Lab #2 MUX Decoder Design

Class : 00

201602004 박태현

I. 실습 목적

MUX 와 Decoder 설계를 통해 3 가지 Verilog 의 작성 스타일에 대한 이해

II. 수행 결과

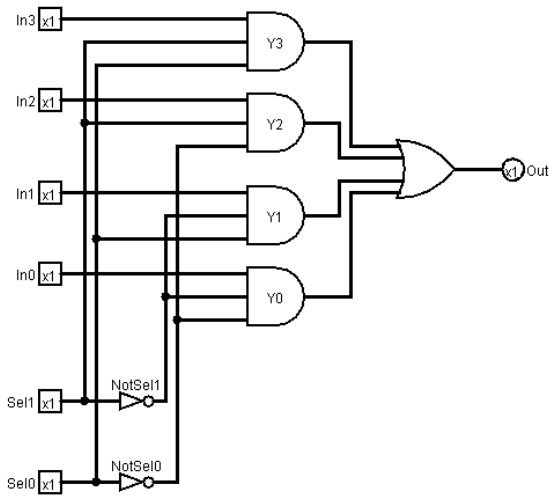
1. 4 to 1 MUX

1) Truth Table

In3	In2	In1	In0	Sel1	Sel0	Out
x	x	x	1	0	0	1
x	x	x	0	0	0	0
x	x	1	x	0	1	1
x	x	0	x	0	1	0
x	1	x	x	1	0	1
x	0	x	x	1	0	0
0	x	x	x	1	1	1
1	x	x	x	1	1	0

Out = In0 Sel1′ Sel0′ + In1 Sel1′ Sel0 + In2 Sel1 Sel0′ + In3 Sel1 Sel0

2) Logic Diagram



3) Verilog Code

Input : In0, In1, In2, In3, Sel[1:0]

Output : Out

A. Structural Style

```
module mux_4_to_1(Out, In0, In1, In2, In3, Sel);
   /* Structural Style */
   output Out;
   input In0, In1, In2, In3, Sel;

   wire [1:0] Sel;
   wire [1:0] NotSel;
   wire Y0, Y1, Y2, Y3;

   not not0(NotSel[0], Sel[0]);
   not not1(NotSel[1], Sel[1]);

   and and0(Y0, In0, NotSel[1], NotSel[0]);
   and and1(Y1, In1, NotSel[1], Sel[0]);
   and and2(Y2, In2, Sel[1], NotSel[0]);
   and and3(Y3, In3, Sel[1], Sel[0]);
   or or0(Out, Y0, Y1, Y2, Y3);
endmodule
```

Sel에 대해서 Not을 만들고 and 및 or하여 로직 다이어그램과 동일하게 만들었습니다

B. Dataflow Style

```
module mux_4_to_1(Out, In0, In1, In2, In3, Sel);
    output Out;
    input In0, In1, In2, In3, Sel;

    wire [1:0] Sel;
    wire Out;

    assign Out =
    (Sel == 2'b00) ? In0 :
    (Sel == 2'b01) ? In1 :
    (Sel == 2'b10) ? In2 :
    (Sel == 2'b11) ? In3 : 1'bx;

endmodule
```

삼항연산자를 활용하여 sel에 따라서 in을 out에 매칭시키도록 했습니다. 마지막 예외로 don't care를 주어 원치 않는 래치의 합성을 막았습니다.

C. Behavioral Style

```
module mux_4_to_1(Out, In0, In1, In2, In3, Sel);
   /* Behavioral Style */
   output Out;
   input In0, In1, In2, In3, Sel;

   wire [1:0] Sel;
   reg Out;

   always @ (In0 or In1 or In2 or In3 or Sel) begin
      casex(Sel)
      2'b00 : Out = In0;
      2'b01 : Out = In1;
      2'b10 : Out = In2;
      2'b11 : Out = In3;
      default : Out = 1'bx;
      endcase
   end

endmodule
```

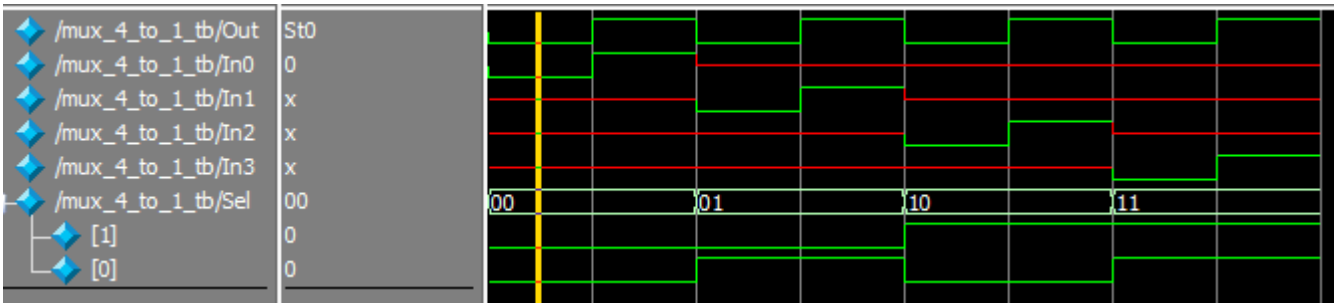
Casex를 이용해 Sel의 값에 따라서 Out에 In을 매칭시켜주었습니다. Default를 넣어서 래치 합성을 방지했습니다

4) Test bench

```
module mux_4_to_1_tb;
wire Out;
reg In0, In1, In2, In3;
reg [1:0] Sel;

mux_4_to_1 mux_4_to_10(Out, In0, In1, In2, In3, Sel);

initial begin

   Sel = 2'b00;

   {In3, In2, In1, In0} = 4'bxxx0;
   #100

   {In3, In2, In1, In0} = 4'bxxx1;
   #100

   Sel = 2'b01;

   {In3, In2, In1, In0} = 4'bxx0x;
   #100

   {In3, In2, In1, In0} = 4'bxx1x;
   #100

   Sel = 2'b10;

   {In3, In2, In1, In0} = 4'bx0xx;
   #100

   {In3, In2, In1, In0} = 4'bx1xx;
   #100

   Sel = 2'b11;

   {In3, In2, In1, In0} = 4'b0xxx;
   #100

   {In3, In2, In1, In0} = 4'b1xxx;
   #100;
   end
endmodule
```

현재 시점의 Sel과 연관되어 있는 값에 대해서만 0또는 1을 주었고, 이외의 값에 대해서는 don't care를 할당하여 테스트를 진행하였습니다

5) Check waveform



Sel = 00인 경우 In0과 일치하여 움직이고

Sel = 01인 경우 In1과 일치하여 움직이고

Sel = 10인 경우 In2과 일치하여 움직이고

Sel = 11인 경우 In3과 일치하여 움직이는 것을 확인했습니다

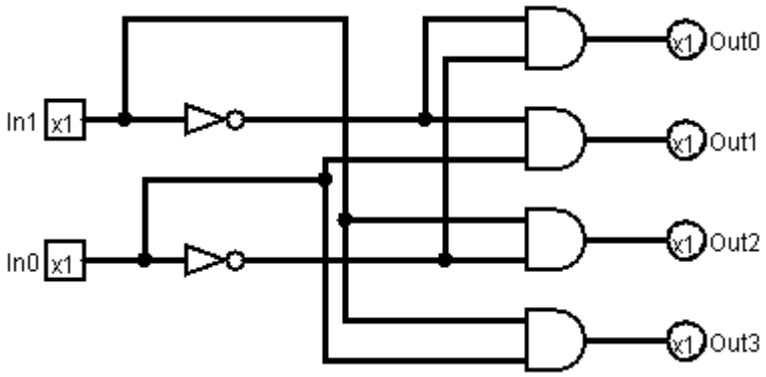
2. 2 to 4 Decoder

1) Truth Table

In1	In0	Out0	Out1	Out2	Out3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Out0 = In1′ In0′, Out1 = In1′ In0, Out2 = In1 In0′, Out3 = In1 In0

2) Logic Diagram



3) Verilog Code

Input : In[1:0] Output : Out[3:0]

A. Structural Style

```
module dec_2_to_4(Out, In);
   /* Structural style */
   output Out;
   input In;

   wire [1:0] In;
   wire [1:0] NotIn;
   wire [3:0] Out;

   not not0(NotIn[0], In[0]);
   not not1(NotIn[1], In[1]);

   and and0(Out[0], NotIn[1], NotIn[0]);
   and and1(Out[1], NotIn[1], In[0]);
   and and2(Out[2], In[1], NotIn[0]);
   and and3(Out[3], In[1], In[0]);
endmodule
```

In에 대해 Not을 만들고, 로직 다이어그램을 만든 대로 wire를 연결해주었습니다

B. Dataflow Style

```
module dec_2_to_4(Out, In);
   /* Dataflow style */
   output Out;
   input In;

   wire [1:0] In;
   wire [3:0] Out;

   assign Out[0] = !In[1] & !In[0];
   assign Out[1] = !In[1] & In[0];
   assign Out[2] = In[1] & !In[0];
   assign Out[3] = In[1] & In[0];

endmodule
```

Out에 대해서 In을 조건에 맞게 할당했습니다

C. Behavioral Style

```
module dec_2_to_4(Out, In);
   /* Behavioral style */
   output Out;
   input In;
   reg [3:0] Out;
   wire [1:0] In;

   always @ (In) begin
      casex(In)
         2'b00 : Out = 4'b0001;
         2'b01 : Out = 4'b0010;
         2'b10 : Out = 4'b0100;
         2'b11 : Out = 4'b1000;
         default: Out=4'b0000;
      endcase
   end

endmodule
```

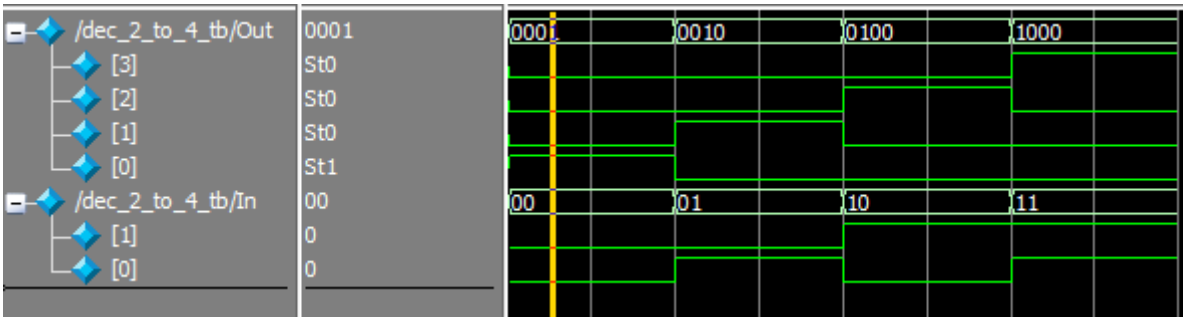
In으로 들어온 값을 읽어서 0인 경우 0번 out을 1인 경우 1번 out을 1로 2인 경우 2번 out을 1로 3인 경우 3번 out을 1로 설정하도록 했습니다

4) Test bench

```
module dec_2_to_4_tb;

wire [3:0] Out;
reg [1:0] In;

dec_2_to_4 dec_2_to_40(Out, In);

initial begin
    In = 2'b00;

    #100
    In = 2'b01;

    #100
    In = 2'b10;

    #100
    In = 2'b11;

    #100;
    end
endmodule
```

In을 00부터 11까지 하나씩 바꿔가면서 수행을 했습니다

5) Check waveform



00일 때 0을

01일 때 1을

10일 때 2를

11일 때 3에서 출력이 나오는 것을 확인할 수 있었습니다

3. Discussions

베릴로그의 세가지 작성 방식인 Structural, Dataflow, Behavioral을 직접 작성해보고 차이를 생각해볼 수 있었습니다

각 방식마다 쓸 수 있는 방법을 세가지를 구분 지어 구현하다 보니 헷갈렸습니다