Федеральное государственное бюджетное образовательное учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт: ИРЭ Кафедра: Электроники и Наноэлектроники

Направление подготовки: 11.04.04 Электроника и Наноэлектроника

ОТЧЕТ по практике

Наименование практики: Производственная практика: научно-исследовательская работа

СТУДЕНТ

/. Маринин Н.С /
(подпись) (*Фамилия и инициалы*)

Группа Эр-05м-21
(номер учебной группы)

ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ ПО ПРАКТИКЕ

(отлично, хорошо, удовлетворительно, неудовлетворительно, зачтено, не зачтено)

/ /
(подпись) (*Фамилия и инициалы члена комиссии*)

/ /
(подпись) *(Фамилия и инициалы члена комиссии)*

Москва
2023

Оглавление

1. Методы

Для реализации Фурье преобразования, то есть для перевода данных из временного в частное измерение был использован алгоритм Гёрцеля. Данный алгоритм позволяет произвести расчет не полного ДПФ, а лишь фиксированного количества спектральных отсчетов.

По алгоритму спектральный отсчет S(k) равен:

$$S(k) = y_{N-1}(k) = W_N^{-k} v(N-1) - v(N-2),$$

Где u - промежуточные значения, которые рассчитываются итерационно:

$$v(r) = s(r) + 2\cos\left(2\pi\frac{k}{N}\right) v(r-1) - v(r-2).$$

W – поворотный коэффициент

$$W_N^{nk} = \exp\left(-j\frac{2\pi}{N}nk\right); \quad k = 0 \ldots N-1.$$

Таким образом, для расчета потребуется N вещественных умножений, а не комплексных. Также требуется одно комплексное умножение на W_N^{-k} на последней итерации.

Также заранее необходимо будет рассчитать sin и cos для нужного отсчёта.

Расчёт будет проводится с помощью алгоритма CORDIC.

Алгоритм был придуман для поворота вектора на плоскости с помощью операций «сдвиг регистра вправо» и «сложение регистров». Другими словами — для реализации поворота вектора аппаратно (при помощи цифровой схемотехники).

Суть заключается в последовательном, итерационном повороте вектора на заранее рассчитанный угол, atan которого кратен степени 2 (для операции сдвига).

С каждой итерацией угол поворота уменьшается, достигая необходимой точности расчета.

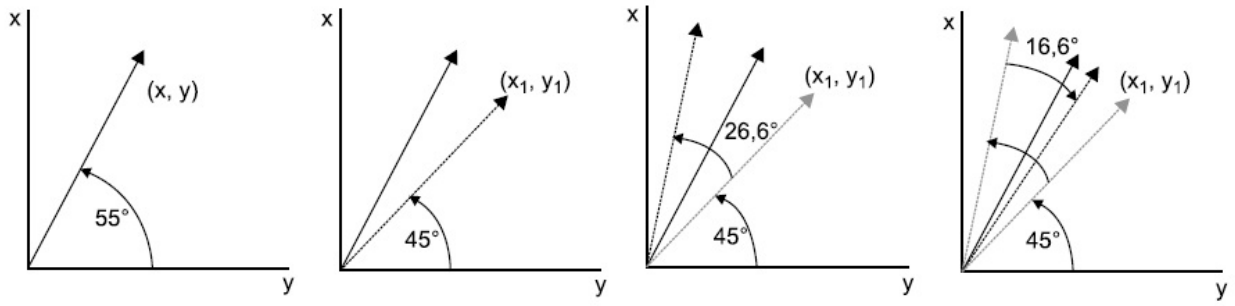

Рисунок 1 CORDIC алгоритм

Координаты x_1 и y_1 вычисляются по формулам:

$$x_1 = \cos(\varphi)\times(x_0 - y_0\times\tan(\varphi)),$$

$$y_1 = \cos(\varphi)\times(y_0 + x_0\times\tan(\varphi)).$$

Умножение на tan заменяется сдвигом, а cos заменяется коэффициентом масштабирования K, который рассчитывается заранее в зависимости от количества итераций.

$$K(n) = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} 1/\sqrt{1+2^{-2i}}$$

На основе данных двух алгоритмов был разработан модуль для вычисления спектров сигнала.

Для примера количество отсчётов NS – 1000, число рассчитываемых частот NF – 11 (6, 60, 80, 100, 200, 300, 400, 500, 600, 800, 1000) Гц, частота дискретизации – 2000 Гц (для данных частот больше не нужно, для больших частот соответственно будет применятся большая частота дискретизации).

1.1. Пример 1

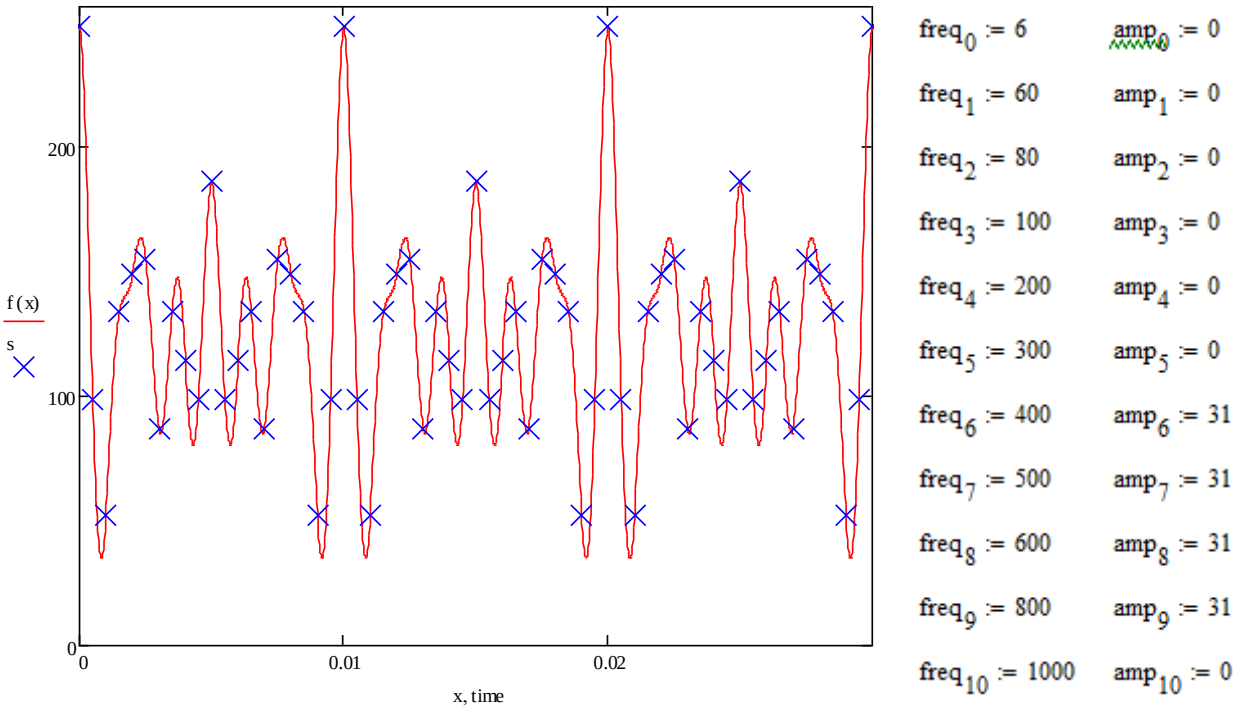

Рисунок 2 Входной сигнал и его параметры

Были получены следующие результаты:

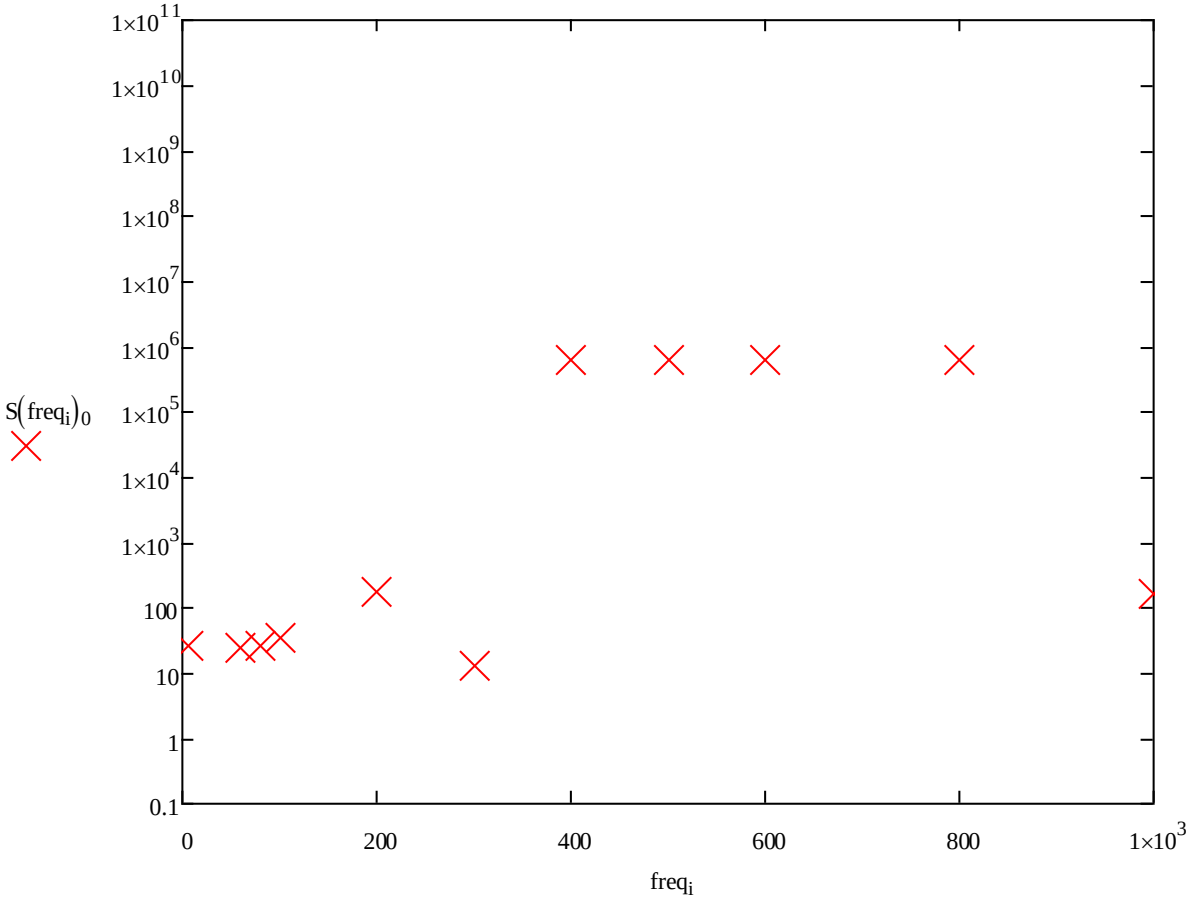

Рисунок 3 Значения полученные в системе matchcad

```
sim:/tb_FourierTransform/data_o @ 13585 ps
10 : 0000009e 00097dd4
 8 : 000975fe 00097445
 6 : 00095b62 0000000c
 4 : 000000b2 00000022
 2 : 00000019 00000018
 0 : 0000000c
```

Рисунок 4 Значения полученные при симуляции модуля в modelsim. Индексы соответствуют индексам частот на рис.4, значения записаны в формате 32.0.

1.2. Пример 2

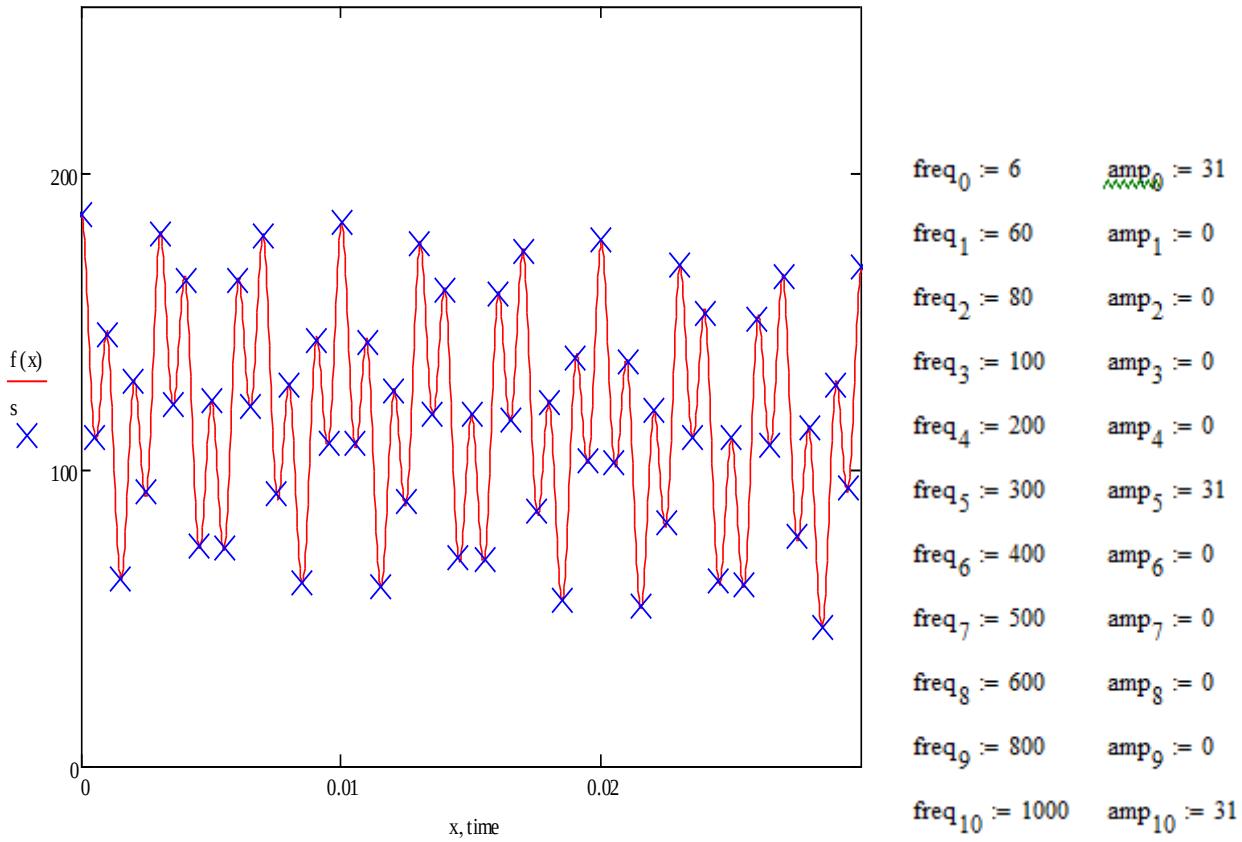

Рисунок 5 Входной сигнал и его параметры

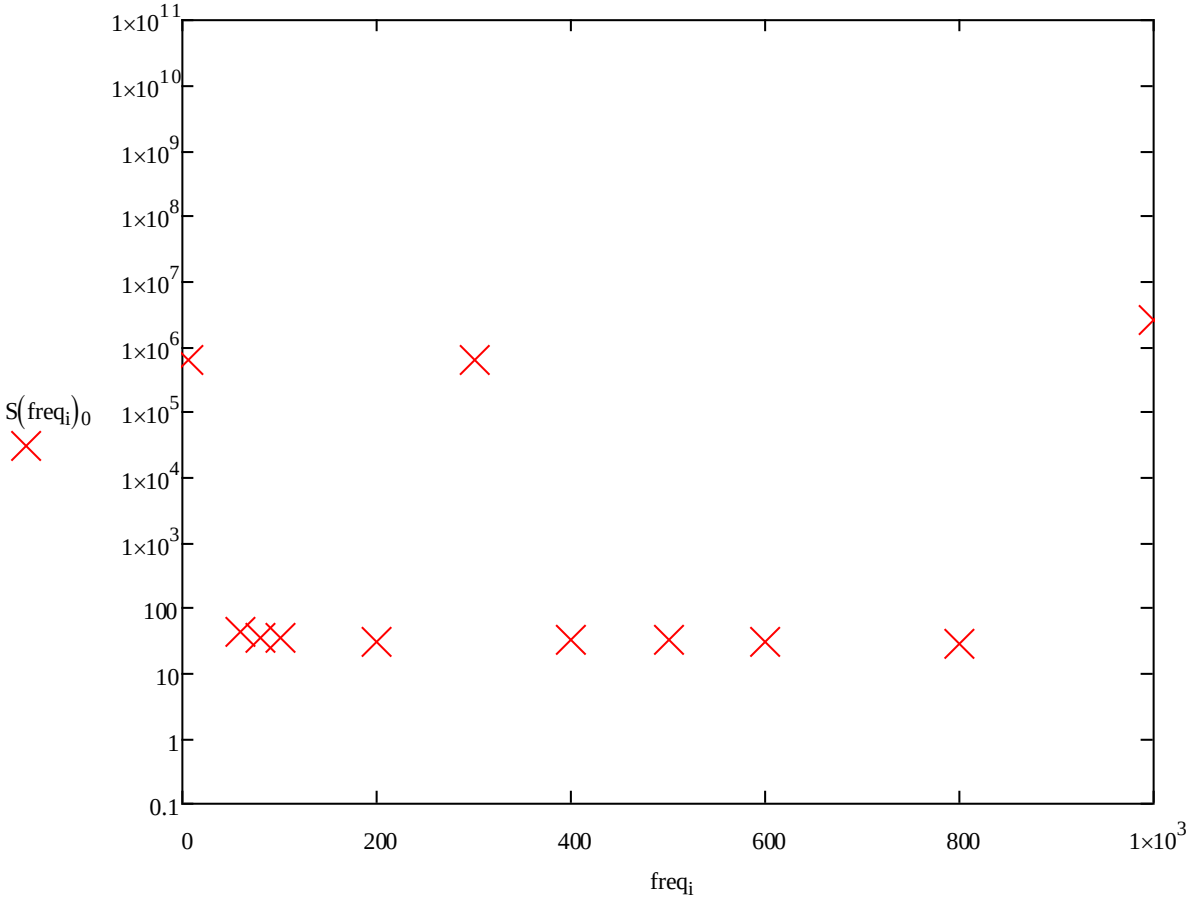

Рисунок 6 Значения полученные в системе matchcad

```
sim:/tb_FourierTransform/data_o @ 14502 ps
10 : 00261708 0000001b
 8 : 0000001f 00000020
 6 : 00000021 00096083
 4 : 0000001e 00000021
 2 : 00000021 00000029
 0 : 00095a33
```

Рисунок 7 Рисунок 4 Значения полученные при симуляции модуля в modelsim.

Ввиду погрешности и частоты 11-ой частоты, значения отличаются от полученных ранее, но в целом результат удовлетворителен.

2. Приложение

2.1. Топ уровень

```
module FourierTransform #(
  parameter NF = 11, // NUM_FREQ
  parameter NS = 10  // NUM_SAMPLE

) (
  // CLK&RST
  input                                rstn  ,
  input                                clk   ,
  // CTRL
  input                                cEn   ,
  input                                hEn   ,
  output logic        [NF-1:0]         ready ,
  // DATA
  input                          [7 :0] sample,
  output logic signed [NF-1:0][31:0] data_o
);

logic        [NF-1:0][31:0] freq_arr ; // (32.0)
logic signed [NF-1:0][31:0] angel_arr;
logic signed [NF-1:0][31:0] coefW_re ;
logic signed [NF-1:0][31:0] coefW_im ;
logic signed [NF-1:0][31:0] alpha    ;
logic signed         [31:0] data     ;

logic ready0     ;
logic ready1     ;
logic herzelEn   ;

assign herzelEn = ready1 && hEn;

Angel #(
  .NF(NF)
) u_Angel (
  .rstn   (rstn     ),
  .clk    (clk      ),
  .en     (cEn      ),
  .ready  (ready0   ),
  .freq_i (freq_arr ),
  .angel_o(angel_arr)
);
```

```
Cordic #(
  .NF(NF)
) u_Cordic (
  .rstn (rstn     ),
  .clk  (clk      ),
  .en   (ready0   ),
  .ready(ready1   ),
  .ang_i(angel_arr),
  .cos_o(coefW_re ),
  .sin_o(coefW_im ),
  .alpha(alpha    )
);

DataScale u_DataScale (
  .rstn  (rstn    ),
  .clk   (clk     ),
  .data_i(sample  ),
  .data_o(data    )
);

logic enn = 0;
always_ff @(posedge clk) if (herzelEn) enn <= 1;

genvar gvar;
generate
  for (gvar = 0; gvar < NF; gvar = gvar + 1) begin : herzel
    Herzel #(
      .NF(NF),
      .NS(NS)
    ) u_Herzel (
      .rstn   (rstn          ),
      .clk    (clk           ),
      .en     (enn           ),
      .ready  (ready[gvar]   ),
      .alpha_i(alpha[gvar]   ),
      .cW_re_i(coefW_re[gvar]),
      .cW_im_i(coefW_im[gvar]),
      .data_i (data          ),
      .data_o (data_o[gvar]  )
    );
  end
endgenerate

initial begin
  freq_arr[0 ] = 32'd6     ;
```

```
  freq_arr[1 ] = 32'd60    ;
  freq_arr[2 ] = 32'd80    ;
  freq_arr[3 ] = 32'd100   ;
  freq_arr[4 ] = 32'd200   ;
  freq_arr[5 ] = 32'd300   ;
  freq_arr[6 ] = 32'd400   ;
  freq_arr[7 ] = 32'd500   ;
  freq_arr[8 ] = 32'd600   ;
  freq_arr[9 ] = 32'd800   ;
  freq_arr[10] = 32'd1000  ;
end

endmodule
```

2.2. Herzel модуль

```
module Herzel #(
  parameter NF = 11,
  parameter NS = 10
)(
  // CLK&RST
  input                          rstn        ,
  input                          clk         ,
  // CTRL
  input                          en          ,
  output logic                   ready       ,
  // DATA
  input          signed [31:0] alpha_i   ,
  input          signed [31:0] cW_re_i   ,
  input          signed [31:0] cW_im_i   ,
  input          signed [31:0] data_i    ,
  output logic unsigned [31:0] data_o
);

logic signed   [63 :0] alpha       ;
logic signed   [63 :0] coefW_re    ;
logic signed   [63 :0] coefW_im    ;
logic signed   [63 :0] data        ;
logic signed   [63 :0] vm1         ;
logic signed   [63 :0] vm2         ;
logic signed   [127:0] vm1_alpha   ;
logic signed   [63 :0] vm1_alpha_32;
logic signed   [127:0] vm1_cW_re   ;
logic signed   [63 :0] vm1_cW_re_32;
logic signed   [63 :0] vm1_cW_re_m2;
logic signed   [127:0] vm1_cW_im   ;
logic signed   [63 :0] vm1_cW_im_32;
logic signed   [127:0] data_re     ;
logic unsigned [31 :0] data_re_32  ;
logic signed   [127:0] data_im     ;
logic unsigned [31 :0] data_im_32  ;
logic          [31 :0] indx1       ;
logic                  vmcw        ;

assign alpha     = {{13{alpha_i[31]}}, alpha_i[30:0], {20{1'b0}}}; //
20.44
assign coefW_re  = {{13{cW_re_i[31]}}, cW_re_i[30:0], {20{1'b0}}}; //
20.44
```

```
assign coefW_im  = {{13{cW_im_i[31]}}, cW_im_i[30:0], {20{1'b0}}}; //
20.44
assign data      = {{13{data_i [31]}}, data_i [30:0], {20{1'b0}}}; //
20.44
assign vm1_alpha = alpha * vm1; // 20.44 * 20.44 = 40.88

assign vm1_alpha_32 = {vm1_alpha[127], vm1_alpha[107:44]}; // 20.44
assign vm1_cW_re_32 = {vm1_cW_re[127], vm1_cW_re[107:44]}; // 20.44
assign vm1_cW_im_32 = {vm1_cW_im[127], vm1_cW_im[107:44]}; // 20.44
assign vm1_cW_re_m2 = vm1_cW_re_32 - vm2; // 20.44
assign data_re_32   = data_re[119:88]; // 32.0
assign data_im_32   = data_im[119:88]; // 32.0
assign data_o       = data_re_32 + data_im_32; // 32.0

always_ff @(posedge clk) begin
  if (!rstn) begin
    ready     <= 0;
    vm1       <= 0;
    vm2       <= 0;
    vm1_cW_re <= 0;
    vm1_cW_im <= 0;
    data_re   <= 0;
    data_im   <= 0;
    indx1     <= 0;
    vmcw      <= 0;
  end
  else if (en && !ready) begin
    if (indx1 < (NS - 1)) begin
      vm1   <= data + vm1_alpha_32 - vm2;
      vm2   <= vm1      ;
      indx1 <= indx1 + 1;
    end
    else if (!vmcw) begin
      vm1_cW_re <= coefW_re * vm1;
      vm1_cW_im <= coefW_im * vm1;
      vmcw      <= 1;
    end
    else if (vmcw) begin
      data_re <= vm1_cW_re_m2 * vm1_cW_re_m2;
      data_im <= vm1_cW_im_32 * vm1_cW_im_32;
      ready   <= 1;
    end
  end
end
endmodule
```

2.3. Angel модуль

```
module Angel #(
  parameter NF = 11
)(
  input                       rstn   ,
  input                       clk    ,
  input                       en     ,
  output logic                ready  ,
  input        [NF-1:0][31:0] freq_i ,
  output logic [NF-1:0][31:0] angel_o
);

logic signed [31:0] ANGEL_COEF = 32'h019BC65b; // 2*pi/1000 (0.32)

logic signed [NF-1:0][31:0] k_arr; // (32.0)
logic signed [NF-1:0][63:0] angel; // (32.32)
logic                [7 :0] indx ;

genvar gvar1;
generate
  for (gvar1 = 0; gvar1 < NF; gvar1 = gvar1 + 1) begin
    assign k_arr[gvar1] = freq_i[gvar1] >> 1;
  end
endgenerate

genvar gvar2;
generate
  for (gvar2 = 0; gvar2 < NF; gvar2 = gvar2 + 1) begin
    assign angel_o[gvar2] = angel[gvar2][39:8];
  end
endgenerate

always_ff @(posedge clk) begin
    if (!rstn) begin
      ready <= 0;
      angel <= 0;
      indx  <= 0;
    end
    else if (en && !ready) begin
      if (indx < NF) begin
        angel[indx] <= k_arr[indx] * ANGEL_COEF;
        indx        <= indx + 1;
      end
      else begin
```

```
            ready <= 1;
        end
    end
end

endmodule
```

2.4. Cordic модуль

```
module Cordic #(
  parameter NF = 11
)(
  // CLK&RST
  input                                rstn ,
  input                                clk  ,
  // CTRL
  input                                en   ,
  output logic                         ready,
  // DATA
  input        signed [NF-1:0][31:0] ang_i,
  output logic signed [NF-1:0][31:0] cos_o,
  output logic signed [NF-1:0][31:0] sin_o,
  output logic signed [NF-1:0][31:0] alpha
);

logic signed [31:0] PI       = 32'h03_243F6A; // (8.24)
logic signed [31:0] PI2      = 32'h01_921FB5; // (8.24)
logic signed [31:0] COEF_DEF = 32'h00_9B74ED; // (0.32)
logic signed [63:0] ZERO     = 64'h0        ; // (0.32)

logic signed         [31:0] ang  ;
logic signed         [31:0] cos  ;
logic signed         [31:0] sin  ;
logic signed [NF-1:0][63:0] cos_m;
logic signed [NF-1:0][63:0] sin_m;
logic                [8 :0] indx0;
logic                [8 :0] indx1;
logic                [31:0] atan ;
logic                       init ;
logic                       norm ;
logic                [1 :0] quad ;

always_ff @(posedge clk) begin
  if (!rstn) begin
    ready <= 0;
    cos   <= 32'h01_000000;
    sin   <= 32'h00_000000;
    sin_m <= 0;
    cos_m <= 0;
    indx0 <= 0;
    indx1 <= 0;
    init  <= 0;
```

```
  norm  <= 0;
  quad  <= 0;
end
else if (en && !init) begin
  ang  <= ang_i[indx0];
  init <= 1;
end
else if (en && !norm) begin
  if (ang > PI) begin
    ang  <= ang - PI;
    quad <= 2'b10   ;
  end
  else if (ang > PI2) begin
    ang  <= ang - PI2;
    quad <= 2'b01    ;
  end
  else begin
    quad <= 2'b00;
  end
  norm <= 1;
end
else if (en && !ready) begin
  if (indx0 < NF) begin
    if (indx1 < 23) begin
      if (ang[31] == 0) begin
        cos <= cos - (sin >>> indx1);
        sin <= sin + (cos >>> indx1);
        ang <= ang - atan           ;
      end
      else begin
        cos <= cos + (sin >>> indx1);
        sin <= sin - (cos >>> indx1);
        ang <= ang + atan           ;
      end
      indx1 <= indx1 + 1;
    end
    else begin
      if (quad == 2'b10) begin
        cos_m[indx0] <= ZERO - cos * COEF_DEF;
        sin_m[indx0] <= ZERO - sin * COEF_DEF;
      end
      else if (quad == 2'b01) begin
        cos_m[indx0] <= ZERO - sin * COEF_DEF;
        sin_m[indx0] <= cos * COEF_DEF;
      end
```

```
        else begin
          cos_m[indx0] <= cos * COEF_DEF;
          sin_m[indx0] <= sin * COEF_DEF;
        end
        cos          <= 32'h01_000000 ;
        sin          <= 32'h00_000000 ;
        indx0        <= indx0 + 1     ;
        indx1        <= 0             ;
        init         <= 0             ;
        norm         <= 0             ;
      end
    end
    else begin
      ready <= 1;
    end
  end
end

genvar gvar;
generate
  for (gvar = 0; gvar < NF; gvar = gvar + 1) begin
    assign cos_o[gvar] = {cos_m[gvar][63], cos_m[gvar][54:24]};
    assign sin_o[gvar] = {sin_m[gvar][63], sin_m[gvar][54:24]};
    assign alpha[gvar] = {cos_m[gvar][63], cos_m[gvar][53:23]};
  end
endgenerate

always_comb begin
  case (indx1)
    0      : atan = 32'h00_C90FDA;  //  atanh(2^(-0))
    1      : atan = 32'h00_76B19C;  //  atanh(2^(-1))
    2      : atan = 32'h00_3EB6EB;  //  atanh(2^(-2))
    3      : atan = 32'h00_1FD5BA;  //  atanh(2^(-3))
    4      : atan = 32'h00_0FFAAD;  //  atanh(2^(-4))
    5      : atan = 32'h00_07FF55;  //  atanh(2^(-5))
    6      : atan = 32'h00_03FFEA;  //  atanh(2^(-6))
    7      : atan = 32'h00_01FFFD;  //  atanh(2^(-7))
    8      : atan = 32'h00_00FFFF;  //  atanh(2^(-8))
    9      : atan = 32'h00_007FFF;  //  atanh(2^(-9))
    10     : atan = 32'h00_003FFF;  //  atanh(2^(-10))
    11     : atan = 32'h00_001FFF;  //  atanh(2^(-11))
    12     : atan = 32'h00_000FFF;  //  atanh(2^(-12))
    13     : atan = 32'h00_0007FF;  //  atanh(2^(-13))
    14     : atan = 32'h00_0003FF;  //  atanh(2^(-14))
    15     : atan = 32'h00_0001FF;  //  atanh(2^(-15))
```

```
    16      : atan = 32'h00_0000FF;  //  atanh(2^(-16))
    17      : atan = 32'h00_00007F;  //  atanh(2^(-17))
    18      : atan = 32'h00_00003F;  //  atanh(2^(-18))
    19      : atan = 32'h00_00001F;  //  atanh(2^(-19))
    20      : atan = 32'h00_00000F;  //  atanh(2^(-20))
    21      : atan = 32'h00_000007;  //  atanh(2^(-21))
    22      : atan = 32'h00_000003;  //  atanh(2^(-22))
    23      : atan = 32'h00_000001;  //  atanh(2^(-23))
    default: atan = 32'h00_000000;
  endcase
end


endmodule
```

2.5. DataScale модуль

```
module DataScale (
  // CLK&RST
  input                        rstn  ,
  input                        clk   ,
  // DATA
  input                [7 :0] data_i,
  output logic signed [31:0] data_o
);

logic [31:0] SCALE_COEF = 32'h00_0D0000; // 13/256 (8.24)
logic [31:0] data  ;
logic [63:0] data_m;

assign data   = {data_i, 24'h00};
assign data_o = data_m[55:24]   ;

always_ff @(posedge clk) begin
  if (!rstn) begin
    data_m <= 0;
  end
  else begin
    data_m <= SCALE_COEF * data;
  end
end

endmodule
```

2.6. TB модуль

```
module tb_FourierTransform;

localparam SIM_TIM    = 2000000;
localparam CLK_PERIOD = 10    ;
localparam NF         = 11    ;
localparam NS         = 1000;

logic                       rstn  ;
logic                       clk   ;
logic                       cEn   ;
logic                       hEn   ;
logic        [NF-1:0]       ready ;
logic                [7 :0] sample;
logic signed [NF-1:0][31:0] data_o;

FourierTransform #(
  .NF(NF),
  .NS(NS)
) DUT (
  .rstn  (rstn  ),
  .clk   (clk   ),
  .cEn   (cEn   ),
  .hEn   (hEn   ),
  .ready (ready ),
  .sample(sample),
  .data_o(data_o)
);

integer fd_r_s;
integer fd_w_v;
integer fd_w_c;
integer fd_w_s;

always #(CLK_PERIOD/2) clk=~clk;

initial begin
  clk    = 0;
  rstn   = 0;
  cEn    = 1;
  hEn    = 0;
  sample = 0;
  repeat(5) @(posedge clk);
  rstn = 1;
```

```
end

initial begin
  fd_r_s =
$fopen("D:/Desktop/Study_now/SRW/Fourier_Transform/src/sim/sample.csv",
"r");
  if (fd_r_s == 0) $finish;
  fd_w_v =
$fopen("D:/Desktop/Study_now/SRW/Fourier_Transform/src/sim/vector.csv",
"w");
  if (fd_w_v == 0) $finish;
  fd_w_c =
$fopen("D:/Desktop/Study_now/SRW/Fourier_Transform/src/sim/cos.csv",
"w");
  if (fd_w_c == 0) $finish;
  fd_w_s =
$fopen("D:/Desktop/Study_now/SRW/Fourier_Transform/src/sim/sin.csv",
"w");
  if (fd_w_s == 0) $finish;
end


initial begin
  wait(DUT.u_Cordic.ready);
  for (int i = 0; i < NF; i = i + 1) begin
    $fwrite(fd_w_c, "%h\n", DUT.u_Cordic.cos_o[i]);
    $fwrite(fd_w_s, "%h\n", DUT.u_Cordic.sin_o[i]);
  end
end

initial begin
  wait(DUT.ready1);
  $fscanf(fd_r_s, "%d\n", sample);
  @(posedge clk);
  hEn = 1;

  fork
    begin
      while (!$feof(fd_r_s)) begin
        @(posedge clk);
        $fscanf(fd_r_s, "%d\n", sample);
        $fwrite(fd_w_v, "%h\n", DUT.herzel[10].u_Herzel.vm1);
      end
      @(&ready);
    end
```

```
    begin
      #(SIM_TIM);
    end
  join_any

  $fwrite(fd_w_v, "%h\n", DUT.herzel[10].u_Herzel.vm1);
  $fclose(fd_r_s);
  $fclose(fd_w_v);
  $fclose(fd_w_c);
  $fclose(fd_w_s);
  #(5000) $stop;
end

endmodule
```

3. Приложение 2

Ссылка на репозиторий для более удобного моделирования:

https://github.com/MarininNS/GoertzelAlgorithm.git

Статьи из интернета с более подробным описанием алгоритмов:

Алгоритм Гёрцеля (dsplib.org)