**7주차 결과보고서**

**전공: 생명과학과 학년: 4학년 학번: 20182186 이름: 김승원**

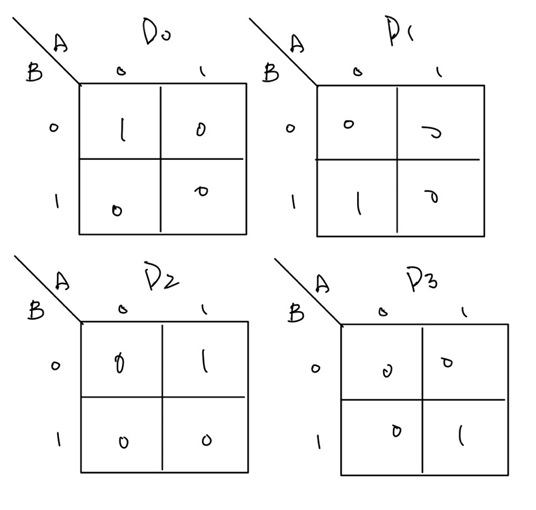
**1. 2 to 4 Decoder의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(Truth table 작성 및 k-map 포함[AND,NAND GATE])**

**<Table 1. 2 to 4 Decoder truth table>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | Output | | | |
| 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 |

2 to 4 Decoder의 진리표이다. Input A, B의 각각의 값에 맞도록 output이 출력되도록 하는데, Decoder의 정의에 따라 Output 들 중 한 Output만 출력되도록 하도록 한다.



**<Figure 1. 2 to 4 Decoder의 K-Map>**

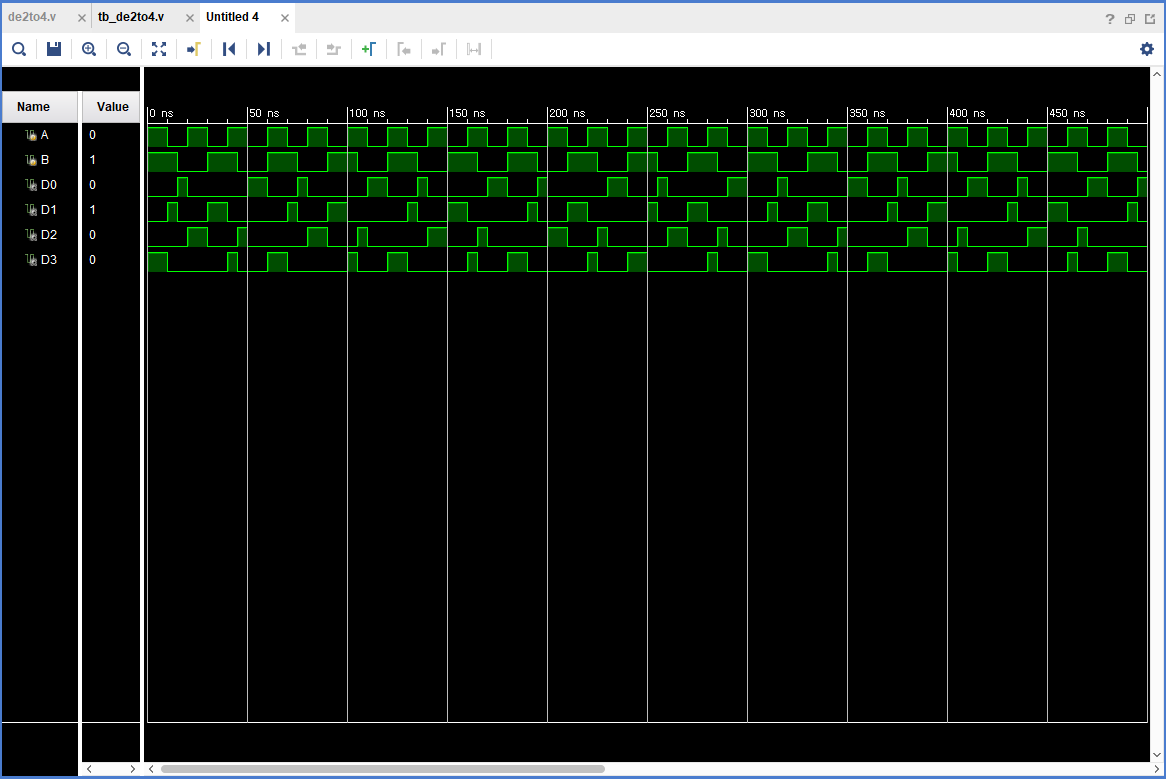
2 to 4 Decoder의 K-map이다. Input A, B에 맞게 D0, D1, D2, D3 차례대로 K-map을 그려서 할당해준다. 왼쪽 위부터 해서 오른쪽, 아래에 왼쪽, 오른쪽 차례대로 D0, D1, D2, D3를 나타내는 K-map이다.

D0 = A’B’

D1 = A’B

D2 = AB’

D3 = AB



**<Figure 2. 2 to 4 Decoder simulation 결과>**

2 to 4 Decoder의 simulation 결과이다. Input A, B의 값을 정해줌에 따라 적절한 Output이 출력되는 것을 확인했다. Decoder의 정의에 따라 Input을 입력했을 때, 하나의 선만 출력이 되는 것을 확인했다.

**2.   4 to 2 Encoder의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(Truth table 작성 및 k-map 포함)**

**<table 2. 4 to 2 Encoder의 truth table>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | | Output | |
| A | B | C | D | E0 | E1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

테이블이(가) 표시된 사진

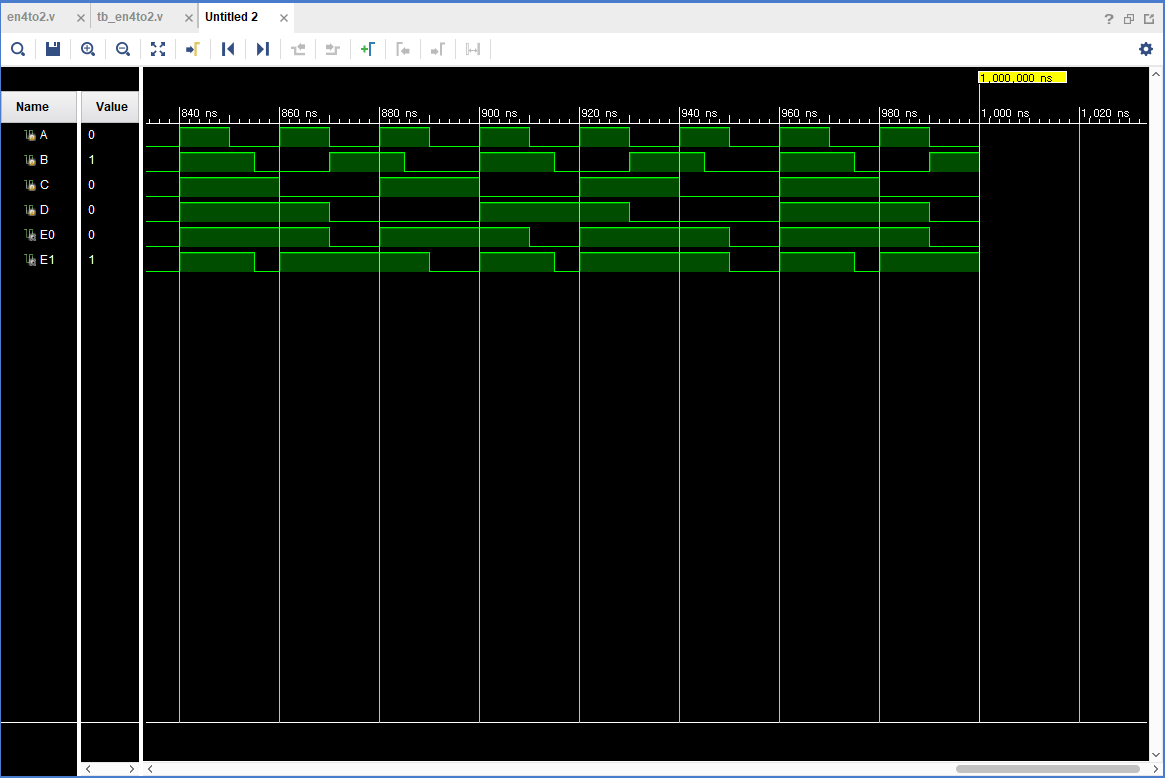
자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

**<Figure 3. 4 to 2 Encoder의 K-map>**

E0 = A+B

E1 = A+C



**<Figure 4. 4 to 2 Encoder의 K-map>**

4 to 2 Encoder를 구현하기 위해서 Input에서 D만 1로 입력하는 경우에는 E0와 E1이 각각 0, 0 이 출력되도록 했다. C만 1로 입력한 경우 E0는 0, E1은 1로 출력되도록 했다. B만 1로 입력한 경우에는 E0를 1, E1을 0으로 출력되도록 했다. 마지막으로 A만 1로 입력한 경우 E0, E1모두 1이 출력되도록 했다. 의도대로 결과가 제대로 simulation상에서 나타난 것을 확인했다.

**3. 4 to 2 Encoder에서 입력 형태 4가지를 제외한 나머지 입력 형태는 무엇을 뜻하는지 설명하시오.**

입력 형태 4가지를 제외하면, 나머지 형태는 두 가지 상태일 것이다. 한 가지는 어떠한 입력도 들어오지 않는 상태이거나, 또는 입력이 2개 이상으로 입력되고 있는 경우일 것이다. 4 to 2 encoder에서는 이러한 경우를 정의하지 못하는 동작이기에, 어떠한 bit가 활성 되어 있는지에 대하여 정보를 똑바로 담고 있지 않은 엉뚱한 출력을 내보낸다. 가령, ABCD를 각각 0110으로 입력 받을 경우 4 to 2 encoder는 0100과 0010이 입력됐다고 판단하여 E0에서의 1과 E1에서의 1 모두 출력하여 11을 출력한다. 따라서 제대로 된 결과를 얻지 못하게 될 수 있으므로, 이러한 오류를 해결하기 위해서 입력들 간에 우선순위를 정해서 해결할 수 있다.

**4. 4 to 2 encoder의 4가지 형태가 아닌 모든 입력 형태(16가지)에 대하여 동작되는 priority encoder의 논리 회로를 구성하여라.**

Priority encoder는 입력들 간의 우선순위가 있다. 따라서 우선순위에 따라서 높은 순위에 있는 입력을 출력한다. 우선 truth table을 작성해보자. Input은 A, B, C, D를 사용하고, output은 E0와 E1로 사용한다.

**<Table 3. 4 to 2 encoder (priority encoder) Truth table>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | | Output | |
| A | B | C | D | E0 | E1 |
| 0 | 0 | 0 | 0 | X | X |
| 1 | 0 | 0 | 0 | 0 | 0 |
| X | 1 | 0 | 0 | 0 | 1 |
| X | X | 1 | 0 | 1 | 0 |
| X | X | X | 1 | 1 | 1 |

그리고 K-map을 작성한다.

광장이(가) 표시된 사진

자동 생성된 설명

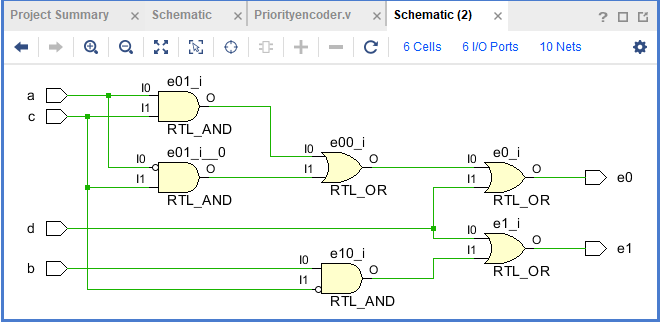
**<Figure 5. 4 to 2 encoder (priority encoder) K-map>**

따라서 E0 = AC + A’C + D이고, E1 = D + BC’ 이다. 이를 통해 vivado에서 source code를 작성해 schematic으로 회로를 시각화한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**<Figure 6. source code>**



**<Figure 7. 4 to 2 encoder (priority encoder) schematic>**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**<Figure 8. Simulation source code>**



**<figure 9. 4 to 2 encoder (priority encoder) simulation 결과>**

추가적으로 Simulation code를 작성한 후 simulation을 한 결과 다음과 같은 truth table을 작성할 수 있다.

**<Table 4. 4 to 2 encoder (priority encoder) truth table>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A | B | C | D | E0 | E1 |
| 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 | 0 |
| 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 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

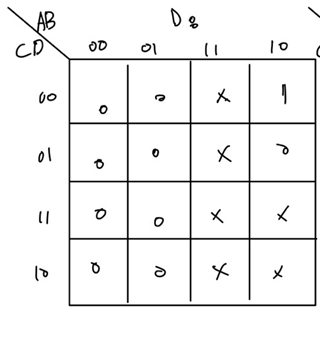
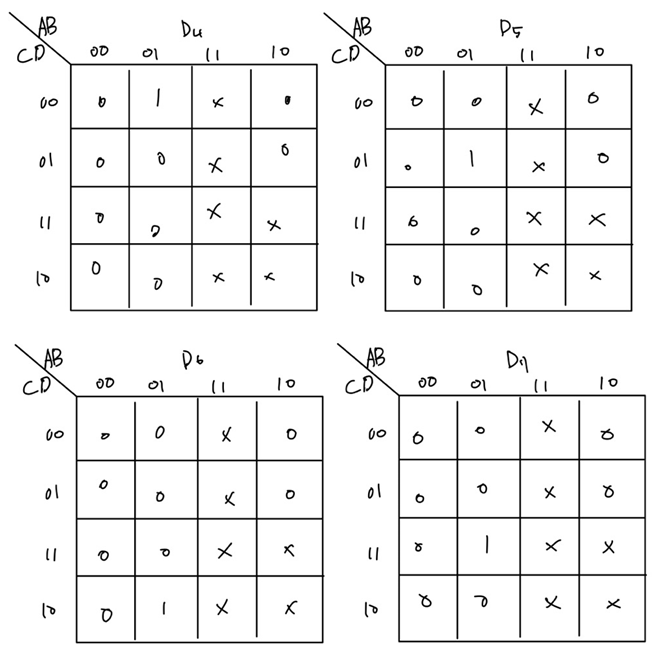
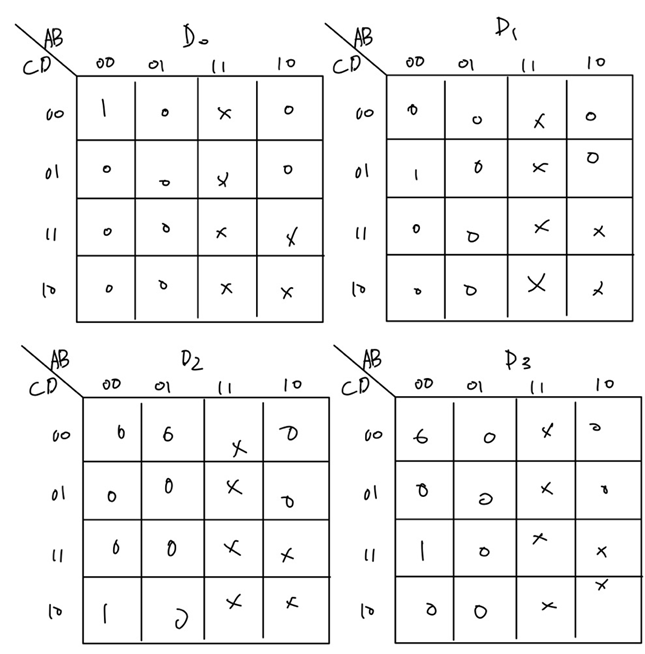
**5. BCD to Decimal decoder의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(Truth table 작성 및 k-map 포함)**

**<Table 5. BCD to Decimal decoder truth table>**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | | | Output | | | | | | | | | |
| A | B | C | D | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 |
| 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 |

input으로 A, B, C, D를 입력 받았고, output으로 D0~D8까지 출력했다. Input으로 BCD code를 받아서 Decimal로 출력하였다.



**<Figure 10. BCD to Decimal decoder K-map>**

BCD to Decimal decoder의 K-map 결과이다.

D0 = A’B’C’D’

D1 = A’B’CD’

D2 = A’B’CD

D3 = A’BC’D’

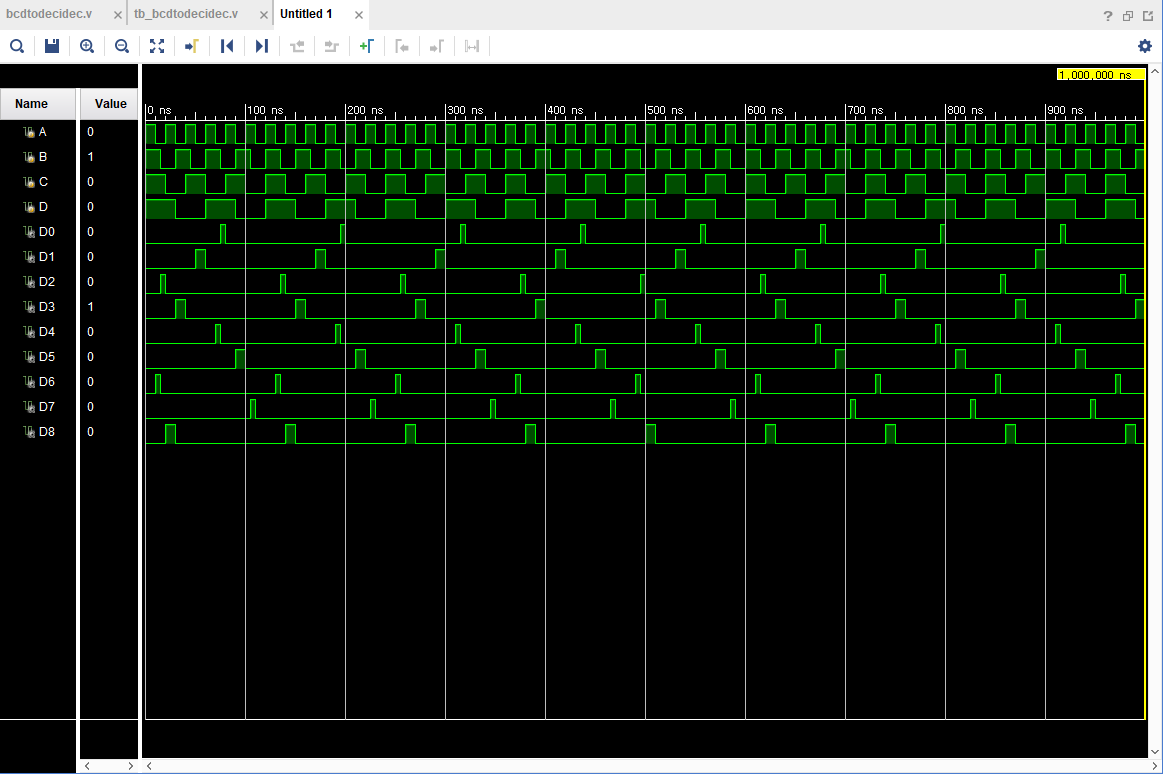
D4 = A’BC’D

D5 = A’BCD’

D6 = A’BCD

D7 = AB’C’D’

D8 = AB’C’D



**<Figure 11. BCD to Decimal decoder의 simulation 결과>**

BCD to Decimal decoder를 통해서 이진수로 1~9까지를 나타내는 9가지의 입력에 대해서 그 번호에 해당하는 십진수를 decoder를 통해 출력해주는 회로다. 예를 들어 ABCD 입력이 0001이 들어온 경우 1을 뜻하는 D0만 출력이 나오게 한다. FPGA로 실제로 테스트한 결과 0001부터 1001까지 모두 decimal로 잘 나타난 것을 확인했다.

**6.   Encoder와 decoder의 주요 응용에 대하여 설명하시오.**

Encoder는 주로 입력신호를 컴퓨터가 알아들을 수 있는 코드로 변경한다. 그리고 Decoder는컴퓨터 내부 코드를 일반적인 다른 신호로 변경하여 출력해준다. Encoder의 경우 decoder와 함께 사용하여 data를 적절한 목적지로 보내는 역할을 수행한다. 그리고 모터처럼 다양한 부품들이 서로 동기화되어서 작동하는 장치에도 쓰이고 있다. 그리고 보안을 위한 암호화에도 사용되고 있다.

Decoder의 경우 multiplexing이나 demultiplexing에 사용하거나, 숫자를 표현하기 위한 n-segment display에 사용하기도 한다. 그리고 컴퓨터 내부에서 memory의 위치를 받은 다음 해당 주소 값을 출력하는 memory addressing 작업에도 이용하기도 한다.

**7.   4 to 1 line MUX의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(code, Truth table 작성)**

**<4 to 1 line MUX source code>**

`timescale 1ns / 1ps

module mux4to1(

input a,b,c,d,s0,s1,

output f

);

assign f = (a&~s0&~s1) | (b & ~s1 & s0) | (c& s1 & ~s0) | (d & s1 & s0);

endmodule

**<4 to 1 line MUX simulation code>**

`timescale 1ns / 1ps

module tb\_mux4to1;

reg A, B, C, D, S0, S1;

wire F;

mux4to1 mygate(.a(A), .b(B), .c(C), .d(D), .s0(S0), .s1(S1), .f(F));

initial begin

A = 1'b1;

B = 1'b1;

C = 1'b1;

D = 1'b1;

S0 = 1'b1;

S1 = 1'b1;

end

always @(A or B or C or D or S0 or S1) begin

A <= #10 ~A;

B <= #20 ~B;

C <= #30 ~C;

D <= #50 ~D;

S0 <= #70 ~S0;

S1 <= #90 ~S1;

end

initial begin

#1000;

$finish;

end

endmodule

**<Table 6. 4 to 1 line MUX의 Truth table>**

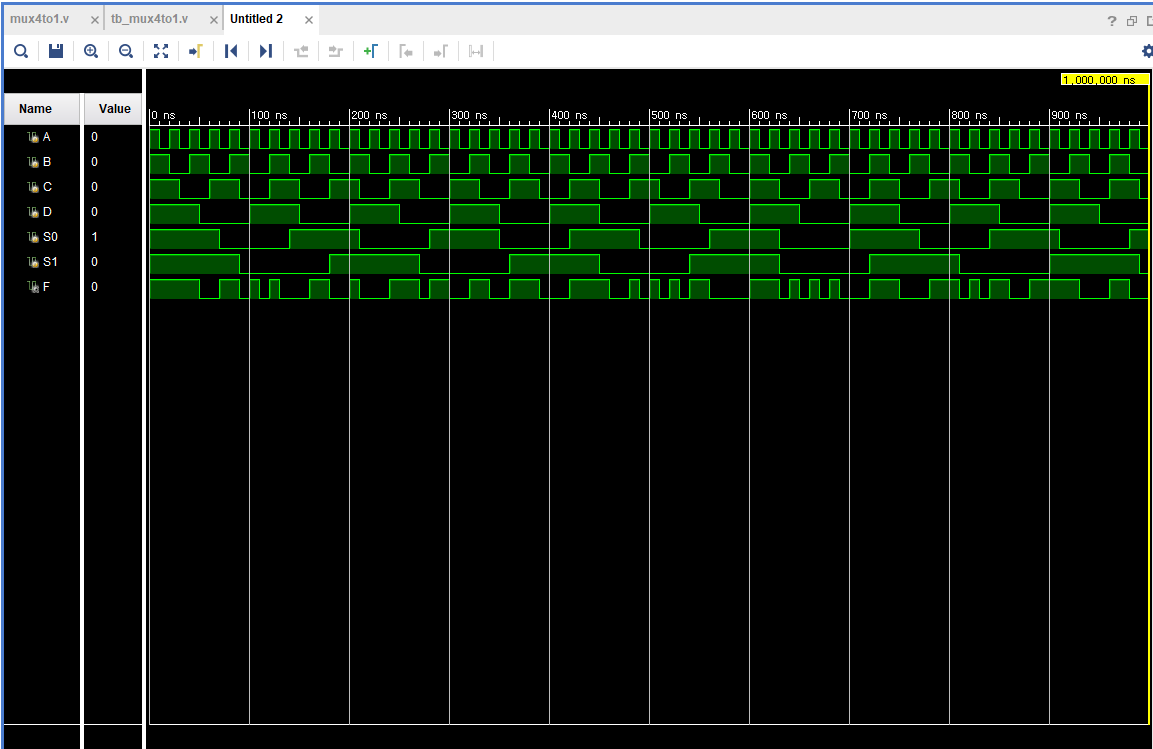
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S0 | S1 | A | B | C | D | F |
| 0 | 0 | 0 | X | X | X | 0 |
| 0 | 0 | 1 | X | X | X | 1 |
| 0 | 1 | X | 0 | X | X | 0 |
| 0 | 1 | X | 1 | X | X | 1 |
| 1 | 0 | X | X | 0 | X | 0 |
| 1 | 0 | X | X | 1 | X | 1 |
| 1 | 1 | X | X | X | 0 | 0 |
| 1 | 1 | X | X | X | 1 | 1 |

우리는 output expression을 다음과 같이 쓸 수 있다.

F = S0’S1’A + S0’S1B + S0S1’C + S0S1D

따라서 위의 source code에 다음과 같이 할당하였다.

assign f = (a&~s0&~s1) | (b & ~s1 & s0) | (c& s1 & ~s0) | (d & s1 & s0);

****

**<figure 12. 4 to 1 line MUX의 simulation 결과>**

Truth table에서 확인할 수 있듯이 S0와 S1의 인풋과 각각 A, B, C, D의 인풋에 따라 output이 어떻게 출력되는지 확인했다. S0와 S1이 각각 0, 0이면 input A, B, C, D중 A만 선택되어 출력 값에 영향을 준다. 그리고 S0와 S1이 각각 0, 1이면 input A, B, C, D중 B만 선택되어 출력 값에 영향을 준다. 그리고 S0와 S1이 각각 1, 0이면 input A, B, C, D중 C만 선택되어 출력 값에 영향을 준다. 그리고 S0와 S1이 각각 1, 1이면 input A, B, C, D중 D만 선택되어 출력 값에 영향을 준다. 실제로 simulation결과를 확인해보면, 우선 S0와 S1이 둘 다 0으로 입력한 경우 A가 0일 때 F가 0으로 출력될 것이고, A가 1일 때 F가 1로 출력될 것이다. 그리고 S0와 S1이 각각 0과 1로 입력됐을 때는, B가 0일 때 F가 0으로 출력되고, B가 1일 때는 F가 1로 출력된다. 그리고 S0와 S1이 각각 1과 0으로 입력됐을 때는, C가 0일 때 F가 0으로 출력되고, C가 1일때는 F가 1로 출력된다. 마지막으로 S0와 S1이 둘 다 1로 입력된 경우 D가 0일 때는 F가 0으로 출력되고, D가 1일로 입력됐을 때는 F가 1로 출력되는 것을 확인할 수 있다.

**8.   1 to 4 line deMUX를 이용하여 4 to 16 decoder를 수행하고 결과를 나타내시오**

**(코드, Truth table 작성)**

**<1 to 4 line deMUX source code>**

`timescale 1ns / 1ps

module demux1to4(

input s0, s1, f,

output a,b,c,d

);

assign a = ~s0 & ~s1 & f;

assign b = s0 & ~s1 & f;

assign c = ~s0 & s1 & f;

assign d = s0 & s1 & f;

endmodule

Demux의 경우 위의 source code로 회로를 구현하고 input에 맞게 output이 잘 출력되는지 확인할 수 있었다. 위를 이용해서 4 to 16 decoder를 만들어본다. Demux의 입력데이터 중 f의 경우 decoder에서 사용되는 enable 입력의 역할과 동일하다고 생각하면 된다. 즉, F가 1인경우에만 출력 값이 나오도록 할 것이므로, simulation source를 작성할 때 F는 1인 경우만 고려한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

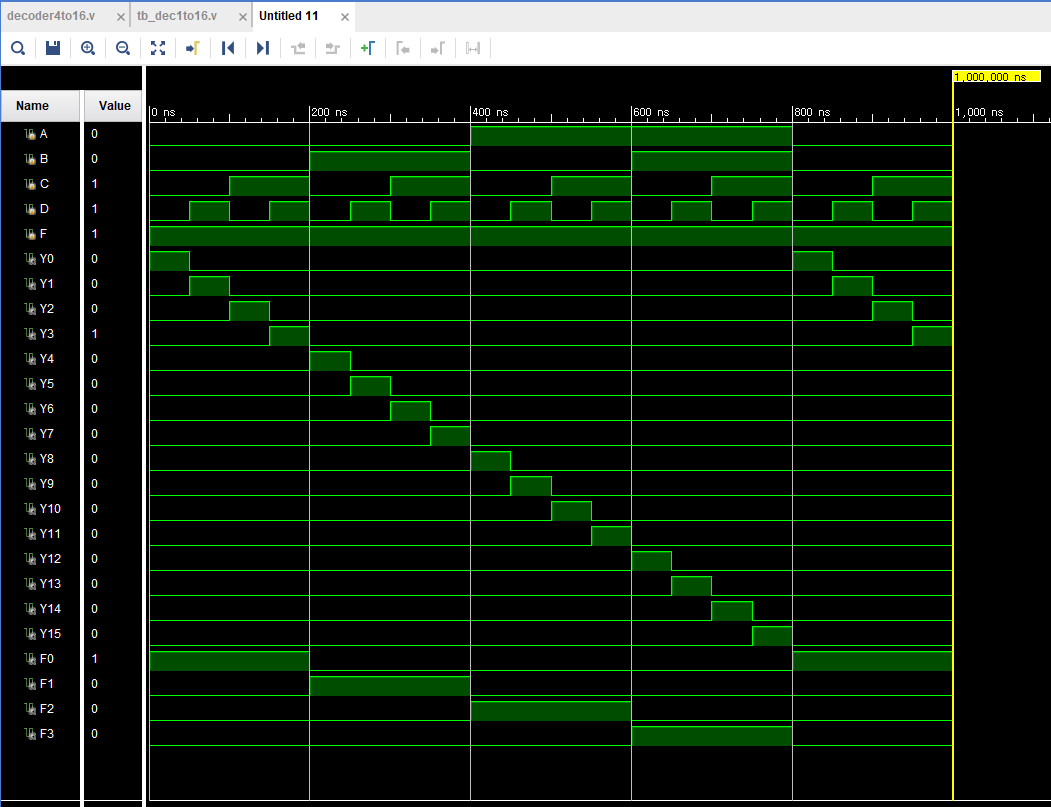
**<Figure 13. 4 to 16 decoder의 source code>**

Source code를 작성해준 후에 정상적으로 돌아가는지 확인하기 위해서 simulation source를 작성하고, simulation결과를 확인한다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**<Figure 14. 4 to 16 decoder simulation sources>**



**<Figure 15. 4 to 16 decoder simulation의 결과>**

**Figure 15**를 확인한 결과 4 to 16 decoder는 정상적으로 작동하는 것을 확인했다. 각 input값에 맞게 output 값은 단 하나만 출력되는 것이 확인됐다.

**<Table 7. 4 to 16 decoder의 truth table>**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | | | | | Output | | | | | | | | | | | | | | | |
| F = 1 | A | B | C | D | Y0 | Y1 | Y2 | Y3 | Y4 | Y5 | Y6 | Y7 | Y8 | Y9 | Y10 | Y11 | Y12 | Y13 | Y14 | Y15 |
| 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 |

**9.  결과 검토 및 논의 사항.**

Decoder의 경우 정해진 input에 따라 하나의 Output만 출력되는 것을 확인했다. 또한 encoder의 경우 decoder와는 반대로 input을 4bit로 줘서 ABCD중 D만 1로 입력한 경우에는 E0, E1가 0, 0이 출력되도록 4 to 2 encoder를 구현했다. 그리고 BCD to decimal decoder도 이진수로 1~9까지 나타내는 9가지 입력에 대해 그 번호에 해당하는 십진수를 decoder를 통해서 출력시켰다. 그리고 4 to 1 line Mux 또한 s0, s1의 input과 ABCD 각각의 input에 따라 output을 결정했고, 어떻게 출력되는지 확인했다.

Decoder와 encoder 그리고 BCD to decimal decoder, Mux와 deMux 모두 의도한 대로 회로들이 기능을 했음을 simulation과 truth table을 통해 확인했다. 카르노 맵을 통해서 논리식을 간소화하여 회로를 설계했고, 각각의 output을 출력하기 위해 input을 적절히 할당해준 다음 simulation 결과를 확인하고, 최종적으로 FPGA를 통해 구현한 것을 확인했다.

Gate의 개수가 많아지면서 비용적인 측면을 고려하지 않을 수 없다. 동일한 gate를 사용할 수 있다면 최대한 사용하도록 하여 설계해야 할 gate의 수를 줄여서 비용적으로 절감할 수 있을 것이라 생각된다.

**10.  추가 이론 조사 및 작성.**

Three state gates는 tri-state buffer라고도 불린다. 이때까지 봤던 모든 회로들은 output으로 대부분 0, 1을 출력했지만, three state gate는 Z라는 output도 있다. 이는 High impedance state이고, 아무 상태에도 연결이 되지 않아 있는 상태를 뜻한다. 즉, Z상태는 우리가 어떤 상태인지 알 수 없는 상태를 말한다.

시계이(가) 표시된 사진

자동 생성된 설명

위의 표와 그림에서 볼 수 있듯이 EN의 상태에 따라서 출력 값이 결정되는 구조로 되어있다.

이때 EN의 상태에 따라 a와 b만이 입력되도록 바꿀 수 있는데, 만약 EN 없이 와이어가 연결되어 있었다면, a와 b 와이어 모두 전류를 보내면서 누전현상이 일어났을 것이다. Three states buffer는 2개 이상의 gate나 다른 논리소자들도 연결시킬 수 있다. 따라서 둘 이상의 장치가 다른 장치에 전기적으로 연결되어 있을 때, three state gates를 이용해서 이러한 비정상적인 현상을 방지할 수 있다.