Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Кафедра електронно обчислювальної апаратури

Лабораторна робота №1

з курсу «Апаратні прискорювачі на мікросхемах програмованої логіки»

Виконав:

Студент 3 курсу

Групи ДК-02

Садко Вячеслав

Варіант 10

Київ 2022

ЗМІСТ

Завдання

1. Реалізація в Simulink підсистеми, що розраховує функцію:

$$Y = W0*X0 + W1*X1 + W2*X2 + W3*X3$$

Типи даних входів: int8

Тип даних виходу: int16

На входах і виході поставити регістри (блок затримки на 1 такт)

Налаштування блоків “Uniform Random Number” згідно номеру варіанту:

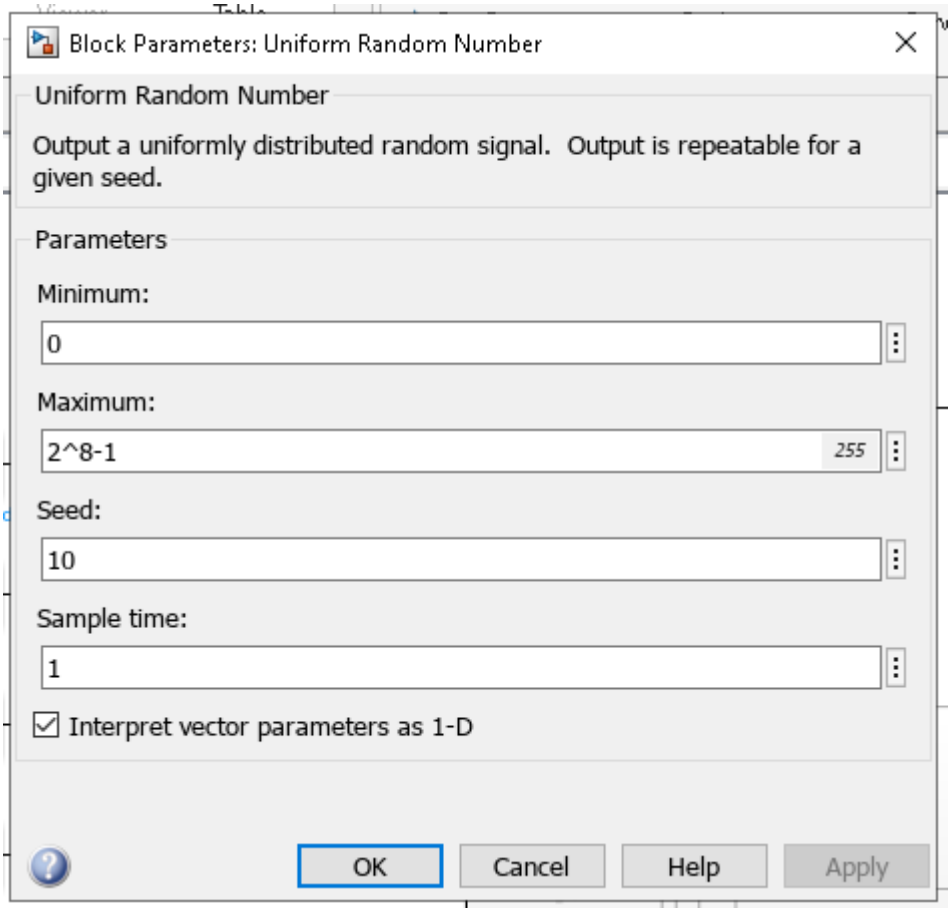

Рис. 1 Налаштування першого по порядку блоку

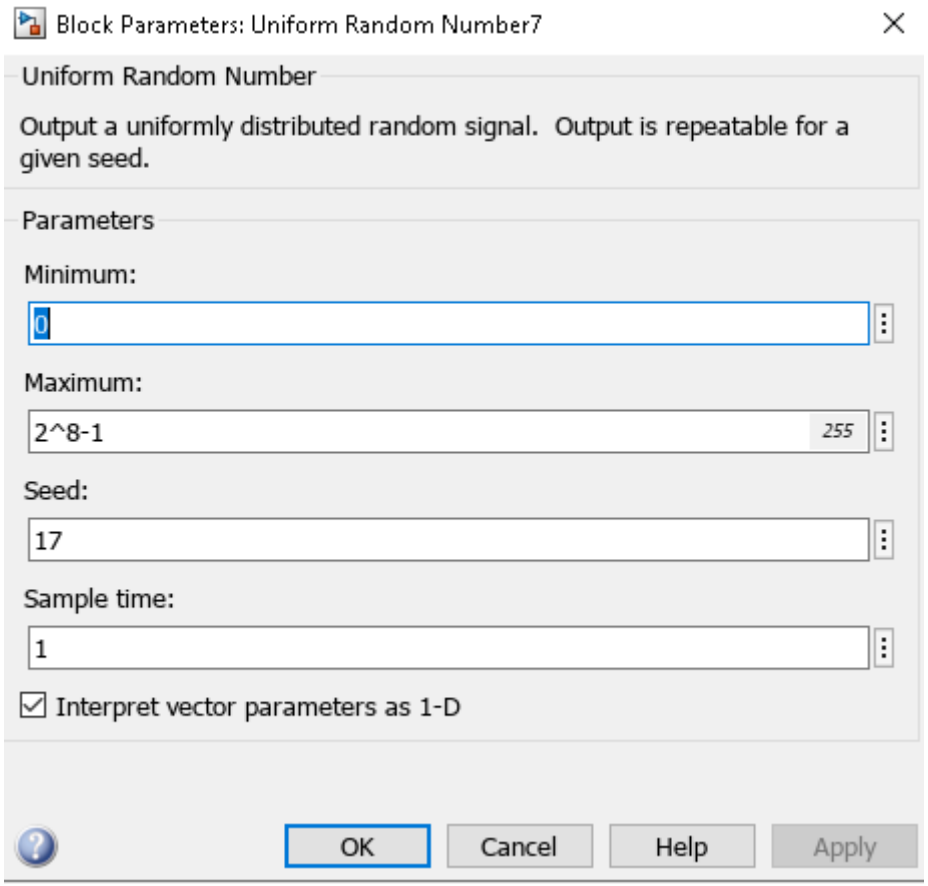

Рис. 2 Налаштування останнього по порядку блоку

Створюємо в Simulink загальну схему згідно умові:

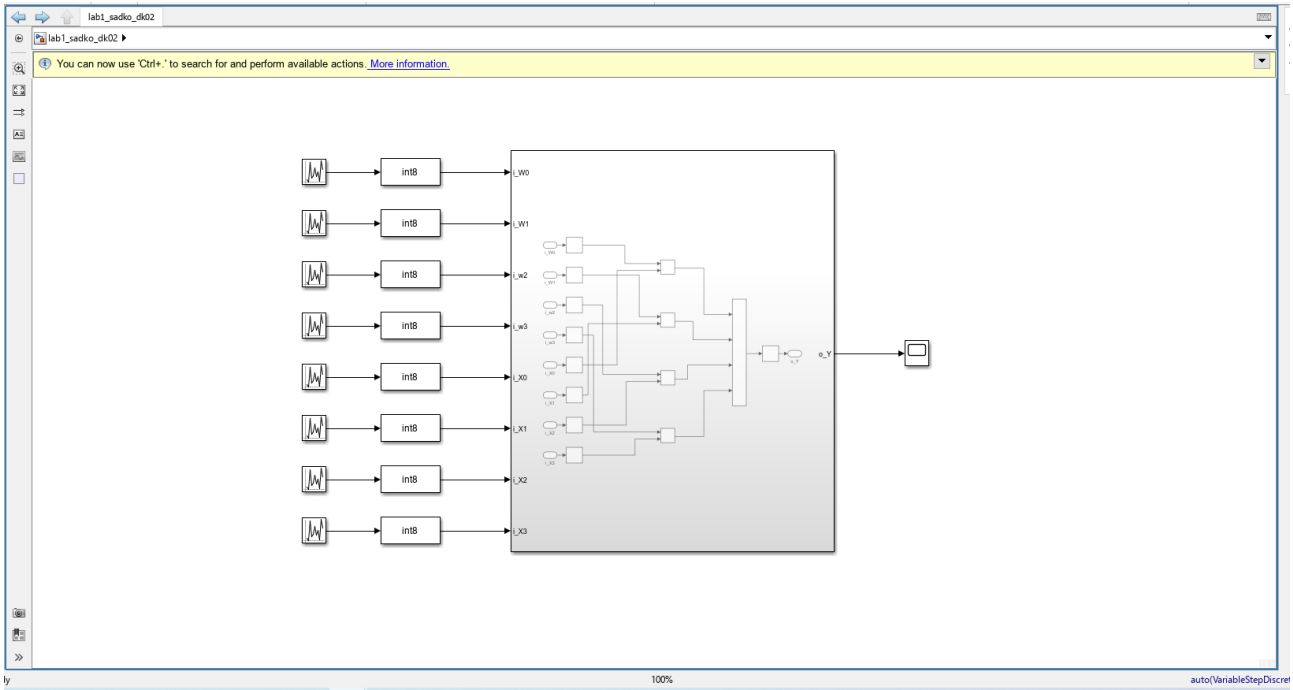

Рис. 3 Загальна схема

З схеми, яка проводить обчислення заданої функції, утворюємо підсистему:

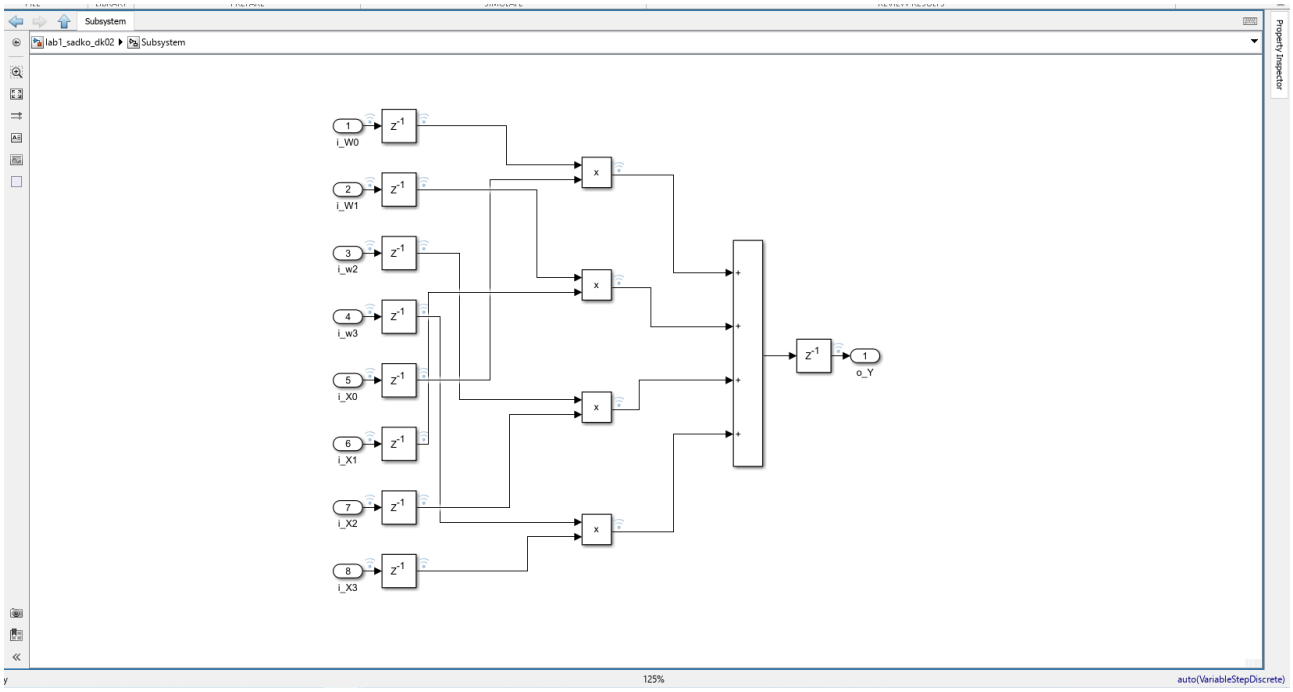

Рис. 4 Підсистема загальної схеми

Налаштування компонентів підсистеми:

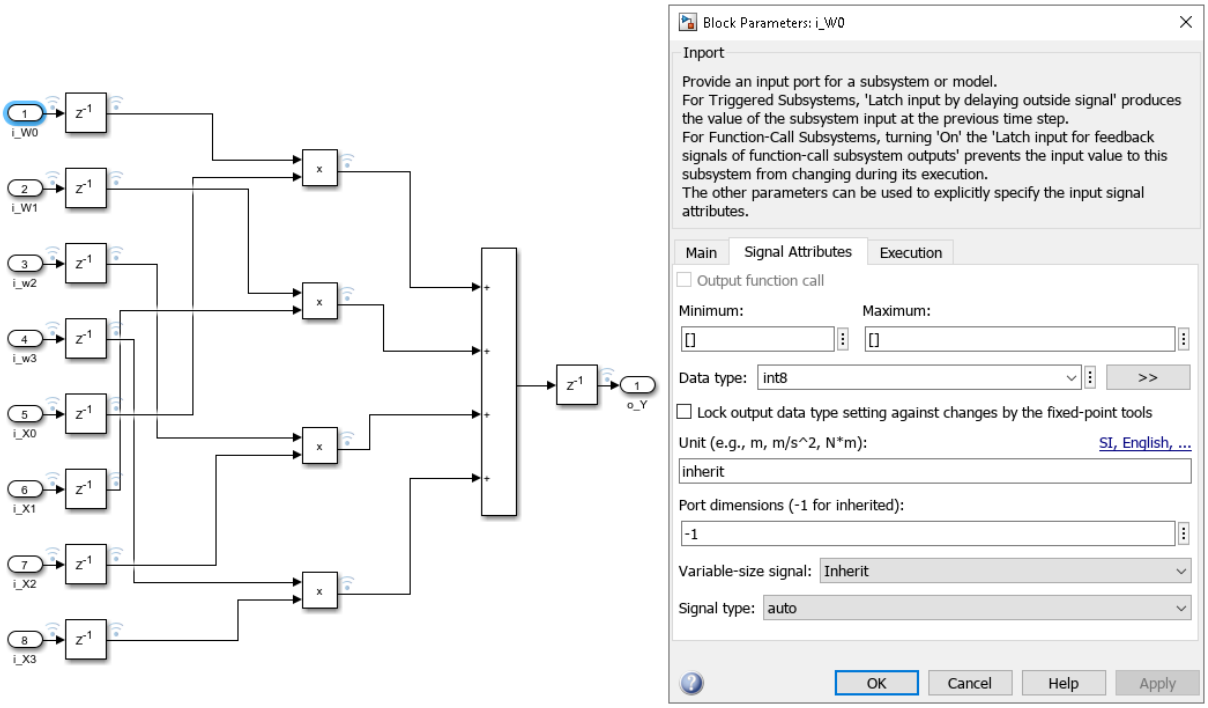

Рис. 5 Налаштування для входів i_W

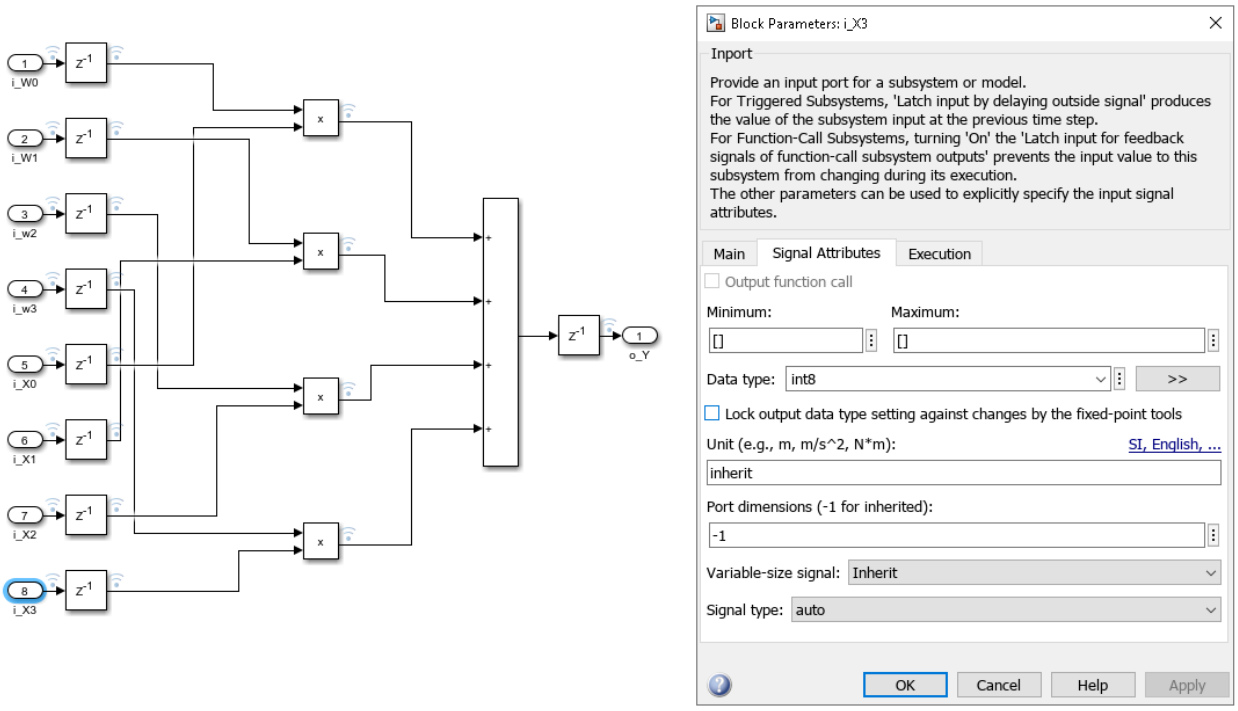

Рис. 6 Налаштування для входів i_X

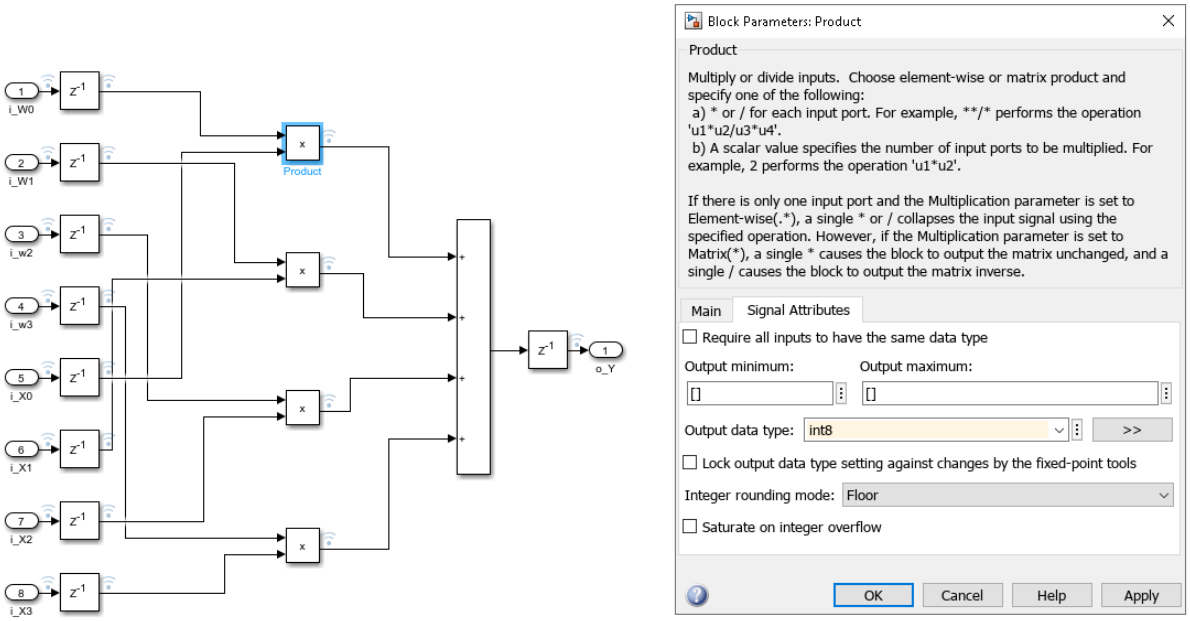

Рис. 7 Налаштування для блоків Product

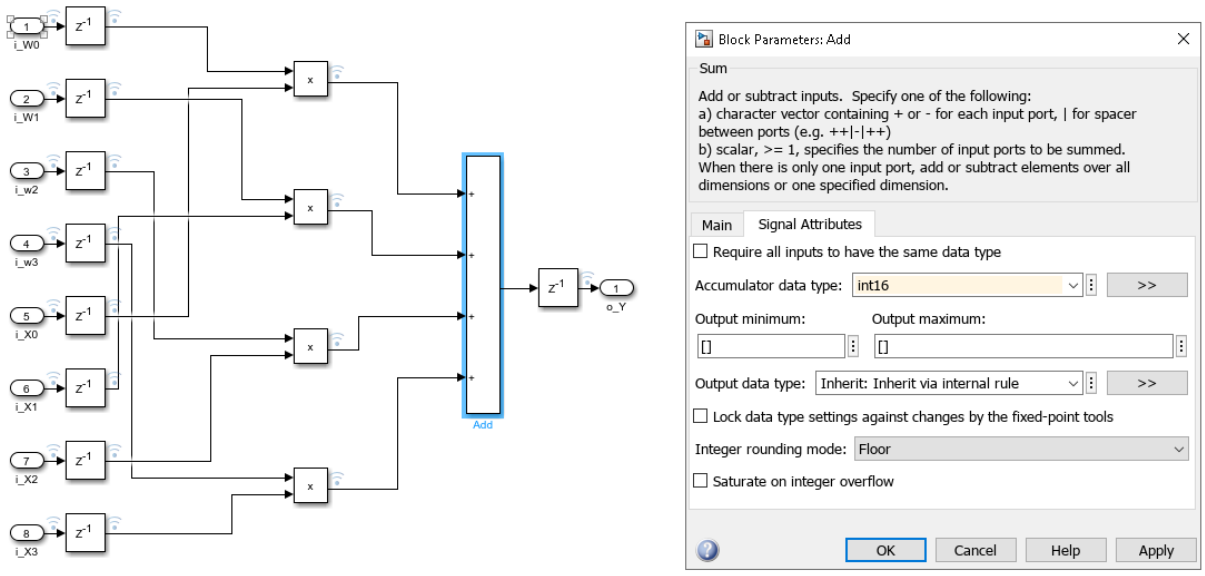

Рис. 8 Налаштування для блоків Add

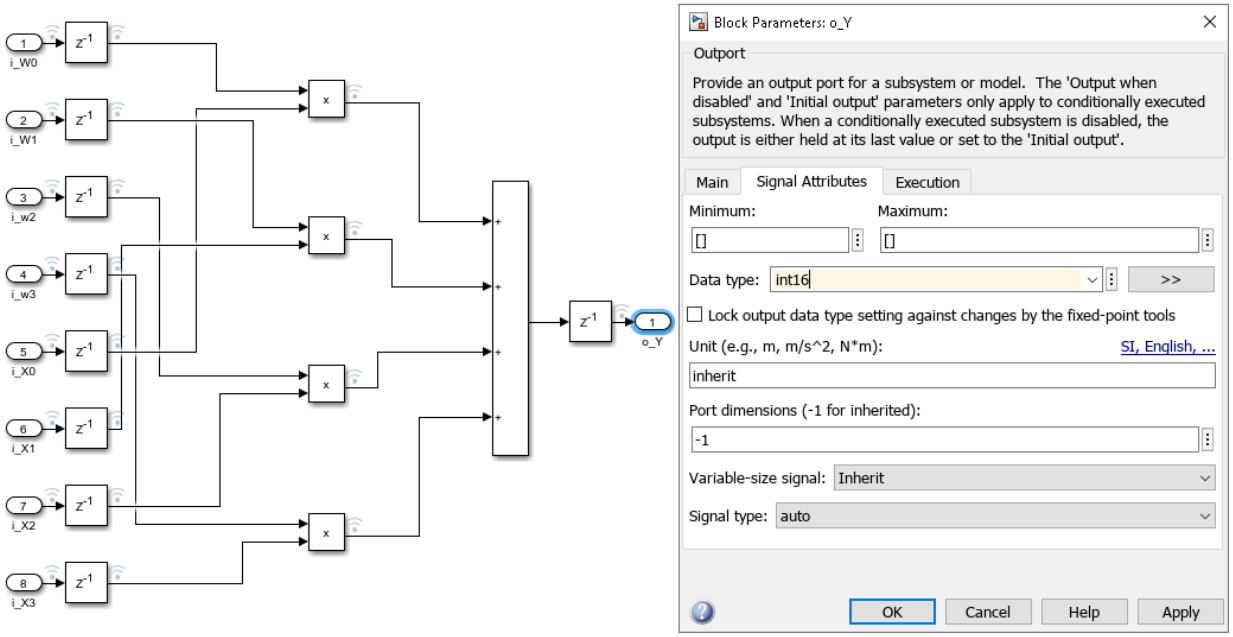

Рис. 9 Налаштування для виходу o_Y

2. Перегляд в логічному аналазаторі даних на входах і на виході створеної підсистеми у знаковому десятковому поданні

Після компіляції системи в логічному аналізаторі можна відстежити результат. На рис. 10 можна побачити, що Delay0 утворює затримку на два такти.

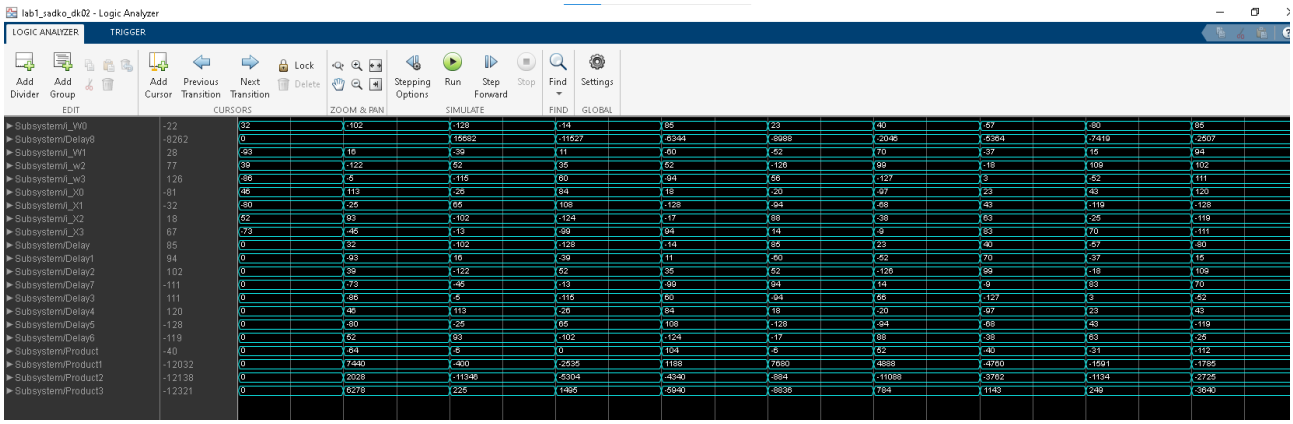

Рис. 10 Результат симуляції схеми

3. Генерування коду на Verilog та синтез згенерованого коду в Quartus для створеної підсистеми

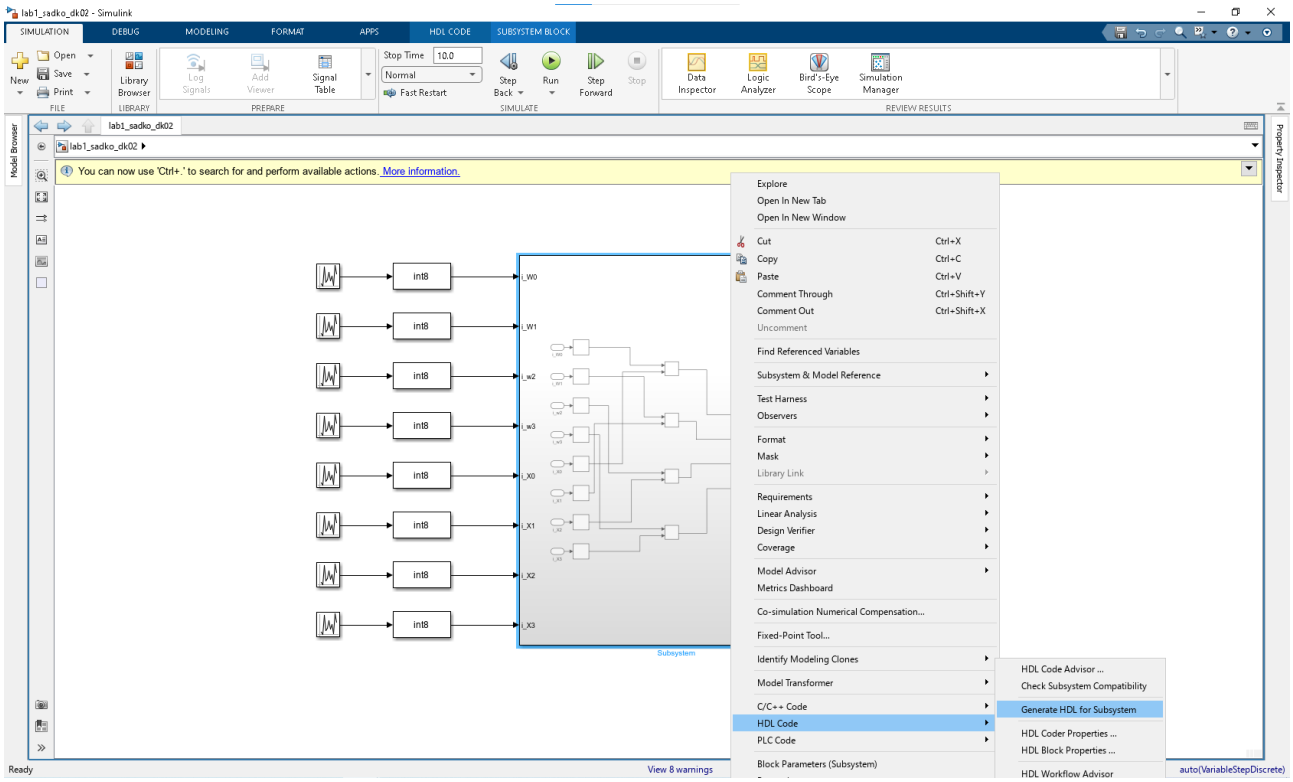

Рис. 11 Генерація коду на Verilog для підсистеми

Лістинг 3.1

```
// 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_W0,

           i_W1,

           i_w2,

           i_w3,
```

```
          i_X0,
          i_X1,
          i_X2,
          i_X3,
          ce_out,
          o_Y);


input   clk;
input   reset;
input   clk_enable;
input   signed [7:0] i_W0;  // int8
input   signed [7:0] i_W1;  // int8
input   signed [7:0] i_w2;  // int8
input   signed [7:0] i_w3;  // int8
input   signed [7:0] i_X0;  // int8
input   signed [7:0] i_X1;  // int8
input   signed [7:0] i_X2;  // int8
input   signed [7:0] i_X3;  // int8
output  ce_out;
output  signed [15:0] o_Y;  // int16
wire enb;
reg signed [7:0] Delay_out1;  // int8
reg signed [7:0] Delay1_out1;  // int8
reg signed [7:0] Delay2_out1;  // int8
reg signed [7:0] Delay3_out1;  // int8
reg signed [7:0] Delay4_out1;  // int8
wire signed [15:0] Product_mul_temp;  // sfix16
wire signed [7:0] Product_out1;  // int8
```

```
reg signed [7:0] Delay5_out1;  // int8

wire signed [15:0] Product1_out1;  // int16

wire signed [15:0] Add_stage2_add_temp;  // sfix16

wire signed [15:0] Add_stage2_1;  // sfix16

wire signed [16:0] Add_op_stage1;  // sfix17

reg signed [7:0] Delay6_out1;  // int8

wire signed [15:0] Product2_out1;  // int16

wire signed [15:0] Add_stage3_add_cast;  // sfix16

wire signed [15:0] Add_stage3_add_temp;  // sfix16

wire signed [17:0] Add_op_stage2;  // sfix18

reg signed [7:0] Delay7_out1;  // int8

wire signed [15:0] Product3_out1;  // int16

wire signed [15:0] Add_stage4_add_cast;  // sfix16

wire signed [15:0] Add_out1;  // int16

reg signed [15:0] Delay8_out1;  // int16

assign enb = clk_enable;

always @(posedge clk or posedge reset)

  begin : Delay_process

    if (reset == 1'b1) begin

      Delay_out1 <= 8'sb00000000;

    end

    else begin

      if (enb) begin

        Delay_out1 <= i_W0;

      end

    end

  end

always @(posedge clk or posedge reset)
```

```
  begin : Delay1_process
    if (reset == 1'b1) begin
      Delay1_out1 <= 8'sb00000000;
    end
    else begin
      if (enb) begin
        Delay1_out1 <= i_W1;
      end
    end
  end
always @(posedge clk or posedge reset)
  begin : Delay2_process
    if (reset == 1'b1) begin
      Delay2_out1 <= 8'sb00000000;
    end
    else begin
      if (enb) begin
        Delay2_out1 <= i_w2;
      end
    end
  end
always @(posedge clk or posedge reset)
  begin : Delay3_process
    if (reset == 1'b1) begin
      Delay3_out1 <= 8'sb00000000;
    end
    else begin
      if (enb) begin
```

```
        Delay3_out1 <= i_w3;
      end
    end
  end
always @(posedge clk or posedge reset)
  begin : Delay4_process
    if (reset == 1'b1) begin
      Delay4_out1 <= 8'sb00000000;
    end
    else begin
      if (enb) begin
        Delay4_out1 <= i_X0;
      end
    end
  end
assign Product_mul_temp = Delay_out1 * Delay4_out1;
assign Product_out1 = Product_mul_temp[7:0];
always @(posedge clk or posedge reset)
  begin : Delay5_process
    if (reset == 1'b1) begin
      Delay5_out1 <= 8'sb00000000;
    end
    else begin
      if (enb) begin
        Delay5_out1 <= i_X1;
      end
    end
  end
```

```
assign Product1_out1 = Delay1_out1 * Delay5_out1;

assign Add_stage2_1 = {{8{Product_out1[7]}}, Product_out1};

assign Add_stage2_add_temp = Add_stage2_1 + Product1_out1;

assign Add_op_stage1 = {Add_stage2_add_temp[15], Add_stage2_add_temp};

always @(posedge clk or posedge reset)
  begin : Delay6_process
    if (reset == 1'b1) begin
      Delay6_out1 <= 8'sb00000000;
    end
    else begin
      if (enb) begin
        Delay6_out1 <= i_X2;
      end
    end
  end
assign Product2_out1 = Delay2_out1 * Delay6_out1;

assign Add_stage3_add_cast = Add_op_stage1[15:0];

assign Add_stage3_add_temp = Add_stage3_add_cast + Product2_out1;

assign Add_op_stage2 = {{2{Add_stage3_add_temp[15]}}, Add_stage3_add_temp};

always @(posedge clk or posedge reset)
  begin : Delay7_process
    if (reset == 1'b1) begin
      Delay7_out1 <= 8'sb00000000;
    end
    else begin
      if (enb) begin
        Delay7_out1 <= i_X3;
      end
```

```
      end
    end
  assign Product3_out1 = Delay3_out1 * Delay7_out1;
  assign Add_stage4_add_cast = Add_op_stage2[15:0];
  assign Add_out1 = Add_stage4_add_cast + Product3_out1;
  always @(posedge clk or posedge reset)
    begin : Delay8_process
      if (reset == 1'b1) begin
        Delay8_out1 <= 16'sb0000000000000000;
      end
      else begin
        if (enb) begin
          Delay8_out1 <= Add_out1;
        end
      end
    end
  assign o_Y = Delay8_out1;
  assign ce_out = clk_enable;
endmodule  // Subsystem
```

4.Результат синтезу в RTL Viewer у квартусі та визначення апаратних витрат:

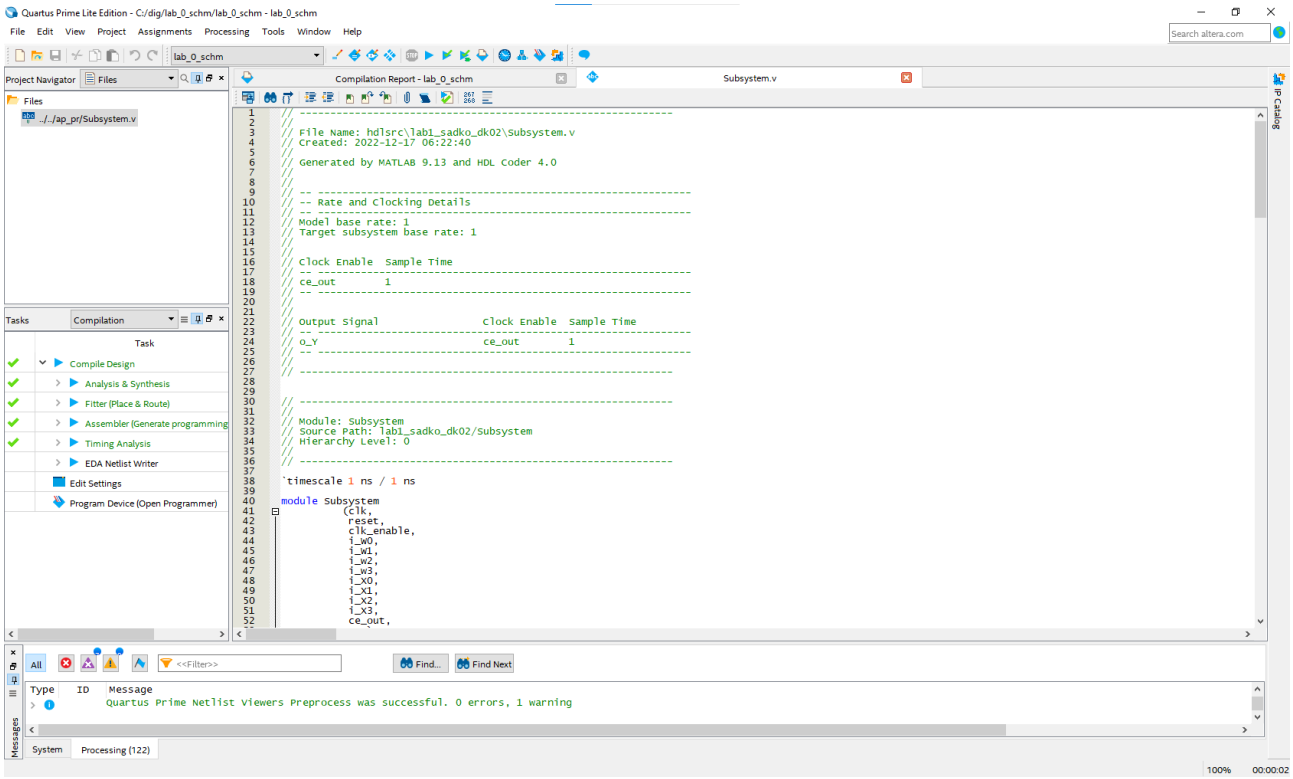

Рис. 12 Результат компіляції згенерованого коду у Quartus Prime Lite

За рис. 13 можна побачити, що у нас все правильно було згенеровано, особлива увага на розрядності, на входах по 8-біт, на Mult, Add та виході 16-біт, як і було задано в умові. На рис. 14 після компіляції файлу, як результат, можемо відстежити які апаратні витрати на цю підсистему.

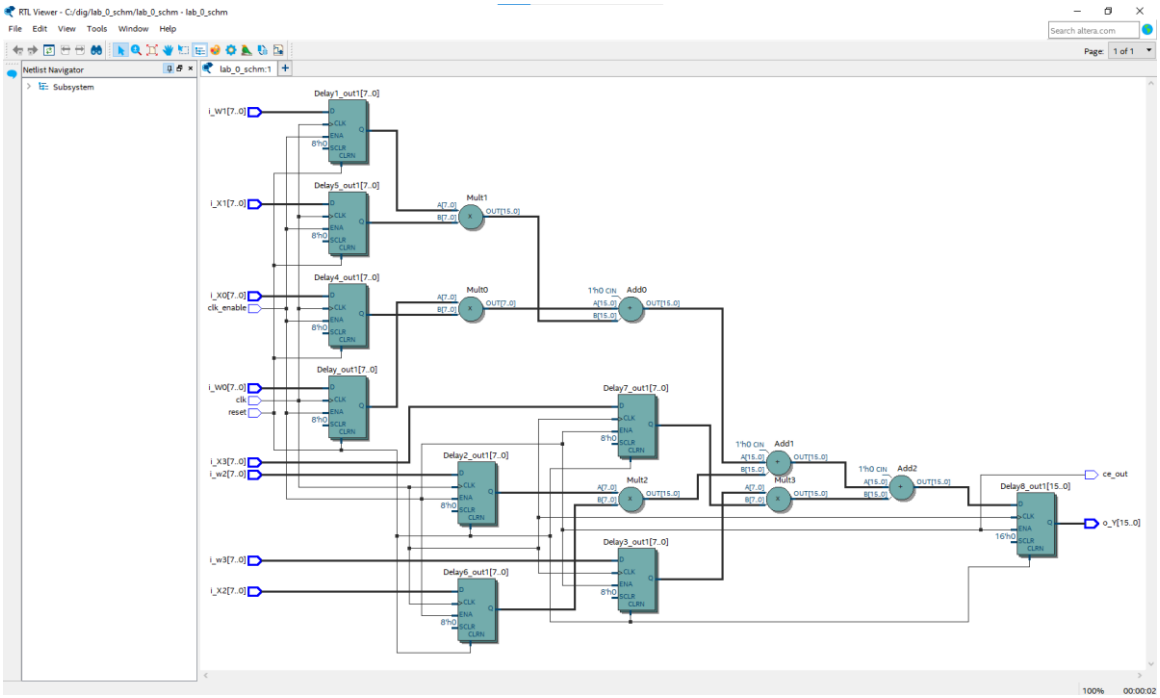

Рис. 13 Синтез згенерованого коду через RTL viewer

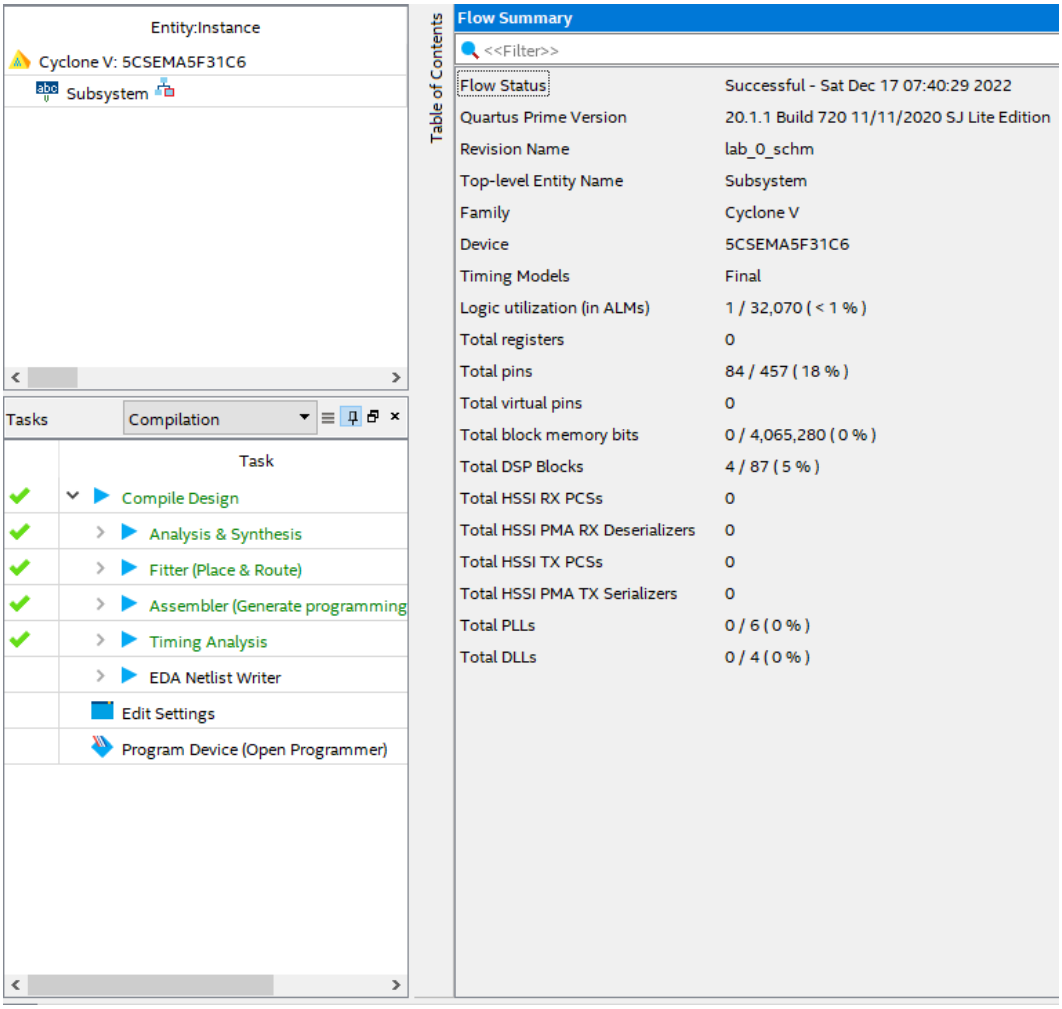

Рис. 14 Результат компіляції та дані про апаратні витрати підсистеми

5. Створення тестбенч файлу в Matlab для створеної підсистеми і результат симуляції тестбенча в Modelsim

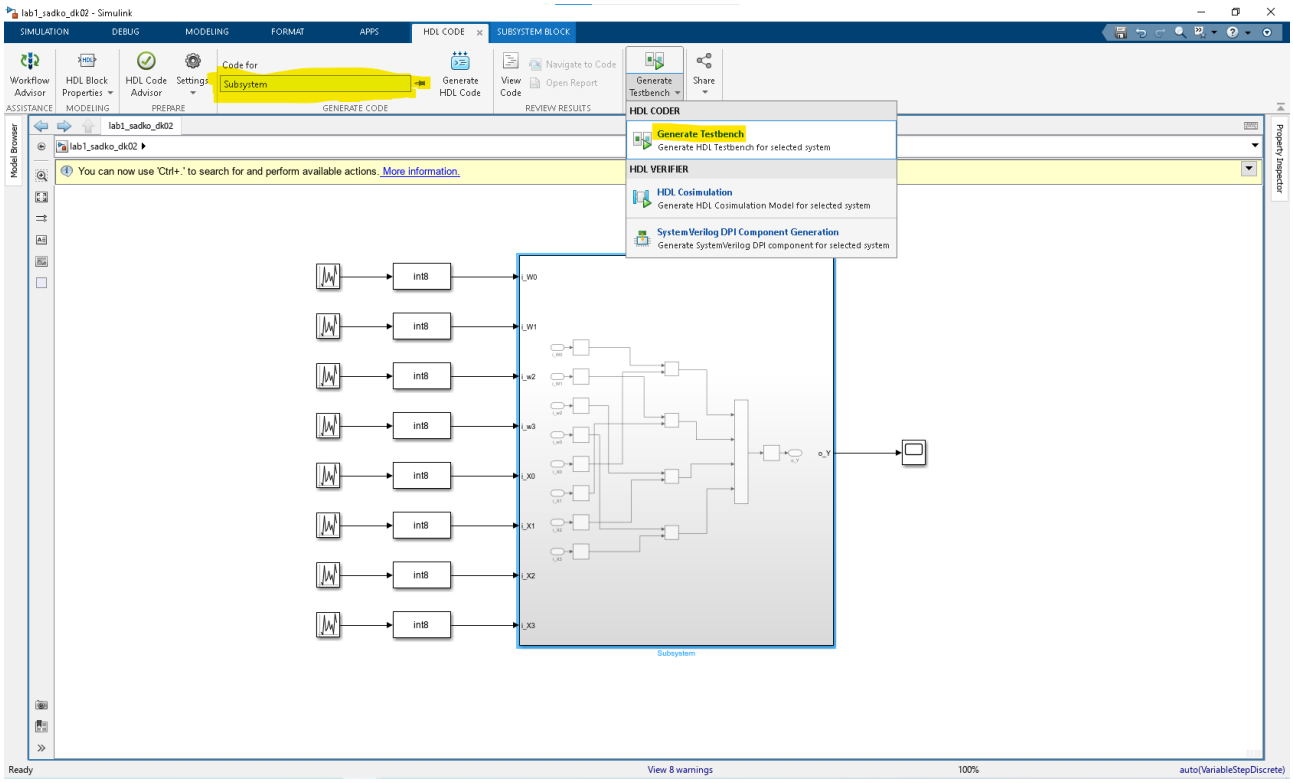

Рис. 15 Генерування тестбенч файлу для згенерованого коду

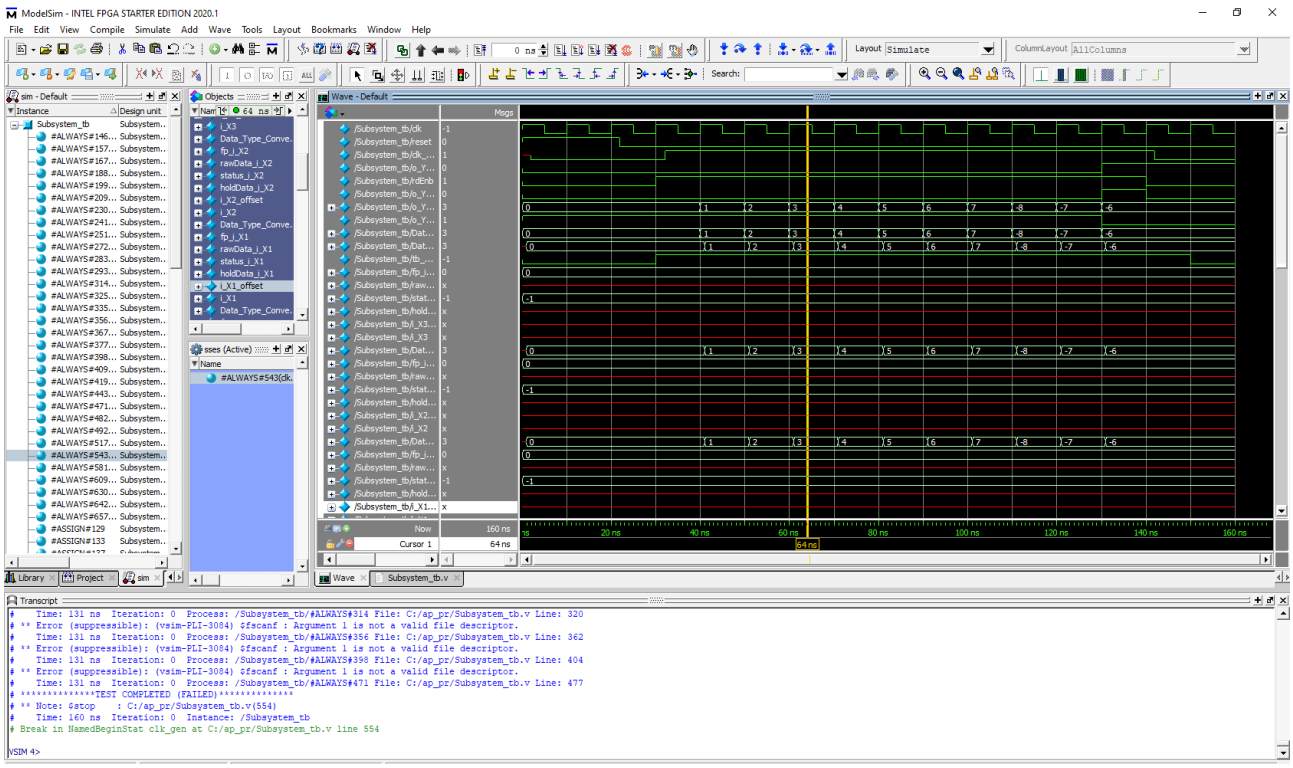

Рис. 16 Результат компіляції та запуску тестбенч файлу

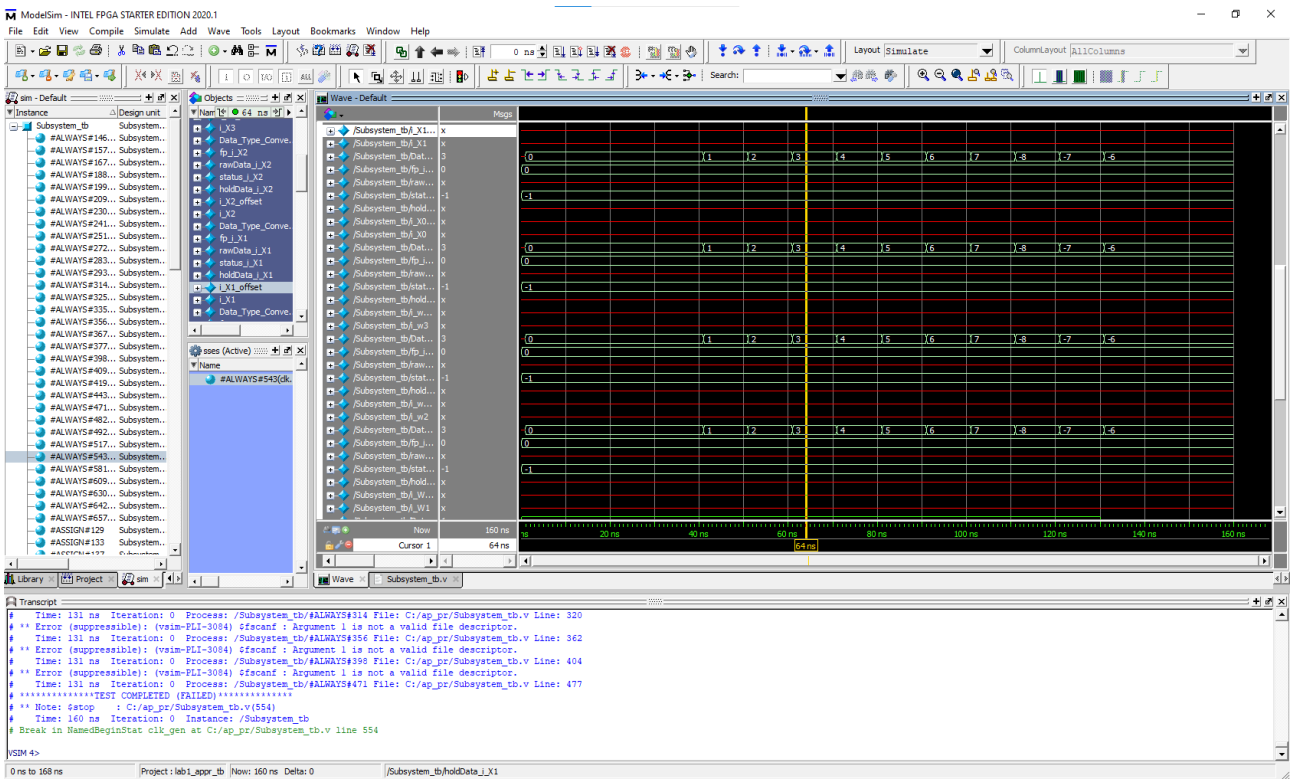

Рис. 17 Результат компіляції та запуску тестбенч файлу

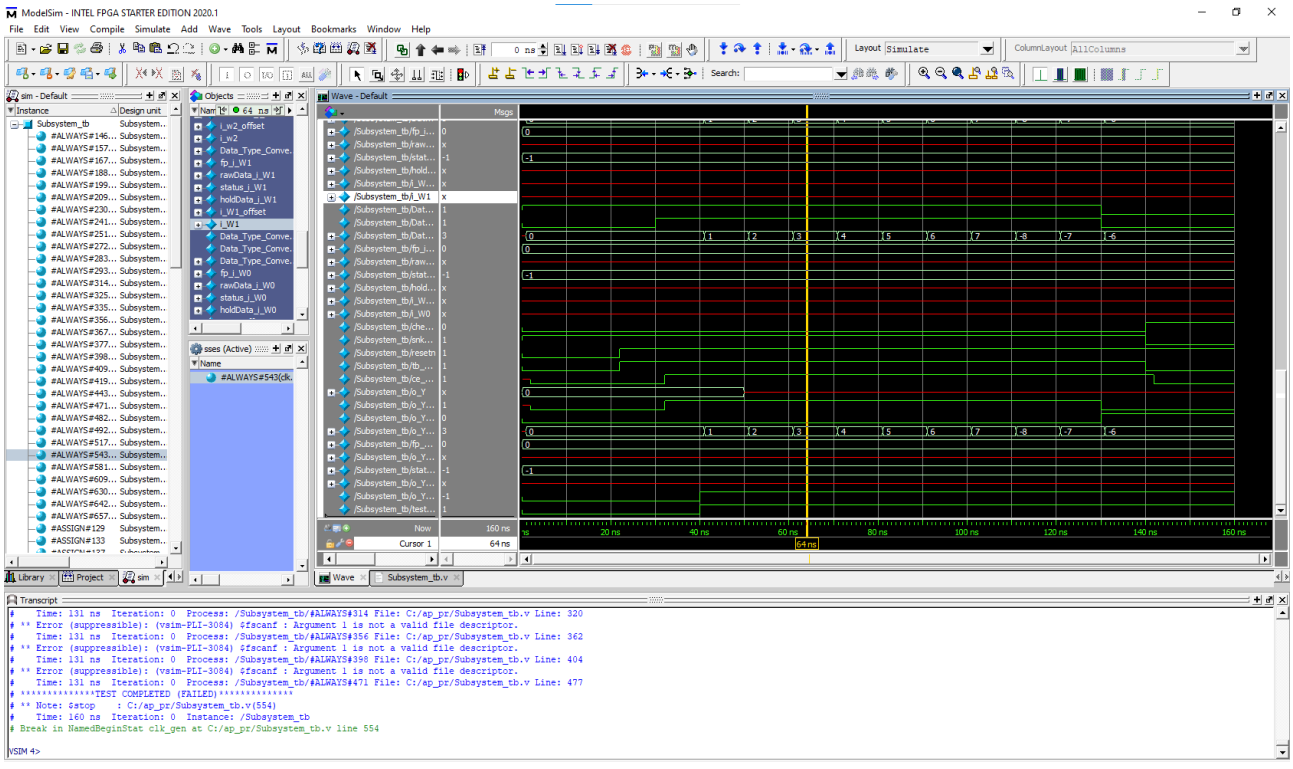

Рис. 18 Результат компіляції та запуску тестбенч файлу

Висновок:

В процесі виконання лабораторної роботи ознайомився з налаштуванням Matlab та Simulink, склавши систему наведену на рис. 3 та підсистему на рис. 4. В результаті зкомпілюваваши цю систему отримав результати обчислень на рис. 10.

Додатково провів окреме дослідження, згенерувавши Verilog код підсистеми і в Quartus отримав досить корисні дані про апартні витрати підсистеми на рис. 14 , це дуже важливо для проведень оптимізацій різних систем в процесі розробки і тестування.

Також був згенерований ще тестбенч файл для вищезгаданого коду, це дало змогу окремо протестувати підсистему користуючись іншим програмним забезпеченням.