4 to 1 Multiplexer

COEN 122 Lab 1

Grading

- Lab 1-4 (10% each) 40%
- Final Project 60%

Late Policy

- Late submissions submitted within 24 hours after the deadline receive 50% credit. After 24 hours, no credit is given.

Demo

- Labs submitted with no demo will receive no credit. Please demo to me before the lab is due.
- You can demo in either TAs Office Hours

Lab 1 Overview

- Objective: Implement 4:1 multiplexer in Verilog using Vivado
- For the first 4 labs, you will be working individually, but will be working in teams for the project

Helpful website: http://www.asic-world.com/verilog/veritut.html

Running Vivado

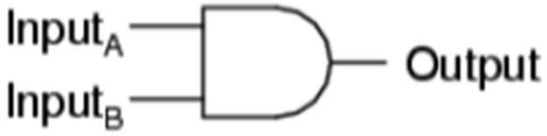
- Open ECC Link -> File Explorer -> Applications -> EE -> Xilinx Design Tools -> Vivado 2020.2 -> Vivado 2020.2 Application
- Wait
- Create New Project
- Name and Location: must not start with samba
- RTL
- For now, ignore add source (will be useful later)
- Hit next until finish

Design Flow

- Write code for a structure model of a 4:1 mux (No if statements)
- Write test bench to verify proper function of logic block
- Run Synthesis
- Run Simulation
- View waveform, verify results

AND Gate Example

2-input AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

AND Gate Example Code

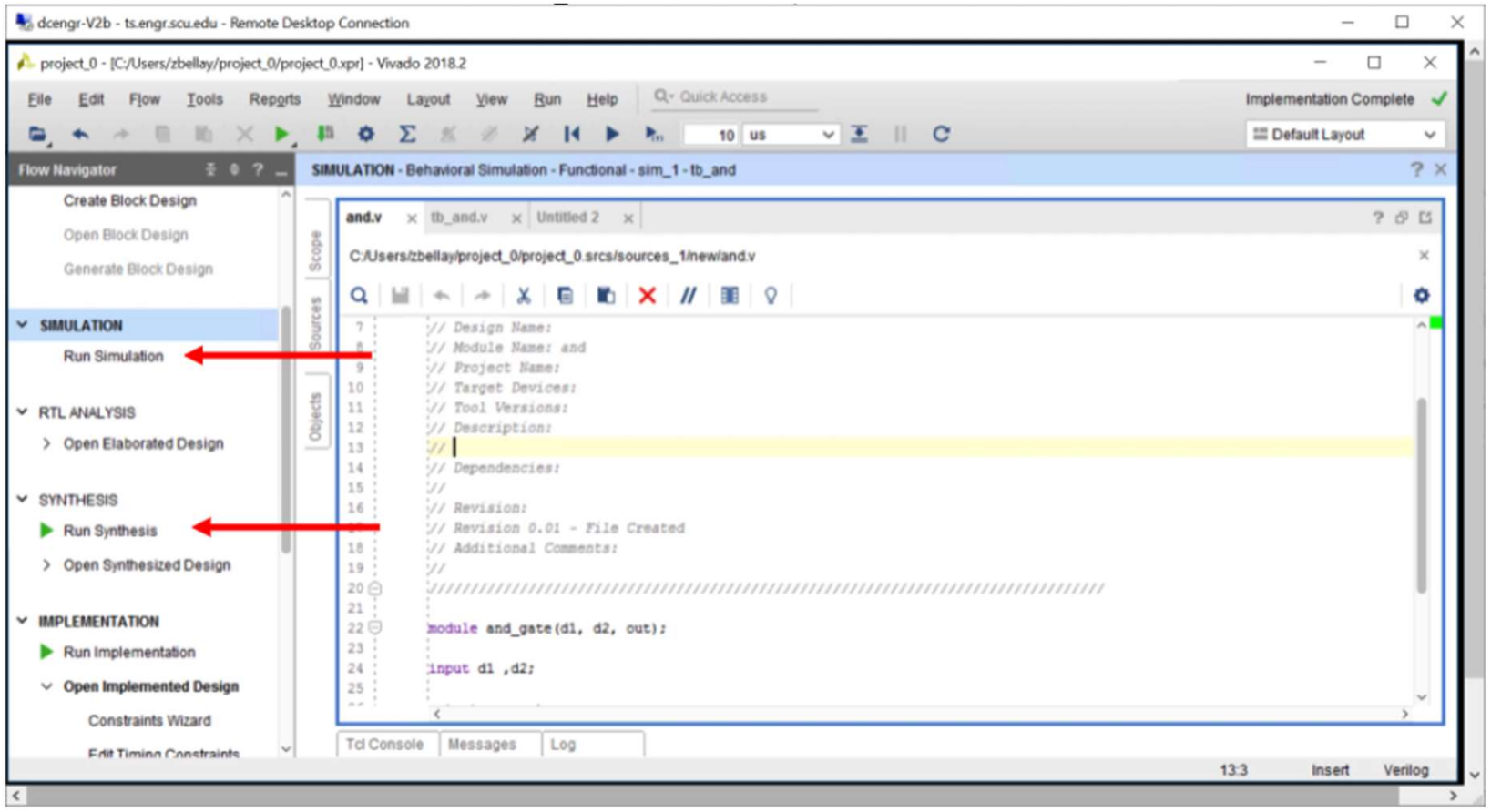
```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Engineer: Zach Bellay
// Create Date: 10/01/2019 07:33:31 PM
// Etc...
//////////////////////////////////////////////////////////////////////////////////

// define and_gate variables
module and_gate(d1, d2, out);

// define d1, d2 as inputs
input d1 ,d2;

// define out to be a reg
// meaning a value can be
// assigned to it
output reg out;

// the 'always@' means that
// every time an input changes,
// run this body of code again
// (so our and gate reruns whenever
// we change the inputs!)
always@(d1, d2)
begin
    out = d1 & d2;
end

endmodule
```

and.v

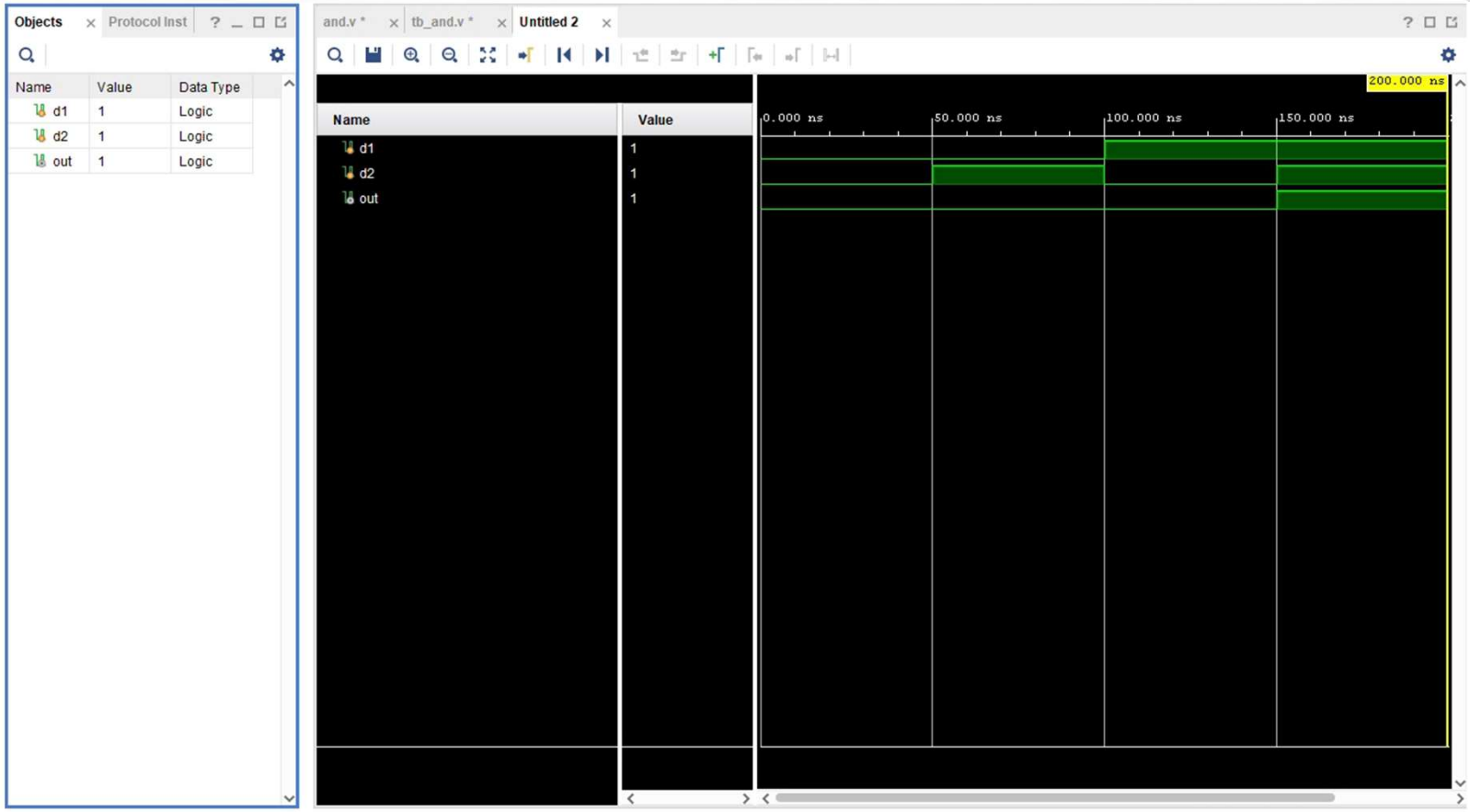
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Zach Bellay
// Create Date: 10/01/2019 07:33:31 PM
// Etc...
//////////////////////////////////////////////////////////////////////////////////
module tb_and(); // define testbench module
// regs can store data
reg d1, d2;
// can't store data, connects modules
wire out;
// and_gate object
and_gate test(d1,d2,out);
// 'initial' means just to
// do it once (unlike 'always')
initial
begin
    // Out should be 0
    d1 = 0;
    d2 = 0;
    #50; // wait 50 nanoseconds to see the change!
    // Out should be 0
    d1 = 0;
    d2 = 1;
    #50;
    // Out should be 0
    d1 = 1;
    d2 = 0;
    #50;
    // Out should be 1
    d1 = 1;
    d2 = 1;
    #50;
    $finish;
end
endmodule
```

tb_and.v

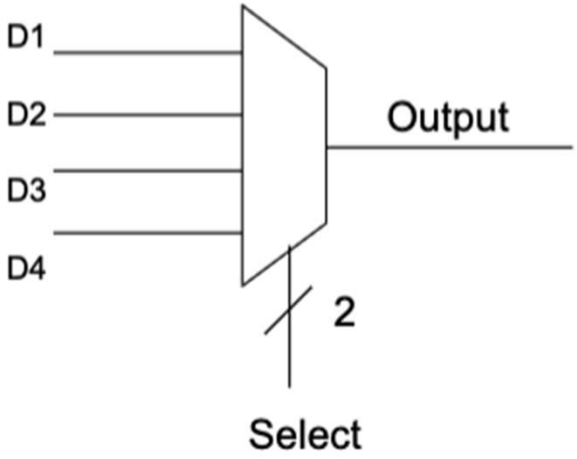
Run Synthesis, then Simulation



Waveform


Objects
Protocol Inst
Name
Value
Data Type
d1
1
Logic
d2
1
Logic
out
1
Logic
and.v *
tb_and.v *
Untitled 2
Name
Value
d1
1
d2
1
out
1
200.000 ns
0.000 ns
50.000 ns
100.000 ns
150.000 ns

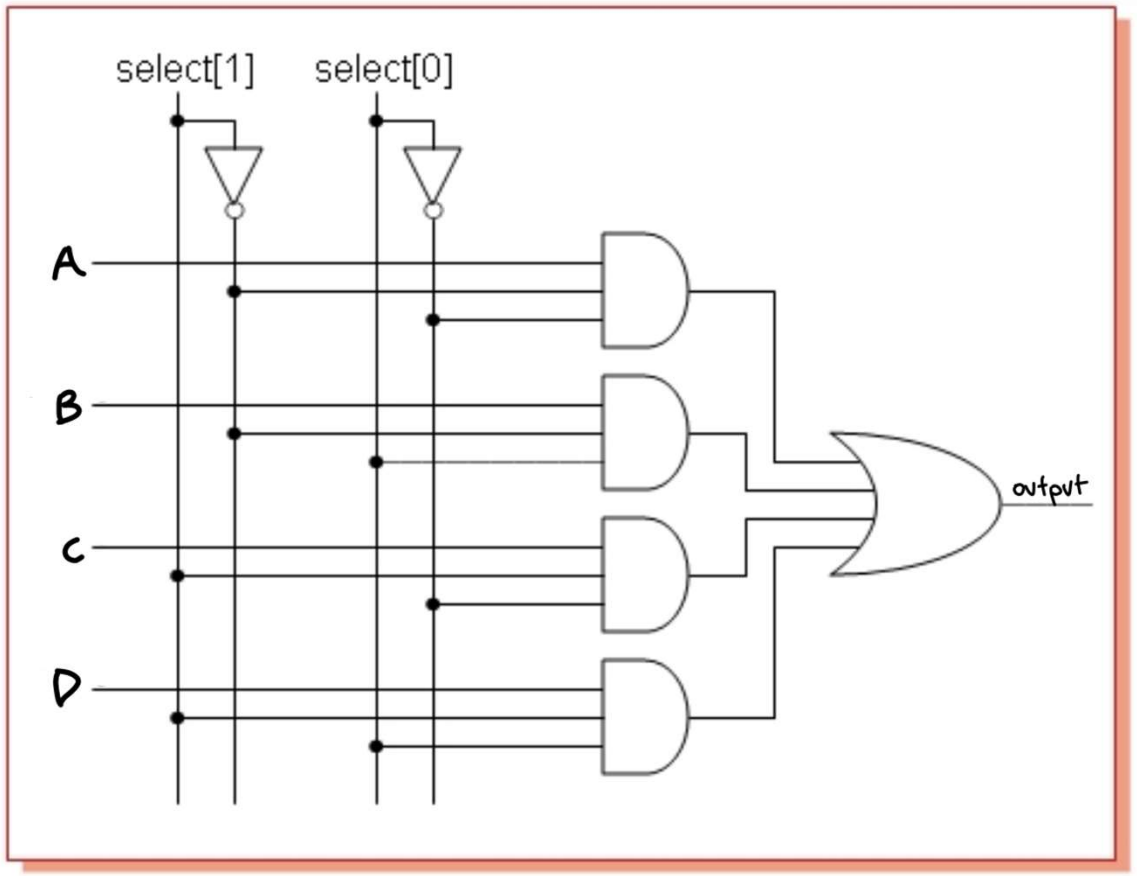
MUX Functionality



Truth Table:

Select Data Inputs		Output
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Logic Circuit for MUX



Important Constructs for Mux

Bit Array:

```
input [1:0] sel;
```

Gates:

```
input A,B,C,D;
wire output1, output2, output3;
output out;

not(output1, A); //not gate
and(output2, A, B, C); //and gate
or(output3, A, B, C); //or gate
```

Testbench:

```
sel = 2'b00;
```

Submission

- mux file as a txt file
- mux_testbench file as a txt file
- Screenshot of waveform