Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Кафедра електронно обчислювальної апаратури

Лабораторна робота №2

з курсу «Апаратні прискорювачі на мікросхемах програмованої логіки»

Виконав:

Студент 3 курсу

Групи ДК-02

Варіант 25

Садко Вячеслав

Київ 2023

Зміст

Мета : В Simulink реалізувати підсистему, що розраховує модуль і аргумент комплексного числа для вхідних даних у форматах з фіксованою комою і плаваючою комою

Розрахунок основних параметрів

- Представлення: знакове
- Розрядність цілої частини: N = 25, з яких старший розряд визнача сигнатуру числа
- Розрядність дробової частини:

$$M = 32 - N = 7 \tag{1}$$

- Максимальна ціла частина вхідного числа:

$$x = 2^{N-1} - 1 = 2^{24} - 1 = 16777215 \tag{2}$$

- Максимальна дробова частина вхідного числа:

$$y = 1 - \frac{1}{2^M} = 1 - \frac{1}{2^7} = 0.9921875 \tag{3}$$

- Максимальне додатнє вхідне число:

$$\max_{positive} = 16777215.9921875 \tag{4}$$

- Максимальне вхідне від’ємне вхідне число:

$$\max_{negative} = -16777216 \tag{5}$$

- Значення seed для першого генератора випадкових чисел: 25

Реалізація блок схеми обчислювача модуля і аргументу комплексного числа для вхідного аргументу з фіксованої комою

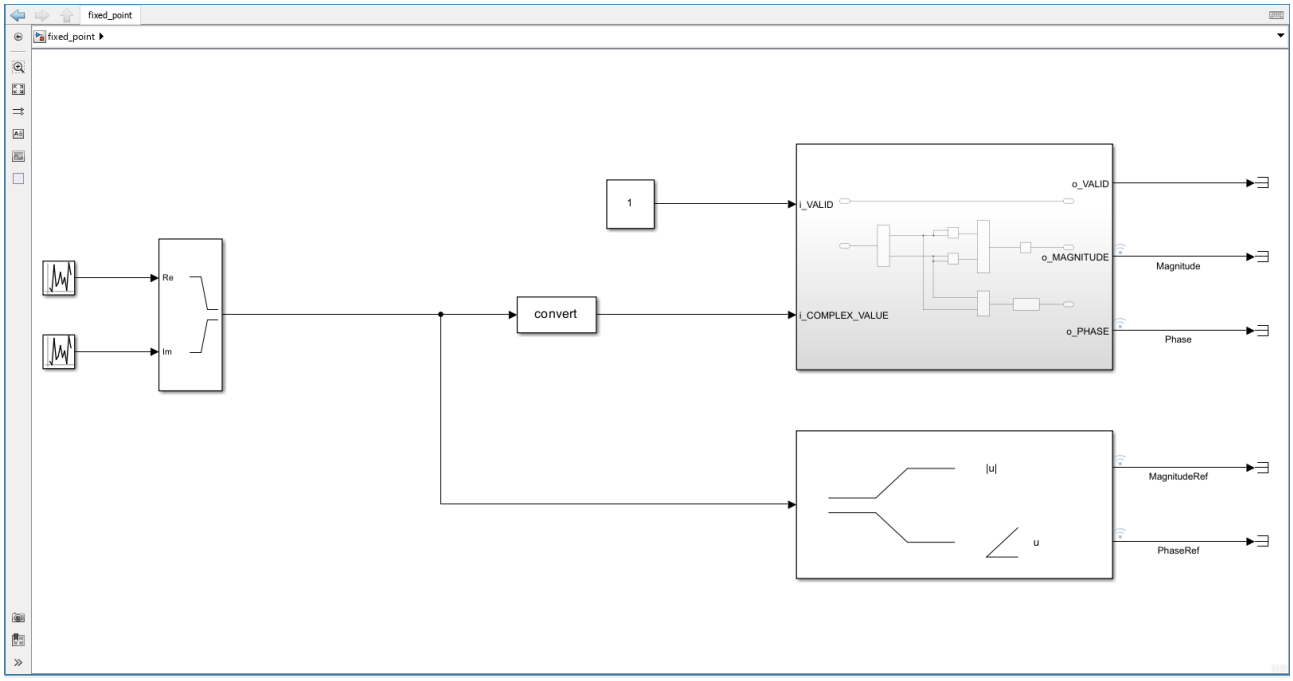

Рис. 1 Загальна блок-схема обчислювача для чисел з фіксованою комою

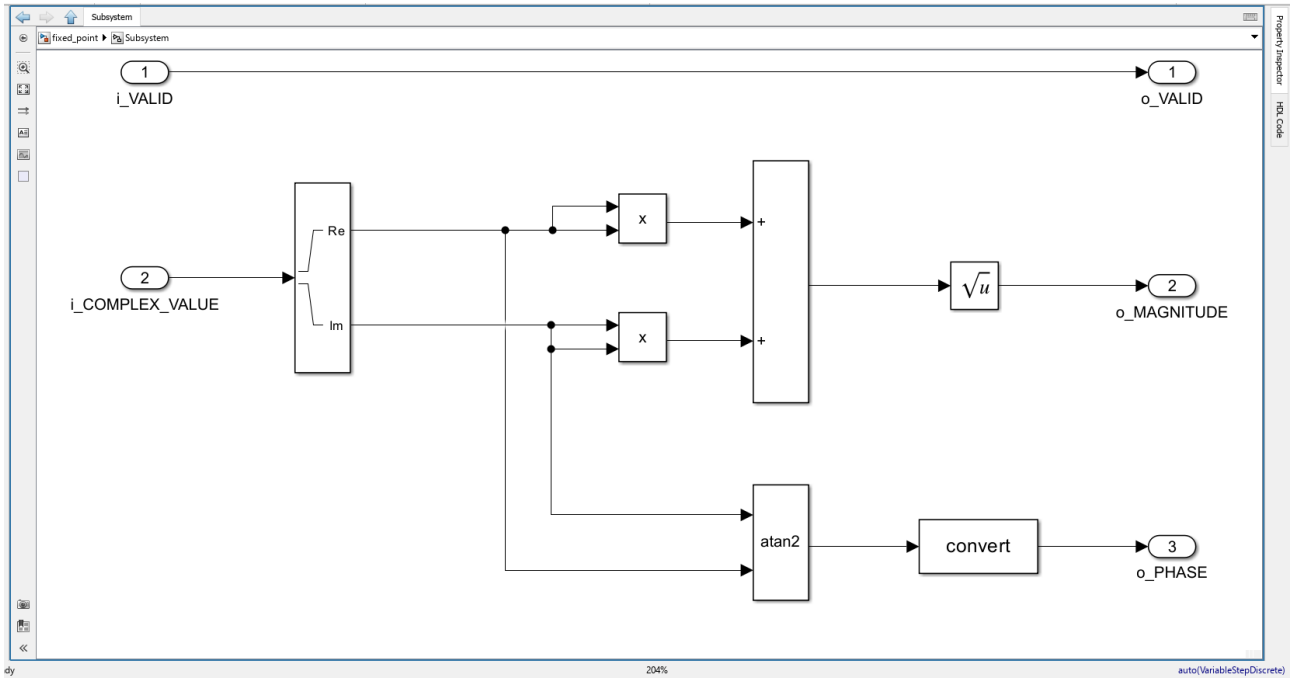

Рис. 2 Підсистема Subsystem

Налаштування блоків

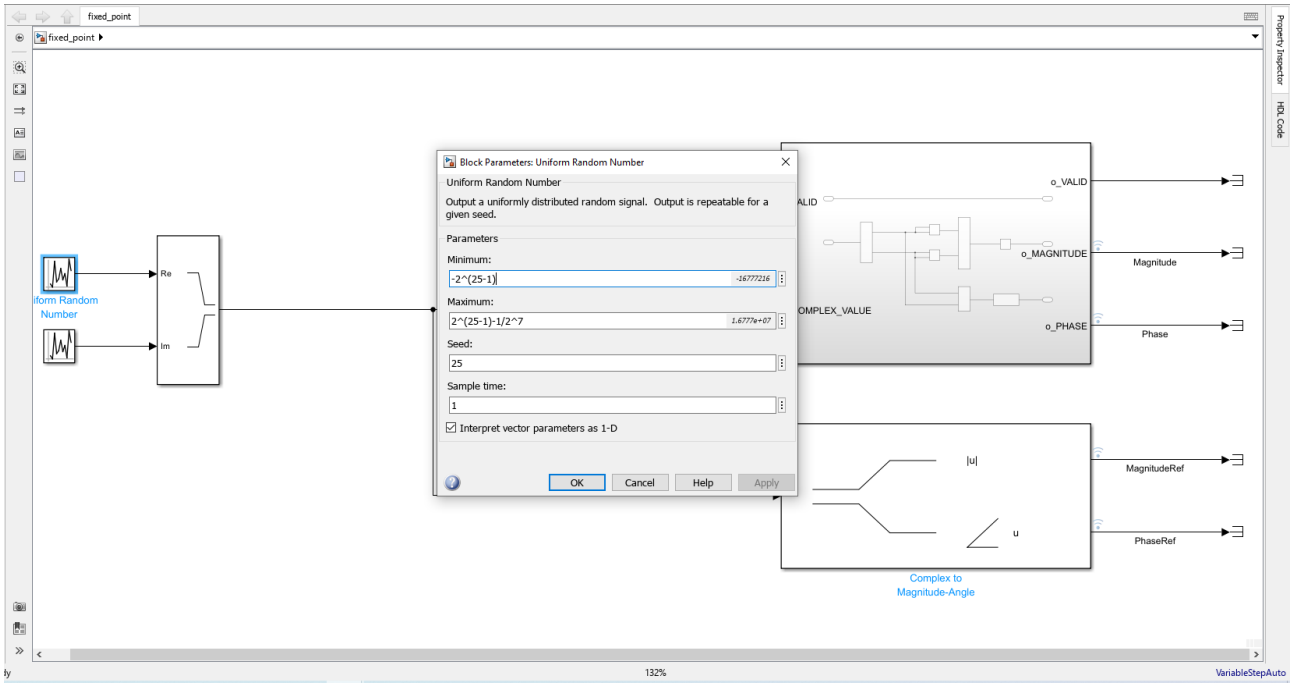

Рис. 3 Налаштування блоків для генерації випадкових чисел з врахуванням допустимого діапазону для знакових чисел

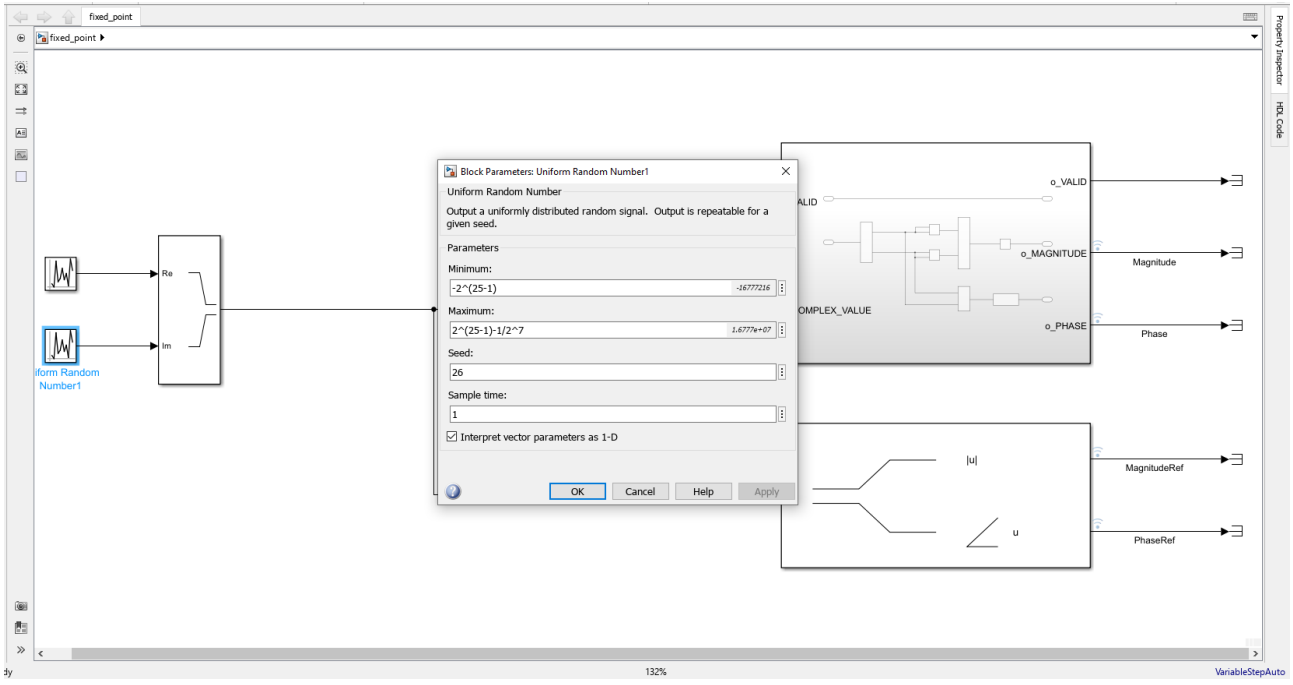

Рис. 4 Налаштування блоків для генерації випадкових чисел з врахуванням допустимого діапазону для знакових чисел

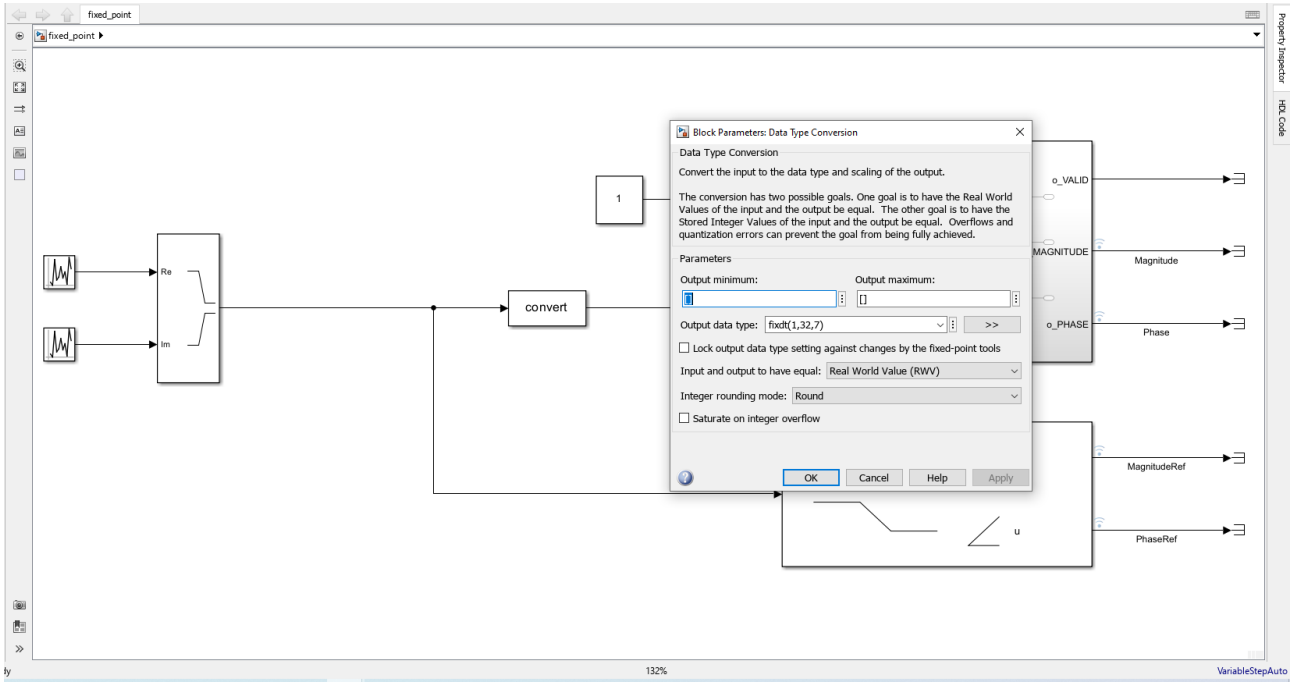

Рис. 5 Налаштування блоку Convert для зведення вхідних чисел в тип Fixed Float

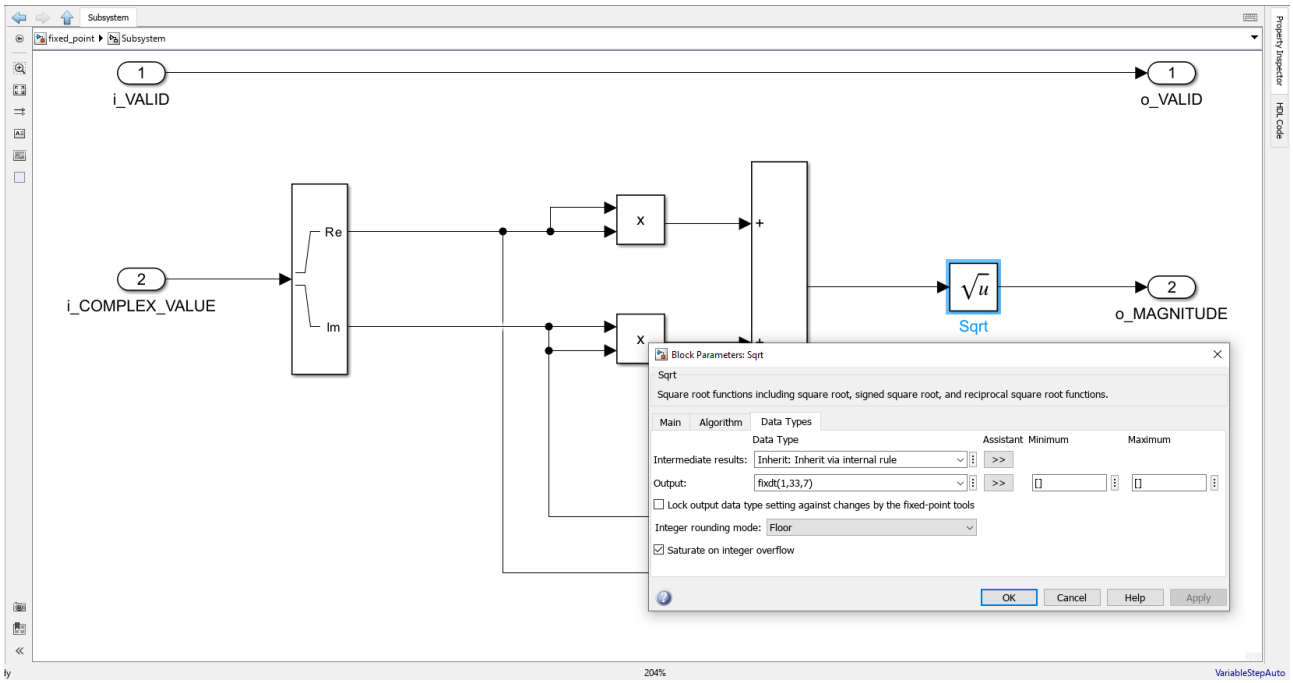

Рис. 6 Налаштування вихідного блоку з результатом обчислення

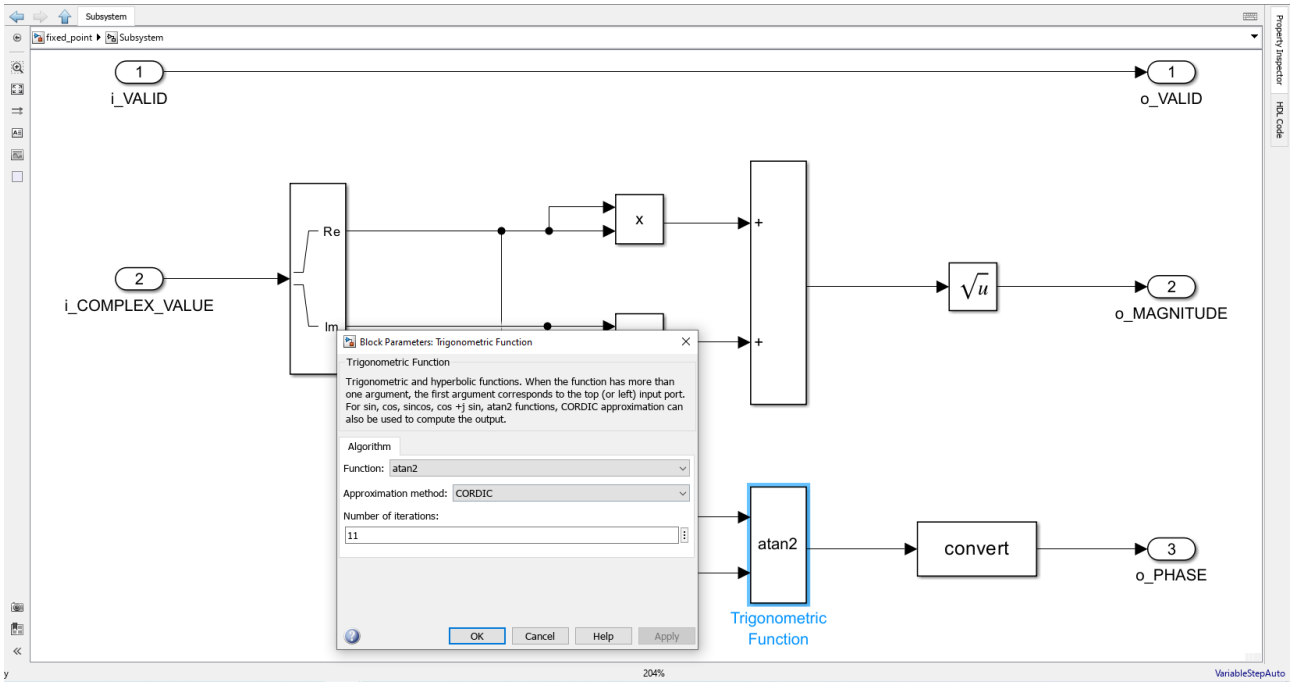

Рис. 7 Налаштування блоку Atan2 з апроксимацією

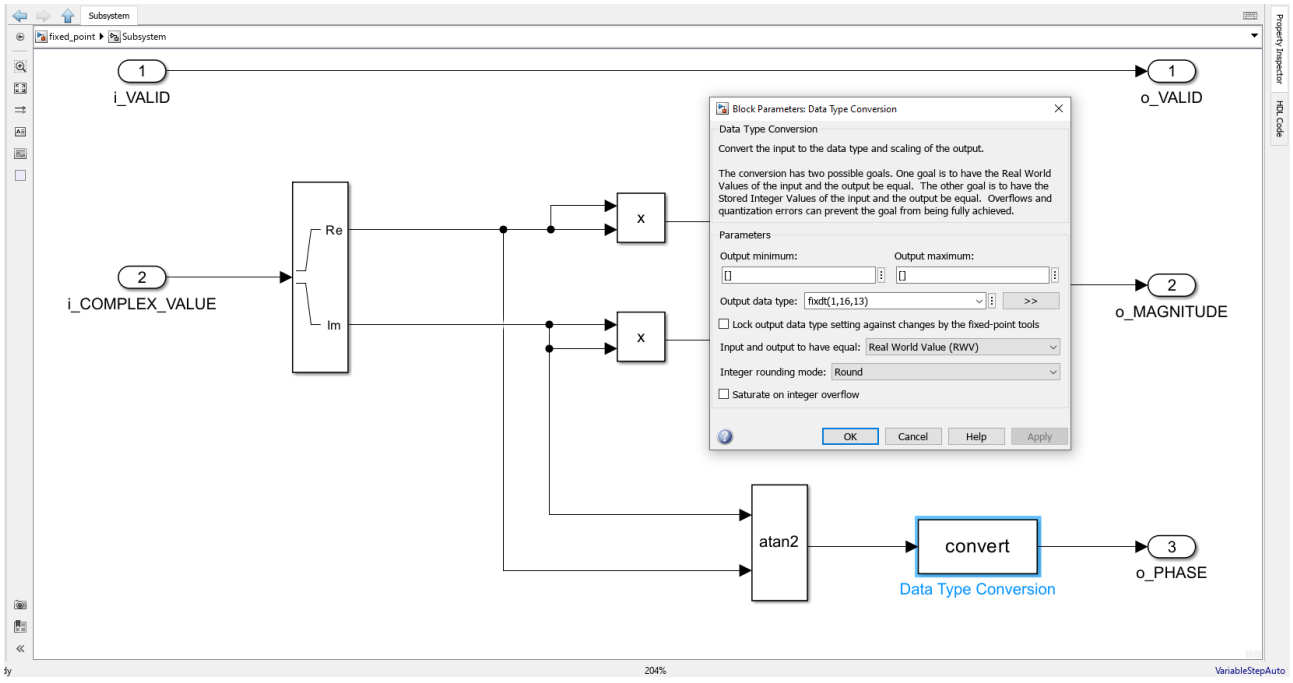

Рис. 8 Налаштування блоку для зведення значення Atan2 в тип Fixed float

Результат моделювання

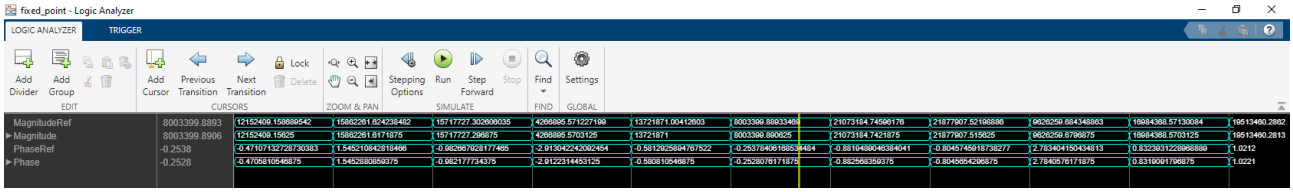

Рис.9 Результат моделювання для чисел з фіксованою комою

Визначення апаратних витрат в Quartus Prime

Flow Summary

<<Filter>>

Flow Status	Successful - Fri Jan 13 06:28:13 2023
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	ap_pr
Top-level Entity Name	Subsystem
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	3,043 / 32,070 (9 %)
Total registers	4066
Total pins	119 / 457 (26 %)
Total virtual pins	0
Total block memory bits	1,116 / 4,065,280 (< 1 %)
Total DSP Blocks	6 / 87 (7 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Рис. 10 Результат компіляції згенерованого HDL файла і визначення апаратних витрат схеми

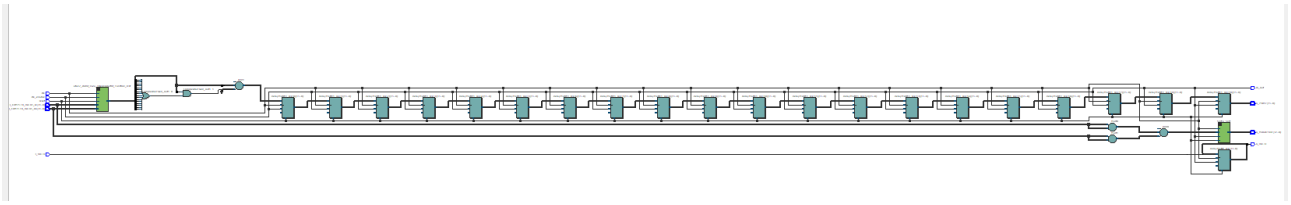

Рис. 11 Синтез згенерованого HDL файла в RTL viewer

Виконання тестового файлу в Modelsim

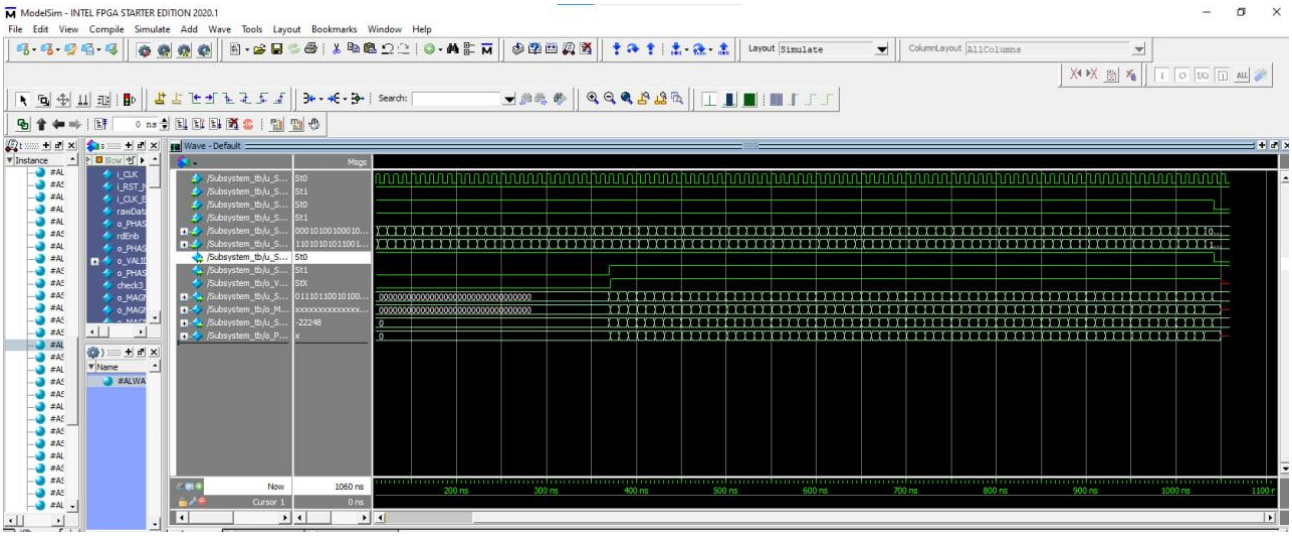

Рис. 12 Загальний вид виконання тестового файлу в Modelsim

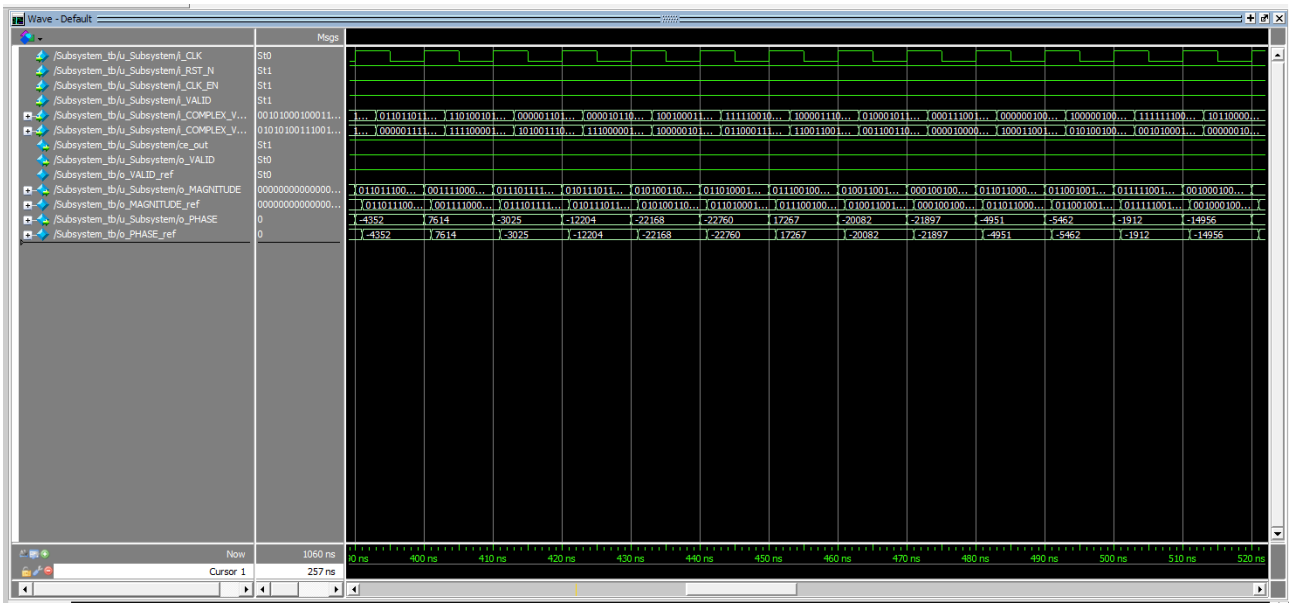

Рис. 13 Детальний вид виконання тестового файлу в Modelsim

Реалізація блок схеми обчислювачів модуля і аргументу комплексного числа для вхідного аргументу з плаваючою комою

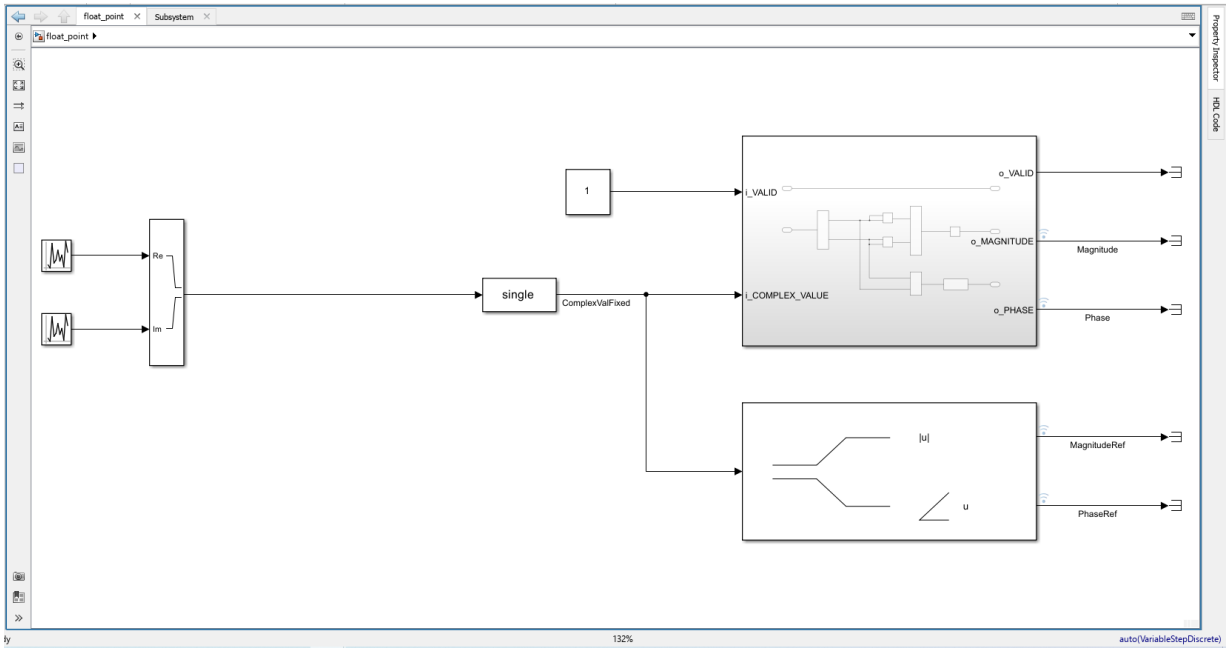

Рис. 14 Загальна блок-схема обчислювача для чисел з плаваючою комою

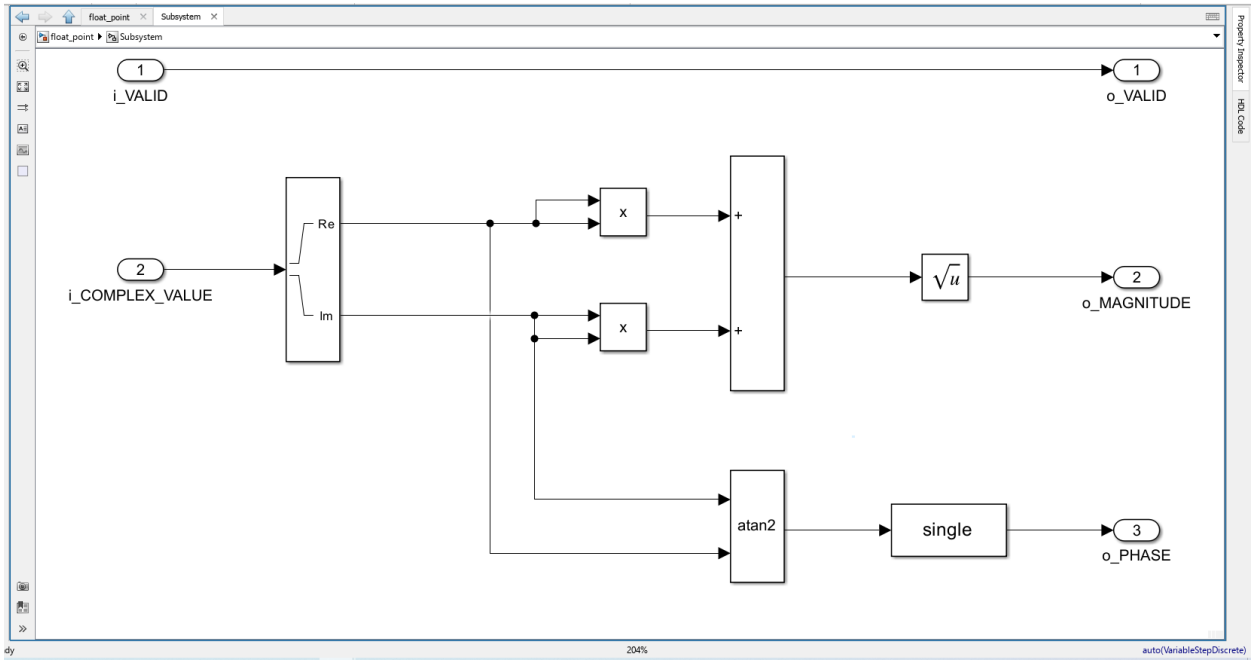

Рис. 15 Підсистема Subsystem

Налаштування блоків

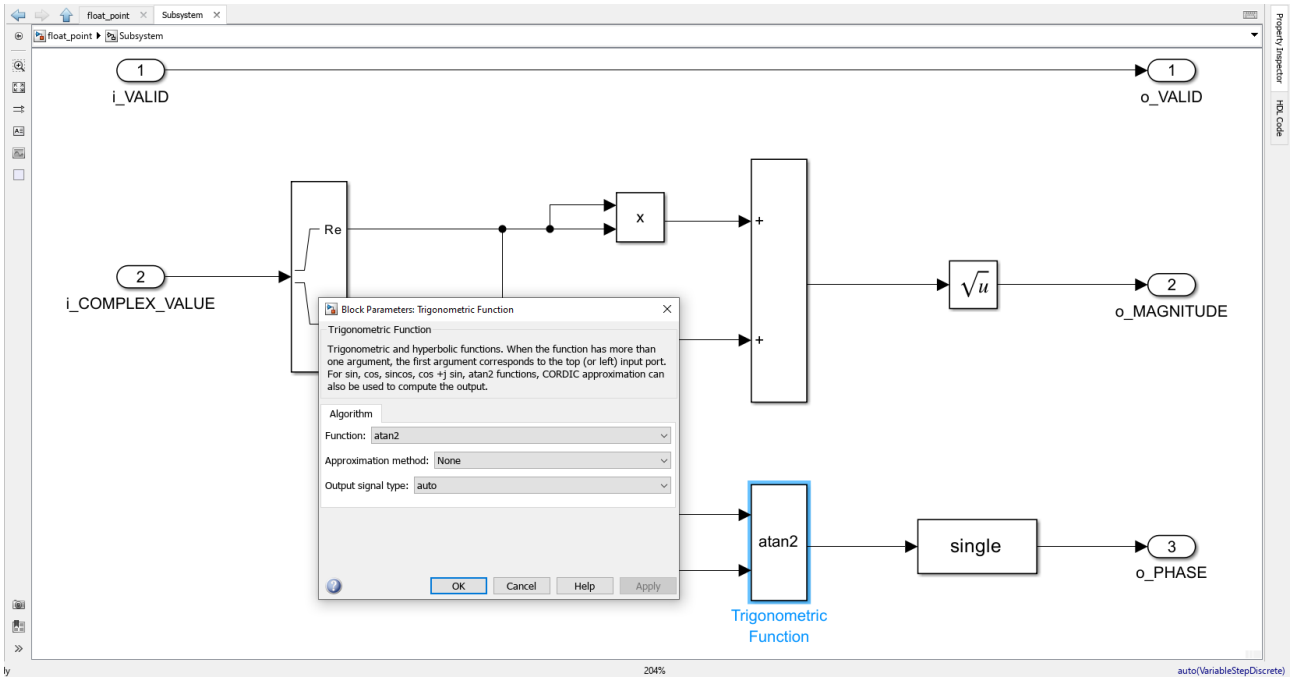

Рис.16 Налаштування блоку Atan2 без апроксимації

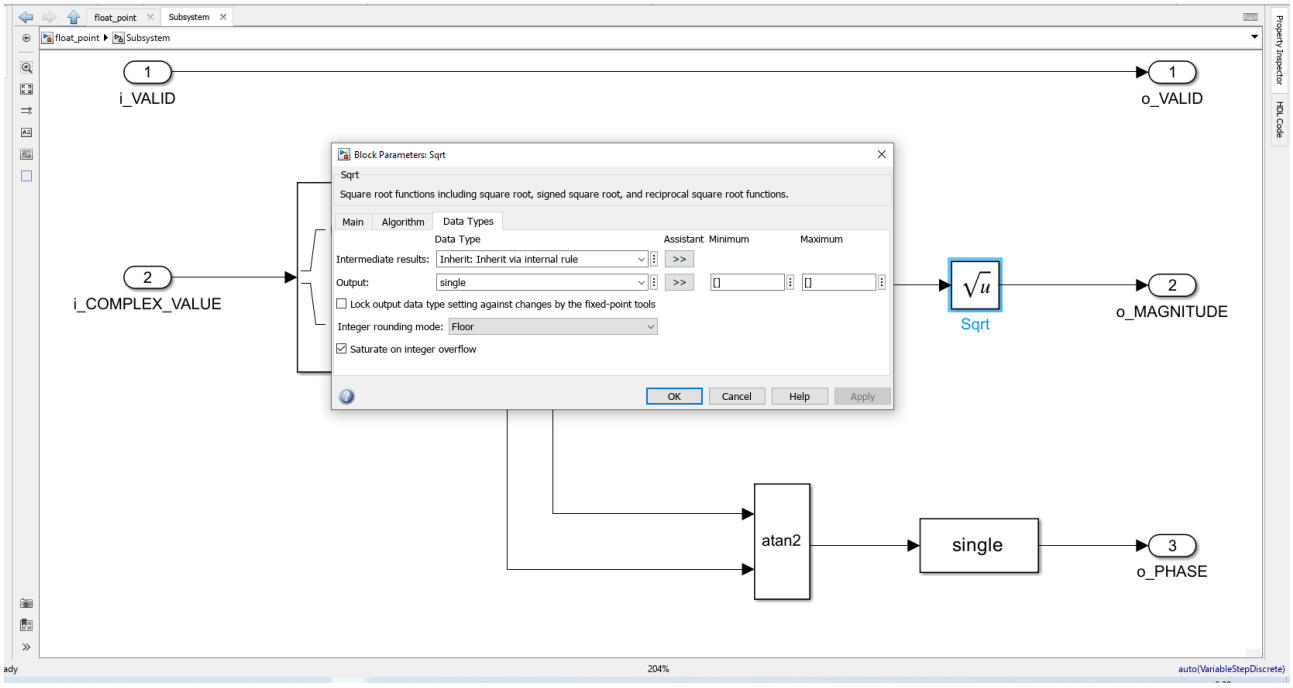

Рис.17 Налаштування вихідного блоку з результатом обчислення

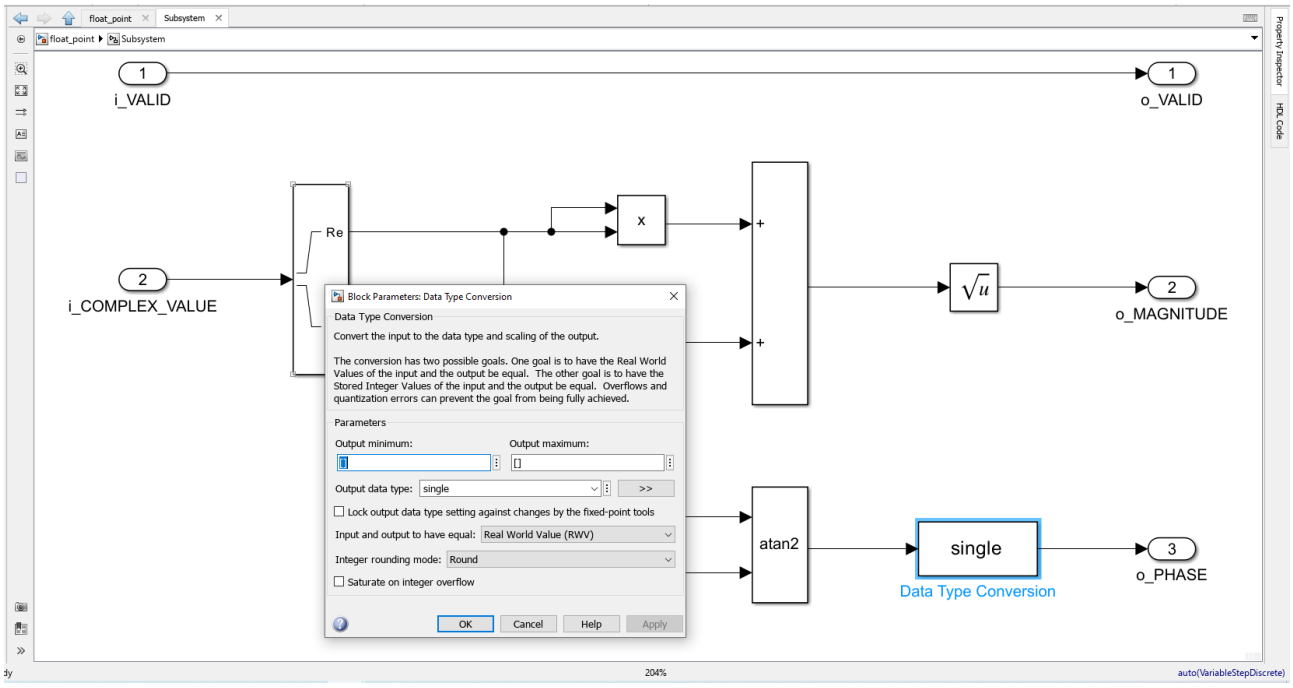

Рис. 18 Налаштування блоку Single, налаштування спільне для всіх блоків, які використовуються в блок-схемі

Результат моделювання

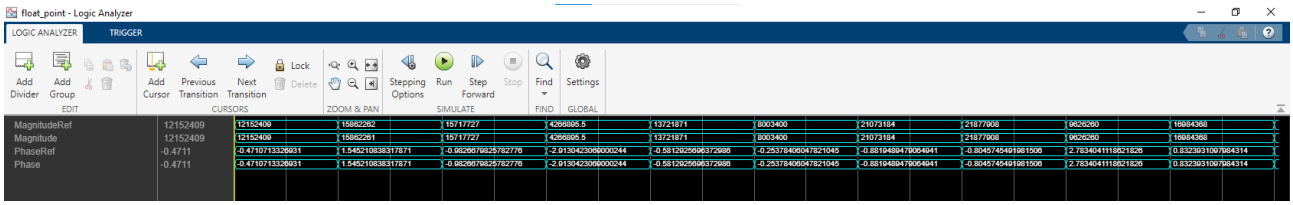

Рис. 19 Результат моделювання для чисел з плаваючою комою

Визначення апаратних витрат в Quartus Prime

Flow Summary

<<Filter>>

Flow Status	Successful - Fri Jan 13 06:51:41 2023
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	ap_pr
Top-level Entity Name	Subsystem
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	2,197 / 32,070 (7 %)
Total registers	4133
Total pins	134 / 457 (29 %)
Total virtual pins	0
Total block memory bits	2,989 / 4,065,280 (< 1 %)
Total DSP Blocks	10 / 87 (11 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Рис. 20 Результат компіляції згенерованого HDL файла і визначення апаратних витрат схеми

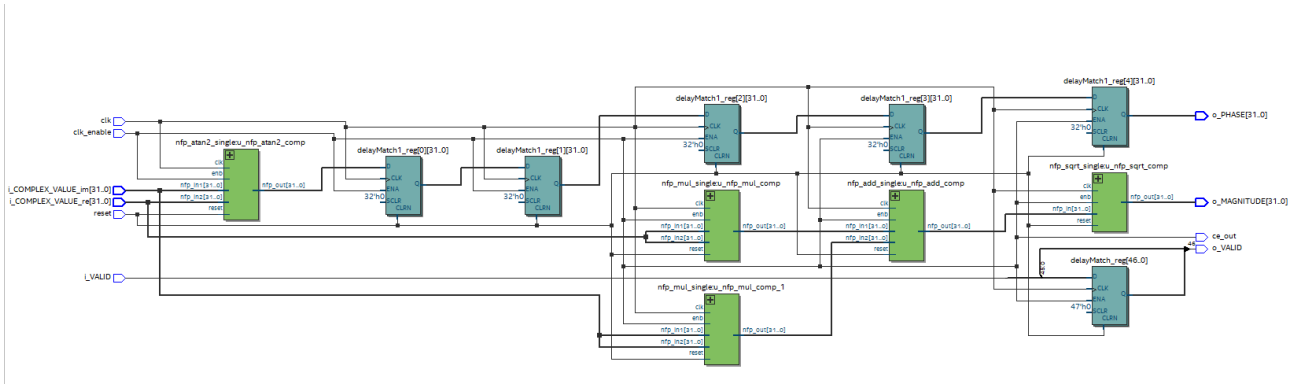

Рис. 21 Синтез згенерованого HDL файла в RTL viewer

Виконання тестового файлу в Modelsim

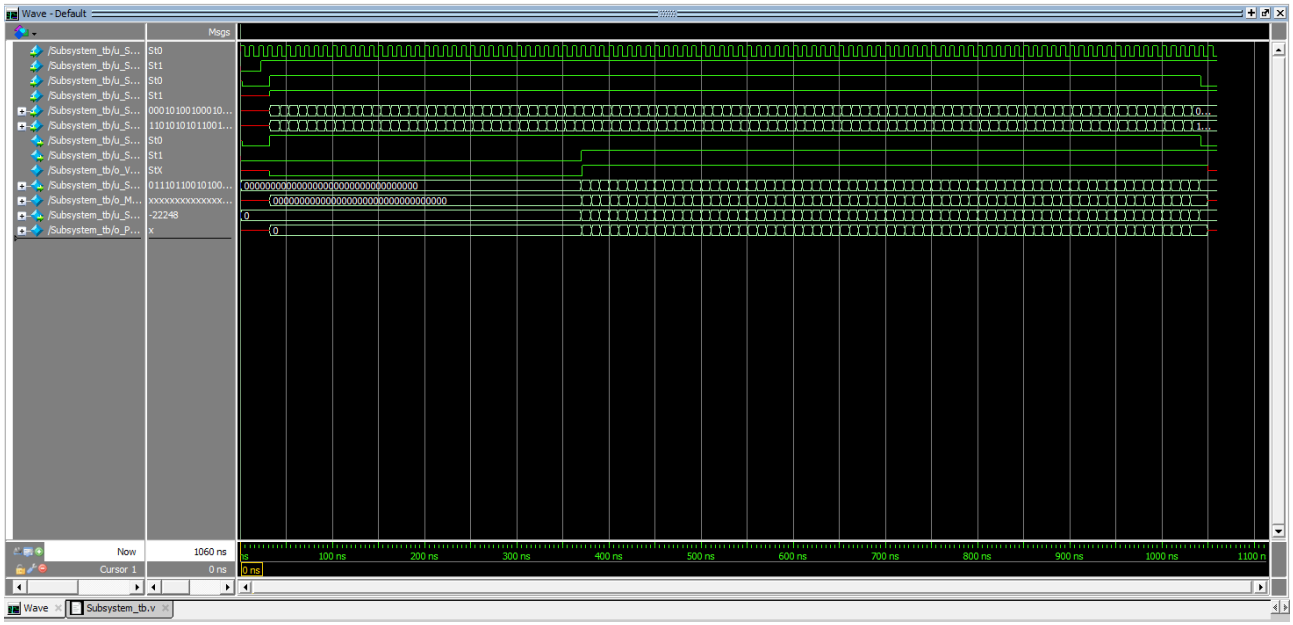

Рис. 22 Загальний вид виконання тестового файлу в Modelsim

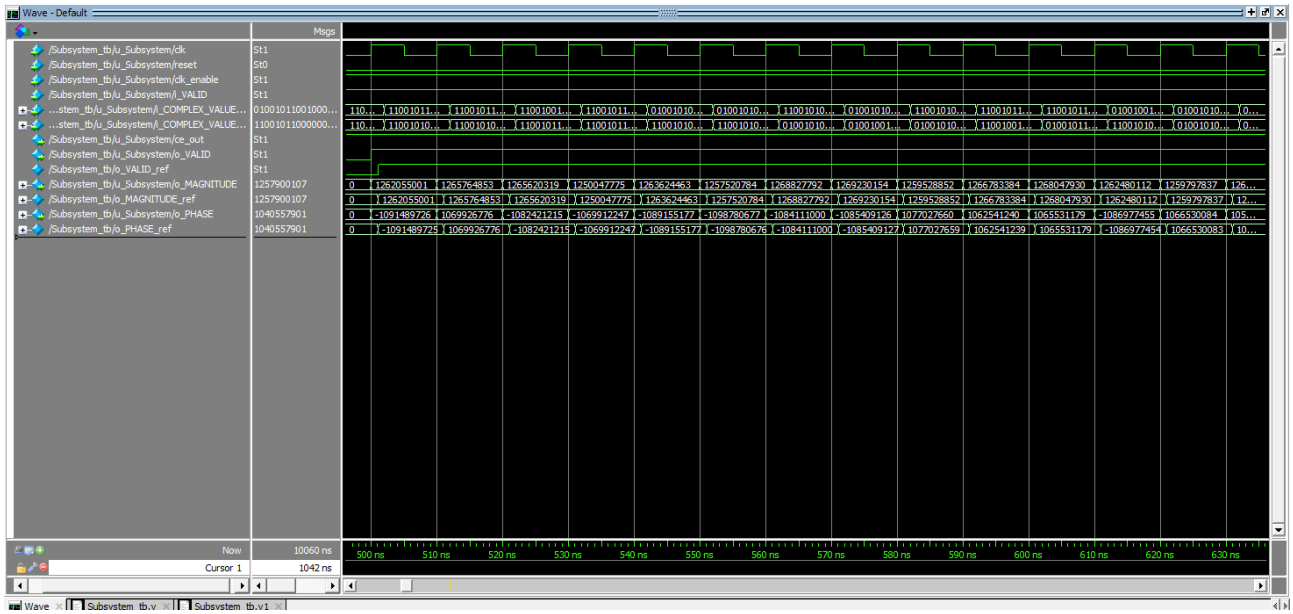

Рис. 23 Детальний вид виконання тестового файлу в Modelsim

Висновок

В ході виконання лабораторної роботи я провів моделювання блок-схем, які обчислюють значення модуль та аргумент вхідних даних для чисел з фіксованою та плаваючою комою. Використовуючи два способа обчислення,

через підсистему та вбудований блок Simulink я отримав еталонні значення модуля та аргумента, а через підсистему наближені.

Першим кроком був розрахунок максимального додатнього значення для генератора випадкових чисел (4). Для цього потрібно було визначити розподіл біт по цілим і дробовим частинам. На цілу частину виділилося 25 біт (1), проте відомо, що якщо представлення знакове, то один старший розряд резервується для інформації по сигнатурі числа, тому на (2) на цілу частину реально відводиться на один біт менше. Також можна помітити, що результат віднімається на одиницю – це потрібно для визначення максимального значення дробової частини, тобто кроку між числами, що прийнято називати квантом чисел. Цей розрахунок (3) показує, що на дробову частину відводиться 7 біт, що дозволя задати задовільну точність, якість точності залежиться від кількості «9» в спадній, по розрядам, послідовності, чим більша буде їх серія підряд, тим точніше можна представити число. Також варто зауважити, що результат (5) є цілим числом. Як правило, для додатніх чисел збільшення розрядності загального числа лиш буде лише наближати до цілого додатнього значення, але ніколи його не досягне, відставання буде на один квант, який визачений на (3), це пов’язано з індексацією числа, для цілого числа починається відлік з 0, а для дробової 1, з таких суджень максимальне додатнє число не можна отримати, а від’ємне можна. Ці висновки були враховані при налаштування блок-схеми для обчислювача з фіксованою та плаваючею комою.

З рис. 9 можна помітити, що числа у форматі чисел з фіксованою комою мають певну розрядність як для цілої частини, так і для дробової. З рис. 19 можна помітити основну відмінність від попереднього формату – числа не можуть одночасно містити багато розрядів як для цілої частини, так і для дробової, для деяких комбінацій можна помітити, що в деяких числах присутня дробова частина, а саме 0.5. З цього можна зробити висновок, що в переважній більшості розподіл динамічного діапазону йде в бік цілої частини і в крайньому випадку виділяється один біт на дробову частину, так як квант для даної серії

чисел – або 1, або 0.5, отже або виділяється 0 біт на дробову частину, або 1 біт. Якщо сильно зменшити діапазон можливих чисел в генераторі випадкових чисел, то в такому випадку більше бітів йтиме на дробову частину, що дозволить детальніше відобразити дробове число.

По рис. 9 та рис. 19 можна зробити висновок, що для обох варіантів значення підсистеми не сильно відрізняються від еталонних, тому дані схеми є прийнятними до користування.

З рис. 10 та рис. 20 можна оцінити апаратні витрати на реалізацію двох схем. Можна виділити те, що реалізація обчислювача з числами у форматі з плаваючею комою потребує більше DSP-блоків та трішки більше потребує пам’яті, на відміну від з фіксованою комою.

Також були згенеровані Verilog файли та тестові файли, для їх відлагодження, такий підхід дозволя зручно моделювати в Matlab/Simulink різні схеми для «важких» обчислень та експортувати у Verilog код, з подальшею можливістю завантажити прошивку у плати FPGA.

Додаток А

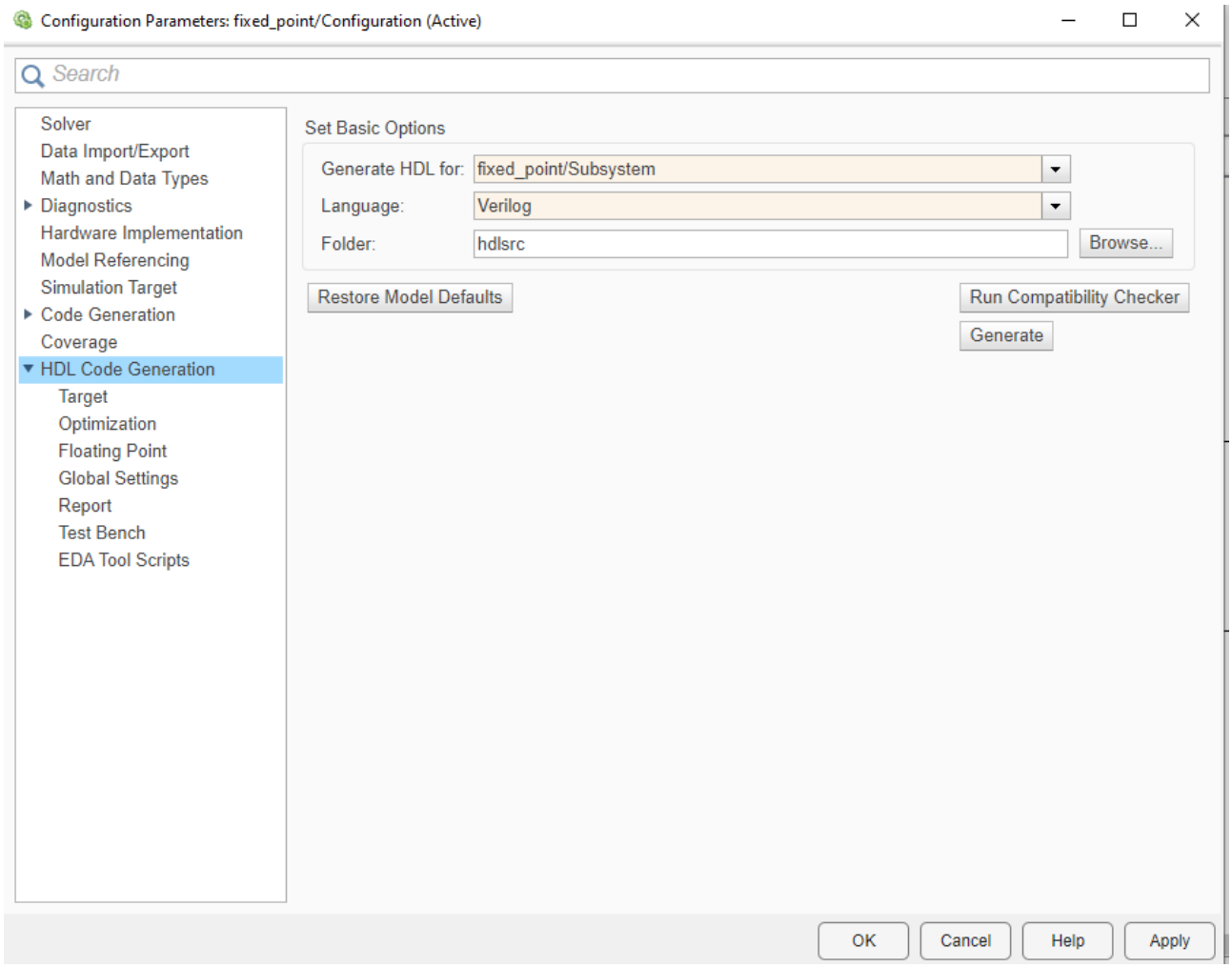

Рис. 24 Налаштування для генерації HDL файлів, аналогічні налаштування для чисел з плаваючою комою

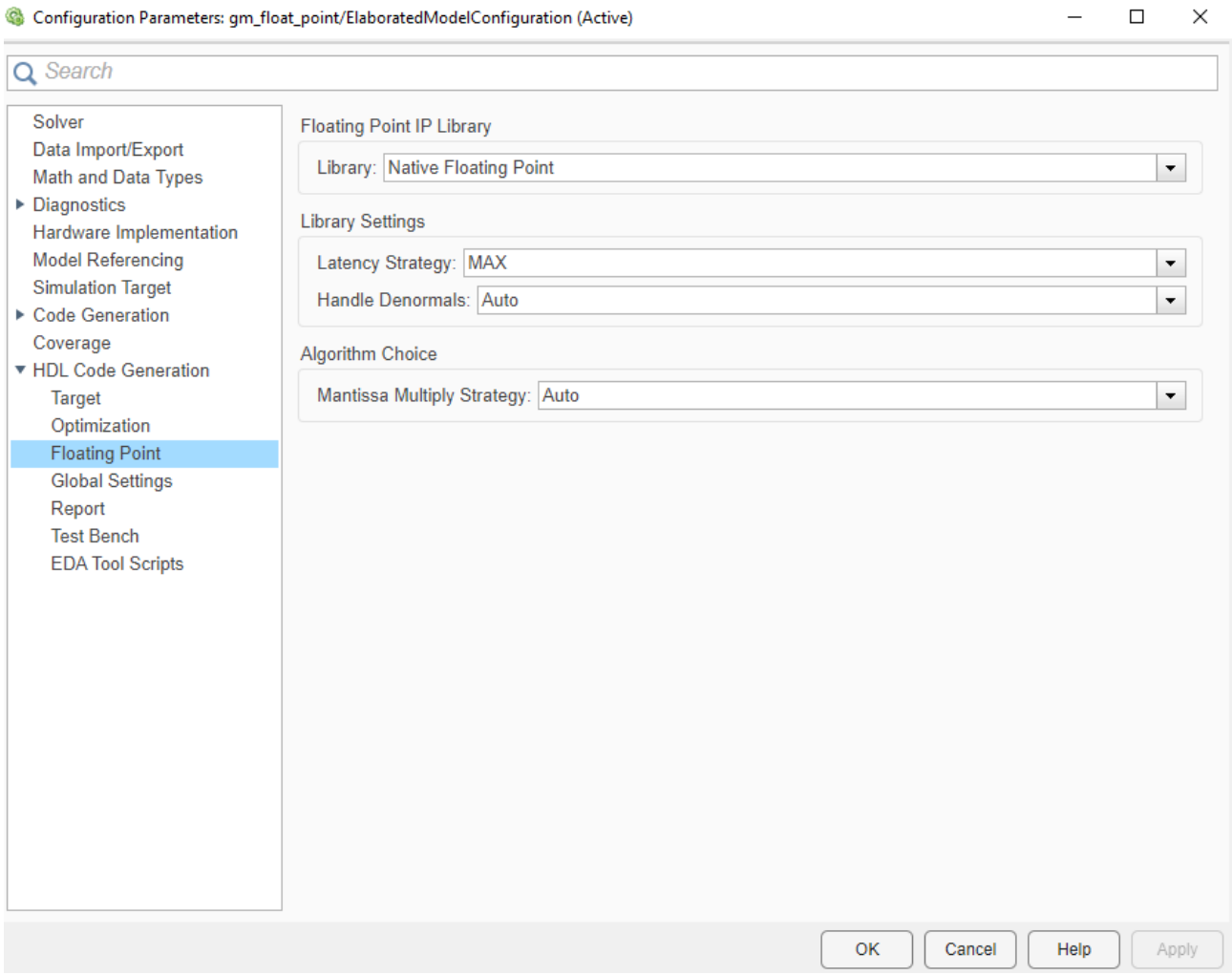

Рис. 25 Налаштування Solver для генерації HDL файлів

Додаток Б

Лістинг 1: Verilog файл Subsystem.v для чисел з фіксованою комою

```
// коментарі прибрав, але вони, як і сам .v файл будуть у директорії, де розміщенні
будуть всі файли проєкту

`timescale 1 ns / 1 ns

module Subsystem

          (clk,

           reset,

           clk_enable,

           i_VALID,

           i_COMPLEX_VALUE_re,

           i_COMPLEX_VALUE_im,

           ce_out,

           o_VALID,

           o_MAGNITUDE,

           o_PHASE);

  input   clk;

  input   reset;

  input   clk_enable;

  input   i_VALID;

  input   signed [31:0] i_COMPLEX_VALUE_re;  // sfix32_En7

  input   signed [31:0] i_COMPLEX_VALUE_im;  // sfix32_En7

  output  ce_out;

  output  o_VALID;

  output  signed [32:0] o_MAGNITUDE;  // sfix33_En7

  output  signed [15:0] o_PHASE;  // sfix16_En13

  wire enb;

  reg  [0:33] delayMatch_reg;  // ufix1 [34]

  wire [0:33] delayMatch_reg_next;  // ufix1 [34]

  wire i_VALID_1;

  wire signed [63:0] Product_out1;  // sfix64_En14

  wire signed [63:0] Product1_out1;  // sfix64_En14

  wire signed [63:0] Add_out1;  // sfix64_En14

  wire signed [32:0] Sqrt_out1;  // sfix33_En7

  wire signed [31:0] Trigonometric_Function_out1;  // sfix32_En29

  wire signed [15:0] Data_Type_Conversion_out1;  // sfix16_En13
```

```
reg signed [15:0] delayMatch1_reg [0:19];  // sfix16 [20]
wire signed [15:0] delayMatch1_reg_next [0:19];  // sfix16_En13 [20]
wire signed [15:0] Data_Type_Conversion_out1_1;  // sfix16_En13
assign enb = clk_enable;
always @(posedge clk or posedge reset)
  begin : delayMatch_process
    if (reset == 1'b1) begin
      delayMatch_reg[0] <= 1'b0;
      delayMatch_reg[1] <= 1'b0;
      delayMatch_reg[2] <= 1'b0;
      delayMatch_reg[3] <= 1'b0;
      delayMatch_reg[4] <= 1'b0;
      delayMatch_reg[5] <= 1'b0;
      delayMatch_reg[6] <= 1'b0;
      delayMatch_reg[7] <= 1'b0;
      delayMatch_reg[8] <= 1'b0;
      delayMatch_reg[9] <= 1'b0;
      delayMatch_reg[10] <= 1'b0;
      delayMatch_reg[11] <= 1'b0;
      delayMatch_reg[12] <= 1'b0;
      delayMatch_reg[13] <= 1'b0;
      delayMatch_reg[14] <= 1'b0;
      delayMatch_reg[15] <= 1'b0;
      delayMatch_reg[16] <= 1'b0;
      delayMatch_reg[17] <= 1'b0;
      delayMatch_reg[18] <= 1'b0;
      delayMatch_reg[19] <= 1'b0;
      delayMatch_reg[20] <= 1'b0;
      delayMatch_reg[21] <= 1'b0;
      delayMatch_reg[22] <= 1'b0;
      delayMatch_reg[23] <= 1'b0;
      delayMatch_reg[24] <= 1'b0;
      delayMatch_reg[25] <= 1'b0;
      delayMatch_reg[26] <= 1'b0;
      delayMatch_reg[27] <= 1'b0;
      delayMatch_reg[28] <= 1'b0;
      delayMatch_reg[29] <= 1'b0;
```

```
        delayMatch_reg[30] <= 1'b0;
        delayMatch_reg[31] <= 1'b0;
        delayMatch_reg[32] <= 1'b0;
        delayMatch_reg[33] <= 1'b0;
      end
      else begin
        if (enb) begin
          delayMatch_reg[0] <= delayMatch_reg_next[0];
          delayMatch_reg[1] <= delayMatch_reg_next[1];
          delayMatch_reg[2] <= delayMatch_reg_next[2];
          delayMatch_reg[3] <= delayMatch_reg_next[3];
          delayMatch_reg[4] <= delayMatch_reg_next[4];
          delayMatch_reg[5] <= delayMatch_reg_next[5];
          delayMatch_reg[6] <= delayMatch_reg_next[6];
          delayMatch_reg[7] <= delayMatch_reg_next[7];
          delayMatch_reg[8] <= delayMatch_reg_next[8];
          delayMatch_reg[9] <= delayMatch_reg_next[9];
          delayMatch_reg[10] <= delayMatch_reg_next[10];
          delayMatch_reg[11] <= delayMatch_reg_next[11];
          delayMatch_reg[12] <= delayMatch_reg_next[12];
          delayMatch_reg[13] <= delayMatch_reg_next[13];
          delayMatch_reg[14] <= delayMatch_reg_next[14];
          delayMatch_reg[15] <= delayMatch_reg_next[15];
          delayMatch_reg[16] <= delayMatch_reg_next[16];
          delayMatch_reg[17] <= delayMatch_reg_next[17];
          delayMatch_reg[18] <= delayMatch_reg_next[18];
          delayMatch_reg[19] <= delayMatch_reg_next[19];
          delayMatch_reg[20] <= delayMatch_reg_next[20];
          delayMatch_reg[21] <= delayMatch_reg_next[21];
          delayMatch_reg[22] <= delayMatch_reg_next[22];
          delayMatch_reg[23] <= delayMatch_reg_next[23];
          delayMatch_reg[24] <= delayMatch_reg_next[24];
          delayMatch_reg[25] <= delayMatch_reg_next[25];
          delayMatch_reg[26] <= delayMatch_reg_next[26];
          delayMatch_reg[27] <= delayMatch_reg_next[27];
          delayMatch_reg[28] <= delayMatch_reg_next[28];
          delayMatch_reg[29] <= delayMatch_reg_next[29];
```

```
        delayMatch_reg[30] <= delayMatch_reg_next[30];
        delayMatch_reg[31] <= delayMatch_reg_next[31];
        delayMatch_reg[32] <= delayMatch_reg_next[32];
        delayMatch_reg[33] <= delayMatch_reg_next[33];
      end
    end
  end
assign i_VALID_1 = delayMatch_reg[33];
assign delayMatch_reg_next[0] = i_VALID;
assign delayMatch_reg_next[1] = delayMatch_reg[0];
assign delayMatch_reg_next[2] = delayMatch_reg[1];
assign delayMatch_reg_next[3] = delayMatch_reg[2];
assign delayMatch_reg_next[4] = delayMatch_reg[3];
assign delayMatch_reg_next[5] = delayMatch_reg[4];
assign delayMatch_reg_next[6] = delayMatch_reg[5];
assign delayMatch_reg_next[7] = delayMatch_reg[6];
assign delayMatch_reg_next[8] = delayMatch_reg[7];
assign delayMatch_reg_next[9] = delayMatch_reg[8];
assign delayMatch_reg_next[10] = delayMatch_reg[9];
assign delayMatch_reg_next[11] = delayMatch_reg[10];
assign delayMatch_reg_next[12] = delayMatch_reg[11];
assign delayMatch_reg_next[13] = delayMatch_reg[12];
assign delayMatch_reg_next[14] = delayMatch_reg[13];
assign delayMatch_reg_next[15] = delayMatch_reg[14];
assign delayMatch_reg_next[16] = delayMatch_reg[15];
assign delayMatch_reg_next[17] = delayMatch_reg[16];
assign delayMatch_reg_next[18] = delayMatch_reg[17];
assign delayMatch_reg_next[19] = delayMatch_reg[18];
assign delayMatch_reg_next[20] = delayMatch_reg[19];
assign delayMatch_reg_next[21] = delayMatch_reg[20];
assign delayMatch_reg_next[22] = delayMatch_reg[21];
assign delayMatch_reg_next[23] = delayMatch_reg[22];
assign delayMatch_reg_next[24] = delayMatch_reg[23];
assign delayMatch_reg_next[25] = delayMatch_reg[24];
assign delayMatch_reg_next[26] = delayMatch_reg[25];
assign delayMatch_reg_next[27] = delayMatch_reg[26];
assign delayMatch_reg_next[28] = delayMatch_reg[27];
```

```
  assign delayMatch_reg_next[29] = delayMatch_reg[28];
  assign delayMatch_reg_next[30] = delayMatch_reg[29];
  assign delayMatch_reg_next[31] = delayMatch_reg[30];
  assign delayMatch_reg_next[32] = delayMatch_reg[31];
  assign delayMatch_reg_next[33] = delayMatch_reg[32];




  assign o_VALID = i_VALID_1;
  assign Product_out1 = i_COMPLEX_VALUE_re * i_COMPLEX_VALUE_re;
  assign Product1_out1 = i_COMPLEX_VALUE_im * i_COMPLEX_VALUE_im;
  assign Add_out1 = Product_out1 + Product1_out1;
  Sqrt u_Sqrt (.clk(clk),
               .reset(reset),
               .enb(clk_enable),
               .din(Add_out1),  // sfix64_En14
               .dout(Sqrt_out1)  // sfix33_En7
               );
  assign o_MAGNITUDE = Sqrt_out1;
  atan2_cordic_nw u_Trigonometric_Function_inst (.clk(clk),
                                                 .reset(reset),
                                                 .enb(clk_enable),
                                                 .y_in(i_COMPLEX_VALUE_im),  //
sfix32_En7
                                                 .x_in(i_COMPLEX_VALUE_re),  //
sfix32_En7
                                                 .angle(Trigonometric_Function_out1)
// sfix32_En29
                                                 );
  assign Data_Type_Conversion_out1 = Trigonometric_Function_out1[31:16] +
$signed({1'b0, Trigonometric_Function_out1[15] & (( ~ Trigonometric_Function_out1[31])
| (|Trigonometric_Function_out1[14:0]))});
  always @(posedge clk or posedge reset)
    begin : delayMatch1_process
      if (reset == 1'b1) begin
        delayMatch1_reg[0] <= 16'sb0000000000000000;
        delayMatch1_reg[1] <= 16'sb0000000000000000;
        delayMatch1_reg[2] <= 16'sb0000000000000000;
        delayMatch1_reg[3] <= 16'sb0000000000000000;
```

```
        delayMatch1_reg[4] <= 16'sb0000000000000000;
        delayMatch1_reg[5] <= 16'sb0000000000000000;
        delayMatch1_reg[6] <= 16'sb0000000000000000;
        delayMatch1_reg[7] <= 16'sb0000000000000000;
        delayMatch1_reg[8] <= 16'sb0000000000000000;
        delayMatch1_reg[9] <= 16'sb0000000000000000;
        delayMatch1_reg[10] <= 16'sb0000000000000000;
        delayMatch1_reg[11] <= 16'sb0000000000000000;
        delayMatch1_reg[12] <= 16'sb0000000000000000;
        delayMatch1_reg[13] <= 16'sb0000000000000000;
        delayMatch1_reg[14] <= 16'sb0000000000000000;
        delayMatch1_reg[15] <= 16'sb0000000000000000;
        delayMatch1_reg[16] <= 16'sb0000000000000000;
        delayMatch1_reg[17] <= 16'sb0000000000000000;
        delayMatch1_reg[18] <= 16'sb0000000000000000;
        delayMatch1_reg[19] <= 16'sb0000000000000000;
      end
      else begin
        if (enb) begin
          delayMatch1_reg[0] <= delayMatch1_reg_next[0];
          delayMatch1_reg[1] <= delayMatch1_reg_next[1];
          delayMatch1_reg[2] <= delayMatch1_reg_next[2];
          delayMatch1_reg[3] <= delayMatch1_reg_next[3];
          delayMatch1_reg[4] <= delayMatch1_reg_next[4];
          delayMatch1_reg[5] <= delayMatch1_reg_next[5];
          delayMatch1_reg[6] <= delayMatch1_reg_next[6];
          delayMatch1_reg[7] <= delayMatch1_reg_next[7];
          delayMatch1_reg[8] <= delayMatch1_reg_next[8];
          delayMatch1_reg[9] <= delayMatch1_reg_next[9];
          delayMatch1_reg[10] <= delayMatch1_reg_next[10];
          delayMatch1_reg[11] <= delayMatch1_reg_next[11];
          delayMatch1_reg[12] <= delayMatch1_reg_next[12];
          delayMatch1_reg[13] <= delayMatch1_reg_next[13];
          delayMatch1_reg[14] <= delayMatch1_reg_next[14];
          delayMatch1_reg[15] <= delayMatch1_reg_next[15];
          delayMatch1_reg[16] <= delayMatch1_reg_next[16];
          delayMatch1_reg[17] <= delayMatch1_reg_next[17];
```

```
          delayMatch1_reg[18] <= delayMatch1_reg_next[18];
          delayMatch1_reg[19] <= delayMatch1_reg_next[19];
        end
      end
    end
  assign Data_Type_Conversion_out1_1 = delayMatch1_reg[19];
  assign delayMatch1_reg_next[0] = Data_Type_Conversion_out1;
  assign delayMatch1_reg_next[1] = delayMatch1_reg[0];
  assign delayMatch1_reg_next[2] = delayMatch1_reg[1];
  assign delayMatch1_reg_next[3] = delayMatch1_reg[2];
  assign delayMatch1_reg_next[4] = delayMatch1_reg[3];
  assign delayMatch1_reg_next[5] = delayMatch1_reg[4];
  assign delayMatch1_reg_next[6] = delayMatch1_reg[5];
  assign delayMatch1_reg_next[7] = delayMatch1_reg[6];
  assign delayMatch1_reg_next[8] = delayMatch1_reg[7];
  assign delayMatch1_reg_next[9] = delayMatch1_reg[8];
  assign delayMatch1_reg_next[10] = delayMatch1_reg[9];
  assign delayMatch1_reg_next[11] = delayMatch1_reg[10];
  assign delayMatch1_reg_next[12] = delayMatch1_reg[11];
  assign delayMatch1_reg_next[13] = delayMatch1_reg[12];
  assign delayMatch1_reg_next[14] = delayMatch1_reg[13];
  assign delayMatch1_reg_next[15] = delayMatch1_reg[14];
  assign delayMatch1_reg_next[16] = delayMatch1_reg[15];
  assign delayMatch1_reg_next[17] = delayMatch1_reg[16];
  assign delayMatch1_reg_next[18] = delayMatch1_reg[17];
  assign delayMatch1_reg_next[19] = delayMatch1_reg[18];
  assign o_PHASE = Data_Type_Conversion_out1_1;
  assign ce_out = clk_enable;
endmodule  // Subsystem
```

Лістинг 2: Тестовий файл Subsystem_tb.v для чисел з фіксованою комою

```
// Коментарі прибрані
`timescale 1 ns / 1 ns
module Subsystem_tb;
  reg  clk;
  reg  reset;
  wire clk_enable;
```

```
wire rawData_i_VALID;
wire o_PHASE_done;  // ufix1
wire rdEnb;
wire o_PHASE_done_enb;  // ufix1
reg [9:0] o_VALID_addr;  // ufix10
wire o_PHASE_lastAddr;  // ufix1
wire resetn;
reg  check3_done;  // ufix1
wire o_MAGNITUDE_done;  // ufix1
wire o_MAGNITUDE_done_enb;  // ufix1
wire o_MAGNITUDE_lastAddr;  // ufix1
reg  check2_done;  // ufix1
wire o_VALID_done;  // ufix1
wire o_VALID_done_enb;  // ufix1
wire o_VALID_active;  // ufix1
wire [9:0] Data_Type_Conversion_out1_addr_delay;  // ufix10
reg  tb_enb_delay;
reg signed [31:0] fp_i_COMPLEX_VALUE_im;  // sfix32
reg signed [31:0] rawData_i_COMPLEX_VALUE_im;  // sfix32_En7
reg signed [31:0] status_i_COMPLEX_VALUE_im;  // sfix32
reg signed [31:0] holdData_i_COMPLEX_VALUE_im;  // sfix32_En7
reg signed [31:0] i_COMPLEX_VALUE_im_offset;  // sfix32_En7
wire signed [31:0] i_COMPLEX_VALUE_im;  // sfix32_En7
reg [9:0] Constant_out1_addr;  // ufix10
wire Constant_out1_active;  // ufix1
wire Constant_out1_enb;  // ufix1
reg signed [31:0] fp_i_COMPLEX_VALUE_re;  // sfix32
reg signed [31:0] rawData_i_COMPLEX_VALUE_re;  // sfix32_En7
reg signed [31:0] status_i_COMPLEX_VALUE_re;  // sfix32
reg signed [31:0] holdData_i_COMPLEX_VALUE_re;  // sfix32_En7
reg signed [31:0] i_COMPLEX_VALUE_re_offset;  // sfix32_En7
wire signed [31:0] i_COMPLEX_VALUE_re;  // sfix32_En7
reg  holdData_i_VALID;
reg  i_VALID_offset;
wire i_VALID_1;
wire snkDone;
wire snkDonen;
```

```
wire tb_enb;
wire ce_out;
wire o_VALID;
wire signed [32:0] o_MAGNITUDE;  // sfix33_En7
wire signed [15:0] o_PHASE;  // sfix16_En13
wire o_VALID_enb;  // ufix1
wire o_VALID_lastAddr;  // ufix1
reg  check1_done;  // ufix1
reg [5:0] o_VALID_chkcnt;  // ufix6
wire o_VALID_ignCntDone;  // ufix1
wire o_VALID_needToCount;  // ufix1
wire o_VALID_chkenb;  // ufix1
wire o_VALID_chkdata;  // ufix1
wire [9:0] o_VALID_addr_delay_1;  // ufix10
reg signed [31:0] fp_o_VALID_expected;  // sfix32
reg  o_VALID_expected;
reg signed [31:0] status_o_VALID_expected;  // sfix32
wire o_VALID_ref;
reg  o_VALID_testFailure;  // ufix1
reg [5:0] o_MAGNITUDE_chkcnt;  // ufix6
wire o_MAGNITUDE_ignCntDone;  // ufix1
wire o_MAGNITUDE_needToCount;  // ufix1
wire o_MAGNITUDE_chkenb;  // ufix1
wire o_MAGNITUDE_chkdata;  // ufix1
wire [9:0] o_MAGNITUDE_addr_delay_1;  // ufix10
reg signed [31:0] fp_o_MAGNITUDE_expected;  // sfix32
reg signed [32:0] o_MAGNITUDE_expected;  // sfix33_En7
reg signed [31:0] status_o_MAGNITUDE_expected;  // sfix32
wire signed [32:0] o_MAGNITUDE_ref;  // sfix33_En7
reg  o_MAGNITUDE_testFailure;  // ufix1
reg [5:0] o_PHASE_chkcnt;  // ufix6
wire o_PHASE_ignCntDone;  // ufix1
wire o_PHASE_needToCount;  // ufix1
wire o_PHASE_chkenb;  // ufix1
wire o_PHASE_chkdata;  // ufix1
wire [9:0] o_PHASE_addr_delay_1;  // ufix10
reg signed [31:0] fp_o_PHASE_expected;  // sfix32
```

```
reg signed [15:0] o_PHASE_expected;  // sfix16_En13
reg signed [31:0] status_o_PHASE_expected;  // sfix32
wire signed [15:0] o_PHASE_ref;  // sfix16_En13
reg  o_PHASE_testFailure;  // ufix1
wire testFailure;  // ufix1
function real absReal(input real num);
begin
  if (num < 0)
    absReal = -num;
  else
    absReal = num;
end
endfunction
function real floatHalfToReal;
input [15:0] x;
reg [63:0] conv;
begin
  conv[63] = x[15]; // sign
  if (x[14:10] == 5'b0) // exp
    conv[62:52] = 11'b0;
  else
    conv[62:52] = 1023 + (x[14:10] - 15);
  conv[51:42] = x[9:0]; // mantissa
  conv[41:0] = 42'b0;
  if (((x[14:10] == 5'h1F) && (x[9:0] != 10'h0))) // check for NaN
  begin
    conv[63] = 1'b0;
    conv[62:52] = 11'h7FF;
    conv[51:0] = 52'h0;
  end
  floatHalfToReal = $bitstoreal(conv);
end
endfunction
function real floatSingleToReal;
input [31:0] x;
reg [63:0] conv;
begin
```

```
  conv[63] = x[31]; // sign
  if (x[30:23] == 8'b0) // exp
    conv[62:52] = 11'b0;
  else
    conv[62:52] = 1023 + (x[30:23] - 127);
  conv[51:29] = x[22:0]; // mantissa
  conv[28:0] = 29'b0;
  if (((x[30:23] == 8'hFF) && (x[22:0] != 23'h0))) // check for NaN
  begin
    conv[63] = 1'b0;
    conv[62:52] = 11'h7FF;
    conv[51:0] = 52'h0;
  end
  floatSingleToReal = $bitstoreal(conv);
end
endfunction
function real floatDoubleToReal;
input [63:0] x;
reg [63:0] conv;
begin
  conv[63:0] = x[63:0];
  if (((x[62:52] == 11'h7FF) && (x[51:0] != 52'h0))) // check for NaN
  begin
    conv[63] = 1'b0;
    conv[62:52] = 11'h7FF;
    conv[51:0] = 52'h0;
  end
  floatDoubleToReal = $bitstoreal(conv);
end
endfunction
function isFloatEpsEqual(input real a, input real b, input real eps);
real absdiff;

begin
  absdiff = absReal(a - b);
  if (absdiff < eps) // absolute error check
    isFloatEpsEqual = 1;
```

```
  else if (a == b) // check infinities
    isFloatEpsEqual = 1;
  else if (a*b == 0.0) // either is zero
    isFloatEpsEqual = (absdiff < eps);
  else if (absReal(a) < absReal(b)) // relative error check
    isFloatEpsEqual = absdiff/absReal(b) < eps;
  else
    isFloatEpsEqual = absdiff/absReal(a) < eps;
end
endfunction
function isFloatHalfEpsEqual;
input [15:0] x;
input [15:0] y;
input real eps;
real a, b;
real absdiff;
begin
  a = floatHalfToReal(x);
  b = floatHalfToReal(y);
  isFloatHalfEpsEqual = isFloatEpsEqual(a, b, eps);
end
endfunction
function isFloatSingleEpsEqual;
input [31:0] x;
input [31:0] y;
input real eps;
real a, b;
real absdiff;
begin
  a = floatSingleToReal(x);
  b = floatSingleToReal(y);
  isFloatSingleEpsEqual = isFloatEpsEqual(a, b, eps);
end
endfunction
function isFloatDoubleEpsEqual;
input [63:0] x;
input [63:0] y;
```

```
input real eps;
real a, b;
real absdiff;

begin
  a = floatDoubleToReal(x);
  b = floatDoubleToReal(y);
  isFloatDoubleEpsEqual = isFloatEpsEqual(a, b, eps);
end
endfunction
// Data source for i_VALID
assign rawData_i_VALID = 1'b1;
assign o_PHASE_done_enb = o_PHASE_done & rdEnb;
assign o_PHASE_lastAddr = o_VALID_addr >= 10'b1111101000;
assign o_PHASE_done = o_PHASE_lastAddr & resetn;
// Delay to allow last sim cycle to complete
always @(posedge clk or posedge reset)
  begin : checkDone_3
    if (reset) begin
      check3_done <= 0;
    end
    else begin
      if (o_PHASE_done_enb) begin
        check3_done <= o_PHASE_done;
      end
    end
  end
assign o_MAGNITUDE_done_enb = o_MAGNITUDE_done & rdEnb;
assign o_MAGNITUDE_lastAddr = o_VALID_addr >= 10'b1111101000;
assign o_MAGNITUDE_done = o_MAGNITUDE_lastAddr & resetn;
// Delay to allow last sim cycle to complete
always @(posedge clk or posedge reset)
  begin : checkDone_2
    if (reset) begin
      check2_done <= 0;
    end
    else begin
```

```
        if (o_MAGNITUDE_done_enb) begin
          check2_done <= o_MAGNITUDE_done;
        end
      end
    end
  assign o_VALID_done_enb = o_VALID_done & rdEnb;
  assign o_VALID_active = o_VALID_addr != 10'b1111101000;
  // Data source for i_COMPLEX_VALUE_im
  initial
    begin : i_COMPLEX_VALUE_im_fileread
      fp_i_COMPLEX_VALUE_im = $fopen("i_COMPLEX_VALUE_im.dat", "r");
      status_i_COMPLEX_VALUE_im = $rewind(fp_i_COMPLEX_VALUE_im);
    end
  always @(Data_Type_Conversion_out1_addr_delay, rdEnb, tb_enb_delay)
    begin
      if (tb_enb_delay == 0) begin
        rawData_i_COMPLEX_VALUE_im <= 32'bx;
      end
      else if (rdEnb == 1) begin
        status_i_COMPLEX_VALUE_im = $fscanf(fp_i_COMPLEX_VALUE_im, "%h",
rawData_i_COMPLEX_VALUE_im);
      end
    end
  // holdData reg for Data_Type_Conversion_out1
  always @(posedge clk or posedge reset)
    begin : stimuli_Data_Type_Conversion_out1
      if (reset) begin
        holdData_i_COMPLEX_VALUE_im <= 32'bx;
      end
      else begin
        holdData_i_COMPLEX_VALUE_im <= rawData_i_COMPLEX_VALUE_im;
      end
    end
  always @(rawData_i_COMPLEX_VALUE_im or rdEnb)
    begin : stimuli_Data_Type_Conversion_out1_1
      if (rdEnb == 1'b0) begin
        i_COMPLEX_VALUE_im_offset <= holdData_i_COMPLEX_VALUE_im;
```

```
    end
    else begin
      i_COMPLEX_VALUE_im_offset <= rawData_i_COMPLEX_VALUE_im;
    end
  end
assign #2 i_COMPLEX_VALUE_im = i_COMPLEX_VALUE_im_offset;
assign Constant_out1_active = Constant_out1_addr != 10'b1111101000;
assign Constant_out1_enb = Constant_out1_active & (rdEnb & tb_enb_delay);
// Count limited, Unsigned Counter
//  initial value   = 0
//  step value      = 1
//  count to value  = 1000
always @(posedge clk or posedge reset)
  begin : Constant_process
    if (reset == 1'b1) begin
      Constant_out1_addr <= 10'b0000000000;
    end
    else begin
      if (Constant_out1_enb) begin
        if (Constant_out1_addr >= 10'b1111101000) begin
          Constant_out1_addr <= 10'b0000000000;
        end
        else begin
          Constant_out1_addr <= Constant_out1_addr + 10'b0000000001;
        end
      end
    end
  end
assign #1 Data_Type_Conversion_out1_addr_delay = Constant_out1_addr;
// Data source for i_COMPLEX_VALUE_re
initial
  begin : i_COMPLEX_VALUE_re_fileread
    fp_i_COMPLEX_VALUE_re = $fopen("i_COMPLEX_VALUE_re.dat", "r");
    status_i_COMPLEX_VALUE_re = $rewind(fp_i_COMPLEX_VALUE_re);
  end
always @(Data_Type_Conversion_out1_addr_delay, rdEnb, tb_enb_delay)
  begin
```

```
      if (tb_enb_delay == 0) begin
        rawData_i_COMPLEX_VALUE_re <= 32'bx;
      end
      else if (rdEnb == 1) begin
        status_i_COMPLEX_VALUE_re = $fscanf(fp_i_COMPLEX_VALUE_re, "%h",
rawData_i_COMPLEX_VALUE_re);
      end
    end
  // holdData reg for Data_Type_Conversion_out1
  always @(posedge clk or posedge reset)
    begin : stimuli_Data_Type_Conversion_out1_2
      if (reset) begin
        holdData_i_COMPLEX_VALUE_re <= 32'bx;
      end
      else begin
        holdData_i_COMPLEX_VALUE_re <= rawData_i_COMPLEX_VALUE_re;
      end
    end
  always @(rawData_i_COMPLEX_VALUE_re or rdEnb)
    begin : stimuli_Data_Type_Conversion_out1_3
      if (rdEnb == 1'b0) begin
        i_COMPLEX_VALUE_re_offset <= holdData_i_COMPLEX_VALUE_re;
      end
      else begin
        i_COMPLEX_VALUE_re_offset <= rawData_i_COMPLEX_VALUE_re;
      end
    end


  assign #2 i_COMPLEX_VALUE_re = i_COMPLEX_VALUE_re_offset;
  // holdData reg for Constant_out1
  always @(posedge clk or posedge reset)
    begin : stimuli_Constant_out1
      if (reset) begin
        holdData_i_VALID <= 1'bx;
      end
      else begin
        holdData_i_VALID <= rawData_i_VALID;
```

```
    end
  end
always @(rawData_i_VALID or rdEnb)
  begin : stimuli_Constant_out1_1
    if (rdEnb == 1'b0) begin
      i_VALID_offset <= holdData_i_VALID;
    end
    else begin
      i_VALID_offset <= rawData_i_VALID;
    end
  end

assign #2 i_VALID_1 = i_VALID_offset;
assign snkDonen =  ~ snkDone;
assign resetn =  ~ reset;
assign tb_enb = resetn & snkDonen;
// Delay inside enable generation: register depth 1
always @(posedge clk or posedge reset)
  begin : u_enable_delay
    if (reset) begin
      tb_enb_delay <= 0;
    end
    else begin
      tb_enb_delay <= tb_enb;
    end
  end
assign rdEnb = (snkDone == 1'b0 ? tb_enb_delay :
            1'b0);
assign #2 clk_enable = rdEnb;
initial
  begin : reset_gen
    reset <= 1'b1;
    # (20);
    @ (posedge clk)
    # (2);
    reset <= 1'b0;
  end
```

```
always
  begin : clk_gen
    clk <= 1'b1;
    # (5);
    clk <= 1'b0;
    # (5);
    if (snkDone == 1'b1) begin
      clk <= 1'b1;
      # (5);
      clk <= 1'b0;
      # (5);
      $stop;
    end
  end
Subsystem u_Subsystem (.clk(clk),
                       .reset(reset),
                       .clk_enable(clk_enable),
                       .i_VALID(i_VALID_1),
                       .i_COMPLEX_VALUE_re(i_COMPLEX_VALUE_re),  // sfix32_En7
                       .i_COMPLEX_VALUE_im(i_COMPLEX_VALUE_im),  // sfix32_En7
                       .ce_out(ce_out),
                       .o_VALID(o_VALID),
                       .o_MAGNITUDE(o_MAGNITUDE),  // sfix33_En7
                       .o_PHASE(o_PHASE)  // sfix16_En13
                       );

assign o_VALID_enb = ce_out & o_VALID_active;
// Count limited, Unsigned Counter
//  initial value   = 0
//  step value      = 1
//  count to value  = 1000
always @(posedge clk or posedge reset)
  begin : c_2_process
    if (reset == 1'b1) begin
      o_VALID_addr <= 10'b0000000000;
    end
    else begin
```

```
      if (o_VALID_enb) begin
        if (o_VALID_addr >= 10'b1111101000) begin
          o_VALID_addr <= 10'b0000000000;
        end
        else begin
          o_VALID_addr <= o_VALID_addr + 10'b0000000001;
        end
      end
    end
  end
assign o_VALID_lastAddr = o_VALID_addr >= 10'b1111101000;
assign o_VALID_done = o_VALID_lastAddr & resetn;
// Delay to allow last sim cycle to complete
always @(posedge clk or posedge reset)
  begin : checkDone_1
    if (reset) begin
      check1_done <= 0;
    end
    else begin
      if (o_VALID_done_enb) begin
        check1_done <= o_VALID_done;
      end
    end
  end
assign snkDone = check3_done & (check1_done & check2_done);
assign o_VALID_ignCntDone = o_VALID_chkcnt != 6'b100010;
assign o_VALID_needToCount = ce_out & o_VALID_ignCntDone;
// Count limited, Unsigned Counter
//  initial value   = 0
//  step value      = 1
//  count to value  = 34
always @(posedge clk or posedge reset)
  begin : o_VALID_IgnoreDataChecking_process
    if (reset == 1'b1) begin
      o_VALID_chkcnt <= 6'b000000;
    end
    else begin
```

```
      if (o_VALID_needToCount) begin
        if (o_VALID_chkcnt >= 6'b100010) begin
          o_VALID_chkcnt <= 6'b000000;
        end
        else begin
          o_VALID_chkcnt <= o_VALID_chkcnt + 6'b000001;
        end
      end
    end
  end
assign o_VALID_chkenb = o_VALID_chkcnt == 6'b100010;
assign o_VALID_chkdata = ce_out & o_VALID_chkenb;
assign #1 o_VALID_addr_delay_1 = o_VALID_addr;
// Data source for o_VALID_expected
initial
  begin : o_VALID_expected_fileread
    fp_o_VALID_expected = $fopen("o_VALID_expected.dat", "r");
    status_o_VALID_expected = $rewind(fp_o_VALID_expected);
  end
always @(o_VALID_addr_delay_1, ce_out, tb_enb_delay)
  begin
    if (tb_enb_delay == 0) begin
      o_VALID_expected <= 1'bx;
    end
    else if (ce_out == 1) begin
      status_o_VALID_expected = $fscanf(fp_o_VALID_expected, "%h", o_VALID_expected);
    end
  end
assign o_VALID_ref = o_VALID_expected;

always @(posedge clk or posedge reset)
  begin : o_VALID_checker
    if (reset == 1'b1) begin
      o_VALID_testFailure <= 1'b0;
    end
    else begin
      if (o_VALID_chkdata == 1'b1 && o_VALID !== o_VALID_ref) begin
```

```
          o_VALID_testFailure <= 1'b1;
          $display("ERROR in o_VALID at time %t : Expected '%h' Actual '%h'", $time,
o_VALID_ref, o_VALID);
        end
      end
    end
  assign o_MAGNITUDE_ignCntDone = o_MAGNITUDE_chkcnt != 6'b100010;
  assign o_MAGNITUDE_needToCount = ce_out & o_MAGNITUDE_ignCntDone;
  // Count limited, Unsigned Counter
  //  initial value   = 0
  //  step value      = 1
  //  count to value  = 34
  always @(posedge clk or posedge reset)
    begin : o_MAGNITUDE_IgnoreDataChecking_process
      if (reset == 1'b1) begin
        o_MAGNITUDE_chkcnt <= 6'b000000;
      end
      else begin
        if (o_MAGNITUDE_needToCount) begin
          if (o_MAGNITUDE_chkcnt >= 6'b100010) begin
            o_MAGNITUDE_chkcnt <= 6'b000000;
          end
          else begin
            o_MAGNITUDE_chkcnt <= o_MAGNITUDE_chkcnt + 6'b000001;
          end
        end
      end
    end
  assign o_MAGNITUDE_chkenb = o_MAGNITUDE_chkcnt == 6'b100010;
  assign o_MAGNITUDE_chkdata = ce_out & o_MAGNITUDE_chkenb;
  assign #1 o_MAGNITUDE_addr_delay_1 = o_VALID_addr;
  // Data source for o_MAGNITUDE_expected
  initial
    begin : o_MAGNITUDE_expected_fileread
      fp_o_MAGNITUDE_expected = $fopen("o_MAGNITUDE_expected.dat", "r");
      status_o_MAGNITUDE_expected = $rewind(fp_o_MAGNITUDE_expected);
    end
```

```
  always @(o_MAGNITUDE_addr_delay_1, ce_out, tb_enb_delay)
    begin
      if (tb_enb_delay == 0) begin
        o_MAGNITUDE_expected <= 33'bx;
      end
      else if (ce_out == 1) begin
        status_o_MAGNITUDE_expected = $fscanf(fp_o_MAGNITUDE_expected, "%h",
o_MAGNITUDE_expected);
      end
    end
  assign o_MAGNITUDE_ref = o_MAGNITUDE_expected;
  always @(posedge clk or posedge reset)
    begin : o_MAGNITUDE_checker
      if (reset == 1'b1) begin
        o_MAGNITUDE_testFailure <= 1'b0;
      end
      else begin
        if (o_MAGNITUDE_chkdata == 1'b1 && o_MAGNITUDE !== o_MAGNITUDE_ref) begin
          o_MAGNITUDE_testFailure <= 1'b1;
          $display("ERROR in o_MAGNITUDE at time %t : Expected '%h' Actual '%h'",
$time, o_MAGNITUDE_ref, o_MAGNITUDE);
        end
      end
    end
  assign o_PHASE_ignCntDone = o_PHASE_chkcnt != 6'b100010;
  assign o_PHASE_needToCount = ce_out & o_PHASE_ignCntDone;
  // Count limited, Unsigned Counter
  //  initial value   = 0
  //  step value      = 1
  //  count to value  = 34
  always @(posedge clk or posedge reset)
    begin : o_PHASE_IgnoreDataChecking_process
      if (reset == 1'b1) begin
        o_PHASE_chkcnt <= 6'b000000;
      end
      else begin
        if (o_PHASE_needToCount) begin
          if (o_PHASE_chkcnt >= 6'b100010) begin
```

```
          o_PHASE_chkcnt <= 6'b000000;
        end
        else begin
          o_PHASE_chkcnt <= o_PHASE_chkcnt + 6'b000001;
        end
      end
    end
  end
assign o_PHASE_chkenb = o_PHASE_chkcnt == 6'b100010;
assign o_PHASE_chkdata = ce_out & o_PHASE_chkenb;
assign #1 o_PHASE_addr_delay_1 = o_VALID_addr;
// Data source for o_PHASE_expected
initial
  begin : o_PHASE_expected_fileread
    fp_o_PHASE_expected = $fopen("o_PHASE_expected.dat", "r");
    status_o_PHASE_expected = $rewind(fp_o_PHASE_expected);
  end

always @(o_PHASE_addr_delay_1, ce_out, tb_enb_delay)
  begin
    if (tb_enb_delay == 0) begin
      o_PHASE_expected <= 16'bx;
    end
    else if (ce_out == 1) begin
      status_o_PHASE_expected = $fscanf(fp_o_PHASE_expected, "%h", o_PHASE_expected);
    end
  end
assign o_PHASE_ref = o_PHASE_expected;
always @(posedge clk or posedge reset)
  begin : o_PHASE_checker
    if (reset == 1'b1) begin
      o_PHASE_testFailure <= 1'b0;
    end
    else begin
      if (o_PHASE_chkdata == 1'b1 && o_PHASE !== o_PHASE_ref) begin
        o_PHASE_testFailure <= 1'b1;
```

```
          $display("ERROR in o_PHASE at time %t : Expected '%h' Actual '%h'", $time,
o_PHASE_ref, o_PHASE);
        end
      end
    end
  assign testFailure = o_PHASE_testFailure | (o_VALID_testFailure |
o_MAGNITUDE_testFailure);
  always @(posedge clk)
    begin : completed_msg
      if (snkDone == 1'b1) begin
        if (testFailure == 1'b0) begin
          $display("**************TEST COMPLETED (PASSED)**************");
        end
        else begin
          $display("**************TEST COMPLETED (FAILED)**************");
        end
      end
    end
endmodule  // Subsystem_tb
```

Лістинг 3: Verilog файл Subsystem.v для чисел з плаваючою комою

```
`timescale 1 ns / 1 ns
module Subsystem
          (clk,
           reset,
           clk_enable,
           i_VALID,
           i_COMPLEX_VALUE_re,
           i_COMPLEX_VALUE_im,
           ce_out,
           o_VALID,
           o_MAGNITUDE,
           o_PHASE);
  input   clk;
  input   reset;
  input   clk_enable;
  input   i_VALID;
  input   [31:0] i_COMPLEX_VALUE_re;  // single
```

```
input   [31:0] i_COMPLEX_VALUE_im;  // single
output  ce_out;
output  o_VALID;
output  [31:0] o_MAGNITUDE;  // single
output  [31:0] o_PHASE;  // single
wire enb;
reg  [0:46] delayMatch_reg;  // ufix1 [47]
wire [0:46] delayMatch_reg_next;  // ufix1 [47]
wire i_VALID_1;
wire [31:0] Product_out1;  // ufix32
wire [31:0] Product1_out1;  // ufix32
wire [31:0] Add_out1;  // ufix32
wire [31:0] Sqrt_out1;  // ufix32
wire [31:0] Trigonometric_Function_out1;  // ufix32
reg [31:0] delayMatch1_reg [0:4];  // ufix32 [5]
wire [31:0] delayMatch1_reg_next [0:4];  // ufix32 [5]
wire [31:0] Trigonometric_Function_out1_1;  // ufix32
assign enb = clk_enable;
always @(posedge clk or posedge reset)
  begin : delayMatch_process
    if (reset == 1'b1) begin
      delayMatch_reg[0] <= 1'b0;
      delayMatch_reg[1] <= 1'b0;
      delayMatch_reg[2] <= 1'b0;
      delayMatch_reg[3] <= 1'b0;
      delayMatch_reg[4] <= 1'b0;
      delayMatch_reg[5] <= 1'b0;
      delayMatch_reg[6] <= 1'b0;
      delayMatch_reg[7] <= 1'b0;
      delayMatch_reg[8] <= 1'b0;
      delayMatch_reg[9] <= 1'b0;
      delayMatch_reg[10] <= 1'b0;
      delayMatch_reg[11] <= 1'b0;
      delayMatch_reg[12] <= 1'b0;
      delayMatch_reg[13] <= 1'b0;
      delayMatch_reg[14] <= 1'b0;
      delayMatch_reg[15] <= 1'b0;
```

```
        delayMatch_reg[16] <= 1'b0;
        delayMatch_reg[17] <= 1'b0;
        delayMatch_reg[18] <= 1'b0;
        delayMatch_reg[19] <= 1'b0;
        delayMatch_reg[20] <= 1'b0;
        delayMatch_reg[21] <= 1'b0;
        delayMatch_reg[22] <= 1'b0;
        delayMatch_reg[23] <= 1'b0;
        delayMatch_reg[24] <= 1'b0;
        delayMatch_reg[25] <= 1'b0;
        delayMatch_reg[26] <= 1'b0;
        delayMatch_reg[27] <= 1'b0;
        delayMatch_reg[28] <= 1'b0;
        delayMatch_reg[29] <= 1'b0;
        delayMatch_reg[30] <= 1'b0;
        delayMatch_reg[31] <= 1'b0;
        delayMatch_reg[32] <= 1'b0;
        delayMatch_reg[33] <= 1'b0;
        delayMatch_reg[34] <= 1'b0;
        delayMatch_reg[35] <= 1'b0;
        delayMatch_reg[36] <= 1'b0;
        delayMatch_reg[37] <= 1'b0;
        delayMatch_reg[38] <= 1'b0;
        delayMatch_reg[39] <= 1'b0;
        delayMatch_reg[40] <= 1'b0;
        delayMatch_reg[41] <= 1'b0;
        delayMatch_reg[42] <= 1'b0;
        delayMatch_reg[43] <= 1'b0;
        delayMatch_reg[44] <= 1'b0;
        delayMatch_reg[45] <= 1'b0;
        delayMatch_reg[46] <= 1'b0;
      end
      else begin
        if (enb) begin
          delayMatch_reg[0] <= delayMatch_reg_next[0];
          delayMatch_reg[1] <= delayMatch_reg_next[1];
          delayMatch_reg[2] <= delayMatch_reg_next[2];
```

```
        delayMatch_reg[3] <= delayMatch_reg_next[3];
        delayMatch_reg[4] <= delayMatch_reg_next[4];
        delayMatch_reg[5] <= delayMatch_reg_next[5];
        delayMatch_reg[6] <= delayMatch_reg_next[6];
        delayMatch_reg[7] <= delayMatch_reg_next[7];
        delayMatch_reg[8] <= delayMatch_reg_next[8];
        delayMatch_reg[9] <= delayMatch_reg_next[9];
        delayMatch_reg[10] <= delayMatch_reg_next[10];
        delayMatch_reg[11] <= delayMatch_reg_next[11];
        delayMatch_reg[12] <= delayMatch_reg_next[12];
        delayMatch_reg[13] <= delayMatch_reg_next[13];
        delayMatch_reg[14] <= delayMatch_reg_next[14];
        delayMatch_reg[15] <= delayMatch_reg_next[15];
        delayMatch_reg[16] <= delayMatch_reg_next[16];
        delayMatch_reg[17] <= delayMatch_reg_next[17];
        delayMatch_reg[18] <= delayMatch_reg_next[18];
        delayMatch_reg[19] <= delayMatch_reg_next[19];
        delayMatch_reg[20] <= delayMatch_reg_next[20];
        delayMatch_reg[21] <= delayMatch_reg_next[21];
        delayMatch_reg[22] <= delayMatch_reg_next[22];
        delayMatch_reg[23] <= delayMatch_reg_next[23];
        delayMatch_reg[24] <= delayMatch_reg_next[24];
        delayMatch_reg[25] <= delayMatch_reg_next[25];
        delayMatch_reg[26] <= delayMatch_reg_next[26];
        delayMatch_reg[27] <= delayMatch_reg_next[27];
        delayMatch_reg[28] <= delayMatch_reg_next[28];
        delayMatch_reg[29] <= delayMatch_reg_next[29];
        delayMatch_reg[30] <= delayMatch_reg_next[30];
        delayMatch_reg[31] <= delayMatch_reg_next[31];
        delayMatch_reg[32] <= delayMatch_reg_next[32];
        delayMatch_reg[33] <= delayMatch_reg_next[33];
        delayMatch_reg[34] <= delayMatch_reg_next[34];
        delayMatch_reg[35] <= delayMatch_reg_next[35];
        delayMatch_reg[36] <= delayMatch_reg_next[36];
        delayMatch_reg[37] <= delayMatch_reg_next[37];
        delayMatch_reg[38] <= delayMatch_reg_next[38];
        delayMatch_reg[39] <= delayMatch_reg_next[39];
```

```
          delayMatch_reg[40] <= delayMatch_reg_next[40];
          delayMatch_reg[41] <= delayMatch_reg_next[41];
          delayMatch_reg[42] <= delayMatch_reg_next[42];
          delayMatch_reg[43] <= delayMatch_reg_next[43];
          delayMatch_reg[44] <= delayMatch_reg_next[44];
          delayMatch_reg[45] <= delayMatch_reg_next[45];
          delayMatch_reg[46] <= delayMatch_reg_next[46];
        end
      end
    end
  assign i_VALID_1 = delayMatch_reg[46];
  assign delayMatch_reg_next[0] = i_VALID;
  assign delayMatch_reg_next[1] = delayMatch_reg[0];
  assign delayMatch_reg_next[2] = delayMatch_reg[1];
  assign delayMatch_reg_next[3] = delayMatch_reg[2];
  assign delayMatch_reg_next[4] = delayMatch_reg[3];
  assign delayMatch_reg_next[5] = delayMatch_reg[4];
  assign delayMatch_reg_next[6] = delayMatch_reg[5];
  assign delayMatch_reg_next[7] = delayMatch_reg[6];
  assign delayMatch_reg_next[8] = delayMatch_reg[7];
  assign delayMatch_reg_next[9] = delayMatch_reg[8];
  assign delayMatch_reg_next[10] = delayMatch_reg[9];
  assign delayMatch_reg_next[11] = delayMatch_reg[10];
  assign delayMatch_reg_next[12] = delayMatch_reg[11];
  assign delayMatch_reg_next[13] = delayMatch_reg[12];
  assign delayMatch_reg_next[14] = delayMatch_reg[13];
  assign delayMatch_reg_next[15] = delayMatch_reg[14];
  assign delayMatch_reg_next[16] = delayMatch_reg[15];
  assign delayMatch_reg_next[17] = delayMatch_reg[16];
  assign delayMatch_reg_next[18] = delayMatch_reg[17];
  assign delayMatch_reg_next[19] = delayMatch_reg[18];
  assign delayMatch_reg_next[20] = delayMatch_reg[19];
  assign delayMatch_reg_next[21] = delayMatch_reg[20];
  assign delayMatch_reg_next[22] = delayMatch_reg[21];
  assign delayMatch_reg_next[23] = delayMatch_reg[22];
  assign delayMatch_reg_next[24] = delayMatch_reg[23];
  assign delayMatch_reg_next[25] = delayMatch_reg[24];
```

```
assign delayMatch_reg_next[26] = delayMatch_reg[25];
assign delayMatch_reg_next[27] = delayMatch_reg[26];
assign delayMatch_reg_next[28] = delayMatch_reg[27];
assign delayMatch_reg_next[29] = delayMatch_reg[28];
assign delayMatch_reg_next[30] = delayMatch_reg[29];
assign delayMatch_reg_next[31] = delayMatch_reg[30];
assign delayMatch_reg_next[32] = delayMatch_reg[31];
assign delayMatch_reg_next[33] = delayMatch_reg[32];
assign delayMatch_reg_next[34] = delayMatch_reg[33];
assign delayMatch_reg_next[35] = delayMatch_reg[34];
assign delayMatch_reg_next[36] = delayMatch_reg[35];
assign delayMatch_reg_next[37] = delayMatch_reg[36];
assign delayMatch_reg_next[38] = delayMatch_reg[37];
assign delayMatch_reg_next[39] = delayMatch_reg[38];
assign delayMatch_reg_next[40] = delayMatch_reg[39];
assign delayMatch_reg_next[41] = delayMatch_reg[40];
assign delayMatch_reg_next[42] = delayMatch_reg[41];
assign delayMatch_reg_next[43] = delayMatch_reg[42];
assign delayMatch_reg_next[44] = delayMatch_reg[43];
assign delayMatch_reg_next[45] = delayMatch_reg[44];
assign delayMatch_reg_next[46] = delayMatch_reg[45];
assign o_VALID = i_VALID_1;
nfp_mul_single u_nfp_mul_comp (.clk(clk),
                               .reset(reset),
                               .enb(clk_enable),
                               .nfp_in1(i_COMPLEX_VALUE_re),  // single
                               .nfp_in2(i_COMPLEX_VALUE_re),  // single
                               .nfp_out(Product_out1)  // single
                               );
nfp_mul_single u_nfp_mul_comp_1 (.clk(clk),
                                 .reset(reset),
                                 .enb(clk_enable),
                                 .nfp_in1(i_COMPLEX_VALUE_im),  // single
                                 .nfp_in2(i_COMPLEX_VALUE_im),  // single
                                 .nfp_out(Product1_out1)  // single
                                 );
```

```
nfp_add_single u_nfp_add_comp (.clk(clk),
                               .reset(reset),
                               .enb(clk_enable),
                               .nfp_in1(Product_out1),  // single
                               .nfp_in2(Product1_out1),  // single
                               .nfp_out(Add_out1)  // single
                               );
nfp_sqrt_single u_nfp_sqrt_comp (.clk(clk),
                                 .reset(reset),
                                 .enb(clk_enable),
                                 .nfp_in(Add_out1),  // single
                                 .nfp_out(Sqrt_out1)  // single
                                 );
nfp_atan2_single u_nfp_atan2_comp (.clk(clk),
                                   .reset(reset),
                                   .enb(clk_enable),
                                   .nfp_in1(i_COMPLEX_VALUE_im),  // single
                                   .nfp_in2(i_COMPLEX_VALUE_re),  // single
                                   .nfp_out(Trigonometric_Function_out1)  // single
                                   );
always @(posedge clk or posedge reset)
  begin : delayMatch1_process
    if (reset == 1'b1) begin
      delayMatch1_reg[0] <= 32'h00000000;
      delayMatch1_reg[1] <= 32'h00000000;
      delayMatch1_reg[2] <= 32'h00000000;
      delayMatch1_reg[3] <= 32'h00000000;
      delayMatch1_reg[4] <= 32'h00000000;
    end
    else begin
      if (enb) begin
        delayMatch1_reg[0] <= delayMatch1_reg_next[0];
        delayMatch1_reg[1] <= delayMatch1_reg_next[1];
        delayMatch1_reg[2] <= delayMatch1_reg_next[2];
        delayMatch1_reg[3] <= delayMatch1_reg_next[3];
        delayMatch1_reg[4] <= delayMatch1_reg_next[4];
      end
```

```
      end
    end
  assign Trigonometric_Function_out1_1 = delayMatch1_reg[4];
  assign delayMatch1_reg_next[0] = Trigonometric_Function_out1;
  assign delayMatch1_reg_next[1] = delayMatch1_reg[0];
  assign delayMatch1_reg_next[2] = delayMatch1_reg[1];
  assign delayMatch1_reg_next[3] = delayMatch1_reg[2];
  assign delayMatch1_reg_next[4] = delayMatch1_reg[3];
  assign ce_out = clk_enable;
  assign o_MAGNITUDE = Sqrt_out1;
  assign o_PHASE = Trigonometric_Function_out1_1;
endmodule  // Subsystem
```

Лістинг 4: Тестовий файл Subsystem_tb.v для чисел з плаваючою комою

```
`timescale 1 ns / 1 ns
module Subsystem_tb;
  reg  clk;
  reg  reset;
  wire clk_enable;
  wire rawData_i_VALID;
  wire o_PHASE_done;  // ufix1
  wire rdEnb;
  wire o_PHASE_done_enb;  // ufix1
  reg [9:0] o_VALID_addr;  // ufix10
  wire o_PHASE_lastAddr;  // ufix1
  wire resetn;
  reg  check3_done;  // ufix1
  wire o_MAGNITUDE_done;  // ufix1
  wire o_MAGNITUDE_done_enb;  // ufix1
  wire o_MAGNITUDE_lastAddr;  // ufix1
  reg  check2_done;  // ufix1
  wire o_VALID_done;  // ufix1
  wire o_VALID_done_enb;  // ufix1
  wire o_VALID_active;  // ufix1
  wire [9:0] ComplexValFixed_addr_delay;  // ufix10
  reg  tb_enb_delay;
```

```
reg signed [31:0] fp_i_COMPLEX_VALUE_im;  // sfix32
reg [31:0] rawData_i_COMPLEX_VALUE_im;  // ufix32
reg signed [31:0] status_i_COMPLEX_VALUE_im;  // sfix32
reg [31:0] holdData_i_COMPLEX_VALUE_im;  // ufix32
reg [31:0] i_COMPLEX_VALUE_im_offset;  // ufix32
reg [31:0] i_COMPLEX_VALUE_im;  // ufix32
wire [31:0] i_COMPLEX_VALUE_im_1;  // ufix32
reg [9:0] Constant_out1_addr;  // ufix10
wire Constant_out1_active;  // ufix1
wire Constant_out1_enb;  // ufix1
reg signed [31:0] fp_i_COMPLEX_VALUE_re;  // sfix32
reg [31:0] rawData_i_COMPLEX_VALUE_re;  // ufix32
reg signed [31:0] status_i_COMPLEX_VALUE_re;  // sfix32
reg [31:0] holdData_i_COMPLEX_VALUE_re;  // ufix32
reg [31:0] i_COMPLEX_VALUE_re_offset;  // ufix32
reg [31:0] i_COMPLEX_VALUE_re;  // ufix32
wire [31:0] i_COMPLEX_VALUE_re_1;  // ufix32
reg  holdData_i_VALID;
reg  i_VALID_offset;
wire i_VALID_1;
wire snkDone;
wire snkDonen;
wire tb_enb;
wire ce_out;
wire o_VALID;
wire [31:0] o_MAGNITUDE;  // ufix32
wire [31:0] o_PHASE;  // ufix32
wire o_VALID_enb;  // ufix1
wire o_VALID_lastAddr;  // ufix1
reg  check1_done;  // ufix1
reg [5:0] o_VALID_chkcnt;  // ufix6
wire o_VALID_ignCntDone;  // ufix1
wire o_VALID_needToCount;  // ufix1
wire o_VALID_chkenb;  // ufix1
wire o_VALID_chkdata;  // ufix1
wire [9:0] o_VALID_addr_delay_1;  // ufix10
reg signed [31:0] fp_o_VALID_expected;  // sfix32
```

```
reg  o_VALID_expected;
reg signed [31:0] status_o_VALID_expected;  // sfix32
wire o_VALID_ref;
reg  o_VALID_testFailure;  // ufix1
reg [5:0] o_MAGNITUDE_chkcnt;  // ufix6
wire o_MAGNITUDE_ignCntDone;  // ufix1
wire o_MAGNITUDE_needToCount;  // ufix1
wire o_MAGNITUDE_chkenb;  // ufix1
wire o_MAGNITUDE_chkdata;  // ufix1
wire [9:0] o_MAGNITUDE_addr_delay_1;  // ufix10
reg signed [31:0] fp_o_MAGNITUDE_expected;  // sfix32
reg [31:0] o_MAGNITUDE_expected;  // ufix32
reg signed [31:0] status_o_MAGNITUDE_expected;  // sfix32
wire [31:0] o_MAGNITUDE_ref;  // ufix32
reg  o_MAGNITUDE_testFailure;  // ufix1
reg [5:0] o_PHASE_chkcnt;  // ufix6
wire o_PHASE_ignCntDone;  // ufix1
wire o_PHASE_needToCount;  // ufix1
wire o_PHASE_chkenb;  // ufix1
wire o_PHASE_chkdata;  // ufix1
wire [9:0] o_PHASE_addr_delay_1;  // ufix10
reg signed [31:0] fp_o_PHASE_expected;  // sfix32
reg [31:0] o_PHASE_expected;  // ufix32
reg signed [31:0] status_o_PHASE_expected;  // sfix32
wire [31:0] o_PHASE_ref;  // ufix32
reg  o_PHASE_testFailure;  // ufix1
wire testFailure;  // ufix1
function real absReal(input real num);
begin
  if (num < 0)
    absReal = -num;
  else
    absReal = num;
end
endfunction

function real floatHalfToReal;
```

```
input [15:0] x;
reg [63:0] conv;
begin
  conv[63] = x[15]; // sign
  if (x[14:10] == 5'b0) // exp
    conv[62:52] = 11'b0;
  else
    conv[62:52] = 1023 + (x[14:10] - 15);
  conv[51:42] = x[9:0]; // mantissa
  conv[41:0] = 42'b0;
  if (((x[14:10] == 5'h1F) && (x[9:0] != 10'h0))) // check for NaN
  begin
    conv[63] = 1'b0;
    conv[62:52] = 11'h7FF;
    conv[51:0] = 52'h0;
  end
  floatHalfToReal = $bitstoreal(conv);
end
endfunction
function real floatSingleToReal;
input [31:0] x;
reg [63:0] conv;
begin
  conv[63] = x[31]; // sign
  if (x[30:23] == 8'b0) // exp
    conv[62:52] = 11'b0;
  else
    conv[62:52] = 1023 + (x[30:23] - 127);
  conv[51:29] = x[22:0]; // mantissa
  conv[28:0] = 29'b0;
  if (((x[30:23] == 8'hFF) && (x[22:0] != 23'h0))) // check for NaN
  begin
    conv[63] = 1'b0;
    conv[62:52] = 11'h7FF;
    conv[51:0] = 52'h0;
  end
  floatSingleToReal = $bitstoreal(conv);
```

```
end
endfunction
function real floatDoubleToReal;
input [63:0] x;
reg [63:0] conv;
begin
  conv[63:0] = x[63:0];
  if (((x[62:52] == 11'h7FF) && (x[51:0] != 52'h0))) // check for NaN
  begin
    conv[63] = 1'b0;
    conv[62:52] = 11'h7FF;
    conv[51:0] = 52'h0;
  end
  floatDoubleToReal = $bitstoreal(conv);
end
endfunction
function isFloatEpsEqual(input real a, input real b, input real eps);
real absdiff;
begin
  absdiff = absReal(a - b);
  if (absdiff < eps) // absolute error check
    isFloatEpsEqual = 1;
  else if (a == b) // check infinities
    isFloatEpsEqual = 1;
  else if (a*b == 0.0) // either is zero
    isFloatEpsEqual = (absdiff < eps);
  else if (absReal(a) < absReal(b)) // relative error check
    isFloatEpsEqual = absdiff/absReal(b) < eps;
  else
    isFloatEpsEqual = absdiff/absReal(a) < eps;
end
endfunction
function isFloatHalfEpsEqual;
input [15:0] x;
input [15:0] y;
input real eps;
real a, b;
```

```
real absdiff;
begin
  a = floatHalfToReal(x);
  b = floatHalfToReal(y);
  isFloatHalfEpsEqual = isFloatEpsEqual(a, b, eps);
end
endfunction
function isFloatSingleEpsEqual;
input [31:0] x;
input [31:0] y;
input real eps;
real a, b;
real absdiff;
begin
  a = floatSingleToReal(x);
  b = floatSingleToReal(y);
  isFloatSingleEpsEqual = isFloatEpsEqual(a, b, eps);
end
endfunction
function isFloatDoubleEpsEqual;
input [63:0] x;
input [63:0] y;
input real eps;
real a, b;
real absdiff;
begin
  a = floatDoubleToReal(x);
  b = floatDoubleToReal(y);
  isFloatDoubleEpsEqual = isFloatEpsEqual(a, b, eps);
end
endfunction
// Data source for i_VALID
assign rawData_i_VALID = 1'b1;
assign o_PHASE_done_enb = o_PHASE_done & rdEnb;
assign o_PHASE_lastAddr = o_VALID_addr >= 10'b1111101000;
assign o_PHASE_done = o_PHASE_lastAddr & resetn;
// Delay to allow last sim cycle to complete
```

```
always @(posedge clk or posedge reset)
  begin : checkDone_3
    if (reset) begin
      check3_done <= 0;
    end
    else begin
      if (o_PHASE_done_enb) begin
        check3_done <= o_PHASE_done;
      end
    end
  end
assign o_MAGNITUDE_done_enb = o_MAGNITUDE_done & rdEnb;
assign o_MAGNITUDE_lastAddr = o_VALID_addr >= 10'b1111101000;
assign o_MAGNITUDE_done = o_MAGNITUDE_lastAddr & resetn;
// Delay to allow last sim cycle to complete
always @(posedge clk or posedge reset)
  begin : checkDone_2
    if (reset) begin
      check2_done <= 0;
    end
    else begin
      if (o_MAGNITUDE_done_enb) begin
        check2_done <= o_MAGNITUDE_done;
      end
    end
  end
assign o_VALID_done_enb = o_VALID_done & rdEnb;
assign o_VALID_active = o_VALID_addr != 10'b1111101000;
// Data source for i_COMPLEX_VALUE_im
initial
  begin : i_COMPLEX_VALUE_im_fileread
    fp_i_COMPLEX_VALUE_im = $fopen("i_COMPLEX_VALUE_im.dat", "r");
    status_i_COMPLEX_VALUE_im = $rewind(fp_i_COMPLEX_VALUE_im);
  end
always @(ComplexValFixed_addr_delay, rdEnb, tb_enb_delay)
  begin
    if (tb_enb_delay == 0) begin
```

```
        rawData_i_COMPLEX_VALUE_im <= 32'bx;
      end
      else if (rdEnb == 1) begin
        status_i_COMPLEX_VALUE_im      =      $fscanf(fp_i_COMPLEX_VALUE_im,      "%h",
rawData_i_COMPLEX_VALUE_im);
      end
    end
  // holdData reg for ComplexValFixed
  always @(posedge clk or posedge reset)
    begin : stimuli_ComplexValFixed
      if (reset) begin
        holdData_i_COMPLEX_VALUE_im <= 32'bx;
      end
      else begin
        holdData_i_COMPLEX_VALUE_im <= rawData_i_COMPLEX_VALUE_im;
      end
    end
  always @(rawData_i_COMPLEX_VALUE_im or rdEnb)
    begin : stimuli_ComplexValFixed_1
      if (rdEnb == 1'b0) begin
        i_COMPLEX_VALUE_im_offset <= holdData_i_COMPLEX_VALUE_im;
      end
      else begin
        i_COMPLEX_VALUE_im_offset <= rawData_i_COMPLEX_VALUE_im;
      end
    end
  always #2 i_COMPLEX_VALUE_im <= i_COMPLEX_VALUE_im_offset;
  assign i_COMPLEX_VALUE_im_1 = i_COMPLEX_VALUE_im;
  assign Constant_out1_active = Constant_out1_addr != 10'b1111101000;
  assign Constant_out1_enb = Constant_out1_active & (rdEnb & tb_enb_delay);
  // Count limited, Unsigned Counter
  //  initial value   = 0
  //  step value      = 1
  //  count to value  = 1000
  always @(posedge clk or posedge reset)
    begin : Constant_process
      if (reset == 1'b1) begin
```

```
        Constant_out1_addr <= 10'b0000000000;
      end
      else begin
        if (Constant_out1_enb) begin
          if (Constant_out1_addr >= 10'b1111101000) begin
            Constant_out1_addr <= 10'b0000000000;
          end
          else begin
            Constant_out1_addr <= Constant_out1_addr + 10'b0000000001;
          end
        end
      end
    end
  assign #1 ComplexValFixed_addr_delay = Constant_out1_addr;

  // Data source for i_COMPLEX_VALUE_re
  initial
    begin : i_COMPLEX_VALUE_re_fileread
      fp_i_COMPLEX_VALUE_re = $fopen("i_COMPLEX_VALUE_re.dat", "r");
      status_i_COMPLEX_VALUE_re = $rewind(fp_i_COMPLEX_VALUE_re);
    end
  always @(ComplexValFixed_addr_delay, rdEnb, tb_enb_delay)
    begin
      if (tb_enb_delay == 0) begin
        rawData_i_COMPLEX_VALUE_re <= 32'bx;
      end
      else if (rdEnb == 1) begin
        status_i_COMPLEX_VALUE_re      =      $fscanf(fp_i_COMPLEX_VALUE_re,      "%h",
rawData_i_COMPLEX_VALUE_re);
      end
    end
  // holdData reg for ComplexValFixed
  always @(posedge clk or posedge reset)
    begin : stimuli_ComplexValFixed_2
      if (reset) begin
        holdData_i_COMPLEX_VALUE_re <= 32'bx;
      end
```

```
    else begin
      holdData_i_COMPLEX_VALUE_re <= rawData_i_COMPLEX_VALUE_re;
    end
  end
always @(rawData_i_COMPLEX_VALUE_re or rdEnb)
  begin : stimuli_ComplexValFixed_3
    if (rdEnb == 1'b0) begin
      i_COMPLEX_VALUE_re_offset <= holdData_i_COMPLEX_VALUE_re;
    end
    else begin
      i_COMPLEX_VALUE_re_offset <= rawData_i_COMPLEX_VALUE_re;
    end
  end
always #2 i_COMPLEX_VALUE_re <= i_COMPLEX_VALUE_re_offset;
assign i_COMPLEX_VALUE_re_1 = i_COMPLEX_VALUE_re;
// holdData reg for Constant_out1
always @(posedge clk or posedge reset)
  begin : stimuli_Constant_out1
    if (reset) begin
      holdData_i_VALID <= 1'bx;
    end
    else begin
      holdData_i_VALID <= rawData_i_VALID;
    end
  end
always @(rawData_i_VALID or rdEnb)
  begin : stimuli_Constant_out1_1
    if (rdEnb == 1'b0) begin
      i_VALID_offset <= holdData_i_VALID;
    end
    else begin
      i_VALID_offset <= rawData_i_VALID;
    end
  end
assign #2 i_VALID_1 = i_VALID_offset;
assign snkDonen =  ~ snkDone;
assign resetn =  ~ reset;
```

```
assign tb_enb = resetn & snkDonen;
// Delay inside enable generation: register depth 1
always @(posedge clk or posedge reset)
  begin : u_enable_delay
    if (reset) begin
      tb_enb_delay <= 0;
    end
    else begin
      tb_enb_delay <= tb_enb;
    end
  end
assign rdEnb = (snkDone == 1'b0 ? tb_enb_delay :
            1'b0);
assign #2 clk_enable = rdEnb;
initial
  begin : reset_gen
    reset <= 1'b1;
    # (20);
    @ (posedge clk)
    # (2);
    reset <= 1'b0;
  end
always
  begin : clk_gen
    clk <= 1'b1;
    # (5);
    clk <= 1'b0;
    # (5);
    if (snkDone == 1'b1) begin
      clk <= 1'b1;
      # (5);
      clk <= 1'b0;
      # (5);
      $stop;
    end
  end
```

```
Subsystem u_Subsystem (.clk(clk),
                       .reset(reset),
                       .clk_enable(clk_enable),
                       .i_VALID(i_VALID_1),
                       .i_COMPLEX_VALUE_re(i_COMPLEX_VALUE_re_1),  // single
                       .i_COMPLEX_VALUE_im(i_COMPLEX_VALUE_im_1),  // single
                       .ce_out(ce_out),
                       .o_VALID(o_VALID),
                       .o_MAGNITUDE(o_MAGNITUDE),  // single
                       .o_PHASE(o_PHASE)  // single
                       );
assign o_VALID_enb = ce_out & o_VALID_active;
// Count limited, Unsigned Counter
//  initial value   = 0
//  step value      = 1
//  count to value  = 1000
always @(posedge clk or posedge reset)
  begin : c_2_process
    if (reset == 1'b1) begin
      o_VALID_addr <= 10'b0000000000;
    end
    else begin
      if (o_VALID_enb) begin
        if (o_VALID_addr >= 10'b1111101000) begin
          o_VALID_addr <= 10'b0000000000;
        end
        else begin
          o_VALID_addr <= o_VALID_addr + 10'b0000000001;
        end
      end
    end
  end
assign o_VALID_lastAddr = o_VALID_addr >= 10'b1111101000;
assign o_VALID_done = o_VALID_lastAddr & resetn;
// Delay to allow last sim cycle to complete
always @(posedge clk or posedge reset)
  begin : checkDone_1
```

```
    if (reset) begin
      check1_done <= 0;
    end
    else begin
      if (o_VALID_done_enb) begin
        check1_done <= o_VALID_done;
      end
    end
  end
assign snkDone = check3_done & (check1_done & check2_done);
assign o_VALID_ignCntDone = o_VALID_chkcnt != 6'b101111;
assign o_VALID_needToCount = ce_out & o_VALID_ignCntDone;
// Count limited, Unsigned Counter
//  initial value   = 0
//  step value      = 1
//  count to value  = 47
always @(posedge clk or posedge reset)
  begin : o_VALID_IgnoreDataChecking_process
    if (reset == 1'b1) begin
      o_VALID_chkcnt <= 6'b000000;
    end
    else begin
      if (o_VALID_needToCount) begin
        if (o_VALID_chkcnt >= 6'b101111) begin
          o_VALID_chkcnt <= 6'b000000;
        end
        else begin
          o_VALID_chkcnt <= o_VALID_chkcnt + 6'b000001;
        end
      end
    end
  end
assign o_VALID_chkenb = o_VALID_chkcnt == 6'b101111;
assign o_VALID_chkdata = ce_out & o_VALID_chkenb;
```

```
  assign #1 o_VALID_addr_delay_1 = o_VALID_addr;


  // Data source for o_VALID_expected
  initial
    begin : o_VALID_expected_fileread
      fp_o_VALID_expected = $fopen("o_VALID_expected.dat", "r");
      status_o_VALID_expected = $rewind(fp_o_VALID_expected);
    end
  always @(o_VALID_addr_delay_1, ce_out, tb_enb_delay)
    begin
      if (tb_enb_delay == 0) begin
        o_VALID_expected <= 1'bx;
      end
      else if (ce_out == 1) begin
        status_o_VALID_expected = $fscanf(fp_o_VALID_expected, "%h", o_VALID_expected);
      end
    end
  assign o_VALID_ref = o_VALID_expected;
  always @(posedge clk or posedge reset)
    begin : o_VALID_checker
      if (reset == 1'b1) begin
        o_VALID_testFailure <= 1'b0;
      end
      else begin
        if (o_VALID_chkdata == 1'b1 && o_VALID !== o_VALID_ref) begin
          o_VALID_testFailure <= 1'b1;
          $display("ERROR in o_VALID at time %t : Expected '%h' Actual '%h'", $time,
o_VALID_ref, o_VALID);
        end
      end
    end
  assign o_MAGNITUDE_ignCntDone = o_MAGNITUDE_chkcnt != 6'b101111;
  assign o_MAGNITUDE_needToCount = ce_out & o_MAGNITUDE_ignCntDone;
  // Count limited, Unsigned Counter
  //  initial value   = 0
  //  step value      = 1
  //  count to value  = 47
```

```
  always @(posedge clk or posedge reset)
    begin : o_MAGNITUDE_IgnoreDataChecking_process
      if (reset == 1'b1) begin
        o_MAGNITUDE_chkcnt <= 6'b000000;
      end
      else begin
        if (o_MAGNITUDE_needToCount) begin
          if (o_MAGNITUDE_chkcnt >= 6'b101111) begin
            o_MAGNITUDE_chkcnt <= 6'b000000;
          end
          else begin
            o_MAGNITUDE_chkcnt <= o_MAGNITUDE_chkcnt + 6'b000001;
          end
        end
      end
    end
  assign o_MAGNITUDE_chkenb = o_MAGNITUDE_chkcnt == 6'b101111;
  assign o_MAGNITUDE_chkdata = ce_out & o_MAGNITUDE_chkenb;
  assign #1 o_MAGNITUDE_addr_delay_1 = o_VALID_addr;
  // Data source for o_MAGNITUDE_expected
  initial
    begin : o_MAGNITUDE_expected_fileread
      fp_o_MAGNITUDE_expected = $fopen("o_MAGNITUDE_expected.dat", "r");
      status_o_MAGNITUDE_expected = $rewind(fp_o_MAGNITUDE_expected);
    end

  always @(o_MAGNITUDE_addr_delay_1, ce_out, tb_enb_delay)
    begin
      if (tb_enb_delay == 0) begin
        o_MAGNITUDE_expected <= 32'bx;
      end
      else if (ce_out == 1) begin
        status_o_MAGNITUDE_expected     =     $fscanf(fp_o_MAGNITUDE_expected,     "%h",
o_MAGNITUDE_expected);
      end
    end
```

```
  assign o_MAGNITUDE_ref = o_MAGNITUDE_expected;


  always @(posedge clk or posedge reset)
    begin : o_MAGNITUDE_checker
      if (reset == 1'b1) begin
        o_MAGNITUDE_testFailure <= 1'b0;
      end
      else begin
        if   (o_MAGNITUDE_chkdata   ==   1'b1   &&   !isFloatSingleEpsEqual(o_MAGNITUDE,
o_MAGNITUDE_ref, 9.9999999999999995e-08)) begin
          o_MAGNITUDE_testFailure <= 1'b1;
          $display("ERROR in o_MAGNITUDE at time %t : Expected '%h' Actual '%h'", $time,
o_MAGNITUDE_ref, o_MAGNITUDE);
        end
      end
    end
  assign o_PHASE_ignCntDone = o_PHASE_chkcnt != 6'b101111;
  assign o_PHASE_needToCount = ce_out & o_PHASE_ignCntDone;
  // Count limited, Unsigned Counter
  //  initial value   = 0
  //  step value      = 1
  //  count to value  = 47
  always @(posedge clk or posedge reset)
    begin : o_PHASE_IgnoreDataChecking_process
      if (reset == 1'b1) begin
        o_PHASE_chkcnt <= 6'b000000;
      end
      else begin
        if (o_PHASE_needToCount) begin
          if (o_PHASE_chkcnt >= 6'b101111) begin
            o_PHASE_chkcnt <= 6'b000000;
          end
          else begin
            o_PHASE_chkcnt <= o_PHASE_chkcnt + 6'b000001;
          end
        end
      end
    end
```

```
  assign o_PHASE_chkenb = o_PHASE_chkcnt == 6'b101111;
  assign o_PHASE_chkdata = ce_out & o_PHASE_chkenb;
  assign #1 o_PHASE_addr_delay_1 = o_VALID_addr;
  // Data source for o_PHASE_expected
  initial
    begin : o_PHASE_expected_fileread
      fp_o_PHASE_expected = $fopen("o_PHASE_expected.dat", "r");
      status_o_PHASE_expected = $rewind(fp_o_PHASE_expected);
    end

  always @(o_PHASE_addr_delay_1, ce_out, tb_enb_delay)
    begin
      if (tb_enb_delay == 0) begin
        o_PHASE_expected <= 32'bx;
      end
      else if (ce_out == 1) begin
        status_o_PHASE_expected = $fscanf(fp_o_PHASE_expected, "%h", o_PHASE_expected);
      end
    end
  assign o_PHASE_ref = o_PHASE_expected;
  always @(posedge clk or posedge reset)
    begin : o_PHASE_checker
      if (reset == 1'b1) begin
        o_PHASE_testFailure <= 1'b0;
      end
      else begin
        if (o_PHASE_chkdata == 1'b1 && !isFloatSingleEpsEqual(o_PHASE, o_PHASE_ref,
9.9999999999999995e-08)) begin
          o_PHASE_testFailure <= 1'b1;
          $display("ERROR in o_PHASE at time %t : Expected '%h' Actual '%h'", $time,
o_PHASE_ref, o_PHASE);
        end
      end
    end

  assign testFailure = o_PHASE_testFailure | (o_VALID_testFailure |
o_MAGNITUDE_testFailure);
  always @(posedge clk)
```

```
    begin : completed_msg
      if (snkDone == 1'b1) begin
        if (testFailure == 1'b0) begin
          $display("**************TEST COMPLETED (PASSED)**************");
        end
        else begin
          $display("**************TEST COMPLETED (FAILED)**************");
        end
      end
    end
endmodule  // Subsystem_tb
```