Научное общество учащихся «Эврика»

МУНИЦИПАЛЬНОе АВТОНОМНОЕ ОБЩЕобразовательное учреждение «ЛИЦЕЙ № 38»

Советского района г. Нижнего Новгорода

Реализация алгоритмов цифровой обработки сигналов на программируемой логике

Выполнил:

Соколов Андрей Валерьевич,

Ученик 9 “Б” класса

Руководитель:

Халепина Светлана Викторовна

учитель информатики и технологии

Нижний Новгород

2024

Содержание

[Введение 3](#_Toc159093653)

[1 Терминология 5](#_Toc159093654)

[2 Целеполагание 6](#_Toc159093655)

[3 ПЛИС (программируемая логическая интегральная схема) 7](#_Toc159093656)

[4 Производители микросхем программируемой логике 8](#_Toc159093657)

[5 Языки описания аппаратуры 10](#_Toc159093658)

[5.1 Общая структура описания модуля: 10](#_Toc159093659)

[5.2 Входы и выходы модуля: 10](#_Toc159093660)

[5.3 Описание провода и шины: 10](#_Toc159093661)

[5.4 Процедурный блок always\_comb 11](#_Toc159093662)

[5.5 Процедурный блок always\_ff @(posedge clk) 11](#_Toc159093663)

[5.6 SystemVerilog блокирующие и неблокирующие присваивание 11](#_Toc159093664)

[6 Структурное представление и поведенческое описание основных конструкций 14](#_Toc159093665)

[7 Среда разработки 19](#_Toc159093666)

[8 Функциональная симуляция HDL кода 21](#_Toc159093667)

[9 Дополнительный двоичный код 25](#_Toc159093668)

[10 Алгоритмы цифровой обработки сигналов 27](#_Toc159093669)

[Заключение 34](#_Toc159093670)

[Список литературы 35](#_Toc159093671)

[Приложения 36](#_Toc159093672)

# Введение

В настоящее время информационные технологии играют значительную роль. "Умные устройства" стремительно заполняют все сферы жизни человека. Устройства становятся сложнее, растут требования к скорости обработки, размерам и потреблению питания. На фоне развития умных устройств,

возрастает потребность к электронным вычислителям - интегральным микросхемам.

Производство интегральных микросхем хоть и остаётся довольно дорогостоящим, но становится доступнее год от года.

Чаще всего разработка микросхемы начинается с создание прототипа устройства на микросхемах программируемой логики (ПЛИС). ПЛИС (программируемые логические интегральные схемы)

как правило уступают серийным микросхемам по размерам, потреблению, скорости. Но одним из главных преимуществ является гибкая структура - ПЛИС можно перепрограммировать, то есть можно тестировать и модифицировать алгоритмы. При нахождении ошибки в процессе работы, можно поменять алгоритм и запрограммировать микросхему заново. Если же ошибка будет обнаружена в готовом серийном изделии, исправить её уже нельзя, придётся переделывать микросхему, фактически запускать весь процесс с нуля.

В последнее время в связи с увеличением разработок спецвычислителей и процессорных ядер - растёт потребность в специалистах по языкам описания аппаратуры (HDL - hardware description language). Основные направления — это разработчики реализующие алгоритмы на RTL (register transfer level) и верификаторы.

Я выбрал тему проекта, связанную с разработкой алгоритмов на ПЛИС (RTL разработкой) поскольку это тесно связано с разработкой интегральных устройств массового применения.

В данной работе я буду изучать принципы разработки на программируемой логике. Планирую освоить язык описания аппаратуры SystemVerilog. Разработка программируемой логики будет осуществляться в среде Vivado. Изучу среду для функциональной симуляции (Active-HDL). Буду знакомиться с несколькими алгоритмами цифровой обработкой сигналов.

# 1 Терминология

* FPGA (field-programmable gate array) – ПЛИС (программируемая логическая интегральная схема).
* CLBs (Configurable logic blocks) – Конфигурированные логические блоки.
* IOB (input/output buffer) – Входные/выходные буфера.
* block RAMs – Блоковая память.
* DSPs (Dynamic Syncopation Productions) – Блоки цифровой обработки сигналов.
* ARM (Advanced RISC Machine) – Встроенные процессорные ядра.
* HDL (hardware description language) – Язык описания аппаратуры.
* RTL (register transfer level) – Уровень регистровых передач.
* DSP (digital signal processing) – ЦОС (цифровая обработка сигналов)

# 2 Целеполагание

Цель проекта:

1. Реализовать алгоритмы на ПЛИС с использованием языка System Verilog.

Задачи проекта:

1. Освоить инструмент для разработки на ПЛИС (Vivado)

2. Освоить инструмент для функциональной симуляции (Active-HDL)

3. Разобраться с алгоритмами цифровой обработки (цифровой фильтр и CORDIC)

4. Освоить язык описания аппаратуры System Verilog

# 3 ПЛИС (программируемая логическая интегральная схема)

FPGA(field-programmable gate array) электронный компонент (интегральная микросхема), используемый для создания конфигурируемых цифровых электронных схем.

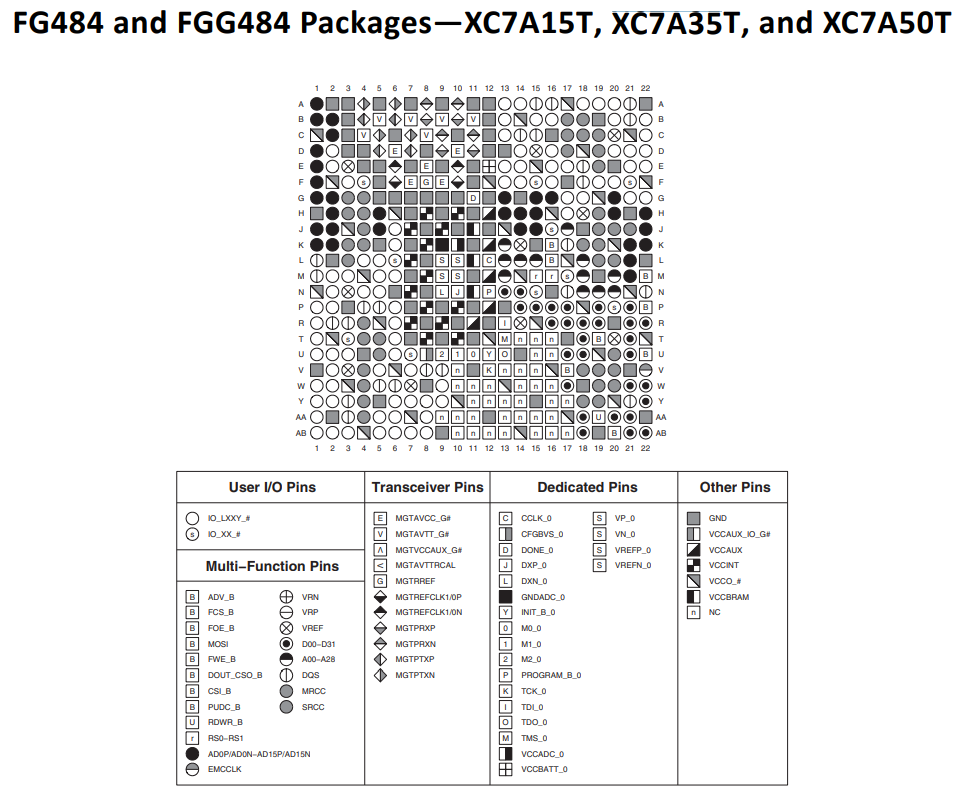
FPGA представляет из себя перепрограммируемую микросхему, которая способна выполнять вычисления с большой скоростью. Поэтому данные типы электронных вычислителей часто используются для цифровой обработки.

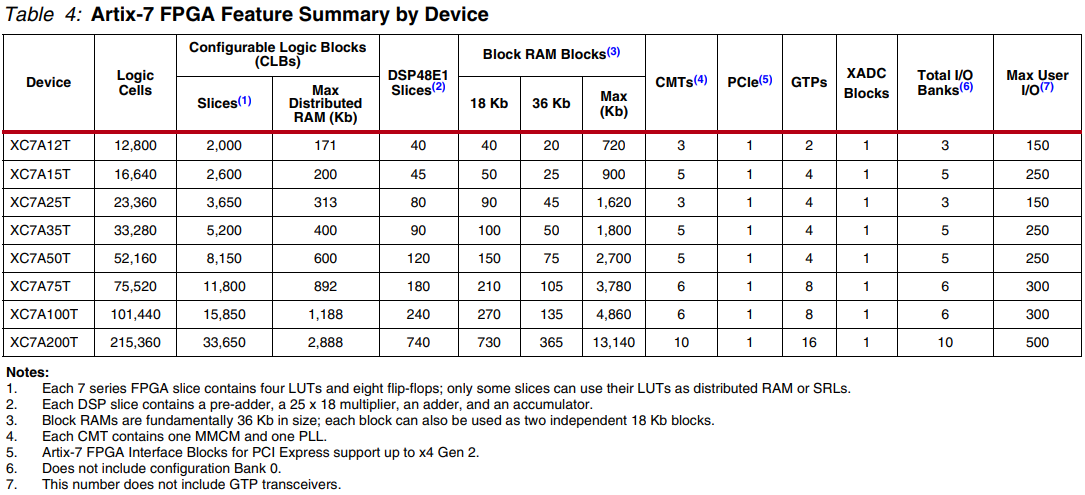
FPGA содержит логические элементы и связи соединяющие их. Их использование контролируется конфигурацией прошивки FPGA. Существует большое количество логических элементов от простых комбинаторных логических функций до встроенных процессоров. Логические элементы и связи соединяющие их расположены в виде сетки из фрагментов. Структура FPGA чрезвычайно регулярная. Каждая плита содержит одну из небольшого разнообразия VLSI, предназначенной для логики и маршрутизации.

* Конфигурированные логические блоки (Configurable logic blocks, CLBs) содержит программируемую логику и память.
* Входные/выходные буфера (IOB).
* Сигналы синхронизации (Clocks)
* Блоки специального назначения, которые включают: Блоковая память (block RAMs), Блоки цифровой обработки сигналов (DSPs), встроенные процессорные ядра (ARM).

Использование вышеупомянутых блоков определяется конфигурационным файлом (Bitstream), который как правило загружается из внешнего устройства (внешняя флэш память)

# 4 Производители микросхем программируемой логике

 На данный момент основными производителями FPGA являются Xilinx (AMD) и Intel. Микросхемы отличаются количеством логических элементов, скоростью работы, наличием интерфейсов и т.д.



# 5 Языки описания аппаратуры

Для описания алгоритмов для FPGA существуют языки описания аппаратуры (HDL – Hardware Description Language). На сегодняшний день широко применяются такие языки: VHDL, Verilog, SystemVerilog.

В данном проекте для разработки мной был выбран язык SystemVerilog.

Структура проекта на SystemVerilog представляет собой модули (Module). Как правило, в одно файле записывают один модуль.

### 5.1 Общая структура описания модуля:

module <Имя модуля>

(

<входы и выходы модуля>

);

Endmodule

### 5.2 Входы и выходы модуля:

Вход описывается как:

Input <провод или шина>

Выход описывается как:

Output <провод или шина>

### 5.3 Описание провода и шины:

Провод (элемент, который переносит 1 бит) описывается как:

Logic <имя провода>

Шина (элемент, который переносит несколько бит) описывается как:

Logic [<старший бит> : <младший бит>] <имя шины>

Шина из проводов описывается как:

Logic <имя шины> [<младший индекс> : <старший индекс>]

Двумерную шину описывается как:

Logic [<старший бит> : <младший бит>] <имя шины> [<младший индекс> : <старший индекс>]

### 5.4 Процедурный блок always\_comb

always\_comb begin

a <= b+c;

end

5.5 Процедурный блок always\_ff @(posedge clk)

always\_ff @(posedge clk) begin

a <= b+c;

end

### 5.6 SystemVerilog блокирующие и неблокирующие присваивание

В SystemVerilog существует два вида присваивания: блокирующие и неблокирующие

Пример блокирующего присваивания:

always @(posedge clk) begin

x = x + 1;

y = x;

end

В данном случае если x = 3, то в результате выполнения данного блока y = 4,

x = 4.

Пример неблокирующего присваивания:

always @(posedge clk) begin

x <= x + 1;

y <= x;

end

В данном случае если x = 3, то в результате выполнения данного блока y = 3,

x = 4.

Отличительная характеристика комбинационных схем состоит в том, что значения их вывода можно задать с помощью булевых функций, зависящихся только от значений входов. Логические функции есть абстрактное описание электронных схем, в которых входы преобразуются в выходы непосредственно, без использования элементов памяти.

Комбинационную схему можно представить с помощью языков описания аппаратуры.

Поведенческое описание D-триггера:

module DFF (

input logic D, clk, rst,

output logic Q

);

always\_ff @(posedge clk, posedge rst) begin

if (rst) begin

Q <= 0;

end else begin

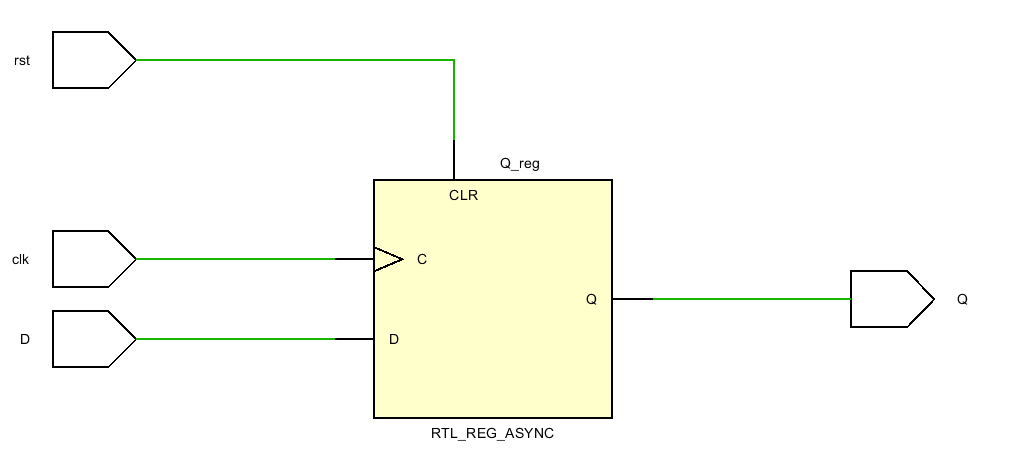
Q <= D;

end

end

endmodule

# 6 Структурное представление и поведенческое описание основных конструкций

Структурное представление поведенческого описания D-триггера:

Поведенческое описание мультиплексора:

module MuxSimple(

input logic a, b, select,

output logic y

);

always\_comb

begin

if(select)

y <= b;

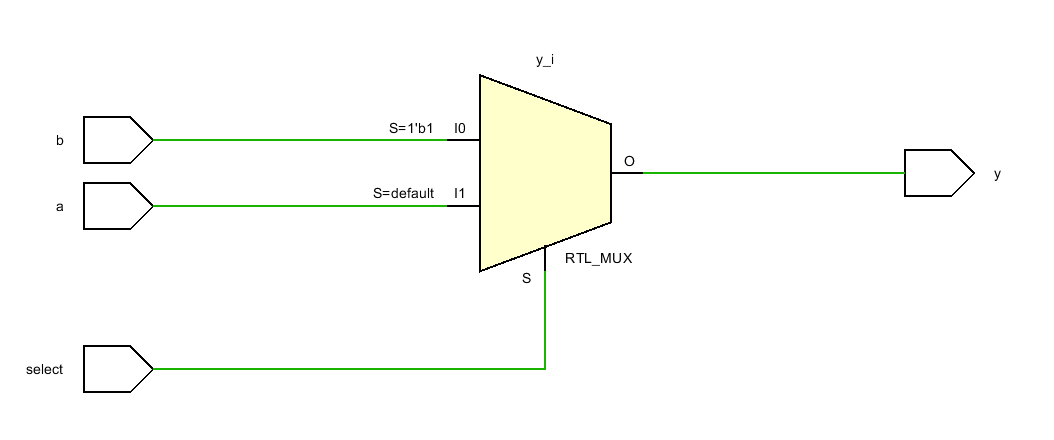
else

y<=a;

end

endmodule

Структурное представление поведенческого описания мультиплексора:



Поведенческое описание памяти:

typedef struct packed {

logic [3:0] addr;

logic [27:0] data;

} Packet;

module rams\_sdp\_struct #(

parameter A\_WID = 10,

D\_WID = 32

)

(

input clk,

input we,

input ena,

input [A\_WID-1:0] raddr, waddr,

input Packet din,

output Packet dout

);

Packet mem [2\*\*A\_WID-1:0];

always @ (posedge clk)

begin

if (ena) begin

if(we)

mem[waddr] <= din;

end

end

always @ (posedge clk)

begin

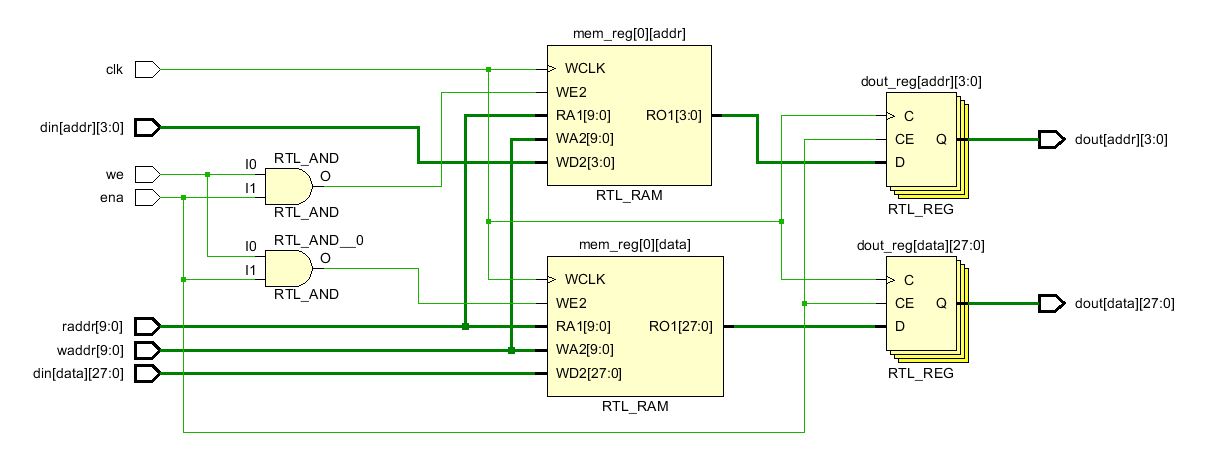
if (ena) begin

dout <= mem[raddr];

end

end

endmodule

Структурное представление поведенческого описания памяти:

Поведенческое описание сумматора:

module Adder (

input logic [7:0] a,

input logic [7:0] b,

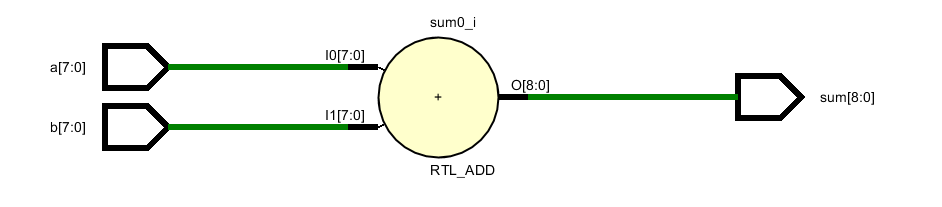
output logic [8:0] sum

);

always\_comb

sum <= a + b;

endmodule

Структурное представления поведенческого описания сумматора:

# 7 Среда разработки

Каждая из упомянутых выше фирм производителей FPGA имеют свою собственную среду разработки: Xilinx – Vivado, Intel – Quartus. Каждая из сред совместима только со своими микросхемами. Среда разработки позволяет описать алгоритм на языке описания аппаратуры и собрать прошивку для программирования ей микросхемы. Также среда позволяет работать с внутрисистемной отладкой.

В сборке проекта в среде Vivado можно выделить три основных этапа.

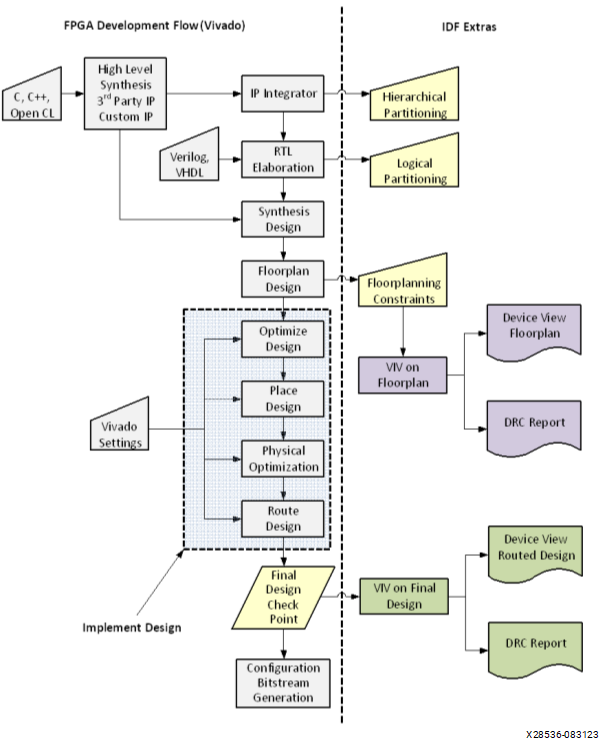
* + Synthesis (синтез) – анализ исходного кода и преобразование его в логические элементы и связи между ними.

Результатом синтеза является файл, называемый в англоязычной литературе netlist

* + Implementation (имплементация) – исходными данными для данного этапа является результат предыдущего этапа – файл netlist. На данном этапе происходит преобразование логических элементов в примитивы внутри конкретной микросхемы. Для данного этапа необходимо выбрать микросхему, на которой производится разработка.
  + Bitstream – бинарный файл содержащий конфигурацию для FPGA. Как правило, Bitstream файл записывается на внешнею энергонезависимую память. После подачи питания на FPGA, Она производит скачивание прошивки из внешней памяти, тем самым конфигурирует свою внутреннюю структуру. Еще сценарий конфигурации FPGA – это

прошивка с помощью JTAG кабеля. Этот метод чаше всего используется

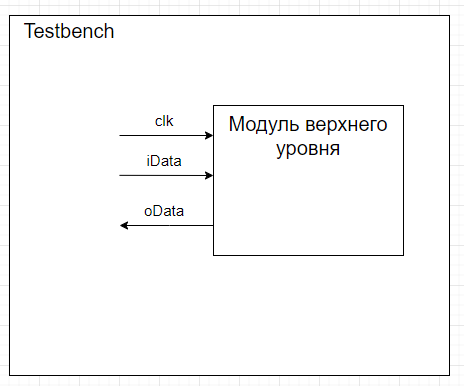
для отладки. При данном методе файл на прямую загружается через отладочный кабель.

Ниже представлена диаграмма сборки проекта в среде Vivado. В ней кроме основных шагов, описанных выше, представлены другие шаги разработки проекта.

# 8 Функциональная симуляция HDL кода

Для отладки кода можно использовать вышеупомянутые среды разработки. Но как правило они используются для сборки проекта и отладки. Для функциональной симуляции используется инструменты других производителей: Aldec – Active-HDL, Mentor – Modelsim. Данные инструменты предоставляют широкий функционал для моделирования и тестирования написанного HDL кода.

Функциональная симуляция используется для проверки работоспособности написанного кода без использования реального железа. То есть является своего рода эмуляции работы кода на персональном компьютере. В реальном железе внешние сигналы подключаются к входам и выходам микросхемы. В функциональной симуляции используется тестовый модуль (Testbench), который включает в себя модуль верхнего уровня



На рисунке выше изображена структурная схема проекта симуляции. Внешние сигналы clk, iData, oData, подключены в модуле созданном специально для симуляции – Testbench. Testbench не участвует в синтезе и поэтому не присутствует в файле прошивки. В Testbench могут использоваться **не синтезируемые конструкции**. Например, чтобы создать сигнал синхронизации (clk) можно использовать следующую конструкцию:

reg clk;

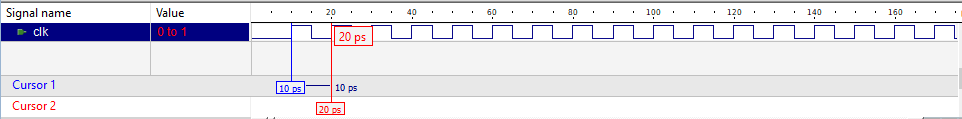
initial

begin

clk <= 0;

#5 forever #5 clk <= ~clk;

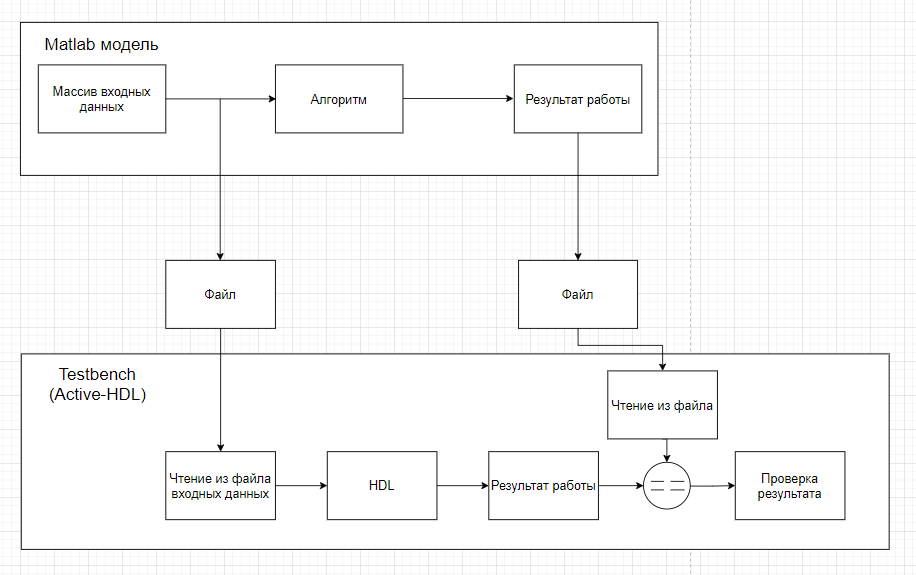
End



Выше на временной диаграмме можно видеть результат выполнения кода. В начале сигналу clk присваивается нулю потом каждые 5 пикосекунд сигнал clk инвертируется. #5 определяет задержку, данная конструкция не может быть синтезируемая в реальном устройстве. В реальном устройстве в качестве источника сигнала синхронизации используется специальные микросхемы – генератор опорного сигнала.

В Testbench также для теста можно сформировать входное воздействие (iData) и проверить результат работы (oData).

В сложных проектах результаты симуляции невозможно проверить без математической модели. Чаше всего перед реализацией алгоритма на HDL создается математическая модель, которая предоставляет исходные данные для алгоритма, а также результаты его работы. Эти данные могут быть сохранены в виде файлов, а потом прочитаны и использованы в HDL testbench. При таком подходе исходные данные одинаковы в математической модели и в HDL коде, следовательно результаты работы должна также совпадать.



Рассмотрим данный механизм на примере фильтра. В качестве среды создание математической модели выбрана Matlab. Математическая модель фильтра выглядит следующим образом:

cfs = [-1,1,-1,-1,1,1,1,-1]; % Коэффиценты фильтра

filter\_in = randsrc(1,100,[-10:10]); % Входные данные фильтра

% filter - библеотечная функция Matlab, реализующая фильтр

filter\_out = filter(cfs,1,filter\_in);

fileID = fopen('filter\_in.txt','w'); % Создаем файл для записи входных данных фильтра

fprintf(fileID,'%d\n',filter\_in);

fclose(fileID);

fileID = fopen('filter\_out.txt','w'); % Создаем файл для записи выхода фильтра

fprintf(fileID,'%d\n',filter\_out);

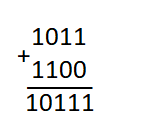
fclose(fileID);

# 9 Дополнительный двоичный код

Дополнительный код – система представления знаковых целых чисел, упрощающая выполнение двоичной арифметики. Чтобы найти дополнительный код числа, представленного в двоичной системе, необходимо инвертировать его биты, после чего прибавить к результату единицу. Это представление упрощает построение вычислительных устройств, позволяя выполнять сложение и вычитание при помощи одной и тоже схемы, поскольку вычитание числа равнозначно прибавлению его дополнительного кода. К тому же этот приём позволяет эффективно обрабатывать изменение знаков и нулевые значения, в связи с чем он так широко используется в вычислительных системах.

|  |  |  |
| --- | --- | --- |
| Двоичный код | Десятичный беззнаковый код | Десятичный знаковый код (дополнительный код) |
| 0000 | 0 | 0 |
| 0001 | 1 | 1 |
| 0010 | 2 | 2 |
| 0011 | 3 | 3 |
| 0100 | 4 | 4 |
| 0101 | 5 | 5 |
| 0110 | 6 | 6 |
| 0111 | 7 | 7 |
| 1000 | 8 | -8 |
| 1001 | 9 | -7 |
| 1010 | 10 | -6 |
| 1011 | 11 | -5 |
| 1100 | 12 | -4 |
| 1101 | 13 | -3 |
| 1110 | 14 | -2 |
| 1111 | 15 | -1 |

Рассмотрим сложение двух беззнаковых чисел: 11 (b1011) и 12 (b1100)



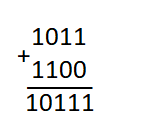
11 + 12 = 23

10111 = 23

Представим данные числа как знаковые:

Inv(1011) + 1 = 0100 + 0001 = 0101 = 5, то есть 1011 = -5

Inv(1100) + 1 = 0011 + 0001 = 0100 = 4, то есть 1100 = -4



-5 + (-4) = -9

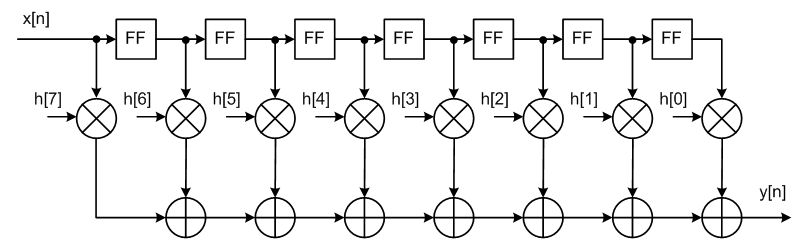
Inv(10111) + 1 = 01000 + 00001 = 1001 = 9, то есть 10111 = -9

# 10 Алгоритмы цифровой обработки сигналов

В современном мире информация как правило обрабатывается в цифровом виде. Аналоговый сигнал – звук, видео, радиоволны преобразуются с помощью аналого-цифрового преобразования, то есть представления сигнала в виде чисел. Поэтому алгоритмы цифровой обработки сигналов находят широкое применение.

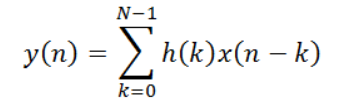
**Цифровой фильтр**. Цифровая фильтрация – один из основных алгоритмов цифровой обработке сигналов (ЦОС). В общих словах цифровой фильтр можно описать как выделение некоторого признака из общего набора данных.

Одна из структур цифрового фильтра может быть представлена следующим образом.



Как видно на рисунке в структуре цифрового фильтра можно отметить такие блоки как задержка на один такт (FF), умножение (на h), сумматоры (+). Входное воздействие x поступает на вход фильтра. Выход фильтра (y) является результатом фильтрации входного воздействия.

Математически фильтр можно представить следующей формулой:



X – отсчеты входного сигнала

Y – oтсчеты выходного сигнала

h – коэффициенты фильтра

Для того чтобы понять как выбираются коэффициенты фильтра разберем следующий пример:

Предположим, что нам нужно определить наличие некоторой последовательности конечной длины в поступающих из вне данных. Для упрощения, данные состоят всего из двух элементов: +1 и -1. Последовательность, наличие которой нужно определить выглядит следующим образом: x = [-1,-1,+1,+1,-1]. В случае наличия данной последовательности, цифровой фильтр должен выдать на выходе максимально возможное значение равное 5. Такой фильтр будет называться **оптимальным** с такой последовательностью. Существует правило для выбора коэффициентов фильтра оптимального с последовательностью: коэффициенты фильтра должны быть зеркально отраженной входной последовательностью, с которой он оптимален. Для нашего примера коэффициенты фильтра h = [-1,+1,+1,-1,-1]

Рассмотрим пример для входной последовательности и входных коэффициентов, описанных выше.

1й отсчёт: (-1)\*(-1) + 0 \*(+1) + 0\*(+1) + 0\*(-1) + 0\*(-1) = 1

2й отсчёт: (-1)\*(-1) + (-1) \*(+1) + 0\*(+1) + 0\*(-1) + 0\*(-1) = 0

3й отсчёт: (+1)\*(-1) + (-1) \*(+1) + (-1)\*(+1) + 0\*(-1) + 0\*(-1) = -3

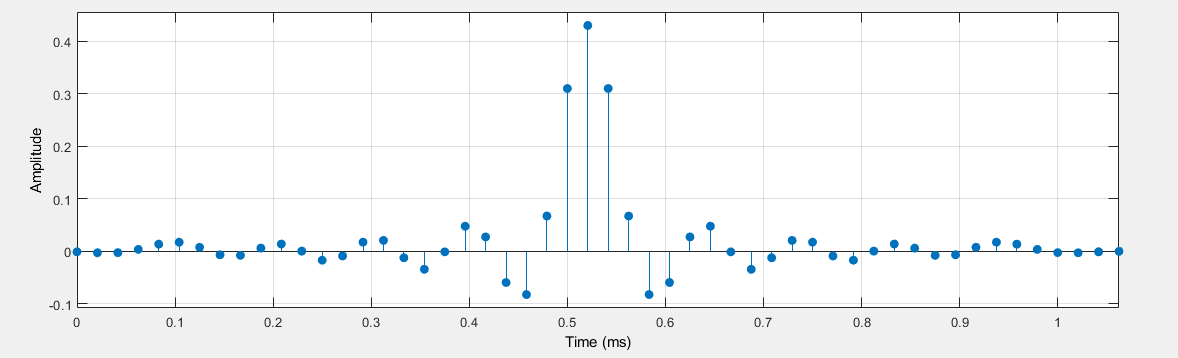
4й отсчёт: (+1)\*(-1) + (+1) \*(+1) + (-1)\*(+1) + (-1)\*(-1) + 0\*(-1) = 0

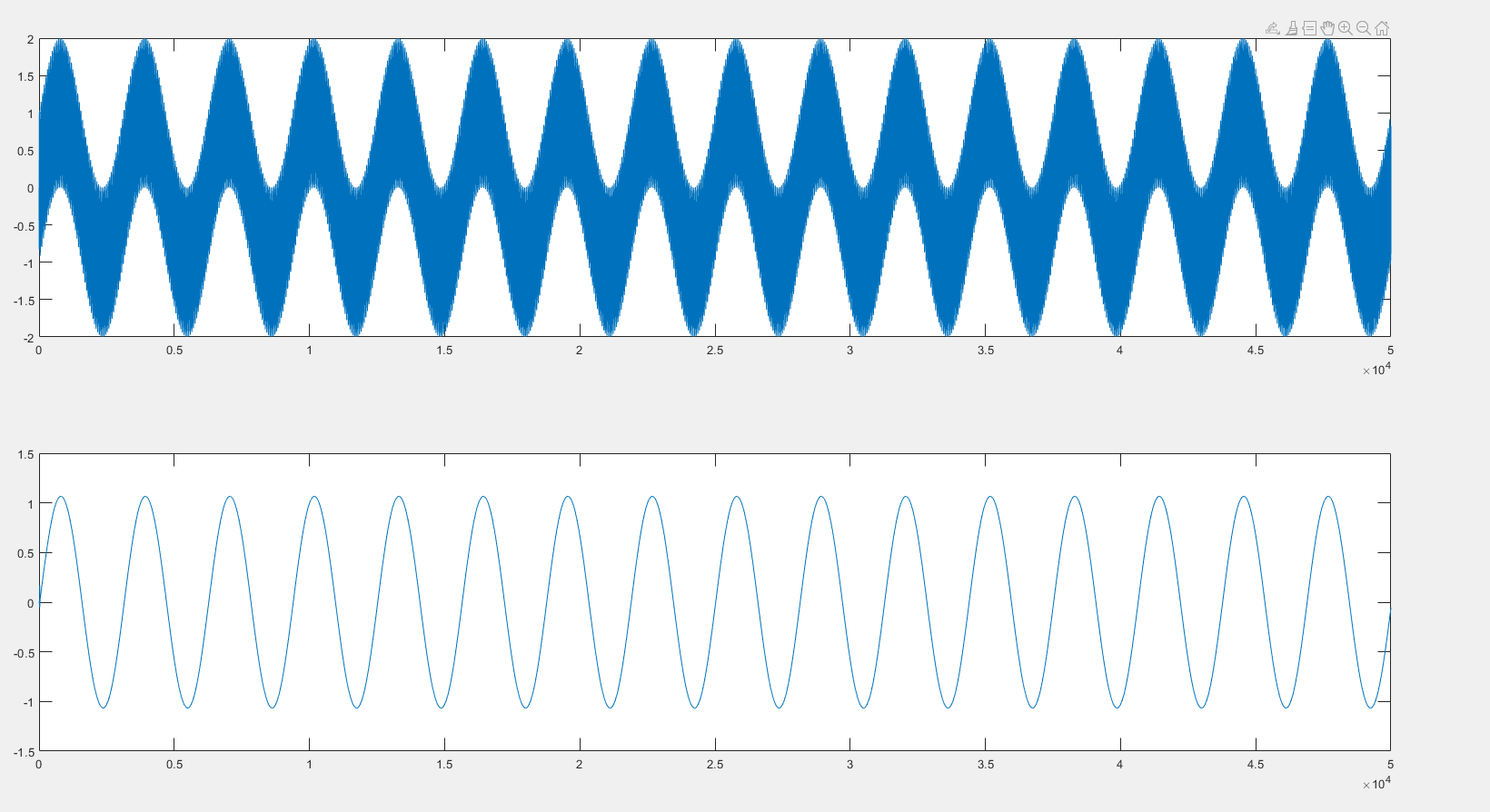
5й отсчёт: (-1)\*(-1) + (+1) \*(+1) + (+1)\*(+1) + (-1)\*(-1) + (-1)\*(-1) = 5

Из примера видно, что максимально возможный результат (5) может получиться только в случаи детектируемой последовательности.

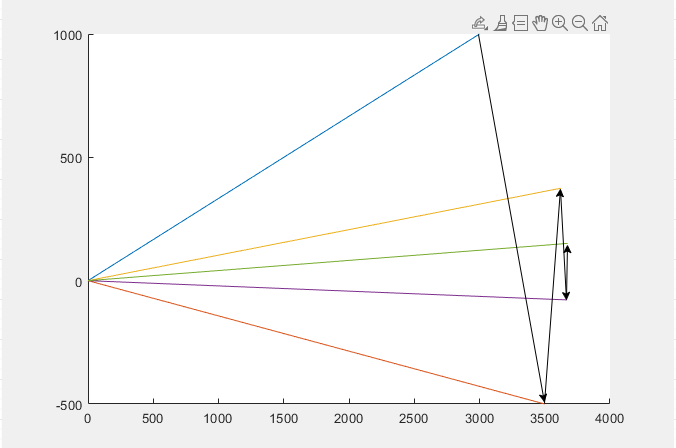
Фильтры могут выполнять более сложные операции. Например, в радиосвязи с помощью фильтра можно выделить интересующую радиостанцию. Для примера возьмём в качестве входных данных два синуса с разной частотой. Коэффициенты фильтра были получены из расчёта подавлении синуса с более высокой частотой.

Коэффициенты фильтра выглядят следующим образом:



Входом фильтра является сумма синусов с частотами f и f\*100. Ниже изображены графики входа и выхода фильтра:

**CORDIC (COordinate Rotation DIgital Computer).** Цифровой вычислитель поворота системы координат; метод «цифра за цифрой», алгоритм Волдера.



В ЦОС часто приходится выполнять такие операции как нахождение амплитуды или угла наклона вектора. Классические методы вычисления требует больших ресурсов. Алгоритм CORDIC может выполнить вычисления использую операции сложения и поразрядного сдвига (что довольно легко реализуется на программируемой логике)

Рассмотрим пример **вычисление угла** между некоторым вектором и осью Ox. Нужно заметить, что алгоритм работает на диапазоне углов от -90 до + 90 градусов. Для работы с произвольными углами требуется не значительная модификация алгоритма

Исходными данными для алгоритма является вектор, в данном случае

X = 1000;

Y = 2000;

Угол наклона данного вектора к оси oX равен 1.1071 радиан.

Также для алгоритма необходима таблица арктангенсов.

atan(1), atan(1/2), atan(1/4), atan(1/8),…

Каждый арктангенс используется на соответствующей итерации.

Алгоритм cordic является итерационным. Результаты текущей итерации являются исходными данными для следующей. В каждой итерации cordic выполняются следующие действия:

acc\_Z = 0;

for L = 1:max\_L

Z = atan\_table(L);

tmp\_X = X;

if (Y > 0)

X = X + bitshift(abs(Y),-(L-1));

Y = Y - bitshift(tmp\_X,-(L-1));

acc\_Z = acc\_Z + Z;

else

X = X + bitshift(abs(Y),-(L-1));

Y = Y + bitshift(tmp\_X,-(L-1));

acc\_Z = acc\_Z - Z;

end

end

Распишем результаты для нескольких итераций.

0: X = 1000; Y = 2000; acc\_Z = 0;

1: X = 3000; Y = 1000; acc\_Z = 0.785;

2: X = 3500; Y = -500; acc\_Z = 1.249;

3: X = 3625; Y = 375; acc\_Z = 1.004;

4: X = 3671; Y = -78; acc\_Z = 1.128;

5: X = 3676; Y = 151; acc\_Z = 1.066;

6: X = 3680; Y = 37; acc\_Z = 1.097;

7: X = 3680; Y = -20; acc\_Z = 1.112;

8: X = 3681; Y = 8; acc\_Z = 1.105;

9: X = 3681; Y = -6; acc\_Z = 1.108;

10: X = 3682; Y = 1; acc\_Z = 1.107;

Результатом работы алгоритма (угла наклона вектора к оси oX) является переменная acc\_Z, которая на 10 итерации является 1.107 радиана, что близко к расчетному значению. Точность алгоритма определяется количеством итераций. Можно заметить, что значение X координаты для вектора растет с каждой итерацией, а Y координата стремиться к нулю. Алгоритм можно представить как стремление вектора быть сонаправленным с осью oX при таком условии угол поворота будет равен искомому.

Используя данный подход можно также найти длину вектора. Известно чтобы найти длину вектора нужно использовать коэффициент коррекции который зависит от количества итераций и рассчитывается следующим образом:

p = 1;

for i = 0:N-1

p = p\*1/sqrt(1+2^(-2\*i));

end

Для 10 итераций результат будет равен 0.6073

В итоге длина вектора равна 3682\*0.6073 = 2236

Длина вектора по теореме Пифагора равна sqrt(1000^2 + 2000^2) = 2236

То есть длина вектора рассчитанной по алгоритму CORDIC совпадает с идеальной.

# Заключение

В результате выполнения проекта были изучены инструменты Vivado от фирмы Xilinx на примере реализации проекта на ПЛИС модели Artix7. Изучен язык описание аппаратуры System Verilog. Разработана математическая модель цифрового фильтра и алгоритма CORDIC. Для проверки алгоритмов применялся инструмент функциональной симуляции – Active-HDL. Так же математическая модель алгоритмов проверялась с помощью инструмента MATLAB.

# Список литературы

1. “Логическое проектирование и верификация систем на SystemVerilog”, книга, Дональд Томас.
2. URL: <https://habr.com/ru/articles/274847/>.
3. URL: <https://www.xilinx.com/video/hardware/getting-started-with-the-vivado-ide.html>.
4. “Реализация CORDIC-алгоритма на ПЛИС”, научная статья, Дмитрий Дайнеко.
5. “7 Series FPGAs Packaging and Pinout”, документация на Artix-7, Xilinx.

# Приложения

**CORDIC HDL код:**

module Cordic

#(

parameter pDataW = 16,

parameter pAngleW = 18

)

(

input logic clk,

input logic [pDataW-1:0] iX,

input logic [pDataW-1:0] iY,

output logic [pAngleW-1:0] oAngle

);

localparam [pAngleW-1:0] atan\_table\_fixp[34] = {32767,19343,10221,5188,2604,1303,652,326,163,81,41,20,10,5,3,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

localparam int pExtW = 10;

// Precorrection

logic [pDataW-1:0] sXprecorr;

logic [pDataW-1:0] sYprecorr;

logic [1:0] sPrecorrType;

logic [pAngleW-1:0] sAtan[34];

always\_ff @(posedge clk)

begin

if(signed'(iX) < 0)

if(signed'(iY) > 0)

begin

sXprecorr <= iY;

sYprecorr <= -iX;

sPrecorrType <= 2'd1;

end

else

begin

sXprecorr <= -iY;

sYprecorr <= iX;

sPrecorrType <= 2'd2;

end

else

begin

sXprecorr <= iX;

sYprecorr <= iY;

sPrecorrType <= 2'd0;

end

end

logic [pDataW+pExtW-1:0] sX [pAngleW+1];

logic [pDataW+pExtW-1:0] sY [pAngleW+1];

logic [pAngleW-1:0] sAngle [pAngleW+1];

always\_comb

begin

sX[0] <= {sXprecorr,{pExtW{1'b0}}};

sY[0] <={sYprecorr,{pExtW{1'b0}}};

sAngle[0] <= '0;

end

genvar i;

generate

for(i = 0;i < pAngleW+1;i++)

begin

always\_comb

sAtan[i] <= atan\_table\_fixp[i];

Cordic\_Core

#(

.pDataW($size(sX[0])),

.pAngleW($size(sAngle[0])),

.pIter(i)

)

Cordic\_Core\_inst(

.clk(clk),

.iX(sX[i]),

.iY(sY[i]),

.iAngle(sAngle[i]),

.iAtan(sAtan[i]),

//.iter(0),

.oX(sX[i+1]),

.oY(sY[i+1]),

.oAngle(sAngle[i+1])

);

end

endgenerate

always\_comb

begin

if(sPrecorrType == 2'd1)

begin

oAngle <= sAngle[pAngleW] + 2\*\*(pAngleW-2);

end

else if(sPrecorrType == 2'd2)

begin

oAngle <= sAngle[pAngleW] - 2\*\*(pAngleW-2);

end

else

oAngle <= sAngle[pAngleW];

end

endmodule

module Cordic\_Core

#(

parameter pDataW = 16,

parameter pAngleW = 18,

parameter pIter = 0

)

(

input logic clk,

input logic [pDataW-1:0] iX,

input logic [pDataW-1:0] iY,

input logic [pAngleW-1:0] iAngle,

input logic [pAngleW-1:0] iAtan,

//input logic [4:0] iter,

output logic [pDataW-1:0] oX,

output logic [pDataW-1:0] oY,

output logic [pAngleW-1:0] oAngle

);

logic [pDataW-1:0] sXshift;

logic [pDataW-1:0] sYshift;

always\_comb

begin

sXshift <= {{(pIter){iX[pDataW-1]}},iX[pDataW-1:pIter]};

sYshift <= {{(pIter){iY[pDataW-1]}},iY[pDataW-1:pIter]};

end

always\_ff @(posedge clk)

begin

if(signed'(iY) > 0)

begin

oX <= signed'(iX) + signed'(sYshift);

oY <= signed'(iY) - signed'(sXshift);

oAngle <= signed'(iAngle) + signed'(iAtan);

end

else

begin

oX <= signed'(iX) - signed'(sYshift);

oY <= signed'(iY) + signed'(sXshift);

oAngle <= signed'(iAngle) - signed'(iAtan);

end

end

endmodule

**CORDIC Testbench код:**

`timescale 1ns / 1ps

module Cordic\_tb;

//Parameters declaration:

defparam UUT.pDataW = 16;

parameter pDataW = 16;

defparam UUT.pAngleW = 18;

parameter pAngleW = 18;

//defparam UUT.atan\_table\_fixp = {262143,154752,81767,41506,20834,10427,5215,2608,1304,652,326,163,81,41,20,10,5,3,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//parameter atan\_table\_fixp = {262143,154752,81767,41506,20834,10427,5215,2608,1304,652,326,163,81,41,20,10,5,3,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//Internal signals declarations:

reg clk;

reg [pDataW-1:0]iX;

reg [pDataW-1:0]iY;

wire [pAngleW-1:0]oAngle;

reg [pDataW-1:0]iXRef;

reg [pDataW-1:0]iYRef;

reg [pAngleW-1:0]oAngleRef;

// Unit Under Test port map

Cordic UUT (

.clk(clk),

.iX(iX),

.iY(iY),

.oAngle(oAngle));

initial

begin

$monitor($realtime,,"ps %h %h %h %h ",clk,iX,iY,oAngle);

iX = 1000;

iY = 2000;

end

initial

begin

clk <= 0;

#5 forever #5 clk <= ~clk;

end

always @(posedge clk)

$fscanf(fd\_X, "%d,", iXRef);

always @(posedge clk)

$fscanf(fd\_Y, "%d,", iYRef);

always @(posedge clk)

$fscanf(fd\_A, "%d,", oAngleRef);

integer fd\_X;

integer fd\_Y;

integer fd\_A;

integer i;

initial

begin

fd\_X = $fopen("./src/matlab/X\_coordinate.txt", "r");

fd\_Y = $fopen("./src/matlab/Y\_coordinate.txt", "r");

fd\_A = $fopen("./src/matlab/angle.txt", "r");

i = 1'b0;

end

endmodule

**CORDIC Matlab код:**

clc

clear;

close all;

K = 1;

for L = 0 : 33

atan\_table(L+1) = atan(K);

K = K \* 0.5;

end

for width = 1:32

atan\_table\_fixp(width, :) = round((atan\_table/atan\_table(1)) \* (2^width-1));

end

phase\_width = 18;

max\_L = phase\_width;

X = 1000;

Y = 2000;

ideal\_angle = angle(X + 1i\*Y);

precor\_type = 0;

if (X < 0)

tmp\_I = X;

if (Y > 0) % -90

X = Y;

Y = -tmp\_I;

precor\_type = 1;

else

X = -Y;

Y = tmp\_I;

precor\_type = 2;

end

end

acc\_Z = 0;

fileID\_X = fopen('X\_coordinate.txt','w');

fprintf(fileID\_X,'%d\n',X);

fileID\_Y = fopen('Y\_coordinate.txt','w');

fprintf(fileID\_Y,'%d\n',Y);

fileID\_A = fopen('angle.txt','w');

for L = 1:max\_L

Z = atan\_table(L);

tmp\_X = X;

if (Y > 0)

X = X + bitshift(abs(Y),-(L-1));

Y = Y - bitshift(tmp\_X,-(L-1));

acc\_Z = acc\_Z + Z;

else

X = X + bitshift(abs(Y),-(L-1));

Y = Y + bitshift(tmp\_X,-(L-1));

acc\_Z = acc\_Z - Z;

end

M = [X;Y];

hold on; plotv(M,'-');

end

fprintf(fileID\_A,'%d\n',acc\_Z);

fclose(fileID\_X);

fclose(fileID\_Y);

fclose(fileID\_A);

if (precor\_type == 1) % +90

acc\_Z = acc\_Z+(2^(phase\_width-2));

elseif (precor\_type == 2) % -90

acc\_Z = acc\_Z-(2^(phase\_width-2));

end

error = ideal\_angle - acc\_Z;

disp(error)

N = 10;

p = 1;

for i = 0:N-1

p = p\*1/sqrt(1+2^(-2\*i));

end

**Фильтр HDL код:**

module DelayLine(clk,rst,iData,oData);

parameter int unsigned pWidth = 8;

parameter int unsigned pLength = 4;

parameter int pCoefs[pLength] = {1,-1,1,1};

input logic clk;

input logic rst;

input logic [pWidth-1:0] iData;

output logic [pWidth\*2:0] oData;

logic [pWidth-1:0] sDelay[pLength] = '{default:0};

logic [pWidth\*2-1:0] sMult[pLength];

logic [pWidth\*2:0] sSum[pLength-1];

logic [pWidth-1:0] cfs\_sig[pLength];

always\_comb

sDelay[0] <= iData;

genvar i;

generate

for(i=1;i < pLength;i++)

begin

always\_ff @(posedge clk)

sDelay[i] <= sDelay[i-1];

end

endgenerate

generate

for(i = 0; i < pLength;i++)

begin

always\_ff @(posedge clk)

begin

sMult[i] <= signed'(sDelay[i]) \* signed'(pCoefs[i]);

cfs\_sig[i] <= signed'(pCoefs[i]);

end

end

endgenerate

always\_comb

sSum[0] <= signed'(sMult[0]) + signed'(sMult[1]);

generate

for(i = 1;i < pLength-1;i++)

begin

always\_comb

sSum[i] <= signed'(sSum[i-1]) + signed'(sMult[i+1]);

end

endgenerate

always\_ff @(posedge clk)

oData <= sSum[pLength-2];

endmodule

**Фильтр Testbench код:**

`timescale 1ns / 1ns

module DelayLine\_tb;

//Parameters declaration:

defparam UUT.pWidth = 8;

parameter pWidth = 8;

defparam UUT.pLength = 8;

parameter pLength = 8;

parameter DataLength = 32;

//localparam int pCoefs[pLength] = {-1,1,-1,-1,1,1,1,-1};

//localparam int srcData[DataLength] = {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,-1,1,1,1,-1,-1,1,-1,1,1,1,1,1,1,1,1};

localparam int pCoefs[pLength] = {-1,1,-1,-1,1,1,1,-1};

localparam int srcData[DataLength] = {0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

//Internal signals declarations:

reg clk;

reg rst;

reg [pWidth-1:0]iData;

wire [pWidth\*2:0]oData;

reg [pWidth\*2:0] oDataRef;

reg oDataRdEn = 0;

reg check;

reg [7:0] cnt = '0;

// Unit Under Test port map

DelayLine

#(

.pCoefs(pCoefs)

)

UUT (

.clk(clk),

.rst(rst),

.iData(iData),

.oData(oData)

);

initial

begin

$monitor($realtime,,"ps %h %h %h %h ",clk,rst,iData,oData);

rst = 1'b0;

iData = 0;

cnt = 0;

end

initial

begin

clk <= 0;

#10 forever #10 clk <= ~clk;

end

always @(posedge clk)

if (cnt == $size(srcData)-1)

cnt <= '0;

else

cnt <= cnt + 1;

always @(posedge clk)

//iData <= srcData[cnt];

scanRet = $fscanf(fd\_in, "%d,", iData);

initial

#60 oDataRdEn = 1'b1;

always @(posedge clk)

if (oDataRdEn == 1'b1)

#1 $fscanf(fd\_out, "%d,", oDataRef);

always @(posedge clk)

if (oDataRef == oData)

check <= 1'b1;

else

check <= 1'b0;

reg signed [15:0] A[0:16383];

integer scanRet;

integer fd\_in;

integer fd\_out;

integer i;

initial

begin

fd\_in = $fopen("./src/matlab/filter\_in.txt", "r");

fd\_out = $fopen("./src/matlab/filter\_out.txt", "r");

i = 0;

//do begin

// scanRet = $fscanf(fd, "%d,", A[i]);

// i = i + 1;

// end while (scanRet > 0);

end

endmodule

**Фильтр Matlab код:**

clc;

clear;

cfs = [-1,1,-1,-1,1,1,1,-1]; % Коэффиценты фильтра

filter\_in = randsrc(1,100,[-10:10]); % Входные данные фильтра

% filter - библеотечная функция Matlab, реализующая фильтр

filter\_out = filter(cfs,1,filter\_in);

fileID = fopen('filter\_in.txt','w'); % Создаем файл для записи входных данных фильтра

fprintf(fileID,'%d\n',filter\_in);

fclose(fileID);

fileID = fopen('filter\_out.txt','w'); % Создаем файл для записи выхода фильтра

fprintf(fileID,'%d\n',filter\_out);

fclose(fileID);

subplot(2,1,1);

plot(filter\_in);

subplot(2,1,2);

plot(filter\_out);