Cover Page

Name: Ethan Lingad

Student ID: 862268541

Course Title: CS120A

Lab Section: 22

Lab Title: Lab 02. Decoder and Multiplexers

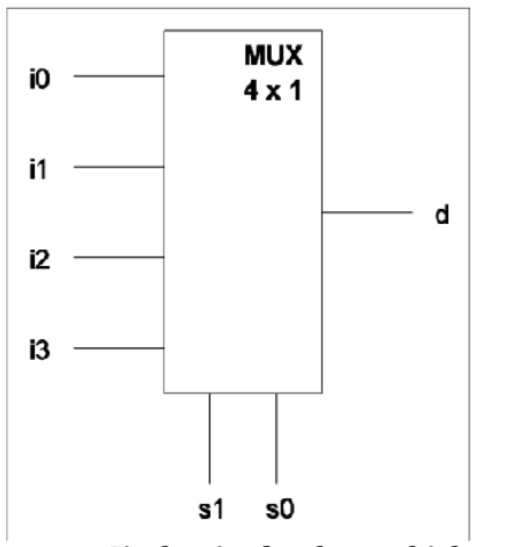
Overview:

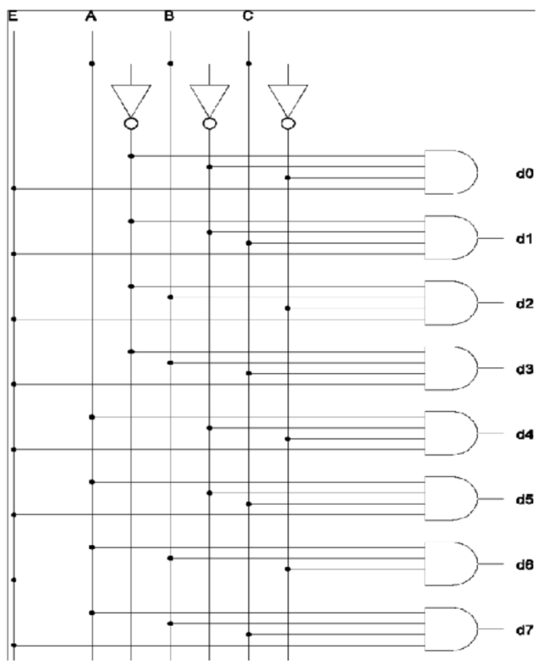
What was done in the lab was that I was prompted to create a 3 x 8 multiplexer and a 4 x 1 decoder. I then was prompted to fill in the code that was missing for creating a structural and behavioral modeling for both the multiplexer and decoder. The results from the waveform showed that the inputs showed the correct result when given different combinations of inputs for both the multiplexer and decoder. The conclusion is that I was able to create a behavioral and structural model in addition to creating test cases for each of them.

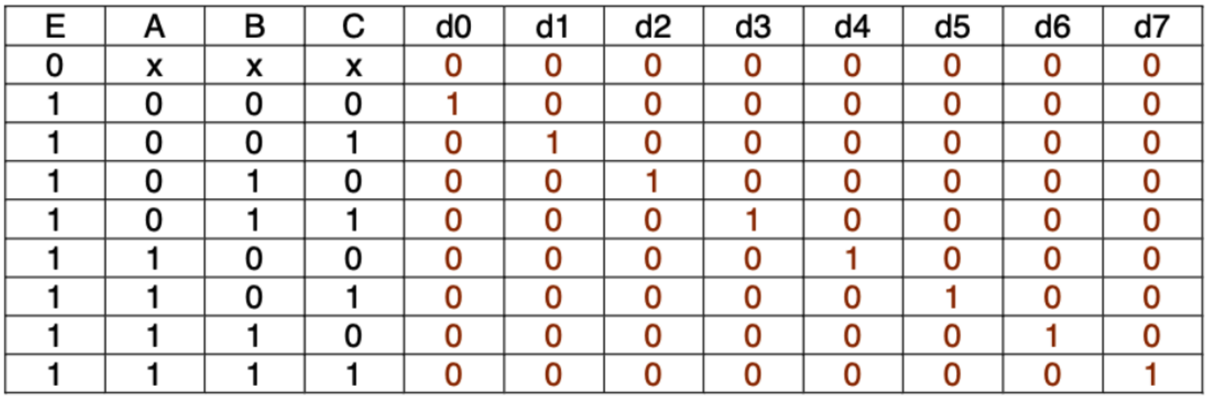
Analysis:

1. For part 1, Copy and paste the code given for the design.sv
2. fill in the rest of the and4 module to set all the outputs in the structural model
3. Copy and paste the code give for the testbench.sv
4. Change the decoder_st to decoder_bh
5. Look at the results for the behavioral simulation and verify that the results are correct
6. For part 2, Copy and paste the code for the design.sv
7. fill in the rest of the parts for the structural model
8. fill in the rest of the test cases for the behavioral model
9. copy and paste the code for the testbench.sv
10. change the mux_bh to mux_st to test the structural model
11. Examine the waveform

Resources that I used to construct Part 1 and Part 2:







E	A	B	C	d0	d1	d2	d3	d4	d5	d6	d7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

S1	S0	i0 / 00	i1 / 01	i2 / 10	i3 / 11	output
0	0	0	x	x	x	0
0	0	1	x	x	x	1
0	1	x	0	x	x	0
0	1	x	1		x	1

1	0	x	x	0	x	0
1	0	x	x	1	x	1
1	1	x	x	x	0	0
1	1	x	x	x	1	1

Records:

Part 1:

design.sv

```
// Code your design here
module and4(
input wire enable ,
input wire a,
input wire b,
input wire c,
output wire r
);
//
assign r = enable & a & b & c ;
endmodule
// structural model
module decoder_st(
// I/0 ports
input wire enable ,
input wire a ,
input wire b ,
input wire c ,
output wire d0,
output wire d1,
output wire d2,
output wire d3,
output wire d4,
output wire d5,
output wire d6,
output wire d7
);

// Using the and4 module to set all outputs
and4 c1(enable, ~a, ~b, ~c, d0) ;
// Your code goes here (7 cases left to implement)
  and4 c2(enable, ~a, ~b,c, d1) ;
  and4 c3(enable, ~a, b, ~c, d2) ;
  and4 c4(enable, ~a, b, c, d3) ;
  and4 c5(enable, a, ~b, ~c, d4) ;
  and4 c6(enable, a, ~b, c, d5) ;
  and4 c7(enable, a, b, ~c, d6) ;
  and4 c8(enable, a, b, c, d7) ;
endmodule
// behavioral model
module decoder_bh(
// I/0 ports
input wire enable ,
input wire a ,
input wire b ,
input wire c ,
output reg d0,
output reg d1,
output reg d2,
output reg d3,
output reg d4,
output reg d5,
output reg d6,
output reg d7
);
// Internal wire
wire [3:0] bundle ;
assign bundle = {enable , a, b, c } ;
// Behavioral description
always @(*) begin
d0 = 1'b0 ;
d1 = 1'b0 ;
d2 = 1'b0 ;
d3 = 1'b0 ;
```

```
d4 = 1'b0 ;
d5 = 1'b0 ;
d6 = 1'b0 ;
d7 = 1'b0 ;
// Setting the correct output
case (bundle)
4'b1000: d0 = 1'b1 ;
4'b1001: d1 = 1'b1 ;
4'b1010: d2 = 1'b1 ;
4'b1011: d3 = 1'b1 ;
4'b1100: d4 = 1'b1 ;
4'b1101: d5 = 1'b1 ;
4'b1110: d6 = 1'b1 ;
4'b1111: d7 = 1'b1 ;
default : begin
d0 = 1'b0 ;
end
endcase
end
endmodule
```

testbench.sv

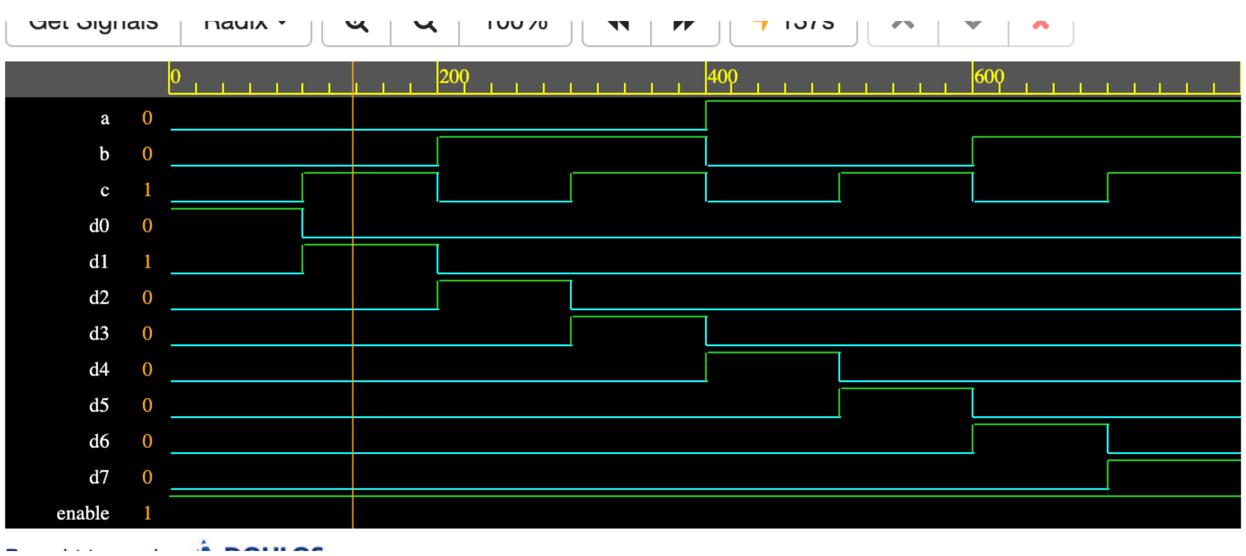
```
// Code your testbench here
// or browse Examples
module decoder_tb;
// Inputs

reg enable;
reg a;
reg b;
reg c;
// Outputs
wire d0;
wire d1;
wire d2;
wire d3;
wire d4;
wire d5;
wire d6;
wire d7;
// Instantiate the Unit Under Test (UUT)
// decoder_st
  decoder_bh uut ( // change to "decoder_bh" for testing your
behavioral model
.enable(enable),
.a(a),
.b(b),
.c(c),
.d0(d0),
.d1(d1),
.d2(d2),
.d3(d3),
.d4(d4),
.d5(d5),
.d6(d6),
.d7(d7)
);
initial begin
$dumpfile("dump.vcd"); $dumpvars;
enable = 1;
a = 0;
b = 0;
c = 0;
#100; // Wait for 100 ns
$display("TC11 ");
if ( d0 != 1'b1 ) $display ("Result is wrong");
a = 0;
b = 0;
c = 1;
#100;
$display("TC12 ");
if ( d1 != 1'b1 ) $display ("Result is wrong");
a = 0;
b = 1;
c = 0;
#100;
$display("TC13 ");
```

```
if ( d2 != 1'b1 ) $display ("Result is wrong");
a = 0;
b = 1;
c = 1;
#100;
$display("TC14 ");
if ( d3 != 1'b1 ) $display ("Result is wrong");
a = 1;
b = 0;
c = 0;
#100;
$display("TC15 ");
if ( d4 != 1'b1 ) $display ("Result is wrong");
a = 1;
b = 0;
c = 1;
#100;
$display("TC16 ");
if ( d5 != 1'b1 ) $display ("Result is wrong");
a = 1;
b = 1;

c = 0;
#100;
$display("TC17 ");
if ( d6 != 1'b1 ) $display ("Result is wrong");
a = 1;
b = 1;
c = 1;
#100;
$display("TC18 ");
if ( d7 != 1'b1 ) $display ("Result is wrong");
// Your test cases *******************
end
endmodule
```

Waveform For Part 1



Part 2:

design.sv

```
// Code your design here
module and3(
input wire a,
input wire b,
input wire c,
output wire r
);
assign r = a & b & c ;
endmodule
// structural model
module mux_st(
// Ports I/O
input wire s1,
input wire s0,
input wire i0,
input wire i1,
input wire i2,
input wire i3,
output wire d
);
wire r1, r2, r3, r4 ;
and3 c1 ( ~s1,~s0, i0, r1 ) ;
```

```
  and3 c2 (~s1, s0, i1, r2) ;
  and3 c3(s1, ~s0, i2, r3) ;
  and3 c4(s1, s0, i3, r4) ;

// Your code goes here (3 cases left)
assign d = r1 | r2 | r3 | r4 ;
endmodule
// behavioral model
module mux_bh(
// Ports I/O
input wire s1,
input wire s0,
input wire i0,
input wire i1,
input wire i2,
input wire i3,
output reg d
) ;
always @(*) begin
d = 1'b0 ;
case ( {s1,s0} )
2'b00 : d = i0 ;
2'b01 : d = i1 ;
2'b10 : d = i2 ;
2'b11 : d = i3 ;
// your code goes here (3 cases left)
endcase
end
endmodule
```

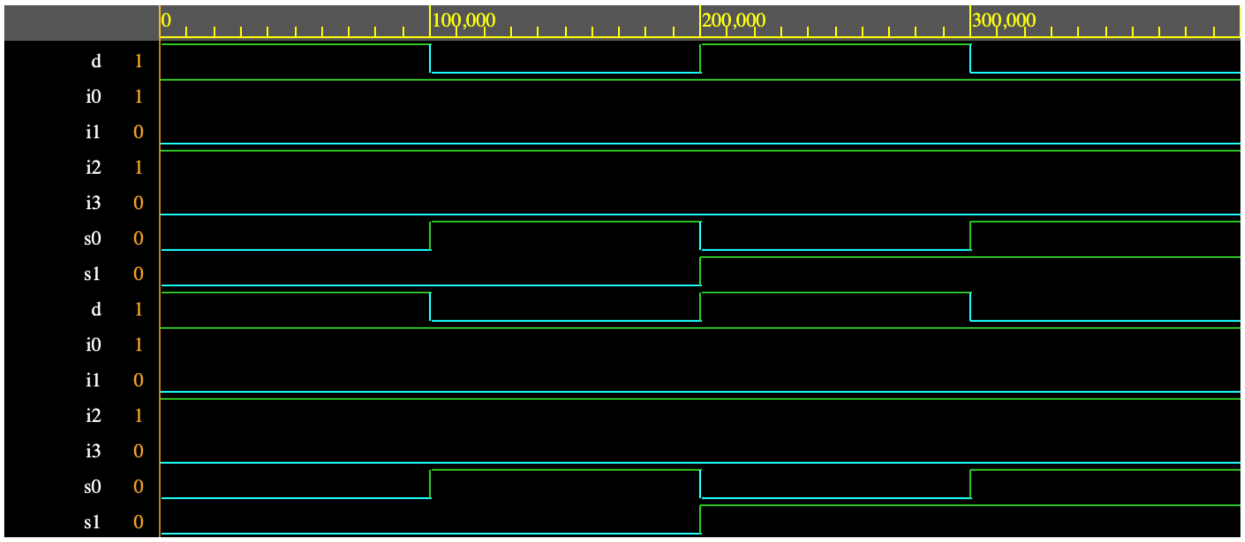
testbench.sv

```
// Code your testbench here
// or browse Examples
`timescale 1ns / 1ps
module mux_tb;
// Inputs
reg s1;
reg s0;
```

```verilog
reg i0;
reg i1;
reg i2;
reg i3;
// Outputs
wire d;
// Instantiate the Unit Under Test (UUT)
mux_bh uut ( // change to "mux_st" for testing your structuralmodel
.s1(s1),
.s0(s0),
.i0(i0),
.i1(i1),
.i2(i2),
.i3(i3),
.d(d)
);
initial begin
$dumpfile("dump.vcd"); $dumpvars;
i0 = 1;
i1 = 0;
i2 = 1;
i3 = 0;
s1 = 0;
s0 = 0;
#100;
$display("TC11 ");
if ( d != i0 ) $display ("Result is wrong");
s1 = 0;
s0 = 1;
#100;
$display("TC12 ");
if ( d != i1 ) $display ("Result is wrong");
s1 = 1;
s0 = 0;
#100;
$display("TC13 ");
if ( d != i2 ) $display ("Result is wrong");
s1 = 1;
s0 = 1;
```

```
#100;
$display("TC14 ");
if ( d != i3 ) $display ("Result is wrong");
// Your test cases
end
endmodule
```

Waveform for Structural and Behavioral Model: Both give the same results



Discussion:

The system does work according to the provided specification. The Mux and Decoder gave the correct results when given a testbench for the design. Some problems dealt within the lab was that I was unable to get the standard Verilog working for this class but was able to figure it out by using a different Verilog version. There is no other way to improve the system since it looks sufficient enough.

Conclusions:

The purpose of the lab was to understand how to implement decoders and mux. On a more comprehensive level, it was to help understand the ways of implementing fundamentally in Verilog. It taught me how to create logic gates and helped me to understand the inner workings of Verilog modules.

Questions:

1. A waveform is a way of showing the results from the testbench code in Verilog. It can help to verify if certain inputs would give a desired result. It shows certain results at a given duration based on the testbench.sv file. The input changes as the code is executed.
2. A testbench is a way to test your Verilog implementation that you have created. It is a way to help ensure that your code is producing the correct output. It creates a mock module in which you pass it into your actual code through instantiation.
3. Yes, we can definitely replace the 4 input AND gates in the circuit with a 2-input AND gate. The way that we would do that is by giving only two inputs. Then with the two inputs, we can use the '&' operator to and the two inputs instead of having to AND four inputs or three. In the examples above, we used either three or four and operators but we can instead use two to make a two input AND gate.