Implementasi Penjumlahan Angka Floating Point FPGA

GROUP BP04

Landasan Teori

FLoat

Float adalah representasi angka real dengan menggunakan biner.

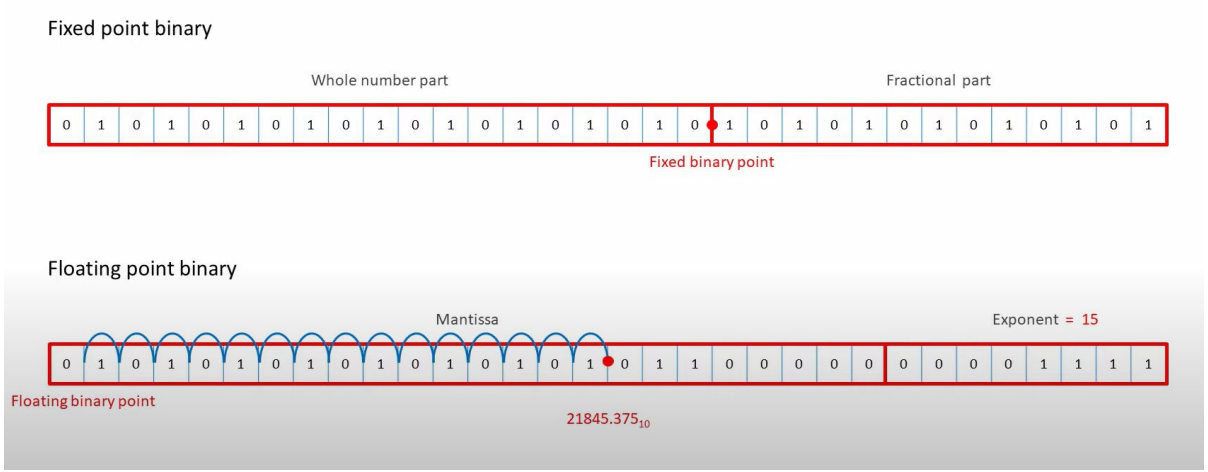
Metodenya adalah menggunakan sistem seperti notasi scientific.

1.5123(10) * 10^-21(10)
(scientific notation)

1.101101(2) * 2^-12(10)
(floating point notation)

Cara Kerja Float

Float adalah singkatan dari floating point number yang artinya titik yang membatasi angka dan desimalnya dapat diubah (floating). Hal ini berbeda dengan representasi angka real yang berupa fixed point number



Standardisasi Float

Float distrandardisasi dengan IEEE 754, dengan format 1 bit sign, 8 bit exponent, dan 23 bit (24 implied) mantissa. Mantissa, atau kadang disebut fraction, adalah bagian angka spesifiknya (kalau dalam notasi scientific yang di kiri). Float memiliki 32 bit, dan juga terdapat double yang memiliki 64 bit.

Mantissa

Mantissa selalu bersifat positif, kenegatifan mantissa diatur pada bit sign. Contohnya jika ingin ditulis -2, maka bit sign di-*set* untuk 1 dan *mantissa*-nya berisi 2.

Untuk melebihkan data yang dapat disimpan, mantissa memiliki 1 implisit pada MSB.

Exponent

Exponent dapat berupa negative. Namun, negativitasnya tidak bersistem 2’s complement tetapi dengan 0111(2) / 7(10) sebagai 0. Jadi 0 adalah 2^7 atau 127.

Jika ingin dicari exponent aslinya, exponent pada float harus dikurangi 127

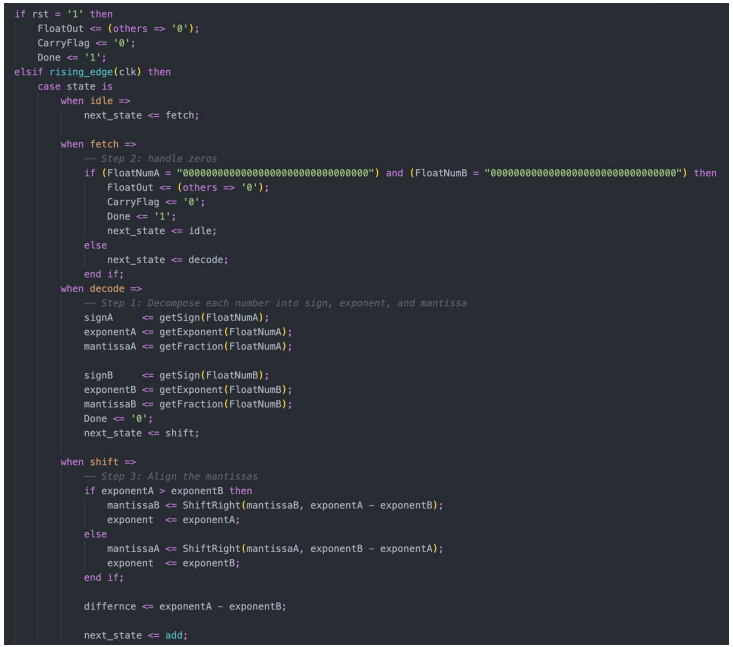
Penjelasan Component

Adder (Async)

- Bertanggung jawab untuk operasi penjumlahan asynchronous pada angka floating point.
- Membentuk pasangan mantissa yang sesuai dan menangani eksponen untuk hasil penjumlahan yang akurat
- Fungsi 'FullAdderBit', implementasi penuh dari operasi penjumlahan bit
- Process, melakukan penjumlahan asynchronous pada setiap bit
- Menggunakan carry untuk menangani perpindahan ke bit berikutnya
- Hasil akhir disimpan dalam sinyal 'sum' dan 'CarryFlag'

```
entity Adder is
    port (
        AddNumA : in std_logic_vector(31 downto 0);
        AddNumB : in std_logic_vector(31 downto 0);
        Sum : out std_logic_vector(31 downto 0); CarryFlag : out std_logic
    );
end entity Adder;

-- FullAdderBit function
architecture rtl of Adder is
    function FullAdderBit(A, B, Cin : std_logic) return std_logic_vector is
        variable SumBit : std_logic_vector(0 to 1);
    begin
        SumBit(0) := A xor B xor Cin;
        SumBit(1) := (A and B) or (B and Cin) or (A and Cin);
        return SumBit;
    end function FullAdderBit;

    signal Carry : std_logic := '0';

begin
    process (AddNumA, AddNumB)
        variable TempSum : std_logic_vector(31 downto 0) := (others => '0');
        variable TempCarry : std_logic := '0';
    begin
        for i in 0 to 31 loop
            TempSum(i) := FullAdderBit(AddNumA(i), AddNumB(i), TempCarry)(0);
            TempCarry := FullAdderBit(AddNumA(i), AddNumB(i), TempCarry)(1);
        end loop;

        Sum <= TempSum;
        CarryFlag <= TempCarry;
    end process;
end architecture rtl;
```

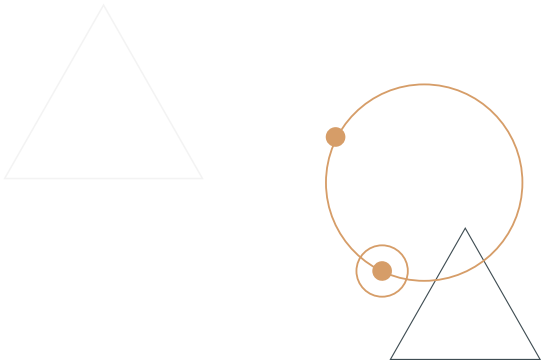
Float Adder (Sync)



```
if rst = '1' then
    FloatOut <= (others => '0');
    CarryFlag <= '0';
    Done <= '1';
elsif rising_edge(clk) then
    case state is
        when idle =>
            next_state <= fetch;

        when fetch =>
            -- Step 2: handle zeros
            if (FloatNumA = "00000000000000000000000000000000") and (FloatNumB = "00000000000000000000000000000000") then
                FloatOut <= (others => '0');
                CarryFlag <= '0';
                Done <= '1';
                next_state <= idle;
            else
                next_state <= decode;
            end if;
        when decode =>
            -- Step 1: Decompose each number into sign, exponent, and mantissa
            signA     <= getSign(FloatNumA);
            exponentA <= getExponent(FloatNumA);
            mantissaA <= getFraction(FloatNumA);

            signB     <= getSign(FloatNumB);
            exponentB <= getExponent(FloatNumB);
            mantissaB <= getFraction(FloatNumB);
            Done <= '0';
            next_state <= shift;

        when shift =>
            -- Step 3: Align the mantissas
            if exponentA > exponentB then
                mantissaB <= ShiftRight(mantissaB, exponentA - exponentB);
                exponent  <= exponentA;
            else
                mantissaA <= ShiftRight(mantissaA, exponentB - exponentA);
                exponent  <= exponentB;
            end if;

            differnce <= exponentA - exponentB;

            next_state <= add;
```

Jika reset aktif, atur output menjadi nol dan tandai operasi selesai. Jika terjadi tepi naik clock, lakukan langkah-langkah berikut berdasarkan keadaan saat ini:

- idle: Pindah ke fetch.
- fetch: Jika kedua angka input adalah nol, atur output menjadi nol dan tandai operasi selesai, lalu kembali ke idle. Jika tidak, pindah ke decode.
- decode: Dekomposisi setiap angka menjadi tanda, eksponen, dan mantissa. Setel Done menjadi '0' dan pindah ke shift.
- shift: Sesuaikan mantissa agar eksponennya sama. Hitung perbedaan eksponen. Pindah ke add.



Float Adder (Sync)

- add: Tambahkan atau kurangkan mantissa berdasarkan tanda angka. Hasilnya disimpan dalam mantissa, dan tanda hasil disimpan dalam signO. Pindah ke normalize.
- normalize: Normalisasi hasil dengan menyesuaikan eksponen dan mantissa. Pindah ke output setelah normalisasi.
- output: Kumpulkan hasil dengan merakit tanda, eksponen, dan mantissa. Atur CarryFlag menjadi '0', tandai operasi selesai, dan kembali ke idle.
- others: Jika keadaan tidak dikenali, kembalikan ke idle.

```
    when add =>
        -- Step 4: Add or subtract the mantissas based on the signs
        if signA = signB then
            mantissa <= std_logic_vector(unsigned(mantissaA) + unsigned(mantissaB));
            signO <= signA;
        else
            if mantissaA > mantissaB then
                mantissa <= std_logic_vector(unsigned(mantissaA) - unsigned(mantissaB));
                signO <= signA;
            else
                mantissa <= std_logic_vector(unsigned(mantissaB) - unsigned(mantissaA));
                signO <= signB;
            end if;
        end if;
        next_state <= normalize;

    when normalize =>
        -- Step 5: Normalize the result
        exponent <= exponent + 127;
        mantissa <= std_logic_vector(Shift_Right(unsigned(mantissa), 1));
        if mantissa(23) = '1' then
            exponent <= exponent + 1;
            mantissa <= ShiftRight(mantissa, 1);
        end if;
        next_state <= output;

    when output =>
        -- Step 6: Assemble the result
        FloatOut <= Assemble(signO, exponent, mantissa);
        CarryFlag <= '0';
        Done <= '1';
        next_state <= idle;

    when others =>
        next_state <= idle;
end case;
```

Decoder (Async)

- Berfungsi sebagai decoder sinyal kontrol dan alamat, mengarahkan input ke komponen yang tepat
- Desain asynchronous untuk respon yang cepat terhadap perubahan status
- Menggunakan ControlWord sebagai input untuk menentukan nilai FloatNumA, FloatNumB, dan SIgnSelect

```
entity Decoder is
    port (
        ControlWord : in std_logic_vector (64 downto 0);
        FloatNumA : out std_logic_vector (31 downto 0);
        FloatNumB : out std_logic_vector (31 downto 0);
        SignSelect : out std_logic
    );
end entity Decoder;

architecture rtl of Decoder is

begin
    FloatNumA <= ControlWord(31 downto 0);
    FloatNumB <= ControlWord(63 downto 32);
    SignSelect <= ControlWord(64);
end architecture rtl;
```

Top Level

Mendefinisikan suatu entitas (TopLevel) yang menggunakan tiga komponen utama: Decoder, FloatAdder, dan satu rangkaian kombinasional untuk mengelola keadaan (state).

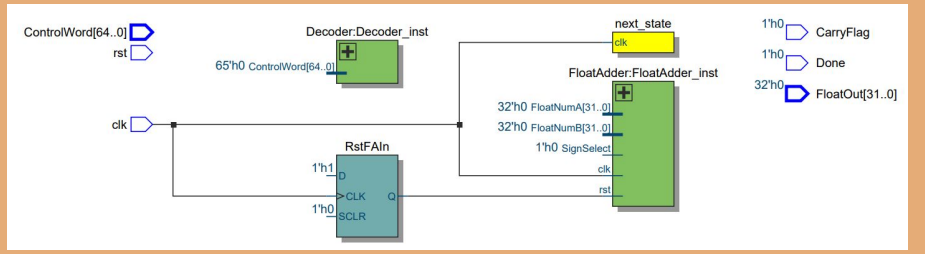
```
entity TopLevel is
    port (
        ControlWord : in std_logic_vector (64 downto 0);
        clk         : in std_logic;
        rst         : in std_logic;
        FloatOut    : out std_logic_vector (31 downto 0);
        CarryFlag   : out std_logic;
        Done        : out std_logic
    );
end entity TopLevel;

architecture rtl of TopLevel is
    type StateType is (Idle, Decode, Execution, Output);
    signal state : StateType := Idle;
    signal next_state : StateType := Idle;

    signal FloatNumADecOut, FloatNumBDecOut : std_logic_vector(31 downto 0);
    signal FloatNumAFAIn, FloatNumBFAIn : std_logic_vector(31 downto 0);
    signal SignSelectDecOut     : std_logic;
    signal SignSelectFAIn     : std_logic;
    signal RstFAIn     : std_logic;

    signal FloatAdderOut        : std_logic_vector(31 downto 0);
    signal FloatAdderCarryFlag  : std_logic;
    signal FloatAdderDone       : std_logic;
    signal ControlWordDecIn : std_logic_vector (64 downto 0);

    -- Component Declarations
    component Decoder
        port (
            ControlWord : in std_logic_vector (64 downto 0);
            FloatNumA   : out std_logic_vector (31 downto 0);
            FloatNumB   : out std_logic_vector (31 downto 0);
            SignSelect  : out std_logic
        );
    end component;

    component FloatAdder
        port (
            FloatNumA   : in std_logic_vector (31 downto 0);
            FloatNumB   : in std_logic_vector (31 downto 0);
            clk         : in std_logic;
            rst         : in std_logic;
            SignSelect  : in std_logic;
            FloatOut    : out std_logic_vector (31 downto 0);
            CarryFlag   : out std_logic;
            Done        : out std_logic
        );
    end component;
```

```
if rst = '1' then
    state <= Idle;
elsif rising_edge(clk) then
    case state is
        when Idle =>
            if rising_edge(clk) then
                -- reset signals
                FloatNumADecOut <= (others => '0');
                FloatNumBDecOut <= (others => '0');
                FloatNumAFAIn <= (others => '0');
                FloatNumBFAIn <= (others => '0');
                SignSelectDecOut <= '0';
                SignSelectFAIn <= '0';
                RstFAIn <= '1';
                FloatAdderOut <= (others => '0');
                FloatAdderCarryFlag <= '0';
                FloatAdderDone <= '0';
                ControlWordDecIn <= (others => '0');

                next_state <= Decode;
            end if;

        when Decode =>
            if rising_edge(clk) then
                -- Input signals
                ControlWordDecIn <= ControlWord;

                next_state <= Execution;
            end if;

        when Execution =>
            if FloatAdderDone = '1' then
                -- Output signals
                FloatNumAFAIn <= FloatNumADecOut;
                FloatNumBFAIn <= FloatNumBDecOut;
                SignSelectFAIn <= SignSelectDecOut;
                RstFAIn <= '0';

                next_state <= Output;
            else
                next_state <= Execution;
            end if;

        when Output =>
            -- Output signals
            FloatOut <= FloatAdderOut;
            CarryFlag <= FloatAdderCarryFlag;
            Done <= FloatAdderDone;

            next_state <= Idle;
    end case;
```

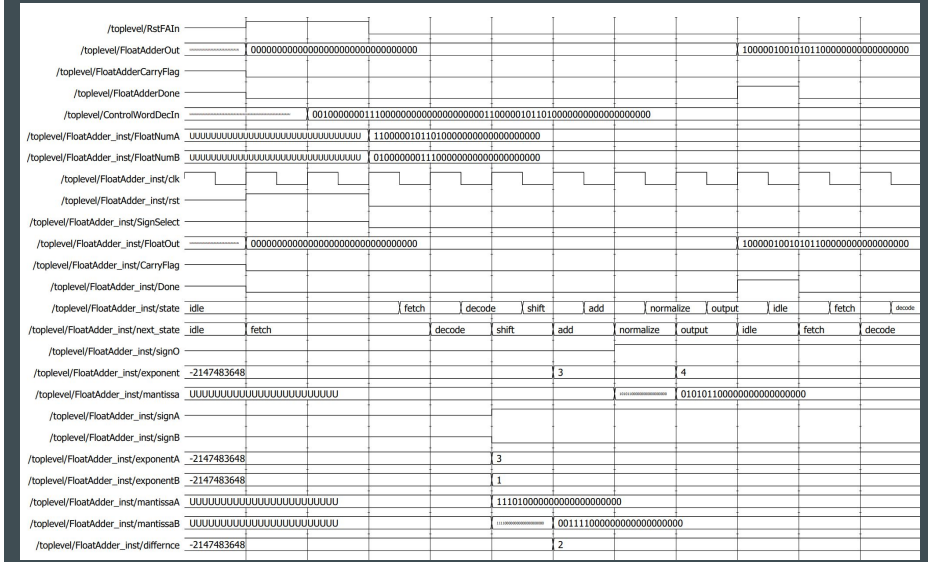
Top Level

1. Jika reset aktif, set keadaan menjadi Idle.
2. Jika terjadi tepi naik clock, jalankan logika berdasarkan state:
    - Pada state Idle, lakukan inisialisasi dan pindah ke Decode.
    - Pada state Decode, terima input dan pindah ke Execution.
    - Pada state Execution, tunggu hingga FloatAdderDone aktif, lalu kirim output dan pindah ke Output.
    - Pada state Output, kirim output dan kembali ke Idle.
    - Jika state unknown, kembali ke Idle.

Testing



Synthesize



Waveform

Test Bench

```
-- Stimulus process
stim_proc: process
begin
    -- Case 1
    AddNumA <= std_logic_vector(to_unsigned(15, 32));
    AddNumB <= std_logic_vector(to_unsigned(10, 32));
    wait for 100 ps;
    assert Sum = std_logic_vector(to_unsigned(25, 32)) report "Test failed for case 1" severity error;

    -- Case 2
    AddNumA <= std_logic_vector(to_unsigned(123, 32));
    AddNumB <= std_logic_vector(to_unsigned(456, 32));
    wait for 100 ps;
    assert Sum = std_logic_vector(to_unsigned(579, 32)) report "Test failed for case 2" severity error;

    -- Case 3
    AddNumA <= std_logic_vector(to_unsigned(1223, 32));
    AddNumB <= std_logic_vector(to_unsigned(4546, 32));
    wait for 100 ps;
    assert Sum = std_logic_vector(to_unsigned(5769, 32)) report "Test failed for case 3" severity error;

    -- Case 4
    AddNumA <= std_logic_vector(to_unsigned(1, 32));
    AddNumB <= "11111111111111111111111111111111";
    wait for 100 ps;
    assert CarryFlag = '1' report "Test failed for case 4" severity error;
```

Adder

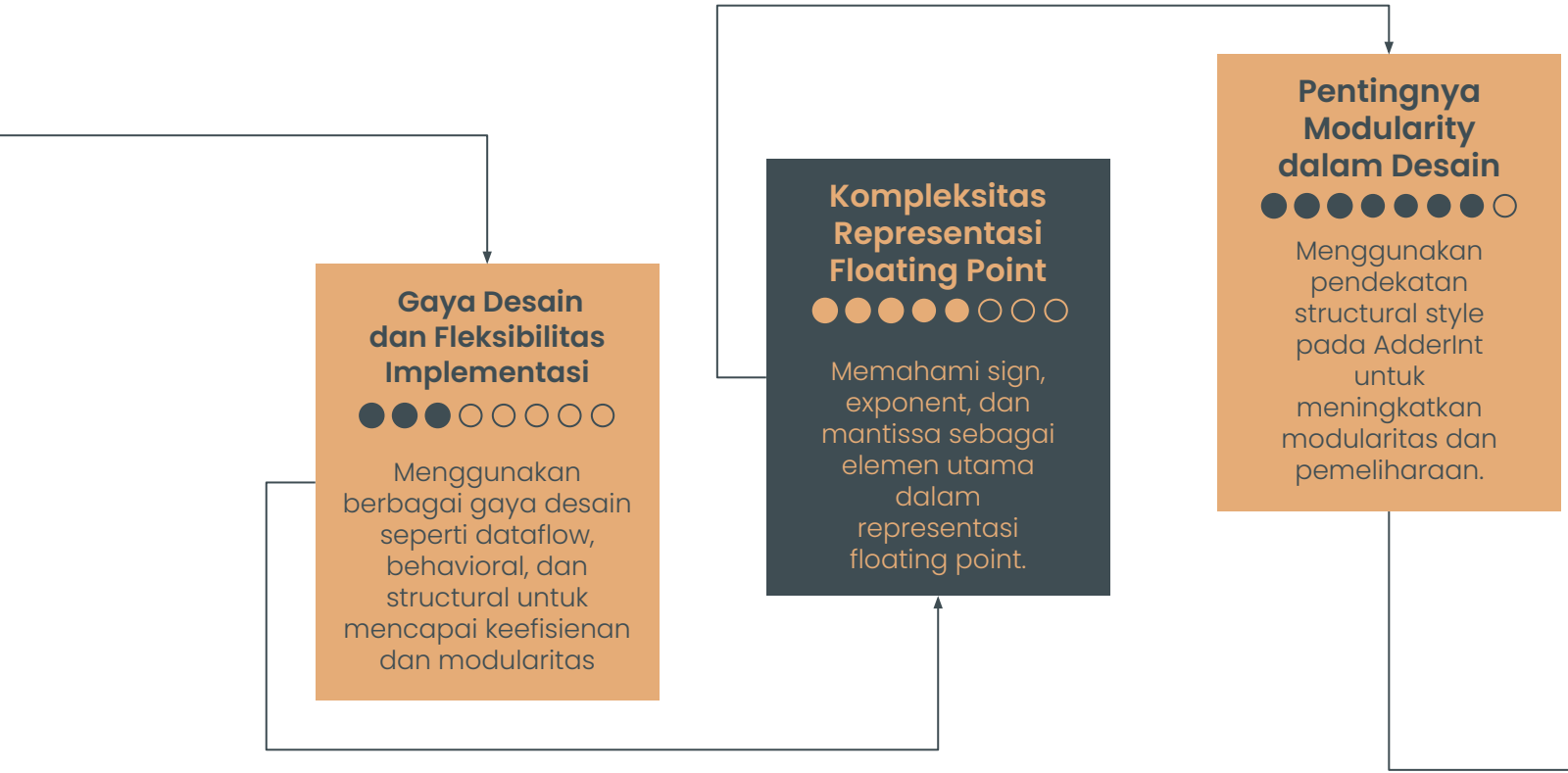
```
begin
    -- Case 1
    AddNumA := std_logic_vector(to_unsigned(15, 32));
    AddNumB := std_logic_vector(to_unsigned(10, 32));
    SignSelectVar := '0';
    ControlWord <= SignSelectVar & AddNumA & AddNumB;
    wait for 100 ps;
    assert FloatNumA = AddNumA report "Test failed for case 1 - FloatNumA" severity error;
    assert FloatNumB = AddNumB report "Test failed for case 1 - FloatNumB" severity error;
    assert SignSelect = SignSelectVar report "Test failed for case 1 - SignSelectVar" severity error;

    -- Case 2
    AddNumA := std_logic_vector(to_unsigned(165, 32));
    AddNumB := std_logic_vector(to_unsigned(410, 32));
    SignSelectVar := '0';
    ControlWord <= SignSelectVar & AddNumA & AddNumB;
    wait for 100 ps;
    assert FloatNumA = AddNumA report "Test failed for case 1 - FloatNumA" severity error;
    assert FloatNumB = AddNumB report "Test failed for case 1 - FloatNumB" severity error;
    assert SignSelect = SignSelectVar report "Test failed for case 1 - SignSelectVar" severity error;


    -- Case 3
    AddNumA := std_logic_vector(to_unsigned(65, 32));
    AddNumB := std_logic_vector(to_unsigned(10, 32));
    SignSelectVar := '1';
    ControlWord <= SignSelectVar & AddNumA & AddNumB;
    wait for 100 ps;
    assert FloatNumA = AddNumA report "Test failed for case 1 - FloatNumA" severity error;
    assert FloatNumB = AddNumB report "Test failed for case 1 - FloatNumB" severity error;
    assert SignSelect = SignSelectVar report "Test failed for case 1 - SignSelectVar" severity error;
    wait;
end process;
```

Decoder

```
-- Test case 1
input1 <= "01000000111010000000000000000000"; --7.25
input2 <= "01000000111010000000000000000000"; --7.25

assert expected_output = "01000001011010000000000000000000"; --14.5
report "Float Adder failed" severity error;

-- Test case 2
input1 <= "11000001011010000000000000000000"; -- -14.5
input2 <= "01000000011100000000000000000000"; -- 3.75

assert expected_output = "10000001000110000000000000000000"; -- -10.75
report "Float Adder failed" severity error;

-- Test case 3
input1 <= "11000010010110011000000000000000"; -- -54.375
input2 <= (others => '0'); -- 0

assert expected_output = "11000010010110011000000000000000"; -- -54.375
report "Float Adder failed" severity error;

-- Test case 4
input1 <= (others => '0'); -- 0
input2 <= "01000010010110011000000000000000"; -- 54.375

assert expected_output = "01000010010110011000000000000000"; -- 54.375
report "Float Adder failed" severity error;

-- Test case 5
input1 <= "01010010000011111011001101110111"; -- 154297810944
input2 <= "11000000000000000000000000000000"; -- -2

assert expected_output = "10010010000011111011001101110111"; -- -154297810944
report "Float Adder failed" severity error;
```

Float Adder

Kesimpulan


Gaya Desain dan Fleksibilitas Implementasi
Menggunakan berbagai gaya desain seperti dataflow, behavioral, dan structural untuk mencapai keefisienan dan modularitas
Kompleksitas Representasi Floating Point
Memahami sign, exponent, dan mantissa sebagai elemen utama dalam representasi floating point.
Pentingnya Modularity dalam Desain
Menggunakan pendekatan structural style pada AdderInt untuk meningkatkan modularitas dan pemeliharaan.

Thank You


slidesgo