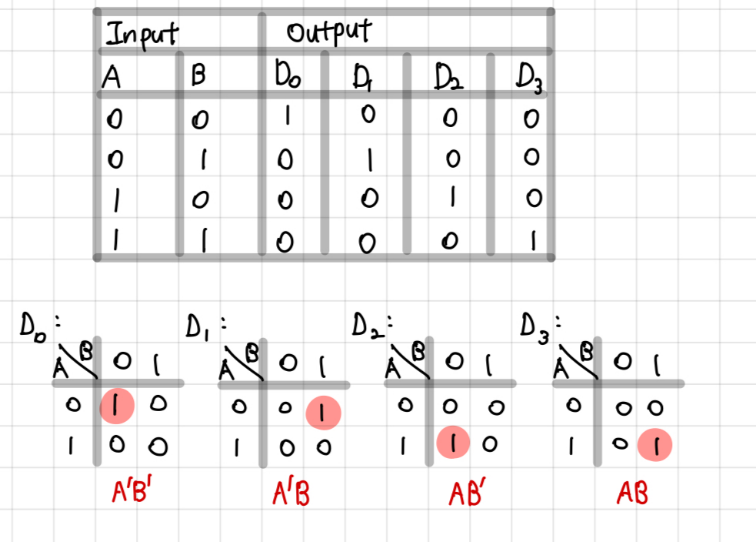
9주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231609 이름: 정희선

**1.**

1.1) active high

- truth table & k-map



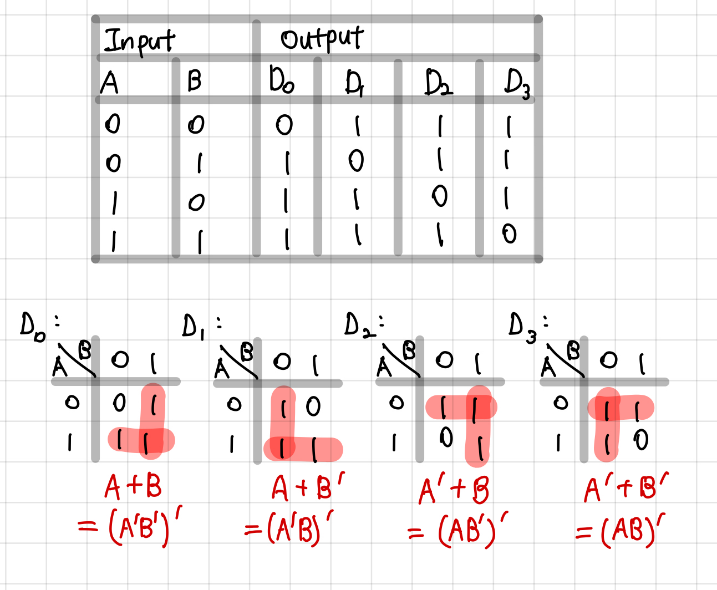
- verilog using AND & simulation

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input A,*  *input B,*  *output D0,*  *output D1,*  *output D2,*  *output D3*    *);*  *assign D0 = (~A) & (~B);*  *assign D1 = (~A) & (B);*  *assign D2 = (A) & (~B);*  *assign D3 = (A) & (B);*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg A,B;*  *wire D0, D1, D2, D3;*  *boolean u\_boolean (*  *.A(A),*  *.B(B),*  *.D0(D0),*  *.D1(D1),*  *.D2(D2),*  *.D3(D3)*  *);*  *initial begin*  *A = 1'b0;*  *B = 1'b0;*  *end*  *always@(A or B) begin*  *A <= #10 ~A;*  *B <= #20 ~B;*  *end*  *initial begin*  *#75*  *$finish;*  *end*  *endmodule* |

**

1.2) active low

- truth table & k-map

****

- verilog using NAND & simulation

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input A,*  *input B,*  *output D0,*  *output D1,*  *output D2,*  *output D3*    *);*  *assign D0 = ~((~(A & A)) & (~(B & B)));*  *assign D1 = ~((~(A & A)) & B);*  *assign D2 = ~(A & (~(B & B)));*  *assign D3 = ~(A & B);*  *assign D0 = ~((~A) & (~B));*  *assign D1 = ~((~A) & (B));*  *assign D2 = ~((A) & (~B));*  *assign D3 = ~((A) & (B));*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg A,B;*  *wire D0, D1, D2, D3;*  *boolean u\_boolean (*  *.A(A),*  *.B(B),*  *.D0(D0),*  *.D1(D1),*  *.D2(D2),*  *.D3(D3)*  *);*  *initial begin*  *A = 1'b0;*  *B = 1'b0;*  *end*  *always@(A or B) begin*  *A <= #10 ~A;*  *B <= #20 ~B;*  *end*  *initial begin*  *#75*  *$finish;*  *end*  *endmodule* |

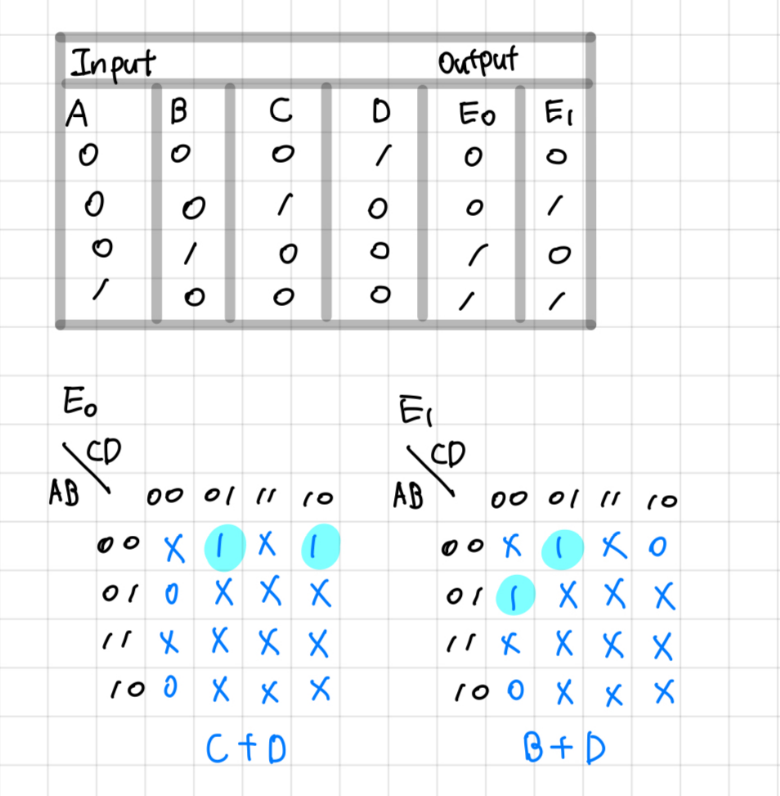
**소프트웨어, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 편집이(가) 표시된 사진

자동 생성된 설명**

2-to-4 디코더(Decoder)는 두 개의 입력을 받아 네 개의 출력 중 하나를 활성화하는 회로이다. 이 회로는 특정 출력이 1이 될 때 활성화되는 Active High 형태와, 특정 출력이 0이 될 때 활성화되는 Active Low 형태로 나눌 수 있다. Active High 형태의 디코더는 AND 게이트를 사용하여 구현하며, 출력 중 하나만 1이 되는 방식으로 특정 출력이 선택되었음을 나타낸다. 입력 A와 B의 조합에 따라 D0에서 D3 중 하나의 출력만 1이 되며 나머지 출력은 0이 된다. 반면, Active Low 형태의 디코더는 NAND 게이트를 사용하여 구현하며, 출력 중 하나만 0이 되는 방식으로 특정 출력이 선택되었음을 나타낸다. A와 B의 조합에 따라 D0에서 D3 중 하나의 출력만 0이 되며 나머지 출력은 모두 1이 된다. 예를 들어, A=0, B=0일 때 D0은 0이 되고, D1, D2, D3은 모두 1이 된다. Active Low 형태의 디코더는 Active Low 신호가 필요한 경우에 적합하며, NAND 게이트로 설계되어 있어 회로가 상대적으로 복잡할 수 있다. Active High AND 디코더와 Active Low NAND 디코더의 차이는 출력이 1로 활성화되는지, 0으로 활성화되는지에 있다. Active High 형태는 특정 출력이 1이 될 때 선택된 것으로 간주하며, 회로가 단순하고 직관적이다. 반면, Active Low 형태는 특정 출력이 0이 될 때 선택된 것으로 간주하여 Active Low 신호를 요구하는 시스템에서 유용하다.

**2.**

- truth table & k-map

****

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input A,*  *input B,*  *input C,*  *input D,*  *output E0,*  *output E1*    *);*  *assign E0 = (C) | (D);*  *assign E1 = (B) | (D);*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg A,B,C,D;*  *wire E0, E1;*  *boolean u\_boolean (*  *.A(A),*  *.B(B),*  *.C(C),*  *.D(D),*  *.E0(E0),*  *.E1(E1)*  *);*  *initial begin*  *A = 1'b0;*  *B = 1'b0;*  *C = 1'b0;*  *D = 1'b0;*  *end*  *always@(A or B or C or D) begin*  *A <= #10 ~A;*  *B <= #20 ~B;*  *C <= #30 ~C;*  *D <= #40 ~D;*  *end*  *initial begin*  *#200*  *$finish;*  *end*  *endmodule* |

****

4-to-2 인코더는 네 개의 입력을 받아 이를 2비트의 출력으로 인코딩하는 장치이다. 이 인코더는 입력 A, B, C, D 중 하나의 입력이 활성화되었을 때 그 입력의 위치를 2비트 출력 E1​과 E0​에 맞추어 나타낸다.

**3.**

4-to-2 인코더의 4-to-2 인코더에서 입력 형태 네 가지(1000, 0100, 0010, 0001)를 제외한 나머지 입력 형태는 동시에 여러 입력이 활성화된 경우를 의미한다. 일반적인 4-to-2 인코더는 입력 중 하나만 활성화되었다고 가정하여 작동하지만, 실제 사용 환경에서는 둘 이상의 입력이 활성화될 수 있다. 이러한 경우 4-to-2 인코더는 단일 활성화 입력만을 인코딩할 수 없으므로 정의되지 않은 출력 상태가 발생하거나 불안정한 출력을 생성할 가능성이 있다. 따라서 여러 입력이 동시에 활성화될 때 인코더가 올바르게 작동하기 위해서는 우선순위를 정해 각 입력에 우선순위를 할당하는 우선순위(priority) 인코더를 사용하게 된다.

**4.**

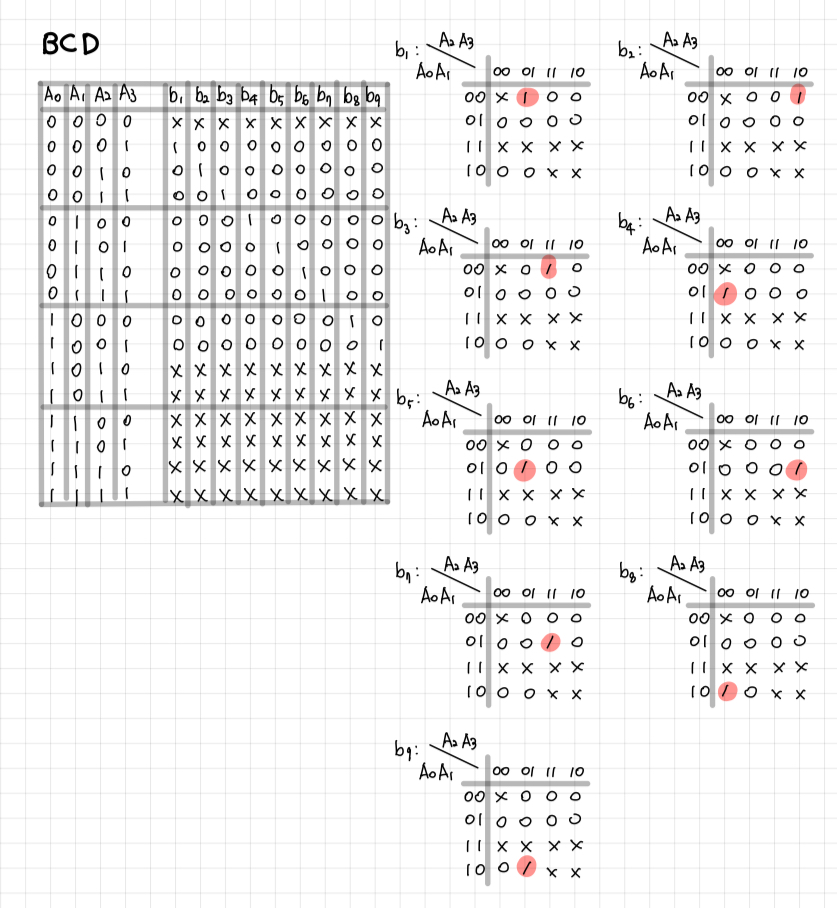
4-to-2 인코더의 네 가지 입력 형태(1000, 0100, 0010, 0001)를 제외한 나머지 16가지 입력 조합에 대해 작동되는 우선순위 인코더(priority encoder)는 동시에 여러 입력이 활성화되었을 때 우선순위가 가장 높은 입력을 기준으로 출력을 생성한다. 우선순위 인코더는 각 입력에 우선순위를 부여하여, 예를 들어 A가 가장 높은 우선순위, D가 가장 낮은 우선순위를 갖는 구조로 동작하도록 구성할 수 있다.

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input A,*  *input B,*  *input C,*  *input D,*  *output E0,*  *output E1,*  *output Valid*  *);*  *assign E0 = D | (~D & C) | (~D & ~C & B);*  *assign E1 = D | (~D & ~C & B);*  *assign Valid = A | B | C | D;*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg A,B,C,D;*  *wire E0,E1,Valid;*  *boolean u\_boolean(*  *.A(A),*  *.B(B),*  *.C(C),*  *.D(D),*  *.E0(E0),*  *.E1(E1),*  *.Valid(Valid)*  *);*  *initial begin*  *A = 1'b0;*  *B = 1'b0;*  *C = 1'b0;*  *D = 1'b0;*  *end*  *always@(A or B or C or D) begin*  *A <= #10 ~A;*  *B <= #20 ~B;*  *C <= #30 ~C;*  *D <= #40 ~D;*  *end*  *initial begin*  *#500*  *$finish;*  *end*  *endmodule* |

스크린샷, 컴퓨터, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**5.**



|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input A0,*  *input A1,*  *input A2,*  *input A3,*  *output b1,*  *output b2,*  *output b3,*  *output b4,*  *output b5,*  *output b6,*  *output b7,*  *output b8,*  *output b9*  *);*  *assign b1 = (A0)&(~A1)&(~A2)&(~A3);*  *assign b2 = (~A0)&(A1)&(~A2)&(~A3);*  *assign b3 = (A0)&(A1)&(~A2)&(~A3);*  *assign b4 = (~A0)&(~A1)&(A2)&(~A3);*  *assign b5 = (A0)&(~A1)&(A2)&(~A3);*  *assign b6 = (~A0)&(A1)&(A2)&(~A3);*  *assign b7 = (A0)&(A1)&(A2)&(~A3);*  *assign b8 = (~A0)&(~A1)&(~A2)&(A3);*  *assign b9 = (A0)&(~A1)&(~A2)&(A3);*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg A0,A1,A2,A3;*  *wire b1, b2, b3, b4, b5, b6, b7, b8, b9;*  *boolean u\_boolean (*  *.A0(A0),*  *.A1(A1),*  *.A2(A2),*  *.A3(A3),*  *.b1(b1),*  *.b2(b2),*  *.b3(b3),*  *.b4(b4),*  *.b5(b5),*  *.b6(b6),*  *.b7(b7),*  *.b8(b8),*  *.b9(b9)*  *);*  *initial begin*  *A0 = 1'b0;*  *A1 = 1'b0;*  *A2 = 1'b0;*  *A3 = 1'b0;*  *end*  *always@(A0 or A1 or A2 or A3) begin*  *A0 <= #10 ~A0;*  *A1 <= #20 ~A1;*  *A2 <= #30 ~A2;*  *A3 <= #40 ~A3;*  *end*  *initial begin*  *#250*  *$finish;*  *end*  *endmodule* |

스크린샷, 소프트웨어, 멀티미디어 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

BCD (Binary-Coded Decimal) to Decimal Decoder는 4비트의 BCD 입력을 받아 이를 10진수 형태의 출력으로 변환하는 디지털 회로이다. BCD는 4비트로 0부터 9까지의 십진수를 이진수로 표현하는 방식이다. 이 디코더의 기능을 구현하기 위해 주로 4개의 입력 비트(A3, A2, A1, A0)를 사용하며, 10개의 출력(0 ~ 9)을 제공한다. 각 출력은 특정 입력 조합에만 활성화되며, 이로써 입력된 BCD 코드에 해당하는 십진수가 출력된다. 이론적 Boolean 함수는 각 출력에 대한 특정 BCD 조합을 정의하는 논리 식으로 나타낼 수 있다. 예를 들어, 0번 출력이 활성화되려면 입력 A3, A2, A1, A0이 모두 0이어야 하므로 D0 = ~A3 & ~A2 & ~A1 & ~A0와 같은 논리식을 갖는다. 각 출력은 BCD 조합에 맞는 논리식으로 정의된다. 시뮬레이션을 수행하여, 결과가 기대한 출력과 동일한지 확인한다. 이로써 회로가 정확히 동작하는지 검증할 수 있다.

**6.**

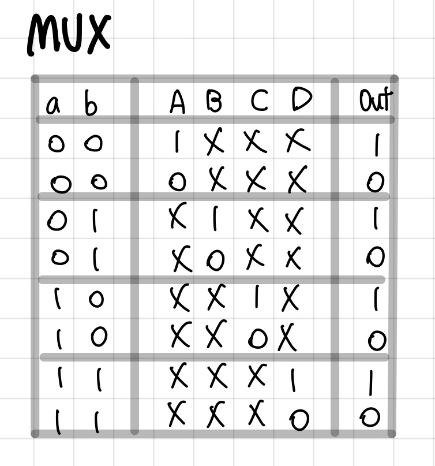
Encoder와 Decoder는 디지털 시스템에서 중요한 역할을 하며, 다양한 응용 분야에서 널리 사용된다.

Encoder의 대표적인 응용은 키보드 스캔과 데이터 압축이다. 키보드 스캔에서는 각 키의 위치에 해당하는 코드를 생성하여 컴퓨터가 어떤 키가 눌렸는지 알 수 있도록 한다. 데이터 압축에서는 다중 신호를 하나의 신호로 변환하여 데이터 전송 효율을 높인다. 또한, 우선순위 인코더는 여러 신호 중에서 우선순위가 가장 높은 신호를 감지하여 특정 이벤트의 발생을 알려주는 데 사용된다. 예를 들어, 인터럽트 처리 시스템에서 어떤 장치가 인터럽트를 요청할 때, 우선순위 인코더는 가장 중요한 인터럽트를 우선적으로 처리하도록 도와준다.

Decoder의 대표적인 응용으로 메모리 주소 지정과 7-세그먼트 디스플레이 구동이 있다. 메모리 주소 지정에서는 Decoder가 특정 주소 입력에 따라 메모리의 해당 영역을 활성화하여 데이터를 읽거나 쓸 수 있게 한다. 7-세그먼트 디스플레이에서는 Decoder가 숫자나 문자의 이진 코드 입력을 받아 특정 세그먼트를 활성화하여 원하는 형태로 표시되도록 한다. 또한, 다중화 회로에서 입력 신호의 특정 선택을 통해 데이터를 특정 출력으로 연결할 때도 Decoder가 사용된다.

**7.**

- truth table



* verilog

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input A,*  *input B,*  *input C,*  *input D,*  *input a,*  *input b,*  *output out*  *);*  *assign out = (A&~a&~b) | (B&~a&b) | (C&a&~b) | (D&a&b);*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg A,B,C,D,a,b;*  *wire out;*  *boolean u\_boolean (*  *.A(A),*  *.B(B),*  *.C(C),*  *.D(D),*  *.a(a),*  *.b(b),*  *.out(out)*  *);*  *initial begin*  *A = 1'b0;*  *B = 1'b0;*  *C = 1'b0;*  *D = 1'b0;*  *a = 1'b0;*  *b = 1'b0;*  *end*  *always@(A or B or C or D or a or b) begin*  *A <= #10 ~A;*  *B <= #20 ~B;*  *C <= #30 ~C;*  *D <= #40 ~D;*  *a <= #50 ~C;*  *b <= #60 ~D;*  *end*  *initial begin*  *#800*  *$finish;*  *end*  *endmodule* |

스크린샷, 컴퓨터, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

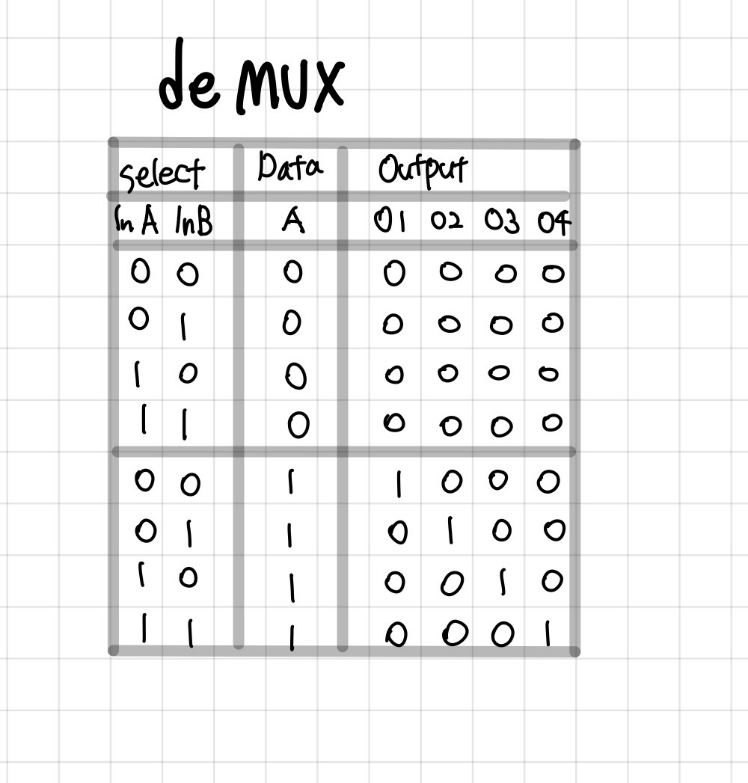
자동 생성된 설명

4 to 1 MUX(Multiplexer)는 여러 개의 입력 신호 중 하나를 선택하여 출력으로 전달하는 디지털 회로이다. MUX의 역할은 다수의 입력 중에서 특정 입력만을 선택하여 단일 출력으로 보내는 것으로, 이 과정을 통해 데이터 경로를 효율적으로 관리할 수 있다. 위 그림의 4 to 1 MUX는 네 개의 입력 신호(A, B, C, D)와 두 개의 선택 신호(a, b)를 가진다. 선택 신호(a, b)의 조합에 따라 MUX는 A, B, C, 또는 D 중 하나를 선택하여 출력으로 보낸다. 선택 신호가 00일 때 A가 출력되고, 01일 때 B가 출력되며, 10일 때 C가 출력되고, 11일 때 D가 출력된다. 이러한 개별 AND 게이트의 출력이 OR 게이트를 통해 결합되면 최종 출력이 선택 신호에 따라 결정된다. 따라서 전체 MUX의 출력은 OR 게이트의 출력으로, 선택된 하나의 입력이 단일 출력으로 전달된다. 이 회로가 실제 Boolean 함수와 일치하는지 확인하기 위해, 선택 신호의 모든 조합에 대해 진리표를 작성하고 각 조합에 대해 예상되는 출력이 실제 출력과 동일한지 검토할 수 있다. Verilog로 시뮬레이션을 통해 이 동작을 확인하고, FPGA를 이용하여 물리적 구현과 동작을 확인할 수도 있다.

**8.**

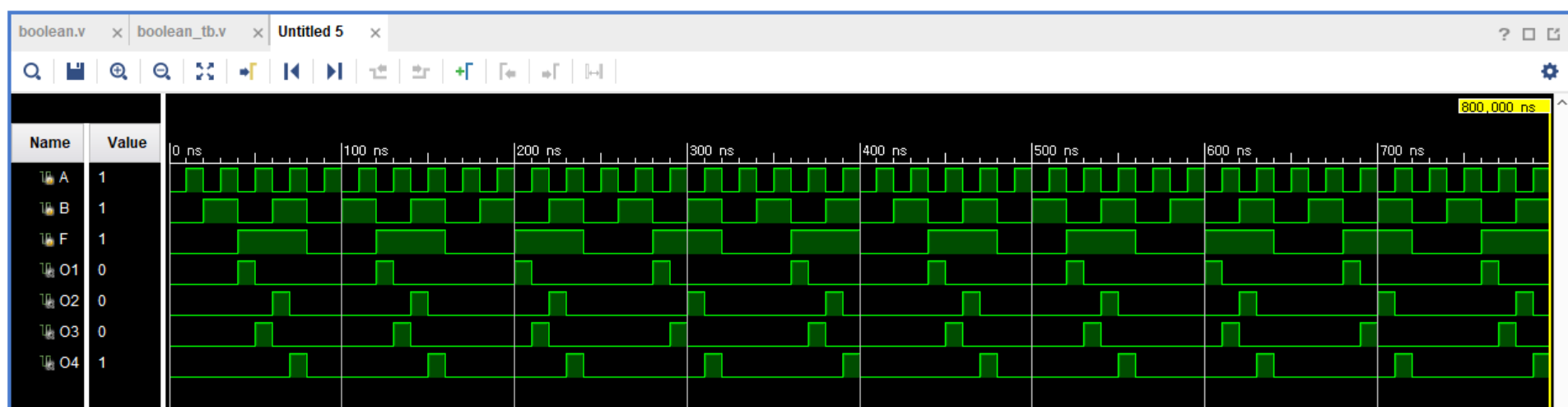
**8.1) 1 to 4 demux**

- truth table



* verilog

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input A,*  *input B,*  *input C,*  *input D,*  *input a,*  *input b,*  *output out*  *);*  *assign out = (A&~a&~b) | (B&~a&b) | (C&a&~b) | (D&a&b);*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg A,B,C,D,a,b;*  *wire out;*  *boolean u\_boolean (*  *.A(A),*  *.B(B),*  *.C(C),*  *.D(D),*  *.a(a),*  *.b(b),*  *.out(out)*  *);*  *initial begin*  *A = 1'b0;*  *B = 1'b0;*  *C = 1'b0;*  *D = 1'b0;*  *a = 1'b0;*  *b = 1'b0;*  *end*  *always@(A or B or C or D or a or b) begin*  *A <= #10 ~A;*  *B <= #20 ~B;*  *C <= #30 ~C;*  *D <= #40 ~D;*  *a <= #50 ~C;*  *b <= #60 ~D;*  *end*  *initial begin*  *#800*  *$finish;*  *end*  *endmodule* |



1 to 4 DeMUX는 하나의 입력 신호를 네 개의 출력 중 하나로 전달하는 회로이다. 이 회로는 선택 라인(InA, InB)에 의해 특정 출력이 활성화되며, 나머지 출력은 비활성화 상태를 유지한다. DeMUX는 Boolean 함수를 통해 정확하게 설명될 수 있으며, 각 출력의 논리 상태는 선택 라인과 입력 신호에 의해 결정된다. 선택 라인의 상태에 따라 입력 A가 해당 출력으로만 전달되도록 제어할 수 있다. 예를 들어, InA=0이고 InB=0인 경우에는 Out1만이 활성화되고, 나머지 출력들은 모두 0이 된다. 이는 입력 신호 A가 Out1로만 전달되도록 한다. 마찬가지로, InA=0, InB=1인 경우에는 Out2가 활성화되며, 나머지 출력은 비활성화된다. 이와 같은 방식으로 모든 출력은 특정 선택 라인 조합에 따라 활성화되거나 비활성화된다.

**8.2) 4 to 16 demux**

* verilog

위에서 사용한 1 to 4 line deMUX의 design source를 dmux라는 모듈로 선언하여 구현한다.

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input a0, a1, a2, a3,*  *output y0, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15*  *);*    *wire en0, en1, en2, en3;*  *dmux dm(a1, a0, 1, en0, en1, en2, en3);*  *dmux dm0(a3, a2, en0, y0, y1, y2, y3);*  *dmux dm1(a3, a2, en1, y4, y5, y6, y7);*  *dmux dm2(a3, a2, en2, y8, y9, y10, y11);*  *dmux dm3(a3, a2, en3, y12, y13, y14, y15);*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg [3:0] in;*  *wire y0, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15;*  *boolean u\_boolean(in[3], in[2], in[1], in[0], y0, y1, y2, y3, y4, y5, y6, y7, y8, y9, y10, y11, y12, y13, y14, y15);*  *initial begin*  *in[3] = 1'b0;*  *in[2] = 1'b0;*  *in[1] = 1'b0;*  *in[0] = 1'b0;*  *end*  *always in[3] = #400 ~in[3];*  *always in[2] = #200 ~in[2];*  *always in[1] = #100 ~in[1];*  *always in[0] = #50 ~in[0];*  *initial begin*  *#1000*  *$finish;*  *end*  *endmodule* |

스크린샷, 디스플레이, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**9.**

2 to 4 Decoder의 결과는 두 개의 입력(A, B)에 따라 네 개의 출력(D0, D1, D2, D3) 중 하나가 활성화되는지를 확인하는 것이었다. 이 디코더는 입력의 조합에 따라 정확히 하나의 출력만 '1'이 되고 나머지 출력이 '0'이 되어야 한다. 실험 결과, 이 회로는 올바르게 동작하였으며, 입력 조합에 맞게 정확한 출력이 활성화되었다. 논의할 사항으로는 디코더가 여러 입력 값을 디코딩하는 데 유용하지만, 여러 비트를 동시에 디코딩해야 하는 경우에는 다단 회로 구성이 필요할 수 있다는 점이다.

4 to 2 Encoder의 결과는 네 개의 입력 중 하나가 '1'일 때, 그 위치에 따라 두 개의 출력(E0, E1)이 이진수로 해당 위치를 나타내도록 구현되었다. 실험을 통해 이 회로는 정확히 기대한 대로 작동하였다. 특히, 4 to 2 Encoder는 입력의 우선순위가 중요하지 않은 단순한 경우에 적합하지만, 여러 입력이 동시에 활성화되는 경우에는 우선순위 인코더를 사용하는 것이 필요할 수 있다.

BCD to Decimal Decoder의 경우, 4비트 BCD 입력을 받아 10개의 디지털 출력 중 하나를 활성화하여 0에서 9까지의 10진수를 표시하는 것이 목적이었다. 실험 결과, BCD to Decimal Decoder는 기대한 대로 동작하며, 각 BCD 입력 값에 맞는 단일 출력만이 '1'로 활성화되었다. 논의할 사항으로는 BCD 입력이 10진수 범위를 벗어난 값일 때, 즉 10~15의 값이 입력될 때 출력이 활성화되지 않도록 해야 한다는 점이다. 이는 BCD Decoder에서 올바르지 않은 입력을 처리하는 방법을 추가해야 함을 시사한다.

4 to 1 MUX의 실험 결과는 두 개의 선택 라인(a, b)에 따라 네 개의 입력 중 하나만이 출력으로 전달되는지 확인하는 것이었다. MUX는 특정 조건에서 하나의 입력을 선택적으로 출력하는 회로로, 실험 결과 각 입력이 올바르게 출력으로 전달되는 것을 확인하였다. 논의할 사항으로는 MUX가 데이터 선택 및 경로 제어에 적합하지만, 선택 라인 수가 많아질수록 회로의 복잡성이 증가한다는 점이다. 따라서, 다중 입력을 다루어야 하는 경우 다단 MUX 구성을 고려할 필요가 있다.

1 to 4 DeMUX의 경우, 하나의 입력을 선택 라인에 따라 네 개의 출력 중 하나로 분배하는 회로로서, 실험 결과 기대한 대로 작동하였다. 선택 라인 조합에 맞게 지정된 출력만 활성화되고 나머지 출력은 비활성화되는 것을 확인하였다. 논의할 사항으로는 DeMUX가 데이터 분배에 매우 유용하지만, 여러 출력을 활성화해야 할 경우 추가적인 회로 구성이 필요하다는 점이다.

**10.**

Active High/Active Low: 디지털 회로에서 신호가 '1'일 때 활성화되는 경우를 Active High라고 하고, 신호가 '0'일 때 활성화되는 경우를 Active Low라고 한다. Decoder, MUX, DeMUX와 같은 회로에서 활성화 방식에 따라 회로의 동작이 달라질 수 있다. 이를 고려하여 설계해야 한다.

Tri-state Buffer: MUX와 DeMUX 같은 회로와 함께 사용되는 Tri-state Buffer는 회로의 출력이 '1', '0', 또는 고저항 상태(high-impedance, Z)로 될 수 있게 하는 장치이다. Tri-state Buffer의 동작 원리는 입력 신호와 함께 Enable(또는 Control) 신호를 받는 구조에 기반한다. Enable 신호에 따라 Tri-state Buffer의 출력은 세 가지 상태로 설정될 수 있다. 첫 번째로, Enable이 '1'일 때, Tri-state Buffer는 일반적인 버퍼처럼 동작하여 입력 신호를 출력으로 전달한다. 즉, 입력이 '1'이면 출력이 '1'이 되고, 입력이 '0'이면 출력이 '0'이 된다. 두 번째로, Enable이 '0'일 때, Tri-state Buffer는 고저항 상태(Z)로 전환된다. 이 상태에서는 출력이 회로에서 '분리'된 것처럼 작동하여, 다른 회로에 영향을 주지 않는다.