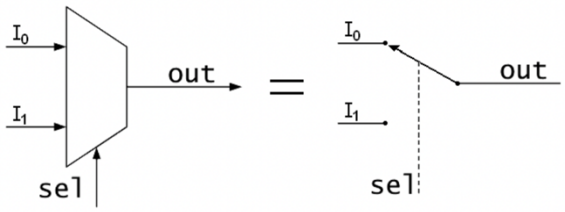
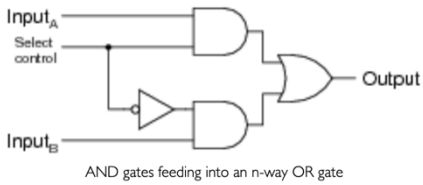
CSULB
CECS225
Lab Mux

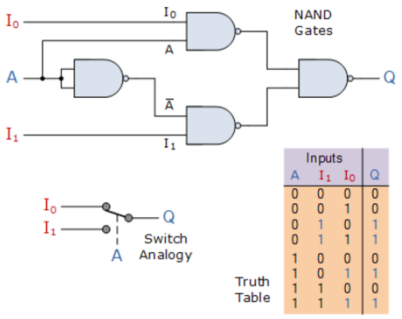
In general, the MUX is looks like the image, as a high-level abstraction:
The image shows the analog design, it selects between multiple input signals.



This is the digital design of the multiplexer



AND gates feeding into an n-way OR gate



Verilog Implementation:
We can use the assign operator with the expression that is a direct Verilog equation, we can use a set of `Use set of predefined primitives (simple gate-level function blocks).`
`Or we can use the followings:`

you need to put it inside a module. (I recommend that you understand and try each implementation using Vivado)

```
wire in0, in1, sel, out;
wire [1:0] out=(sel)?in1:in0;
```

```
wire in0, in1, sel, out;
d = {in0, in1};
out = d[sel];
```

The main issue is when you go to larger numbers of inputs, like 4, 8, 16, 32.

Another good option is the *case* statement, which is extremely useful in module format:

```
module (in0, in1, in2, in3, in4, in5, in6, in7, sel, out);

input [63:0] in0, in1, in2, in3, in4, in5, in6, in7; [2:0] sel;
input [2:0] sel;
output[63:0] out;

always @(in0 or in1 or in2 or in3 or in4 or in5 or in6 or in7 or sel)

 case (sel)

   3'b000  : out = in0;
   3'b001  : out = in1;
   3'b010  : out = in2;
   3'b011  : out = in3;
   3'b100  : out = in4;
   3'b101  : out = in5;
   3'b110  : out = in6;
   3'b111  : out = in7;

endcase

endmodule
```

For this lab
Design a 2-to-1 mux.

```
module mux2to1();
input a, b, sel;
output y;

//1- use
// assign [signal_name] = [expression];

//2- Use set of predefined primitives (simple gate-level function blocks)
//to define your mux2to1.

//3- Try all previous implementations

//4- use a simple testbench to test the 2-to-1 mux and to make sure it is
working fine -generate the time line and output the truth table using the
$display statement

When you try a new implementation comment the previous one to have one
implementation only in the design.
```

5- Design a 4-to-1 mux by cascading 2-to-1 muxes. Use testbench at the end of this document
6- Then design an 8-to-1 mux by cascading the 4-to-1 mux (for the last one you have to modify the testbench as well)

This is the testbench for 4 to 1 multiplexer. Make sure to fill in the lines that say **"your module name here"** or **"your input x here",** etc. with the correct port name in your 4:1 mux module.

```
module mux4to1tester;
    reg d3, d2, d1, d0, s1, s0;
    wire y;

    "your module name here" dut(."your input 0 name"(d0),
                                ."your input 1 name"(d1),
                                ."your input 2 name"(d2),
                                ."your input 3 name"(d3),
                                ."your select bit 0 name"(s0),
                                ."your select bit 1 name"(s1),
                                ."your output name"(y));
    integer i;

    initial begin
        $display("Mux 4 to 1 tester!");
        d0 = 1'b1; d1= 1'b0;
        d2 = 1'bx; d3= 1'b1;
        $display("\tTest case\td3\td2\td1\td2\ts1\ts0\t|\ty");
        for(i = 0; i<4; i = i + 1)
        begin
            {s1,s0} = i;
            #1 $display("%d\t\t%b\t%b\t%b\t%b\t|\t%b",i,d3,d2,d1,d0,s1,s0,y);
        end
        $finish;
    end
endmodule
```

Please submit a copy of your code and a link to a video explaining your code