Lab 5 – Basic Processor Modules

Name: Breeha and Namel	ID: 08283 and 08327

Task 1

Code

Codes should contain meaningful variable naming and comments

**Add snip of code or copy-paste the code written in the Editor window. Make sure the irrelevant area of snip is cropped.*

Design Modules

```
module mux(
    input[63:0] a, //input for a
    input [63:0]b, //input for b
    input sel, //selection bit
    output[0:63] data_out //output
    );
    assign data_out = sel? a:b; //if selection bit is 1, then assign the value of a to data_out, else assign b.
endmodule
```

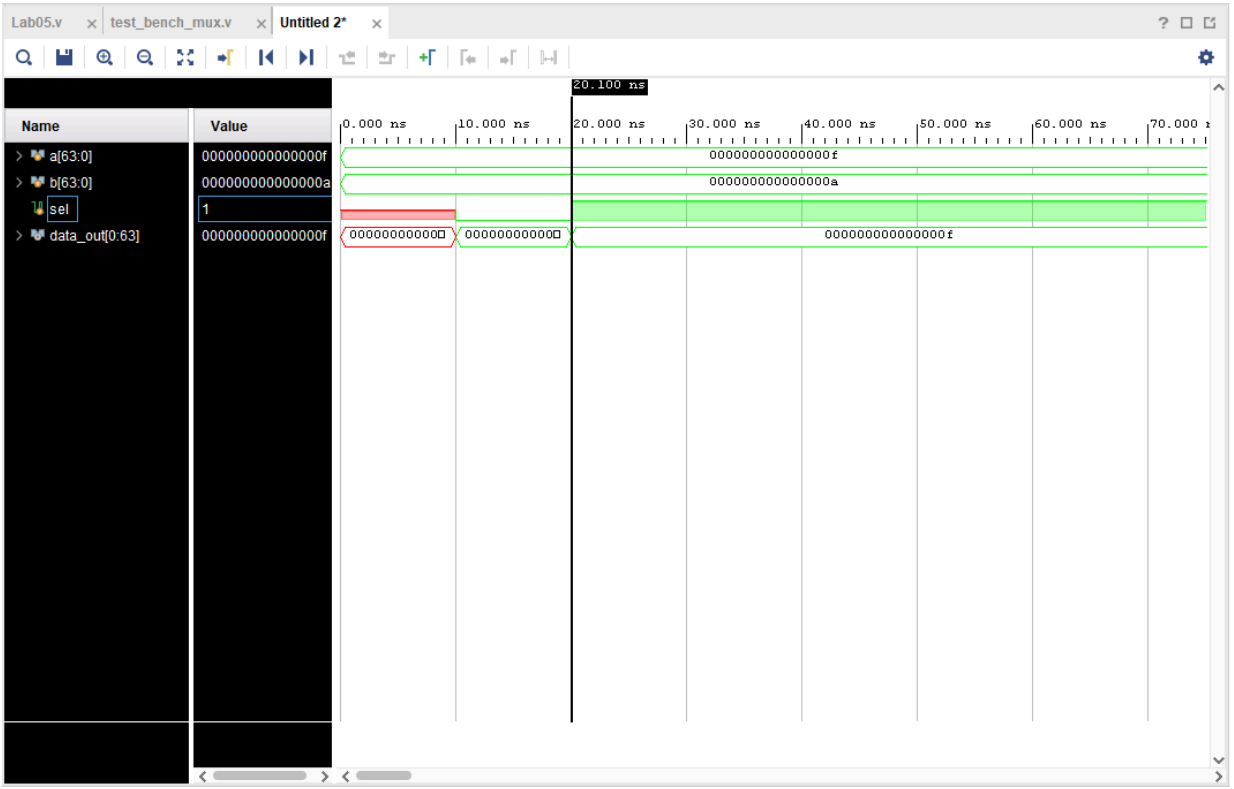
Testbench

```
module test_bench_mux(
    );
    reg[63:0] a; //input for a
    reg [63:0] b; //input for b
    reg sel; //selection bit
    wire[0:63] data_out; //output

      mux m1(a, b , sel, data_out); //calling mux function

    initial
    begin
    a= 4'hf; //the value we took for a is 15
    b= 4'ha; //the value we took for b is 10

    #10
    sel=1'b0; //assigning selection bit 0

    #10
    sel=1'b1; //assigning selection bit 1

    end

endmodule
```

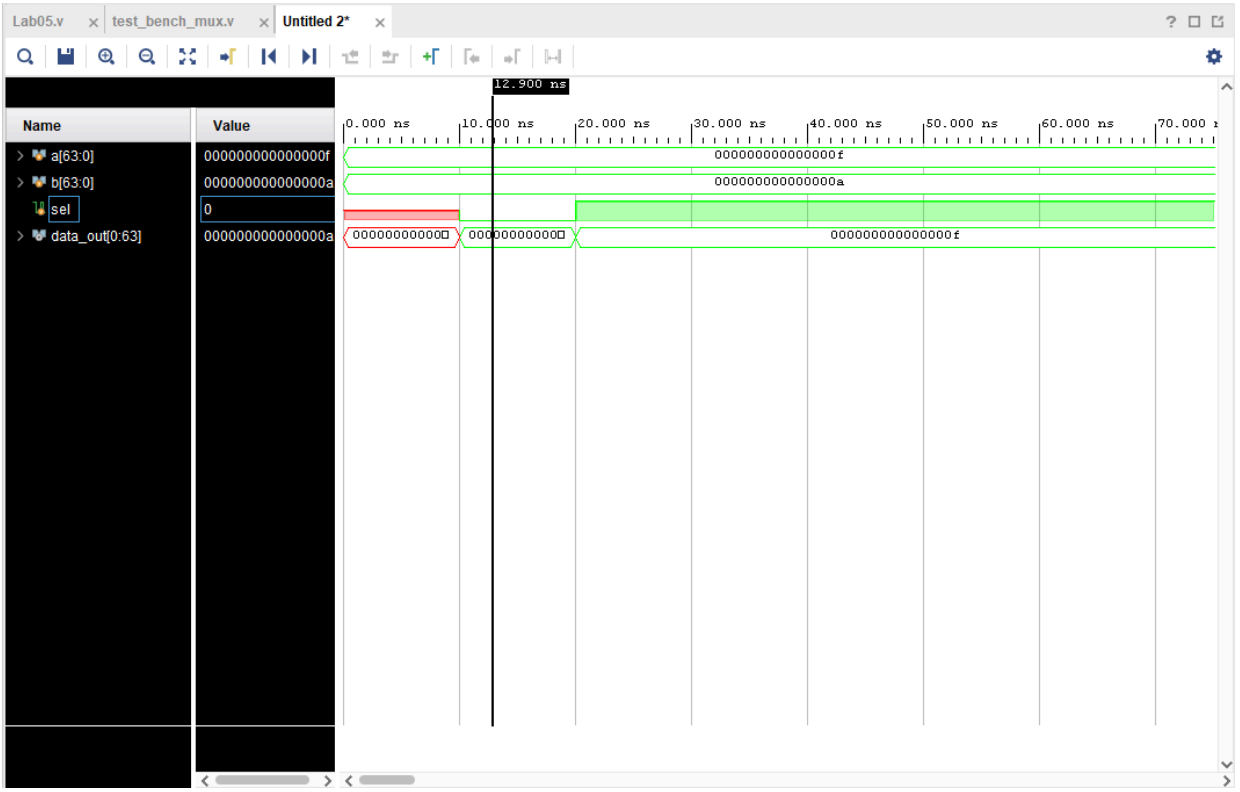
Results (Waveforms)

**Add snip of relevant signals' waveforms. Results in Log Window are optional. Make sure the irrelevant area of snip is cropped.*

when selection bit is 1



when selection bit is 0



Comments

**Comment on the obtained results/working of code*

We took 64 bits for input a and b, and 1 bit for selection bit. Our output is also in 64 bits.
When selection bit is 0 we assigned value of **a** which is 10 (hexadecimal: A) to data_out and when selection bit is 1 we assigned value of **b** which is 15 (hexadecimal: F) to data_out.

Task 2

Code

Codes should contain meaningful variable naming and comments

**Add snip of code or copy-paste the code written in the Editor window. Make sure the irrelevant area of snip is cropped.*

Design Modules

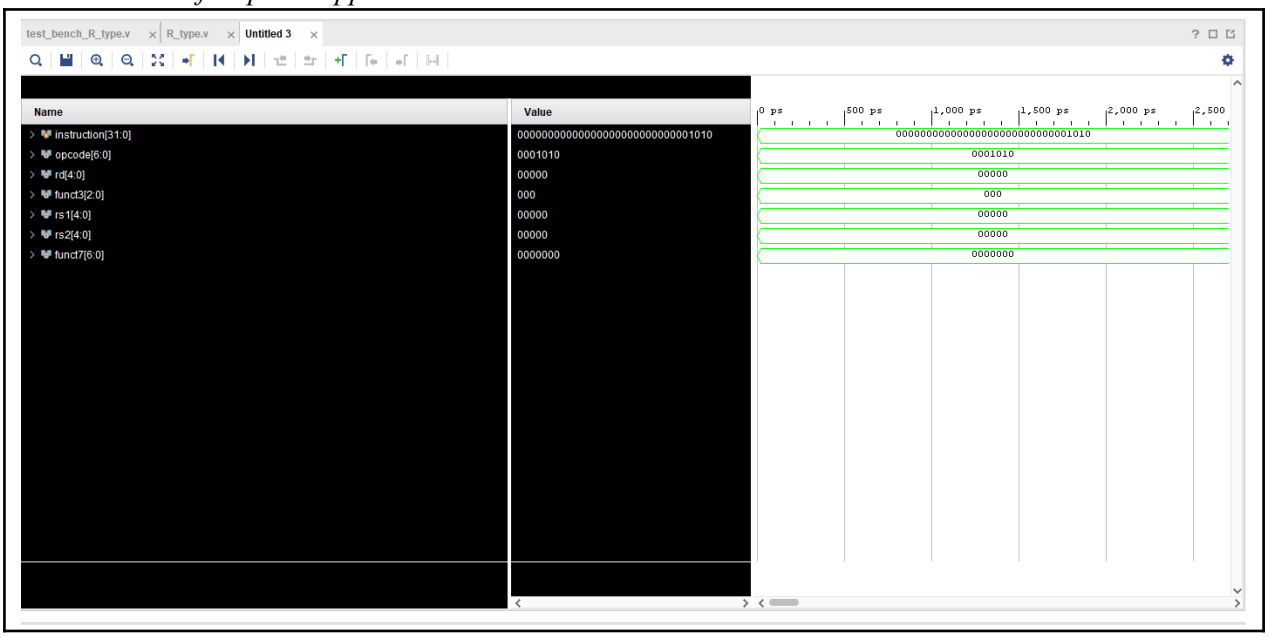
```
module R_type(
    input[31:0] instruction, // we took 32 bits for instruction input
    output[6:0] opcode, // assigned 7 bits to opcode
    output[4:0] rd, // assigned 5 bits to rd
    output [2:0] funct3, //assigned 3 bits to funct3
    output[4:0] rs1, //assigned 5 bits to rs1
    output[4:0] rs2, //assigned 5 bits to rs2
    output[6:0] funct7 //assigned 7 bits to funct7
    );
    //here we will slice the instruction bits according to the R-type instruction
    assign opcode= instruction[6:0];
    assign rd= instruction[11:7];
    assign funct3 =instruction[14:12];
    assign rs1= instruction [19:15];
    assign rs2= instruction [24:20];
    assign funct7= instruction [31:25];

endmodule
```

Testbench

```
module test_bench_R_type(

    );
      reg [31:0] instruction; // we took 32 bits for instruction input
    wire [6:0] opcode; // assigned 7 bits to opcode
    wire[4:0] rd; // assigned 5 bits to rd
    wire [2:0] funct3; //assigned 3 bits to funct3
    wire [4:0] rs1; //assigned 5 bits to rs1
    wire [4:0] rs2; //assigned 5 bits to rs2
    wire [6:0] funct7; //assigned 7 bits to funct7

      R_type r1(instruction, opcode, rd, funct3, rs1, rs2, funct7);

    initial
    begin

    //assigning random value for instruction
                                        instruction = 32'b00000000000000000000000000001010; // we assigned value 10


    end
endmodule
```

Results (Waveforms)

**Add snip of relevant signals' waveforms. Results in Log Window are optional. Make sure the irrelevant area of snip is cropped.*



Comments

**Comment on the obtained results/working of code*

In this instruction parser module we passed a 32 bit instruction in binary, and output the different fields of instruction format.

- we assigned the first 7 bits of instruction to opcode
- the next 5 bits of instructions to rd
- the next 3 bits of instructions to function 3
- the next 10 bits of instructions to rs1 and rs2
- finally the last 7 bits of instruction to funct7

Task 3

Code

Codes should contain meaningful variable naming and comments

**Add snip of code or copy-paste the code written in the Editor window. Make sure the irrelevant area of snip is cropped.*

Design Modules

```
module immidiate_data_generator(
   input  [31:0]  instruction,    // 32-bit instruction input
   output reg [11:0] imm_data,    // 12-bit immediate data output
   output reg selection_bit_1,    // Control signal output 1
   output reg selection_bit_2,    // Control signal output 2
   output reg [63:0] final_output // 64-bit final output, sign-extended immediate data
   );

   always @* begin
      // Extract selection bits from the instruction
      selection_bit_1 = instruction[5]; // Extract 6th bit for selection_bit_1
      selection_bit_2 = instruction[6]; // Extract 7th bit for selection_bit_2

      // Determine how to construct immediate data based on selection bits
      if(selection_bit_2) begin
         // If selection_bit_2 is set, format immediate data for a specific instruction type
         imm_data = {instruction[31],instruction[7],instruction[30:25],instruction[11:8]};
      end
      else begin
         // If selection_bit_2 is not set, further check selection_bit_1
         if(selection_bit_1) begin
            // If selection_bit_1 is set, format immediate data for another instruction type
            imm_data ={instruction[31:25],instruction[11:7]};
         end
         else begin
            // If neither selection bit is set, use a default immediate data format
            imm_data = instruction[31:20];
         end
      end
```

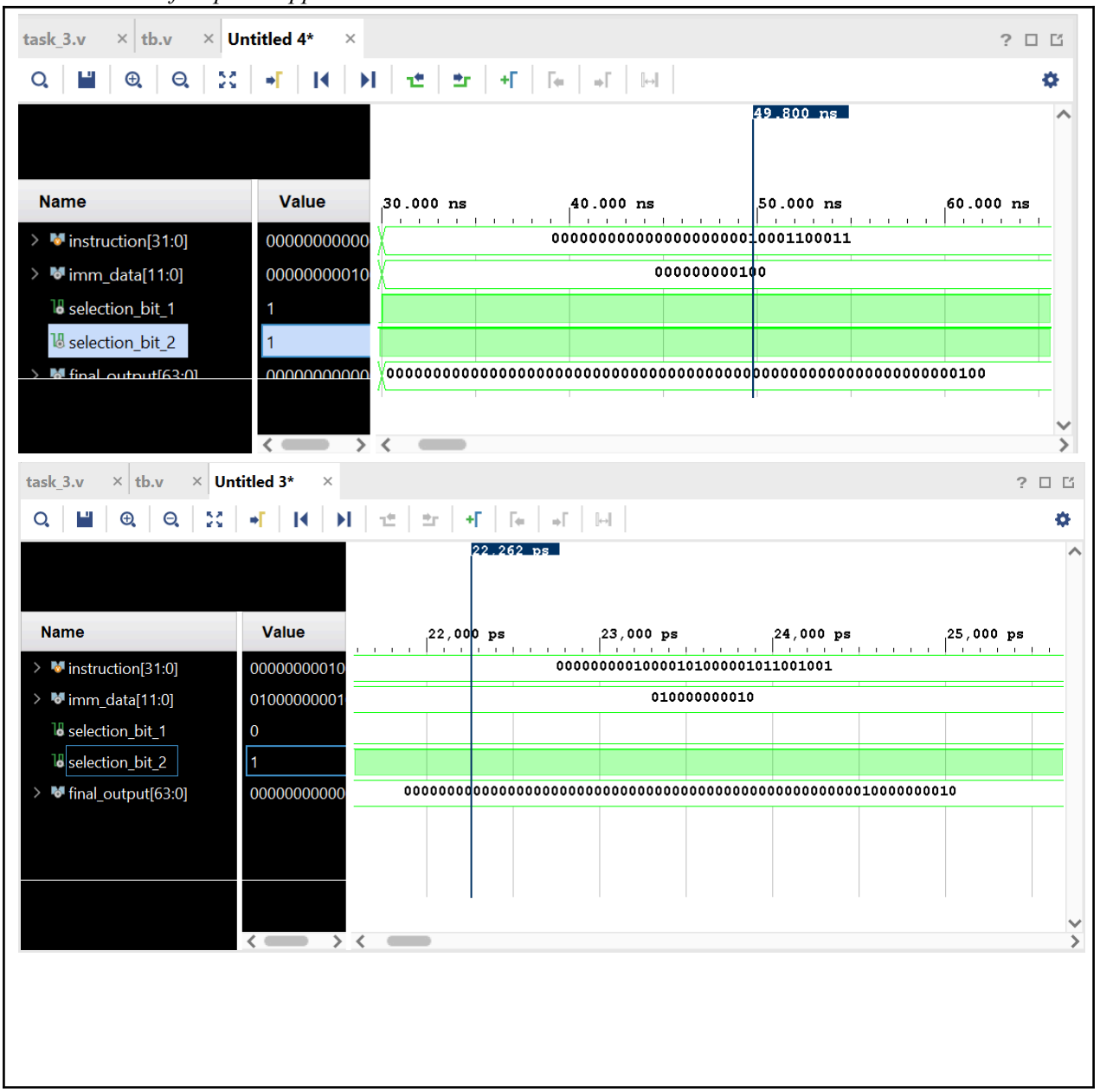
Testbench

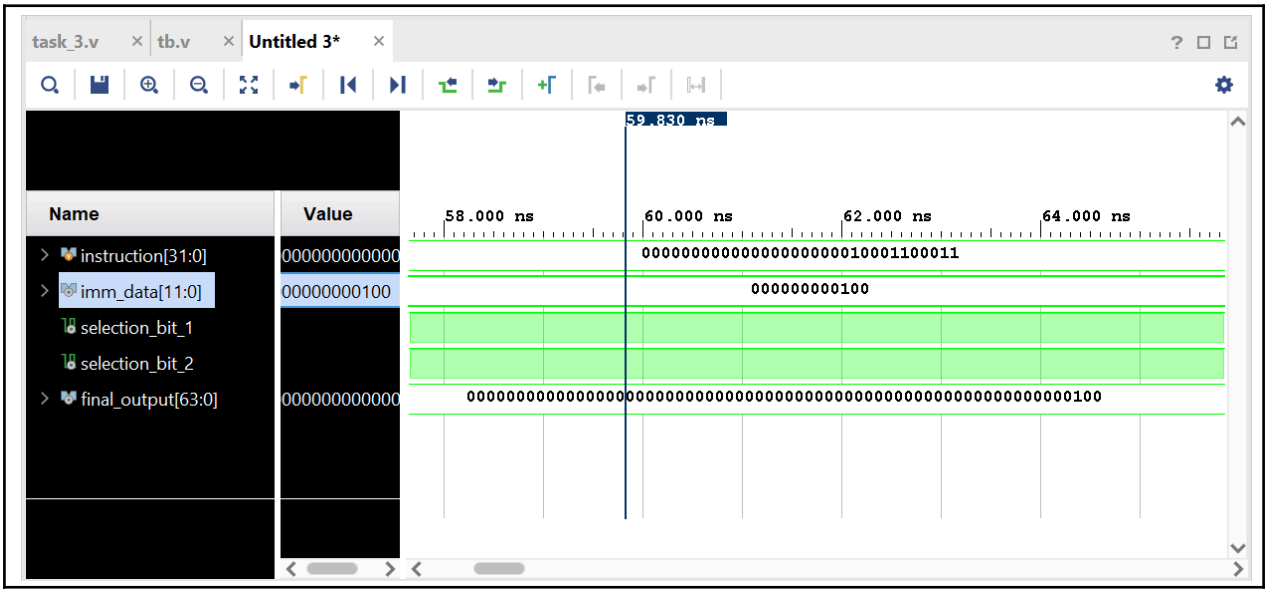
```
module tb(
);
reg [31:0] instruction;
wire [11:0] imm_data;
wire selection_bit_1;
wire selection_bit_2;
wire[63:0] final_output;
immidiate_data_generator imm1(instruction,imm_data,selection_bit_1,selection_bit_2,final_output);
initial
 begin
 #10
                        instruction                        = 32'b00000110110011110011000000010001 ;
 #10
                        instruction                        = 32'b00000000100001010000010110010001 ;
 #10
 instruction = 32'h00000463;

 end

endmodule
```

```
        // Construct the final output by
sign-extending the 12-bit immediate data to 64
bits
        final_output =
{{52{imm_data[11]}},imm_data}; // Sign-extend
using the MSB of imm_data
    end

endmodule
```

Results (Waveforms)

**Add snip of relevant signals' waveforms. Results in Log Window are optional. Make sure the irrelevant area of snip is cropped.*





Comments

**Comment on the obtained results/working of code*

The code snippet demonstrates how to derive three distinct forms of immediate data based on the values of bits 6 and 5 of the input instruction. Specifically, when bits 6 and 5 are set to (00), the immediate data is formatted according to the I-type instruction format. If these bits are (01), the data adheres to the S-type format. Lastly, when the bits are either (11) or (10), the immediate data corresponds to the SB-type format.

Lab 05 - Basic Processor Modules

Assessment Rubric

Name:	**Student ID:**

Points Distribution

Task No.	**LR2 Code**	**LR 5 Results**	**AR 7 Report Submission/Code Comments**
Task 1	/10	/10	/20
Task 2	/20	/10	
Task 3	/20	/10	
Total Points	/100 Points		
CLO Mapped	CLO 1		

For description of different levels of the mapped rubrics, please refer the provided Lab Evaluation Assessment Rubrics and Affective Domain Assessment Rubrics.

#	Assessment Elements	Level 1: Unsatisfactory	Level 2: Developing	Level 3: Good	Level 4: Exemplary
LR2	Program/Code/ Simulation Model/ Network Model	Program/code/simulation model/network model does not implement the required functionality and has several errors. The student is not able to utilize even the basic tools of the software.	Program/code/simulation model/network model has some errors and does not produce completely accurate results. Student has limited command on the basic tools of the software.	Program/code/simulation model/network model gives correct output but not efficiently implemented or implemented by computationally complex routine.	Program/code/simulation /network model is efficiently implemented and gives correct output. Student has full command on the basic tools of the software.
LR5	Results & Plots	Figures/ graphs / tables are not developed or are poorly constructed with erroneous results. Titles, captions, units are not mentioned. Data is presented in an obscure manner.	Figures, graphs and tables are drawn but contain errors. Titles, captions, units are not accurate. Data presentation is not too clear.	All figures, graphs, tables are correctly drawn but contain minor errors or some of the details are missing.	Figures / graphs / tables are correctly drawn and appropriate titles/captions and proper units are mentioned. Data presentation is systematic.

AR9	Report Content/Code Comments	No summary provided. The number/amount of tasks completed below the level of satisfaction and/or submitted late	Couldn't provide good summary of in-lab tasks. Some major tasks were completed but not explained well. Submission on time. Some major plots and figures provided	Good summary of In-lab tasks. All major tasks completed except few minor ones. The work is supported by some decent explanations, Submission on time, All necessary plots, and figures provided	Outstanding Summary of In-Lab tasks. All task completed and explained well, submitted on time, good presentation of plots and figure with proper label, titles and captions