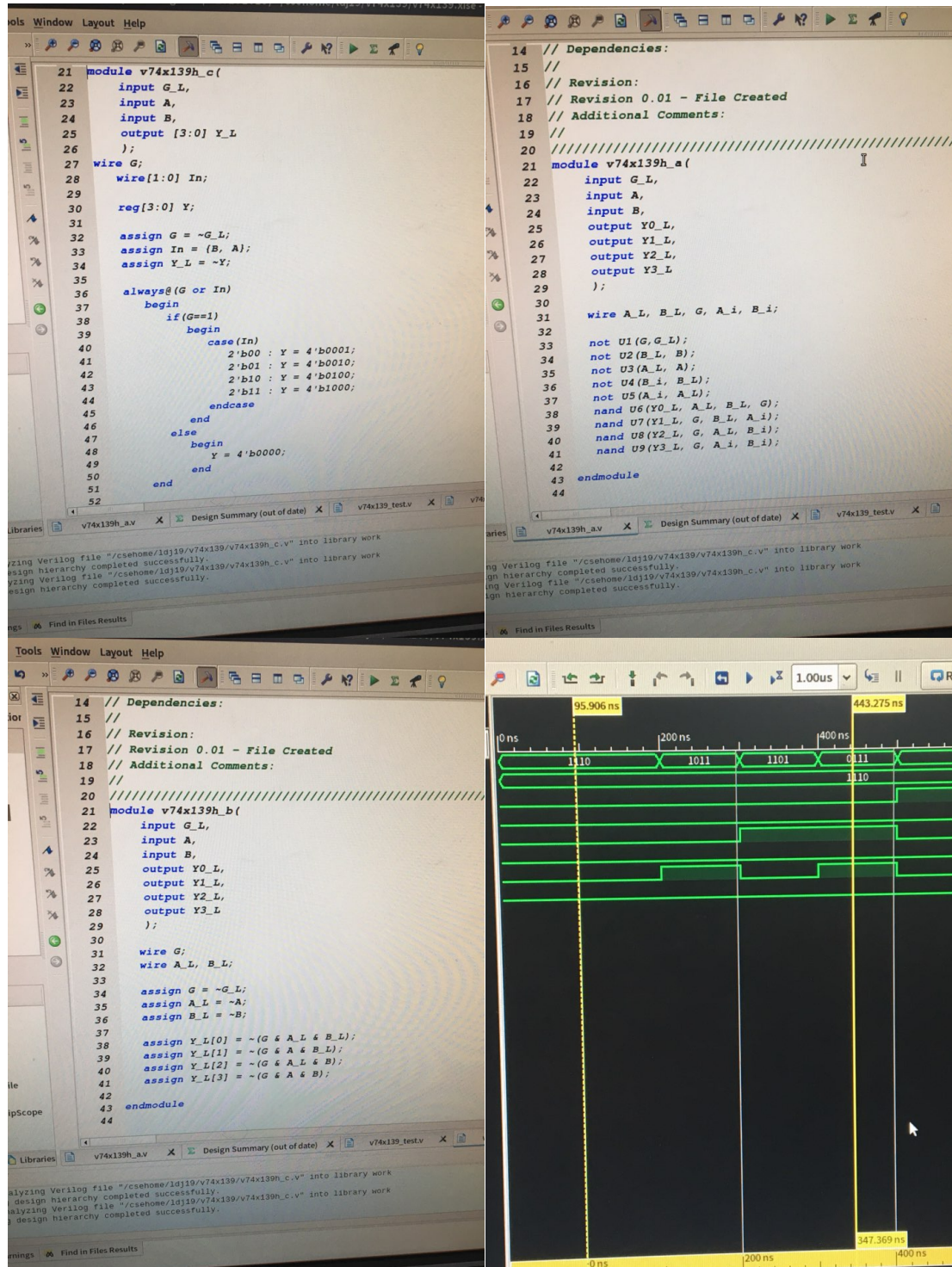
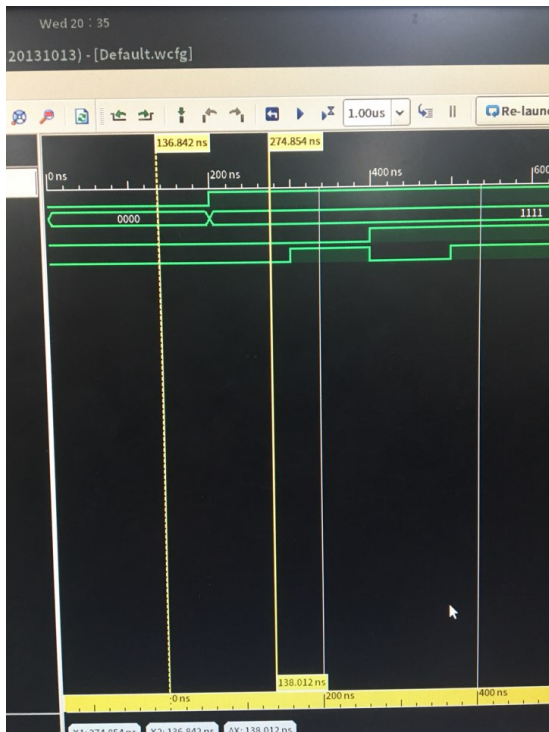


0. Lab - v74x139

Structural, Data Flow, Behavioral 의 세 가지 방법으로 만든 74x139 두 개를 이어 붙여 마지막 사진과 같은 시뮬레이션 결과를 얻었습니다.



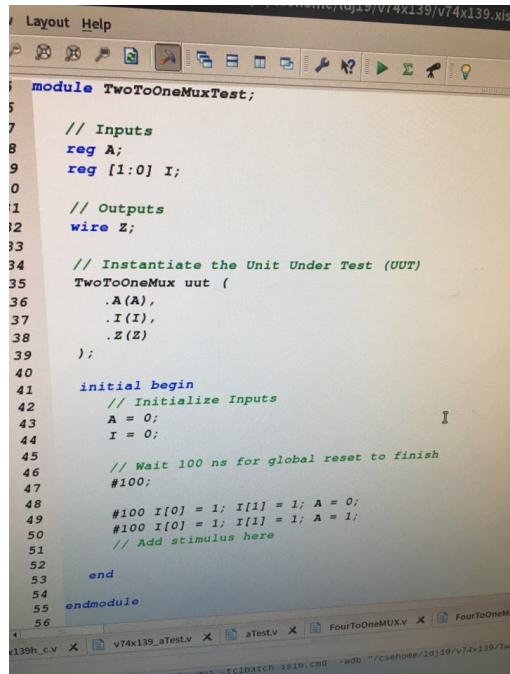
1. 4-to-1 MUX



4-to-1 MUX 를 structural, data flow, behavioral 의 세 가지 방법으로 구현하였고 위와 같은 시뮬레이션 결과를 통해 확인하였습니다. Circuit diagram 과 truth table, 그리고 동작 방식을 모두 수업 자료를 통해 확인할 수 있었던 상황에서 세 가지 방법의 구현이 크게 다르지 않았습니다. Circuit diagram 을 structural description 으로, truth table 를 data flow description 으로, 둘을 적절히 조합한 것을 behavioral description 으로 나타낼 수 있기 때문입니다. 그러나 만일 circuit function 이 복잡하고 간소화하기 힘든 상황이라면 structural description 이, 높은 레벨 gate 로 간단히 나타낼 수 있으나 truth table 의 경우의 수가 너무 많은 상황이라면 data flow description 이 불편할 수 있을 것 같습니다.

2. 16-to-1 MUX

우선 2-to-1 MUX 를 구현하였습니다.



```
module TwoToOneMuxTest;

// Inputs
reg A;
reg [1:0] I;

// Outputs
wire Z;

// Instantiate the Unit Under Test (UUT)
TwoToOneMux uut (
    .A(A),
    .I(I),
    .Z(Z)
);

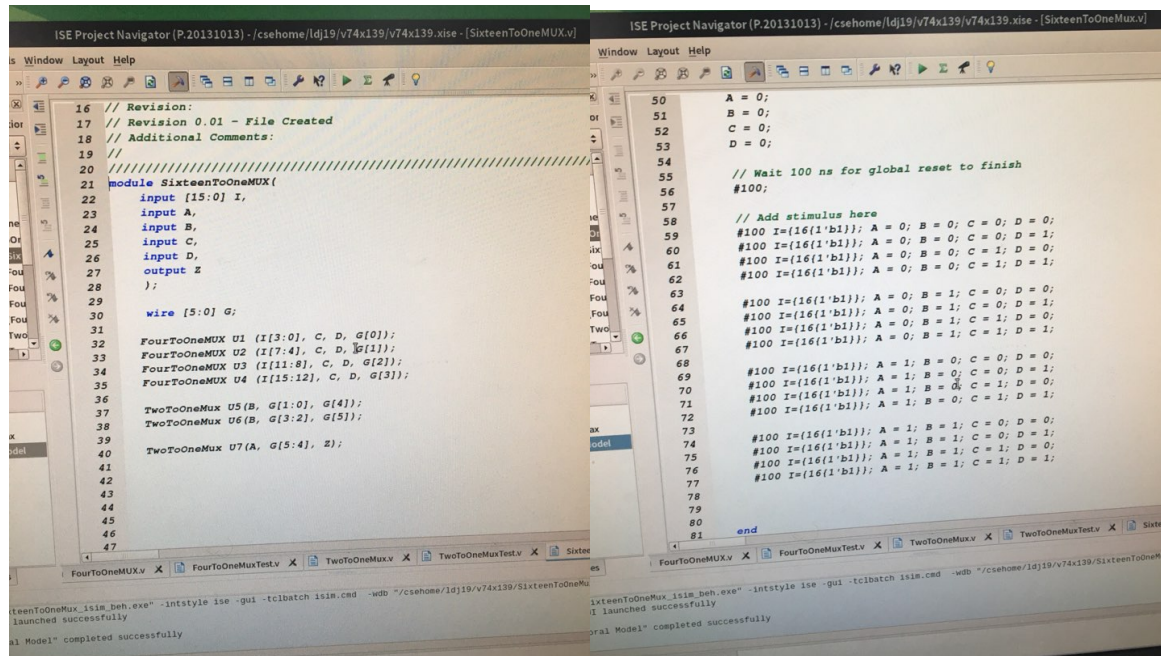
initial begin
    // Initialize Inputs
    A = 0;
    I = 0;

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

    #100 I[0] = 1; I[1] = 1; A = 0;
    #100 I[0] = 1; I[1] = 1; A = 1;
    // Add stimulus here

end
endmodule
```

이것과 1 번 문제에서 구현한 4-to-1 MUX 를 조합하여 16-to-1 MUX 를 구현하였습니다.



```
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
// =====
module SixteenToOneMUX(
    input [15:0] I,
    input A,
    input B,
    input C,
    input D,
    output Z
);
    wire [5:0] G;

    FourToOneMUX U1 (I[3:0], C, D, G[0]);
    FourToOneMUX U2 (I[7:4], C, D, G[1]);
    FourToOneMUX U3 (I[11:8], C, D, G[2]);
    FourToOneMUX U4 (I[15:12], C, D, G[3]);

    TwoToOneMux U5 (B, G[1:0], G[4]);
    TwoToOneMux U6 (B, G[3:2], G[5]);

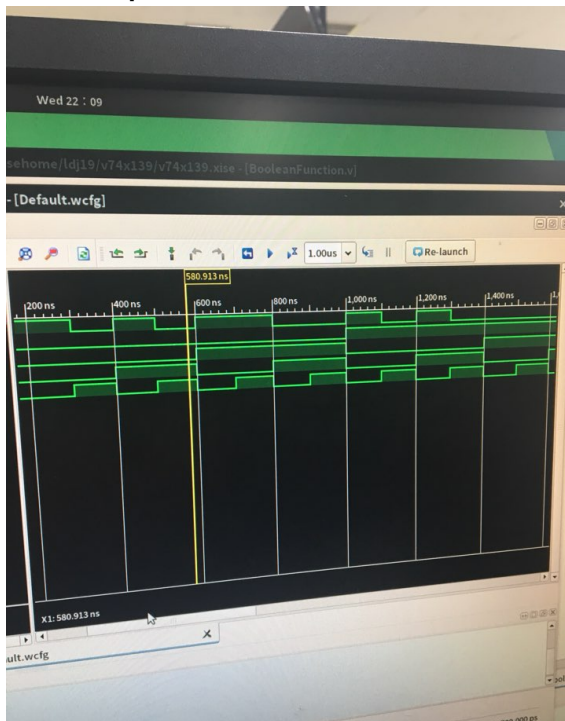
    TwoToOneMux U7 (A, G[5:4], Z);
endmodule

// Stimulus
A = 0;
B = 0;
C = 0;
D = 0;

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

// Add stimulus here
#100 I={16(1'b1)}; A = 0; B = 0; C = 0; D = 1;
#100 I={16(1'b1)}; A = 0; B = 0; C = 1; D = 0;
#100 I={16(1'b1)}; A = 0; B = 0; C = 1; D = 1;
#100 I={16(1'b1)}; A = 0; B = 1; C = 0; D = 0;
#100 I={16(1'b1)}; A = 0; B = 1; C = 0; D = 1;
#100 I={16(1'b1)}; A = 0; B = 1; C = 1; D = 0;
#100 I={16(1'b1)}; A = 0; B = 1; C = 1; D = 1;
#100 I={16(1'b1)}; A = 1; B = 0; C = 0; D = 0;
#100 I={16(1'b1)}; A = 1; B = 0; C = 0; D = 1;
#100 I={16(1'b1)}; A = 1; B = 0; C = 1; D = 0;
#100 I={16(1'b1)}; A = 1; B = 0; C = 1; D = 1;
#100 I={16(1'b1)}; A = 1; B = 1; C = 0; D = 0;
#100 I={16(1'b1)}; A = 1; B = 1; C = 0; D = 1;
#100 I={16(1'b1)}; A = 1; B = 1; C = 1; D = 0;
#100 I={16(1'b1)}; A = 1; B = 1; C = 1; D = 1;
end
```


3. Four-input Boolean function



4. 2x2 bit multiplier

```
Module Name: TwoBitMultiplier
Project Name:
Target Devices:
Tool versions:
Description:

Dependencies:

Revision:
Revision 0.01 - File Created
Additional Comments:

////////////////////////////////////
module TwoBitMultiplier(
    input A1,
    input A2,
    input B1,
    input B2,
    output P1,
    output P2,
    output P4,
    output P8
);

    assign P1 = A1 & B1;
    assign P2 = (~A2 & A1 & B2) | (A1 & B2 & ~B1) | (A2 & ~B2 & B1) | (A2 & ~A1 & B2);
    assign P4 = (A2 & B2 & ~B1) | (A2 & ~A1 & B2);
    assign P8 = (A2 & A1 & B2 & B1);

endmodule

v X TwoToOneMuxTest.v X SixteenToOneMux.v X SixteenToOneMux.v X BooleanFunction.v
e" -intstyle ise -gui -tclbatch isim.cmd -wdb ~/csehome/ldj19/v74x139/MultiplierTest_isim_beh.wdb
essfully
```