CS 207 Digital Logic - Spring 2020
Lab 5

James Yu

Mar. 18, 2020

Objective

1. Implement basic combinational logic circuits.

Lab Exercise Submission

1. You should name all source code as instructed. Mis-named files will not be recognized.
2. You should submit all source code files with an extension "`.v`".
3. You should submit all source code directly into the sakai system below. **Do not compress them into one folder.**

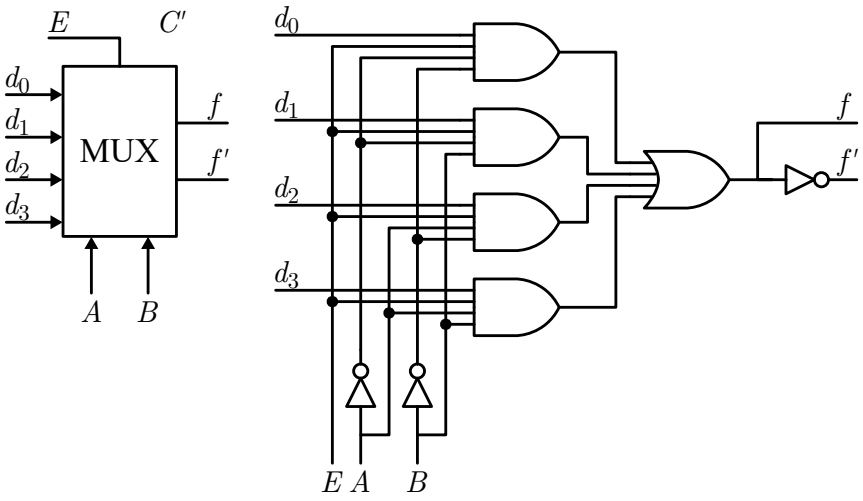
   `https://sakai.sustech.edu.cn/portal/site/ebf68254-68c5-4cfe-9d42-b26758f854ee`
4. Lab exercises should be submitted before the deadline, typically one week after the lab session. No late submission policy applies to lab exercises.

1 4-1 Multiplexer

In Lab 3 we tried to implement a 4-1 MUX using UDP. During the lecture we know how MUX can be implemented using primitive gates as shown in the next page. In this lab, we first implement the 4-1 MUX using both gate-level design. The Verilog code is given as follows:

`mux4to1.v`

```
module mux4to1(Y, D, A);
output Y;
input [1:0] A;
input [3:0] D;
```




```
wire na0, na1;
wire y0, y1, y2, y3;

not n_0(na0, A[0]);
not n_1(na1, A[1]);
and and_0(y0, D[0], na1, na0);
and and_1(y1, D[1], na1, A[0]);
and and_2(y2, D[2], A[1], na0);
and and_3(y3, D[3], A[1], A[0]);
or or_0(Y, y0, y1, y2, y3);
endmodule
```

To test the code, we have the following testbench:

mux4to1.v

```
module mux4to1_tb;
reg [3:0] IN;
reg [1:0] S;
wire OUT;

mux4to1 mux(OUT, IN, S);
initial begin
  IN[0]=1; IN[1]=1; IN[2]=1; IN[3]=0;
  $display("IN0 = %b, IN1 = %b, IN2 = %b, IN3 = %b", IN[0], IN[1], IN[2], IN[3]);
  $monitor("%3t: S1 = %b, S0 = %b, OUT = %b", $time, S[1], S[0], OUT);
  #5 S[0]=0; S[1]=0;
  #5 S[0]=1; S[1]=0;
  #5 S[0]=0; S[1]=1;
  #5 S[0]=1; S[1]=1;
```

```
  #10 $finish;
end
endmodule
```

The above example shows how we can use gate-level modeling to implement a 4-1 MUX. It is also possible to do the same with behavioral modeling:

mux4to1.v

```
module mux4to1(Y, D, A);
output Y;
input [1:0] A;
input [3:0] D;
reg Y;

always @(A)
  case(A)
    0: Y <= D[0];
    1: Y <= D[1];
    2: Y <= D[2];
    3: Y <= D[3];
  endcase
endmodule
```

We can use the same testbench as shown above, or we can use behavioral modeling thinking and implement the following:

mux4to1.v

```
module mux4to1_tb;
reg [3:0] IN;
reg [1:0] S;
wire OUT;

mux4to1 mymux(OUT, IN, S);
initial begin
  IN <= 4'b1010;
  S <= 2'b00;
  #5 $display("IN[3] = %b, IN[2] = %b, IN[1] = %b, IN[0] = %b", IN[3], IN[2], IN[1],
      IN[0]);
     $monitor("%3t: S[1] = %b, S[0] = %b, OUT = %b", $time, S[1], S[0], OUT);

  #5 S <= 2'b01;
  #5 S <= 2'b10;
  #5 S <= 2'b11;
  #10 $finish;
```

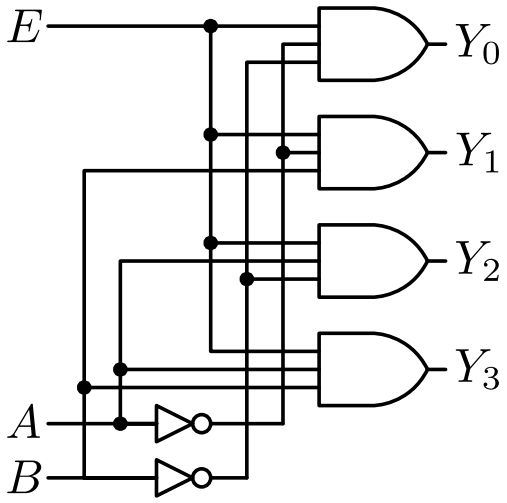
```
end
endmodule
```

Try them out and see if the results are same.

2 Exercise

2.1 Exercise 1

Implement a 1-4 demultiplexer shown as follows:



Implement the DEMUX in both gate-level design and behavioral modeling. The gate-level design module should follow the design below:

demuxgate.v

```
module demuxgate(Y, E, A, B);
// Module Code
endmodule
```

and save in file `demuxgate.v`. The behavioral modeling module should follow the design below:

demuxbehav.v

```
module demuxbehav(Y, E, A, B);
// Module Code
endmodule
```

and save in file `demuxbehav.v`.

Assignment
Save the source code in `demuxgate.v` and `demuxbehav.v`. Upload these files to Sakai under **Assignments → Lab Exercise 5**.

2.2 Exercise 2

Use the previous 4-1 multiplexer to implement a 8-1 multiplexer. The module should follow the design below:

mux8to1.v

```
module mux8to1(Y, A, D);
// Module Code
endmodule
```

and save in file `mux8to1.v`. Do remember to include the 4-1 MUX code in your source file. The input `A` is the three selection lines, and the input `D` is the eight data lines.

Assignment
Save the source code in `mux8to1.v`. Upload this file to Sakai under **Assignments → Lab Exercise 5**.

Assignment
When you finished Lab Exercise 5, there should be three `.v` files in the Sakai system: `demuxgate.v`, `demuxbehav.v`, and `mux8to1.v`.