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

Институт ИРЭ
Кафедра ЭиН

ВЫПУСКНАЯКВАЛИФИКАЦИОННАЯ РАБОТА
(МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

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

«Работа допущена к защите»

Заведующий кафедрой д.т.н. профессор _____ Мирошникова И.Н.
уч. степень звание подпись фамилия и инициалы

Дата

Москва, 2023

1. Аннотация

Для решения задачи 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.

В результате разработки были получены следующие характеристики:

Здесь будут точные характеристики по разным режимам работы

2. Оглавление

3. Введение

Данная работа относится к области цифровой схемотехнике, 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.

4. Основная часть

4.1. Типы шумов и их природа

Шум - 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.

Спектральная плотность внешнего шума равна:

$$P(f) = 2J \cdot \frac{exp\,(h\nu/kT)}{exp\,(h\nu/kT)-1} \qquad (1.1)$$

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

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

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

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

$$\mathrm{P_U(f)} = \frac{\overline{\mathrm{U^2(f)}}}{\Delta \mathrm{f}}, \Delta \mathrm{f} \to 0 \qquad (1.2)$$

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

$$\overline{\mathrm{U^2(f)}} = \mathrm{F}(\omega; \mathrm{t})\overline{\mathrm{U^2(t)}} \qquad (1.3)$$

Различные виду 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.

Общий спектр плотности мощности шума следующий:

$$P_U(f) = \frac{\overline{U_{\text{ш}}^2}}{\Delta f} = \frac{A \cdot U_{\text{ФР}}^2}{f^{\alpha} \cdot \mathrm{V}} + \frac{4U_{\text{ФР}}^2 \overline{\Delta p^2}}{\mathrm{p}^2 \cdot V} \cdot \frac{\tau}{1+(2\pi \cdot \mathrm{f})^2 \tau^2} + 4kT_d R \qquad (1.4)$$

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

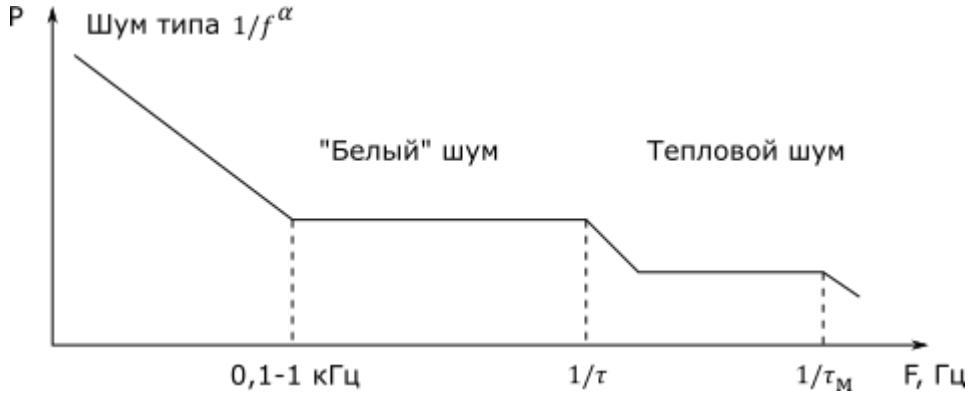


Рисунок 1 Структура спектра плотности мощности шума фотоприемника

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

4.1.1. 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.

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

$$P_u(f) = \frac{\overline{U_{\text{ш}}^2}}{\Delta f} = 4kT_dR \qquad (1.5)$$

$U_{\text{ш}}$ – среднеквадратическое значение напряжения шума

Δf – полоса пропускания измерительного тракта

T_d – рабочая температура ФР

R – электрическое сопротивление ФР

4.1.2. 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 появлении 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.

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.

$$P_u(f) = \frac{4U_{\text{ФР}}^2}{V} \cdot \frac{(b+1)^2}{(bn+p)} \cdot \frac{np}{n+p} \cdot \frac{\tau}{1+\omega^2\tau^2} \quad (1.6)$$

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

n, p – концентрация электронов и дырок

4.1.3. Фликкер-шум

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

$$P_u(f) = \frac{\overline{U_{\text{ш}}^2}}{\Delta f} = \frac{AU_{\text{ФР}}^2}{f^\alpha} \quad (1.6)$$

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

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,

имеющихся в данном образце. И происходит косвенное изменение числа носителей в образце. Изменение заполнения поверхностных ловушек влияет на генерацию неосновных носителей в центрах быстрой рекомбинации вблизи поверхности.

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

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

Формула для расчёта частотной характеристики шума из теории Хоухе:

$$P_i(f) = \frac{\alpha_H I_{\text{н}}^2}{N \cdot f^{\alpha}} \qquad (1.7)$$

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

V – объем исследуемого образца

n – концентрация носителей

α_H– постоянная Хоухе.

4.2. 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.

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

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

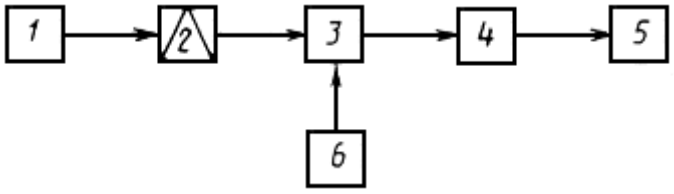


Рисунок 2 1 - источник излучения; 2 - модулятор; 3 - испытуемый фотоприёмник; 4 - усилительное устройство; 5 - регистрирующий прибор; 6 - источник питания фотоприёмника

При выборе источника излучения рекомендуется применять следующие источники излучения:

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

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

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

В состав источников излучения для ослабления потока излучения или для увеличения плотности мощности могут входить ослабители

(аттенюаторы), зеркала, линзы, объективы и другие оптические элементы. Влияние оптических элементов не должно учитываться, если они изменяют коэффициент использования излучения за счет изменения его спектрального состава не более чем на 2%. Под коэффициентом использования излучения следует понимать

$$\varphi = \frac{\int_0^\infty S_{\text{отн}}(\lambda) r_\lambda d\lambda}{\int_0^\infty r_\lambda d\lambda} \qquad (2.1)$$

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 ±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 ±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 ±3%.

Коэффициент пульсации должен находиться в пределах ±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 ТУ на фотоприёмник конкретного типа:

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

К испытуемому 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 близки 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:

$$U_{\text{ш}} = \sqrt{U_{\text{ш}2}^2 - U_{\text{ш}1}^2} \qquad (2.2)$$

где $U_{\text{ш}2}$ - суммарное напряжение шума при подаче напряжения, питания фотоприёмника.

$U_{\text{ш}1}$ - 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.

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

4.2.1. Установка

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.

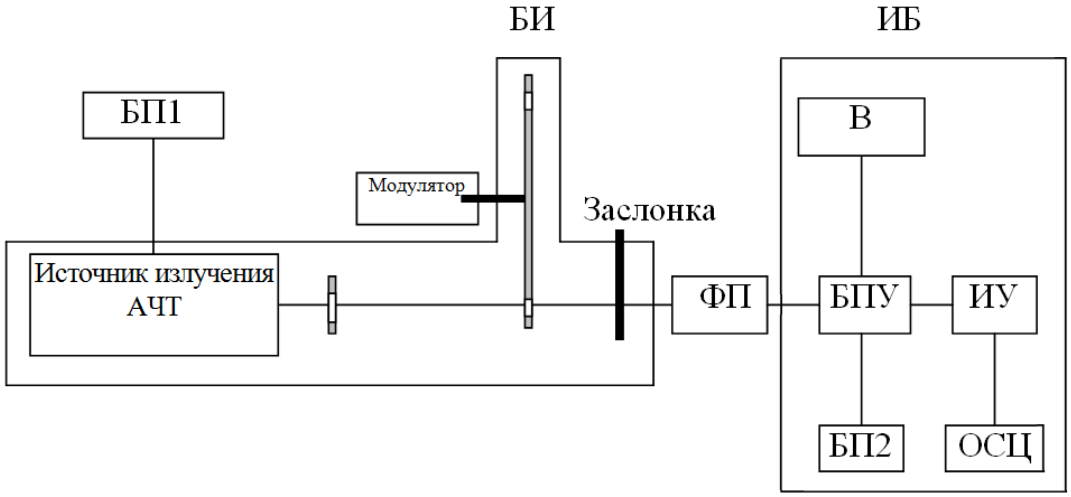


Рисунок 3 Структурная схема установки для измерения параметров ФЧЭ

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

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

4.3. 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 (SAR) показан рис.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.

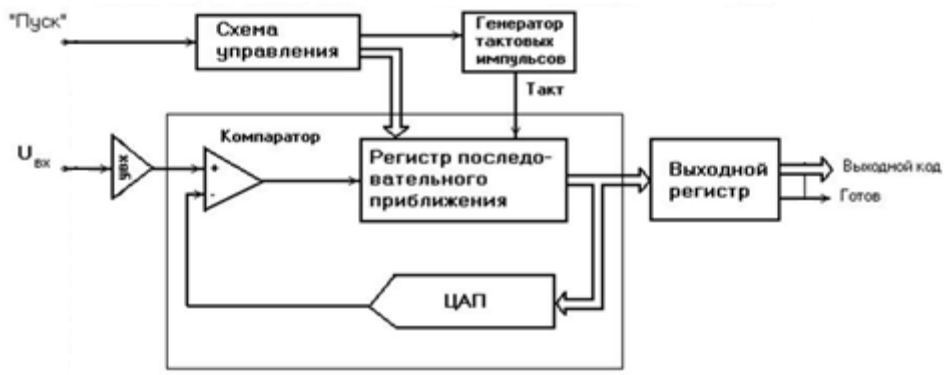


Рисунок 5 АЦП последовательного приближения

4.4. 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.

$$f(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-ix\omega}\, dx \qquad (4.1)$$

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

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

На рис. 4 примерно представлено примерный принцип преобразования временного сигнала в частотный.

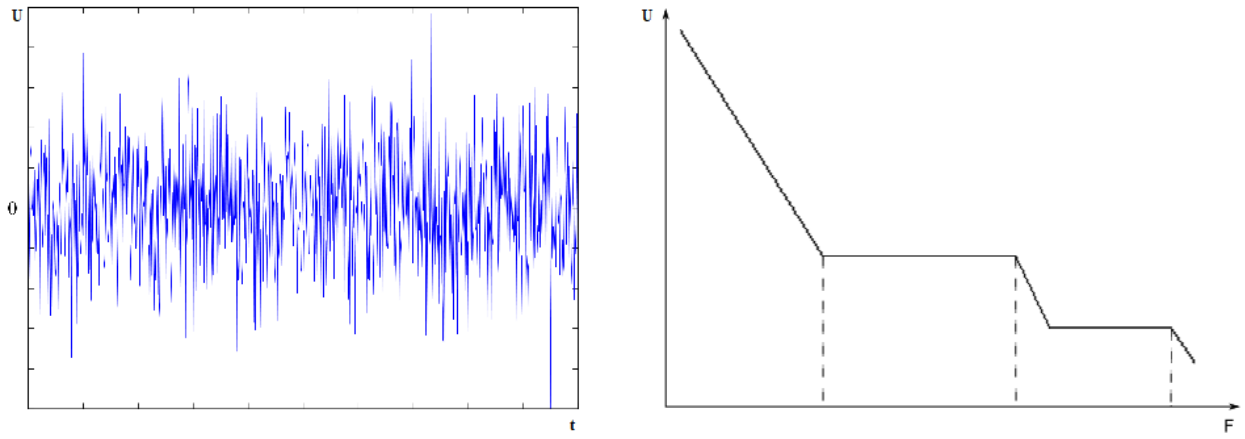


Рисунок 6 Входной сигнал слева, преобразованный через Фурье справа

Для цифровой схемотехники применяют дискретное преобразование Фурье:

$$M(f) = \frac{1}{N} \sum_{k=0}^{N-1} g(t_k) \cdot [cos(2\pi f t_k) + i \cdot sin(2\pi f t_k)], \qquad (4.2)$$

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:

$$[cos(2\pi f t_k) + i \cdot sin(2\pi f t_k)] \qquad (4.3)$$

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.

4.5. Устройство устройства

Структура проекта выглядит следующим образом:

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

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

Моделирование проводилось в MicroCap.

Интерфейс модуля представлен на 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.

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

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

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

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

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

Далее начнётся оцифровка нового значения.

Полученный код:

$$10010110_2 = 150_{10}$$

Соответствующее ему напряжение:

$$150 \cdot \frac{13}{256} = 7.62$$

4.5.2. 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 — VOD. Сигнал в каждой из двух сигнальных линий имеет максимальный размах |VOD| и центрирован относительно синфазного напряжения VOC. Стандартные значения этих напряжений показаны на рис. 7.

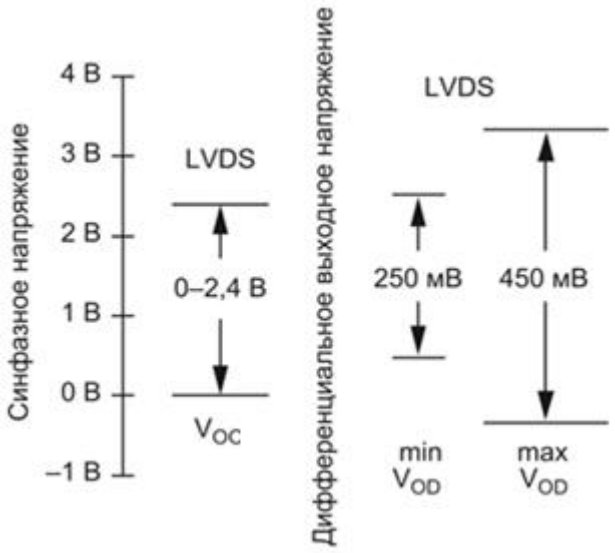


Рисунок 7 Уровни сигналов в LVDS и M-LVDS

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

Данные LVDS драйверы устанавливаются для каждого бита выходного кода АЦП и параллельно передаются на FPGA. Общая схема данного соединения показана рис. 10. Терминирующий резистор R_T имеет номинал в 100 Ом и необходим для согласования.

Рисунок 8 Канал связи между приёмником и передатчиком

4.5.3. Модуль FourierTransform

Данный модуль является топ-уровнем всего RTL дизайна.

Интерфейс модуля представлен на 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:

- IBUFDS
- resync_nrst
- resync_data
- spi2axi
- HerzelRegs
- div_all
- Angel
- Cordic
- DataScale
- Herzel

Для управления модулем предусмотрен SPI интерфейс.

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

Карта памяти топ-уровня представлена в таблице 3.

Таблица 3 Карта памяти модуля FourierTransform

Адрес	Тип	Сброс	Название	Описание
0x00	RW	0x29042023	VERSION	Версия модуля
0x04	RW	0xF0F0F0F0	DEBUG	Тестовый регистр
0x08	RW	0x0	MODE	Режим работы 0 – данные поступают постепенно 1 – данные поступают непрерывно каждый такт
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 18-8 биты – вычислены данные по каждому модулю Herzel
0x1C	RW	0x0	RESET_ALL	Сброс всех модулей кроме регистров для нового расчёта или перерасчёта

0x20	RW	0x0	RESET_H	Сброс только модуля Herzel для нового расчёта или перерасчёта
от 0x10000000 до 0x1000000 + 0x4 * NF	RW	0x0	FREQ	Частота. Число регистров и соответственно число частот для расчёта задаётся параметром NF главного модуля.
от 0x20000000 до 0x2000000 + 0x4 * NF	RW	0x0	DATA	Рассчитанные данные для частот.

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. Если необходимо также выбрать режим работы через регистр MODE. 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 битах 18-8 регистра STATUS установятся 1, после чего можно будет считывать данные с регистров DATA_1- DATA_NF.

4.5.3.1. 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 на в таблице 4.

Таблица 4 Интерфейс модуля IBUFDS

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

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

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

```
set_property -dict {IOSTANDARD LVDS PACKAGE_PIN ???} [get_ports enable_p]
set_property -dict {IOSTANDARD LVDS PACKAGE_PIN ???} [get_ports enable_n]
set_property -dict {IOSTANDARD LVDS PACKAGE_PIN ???} [get_ports sample_n[i]]
```

И так далее для каждого дифференциального сигнала.

4.5.3.2. resync_nrst

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

Интерфейс модуля представлен на 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

4.5.3.3. 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 на 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.

4.5.3.4. spi2axi

Данный модуль преобразует SPI транзакции в AXI-lite транзакции.

Интерфейс модуля представлен на 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 рис. 9.

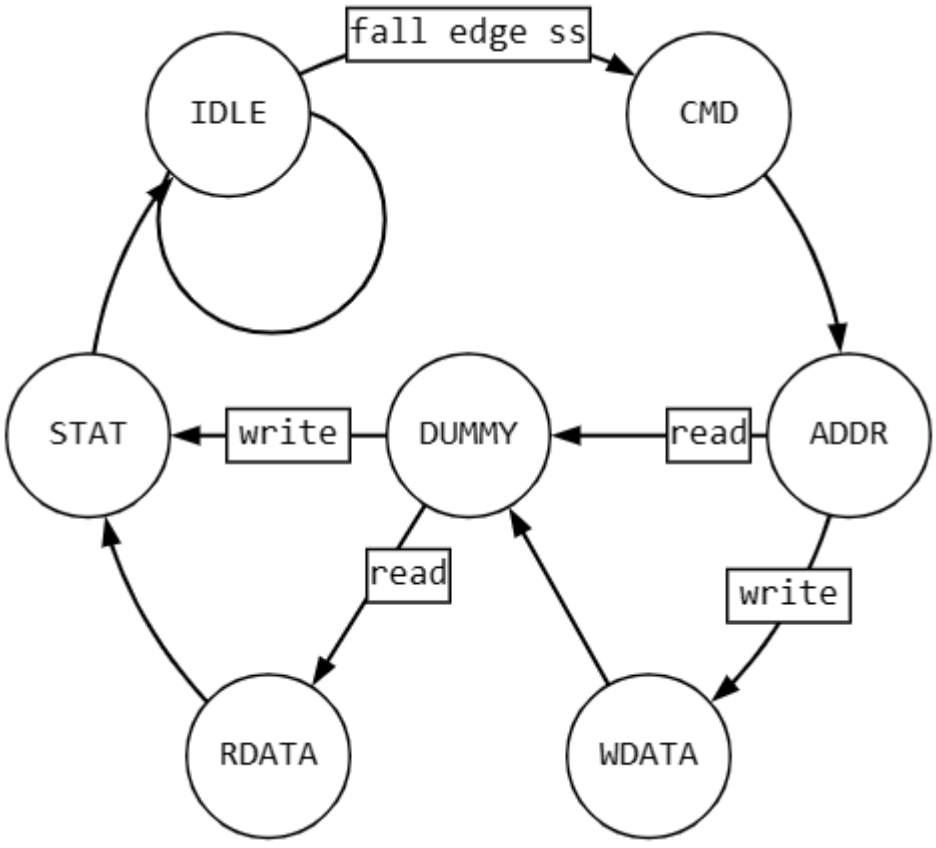


Рисунок 9 Граф состояний модуля HerzelRegs

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 рис. 9.

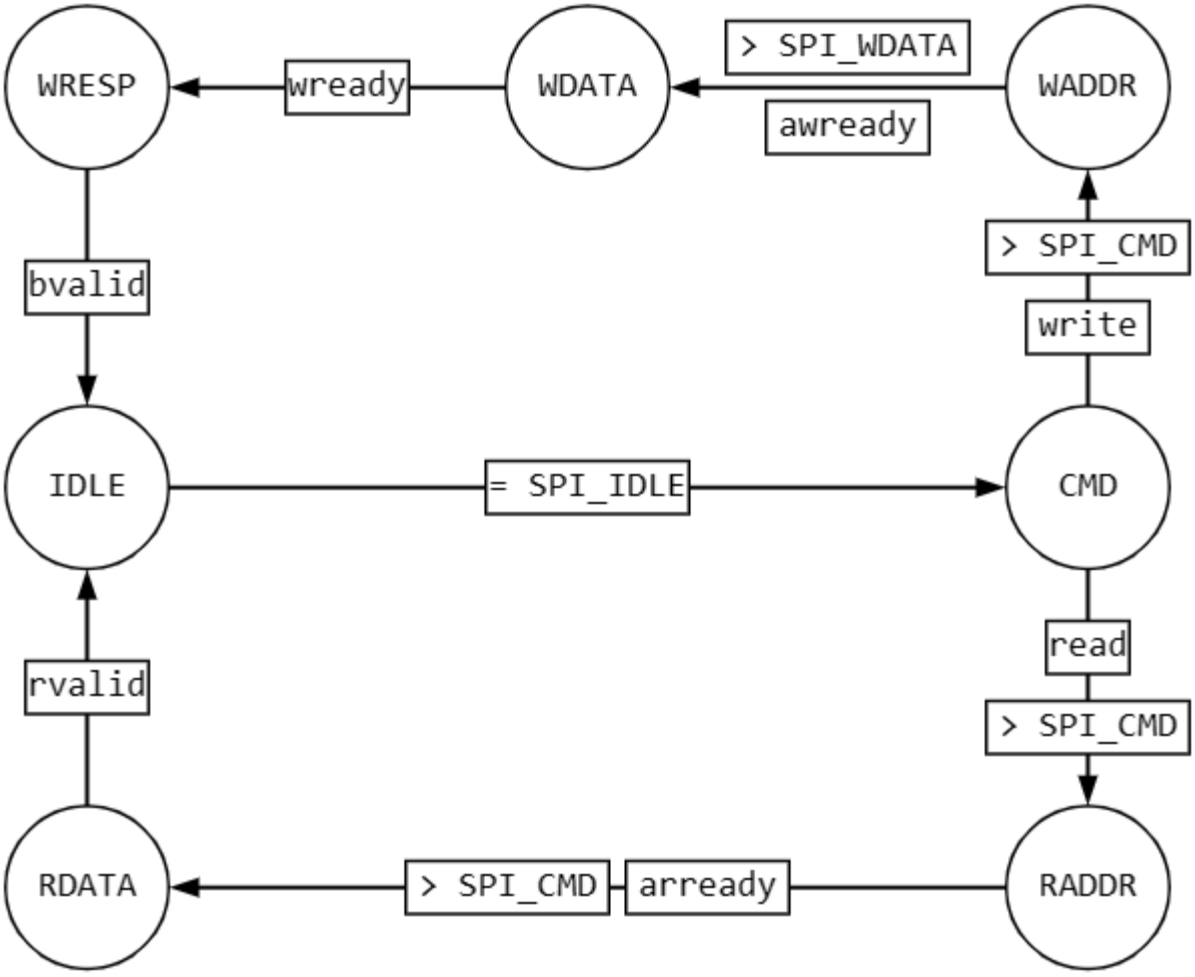


Рисунок 10 Граф состояний модуля HerzelRegs

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

4.5.3.5. HerzelRegs

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

Интерфейс модуля представлен на в таблице 01.

Таблица 10 Интерфейс модуля resync_data

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
mode_o	ВЫХОД	[0]	Режим работы
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

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 (wwalid). После запись данных по принятому адресу, отправление ответа (wready) и переход в следующее состояние.
- WRESP – отправка статуса (bresp) с сигналом действительности данных (bvalid) и ожидание ответа (bready). После получения ответа (bready) транзакция завершается и переходит в IDLE.

  Граф машины состояний показан на рис.11.

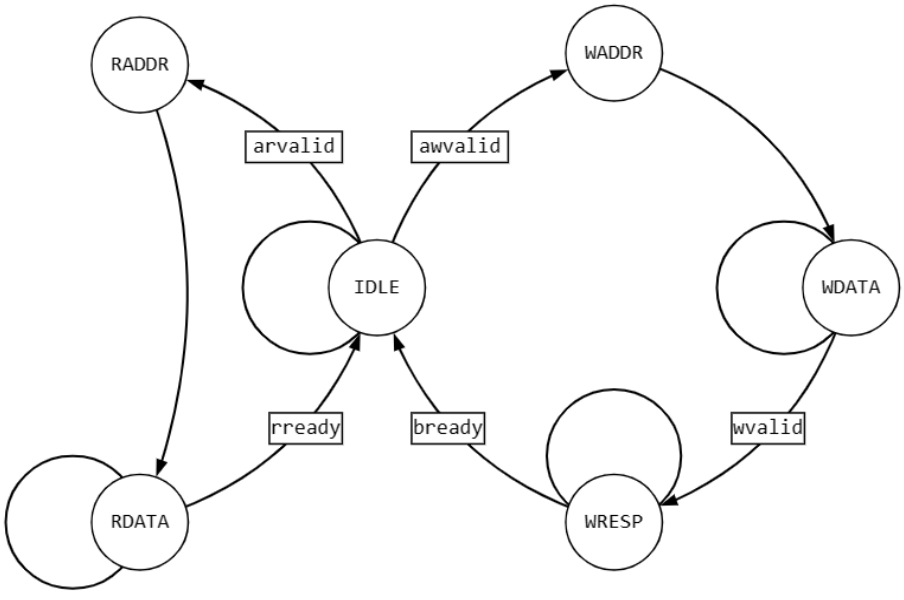


Рисунок 11 Граф состояний модуля HerzelRegs

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

4.5.3.6. div_all

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

Интерфейс модуля представлен на 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 samp_freq_i/num_samp_i.

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

Граф машины состояний показан на рис.12.

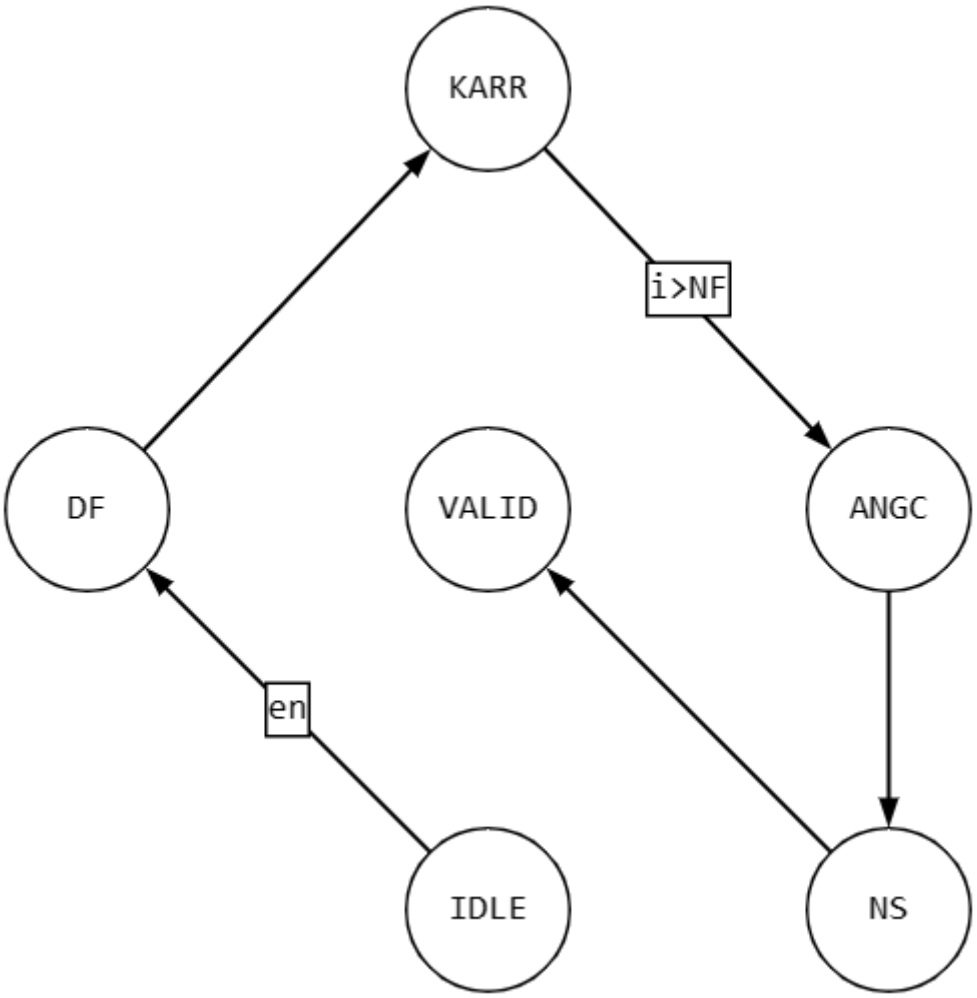


Рисунок 12 Граф состояний модуля div_all

4.5.3.7. 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 на 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:

$$k = \frac{FREQ}{df}, \text{где } df = \frac{f}{NS} = 2Hz \qquad (5.1)$$

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.

4.5.3.8. 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 на 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.

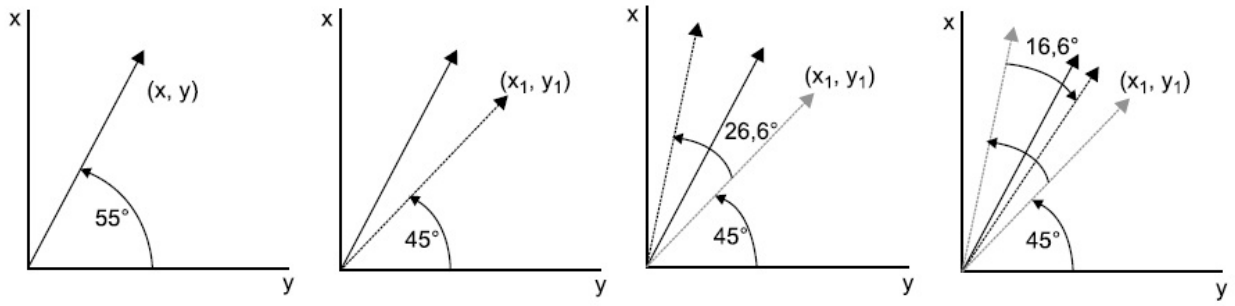


Рисунок 13 CORDIC алгоритм

Координаты x и y вычисляются по формулам:

$$x_{i+1} = x_i \cdot \cos\varphi - y_i \cdot \sin\varphi \quad (5.2)$$

$$y_{i+1} = x_i \cdot \sin\varphi + y_i \cdot \cos\varphi \quad (5.3)$$

После преобразования:

$$x_{i+1} = \cos\varphi \cdot (x_i - y_i \cdot \tan\varphi) \quad (5.4)$$

$$y_{i+1} = \cos\varphi \cdot (y_i + x_i \cdot \tan\varphi) \quad (5.5)$$

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:

$$x_{i+1} = x_i - \sigma_i \cdot y_i \cdot 2^{-i} \quad (5.6)$$

$$y_{i+1} = y_i + \sigma_i \cdot x_i \cdot 2^{-i} \quad (5.7)$$

$$ang_{i+1} = ang_i - \tan\varphi_i \quad (5.8)$$

$$\sigma_i = \frac{ang}{|ang|} = \pm 1 \quad (5.9)$$

$$K(n) = \prod_{i=0}^{ns-1} \frac{1}{\sqrt{1+2^{-2i}}} = 0.607253, ns = 24 \quad (5.10)$$

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 рис. 14.

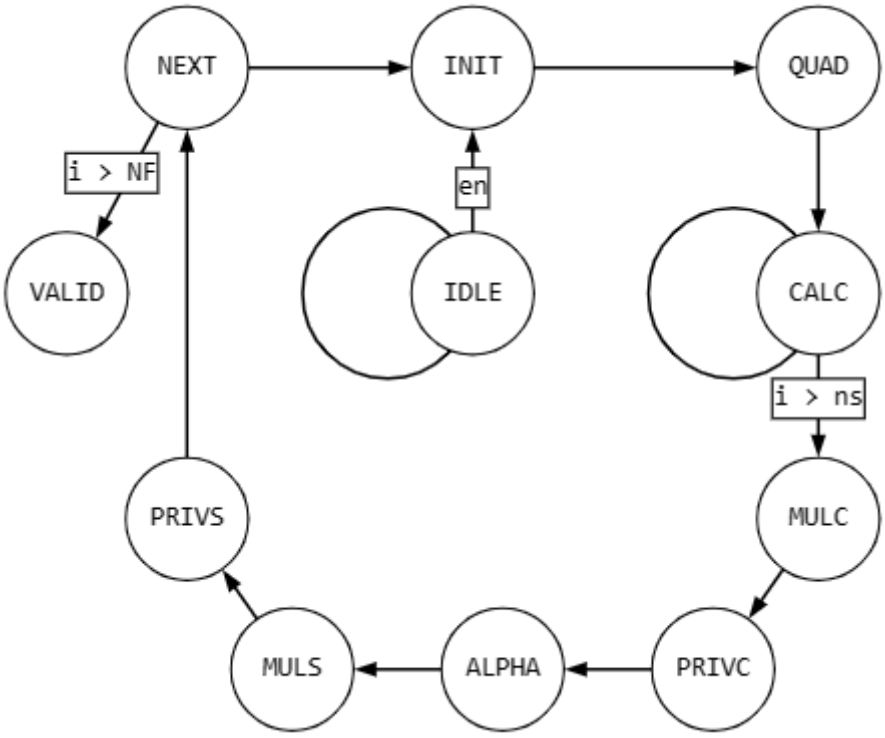


Рисунок 14 Граф состояний модуля Cordic

- 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, определённому в QUAD состоянии, по формулам приведения.
- ALPHA – вычисление коэффициента альфа.
- MULS – умножение рассчитанного синуса на коэффициент масштабирования K.
- PRIVS – приведение синуса к начальному квадранту, определённому в QUAD состоянии, по формулам приведения.

- NEXT – сброс значений, увеличение счётчика выбора угла.
- VALID – завершение вычислений, установка сигнала действительности данных valid.

4.5.3.9. DataScale

Данный модуль предназначен для масштабирования входных данных по следующей формуле:

$$data = \frac{13}{256} \cdot sample \quad (5.11)$$

13/256 – коэффициент, определяемый АЦП.

Также выходные данные доводятся до нужного формата и ширины.

Интерфейс модуля представлен на 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
mode	ВХОД	[0]	Режим работы
data_i	ВХОД	[7:0]	Данные с АЦП
data_o	ВЫХОД	[31:0]	Преобразованные данные

В случае последовательного приёма данных на порт mode подаётся 1, что обеспечивает правильную передачу сигнала valid и входных данных. Иначе из-за разницы частот АЦП и цифровой схемы сигнал valid будет слишком долгим, что приведёт к повторной записи одной выборки и нарушению расчётов.

4.5.3.10. Herzel

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

Подробный вывод формул – [4]

Интерфейс модуля представлен на 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 не полного ДПФ, а лишь фиксированного количества спектральных отсчетов.

По алгоритму спектральный отсчет S(k) равен:

$$S(k) = y_{N-1}(k) = W_N^{-k} \cdot \upsilon(N-1) - \upsilon(N-2) \quad (5.12)$$

Где υ - промежуточные значения, которые рассчитываются итерационно:

$$\upsilon(r) = s(r) + 2\cos\left(2\pi\frac{k}{N}\right)\upsilon(r-1) - \upsilon(r-2) \quad (5.13)$$

W – поворотный коэффициент

$$W_N^{nk} = exp\left(-j\frac{2\pi}{N}nk\right), k = 0 \dots N-1 \quad (5.14)$$

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 υ.
- MULR – умножение υ на Re часть поворотного коэффициента.
- MULI – умножение υ 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 рис.15.

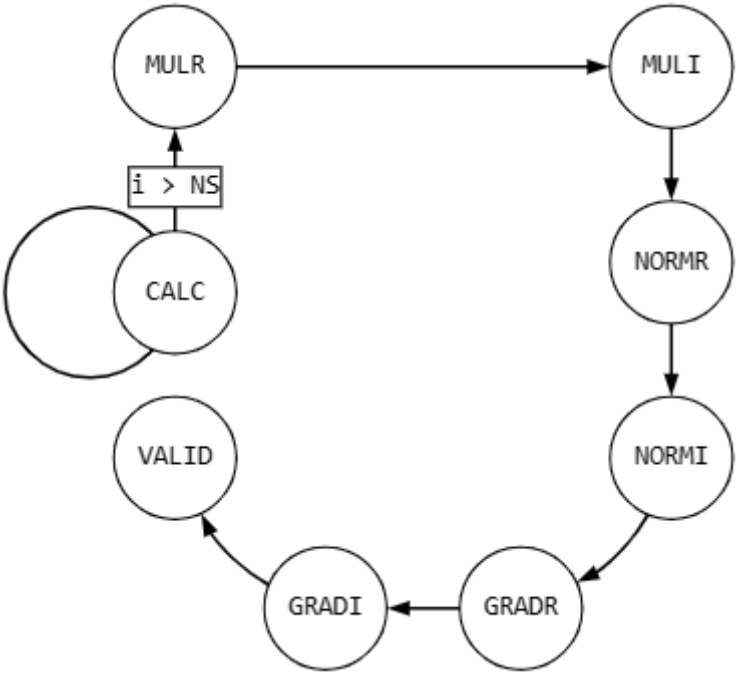


Рисунок 15 Граф состояний модуля Herzel

4.6. Моделирование FourierTransform

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

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

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

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

data.csv – рассчитанные данные из MatchCad.

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

Примеры некоторых моделирований приведены в приложении 2.

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

$$f(t) = \sum_{i=0}^{NF-1}[amp_i \cdot (cos(2\pi \cdot freq_i) + 1)] \qquad (5.16)$$

5. Заключение

Пока не придумал

6. Приложение 1

Была синтезирована схема рис.16. Число используемых элементов на ПЛИС приведено на рис.17.

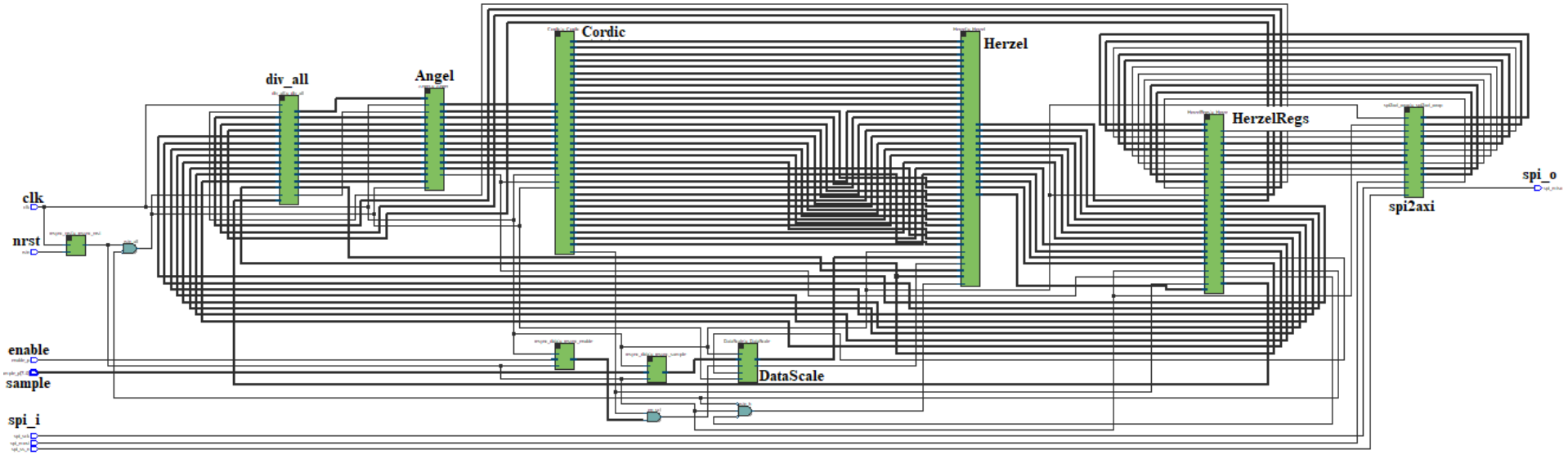


Рисунок 16 Синтезированная схема проекта

Flow Status	Flow Failed - Mon Jun 05 23:30:55 2023	Total logic elements	15,289 / 10,320 (148 %)
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition	Total registers	8131
Revision Name	SRW	Total pins	33 / 92 (36 %)
Top-level Entity Name	FourierTransform	Total virtual pins	0
Family	Cyclone IV E	Total memory bits	0 / 423,936 (0 %)
Device	EP4CE10E22C8	Embedded Multiplier 9-bit elements	46 / 46 (100 %)
Timing Models	Final	Total PLLs	0 / 2 (0 %)

Рисунок 17 Используемые элементы ПЛИС

7. Приложение 2

7.1. Модуль FourierTransform_tb

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

`define TEST

module FourierTransform_tb;

localparam CLK_PER     = 10  ;
localparam ADC_CLK_PER = 100 ;
localparam SPI_CLK_PER = 1000;
localparam NF          = 14  ;

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.mst.sck ;
assign spi_ss_n    = spi_if.mst.ss_n;
assign spi_mosi    = spi_if.mst.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)
) 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.cs
v", "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.herzel[9].u_Herzel.valid)) begin
    @(posedge clk);
    $fwrite(fd_w_v, "%h\n", DUT.herzel[8].u_Herzel.vm1);
  end
  $fwrite(fd_w_v, "%h\n", DUT.herzel[8].u_Herzel.vm1);
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 clk);
  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");
  if (DUT.u_DataScale.mode == 1) begin
    $fscanf(fd_r_s, "%d\n", sample_p);
    enable_p = 1;
    while (!$feof(fd_r_s)) begin
      @(posedge clk);
      $fscanf(fd_r_s, "%d\n", sample_p);
    end
    enable_p = 0;
  end
  else begin
    while (!$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
  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(5) @(posedge clk);
  rstn = 1;
  repeat(20) @(posedge clk);

  spi_if.write_data(NUM_SAMP , 32'd5000, spi_stat);
  spi_if.write_data(SAMP_FREQ, 32'd100000, spi_stat);
  spi_if.write_data(MODE     , 32'd1, 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

  repeat(1) begin
    spi_if.write_data(RESET_ALL, 32'd1, spi_stat);
    repeat(5) @(posedge clk);
    spi_if.write_data(RESET_ALL, 32'd0, 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

  # 5000;
  end_of_test();
  $stop;
end

endmodule
```

7.2. spi2axi интерфейс

```
`timescale 1ns/1ps

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

interface spi_if #(SPI_CLK_PER, DISPLAY = 0) ();

logic [31:0] write_instr = 32'h0000_0000;
logic [31:0] read_instr  = 32'h0000_0001;
logic [31:0] dummy_word  = 32'h0000_0001;

logic sck ;
logic ss_n;
logic mosi;
logic miso;

modport mst (
  output sck ,
  output ss_n,
  output mosi,
  input  miso
);

modport slv (
  input  sck ,
  input  ss_n,
  input  mosi,
  output miso
);

task init();
  mst.sck  = 0;
  mst.ss_n = 1;
  mst.mosi = 0;
endtask

task clk_en(input en);
  if (en) begin
    forever #(SPI_CLK_PER/2) sck = !sck;
  end
  else begin
    forever #(SPI_CLK_PER/2) sck = 0;
  end
endtask
```

```
task write(input int num_bit, input [31:0] data);
  for (int i=num_bit-1; i>=0; i=i-1) begin
    mst.mosi = data[i];
    @(posedge sck);
  end
  mst.mosi = 0;
endtask

task read(input int num_bit, output [31:0] data);
  data = 0;
  for (int i=num_bit-1; i>=0; i=i-1) begin
    data[i] = mst.miso;
    @(posedge sck);
  end
endtask

task read_data(input [31:0] addr, output [31:0] data, output [31:0]
status);
  fork
    begin
      clk_en(1);
    end
    begin
      // slave select
      mst.ss_n = 0;
      // instruction byte
      write(8, read_instr);
      // 4 address bytes
      @(posedge sck);
      write(32, addr);
      // dummy byte
      write(8, dummy_word);
      // 4 data bytes
      read(32, data);
      // status byte
      @(posedge sck);
      read(8, status);
      // slave select
      mst.ss_n = 1;
    end
  join_any
  @(negedge sck);
  disable fork;
```

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

endinterface
```

7.3. Результаты моделирования

Тест 1. По формуле 5.16 формируется сигнал с параметрами, показанными на рис.18. После чего с сигнала делаются выборки с частотой 200 кГц, число выборок – 100 т.шт. Данные параметры записываются в нужные регистры и начинается расчёт.

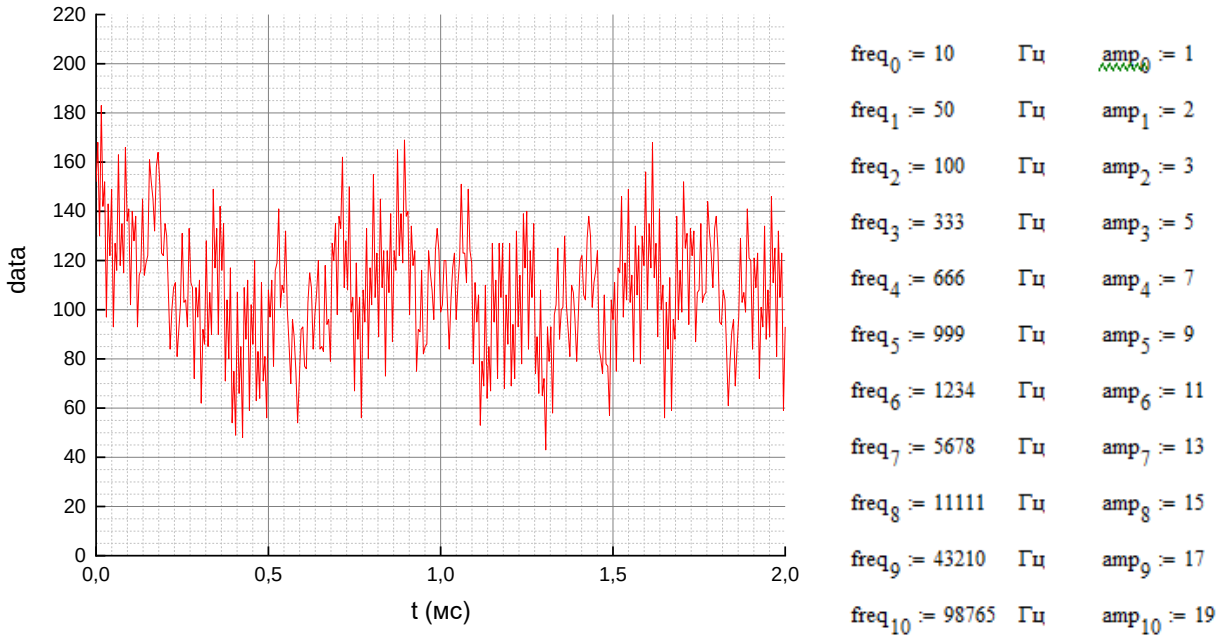


Рисунок 18 Сгенерированный сигнал для теста 1 и его параметры

Итоговый результат можно увидеть на рис.19. Как видно рассчитанные частоты и амплитуды соответствуют заданным значениям.

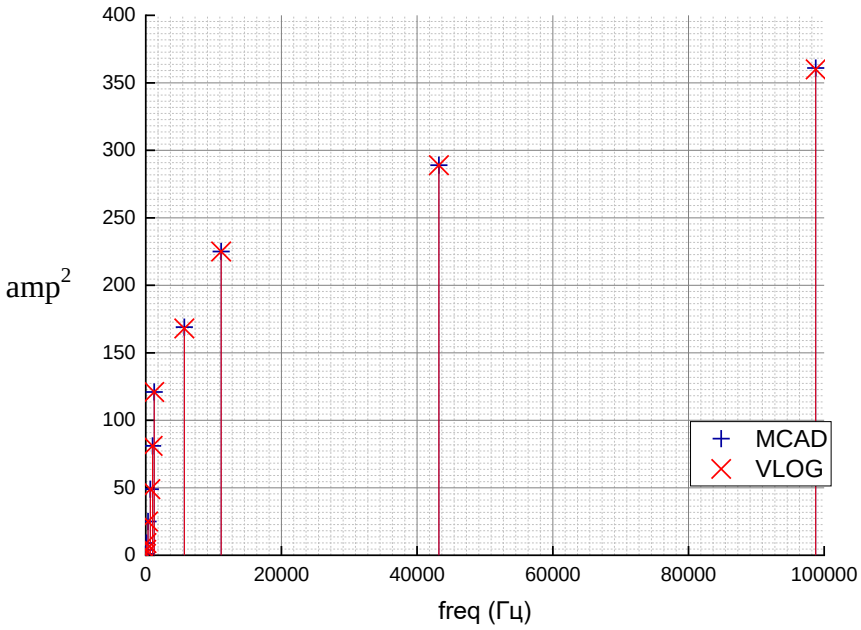


Рисунок 19 Результат расчёта Verilog модуля и MatchCad

Тест 2. На рис. 20-21. Приведён тот же тест, но с другими параметрами входного сигнала.

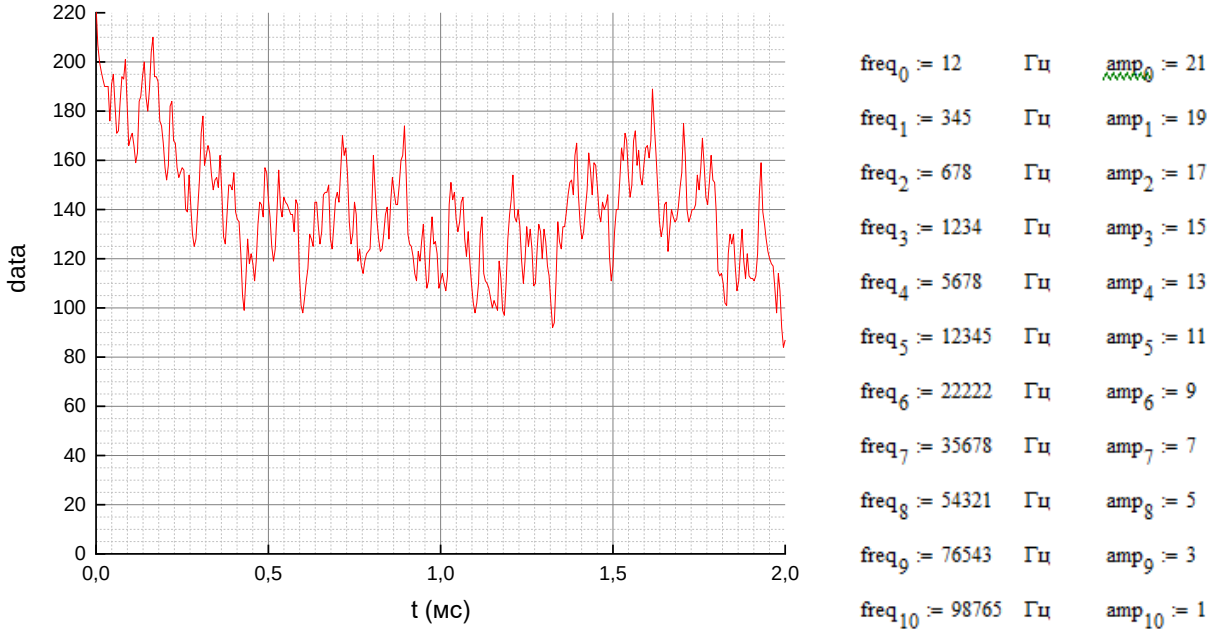


Рисунок 20 Сгенерированный сигнал для теста 2 и его параметры

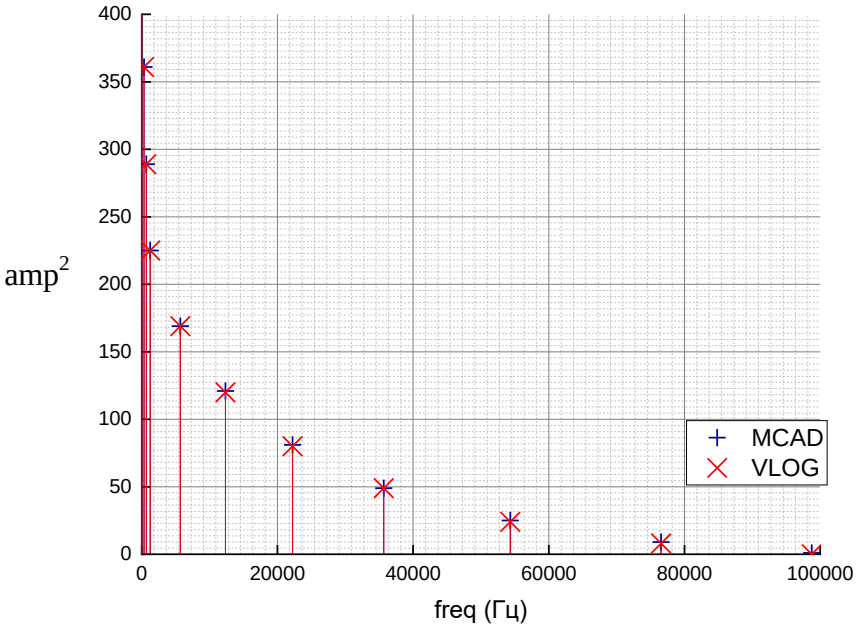


Рисунок 21 Результат расчёта Verilog модуля и MatchCad

На рис.22. 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 22 Вывод в консоль

На рис.23. 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.

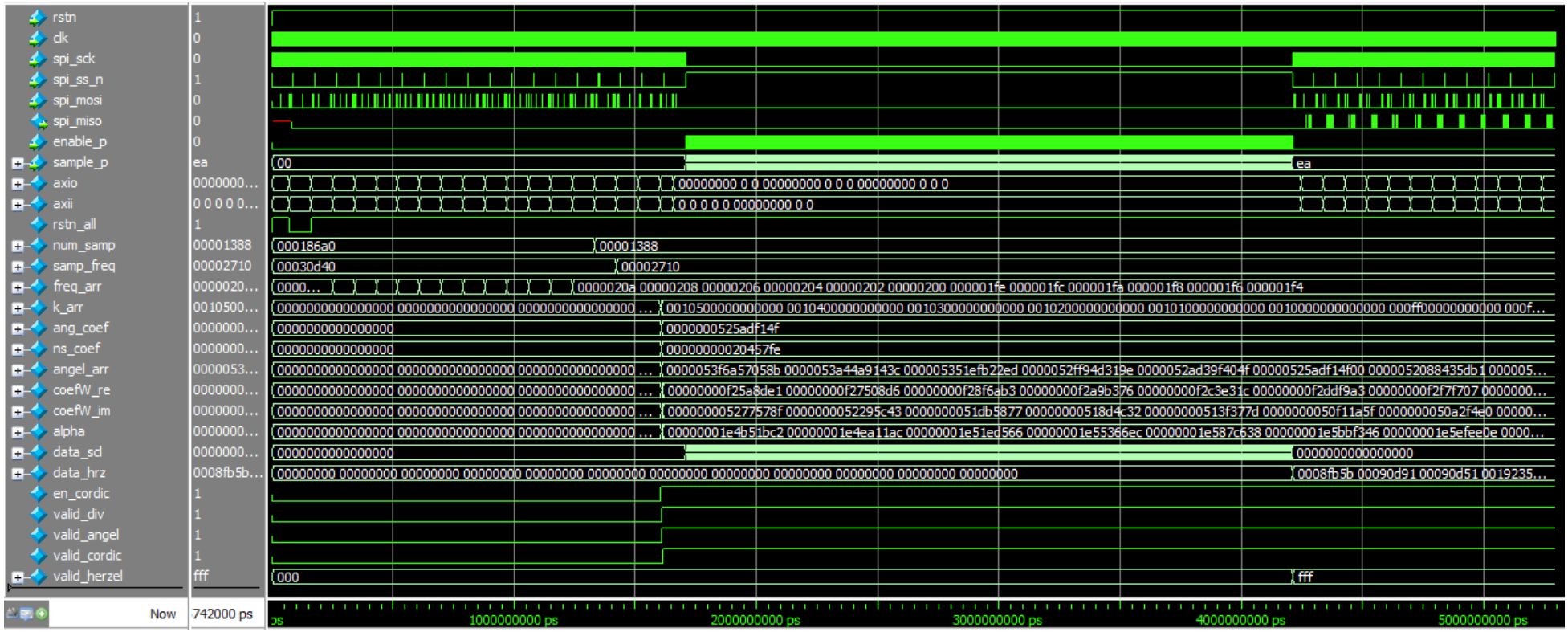


Рисунок 23 Осцилограмма модуля FourierTransform

На рис.24. показана AXI-lite ранзакция записи данный в регистры. На ней можно увидеть соответствующие запросы от ведущего и следующие за ними ответы от ведомого и наоборот.

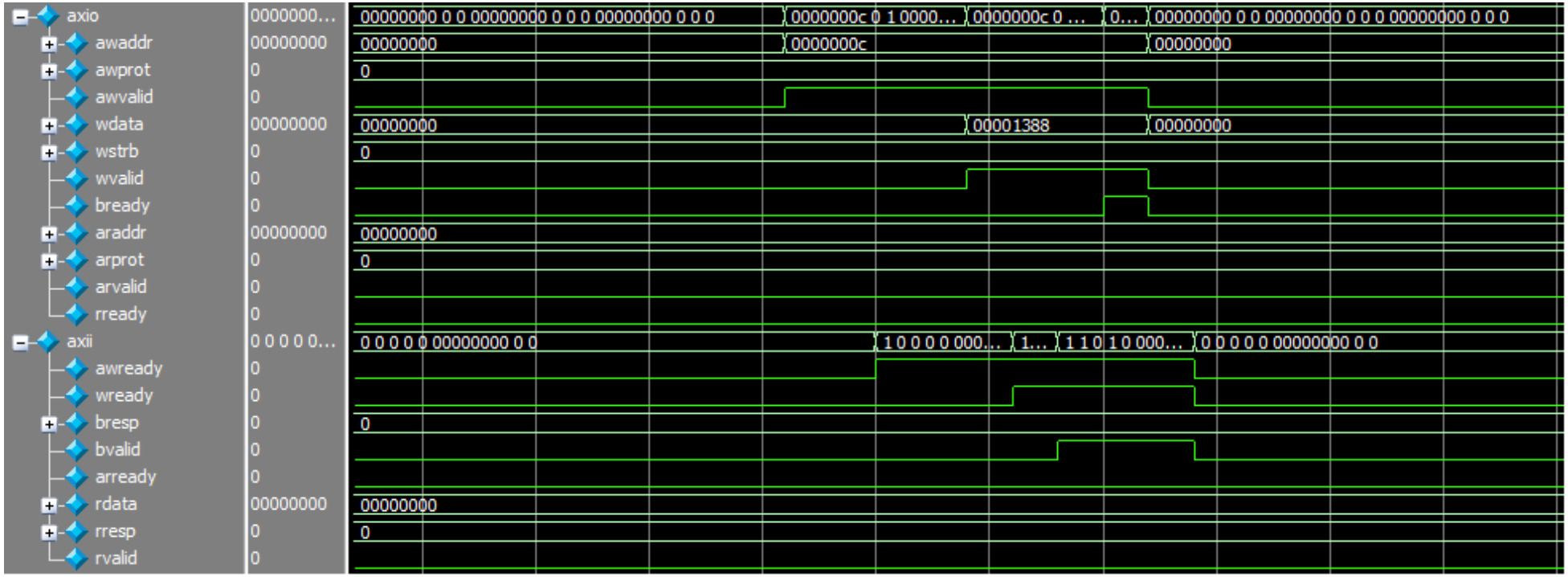


Рисунок 24 Осцилограмма AXI-lite транзакции

На рис.25. 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.

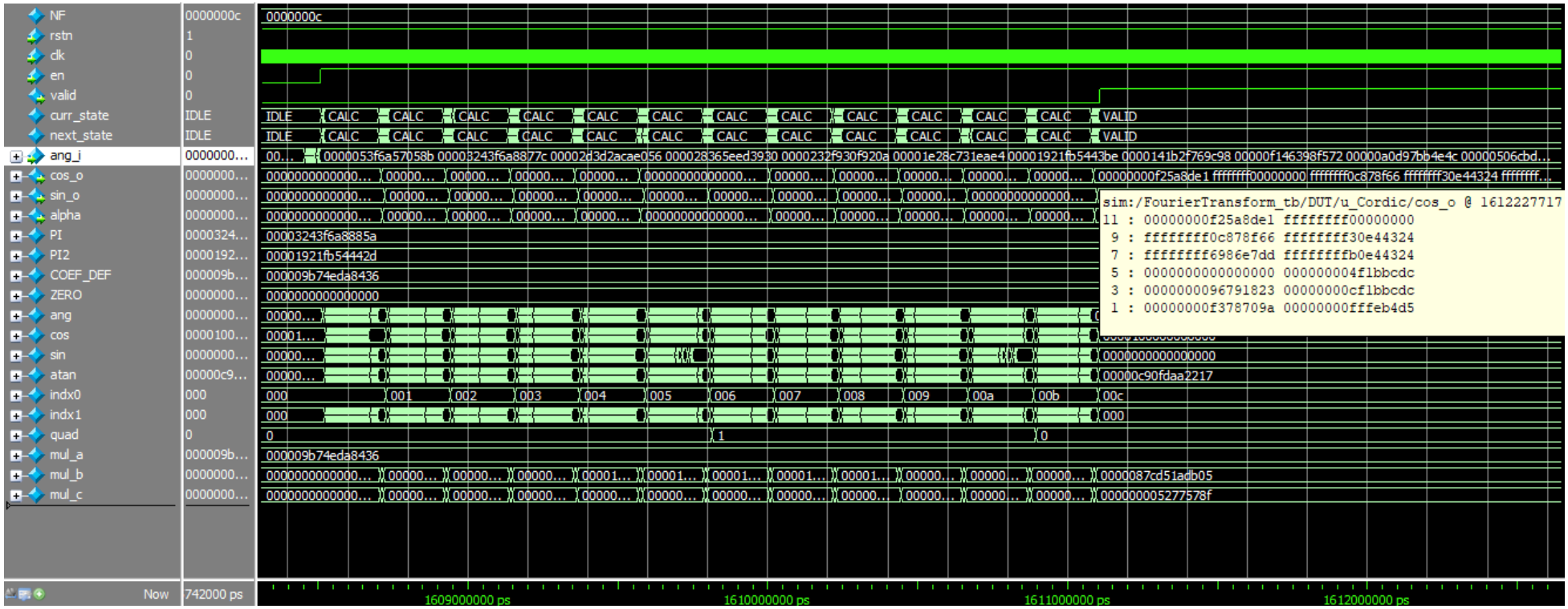


Рисунок 25 Осцилограмма модуля Cordic

На рис.26. 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.

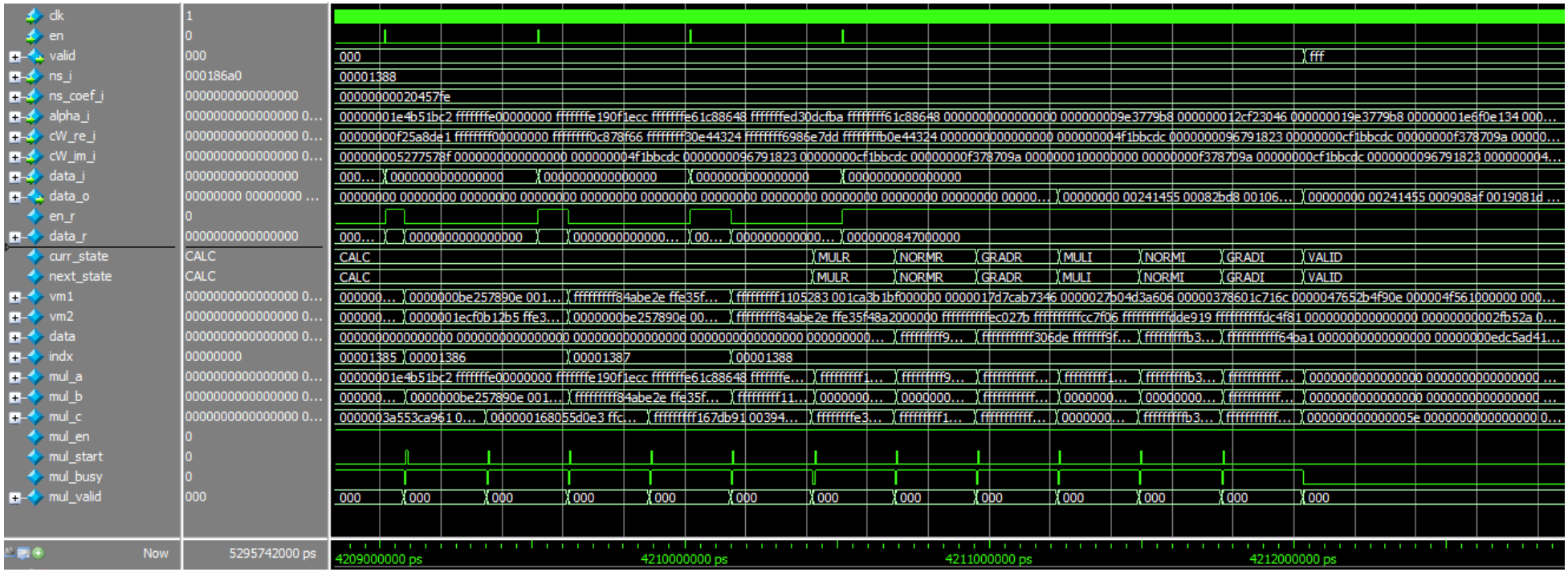


Рисунок 26 Осцилограмма модуля Herzel

8. Приложение 3

8.1. АЦП

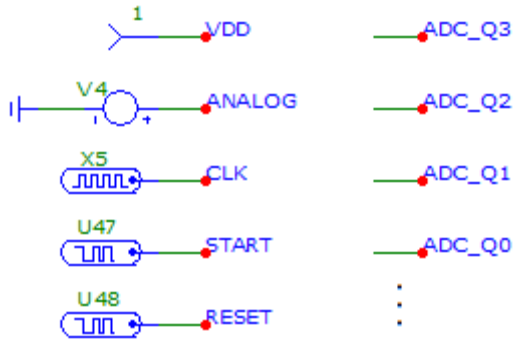


Рисунок 27 Интерфейс АЦП модуля

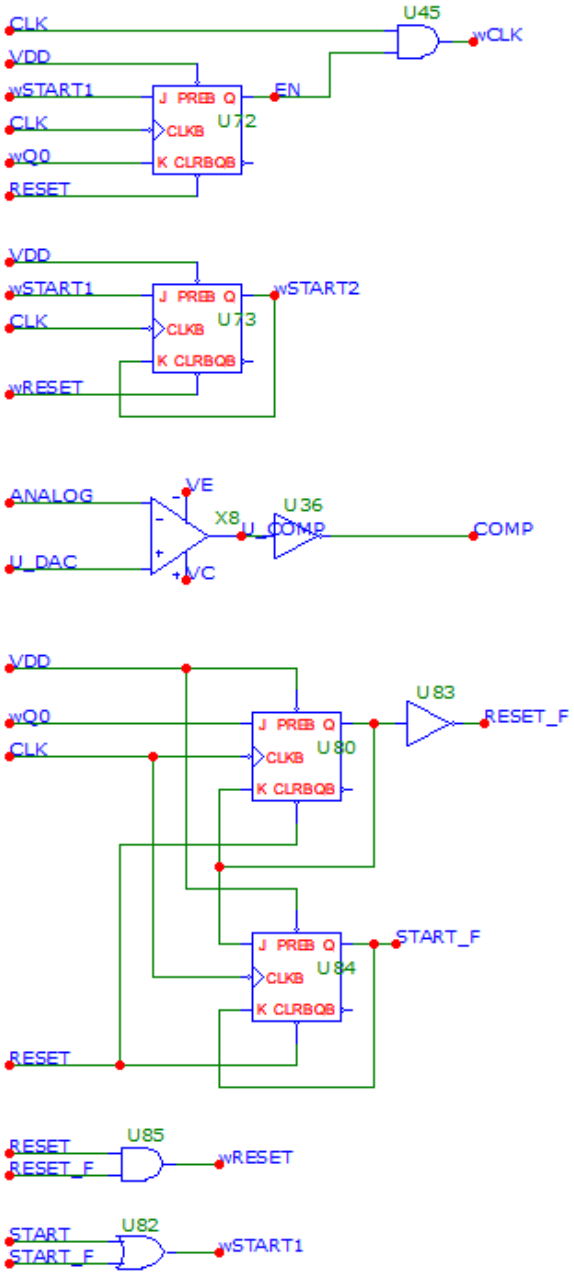


Рисунок 28 Схема управления

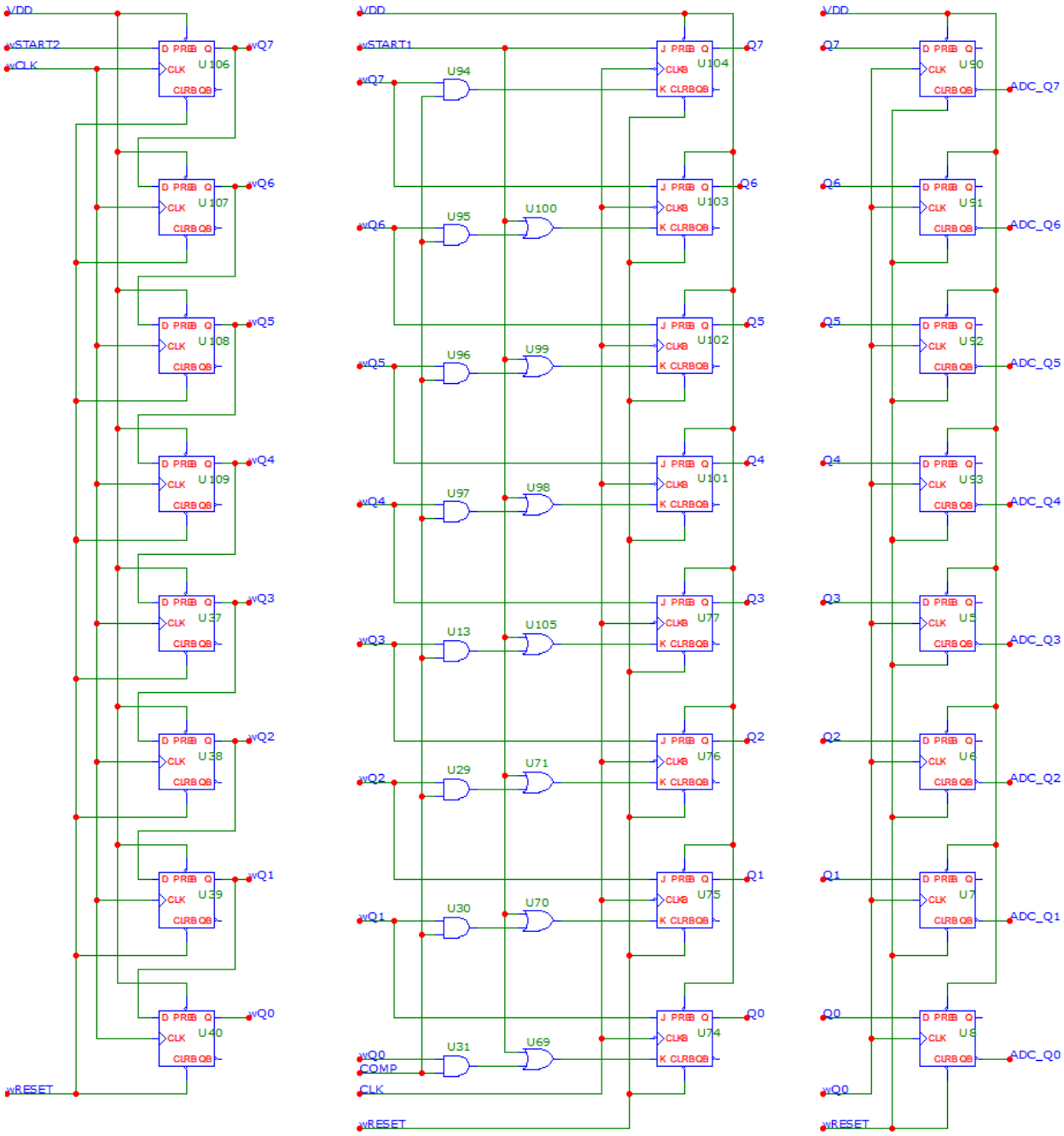


Рисунок 29 Слева на право: сдвиговый регистр, схема изменения кода, выходной регистр

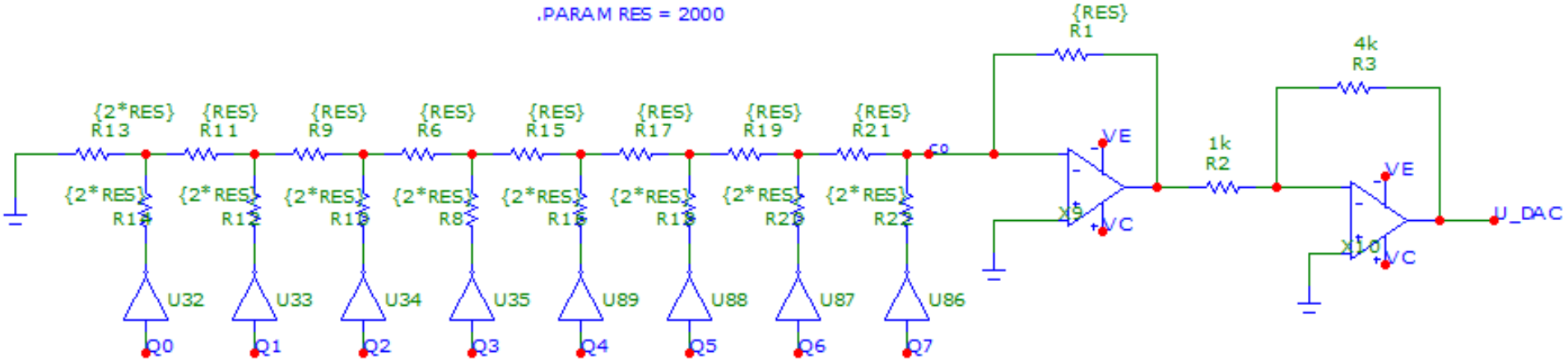
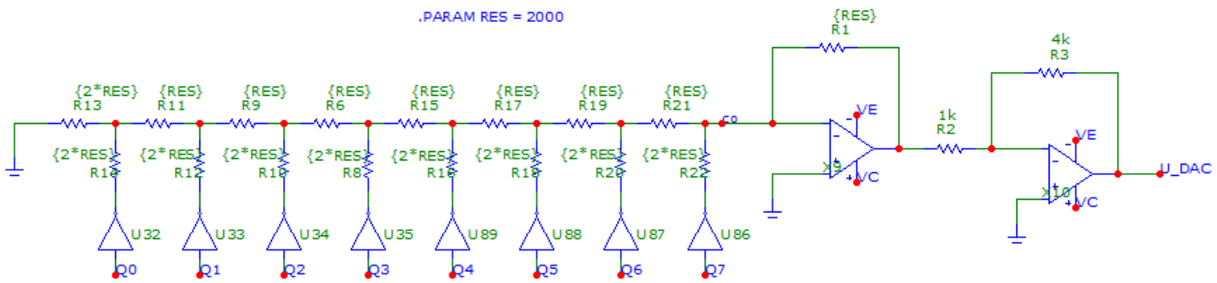
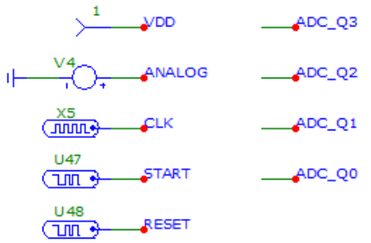
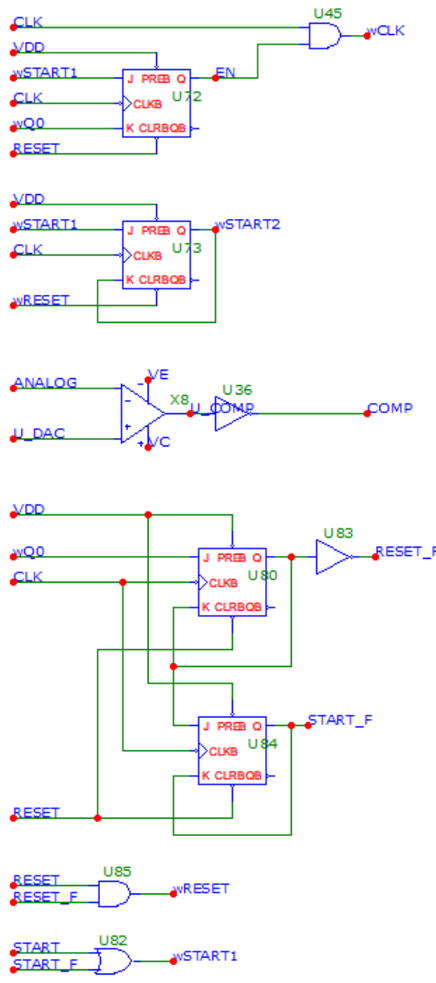


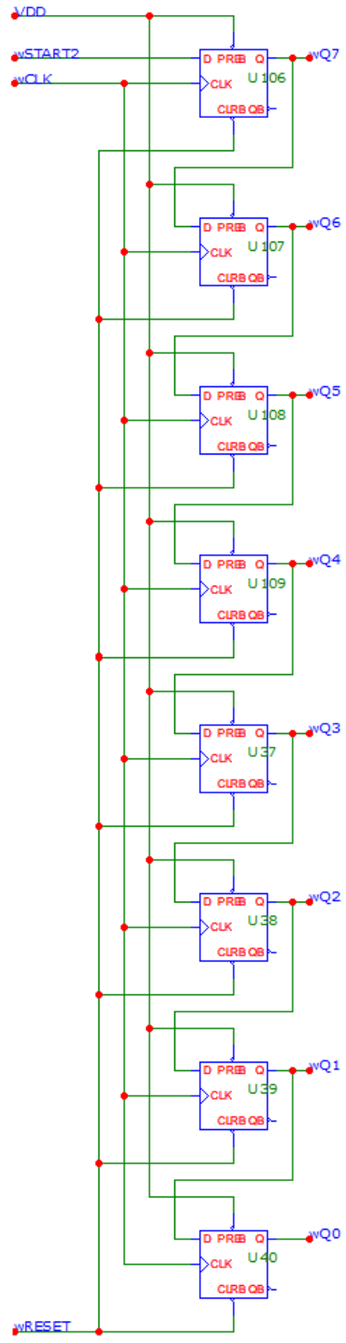
Рисунок 30 Схема ЦАП

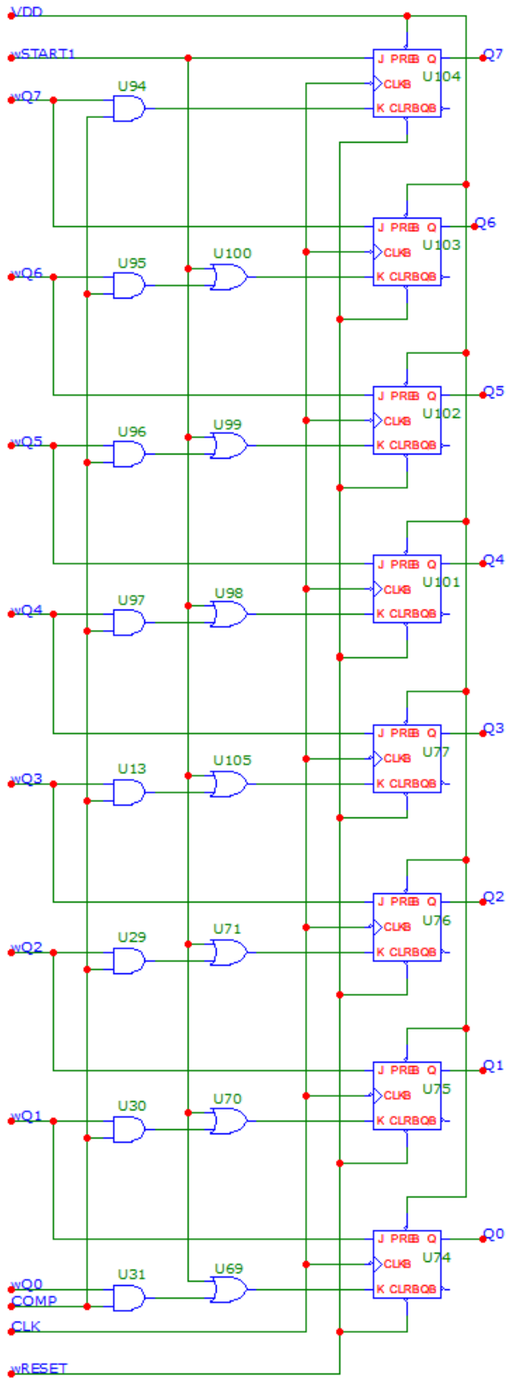

.PARAM RES = 2000
{RES}
R1
4k
R3
1k
R2
{2*RES} {RES} {RES} {RES} {RES} {RES} {RES} {RES}
R13 R11 R9 R6 R15 R17 R19 R21
{2*RES} {2*RES} {2*RES} {2*RES} {2*RES} {2*RES} {2*RES} {2*RES}
R14 R12 R10 R8 R16 R18 R20 R22
U32 U33 U34 U35 U89 U88 U87 U86
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7
VE
VC
X5
X10
U_DAC

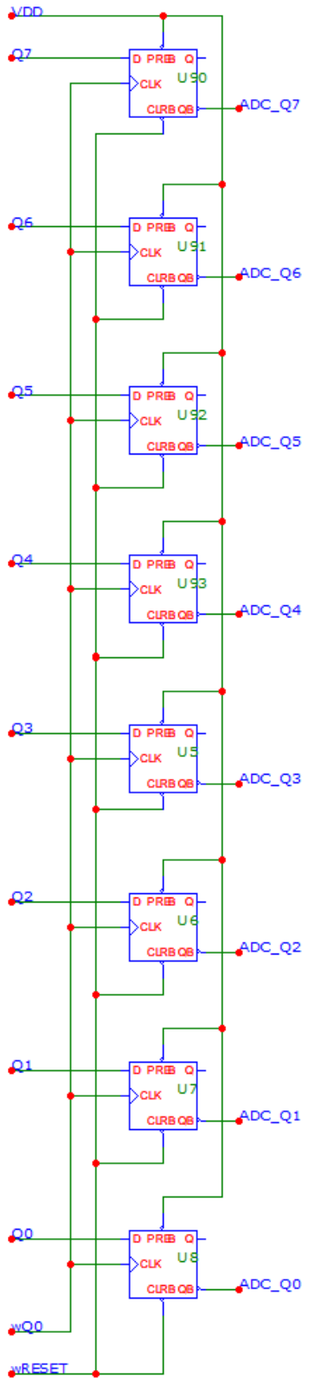

1
VDD
V4
ANALOG
X5
CLK
U47
START
U48
RESET

ADC_Q3
ADC_Q2
ADC_Q1
ADC_Q0


CLK
VDD
wSTART1
CLK
wQ0
RESET
U45
U72
EN
wCLK
VDD
wSTART1
CLK
wRESET
U73
wSTART2
ANALOG
U_DAC
VE
VC
X8
U_COMP
U36
COMP
VDD
wQ0
CLK
RESET
U80
U83
RESET_F
START_F
U84
RESET
RESET
RESET_F
U85
wRESET
START
START_F
U82
wSTART1


VDD
wSTART2
wCLK
U106
U107
U108
U109
U37
U38
U39
U40
wQ7
wQ6
wQ5
wQ4
wQ3
wQ2
wQ1
wQ0
wRESET


VDD
wSTART1
wQ7
wQ6
wQ5
wQ4
wQ3
wQ2
wQ1
wQ0
COMP
CLK
wRESET
U94
U95
U96
U97
U13
U29
U30
U31
U100
U99
U98
U105
U71
U70
U69
U104
U103
U102
U101
U77
U76
U75
U74
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0


VDD
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
wRESET
U90
U91
U92
U93
U5
U6
U7
U8
ADC_Q7
ADC_Q6
ADC_Q5
ADC_Q4
ADC_Q3
ADC_Q2
ADC_Q1
ADC_Q0

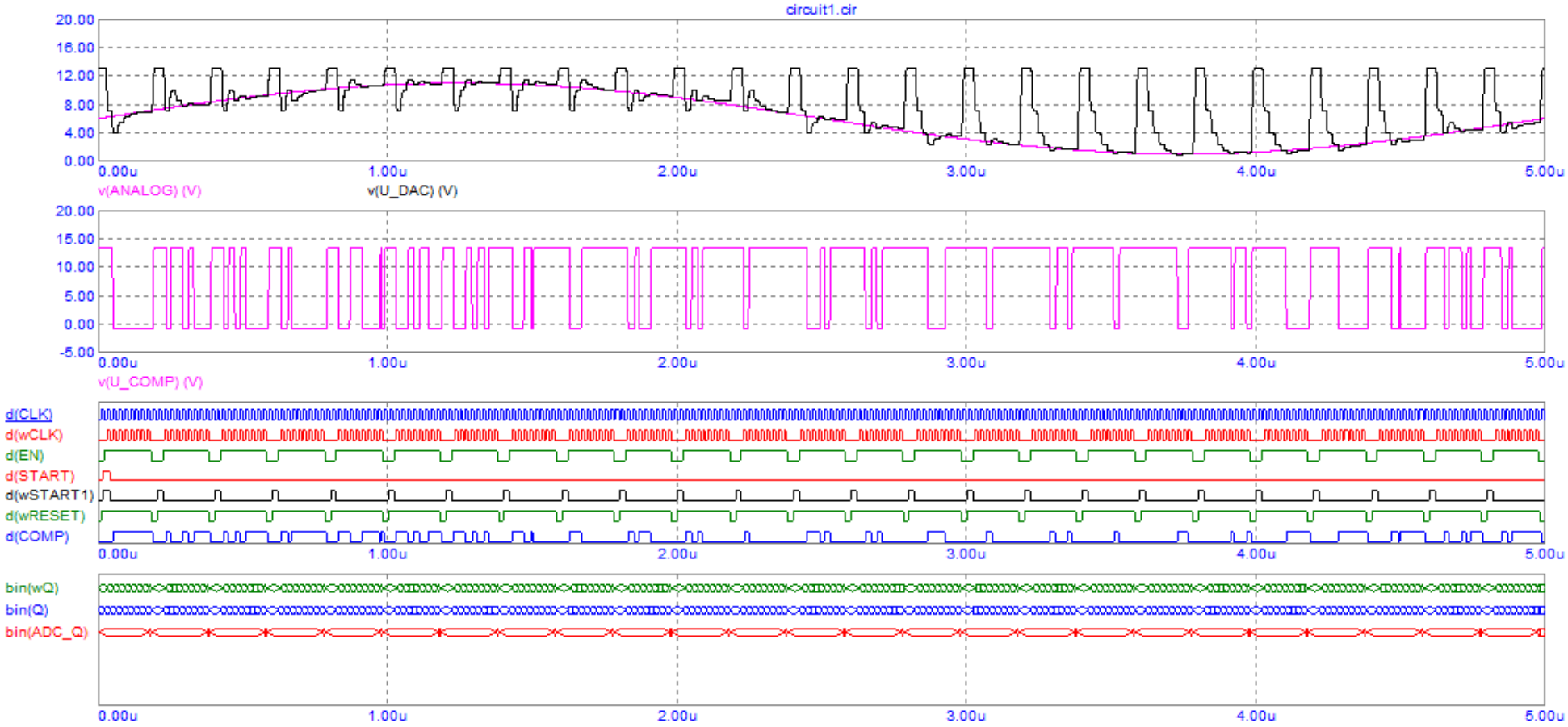


Рисунок 31 Осцилограмма

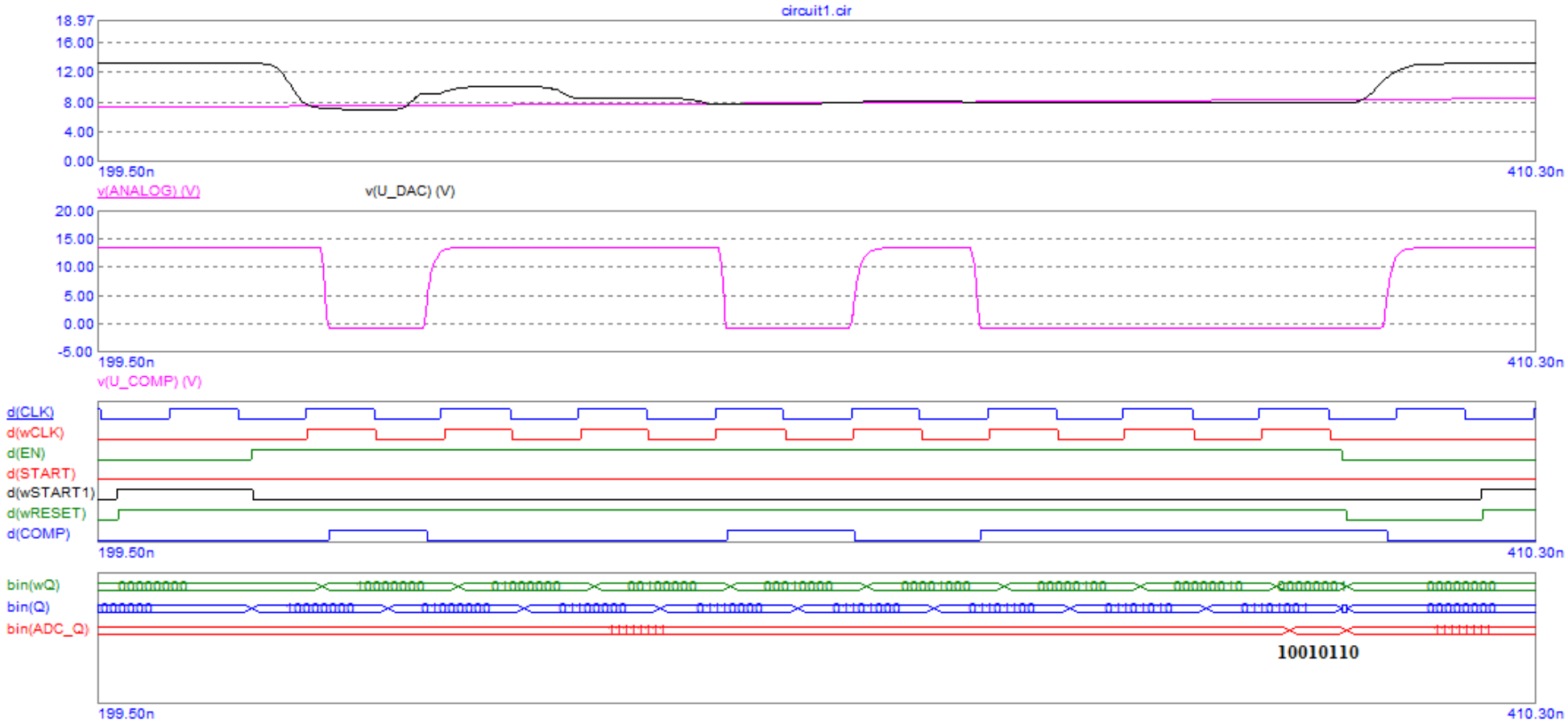


Рисунок 32 Осцилограмма

8.2. LVDS

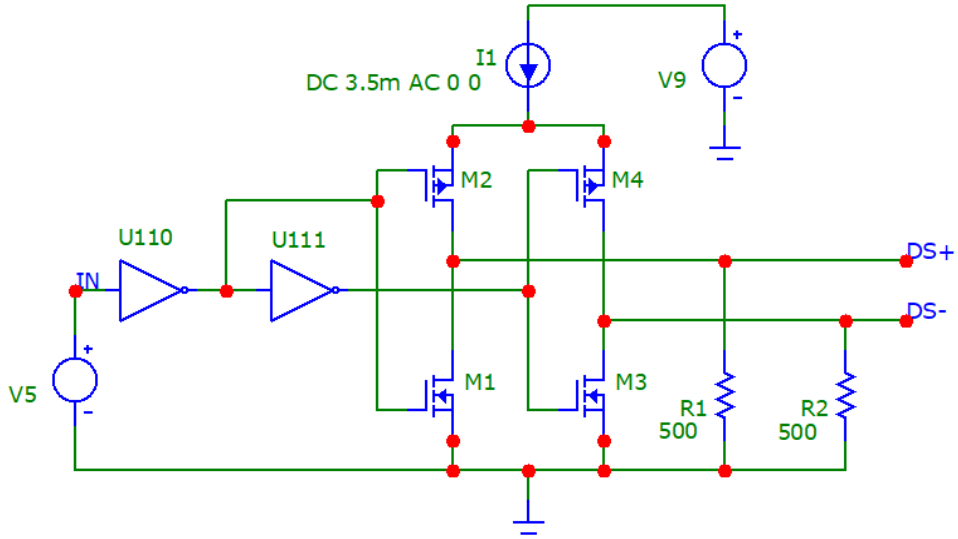


Рисунок 33 Простейшая схема LVDS драйвера

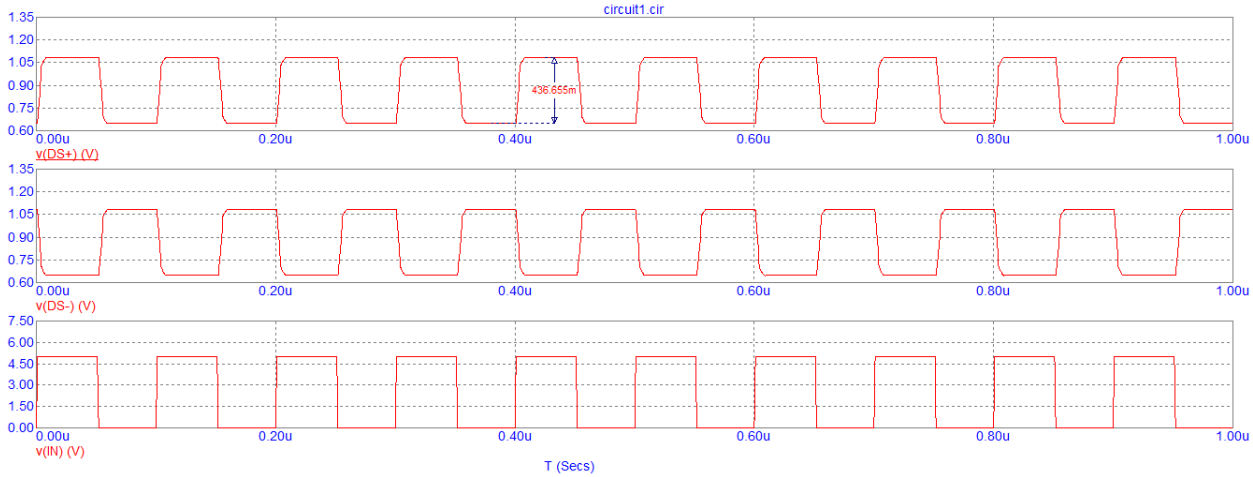


Рисунок 34 Моделирование драйвера LVDS интерфейса

8.3. Модуль преобразования Фурье

8.3.1. FourierTransform

```
import axi_pkg::*;
module FourierTransform #(
  parameter NF = 11 // NUM_FREQ
) (
  // 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       enable    ;
logic [7:0] sample    ;
logic       enable_syn;
logic [7:0] sample_syn;
logic       rstn_syn  ;

logic mode       ;
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         [31:0] 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);

`ifndef TEST
  IBUFDS IBUFDS_inst
  (
     .I (enable_p),
     .IB(enable_n),
     .O (enable  )
  );
  genvar gvar_buf;
  generate
    for (gvar_buf = 0; gvar_buf < 8; gvar_buf = gvar_buf + 1) begin :
IBUFDS_sample
      IBUFDS IBUFDS_inst
      (
        .I (sample_p[gvar_buf]),
        .IB(sample_n[gvar_buf]),
        .O (sample  [gvar_buf])
      );
    end
  endgenerate
`else
  assign enable = enable_p;
  genvar gvar_buf;
  generate
    for (gvar_buf = 0; gvar_buf < 8; gvar_buf = gvar_buf + 1) begin :
assign_sample
```

```
      assign sample[gvar_buf] = sample_p[gvar_buf];
    end
  endgenerate
`endif

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

resync_data #(
  .DW(1)
) u_resync_enable (
  .rstn  (rstn_syn  ),
  .clk   (clk       ),
  .data_i(enable    ),
  .data_o(enable_syn)
);

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

spi2axi_wrap u_spi2axi_wrap (
  .axi_rstn_i (rstn_syn),
  .axi_clk_i  (clk     ),
  .spi_sck_i  (spi_sck ),
  .spi_ss_n_i (spi_ss_n),
  .spi_mosi_i (spi_mosi),
  .spi_miso_o (spi_miso),
  .axio_o     (axio    ),
  .axii_i     (axii    )
);

HerzelRegs #(
  .NF(NF)
) u_HerzelRegs (
  .rstn          (rstn_syn    ),
  .clk           (clk         ),
```

```
  .freq_arr_o    (freq_arr    ),
  .en_cordic_o   (en_cordic   ),
  .valid_angel_i (valid_angel ),
  .valid_cordic_i(valid_cordic),
  .valid_herzel_i(valid_herzel),
  .data_arr_i    (data_hrz    ),
  .num_samp_o    (num_samp    ),
  .samp_freq_o   (samp_freq   ),
  .mode_o        (mode        ),
  .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        (clk      ),
  .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       (clk        ),
  .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  (clk         ),
  .en   (valid_angel ),
  .valid(valid_cordic),
  .ang_i(angel_arr   ),
  .cos_o(coefW_re    ),
  .sin_o(coefW_im    ),
  .alpha(alpha       )
);

assign en_scl = valid_cordic && enable_syn;
DataScale u_DataScale (
  .rstn  (rstn_all  ),
  .clk   (clk       ),
  .enable(en_scl    ),
  .valid (valid_scl ),
  .mode  (mode      ),
  .data_i(sample_syn),
  .data_o(data_scl  )
);

genvar gvar;
generate
  for (gvar = 0; gvar < NF; gvar = gvar + 1) begin : herzel
    Herzel #(
      .NF(NF)
    ) u_Herzel (
      .rstn     (rstn_h            ),
      .clk      (clk               ),
      .en       (valid_scl         ),
      .valid    (valid_herzel[gvar]),
      .ns_i     (num_samp          ),
      .ns_coef_i(ns_coef           ),
      .alpha_i  (alpha[gvar]       ),
      .cW_re_i  (coefW_re[gvar]    ),
      .cW_im_i  (coefW_im[gvar]    ),
      .data_i   (data_scl          ),
      .data_o   (data_hrz[gvar]    )
    );
  end
endgenerate
endmodule
```

8.3.2. IBUFDS

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

8.3.3. 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
```

8.3.1. 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
```

8.3.2. 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_sck_old <= 0;
    spi_sck_re  <= 0;
    spi_sck_fe  <= 0;
    spi_ss_old  <= 0;
    spi_ss_fe   <= 0;
  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 < 7) 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 if (spi_sck_re) 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
      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) begin : axi_next_state
  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;
      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
    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
    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
    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
    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

endmodule
```

8.3.3. 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                        mode_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;

logic bresp;

// addres map of regs and default value
logic [31:0] version   = 32'h2904_2023; // RW 0x0000_0000
logic [31:0] debug     = 32'hF0F0_F0F0; // RW 0x0000_0004
logic        mode      = 32'h0000_0000; // RW 0x0000_0008
logic [31:0] num_samp  = 32'h0001_86A0; // RW 0x0000_000C
logic [31:0] samp_freq = 32'h0003_0D40; // RW 0x0000_0010
logic [31:0] en_cordic = 32'h0000_0000; // RW 0x0000_0014
logic [31:0] status    = 32'h0000_0000; // R  0x0000_0018
logic        reset_all = 32'h0000_0000; // RW 0x0000_001C
logic        reset_h   = 32'h0000_0000; // RW 0x0000_0020

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

```
always_comb begin
  mode_o          = mode              ;
  num_samp_o      = num_samp          ;
  samp_freq_o     = samp_freq         ;
  en_cordic_o     = en_cordic[0]      ;
  status[0]       = valid_angel_i     ;
  status[1]       = valid_cordic_i    ;
  status[3]       = &valid_herzel_i   ;
  reset_all_o     = reset_all         ;
  reset_h_o       = reset_h           ;

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

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

state curr_state;
state next_state;

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

always_comb begin
  axii_o.awready = 0;
  axii_o.wready  = 0;
  axii_o.bvalid  = 0;
  axii_o.arready = 0;
  axii_o.rdata   = 0;
```

```
axii_o.rresp   = 0;
axii_o.rvalid  = 0;
bresp = 0;
case (curr_state)
  IDLE : begin
    axii_o.bresp   = 0;
    if (axio_i.arvalid)
      next_state = RADDR;
    else if (axio_i.awvalid)
      next_state = WADDR;
    else
      next_state = IDLE ;
  end
  RADDR: begin
    axii_o.arready = 1;
    next_state = RDATA;
  end
  RDATA: begin
    axii_o.rvalid = 1;
    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_0008: axii_o.rdata  = mode     ;
        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];
          bresp = 1;
        end
      end
      if (!bresp) begin
        axii_o.rresp = 2'h3;
      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];
        bresp = 1;
      end
    end
    if (!bresp) begin
      axii_o.rresp = 2'h3;
    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_0008: mode      = 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;
          bresp = 1;
        end
      end
      if (!bresp) begin
        axii_o.bresp = 2'h3;
      end
    end else begin
```

```
            axii_o.bresp = 2'h3;
          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

endmodule
```

8.3.4. 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
```

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

endmodule
```

8.3.5. 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, // (20.44)
  output logic signed [NF-1:0][63:0] sin_o, // (20.44)
  output logic signed [NF-1:0][63:0] alpha  // (20.44)
);

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'h00000_00000000000; // (20.44)

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;

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 @(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_00000000000;
    sin   <= 64'h00000_00000000000;
    indx0 <= 0;
    indx1 <= 0;
    quad  <= 0;
    mul_b <= 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
  next_state <= PRIVC;
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
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
  next_state <= PRIVS;
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
        next_state <= NEXT;
      end
      NEXT : begin
        if (indx0 < NF) begin
          cos   <= 64'h00001_00000000000;
          sin   <= 64'h00000_00000000000;
          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_0FFAADDB968;  //  atanh(2^(-4 ))
    5      : atan = 64'h00000_07FF556EEA6;  //  atanh(2^(-5 ))
    6      : atan = 64'h00000_03FFEAAB777;  //  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_000FFFFFFAB;  //  atanh(2^(-12))
    13     : atan = 64'h00000_0007FFFFFF5;  //  atanh(2^(-13))
    14     : atan = 64'h00000_0003FFFFFFF;  //  atanh(2^(-14))
```

```
    15      : atan = 64'h00000_00020000000;  //  atanh(2^(-15))
    16      : atan = 64'h00000_00010000000;  //  atanh(2^(-16))
    17      : atan = 64'h00000_00008000000;  //  atanh(2^(-17))
    18      : atan = 64'h00000_00004000000;  //  atanh(2^(-18))
    19      : atan = 64'h00000_00002000000;  //  atanh(2^(-19))
    20      : atan = 64'h00000_00001000000;  //  atanh(2^(-20))
    21      : atan = 64'h00000_00000800000;  //  atanh(2^(-21))
    22      : atan = 64'h00000_00000400000;  //  atanh(2^(-22))
    23      : atan = 64'h00000_00000200000;  //  atanh(2^(-23))
    24      : atan = 64'h00000_00000100000;  //  atanh(2^(-24))
    25      : atan = 64'h00000_00000080000;  //  atanh(2^(-25))
    26      : atan = 64'h00000_00000040000;  //  atanh(2^(-26))
    27      : atan = 64'h00000_00000020000;  //  atanh(2^(-27))
    28      : atan = 64'h00000_00000010000;  //  atanh(2^(-28))
    29      : atan = 64'h00000_00000008000;  //  atanh(2^(-29))
    30      : atan = 64'h00000_00000004000;  //  atanh(2^(-30))
    31      : atan = 64'h00000_00000002000;  //  atanh(2^(-31))
    32      : atan = 64'h00000_00000001000;  //  atanh(2^(-32))
    33      : atan = 64'h00000_00000000800;  //  atanh(2^(-33))
    34      : atan = 64'h00000_00000000400;  //  atanh(2^(-34))
    35      : atan = 64'h00000_00000000200;  //  atanh(2^(-35))
    36      : atan = 64'h00000_00000000100;  //  atanh(2^(-36))
    37      : atan = 64'h00000_00000000080;  //  atanh(2^(-37))
    38      : atan = 64'h00000_00000000040;  //  atanh(2^(-38))
    39      : atan = 64'h00000_00000000020;  //  atanh(2^(-39))
    40      : atan = 64'h00000_00000000010;  //  atanh(2^(-40))
    41      : atan = 64'h00000_00000000008;  //  atanh(2^(-41))
    42      : atan = 64'h00000_00000000004;  //  atanh(2^(-42))
    43      : atan = 64'h00000_00000000002;  //  atanh(2^(-43))
    44      : atan = 64'h00000_00000000001;  //  atanh(2^(-44))
    default: atan = 64'h00000_00000000000;
  endcase
end

endmodule
```

8.3.6. DataScale

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

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

logic enable_syn;
logic enable_old;
logic enable_re ;

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

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 || mode) begin
      valid  <= enable           ;
      data_m <= SCALE_COEF * data;
    end
    else begin
      valid  <= 0;
      data_m <= 1;
    end

  end
end

endmodule
```

8.3.7. Herzel

```
module Herzel #(
  parameter NF = 11
)(
  // CLK&RST
  input                          rstn     ,
  input                          clk      ,
  // CTRL
  input                          en       ,
  output logic                   valid    ,
  // DATA
  input                  [31:0] ns_i     , // (32.0 )
  input                  [63:0] ns_coef_i, // (20.44)
  input           signed [63:0] alpha_i  , // (20.44)
  input           signed [63:0] cW_re_i  , // (20.44)
  input           signed [63:0] cW_im_i  , // (20.44)
  input           signed [31:0] data_i   , // ( 8.24)
  output logic unsigned [31:0] data_o     // (16.16)
);

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

state curr_state;
state next_state;

logic signed [63:0] ns_coef  ; // (32.32)
logic signed [63:0] alpha    ; // (32.32)
logic signed [63:0] cW_re    ; // (32.32)
logic signed [63:0] cW_im    ; // (32.32)
logic signed [63:0] data     ; // (32.32)
logic signed [63:0] vm1      ; // (32.32)
logic signed [63:0] vm2      ; // (32.32)
logic signed [63:0] tmp      ; // (32.32)
logic signed [63:0] vm1_cW_re; // (32.32)
logic signed [63:0] vm1_cW_im; // (32.32)
```

```
logic signed [63:0] data_re  ; // (32.32)
logic signed [63:0] data_im  ; // (32.32)
logic        [31:0] indx     ; // (32.32)

assign ns_coef = {{13{ns_coef_i[63]}}, ns_coef_i[62:12]};
assign alpha   = {{13{alpha_i[63]}}, alpha_i[62:12]};
assign cW_re   = {{13{cW_re_i[63]}}, cW_re_i[62:12]};
assign cW_im   = {{13{cW_im_i[63]}}, cW_im_i[62:12]};
assign data    = {{25{data_i[31]}}, data_i[30:0], {8{1'b0}}};

logic signed [63 :0] mul_a; // (32.32)
logic signed [63 :0] mul_b; // (32.32)
logic signed [63 :0] mul_c; // (32.32)

mult_sign #(
  .DW    (64),
  .INT1_I(32),
  .INT2_I(32),
  .INT3_O(32)
) u_mult_sign (
  .a_in (mul_a),
  .b_in (mul_b),
  .c_out(mul_c),
  .c_ful(     )
);

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
    valid      <= 0;
    data_o     <= 0;
    vm1        <= 0;
    vm1_cW_re  <= 0;
    vm1_cW_im  <= 0;
    data_re    <= 0;
    data_im    <= 0;
    indx       <= 0;
```

```
  next_state <= CALC;
end
else begin
  case (curr_state)
    CALC : begin
      if (en) begin
        if (indx < ns_i) begin
          if (indx == 0) begin
            vm1 <= data;
          end
          else if (indx == 1) begin
            vm1 <= data + mul_c;
          end
          else begin
            vm1 <= data + mul_c - vm2;
          end
          indx <= indx + 1;
        end
      end
      if (indx < ns_i - 1)
        next_state <= CALC;
      else
        next_state <= MULR;
    end
    MULR : begin
      vm1_cW_re <= mul_c - vm2;
      next_state <= MULI;
    end
    MULI : begin
      vm1_cW_im <= mul_c;
      next_state <= NORMR;
    end
    NORMR : begin
      data_re <= mul_c;
      next_state <= NORMI;
    end
    NORMI : begin
      data_im <= mul_c;
      next_state <= GRADR;
    end
    GRADR : begin
      data_o <= mul_c[47:16];
      next_state <= GRADI;
    end
    GRADI : begin
```

```
        data_o <= data_o + mul_c[47:16];
        next_state <= VALID;
      end
      VALID : begin
        valid  <= 1;
        next_state <= VALID;
      end
      default: next_state <= curr_state;
    endcase
  end
end

always_ff @(negedge clk, negedge rstn) begin
    if (!rstn) begin
    vm2        <= 0;
    tmp        <= 0;
  end
  else begin
    case (curr_state)
      CALC : begin
        if (en) begin
          tmp <= vm1;
          vm2 <= tmp;
        end
      end
      default:;
    endcase
  end
end

always_comb begin
  case (curr_state)
    CALC : begin
      mul_a = alpha;
      mul_b = vm1  ;
    end
    MULR : begin
      mul_a = vm1  ;
      mul_b = cW_re;
    end
    MULI : begin
      mul_a = vm1  ;
      mul_b = cW_im;
    end
    NORMR : begin
```

```
      mul_a = vm1_cW_re;
      mul_b = ns_coef;
    end
    NORMI : begin
      mul_a = vm1_cW_im;
      mul_b = ns_coef;
    end
    GRADR : begin
      mul_a = data_re;
      mul_b = data_re;
    end
    GRADI : begin
      mul_a = data_im;
      mul_b = data_im;
    end
    default: begin
      mul_a = 0;
      mul_b = 0;
    end
  endcase
end

endmodule
```

8.3.8. 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
```

9. 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).