Pre Laboratory 3:

FLOATING-POINT ADDER/SUBTRACTOR

OBJECTIVES

- Getting to know how to describe a floating-point adder and subtractor.
- Design and implement digital circuits using FSM.
- Download the circuit into the FPGA chip and test its functionality.

PREPARATION FOR LAB 3

- Students have to simulate all the exercises in Pre Lab 3 at home. All results (codes, waveform, RTL viewer, …) have to be captured and submitted to instructors prior to the lab session. *If not, students will not participate in the lab and be considered absent this session.*

Pre Laboratory 3:

FLOATING-POINT ADDER/SUBTRACTOR

Floating point numbers allow computers to perform operations on a wide range of numbers.

According to the IEEE standards, floating point numbers are of the form

(-1)^S * (1+F) *2^E

S is the sign bit, which determines whether the number is positive or negative.

F – fraction – holds the significant bits of the floating point number.

E is the exponent that (1+F) is raised to.

A 32 bit floating point number is standard, however fo simpification, we will be using an 8 bit representation. We will have a sign bit, 3 bits for the exponent, and the remaining 4 bits will be devoted to the fraction. This will allow us to represent a resolution as small as 1/128 and the smallest number we can represent is 1/8. Eight-bit floating point numbers are not useful for performing extremely accurate calculations, but it does demonstrate the operation of a floating point adder.

You are desired to design a floating-point adder/subtractor should perform correctly in the normal cases. Besides several pins needs to indicate some extreme cases:

- Zero detection: when the result is zero, zero detection pin will be 1.
- Overflow detection: when the result is out of range, the overflow occurs. The overflow detection will be 1 to indicate that an overflow has occurred, and the result is not reliable.

Inputs	Operand 1	A[7:0]	8-bit normalized input
	Operand 2	B[7:0]	8-bit normalized input
	Selection	S	1-bit input, addition/subtraction selection
Outputs	Operation result	Result[7:0]	8-bit normalized output
	Zero detection	Z	1-bit ouput
	Overflow detection	OV	1-bit output

Table 1: IO definition

Pre Laboratory 3:

FLOATING-POINT ADDER/SUBTRACTOR

EXERCISE 1:

Objective: Identify the final exponent value.

Requirement: Write a System Verilog module which receive exponent values of two operands and identify the final exponent value. This module has following IO definitions:

Inputs	Exp_A[2:0]	3-bit input, exponent value of operand A
	Exp_B[2:0]	3-bit input, exponent value of operand B
Outputs	Exp_diff[3:0]	4-bit output, exponent value difference (exp_A – exp_B) Exp_diff[3] = 0: exp_A > exp_B Exp_diff[3] = 1: exp_A < exp_B
	Final_exp[2:0]	3-bit output, exponent value of the final result

Instruction:

This module find the difference between two exponent value Exp_A, Exp_B as well as identify the exponent value of the result: final_exp equal the larger exponent value.

1. Create a new Quartus project for your circuit.
2. Write a System Verilog description for the circuit.
3. Compile the code. Use the Quartus RTL Viewer tool to examine the gate-level circuit produced from the code.
4. Simulate the behavior of your code. Test the functionality of your design by inputting various data values and observing the generated outputs.

Check: Your report has to show two results:

- RTL viewer.
- Simulation results.

Pre Laboratory 3:

FLOATING-POINT ADDER/SUBTRACTOR

EXERCISE 2:

Objective: Identifying the bigger fraction and smaller fraction.

Requirement: Create a System Verilog module which identify the bigger fraction and smaller fraction. This module has following IO definitions:

Inputs	Fract_A[3:0]	4-bit input, fraction of operand A
	Fract_B[3:0]	4-bit input, fraction of operand B
	Exp_diff[3:0]	4-bit input, exponent value difference (exp_A – exp_B)
Outputs	Bigger_fract[3:0]	4-bit output, bigger fraction
	Smaller_fract[3:0]	4-bit output, smaller fraction

Instruction: There are two cases:

- ✓ Two exponent value are equal (exp_diff = 0):

- If the difference Fract_A – Fract_B is positive: Bigger_fract = Fract_A; Smaller_fract = Fract_B.

- If the difference Fract_A – Fract_B is negative: Bigger_fract = Fract_B; Smaller_fract = Fract_A.

- ✓ Two exponent value are not equal (exp_diff ≠ 0): Bigger_fract is the operand which has larger exponent.

1. Create a new Quartus project for your circuit.
2. Write a System Verilog description for the circuit.
3. Compile the code. Use the Quartus RTL Viewer tool to examine the gate-level circuit produced from the code.
4. Simulate the behavior of your code. Test the functionality of your design by inputting various data values and observing the generated outputs.

Check: Your report has to show two results: RTL viewer and Simulation results.

Pre Laboratory 3:

FLOATING-POINT ADDER/SUBTRACTOR

EXERCISE 3

Objective: Shift a given number to the right by an amount.

Requirement: Create a System Verilog module which shift a given number to the right by an amount. This module has following IO definitions:

Inputs	Smaller_fract[3:0]	4-bit input
Outputs	Shifted_fract[5:0]	4-bit output

Instruction:

- ✓ Anwers following questions:
  1. Why do we only need to shift the smaller fraction?
  2. Why do we reserve 6 bits to represent the shifted_fract?
  3. How many bits we need to represent the amount to right shift?
- ✓ Identify the amount to right shift: the amount to right shift is the value of the result exp_diff. If the exp_diff is negative (i.e. exp_B is larger), the exp_diff is in two's compliment form. Hence, perform another two's compliment to get a positive shift value.

  (Hint: two compliment can simply be implemented by adding minus sign before exp_diff)
- ✓ Write System Verilog code that create shifted_fract signal.

  (Note:

  - Because the fraction is a 4-bit signal, the shifted_fract is 0 when the shift amount is larger than 4.
  - Normalized input has suppress storage of the leading 1 (the significand is **1+F**). Remember to add this **bit 1** when shifting the fraction.)

1. Create a new Quartus project for your circuit.
2. Write a System Verilog description for the circuit.
3. Compile the code. Use the Quartus RTL Viewer tool to examine the gate-level circuit produced from the code.
4. Simulate the behavior of your code. Test the functionality of your design by inputting various data values and observing the generated outputs.

Check: Your report has to show two results:

- ➢ The waveform to prove the circuit works correctly.
- ➢ The result of RTL viewer.

Pre Laboratory 3:

FLOATING-POINT ADDER/SUBTRACTOR

EXERCISE 4

Objective: Normalize the final exponent and fraction value so it can be represented in floating point format.

Requirement: Create a System Verilog module normalizing the final exponent and fraction value so it can be represented in floating point format. This module has following IO definitions:

Inputs	Temp_fract[5:0]	6-bit input, result of the addition/subtraction between 2 fraction
Outputs	Result_fract[3:0]	4-bit output, fraction of the result
	Result_exp[2:0]	3-bit output, exponent of the result

Instruction:

You have to find the first "1" - i.e. locate a one and shift accordingly in order to get the implicite leading one for floating point representation. Besides, students should identify whether the exponent value increases/decreases and the amount to be added/suctracted to the exponent value.

1. Create a new Quartus project for your circuit.
2. Write a System Verilog description for the circuit.
3. Compile the code. Use the Quartus RTL Viewer tool to examine the gate-level circuit produced from the code.
4. Simulate the behavior of your code. Test the functionality of your design by inputting various data values and observing the generated outputs.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.

Pre Laboratory 3:

FLOATING-POINT ADDER/SUBTRACTOR

EXERCISE 5

Objective: Determines the sign of the resulting value.

Requirement: Create a System Verilog module determining the sign of the resulting value:

Instruction:

If exp_B > exp_A: Result_sign = sign_B, otherwise Result_sign = sign_A

If exp_A = exp_B, continue to check the fraction value: If Fract_B is bigger than Fract_A, Result_sign = sign_B, otherwise, Result_sign = sign_A

1. Create a new Quartus project for your circuit.
2. Write a System Verilog description for the circuit.
3. Compile the code. Use the Quartus RTL Viewer tool to examine the gate-level circuit produced from the code.
4. Simulate the behavior of your code. Test the functionality of your design by inputting various data values and observing the generated outputs.

Check: Your report has to show two results:

- The waveform to prove the circuit works correctly.
- The result of RTL viewer.