



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт	ИРЭ
Кафедра	ЭиН

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)**

Направление	11.04.04 Электроника и нанoeлектроника
	(код и наименование)

Образовательная программа	Твердотельная микро- и нанoeлектроника
---------------------------	--

Форма обучения	очная
	(очная/очно-заочная/заочная)

Тема:	Проектирование и моделирование малошумящего устройства для АЦП и Фурье преобразования
-------	---

Студент	ЭР-05м-21	Маринин Н.С.
	группа	подпись фамилия и инициалы

Руководитель ВКР	К.Т.Н.	доцент	Мирошников Б.Н.
	уч. степень	должность	подпись фамилия и инициалы

Консультант				
	уч. степень	должность	подпись	фамилия и инициалы

Внешний консультант				
	уч. степень	должность	подпись	фамилия и инициалы

организация

«Работа допущена к защите»

Заведующий кафедрой	Д.Т.Н.	профессор	Мирошникова И.Н.
	уч. степень	звание	подпись фамилия и инициалы

Дата

Москва, 2023



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт	ИРЭ
Кафедра	ЭиН

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
(МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ)

Направление	11.04.04 Электроника и наноэлектроника
	(код и наименование)

Образовательная программа	Твердотельная микро- и наноэлектроника
---------------------------	--

Форма обучения	очная
	(очная/очно-заочная/заочная)

Тема:	Проектирование и моделирование малошумящего устройства для АЦП и Фурье преобразования
-------	---

Студент	ЭР-05М-21	Маринин Н.С.
	группа	подпись фамилия и инициалы

Руководитель ВКР	К.Т.Н.	доцент	Мирошников Б.Н.
	уч. степень	должность	подпись фамилия и инициалы

Консультант				
	уч. степень	должность	подпись	фамилия и инициалы

Внешний консультант				
	уч. степень	должность	подпись	фамилия и инициалы

		организация		
Заведующий кафедрой	Д.Т.Н.	профессор	Мирошникова И.Н.	
	уч. степень	звание	подпись фамилия и инициалы	

Место выполнения работы	ФГБОУ ВО «НИУ «МЭИ»
-------------------------	---------------------

1.Обоснование выбора темы выпускной квалификационной работы

Тема была выбрана исходя из личного интереса студента в изучении алгоритмов цифровой обработки сигналов, в данном случае – алгоритм Фурье преобразования. А также в целях общего развития навыков описания цифровой аппаратуры и верификации цифровых схем.

Научный руководитель _____ дата _____

Студент _____ дата _____

2.Консультации по разделу

Подпись консультанта _____ дата _____

3.Консультации по разделу

Подпись консультанта _____ дата _____

4. План выполнения выпускной квалификационной работы

№ п\п	Содержание разделов	Срок выпол- нения	Трудоём- кость, %
I.	Теоретическая часть	с 16.02 по 16.03	20
II.	Экспериментальная часть	с 16.03 по 16.04	70
III.	Публикации		
IV.	Оформление диссертации	с 16.05 по 16.06	10

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

1. Задание брошюруется вместе с выпускной работой после титульного листа (страницы задания имеют номера 2, 3, 4, 5).
2. Отзыв руководителя, рецензия(и), отчет о проверке на объем заимствований и согласие студента на размещение работы в открытом доступе вкладываются в конверт (файловую папку) под обложкой работы.

1. Аннотация

Для решения задачи получения спектральной плотности мощности шума фоторезистора была изучена методика измерения шума, спроектирован АЦП последовательного приближения и спроектирована цифровая схема Фурье преобразования на основе метода Гёрцеля.

Также цифровая схема Фурье преобразования имеет регистровую модель для более широкого спектра измерений.

Также для связи с внешним устройством управления, в цифровой схеме встроен SPI интерфейс.

2. Сокращения

СПМШ	–	спектральная плотность мощности шума
ФР	–	фоторезист
ПЛИС	–	программируема логическая интегральная схема
АЦП	–	аналого-цифровой преобразователь
SPI	–	Serial Peripheral Interface, последовательный периферийный интерфейс
LVDS	–	low-voltage differential signaling, низковольтная дифференциальная передача сигналов
AXI	–	Advanced eXtensible Interfac, расширенный расширяемый интерфейс

3. Оглавление

1. Аннотация	6
2. Сокращения.....	7
3. Оглавление	8
4. Введение.....	9
5. Литературный обзор	10
5.1. Типы шумов и их природа	10
5.2. Метод измерения шума	16
5.3. Установка измерения шума.....	20
5.4. АЦП	22
5.5. Преобразование Фурье	24
6. Разработка схемы	27
6.1. АЦП	28
6.2. LVDS	30
6.3. Модуль Фурье преобразования	32
6.4. Тестирование модуля Фурье преобразования.....	58
7. Реализация на ПЛИС	59
8. Заключение	60
9. Приложение 1	61
9.1. Модель АЦП.....	63
9.2. Тест ПЛИС	65
10. Приложение 2	67
10.1. Тест модуля Фурье преобразования.....	67
10.2. Описание SPI интерфейса для тестирования.....	73
10.3. Результаты моделирования	76
11. Приложение 3	83
11.1. АЦП.....	83
11.2. LVDS.....	89
11.3. Модуль преобразования Фурье.....	90
12. Список литературы	127

4. Введение

Данная работа относится к области цифровой схемотехники, цифровой обработке сигналов. А именно рассматривается метод цифрового Фурье преобразования.

В качестве основной задачи необходимо перевести данные полученные при измерении сигнала шума фоторезистора из временной области в частотную. Для этого необходимо провести измерение шума, оцифровать полученные данные, преобразовать одним из методов Фурье преобразования и передать полученные данные на персональный компьютер для последующего анализа и использования.

Для решения этой задачи была изучена методика измерения шума, спроектирован АЦП последовательного приближения и спроектирована цифровая схема Фурье преобразования на основе метода Гёрцеля.

Главной особенностью метода Гёрцеля является то, что вычисления могут производиться по мере поступления входных данных, что является довольно привлекательным в связке с АЦП, с которого поступают данные.

Для связи с внешним устройством управления, в нашем случае это персональный компьютер, в цифровой схеме встроен SPI интерфейс.

5. Литературный обзор

5.1. Типы шумов и их природа

Шум напряжения и тока - это важная характеристика фотоприемника, которая определяет пороговую чувствительность, то есть минимальный полезный сигнал который можно различить. Шум проявляется в виде случайных флуктуаций напряжения или тока на клеммах прибора.

Выделяют несколько групп шума. Шум бывает радиационным (внешний) - флуктуации потока излучения, падающего на фотоприемник и приводящие к флуктуациям напряжения. Шум этого вида не может быть полностью исключен, но можно использовать оптимальную конструкцию оптической системы и фотоприемника для минимизации его влияния. Внешний шум в идеальных условиях, то есть при отсутствии внутреннего шума является единственным пределом для фотоприёмника. Но реальный шум в большинстве случаев значительно превосходит внешний шум. [1]

Спектральная плотность внешнего шума равна :

$$P(f) = 2J \cdot \frac{\exp(h\nu/kT)}{\exp(h\nu/kT)-1}$$

Внутренний шум – это шум возникающий в фотоприёмнике в результате случайных, флуктационных физических процессах. Для фоторезисторных фотоприемников выделяют следующие виды шумов:

- шум типа $1/f^\alpha$
- тепловой шум
- генерационно-рекомбинационный

Также существует дробовой (барьерный) шум, но он в основном проявляется в фотодиодах. В фоторезистах он может проявиться только в случае плохих контактов.

Для характеристики шума используют среднеквадратичное значение, то есть его мощность. Под спектральной плотностью мощности шума (СПМШ) понимают следующую величину:

$$P_U(f) = \frac{\overline{U^2(f)}}{\Delta f}, \Delta f \rightarrow 0$$

$\overline{U^2(f)}$ - Фурье-образ временных флуктуаций напряжения:

$$\overline{U^2(f)} = F(\omega; t) \overline{U^2(t)}$$

Различные виды шумов имеют свою различную природу происхождения, что отражается на СПМШ. Таким образом, исследуя СПМШ в различных условиях (температура, освещенность и так далее), можно делать выводы о природе шума и методах его уменьшения [1].

Общий спектр плотности мощности шума следующий:

$$P_U(f) = \frac{\overline{U_{ш}^2}}{\Delta f} = \frac{A \cdot U_{ФР}^2}{f^\alpha \cdot V} + \frac{4U_{ФР}^2 \overline{\Delta p^2}}{p^2 \cdot V} \cdot \frac{\tau}{1 + (2\pi \cdot f)^2 \tau^2} + 4kT_d R$$

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

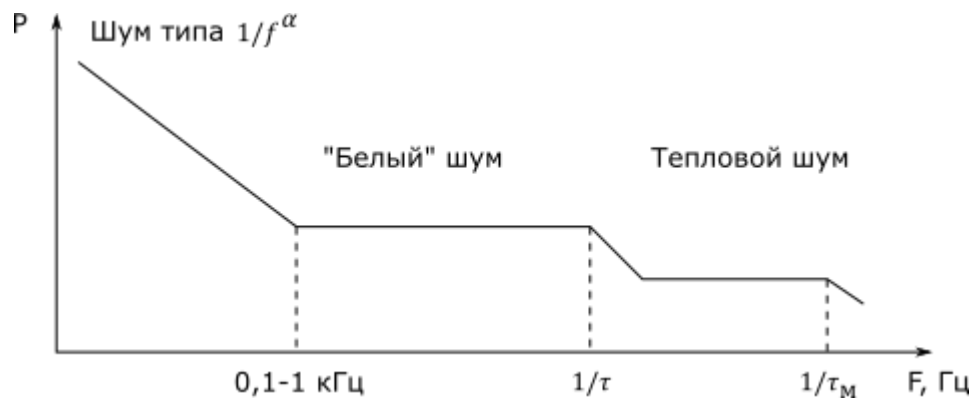


Рисунок 1 Структура спектра плотности мощности шума фотоприемника

Получив СПМШ определённого фотоприемника, можно выбрать оптимальную частоту модуляции сигнала и полосу пропускания системы, чтобы минимизировать влияние шума на чувствительность прибора.

5.1.1. Тепловой шум

Тепловой шум или шум Джонсона-Найквиста является результатом флуктуации проводимости в материалах, вызванных случайными изменениями скорости свободных носителей заряда. Это приводит к появлению случайных микроскопических диффузионных токов, которые вызывают флуктуацию тока в системе и флуктуацию напряжения на электрических контактах [1].

Спектр плотности мощности этого шума может быть определен с помощью формулы Найквиста:

$$P_u(f) = \frac{\overline{U_{ш}^2}}{\Delta f} = 4kT_d R$$

$U_{ш}$ – среднеквадратическое значение напряжения шума

Δf – полоса пропускания измерительного тракта

T_d – рабочая температура ФР

R – электрическое сопротивление ФР

5.1.2. Генерационно-рекомбинационный шум

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

Генерационно-рекомбинационный возникает из-за случайных флуктуаций скоростей генерации и рекомбинации носителей заряда, вызванных различными причинами, такими как неоднородности в структуре материала. Что приводит к флуктуации плотности заряда и появлению шума.

Этот тип шума может проявляться как при генерации через запрещенную зону материала, так и через разрешенные уровни в запрещенной зоне, обусловленные дефектами структуры, то есть через ловушки. Генерационно-рекомбинационный шум на ловушечных центрах является наиболее распространенным типом шума в полупроводниках так как требует меньше энергии для генерационно-рекомбинационных процессов. Шум же при генерации через запрещённую зону существенен в основном для материалов с узкой запрещенной зоной и низкой концентрацией дефектов, образующих разрешенные уровни в запрещенной зоне [1].

Он может проявляться по-разному, в зависимости от внешних условий, таких как температура, внешнее электрическое поле, концентрация носителей заряда и других параметрах влияющих на процессы генерации. Данный шум не наблюдается при малом времени жизни носителей заряда.

$$P_u(f) = \frac{4U_{\Phi p}^2}{V} \cdot \frac{(b+1)^2}{(bn+p)} \cdot \frac{np}{n+p} \cdot \frac{\tau}{1+\omega^2\tau^2}$$

$V = A_{\text{эфф}} \cdot d$ – объём полупроводникового слоя

n, p – концентрация электронов и дырок

5.1.3. Фликкер-шум

Фликкер-шум проявляется при низких частотах и имеет обратную зависимость от частоты:

$$P_u(f) = \frac{\overline{U_{ш}^2}}{\Delta f} = \frac{AU_{ФР}^2}{f^\alpha}$$

A и α – коэффициенты, зависящие от многих факторов параметра A , показателя степени α и частоты перехода шума $\frac{1}{f^\alpha}$ в «белый» шум. Параметр A варьируется от $2 \cdot 10^{-6}$ до $5 \cdot 10^{-2}$; значение α – от 0,5 до 3 [1].

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.

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 $1/f^\alpha$ флуктуациями числа носителей заряда в результате захвата части носителей «глубоко лежащими ловушками», которые могут находиться в слое над поверхностью полупроводника.

Захват носителей ловушками имеет два следствия. Захват вызывает непосредственное изменение числа свободных носителей основного типа, имеющих 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:

$$P_i(f) = \frac{\alpha_H I_H^2}{N \cdot f^\alpha}$$

$N = V \cdot n$ – число носителей заряда

V – объем исследуемого образца

n – концентрация носителей

α_H – постоянная Хоухе.

5.2. Метод измерения шума

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

Точные значения условий измерения, а также схемы измерений были взяты из ГОСТа [5].

Измерение следует проводить на установке, структурная схема которой приведена на рисунке 2.

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

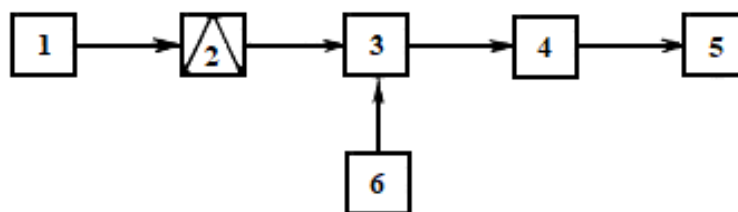


Рисунок 2 1 - источник излучения; 2 - модулятор; 3 - испытуемый фотоприёмник; 4 - усилительное устройство; 5 - регистрирующий прибор; 6 - источник питания фотоприёмника

При выборе источника излучения рекомендуется применять лампу накаливания типа при цветовой температуре 2856 ± 100 К

В качестве источников излучения рекомендуется использовать полупроводниковые лазерные диоды или светодиоды, нестабильность выходной мощности которых не должна выходить за пределы интервала $\pm 5\%$ 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

$$\varphi = \frac{\int_0^{\infty} S_{\text{отн}}(\lambda) r_{\lambda} d\lambda}{\int_0^{\infty} r_{\lambda} d\lambda}$$

где $S_{\text{отн}}(\lambda)$ - относительная спектральная характеристика чувствительности фотоприёмника.

r_{λ} - спектральная плотность потока излучения.

Максимальное значение потока излучения, падающего на фотоприёмник, должно выбираться из условия работы фотоприёмника 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 $\pm 1,5\%$.

Тип усилительного устройства должен выбираться в зависимости от требований к частоте и форме модуляции потока излучения, уровня регистрируемого сигнала, вида измеряемого параметра. 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 $\pm 3\%$. Полоса пропускания таких приборов при измерении напряжения (тока) шума должна не менее чем в десять раз превышать эквивалентную шумовую полосу измерительной цепи.

Источник питания фотоприёмника должен обеспечивать установление напряжения питания фотоприёмника с погрешностью, которая не должна выходить за пределы интервала $\pm 3\%$.

Коэффициент пульсации должен находиться в пределах $\pm 10\%$ и не оказывать влияние на результат измерения параметров фотоприёмника.

Климатические условия проведения измерений Климатические условия окружающей среды, в которых проводят измерение, должны соответствовать следующим требованиям, если иные не оговорены в технических условиях на фотоприёмник конкретного типа:

температура, °C	20 ± 5
относительная влажность, %	65 ± 15
атмосферное давление, кПа	100 ± 4

К испытываемому фотоприёмнику подключают сопротивление нагрузки. Устанавливают режим питания на фотоприёмник и регистрируют значение напряжения шума по показаниям регистрирующего измерительного прибора. Продолжительность измерения и значения напряжения шума следует регистрировать по максимальным повторяющимся показаниям прибора за время не менее 10 с.

Если напряжение шума близко к напряжению шума измерительной установки, то сначала необходимо зарегистрировать напряжение шума без подачи напряжения на фотоприёмник, затем суммарное напряжение шума при подаче на него напряжения.

Напряжение шума в вольтах следует вычислять по формуле:

$$U_{ш} = \sqrt{U_{ш2}^2 - U_{ш1}^2}$$

где $U_{ш2}$ - суммарное напряжение шума при подаче напряжения, питания фотоприёмника.

$U_{ш1}$ - напряжение шума без подачи напряжения питания фотоприёмника.

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

На испытуемый фотоприёмник подают модулированный поток излучения и регистрируют напряжение фотосигнала по показаниям регистрирующего прибора.

5.3. Установка измерения шума

Основными измерительными стендами для измерения фотоэлектрических параметров и оценки качества промышленных фоторезисторов 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 3.

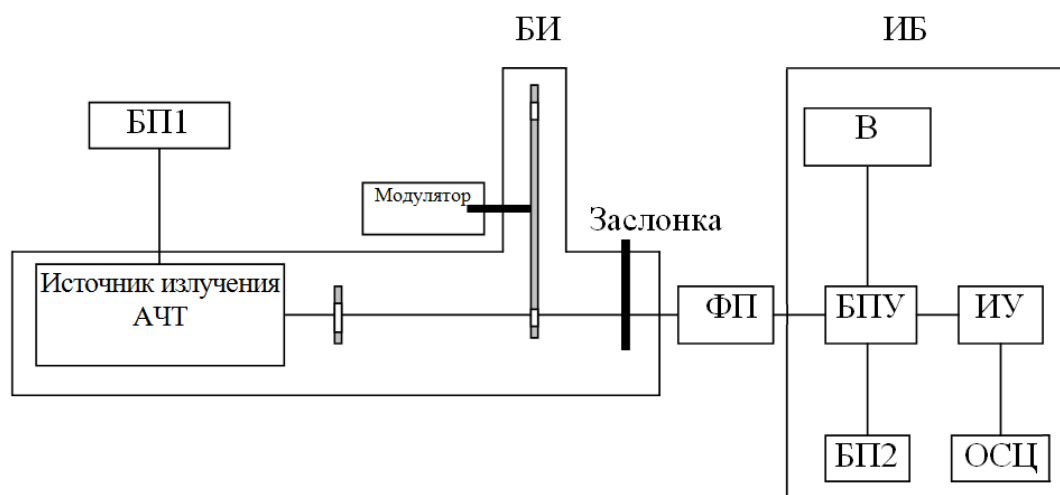


Рисунок 3 Структурная схема установки для измерения параметров ФЧЭ

В состав блока излучателя (БИ) входят: источник излучения – абсолютно чёрное тело (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 фоточувствительного элемента 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, 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 шума, усиленный и отфильтрованный сигнал поступает на 8-ти разрядный АЦП последовательного типа.

Процесс измерения заключается в последовательной регистрации и запоминании шумовых сигналов через интервал времени. Далее спектр плотности мощности шума вычисляется по алгоритму преобразования Фурье.

5.4. АЦП

Существуют АЦП встроенные в микроконтроллеры, ПЛИСы, микропроцессоры, системы-на-кристалле, АЦП последовательного приближения (SAR) другие. 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. На рисунке 4 приведено примерное сравнение трёх самых распространенных типов АЦП.

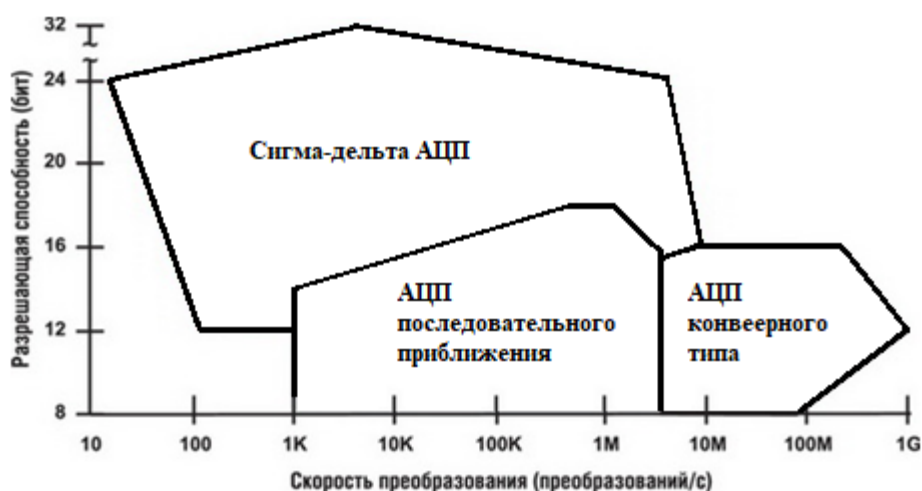


Рисунок 4 Соответствие скорости преобразования и разрешающей способности для трех основных типов интегральных аналого-цифровых преобразователей

АЦП последовательного приближения имеет разрядность данных, как правило, в пределах 6...8 до 20 бит, скорость же – от нескольких Квыб/с до 10 Мвыб/с. SAR АЦП подходит для приложений со средним диапазоном скоростей.

АЦП последовательного приближения показан рисунке 5 и работает по принципу весов. В роли неизвестного веса выступает входной сигнал $U_{вх}$, из которого происходит выборка и хранение значений напряжения. Для этого это напряжение сравнивается эталонным напряжением полученного с ЦАП из цифрового начального кода с помощью компаратора. В случае несовпадения напряжений, эталонное напряжение с помощью регистра последовательного приближения изменяется на половину своего предыдущего значения, за счёт изменения цифрового кода с которого оно

получено. С каждой итерацией точность будет увеличиваться пока цифровой код не исчерпает свою размерность и не будет получена выборка. Итоговый код запишется в выходной регистр и будет передан в данном проекте на LVDS драйвер для передачи в цифровую схему на ПЛИС. Весь необходимый контроль производится с помощью схемы управления.

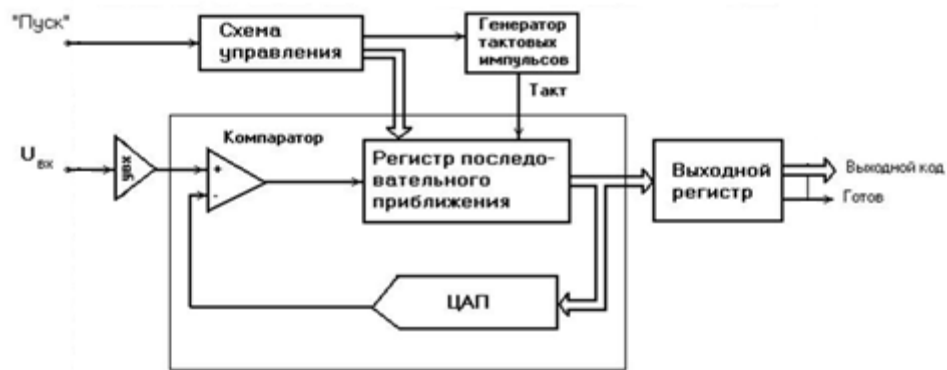


Рисунок 5 АЦП последовательного приближения

Реализация данной схемы представлена в приложении 3 на рисунках 31-35. Работа схемы представлена на рисунках 36-37.

5.5. 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.

$$f(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ix\omega} dx$$

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 рисунке 6 примерно представлено примерный принцип преобразования временного сигнала в частотный.

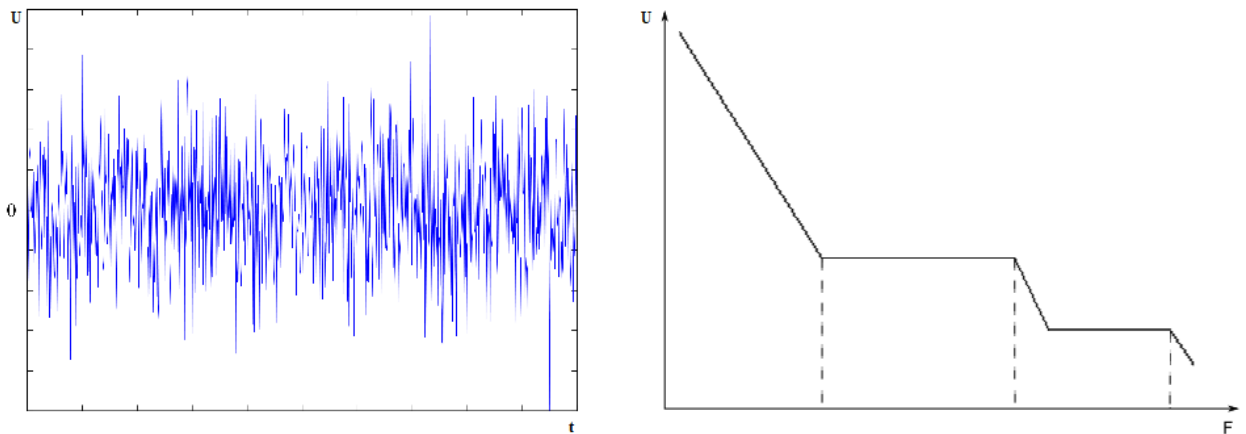


Рисунок 6 Входной сигнал слева, преобразованный через Фурье справа

Для цифровой схемотехники применяют дискретное преобразование Фурье:

$$M(f) = \frac{1}{N} \sum_{k=0}^{N-1} g(t_k) \cdot [\cos(2\pi f t_k) + i \cdot \sin(2\pi f t_k)],$$

Где $g(t_k)$ – входной дискретный сигнал (шум фоторезиста)

N – количество значений входного дискретного сигнала

t_k - k -ое время дискретного сигнала

$f = 0, 1, \dots, N$ - частота

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

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

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

В самом простом случае поступивший код $g(t_k)$ должен будет затем умножен на поворотные коэффициент:

$$[\cos(2\pi f t_k) + i \cdot \sin(2\pi f t_k)]$$

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 N для получения среднего значения.

Таким образом, простейший алгоритм следующий:

- Приём оцифрованного кода с АЦП
- Вычисление $\cos(2\pi f t_k)$ и $\sin(2\pi f t_k)$
- Умножение кода на комплексные коэффициенты
- Сложение с предыдущим значением
- Вычисление модуля
- Деление для получения среднего

Для всего этого потребуется N операций вычисления \cos , \sin , умножения для вычисления одной частоты. Что довольно затратно. Поэтому для

вычислений спектра будет использоваться алгоритм Герцеля. Данный алгоритм 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 - $\Delta t = 0.5$ мкс.

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

6. Разработка схемы

Для решения поставленной задачи был разработан проект, структура которого выглядит следующим образом:

- АЦП
 - АЦП
 - LVDS
- FourierTransform
 - resync_nrst
 - resync_data
 - spi2axi_wrap
 - HerzelRegs
 - div_all
 - Angel
 - Cordic
 - DataScale
 - Herzel

6.1. 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 моделировалась с учётом задержек логических элементов. А также с применением библиотечных реально существующих моделей операционного усилителя (AD8045) и компаратора (MAX9010).

Моделирование проводилось в MicroCap.

Интерфейс модуля представлен в таблице 1.

Таблица 1 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.

Сигнал ANALOG сравнивается с начальным значением напряжения с ЦАП на компараторе как показано на рисунке 5. Начальное значение

напряжения 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 последовательного приближения.

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 и поступит на выходы ADC_Q.

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

Далее начнётся оцифровка нового значения.

6.2. LVDS

Интерфейс LVDS широко применяется для высокоскоростной передачи данных и распределения тактовых сигналов по соединительным линиям, кабелям и межплатным соединениям и других соединениях.

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 — V_{OD} . Сигнал в каждой из двух сигнальных линий имеет максимальный размах $|V_{OD}|$ и центрирован относительно синфазного напряжения V_{OC} . Стандартные значения этих напряжений показаны на рисунке 7.

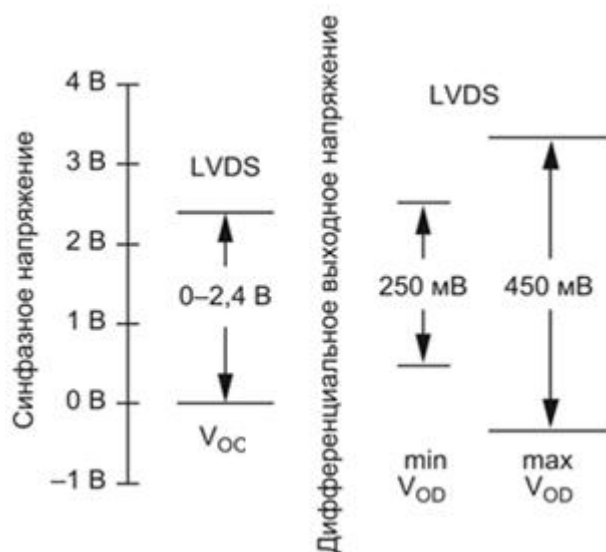


Рисунок 7 Уровни сигналов в LVDS и M-LVDS

Выходные данные с АЦП поступают на вход LVDS драйвера и далее по дифференциальной линии на вход ПЛИС. 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 ПЛИС. Общая схема данного соединения показана на рисунке 8. Терминирующий резистор R_T имеет номинал в 100 Ом и необходим для согласования.



Рисунок 8 Канал связи между приёмником и передатчиком

Схема драйвера LVDS и его моделирования показанна на рисунках 38-39.

6.3. Модуль Фурье преобразования

Данный модуль назван FourierTransform и является главным уровнем всего RTL дизайна. Интерфейс модуля представлен в таблице 2.

Таблица 2 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:

- resync_nrst
- resync_data
- spi2axi_wrap
- HerzelRegs
- div_all
- Angel
- Cordic
- DataScale
- Herzel

Упрощённая структурная схема модуля FourierTransform показана на рисунке 9.

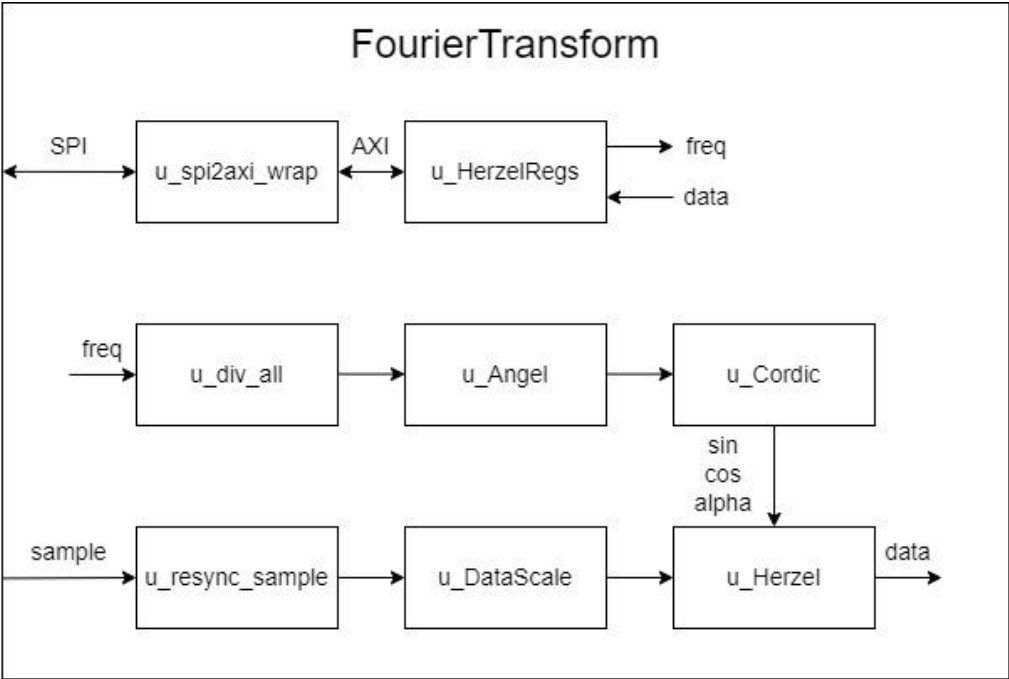


Рисунок 9 Структурная схема модуля FourierTransform

Для управления модулем предусмотрен SPI интерфейс.

Также содуль имеет следующие входные параметры:

NF – количество частот для вычисления.

DW – разрядность умножителя в модуле Herzel.

В параметре необходимо использовать 32 или 64 бита. При других разрядностях модуль не проверялся.

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 главного уровня представлена в таблице 3.

Таблица 3 Карта памяти модуля FourierTransform

Адрес	Тип	Сброс	Название	Описание
0x00	RW	0x29042023	VERSION	Версия модуля

Адрес	Тип	Сброс	Название	Описание
0x04	RW	0xF0F0F0F0	DEBUG	Тестовый регистр
0x0C	RW	0x000186A0	NUM_SAMP	Количество выборок для измерения
0x10	RW	0x00030D40	SAMP_FREQ	Частота выборки
0x14	RW	0x0	EN_CORDIC	Разрешение начала вычисления поворотных коэффициентов
0x18	R	0x0	STATUS	Регистр с информацией о стадиях вычисления: 0 бит – вычислены углы в Angel 1 бит – вычислены коэффициенты в Cordic 2 бит – вычислены данные по каждому модулю Herzel
0x1C	RW	0x0	RESET_ALL	Сброс всех модулей кроме регистров для нового расчёта
0x20	RW	0x0	RESET_H	Сброс только модуля Herzel для нового расчёта
от 0x10000000 до 0x10000000 + 0x4 * NF	RW	0x0	FREQ	Частота. Число регистров и соответственно число частот для расчёта задаётся параметром NF главного модуля.
от 0x20000000 до 0x20000000 + 0x4 * NF	RW	0x0	DATA	Рассчитанные данные для частот.

RW – регистр для записи и чтения.

R – регистр только для чтения.

Для начала работы сначала необходимо записать в регистры `FREQ_1-FREQ_NF` необходимые значения частот. А также если необходимо записать в регистры `NUM_SAMP`, `SAMP_FREQ` число выборок, которые будут считаны с АЦП и частоту с которой эти выборки будут производиться. 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 биты) и с регистра будет считываться 0x3.

После считывания 0x3 с регистра `STATUS` можно будет начинать отправлять действительные данные выборок с АЦП, используя `sample_p`, `sample_n`, `enable_p`, `enable_n` порты. Полученные данные, пройдя несколько вспомогательных модулей, будут последовательно поступать на модуль `Herzel` для расчёта. После прихода необходимого количества данных и завершения вычислений в бите 2 регистра `STATUS` установятся 1, после чего можно будет считывать данные с регистров `DATA_1- DATA_NF`.

6.3.1. IBUFDS

Входы `sample_p`, `sample_n`, `enable_p`, `enable_n` являются дифференциальными и для их преобразования в обычные сигналы используются специальный модуль IBUFDS.

Интерфейс модуля представлен в таблице 4.

Таблица 4 Интерфейс модуля IBUFDS

Название	Тип	Разрядность	Описание
I	ВХОД	[0]	Положительный вход дифференциального сигнала
IV	ВХОД	[0]	Отрицательный вход дифференциального сигнала
O	ВЫХОД	[0]	Выходной цифровой сигнал

Данный модуль является библиотечным элементом и необходим для синтеза, но не моделирования.

Данный модуль является одним из требований для приёма ПЛИС 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]
set_property -dict {IOSTANDARD LVDS PACKAGE_PIN ???} [get_ports sample_n[i]]
```

И так далее для каждого дифференциального сигнала.

При этом, стоит отметить, что данный модуль может не использоваться в определённых программах синтеза.

6.3.2. resync_nrst

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

Интерфейс модуля представлен в таблице 5.

Таблица 5 Интерфейс модуля resync_nrst

Название	Тип	Разрядность	Описание
clk	ВХОД	[0]	Тактовый сигнал
rstn_i	ВХОД	[0]	Асинхронный сброс
rstn_o	ВЫХОД	[0]	Синхронный сброс

6.3.3. resync_data

Данный модуль просто пересинхронизирует входной сигнал данных.

Делается это из-за того, что данные выходящие с АЦП и данные обрабатываемые в цифровой схеме находятся в асинхронных тактовых доменах.

Интерфейс модуля представлен в таблице 6.

Таблица 6 Интерфейс модуля resync_data

Название	Тип	Разрядность	Описание
rstn	ВХОД	[0]	Сброс
clk	ВХОД	[0]	Тактовый сигнал
data_i	ВХОД	[DW-1:0]	Асинхронные данные
data_o	ВЫХОД	[DW-1:0]	Синхронные данные

Параметр DW – ширина пересинхронизируемых данных.

6.3.4. spi2axi

Данный модуль преобразует SPI транзакции в AXI-lite транзакции.

Интерфейс модуля представлен в таблице 7.

Таблица 7 Интерфейс модуля resync_data

Название	Тип	Разрядность	Описание
axi_arstn_i	ВХОД	[0]	Сброс
axi_aclk_i	ВХОД	[0]	Тактовый сигнал
spi_sck_i	ВХОД	[0]	Тактовый сигнал SPI
spi_ss_n_i	ВХОД	[0]	Разрешение приёма/передачи данных SPI
spi_mosi_i	ВХОД	[0]	Данные от ведущего
spi_miso_o	ВЫХОД	[0]	Данные от ведомого
axio_o	ВЫХОД	axi_lite_mosi	AXI-lite шина от ведущего
axii_i	ВХОД	axi_lite_miso	AXI-lite шина от ведомого

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.

Модуль основан на машине состояний состоящей из двух частей:

1. Машина состояний SPI:

- SPI_IDLE – ожидание падения сигнала spi_ss_n_i, сигнализирующего начала spi2axi транзакции.
- SPI_CMD – считывание с spi_mosi_i байта инструкции для последующей передачи по axi.

- SPI_ADDR – считывание с spi_mosi_i четырёх байт адреса для последующей передачи по axi.
- SPI_WDATA – считывание с spi_mosi_i четырёх байт данных для последующей передачи по axi.
- SPI_RDATA – считывание с axi четырёх байт данных для последующей передачи по spi_miso_i.
- SPI_DUMM – пустая передача одного байта с целью дать axi интерфейсу завершить свою транзакцию считывания/записи данных.
- SPI_STAT – передача статуса spi2axi транзакции. В реализации передаёт статус axi транзакции: 0 – всё хорошо, 3 – всё плохо.

Граф машины состояний SPI показан на рисунке 10.

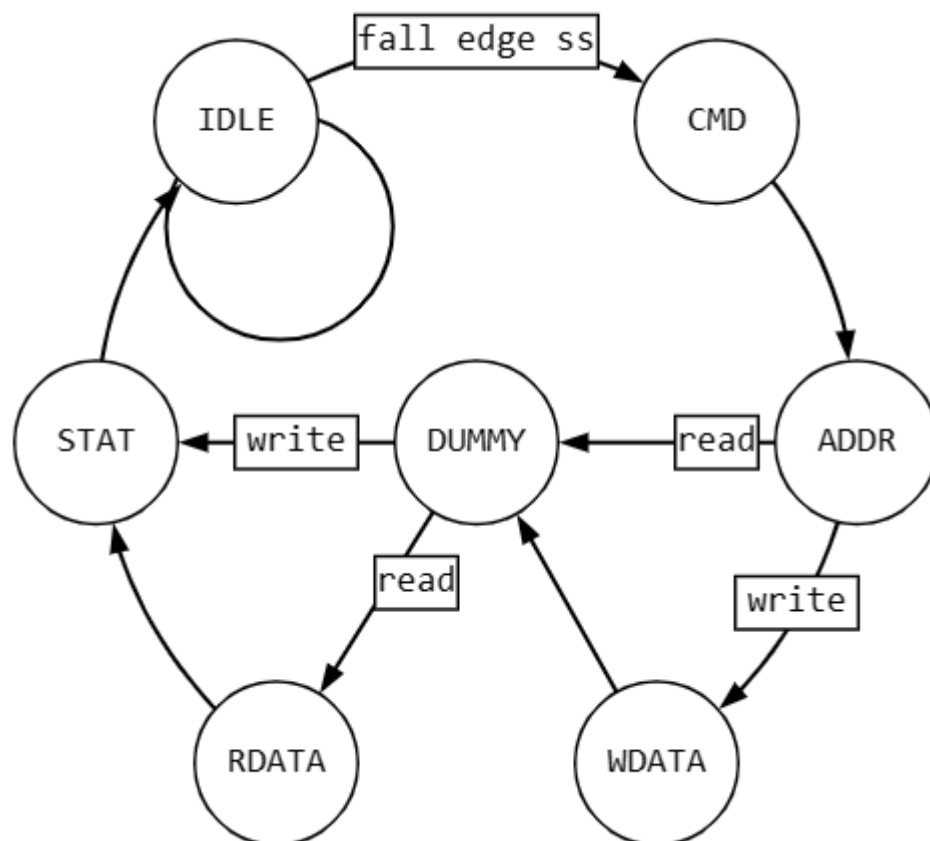


Рисунок 10 Граф состояний модуля HerzelRegs

2. Машина состояний AXI

- AXI_IDLE – сброс axi сигналов.

- AXI_CMD – ожидание записи cmd по spi и сброс некоторых регистров.
- AXI_RADDR – отправка по axi адреса.
- AXI_RDATA – считывание по axi данных.
- AXI_WADDR – отправка по axi адреса.
- AXI_WDATA – запись по axi данных.
- AXI_WRESP – считывание по axi ответа.

Граф машины состояний AXI показан на рисунке 11.

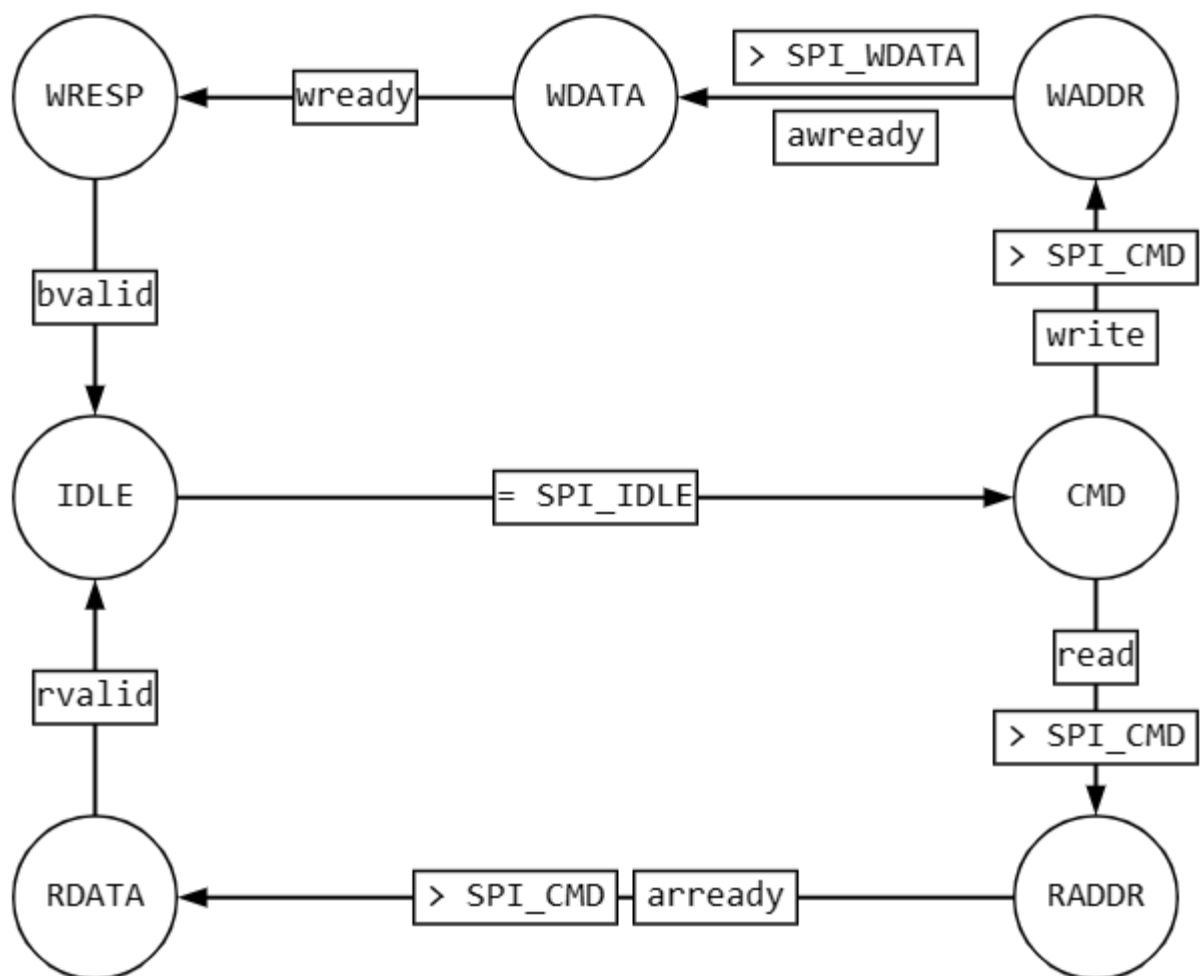


Рисунок 11 Граф состояний модуля HerzelsReg

Для передачи транзакции чтения/записи необходимо соблюсти протокол – таблица 8 и таблица 9. Данный протокол используется в SPI интерфейсе spi_if во время моделирования для создания транзакций.

Таблица 8 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	Статус транзакции от ведомого

Таблица 9 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	Статус транзакции от ведомого

6.3.5. HerzelRegs

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

Интерфейс модуля представлен в таблице 10.

Таблица 10 Интерфейс модуля HerzelRegs

Название	Тип	Разрядность	Описание
rstn	ВХОД	[0]	Сброс
clk	ВХОД	[0]	Тактовый сигнал
freq_arr_o	ВЫХОД	[NF-1:0][31:0]	Массив частот для вычисления
en_cordic_o	ВЫХОД	[0]	Разрешение работы модуля Cordic
valid_angel_i	ВХОД	[0]	Сигнал завершения вычисления модуля Angel
valid_cordic_i	ВХОД	[0]	Сигнал завершения вычисления модуля Cordic
valid_herzel_i	ВХОД	[NF-1:0]	Сигналы завершения вычисления модулей Herzel
data_arr_i	ВХОД	[NF-1:0][31:0]	Массив данных по каждой частоте
num_samp_o	ВЫХОД	[31:0]	Количество выборок
samp_freq_o	ВЫХОД	[31:0]	Частота выборки
reset_all_o	ВЫХОД	[0]	Сброс всех модулей
reset_h_o	ВЫХОД	[0]	Сброс модуля Herzel
axio_i	ВХОД	axi_lite_miso	AXI-lite шина от ведущего

Название	Тип	Разрядность	Описание
axii_o	ВЫХОД	axi_lite_miso	AXI-lite шина от ведомого

Модуль основан на машине состояний:

- 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 (wwvalid). 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.

Граф машины состояний показан на рисунке 12.

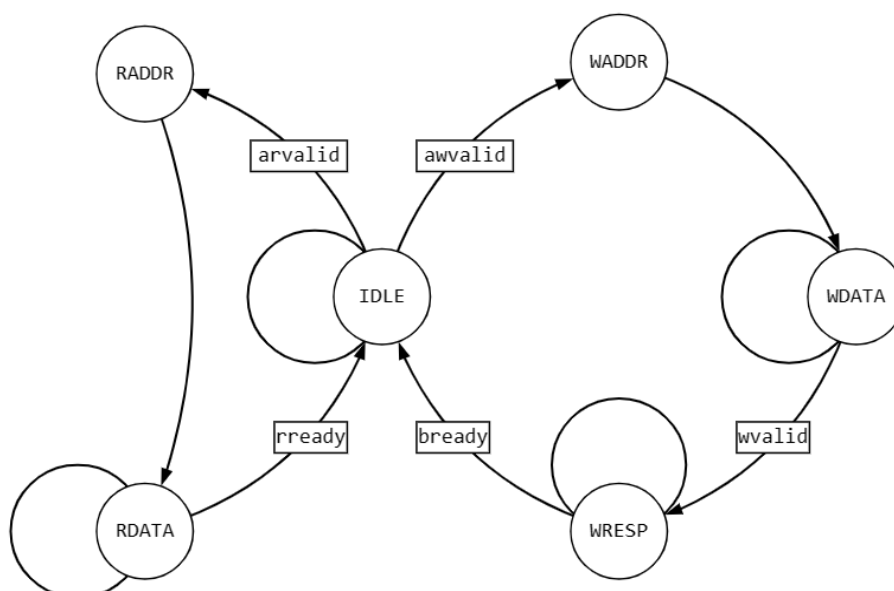


Рисунок 12 Граф состояний модуля HerzelRegs

Также поток данных удобно представить в таблице 11.

Таблица 11 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

6.3.6. div_all

Данный модуль предназначен для деления различных данных в схеме. Для этого модуль включает в себя делитель divu.sv [2] с целой и дробной частью.

Интерфейс модуля представлен в таблице 12.

Таблица 12 Интерфейс модуля resync_data

Название	Тип	Разрядность	Описание
rstn	ВХОД	[0]	Сброс
clk	ВХОД	[0]	Тактовый сигнал
en	ВХОД	[0]	Разрешение начала вычисления
valid	ВЫХОД	[0]	Сигнал завершения расчёта
num_samp_i	ВХОД	[31:0]	Количество выборок
samp_freq_i	ВХОД	[31:0]	Частота выборки
freq_i	ВХОД	[NF-1:0][31:0]	Массив частот для расчёта
k_arr_o	ВЫХОД	[NF-1:0][31:0]	Индексы частот
ang_coef_o	ВЫХОД	[63:0]	Коэффициент для Cordic
ns_coef_o	ВЫХОД	[63:0]	Коэффициент для Herzel

Модуль основан на машине состояний:

- IDLE – ожидание сигнала разрешения начала вычислений.
- DF – вычисление минимальной разницы между двумя частотами по $\text{samp_freq_i}/\text{num_samp_i}$.

- KARR – вычисление индексов частот по freq/df .
- ANGС – вычисление коэффициента для Cordic модуля.
- NS – вычисление коэффициента для Herzel модуля.
- VALID – установка сигнала готовности данных и завершение вычислений.

Граф машины состояний показан на рисунке 13.

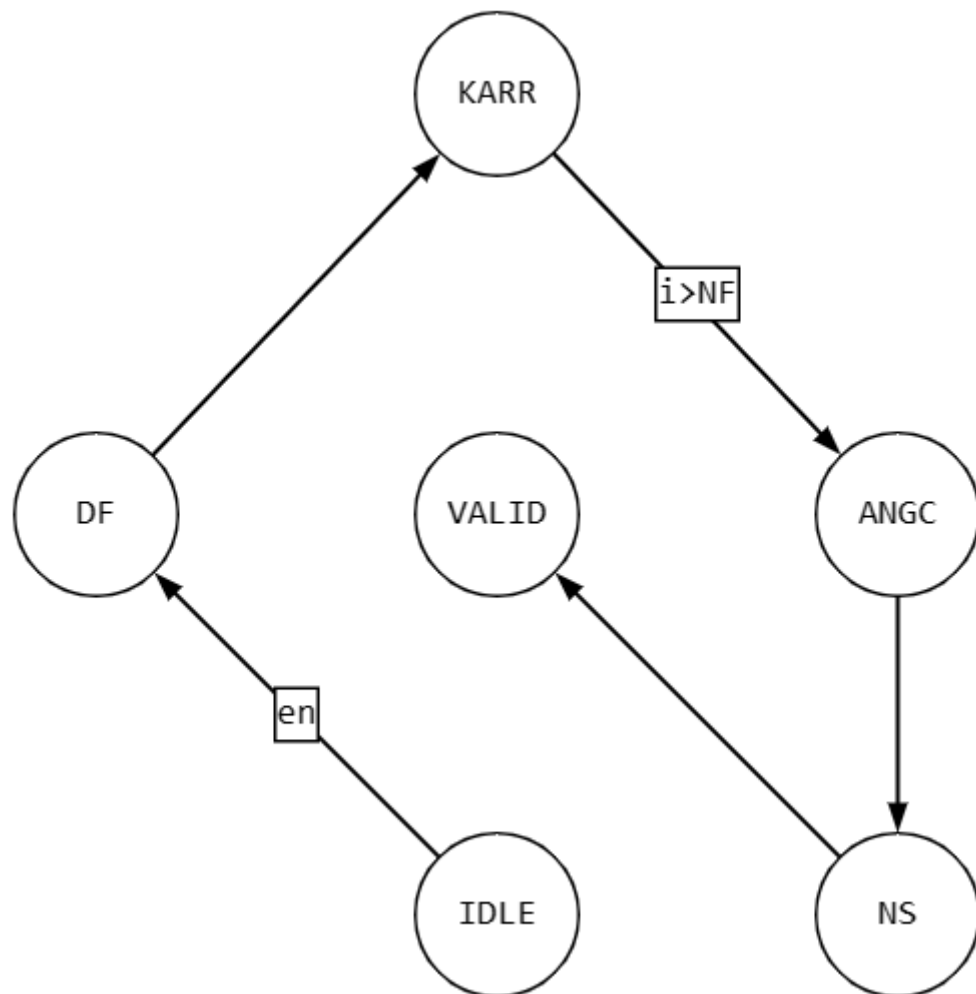


Рисунок 13 Граф состояний модуля `div_all`

6.3.7. 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.

Интерфейс модуля представлен в таблице 13.

Таблица 13 Интерфейс модуля Angel

Название	Тип	Разрядность	Описание
rstn	ВХОД	[0]	Сброс
clk	ВХОД	[0]	Тактовый сигнал
en	ВХОД	[0]	Разрешение начала вычисления
valid	ВЫХОД	[0]	Сигнал завершения расчёта
k_arr_i	ВХОД	[NF-1:0][31:0]	Массив индексов частот
ang_coef_i	ВХОД	[63:0]	Коэффициент для вычисления
angel_o	ВЫХОД	[NF-1:0][63:0]	Вычисленный массив углов

В модуль заходит массив частот, записанных в регистры $FREQ_1$ - $FREQ_NF$. До этого для каждой частоты рассчитывается в модуле div_all номерной коэффициент k по формуле:

$$k = \frac{FREQ}{df}, \text{ где } df = \frac{f}{NS} = 2Hz$$

f – частота дискретизации (200 кГц).

NS – число выборок (100 т.шт).

Так как $df = 2Hz$, то данная операция заменяется сдвигом вправо.

После коэффициент k домножается на $2\pi/N$ – коэффициент (ang_coef_i), который также заранее рассчитывается в модуле div_all. Таким образом, получается угол для последующих расчётов.

6.3.8. Cordic

Данный модуль производит поворотных коэффициентов, а также специального коэффициента alpha по углам передаваемых с модуля Angel. Расчёт производится с помощью алгоритма CORDIC [3].

Интерфейс модуля представлен в таблице 14.

Таблица 14 Интерфейс модуля Cordic

Название	Тип	Разрядность	Описание
rstn	ВХОД	[0]	Сброс
clk	ВХОД	[0]	Тактовый сигнал
en	ВХОД	[0]	Разрешение начала вычисления
valid	ВЫХОД	[0]	Сигнал завершения расчёта
ang_i	ВХОД	[NF-1:0][31:0]	Массив углов для вычисления
cos_o	ВЫХОД	[NF-1:0][63:0]	Вычисленный массив косинусов
sin_o	ВЫХОД	[NF-1:0][63:0]	Вычисленный массив синусов
alpha	ВЫХОД	[NF-1:0][63:0]	Вычисленный массив коэффициента альфа

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, арктангенс которого кратен степени 2 (для операции сдвига).

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

Вышеизложенную последовательность алгоритма можно увидеть на рисунке 14.

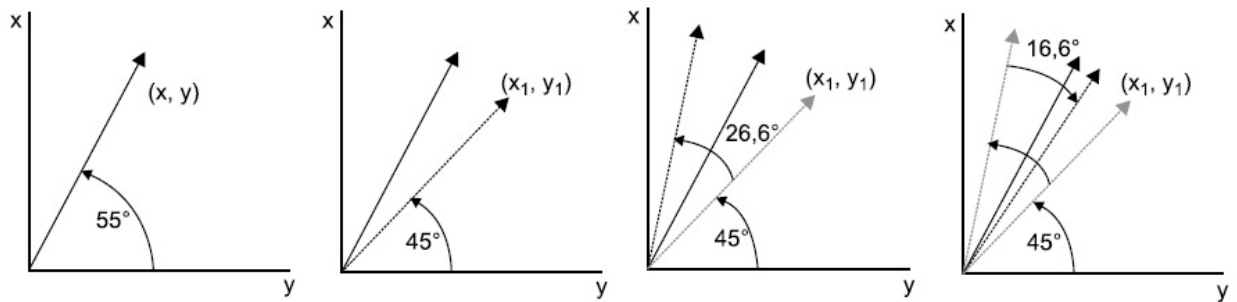


Рисунок 14 CORDIC алгоритм

Координаты x и y вычисляются по формулам:

$$x_{i+1} = x_i \cdot \cos \varphi - y_i \cdot \sin \varphi$$

$$y_{i+1} = x_i \cdot \sin \varphi + y_i \cdot \cos \varphi$$

После преобразования:

$$x_{i+1} = \cos \varphi \cdot (x_i - y_i \cdot \tan \varphi)$$

$$y_{i+1} = \cos \varphi \cdot (y_i + x_i \cdot \tan \varphi)$$

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

Итоговая формула:

$$x_{i+1} = x_i - \sigma_i \cdot y_i \cdot 2^{-i}$$

$$y_{i+1} = y_i + \sigma_i \cdot x_i \cdot 2^{-i}$$

$$ang_{i+1} = ang_i - \tan \varphi_i$$

$$\sigma_i = \frac{ang}{|ang|} = \pm 1$$

$$K(n) = \prod_{i=0}^{ns-1} \frac{1}{\sqrt{1+2^{-2i}}} = 0.607253, ns = 24$$

При этом данный алгоритм верен для случая $\varphi < \pi/2$, в ином случае сперва надо определить квадрант, и с помощью вычитания перевести угол в

первый квадрант, и после стандартного вычисления воспользоваться формулами приведения.

Модуль основан на машине состояний, её граф приведён на рисунке 15.

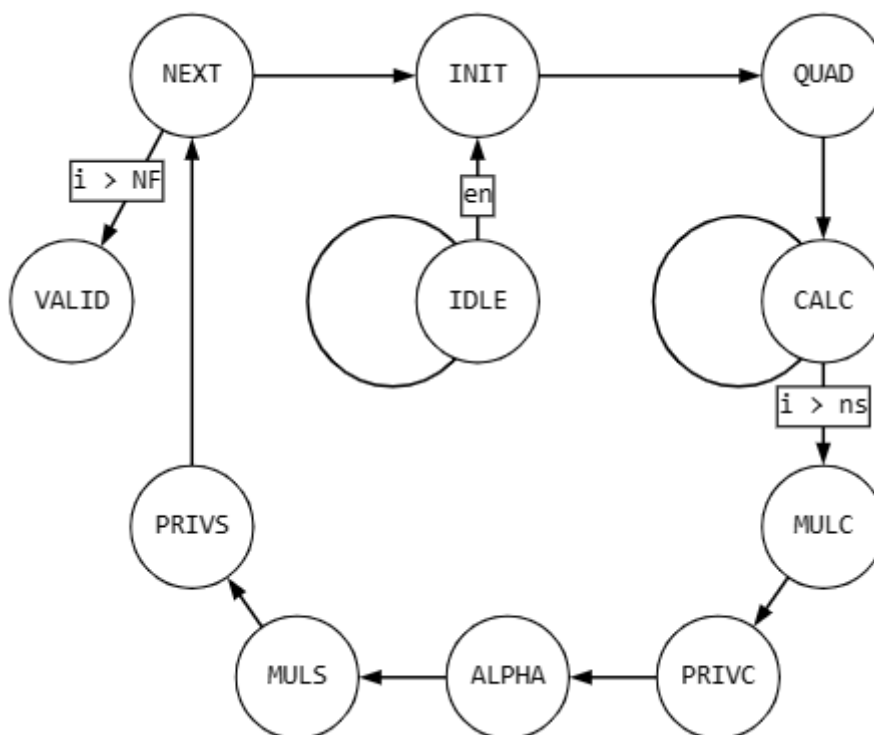


Рисунок 15 Граф состояний модуля Cordic

- IDLE – ожидание сигнала разрешения начала вычислений.
- INIT – выбор угла для расчёта.
- QUAD – расчёт квадранта выбранного угла. И приведение угла к первому квадранту.
- CALC – расчёт косинуса и синуса приведённого угла по алгоритму CORDIC.
- MULC – умножение рассчитанного косинуса на коэффициент масштабирования K .
- PRIVC – приведение косинуса к начальному квадранту, определённом в QUAD состоянии, по формулам приведения.
- ALPHA – вычисление коэффициента альфа.
- MULS – умножение рассчитанного синуса на коэффициент масштабирования K .

- **PRIVS** – приведение синуса к начальному квадранту, определённого в QUAD состоянии, по формулам приведения.
- **NEXT** – сброс значений, увеличение счётчика выбора угла.
- **VALID** – завершение вычислений, установка сигнала действительности данных valid.

6.3.9. DataScale

Данный модуль предназначен для масштабирования входных данных по следующей формуле:

$$data = \frac{13}{256} \cdot sample$$

13/256 – коэффициент, определяемый АЦП.

Также выходные данные доводятся до нужного формата и ширины.

Интерфейс модуля представлен в таблице 15.

Таблица 15 Интерфейс модуля DataScale

Название	Тип	Разрядность	Описание
rstn	ВХОД	[0]	Сброс
clk	ВХОД	[0]	Тактовый сигнал
enable	ВХОД	[0]	Разрешение начала вычисления
valid	ВЫХОД	[0]	Сигнал завершения расчёта
data_i	ВХОД	[7:0]	Данные с АЦП
data_o	ВЫХОД	[31:0]	Преобразованные данные

6.3.10. Herzel

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

Подробный вывод формул – [4]

Интерфейс модуля представлен в таблице 16.

Таблица 16 Интерфейс модуля Herzel

Название	Тип	Разрядность	Описание
rstn	ВХОД	[0]	Сброс
clk	ВХОД	[0]	Тактовый сигнал
en	ВХОД	[0]	Разрешение начала вычисления
valid	ВХОД	[0]	Сигнал завершения расчёта
ns_i	ВХОД	[31:0]	Количество выборок
ns_coef_i	ВХОД	[63:0]	Специальный коэффициент
alpha_i	ВХОД	[63:0]	Коэффициент альфа
cW_re_i	ВХОД	[63:0]	Косинус
cW_im_i	ВХОД	[63:0]	Синус
data_i	ВХОД	[31:0]	Входные данные
data_o	ВЫХОД	[31:0]	Выходные данные

Для реализации Фурье преобразования, то есть для перевода данных из временного в частное измерение был использован алгоритм Гёрцеля. Данный

алгоритм позволяет произвести расчет лишь фиксированного количества спектральных отсчетов.

По алгоритму спектральный отсчет $S(k)$ равен:

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

Где v - промежуточные значения, которые рассчитываются итерационно:

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

W – поворотный коэффициент

$$W_N^{nk} = \exp\left(-j \frac{2\pi}{N} nk\right), k = 0 \dots N-1$$

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

Коэффициент ns_coef_i рассчитывается как $1/NS$. Необходим для вычисления точного выходного значения, которое искажается из-за рекурсивного характера преобразования.

Таким образом, понадобится несколько операций умножения. Для уменьшения использования площади схемы и потребляемых ресурсов платы модуль основан на машине состояний, которая позволяет использовать один умножитель на одну расчётную частоту.

Модуль основан на машине состояний:

- CALC – вычисление промежуточных значений v .
- MULR – умножение v на Re часть поворотного коэффициента.
- MULI – умножение v на Im часть поворотного коэффициента.
- NORMR – деление на NS Re часть.
- NORMI – деление на NS Im часть.
- GRADR – возведение в степень Re часть.
- GRADI – возведение в степень Im часть.

- **VALID** – завершение вычислений, установка сигнала действительности данных valid.

Граф машины состояний представлен на рисунке 16.

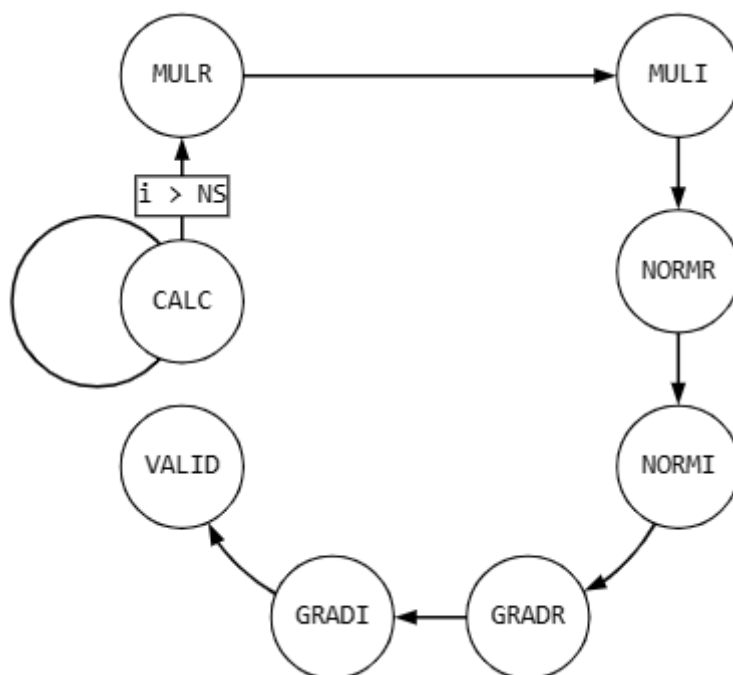


Рисунок 16 Граф состояний модуля Herzel

Для реализации операции умножения используется модуль mult_sign_clk [6].

6.4. Тестирование модуля Фурье преобразования

Алгоритм тестирования модуля Фурье преобразования описан в модуле `FourierTransform_tb` и приведён в приложении 2. Также в этом приложении приведены результаты теста и описание тестового SPI интерфейса - `spi_if` для моделирования SPI транзакций во время теста.

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` которые генерируются в `MatchCad`.

`sample.csv` – содержит данные выборки, которые поступают на вход `sample_p`.

`freq.csv` – содержит значения частот, которые будут вычисляться. Для этого они записываются в регистры `FREQ`.

`data.csv` – рассчитанные данные из `MatchCad` по эталонной модели.

После того как `FourierTransform` завершит расчёт, данные для каждой частоты из `FourierTransform` будут сравниваться с данными из `MatchCad` (`data.csv`). Если значения не будут совпадать с 5% погрешностью, то тест будет провален.

Примеры некоторых моделирований приведены в приложении 2.

Выборки выбирались с разной частотой из сигнала сгенерированного по следующей формуле:

$$f(t) = \sum_{i=0}^{NF-1} [amp_i \cdot (\cos(2\pi \cdot freq_i) + 1)] \quad (1)$$

7. Реализация на ПЛИС

Структурная схема реализации на ПЛИС показана на рисунке 17.

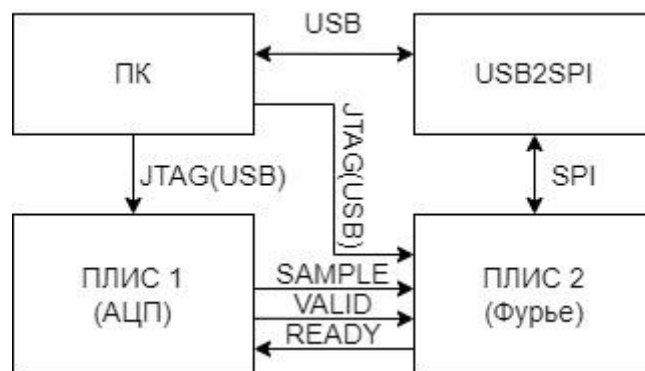


Рисунок 17 Структурная схема проекта

Для тестирования используется две отладочные платы. Первая используется для реализации модели АЦП, аппаратное описание которого приведено в приложении 1. Данная модель загружает в память выборки сгенерированные по формуле 1. Далее модель ждёт пока придёт сигнал READY от второй плис. Данный сигнал отсутствует в основном проекте и введён только для реализации на ПЛИС, для того, чтобы АЦП раньше времени не отправил все выборки впустую, а отправил когда будет возможность их принять, так как количество выборок, которое можно записать ограничено. Когда придёт сигнал READY ПЛИС 1 начнёт с некоторым периодом отправлять выборки SAMPLE с сигналом VALID в ПЛИС 2.

Вторая ПЛИС реализует в себе модуль FourierTransform и отличается только наличием дополнительного сигнала READY, а также отсутствием дифференциальных составляющих сигнала для удобства соединения двух ПЛИС. В остальном данный модуль ничем не отличается от изложенного в предыдущем разделе.

При этом для управления ПЛИС 2 необходим USB2SPI мост. Данную роль выполняет плата UMFT4222EV [7]. Для управления платой предусмотрена библиотека [8], которая использовалась для тестирования. Тест и результат приведён в приложении 1.

8. Заключение

Был спроектирован АЦП последовательного приближения и спроектирована цифровая схема Фурье преобразования на основе метода Гёрцеля.

В цифровой схеме Фурье преобразования была реализованна регистровая модель, а также SPI интерфейс для связи с внешним устройством управления.

Полученный модуль преобразования Фурье последовательно обрабатывает данные приходящие с АЦП, что не требует в свою очередь хранения оцифрованных данных в памяти и ожидания конца оцифровывания всех необходимых данных.

Недостатком данного метода является его рекурсивная природа. Из-за чего при большом количестве измерений требуется большая точность, что в свою очередь увеличивает число используемых элементов. Так для добавления одной частоты для расчёта требуется 800 элементов (при реализации на ПЛИС).

Полученна схема была реализована на ПЛИС и протестированна.

9. Приложение 1

Была синтезирована схема, которая показана на рисунке 18. Число используемых элементов на ПЛИС приведено на рисунке 19.

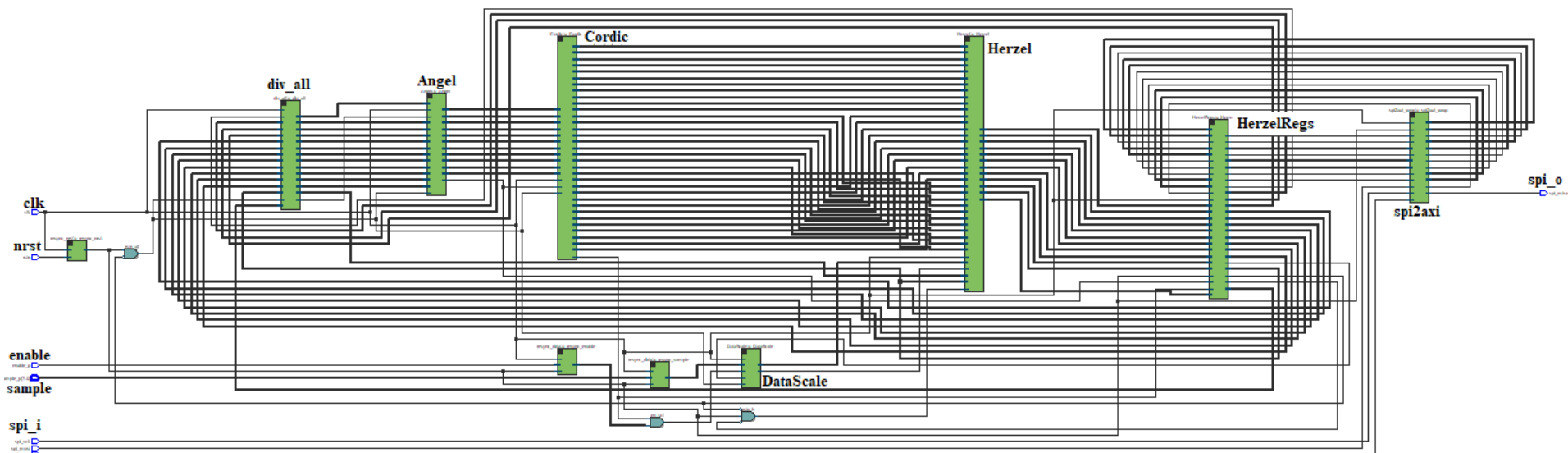


Рисунок 18 Синтезированная схема проекта

		Timing Models		Final
Flow Status	Successful - Fri Jun 16 22:59:59 2023	Total logic elements	5,524 / 6,272 (88 %)	
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition	Total registers	2798	
Revision Name	SRW	Total pins	16 / 92 (17 %)	
Top-level Entity Name	FourierTransform	Total virtual pins	0	
Family	Cyclone IV E	Total memory bits	0 / 276,480 (0 %)	
Device	EP4CE6E22C8	Embedded Multiplier 9-bit elements	30 / 30 (100 %)	

Рисунок 19 Используемые элементы ПЛИС

Реализация проекта на плис показана на рисунке 20. Результат теста на рисунке 21.

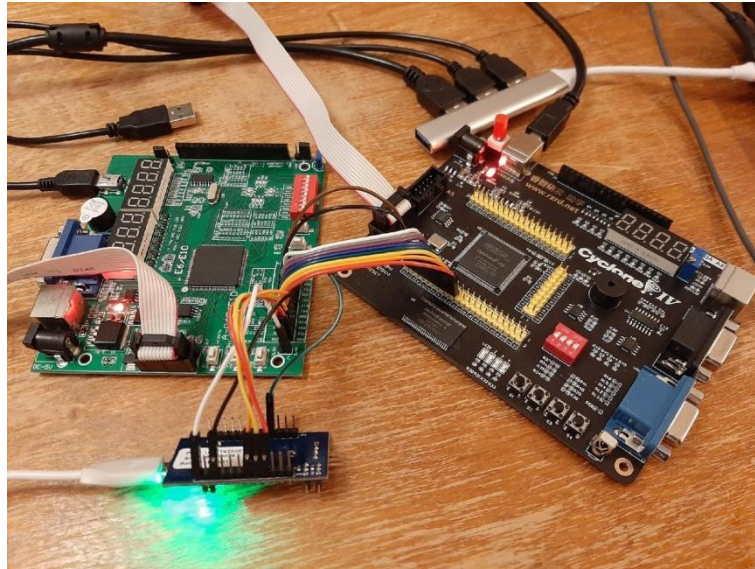


Рисунок 20 Реализация проекта на ПЛИС

```
Write addr - 000c, data - 1388  
Status - 00
```

```
Write addr - 0010, data - 2710  
Status - 00
```

```
Write addr - 10000000, data - 07d0  
Status - 00
```

```
Write addr - 10000004, data - 0fa0  
Status - 00
```

```
Write addr - 0014, data - 0001  
Status - 00
```

```
Read addr - 20000000  
Status - 00
```

```
Read addr - 20000004  
Status - 00
```

```
Result of test:
```

```
Freq 1 : 008fcddd
```

```
Freq 2 : 00190045
```

```
Process finished with exit code 0
```

Рисунок 21 Вывод в консоль

9.1. Модель АЦП

```
module adc #(
    parameter MW = 5000
) (
    input          nrst ,
    input          clk   ,
    input          ready ,
    output logic    valid ,
    output logic [7:0] sample,
    output logic    led
);

logic nrst_syn ;
logic ready_syn;

logic [7 :0] memory [MW-1:0];
logic [31:0] addr;
logic        en   ;
logic [31:0] cntr;

resync_nrst u_resync_nrst (
    .clk   (clk      ),
    .rstn_i(nrst     ),
    .rstn_o(nrst_syn)
);

resync_data u_resync_data (
    .rstn   (nrst_syn ),
    .clk    (clk      ),
    .data_i (ready     ),
    .data_o (ready_syn)
);

initial begin
    $readmemb("D:/Desktop/Study_now/SRW/GoertzelAlgorithm/src/sim/data/sample_bin.csv", memory);
end

always_ff @(posedge clk, negedge nrst_syn) begin
    if (!nrst_syn) begin
        cntr <= 0;
        en   <= 0;
        led <= 0;
    end
end
```

```

else begin
    led <= 1;
    if (cntr < 250) begin
        cntr <= cntr + 1;
        en    <= 0;
    end
    else begin
        cntr <= 0;
        en    <= 1;
    end
end
end
end

always_ff @(posedge clk, negedge nrst_syn) begin
    if (!nrst_syn) begin
        valid  <= 0;
        sample <= 0;
        addr   <= 0;
    end
    else begin
        if (ready_syn && en && addr < MW) begin
            valid  <= 1;
            sample <= memory[addr];
            addr   <= addr + 1;
        end
        else if (cntr > 20) begin
            valid  = 0;
            sample = 0;
        end
    end
end
end

endmodule

```


9.2. Тест ПЛИС

```
import ft4222
from ft4222.SPI import Cpha, Cpol
from ft4222.SPIMaster import Mode, Clock, SlaveSelect
from time import sleep

VERSION      = 0x0000_0000  # RW
DEBUG        = 0x0000_0004  # RW
MODE         = 0x0000_0008  # RW
NUM_SAMP     = 0x0000_000C  # RW
SAMP_FREQ    = 0x0000_0010  # RW
EN_CORDIC    = 0x0000_0014  # RW
STATUS       = 0x0000_0018  # R
RESET_ALL    = 0x0000_001C  # RW
RESET_H      = 0x0000_0020  # RW

FREQ_1       = 0x1000_0000  # RW
FREQ_2       = 0x1000_0004  # RW

DATA_1       = 0x2000_0000  # R
DATA_2       = 0x2000_0004  # R

read_byte = 0x01
write_byte = 0x00
dummy_byte = 0x00

def write(device, addr, data):
    print(f'Write addr - {addr:04x}, data - {data:04x}')
    device.spiMaster_SingleWrite(write_byte, False)
    byte = addr.to_bytes(4, byteorder="big")
    device.spiMaster_SingleWrite(byte[0:1], False)
    device.spiMaster_SingleWrite(byte[1:2], False)
    device.spiMaster_SingleWrite(byte[2:3], False)
    device.spiMaster_SingleWrite(byte[3:4], False)
    byte = data.to_bytes(4, byteorder="big")
    device.spiMaster_SingleWrite(byte[0:1], False)
    device.spiMaster_SingleWrite(byte[1:2], False)
    device.spiMaster_SingleWrite(byte[2:3], False)
    device.spiMaster_SingleWrite(byte[3:4], False)
    device.spiMaster_SingleWrite(dummy_byte, False)
    print(f'Status - {device.spiMaster_SingleRead(0x01,
True)}.hex()}\n')

def read(device, addr):
    print(f'Read addr - {addr:04x}')
    device.spiMaster_SingleWrite(read_byte, False)
    byte = addr.to_bytes(4, byteorder="big")
    device.spiMaster_SingleWrite(byte[0:1], False)
    device.spiMaster_SingleWrite(byte[1:2], False)
```

```

device.spiMaster_SingleWrite(byte[2:3], False)
device.spiMaster_SingleWrite(byte[3:4], False)
device.spiMaster_SingleWrite(dummy_byte, False)
data = bytearray()
data.append(device.spiMaster_SingleRead(0x01, False)[0])
data.append(device.spiMaster_SingleRead(0x01, False)[0])
data.append(device.spiMaster_SingleRead(0x01, False)[0])
data.append(device.spiMaster_SingleRead(0x01, False)[0])
print(f'Status - {device.spiMaster_SingleRead(0x01,
True)}.hex()}\n')
return data.hex()

def test():
    # open 'device' with default description 'FT4222 A'
    device = ft4222.openByDescription('FT4222 A')

    # init spi master
    device.spiMaster_Init(Mode.SINGLE, Clock.DIV_128,
Cpol.IDLE_LOW, Cpha.CLK_LEADING, SlaveSelect.SS0)

    write(device, NUM_SAMP , 5000 )
    write(device, SAMP_FREQ, 10000)

    write(device, FREQ_1, 2000)
    write(device, FREQ_2, 4000)

    write(device, EN_CORDIC, 1)

    sleep(1)

    data1 = read(device, DATA_1)
    data2 = read(device, DATA_2)

    print('Result of test:')
    print(f'Freq 1 : {data1}')
    print(f'Freq 2 : {data2}')

if __name__ == '__main__':
    test()

```

10. Приложение 2

10.1. Тест модуля Фурье преобразования

```
`timescale 1ns/1ns
`include "./FourierTransformAM.sv"

`define TEST

module FourierTransform_tb;

localparam CLK_PER      = 5    ;
localparam ADC_CLK_PER = 500   ;
localparam SPI_CLK_PER = 1000;
localparam NF           = 12   ;
localparam DW           = 64   ;

logic ok    ;

integer fd_r_s;
integer fd_r_f;
integer fd_r_d;
integer fd_r_l;
integer fd_w_v;
integer fd_w_r;

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] mcad_flog;
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.sck ;
assign spi_ss_n     = spi_if.ss_n;
assign spi_mosi     = spi_if.mosi;
assign spi_if.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),
    .DW(DW)
) 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;
    fd_r_l =
$ fopen("D:/Desktop/Study_now/SRW/GoertzelAlgorithm/src/sim/data/logic.csv"
, "r");
    if (fd_r_l == 0) $finish;
    fd_w_v =
$ fopen("D:/Desktop/Study_now/SRW/GoertzelAlgorithm/src/sim/data/vector.cs
v", "w");
    if (fd_w_v == 0) $finish;
    fd_w_r =
$ fopen("D:/Desktop/Study_now/SRW/GoertzelAlgorithm/src/sim/data/result.cs
v", "w");
    if (fd_w_r == 0) $finish;
end
final begin
    $fclose(fd_r_s);
    $fclose(fd_r_f);
    $fclose(fd_r_d);
    $fclose(fd_r_l);
    $fclose(fd_w_v);
    $fclose(fd_w_r);
end

initial begin
    @(posedge enable_p);
    while (!(DUT.u_Herzel.valid[0])) begin
        wait(&DUT.u_Herzel.mul_valid)
        @(posedge DUT.clkd);
        $fwrite(fd_w_v, "%h\n", DUT.u_Herzel.vm1[0]);
    end
    $fwrite(fd_w_v, "%h\n", DUT.u_Herzel.vm1[0]);
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]);
        mcad_data[i] = mcad_data[i] * 32'h1_0000;
    end
    for (int i = 0; i < NF; i = i + 1) begin

```

```

        $fscanf(fd_r_l, "%d\n", mcad_flog[i]);
    end
end

task end_of_test();
    if (ok)
        $display("[%010t] TEST SUCCESS", $time);
    else
        $display("[%010t] TEST FAILED", $time);
        $display("Result:");
        for (int i = 0; i < NF; i = i + 1) begin
            if (mcad_flog[i] == 1) begin
                $display("  %02d: %05d. MCAD - %08h, VLOG - %08h", i, mcad_freq[i],
mcad_data[i], vlog_data[i]);
            end
        end
    endtask

task herzel_wait_all_valid();
    while (spi_data&STATUS_HERZEL_MSK != STATUS_HERZEL_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
    @(posedge DUT.clkd);
    if (fd_r_s != 0)
        $fclose(fd_r_s);
    fd_r_s =
    $fopen("D:/Desktop/Study_now/SRW/GoertzelAlgorithm/src/sim/data/sample.cs
v", "r");
    while (!$feof(fd_r_s)) begin
        #(ADC_CLK_PER/2);
        enable_p = 1;
        $fscanf(fd_r_s, "%d\n", sample_p);
        #(ADC_CLK_PER/2);
        enable_p = 0;
    end
    herzel_wait_all_valid();
endtask

```

```

initial begin
    ok          = 1;
    clk         = 0;
    rstn        = 0;
    enable_p    = 0;
    sample_p    = 0;
    spi_data    = 0;
    vlog_data    = 0;
    spi_if.init();
    repeat(20) @(posedge DUT.clkd);
    rstn = 1;
    repeat(20) @(posedge DUT.clkd);

    repeat(1) begin
        spi_if.write_data(RESET_ALL, 32'd1, spi_stat);
        repeat(20) @(posedge DUT.clkd);
        spi_if.write_data(RESET_ALL, 32'd0, spi_stat);

        $display("[%010t] 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

        spi_if.write_data(NUM_SAMP , 32'd5000, spi_stat);
        spi_if.write_data(SAMP_FREQ, 32'd10000, spi_stat);

        $display("[%010t] Start Herzel", $time);
        herzel();

        $display("[%010t] Read result", $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);
            $fwrite(fd_w_r, "%d\n", vlog_data[i][31:16]);
        end

        $display("[%010t] Start check", $time);
        for (int i = 0; i < NF; i = i + 1) begin
            if (mcad_flog[i] == 1) begin
                if ((vlog_data[i] < mcad_data[i]*0.95) | (vlog_data[i] >
mcad_data[i]*1.05)) begin
                    $display("[%010t] Error at freq %0d: %0d. MCAD - %0h, VLOG -
%0h", $time, i, mcad_freq[i], mcad_data[i], vlog_data[i]);
                    ok = 0;
                end
            end
        end
    end
end

```

```
        end
    end
end

    # 5000;
    end_of_test();
    $stop;
end

endmodule
```


10.2. Описание SPI интерфейса для тестирования

```
`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_0000;

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];
        @(negedge 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
        @(posedge sck);
        data[i] = mst.miso;
    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
            write(32, addr);
            // dummy byte
            write(8, dummy_word);
            // 4 data bytes
            read(32, data);
            // status byte
            read(8, status);
            #(SPI_CLK_PER/2);
        end
    join_any
    disable fork;
    #(SPI_CLK_PER/2);
    // slave select
    mst.ss_n = 1;
    if (DISPLAY) $display("[%010t] spi read : addr = 0x%08h, data = 0x%08h,
status = 0x%02h", $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
      write(32, addr);
      // 4 data bytes
      write(32, data);
      // dummy byte
      write(8, dummy_word);
      // status byte
      read(8, status);
      #(SPI_CLK_PER/2);
    end
  join_any
  disable fork;
  #(SPI_CLK_PER/2);
  // slave select
  mst.ss_n = 1;
  if (DISPLAY) $display("[%010t] spi write: addr = 0x%08h, data = 0x%08h,
status = 0x%02h", $time, addr, data, status);
endtask
```

```
endinterface
```

10.3. Результаты моделирования

Тест 1. По формуле 1 формируется сигнал с параметрами, показанными на рисунке 22. После чего с сигнала делаются выборки с частотой 200 кГц, число выборок – 100 т.шт. Данные параметры записываются в нужные регистры и начинается расчёт.

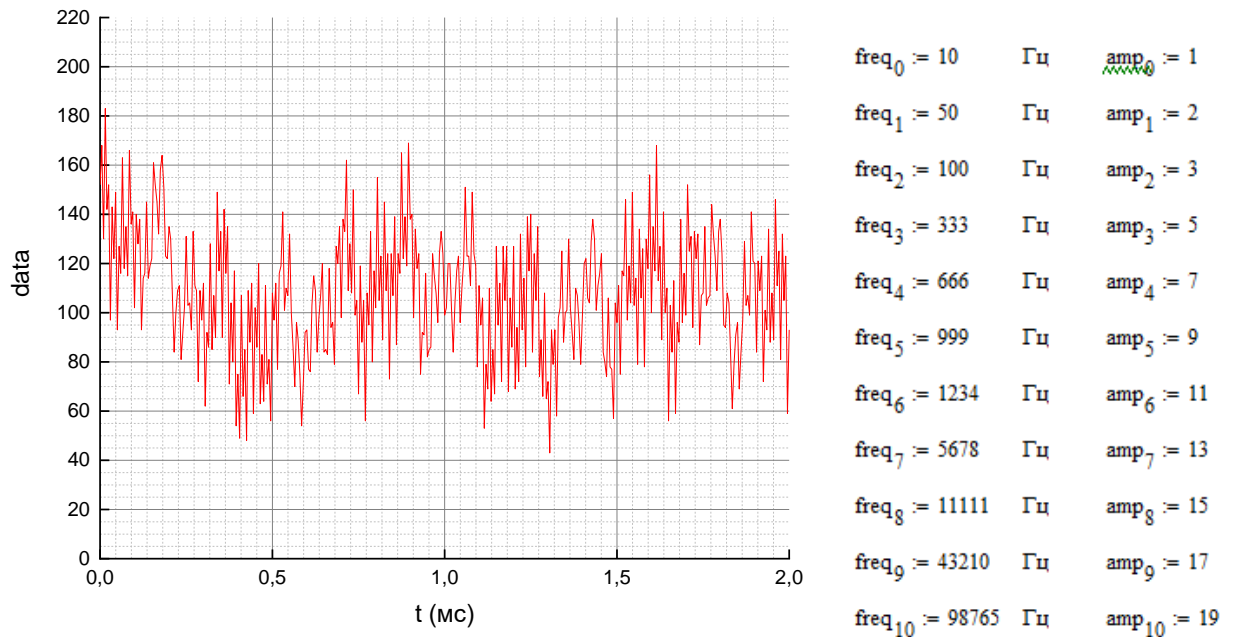


Рисунок 22 Сгенерированный сигнал для теста 1 и его параметры

Итоговый результат можно увидеть на рисунке 23. Как видно рассчитанные частоты и амплитуды соответствуют заданным значениям.

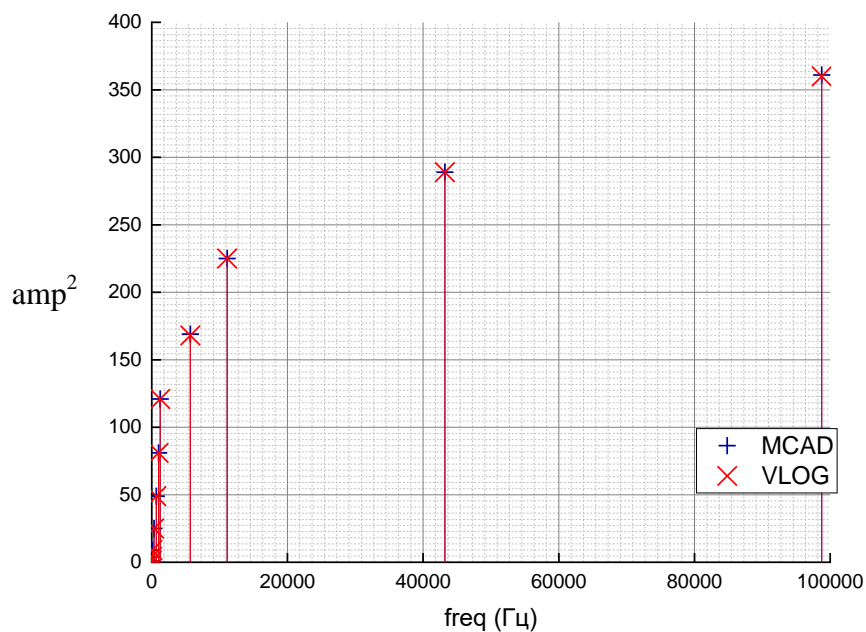


Рисунок 23 Результат расчёта Verilog модуля и MatchCad

Тест 2. На рисунках 24-25. Приведён тот же тест, но с другими параметрами входного сигнала.

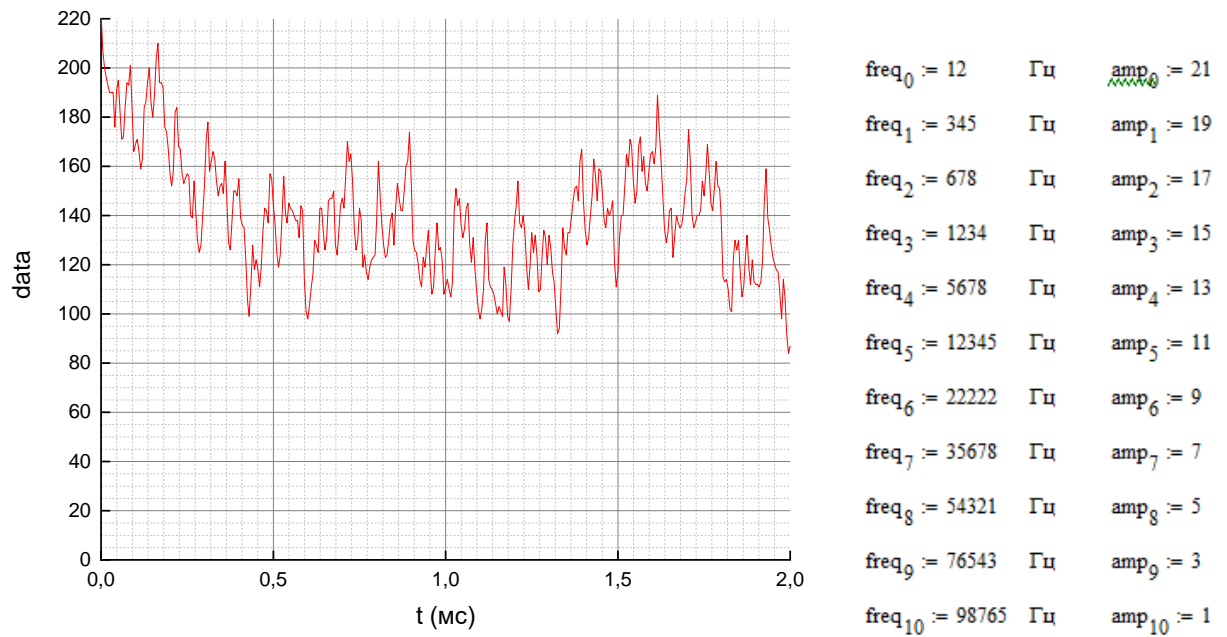


Рисунок 24 Сгенерированный сигнал для теста 2 и его параметры

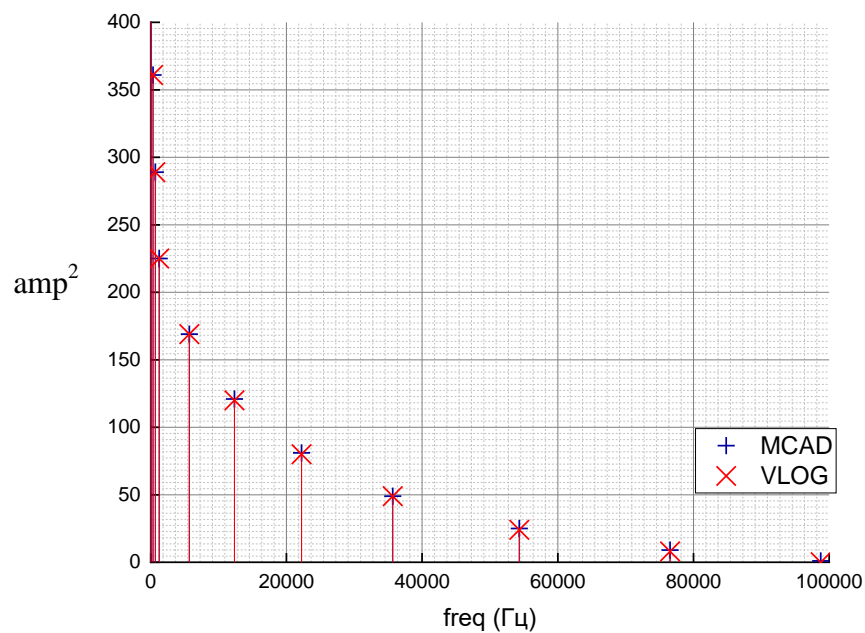


Рисунок 25 Результат расчёта Verilog модуля и MatchCad

На рисунке 26. приведён вывод информации теста в консоль при выполнении тестов. По нему можно увидеть последовательность работы модуля.

```
# [ 90158000] spi write: addr = 0x0000001c, data = 0x00000001, status = 0x00
# [ 180238000] spi write: addr = 0x0000001c, data = 0x00000000, status = 0x00
# [ 180238000] Write freq
# [ 270238000] spi write: addr = 0x10000000, data = 0x000001f4, status = 0x00
# [ 360238000] spi write: addr = 0x10000004, data = 0x000001f6, status = 0x00
# [ 450238000] spi write: addr = 0x10000008, data = 0x000001f8, status = 0x00
# [ 540238000] spi write: addr = 0x1000000c, data = 0x000001fa, status = 0x00
# [ 630238000] spi write: addr = 0x10000010, data = 0x000001fc, status = 0x00
# [ 720238000] spi write: addr = 0x10000014, data = 0x000001fe, status = 0x00
# [ 810238000] spi write: addr = 0x10000018, data = 0x00000200, status = 0x00
# [ 900238000] spi write: addr = 0x1000001c, data = 0x00000202, status = 0x00
# [ 990238000] spi write: addr = 0x10000020, data = 0x00000204, status = 0x00
# [1080238000] spi write: addr = 0x10000024, data = 0x00000206, status = 0x00
# [1170238000] spi write: addr = 0x10000028, data = 0x00000208, status = 0x00
# [1260238000] spi write: addr = 0x1000002c, data = 0x0000020a, status = 0x00
# [1350238000] spi write: addr = 0x0000000c, data = 0x00001388, status = 0x00
# [1440238000] spi write: addr = 0x00000010, data = 0x00002710, status = 0x00
# [1530238000] spi write: addr = 0x00000008, data = 0x00000000, status = 0x00
# [1530238000] Start Herzel
# [1620238000] spi write: addr = 0x00000014, data = 0x00000001, status = 0x00
# [1710238000] spi read : addr = 0x00000018, data = 0x00000003, status = 0x00
# [4210742000] Read result
# [4300742000] spi read : addr = 0x20000000, data = 0x01901d72, status = 0x00
# [4390742000] spi read : addr = 0x20000004, data = 0x014440bb, status = 0x00
# [4480742000] spi read : addr = 0x20000008, data = 0x01008d0e, status = 0x00
# [4570742000] spi read : addr = 0x2000000c, data = 0x00c47e61, status = 0x00
# [4660742000] spi read : addr = 0x20000010, data = 0x009004c0, status = 0x00
# [4750742000] spi read : addr = 0x20000014, data = 0x006418ce, status = 0x00
# [4840742000] spi read : addr = 0x20000018, data = 0x00510312, status = 0x00
# [4930742000] spi read : addr = 0x2000001c, data = 0x0030f4eb, status = 0x00
# [5020742000] spi read : addr = 0x20000020, data = 0x0019235c, status = 0x00
# [5110742000] spi read : addr = 0x20000024, data = 0x00090d51, status = 0x00
# [5200742000] spi read : addr = 0x20000028, data = 0x00090d91, status = 0x00
# [5290742000] spi read : addr = 0x2000002c, data = 0x0008fb5b, status = 0x00
# [5290742000] Start check
# [5295742000] TEST SUCCESS
# Result:
# 0: 500. MCAD - 01900000, VLOG - 01901d72
# 1: 502. MCAD - 01440000, VLOG - 014440bb
# 2: 504. MCAD - 01010000, VLOG - 01008d0e
# 3: 506. MCAD - 00c40000, VLOG - 00c47e61
# 4: 508. MCAD - 00900000, VLOG - 009004c0
# 5: 510. MCAD - 00640000, VLOG - 006418ce
# 6: 512. MCAD - 00510000, VLOG - 00510312
# 7: 514. MCAD - 00310000, VLOG - 0030f4eb
# 8: 516. MCAD - 00190000, VLOG - 0019235c
# 9: 518. MCAD - 00090000, VLOG - 00090d51
# 10: 520. MCAD - 00090000, VLOG - 00090d91
# 11: 522. MCAD - 00090000, VLOG - 0008fb5b
# ** Note: $stop : D:/Desktop/Study_now/SRW/GoertzelAlgorithm/src/sim/FourierTransform_tb.sv(256)
# Time: 5295742 ns Iteration: 0 Instance: /FourierTransform_tb
```

Рисунок 26 Вывод в консоль

На рисунке 27. представлена осциллограмма модуля FourierTransform. На ней можно увидеть SPI транзакции чтения и записи регистров модуля, и следующие за ними AXI-lite транзакции. Начало работы модуля Cordic, его завершение. А также работу модуля Herzel, принимающего выборки с sample_p и чтение его результатов после завершения расчётов.

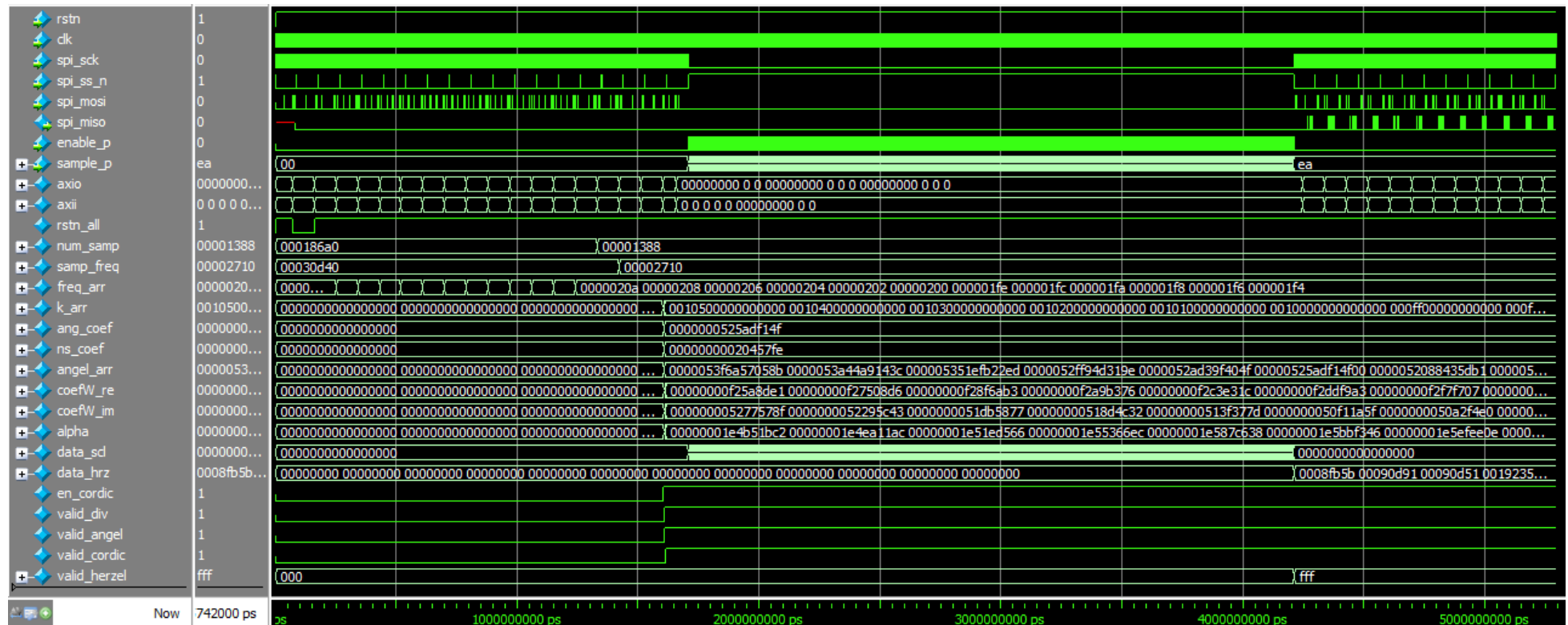


Рисунок 27 Осциллограмма модуля FourierTransform

На рисунке 28 показана AXI-lite транзакция записи данных в регистры. На ней можно увидеть соответствующие запросы от ведущего и следующие за ними ответы от ведомого и наоборот.

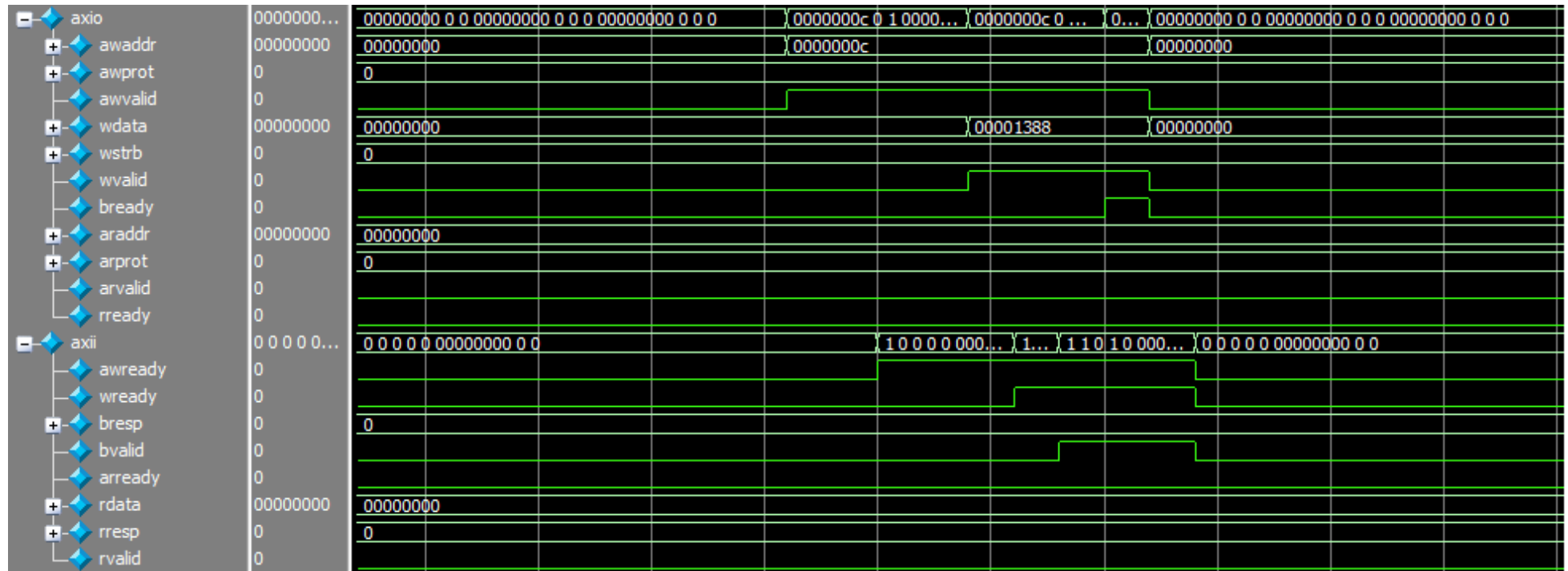


Рисунок 28 Осциллограмма AXI-lite транзакции

На рисунке 29. показана работа модуля Cordic. Последовательно происходит вычисление синусов и косинусов входящих углов. На схему при этом подавались частоты от 10, 500, 1000...5000 Гц при частоте выборки 10000 Гц и числа выборок 5000 шт.



Рисунок 29 Осциллограмма модуля Cordic

На рисунке 30. показана работа модуля Herzel. Можно увидеть работу умножителя, и машины состояний. Данные при en записываются в временные регистры en_r и data_r и ждут пока завершит свою работу умножитель. После завершения умножения, вычисляются vm1, vm2 и снова идёт умножение. Также идёт работа и в остальных состояниях пока не будет получен окончательный результат.

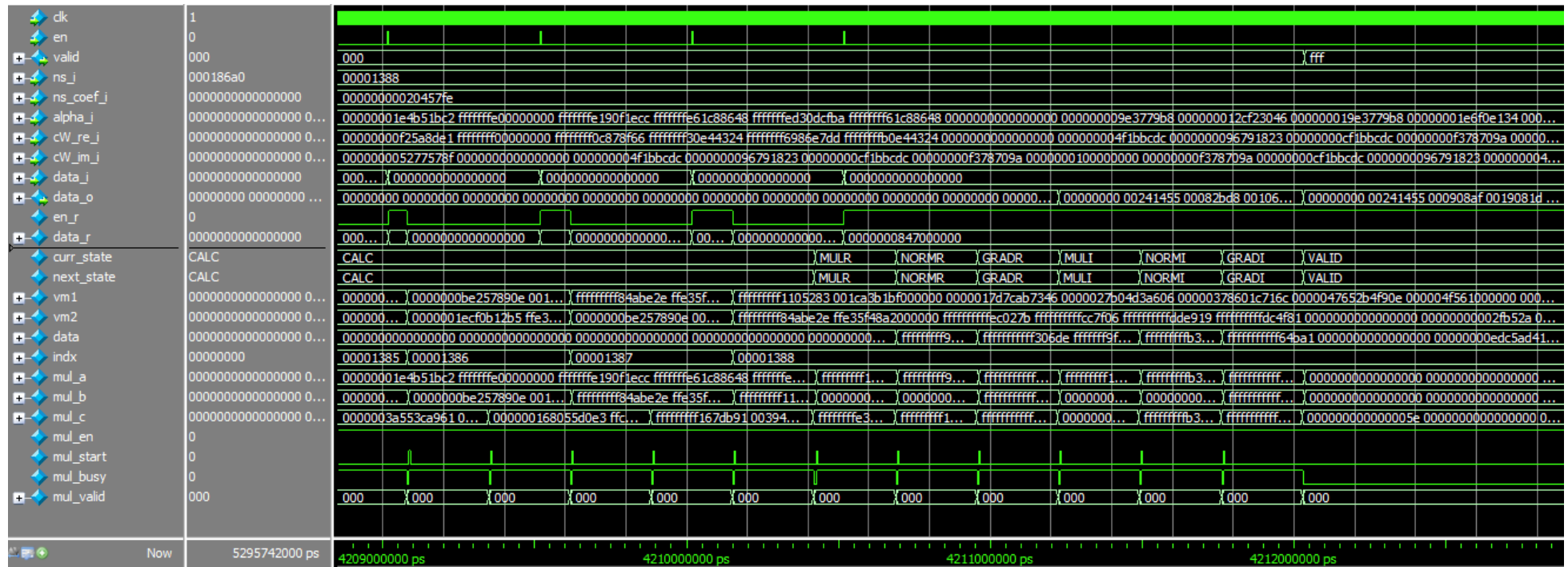


Рисунок 30 Осциллограмма модуля Herzel

11. Приложение 3

11.1. АЦП

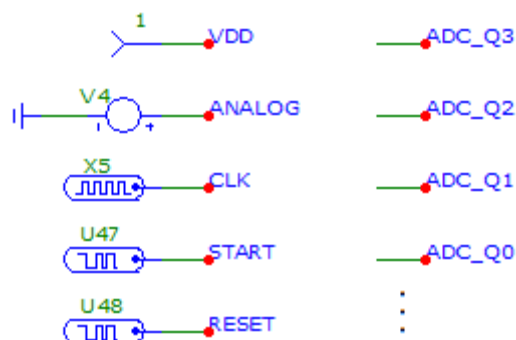


Рисунок 31 Интерфейс АЦП модуля и генераторы тактовых импульсов

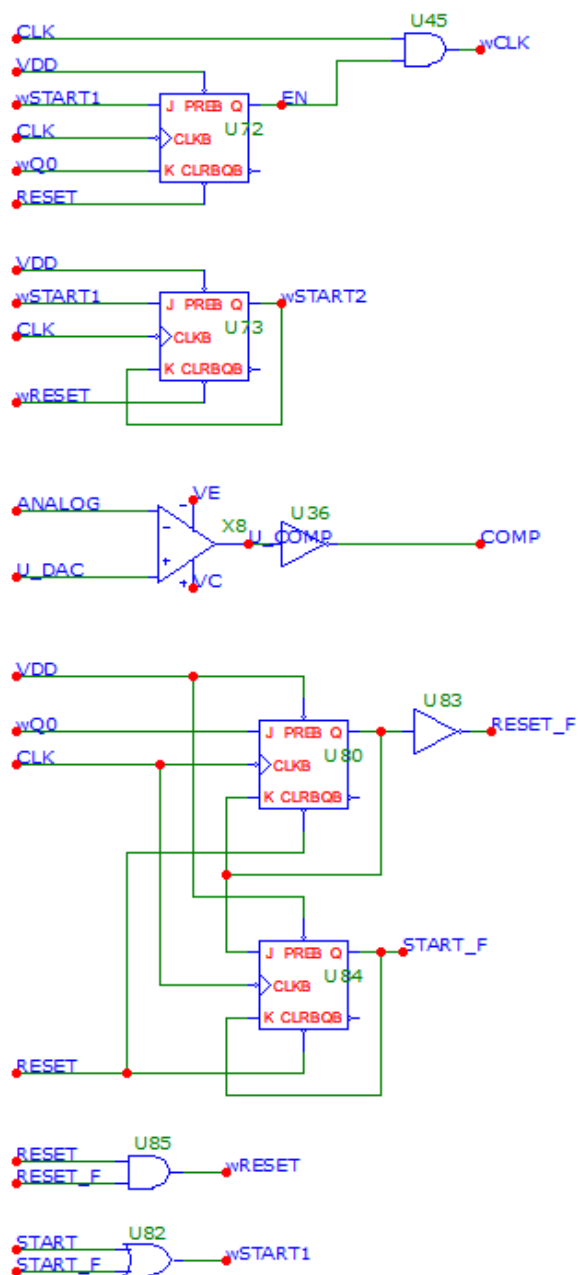


Рисунок 32 Схема управления

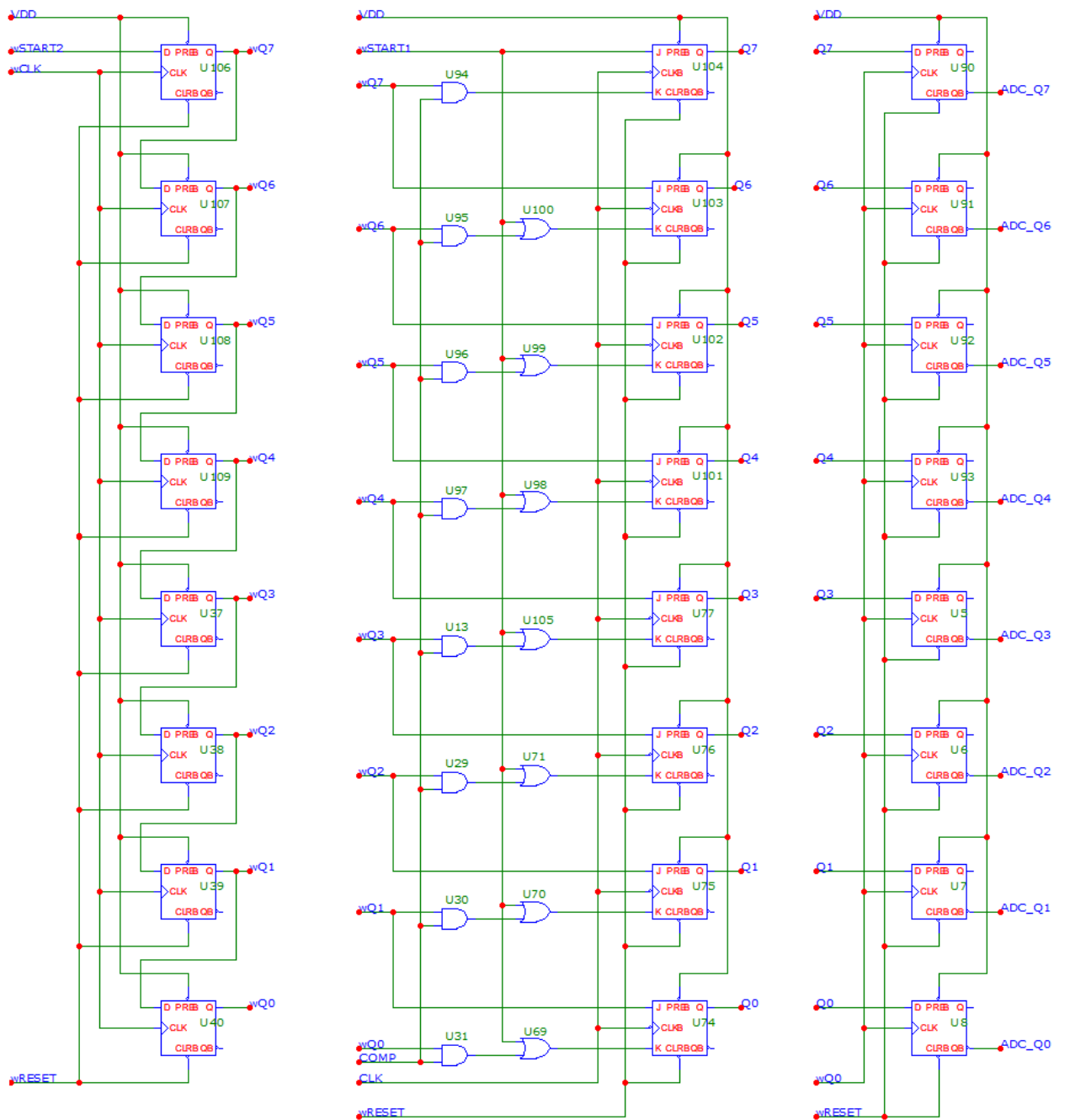


Рисунок 33 Слева на право: сдвиговый регистр, схема изменения кода, выходной регистр.
В целом представляют регистр последовательного приближения и выходной регистр.

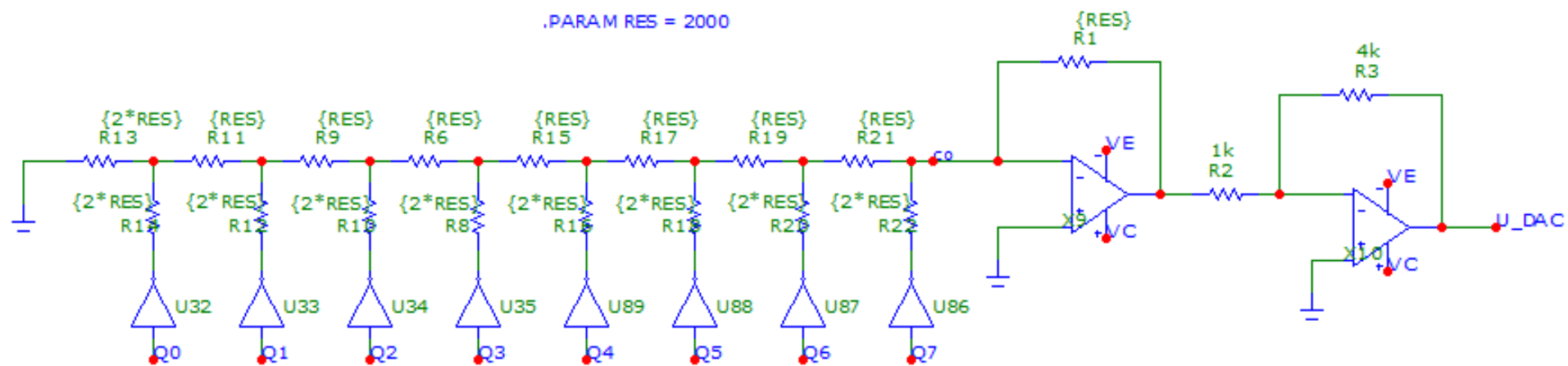


Рисунок 34 Схема ЦАП

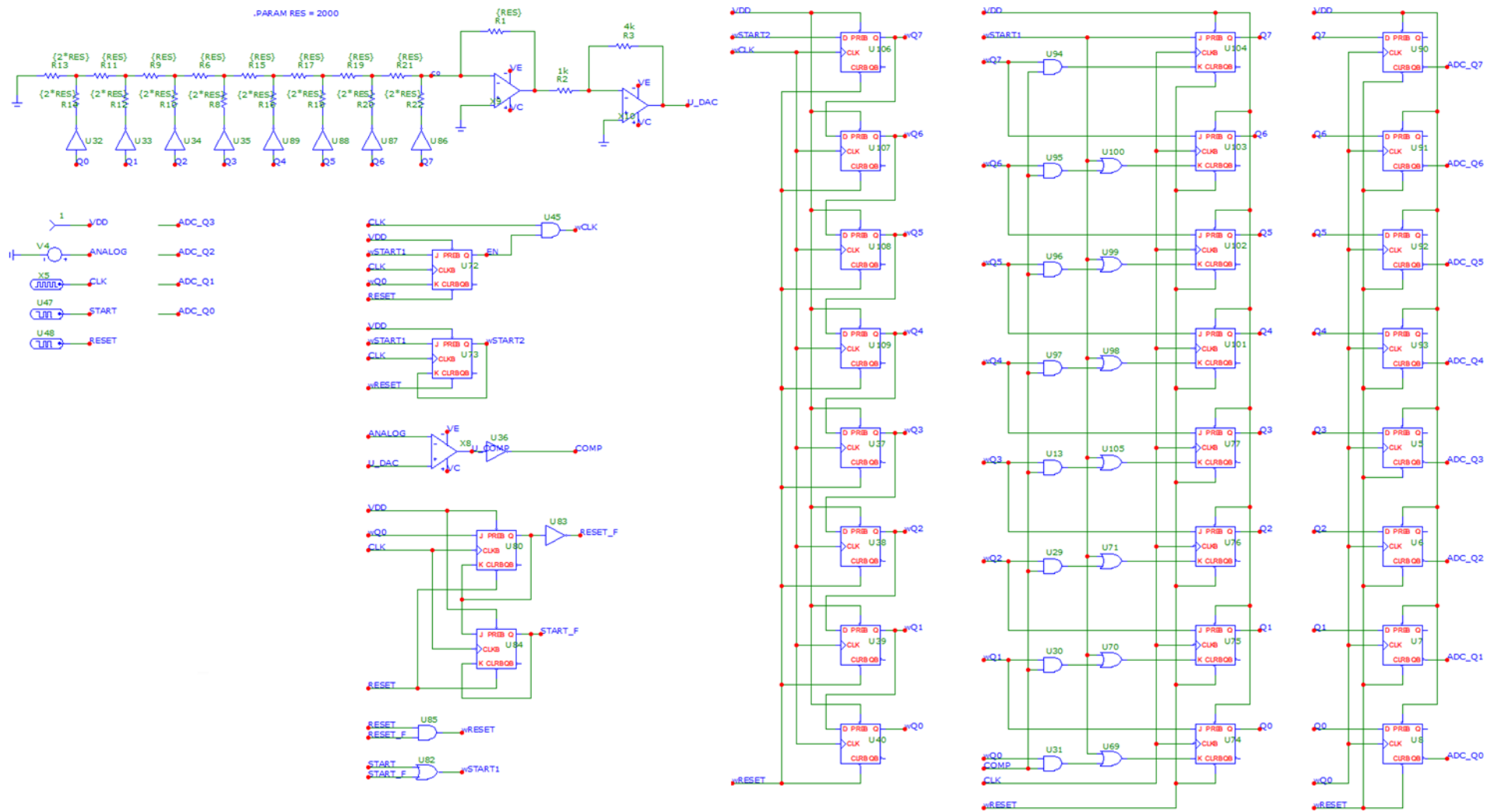


Рисунок 35 Общя схема АЦП последовательного приближения

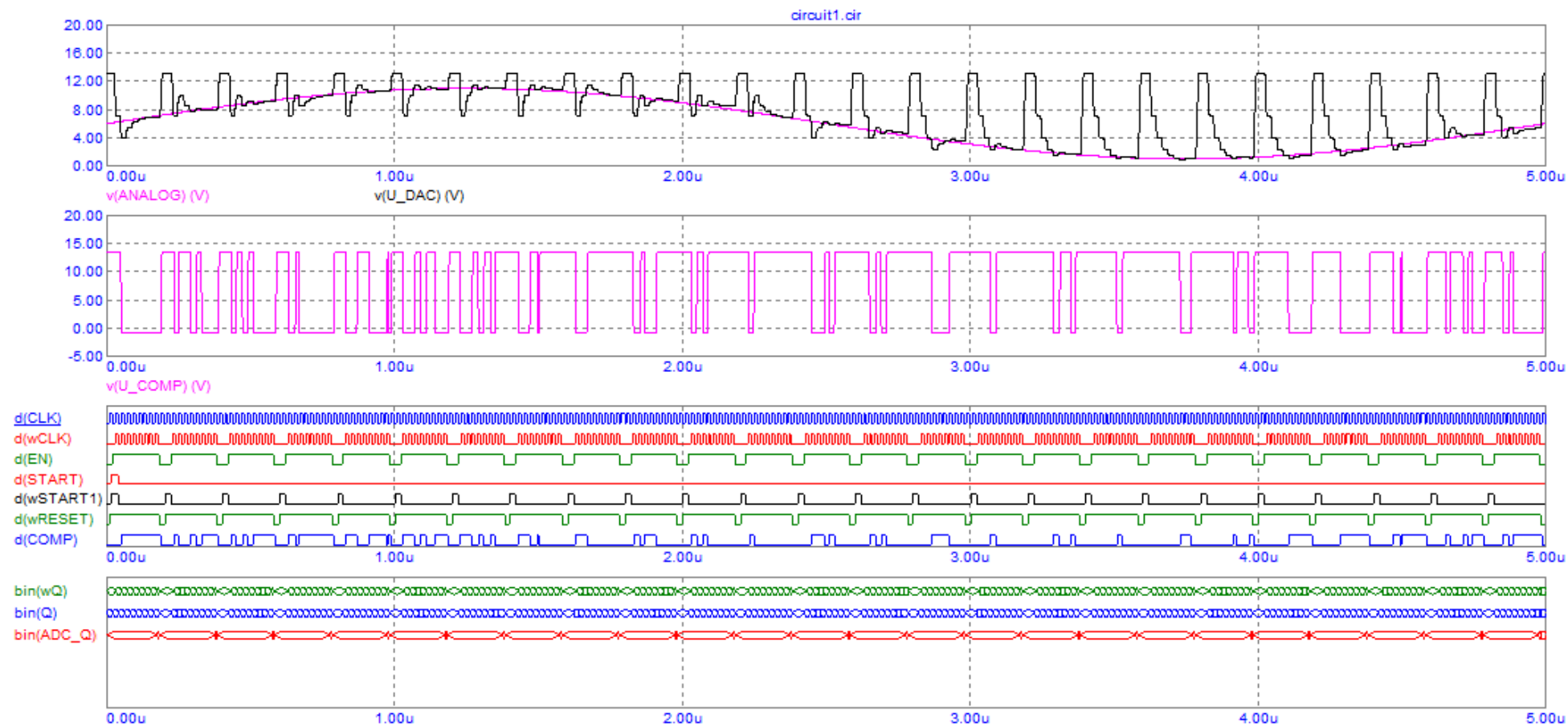


Рисунок 36 Осцилограмма

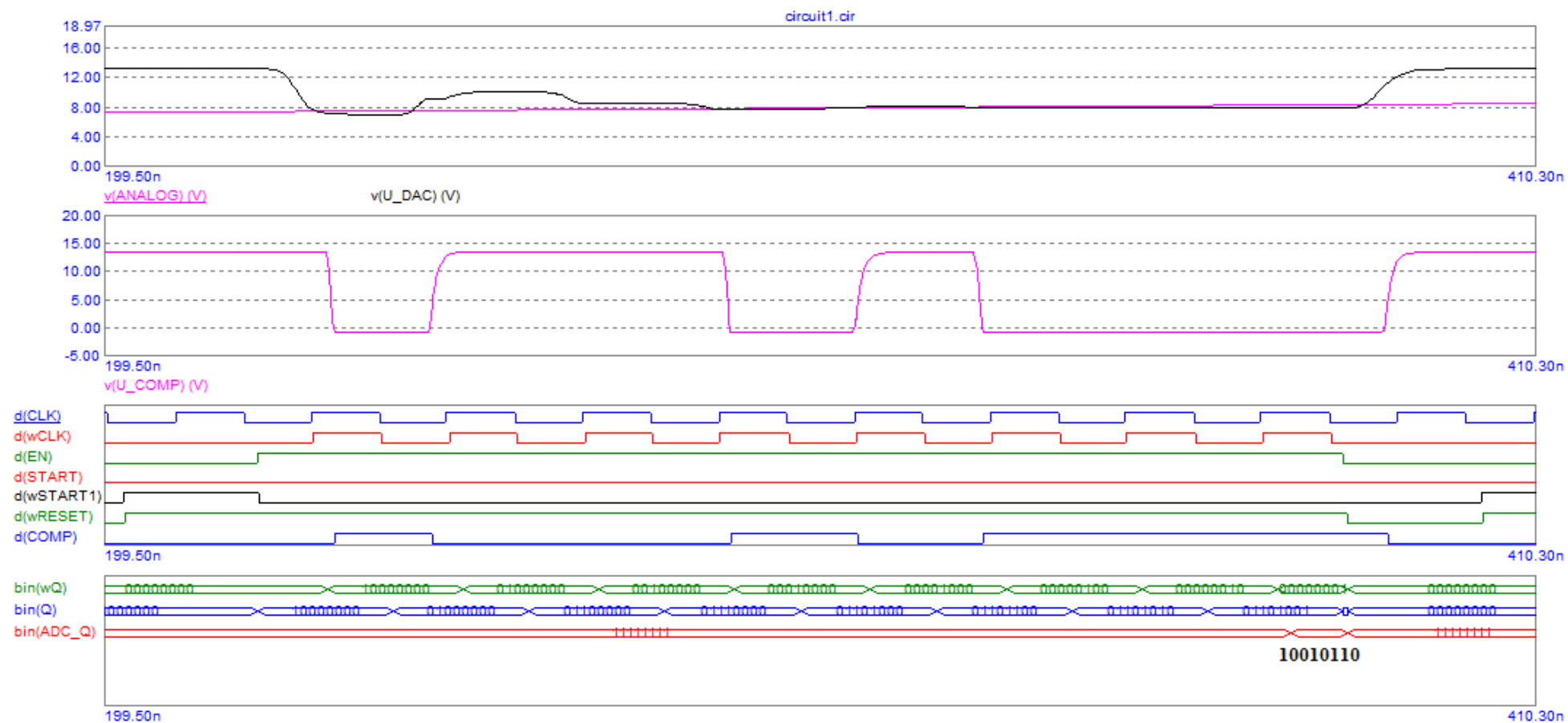


Рисунок 37 Осцилограмма

11.2. LVDS

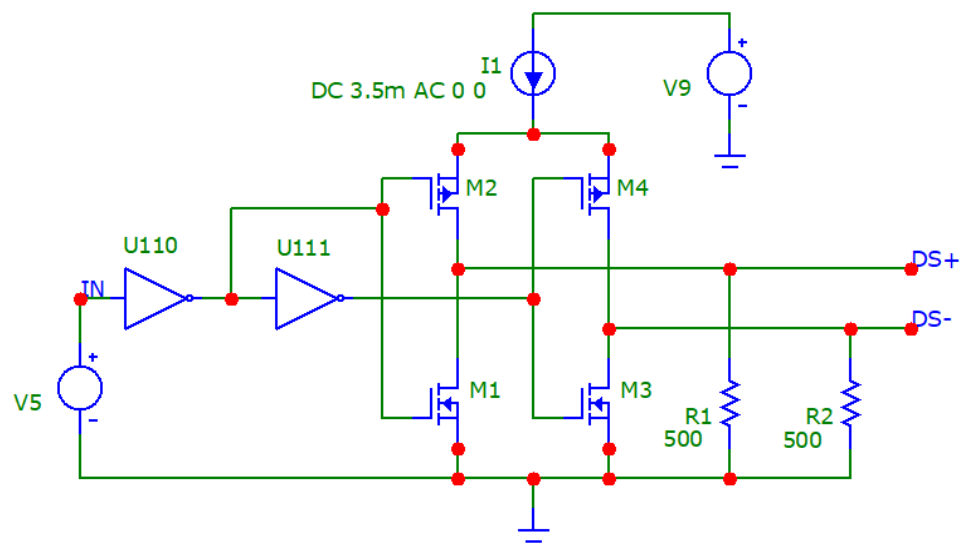


Рисунок 38 Простейшая схема LVDS драйвера

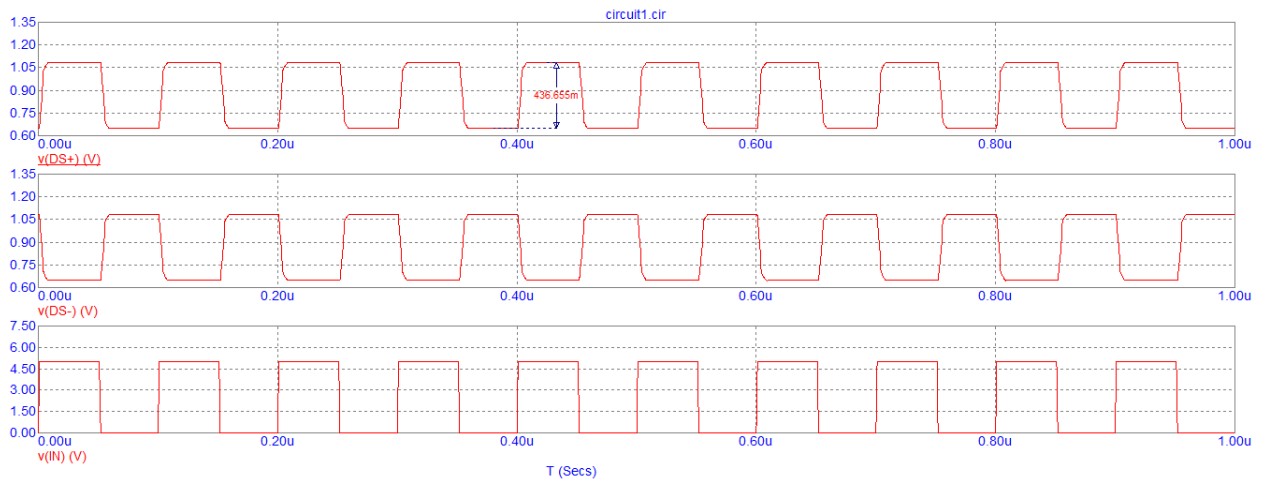


Рисунок 39 Моделирование драйвера LVDS интерфейса

11.3. Модуль преобразования Фурье

11.3.1. FourierTransform

```
import axi_pkg::*;

module FourierTransform #(
    parameter NF = 11 , // NUM_FREQ
    parameter DW = 32 //
) (
    // CLK&RST
    input rstn ,
    input clk ,
    // SPI
    input spi_sck ,
    input spi_ss_n,
    input spi_mosi,
    output spi_miso,
    // CTRL
    input enable_p,
    input enable_n,
    // // DATA
    input [7:0] sample_p,
    input [7:0] sample_n
);

logic clkd;

logic enable ;
logic [7:0] sample ;
logic enable_syn;
logic [7:0] sample_syn;
logic rstn_syn ;

logic reset_all_r;
logic reset_h_r ;
logic rstn_all ;
logic rstn_h ;

logic [31:0] num_samp ;
logic [31:0] samp_freq;
logic [NF-1:0][31:0] freq_arr ;
logic [NF-1:0][63:0] k_arr ;
```

```

logic          [63:0] ang_coef ;
logic          [63:0] ns_coef  ;
logic signed [NF-1:0][63:0] angel_arr;
logic signed [NF-1:0][63:0] coefw_re ;
logic signed [NF-1:0][63:0] coefw_im ;
logic signed [NF-1:0][63:0] alpha    ;
logic signed          [63:0] data_scl ;
logic signed [NF-1:0][31:0] data_hrz ;

logic          en_cordic  ;
logic          en_scl     ;
logic          valid_div  ;
logic          valid_angel ;
logic          valid_cordic;
logic          valid_scl  ;
logic [NF-1:0] valid_herzel;

axi_lite_mosi axio;
axi_lite_miso axii;

assign rstn_all = rstn_syn && (~reset_all_r);
assign rstn_h   = rstn_syn && (~reset_all_r) && (~reset_h_r);

assign clkd = clk;

    assign enable = enable_p;
    genvar gvar_buf;
    generate
        for (gvar_buf = 0; gvar_buf < 8; gvar_buf = gvar_buf + 1) begin :
assign_sample
            assign sample[gvar_buf] = sample_p[gvar_buf];
        end
    endgenerate

resync_nrst u_resync_nrst(
    .clk    (clkd    ),
    .rstn_i(rstn    ),
    .rstn_o(rstn_syn)
);

resync_data #(
    .DW(1)
) u_resync_enable (
    .rstn  (rstn_syn ),
    .clk   (clkd     ),

```

```

        .data_i(enable    ),
        .data_o(enable_syn)
    );

    resync_data #(
        .DW(8)
    ) u_resync_sample (
        .rstn  (rstn_syn  ),
        .clk   (clkd      ),
        .data_i(sample    ),
        .data_o(sample_syn)
    );

    spi2axi_wrap u_spi2axi_wrap (
        .axi_rstn_i (rstn_syn),
        .axi_clk_i  (clkd    ),
        .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    )
    );

    HerzlRegs #(
        .NF(NF)
    ) u_HerzlRegs (
        .rstn      (rstn_syn  ),
        .clk       (clkd      ),
        .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  ),
        .num_samp_o    (num_samp  ),
        .samp_freq_o   (samp_freq ),
        .reset_all_o   (reset_all_r ),
        .reset_h_o     (reset_h_r  ),
        .axio_i        (axio     ),
        .axii_o        (axii     )
    );

    div_all #(
        .NF(NF)

```

```

) u_div_all(
    .rstn      (rstn_all ),
    .clk       (clkd      ),
    .en        (en_cordic),
    .valid     (valid_div),
    .num_samp_i (num_samp ),
    .samp_freq_i(samp_freq),
    .freq_i     (freq_arr ),
    .k_arr_o    (k_arr     ),
    .ang_coef_o (ang_coef ),
    .ns_coef_o  (ns_coef  )
);

```

```

Angel #(
    .NF(NF)
) u_Angel (
    .rstn      (rstn_all  ),
    .clk       (clkd      ),
    .en        (valid_div ),
    .valid     (valid_angel),
    .k_arr_i    (k_arr     ),
    .ang_coef_i(ang_coef  ),
    .angel_o    (angel_arr )
);

```

```

Cordic #(
    .NF(NF)
) u_Cordic (
    .rstn (rstn_all  ),
    .clk  (clkd      ),
    .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_syn;

```

```

DataScale u_DataScale (
    .rstn  (rstn_all  ),
    .clk   (clkd      ),
    .enable(en_scl    ),
    .valid (valid_scl ),

```

```

        .data_i(sample_syn),
        .data_o(data_scl )
    );

    logic [DW-1:0] tns_coef ;
    logic signed [NF-1:0][DW-1:0] talpha ;
    logic signed [NF-1:0][DW-1:0] tcoefw_re ;
    logic signed [NF-1:0][DW-1:0] tcoefw_im ;
    logic signed [DW-1:0] tdata_scl ;

    assign tns_coef = ns_coef [64/2+DW/2-1:64/2-DW/2];
    assign tdata_scl = data_scl[64/2+DW/2-1:64/2-DW/2];

    genvar gvar;
    generate
        for (gvar=0;gvar<NF;gvar=gvar+1) begin : bits_slice
            assign talpha [gvar] = alpha [gvar][64/2+DW/2-1:64/2-DW/2];
            assign tcoefw_re[gvar] = coefw_re[gvar][64/2+DW/2-1:64/2-DW/2];
            assign tcoefw_im[gvar] = coefw_im[gvar][64/2+DW/2-1:64/2-DW/2];
        end
    endgenerate

    Herzal #(
        .NF(NF),
        .DW(DW)
    ) u_Herzal (
        .rstn (rstn_h ),
        .clk (clkd ),
        .en (valid_scl ),
        .valid (valid_herzel),
        .ns_i (num_samp ),
        .ns_coef_i(tns_coef ),
        .alpha_i (talpha ),
        .cW_re_i (tcoefw_re ),
        .cW_im_i (tcoefw_im ),
        .data_i (tdata_scl ),
        .data_o (data_hrz )
    );

endmodule

```

11.3.2. IBUFDS

```
IBUFDS IBUFDS_inst
(
    .I (enable_p),
    .IB(enable_n),
    .O (enable )
);
```

11.3.3. resync_nrst

```
module resync_nrst #(
    parameter NUM_STAGE = 3
) (
    input  clk  ,
    input  rstn_i,
    output rstn_o
);

logic [NUM_STAGE-1:0] nrst_stg;

always_ff @(posedge clk, negedge rstn_i) begin
    if (!rstn_i) begin
        nrst_stg <= 0;
    end
    else begin
        nrst_stg <= {nrst_stg[NUM_STAGE-2:0], 1'b1};
    end
end

assign rstn_o = nrst_stg[NUM_STAGE-1];

endmodule
```

11.3.1. resync_data

```
module resync_data #(
    parameter NUM_STAGE = 3,
    parameter DW         = 1
) (
    input          rstn  ,
    input          clk   ,
    input  [DW-1:0] data_i,
    output [DW-1:0] data_o
);

localparam STG_DW = NUM_STAGE * DW;

logic [STG_DW-1:0] data_stg;

always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        data_stg <= 0;
    end
    else begin
        data_stg <= {data_stg[STG_DW-DW-1:0], data_i};
    end
end

assign data_o = data_stg[STG_DW-1:STG_DW-DW];

endmodule
```


11.3.2. spi2axi

```
module spi2axi (  
    // SPI  
    input logic      spi_sck      ,  
    input logic      spi_ss       ,  
    input logic      spi_mosi     ,  
    output logic     spi_miso     ,  
    // AXI  
    input logic      axi_clk      ,  
    input logic      axi_rstn     ,  
    output logic     axi_awvalid  ,  
    output logic [31:0] axi_awaddr ,  
    output logic [2 :0] axi_awprot , //  
    input logic      axi_awready  ,  
    output logic     axi_wvalid   ,  
    output logic [31:0] axi_wdata  ,  
    output logic [3 :0] axi_wstrb  , //  
    input logic      axi_wready   ,  
    input logic      axi_bvalid   ,  
    input logic [1 :0] axi_bresp   ,  
    output logic     axi_bready   ,  
    output logic     axi_arvalid  ,  
    output logic [31:0] axi_araddr ,  
    output logic [2 :0] axi_arprot , //  
    input logic      axi_arready  ,  
    input logic      axi_rvalid   ,  
    input logic [31:0] axi_rdata   ,  
    input logic [1 :0] axi_rresp   ,  
    output logic     axi_rready  
);
```

```
typedef enum {  
    SPI_IDLE ,  
    SPI_CMD  ,  
    SPI_ADDR ,  
    SPI_WDATA,  
    SPI_RDATA,  
    SPI_DUMM ,  
    SPI_STAT  
} spi_state;
```

```
typedef enum {
```

```

    AXI_IDLE ,
    AXI_CMD  ,
    AXI_RADDR,
    AXI_RDATA,
    AXI_WADDR,
    AXI_WDATA,
    AXI_WRESP
} axi_state;

spi_state spi_cstate;
spi_state spi_nstate;
axi_state axi_cstate;
axi_state axi_nstate;

logic      spi_sck_syn;
logic      spi_sck_old;
logic      spi_sck_re ;
logic      spi_sck_fe ;
logic [7:0] spi_cntr  ;

logic [7:0] spi_cmd   ;
logic [31:0] spi_addr ;
logic [31:0] spi_wdata ;
logic [31:0] spi_rdata ; //
logic [7:0] spi_dumm  ;
logic [7:0] spi_stat  ; //

logic      spi_ss_syn ;
logic      spi_ss_old ;
logic      spi_ss_fe  ;

resync_data u_resync_sck (
    .rstn (axi_rstn ),
    .clk  (axi_clk  ),
    .data_i(spi_sck  ),
    .data_o(spi_sck_syn)
);

resync_data u_resync_ss (
    .rstn (axi_rstn ),
    .clk  (axi_clk  ),
    .data_i(spi_ss   ),
    .data_o(spi_ss_syn)

```

```
);
```

```
always_ff @(negedge axi_clk, negedge axi_rstn) begin : spi_current_state
    if (!axi_rstn) begin
        spi_cstate <= SPI_IDLE;
    end
    else begin
        spi_cstate <= spi_nstate;
    end
end
```

```
always_ff @(posedge axi_clk, negedge axi_rstn) begin : spi_next_state
    if (!axi_rstn) begin
        spi_miso      <= 0;
        spi_sck_old <= 0;
        spi_sck_re  <= 0;
        spi_sck_fe  <= 0;
        spi_ss_old  <= 0;
        spi_ss_fe   <= 0;
        spi_nstate <= SPI_IDLE;
    end
    else begin
```

```
        spi_sck_old <= spi_sck_syn;
        spi_sck_re  <= ((spi_sck_old == 1'b0) && (spi_sck_syn == 1'b1));
        spi_sck_fe  <= ((spi_sck_old == 1'b1) && (spi_sck_syn == 1'b0));
```

```
        spi_ss_old <= spi_ss_syn;
        spi_ss_fe   <= ((spi_ss_old == 1'b1) && (spi_ss_syn == 1'b0));
```

```
        case (spi_cstate)
            SPI_IDLE : begin
                spi_cntr  <= 0;
                spi_cmd   <= 0;
                spi_addr  <= 0;
                spi_wdata <= 0;
                spi_dumm  <= 0;
                if (spi_ss_fe)
                    spi_nstate <= SPI_CMD;
                else
                    spi_nstate <= SPI_IDLE;
            end
            SPI_CMD : begin
                if (spi_cntr < 8) begin
```

```

        if (spi_sck_re) begin
            spi_cmd    <= {spi_cmd[6:0], spi_mosi};
            spi_cntr    <= spi_cntr + 1;
            spi_nstate <= SPI_CMD;
        end
    end
else begin
    spi_cntr    <= 0;
    spi_nstate <= SPI_ADDR;
end
end
SPI_ADDR : begin
    if (spi_cntr < 32) begin
        if (spi_sck_re) begin
            spi_addr    <= {spi_addr[30:0], spi_mosi};
            spi_cntr    <= spi_cntr + 1;
            spi_nstate <= SPI_ADDR;
        end
    end
else begin
    spi_cntr    <= 0;
    if (spi_cmd == 1'b1)
        spi_nstate <= SPI_DUMM;
    else
        spi_nstate <= SPI_WDATA;
    end
end
SPI_WDATA : begin
    if (spi_cntr < 32) begin
        if (spi_sck_re) begin
            spi_wdata    <= {spi_wdata[30:0], spi_mosi};
            spi_cntr    <= spi_cntr + 1;
            spi_nstate <= SPI_WDATA;
        end
    end
else begin
    spi_cntr    <= 0;
    spi_nstate <= SPI_DUMM;
end
end
SPI_RDATA : begin
    if (spi_cntr < 32) begin
        if (spi_sck_fe) begin
            spi_miso    <= spi_rdata[31-spi_cntr];
            spi_cntr    <= spi_cntr + 1;

```

```

        spi_nstate <= SPI_RDATA;
    end
    else if (spi_sck_re) begin
        spi_miso    <= 0;
    end
end
else begin
    spi_cntr    <= 0;
    spi_nstate <= SPI_STAT;
end
end
SPI_DUMM : begin
    if (spi_cntr < 8) begin
        if (spi_sck_re) begin
            spi_dumm    <= {spi_dumm[6:0], 1'b0};
            spi_cntr    <= spi_cntr + 1;
            spi_nstate <= SPI_DUMM;
        end
    end
    else begin
        spi_cntr    <= 0;
        if (spi_cmd == 1'b1)
            spi_nstate <= SPI_RDATA;
        else
            spi_nstate <= SPI_STAT;
        end
    end
end
SPI_STAT : begin
    if (spi_cntr < 8) begin
        if (spi_sck_re) begin
            spi_miso    <= spi_stat[7-spi_cntr];
            spi_cntr    <= spi_cntr + 1;
            spi_nstate <= SPI_STAT;
        end
    end
    else begin
        spi_cntr    <= 0;
        spi_nstate <= SPI_IDLE;
    end
end
default: spi_nstate <= spi_cstate;
endcase
end
end

```

```

always_ff @(negedge axi_clk, negedge axi_rstn) begin : axi_current_state
    if (!axi_rstn) begin
        axi_cstate <= AXI_IDLE;
    end
    else begin
        axi_cstate <= axi_nstate;
    end
end

always_ff @(posedge axi_clk, negedge axi_rstn) begin : axi_next_state
    if (!axi_rstn) begin
        axi_awvalid <= 0;
        axi_awaddr  <= 0;
        axi_awprot  <= 0;
        axi_wvalid  <= 0;
        axi_wdata   <= 0;
        axi_wstrb   <= 0;
        axi_bready  <= 0;
        axi_araddr  <= 0;
        axi_arprot  <= 0;
        axi_arvalid <= 0;
        axi_rready  <= 0;
        spi_rdata   <= 0;
        spi_stat    <= 0;
        axi_nstate <= AXI_IDLE;
    end
    else begin
        case (axi_cstate)
            AXI_IDLE : begin
                axi_awvalid <= 0;
                axi_awaddr  <= 0;
                axi_awprot  <= 0;
                axi_wvalid  <= 0;
                axi_wdata   <= 0;
                axi_wstrb   <= 0;
                axi_bready  <= 0;
                axi_araddr  <= 0;
                axi_arprot  <= 0;
                axi_arvalid <= 0;
                axi_rready  <= 0;
                if (spi_cstate == SPI_IDLE)
                    axi_nstate <= AXI_CMD;
            end
        end
    end
end

```

```

        else
            axi_nstate <= AXI_IDLE;
        end
    AXI_CMD : begin
        spi_rdata <= 0;
        spi_stat  <= 0;
        if (spi_cstate > SPI_CMD) begin
            if (spi_cmd == 1)
                axi_nstate <= AXI_RADDR;
            else
                axi_nstate <= AXI_WADDR;
            end
        end
    end
    AXI_RADDR : begin
        if (spi_cstate > SPI_ADDR) begin
            axi_araddr  <= spi_addr;
            axi_arvalid <= 1;
            if (axi_arready == 1)
                axi_nstate <= AXI_RDATA;
            else
                axi_nstate <= AXI_RADDR;
            end
        end
    end
    AXI_RDATA : begin
        if (axi_rvalid == 1) begin
            spi_rdata  <= axi_rdata;
            spi_stat   <= axi_rresp;
            axi_rready <= 1;
            axi_nstate <= AXI_IDLE;
        end
        else
            axi_nstate <= AXI_RDATA;
        end
    end
    AXI_WADDR : begin
        if (spi_cstate > SPI_WDATA) begin
            axi_awaddr  <= spi_addr;
            axi_awvalid <= 1;
            if (axi_awready == 1)
                axi_nstate <= AXI_WDATA;
            else
                axi_nstate <= AXI_WADDR;
            end
        end
    end
    AXI_WDATA : begin
        if (spi_cstate > SPI_WDATA) begin

```

```

        axi_wdata <= spi_wdata;
        axi_wvalid <= 1;
        if (axi_wready == 1)
            axi_nstate <= AXI_WRESP;
        else
            axi_nstate <= AXI_WDATA;
        end
    end
    AXI_WRESP : begin
        if (axi_bvalid == 1) begin
            spi_stat <= axi_bresp;
            axi_bready <= 1;
            axi_nstate <= AXI_IDLE;
        end
        else
            axi_nstate <= AXI_WRESP;
        end
        default: axi_nstate <= axi_cstate;
    endcase
end
end

endmodule

```


11.3.3. HerzelRegs

```
import axi_pkg::*;

module HerzelRegs #(
    parameter NF = 11
) (
    // 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    ,
    output logic                        [31:0] num_samp_o    ,
    output logic                        [31:0] samp_freq_o    ,
    output logic                        reset_all_o         ,
    output logic                        reset_h_o          ,
    // AXI
    input  axi_lite_mosi                    axio_i          ,
    output axi_lite_miso                    axii_o
);

localparam FREQ_BA = 32'h1000_0000;
localparam DATA_BA = 32'h2000_0000;

// addres map of regs and default value
logic [31:0] version    ; // RW 0x0000_0000
logic [31:0] debug      ; // RW 0x0000_0004
logic [31:0] num_samp    ; // RW 0x0000_000C
logic [31:0] samp_freq   ; // RW 0x0000_0010
logic [31:0] en_cordic   ; // RW 0x0000_0014
logic [31:0] status      ; // R  0x0000_0018
logic [31:0] reset_all   ; // RW 0x0000_001C
logic [31:0] reset_h     ; // RW 0x0000_0020

logic [NF-1:0][31:0] freq ; // RW 0x1000_0000
logic [NF-1:0][31:0] data ; // R  0x2000_0000

always_comb begin
    num_samp_o      = num_samp            ;
    samp_freq_o     = samp_freq           ;
```

```

en_cordic_o    = en_cordic[0]      ;
status         = 0                  ;
status[0]      = valid_angel_i     ;
status[1]      = valid_cordic_i    ;
status[2]      = &valid_herzel_i   ;
reset_all_o    = reset_all[0]      ;
reset_h_o      = reset_h[0]        ;

for (int i=0; i<NF; i=i+1) begin
    freq_arr_o[i] = freq[i];
end
end

typedef enum {
    IDLE ,
    RADDR,
    RDATA,
    WADDR,
    WDATA,
    WRESP
} state;

state curr_state;
state next_state;

always_ff @(negedge clk, negedge rstn) begin
    if (!rstn) begin
        curr_state <= IDLE;
    end
    else begin
        curr_state <= next_state;
    end
end

always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        version    <= 32'h2904_2023;
        debug      <= 32'hF0F0_F0F0;
        num_samp   <= 32'h0001_86A0;
        samp_freq  <= 32'h0003_0D40;
        en_cordic  <= 32'h0000_0000;
        reset_all  <= 32'h0000_0000;
        reset_h    <= 32'h0000_0000;
        freq       <= 32'h0000_0000;
        data       <= 32'h0000_0000;
    end
end

```

```

next_state <= IDLE;
end else begin
  case (curr_state)
    IDLE : begin
      if (status[2]) begin
        for (int i=0; i<NF; i=i+1) begin
          data[i] = data_arr_i[i];
        end
      end
      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;
      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;
      if (axio_i.araddr[31:28] == 0) begin
        case (axio_i.araddr)
          32'h0000_0000: axii_o.rdata <= version ;
          32'h0000_0004: axii_o.rdata <= debug ;
          32'h0000_000C: axii_o.rdata <= num_samp ;
          32'h0000_0010: axii_o.rdata <= samp_freq;
          32'h0000_0014: axii_o.rdata <= en_cordic;
          32'h0000_0018: axii_o.rdata <= status ;
          32'h0000_001C: axii_o.rdata <= reset_all;
          32'h0000_0020: axii_o.rdata <= reset_h ;
          default: axii_o.rresp <= 2'h3;
        endcase
      end else if (axio_i.araddr[31:28] == 1) begin
        for (int i=0; i<NF; i=i+1) begin
          if (axio_i.araddr == (FREQ_BA + 4*i)) begin

```

```

        axii_o.rdata <= freq[i];
    end
end
end else if (axio_i.araddr[31:28] == 2) begin
    for (int i=0; i<NF; i=i+1) begin
        if (axio_i.araddr == (DATA_BA + 4*i)) begin
            axii_o.rdata <= data[i];
        end
    end
end else begin
    axii_o.rresp <= 2'h3;
end
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;
        if (axio_i.awaddr[31:28] == 0) begin
            case (axio_i.awaddr)
                32'h0000_0000: version    <= axio_i.wdata;
                32'h0000_0004: debug      <= axio_i.wdata;
                32'h0000_000C: num_samp   <= axio_i.wdata;
                32'h0000_0010: samp_freq  <= axio_i.wdata;
                32'h0000_0014: en_cordic  <= axio_i.wdata;
                32'h0000_001C: reset_all  <= axio_i.wdata;
                32'h0000_0020: reset_h    <= axio_i.wdata;
                default: axii_o.bresp <= 2'h3;
            endcase
        end else if (axio_i.awaddr[31:28] == 1) begin
            for (int i=0; i<NF; i=i+1) begin
                if (axio_i.awaddr == (FREQ_BA + 4*i)) begin
                    freq[i] <= axio_i.wdata;
                end
            end
        end else begin
            axii_o.bresp <= 2'h3;
        end
    end
end
end
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
end
endmodule

```

11.3.4. Angel

```
module Angel #(
    parameter NF = 11
)(
    input                rstn        ,
    input                clk         ,
    input                en          ,
    output logic         valid       ,
    input                [NF-1:0][63:0] k_arr_i    , // (20.44)
    input                [63:0] ang_coef_i, // (20.44)
    output logic signed [NF-1:0][63:0] angel_o    // (20.44)
);

logic                [7 :0] indx ;
logic signed [63:0] mul_a; // (20.44)
logic signed [63:0] mul_b; // (20.44)
logic signed [63:0] mul_c; // (20.44)

assign mul_a = ang_coef_i ;
assign mul_b = (indx < NF) ? k_arr_i[indx] : 0;

mult_sign #(
    .DW      (64),
    .INT1_I(20),
    .INT2_I(20),
    .INT3_O(20)
) u_mult_sign (
    .a_in (mul_a),
    .b_in (mul_b),
    .c_out(mul_c),
    .c_ful(      )
);

always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        valid    <= 0;
        angel_o <= 0;
        indx     <= 0;
    end
    else if (en && !valid) begin
        if (indx < NF) begin
            angel_o[indx] <= mul_c ;
            indx          <= indx + 1;
        end
    end
end
```

```
        else begin
            valid <= 1;
        end
    end
end

endmodule
```

11.3.5. Cordic

```
module Cordic #(
    parameter NF = 11
)(
    // CLK&RST
    input                rstn ,
    input                clk  ,
    // CTRL
    input                en   ,
    output logic         valid,
    // DATA
    input    signed [NF-1:0][63:0] ang_i, // (20.44)
    output logic signed [NF-1:0][63:0] cos_o, // (32.32)
    output logic signed [NF-1:0][63:0] sin_o, // (32.32)
    output logic signed [NF-1:0][63:0] alpha // (32.32)
);

typedef enum {
    IDLE ,
    INIT ,
    QUAD ,
    CALC ,
    MULC ,
    PRIVC,
    ALPHA,
    MULS ,
    PRIVS,
    NEXT ,
    VALID
} state;

state curr_state;
state next_state;

logic signed [63:0] PI      = 64'h00003_243F6A8885A; // (20.44)
logic signed [63:0] PI2     = 64'h00001_921FB54442D; // (20.44)
logic signed [63:0] COEF_DEF = 64'h00000_9B74EDA8436; // (20.44)
logic signed [63:0] ZERO    = 64'h0;

logic signed [63:0] ang  ; // (20.44)
logic signed [63:0] cos  ; // (20.44)
logic signed [63:0] sin  ; // (20.44)
logic signed [63:0] atan ; // (20.44)
logic          [8 :0] indx0;
```



```

logic          [8 :0] indx1;
logic          [1 :0] quad ;

logic signed [63:0] mul_a; // (20.44)
logic signed [63:0] mul_b; // (20.44)
logic signed [63:0] mul_c; // (20.44)

assign mul_a = COEF_DEF;

logic mul_start;
logic mul_busy ;
logic mul_valid;

mult_sign_clk #(
    .DW      (64),
    .INT1_I(20),
    .INT2_I(20),
    .INT3_O(32)
) u_mult_sign_clk (
    .rstn (rstn      ),
    .clk  (clk       ),
    .en   (mul_start),
    .a_in (mul_a     ),
    .b_in (mul_b     ),
    .valid(mul_valid),
    .c_out(mul_c     )
);

always_ff @(negedge clk, negedge rstn) begin
    if (!rstn) begin
        curr_state <= IDLE;
    end
    else begin
        curr_state <= next_state;
    end
end

always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        cos_o <= 0;
        sin_o <= 0;
        alpha <= 0;
        valid <= 0;
        ang   <= 0;
        cos    <= 64'h00001_000000000000;
    end
end

```

```

sin    <= 64'h00000_000000000000;
indx0 <= 0;
indx1 <= 0;
quad  <= 0;
mul_b <= 0;
mul_start <= 0;
mul_busy <= 0;
next_state <= IDLE;
end
else begin
    case (curr_state)
        IDLE : begin
            if (en)
                next_state <= INIT;
            else
                next_state <= IDLE;
            end
        INIT : begin
            ang <= ang_i[indx0];
            next_state <= QUAD;
        end
        QUAD : 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
            next_state <= CALC;
        end
        CALC : begin
            if (indx1 < 45) begin
                if (ang[63] == 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
if (indx1 < 45)
    next_state <= CALC;
else
    next_state <= MULC;
end
MULC : begin
    if (quad == 2'b10) begin
        mul_b <= cos;
    end
    else if (quad == 2'b01) begin
        mul_b <= sin;
    end
    else begin
        mul_b <= cos;
    end
    if (!mul_busy)
        mul_start <= 1;
    if (mul_start) begin
        mul_busy <= 1;
        mul_start <= 0;
    end
    if (mul_valid) begin
        mul_busy <= 0;
        next_state <= PRIVC;
    end
end
PRIVC : begin
    if (quad == 2'b10) begin
        cos_o[indx0] <= ZERO - mul_c;
    end
    else if (quad == 2'b01) begin
        cos_o[indx0] <= ZERO - mul_c;
    end
    else begin
        cos_o[indx0] <= mul_c;
    end
    next_state <= ALPHA;
end
ALPHA : begin
    alpha[indx0] <= cos_o[indx0] <<< 1;
    next_state <= MULS;
end

```

```

end
MULS : begin
    if (quad == 2'b10) begin
        mul_b <= sin;
    end
    else if (quad == 2'b01) begin
        mul_b <= cos;
    end
    else begin
        mul_b <= sin;
    end
    if (!mul_busy)
        mul_start <= 1;
    if (mul_start) begin
        mul_busy <= 1;
        mul_start <= 0;
    end
    end
    if (mul_valid) begin
        mul_busy <= 0;
        next_state <= PRIVS;
    end
end
PRIVS : begin
    if (quad == 2'b10) begin
        sin_o[indx0] <= ZERO - mul_c;
    end
    else if (quad == 2'b01) begin
        sin_o[indx0] <= mul_c;
    end
    else begin
        sin_o[indx0] <= mul_c;
    end
    end
    next_state <= NEXT;
end
NEXT : begin
    if (indx0 < NF) begin
        cos    <= 64'h00001_000000000000;
        sin    <= 64'h00000_000000000000;
        indx0 <= indx0 + 1                ;
        indx1 <= 0                        ;
    end
    if (indx0 < NF - 1)
        next_state <= INIT;
    else
        next_state <= VALID;

```

```

end
VALID : begin
    valid <= 1;
    next_state <= VALID;
end
default: next_state <= curr_state;
endcase
end
end

always_comb begin
    case (indx1)
        0      : atan = 64'h00000_C90FDAA2217; // atanh(2^(-0 ))
        1      : atan = 64'h00000_76B19C1586F; // atanh(2^(-1 ))
        2      : atan = 64'h00000_3EB6EBF2590; // atanh(2^(-2 ))
        3      : atan = 64'h00000_1FD5BA9AAC3;  // atanh(2^(-3 ))
        4      : atan = 64'h00000_0FFAADDDB968; // atanh(2^(-4 ))
        5      : atan = 64'h00000_07FF556EEA6;  // atanh(2^(-5 ))
        6      : atan = 64'h00000_03FFFEAAB777; // atanh(2^(-6 ))
        7      : atan = 64'h00000_01FFFD555BC;  // atanh(2^(-7 ))
        8      : atan = 64'h00000_00FFFFAAAAE;  // atanh(2^(-8 ))
        9      : atan = 64'h00000_007FFFF5555;  // atanh(2^(-9 ))
        10     : atan = 64'h00000_003FFFFEAAB;   // atanh(2^(-10))
        11     : atan = 64'h00000_001FFFFFD55;   // atanh(2^(-11))
        12     : atan = 64'h00000_000FFFFFAB;    // atanh(2^(-12))
        13     : atan = 64'h00000_0007FFFFF5;    // atanh(2^(-13))
        14     : atan = 64'h00000_0003FFFFF5;    // atanh(2^(-14))
        15     : atan = 64'h00000_000200000000;  // atanh(2^(-15))
        16     : atan = 64'h00000_000100000000;  // atanh(2^(-16))
        17     : atan = 64'h00000_000080000000;  // atanh(2^(-17))
        18     : atan = 64'h00000_000040000000;  // atanh(2^(-18))
        19     : atan = 64'h00000_000020000000;  // atanh(2^(-19))
        20     : atan = 64'h00000_000010000000;  // atanh(2^(-20))
        21     : atan = 64'h00000_000008000000;  // atanh(2^(-21))
        22     : atan = 64'h00000_000004000000;  // atanh(2^(-22))
        23     : atan = 64'h00000_000002000000;  // atanh(2^(-23))
        24     : atan = 64'h00000_000001000000;  // atanh(2^(-24))
        25     : atan = 64'h00000_000000800000;  // atanh(2^(-25))
        26     : atan = 64'h00000_000000400000;  // atanh(2^(-26))
        27     : atan = 64'h00000_000000200000;  // atanh(2^(-27))
        28     : atan = 64'h00000_000000100000;  // atanh(2^(-28))
        29     : atan = 64'h00000_000000080000;  // atanh(2^(-29))
        30     : atan = 64'h00000_000000040000;  // atanh(2^(-30))
        31     : atan = 64'h00000_000000020000;  // atanh(2^(-31))
        32     : atan = 64'h00000_000000010000;  // atanh(2^(-32))
    endcase
end

```

```

33      : atan = 64'h00000_000000000800; // atanh(2^(-33))
34      : atan = 64'h00000_000000000400; // atanh(2^(-34))
35      : atan = 64'h00000_000000000200; // atanh(2^(-35))
36      : atan = 64'h00000_000000000100; // atanh(2^(-36))
37      : atan = 64'h00000_000000000080; // atanh(2^(-37))
38      : atan = 64'h00000_000000000040; // atanh(2^(-38))
39      : atan = 64'h00000_000000000020; // atanh(2^(-39))
40      : atan = 64'h00000_000000000010; // atanh(2^(-40))
41      : atan = 64'h00000_000000000008; // atanh(2^(-41))
42      : atan = 64'h00000_000000000004; // atanh(2^(-42))
43      : atan = 64'h00000_000000000002; // atanh(2^(-43))
44      : atan = 64'h00000_000000000001; // atanh(2^(-44))
    default: atan = 64'h00000_000000000000;
endcase
end

endmodule

```

11.3.6. DataScale

```
module DataScale (  
    // CLK&RST  
    input                rstn  ,  
    input                clk   ,  
    // CTRL  
    input                enable,  
    output logic         valid ,  
    // DATA  
    input                [7 :0] data_i, // (8.0)  
    output logic signed [63:0] data_o  // (32.32)  
);  
  
logic [7 :0] SCALE_COEF = 8'h0D; // 13/256 (0.8)  
logic [15:0] data_m; // (8.8)  
  
logic enable_syn;  
logic enable_old;  
logic enable_re ;  
  
assign data_o = {24'h0, data_m[15:0], 24'h0};  
  
resync_data #(  
    .NUM_STAGE(2)  
) u_resync_enable (  
    .rstn  (rstn      ),  
    .clk   (clk       ),  
    .data_i(enable     ),  
    .data_o(enable_syn)  
);  
  
always_ff @(negedge clk, negedge rstn) begin  
    if (!rstn) begin  
        valid      <= 0;  
        data_m      <= 0;  
        enable_old <= 0;  
        enable_re  <= 0;  
    end  
    else begin  
  
        enable_old <= enable_syn;  
        enable_re  <= ((enable_old == 1'b0) && (enable_syn == 1'b1));  
  
        if (enable_re) begin
```

```
        valid  <= enable          ;  
        data_m <= SCALE_COEF * data_i;  
    end  
    else begin  
        valid  <= 0;  
        data_m <= 0;  
    end  
  
    end  
end  
  
endmodule
```


11.3.7. Herzel

```
module Herzel #(
    parameter NF = 11,
    parameter DW = 64
)(
    // CLK&RST
    input                rstn    ,
    input                clk     ,
    // CTRL
    input                en      ,
    output logic         [NF-1:0] valid ,
    // data_i
    input                [31 :0] ns_i    , // (32.0 )
    input                [DW-1:0] ns_coef_i, // (32.32)
    input                signed [NF-1:0][DW-1:0] alpha_i , // (32.32)
    input                signed [NF-1:0][DW-1:0] cw_re_i , // (32.32)
    input                signed [NF-1:0][DW-1:0] cw_im_i , // (32.32)
    input                signed [DW-1:0] data_i , // (32.32)
    output logic unsigned [NF-1:0][31 :0] data_o    // (16.16)
);

typedef enum {
    CALC ,
    MULR ,
    NORMR,
    GRADR,
    MULI ,
    NORMI,
    GRADI,
    VALID
} state;

state curr_state;
state next_state;

logic signed [NF-1:0][DW-1:0] vm1 ; // (32.32)
logic signed [NF-1:0][DW-1:0] vm2 ; // (32.32)
logic signed [NF-1:0][DW-1:0] data; // (32.32)
logic                [31 :0] indx; // (32.32)

logic signed [NF-1:0][DW-1:0] mul_a; // (32.32)
logic signed [NF-1:0][DW-1:0] mul_b; // (32.32)
logic signed [NF-1:0][DW-1:0] mul_c; // (32.32)
```

```

logic          en_r      ;
logic [DW-1:0] data_r    ;
logic          mul_en    ;
logic          mul_start;
logic          mul_busy  ;
logic [NF-1:0] mul_valid;

genvar gvar;
generate
    for (gvar = 0; gvar < NF; gvar = gvar + 1) begin : herzel
        mult_sign_clk #(
            .DW      (DW ),
            .INT1_I(DW/2),
            .INT2_I(DW/2),
            .INT3_O(DW/2)
        ) u_mult_sign_clk(
            .rstn (rstn          ),
            .clk  (clk           ),
            .en   (mul_start     ),
            .a_in (mul_a[gvar]   ),
            .b_in (mul_b[gvar]   ),
            .valid(mul_valid[gvar]),
            .c_out(mul_c[gvar]   )
        );
    end
endgenerate

always_ff @(negedge clk, negedge rstn) begin
    if (!rstn) begin
        curr_state <= CALC;
    end
    else begin
        curr_state <= next_state;
    end
end

always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        en_r    <= 0;
        data_r  <= 0;
        valid   <= 0;
        data_o  <= 0;
        vm1     <= 0;
        vm2     <= 0;
        data    <= 0;
    end
end

```

```

    indx    <= 0;
    next_state <= CALC;
end
else begin
    if (en) begin
        en_r    <= 1;
        data_r <= data_i;
    end
    if (&mul_valid || next_state == VALID) begin
        case (curr_state)
            CALC : begin
                if (en_r) begin
                    if (indx < ns_i) begin
                        for (int i=0;i<NF;i++) begin
                            vm1[i] <= data_r + mul_c[i] - vm2[i];
                            vm2[i] <= vm1[i];
                        end
                        indx <= indx + 1;
                    end
                    en_r    <= 0;
                    data_r <= 0;
                end
                if (indx < ns_i)
                    next_state <= CALC;
                else
                    next_state <= MULR;
            end
            MULR : begin
                data <= mul_c - vm2;
                next_state <= NORMR;
            end
            NORMR : begin
                data <= mul_c;
                next_state <= GRADR;
            end
            GRADR : begin
                for (int i=0;i<NF;i++)
                    data_o[i] <= mul_c[i][DW/2+16-1:DW/2-16];
                next_state <= MULI;
            end
            MULI : begin
                data <= mul_c;
                next_state <= NORMI;
            end
            NORMI : begin

```

```

        data <= mul_c;
        next_state <= GRADI;
    end
    GRADI : begin
        for (int i=0;i<NF;i++)
            data_o[i] <= data_o[i] + mul_c[i][DW/2+16-1:DW/2-16];
            next_state <= VALID;
        end
    VALID : begin
        for (int i=0;i<NF;i++)
            valid[i] <= 1;
            next_state <= VALID;
        end
        default: next_state <= curr_state;
    endcase
end
end
end

```

```

always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        mul_a      <= 0;
        mul_b      <= 0;
        mul_en      <= 0;
        mul_start   <= 0;
        mul_busy    <= 0;
    end
    else begin
        if (en) begin
            mul_en <= 1;
        end
        if (!mul_busy && (next_state != VALID) && mul_en) begin
            mul_start <= 1;
            mul_busy  <= 1;
        end
        else begin
            mul_start <= 0;
        end
        if (&mul_valid) begin
            mul_busy <= 0;
        end
        case (curr_state)
            CALC : begin
                mul_a <= alpha_i;
            end
        endcase
    end
end

```

```

        mul_b <= vm1;
    end
    MULR : begin
        mul_a <= vm1;
        mul_b <= cW_re_i;
    end
    NORMR : begin
        mul_a <= data;
        for (int i=0;i<NF;i++)
            mul_b[i] <= ns_coef_i;
        end
    end
    GRADR : begin
        mul_a <= data;
        mul_b <= data;
    end
    end
    MULI : begin
        mul_a <= vm1;
        mul_b <= cW_im_i;
    end
    end
    NORMI : begin
        mul_a <= data;
        for (int i=0;i<NF;i++)
            mul_b[i] <= ns_coef_i;
        end
    end
    GRADI : begin
        mul_a <= data;
        mul_b <= data;
    end
    end
    default: begin
        mul_a <= 0;
        mul_b <= 0;
    end
    end
endcase
end
end

endmodule

```

11.3.8. mult_sign

```
module mult_sign #(
    parameter DW      = 32, //
    parameter INT1_I = 32, // (32.0)
    parameter INT2_I = 32, // (32.0)
    parameter INT3_O = 32 // (32.0)
) (
    input  signed [DW-1 :0] a_in , // (INT1_I.FRQ1_I)
    input  signed [DW-1 :0] b_in , // (INT2_I.FRQ2_I)
    output signed [DW-1 :0] c_out, // (INT3_O.FRQ3_O)
    output signed [DW*2-1:0] c_ful // (INT3_O.FRQ3_O)
);

localparam FRQ1_I = DW      - INT1_I; //
localparam FRQ2_I = DW      - INT2_I; //
localparam FRQ3_O = DW      - INT3_O; // 16
localparam INT3   = INT1_I + INT2_I; // 20
localparam FRQ3   = DW      - INT3_O; // 44

logic signed [DW*2-1:0] c; // (INT3.FRQ3)

assign c = a_in * b_in;

assign c_out = {c[DW*2-1], c[DW*2-INT3+INT3_O-2:DW*2-INT3], c[DW*2-INT3-1:DW*2-INT3-FRQ3_O]};
assign c_ful = c;

endmodule
```

12.Список литературы

1. Мирошников Б. Н. Методы управления фотоэлектрическими параметрами фоторезисторов на основе PbS для импульсных оптикоэлектронных систем: дис. канд. тех. наук. М., 2016. 206 с.
2. Fixed-Point Division. [Электронный ресурс] URL: <https://projectf.io/posts/division-in-verilog/> (дата обращения: 20.05.2023).
3. Дайнеко Д. Реализация CORDIC-алгоритма на ПЛИС. [Электронный ресурс] URL: <https://kit-e.ru/fpga/cordic/> (дата обращения: 20.05.2023).
4. Алгоритм Гёрцеля. [Электронный ресурс] URL: <https://ru.dsplib.org/content/goertzel/goertzel.html> (дата обращения: 20.05.2023).
5. ГОСТ Р 59606-2021. Оптика и фотоника. Устройства фотоприёмные второго и последующих поколений. Методы измерений фотоэлектрических параметров и определения характеристик. 12 октября 2021 г. N 132-ст.
6. Booth Multipliers in Verilog 2001. [Электронный ресурс] URL: https://github.com/MorrisMA/Booth_Multipliers.git (дата обращения: 20.05.2023).
7. Future Technology Devices International Ltd UMFT4222EV USB2.0 to QuadSPI/I2C Bridge Development Module Datasheet. Document No.: FT_001057 Clearance No.: FTDI#407
8. python-ft4222. [Электронный ресурс] URL: <https://msrelectronics.gitlab.io/python-ft4222/> (дата обращения: 20.05.2023).