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. Типы шумов и их природа 2](#_Toc105242510)

[1.1. Тепловой шум или шум Джонсона-Найквиста 4](#_Toc105242511)

[1.2. Генерационно-рекомбинационный шум 5](#_Toc105242512)

[1.3. Шум типа или фликкер-шум 5](#_Toc105242513)

[2. Методы измерения шумов и установки 8](#_Toc105242514)

[2.1. Метод измерения напряжения (тока) фотосигнала и напряжения (тока) шума ФЭПП и (ФПУ) 8](#_Toc105242515)

[2.2. Измерение напряжения шума К54.410 11](#_Toc105242516)

[2.3. Исследования спектральной плотности мощности шума фотоприемника 13](#_Toc105242517)

# 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:

equation 13

Где u - промежуточные значения, которые рассчитываются итерационно:

equation 14

W – поворотный коэффициент



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.

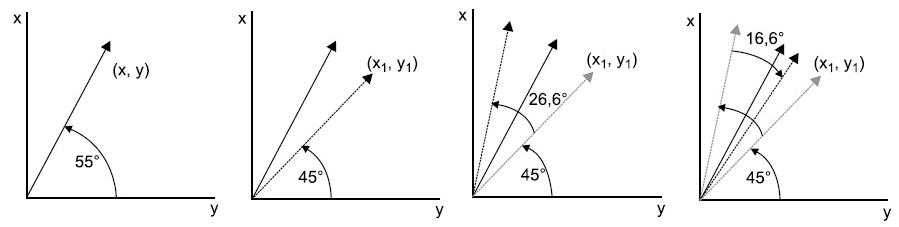
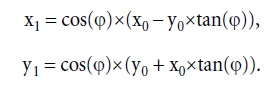


Рисунок CORDIC алгоритм

Координаты x1 и y1 вычисляются по формулам:



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



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).

## Пример 1

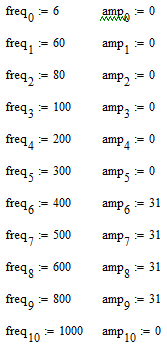


Рисунок Входной сигнал и его параметры

Были получены следующие результаты:



Рисунок Значения полученные в системе matchcad

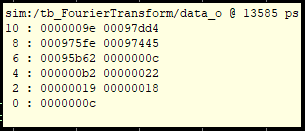


Рисунок Значения полученные при симуляции модуля в modelsim. Индексы соответствуют индексам частот на рис.4, значения записаны в формате 32.0.

## Пример 2

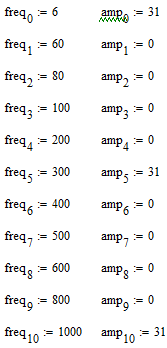
 

Рисунок Входной сигнал и его параметры



Рисунок Значения полученные в системе matchcad

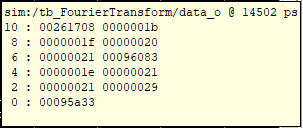


Рисунок Рисунок 4 Значения полученные при симуляции модуля в modelsim.

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

# Приложение

## 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

## 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

## 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

## 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

## 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

## 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

# 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)](https://ru.dsplib.org/content/goertzel/goertzel.html)