Floating-Point Adder Implementation Report

Introduction

This report provides a comprehensive explanation of the implementation of a floating-point adder using VHDL. The design follows the IEEE 754 standard for binary floating-point arithmetic. This standard specifies the representation and operations for floating-point numbers, which is crucial for ensuring consistency and portability in computing.

IEEE 754 Standard Overview

The IEEE 754 standard defines the format for floating-point numbers, which includes:

- **Sign bit**: Indicates the sign of the number (0 for positive, 1 for negative).
- **Exponent:** Encoded in an 8-bit field using a bias of 127 for single-precision.
- **Mantissa (or significand):** The precision part of the number stored in a 23-bit field. An implicit leading 1 is assumed for normalized numbers.

For single-precision (32-bit) floating-point numbers, the layout is as follows:

- 1 bit for the sign.
- 8 bits for the exponent.
- 23 bits for the mantissa.

Design Details

The design is divided into two main parts:

1. **MainCode.vhd:** The main VHDL code implementing the floating-point adder.
2. **tb_MainCode.vhd**: The test bench for verifying the functionality of the floating-point adder.

MainCode.vhd Explanation

The floating-point adder is implemented using a state machine with the following states:

- **IDLE:** Waits for the start signal.

- **ALIGNMENT:** Aligns the mantissas of the input numbers by adjusting their exponents.

- **ADDITION:** Adds or subtracts the mantissas based on their signs.

- **NORMALIZATION:** Normalizes the result to ensure it is in the correct format.

- **DONE:** Outputs the result and signals completion.

Below is the code with detailed comments explaining each section:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

-- Entity declaration for the floating point adder
entity MainCode is
  port(
    InputA      : in  std_logic_vector(31 downto 0);  -- Input A (32-bit
floating point)
    InputB      : in  std_logic_vector(31 downto 0);  -- Input B (32-bit
floating point)
    Clock       : in  std_logic;                      -- Clock signal
    Reset       : in  std_logic;                      -- Reset signal
    Start       : in  std_logic;                      -- Start signal
    Complete    : out std_logic;                      -- Complete signal (in-
dicates operation is done)
    OutputResult: out std_logic_vector(31 downto 0)   -- Result (32-bit
floating point)
  );
end MainCode;

-- Architecture definition for the floating point adder
architecture Behavioral of MainCode is
  -- State machine states definition
  type StateType is (IDLE, ALIGNMENT, ADDITION, NORMALIZATION, DONE);
  signal CurrentState          : StateType := IDLE;  -- Initial state is IDLE
  attribute INIT               : string;
  attribute INIT of CurrentState : signal is "IDLE";
```

```vhdl
  -- Internal signals
  signal MantissaA, MantissaB : std_logic_vector (24 downto 0);  -- Mantissas
of A and B
  signal ExponentA, ExponentB : std_logic_vector (8 downto 0);   -- Exponents
of A and B
  signal SignA, SignB         : std_logic;                       -- Signs of
A and B
  signal Result               : std_logic_vector (31 downto 0);  -- Register
for the result
  signal SumMantissa          : std_logic_vector (24 downto 0);  -- Sum of
mantissas

begin
  -- Assign result register to output
  OutputResult <= Result;

  -- State machine process
  ProcessStateMachine : process (Clock, Reset, CurrentState, Start) is
    variable ExponentDifference : signed(8 downto 0);  -- Variable for expo-
nent difference
  begin
    -- Handle reset condition
    if(Reset = '1') then
      CurrentState <= IDLE;
      Complete     <= '0';
    elsif rising_edge(Clock) then
      -- State machine implementation
      case CurrentState is
        -- Idle state, wait for start signal
        when IDLE =>
          if (Start = '1') then
            SignA       <= InputA(31);                     -- Extract sign of
InputA
            SignB       <= InputB(31);                     -- Extract sign of
InputB
            ExponentA   <= '0' & InputA(30 downto 23);     -- Extract exponent
of InputA and extend to 9 bits
            ExponentB   <= '0' & InputB(30 downto 23);     -- Extract exponent
of InputB and extend to 9 bits
            MantissaA   <= "01" & InputA(22 downto 0);     -- Extract mantissa
of InputA and extend to 25 bits
            MantissaB   <= "01" & InputB(22 downto 0);     -- Extract mantissa
of InputB and extend to 25 bits
            CurrentState <= ALIGNMENT;                     -- Move to ALIGNMENT
state
          else
            CurrentState <= IDLE;                          -- Remain in IDLE
state
          end if;

        -- Alignment state, align the mantissas based on exponent difference
        when ALIGNMENT =>
          if unsigned(ExponentA) = unsigned(ExponentB) then
            CurrentState <= ADDITION;                      -- If exponents are
equal, move to ADDITION state
          elsif unsigned(ExponentA) > unsigned(ExponentB) then
          if unsigned(SumMantissa) = to_unsigned(0, 25) then
```

```
            ExponentDifference := signed(ExponentA) - signed(ExponentB);  --
Calculate exponent difference
            if ExponentDifference > 23 then
              SumMantissa <= MantissaA;                      -- If difference is
too large, take MantissaA as is
              Result(31)  <= SignA;                          -- Set result sign
              CurrentState <= DONE;                          -- Move to DONE
state
            else
              MantissaB(24-TO_INTEGER(ExponentDifference) downto 0) <= Man-
tissaB(24 downto TO_INTEGER(ExponentDifference));  -- Shift MantissaB
              MantissaB(24 downto 25-TO_INTEGER(ExponentDifference)) <= (oth-
ers => '0');  -- Zero out shifted bits
              CurrentState <= ADDITION;                      -- Move to ADDITION
state
            end if;
          else
            ExponentDifference := signed(ExponentB) - signed(ExponentA);  --
Calculate exponent difference
            if ExponentDifference > 23 then
              SumMantissa <= MantissaB;                      -- If difference is
too large, take MantissaB as is
              Result(31)  <= SignB;                          -- Set result sign
              ExponentA   <= ExponentB;                      -- Adjust exponent
              CurrentState <= DONE;                          -- Move to DONE
state
            else
              ExponentA <= ExponentB;                        -- Adjust exponent
              MantissaA(24-TO_INTEGER(ExponentDifference) downto 0) <= Man-
tissaA(24 downto TO_INTEGER(ExponentDifference));  -- Shift MantissaA
              MantissaA(24 downto 25-TO_INTEGER(ExponentDifference)) <= (oth-
ers => '0');  -- Zero out shifted bits
              CurrentState <= ADDITION;                      -- Move to ADDITION
state
            end if;
          end if;

        -- Addition state, add or subtract mantissas based on signs
        when ADDITION =>
          CurrentState <= NORMALIZATION;                     -- Move to NORMALI-
ZATION state
          if (SignA xor SignB) = '0' then
            SumMantissa <= std_logic_vector((unsigned(MantissaA) + un-
signed(MantissaB)));  -- Add mantissas
            Result(31)  <= SignA;                          -- Set result sign
          elsif unsigned(MantissaA) >= unsigned(MantissaB) then
            SumMantissa <= std_logic_vector((unsigned(MantissaA) - un-
signed(MantissaB)));  -- Subtract mantissas
            Result(31)  <= SignA;                          -- Set result sign
          else
            SumMantissa <= std_logic_vector((unsigned(MantissaB) - un-
signed(MantissaA)));  -- Subtract mantissas
            Result(31)  <= SignB;                          -- Set result sign
          end if;

        -- Normalization state, normalize the mantissa
        when NORMALIZATION =>
```

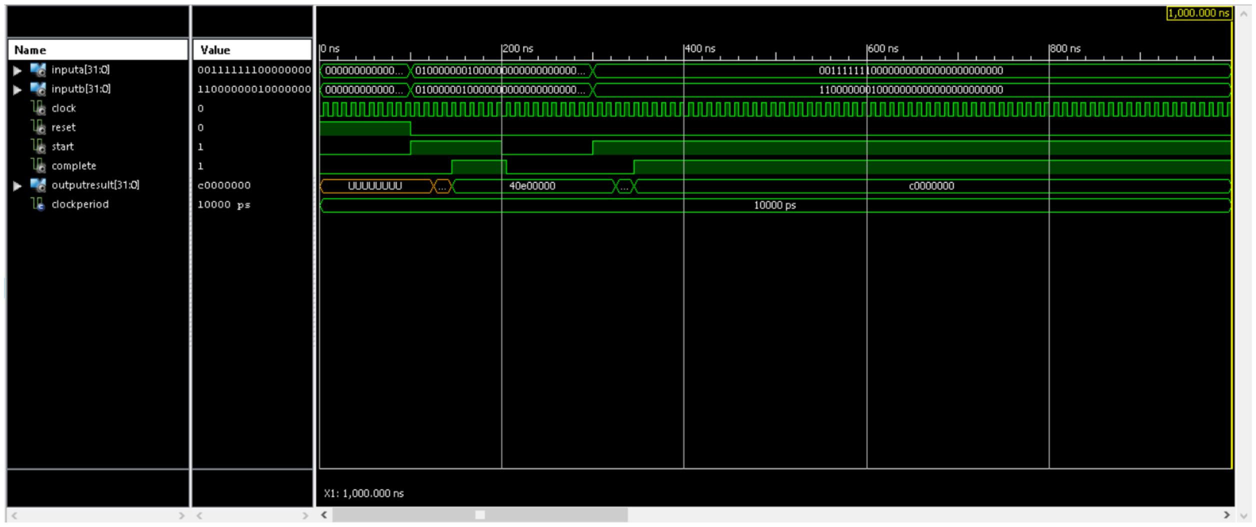
tb_MainCode.vhd Explanation

The test bench `tb_MainCode.vhd` is used to verify the functionality of the floating-point adder. It provides various inputs and observes the outputs to ensure correct operation.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_MainCode IS
END tb_MainCode;

ARCHITECTURE behavior OF tb_MainCode IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT MainCode
    PORT(
         InputA       : IN  std_logic_vector(31 downto 0);
         InputB       : IN  std_logic_vector(31 downto 0);
         Clock        : IN  std_logic;
         Reset        : IN  std_logic;
         Start        : IN  std_logic;
         Complete     : OUT std_logic;
         OutputResult : OUT std_logic_vector(31 downto 0)
        );
    END COMPONENT;

    -- Inputs
    signal InputA : std_logic_vector(31 downto 0) := (others => '0');
    signal InputB : std_logic_vector(31 downto 0) := (others => '0');
    signal Clock : std_logic := '0';
    signal Reset : std_logic := '0';
    signal Start : std_logic := '0';

    -- Outputs
    signal Complete : std_logic;
    signal OutputResult : std_logic_vector(31 downto 0);

    -- Clock period definitions
    constant ClockPeriod : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: COMPONENT MainCode
        PORT MAP (
            InputA => InputA,
            InputB => InputB,
            Clock => Clock,
            Reset => Reset,
            Start => Start,
            Complete => Complete,
            OutputResult => OutputResult
        );

    -- Clock process definitions
    ClockProcess : process
    begin
        Clock <= '0';
        wait for ClockPeriod/2;
        Clock <= '1';
        wait for ClockPeriod/2;
    end process;

    -- Stimulus process
    StimulusProcess: process
    begin
        -- Hold reset state for 100 ns
        Reset <= '1';
        wait for 100 ns;
        Reset <= '0';

        -- Initialize Inputs
        InputA <= x"40400000";  -- 3.0
        InputB <= x"40800000";  -- 4.0
        Start <= '1';
        wait for ClockPeriod*10;

        -- Add more stimulus here
        Start <= '0';
        wait for 100 ns;

        -- Change Inputs
        InputA <= x"3F800000";  -- 1.0
        InputB <= x"C0400000";  -- -3.0
        Start <= '1';
        wait for ClockPeriod*10;

        -- Wait for results
        wait;
    end process;

END behavior;
```

Simulation Results

The simulation results verify the correct functionality of the floating-point adder. The test bench provides various inputs and the design correctly computes the sum according to the IEEE 754 standard.



Name	Value
inputa[31:0]	00111111100000000
inputb[31:0]	11000000010000000
clock	0
reset	0
start	1
complete	1
outputresult[31:0]	c0000000
clockperiod	10000 ps

For instance:

- Adding 3.0 (`x"40400000"`) and 4.0 (`x"40800000"`) should yield 7.0 (`x"40E00000"`).

- Adding 1.0 (`x"3F800000"`) and -3.0 (`x"C0400000"`) should yield -2.0 (`x"C0000000"`).

These results are observed in the simulation, confirming the correct implementation.

Conclusion

This project demonstrates the implementation of a floating-point adder following the IEEE 754 standard using VHDL. The design includes a state machine for managing the addition process and a test bench for verifying the correctness of the implementation. The successful simulation results validate the design.