**컴퓨터공학 설계 및 실험Ⅱ**

9주차 결과보고서

서강대학교 공학부 컴퓨터공학 전공

20171646 박태윤

**1. 2 to 4 Decoder 의 결과 및 Simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함,[AND,NAND GATE])**

**(1) 2 to 4 Decoder(AND Gate)**

- 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | | **Output** | | | |
| A | B | W | X | Y | Z |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

- 카르노 맵

(W)

|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 1 | 0 |
| **1** | 0 | 0 |

W = A’B’

(X)

|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 0 | 1 |
| **1** | 0 | 0 |

X = A’B

(Y)

|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 0 | 0 |
| **1** | 1 | 0 |

Y = AB’

(Z)

|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 0 | 0 |
| **1** | 0 | 1 |

Z = AB

- Verilog Code

|  |  |
| --- | --- |
| **AND\_Decoder** | **AND\_Decoder\_tb** |
| `timescale 1ns / 1ps  module AND\_Decoder(A,B,W,X,Y,Z);  input A,B;  output W,X,Y,Z;  assign W = (~A)&(~B);  assign X = (~A)&B;  assign Y = A&(~B);  assign Z = A&B;  endmodule | `timescale 1ns / 1ps  module AND\_Decoder\_tb;  reg A,B;  wire W,X,Y,Z;  AND\_Decoder u\_AND\_Decoder(  .A(A), .B(B),  .W(W),.X(X), .Y(Y), .Z(Z)  );  initial A = 1'b0; initial B = 1'b0;  always A = #20 ~A;  always B = #10 ~B;  initial begin  #100  $finish;  end  endmodule |

- Simulation

스크린샷, 컴퓨터, 노트북이(가) 표시된 사진

자동 생성된 설명

**(2) 2 to 4 Decoder(NAND Gate)**

- 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | | **Output** | | | |
| A | B | W | X | Y | Z |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

- 카르노 맵

(W)

|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 0 | 1 |
| **1** | 1 | 1 |

W = A + B = (A’B’)’

(X)

|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 1 | 0 |
| **1** | 1 | 1 |

X = A + B’ = (A’B)’

(Y)

|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 1 | 1 |
| **1** | 0 | 1 |

Y = A’ + B = (AB’)’

(Z)

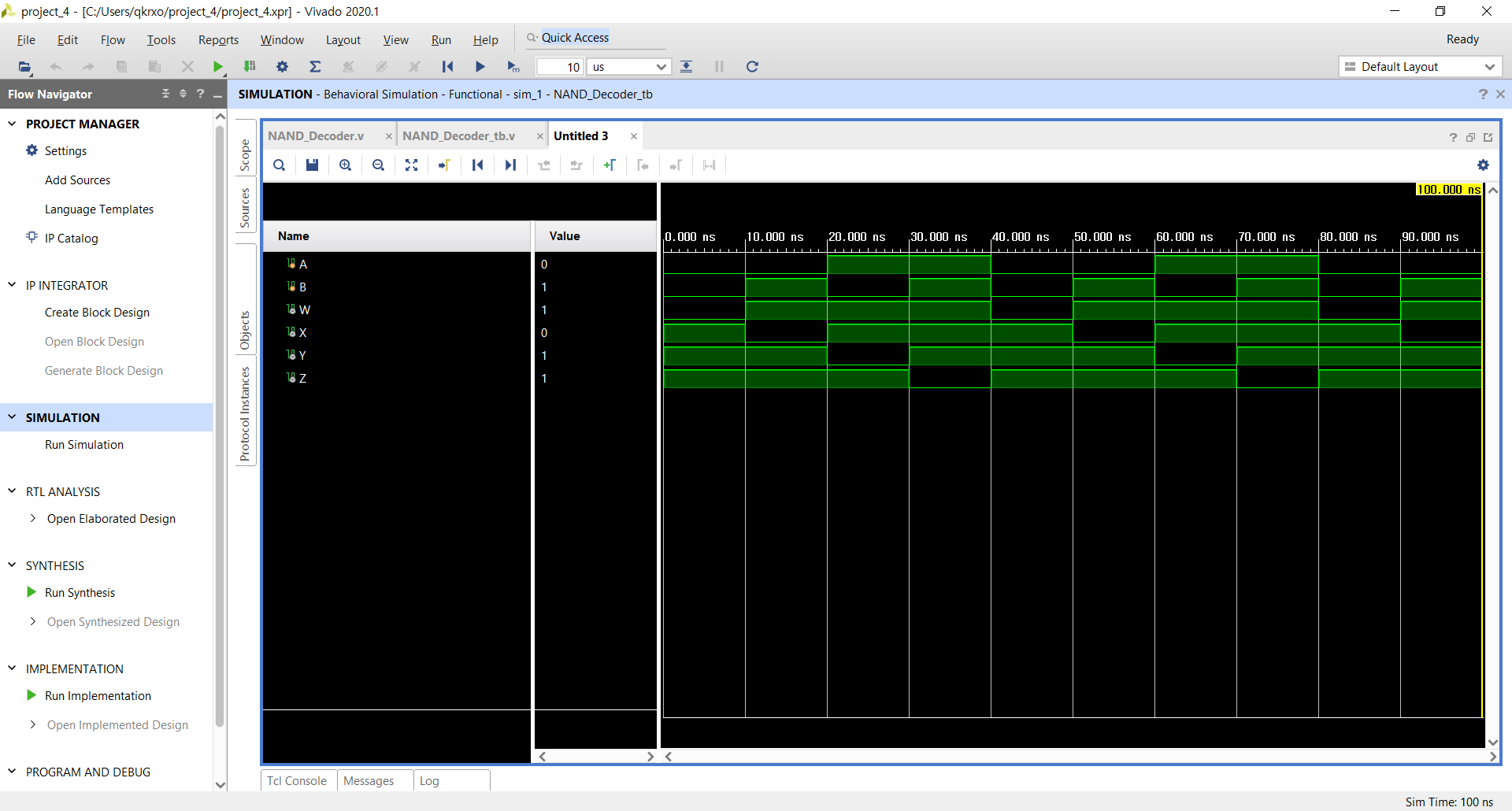
|  |  |  |
| --- | --- | --- |
| **A/B** | **0** | **1** |
| **0** | 1 | 1 |
| **1** | 1 | 0 |

Z = A’ + B’ = (AB)’

- Verilog Code

|  |  |
| --- | --- |
| **NAND\_Decoder** | **NAND\_Decoder\_tb** |
| `timescale 1ns / 1ps  module NAND\_Decoder(A,B,W,X,Y,Z);  input A,B;  output W,X,Y,Z;  assign W = ~((~A)&(~B));  assign X = ~((~A)&B);  assign Y = ~(A&(~B));  assign Z = ~(A&B);  endmodule | `timescale 1ns / 1ps  module NAND\_Decoder\_tb;  reg A,B;  wire W,X,Y,Z;  NAND\_Decoder u\_NAND\_Decoder(  .A(A), .B(B),  .W(W),.X(X), .Y(Y), .Z(Z)  );  initial A = 1'b0; initial B = 1'b0;  always A = #20 ~A;  always B = #10 ~B;  initial begin  #100  $finish;  end  endmodule |

- Simulation



- and gate를 사용한 디코더와 nand gate를 사용한 디코더 비교

and gate를 사용하여 구현한 디코더와 nand gate를 사용하여 구현한 디코더의 가장 큰 차이는 출력에 있다. and gate를 사용한 디코더는 입력에 따라 1을 나타내는 출력이 모두 다르며 nand gate를 사용한 디코더는 0을 나타내는 출력이 모두 다르다.

**2. 4 to 2 Encoder의 결과 및 Simulation 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함)**

**(1) 2 to 4 Encoder(OR Gate)**

- 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input** | | | | **Output** | |
| A | B | C | D | X | Y |
| 0 | 0 | 0 | 0 | x | x |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | x | x |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | x | x |
| 0 | 1 | 1 | 0 | x | x |
| 0 | 1 | 1 | 1 | x | x |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | x | x |
| 1 | 0 | 1 | 0 | x | x |
| 1 | 0 | 1 | 1 | x | x |
| 1 | 1 | 0 | 0 | x | x |
| 1 | 1 | 0 | 1 | x | x |
| 1 | 1 | 1 | 0 | x | x |
| 1 | 1 | 1 | 1 | x | x |

- 카르노 맵

(X)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB/CD** | **00** | **01** | **11** | **10** |
| **00** | x | 0 | x | 0 |
| **01** | 1 | x | x | x |
| **11** | x | x | x | x |
| **10** | 1 | x | x | x |

X = A + B

(Y)

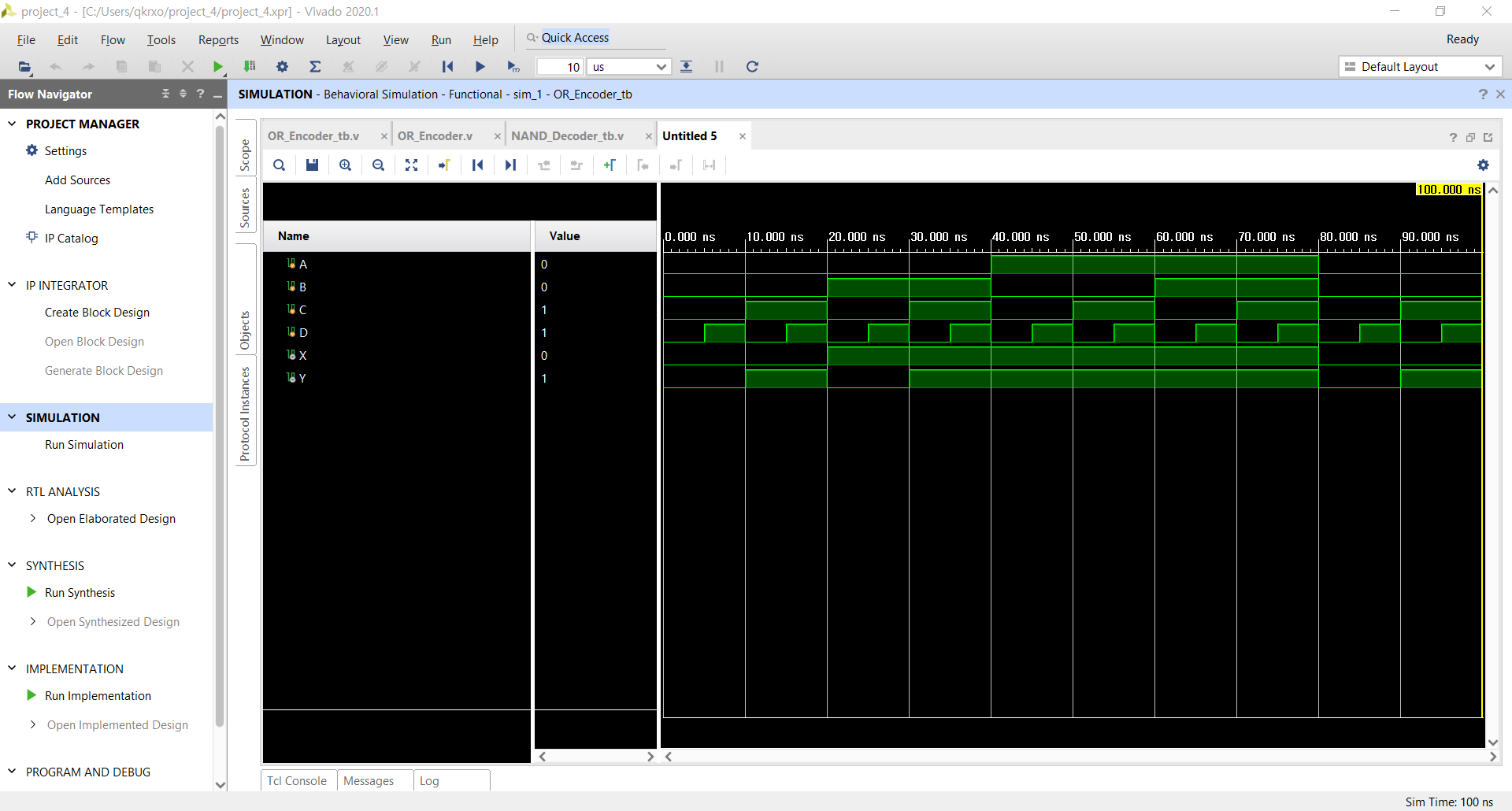
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB/CD** | **00** | **01** | **11** | **10** |
| **00** | x | 0 | x | 1 |
| **01** | 0 | x | x | x |
| **11** | x | x | x | x |
| **10** | 1 | x | x | x |

Y = A + C

- Verilog Code

|  |  |
| --- | --- |
| **OR\_Encoder** | **OR\_Encoder\_tb** |
| `timescale 1ns / 1ps  module OR\_Encoder(A,B,C,D,X,Y);  input A,B,C,D;  output X,Y;  assign X = A|B;  assign Y = A|C;  endmodule | `timescale 1ns / 1ps  module OR\_Encoder\_tb;  reg A,B,C,D;  wire X,Y;  OR\_Encoder u\_OR\_Encoder(  .A(A), .B(B), .C(C), .D(D),  .X(X), .Y(Y)  );  initial A = 1'b0; initial B = 1'b0; initial C = 1'b0; initial D = 1'b0;  always A = #40 ~A;  always B = #20 ~B;  always C = #10 ~C;  always D = #5 ~D;  initial begin  #100  $finish;  end  endmodule |

- Simulation



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

진리표에 따르면 입력 중 하나만 1값을 나타내고 나머지 입력은 모두 0을 나타낼 때만 출력을 하고 나머지는 모두 don’t care로 처리하였다. Encoder와 Decoder는 서로 반대되어 4 to 2 인코더에서 입력 중 하나만 1을 나타내고 나머지 입력들은 모두 0을 나타내야 한다. 하지만 입력이 4개인 경우에 실제 입력의 경우의 수는 2^4 = 16개 이므로 나머지 12개의 입력에 대해서도 처리를 해주어야 한다. 실제로 인코더에서는 입력에 우선순위를 부여하여 구현하는 방식을 택한다. 예를 들어 우선순위가 입력 중 가장 높은 A가 1을 나타낸다면 B, C, D가 어떤 값이든 입력이 (1,0,0,0)일 때와 같은 출력을 나타낸다.

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

- 진리표

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Input** | | | | **Output** | | |
| **A** | **B** | **C** | **D** | **X** | **Y** | **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 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Don’t care가 아닌 실제 우선순위를 고려하여 진리표를 다음과 같이 만들 수 있다. 우선순위는 A가 가장 높으며 D가 가장 낮은 순서로 구성하였으며 우선순위가 높은 입력이 1값을 나타낸다면 나머지 우선순위가 낮은 입력이 어떤 값을 나타내는지는 고려하지 않는다. 출력에서 V는 입력이 유효한지 아닌지를 나타내는데, 입력이 모두 0일 때 0, 입력 중 하나라도 1을 나타낼 경우 1을 출력한다. V가 0일 때 나머지 출력 X, Y는 don’t care로 처리할 수 있다.

- 카르노 맵

(X)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB/CD** | **00** | **01** | **11** | **10** |
| **00** | x | 0 | 0 | 0 |
| **01** | 1 | 1 | 1 | 1 |
| **11** | 1 | 1 | 1 | 1 |
| **10** | 1 | 1 | 1 | 1 |

X = A + B

(Y)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB/CD** | **00** | **01** | **11** | **10** |
| **00** | x | 0 | 1 | 1 |
| **01** | 0 | 0 | 0 | 0 |
| **11** | 1 | 1 | 1 | 1 |
| **10** | 1 | 1 | 1 | 1 |

Y = A + B’C

(V)

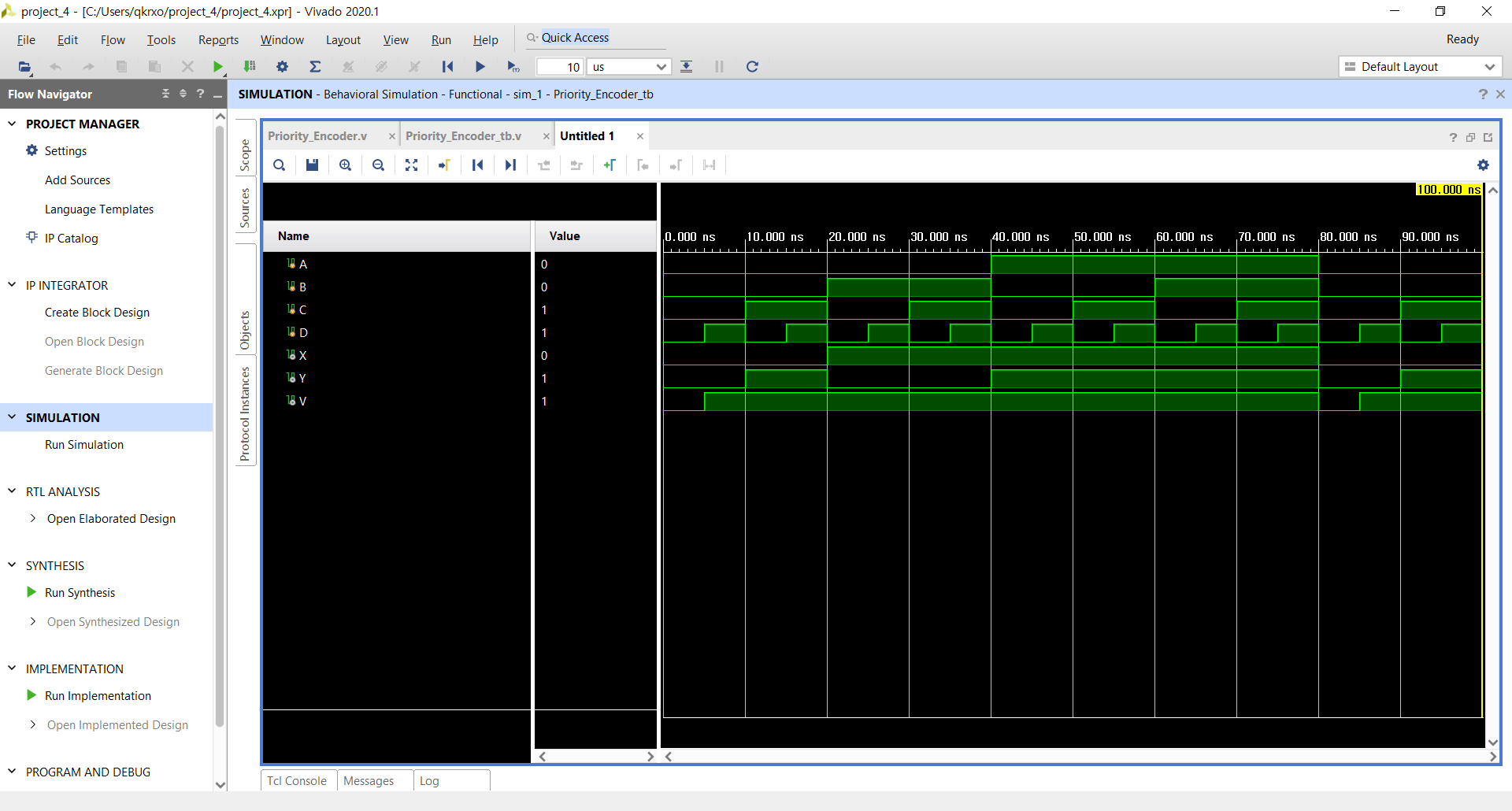
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB/CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | 1 | 1 |
| **01** | 1 | 1 | 1 | 1 |
| **11** | 1 | 1 | 1 | 1 |
| **10** | 1 | 1 | 1 | 1 |

V = A + B + C + D

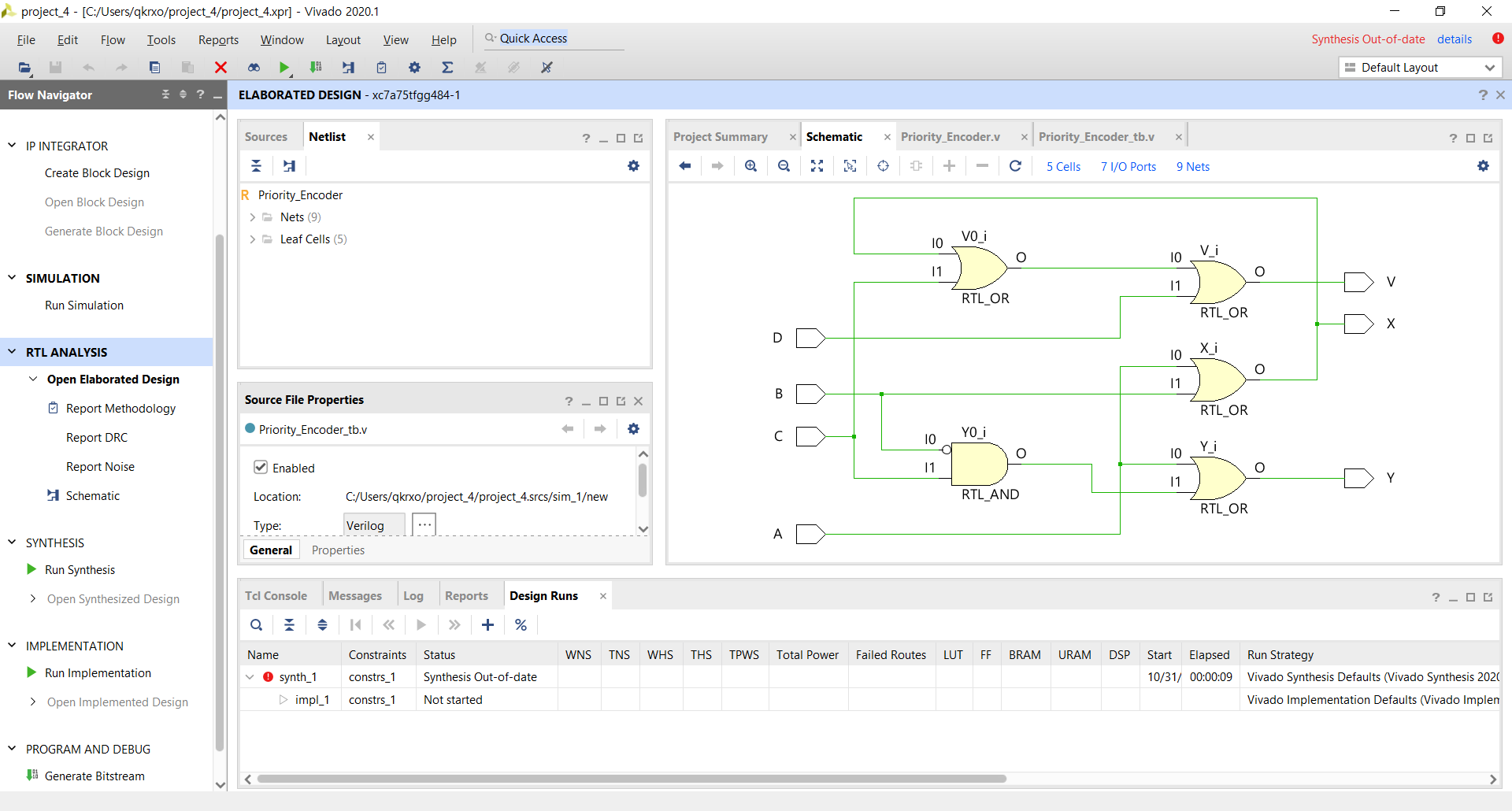
- Verilog Code

|  |  |
| --- | --- |
| **Priority\_Encoder** | **Priority\_Encoder\_tb** |
| `timescale 1ns / 1ps  module Priority\_Encoder(A,B,C,D,X,Y,V);  input A,B,C,D;  output X,Y,V;  assign X = A|B;  assign Y = A|((~B)&C);  assign V = A|B|C|D;  endmodule | `timescale 1ns / 1ps  module Priority\_Encoder\_tb;  reg A,B,C,D;  wire X,Y,V;  Priority\_Encoder u\_Priority\_Encoder(  .A(A), .B(B), .C(C), .D(D),  .X(X), .Y(Y), .V(V)  );  initial A = 1'b0; initial B = 1'b0; initial C = 1'b0; initial D = 1'b0;  always A = #40 ~A;  always B = #20 ~B;  always C = #10 ~C;  always D = #5 ~D;  initial begin  #100  $finish;  end  endmodule |

- Simulation



- Schematic



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

인코더는 주로 입력 신호를 컴퓨터 내부에서 사용하는 코드로 변경하는 역할을 해준다. 즉, 인코더는 어떤 자료를 다른 장치에 전달할 때 그 장치가 잘 활용할 수 있도록 신호를 바꿔주는 역할을 한다. 예를 들어 10진수로 표현된 숫자를 컴퓨터가 활용할 수 있는 2진수로 바꿔주는 것 또한 인코더라고 할 수 있다.

디코더는 인코더와 반대로 컴퓨터 내부의 코드를 일반적인 신호로 변경하여 출력하는 역할을 한다. 위의 예시와 연결하면 인코더에서 10진수로 표현된 숫자를 컴퓨터가 활용가능한 2진수로 바꿔 전달해주었다면 디코더는 컴퓨터에서 2진수를 받아 사용자가 활용할 수 있는 10진수로 변경하여 출력해준다.

그 이외에 다음과 같은 분야에서 인코더와 디코더가 쓰이는 것을 확인할 수 있다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **목적** | **원본** | **처리방식** | **부호화(Encoding)** | **복호화**  **(Decoding)** |
| 동영상 파일 압축 | 압축되지 않은 파일 | 변환 알고리즘(표준화) | 압축 알고리즘으로 변환 | 코덱으로 압축을 푼다. |
| 통신보안 | 암호화되지 않은 신호 | 암호화 | 비화기로 암호화 | 암호해제를 통해 원래의 신호로 변환 |
| 아파트의 층을 숫자화 | 아파트 층 | 숫자화 | 각 층을 십진수화 | 해당 층을 지목 |

**6. 8 to 1 line MUX의 결과 및 Simulation 과정에 대해서 설명하시오. (code, Truth table 작성)**

**(1) 8 to 1 line MUX**

- 진리표

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | | | | | | | | | | | **Output** |
| x | y | z | A | B | C | D | E | F | G | H | O |
| 0 | 0 | 0 | 0 | x | x | x | x | x | x | x | 0(A) |
| 0 | 0 | 0 | 1 | x | x | x | x | x | x | x | 1(A) |
| 0 | 0 | 1 | x | 0 | x | x | x | x | x | x | 0(B) |
| 0 | 0 | 1 | x | 1 | x | x | x | x | x | x | 1(B) |
| 0 | 1 | 0 | x | x | 0 | x | x | x | x | x | 0(C) |
| 0 | 1 | 0 | x | x | 1 | x | x | x | x | x | 1(C) |
| 0 | 1 | 1 | x | x | x | 0 | x | x | x | x | 0(D) |
| 0 | 1 | 1 | x | x | x | 1 | x | x | x | x | 1(D) |
| 1 | 0 | 0 | x | x | x | x | 0 | x | x | x | 0(E) |
| 1 | 0 | 0 | x | x | x | x | 1 | x | x | x | 1(E) |
| 1 | 0 | 1 | x | x | x | x | x | 0 | x | x | 0(F) |
| 1 | 0 | 1 | x | x | x | x | x | 1 | x | x | 1(F) |
| 1 | 1 | 0 | x | x | x | x | x | x | 0 | x | 0(G) |
| 1 | 1 | 0 | x | x | x | x | x | x | 1 | x | 1(G) |
| 1 | 1 | 1 | x | x | x | x | x | x | x | 0 | 0(H) |
| 1 | 1 | 1 | x | x | x | x | x | x | x | 1 | 1(H) |

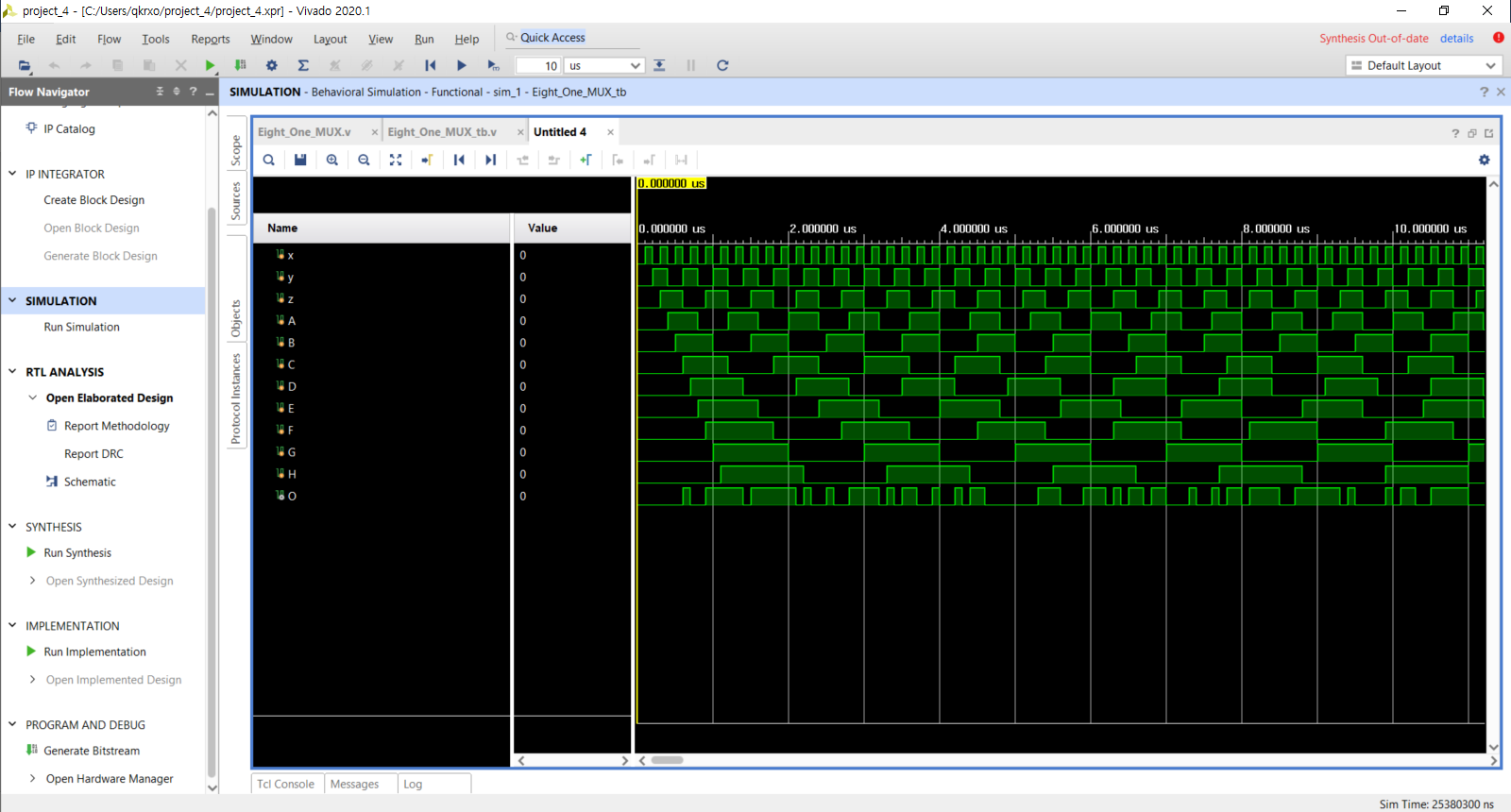
다음과 같이 진리표를 작성할 수 있다. 입력 x,y,z에 따라 (A~H)까지의 입력 중 하나를 선택하여 출력해준다.

- Verilog Code

|  |
| --- |
| **Eight\_One\_MUX** |
| `timescale 1ns / 1ps  module Eight\_One\_MUX(x,y,z,A,B,C,D,E,F,G,H,O);  input x,y,z,A,B,C,D,E,F,G,H;  output O;  assign O = (~x&~y&~z&A)|(~x&~y&z&B)|(~x&y&~z&C)|(~x&y&z&D)|(x&~y&~z&E)|(x&~y&z&F)|(x&y&~z&G)|(x&y&z&H);  endmodule |
| **Eight\_One\_MUX\_tb** |
| `timescale 1ns / 1ps  module Eight\_One\_MUX\_tb;  reg x,y,z,A,B,C,D,E,F,G,H;  wire O;  Eight\_One\_MUX u\_Eight\_One\_MUX(  .x(x), .y(y), .z(z),  .A(A), .B(B), .C(C), .D(D),  .E(E), .F(F), .G(G), .H(H),  .O(O)  );  initial x = 1'b0; initial y = 1'b0; initial z = 1'b0;  initial A = 1'b0; initial B = 1'b0; initial C = 1'b0; initial D = 1'b0;  initial E = 1'b0; initial F = 1'b0; initial G = 1'b0; initial H = 1'b0;  always x = #100 ~x;  always y = #200 ~y;  always z = #300 ~z;  always A = #400 ~A;  always B = #500 ~B;  always C = #600 ~C;  always D = #700 ~D;  always E = #800 ~E;  always F = #900 ~F;  always G = #1000 ~G;  always H = #1100 ~H;  initial begin  #1000  $finish;  end  endmodule |

입력을 x,y,z,A,B,C,D,E,F,G,H 출력을 O로 선언하여 코딩하였다. x,y,z에 따라 출력 O에 A~H중 하나를 assign해주어야 하므로, 위의 코드와 같이 선언할 수 있다. 예를 들어, x,y,z는 모두 0일 때 A를 선택해야 하므로 ~x&~y&~z&A와 같이 코드를 짤 수 있으며 이와 비슷하게 각 입력에 대하여 항들을 만들 수 있다.

- Simulation



**7. 결과 검토 및 논의 사항.**

2 to 4 Decoder, 4 to 2 Encoder, Priority Encoder, 8 to 1 line MUX에 대해서 알아보았다. 각각 진리표를 작성한 이후 카르노 맵을 만들어 식을 간소화시킨 뒤 Verilog로 구현하여 Simulation동작을 통해 결과를 확인하였다. 2 to 4 디코더는 AND게이트와 NAND로 구성된 두 가지의 회로를 모두 살펴보았다. 디코더와 인코더는 서로 반대되는 개념임을 확인하였으며 특히 인코더에서 입력의 경우의 수를 고려하여 각 입력에 해당하는 우선 순위와 입력이 유효한 값인지를 나타내는 V(valid)를 넣어 모든 입력에 대해 출력 값을 나타낼 수 있는 우선 순위 인코더 또한 구현해보았다. 추가적으로 디코더와 인코더의 응용 방식과 실제 사례 또한 살펴보았다. 멀티플렉서에서는 선택선에 따라 출력이 8개의 입력 중 하나를 나타내는 것을 확인하였다. 선택선의 값이 정해지면 출력은 8개 중 단 하나의 입력을 나타내고 그 이외의 출력이 되지 않는 입력들은 모두 don’t care로 처리하였다.

**8. 추가 이론 조사 및 작성.**

코덱이란 Coder-Decoder의 약자로, 음성 또는 영상을 디지털 신호로 변환해주는 코더와 그 반대로 변환시켜 주는 디코더의 기능을 함께 갖춘 기술이다. 즉, 사람이 보고 들을 수 있는 아날로그 신호를 컴퓨터가 처리할 수 있는 디지털 신호로 변환시켜주거나 그 반대로 해석해주는 것을 의미한다. 압축되지 않은 영상이나 음성매체는 그 데이터의 크기가 상당히 크기 때문에 코덱은 크 파일들을 압축시켜 작게 묶어주거나 압축 해제를 통해 다시 본래대로 재생할 수 있게 해준다. 코덱은 크게 오디오 코덱과 비디오 코덱으로 나눌 수 있다. 오디오 코덱은 대표적으로 MP3가 존재하는데, MP3는 PCM 오디오 데이터에서 실질적으로 사람이 들을 수 없는 부분이나 필요하지 않은 부분을 버리고 다시 인코딩 하는 방식을 사용한다. 이외에 AC3, AAC, OGG, FLAC등 다양한 오디오 코덱이 존재한다. 비디오 코덱은 대표적으로 MPEG이 존재하며 이외에 ASF, DivX, Xvid등이 존재한다.

코덱은 또한 압축 품질에 따라 손실 코덱과 비손실 코덱으로 나눌 수 있다. 손실 코덱은 대부분의 오디오 코덱과 비디오 코덱이 채택한 방식으로 사람이 잘 느끼지 못하는 부분을 우선적으로 손실시켜 압축률을 크게 올리는 코덱이다. 비손실 코덱은 압축된 데이터가 원본 스트림에 존재하는 모든 정보를 보유하고 있는 코덱으로 빠른 처리가 필요한 경우 사용한다.