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

2022

Оглавление

[1. Типы шумов и их природа 4](#_Toc134552041)

[1.1. Тепловой шум или шум Джонсона-Найквиста 6](#_Toc134552042)

[1.2. Генерационно-рекомбинационный шум 7](#_Toc134552043)

[1.3. Шум типа или фликкер-шум 8](#_Toc134552044)

[2. Методы измерения шумов и установки 11](#_Toc134552045)

[2.1. Метод измерения напряжения (тока) фотосигнала и напряжения (тока) шума ФЭПП и (ФПУ) 11](#_Toc134552046)

[2.2. Измерение напряжения шума К54.410 14](#_Toc134552047)

[2.3. Исследования спектральной плотности мощности шума фотоприемника 16](#_Toc134552048)

[3. Выбор АЦП 18](#_Toc134552049)

[4. Преобразование Фурье 21](#_Toc134552050)

[5. Устройство устройства 25](#_Toc134552051)

[5.1. АЦП 26](#_Toc134552052)

[5.2. LVDS 28](#_Toc134552053)

[5.3. Модуль FourierTransform 30](#_Toc134552054)

[5.3.1. IBUFDS 33](#_Toc134552055)

[5.3.2. resync\_nrst 33](#_Toc134552056)

[5.3.3. spi2axi 34](#_Toc134552057)

[5.3.4. HerzelRegs 36](#_Toc134552058)

[5.3.5. Angel 38](#_Toc134552059)

[5.3.6. Cordic 39](#_Toc134552060)

[5.3.7. DataScale 41](#_Toc134552061)

[5.3.8. Herzel 42](#_Toc134552062)

[6. Приложение 1 43](#_Toc134552063)

[6.1. АЦП 49](#_Toc134552064)

[6.2. LVDS 55](#_Toc134552065)

[6.3. Модуль преобразования Фурье 56](#_Toc134552066)

[6.3.1. FourierTransform 56](#_Toc134552067)

[6.3.2. IBUFDS 60](#_Toc134552068)

[6.3.3. resync\_nrst 60](#_Toc134552069)

[6.3.4. spi2axi интерфейс 61](#_Toc134552070)

[6.3.5. HerzelRegs 64](#_Toc134552071)

[6.3.6. Angel 69](#_Toc134552072)

[6.3.7. Cordic 71](#_Toc134552073)

[6.3.8. DataScale 75](#_Toc134552074)

[6.3.9. Herzel 76](#_Toc134552075)

[7. Источники 78](#_Toc134552076)

# Типы шумов и их природа

Шум является одной из основных характеристик фотоприемника, определяющей пороговую чувствительность то есть минимальный полезный сигнал, который равен или несколько превышает шум.

Шумом принято называть самопроизвольные флуктуации напряжения или тока на выходных клеммах прибора. Можно выделить две группы шумов:

Радиационный (внешний) шум – флуктуации потока излучения, падающего на фотоприемник, приводят к флуктуациям напряжения сигнала. Этот источник шума принципиально неустраним и характерен для любого типа фотопроводника. Минимизация шума этого вида достигается выбором оптимальной конструкции оптической системы, где используется фотоприемник. Шум, обусловленный флуктуациями потока фонового излучения, является идеальным пределом для фотоприемника. Реальный фотопреобразователь всегда обладает избыточным (внутренним), свойственным для данного типа фотоприёмника шумом, который зачастую намного превышает фоновый шум.

Спектральная плотность этого шума равна:

Реальный фотопреобразователь всегда обладает избыточным, внутренним, свойственным для данного типа ФП-шумом, который зачастую намного превышает фоновый шум.

Внутренний шум – шум возникающий в фотоприёмнике. Флуктуационные явления (шум) в полупроводниковых приборах обусловлены случайным характером происходящих в них физических процессов.

Для фотоприемника на основе фоторезистора характерными являются следующие виды шумов: джонсоновский (тепловой шум), генерационно-рекомбинационный шум (ГР), шум вида 1/f (фликершум), а также в случае плохих контактов дробовой или барьерный шум.

Существует несколько видов внутренних шумов. В фоторезистах шум определяется тремя составляющими:

* шум типа
* тепловой шум
* генерационно-рекомбинационный

Также сущёствует дробовой (барьерный) шум, но он в основном проявляется в фотодиодах. В фоторезистах он может проявиться только в случае плохих контактов, поэтому рассматриваться не будет.

Поскольку среднеарифметическое от случайной величины, какой является напряжение шумов равно нулю, для характеристики шумов используют среднеквадратичное, т. е. мощность шума. Под спектральной плотностью шума понимают следующую величину:

где под частотной зависимостью подразумевают фурье-образ временных флуктуаций напряжения:

Каждый из видов шума имеет свою природу и определенный вид спектральной плотности . Исследуя спектральную плотность шума в различных условиях (температура, освещенность и т. д.), можно делать выводы о природе шума и методах его уменьшения.

Общий спектр плотности мощности шума для фоторезиста следующий:

На рисунке 1 приведен классический вид спектра плотности мощности шума фотоприемника.

Зная такую характеристики для конкретных приемников, можно выбрать частоту модуляции сигнала и полосу пропускания системы так, чтобы по возможности уменьшить влияние собственных шумов на чувствительность прибора.

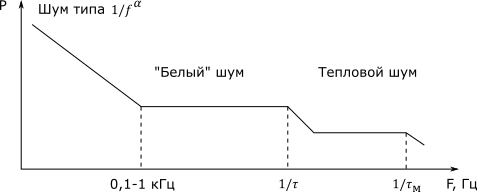


Рисунок Структура спектра плотности мощности шума фотоприемника

## Тепловой шум или шум Джонсона-Найквиста

Тепловой шум или шум Джонсона-Найквиста определяется процессами флуктуации проводимости образца за счет флуктуации числа носителей и/или их подвижности. Случайные изменения скорости свободных носителей заряда, приводят к флуктуациям их локальной плотности и дрейфовой подвижности, в результате чего появляются случайные микроскопические диффузионные токи.

Поэтому в системе, содержащей свободные носители заряда, существует флуктуирующий ток, а на электрических контактах возникает флуктуирующее напряжение.

Спектр плотности мощности этого шума можно определить по формуле Найквиста:

– среднеквадратическое значение напряжения шума

– полоса пропускания измерительного тракта

– рабочая температура ФР

– электрическое сопротивление ФР

## Генерационно-рекомбинационный шум

В среднем диапазоне частот усилительного устройства можно выделить участок «белого» шума, который не зависит от частоты. Это – генерационно-рекомбинационный шум.

Генерационно-рекомбинационный шум обусловлен спонтанными флуктуациями скоростей генерации и рекомбинации носителей заряда. Генерационно-рекомбинационный шум наблюдается в полупроводниках различных типов и многочисленных полупроводниковых приборах. В его основе лежат флуктуации концентрации свободных носителей заряда, вызываемые флуктуациями скоростей процессов генерации и рекомбинации. Шум данного типа может вызываться как прямой генерацией-рекомбинацией (через запрещенную зону), так и генерационно-рекомбинационными процессами через разрешенные уровни в запрещенной зоне, образованные дефектами структуры, иначе говоря, генерационно-рекомбинационными процессами через ловушки. Шум, вызванный прямыми генерационно-рекомбинационнми процессами, существенен в основном для материалов с узкой запрещенной зоной и низкой концентрацией дефектов, образующих разрешенные уровни в запрещенной зоне. В большей части случаев генерационно-рекомбинационный шум обусловлен процессами на ловушечных центрах. Это связано с тем, что для генерации и рекомбинации носителей через разрешенные уровни в запрещенной зоне требуется меньше энергии, чем для прямых генерационно-рекомбинационных процессов.

Данный шум практческие не наблюдается при малом времени жизни носителей заряда.

При возбуждении избыточных носителей в полупроводнике случайный характер электронных переходов между зонами, либо между зонами и локальными энергетическими уровнями, обусловленный статистической природой процессов генерации, рекомбинации и прилипания носителей тока, вызывает генерационно-рекомбинационный шум. В этих случаях наблюдаются флуктуации проводимости полупроводника, а следовательно и флуктуации напряжения.

Поскольку носители могут вытягиваться из объема внешним электрическим полем, рекомбинировать на поверхностных центрах и т. д., генерационно-рекомбинационный шум может проявляться по-разному и сильно зависит от внешних условий.

– объём полупроводникового слоя

– концентрация электронов и дырок

## Шум типа или фликкер-шум

При протекании постоянного тока через полупроводниковый материал между его контактами возникает значительное низкочастотное шумовое напряжение, которое действует наряду с тепловым и дробовым шумом.

Этот шум имеет обратную зависимость от частоты:

– коэффициенты, зависящие от многих факторов параметра A, показателя степени α и частоты перехода шума  в “белый” шум. В литературе приводятся данные, что A варьируется от 2·10-6 до 5·10-2; значение α – от 0,5 до 3.

При измерении фликкер-шума, возникающего в объеме полупроводникового материала, часто оказывается, что этот шум маскируется фликкер-шумом, возникающим в концевых контактах. Для исключения этого контактного шума генератор постоянного тока присоединяют к концевым контактам, а шум измеряют между двумя зондами, не несущими постоянного тока.

Данный тип шума сильно зависит от условий окружающей среды, но при этом слабо зависит от температуры.

Фликкер-шум имеет важное значение для полупроводников с малым объемом или для тонкослойных полупроводников.

Шум возникает из-за флуктуации плотности носителей заряд, преимущественно основного типа.

Есть четыре теории происхождения этих шумов: контактная теория; теория модуляции проводимости полупроводника; теория флуктуации концентрации носителей и теория флуктуации подвижности носителей.

Модель флуктуации концентрации носителей, предложенная Мак-Уортером, объясняет шум флуктуациями числа носителей заряда в результате захвата части носителей «глубоко лежащими ловушками», которые могут находиться в слое над поверхностью полупроводника.

Поверхностные ловушки, которые обусловливают частотную зависимость полевого эффекта, участвуют и в образовании фликкер-шума.

Захват носителей ловушками имеет два следствия. Захват вызывает непосредственное изменение числа свободных носителей основного типа, имеющихся в данном образце. И происходит косвенное изменение числа носителей в образце. Изменение заполнения поверхностных ловушек влияет на генерацию неосновных носителей в центрах быстрой рекомбинации вблизи поверхности.

Данная теория объясняет также сильное влияние на величину шума окружающей среды, которая влияет на состояние поверхности.

Альтернативная модель флуктуации подвижности носителей Хоухе не связана с поверхностными явлениями, а определяется только рассеянием носителей на акустических фононах.

Формула для расчёта частотной характеристики шума из теории Хоухе:

– число носителей заряда

– объем исследуемого образца

– концентрация носителей

– постоянная Хоухе.

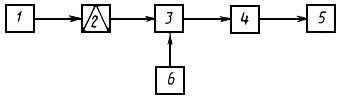
# Методы измерения шумов и установки

Шум фотоприемников (напряжение или ток) измеряют, используя либо узкополосные усилители с полосой пропускания , либо метод дискретных измерений шумового аналогового сигнала с последующей математической обработкой по алгоритму преобразования Фурье для получения частотного спектра мощности шума.

## Метод измерения напряжения (тока) фотосигнала и напряжения (тока) шума ФЭПП и (ФПУ)

Измерение следует проводить на установке, структурная схема которой приведена на черт.5.

Конструкция измерительной установки должна исключать влияние рассеянного света и посторонних источников излучения на результаты измерений.

  
Черт.5 1 - источник излучения; 2 - модулятор; 3 - испытуемый ФЭПП (ФПУ); 4 - усилительное устройство; 5 - регистрирующий прибор; 6 - источник питания ФЭПП (ФПУ)

Тип источника излучения следует выбирать с учетом параметров измеряемого изделия, установленных в стандартах или ТУ на изделие конкретных типов. Рекомендуется применять следующие источники излучения:

1) лампу накаливания типа СИС или РН при цветовой температуре (2856 ± 100) К;

2) полный излучатель - абсолютно черное тело (АЧТ) с температурой полости (500 ± 2) К или =(1273 ± 15) К;

Требования к источнику импульсного излучения - Тип источника излучения, тип контрольного ФЭПП (ФПУ), длительность и форма импульсов, уровень мощности излучения следует выбирать с учетом параметров измеряемого изделия, установленных в ТУ на ФЭПП (ФПУ) конкретных типов.

В качестве источников излучения рекомендуется использовать полупроводниковые лазерные диоды или светодиоды, нестабильность выходной мощности которых не должна выходить за пределы интервала ±5% за время измерений.

В состав источников излучения для ослабления потока излучения или для увеличения плотности мощности могут входить ослабители (аттенюаторы), зеркала, линзы, объективы и другие оптические элементы. Влияние оптических элементов не должно учитываться, если они изменяют коэффициент использования излучения за счет изменения его спектрального состава не более чем на 2%. Под коэффициентом использования излучения следует понимать

где  - относительная спектральная характеристика чувствительности ФЭПП, отн. ед.

 - спектральная плотность потока излучения, Вт·см·мкм.

Максимальное значение потока излучения, падающего на ФЭПП, должно выбираться из условия работы ФЭПП на линейном участке его энергетической характеристики.

Частота модуляции должна быть (800 ± 12) Гц. Конструкция модулятора должна быть такой, чтобы закон изменения потока излучения приближался к синусоидальному. Нестабильность частоты модуляции не должна выходить за пределы интервала ±1,5%.

Тип усилительного устройства должен выбираться в зависимости от требований к частоте и форме модуляции потока излучения, уровня регистрируемого сигнала, вида измеряемого параметра и других требований, приведенных в ТУ на ФЭПП (ФПУ) конкретных типов. В зависимости от этих требований в состав усилительного устройства могут входить селективные и широкополосные усилители, а также регистрирующие приборы.

Измерители тока и напряжения должны обеспечивать измерение, погрешность которых не должна выходить за пределы ±3%. Полоса пропускания таких приборов при измерении напряжения (тока) шума должна не менее чем в десять раз превышать эквивалентную шумовую полосу измерительной цепи.

Источник питания ФЭПП должен обеспечивать установление напряжения питания ФЭПП с погрешностью, которая не должна выходить за пределы интервала ±3%, если иное не оговорено в ТУ на ФЭПП конкретного типа.

Коэффициент пульсации должен находиться в пределах ±10% и не оказывать влияние на результат измерения параметров ФЭПП.

Климатические условия проведения измерений Климатические условия окружающей среды, в которых проводят измерение, должны соответствовать следующим требованиям, если иные не оговорены в ТУ на ФЭПП или ФПУ конкретного типа:

|  |  |
| --- | --- |
|  |  |
| температура, °С | 20±5 |
| относительная влажность, % | 65±15 |
| атмосферное давление, кПа | 100±4 (750±30 мм рт.ст.) |

К испытуемому ФЭПП подключают сопротивление нагрузки, значение которого указывается в ТУ на ФЭПП конкретного типа. Устанавливают режим питания на ФЭПП (ФПУ) в соответствии с требованиями ТУ на ФЭПП (ФПУ) конкретных типов и регистрируют значение напряжения (тока) шума по показаниям регистрирующего измерительного прибора. Продолжительность измерения и значения напряжения (тока) шума должны соответствовать требованиям ТУ на ФЭПП (ФПУ) конкретных типов. Если эти требования отсутствуют, то напряжение (ток) шума следует регистрировать по максимальным повторяющимся показаниям прибора за время не менее 10 с.

Если напряжение (ток) шума близки к напряжению (току) шума измерительной установки, то сначала необходимо зарегистрировать напряжение шума без подачи напряжения на ФЭПП, затем суммарное напряжение шума при подаче на него напряжения.

Напряжение шума в вольтах следует вычислять по формуле:

где  - суммарное напряжение шума при подаче напряжения, питания ФЭПП, В;

 - напряжение шума без подачи напряжения питания ФЭПП, В.

Если в состав измерительной установки входит преобразователь ток-напряжение, то сначала необходимо измерить ток без подключения ФЭПП, а затем ток с подключенным ФЭПП.

На испытуемый ФЭПП (ФПУ) подают модулированный поток излучения и регистрируют напряжение (ток) фотосигнала по показаниям регистрирующего прибора.

## Измерение напряжения шума К54.410

Основными измерительными стендами для измерения фотоэлектрических параметров и оценки качества промышленных ФР в России являются две установки: К54.410, разработанная НПО «Орион» и изготовленная на заводе «Кварц» и изготовленная на заводе «Сапфир» установка ИФР-3. Обе установки позволяют определить значение напряжения сигнала и шума на разных частотах (установка К54.410 на частотах 400, 800 и 1200 Гц, а установка ИФР-3 на частотах 800 и 2000 Гц). Структурная схема установок показана на рисунке 2.1.



Рисунок 2.1 – Структурная схема установки для измерения параметров ФЧЭ

В состав блока излучателя (БИ) входят: источник излучения – АЧТ с подключенными к нему контактным термометром и блоком питания (БП1), благодаря которым осуществляется точная настройка температуры АЧТ; модулятор излучения – перфорированный диск, вращающийся с фиксированной частотой (в зависимости от модели она варьируется от 400 до 2000 Гц); заслонка, которая перекрывает поток излучения во время измерения шумового сигнала на ФП.

В состав измерительного блока (ИБ) входит: БПУ – блок предварительного усиления; БП2 – блок питания измерительной части установки; ИУ – измерительный усилитель; ОСЦ – осциллограф; В – вольтметр; ФП – фотоприемник, в специальном металлическом корпусе.

При измерении параметров ФЧЭ в качестве источника излучения обычно используется модель абсолютно черного тела (АЧТ), представляющая собой изотермическую замкнутую полость с площадью отверстия много меньше внутренней поверхности полости.

Излучение, соответствующее заданной температуре АЧТ, проходит через модулятор и при открытой заслонке попадает на фоточувствительную площадку образца. При этом происходит изменение концентрации носителей заряда, изменяется проводимость фоточувствительного слоя. Это изменение проводимости фиксируется как изменение напряжения, падающего на нагрузочном сопротивлении Rн, включенном последовательно с ФР.

Действующее значение изменения напряжения измеряется с помощью встроенного вольтметра, форма сигнала контролируется осциллографом.

Измерение шумовых характеристик идентично измерению параметров сигнала, за исключением положения заслонки, установленной после диафрагмы модулятора. Для снятия шумовых характеристик она закрыта и тем самым перекрывает поток излучения от АЧТ.

Измерив напряжение сигнала и шума можно определить такие параметры как вольтовая чувствительность и обнаружительная способность. А также значение постоянной времени релаксации фотопроводимости (времени жизни носителей тока) (*τ*), используя значения фотосигнала фотоприемника на двух частотах.

## Исследования спектральной плотности мощности шума фотоприемника

В данной установке используется метод дискретных измерений шумового аналогового сигнала с последующей математической обработкой по алгоритму быстрого преобразования Фурье для получения частотного спектра мощности шума. Структурная схема установки приведена на рисунке.

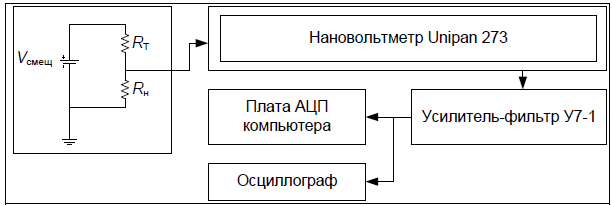


Рисунок Структурная схема установки для снятия СПМШ

Исследуемый ФП находится в заземленном металлическом стакане, который позволяет подвести требуемую температуру фона и обеспечить экранировку от внешних наводок. В параллель с ним стоит нагрузка 𝑅н. Обычно она выбирается согласованной с темновым сопротивление ФП.

С нагрузочного резистора снимается сигнал, затем он усиливается нановольтметром Unipan 273, который способен работать в широком диапазоне частот от 0,3 до 105 Гц. Далее сигнал поступает на У7-1 – широкополосный усилитель с регулируемым дискретно коэффициентом усиления и узкополосным фильтром с частотами среза 10, 102, 103, 104, 105 Гц.

Усиленный и отфильтрованный сигнал поступает на 12-ти разрядный последовательный АЦП компьютера, имеющий полосу пропускания 0-250 кГц и рассчитанный на напряжения сигнала до 7 В, которое контролируется осциллографом.

Процесс измерения заключается в последовательной регистрации и запоминании 2N шумовых сигналов через интервал времени Т.

Спектр плотности мощности шума вычисляется по алгоритму быстрого преобразования Фурье.

Измерения на установке для снятия СПМШ начинаются с определения шума измерительного тракта. При этом на входе нановольметра Unipan обеспечиваются “холостой ход” и “короткое замыкание” и снимается спектр плотности мощности шума.

# Выбор АЦП

Существуют АЦП встроенные в микроконтроллеры, ПЛИСы, микропроцессоры, системы-на-кристалле,  АЦП последовательного приближения (SAR) и сигма-дельта-версии. Конвейерные АЦП используются в тех приложениях где требуется высочайшая скорость выборок. Диапазон скоростей выборок АЦП лежит в пределах от 10 выб/с до свыше 10 Гвыб/с. Разрядность от 8 до 32 бит.

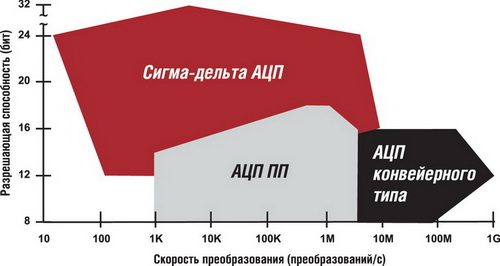


Рисунок соответствие скорости преобразования и разрешающей способности для трех основных типов интегральных аналого-цифровых преобразователей [1]

**АЦП последовательного приближения** (SAR АЦП) выпускаются в широком диапазоне значений разрешения и скорости. Разрядность, как правило, лежит в пределах 6…8 до 20 бит, скорость же – от нескольких Квыб/с до 10 Мвыб/с. SAR АЦП – хороший выбор для приложений со средним диапазоном скоростей. Они не столь быстродействующие, как конвейерные АЦП, но их быстродействие немного выше, чем у сигма-дельта-АЦП.

АЦП последовательного приближения (SAR) работает по принципу весов. В преобразователе последовательного приближения в роли неизвестного веса выступает входной сигнал, из которого происходит выборка и хранение. Затем это напряжение сравнивается с последующими известными значениями напряжения, и результаты выводятся преобразователем.

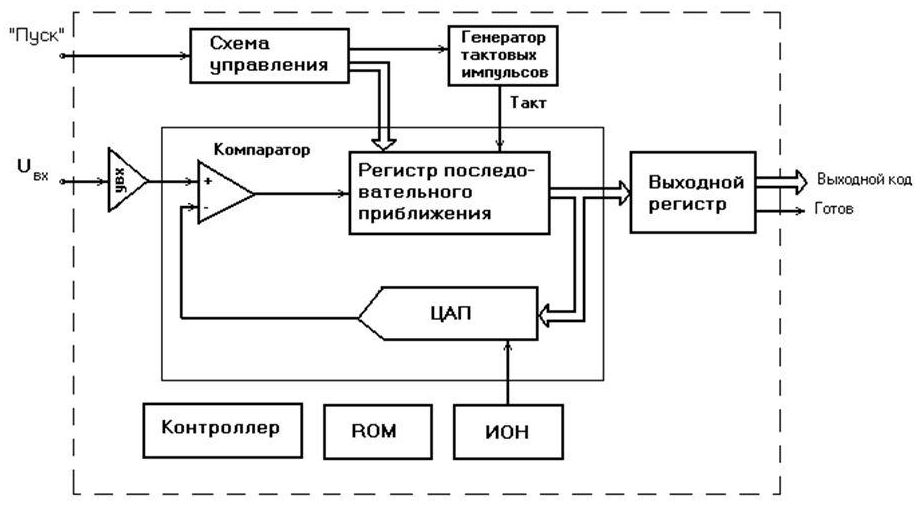


Рисунок АЦП последовательного приближения

**Сигма-дельта АЦП** позволяет уменьшить погрешность, вносимую шумами. При этом можно повысить разрешающую способность за счет усреднения результатов измерений. Сигма-дельта преобразователь измеряет входной сигнал в течение определенного времени и формирует цифровой код, соответствующий среднему значению сигнала за это время.

Если необходима повышенная точность за счет более высокого уровня семплирования или максимальное значение эффективного количества бит, наилучшим выбором станет сигма-дельта-АЦП, особенно для малошумящих точных приложений. Когда скорость не так критична, передискретизация и формирование шума в сигма-дельта-АЦП дают очень высокую точность.

Сегодняшний результат этого процесса – очень качественные АЦП с разрядностью до 24 или 32 бит и частотой дискретизации от 10 выб/с до 10 Мвыб/с.

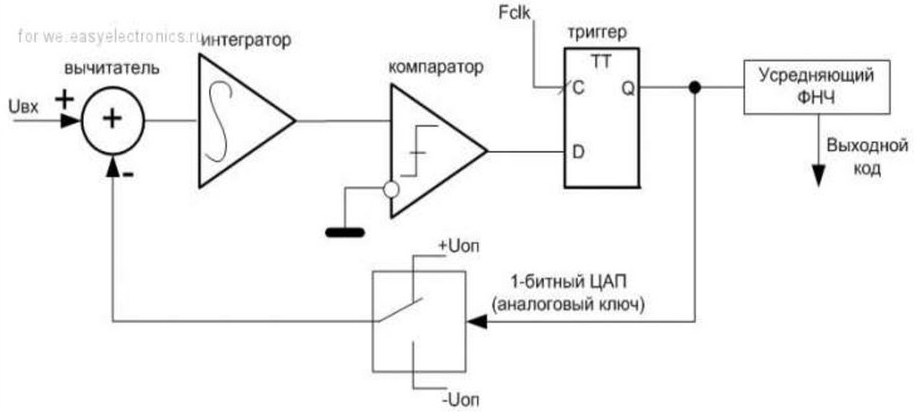


Рисунок Сигма-дельта АЦП

**Конвейерный АЦП** - принцип многоступенчатой обработки входного сигнала позволяет найти оптимальный компромисс между разрядностью, быстродействием и сложностью топологии АЦП. На первом шаге производится грубое преобразование (с низким разрешением). Далее определяется разница между входным сигналом и аналоговым сигналом, соответствующим результату грубого кода. На втором шаге найденная разница подвергается преобразованию, и полученный код объединяется с грубым кодом для получения полного выходного цифрового значения.

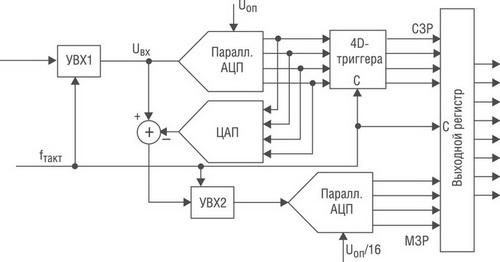


Рисунок АЦП конвейерного типа

Для реализации был выбран АЦП последовательного приближения в виду его компромиссных свойств – быстродействие, точность, простота.

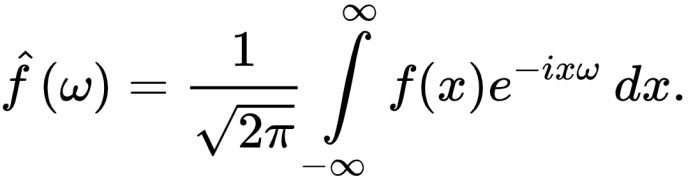
Схема и осциллограммы работы АЦП приведены в приложении.

Полученная частота выборки может быть более 400 КГц, чего достаточно для оцифровки шумов фоторезиста частотой 200 КГц.

Разрядность – 8 бит, но может быть легко масштабирована добавлением новых триггеров.

# Преобразование Фурье

Преобразование Фурье — операция, сопоставляющая одной функции вещественной переменной другую функцию вещественной переменной. Эта новая функция описывает коэффициенты («амплитуды») при разложении исходной функции на элементарные составляющие — гармонические колебания с разными частотами.



То есть данное преобразование раскладывает исходный сигнал на гармонические функции с разными частотами.

По полученным частотам далее можно построить спектральную характеристику входного сигнала и далее в отдельных случаях построить и спектральную характеристику мощности. В нашем случае СПМШ фоторезиста.

На рис. 4 примерно представлено преобразование временного сигнала в частотный.

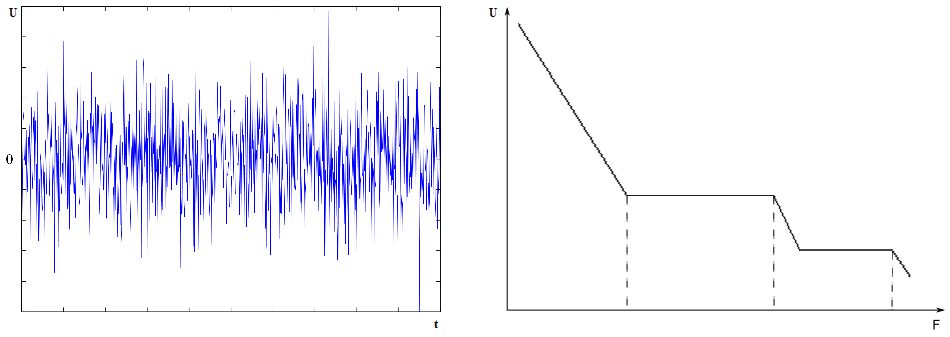


Рисунок Входной сигнал слева, преобразованный через Фурье справа

Для цифровой схемотехники применяют дискретное преобразование Фурье:

Где – входной дискретный сигнал (шум фоторезиста)

- количество значений входного дискретного сигнала

- k-ое время дискретного сигнала

- частота

Данное преобразование, то есть нахождение из можно представить как вычисление среднего значения (центра масс) всех элементов дискретного сигнала на комплексной плоскости при текущей .

Если не совпадает с частотой сигнала, то среднее значение на комплексной плоскости будет равно нулю. И на спектральной характеристике при текущей будет ноль. Если совпадает с частотой сигнала, то среднее значение на комплексной плоскости не будет равно нулю рис. 9. И на спектральной характеристике при текущей будет пик.

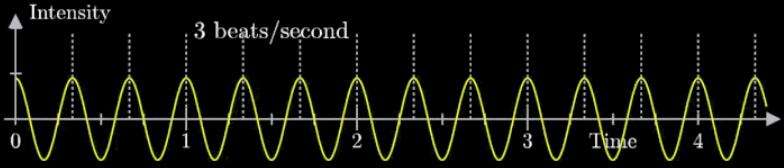


Рисунок Входной сигнал

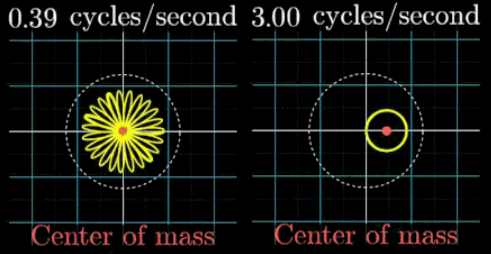


Рисунок Представление суммы частей сигнала на комплексной плоскости

То есть по сути дела алгоритм нахождения Фурье образа дискретного сигнала сводится к суммированию всех комплексных чисел при текущей частоте и делении на количество суммирований.

В нашем случае дискретный сигнал будет сигналом с АЦП. Данные с АЦП будут поступать последовательно по мере их оцифровки.

Поступивший код должен будет затем умножен на

После ожидать появления нового оцифрованного кода. Новый код также умножается на коэффициент, и далее суммируется с предыдущим значением.

Так продолжается до получения необходимого количества значений. После модуль суммы делится на для получения среднего значения.

Таким образом, алгоритм следующий:

* Приём оцифрованного кода с АЦП
* Вычисление и
* Умножение кода на комплексные коэффициенты
* Сложение с предыдущим значением
* Вычисление модуля
* Деление для получения среднего

При этом будет вычислено значение только для одной частоты. Для вычисления на другой частоте надо будет повторить алгоритм. Для ускорения процесса можно либо добавить конвейер либо сделать несколько параллельных модулей.

Но всё же потребуется N операций cos, sin, умножения для вычисления одной частоты. Что довольно затратно. Поэтому для вычислений спектра будет использоваться алгоритм Герцеля. Данный алгоритм также позволяет вычислить значения для отдельного набора частот, но с меньшими вычислительными затратами.

Учитывая, что СПМШ фоторезиста обычно строится в логарифмическом масштабе, в нашем случае от 1 Гц до 100 КГц. То для построения графика можно использовать соответствующие частоты 1, 10, 100…100000.

То есть, понадобится 7 вычислений или для большей точности провести вычисление ещё и промежуточных значений. Тогда понадобится 13 вычислений частот.

Также для облегчения вычислений можно заранее определиться с числом измерений, то есть с частотой выборки. В данном случае она должна составлять как минимум 200 КВыб/с для оцифровки сигнала частотой в 100 КГц. Таким образом, время одной выборки - .

Далее полученные данные будут переданы в графическую систему отображения информации для построения графика.



Рисунок Теоретическая СПМШ

# Устройство устройства

Структура проекта выглядит следующим образом:

* АЦП
  + АЦП
  + LVDS
* FourierTransform
  + IBUFDS
  + resync\_nrst
  + spi2axi\_wrap
  + HerzelRegs
  + Angel
  + Cordic
  + DataScale
  + Herzel

## АЦП

Полученная частота выборки может быть более 200 КГц, чего достаточно для оцифровки шумов фоторезиста частотой 100 КГц.

Разрядность – 8 бит, но может быть легко масштабирована добавлением новых триггеров.

Схема моделировалось с учётом задержек логических элементов. А также с применением библиотечных реально существующих моделей ОУ (AD8045) и компаратора (MAX9010).

Моделирование проводилось в MicroCap.

Интерфейс модуля представлен на рис.1 и в таблице 1.

Таблица Описание сигналов АЦП

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Разрядность | Описание |
| ANALOG | ВХОД | [0] | Аналоговый сигнал который необходимо оцифровать |
| CLK | ВХОД | [0] | Тактовый сигнал |
| START | ВХОД | [0] | Сигнал запуска преобразования |
| RESET | ВХОД | [0] | Сигнал сброса |
| VALID | ВЫХОД | [0] | Готовность данных |
| ADC\_Q | ВЫХОД | [7:0] | Выходной цифровой код |

Предварительно произведя инициализацию всех триггеров с помощью RESET, преобразование начинается с приходом импульса на START.

ANALOG сравнивается с начальным значением напряжения с ЦАП рис.4 на компараторе. Которое в начале равняется половине максимального напряжения. Максимальное напряжение в свою очередь регулируется резисторами ЦАП.

В зависимости от результата текущий выбранный с помощью сдвигового регистра бит меняется или нет на схеме изменения кода рис.3.

Изменённый или нет код поступает на ЦАП и снова сравнивается с входным сигналом.

Так продолжается пока не будут пройдены все биты. После полученный код запишется в выходные регистры рис.3 и поступит на выходы ADC\_Q.

Также установится сигнал VALID, говорящий о готовности передачи оцифрованных данных.

Далее начнётся оцифровка нового значения.

Полученный код:

Соответствующее ему напряжение:

## LVDS

Интерфейс LVDS широко применяется для высокоскоростной передачи данных и распределения тактовых сигналов по соединительным линиям, кабелям и межплатным соединениям, а также в каналах связи в пределах одной печатной платы.

Интерфейс LVDS обладает следующими достоинствами:

* Скорость передачи — до 1 Гбит/с и выше.
* Пониженный уровень электромагнитных излучений.
* Повышенная устойчивость к шуму.
* Низкое энергопотребление.

В LVDS одна сигнальная линия является неинвертирующей (то есть при передаче логической единицы на ней устанавливается высокий уровень напряжения, а при передаче логического нуля — низкий уровень напряжения), а другая линия — инвертирующей (то есть сигнал, передаваемый по ней, является комплементарным по отношению к сигналу в неинвертирующей линии). Разность напряжений между двумя сигнальными линиями называется дифференциальным напряжением — VOD. Сигнал в каждой из двух сигнальных линий имеет максимальный размах |VOD| и центрирован относительно синфазного напряжения VOC.

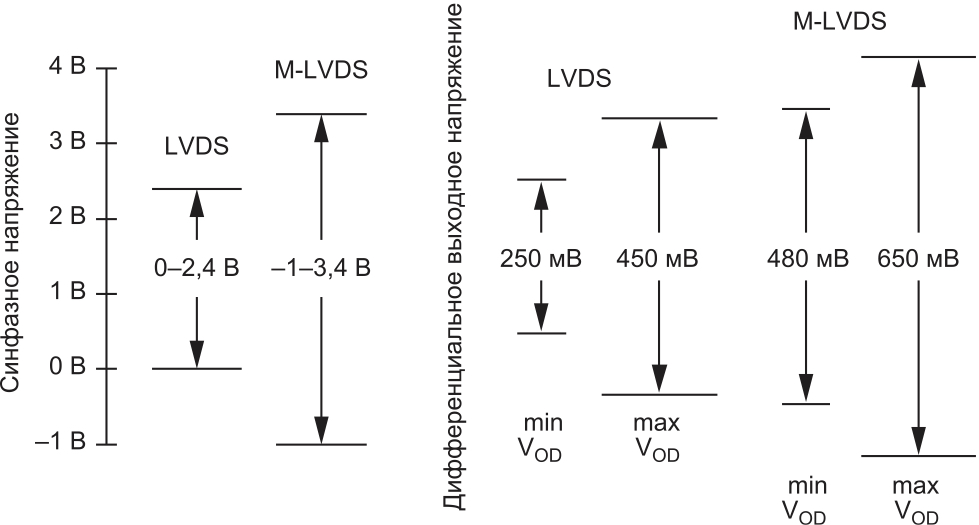


Рисунок Уровни сигналов в LVDS и M-LVDS

Выходные данные с АЦП поступают на вход LVDS драйвера и далее по дифференциальной линии на вход FPGA. Большинство плат поддерживают стандарт входа LVDS, необходимо только в файле с ограничениями (constraints.xdc) указать необходимый стандарт у соответствующего порта. А также добавить входные дифференциальные буферы (IBUFDS) в Verilog дизайне, чтобы плата смогла правильно всё развести.

Данные LVDS драйверы устанавливаются для каждого бита выходного кода АЦП и параллельно передаются на FPGA.

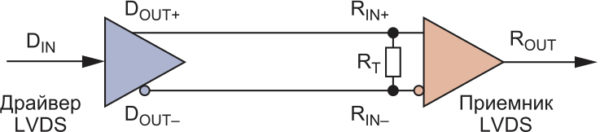


Рисунок Канал связи между приёмником и передатчиком

## Модуль FourierTransform

Данный модуль является топ-уровнем всего RTL дизайна.

Интерфейс модуля представлен в таблице 2.

Таблица Интерфейс модуля FourierTransform

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Разрядность | Описание |
| rstn | ВХОД | [0] | Сигнал сброса |
| clk | ВХОД | [0] | Тактовый сигнал |
| spi\_sck | ВХОД | [0] | Тактовый сигнал SPI |
| spi\_ss\_n | ВХОД | [0] | Разрешение приёма/передачи данных SPI |
| spi\_mosi | ВХОД | [0] | Данные от ведущего |
| spi\_miso | ВЫХОД | [0] | Данные от ведомого |
| enable\_p  enable\_n | ВХОД | [0] | Разрешение приёма данных |
| sample\_p  sample\_n | ВХОД | [7:0] | Входной цифровой код |

Модуль включает в себя следующие подмодули:

* + IBUFDS
  + resync\_nrst
  + spi2axi
  + HerzelRegs
  + Angel
  + Cordic
  + DataScale
  + Herzel

Для управления модулем предусмотрен SPI интерфейс.

Соответствующие сигналы SPI интерфейса поступают на модуль spi2axi для преобразования SPI транзакции в AXI-lite транзакции, которые в свою очередь идут на модуль HerzelRegs для дешифрации и записи/считывания соответствующих регистров.

Карта памяти топ-уровня представлена в таблице 3.

Таблица Карта памяти модуля FourierTransform

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Адрес | Тип | Сброс | Название | Описание |
| 0x00 | RW | 0x29042023 | VERSION | Версия модуля |
| 0x04 | RW | 0xF0F0F0F0 | DEBUG | Тестовый регистр |
| 0x08 | RW | 0x0 | FREQ\_1 | Частота 1 |
| 0x0C | RW | 0x0 | FREQ\_2 | Частота 2 |
| 0x10 | RW | 0x0 | FREQ\_3 | Частота 3 |
| 0x14 | RW | 0x0 | FREQ\_4 | Частота 4 |
| 0x18 | RW | 0x0 | FREQ\_5 | Частота 5 |
| 0x1C | RW | 0x0 | FREQ\_6 | Частота 6 |
| 0x20 | RW | 0x0 | FREQ\_7 | Частота 7 |
| 0x24 | RW | 0x0 | FREQ\_8 | Частота 8 |
| 0x28 | RW | 0x0 | FREQ\_9 | Частота 9 |
| 0x2C | RW | 0x0 | FREQ\_10 | Частота 10 |
| 0x30 | RW | 0x0 | FREQ\_11 | Частота 11 |
| 0x34 | RW | 0x0 | EN\_CORDIC | Разрешение начала вычисления поворотных коэффициентов |
| 0x38 | R | 0x0 | STATUS | Регистр с информацией о стадиях вычисления:  0 бит – вычислены углы в Angel  1 бит – вычислены коэффициенты в Cordic  18-8 биты – вычислены данные по каждому модулю Herzel |
| 0x3C | R | 0x0 | DATA\_1 | Данные для частоты 1 |
| 0x40 | R | 0x0 | DATA\_2 | Данные для частоты 2 |
| 0x44 | R | 0x0 | DATA\_3 | Данные для частоты 3 |
| 0x48 | R | 0x0 | DATA\_4 | Данные для частоты 4 |
| 0x4C | R | 0x0 | DATA\_5 | Данные для частоты 5 |
| 0x50 | R | 0x0 | DATA\_6 | Данные для частоты 6 |
| 0x54 | R | 0x0 | DATA\_7 | Данные для частоты 7 |
| 0x58 | R | 0x0 | DATA\_8 | Данные для частоты 8 |
| 0x5C | R | 0x0 | DATA\_9 | Данные для частоты 9 |
| 0x60 | R | 0x0 | DATA\_10 | Данные для частоты 10 |
| 0x64 | R | 0x0 | DATA\_11 | Данные для частоты 11 |

Для начала работы сначала необходимо записать в регистры FREQ\_1- FREQ\_11 необходимые значения частот. Далее необходимо записать в EN\_CORDIC 0x1 для запуска вычисления поворотных коэффициентов. После того как коэффициенты будут вычислены в регистре STATUS выставятся соответствующие биты (0 и 1) и с регистра будет считываться 0x3.

После считывания 0x3 с регистра STATUS можно будет начинать отправлять действительные данные, используя sample\_p, sample\_n, enable\_p, enable\_n порты. Полученные данные, пройдя несколько вспомогательных будут последовательно поступать на модуль Herzel для расчёта. Спустя прихода необходимого количества данных и завершения вычислений в битах 18-8 регистра STATUS установятся 1, после чего можно будет считывать данные с регистров DATA\_1- DATA\_11.

### IBUFDS

Входы sample\_p, sample\_n, enable\_p, enable\_n являются дифференциальными и для их преобразования в обычные сигналы используются специальный модуль IBUFDS.

Данный модуль является библиотечным элементом и необходим для синтеза, но не моделирования. В отдельных симуляторах данный модуль может вызвать ошибку, для устранения этого в FourierTransform прописан специальный `define TEST, который нужно менять при синтезе и модолеровании.

Данный модуль является одним из требований для приёма FPGA LVDS сигнала. Одновременно с его установкой, в файле с ограничениями необходимо указать необходимый стандарт принимаемого сигнала:

set\_property -dict {IOSTANDARD LVDS PACKAGE\_PIN ???} [get\_ports enable\_p]

set\_property -dict {IOSTANDARD LVDS PACKAGE\_PIN ???} [get\_ports enable\_n]

### resync\_nrst

Данный модуль просто пересинхронизирует входной сигнал сброса для безопасной инициализации схемы.

### spi2axi

Данный модуль преобразует SPI транзакции в AXI-lite транзакции.

Модуль был взят из интернета [3].

Модуль работает в режиме 00: Clock polarity (CPOL) = 0, Clock Phase (CPHA) = 0. То есть Уровень ожидания равен 0, данные записываются по переднему (нарастающему) фронту тактового сигнала.

Для передачи транзакции чтения/записи необходимо соблюсти протокол – таблица 4/5.

Данный протокол используется в SPI интерфейсе spi\_if во время моделирования для создания транзакций.

Таблица SPI транзакция записи

| Байт | MOSI | MISO | Комментарий |
| --- | --- | --- | --- |
| 0 | 0x00 | 0x00 | Байт инструкции записи |
| 1 | address[31:24] | 0x00 | Адрес записи (старший байт) |
| 2 | address[23:16] | 0x00 | Адрес |
| 3 | address[15:8] | 0x00 | Адрес |
| 4 | address[7:0] | 0x00 | Адрес записи (младший байт) |
| 5 | wr\_data[31:24] | 0x00 | Запись данных (старший байт) |
| 6 | wr\_data[23:16] | 0x00 | Данные |
| 7 | wr\_data[15:8] | 0x00 | Данные |
| 8 | wr\_data[7:0] | 0x00 | Запись данных (младший байт) |
| 9 | don't care | 0x00 | Пустой байт |
| 10 | don't care | status | [2] timeout ожидания ответа на запись [1:0] BRESP AXI4 ответ на запись (действительно только при timeout = 0) |

Таблица SPI транзакция чтения

| Байт | MOSI | MISO | Комментарий |
| --- | --- | --- | --- |
| 0 | 0x01 | 0x00 | Байт инструкции чтения |
| 1 | address[31:24] | 0x00 | Адрес чтения (старший байт) |
| 2 | address[23:16] | 0x00 | Адрес |
| 3 | address[15:8] | 0x00 | Адрес |
| 4 | address[7:0] | 0x00 | Адрес чтения (младший байт) |
| 5 | don't care | 0x00 | Пустой байт |
| 6 | don't care | rd\_data[31:24] | Чтение данных (старший байт) |
| 7 | don't care | rd\_data[23:16] | Данные |
| 8 | don't care | rd\_data[15:8] | Данные |
| 9 | don't care | rd\_data[7:0] | Чтение данных (младший байт) |
| 10 | don't care | status | [2] timeout произошел во время ожидания ответа на чтение [1:0] RRESP Ответ на чтение AXI4 |

### HerzelRegs

Данный модуль предназначен для приёма AXI-lite транзакций, их дешифрации, записи/чтения регистров и возвращения ответа.

Модуль основан на машине состояний:

* IDLE
* RADDR
* RDATA
* WADDR
* WDATA
* WRESP

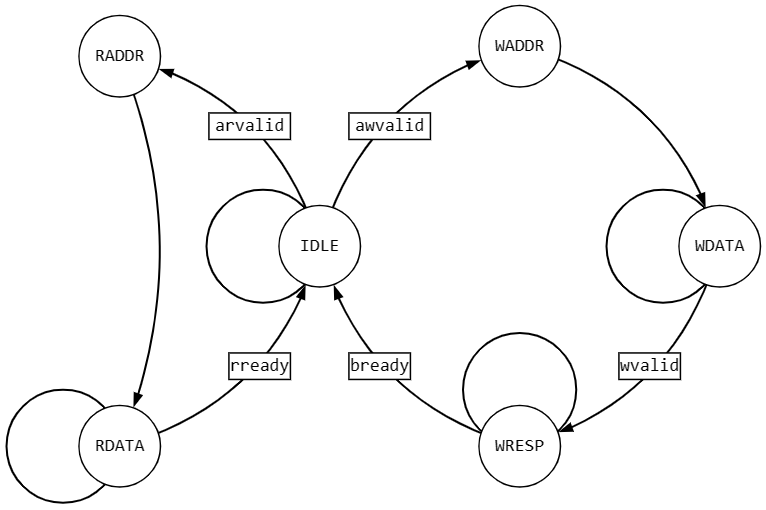


Рисунок Граф состояний модуля HerzelRegs

IDLE – ожидание появления транзакции чтения/записи (arvalid/awvalid).

RADDR – считывание адреса и отправление ответа (arready)

RDATA – отправка данных по принятому адресу с сигналом действительности данных (rvalid) и сигналом ответа (rresp). Ожидание ответа (rready). После получения ответа (rready) транзакция завершается и переходит в IDLE.

WADDR – считывание адреса и отправление ответа (awready)

WDATA – ожидание сигнала записи (wwalid). После запись данных по принятому адресу, отправление ответа (wready) и переход в следующее состояние.

WRESP – отправка статуса (bresp) с сигналом действительности данных (bvalid) и ожидание ответа (bready). После получения ответа (bready) транзакция завершается и переходит в IDLE.

Таблица Сигналы участвующие в обмене транзакций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Запись  Адрес | Запись  Данные | Запись  Отчёт | Чтение  Адрес | Чтение  Данные |
| AWVALID | WVALID | BVALID | ARVALID | RVALID |
| AWREADY | WREADY | BREADY | ARREADY | RREADY |
| AWADDR | WDATA | BRESP | ARADDR | RDATA |
|  |  |  |  | RRESP |

### Angel

Данный модуль является одним из этапов вычисления поворотных коэффициентов, а также специального коэффициента alpha. В частности данный модуль вычисляет угол, который будет передан модулю Cordic для вычисления углов.

В модуль заходит массив частот, записанных в регистры FREQ\_1-FREQ\_11. Далее для каждой частоты рассчитывается номерной коэффициент k по формуле:

– частота дискретизации (200 МГц).

– число выборок (100 т.шт).

Так как , то данная операция заменяется сдвигом вправо.

После коэффициент домножается на . Таким образом, получается угол для последующих расчётов.

### Cordic

Данный модуль производит поворотных коэффициентов, а также специального коэффициента alpha по углам передаваемых с модуля Angel. Расчёт производится 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.

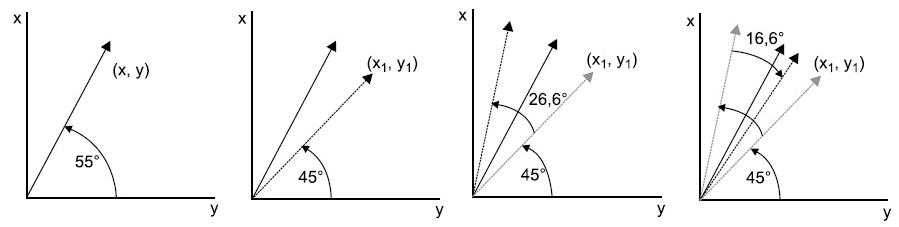


Рисунок CORDIC алгоритм

Координаты x и y вычисляются по формулам:

После преобразования:

Умножение на tan заменяется сдвигом, а cos заменяется коэффициентом масштабирования K, который рассчитывается заранее в зависимости от количества итераций. Умножение на K происходит в самом конце только один раз.

Итоговая формула:

При этом данный алгоритм верен для случая , в ином случае сперва надо определить квадрант, и с помощью вычитания перевести угол в первый квадрант, и после стандартного вычисления воспользоваться формулами приведения.

### DataScale

Данный модуль предназначен для масштабирования входных данных по следующей формуле:

13/256 – коэффициент, определяемый АЦП.

Также выходные данные доводятся до нужного формата и ширины.

### Herzel

Данный модуль по поступающим поворотным коэффициентам и коэффициенту alpha, а также данным вычисляет спектр.

Подробный вывод формул – [5]

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:

Где - промежуточные значения, которые рассчитываются итерационно:

W – поворотный коэффициент

Таким образом, для расчета потребуется N вещественных умножений, а не комплексных. Также требуется одно комплексное умножение на  на последней итерации.

Условие эффективности алгоритма Гёрцеля по сравнению БПФ-алгоритмом Кули:

# Моделирование FourierTransform

Работа модуля тестирования основана на сравнении результатов вычисления из MatchCad и результатов моделирования FourierTransform.

Для этого считываются данные из файлов sample.csv, freq.csv, data.csv.

sample.csv – содержит данные выборок, которые поступают на вход sample\_p.

freq.csv – содержит значения частот, которые будут вычисляться. Для этого они записываются в регистры FREQ1-FREQ11.

data.csv – рассчитанные данные из MatchCad.

После того как FourierTransform завершит расчёт, данные для каждой частоты из FourierTransform будут стравниваться с данными из MatchCad (data.csv). Если значения не будут совпадать с 10% погрешностью, то тест будет провален.

Примеры некоторых моделирований приведены в приложении 2.

# Приложение 1

## Модуль FourierTransform\_tb

`timescale 1ns/1ps

`include "./FourierTransformAM.sv"

module FourierTransform\_tb;

localparam CLK\_PER     = 5     ;

localparam SPI\_CLK\_PER = 1000  ;

localparam NF          = 11    ; // dont touch

localparam NS          = 100000; // dont touch

logic ok   ;

logic error;

integer fd\_r\_s;

integer fd\_r\_f;

integer fd\_r\_d;

spi\_if #(.SPI\_CLK\_PER(SPI\_CLK\_PER), .DISPLAY(1)) spi\_if();

logic [31:0] spi\_data;

logic [31:0] spi\_stat;

logic [NF-1:0][31:0] mcad\_freq;

logic [NF-1:0][31:0] mcad\_data;

logic [NF-1:0][31:0] vlog\_data;

logic        rstn    ;

logic        clk     ;

logic        spi\_sck ;

logic        spi\_ss\_n;

logic        spi\_mosi;

logic        spi\_miso;

logic        enable\_p;

logic        enable\_n;

logic [7 :0] sample\_p;

logic [7 :0] sample\_n;

assign spi\_sck         = spi\_if.mst.sck ;

assign spi\_ss\_n        = spi\_if.mst.ss\_n;

assign spi\_mosi        = spi\_if.mst.mosi;

assign spi\_if.mst.miso = spi\_miso       ;

assign enable\_n    = ~enable\_p   ;

assign sample\_n[0] = ~sample\_p[0];

assign sample\_n[1] = ~sample\_p[1];

assign sample\_n[2] = ~sample\_p[2];

assign sample\_n[3] = ~sample\_p[3];

assign sample\_n[4] = ~sample\_p[4];

assign sample\_n[5] = ~sample\_p[5];

assign sample\_n[6] = ~sample\_p[6];

assign sample\_n[7] = ~sample\_p[7];

FourierTransform #(

  .NF(NF),

  .NS(NS)

) DUT (

  .rstn    (rstn    ),

  .clk     (clk     ),

  .spi\_sck (spi\_sck ),

  .spi\_ss\_n(spi\_ss\_n),

  .spi\_mosi(spi\_mosi),

  .spi\_miso(spi\_miso),

  .enable\_p(enable\_p),

  .enable\_n(enable\_n),

  .sample\_p(sample\_p),

  .sample\_n(sample\_n)

);

initial forever #(CLK\_PER/2) clk=~clk;

initial begin

  fd\_r\_s = $fopen("D:/Desktop/Study\_now/SRW/GoertzelAlgorithm/src/sim/data/sample.csv", "r");

  if (fd\_r\_s == 0) $finish;

  fd\_r\_f = $fopen("D:/Desktop/Study\_now/SRW/GoertzelAlgorithm/src/sim/data/freq.csv", "r");

  if (fd\_r\_f == 0) $finish;

  fd\_r\_d = $fopen("D:/Desktop/Study\_now/SRW/GoertzelAlgorithm/src/sim/data/data.csv", "r");

  if (fd\_r\_d == 0) $finish;

end

final begin

  $fclose(fd\_r\_s);

  $fclose(fd\_r\_f);

  $fclose(fd\_r\_d);

end

initial begin

  for (int i = 0; i < NF; i = i + 1) begin

    $fscanf(fd\_r\_f, "%d\n", mcad\_freq[i]);

  end

  for (int i = 0; i < NF; i = i + 1) begin

    $fscanf(fd\_r\_d, "%d\n", mcad\_data[i]);

  end

end

task end\_of\_test();

  if (ok)

    $display("[%0t] TEST SUCCESS", $time);

  else

    $display("[%0t] TEST FAILED", $time);

    $display("[%0t] NUM OF ERROR: %0d", $time, error);

endtask

task herzel\_wait\_all\_valid();

  while (spi\_data&STATUS\_HERZEL\_ALL\_MSK != STATUS\_HERZEL\_ALL\_MSK) begin

    spi\_if.read\_data(STATUS, spi\_data, spi\_stat);

  end

endtask

task herzel();

  spi\_if.write\_data(EN\_CORDIC, 1       , spi\_stat);

  spi\_if.read\_data (STATUS   , spi\_data, spi\_stat);

  while (!(spi\_data&STATUS\_CORDIC\_MSK)) begin

    spi\_if.read\_data(STATUS, spi\_data, spi\_stat);

  end

  $fscanf(fd\_r\_s, "%d\n", sample\_p);

  @(posedge clk);

  enable\_p = 1;

  while (!$feof(fd\_r\_s)) begin

    @(posedge clk);

    $fscanf(fd\_r\_s, "%d\n", sample\_p);

  end

  herzel\_wait\_all\_valid();

endtask

initial begin

  ok       = 1;

  error    = 0;

  clk      = 0;

  rstn     = 0;

  enable\_p = 0;

  sample\_p = 0;

  spi\_if.init();

  repeat(5) @(posedge clk);

  rstn = 1;

  $display("[%0t] Write freq", $time);

  for (int i = 0; i < NF; i = i + 1) begin

    spi\_if.write\_data(FREQ\_1 + 32'h4\*i, mcad\_freq[i], spi\_stat);

  end

  $display("[%0t] Start Herzel", $time);

  herzel();

  $display("[%0t] Read data", $time);

  for (int i = 0; i < NF; i = i + 1) begin

    spi\_if.read\_data(DATA\_1 + 32'h4\*i, vlog\_data[i], spi\_stat);

  end

  $display("[%0t] Start check", $time);

  for (int i = 0; i < NF; i = i + 1) begin

    if ((vlog\_data[i] < mcad\_data[i]\*0.90) | (vlog\_data[i] > mcad\_data[i]\*1.10)) begin

      $display("[%0t] Error at freq %0d: %0d. MCAD - %0d, VLOG - %0d", $time, i, mcad\_freq[i], mcad\_data[i], vlog\_data[i]);

      error = error + 1;

      ok    = 0;

    end

  end

  # 5000;

  end\_of\_test();

  $stop;

end

endmodule

## Результаты моделирования

# Приложение 2

## АЦП

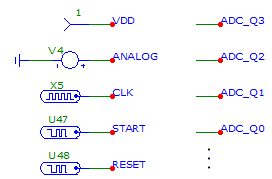


Рисунок Интерфейс АЦП модуля

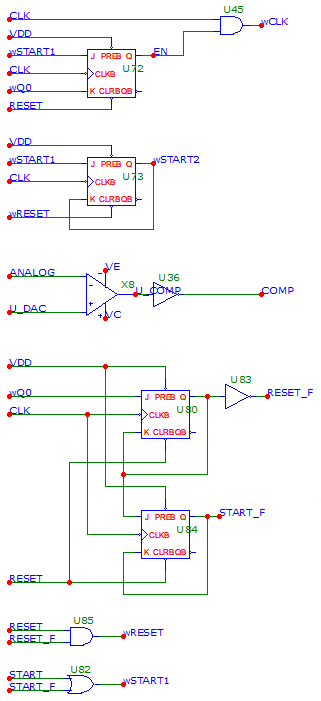


Рисунок Схема управления

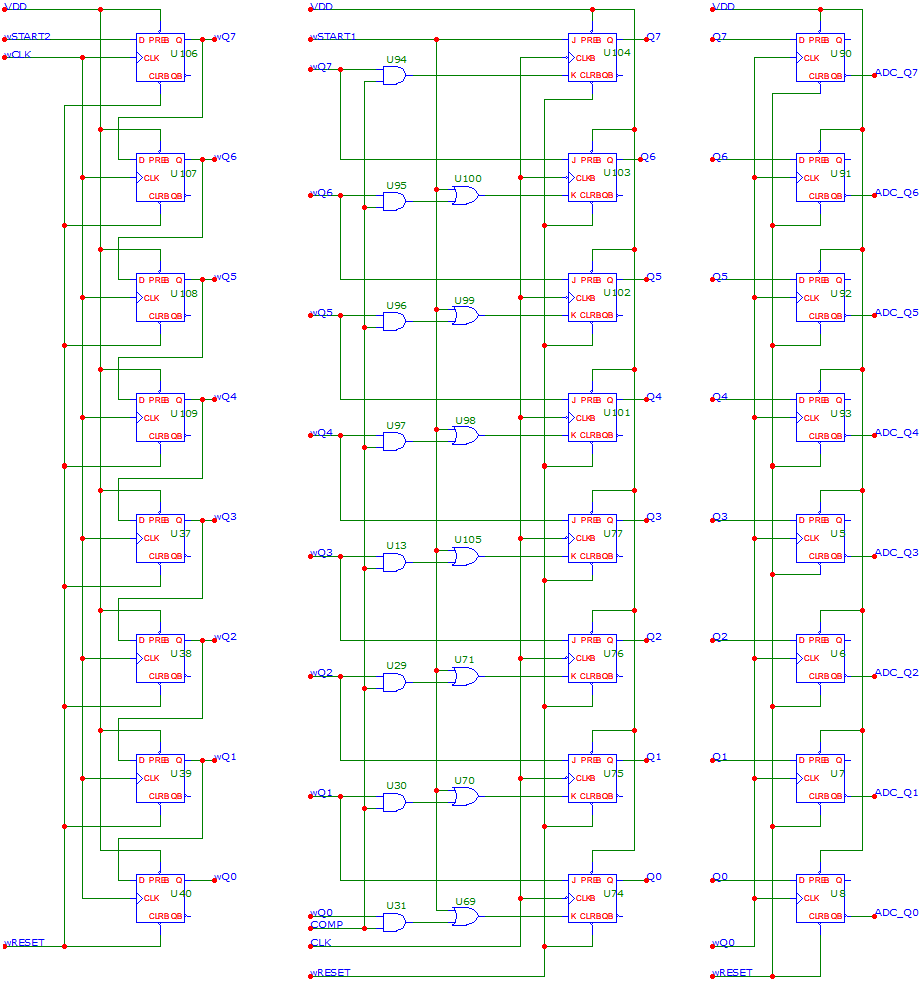


Рисунок Слева на право: сдвиговый регистр, схема изменения кода, выходной регистр

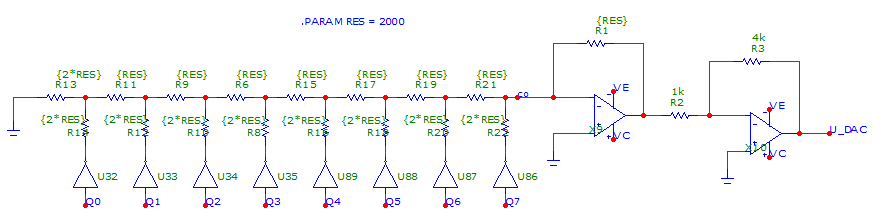
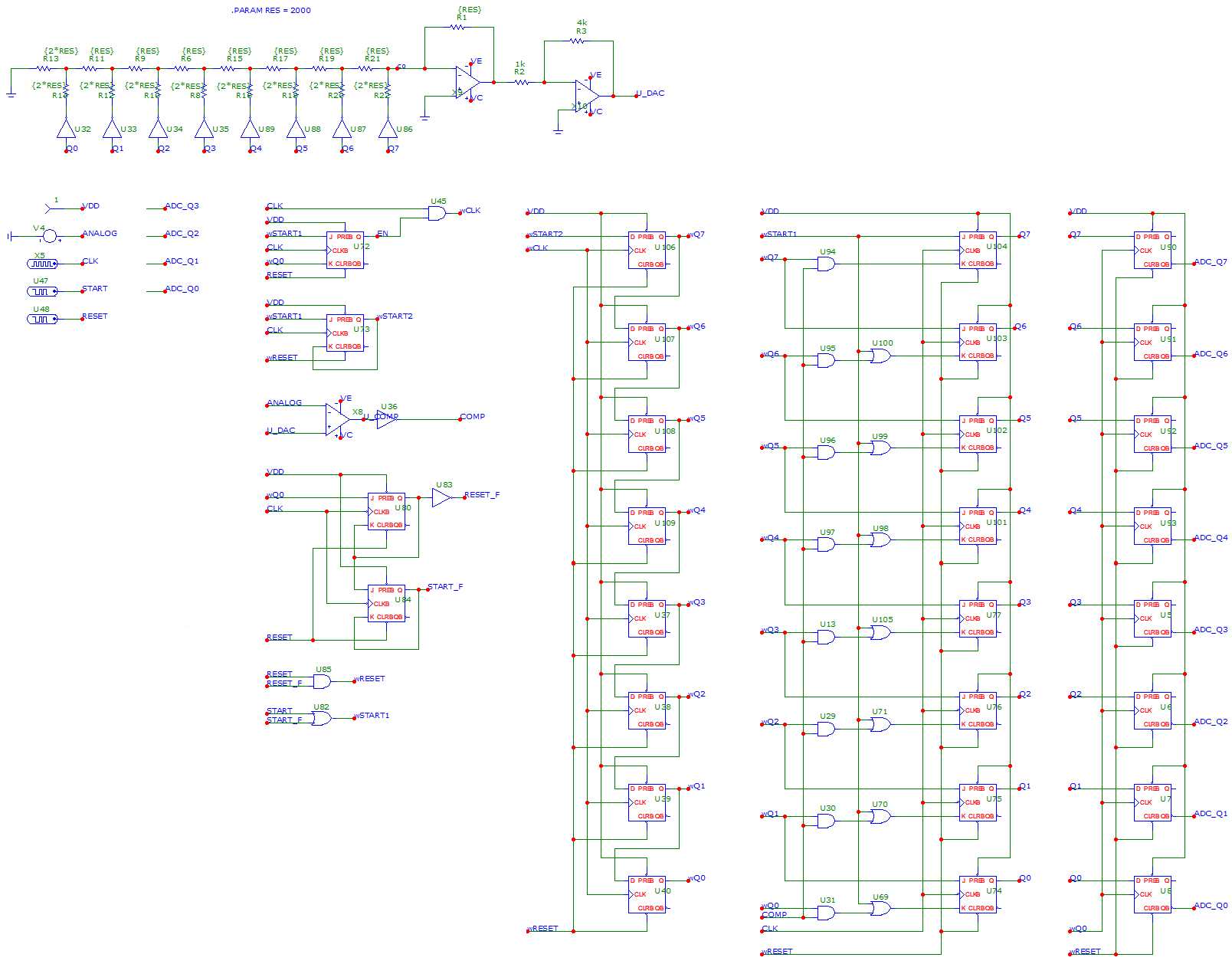


Рисунок Схема ЦАП



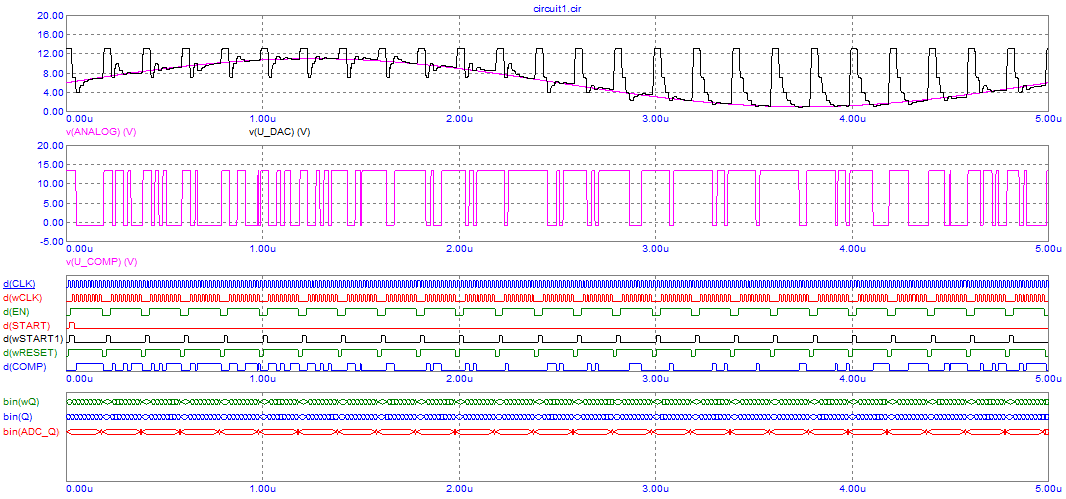


Рисунок Осцилограмма

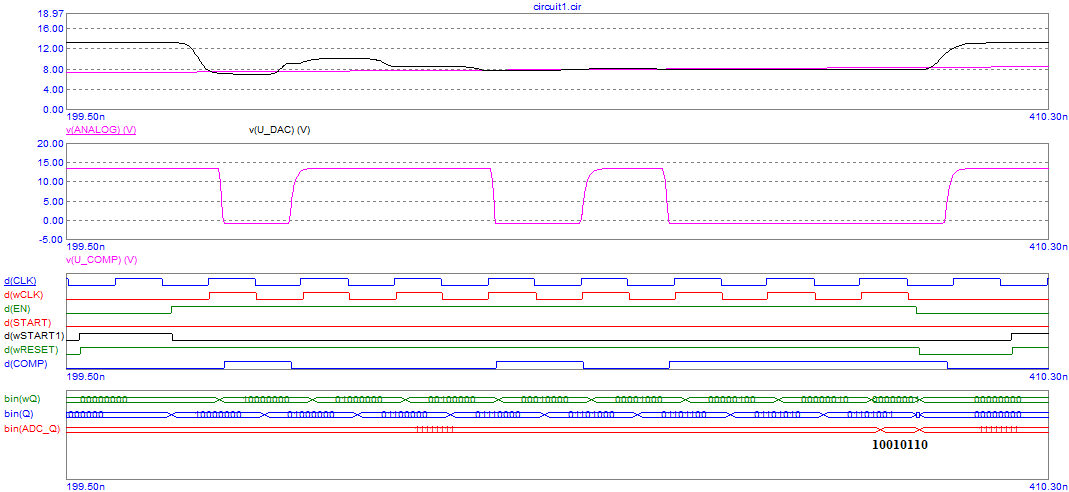


Рисунок Осцилограмма

## LVDS

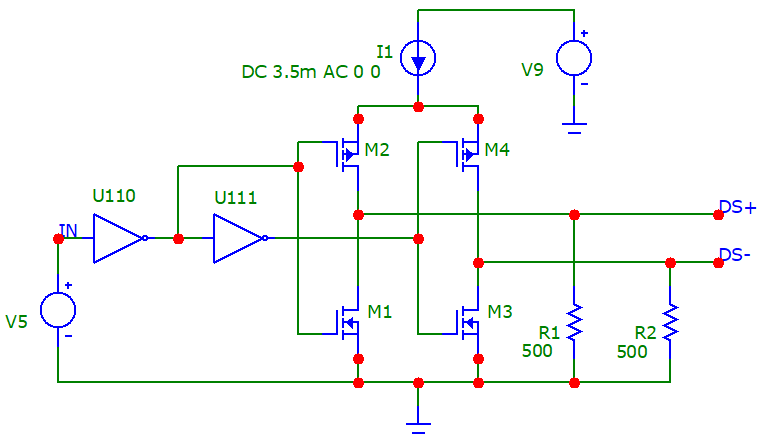


Рисунок Простейшая схема LVDS драйвера

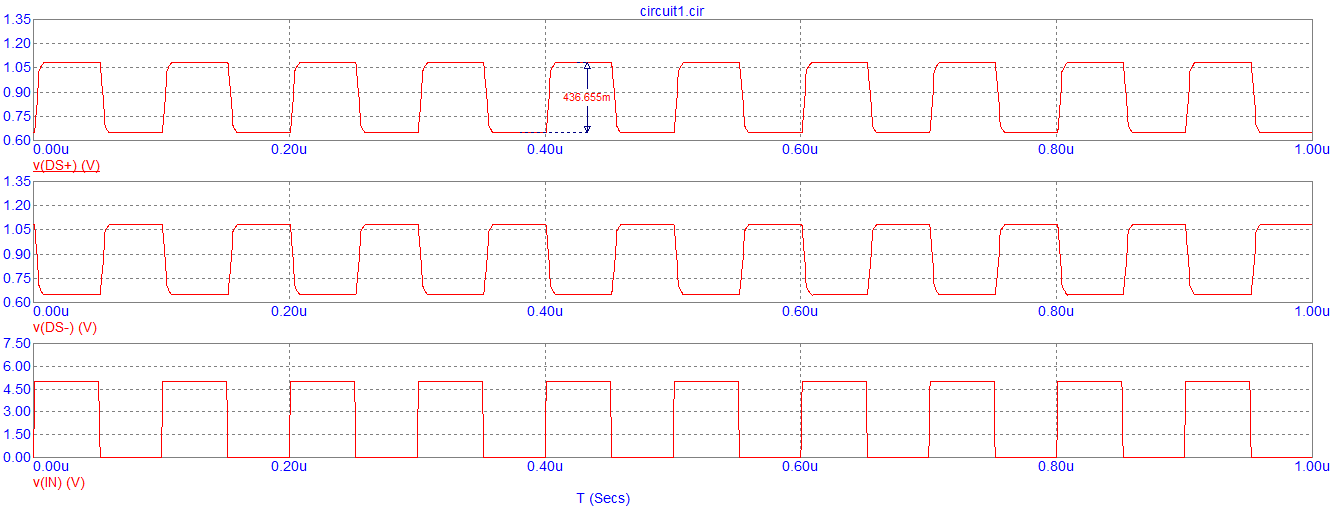


Рисунок Моделирование драйвера LVDS интерфейса

## Модуль преобразования Фурье

### FourierTransform

import axi\_pkg::\*;

`define TEST 1

module FourierTransform #(

  parameter NF = 11  , // NUM\_FREQ

  parameter NS = 1000  // NUM\_SAMPLE

) (

  input        rstn    ,

  input        clk     ,

  input        spi\_sck ,

  input        spi\_ss\_n,

  input        spi\_mosi,

  output       spi\_miso,

  input        enable\_p,

  input        enable\_n,

  input  [7:0] sample\_p,

  input  [7:0] sample\_n

);

logic       rstn\_syn;

logic       enable  ;

logic [7:0] sample  ;

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\_scl ;

logic signed [NF-1:0][31:0] data\_hrz ;

logic          en\_cordic   ;

logic          en\_scl      ;

logic          valid\_angel ;

logic          valid\_cordic;

logic          valid\_scl   ;

logic [NF-1:0] valid\_herzel;

axi\_lite\_mosi axio;

axi\_lite\_miso axii;

`ifndef TEST

  IBUFDS IBUFDS\_inst

  (

     .I (enable\_p),

     .IB(enable\_n),

     .O (enable  )

  );

  genvar gvar\_buf;

  generate

    for (gvar\_buf = 0; gvar\_buf < 8; gvar\_buf = gvar\_buf + 1) begin : buf

      IBUFDS IBUFDS\_inst

      (

        .I (sample\_p[gvar\_buf]),

        .IB(sample\_n[gvar\_buf]),

        .O (sample  [gvar\_buf])

      );

    end

  endgenerate

`else

  assign enable = enable\_p;

  genvar gvar\_buf;

  generate

    for (gvar\_buf = 0; gvar\_buf < 8; gvar\_buf = gvar\_buf + 1) begin

      assign sample[gvar\_buf] = sample\_p[gvar\_buf];

    end

  endgenerate

`endif

resync\_nrst u\_resync\_nrst(

  .clk   (clk     ),

  .nrst\_i(rstn    ),

  .nrst\_o(rstn\_syn)

);

spi2axi\_wrap #(

  .AXI\_ADDR\_WIDTH(32)

) u\_spi2axi\_wrap (

  .axi\_arstn\_i(rstn\_syn),

  .axi\_aclk\_i (clk     ),

  .spi\_sck\_i  (spi\_sck ),

  .spi\_ss\_n\_i (spi\_ss\_n),

  .spi\_mosi\_i (spi\_mosi),

  .spi\_miso\_o (spi\_miso),

  .axio\_o     (axio    ),

  .axii\_i     (axii    )

);

HerzelRegs u\_HerzelRegs (

  .rstn          (rstn\_syn    ),

  .clk           (clk         ),

  .freq\_arr\_o    (freq\_arr    ),

  .en\_cordic\_o   (en\_cordic   ),

  .valid\_angel\_i (valid\_angel ),

  .valid\_cordic\_i(valid\_cordic),

  .valid\_herzel\_i(valid\_herzel),

  .data\_arr\_i    (data\_hrz    ),

  .axio\_i        (axio        ),

  .axii\_o        (axii        )

);

Angel #(

  .NF(NF)

) u\_Angel (

  .rstn   (rstn\_syn   ),

  .clk    (clk        ),

  .en     (en\_cordic  ),

  .valid  (valid\_angel),

  .freq\_i (freq\_arr   ),

  .angel\_o(angel\_arr  )

);

Cordic #(

  .NF(NF)

) u\_Cordic (

  .rstn (rstn\_syn    ),

  .clk  (clk         ),

  .en   (valid\_angel ),

  .valid(valid\_cordic),

  .ang\_i(angel\_arr   ),

  .cos\_o(coefW\_re    ),

  .sin\_o(coefW\_im    ),

  .alpha(alpha       )

);

assign en\_scl = valid\_cordic && enable;

DataScale u\_DataScale (

  .rstn  (rstn\_syn ),

  .clk   (clk      ),

  .enable(en\_scl   ),

  .valid (valid\_scl),

  .data\_i(sample   ),

  .data\_o(data\_scl )

);

genvar gvar;

generate

  for (gvar = 0; gvar < NF; gvar = gvar + 1) begin : herzel

    Herzel #(

      .NF(NF),

      .NS(NS)

    ) u\_Herzel (

      .rstn   (rstn\_syn          ),

      .clk    (clk               ),

      .en     (valid\_scl         ),

      .valid  (valid\_herzel[gvar]),

      .alpha\_i(alpha[gvar]       ),

      .cW\_re\_i(coefW\_re[gvar]    ),

      .cW\_im\_i(coefW\_im[gvar]    ),

      .data\_i (data\_scl          ),

      .data\_o (data\_hrz[gvar]    )

    );

  end

endgenerate

endmodule

### IBUFDS

  IBUFDS IBUFDS\_inst

  (

     .I (enable\_p),

     .IB(enable\_n),

     .O (enable  )

  );

### resync\_nrst

module resync\_nrst #(

  parameter NUM\_STAGE = 3

) (

  input  clk   ,

  input  nrst\_i,

  output nrst\_o

);

logic [NUM\_STAGE-1:0] nrst\_stg;

always\_ff @(posedge clk, negedge nrst\_i) begin

  if (!nrst\_i) begin

    nrst\_stg <= 0;

  end

  else begin

    nrst\_stg <= {nrst\_stg[NUM\_STAGE-2:0], 1'b1};

  end

end

assign nrst\_o = nrst\_stg[NUM\_STAGE-1];

endmodule

### spi2axi интерфейс

`timescale 1ns/1ps

// !!! CPOL and CPHA - 0 and 0 ONLY !!!

interface spi\_if #(SPI\_CLK\_PER, DISPLAY = 0) ();

logic [31:0] write\_instr = 32'h0000\_0000;

logic [31:0] read\_instr  = 32'h0000\_0001;

logic [31:0] dummy\_word  = 32'h0000\_0001;

logic sck ;

logic ss\_n;

logic mosi;

logic miso;

modport mst (

  output sck ,

  output ss\_n,

  output mosi,

  input  miso

);

modport slv (

  input  sck ,

  input  ss\_n,

  input  mosi,

  output miso

);

task init();

  mst.sck  = 0;

  mst.ss\_n = 1;

  mst.mosi = 0;

endtask

task clk\_en(input en);

  if (en) begin

    forever #(SPI\_CLK\_PER/2) sck = !sck;

  end

  else begin

    forever #(SPI\_CLK\_PER/2) sck = 0;

  end

endtask

task write(input int num\_bit, input [31:0] data);

  for (int i=num\_bit-1; i>=0; i=i-1) begin

    mst.mosi = data[i];

    @(posedge sck);

  end

  mst.mosi = 0;

endtask

task read(input int num\_bit, output [31:0] data);

  data = 0;

  for (int i=num\_bit-1; i>=0; i=i-1) begin

    data[i] = mst.miso;

    @(posedge sck);

  end

endtask

task read\_data(input [31:0] addr, output [31:0] data, output [31:0] status);

  fork

    begin

      clk\_en(1);

    end

    begin

      // slave select

      mst.ss\_n = 0;

      // instruction byte

      write(8, read\_instr);

      // 4 address bytes

      @(posedge sck);

      write(32, addr);

      // dummy byte

      write(8, dummy\_word);

      // 4 data bytes

      read(32, data);

      // status byte

      read(8, status);

      // slave select

      mst.ss\_n = 1;

    end

  join\_any

  @(negedge sck);

  disable fork;

  if (DISPLAY) $display("[%0t] spi read: addr = 0x%0h, data = 0x%0h, status = 0x%0h", $time, addr, data, status);

endtask

task write\_data(input [31:0] addr, input [31:0] data, output [31:0] status);

  fork

    begin

      clk\_en(1);

    end

    begin

      // slave select

      mst.ss\_n = 0;

      // instruction byte

      write(8, write\_instr);

      // 4 address bytes

      @(posedge sck);

      write(32, addr);

      // 4 data bytes

      write(32, data);

      // dummy byte

      write(8, dummy\_word);

      // status byte

      read(8, status);

      // slave select

      mst.ss\_n = 1;

    end

  join\_any

  @(negedge sck);

  disable fork;

  if (DISPLAY) $display("[%0t] spi write: addr = 0x%0h, data = 0x%0h, status = 0x%0h", $time, addr, data, status);

endtask

endinterface

### HerzelRegs

import axi\_pkg::\*;

module HerzelRegs #(

  parameter NF = 11  ,

  parameter NS = 1000

) (

  // CLK&RST

  input                               rstn          ,

  input                               clk           ,

  // REGS

  output logic         [NF-1:0][31:0] freq\_arr\_o    ,

  output logic                        en\_cordic\_o   ,

  input                               valid\_angel\_i ,

  input                               valid\_cordic\_i,

  input                [NF-1:0]       valid\_herzel\_i,

  input                [NF-1:0][31:0] data\_arr\_i    ,

  // AXI

  input  axi\_lite\_mosi                axio\_i        ,

  output axi\_lite\_miso                axii\_o

);

// addres map of regs and default value

logic [31:0] version   = 32'h2904\_2023; // RW 0x00

logic [31:0] debug     = 32'hF0F0\_F0F0; // RW 0x04

logic [31:0] freq\_1    = 32'h0000\_0000; // RW 0x08

logic [31:0] freq\_2    = 32'h0000\_0000; // RW 0x0C

logic [31:0] freq\_3    = 32'h0000\_0000; // RW 0x10

logic [31:0] freq\_4    = 32'h0000\_0000; // RW 0x14

logic [31:0] freq\_5    = 32'h0000\_0000; // RW 0x18

logic [31:0] freq\_6    = 32'h0000\_0000; // RW 0x1C

logic [31:0] freq\_7    = 32'h0000\_0000; // RW 0x20

logic [31:0] freq\_8    = 32'h0000\_0000; // RW 0x24

logic [31:0] freq\_9    = 32'h0000\_0000; // RW 0x28

logic [31:0] freq\_10   = 32'h0000\_0000; // RW 0x2C

logic [31:0] freq\_11   = 32'h0000\_0000; // RW 0x30

logic [31:0] en\_cordic = 32'h0000\_0000; // RW 0x34

logic [31:0] status    = 32'h0000\_0000; // R  0x38

logic [31:0] data\_1    = 32'h0000\_0000; // R  0x3C

logic [31:0] data\_2    = 32'h0000\_0000; // R  0x40

logic [31:0] data\_3    = 32'h0000\_0000; // R  0x44

logic [31:0] data\_4    = 32'h0000\_0000; // R  0x48

logic [31:0] data\_5    = 32'h0000\_0000; // R  0x4C

logic [31:0] data\_6    = 32'h0000\_0000; // R  0x50

logic [31:0] data\_7    = 32'h0000\_0000; // R  0x54

logic [31:0] data\_8    = 32'h0000\_0000; // R  0x58

logic [31:0] data\_9    = 32'h0000\_0000; // R  0x5C

logic [31:0] data\_10   = 32'h0000\_0000; // R  0x60

logic [31:0] data\_11   = 32'h0000\_0000; // R  0x64

always\_comb begin

  freq\_arr\_o[0 ] = freq\_1            ;

  freq\_arr\_o[1 ] = freq\_2            ;

  freq\_arr\_o[2 ] = freq\_3            ;

  freq\_arr\_o[3 ] = freq\_4            ;

  freq\_arr\_o[4 ] = freq\_5            ;

  freq\_arr\_o[5 ] = freq\_6            ;

  freq\_arr\_o[6 ] = freq\_7            ;

  freq\_arr\_o[7 ] = freq\_8            ;

  freq\_arr\_o[8 ] = freq\_9            ;

  freq\_arr\_o[9 ] = freq\_10           ;

  freq\_arr\_o[10] = freq\_11           ;

  en\_cordic\_o    = en\_cordic[0]      ;

  status[0]      = valid\_angel\_i     ;

  status[1]      = valid\_cordic\_i    ;

  status[8 ]     = valid\_herzel\_i[0 ];

  status[9 ]     = valid\_herzel\_i[1 ];

  status[10]     = valid\_herzel\_i[2 ];

  status[11]     = valid\_herzel\_i[3 ];

  status[12]     = valid\_herzel\_i[4 ];

  status[13]     = valid\_herzel\_i[5 ];

  status[14]     = valid\_herzel\_i[6 ];

  status[15]     = valid\_herzel\_i[7 ];

  status[16]     = valid\_herzel\_i[8 ];

  status[17]     = valid\_herzel\_i[9 ];

  status[18]     = valid\_herzel\_i[10];

  data\_1         = data\_arr\_i[0 ]    ;

  data\_2         = data\_arr\_i[1 ]    ;

  data\_3         = data\_arr\_i[2 ]    ;

  data\_4         = data\_arr\_i[3 ]    ;

  data\_5         = data\_arr\_i[4 ]    ;

  data\_6         = data\_arr\_i[5 ]    ;

  data\_7         = data\_arr\_i[6 ]    ;

  data\_8         = data\_arr\_i[7 ]    ;

  data\_9         = data\_arr\_i[8 ]    ;

  data\_10        = data\_arr\_i[9 ]    ;

  data\_11        = data\_arr\_i[10]    ;

end

typedef enum {

  IDLE ,

  RADDR,

  RDATA,

  WADDR,

  WDATA,

  WRESP

} state;

state curr\_state;

state next\_state;

always\_ff @(posedge clk, negedge rstn) begin

  if (!rstn) begin

    curr\_state <= IDLE;

  end

  else begin

    curr\_state <= next\_state;

  end

end

always\_comb begin

  axii\_o.awready = 0;

  axii\_o.wready  = 0;

  axii\_o.bvalid  = 0;

  axii\_o.arready = 0;

  axii\_o.rdata   = 0;

  axii\_o.rresp   = 0;

  axii\_o.rvalid  = 0;

  case (curr\_state)

    IDLE : begin

      axii\_o.bresp   = 0;

      if (axio\_i.arvalid)

        next\_state = RADDR;

      else if (axio\_i.awvalid)

        next\_state = WADDR;

      else

        next\_state = IDLE ;

    end

    RADDR: begin

      axii\_o.arready = 1;

      next\_state = RDATA;

    end

    RDATA: begin

      axii\_o.rvalid = 1;

      case (axio\_i.araddr)

        32'h00: axii\_o.rdata  = version  ;

        32'h04: axii\_o.rdata  = debug    ;

        32'h08: axii\_o.rdata  = freq\_1   ;

        32'h0C: axii\_o.rdata  = freq\_2   ;

        32'h10: axii\_o.rdata  = freq\_3   ;

        32'h14: axii\_o.rdata  = freq\_4   ;

        32'h18: axii\_o.rdata  = freq\_5   ;

        32'h1C: axii\_o.rdata  = freq\_6   ;

        32'h20: axii\_o.rdata  = freq\_7   ;

        32'h24: axii\_o.rdata  = freq\_8   ;

        32'h28: axii\_o.rdata  = freq\_9   ;

        32'h2C: axii\_o.rdata  = freq\_10  ;

        32'h30: axii\_o.rdata  = freq\_11  ;

        32'h34: axii\_o.rdata  = en\_cordic;

        32'h38: axii\_o.rdata  = status   ;

        32'h3C: axii\_o.rdata  = data\_1   ;

        32'h40: axii\_o.rdata  = data\_2   ;

        32'h44: axii\_o.rdata  = data\_3   ;

        32'h48: axii\_o.rdata  = data\_4   ;

        32'h4C: axii\_o.rdata  = data\_5   ;

        32'h50: axii\_o.rdata  = data\_6   ;

        32'h54: axii\_o.rdata  = data\_7   ;

        32'h58: axii\_o.rdata  = data\_8   ;

        32'h5C: axii\_o.rdata  = data\_9   ;

        32'h60: axii\_o.rdata  = data\_10  ;

        32'h64: axii\_o.rdata  = data\_11  ;

        default: axii\_o.rresp = 2'h3;

      endcase

      if (axio\_i.rready)

        next\_state = IDLE;

      else

        next\_state = RDATA;

    end

    WADDR: begin

      axii\_o.awready = 1;

      next\_state = WDATA;

    end

    WDATA: begin

      if (axio\_i.wvalid) begin

        axii\_o.wready = 1;

        case (axio\_i.awaddr)

          32'h00: version   = axio\_i.wdata;

          32'h04: debug     = axio\_i.wdata;

          32'h08: freq\_1    = axio\_i.wdata;

          32'h0C: freq\_2    = axio\_i.wdata;

          32'h10: freq\_3    = axio\_i.wdata;

          32'h14: freq\_4    = axio\_i.wdata;

          32'h18: freq\_5    = axio\_i.wdata;

          32'h1C: freq\_6    = axio\_i.wdata;

          32'h20: freq\_7    = axio\_i.wdata;

          32'h24: freq\_8    = axio\_i.wdata;

          32'h28: freq\_9    = axio\_i.wdata;

          32'h2C: freq\_10   = axio\_i.wdata;

          32'h30: freq\_11   = axio\_i.wdata;

          32'h34: en\_cordic = axio\_i.wdata;

          default: axii\_o.bresp = 2'h3;

        endcase

      end

      if (axio\_i.wvalid)

        next\_state = WRESP;

      else

        next\_state = WDATA;

    end

    WRESP: begin

      axii\_o.bvalid = 1;

      if (axio\_i.bready)

        next\_state = IDLE;

      else

        next\_state = WRESP;

    end

    default: next\_state = curr\_state;

  endcase

end

endmodule

### Angel

module Angel #(

  parameter NF = 11

)(

  input                       rstn   ,

  input                       clk    ,

  input                       en     ,

  output logic                valid  ,

  input        [NF-1:0][31:0] freq\_i ,

  output logic [NF-1:0][31:0] angel\_o

);

// logic signed [31:0] ANGEL\_COEF = 32'h019B\_C65b; // 2\*pi/1000 (0.32)

logic signed [31:0] ANGEL\_COEF = 32'h0004\_1E24; // 2\*pi/100000 (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, negedge rstn) begin

    if (!rstn) begin

      valid <= 0;

      angel <= 0;

      indx  <= 0;

    end

    else if (en && !valid) begin

      if (indx < NF) begin

        angel[indx] <= k\_arr[indx] \* ANGEL\_COEF;

        indx        <= indx + 1;

      end

      else begin

        valid <= 1;

      end

    end

end

endmodule

### Cordic

module Cordic #(

  parameter NF = 11

)(

  // CLK&RST

  input                              rstn ,

  input                              clk  ,

  // CTRL

  input                              en   ,

  output logic                       valid,

  // 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, negedge rstn) begin

  if (!rstn) begin

    valid <= 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 && !valid) 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

      valid <= 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

module DataScale (

  // CLK&RST

  input                      rstn  ,

  input                      clk   ,

  // CTRL

  input                      enable,

  output logic               valid ,

  // 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, negedge rstn) begin

  if (!rstn) begin

    valid  <= 0;

    data\_m <= 0;

  end

  else begin

    valid  <= enable           ;

    data\_m <= SCALE\_COEF \* data;

  end

end

endmodule

### Herzel

module Herzel #(

  parameter NF = 11  ,

  parameter NS = 1000

)(

  // CLK&RST

  input                      rstn        ,

  input                      clk         ,

  // CTRL

  input                      en          ,

  output logic               valid       ,

  // 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, negedge rstn) begin

  if (!rstn) begin

    valid     <= 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 && !valid) 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;

      valid   <= 1;

    end

  end

end

endmodule

# Источники

[1] <https://www.compel.ru/lib/54830>

[2] <https://www.compel.ru/lib/87061?ysclid=laf3i4m5ow148073174>

[3] <https://github.com/airhdl/spi-to-axi-bridge/tree/main>

[4] <https://kit-e.ru/fpga/cordic/>

[5] <https://ru.dsplib.org/content/goertzel/goertzel.html>

Ссылка на репозиторий для моделирования:

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