SoC설계

Lab#5

컴퓨터공학과

201402439

천원준

1. Purpose of the lab

이번 과제의 목표는 4-to-1 MUX와 2-to-4 Decoder을 Structural, Dataflow, Behavioral Style로 구현하고 정상 작동하는지 확인해보는 것이다.

2. Design procedure

(1) 4-to-1 MUX

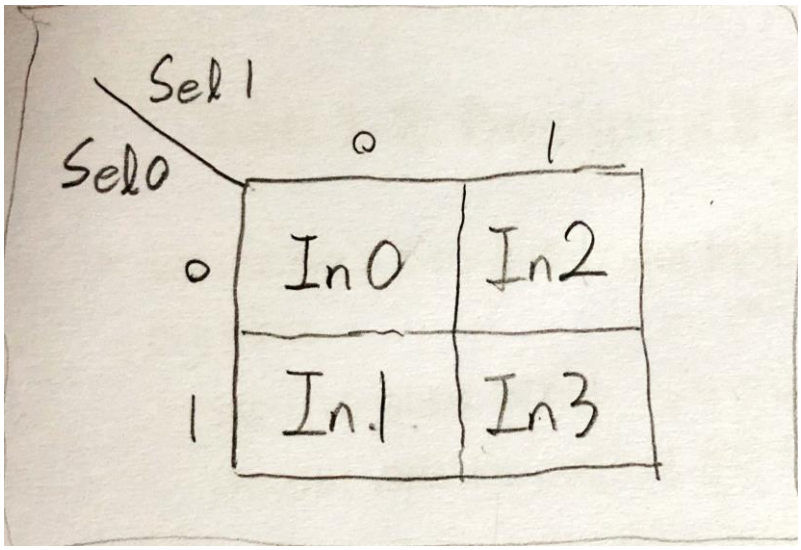
1) Truth table

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

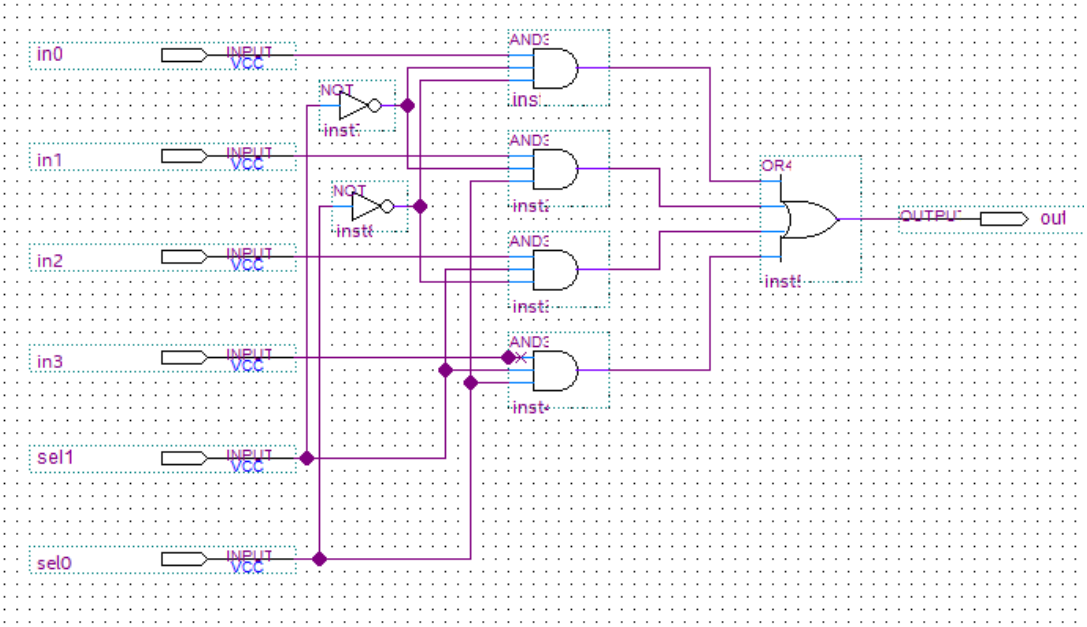
2) Boolean Equation

Out = (In0 * ~(Sel0) * ~(Sel1)) + (In1 * Sel0 * ~(Sel1)) + (In2 * ~(Sel0) * Sel1) + (In3 * Sel0 * Sel1)

3) K-map



4) Logic diagram



(2) 2-to-4 Decoder

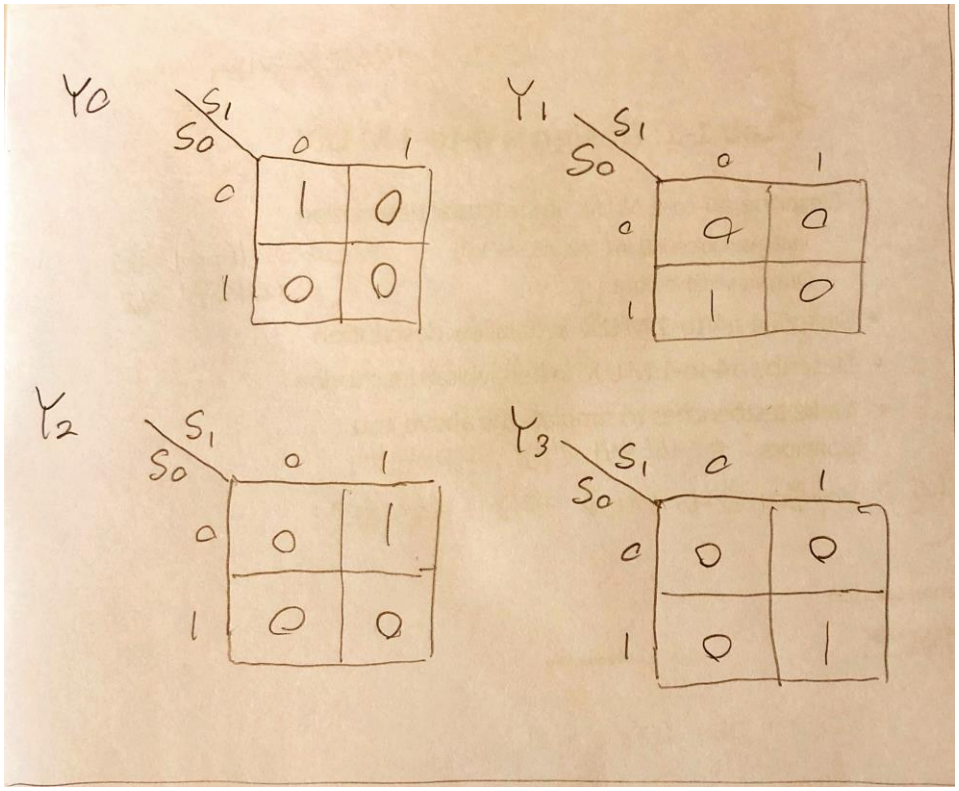
1) Truth table

S1	S0	Y0	Y1	Y2	Y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

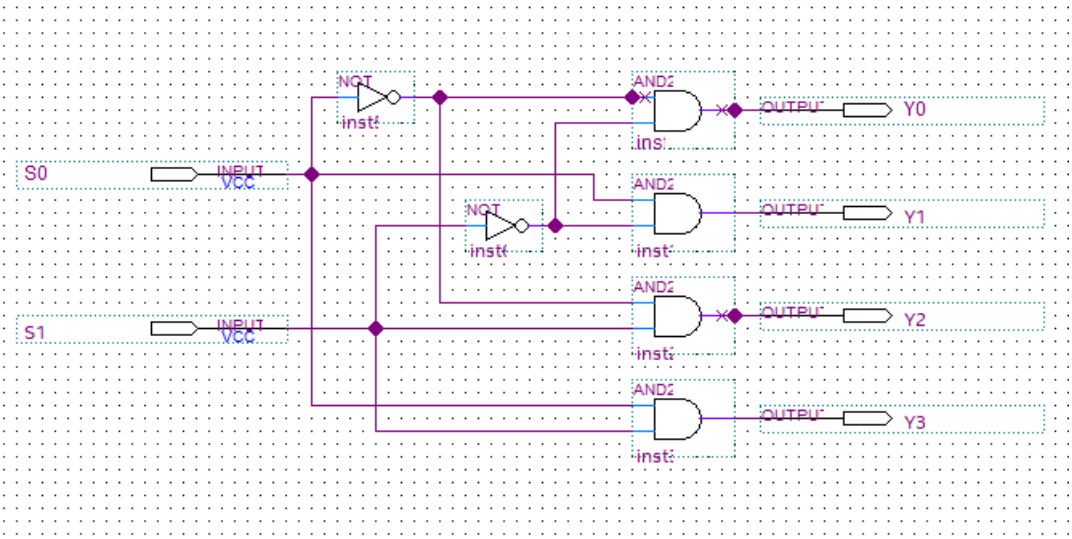
2) Boolean Equation

- Y0 = ~(S0) * ~(S1)
- Y1 = S0 * ~(S1)
- Y2 = ~(S0) * S1
- Y3 = S0 * S1

3) K-map



4) Logic diagram



3. Simulation

(1) MUX_Structural

- MUX_Structural Top module

```
module MUX_Structural(Out, In0, In1, In2, In3, Sel1, Sel0);

//input & output
output Out;
input In0, In1, In2, In3, Sel1, Sel0;

//make NotSel
wire NotSel0, NotSel1;

wire Y0, Y1, Y2, Y3;

//declare all gates(Structural Style)
not not0 (NotSel0, Sel0); //parameter (output, input)
not not1 (NotSel1, Sel1);

and and0(Y0, In0, NotSel1, NotSel0);
and and1(Y1, In1, NotSel1, Sel0);
and and2(Y2, In2, Sel1, NotSel0);
and and3(Y3, In3, Sel1, Sel0);

or or0(Out, Y0, Y1, Y2, Y3);

endmodule
```

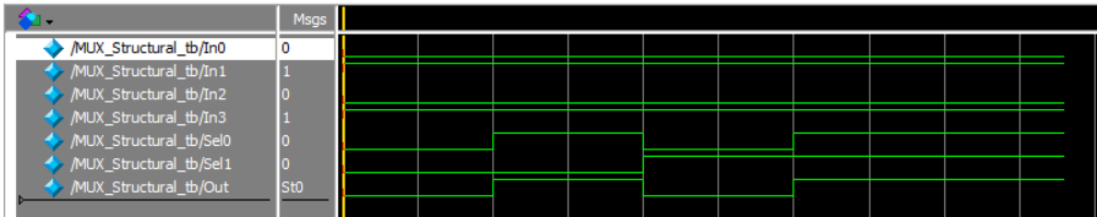
우선 각 입출력을 선언 후, 위의 Logic diagram에 따라 필요한 게이트들을 모두 선언 후 입출력을 연결해 주었습니다.

- MUX_Structural_testbench

```
`timescale 1ns/100ps

module MUX_Structural_tb;

//input
reg In0, In1, In2, In3;
reg Sel0, Sel1;

//output
wire Out;

//MUX_Structural module
MUX_Structural Mux(Out, In0, In1, In2, In3, Sel1, Sel0);

//input signal begin
initial begin
        In0 = 1'b0;
        In1 = 1'b1;
        In2 = 1'b0;
        In3 = 1'b1;

        Sel1 = 1'b0;
        Sel0 = 1'b0;

        #10
        Sel0 = 1'b1;
        #10
        Sel0 = 1'b0;
        Sel1 = 1'b1;
        #10
        Sel0 = 1'b1;
        #10; //should have semicolon at last time delay

    end
endmodule
```

Timescale는 1ns/100ps, 입력은 register로 선언, 출력은 wire로 선언 후, testbench를 실행시켰습니다. #10 간격을 두고 Sel 입력을 00, 01, 10, 11 순으로 주었습니다.

- 결과



MUX test의 목적은 Sel 입력에 따라 어떤 입력이 출력되는지를 확인하는 것이므로, Sel 입력만 #10 간격으로 변화시켰습니다. Sel 입력을 00, 01, 10, 11로 변화시키자, 출력으로 In0, In1, In2, In3 순으로 값이 나오는 것을 확인할 수 있었습니다

(2) MUX_Dataflow

- MUX_Dataflow Top module

```
1   module MUX_Dataflow(Out, In0, In1, In2, In3, Sel);
2
3   //input & output
4   output Out;
5   input In0, In1, In2, In3;
6   input [1:0] Sel; //able to declare 2bit input
7   wire Out;
8
9   //define module
10  assign Out =
11     (Sel == 2'b00) ? In0 :
12     (Sel == 2'b01) ? In1 :
13     (Sel == 2'b10) ? In2 :
14     (Sel == 2'b11) ? In3 : 1'bx;
15
16  endmodule
```

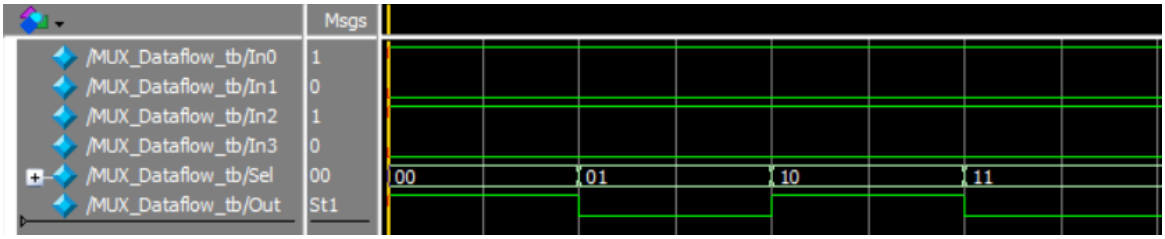
Structural Style과는 다르게, Sel 입력을 2비트짜리 신호 한 묶음으로 받습니다. 3항 연산자를 이용하여 각 게이트의 선언 없이 간단하게 입력 신호에 따른 출력을 구현하였습니다.

- MUX_Dataflow_testbench

```
1   `timescale 1ns/100ps
2
3   module MUX_Dataflow_tb;
4
5   //input
6   reg In0, In1, In2, In3;
7   reg [1:0] Sel;
8
9   //output
10  wire Out;
11
12  //MUX_Dataflow module
13  MUX_Dataflow Mux(Out, In0, In1, In2, In3, Sel);
14
15  //input signal begin
16  initial begin
17      In0 = 1'b1;
18      In1 = 1'b0;
19      In2 = 1'b1;
20      In3 = 1'b0;
21
22      Sel = 2'b00;
23
24      #10
25      Sel = 2'b01;
26      #10
27      Sel = 2'b10;
28      #10
29      Sel = 2'b11;
30      #10;
31  end
32  endmodule
```

Timescale는 1ns/100ps, 입력은 register로 선언, 출력은 wire로 선언 후, testbench를 실행시켰습니다. #10 간격으로 Sel입력에 변화를 주었습니다. Sel 입력은 00, 01, 10, 11 순으로 주었습니다.

- 결과



Sel 입력을 00, 01, 10, 11로 변화시키자, 출력으로 In0, In1, In2, In3 순으로 값이 나오는 것을 확인할 수 있었습니다

(3) MUX_Behavioral

- MUX_Behavioral Top module

```
module MUX_Behavioral(Out, In0, In1, In2, In3, Sel1, Sel0);

//input & output
output Out;
input In0, In1, In2, In3, Sel0, Sel1;
reg Out;

always @(Sel1 or Sel0 or In0 or In1 or In2 or In3)
   begin
      case ({Sel1, Sel0})
      2'b00 : Out = In0;
      2'b01 : Out = In1;
      2'b10 : Out = In2;
      2'b11 : Out = In3;
      default : Out = 1'bx;
      endcase
   end
endmodule
```

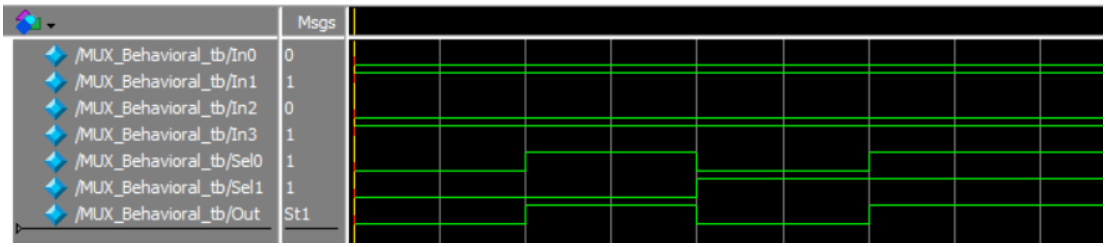
Behavioral Style에서는 case문을 이용하여 간단하게 각 입력에 따른 출력을 지정할 수 있었습니다. default문을 사용하여 불필요한 latch가 생기지 않도록 만들었습니다.

- MUX_Behavioral_testbench

```
`timescale 1ns/100ps

module MUX_Behavioral_tb;

//input
reg In0, In1, In2, In3;
reg Sel0, Sel1;

//output
wire Out;

//MUX_Behavioral module
MUX_Behavioral Mux(Out, In0, In1, In2, In3, Sel1, Sel0);

//input signal begin
initial begin
    In0 = 1'b0;
    In1 = 1'b1;
    In2 = 1'b0;
    In3 = 1'b1;

    Sel1 = 1'b0;
    Sel0 = 1'b0;

    #10
    Sel0 = 1'b1;
    #10
    Sel0 = 1'b0;
    Sel1 = 1'b1;
    #10
    Sel0 = 1'b1;
    #10;

   end
endmodule
```

Timescale는 1ns/100ps, 입력은 register로 선언, 출력은 wire로 선언 후, testbench를 실행시켰습니다. #10 간격을 두고 Sel 입력을 00, 01, 10, 11 순으로 주었습니다.

- 결과



Sel 입력을 00, 01, 10, 11로 변화시키자, 출력으로 In0, In1, In2, In3 순으로 값이 나오는 것을 확인할 수 있었습니다.

(4) Decoder_Structural

- Decoder_Structural Top module

```
1   module Decoder_Structural(Out0, Out1, Out2, Out3, In0, In1);
2
3   //input & output
4   output Out0, Out1, Out2, Out3; //4bit
5   input In0, In1; //2bit
6
7   wire NotIn0, NotIn1;
8
9   not not0 (NotIn0, In0);
10  not not1 (NotIn1, In1);
11
12  and and0 (Out0, NotIn0, NotIn1);
13  and and1 (Out1, In0, NotIn1);
14  and and2 (Out2, NotIn0, In1);
15  and and3 (Out3, In0, In1);
16
17  endmodule
18
```

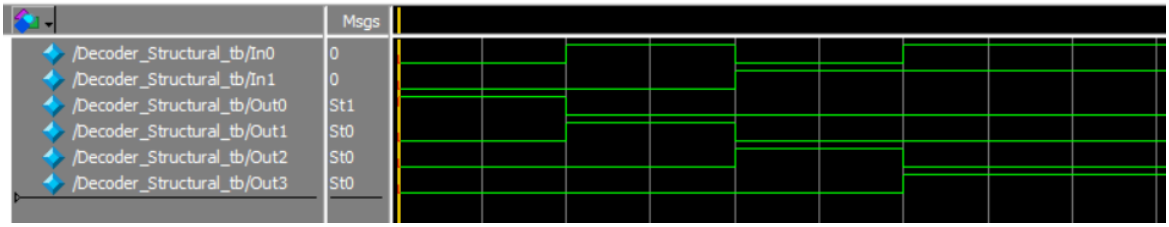
우선 각 입출력을 선언 후, 위의 Logic diagram에 따라 필요한 게이트들을 모두 선언 후 입출력을 연결해 주었습니다.

- Decoder_Structural_testbench

```
1   `timescale 1ns/100ps
2
3   module Decoder_Structural_tb;
4
5   //input
6   reg In0, In1;
7   //In0 is lowest bit
8
9   //output
10  wire Out0, Out1, Out2, Out3;
11
12  //Decoder_Structural module
13  Decoder_Structural Dec(Out0, Out1, Out2, Out3, In0, In1);
14
15  //input signal begin
16  initial begin
17
18      In0 = 1'b0;
19      In1 = 1'b0;
20
21      #10
22      In0 = 1'b1;
23      #10
24      In0 = 1'b0;
25      In1 = 1'b1;
26      #10
27      In0 = 1'b1;
28      #10;
29
30  end
31  endmodule
32
```

Timescale는 1ns/100ps, 입력은 register로 선언, 출력은 wire로 선언 후, testbench를 실행시켰습니다. #10 간격을 두고 In 입력을 00, 01, 10, 11 순으로 주었습니다.

- 결과



In 입력을 00, 01, 10, 11로 변화시키자, 출력으로 Out0은 1, 0, 0, 0, Out1은 0, 1, 0, 0, Out2는 0, 0, 1, 0, Out3은 0, 0, 0, 1로 값이 나오는 것을 확인할 수 있었습니다.

(5) Decoder_Dataflow

- Decoder_Dataflow Top module

```
module Decoder_Dataflow(Out0, Out1, Out2, Out3, In);

//input & output
output Out0, Out1, Out2, Out3;
input [1:0] In; //2bit
wire Out0, Out1, Out2, Out3;

//define module
assign Out0 =
   (In == 2'b00) ? 1'b1 : 1'b0;

assign Out1 =
   (In == 2'b01) ? 1'b1 : 1'b0;

assign Out2 =
   (In == 2'b10) ? 1'b1 : 1'b0;

assign Out3 =
   (In == 2'b11) ? 1'b1 : 1'b0;

endmodule
```

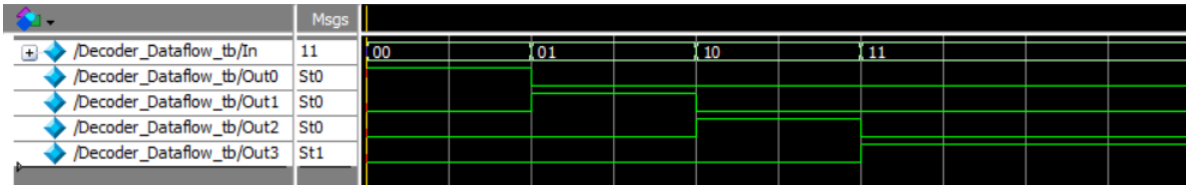
Structural Style과는 다르게, In 입력을 2비트짜리 신호 한 묶음으로 받습니다. 3항 연산자를 이용하여 각 게이트의 선언 없이 간단하게 입력 신호에 따른 출력을 구현하였습니다.

- Decoder_Dataflow_testbench

```
`timescale 1ns/100ps

module Decoder_Dataflow_tb;

//input
reg [1:0] In;

//output
wire Out0, Out1, Out2, Out3;

//Decoder_Dataflow module
Decoder_Dataflow Dec(Out0, Out1, Out2, Out3, In);

//input signal begin
initial begin
   In = 2'b00;

   #10
   In = 2'b01;
   #10
   In = 2'b10;
   #10
   In = 2'b11;
   #10;
end
endmodule
```

Timescale는 1ns/100ps, 입력은 register로 선언, 출력은 wire로 선언 후, testbench를 실행시켰습니다. #10 간격을 두고 In 입력을 00, 01, 10, 11 순으로 주었습니다.

- 결과



In 입력을 00, 01, 10, 11로 변화시키자, 출력으로 Out0은 1, 0, 0, 0, Out1은 0, 1, 0, 0, Out2는 0, 0, 1, 0, Out3은 0, 0, 0, 1로 값이 나오는 것을 확인할 수 있었습니다.

(6) Decoder_Behavioral

- Decoder_Behavioral Top module

```
1   module Decoder_Behavioral(Out0, Out1, Out2, Out3, In0, In1);
2
3   //input & output
4   output Out0, Out1, Out2, Out3;
5   input In0, In1;
6   reg Out0, Out1, Out2, Out3;
7
8   always @(In0 or In1) //while signal is exist
9      begin
10        case ({In1, In0}) //concatenate In1, In0
11        2'b00 : Out0 = 1'b1;
12        default : Out0 = 1'b0;
13        endcase
14     end
15
16  always @(In0 or In1) //while signal is exist
17     begin
18        case ({In1, In0}) //concatenate In1, In0
19        2'b01 : Out1 = 1'b1;
20        default : Out1 = 1'b0;
21        endcase
22     end
23
24  always @(In0 or In1) //while signal is exist
25     begin
26        case ({In1, In0}) //concatenate In1, In0
27        2'b10 : Out2 = 1'b1;
28        default : Out2 = 1'b0;
29        endcase
30     end
31
32  always @(In0 or In1) //while signal is exist
33     begin
34        case ({In1, In0}) //concatenate In1, In0
35        2'b11 : Out3 = 1'b1;
36        default : Out3 = 1'b0;
37        endcase
38     end
39  endmodule
```

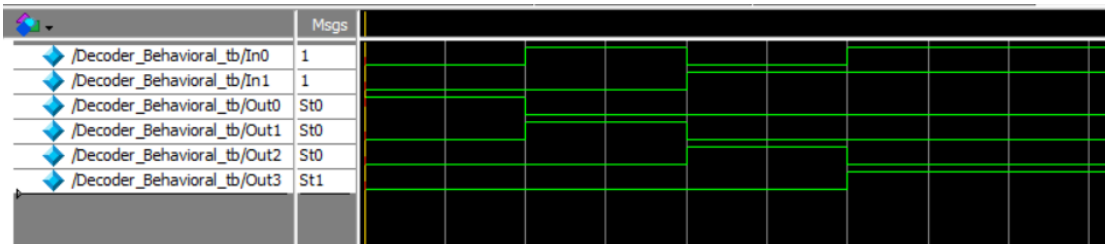
Behavioral Style에서는 case문을 이용하여 간단하게 각 입력에 따른 출력을 지정할 수 있었습니다. default문을 사용하여 불필요한 latch가 생기지 않도록 만들었습니다.

- Decoder_Behavioral_testbench

```
1   `timescale 1ns/100ps
2
3   module Decoder_Behavioral_tb;
4
5   //input
6   reg In0, In1;
7
8   //output
9   wire Out0, Out1, Out2, Out3;
10
11  //Decoder_Behavioral module
12  Decoder_Behavioral Dec(Out0, Out1, Out2, Out3, In0, In1);
13
14  //input signal begin
15  initial begin
16      In0 = 1'b0;
17      In1 = 1'b0;
18
19      #10
20      In0 = 1'b1;
21      #10
22      In0 = 1'b0;
23      In1 = 1'b1;
24      #10
25      In0 = 1'b1;
26      #10;
27
28  end
29  endmodule
```

Timescale는 1ns/100ps, 입력은 register로 선언, 출력은 wire로 선언 후, testbench를 실행시켰습니다. #10 간격을 두고 In 입력을 00, 01, 10, 11 순으로 주었습니다.

- 결과



In 입력을 00, 01, 10, 11로 변화시키자, 출력으로 Out0은 1, 0, 0, 0, Out1은 0, 1, 0, 0, Out2는 0, 0, 1, 0, Out3은 0, 0, 0, 1로 값이 나오는 것을 확인할 수 있었습니다.

4. Evaluation

MUX, Decoder 두 가지 모듈을 Structural, Dataflow, Behavioral Style로 구현 후 출력을 확인해보니, 모두 원하는 결과값을 출력함을 확인해 볼 수 있었습니다.

5. Discussions

Structural Style는 하드웨어 설계도 그대로 구현을 하는 방식이다 보니, 구조가 비교적 단순하고, 게이트가 적게 쓰이는 모듈을 구현 시 효율이 좋아 보였습니다.

Dataflow 방식과 Behavioral 방식은 일반적인 프로그래밍적인 방식을 이용하여 다소 내부 구조가 복잡한 모듈을 구현 시, 생산 효율성이 높아 보였습니다.