|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |

Институт информационных технологий

Кафедра вычислительной техники

|  |
| --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**  **Разработка сумматора чисел с плавающей точкой** |
| *Разработка программно-аппаратного обеспечения информационных и автоматизированных систем* |

|  |  |  |
| --- | --- | --- |
| Отчет представлен  к рассмотрению:  Студент группы ИКМО-05-18 | « » декабря 2022 г. | Пленкин А.В. |
|  |  | (подпись и расшифровка подписи) |
|  |  |  |
| Отчет допущен  к защите: | « » декабря 2022 г. |  |
|  |  | (подпись и расшифровка подписи) |
| Руководитель  от кафедры | « » декабря 2022 г. | Унгер А.Ю. |
|  |  | (подпись и расшифровка подписи) |
|  |  |  |

Москва 2022

**Оглавление**

[Введение 3](#_Toc122278907)

[Использованная литература 11](#_Toc122278908)

[Листинг файлов с исходным кодом 12](#_Toc122278909)

# Введение

Компьютерные программы, также как и прошивки для микросхем, представляют собой последовательность нулей и единиц. Интерпретация того, чем является та или иная последовательность есть забота разработчика данной программы или данной микросхемы. В соответствии с решаемой задачей разработчик может выбирать между целочисленными данными и данными в формате с плавающей точкой (floating point). Стандарт IEEE 754 определяет формат и правила работы для чисел с плавающей точкой одинарной и двойной точности. Язык VHDL имеет встроенный тип для работы с числами с плавающей точкой. Однако данный тип весьма сложен и иногда может приводить к ошибкам синтеза. Кроме того, для того чтобы использовать встроенный тип правильно, необходимо хорошо понимать принципы работы с числами с плавающей точкой.

**Справка.** При *нормальной* форме записи величина мантиссы лежит в диапазоне , т.е. . Такая форма порождает неоднозначность, при которой возможно записать одно число несколькими эквивалентными способами. Например, 0.0001 можно записать как или или . Для устранения неоднозначности применяют нормализованную форму записи, при которой мантисса двоичного числа принимает значения в диапазоне [1,2). При этом, все числа (кроме 0) можно записать единственным образом.

**Цель работы:** на базе программируемой логической интегральной схемы (ПЛИС) спроектировать сумматор чисел с плавающей точкой следующего формата.

Мантисса 8 бит.

Порядок 4 бита. Знак порядка представлен смещением на половину от длины интервала возможных значений.

Само число может быть положительным или отрицательным, о чем свидетельствует бит знака.

**Итого:** 8 бит мантисса + 4 бита порядок + 1 бит знака = 13 бит.

Учитывая эти замечания, наибольшее и наименьшие по модулю числа таковы: и , соответственно. Разработку сумматора разумно проводить в 4 этапа.

**Порядок выполнения.**

1. Отсортировать числа на большее/меньшее.

Сортировка чисел на большее/меньшее – осуществляется с помощью сравнения отдельно порядков и, в случае если порядки одинаковы, мантисс исходных чисел.

1. Выровнять порядки. При этом, порядок меньшего числа должен соответствовать порядку большего.

Выравнивание порядков – реализуется с помощью логического сдвига порядка меньшего числа вправо.

1. Сложить мантиссы обоих чисел с учетом знака.

Сложение с учетом знака – реализует сложение в том случае, если знаки обоих чисел одинаковы, и вычитание из большего числа меньшее, если знаки различаются. Результат помещается в шину увеличенной разрядности (sum), которая позволит учесть случай переноса в следующий разряд. В зависимости от того, в какой позиции сигнала sum появляется первая единица, устанавливается значение сигнала lead0.

1. Нормализовать результат.

Нормализация результата – начинается со сдвига влево мантиссы суммы на lead0 битов. Далее, необходимо учесть особые случаи:

* В случае переноса (наличие единицы в старшем бите sum) порядок результата увеличивается на 1;
* В случае, если количество ведущих нулей результата сложения (lead0) больше порядка большего числа, результат обнуляется (мантисса и порядок становятся равными нулю);
* В случае, если количество ведущих нулей результата сложения (lead0) меньше порядка большего числа, порядок уменьшается на lead0 и происходит сдвиг мантиссы влево на lead0 битов.

На рисунках 1-4 показаны фрагменты кода программы по этапам, с помощью которого спроектирован сумматор чисел с плавающей точкой. Полный листинг представлен в приложении к отчету.

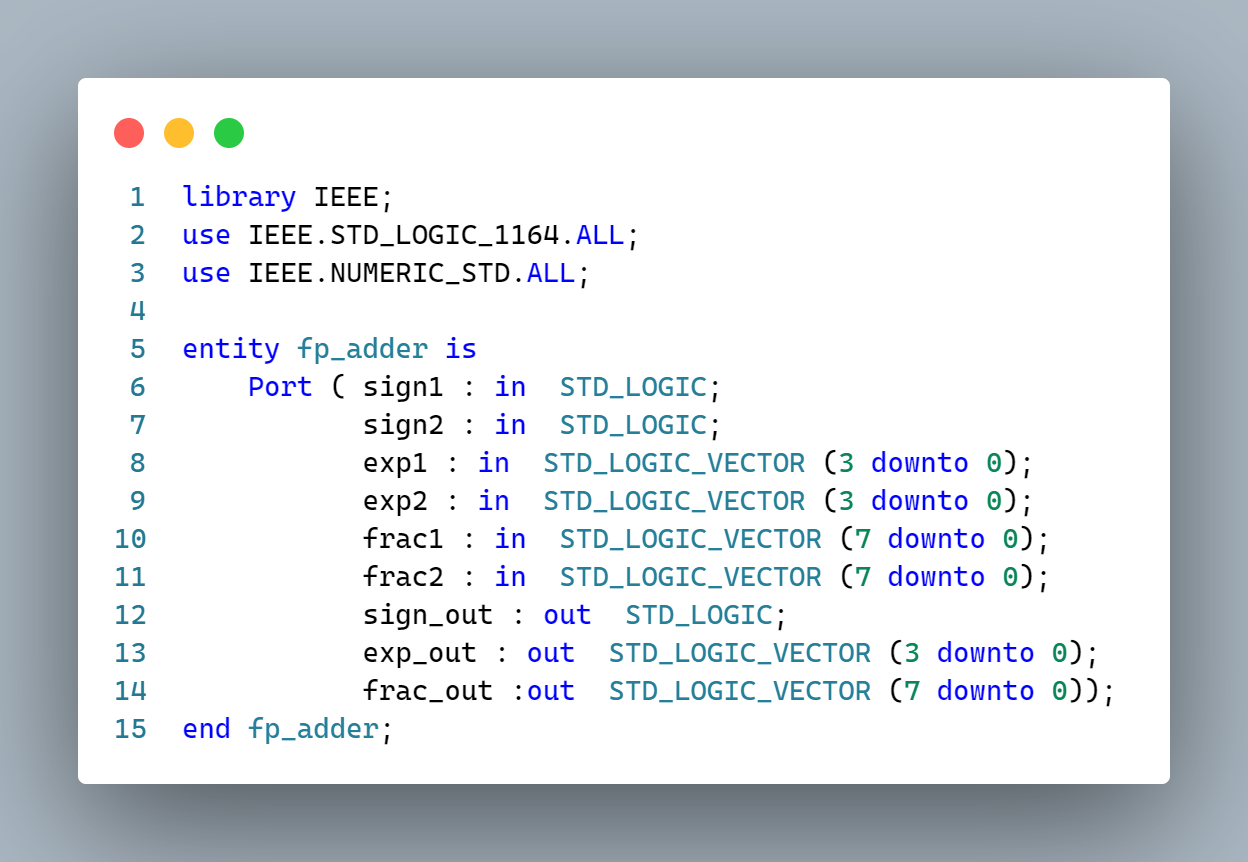


Рисунок 1 – описание интерфейса сумматора.

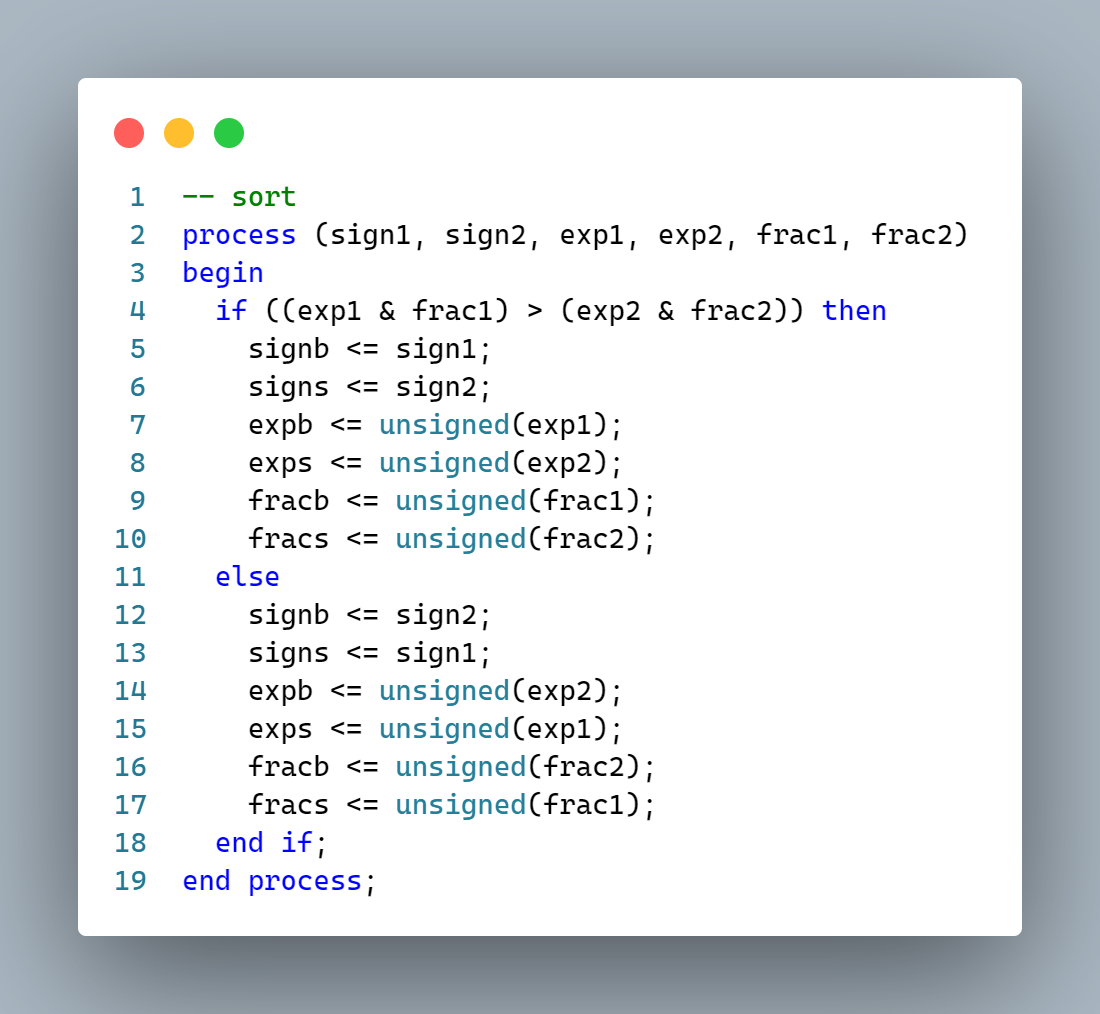


Рисунок 2 – этап сортировки

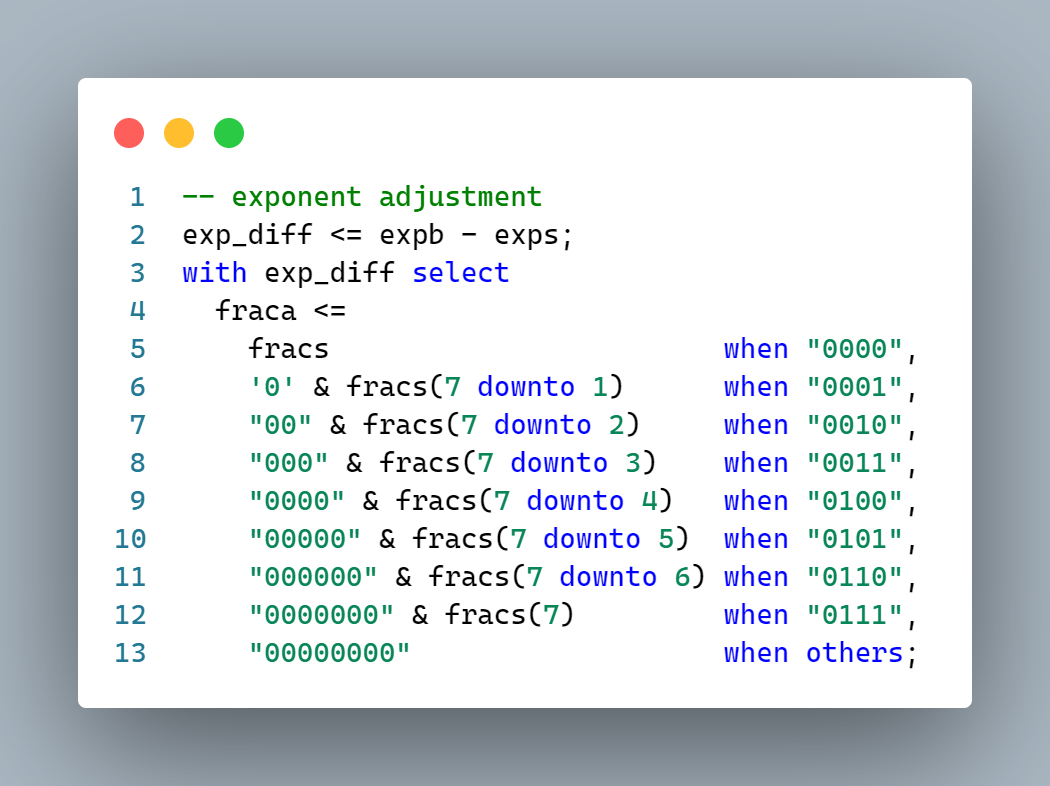


Рисунок 3 – выравнивание порядков

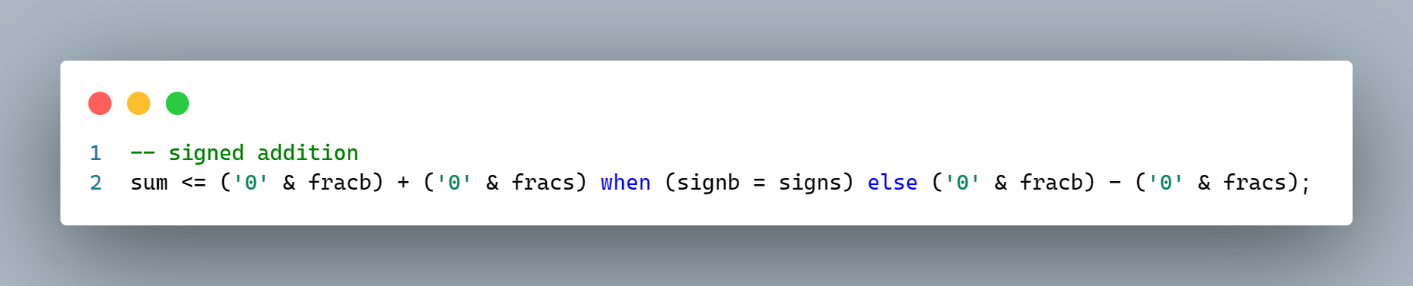


Рисунок 4 – сложение порядков с учетом знака

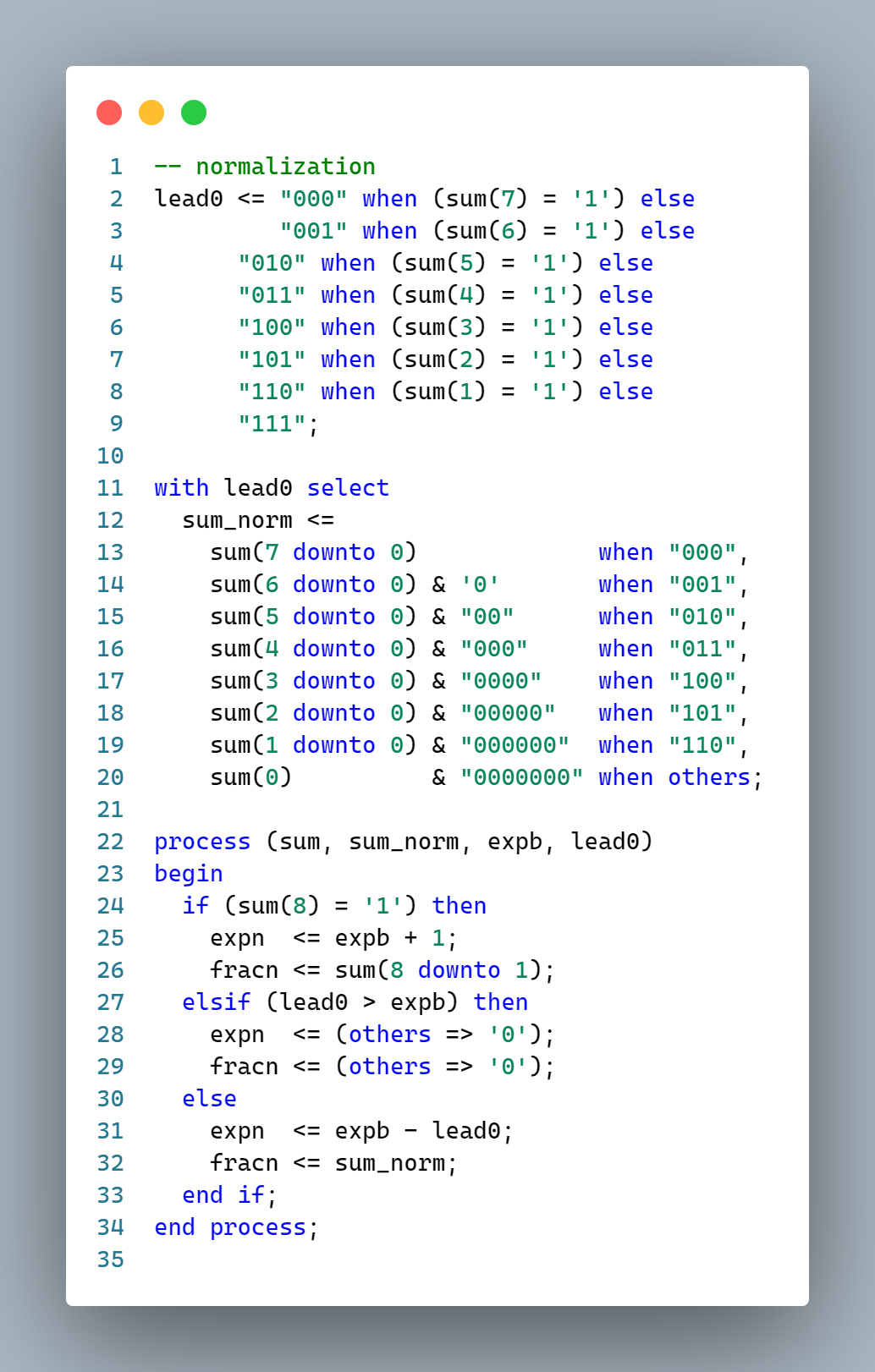


Рисунок 5 – нормализация результата

1. Написать Test Bench для сумматора

Для сумматора написано 5 различных тестов, которые проверяют работу устройства при различных входных данных. Пример кода теста приведен на рисунке. Общий код всех Test Bench’ей приведен в приложении к отчету.

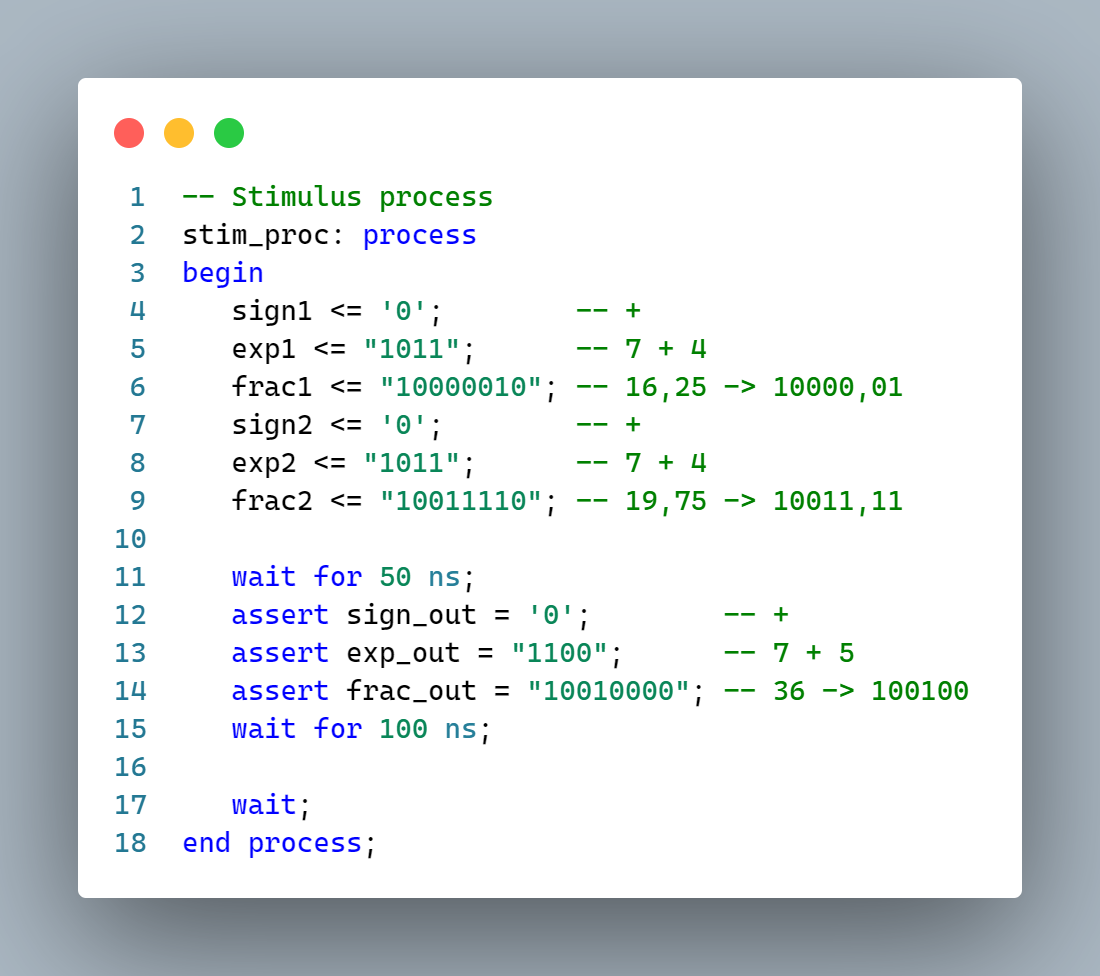


Рисунок 5 – фрагмент кода Test Bench

1. Протестировать работу сумматора с помощью симулятора ISim

На рисунках 6-11 представлены результаты работы сумматора с помощью симулятора ISim.

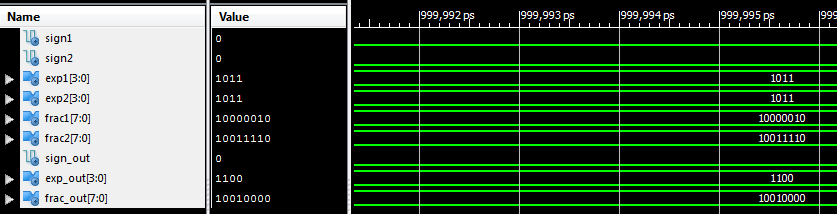


Рисунок 6 – сложение чисел 16,25 и 19,75

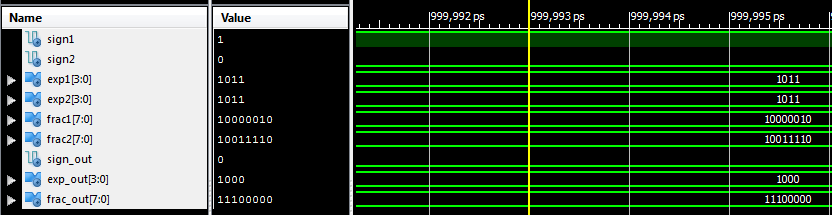


Рисунок 7 – сложение чисел -16,25 и 19,75

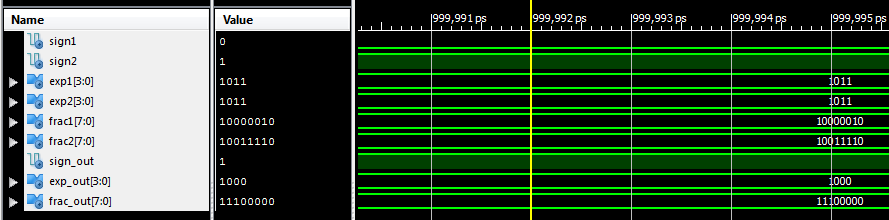


Рисунок 8 – сложение чисел 16,25 и -19,75

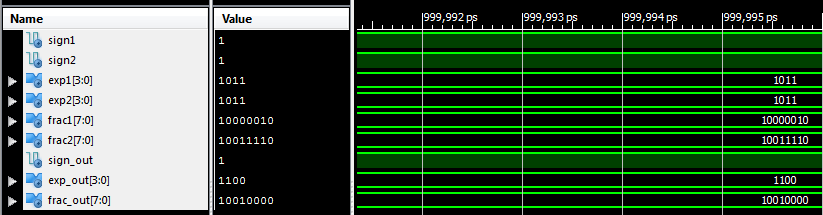


Рисунок 9 – сложение чисел -16,25 и -19,75

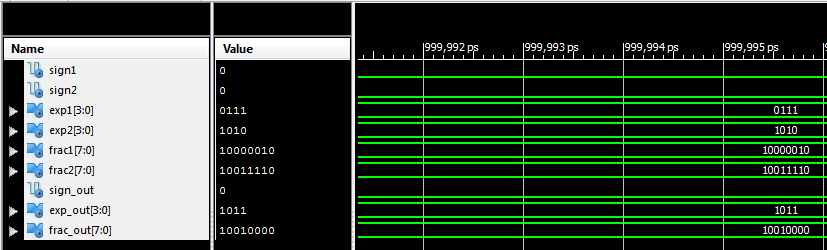


Рисунок 10 – сложение чисел 1,25 и 15,125

# Использованная литература

1. Потехин Д. С., Тарасов И. Е. Разработка программно-аппаратного обеспечения информационных и автоматизированных систем [Электронный ресурс]:конспект лекций. – М.: РТУ МИРЭА, 2020.

2. Строгонов А. В. Цифровая обработка сигналов в базисе программируемых логических интегральных схем [Электронный ресурс]:. - Санкт-Петербург: Лань, 2021.

3. Зайцев Е. И., Халабия Р. Ф. Организация ЭВМ и систем:учебное пособие. - М.: РТУ МИРЭА, 2019.

4. Певцов Е. Ф., Тарасов И. Е., Миннебаев В. М. Автоматизированное проектирование цифровых схем [Электронный ресурс]:учебное пособие. - М.: МИРЭА, 2016.

5. Красников А. К., Волосенков В. О., Никончук Я. С. Программирование микропроцессорных систем [Электронный ресурс]:учебное пособие. - М.: МГТУ МИРЭА, 2014.

6. Деменкова Т. А., Яровов Д. Д. Аппаратная реализация алгоритмов на FPGA [Электронный ресурс]:монография. - М.: МИРЭА, 2016.

# Листинг файлов с исходным кодом

l2\_fp\_adder.vhd

|  |
| --- |
| library IEEE;  use IEEE.STD\_LOGIC\_1164.ALL;  use IEEE.NUMERIC\_STD.ALL;  entity fp\_adder is      Port ( sign1 : in  STD\_LOGIC;             sign2 : in  STD\_LOGIC;             exp1 : in  STD\_LOGIC\_VECTOR (3 downto 0);             exp2 : in  STD\_LOGIC\_VECTOR (3 downto 0);             frac1 : in  STD\_LOGIC\_VECTOR (7 downto 0);             frac2 : in  STD\_LOGIC\_VECTOR (7 downto 0);             sign\_out : out  STD\_LOGIC;             exp\_out : out  STD\_LOGIC\_VECTOR (3 downto 0);             frac\_out :out  STD\_LOGIC\_VECTOR (7 downto 0));  end fp\_adder;  architecture fp\_adder\_arch of fp\_adder is  -- prefix 'b' stands for bigger  -- s - smaller  -- a - adjusted  -- n - normzlized  signal signb, signs : std\_logic;  signal expb, exps, expn : unsigned(3 downto 0);  signal fracb, fracs, fraca, fracn : unsigned(7 downto 0);  signal sum\_norm : unsigned(7 downto 0);  signal exp\_diff : unsigned(3 downto 0);  signal sum : unsigned(8 downto 0);  signal lead0 : unsigned(2 downto 0);  begin  -- sort  process (sign1, sign2, exp1, exp2, frac1, frac2)  begin    if ((exp1 & frac1) > (exp2 & frac2)) then      signb <= sign1;      signs <= sign2;      expb <= unsigned(exp1);      exps <= unsigned(exp2);      fracb <= unsigned(frac1);      fracs <= unsigned(frac2);    else      signb <= sign2;      signs <= sign1;      expb <= unsigned(exp2);      exps <= unsigned(exp1);      fracb <= unsigned(frac2);      fracs <= unsigned(frac1);    end if;  end process;  -- exponent adjustment  exp\_diff <= expb - exps;  with exp\_diff select    fraca <=      fracs                        when "0000",      '0' & fracs(7 downto 1)      when "0001",      "00" & fracs(7 downto 2)     when "0010",      "000" & fracs(7 downto 3)    when "0011",      "0000" & fracs(7 downto 4)   when "0100",      "00000" & fracs(7 downto 5)  when "0101",      "000000" & fracs(7 downto 6) when "0110",      "0000000" & fracs(7)         when "0111",      "00000000"                   when others;  -- signed addition  sum <= ('0' & fracb) + ('0' & fracs) when (signb = signs) else ('0' & fracb) - ('0' & fracs);  -- normalization  lead0 <= "000" when (sum(7) = '1') else           "001" when (sum(6) = '1') else        "010" when (sum(5) = '1') else        "011" when (sum(4) = '1') else        "100" when (sum(3) = '1') else        "101" when (sum(2) = '1') else        "110" when (sum(1) = '1') else        "111";  with lead0 select    sum\_norm <=      sum(7 downto 0)             when "000",      sum(6 downto 0) & '0'       when "001",      sum(5 downto 0) & "00"      when "010",      sum(4 downto 0) & "000"     when "011",      sum(3 downto 0) & "0000"    when "100",      sum(2 downto 0) & "00000"   when "101",      sum(1 downto 0) & "000000"  when "110",      sum(0)          & "0000000" when others;  process (sum, sum\_norm, expb, lead0)  begin    if (sum(8) = '1') then      expn  <= expb + 1;      fracn <= sum(8 downto 1);    elsif (lead0 > expb) then      expn  <= (others => '0');      fracn <= (others => '0');    else      expn  <= expb - lead0;      fracn <= sum\_norm;    end if;  end process;  -- output  sign\_out <= signb;  exp\_out <= std\_logic\_vector(expn);  frac\_out <= std\_logic\_vector(fracn);  end fp\_adder\_arch; |

l2\_fp\_adder\_tb\_1.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY fp\_addert\_test IS  END fp\_addert\_test;  ARCHITECTURE behavior OF fp\_addert\_test IS      COMPONENT fp\_adder      PORT(           sign1 : IN  std\_logic;           sign2 : IN  std\_logic;           exp1 : IN  std\_logic\_vector(3 downto 0);           exp2 : IN  std\_logic\_vector(3 downto 0);           frac1 : IN  std\_logic\_vector(7 downto 0);           frac2 : IN  std\_logic\_vector(7 downto 0);           sign\_out : OUT  std\_logic;           exp\_out : OUT  std\_logic\_vector(3 downto 0);           frac\_out : OUT  std\_logic\_vector(7 downto 0)          );      END COMPONENT;     --Inputs     signal sign1 : std\_logic := '0';     signal sign2 : std\_logic := '0';     signal exp1 : std\_logic\_vector(3 downto 0) := (others => '0');     signal exp2 : std\_logic\_vector(3 downto 0) := (others => '0');     signal frac1 : std\_logic\_vector(7 downto 0) := (others => '0');     signal frac2 : std\_logic\_vector(7 downto 0) := (others => '0');     --Outputs     signal sign\_out : std\_logic;     signal exp\_out : std\_logic\_vector(3 downto 0);     signal frac\_out : std\_logic\_vector(7 downto 0);  BEGIN     -- Instantiate the Unit Under Test (UUT)     uut: fp\_adder PORT MAP (            sign1 => sign1,            sign2 => sign2,            exp1 => exp1,            exp2 => exp2,            frac1 => frac1,            frac2 => frac2,            sign\_out => sign\_out,            exp\_out => exp\_out,            frac\_out => frac\_out          );     -- Stimulus process     stim\_proc: process     begin        sign1 <= '0';        -- +        exp1 <= "1011";      -- 7 + 4        frac1 <= "10000010"; -- 16,25 -> 10000,01        sign2 <= '0';        -- +        exp2 <= "1011";      -- 7 + 4        frac2 <= "10011110"; -- 19,75 -> 10011,11        wait for 50 ns;        assert sign\_out = '0';        -- +        assert exp\_out = "1100";      -- 7 + 5        assert frac\_out = "10010000"; -- 36 -> 100100        wait for 100 ns;        wait;     end process;  END; |

l2\_fp\_adder\_tb\_2.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY fp\_adder\_test\_2 IS  END fp\_adder\_test\_2;  ARCHITECTURE behavior OF fp\_adder\_test\_2 IS      COMPONENT fp\_adder      PORT(           sign1 : IN  std\_logic;           sign2 : IN  std\_logic;           exp1 : IN  std\_logic\_vector(3 downto 0);           exp2 : IN  std\_logic\_vector(3 downto 0);           frac1 : IN  std\_logic\_vector(7 downto 0);           frac2 : IN  std\_logic\_vector(7 downto 0);           sign\_out : OUT  std\_logic;           exp\_out : OUT  std\_logic\_vector(3 downto 0);           frac\_out : OUT  std\_logic\_vector(7 downto 0)          );      END COMPONENT;     --Inputs     signal sign1 : std\_logic := '0';     signal sign2 : std\_logic := '0';     signal exp1 : std\_logic\_vector(3 downto 0) := (others => '0');     signal exp2 : std\_logic\_vector(3 downto 0) := (others => '0');     signal frac1 : std\_logic\_vector(7 downto 0) := (others => '0');     signal frac2 : std\_logic\_vector(7 downto 0) := (others => '0');     --Outputs     signal sign\_out : std\_logic;     signal exp\_out : std\_logic\_vector(3 downto 0);     signal frac\_out : std\_logic\_vector(7 downto 0);  BEGIN     -- Instantiate the Unit Under Test (UUT)     uut: fp\_adder PORT MAP (            sign1 => sign1,            sign2 => sign2,            exp1 => exp1,            exp2 => exp2,            frac1 => frac1,            frac2 => frac2,            sign\_out => sign\_out,            exp\_out => exp\_out,            frac\_out => frac\_out          );     -- Stimulus process     stim\_proc: process     begin        sign1 <= '1';        -- -        exp1 <= "1011";      -- 7 + 4        frac1 <= "10000010"; -- 16,25 -> 10000,01        sign2 <= '0';        -- +        exp2 <= "1011";      -- 7 + 4        frac2 <= "10011110"; -- 19,75 -> 10011,11        wait for 50 ns;        assert sign\_out = '0';        -- +        assert exp\_out = "1000";      -- 7 + 1        assert frac\_out = "11100000"; -- 3.5 -> 11.1        wait for 100 ns;        wait;     end process;  END; |

l2\_fp\_adder\_tb\_3.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY fp\_adder\_test\_3 IS  END fp\_adder\_test\_3;    ARCHITECTURE behavior OF fp\_adder\_test\_3 IS      COMPONENT fp\_adder      PORT(           sign1 : IN  std\_logic;           sign2 : IN  std\_logic;           exp1 : IN  std\_logic\_vector(3 downto 0);           exp2 : IN  std\_logic\_vector(3 downto 0);           frac1 : IN  std\_logic\_vector(7 downto 0);           frac2 : IN  std\_logic\_vector(7 downto 0);           sign\_out : OUT  std\_logic;           exp\_out : OUT  std\_logic\_vector(3 downto 0);           frac\_out : OUT  std\_logic\_vector(7 downto 0)          );      END COMPONENT;     --Inputs     signal sign1 : std\_logic := '0';     signal sign2 : std\_logic := '0';     signal exp1 : std\_logic\_vector(3 downto 0) := (others => '0');     signal exp2 : std\_logic\_vector(3 downto 0) := (others => '0');     signal frac1 : std\_logic\_vector(7 downto 0) := (others => '0');     signal frac2 : std\_logic\_vector(7 downto 0) := (others => '0');     --Outputs     signal sign\_out : std\_logic;     signal exp\_out : std\_logic\_vector(3 downto 0);     signal frac\_out : std\_logic\_vector(7 downto 0);  BEGIN     uut: fp\_adder PORT MAP (            sign1 => sign1,            sign2 => sign2,            exp1 => exp1,            exp2 => exp2,            frac1 => frac1,            frac2 => frac2,            sign\_out => sign\_out,            exp\_out => exp\_out,            frac\_out => frac\_out          );     -- Stimulus process     stim\_proc: process     begin        sign1 <= '0';        -- +        exp1 <= "1011";      -- 7 + 4        frac1 <= "10000010"; -- 16,25 -> 10000,01        sign2 <= '1';        -- -        exp2 <= "1011";      -- 7 + 4        frac2 <= "10011110"; -- 19,75 -> 10011,11        wait for 50 ns;        assert sign\_out = '1';        -- -        assert exp\_out = "1000";      -- 7 + 1        assert frac\_out = "11100000"; -- 3.5 -> 11.1        wait for 100 ns;        wait;     end process;  END; |

l2\_fp\_adder\_tb\_4.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY fp\_adder\_test\_4 IS  END fp\_adder\_test\_4;  ARCHITECTURE behavior OF fp\_adder\_test\_4 IS      COMPONENT fp\_adder      PORT(           sign1 : IN  std\_logic;           sign2 : IN  std\_logic;           exp1 : IN  std\_logic\_vector(3 downto 0);           exp2 : IN  std\_logic\_vector(3 downto 0);           frac1 : IN  std\_logic\_vector(7 downto 0);           frac2 : IN  std\_logic\_vector(7 downto 0);           sign\_out : OUT  std\_logic;           exp\_out : OUT  std\_logic\_vector(3 downto 0);           frac\_out : OUT  std\_logic\_vector(7 downto 0)          );      END COMPONENT;     --Inputs     signal sign1 : std\_logic := '0';     signal sign2 : std\_logic := '0';     signal exp1 : std\_logic\_vector(3 downto 0) := (others => '0');     signal exp2 : std\_logic\_vector(3 downto 0) := (others => '0');     signal frac1 : std\_logic\_vector(7 downto 0) := (others => '0');     signal frac2 : std\_logic\_vector(7 downto 0) := (others => '0');     --Outputs     signal sign\_out : std\_logic;     signal exp\_out : std\_logic\_vector(3 downto 0);     signal frac\_out : std\_logic\_vector(7 downto 0);  BEGIN     -- Instantiate the Unit Under Test (UUT)     uut: fp\_adder PORT MAP (            sign1 => sign1,            sign2 => sign2,            exp1 => exp1,            exp2 => exp2,            frac1 => frac1,            frac2 => frac2,            sign\_out => sign\_out,            exp\_out => exp\_out,            frac\_out => frac\_out          );     -- Stimulus process     stim\_proc: process     begin        sign1 <= '1';        -- -        exp1 <= "1011";      -- 7 + 4        frac1 <= "10000010"; -- 16,25 -> 10000,01        sign2 <= '1';        -- -        exp2 <= "1011";      -- 7 + 4        frac2 <= "10011110"; -- 19,75 -> 10011,11        wait for 50 ns;        assert sign\_out = '1';        -- -        assert exp\_out = "1100";      -- 7 + 5        assert frac\_out = "10010000"; -- 36 -> 100100        wait for 100 ns;        wait;     end process;  END; |

l2\_fp\_adder\_tb\_5.vhd

|  |
| --- |
| LIBRARY ieee;  USE ieee.std\_logic\_1164.ALL;  ENTITY fp\_adder\_test\_5 IS  END fp\_adder\_test\_5;  ARCHITECTURE behavior OF fp\_adder\_test\_5 IS      COMPONENT fp\_adder      PORT(           sign1 : IN  std\_logic;           sign2 : IN  std\_logic;           exp1 : IN  std\_logic\_vector(3 downto 0);           exp2 : IN  std\_logic\_vector(3 downto 0);           frac1 : IN  std\_logic\_vector(7 downto 0);           frac2 : IN  std\_logic\_vector(7 downto 0);           sign\_out : OUT  std\_logic;           exp\_out : OUT  std\_logic\_vector(3 downto 0);           frac\_out : OUT  std\_logic\_vector(7 downto 0)          );      END COMPONENT;     signal sign1 : std\_logic := '0';     signal sign2 : std\_logic := '0';     signal exp1 : std\_logic\_vector(3 downto 0) := (others => '0');     signal exp2 : std\_logic\_vector(3 downto 0) := (others => '0');     signal frac1 : std\_logic\_vector(7 downto 0) := (others => '0');     signal frac2 : std\_logic\_vector(7 downto 0) := (others => '0');     signal sign\_out : std\_logic;     signal exp\_out : std\_logic\_vector(3 downto 0);     signal frac\_out : std\_logic\_vector(7 downto 0);  BEGIN     uut: fp\_adder PORT MAP (            sign1 => sign1,            sign2 => sign2,            exp1 => exp1,            exp2 => exp2,            frac1 => frac1,            frac2 => frac2,            sign\_out => sign\_out,            exp\_out => exp\_out,            frac\_out => frac\_out          );     -- Stimulus process     stim\_proc: process     begin        sign1 <= '0';        -- +        exp1 <= "0111";      -- 7 + 0        frac1 <= "10000010"; -- 1,25 -> 1,01        sign2 <= '0';        -- +        exp2 <= "1010";      -- 7 + 3        frac2 <= "10011110"; -- 15,125 -> 1111,001        wait for 50 ns;        assert sign\_out = '0';        -- +        assert exp\_out = "1011";      -- 7 + 4        assert frac\_out = "10010000"; -- 16,375 -> 10000,011        wait for 100 ns;        wait;     end process;  END; |