CS 223 - Digital Design

Laboratory Assignment 3

Modeling Decorders and MUXs in SystemVerilog

Arda İçöz

21901443

Section 02

24/10/2020

Part B: Behavioural SystemVerilog module for 2-to-4 decoder and a testbench

Behavioural module:

```
module twoToFourDecoder_behavioral(
    input logic a0, a1,
    output logic d[3:0]);

    assign d[0] = ~a0 & ~a1;
    assign d[1] = a0 & ~a1;
    assign d[2] = a1 & ~a0;
    assign d[3] = a1 & a0;
endmodule
```

Testbench:

```
module twoToFour_behavioralTestbench();
    logic a0, a1, d[3:0];

    //instantiate device under test
    twoToFourDecoder_behavioral dut(a0, a1, d);

    //apply inputs one at a time
    //checking results
    initial begin
        a1 = 0; a0 = 0; #10;
        assert (d[3] === 0 && d[2] === 0 && d[1] === 0 && d[0] === 1) else
$error("00 failed");
        a1 = 0; a0 = 1; #10;
        assert (d[3] === 0 && d[2] === 0 && d[1] === 1 && d[0] === 0) else
$error("01 failed");
        a1 = 1; a0 = 0; #10;
        assert (d[3] === 0 && d[2] === 1 && d[1] === 0 && d[0] === 0) else
$error("10 failed");
        a1 = 1; a0 = 1; #10;
        assert (d[3] === 1 && d[2] === 0 && d[1] === 0 && d[0] === 0) else
$error("11 failed");
    end
endmodule
```

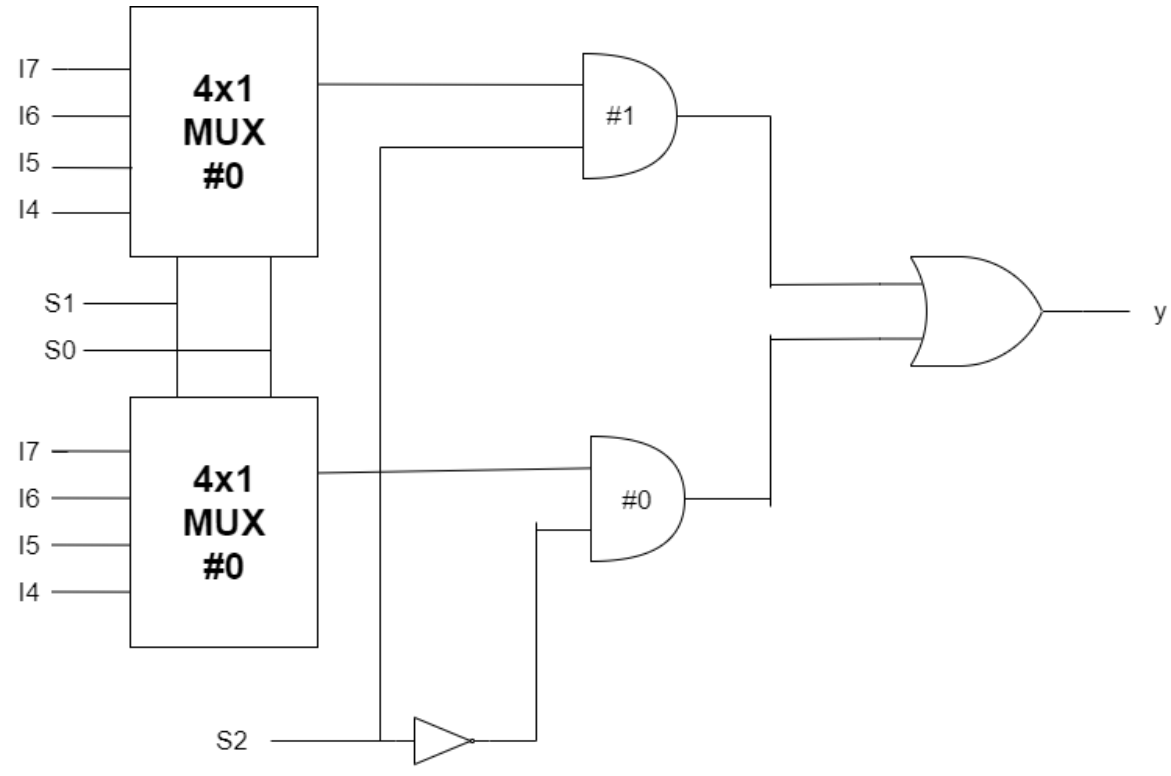
Part C: Behavioural SystemVerilog module for 4-to-1 multiplexer

Behavioural module:

```
module fourToOneMultiplexer_behavioral(
    input logic i0, i1, i2, i3, s0, s1,
    output logic y);

    logic and0, and1, and2, and3;

    //assigning AND gates, starting from the very below AND gate
    assign and0 = ~s1 & ~s0 & i0;
    assign and1 = ~s1 & s0 & i1;
    assign and2 = s1 & ~s0 & i2;
    assign and3 = s1 & s0 & i3;

    //assigning the last OR gate
    assign y = and0 | and1 | and2 | and3;
endmodule
```

Part D: Block diagram and structural SystemVerilog module of 8-to-1 MUX

Block diagram:



Structural module:

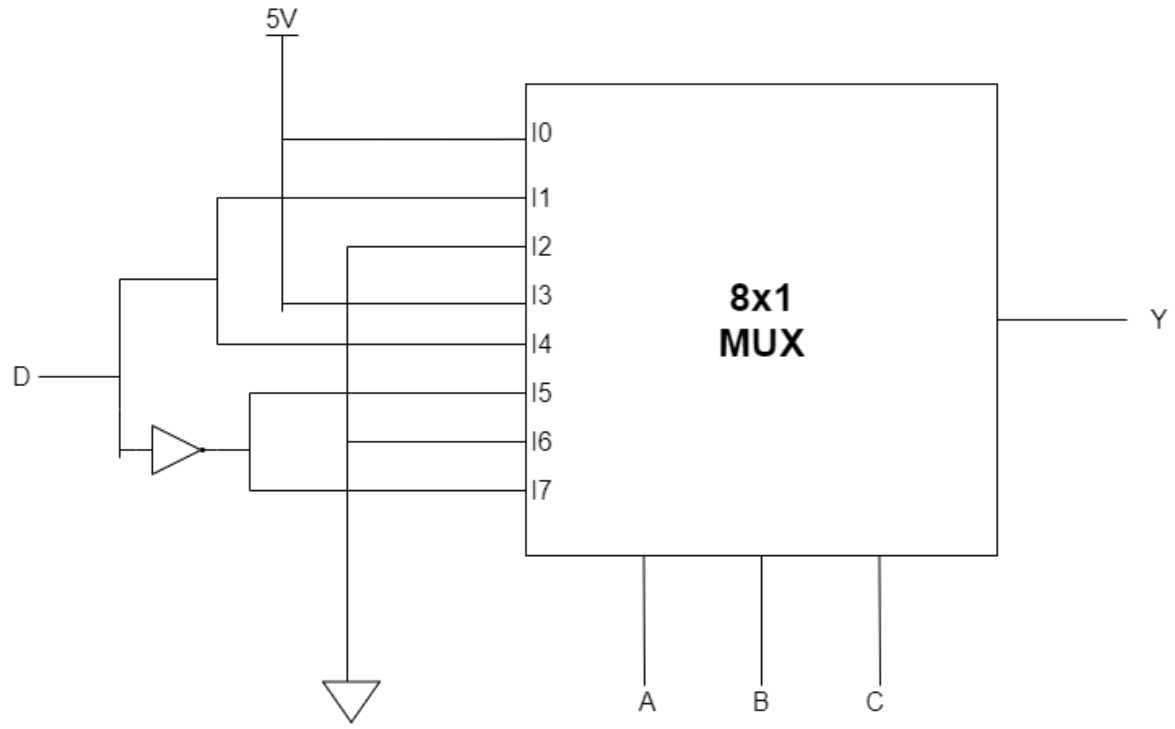
```
module eightToOneMUX_structural(
    input logic i0, i1, i2, i3, i4, i5, i6, i7,
                s0, s1, s2,
    output logic y);

    //outputs of two 4-1 multiplexer
    logic fourToOneOutput0;
    logic fourToOneOutput1;

    //outputs of two AND gates
    logic and0;
    logic and1;

    //creating instances of 4-1 multiplexer
    fourToOneMultiplexer_behavioral fourToOne0(i0, i1, i2, i3, s0, s1,
fourToOneOutput0);
    fourToOneMultiplexer_behavioral fourToOne1(i4, i5, i6, i7, s0, s1,
fourToOneOutput1);

    //processing in the AND gates
    and and0_and(and0, fourToOneOutput0, ~s2);
    and and1_and(and1, fourToOneOutput1, s2);

    //and the very last OR gate
    or y_or(y, and0, and1);
endmodule
```

Testbench:

```
`timescale 1ns / 1ps

module eightToOneMUX_structuralTestbench();
    logic i0, i1, i2, i3, i4, i5, i6, i7, s0, s1, s2, y;

    //instantiate device under test
    eightToOneMUX_structural dut(i0, i1, i2, i3, i4, i5, i6, i7, s0, s1,
s2, y);

    //apply inputs using a for loop (total of 11 inputs)
    //checking results
    initial begin
        for (int i = 1; i < 2048; i = i + 1) begin //11 input will have
2^11 (2048) different situations
            {i0, i1, i2, i3, i4, i5, i6, i7, s0, s1, s2} = i;
            #10;
        end
    end
endmodule
```

Part E: Block diagram of $F(A, B, C, D) = \Sigma(0,1,3,4,7,8,10,11,15)$ *function and SystemVerilog module*

Block diagram:



Structural module:

```
module functionABCD(
    input logic a, b, c, d,
    output logic y);

    //inputs of the D
    logic i0, i1, i2, i3, i4, i5, i6, i7, inv_d;

    //output of the inverter, inverting the input d
    not (inv_d, d);

    //creating the instance of 8-1 multiplexer
    eightToOneMUX_structural(1, d, 0, 1, d, inv_d, 0, inv_d, a, b, c, y);
endmodule
```