LDLAB3

수리과학부 2019-16022 박채연

0. Discussion

(1) Concepts I learned during the lab session

Hardware Description Language에는 대표적으로 IEEE 표준인 Verilog가 있는데, C 언어와 유사한 문법을 가지고 있으며 프로그래밍 방법은 크게 세 가지가 있다는 것을 배웠다. Structural Description 방법은 schematic을 대체하는 것이며 Data Flow Description 방법은 진리표를 대체하는 방법이다. 마지막으로 Behavioral Description은 'How' 에 초점을 둔 프로그래밍 방법이다.

Verilog의 기본 문법에 대해 학습하였다. Verilog는 C언어와는 다르게 Blocking assignment와 Non-blocking assignment를 모두 지원하는데, Blocking assignment는 프로그래밍 언어에서와 같이 순차적으로 실행되는 반면 Non-blocking assignment는 병렬적으로 실행된다는 것을 새로 알게되었다. 또한, 일반적인 프로그래밍 언어와는 다르게 propagation delay 기능도 지원함을 배웠다.

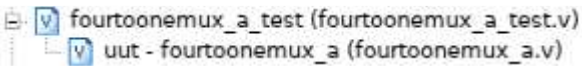
Xilinx 기본 조작법을 배웠다. 모듈을 Structural Description, Data Flow Description, Behavioral Description 중 한 가지 방법을 이용하여 프로그래밍을 한 후 구현한 모듈이 설계대로 올바르게 작동하는지 검증하기 위해 test bench code를 작성해야 한다. 시뮬레이션을 수행하여 정상적으로 작동하는지 확인하고 그렇지 않은 경우 수정하는 과정을 거쳐야 한다.

(2) Any errors I made

74x139 모듈을 세 가지 프로그래밍 방법(Structural Description, Data Flow Description, Behavioral Description)을 이용하여 구현하고 나서 test bench code를 작성하였다. Simulation을 실행시켰을 때 처음에는 테스트 결과가 정상적으로 나오다가 테스트 코드를 변경하지 않았음에도 불구하고 어느 순간 시뮬레이션 결과가 아예 나오지 않았는데 원인을 알지 못해 헤맸다.

74x139 하프 모듈은 data input이 2개, control input이 1개, output이 4개로 이루어져있다. 2-to-4 decoder 역시 data input이 2개, control input이 1개, output이 4개로 이루어져 있지만 output은 74x139 하프 모듈의 output과는 다르다. output을 고려하지 않은 채 input과 output 개수만을 보고 74x139 하프 모듈이 2-to-4 decoder와 동일하다고 생각했고 이를 이용하여 3-to-8 decoder를 구현하였다. 결과를 정리하는 과정에서 해당 모듈이 올바르게 동작하지 않는다는 것을 발견하고 수정하였다.

(3) How to correct my errors



fourtoonemux_a.v 파일을 생성하여 fourtoonemux_a 모듈을 구현하고 fourtoonemux_a_test.v 라는 테스트 파일을 생성하여 해당 모듈에 대한 테스트 코드를 작성하면, Design의 Simulation 탭에

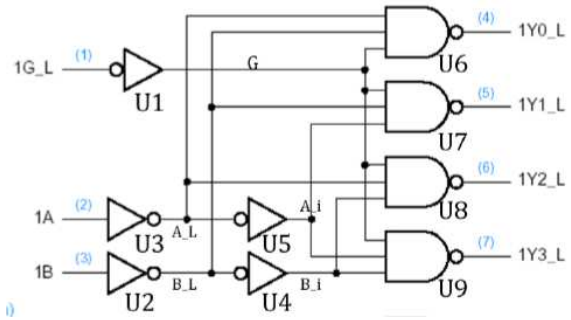
서는 이 두 파일이 위와 같이 계층적인 구조로 나타난다. 테스트를 수행할 때는 'fourtoonemux_a_test (fourtoonemux_a_test.v)'를 클릭하고 시뮬레이션을 실행시켜야 한다. 만약 'uut-fourtoonemux_a (fourtoonemux_a.v)'을 클릭하고 시뮬레이션을 실행할 경우 시뮬레이션 결과가 아예 나오지 않는데, 이 때문에 어느 순간 테스트가 정상적으로 실행되지 않았음을 깨닫고 다시 테스트를 정상적으로 실행시킬 수 있었다.

보고서를 작성하면서 74x139 하프 모듈이 2-to-4 decoder와 output이 달라 이를 이용하여 구현한 3-to-8 decoder 역시 올바르게 동작하지 않음을 확인하였다. 노트북에 Xilinx를 설치하려고 시도하였으나 실패했고, 다음날 302동 310-2호에 출석하여 2-to-4 decoder를 Data Flow Description 방법을 이용하여 새로 구현하였다. 또한, 2개의 2-to-4 decoder에 각각 enable로 G와 ~G을 연결함으로써 3-to-8 decoder를 구현하고 테스트를 수행함으로써 오류를 고칠 수 있었다.

1. Lab practice

(1) Implementations of 74x139 with 3 method and simulations

- Structural Description



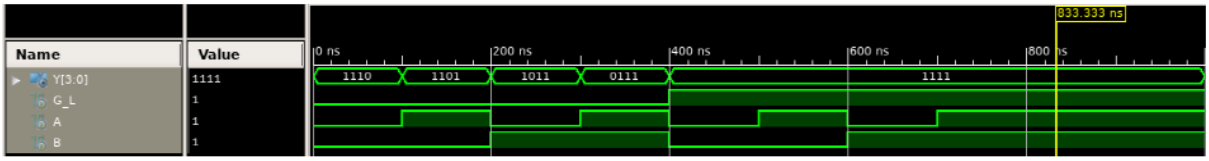
위의 Schematic을 참고하여 Structural Description 방법으로 구현한 1/2 * 74x139 모듈에 대한 코드는 아래와 같다.

```
21  module v74x139h_a(
22      input G_L,
23      input A,
24      input B,
25      output [3:0] Y
26      );
27
28     wire N_A, N_B, N_G;
29
30     not T1(N_G, G_L);
31     not T2(N_A, A);
32     not T3(N_B, B);
33
34     nand T4(Y[0], N_G, N_A, N_B);
35     nand T5(Y[1], N_G, A, N_B);
36     nand T6(Y[2], N_G, N_A, B);
37     nand T7(Y[3], N_G, A, B);
38
39  endmodule
```

아래와 같이 테스트 코드를 작성하여 input 값에 따른 8가지 케이스 모두에 대해 검증하였다.

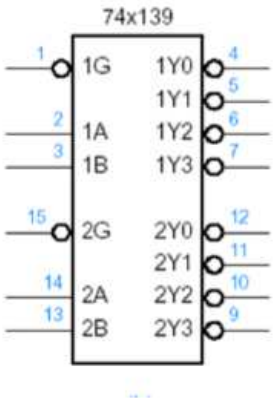
```
25  module v74x139_a_test;
26
27      // Inputs
28      reg G_L;
29      reg A;
30      reg B;
31
32      // Outputs
33      wire [3:0] Y;
34
35      // Instantiate the Unit Under Test (UUT)
36      v74x139h_a uut (
37          .G_L(G_L),
38          .A(A),
39          .B(B),
40          .Y(Y)
41      );
42
43      initial begin
44          // Initialize Inputs
45          G_L = 0;
46          A = 0;
47          B = 0;
48
49          // Wait 100 ns for global reset to finish
50          #100;
51
52          // Add stimulus here
53          G_L = 0;
54          A = 1;
55          B = 0;
56
57          #100;
58          G_L = 0;
59          A = 0;
60          B = 1;

62          #100;
63          G_L = 0;
64          A = 1;
65          B = 1;
66
67          #100;
68          G_L = 1;
69          A = 0;
70          B = 0;
71
72          #100;
73          G_L = 1;
74          A = 1;
75          B = 0;
76
77          #100;
78          G_L = 1;
79          A = 0;
80          B = 1;
81
82          #100;
83          G_L = 1;
84          A = 1;
85          B = 1;
86
87          #100;
88
89      end
90
91  endmodule
```

위의 테스트 코드를 통해 Structural Description 방법으로 구현한 1/2 * 74x139 모듈에 대한 시뮬레이션 결과를 확인하면 아래와 같다.



즉, G_L, A, B 값에 따른 output Y 값을 표로 정리하면 아래와 같은데 구현한 half 모듈이 올바르게 동작하는 모습을 확인할 수 있다.

G_L	A	B	Y[3:0]
0	0	0	1110
0	1	0	1101
0	0	1	1011
0	1	1	.0111
1	0	0	1111
1	1	0	1111
1	0	1	1111
1	1	1	1111



위의 Block Diagram을 참고하여 지금까지 Structural Description 방법으로 구현한 1/2 * 74x139 모듈 2개를 이용해 74x139 모듈을 구현할 수 있는데, 이에 대한 코드는 다음과 같다.

```
21  module v74x139h_aa(
22      input G1,
23      input G2,
24      input B1,
25      input B2,
26      input A1,
27      input A2,
28      output [3:0] Y1,
29      output [3:0] Y2
30      );
31
32     v74x139h_a T1(.G_L(G1), .A(A1), .B(B1), .Y(Y1));
33     v74x139h_a T2(.G_L(G2), .A(A2), .B(B2), .Y(Y2));
34
35  endmodule
```

이때, 74x139 모듈의 input 개수가 6개이므로 총 64개의 케이스가 존재한다. 그 중 A1 = B1 = 1,

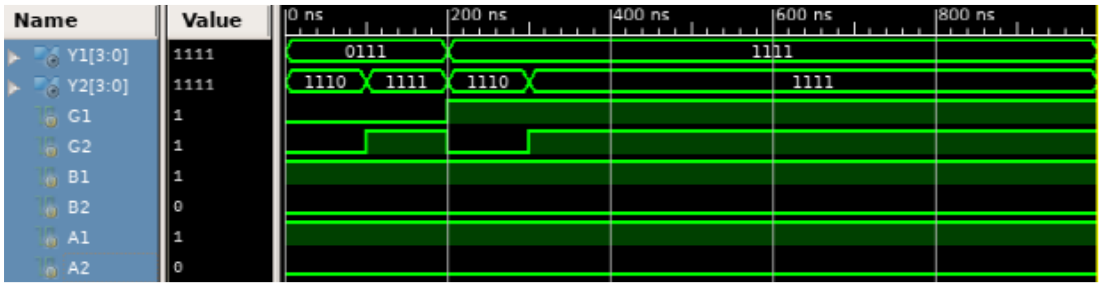
A2 = B2 = 0으로 고정시키고 control input인 G1과 G2 값을 변경해가며 Structural Description 방법으로 구현한 74x139 모듈을 테스트하는 코드를 아래와 같이 작성하였다.

```
25  module v74x139h_aa_test;
26
27      // Inputs
28      reg G1;
29      reg G2;
30      reg B1;
31      reg B2;
32      reg A1;
33      reg A2;
34
35      // Outputs
36      wire [3:0] Y1;
37      wire [3:0] Y2;
38
39      // Instantiate the Unit Under Test (UUT)
40      v74x139h_aa uut (
41          .G1(G1),
42          .G2(G2),
43          .B1(B1),
44          .B2(B2),
45          .A1(A1),
46          .A2(A2),
47          .Y1(Y1),
48          .Y2(Y2)
49      );
50
51      initial begin
52          // Initialize Inputs
53          G1 = 0;
54          G2 = 0;
55          B1 = 1;
56          B2 = 0;
57          A1 = 1;
58          A2 = 0;
```

```
60          // Wait 100 ns for global reset to finish
61          #100;
62
63          // Add stimulus here
64
65          G1 = 0;
66          G2 = 1;
67          B1 = 1;
68          B2 = 0;
69          A1 = 1;
70          A2 = 0;
71
72          #100;
73          G1 = 1;
74          G2 = 0;
75          B1 = 1;
76          B2 = 0;
77          A1 = 1;
78          A2 = 0;
```

```
80          #100;
81          G1 = 1;
82          G2 = 1;
83          B1 = 1;
84          B2 = 0;
85          A1 = 1;
86          A2 = 0;
87
88      end
89
90  endmodule
```

시뮬레이션 결과는 아래와 같다.



위의 시뮬레이션 결과를 표로 정리하면 아래와 같은데 구현한 74x139 모듈이 올바르게 작동함을 알 수 있다. 이때, A1 = B1 = 1, A2 = B2 = 0이다.

G1	G2	Y1	Y2
0	0	0111	1110
0	1	0111	1111
1	0	1111	1110
1	1	1111	1111

- Data Flow Description

G	A	B	Y[3:0]
0	0	0	1110
0	1	0	1101
0	0	1	1011
0	1	1	.0111
1	- (don't care)	- (don't care)	1111

위의 진리표를 참고하여 Data Flow Description 방법으로 작성한 1/2 * 72x139 모듈에 대한 코드는 아래와 같다.

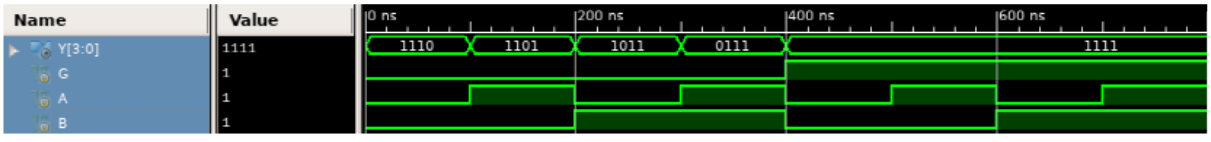
```
21  module v74x139h_b(
22      input G,
23      input A,
24      input B,
25      output [3:0] Y
26      );
27
28     wire [1:0] sel;
29     wire [3:0] out;
30
31     assign sel = {B, A};
32     assign Y = ~out;
33
34     assign out = (sel == 2'b00 && G == 1'b0) ? 4'b0001:
35                  (sel == 2'b01 && G == 1'b0) ? 4'b0010:
36                  (sel == 2'b10 && G == 1'b0) ? 4'b0100:
37                  (sel == 2'b11 && G == 1'b0) ? 4'b1000:
38                  4'b0000;
39
40  endmodule
```

아래와 같이 테스트 코드를 작성하여 input 값에 따른 8가지 케이스 모두에 대해 검증하였다.

```
25  module v74x139h_b_test;
26
27      // Inputs
28      reg G;
29      reg A;
30      reg B;
31
32      // Outputs
33      wire [3:0] Y;
34
35      // Instantiate the Unit Under Test (UUT)
36      v74x139h_b uut (
37          .G(G),
38          .A(A),
39          .B(B),
40          .Y(Y)
41      );
42
43      initial begin
44          // Initialize Inputs
45          G = 0;
46          A = 0;
47          B = 0;
48
49          // Wait 100 ns for global reset to finish
50          #100;
51
52          // Add stimulus here
53          G = 0;
54          A = 1;
55          B = 0;
56
57          #100;
58          G = 0;
59          A = 0;
60          B = 1;
```

```
62          #100;
63          G = 0;
64          A = 1;
65          B = 1;
66
67          #100;
68          G = 1;
69          A = 0;
70          B = 0;
71
72          #100;
73          G = 1;
74          A = 1;
75          B = 0;
76
77          #100;
78          G = 1;
79          A = 0;
80          B = 1;
81
82          #100;
83          G = 1;
84          A = 1;
85          B = 1;
86
87          #100;
88      end
89
90  endmodule
```

위의 테스트 코드를 통해 Data Flow Description 방법으로 구현한 1/2 * 74x139 모듈에 대한 시뮬레이션 결과를 확인하면 아래와 같다. 시뮬레이션 결과가 앞에서 제시한 진리표와 같이 올바르게 동작하는 것을 확인할 수 있다.



지금까지 Data Flow Description 방법으로 구현한 1/2 * 74x139 모듈 2개를 이용하여 74x139 모듈

을 구현할 수 있는데, 이에 대한 코드는 다음과 같다.

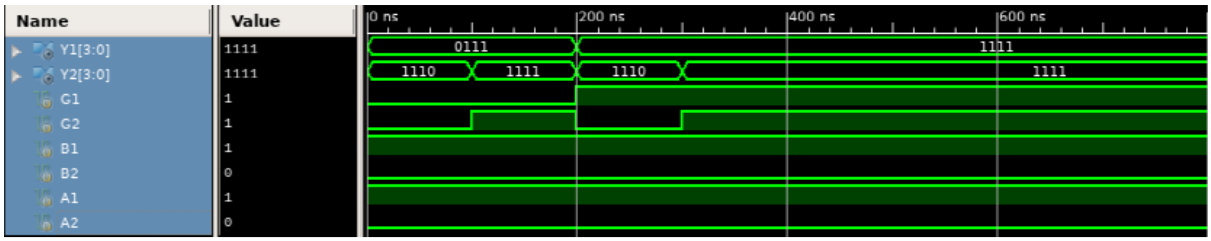
```
module v74x139h_bb(
    input G1,
    input G2,
    input B1,
    input B2,
    input A1,
    input A2,
    output [3:0] Y1,
    output [3:0] Y2
    );

   v74x139h_b T1(.G(G1), .A(A1), .B(B1), .Y(Y1));
   v74x139h_b T2(.G(G2), .A(A2), .B(B2), .Y(Y2));

endmodule
```

이때, 74x139 모듈의 input 개수가 6개이므로 총 64개의 케이스가 존제한다. 그 중 A1 = B1 = 1, A2 = B2 = 0으로 고정시키고 G1과 G2 값을 변경해가며 Data Flow Description 방법으로 구현한 74x139 모듈을 테스트하는 코드를 아래와 같이 작성하였다.

```
module v74x139h_bb_test;

   // Inputs
   reg G1;
   reg G2;
   reg B1;
   reg B2;
   reg A1;
   reg A2;

   // Outputs
   wire [3:0] Y1;
   wire [3:0] Y2;

   // Instantiate the Unit Under Test (UUT)
   v74x139h_bb uut (
      .G1(G1),
      .G2(G2),
      .B1(B1),
      .B2(B2),
      .A1(A1),
      .A2(A2),
      .Y1(Y1),
      .Y2(Y2)
   );

   initial begin
      // Initialize Inputs

      G1 = 0;
      G2 = 0;
      B1 = 1;
      B2 = 0;
      A1 = 1;
      A2 = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here

      G1 = 0;
      G2 = 1;
      B1 = 1;
      B2 = 0;
      A1 = 1;
      A2 = 0;

      #100;
      G1 = 1;
      G2 = 0;
      B1 = 1;
      B2 = 0;
      A1 = 1;
      A2 = 0;
```

```
80        #100;
81        G1 = 1;
82        G2 = 1;
83        B1 = 1;
84        B2 = 0;
85        A1 = 1;
86        A2 = 0;
87
88    end
89
90  endmodule
```

시뮬레이션 결과는 아래와 같다.



위의 시뮬레이션 결과를 표로 정리하면 아래와 같은데 Data Flow Description 방법으로 구현한 74x139 모듈이 올바르게 작동함을 알 수 있다. 이때, A1 = B1 = 1, A2 = B2 = 0이다.

G1	G2	Y1	Y2
0	0	0111	1110
0	1	0111	1111
1	0	1111	1110
1	1	1111	1111

- Behavioral Description

Behavioral Description 방법으로 작성한 1/2 * 72x139 모듈에 대한 코드는 아래와 같다.

```
21  module v74x139h_c(
22      input G,
23      input A,
24      input B,
25      output [3:0] Y
26      );
27
28      wire [1:0] sel;
29      reg [3:0] out;
30
31      assign sel = {B, A};
32      assign Y = ~out;
33
34      always@(G or sel)
35        begin
36          if (G == 1'b0)
37            begin
38              case(sel)
39                  2'b00 : out = 4'b0001;
40                  2'b01 : out = 4'b0010;
41                  2'b10 : out = 4'b0100;
42                  2'b11 : out = 4'b1000;
43              endcase
44            end
```

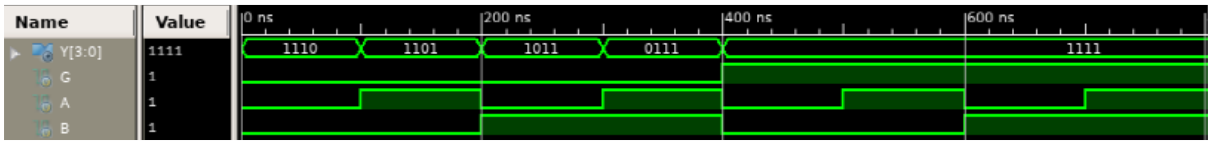
```
45          else
46              begin
47                  out=4'b0000;
48              end
49      end
50
51  endmodule
```

아래와 같이 테스트 코드를 작성하여 input 값에 따른 8가지 케이스 모두에 대해 검증하였다.

```
25  module v74x139h_c_test;
26
27      // Inputs
28      reg G;
29      reg A;
30      reg B;
31
32      // Outputs
33      wire [3:0] Y;
34
35      // Instantiate the Unit Under Test (UUT)
36      v74x139h_c uut (
37          .G(G),
38          .A(A),
39          .B(B),
40          .Y(Y)
41      );
42
43      initial begin
44          // Initialize Inputs
45          G = 0;
46          A = 0;
47          B = 0;
48
49          // Wait 100 ns for global reset to finish
50          #100;
51
52          // Add stimulus here
53          G = 0;
54          A = 1;
55          B = 0;
56
57          #100;
58          G = 0;
59          A = 0;
60          B = 1;
```

```
62          #100;
63          G = 0;
64          A = 1;
65          B = 1;
66
67          #100;
68          G = 1;
69          A = 0;
70          B = 0;
71
72          #100;
73          G = 1;
74          A = 1;
75          B = 0;
76
77          #100;
78          G = 1;
79          A = 0;
80          B = 1;
81
82          #100;
83          G = 1;
84          A = 1;
85          B = 1;
86
87          #100;
88      end
89
90  endmodule
```

위의 테스트 코드를 통해 Behavioral Description 방법으로 구현한 1/2 * 74x139 모듈에 대한 시뮬레이션 결과를 확인하면 아래와 같다.



G, A, B 값에 따른 output Y 값을 표로 정리하면 아래와 같은데 Behavioral Description 방법으로 구현한 1/2 * 74x139 모듈이 올바르게 동작하는 모습을 확인할 수 있다.

G	A	B	Y[3:0]
0	0	0	1110
0	1	0	1101
0	0	1	1011
0	1	1	.0111
1	0	0	1111
1	1	0	1111
1	0	1	1111
1	1	1	1111

지금까지 Behavioral Description 방법으로 구현한 1/2 * 74x139 모듈 2개를 이용하여 74x139 모듈을 구현할 수 있는데, 이에 대한 코드는 다음과 같다.

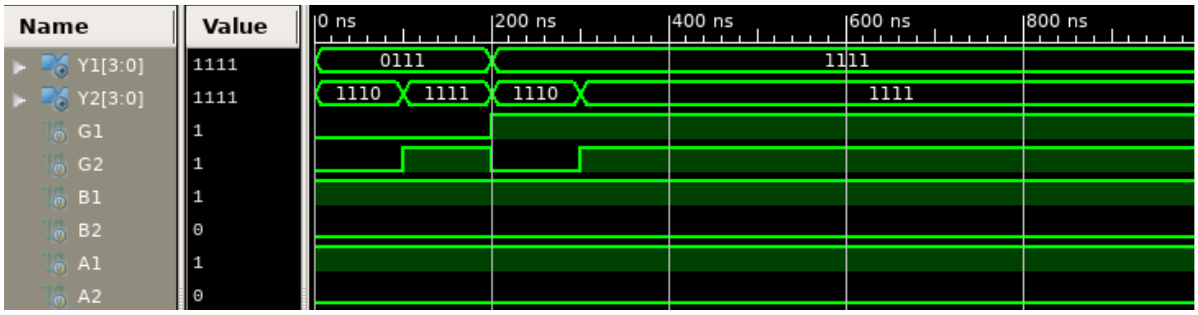
```
21  module v74x139h_cc(
22      input G1,
23      input G2,
24      input B1,
25      input B2,
26      input A1,
27      input A2,
28      output [3:0] Y1,
29      output [3:0] Y2
30      );
31
32     v74x139h_c T1(.G(G1), .A(A1), .B(B1), .Y(Y1));
33     v74x139h_c T2(.G(G2), .A(A2), .B(B2), .Y(Y2));
34
35  endmodule
```

이때, 74x139 모듈의 input 개수가 6개이므로 총 64개의 케이스가 존제한다. 그 중 A1 = B1 = 1, A2 = B2 = 0으로 고정시키고 G1과 G2 값을 변경해가며 Behavioral Description 방법으로 구현한 74x139 모듈을 테스트하는 코드를 아래와 같이 작성하였다.

```
25  module v74x139h_cc_test;
26
27      // Inputs
28      reg G1;
29      reg G2;
30      reg B1;
31      reg B2;
32      reg A1;
33      reg A2;
34
35      // Outputs
36      wire [3:0] Y1;
37      wire [3:0] Y2;
38
39      // Instantiate the Unit Under Test (UUT)
40      v74x139h_cc uut (
41          .G1(G1),
42          .G2(G2),
43          .B1(B1),
44          .B2(B2),
45          .A1(A1),
46          .A2(A2),
47          .Y1(Y1),
48          .Y2(Y2)
49      );
50
51      initial begin
52          // Initialize Inputs
53          G1 = 0;
54          G2 = 0;
55          B1 = 1;
56          B2 = 0;
57          A1 = 1;
58          A2 = 0;
```

```
61          #100;
62
63          // Add stimulus here
64
65          G1 = 0;
66          G2 = 1;
67          B1 = 1;
68          B2 = 0;
69          A1 = 1;
70          A2 = 0;
71
72          #100;
73          G1 = 1;
74          G2 = 0;
75          B1 = 1;
76          B2 = 0;
77          A1 = 1;
78          A2 = 0;
79
80          #100;
81          G1 = 1;
82          G2 = 1;
83          B1 = 1;
84          B2 = 0;
85          A1 = 1;
86          A2 = 0;
87
88          #100;
89
90      end
91
92  endmodule
```

시뮬레이션 결과는 아래와 같다.



위의 시뮬레이션 결과를 표로 정리하면 다음과 같은데 Behavioral Description 방법으로 구현한 74x139 모듈이 올바르게 작동함을 알 수 있다. 이때, A1 = B1 = 1, A2 = B2 = 0이다.

G1	G2	Y1	Y2
0	0	0111	1110
0	1	0111	1111
1	0	1111	1110
1	1	1111	1111

(2) Implment a 3-to-8 decoder using 2-to-4 decoders only and simulate it (one of 3 designing methods)

G	A	B	Y
1	0	0	1000
1	0	1	0100
1	1	0	0010
1	1	1	0001
0	- (don't care)	- (don't care)	0000

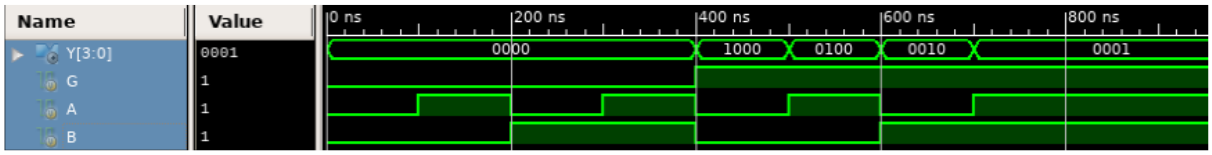
우선, 위의 진리표를 참고하여 Data Flow Description 방법을 이용하여 2-to-4 decoder를 아래와 같이 구현하였다.

```
21  module twotofour(
22      input G,
23      input A,
24      input B,
25      output [3:0] Y
26      );
27
28     wire [1:0] sel;
29     wire [3:0] out;
30
31     assign sel = {B, A};
32     assign Y = out;
33
34     assign out = (sel == 2'b00 && G == 1'b1) ? 4'b1000:
35                  (sel == 2'b01 && G == 1'b1) ? 4'b0100:
36                  (sel == 2'b10 && G == 1'b1) ? 4'b0010:
37                  (sel == 2'b11 && G == 1'b1) ? 4'b0001:
38                  4'b0000;
39
40  endmodule
```

Data Flow Description 방법으로 구현한 2-to-4 decoder가 올바르게 동작하는지 확인하기 위해 아래와 같이 테스트 코드를 작성하여 input 값에 따른 8가지 케이스 모두에 대해 검증하였다.

```
module twotofour_test;

    // Inputs
    reg G;
    reg A;
    reg B;

    // Outputs
    wire [3:0] Y;

    // Instantiate the Unit Under Test (UUT)
    twotofour uut (
        .G(G),
        .A(A),
        .B(B),
        .Y(Y)
    );

    initial begin
        // Initialize Inputs
        G = 0;
        A = 0;
        B = 0;

        // Add stimulus here
        #100;

        G = 0;
        A = 1;
        B = 0;

        #100;

        G = 0;
        A = 0;
        B = 1;

        #100;

        G = 0;
        A = 1;
        B = 1;

        #100;

        G = 1;
        A = 0;
        B = 0;

        #100;
        G = 1;
        A = 1;
        B = 0;

        #100;
        G = 1;
        A = 0;
        B = 1;

        #100;
        G = 1;
        A = 1;
        B = 1;

        #100;
    end

endmodule
```

위의 테스트 코드를 통해 Data Flow Description 방법으로 구현한 2-to-4 decoder에 대한 시뮬레이션 결과를 확인하면 아래와 같다. Data Flow Description 방법으로 구현한 2-to-4 decoder가 앞에서 제시한 진리표대로 올바르게 동작하는 것을 확인할 수 있다.



다음으로, Data Flow Description 방법으로 구현한 위의 2-to-4 decoder를 활용하여 3-to-8 decoder를 아래의 코드를 통해 구현하였다. 2개의 2-to-4 decoder를 이용하면 각각의 decoder에 enable로 G와 ~G를 연결함으로써 3-to-8 decoder를 만들 수 있다.

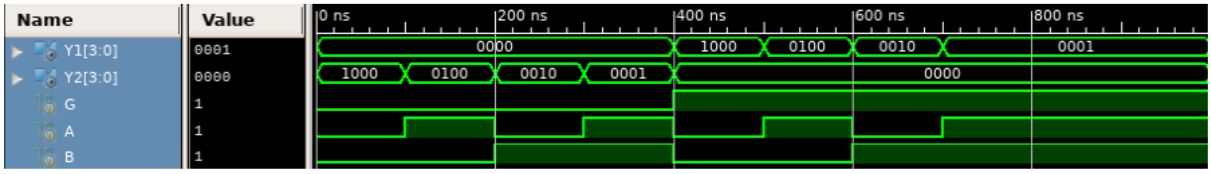
```
21  module threetoeight(
22      input G,
23      input A,
24      input B,
25      output [3:0] Y1,
26      output [3:0] Y2
27      );
28
29
30      twotofour T1(.G(G), .A(A), .B(B), .Y(Y1));
31      twotofour T2(.G(~G), .A(A), .B(B), .Y(Y2));
32
33
34
35  endmodule
```

Data Flow Description 방법으로 구현한 3-to-8 decoder가 올바르게 동작하는지 확인하기 위해 아래와 같이 테스트 코드를 작성하여 input 값에 따른 8가지 케이스 모두에 대해 검증하였다.

```
25  module threetoeight_test;
26
27     // Inputs
28     reg G;
29     reg A;
30     reg B;
31
32     // Outputs
33
34     wire [7:4] Y1;
35     wire [3:0] Y2;
36
37     // Instantiate the Unit Under Test (UUT)
38     threetoeight uut (
39        .G(G),
40        .A(A),
41        .B(B),
42        .Y1(Y1),
43        .Y2(Y2)
44     );
45
46     initial begin
47        // Initialize Inputs
48        G = 0;
49        A = 0;
50        B = 0;
51
52        // Wait 100 ns for global reset to finish
53        #100;
54
55        G = 0;
56        A = 1;
57        B = 0;
58
59        #100;
60
61        G = 0;
62        A = 0;
63        B = 1;
```

```
65          #100;
66
67          G = 0;
68          A = 1;
69          B = 1;
70
71          #100;
72
73          G = 1;
74          A = 0;
75          B = 0;
76
77          #100;
78          G = 1;
79          A = 1;
80          B = 0;
81
82          #100;
83          G = 1;
84          A = 0;
85          B = 1;
86
87          #100;
88          G = 1;
89          A = 1;
90          B = 1;
91
92          #100;
93
94      end
95
96  endmodule
```

위의 테스트 코드를 통해 Data Flow Description 방법으로 구현한 3-to-8 decoder에 대한 시뮬레이션 결과를 확인하면 아래와 같다.

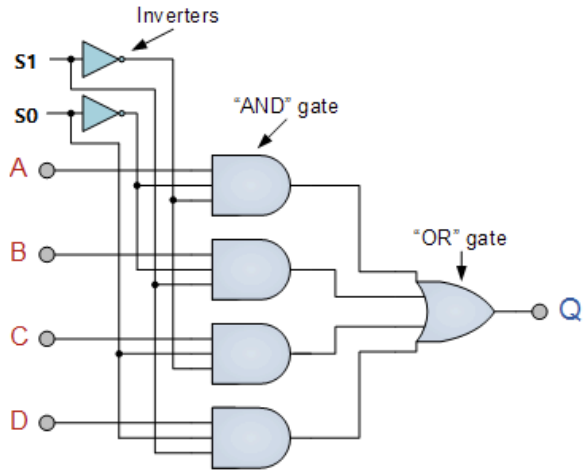


위의 시뮬레이션 결과를 표로 정리하면 아래와 같은데 Data Flow Description 방법으로 구현한 3-to-8 decoder가 올바르게 동작함을 확인할 수 있다.

G	A	B	Y (=Y1+Y2)
0	0	0	00001000
0	1	0	00000100
0	0	1	00000010
0	1	1	00000001
1	0	0	10000000
1	1	0	01000000
1	0	1	00100000
1	1	1	00010000

2. Implement 4-to-1 MUX with all the designing methods that we practiced and simulate it.

- Structural Description



위의 Circuit Diagram을 참고하여 Structural Description 방법으로 구현한 4-to-1 멀티플렉서에 대한 코드는 아래와 같다.

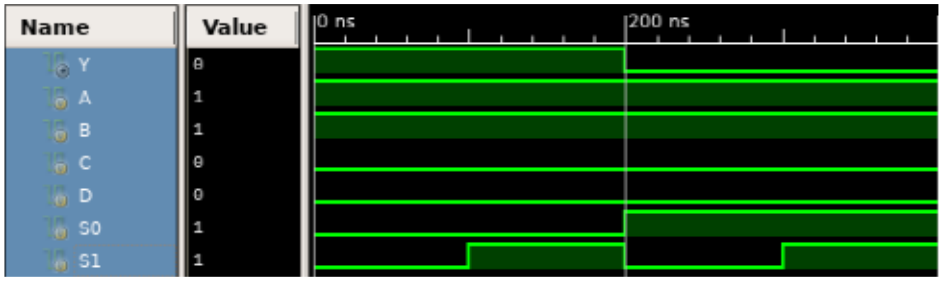
```
21  module fourtoonemux_a(
22      input A,
23      input B,
24      input C,
25      input D,
26      input S0,
27      input S1,
28      output Y
29      );
30
31      wire N_S0, N_S1;
32      wire [3:0] temp;
33
34     not T1(N_S0, S0);
35     not T2(N_S1, S1);
36
37     and T3(temp[0], A, N_S0, N_S1);
38     and T4(temp[1], B, N_S0, S1);
39     and T5(temp[2], C, S0, N_S1);
40     and T6(temp[3], D, S0, S1);
41
42     or(Y, temp[0], temp[1], temp[2], temp[3]);
43
44  endmodule
```

이때, 4-to-1 멀티플렉서의 data input과 control input이 총 6개이므로 64개의 케이스가 존제한다. 그 중 A = B = 1, C = D = 0으로 고정시키고 control input인 S0과 S1 값을 변경해가며 Structural Description 방법으로 구현한 4-to1 멀티플렉서를 테스트하는 코드를 아래와 같이 작성하였다.

```
25  module fourtoonemux_a_test;
26
27      // Inputs
28      reg A;
29      reg B;
30      reg C;
31      reg D;
32      reg S0;
33      reg S1;
34
35      // Outputs
36      wire Y;
37
38      // Instantiate the Unit Under Test (UUT)
39      fourtoonemux_a uut (
40          .A(A),
41          .B(B),
42          .C(C),
43          .D(D),
44          .S0(S0),
45          .S1(S1),
46          .Y(Y)
47      );
48
49      initial begin
50          // Initialize Inputs
51          A = 1;
52          B = 1;
53          C = 0;
54          D = 0;
55          S0 = 0;
56          S1 = 0;
57
58          // Wait 100 ns for global reset to finish
59          #100;
```

```
63          A = 1;
64          B = 1;
65          C = 0;
66          D = 0;
67          S0 = 0;
68          S1 = 1;
69
70          #100
71
72          A = 1;
73          B = 1;
74          C = 0;
75          D = 0;
76          S0 = 1;
77          S1 = 0;
78
79          #100
80          A = 1;
81          B = 1;
82          C = 0;
83          D = 0;
84          S0 = 1;
85          S1 = 1;
86
87          #100;
88      end
89
90  endmodule
```

시뮬레이션 결과는 아래와 같다.



위의 시뮬레이션 결과를 표로 정리하면 아래와 같은데 Structural Description 방법으로 구현한 4-to-1 멀티플렉서가 올바르게 동작함을 확인할 수 있다. (이때, A = B = 1, C = D = 0이다.)

S0	S1	Y
0	0	1 (=A)
0	1	1 (=B)
1	0	0 (=C)
1	1	0 (=D)

- Data Flow Description

S[1:0]	Y
00	A
01	B
10	C
11	D

위의 진리표를 참고하여 Data Flow Description 방법으로 구현한 4-to-1 멀티플렉서에 대한 코드는 아래와 같다.

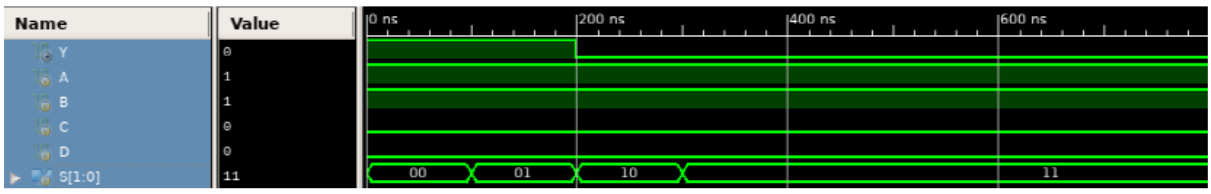
```
21  module fourtoonemux_b(
22      input A,
23      input B,
24      input C,
25      input D,
26      input [1:0] S,
27      output Y
28      );
29
30     assign Y = (S == 2'b00) ? A:
31                (S == 2'b01) ? B:
32                (S == 2'b10) ? C:
33                 D ;
34
35  endmodule
```

이때, 4-to-1 멀티플렉서의 data input과 control input이 총 6개이므로 64개의 케이스가 존제한다. 그 중 A = B = 1, C = D = 0으로 고정시키고 control input인 S0과 S1 값을 변경해가며 Data Flow Description 방법으로 구현한 4-to1 멀티플렉서를 테스트하는 코드를 아래와 같이 작성하였다.

```
25  module fourtoonemux_b_test;
26
27     // Inputs
28     reg A;
29     reg B;
30     reg C;
31     reg D;
32     reg [1:0] S;
33
34     // Outputs
35     wire Y;
36
37     // Instantiate the Unit Under Test (UUT)
38     fourtoonemux_b uut (
39        .A(A),
40        .B(B),
41        .C(C),
42        .D(D),
43        .S(S),
44        .Y(Y)
45     );
46
47     initial begin
48        // Initialize Inputs
49        A = 1;
50        B = 1;
51        C = 0;
52        D = 0;
53        S = 0;
54
55        // Wait 100 ns for global reset to finish
56        #100;
57
58        // Add stimulus here
59        A = 1;
60        B = 1;
61        C = 0;
62        D = 0;
63        S = 1;
```

```
65        #100;
66
67        A = 1;
68        B = 1;
69        C = 0;
70        D = 0;
71        S = 2;
72
73        #100;
74
75        A = 1;
76        B = 1;
77        C = 0;
78        D = 0;
79        S = 3;
80
81        #100;
82     end
83
84  endmodule
```

시뮬레이션 결과는 아래와 같은데, Data Flow Description 방법으로 구현한 4-to-1 멀티플렉서가 올바르게 동작함을 확인할 수 있다.



- Behavioral Description

Behavioral Description 방법으로 구현한 4-to-1 멀티플렉서에 대한 코드는 아래와 같다.

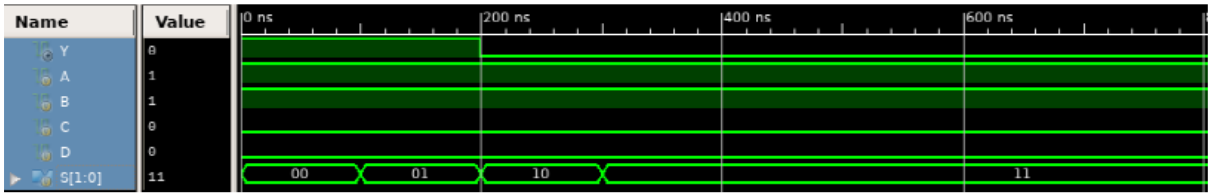
```
21  module fourtoonemux_c(
22      input A,
23      input B,
24      input C,
25      input D,
26      input [1:0] S,
27      output Y
28      );
29
30      reg out;
31
32      assign Y = out;
33
34      always@(S or A or B or C or D)
35        begin
36           case(S)
37              2'b00 : out = A;
38              2'b01 : out = B;
39              2'b10 : out = C;
40              2'b11 : out = D;
41           endcase
42        end
43
44  endmodule
```

이때, 4-to-1 멀티플렉서의 data input과 control input이 총 6개이므로 64개의 케이스가 존제한다. 그 중 A = B = 1, C = D = 0으로 고정시키고 control input인 S0과 S1 값을 변경해가며 Behaavioral Description 방법으로 구현한 4-to1 멀티플렉서를 테스트하는 코드를 아래와 같이 작성하였다.

```
25  module fourtoonemux_c_test;
26
27      // Inputs
28      reg A;
29      reg B;
30      reg C;
31      reg D;
32      reg [1:0] S;
33
34      // Outputs
35      wire Y;
36
37      // Instantiate the Unit Under Test (UUT)
38      fourtoonemux_c uut (
39          .A(A),
40          .B(B),
41          .C(C),
42          .D(D),
43          .S(S),
44          .Y(Y)
45      );
46
47      initial begin
48          // Initialize Inputs
49          A = 1;
50          B = 1;
51          C = 0;
52          D = 0;
53          S = 0;
54
55          // Wait 100 ns for global reset to finish
56          #100;
```

```
60        A = 1;
61        B = 1;
62        C = 0;
63        D = 0;
64        S = 1;
65
66        #100;
67
68        A = 1;
69        B = 1;
70        C = 0;
71        D = 0;
72        S = 2;
73
74        #100;
75
76        A = 1;
77        B = 1;
78        C = 0;
79        D = 0;
80        S = 3;
81
82        #100;
83    end
84
85  endmodule
```

시뮬레이션 결과는 아래와 같다.



위의 시뮬레이션 결과를 표로 정리하면 아래와 같은데 Behavioral Description 방법으로 구현한 4-to-1 멀티플렉서가 올바르게 동작함을 확인할 수 있다. (이때, A = B = 1, C = D = 0이다.)

S	Y
00	1 (=A)
01	1 (=B)
10	0 (=C)
11	0 (=D)

3. Discuss about each type of descriptions (Pros & Cons, etc) Study part of the verilog grammar and discuss about why it was designed that way. (ex, for loop, while loop, blocking, non-blocking assignments, module based feature, etc)

Verilog를 이용한 프로그래밍 방법은 크게 Structural Description, Data Flow Description과 Behavioral Description으로 분류할 수 있다.

Structural Description은 Schematic을 대체하는 방법으로 하드웨어 컴포넌트와 그들 사이의 연결을 간결하고 명확하게 표현한다는 장점이 있다. 이는 계층적인 디자인과 모듈화를 가능하게 한다. 한편, 규모가 큰 하드웨어 설계 시 시간이 오래 걸리며 Data Flow Description과 Behavioral

Description에 비해 유연성이 떨어진다는 단점이 있다.

Data Flow Description은 진리표를 대체하는 방법으로서 하드웨어 구조가 복잡할 때 Structural description에 비해 디자인 속도가 빠르다는 장점이 있다. 그러나 기저 구조 혹은 하드웨어 컴포넌트 간 연결에 대한 구체적인 정보를 제공하지 않기 때문에 최적화하기 어렵다는 단점이 있다.

Behavioral Description은 'How'에 초점을 맞춘 방법으로 설계 변경 시 기저 구조에 영향을 끼치지 않고 higher level에서 대처할 수 있어서 유연성이 높다는 장점이 있다. 한편, Data Flow Description과 마찬가지로 기저 구조 혹은 하드웨어 컴포넌트 간 연결에 대한 구체적인 정보를 제공하지 않기 때문에 최적화하기 어렵다는 단점이 있다.

Verilog는 for loop, while loop를 지원하는 등 C언어 문법과 유사하지만 C언어와는 다르게 non-blocking assignment 기능을 지원한다. Verilog에서 non-blocking assignment를 사용하지 않을 경우 모든 statement들이 순차적으로 진행되는데 실제 디지털 하드웨어에서는 많은 값들이 동시에 업데이트 되기 때문에 non-blocking assignment을 사용하지 않으면 statement의 실행 순서에 따라 시뮬레이션 결과가 실제 결과와 다르다는 이슈가 발생할 위험이 있다. 따라서 하드웨어의 동작을 보다 정확하게 모델링하기 위해 Verilog가 non-blocking assignment를 지원하는 것이라고 예상할 수 있다.