  end  endmodule |

4 to 1 line MUX의 코드

스크린샷, 번호, 텍스트이(가) 표시된 사진

자동 생성된 설명

4 to 1 line MUX의 진리표

Select signal 두 개에 따라 입력값이 대응되는 것을 확인할 수 있다.

.........................

**8.**

.......................

4 to 16 decoder의 진리표는 다음과 같다.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out a | Out b | Out c | Out d | Out e | Out f | Out g | Out h | Out i | Out j | Out k | Out l | Out m | Out n | Out o | Out p |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 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 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 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 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

1 to 4 line demux의 진리표

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

Demux의 입력 데이터를 1로 고정한다면 decoder와 같은 작업을 수행하게 된다.

이를 조합하여 1 to 4 line demux를 이용한 4 to 16 decoder를 구현하면 다음과 같은 source 코드를 생각할 수 있다.

|  |
| --- |
| module sixt(  input a,b,c,d,  output A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P  );  assign A = ~a&~b&~c&~d&1;  assign B = ~a&~b&~c&d&1;  assign C = ~a&~b&c&~d&1;  assign D = ~a&~b&c&d&1;  assign E = ~a&b&~c&~d&1;  assign F = ~a&b&~c&d&1;  assign G = ~a&b&c&~d&1;  assign H = ~a&b&c&d&1;  assign I = a&~b&~c&~d&1;  assign J = a&~b&~c&d&1;  assign K = a&~b&c&~d&1;  assign L = a&~b&c&d&1;  assign M = a&b&~c&~d&1;  assign N = a&b&~c&d&1;  assign O = a&b&c&~d&1;  assign P = a&b&c&d&1;    endmodule |

.........................

**9.**

.......................

디코더와 인코더를 구현하며 디멀티플렉서, 멀티플렉서와의 관계 및 개념에 대해 공부할 수 있었다. 실습 외에 추가적으로 priority encoder 및 디멀티플렉서를 이용한 4 to decoder를 만들며 정형적이고 기본적인 회로가 가지는 문제점을 해결할 수 있는 방식에 대해 한 번 더 고민해 보게 되었다.

.........................

**10.**

.......................

1 to 8 demultiplexer의 대표적인 예로는 74LS138이 있으며, 이는 핀 A0, A1, A2의 상태에 따라 Enable 입력들 중 하나를 8개의 출력핀 중 한 개로 선택하는 논리 회로이다.

도표, 라인, 스크린샷, 기술 도면이(가) 표시된 사진

자동 생성된 설명

.........................