Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт цифровых технологий, электроники и физики

Кафедра вычислительной техники и электроники (ВТиЭ)

УДК 519.688

Работа защищена
«___» _____ 2024 г.
Оценка _____
Председатель ГЭК, д.т.н., проф.
_____ С. П. Пронин

Допустить к защите
«___» _____ 2024 г.
Заведующий кафедрой ВТиЭ,
к.ф.-м.н., доцент
_____ В. В. Пашнев

ПРОГРАММНЫЙ ГЕНЕРАТОР СИГНАЛОВ НА ОСНОВЕ МИКРОКОНТРОЛЛЕРА STM32

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

БР 09.03.01.506.294 ПЗ

Студент группы 506 Д. С. Вебер

Руководитель работы ст. преп. П. Н. Уланов

Консультант:

Нормоконтролер к.ф.-м.н., доцент А. В. Калачёв

Барнаул 2024

РЕФЕРАТ

Объем работы листов 77

Количество рисунков 54

Количество используемых источников 37

Количество таблиц 4

ГЕНЕРАТОР СИГНАЛОВ, МИКРОКОНТРОЛЛЕР, ЦИФРО-АНАЛОГОВЫЙ ПРЕОБРАЗОВАТЕЛЬ, ПРЯМОЙ ЦИФРОВОЙ СИНТЕЗ, МАКЕТ.

Данная выпускная квалификационная работа посвящена разработке программного генератора сигналов. В рамках работы были исследованы теоретические основы и методологии генерации сигналов, проведён обзор аналогичных генераторов сигналов, рассмотрены микроконтроллеры для реализации.

Разработан макет программного генератора сигналов. Генератор способен генерировать различные формы сигналов (синусоидальная, треугольная, прямоугольная, пилообразная, обратно пилообразная) с частотой от 125 до 50 000 Гц, амплитудой 3 В и шагом по частоте 125, 250, 500, 1000 Гц. Для управления генератором используются пять кнопок, а информация о параметрах сигнала отображается на экране с разрешением 128x64 пикселя через интерфейс I2C.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В ходе эксплуатации электронных устройств регулярно возникает необходимость в настройке. На входы устройства принимают сигналы, форма которых задается напряжением. Для тестирования и отладки могут понадобиться как цифровые, так и аналоговые сигналы разной формы. Формирование аналоговых сигналов может обеспечить специализированное устройство — генератор сигналов.

Генератор сигналов — это неотъемлемый инструмент для любого специалиста в области электроники. На сегодняшний день разрабатывается достаточно много генераторов сигналов, но не все генераторы, которые есть на рынке, обладают компактными размерами, лёгкостью транспортировки и доступностью в цене.

Ранее практически все лабораторные генераторы были аналоговыми и конструировались на различных схемах. К их достоинствам можно отнести простоту и надёжность, но у них есть существенные недостатки в виде меньшей стабильности и более тщательной настройке. Сейчас практически все генераторы, которые есть на рынке создаются на основе цифровых методов синтеза аналоговых сигналов, т. к. они стабильные и точные. Такого рода генераторы могут найти применение и в промышленности, но не всем пользователям требуются такие высокие характеристики. Разработанный в данной работе генератор претендует на применение в домашней лаборатории в качестве простого и функционального дешёвого генератора сигналов.

Применением такого генератора может быть генерация сигналов разных форм, работа с аналоговыми системами для исследования влияния сигналов на них, изучение методов обработки сигнала или основ электроники.

Цель выпускной квалификационной работы состоит в разработке программного генератора сигналов на микроконтроллере.

Задачи

1. Исследовать методы генерации сигналов и осуществить выбор;

2. Рассмотреть семейства микроконтроллеров и осуществить выбор;
3. Выбрать среду разработки;
4. Разработать программу;
5. Спроектировать устройство;
6. Протестировать генератор.

1. ТЕОРИЯ И МЕТОДОЛОГИЯ ГЕНЕРАЦИИ СИГНАЛОВ

1.1. Развитие генераторов сигналов

История развития генераторов сигналов начинается с аналоговых устройств, которые использовались для генерации различных форм сигналов, включая низкочастотные, высокочастотные, сверхвысокочастотные и импульсные. Во времена СССР разрабатывалось большое количество аналоговых генераторов сигналов [1]. Однако, с развитием технологий и потребностями в более сложных и модулируемых сигналах, стала очевидна необходимость в усовершенствовании источников сигнала.

В результате развития технологий появились цифровые генераторы на основе прямого цифрового синтеза частот и форм сигналов. Цифровые генераторы сигналов используют минимальное количество аналоговой элементной базы и основываются на специализированных сверхскоростных цифровых микросхемах, а также аналого-цифровых (АЦП) и цифро-аналоговых (ЦАП) преобразователях. Благодаря этому, интеграция данных генераторов с цифровыми системами становится легкой и позволяет открыть массу перспектив их использования в процессе тестирования и наладки разнообразных электронных и радиотехнических устройств.

В современной измерительной технике генераторы сигналов играют ключевую роль во многих сферах. В целом, история развития генераторов сигналов отражает эволюцию технологий, потребностей в модулируемых сигналах и влияние глобальных изменений в науке и технике.

1.2. Основные типы сигналов

Для начала стоит дать определение, что такое сигнал. Сигнал — это носитель информации. Он является переносчиком знаков, которые вместе образуют основу информации для передачи сообщения [1]. Исходя из этого можно сделать вывод, что постоянные токи и напряжения сигналами не являются, т.к. их параметры во времени не меняются, но впрочем их можно отнести к

простейшим сигналам, которые несут в себе информацию о полярности величины. В качестве сигналов они конечно не используются, но с помощью них можно задавать смещение сигналам.

Рассмотрим некоторые распространённые типы сигналов. Именно синусоидальные сигналы мы извлекаем из розетки. Математическое выражение, описывающее синусоидальное напряжение, имеет вид:

$$U = Asin2\pi ft, \tag{1.1}$$

где A — амплитуда сигнала,

f — частота в герцах.

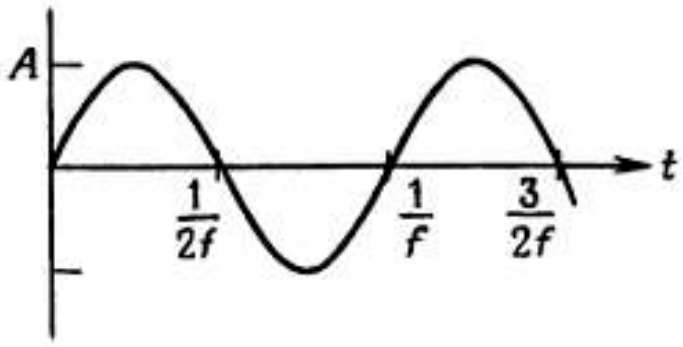


Рис. 1.1 Синусоидальный сигнал.

Эффективное значение равняется двойной амплитуде, то есть размаху сигнала.

Если нужно переместить начало координат ($t = 0$) в какой-то момент времени, то в формулу следует добавить фазу:

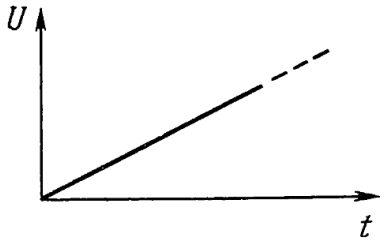
$$U = Asin2\pi ft + \theta. \tag{1.2}$$

Синусоидальные сигналы характеризуются тремя параметрами:

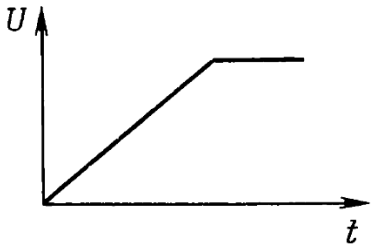
- U_M или I_M — амплитуда переменного напряжения или тока;
- f — частота (период);
- θ — фазовый сдвиг.

У синусоиды есть своё достоинство в том, что функция данной формы сигнала является решением многих дифференциальных уравнений, которые описывают как физические явления, так и свойства линейных цепей [2]. На практике поведение схемы оценивают по её амплитудно-частотной характеристике (АЧХ), которая показывает, как изменяется амплитуда синусоидального сигнала в зависимости от частоты. Для примера на усилителе звуковых частот амплитудно-частотная характеристика в идеале имеет ровную линию в диапазоне от 20 Гц до 20 кГц. Чаще всего частоты, с которыми приходится работать на синусоидальном сигнале, лежат в диапазоне от нескольких герц до нескольких мегагерц.

Линейно-меняющийся сигнал — это напряжение, возрастающее (или убывающее) с постоянной скоростью.



(а) Возрастающее напряжение в виде сигнала.



(б) Ограниченный сигнал.

Рис. 1.2 Линейно-меняющийся сигнал.

Напряжение не может, конечно, расти бесконечно. Поэтому обычно данная величина имеет конечное значение (рис. 1.2 (б)) или сигнал становиться пилообразным (рис. 1.3).

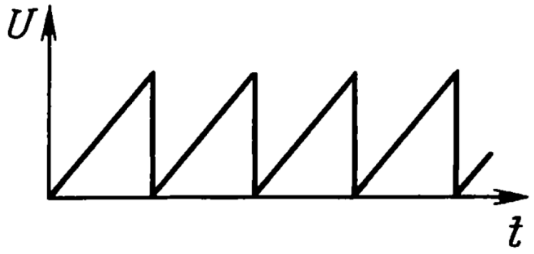


Рис. 1.3 Пилообразный сигнал.

Треугольный сигнал очень похож на линейно-меняющийся, но его отличие в том, что он симметричный.

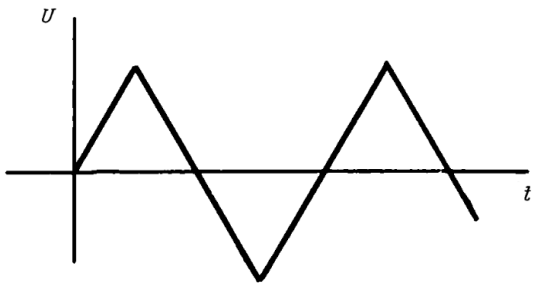


Рис. 1.4 Треугольный сигнал.

Прямоугольный сигнал или как его ещё называют меандр, характеризуется так же как и синусоидальный сигнал частотой и амплитудой.

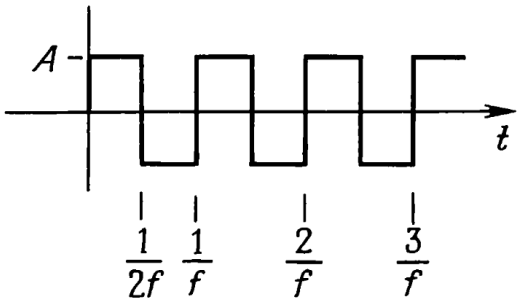


Рис. 1.5 Прямоугольный сигнал.

Эффективным значением для данного сигнала является значение его амплитуды. На самом деле прямоугольный сигнал не идеален. Его форма отличается от прямоугольника, т.к. присутствует время нарастания t_H, которое может быть от нескольких наносекунд до нескольких микросекунд [2].

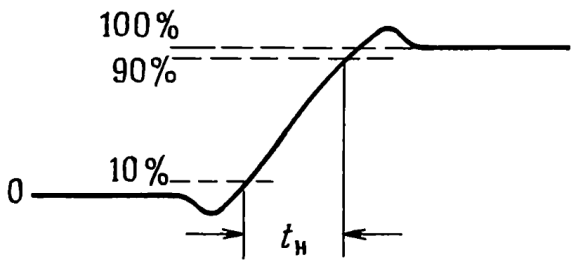


Рис. 1.6 Время нарастания скачка прямоугольного сигнала.

На рисунке 1.6 изображено как обычно выглядит скачок сигнала прямоугольника. Время когда сигнал нарастёт определяется в промежутке от 10 до 90% максимальной амплитуды сигнала.

Сигналы в виде импульса изображены на рисунке 1.7.

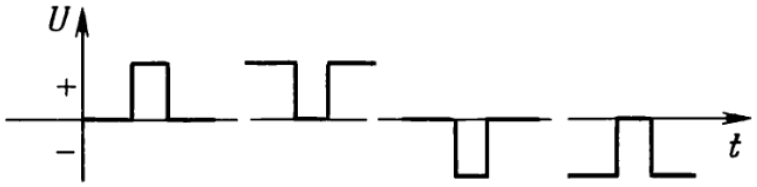


Рис. 1.7 Импульсы.

Данный вид сигналов характеризуется амплитудой и длительностью импульса. Можно генерировать последовательность периодических импульсов и тогда можно ещё характеризовать сигнал частотой (повторением импульса). У импульсов есть полярность — положительная и отрицательная. Кроме этого импульс может спадать, а может нарастать.

1.3. Виды генераторов

Источник сигнала часто является неотъемлемой частью схемы, но для тестирования работы удобно иметь отдельный, независимый источник сигнала. В качестве такого источника могут использоваться следующие виды генераторов.

1. Генераторы синусоидальных сигналов;
2. Функциональные генераторы;
3. Генераторы сигналов произвольной формы;
4. Генераторы импульсов.

1.3.1. Генераторы синусоидальных сигналов

Генераторы таких сигналов широко применяются при тестировании различных радиоэлектронных устройств. Сами же синусоидальные сигналы являются простейшими. Они изменяются во времени, но их параметры —

амплитуда, частота и фаза остаются постоянными [1]. Изменяя эти параметры, возможно осуществить модуляцию синусоидальных сигналов и использовать их для переноса информации. На таком принципе построены разнообразные области применения синусоидальных сигналов в радиотехнике.

В области измерительных приборов существуют различные виды генераторов синусоидального напряжения. Одни из них схемы на RC-цепи для генерации низкочастотных сигналов и на основе LC-контуров для высокочастотных сигналов, далее конструировались схемы на основе разных типов резонаторов, но всё это уже прошлый век и на данный момент генераторы синусоидального сигнала строятся на основе цифровых методах синтеза.

1.3.2. Функциональные генераторы

Функциональными генераторами обычно называют генераторы, которые могут создавать несколько функциональных зависимостей. Данные устройства генерируют сигналы разной формы. Их простота и плавная регулировка частоты в большом диапазоне привела к массовому применению генераторов такого типа. Из всех генераторов, генераторы функций являются очень гибкими. Они позволяют генерировать синусоидальные, треугольные и прямоугольные сигналы в широком спектре частот. Благодаря такому разнообразию сигналов, сфера применения таких генераторов сильно расширяется. Данный вид источника сигнала может быть одним на все случаи жизни. Их можно использовать для тестирования, исследования и отладки абсолютно разной электронной аппаратуры.

Функциональные генераторы также существуют как аналоговые так и цифровые, но в настоящее время аналоговые неактуальны. Переход к функциональным генераторам с цифровым синтезов выходных сигналов и цифровой элементной базой связан с растущими требованиями к сигналам источника [1]. У сигнала должна быть стабильная частота с амплитудой и верная форма. Благодаря применению цифровых элементов в массовой продукции (персональный компьютер, мобильный телефон), цифровые интеграль-

ные схемы стали бурно развиваться. Стала повышаться функциональность схем и понижаться их стоимость.

1.3.3. Генераторы сигналов произвольной формы

Данный вид генератора дополняет функциональный генератор. Достаточно новое направление в генераторах сигналов, которое основывается на прямом цифровом синтезе различных сигналов, по сути произвольных форм. Прямой цифровой синтез открыл возможность воплотить как обилие стандартных функций, так и произвольных форм. Однако синтез сигналов произвольных форм неминуемо усложняет устройство. Ему нужно часто перезаписывать память и должен быть организован какой-нибудь редактор форм с отображением формы сигнала, чтобы строить сигнал по точкам [1]. Следовательно, генераторы такого типа относятся к достаточно сложным и дорогим приборам. И всё же в ряде случаев данный вид генератора сигналов бывает очень необходим. С ростом сложности многих сфер техники, увеличивается разнообразие форм сигналов.

1.3.4. Генераторы импульсов

Важно иногда передавать значительное количество энергии за короткий промежуток времени. Генерация импульсов необходима для тестирования и отладки импульсных систем. К примеру это может быть радиолокатор. В радиолокации импульс направляется в пространство затем отражается от достигнутой цели и воспринимается радиолокационным приёмником. Получив информацию о времени задержки отражённого сигнала, можно оценить расстояние до цели, а проанализировав отражённый импульс можно сделать какие-то выводы о характере цели [1]. Такого рода генераторы находят большое применение в качестве источников несинусоидальных сигналов.

1.4. Методы цифровой генерации сигнала

После рассмотрения видов генераторов сигналов можно сделать вывод о том, что способы получения сигнала также делятся на аналоговые и цифро-

вые. Однако, в настоящее время аналоговые генераторы неактуальны и изучать способы генерации и схемы на аналоговой элементной базе большого смысла не имеет. Следует провести исследование цифровых методов генерации сигнала.

1.4.1. Метод аппроксимации

Метод аппроксимации подразумевает собой вычисление отсчётов функции по заданным параметрам. В устройстве хранятся только параметры, определяющие генерируемый сигнал. Программа рассчитывает значения функции с определенным интервалом [3]. Исходя из этого, данный метод позволяет затратить небольшой объём памяти, но его недостаток это затраты на вычисления, что ограничивает максимальную частоту сигнала. Одним из видов аппроксимации является ступенчатая. Ступенчатая аппроксимация заключает в себе замену функции напряжением ступенчатой формы, которая будет мало отличаться. При данном виде аппроксимации напряжение разбивается по времени с определённым шагом. Интервал между двумя точками заменяется напряжением постоянного тока, то есть ступенька, высота которой означает аппроксимируемое напряжение в момент времени t [4]. В результате замены получим ступенчатую линию вместо кривой. Число ступенек при заданном периоде определяется шагом дискретизации $p = \dfrac{T}{\Delta t}$.

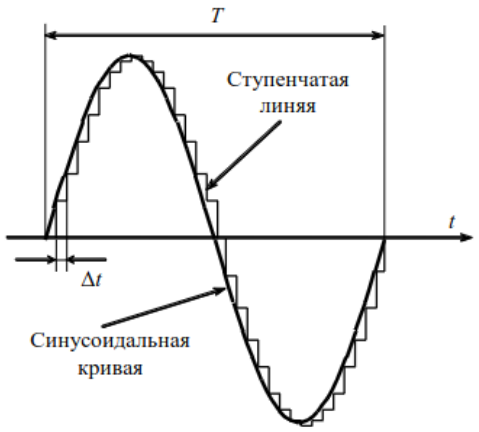


Рис. 1.8 Ступенчатая аппроксимация.

1.4.2. CORDIC

Следующий метод тоже предполагает вычисление отсчётов. Для генерации сигналов также применяется итерационный метод CORDIC. Аббревиатура расшифровывается как Coordinate Rotation Digital Computer, что означает цифровое вычисление поворота системы координат. Ещё этот алгоритм называют «цифра за цифрой». Он был разработан для аппаратного поворота вектора на плоскости [5; 6]. Для этого использовались простые операции сдвиг вправо и сложение или вычитание регистров. Смысл итерационного метода заключается в том, чтобы построить следующую последовательность: $y_{i+1} = f(y_i)$, сходящейся к функции $y(x)$ [7]. Математической моделью в данном методе является единичная окружность с парой векторов, исходящих из центра.

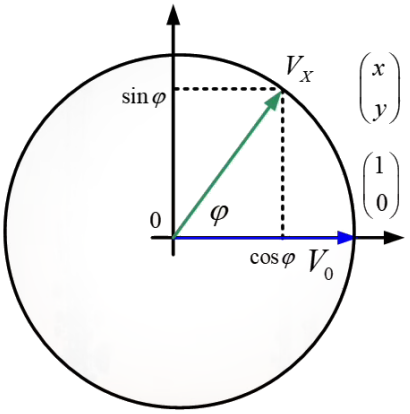


Рис. 1.9 Математическая модель CORDIC.

Вектор V_x отклонён от горизонтальной оси на угол являющимся аргументов функции. Второй вектор V_0 будет производить вращение от начальной точки относительно начала координат. Координаты векторов имеют значения sin и cos угла, на который вектор отклоняется от горизонтальной оси.

Для вектора V_0: $cos\ 0 = 1$, $sin\ 0 = 0$.

Для вектора V_x: $cos\ \phi = x$, $sin\ \phi = y$.

Необходимо найти координаты вектора V_x x и y после поворота на угол ϕ. Координаты вычисляются по тригонометрическим формулам:

$$x = x_0 * cos\ \phi - y_0 * sin\ \phi, \tag{1.3}$$

$$y = x_0 * sin\ \phi + y_0 * cos\ \phi. \tag{1.4}$$

Так как $tan\ \phi = \dfrac{sin\ \phi}{cos\ \phi}$, то можно выразить $sin\ \phi = tan\ \phi * cos\ \phi$ и выполнить преобразование формул. Тогда получим:

$$x = cos\ \phi(x_0 - y_0 * tan\ \phi), \tag{1.5}$$

$$y = cos\ \phi(y_0 + x_0 * tan\ \phi). \tag{1.6}$$

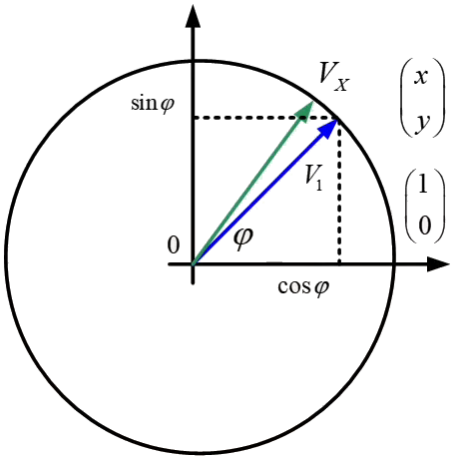


Рис. 1.10 Поворот вектора.

Если задавать такой угол поворота, что $tan\ \phi = \pm 2^{-i}$, где i — целое число, то умножение x_0 и y_0 сведётся к простому сдвигу их значений вправо на i разрядов, так как деление на 2 представляет из себя побитовый сдвиг числа право.

Произвольный угол можно представить в виде суммы углов:

$$\phi_i = \pm atan 2^{-i}, \tag{1.7}$$

где $i = 0, 1, 2$, и т.д.

Тогда операция поворота вектора будет состоять из последовательных простых поворотов. В каждой итерации проводятся следующие вычисления:

1. Направление поворота (1.8);
2. Значение координаты x (1.8);
3. Значение координаты y (1.10);
4. Отклонение вектора (1.11).

$$\sigma_i = sign(z_i). \tag{1.8}$$

$$x_{i+1} = x_i - \sigma_i * y_i * 2^{-i}. \tag{1.9}$$

$$y_{i+1} = y_i + \sigma_i * x_i * 2^{-i}. \tag{1.10}$$

$$z_{i+1} = z_i - \sigma_i * atan(z^{-i}). \tag{1.11}$$

Данный алгоритм применим для генерации синуса и его применение целесообразно только при необходимости быстродействия и высокой точности системы.

1.4.3. Табличный метод

В табличном методе генерации сигналов предполагается, что заранее вычисленные отсчёты хранятся в памяти. То есть никаких вычислений не требуется и генерация сводится к тому, что в порт цифро-аналогового преобразователя нужно вывести ячейку по заданному адресу [8]. Плюсом метода является то, что ему нужно меньше времени, чтобы сформировать отсчёт, т.к. он уже посчитан, следовательно, можно добиться более высокой частоты сигнала. Минусом же является необходимость хранения отсчётов, что может затратить объём памяти [3].

Частота сигнала будет зависеть от опорной частоты устройства.

$$f_{out} = \frac{f_{clk}}{n}. \tag{1.12}$$

где n — количество отсчётов (длина таблицы).

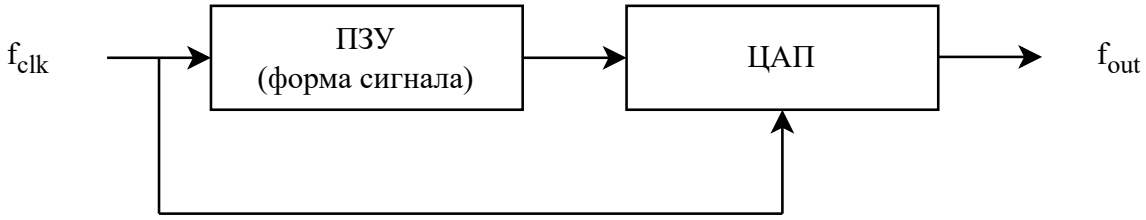


Рис. 1.11 Структурная схема табличного метода.

Управлять частотой устройства не всегда удобно. При желании уменьшить частоту сигнала придётся добавлять какую-то задержку в цикл, а что делать, если появилась необходимость увеличить частоту и код уже максимально оптимизирован. К примеру максимальная частота, которой удалось достигнуть 10 кГц и на большее наше устройство уже не способно. Так как увеличить частоту опроса таблицы уже невозможно, то нужно уменьшить её длину. То есть чтобы нам получить на выходе 20 кГц мы должны будем выводить каждый второй отсчёт таблицы, если 30 кГц, то каждый третий и т. д. Это хороший вариант, но тогда возникает проблема как дополнить программу, чтобы она пропускала нужное количество отсчётов.

1.4.4. Метод DDS

К табличным методам относится также метод прямого цифрового синтеза или как его ещё называют метод DDS и он решает проблему, в которую упирается обычный табличный метод. DDS (Direct Digital Synthesizer) или прямой цифровой синтез, в переводе с английского, представляет собой метод, который позволяет создавать аналоговые сигналы путем генерации цифровой последовательности отсчётов и последующего преобразования этих отсчетов из цифрового вида в аналоговый с помощью ЦАП [3; 9].

На рисунке 1.12 изображена структурная схема DDS с аккумулятором фазы.

Частота сигнала в этой архитектуре определяется следующей формулой:

$$f_{out} = \frac{D * f_{clk}}{2^A}, \tag{1.13}$$

где f_{out} — выходная частота,

f_{clk} — частота устройства,

D — код частоты,

A — разрядность аккумулятора фазы.

Благодаря разрядности аккумулятора фазы можно определять насколько точно будет регулироваться частота выходного сигнала.

Рис. 1.12 Структурная схема DDS с аккумулятором фазы.

В аккумуляторе фазы и есть ключевое отличие метода DDS от простого табличного синтеза. Аккумулятор фазы представляет из себя регистр, в котором в каждом такте работы устройства происходит перезагрузка величины и прибавляется заданный код частоты. Приращение зависит как раз-таки от кода частоты и регулирует это значение [5; 10]. Таким образом, происходит вычисление какой отсчёт нужно отправить в порт цифро-аналогового преобразователя. Ещё одним отличием от табличного способа генерации является работа на фиксированной частоте. Алгоритм метода DDS можно описать блок-схемой на рисунке 1.13.

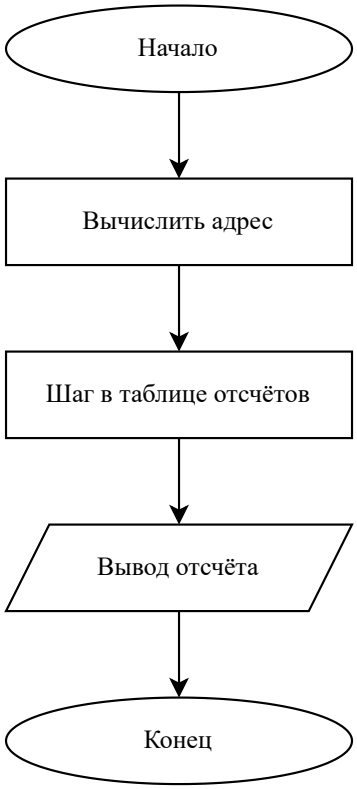


Рис. 1.13 Блок-схема алгоритма метода DDS.

С помощью данного метода можно производить синтез не только стандартных форм сигналов, но и создавать произвольные формы. Метод DDS позволяет управлять цифровым способом амплитудой и фазой сигнала, а также лежит во основе многих приборов [11].

1.5. Обзор существующих генераторов сигналов

В настоящее время существуют полноразмерные генераторы сигналов и генераторы реализованные всего лишь на одной микросхеме.

Рассмотрим полноразмерный генератор отечественного производства от АКТАКОМ — AWG-4112 [12]. Данный источник сигналов умеет генерировать не только стандартные формы, но и произвольные, поэтому относится к генераторам специальных форм. Использует метод DDS, благодаря которому была достигнута высокая стабильность выходных параметров и высокое разрешение по частоте. Диапазон частот: 1 мкГц — 10 МГц.

Рис. 1.14 АКТАКОМ AWG-4112.

К достоинствам данного генератора можно отнести его высокие характеристики, но не у всех пользователей есть в них надобность и тем более стоимость за такие характеристики очень существенная, а также генератор громоздкий по своей конструкции его габариты 235 х 110 х 295 мм и вес 3 кг.

- достоинства: высокая точность, широкий диапазон частот;
- недостатки: большие размеры, высокая стоимость.

Быстрое и непрерывное развитие схемотехники привело к появлению маленьких микросхем, реализующих функционал генератора сигнала [13]. Например, микросхема программируемого генератора AD9833. Данная микросхема тоже использует метод DDS. Управляется она посредством интерфейса SPI. Диапазон частот 0.1 Гц — 25 МГц, формы сигналов стандартные: синусоидальная, треугольная, прямоугольная [14].

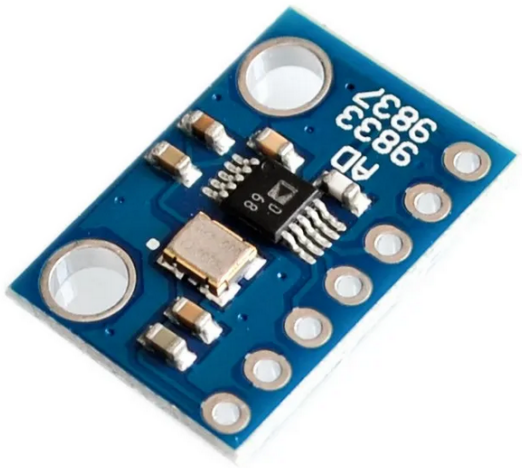


Рис. 1.15 Модуль AD9833.

Реализована микросхема как правило в виде готового модуля с обвязкой. Несомненными плюсами является низкая стоимость, маленькие габариты, малое энергопотребление в сочетании с хорошим качеством выходного сигнала, но недостатком такого генератора является необходимость в управляющем устройстве. Для работы с ним понадобится, например, микроконтроллер, который будет отправлять команды по интерфейсу в данном случае SPI для установки параметров выходного сигнала, то есть помимо самого микроконтроллера потребуется разработать программу для работы с таким видом синтезатора.

- достоинства: компактность, возможность интеграции с другими компонентами;
- недостатки: необходимость в дополнительном управляющем устройстве

Как видно из рассмотренных генераторов они хороши по своему, но под наши требования не подходят. Гораздо эффективнее будет разработать генератор сигналов на одном микроконтроллере тем самым будут сэкономлены ресурсы и система получится довольно гибкой.

1.6. Вывод по первой главе

Таким образом, можно сделать вывод о том, что среди генераторов сигналов наиболее выделяются функциональные генераторы своей универсальностью и гибкостью. Они способны создавать различные функциональные зависимости, что позволяет генерировать сигналы разной формы, включая синусоидальные, треугольные и прямоугольные сигналы в широком спектре частот. Это делает их очень полезными для тестирования, исследования и отладки электронной аппаратуры. В следствие этого было принято решение разрабатывать функциональный генератор сигналов и рассмотрев существующие аналоги стало понятно, что они не совершенны в своей конструкции. В качестве метода генерации сигнала был выбран метод DDS за его простоту реализации и гибкость.

2. РАЗРАБОТКА ПРОГРАММЫ ДЛЯ ГЕНЕРАЦИИ СИГНАЛОВ

2.1. Моделирование DDS

Для начала потребуется таблица отсчётов, чтобы её вычислить используем готовый инструмент [15].

У таблицы есть 5 параметров:

1. Максимальное значение;
2. Количество значений;
3. Смещение от нуля;
4. Разрядность ЦАП: 8 или 12 бит;
5. Форма сигнала.

В данной работе будут использоваться 12-битные значения в количестве 256 чисел. Максимальное значение амплитуды сигнала может быть 4095. Для примера вычислим таблицу значений для синусоиды.

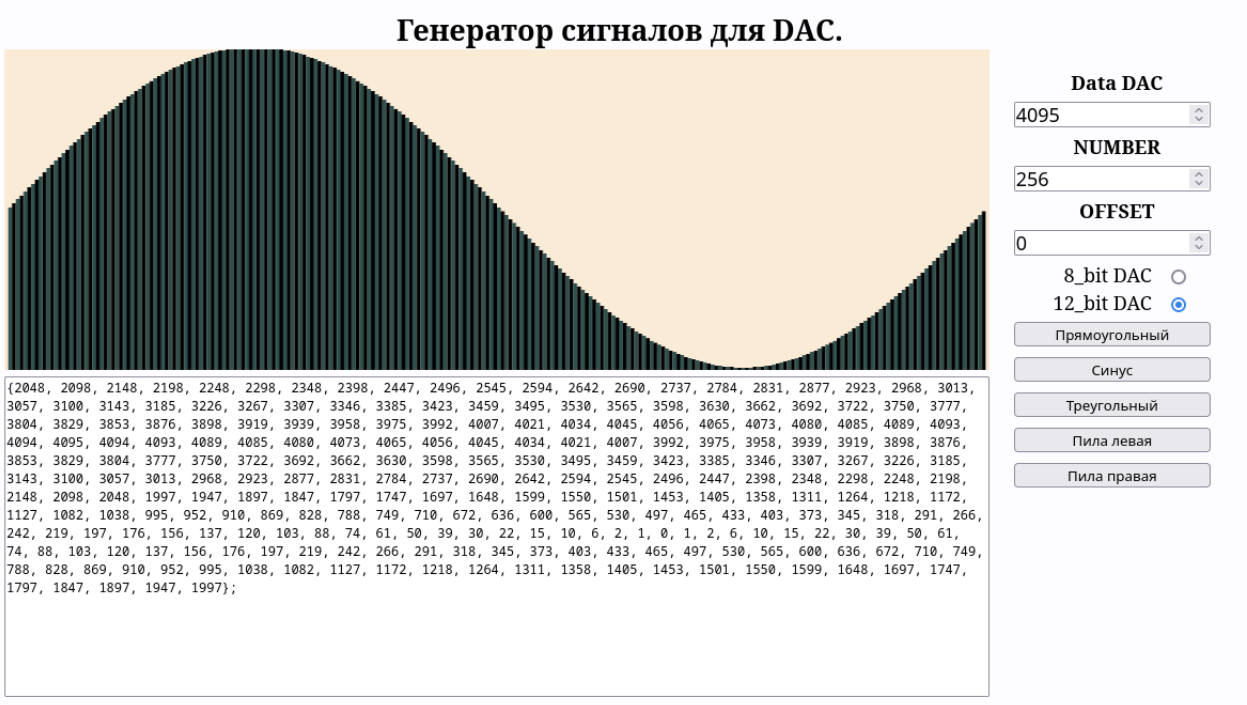


Рис. 2.1 Вычисление таблицы сигнала.

Теперь у нас есть данные для генерации сигнала. Смоделируем алгоритм метода прямого цифрового синтеза по блок-схеме на рис. 1.15 на языке Си для дальнейшей реализации на микроконтроллере.

Листинг 2.1

Метод DDS

```
int main() {
  uint16_t p_acc, p_step;
  uint8_t addr = 0; // адрес ячейки

  p_acc = 0;    // аккумулятор фазы
  p_step = 128; // код частоты

  while(1) {
    addr = p_acc >> 8; // выделение старшей части аккумулятора фазы
    p_acc += p_step;   // шаг
    printf("%d 0x%X\n", addr, sinus[addr]); // вывод отсчёта
  }

  return 0;
}
```

Код частоты задаёт выходную частоту генератора. При значении 256 вывод будет следующий:

```
kenny@desktop:~/workspace/vkr/dds
❯ gcc dds.c -o dds && ./dds
0 0x800
1 0x82C
2 0x858
3 0x884
4 0x8B0
```

Рис. 2.2 Формирование отсчётов при коде частоты 256.

Увеличим код частоты в два раза и получим следующее:

```
kenny@desktop:~/workspace/vkr/dds
❯ gcc dds.c -o dds && ./dds
0 0x800
2 0x858
4 0x8B0
6 0x908
8 0x95F
```

Рис. 2.3 Формирование отсчётов при коде частоты 512.

Как можно заметить отсчёты стали формироваться через один, соответственно частота вырастит в два раза. Теперь уменьшим частоту в два раза выставив код частоты 128.

Рис. 2.4 Формирование отсчётов при коде частоты 128.

Программа стала выводить каждый отсчёт по два раза тем самым, понизив частоту.

В данном виде модуляции код частоты просто абстрактное число, которое добавляется к аккумулятору фазы и узнать реальную частоту проблематично. Результат синтеза будет проверен опытным путём на микроконтроллере.

2.2. Обзор микроконтроллеров

Так как генератор сигналов будет реализовываться на микроконтроллере следует провести обзор и осуществить выбор. Рассмотрим два популярных семейства микроконтроллеров AVR и STM32.

2.2.1. AVR

Микроконтроллеры AVR — это 8-разрядные микроконтроллеры с архитектурой RISC. Данное семейство представляет собой хорошую основу для создания высокопроизводительных и экономичных встраиваемых систем [16]. Подразделяется семейство на две группы: Tiny и Mega.

Микроконтроллеры Tiny имеют небольшую память для программ и их периферия ограничена. Большинство микроконтроллеров данной серии выпускаются в 8-выводных корпусах и предназначены для систем с ограниченным бюджетом. Областью их применения являются различные датчики и бытовая техника [17].

Группа Mega наоборот имеет большую память и развитую периферию. Соответственно область применения гораздо шире и предназначены они для более сложных систем. В таблице 2.1 приведены серии микроконтроллеров и коротко описан их приоритет.

Таблица 2.1

Микроконтроллеры AVR

Группа	Приоритет	Название серий
Tiny	Энергоэффективность, компактность, низкая стоимость	tiny1, tiny2, tiny4, tiny8
Mega	Производительность, гибкость	mega4, mega8, mega16, mega32, mega64, mega128, mega256

2.2.2. STM32

Микроконтроллеры STM32 — это 32-разрядные микроконтроллеры, имеющие процессорное ядро с архитектурой ARM Cortex-M. В настоящее время существует множество микроконтроллеров STM32. Они делятся на семейства в зависимости от версии архитектуры (табл. 2.2).

Таблица 2.2

Семейства STM32

Серия	Ядро
F0	Cortex-M0
G0, L0	Cortex-M0+
F1, F2	Cortex-M3
F3, F4, L4, G4	Cortex-M4
F7, H7	Cortex-M7

Ядро Cortex-M обеспечивает программную совместимость во всех семействах. Кроме этого, для микроконтроллеров выпущенных в одинаковых

корпусах присутствует и аппаратная совместимость, так как на выводах сохраняются одни и те же функции [16; 18]. Будем рассматривать серии микроконтроллеров схожие по функциональным возможностям с Tiny и Mega для дальнейшего сравнения. В таблице 2.3 указаны серии STM32 по группам.

Таблица 2.3

Микроконтроллеры STM32

Группа	Приоритет	Название серий
Широкого применения	Баланс между производительностью и энергоэффективностью	F0, G0, F1, F3, G4
Сверхнизкого энергопотребления	Энергоэффективность, компактность, низкая стоимость	L0, L4

2.3. Сравнение семейств AVR и STM32

Для осуществления выбора проведём сравнение микроконтроллеров, взяв параметры наиболее используемых серий из каждой группы (табл. 2.4). Параметры получены из спецификаций на микроконтроллеры [19—22].

Таблица 2.4

Параметры микроконтроллеров

Параметр	ATtiny10	ATmega32	STM32L010F4	STM32F103xC
Частота	20 МГц	20 МГц	32 МГц	72 МГц
FLASH	1 Кбайт	32 Кбайт	16 Кбайт	256 Кбайт
RAM	64 байт	2 Кбайт	2 Кбайт	48 Кбайт
SPI	-	+	+	+
I2C	-	+	+	+
Питание	1,8 — 5,5 В	1,8 — 5,5 В	1,8 — 3,6 В	1,8 — 3,6 В

Исходя из таблицы можно сделать вывод, что микроконтроллеры AVR применимы в малом спектре задач где скорость не так важна. В нашем же случае скорость работы микроконтроллера может сильно влиять на генерацию

сигнала, а также требуется объём памяти для хранения отсчётов сигналов. В микроконтроллерах STM32 с частотой и объёмом памяти проблем нет и они имеют широкое применение. Серию же выберем F103xC за её характеристики. В связи с этим, а также доступностью отладочных плат будет применён микроконтроллер STM32F103RCT6.

2.4. Среды разработки для STM32

Среда разработки является не маловажным инструментом для создания программной части устройства. В связи с выбором микроконтроллера STM32 рассмотрим популярные бесплатные среды для создания программы на этом семействе микроконтроллеров.

2.4.1. STM32CubeIDE

STM32CubeIDE — это продвинутая платформа разработки на C/C++ с функциями настройки периферийных устройств, генерации кода, компиляции кода и отладки для микроконтроллеров и микропроцессоров STM32 [23]. Среда разработки основана на платформе Eclipse и GCC toolchain для разработки и GDB для отладки. Она позволяет интегрировать сотни существующих плагинов, которые дополняют возможности Eclipse IDE. Имеет расширенные функции отладки, включая: просмотр ядра ЦП, регистров периферийных устройств и памяти, анализ системы просмотра переменных в режиме реального времени. Поддерживается на операционных системах: Linux, macOS, Windows.

После выбора микроконтроллера STM32 создается проект и генерируется код инициализации. В любой момент разработки пользователь может вернуться к инициализации и настройке периферийных устройств и повторно создать код инициализации без какого-либо влияния на пользовательский код. Для разработки используется библиотека HAL.

Драйверы HAL включают в себя полный набор готовых к использованию функций, которые упрощают реализацию пользовательских приложений. Например, коммуникационные периферийные устройства содержат

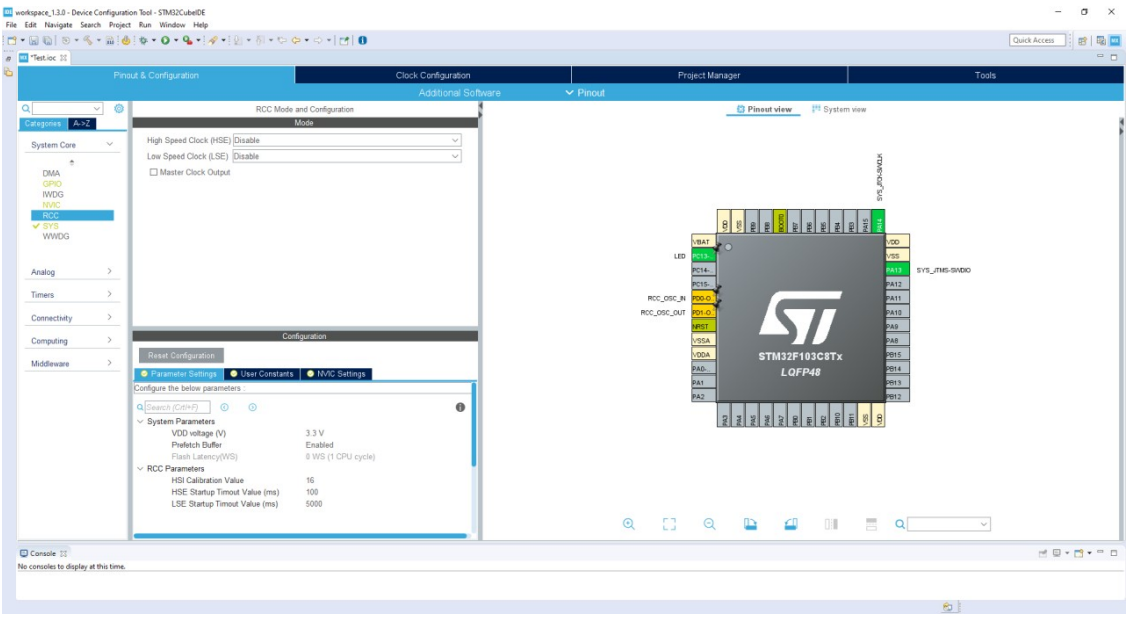


Рис. 2.5 Интерфейс STM32CubeIDE.

функции для инициализации и настройки периферийного устройства, управления передачей данных, обработки прерываний или DMA [24].

Достоинства:

- поддержка различных ОС;
- расширенные возможности отладки;
- большое сообщество;
- автогенерация кода.

Недостатки:

- требовательность к ресурсам ПК;
- сложность настройки.

2.4.2. PlatformIO

PlatformIO — удобная и расширяемая интегрированная среда разработки с набором профессиональных инструментов разработки, предоставляющая современные и мощные функции для ускорения и упрощения процесса разработки встраиваемых продуктов [25].

Данная среда разработки является расширением для текстового редактора Visual Studio Code. VS Code — это легкий, но мощный редактор кода, имеющий богатую экосистему расширений [26]. Доступен для Windows,

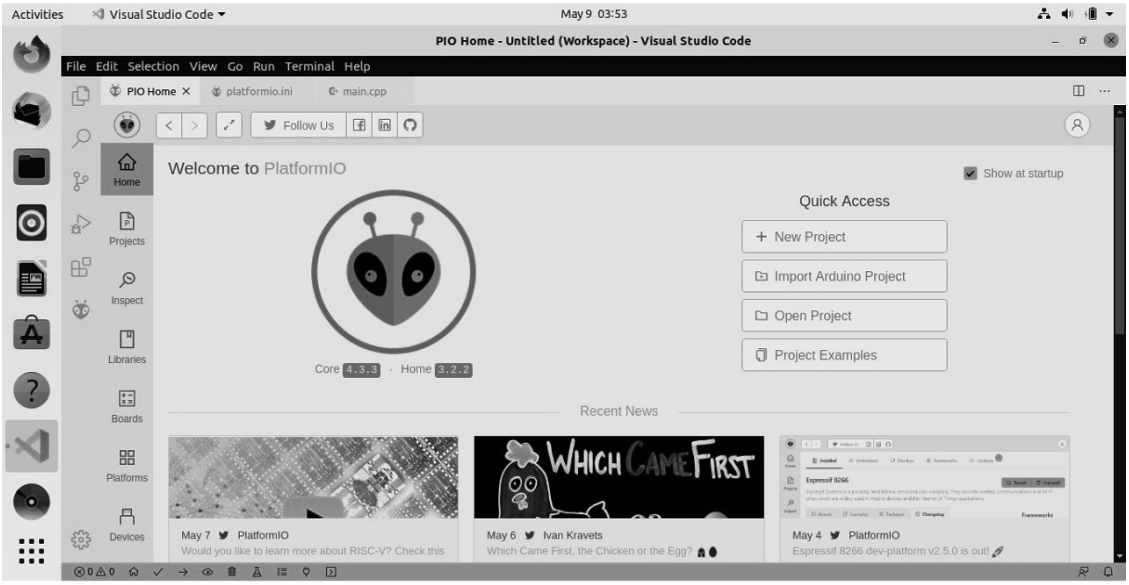


Рис. 2.6 Интерфейс PlatformIO.

macOS и Linux. Работа в паре с VS Code позволяет удобно форматировать код и пользоваться расширениями для языков программирования.

PlatformIO позволяет работать со многими микроконтроллерами и поддерживает множество фреймворков для них, а также библиотек [27]. Ввиду такой широкой поддержки, для STM32 можно разрабатывать с удобной для себя библиотекой. Это может быть к примеру тот же HAL, что и в STM32CubeIDE или libopencm3. Проект libopencm3 (ранее известный как libopenstm32) направлен на создание бесплатной библиотеки микропрограмм с открытым исходным кодом (LGPL версии 3 или более поздней) для различных микроконтроллеров ARM Cortex-M3, включая ST STM32 [28].

Достоинства:

- поддержка различных ОС;
- быстрая компиляция;
- поддержка GitHub;
- возможность работать с разными фрэймворками и платформами.

Недостатки:

- высокий порог вхождения;
- сложность установки.

Попользовавшись обеими средами разработки и разными библиотеками, а также основываясь на достоинствах и недостатках была выбрана среда разработки PlatformIO в связке с библиотекой libopencm3, использующая язык программирования Си [29; 30].

2.5. Алгоритм работы программы для генерации сигналов

Структурно устройство будет выглядеть следующим образом (рис. 2.8). Цифро-аналоговый преобразователь будет использоваться встроенный в микроконтроллер, а в качестве дисплея будет выступать OLED экран с разрешением 128 на 64 пикселя, работающий по интерфейсу I2C на контроллере SSD1306.

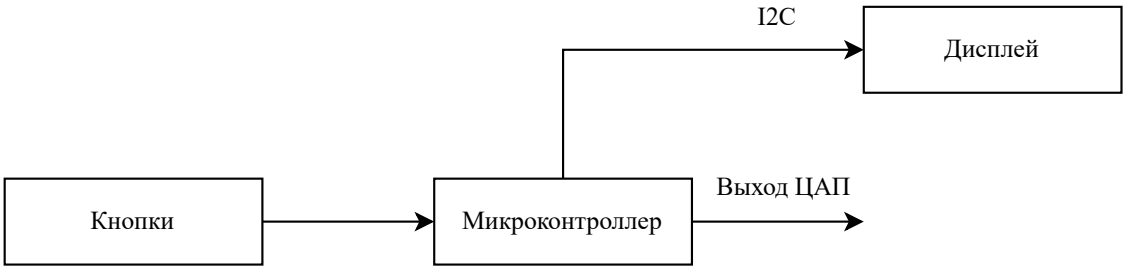


Рис. 2.7 Структурная схема генератора сигналов.

Дисплей понадобится для вывода информации о форме сигнала, частоте и шаге регулировки частоты. Кнопки будут подключены напрямую к микроконтроллеру в количестве пяти штук для выполнения действий:

1. Уменьшить частоту;
2. Увеличить частоту;
3. Предыдущий сигнал;
4. Следующий сигнал;
5. Выбор шага.

Программа должна выполнять три действия:

1. Вывод отсчёта в ЦАП (генерация сигнала);
2. Обработка кнопок;
3. Вывод информации на дисплей.

Для цифро-аналогового преобразователя и кнопок выделим два таймера общего назначения. Таймер номер два будет использоваться для ЦАПа, а номер три для обработки кнопок. На примере третьего таймера рассмотрим его настройку. Алгоритм следующий:

1. Включить тактирование таймера;
2. Задать стартовое значение;
3. Установить предделитель;
4. Установить период таймера;
5. Включить прерывания от таймера;
6. Запустить таймер.

Кроме этого потребуется активировать прерывания и установить им приоритеты.

Таймеры относятся к периферийным устройствам, а вся внутренняя периферия STM32 тактируется от шин APB1 и APB2 (Advanced Peripheral Bus) [31]. В нашем случае таймеры тактируются от шины APB1, частота которой составляет 36 МГц [19]. От предделителя зависит количество отсчётов таймера в секунду. Следовательно, если выставить предделитель 36000 получим следующее:

$$\frac{36 * 10^6}{36 * 10^3} = 1000.$$

Таймер будет делать 1000 отсчётов в секунду. Таким образом, 1 отсчёт таймера будет равняться 1 миллисекунде реального времени. Выставив период 250 получим прерывание от таймера каждые 250 мс. Данного времени хватит для обработки нажатия кнопки, тем самым решается проблема дребезга кнопок без использования программных или аппаратных задержек.

Листинг 2.2

Настройка таймеров и прерываний

```
1 static void timers_setup(void)
2 {
3   rcc_periph_clock_enable(RCC_TIM2);
4   rcc_periph_clock_enable(RCC_TIM3);
```

```
5
6     /* Стартовое значение таймера */
7     TIM_CNT(TIM2) = 0;
8     TIM_CNT(TIM3) = 0;
9
10    /* Предделитель 36MHz/36000 => 1000 отсчетов в секунду (счет начинается с 0, поэтому в предделителе и периоде нужно
      ↪  отнимать единичку) */
11    TIM_PSC(TIM2) = 17;
12    TIM_PSC(TIM3) = 35999;
13
14    /* Период таймера */
15    TIM_ARR(TIM2) = 9;
16    TIM_ARR(TIM3) = 249;
17
18    /* Включить прерывания */
19    TIM_DIER(TIM2) |= TIM_DIER_UIE;
20    TIM_DIER(TIM3) |= TIM_DIER_UIE;
21
22    /* Запустить таймер */
23    TIM_CR1(TIM2) |= TIM_CR1_CEN;
24    TIM_CR1(TIM3) |= TIM_CR1_CEN;
25 }
26
27 static void nvic_setup(void)
28 {
29    /* Активировать прерывания и установить приоритеты */
30    nvic_enable_irq(NVIC_TIM2_IRQ);
31    nvic_set_priority(NVIC_TIM2_IRQ, 2);
32
33    nvic_enable_irq(NVIC_TIM3_IRQ);
34    nvic_set_priority(NVIC_TIM3_IRQ, 1);
35 }
```

Таким образом, подпрограмма обработки кнопок будет размещена в обработчике прерывания от третьего таймера и будет состоять из функций для каждой кнопки.

Листинг 2.3

Обработка кнопок

```
1  void tim3_isr(void) // обработчик прерывания таймера3 (обработка кнопок)
2  {
3     minus_freq();
4     plus_freq();
5     minus_signal(); // функции кнопок
6     plus_signal();
7     step_select();
8     TIM_SR(TIM3) &= ~TIM_SR_UIF; // очистка флага прерывания
9  }
```

Обработка кнопок представлена следующей блок-схемой.

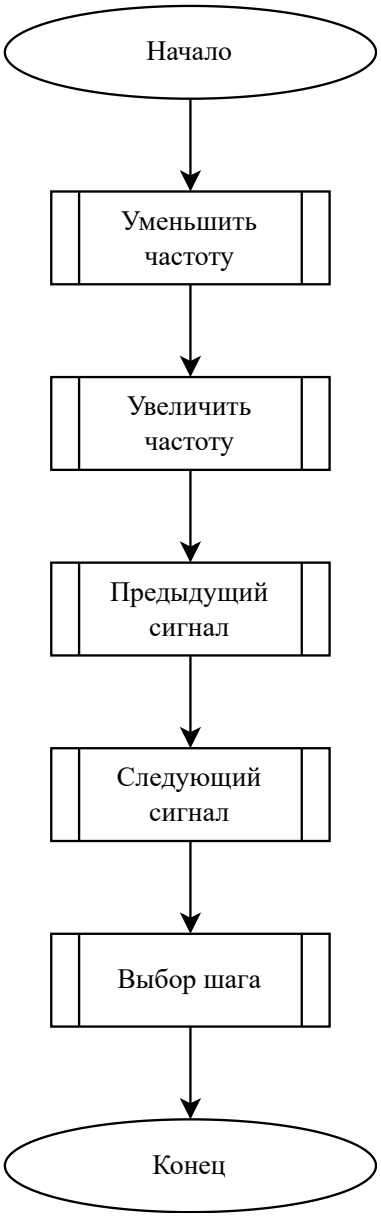


Рис. 2.8 Блок-схема алгоритма функции кнопок.

Подпрограмма для вывода отсчёта в ЦАП будет размещена в обработчике прерывания второго таймера

Листинг 2.4

Генерация сигнала

```
1 void tim2_isr(void) // обработчик прерывания таймера2 (ЦАП)
2 {
3    dac_load_data_buffer_single(signal[p_acc >> 8], RIGHT12, CHANNEL_2); // загрузка буфера в цап
4    p_acc += p_step;            // шаг
```

```
5     TIM_SR(TIM2) &= ~TIM_SR_UIF; // очистка флага прерывания
6  }
```

Работу подпрограммы генерации сигнала описывает следующая блок-схема.

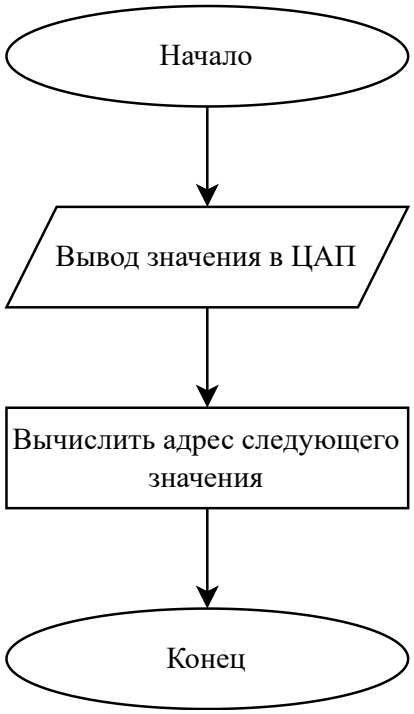


Рис. 2.9 Блок-схема алгоритма функции генерации сигнала.

Блок вычисления адреса содержит в себе выбранный метод прямого цифрового синтеза. Для аккумулятора фазы выделена переменная размером 2 байта. Старшая часть отвечает за адрес ячейки массива сигнала, состоящего из 256 отсчётов, а младшая за шаг в массиве. Размером же шага является код частоты, который прибавляется к аккумулятору фазы. Ввиду того, что размерность массива 256 точек и старшая часть аккумулятора больше 255 быть не может, то проблемы с выходом за границы массива не возникнет.

Главный блок программы будет содержать в себе основные настройки периферии и вывод информации на дисплей. К периферии здесь относится настройка выводов микроконтроллера, таймеров, прерываний, ЦАП и интерфейса I2C для связи с дисплеем.

Интерфейс I2C широко применяется в микропроцессорных системах и его достоинство состоит в том, что передача данных идёт всего через две

линии [32]. Одна линия для информации (SDA), вторая для синхросигнала (SCL). Для вывода информации на дисплей будет использоваться внешняя библиотека с функциями обновления, очистки и вывода строки [33].

В настройке выводов потребуется включить тактирование порта и выделить пять входов микроконтроллера для кнопок. Для цифро-аналогового преобразователя также потребуется включить тактирование, настроить выход и активировать его работу. Настройка I2C будет проходить по следующему алгоритму:

1. Включение тактирования;
2. Настройка альтернативных функций SCL и SDA;
3. Отключение I2C перед изменением конфигурации;
4. Сброс состояния;
5. Установка стандартного режима работы;
6. Установка частоты периферии;
7. Настройка тактовой частоты шины;
8. Задать время нарастания сигналов;
9. Включить подтверждение при получении данных;
10. Запуск интерфейса I2C.

Стандартный режим работы I2C — стандартная тактовая частота работы 100 кГц. Тактируется интерфейс также от шины APB1 — 36 МГц. По спецификации микроконтроллера устанавливается частота шины [19].

Table 52. SCL frequency (f_{PCLK1}= 36 MHz.,V_{DD_I2C} = 3.3 V)(1)(2)

f_{SCL} (kHz)	I2C_CCR value
	R_P = 4.7 kΩ
400	0x801E
300	0x8028
200	0x803C
100	0x00B4
50	0x0168
20	0x0384

Рис. 2.10 Частота шины.

Так как частота тактирования 36 МГц, то время цила будет $\frac{1}{36} = 28$ нс. В характеристиках I2C указано время нарастания сигналов $t_r = 300$ нс [19].

Table 51. I²C characteristics

Symbol	Parameter	Standard mode I²C(1)(2)		Fast mode I²C(1)(2)		Unit
		Min	Max	Min	Max	
$t_{w(SCLL)}$	SCL clock low time	4.7	-	1.3	-	µs
$t_{w(SCLH)}$	SCL clock high time	4.0	-	0.6	-	
$t_{su(SDA)}$	SDA setup time	250	-	100	-	ns
$t_{h(SDA)}$	SDA data hold time	-	3450(3)	-	900(3)	
$t_{r(SDA)}$ $t_{r(SCL)}$	SDA and SCL rise time	-	1000	-	300	
$t_{f(SDA)}$ $t_{f(SCL)}$	SDA and SCL fall time	-	300	-	300	
$t_{h(STA)}$	Start condition hold time	4.0	-	0.6	-	µs
$t_{su(STA)}$	Repeated Start condition setup time	4.7	-	0.6	-	
$t_{su(STO)}$	Stop condition setup time	4.0	-	0.6	-	µs
$t_{w(STO:STA)}$	Stop to Start condition time (bus free)	4.7	-	1.3	-	µs
C_b	Capacitive load for each bus line	-	400	-	400	pF
t_{SP}	Pulse width of the spikes that are suppressed by the analog filter for standard and fast mode	0	50(4)	0	50(4)	µs

Рис. 2.11 Характеристики I2C.

По формуле рассчитаем нужное значение для установки времени нарастания [34].

$$I2C\_TRISE = \frac{t_r}{TPCL1} + 1 = \frac{300}{28} + 1 = 37.$$

Листинг 2.5

Функции настроек выводов, ЦАП и I2C

```
1 static void gpio_setup(void)
2 {
3   // rcc_periph_clock_enable(RCC_GPIOD); // тактирование портов
4   rcc_periph_clock_enable(RCC_GPIOB);
5   gpio_set_mode(GPIOB, GPIO_MODE_INPUT, GPIO_CNF_INPUT_PULL_UPDOWN, GPIO9 | GPIO5 | GPIO6 | GPIO7 | GPIO8); //
    ↪  входы для кнопок, подтянуты к земле
6   // gpio_set_mode(GPIOD, GPIO_MODE_OUTPUT_50_MHZ, GPIO_CNF_OUTPUT_PUSHPULL, GPIO2);
7 }
8
```

```
9  static void dac_setup(void)
10 {
11    rcc_periph_clock_enable(RCC_GPIOA);
12    gpio_set_mode(GPIOA, GPIO_MODE_OUTPUT_2_MHZ, GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO5);
13    rcc_periph_clock_enable(RCC_DAC); // тактирование цапа и настройка вывода
14    dac_enable(CHANNEL_2);            // включить цап
15 }
16
17 static void i2c_setup(void){
18    // Включение тактирования периферийного оборудования для I2C2
19    rcc_periph_clock_enable(RCC_I2C2);
20
21    /*
22     * Настройка альтернативных функций для пинов SCL и SDA интерфейса I2C2.
23     * Это необходимо для подключения I2C устройств к микроконтроллеру через эти
24  пины.
25     */
26    gpio_set_mode(GPIOB, GPIO_MODE_OUTPUT_50_MHZ,
27                  GPIO_CNF_OUTPUT_ALTFN_OPENDRAIN,
28                  GPIO_I2C2_SCL | GPIO_I2C2_SDA);
29
30    // Отключение I2C перед изменением конфигурации
31    i2c_peripheral_disable(I2C2);
32
33    // Сброс состояния периферийного устройства I2C2
34    i2c_reset(I2C2);
35
36    // Установка стандартного режима работы I2C
37    i2c_set_standard_mode(I2C2);
38
39    // Установка частоты периферии
40    i2c_set_clock_frequency(I2C2, I2C_CR2_FREQ_36MHZ);
41
42    // Настройка тактовой частоты шины;
43    i2c_set_ccr(I2C2, 0xB4);
44
45    // Установка времени подъема сигнала SDA после завершения операции чтения/записи
46    i2c_set_trise(I2C2, 0x25);
47
48    // Включение подтверждения при получении данных от устройства
49    i2c_enable_ack(I2C2);
50
51    // Включение периферийного устройства I2C2
52    i2c_peripheral_enable(I2C2);
53 }
```

На рис. 2.12 представлена блок-схема алгоритма главной функции.

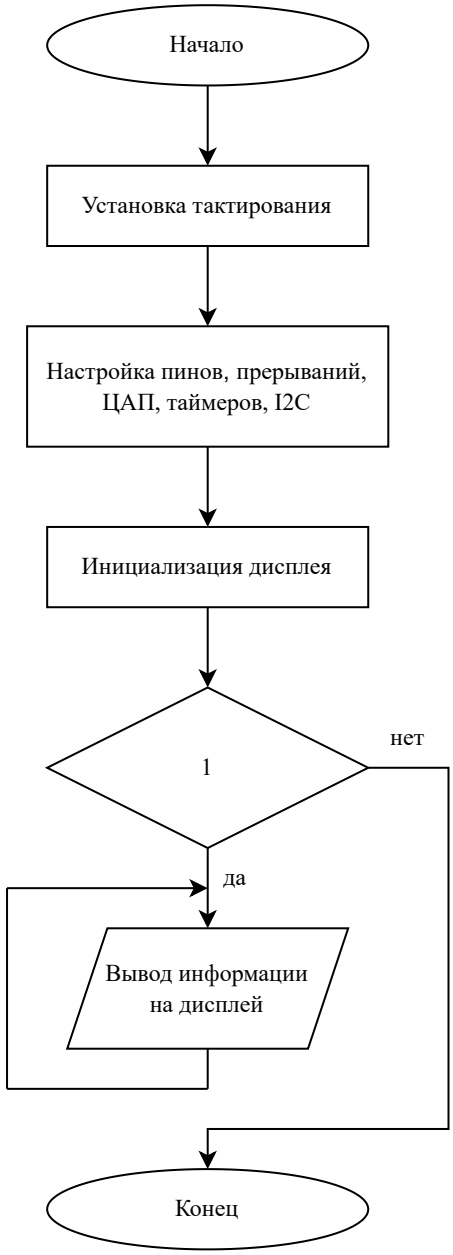


Рис. 2.12 Блок-схема алгоритма главной функции.

Листинг 2.6

Главная функция программы

```
1  int main(void)
2  {
3    rcc_clock_setup_in_hse_8mhz_out_72mhz(); // установка тактирования
4    gpio_setup();
5    nvic_setup();
6    dac_setup();
7    timers_setup();
8    i2c_setup();
9    ssd1306_init(I2C2, DEFAULT_7bit_OLED_SLAVE_ADDRESS, 128, 64); // инициализация дисплея
```

```
10
11    int f = 0;       // переменная частоты
12    wchar_t freq[8]; // буфер для wchar_t строки
13    while (1)
14    {
15       f = p_step / 24 * 125;
16       swprintf(freq, sizeof(freq) / sizeof(wchar_t), L"%d", f); // Использование swprintf для преобразования int в wchar_t*
17       /*  вывод информации на дисплей  */
18       ssd1306_clear();
19       ssd1306_drawWCharStr(0, 0, white, nowrap, L"Форма сигнала:");
20       switch (num_sig)
21       {
22       case 1:
23          ssd1306_drawWCharStr(0, 8, white, nowrap, L"Синус");
24          break;
25       ^^I...
26       case 5:
27          ssd1306_drawWCharStr(0, 8, white, nowrap, L"Пила Правая");
28          break;
29       }
30       ssd1306_drawWCharStr(0, 16, white, nowrap, L"Частота(Гц)");
31       ssd1306_drawWCharStr(64, 16, white, nowrap, freq);
32       ssd1306_drawWCharStr(0, 32, white, nowrap, L"Шаг(Гц)");
33       switch (num_step)
34       {
35       case 1:
36          ssd1306_drawWCharStr(64, 32, white, nowrap, L"125");
37          break;
38       ...
39       case 4:
40          ssd1306_drawWCharStr(64, 32, white, nowrap, L"1000");
41          break;
42       }
43       ssd1306_refresh();
44    }
45
46    return 0;
47 }
```

Полный код программы содержится в приложении. После написания программы произведём сборку проекта и получим сообщение об успешной компиляции (рис. 2.13).

```
ПРОБЛЕМЫ   ВЫХОДНЫЕ ДАННЫЕ   КОНСОЛЬ ОТЛАДКИ   ТЕРМИНАЛ   ПОРТЫ

Scanning dependencies...
No dependencies
Building in release mode
Compiling .pio/build/genericSTM32F103RC/src/main.o
Checking size .pio/build/genericSTM32F103RC/firmware.elf
Advanced Memory Usage is available via "PlatformIO Home > Project Inspect"
RAM:   [=         ]  10.9% (used 5372 bytes from 49152 bytes)
Flash: [=         ]   8.7% (used 22900 bytes from 262144 bytes)
========================================================================= [SUCCESS] Took 0.75 seconds
```

Рис. 2.13 Компиляция проекта.

Для более подробной информации проверим проект с помощью функции «Inspect».

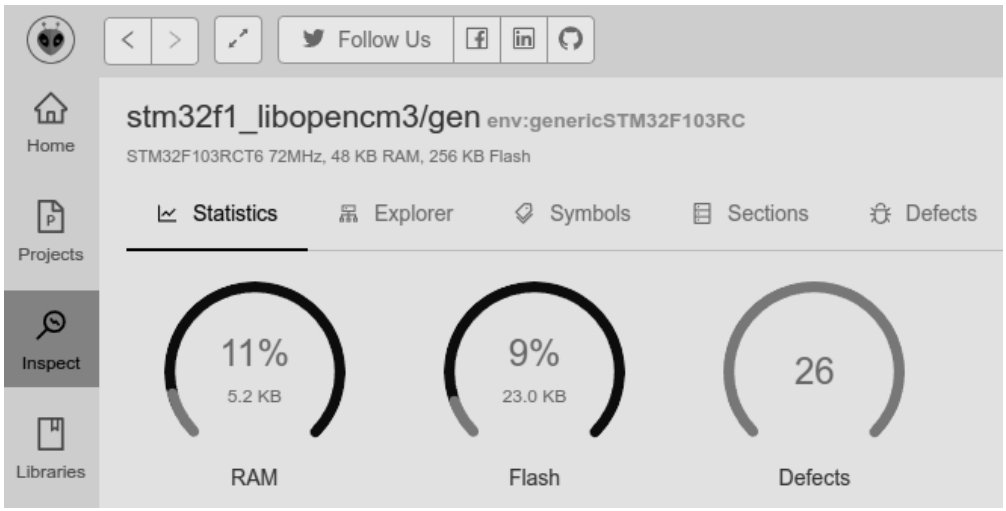


Рис. 2.14 Проверка проекта.

Среда разработки описала занимаемую память и нашла дефекты в проекте.

Рис. 2.15 Дефекты проекта.

Дефекты оказались незначительные и на работу программы влияния не оказывают.

2.6. Вывод по второй главе

Таким образом, для реализации генератора сигналов был выбран микроконтроллер STM32F103RCT6 с использованием среды разработки PlatformIO и библиотеки libopencm3. Определен четкий план действий, включающий структуру программы, алгоритмы работы с цифро-аналоговым преобразователем, кнопками и дисплеем. Была разработана и скомпилирована результирующая программа, которая будет протестирована на макете устройства.

3. ПРОЕКТИРОВАНИЕ И ТЕСТИРОВАНИЕ ГЕНЕРАТОРА СИГНАЛОВ

3.1. Проектирование генератора сигналов

По структурной схеме (рис. 2.7) создадим фрагмент схемы электрической принципиальной.

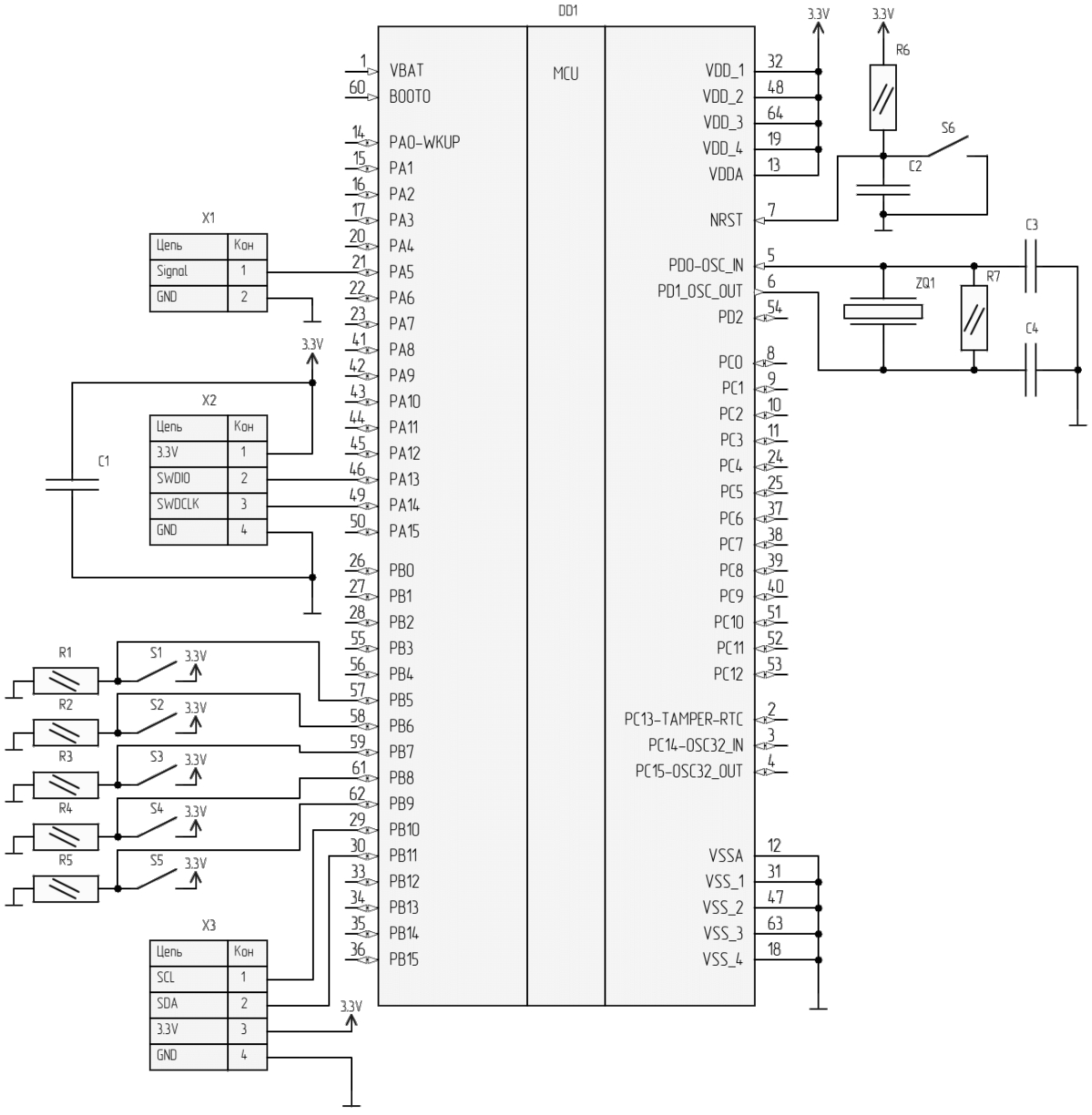


Рис. 3.1 Фрагмент схемы электрической принципиальной.

Просто так контроллер работать не сможет. Необходим внешний кварцевый резонатор для стабилизации частоты системного тактового генерато-

ра, а также функция сброса. Эти узлы уже реализованы на отладочной плате микроконтроллера. Питание схемы будет подаваться через разъём SWD.

Для улучшения генерации будет задействован встроенный в цифро-аналоговый преобразователь выходной буфер. При его использовании он будет срезать сигнал сверху и снизу на 0.2В, поэтому значения тоже следует срезать на эту же величину для корректной генерации.

В документе от STM про работу с цифро-аналоговым преобразователем есть формула для расчета выходного напряжения [35].

$$DAC_{output} = V_{REF} * \frac{DOR}{DAC_{MaxDigitalValue} + 1}, \tag{3.1}$$

где DAC_{output} — выходное напряжение ЦАП,

V_{REF} — опорное напряжение,

DOR — цифровое значение выходного напряжения,

$DAC_{MaxDigitalValue}$ — максимальное значение DOR.

Нам нужно найти какое значение соответствует напряжению 0.2В. Выразим DOR и подставим имеющиеся значения.

$$DOR = \frac{V_{REF}}{DAC_{output}} * DAC_{MaxDigitalValue} + 1 = \frac{3.3}{0.2} * (4095 + 1) = 248.$$

Поэтому для отсчётов нужно будет указать смещение от нуля 248, а максимальное значение 4095 меньше на 248, то есть 3847.

Для управления установлены 5 кнопок, для которых выделены выводы PB5 - PB9. Постоянно быть подключенными к напряжению или земле выводы не могут, т. к. не будет возможности подавать на них какой-либо информационный сигнал. На выводы могут наводиться произвольные потенциалы, что негативно влияет на работу схемы. Подобные потенциалы имитируют сигналы, которые не предусмотрены. Из-за них может нарушиться логика работы, поэтому их принято фиксировать [32].

Для этого используют подтягивающие (Pull - up) или заземляющие (Pull - down) резисторы. Они создают цепь, которая обеспечивает подтяжку сигнала к напряжению питания или земле. Если резисторы имеют большие сопротивления, то сигналы относят к слабым. При подключении сильных информационных сигналов происходит преодоление слабых и функциональность схемы не нарушается [36].

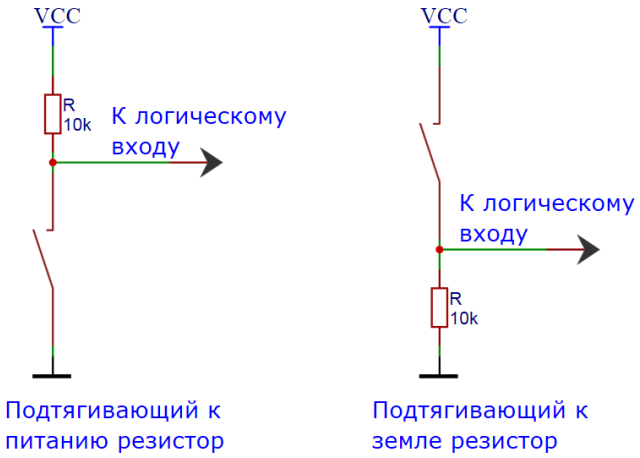


Рис. 3.2 Pull-up и Pull - down резисторы.

В схеме устройства будет использоваться подтяжка к земле для считывания высокого уровня сигнала, то есть будет использоваться прямая логика. Рассчитаем минимальное и максимальное сопротивление заземляющего резистора.

$$R_{min} = \frac{V_0}{I_{max}}, \tag{3.2}$$

где V_0 — напряжение логического нуля,

I_{max} — максимальный ток вывода.

Согласно спецификации микроконтроллера напряжение логического нуля составляет 1,16 В, а максимальный протекающий ток через пин может быть 25 мА [19].

$$R_{min} = \frac{1{,}16}{25 * 10^{-3}} = 46{,}4 \text{ Ом}.$$

Для расчета максимального сопротивления формула следующая:

$$R_{max} = \frac{t}{C_{I/O}}, \tag{3.3}$$

где t — время нарастания сигнала,

$C_{I/O}$ — ёмкость вывода.

Ёмкость вывода составляет 5 пФ, время нарастания возьмём 1 микросекунду (стандартный сигнал 100 кГц).

$$R_{max} = \frac{1 * 10^{-6}}{5 * 10^{-12}} = 200 \text{ кОм}.$$

Из расчётов можно сделать вывод, что номинал резистора расположен в следующих границах.

$$46{,}4 \text{ Ом} < R < 200 \text{ кОм}.$$

Разброс довольно большой, но из практики известно, что подтягивающий резистор имеет номинал 1 - 10 кОм.

Для линий I2C также используются подтягивающие резисторы, которые уже установлены на дисплее.

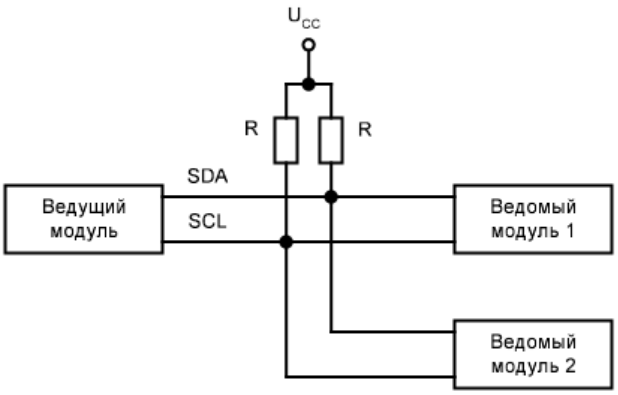


Рис. 3.3 Организация интерфейса.

Интерфейс в микроконтроллере расположен на выводах PB10 (SCL) и PB11 (SDA). Также для дисплея потребуется питание 3.3 В.

Дисплей и кнопки расположим на макетной плате размером 50 на 50 мм.

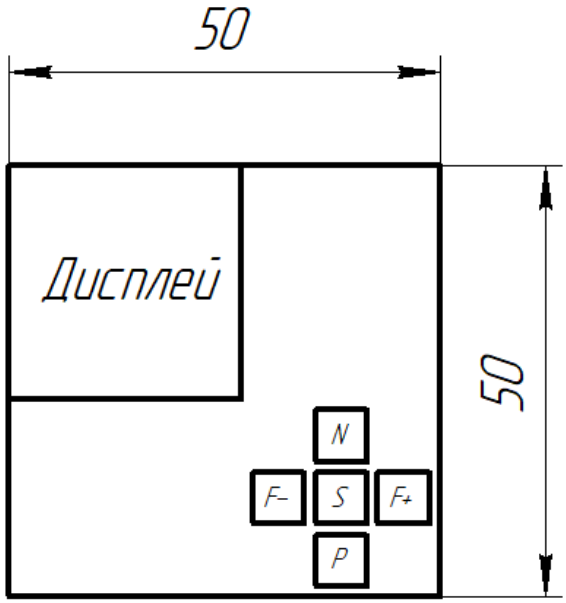


Рис. 3.4 Схема расположения периферии.

Назначения кнопок:

1. F- — уменьшить частоту;
2. F+ — увеличить частоту;
3. P — предыдущий сигнал;
4. N — следующий сигнал;
5. S — переключить шаг по частоте.

В результате сборки получилась плата с периферией.

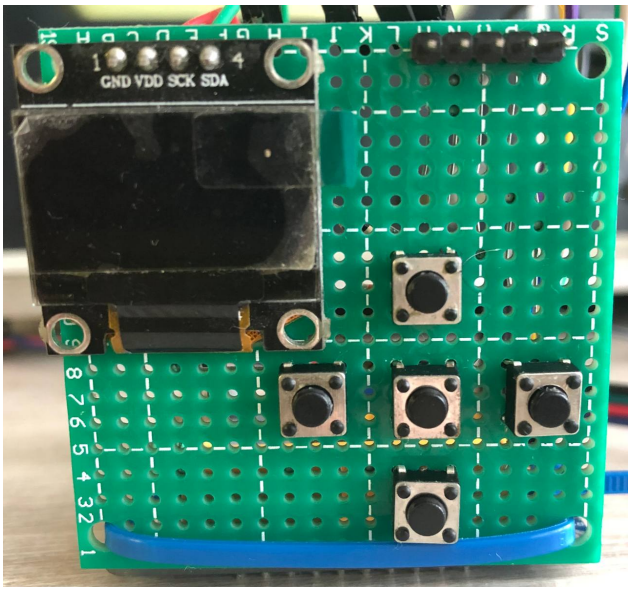


Рис. 3.5 Плата периферии.

Макет устройства будет состоять из отладочной платы микроконтроллера и полученной платы периферии. Обе части будут соединены проводами. Выход цифро-аналогового преобразователя, на котором генерируется сигнал, расположен на отладочной плате. В результате конструирования получился следующий макет устройства (рис. 3.6).

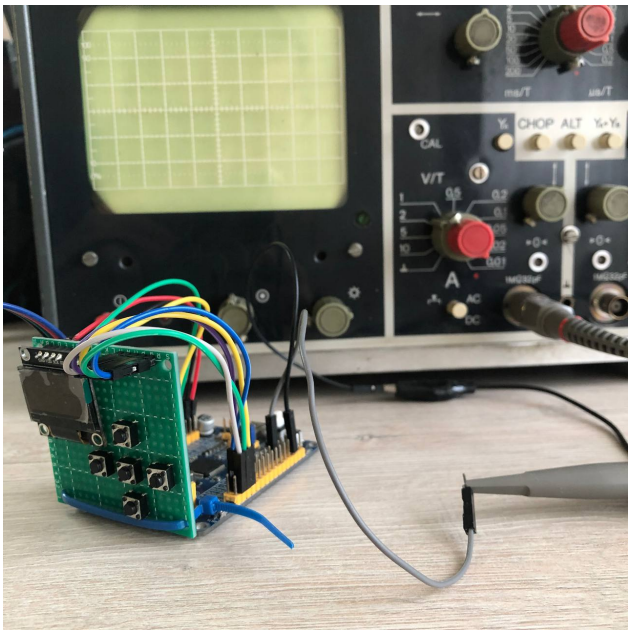


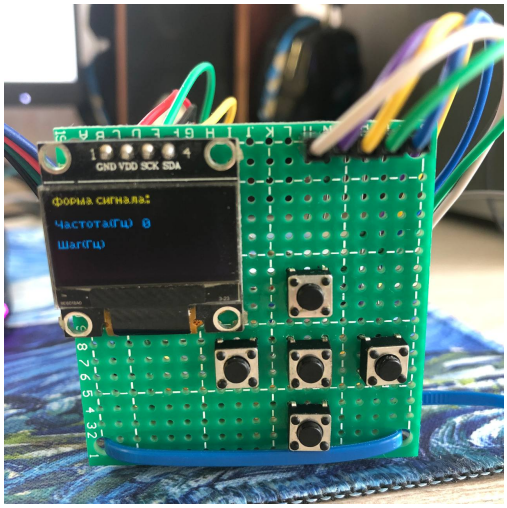
Рис. 3.6 Макет устройства.

3.2. Тестирование генератора сигналов

Протестируем работоспособность полученного устройства. Будем отслеживать состояние устройства и информацию в отладчике. В отладчике будем отслеживать следующие переменные:

- f — расчетная частота;
- step — код частоты (шаг по частоте);
- num_step — номер шага;
- num_sig — номер сигнала;
- signal — буфер сигнала.

После запуска устройства на экране появляются строки с пустыми параметрами формы сигнала, частоты и шага.



(а) Состояние устройства.

(б) Состояние в отладчике.

Рис. 3.7 Начальное состояние устройства при запуске.

Логика работы дисплея описана блок-схемой (рис. 3.8).

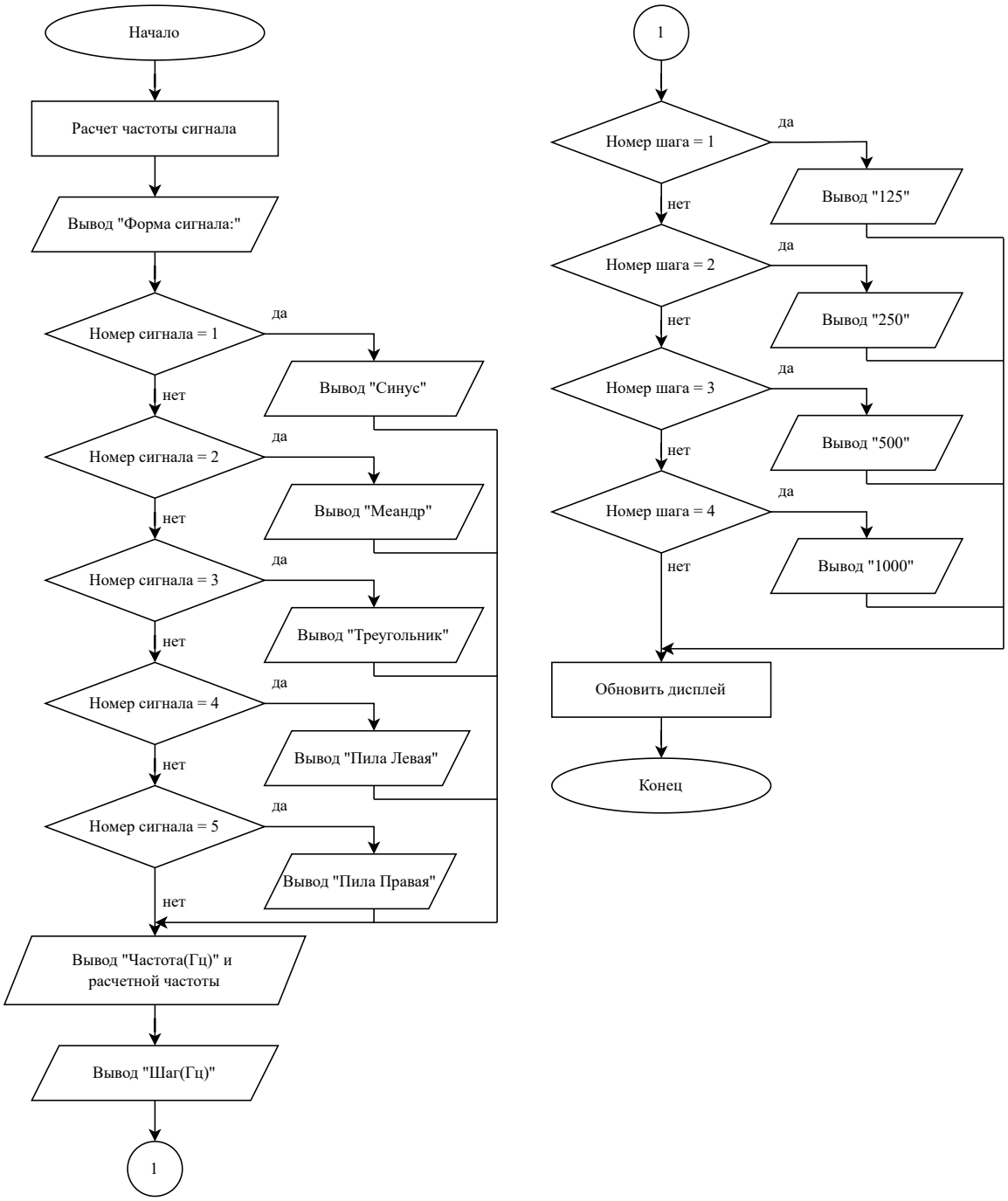


Рис. 3.8 Блок-схема алгоритма работы дисплея.

Требуется выставить параметры сигнала. Клавишами N и P выбирается форма сигнала. При нажатии клавиши N выполняется алгоритм, описанный блок-схемой (рис. 3.10). Для клавиши P алгоритм отличается только тем, что номер сигнала нужно декрементировать.

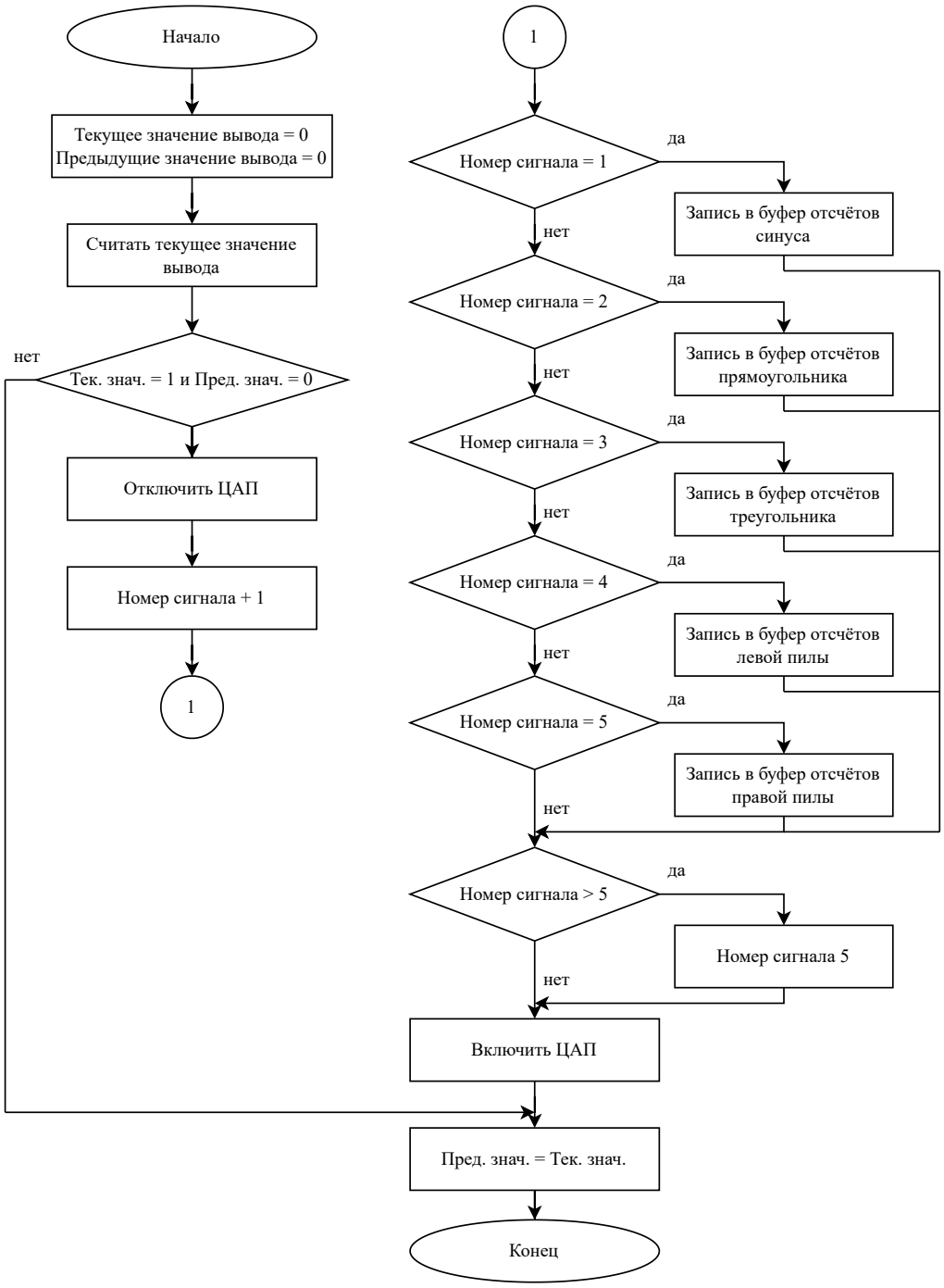
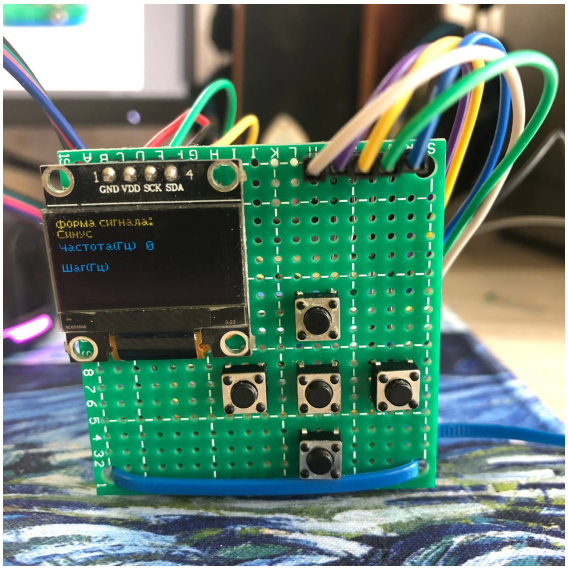


Рис. 3.9 Блок-схема алгоритма выбора следующего сигнала.

При выборе сигнала пользователь листает формы сигнала, а микроконтроллер заполняет буфер отсчётами выбранного сигнала. Выберем сигнал синуса.



(а) Состояние устройства.

КОНТРОЛЬНОЕ ЗНАЧЕНИЕ

```
f:
step: 0
num_step: 0 '\000'
num_sig: 1 '\001'
signal: [256]
  0: 2048
  1: 2092
  2: 2136
  3: 2180
  4: 2224
  5: 2268
  6: 2312
  7: 2355
```

(б) Состояние в отладчике.

Рис. 3.10 Процедура выбора синусоидального сигнала.

Теперь нужно выбрать величину шага, с которым будет регулироваться частота сигнала. Алгоритм для установки шага следующий (рис. 3.16).

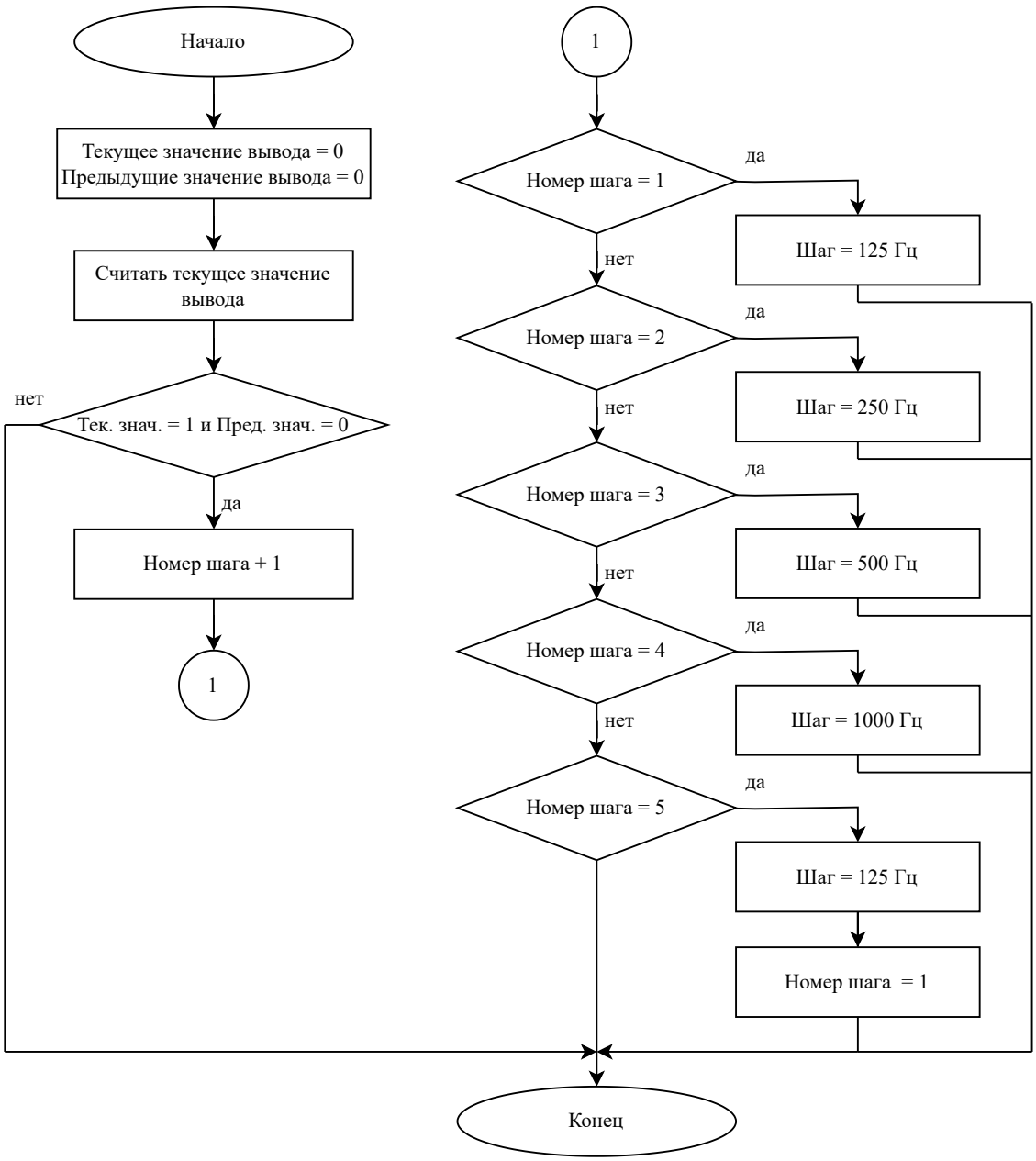
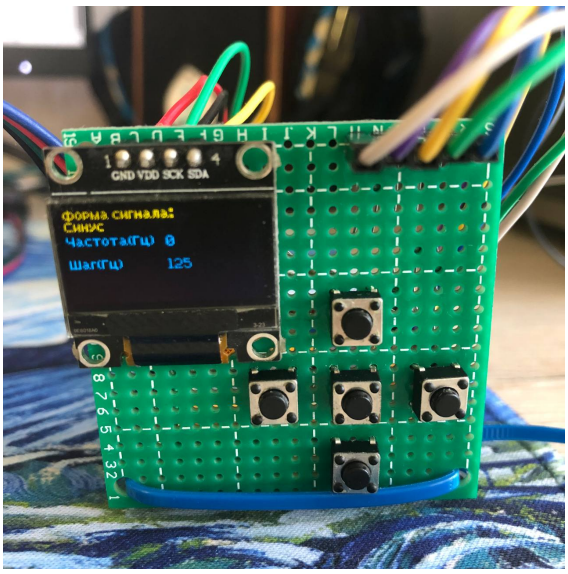


Рис. 3.11 Блок-схема алгоритма выбора шага.

Как можно заметить из блок-схемы всего есть 4 шага:

1. 125 Гц;
2. 250 Гц;
3. 500 Гц;
4. 1000 Гц.

Выберем для примера шаг 125 Гц.



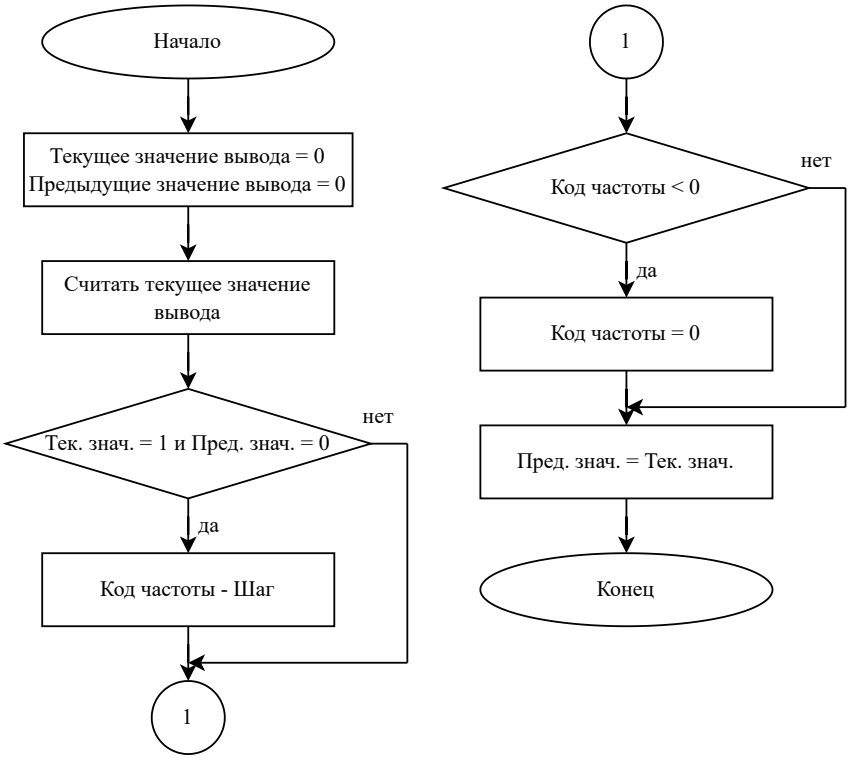
(а) Состояние устройства.

(б) Состояние в отладчике.

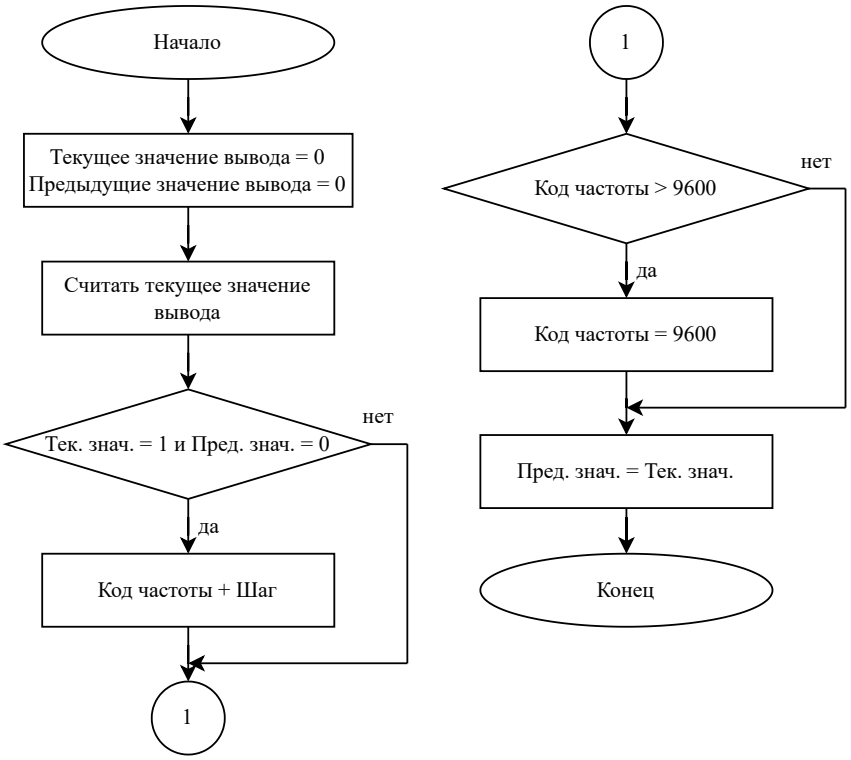
Рис. 3.12 Процедура выбора шага частотой 125 Гц.

Выбор шага происходит циклично, поэтому для него требуется только одна кнопка. Самим же шагом по частоте является переменная step, которая содержит в себе для микроконтроллера код частоты, соответствующий частотам генерируемого сигнала.

С помощью клавиш F- и F+ регулируется частота с заданным шагом. Возможный диапазон частот, который можно выставить 125 — 50000 Гц. Алгоритм представлен блок-схемами (рис. 3.20).



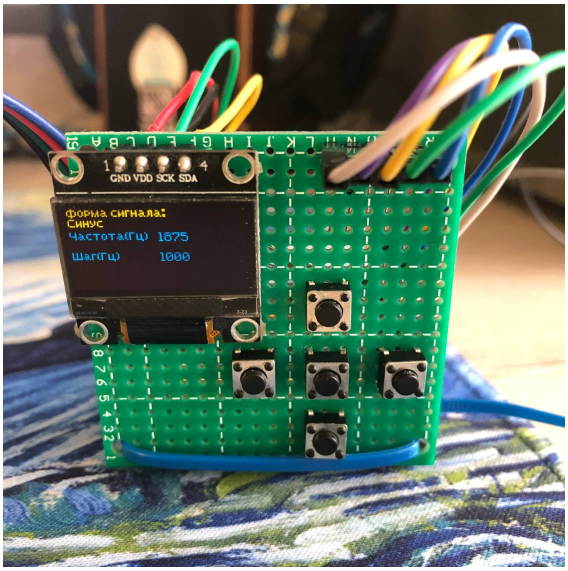
(а) Блок-схема алгоритма уменьшения частоты.



(б) Блок-схема алгоритма увеличения частоты.

Рис. 3.13 Блок-схемы алгоритмов регулировки частоты.

Все функции отрабатывают корректно. Выставим сигнал синуса с частотой 1875 Гц.



(а) Состояние устройства.

(б) Состояние в отладчике.

Рис. 3.14 Выставленные параметры.

Снимем сигнал с осциллографа.

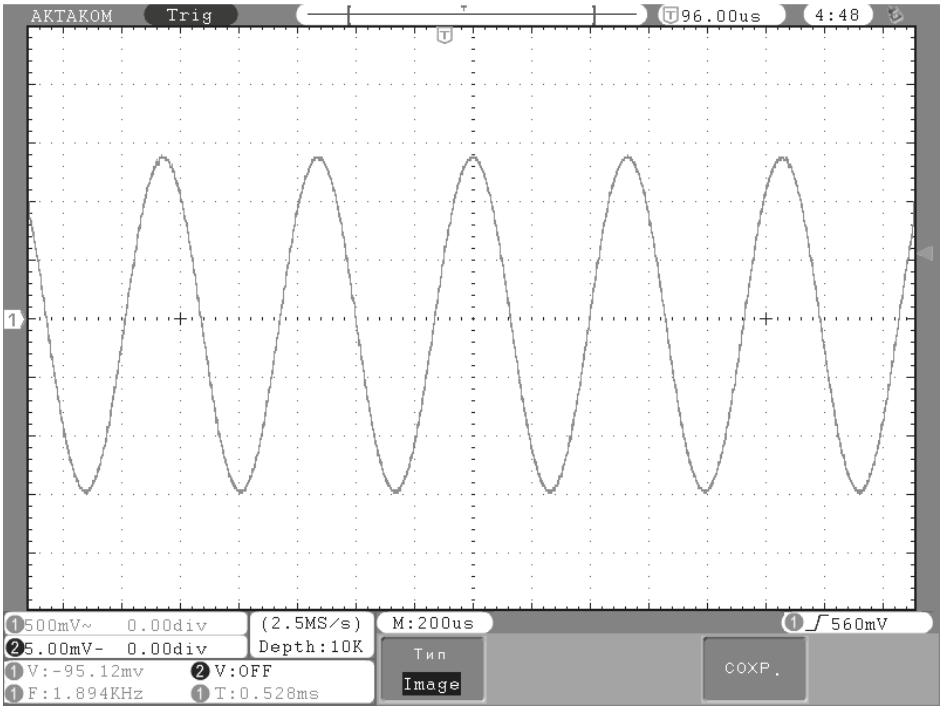


Рис. 3.15 Синусоидальный сигнал с частотой 1875 Гц.

Рассмотрим все формы сигналов на частоте 1 кГц.

На рис. 3.16 изображён синусоидальный сигнал с частотой 1 кГц.

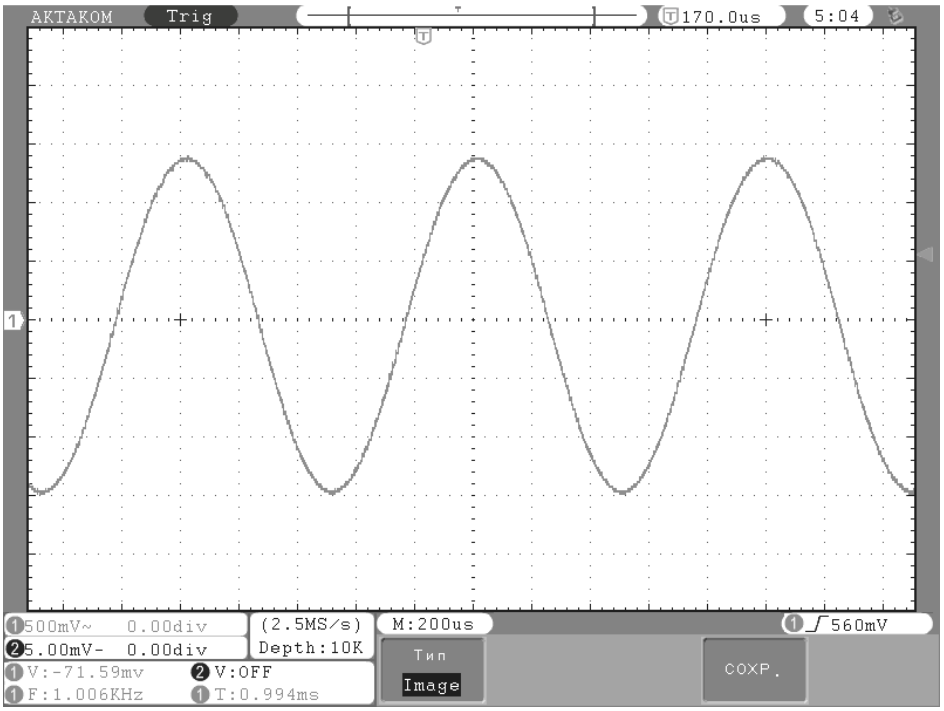


Рис. 3.16 Синусоидальный сигнал с частотой 1 кГц.

На рис. 3.17 изображён прямоугольный сигнал с частотой 1 кГц.

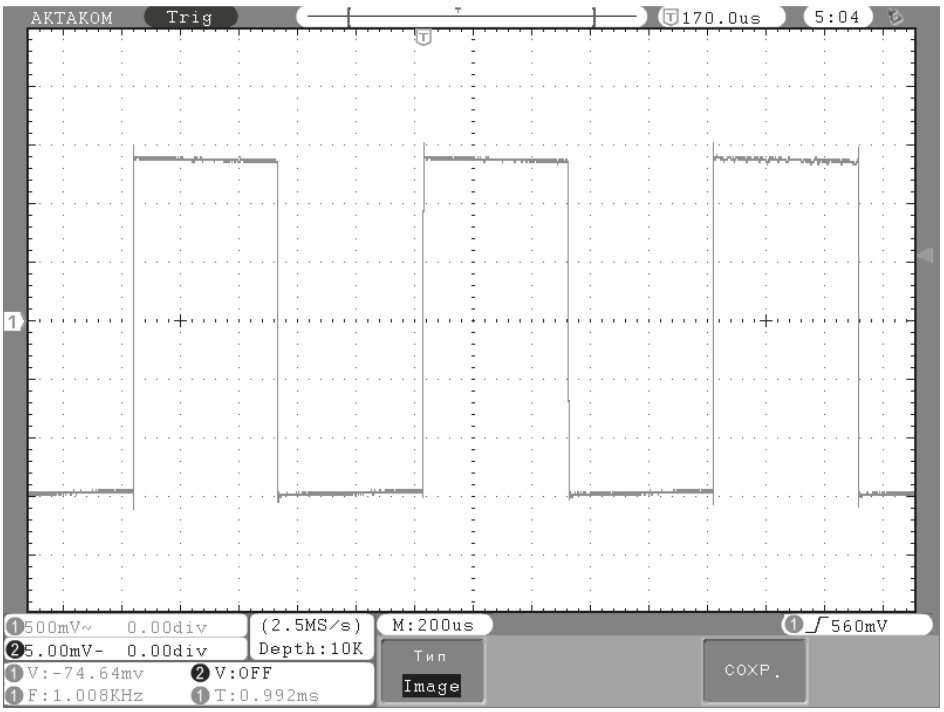


Рис. 3.17 Прямоугольный сигнал с частотой 1 кГц.

На рис. 3.18 изображён треугольный сигнал с частотой 1 кГц.

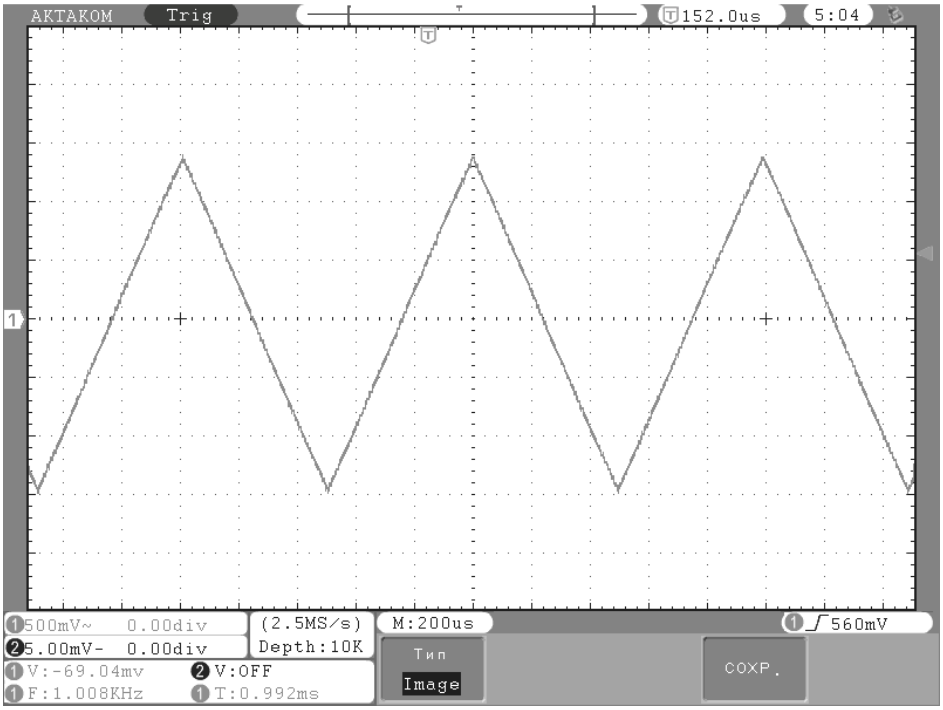


Рис. 3.18 Треугольный сигнал с частотой 1 кГц.

На рис. 3.19 изображён обратный пилообразный сигнал с частотой 1 кГц.

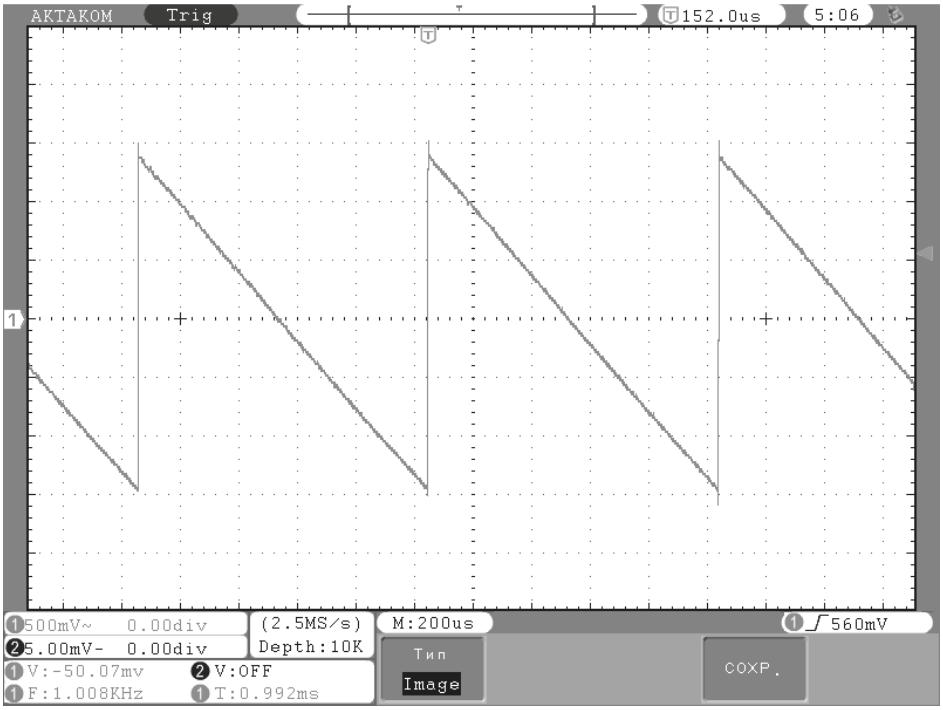


Рис. 3.19 Обратный пилообразный сигнал с частотой 1 кГц.

На рис. 3.20 изображён пилообразный сигнал с частотой 1 кГц.

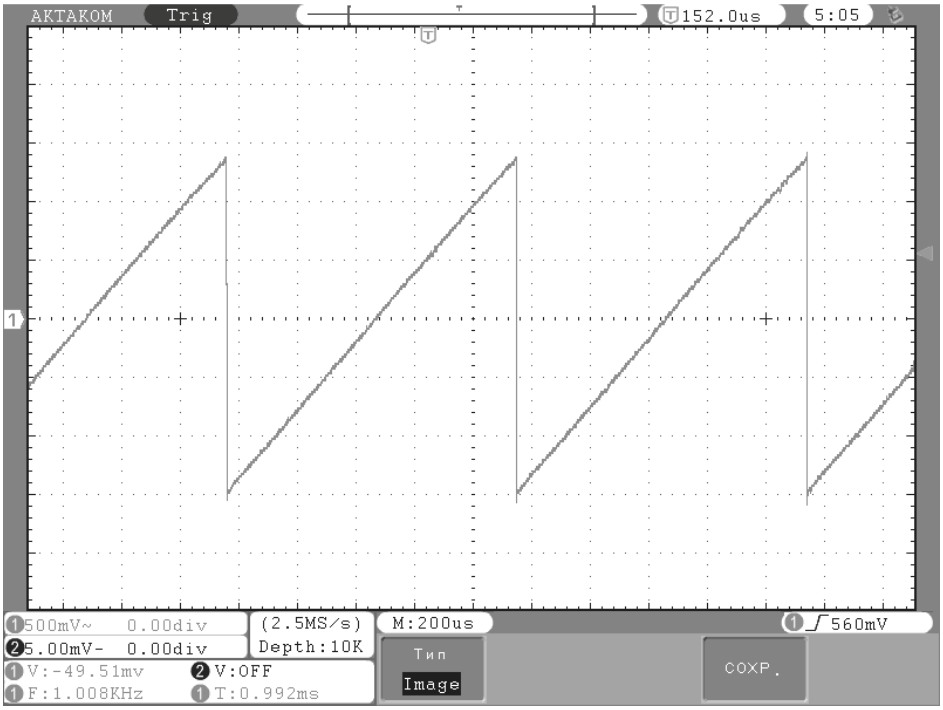


Рис. 3.20 Пилообразный сигнал с частотой 1 кГц.

Как можно заметить устройство генерирует все заданные формы сигналов. Рассмотрим более высокие частоты синуса (рис 3.21).

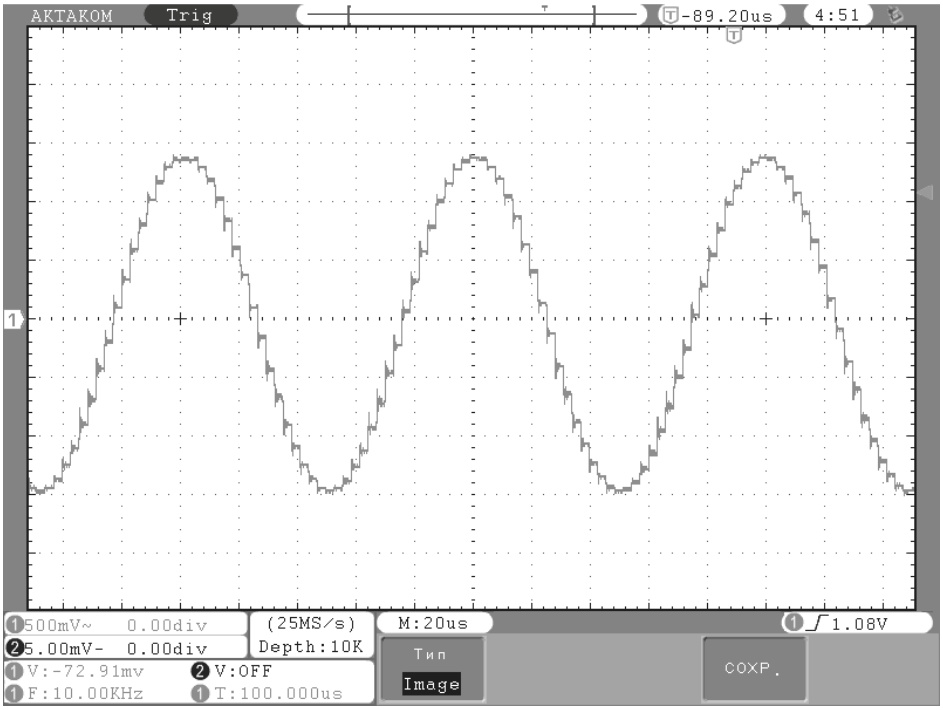


Рис. 3.21 Синусоидальный сигнал с частотой 10 кГц.

На 10 кГц уже можно наблюдать, что отсчёты сигнала не такие ровные. На 50 кГц синусоиду уже трудно узнать (рис. 3.22). На её период приходится 7 отсчётов.

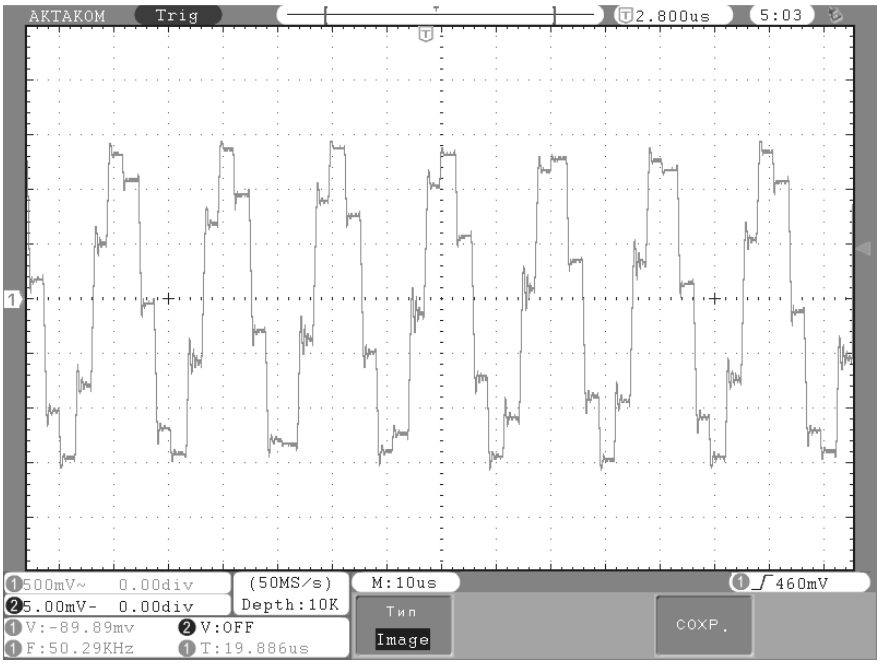


Рис. 3.22 Синусоидальный сигнал с частотой 50 кГц.

Исходя из этого можно сделать вывод, что присутствует шум цифро-аналогового преобразователя.

Проведём анализ сигнала в программе GNU Octave [37]. Возьмём записанные осциллографом отсчёты синусоидального сигнала на частоте 1 кГц и построим для сравнения такую же форму сигнала, но с заведомо большей частотой дискретизации. Будем считать построенную форму идеальным сигналом. Частота дискретизации ЦАП составляет 1 МГц, а для построения возьмём 5 МГц.

Листинг 3.1

Сравнение сигналов

```
1 clear all
2 sig = csvread('data.csv'); % сигнал с осциллографа
3 Td = sig(2,1)-sig(1,1); % период дискретизаци
4 t=linspace(0, 24999*Td, 25000); % интервал времени для идеального сигнала
5 f = 1000; % частота идеального сигнала
6 ideal_sin = -2.9 / 2 * sin(2 * pi * f * t) + 3.3 / 2 + 0.05; % идеальный сигнал
7 ideal_sin=ideal_sin.';
```

```
plot(t, sig(:,2), 'LineWidth', 0.5)
hold on;
plot(t, ideal_sin, 'LineWidth', 2)
xlabel('Время (с)');
ylabel('Напряжение (В)');
```

В итоге получим следующее изображение, из которого можно заметить, что сигнал с генератора не совпадает с идеальным, то есть его частота не составляет ровно 1 кГц (рис. 3.23.).

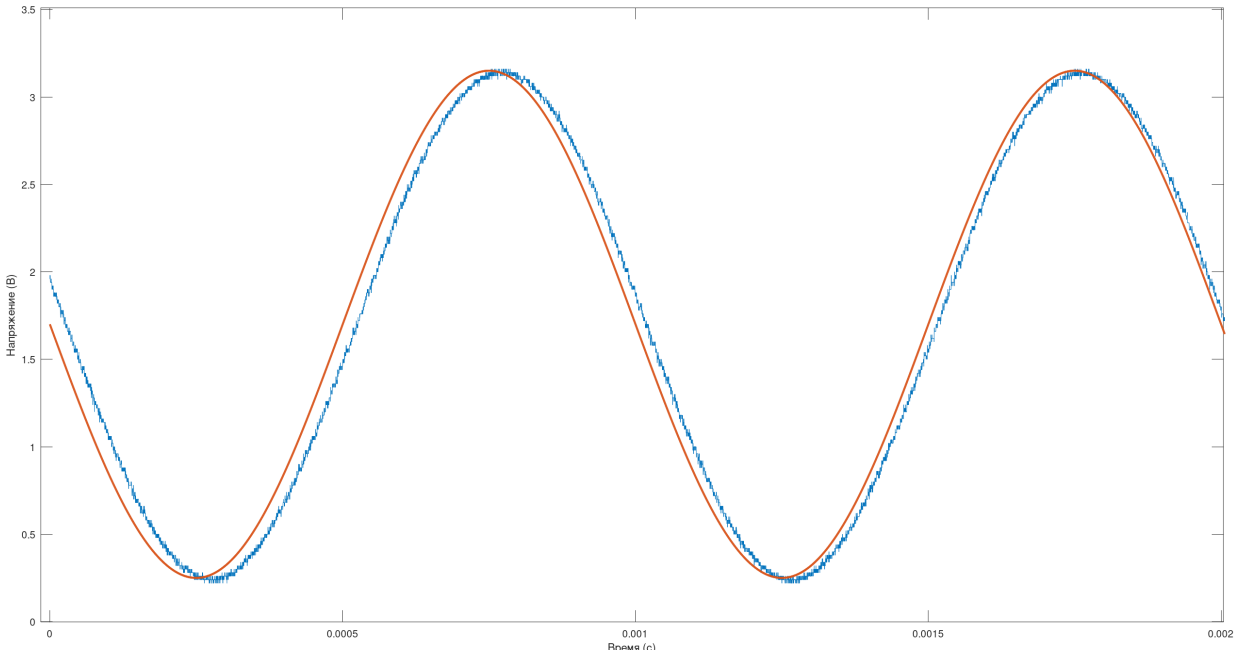


Рис. 3.23 Сравнение записанного и идеального сигналов.

Проведём спектральный анализ нашего сигнала, перейдя от временной составляющей к частотной с помощью быстрого преобразования Фурье [38].

Листинг 3.2

Спектральный анализ

```
fft_ideal = abs(fft(ideal_sin)/25000);
data_sig = sig(:, 2);
fft_sig = abs(fft(data_sig)/25000);
ff=linspace(0,24999/25000*1/Td, 1/Td/25000);
ff=ff.'

output_data_fft = zeros(length(fft_ideal), 2);
output_data_fft(:, 1) = fft_ideal;
output_data_fft(:, 2) = fft_sig;
```

```
10
11 csvwrite('fft.CSV', output_data_fft);
12 csvwrite('ff.CSV', ff);
```

На рис. 3.24 можно заметить смещение по частоте и утечку фазы.

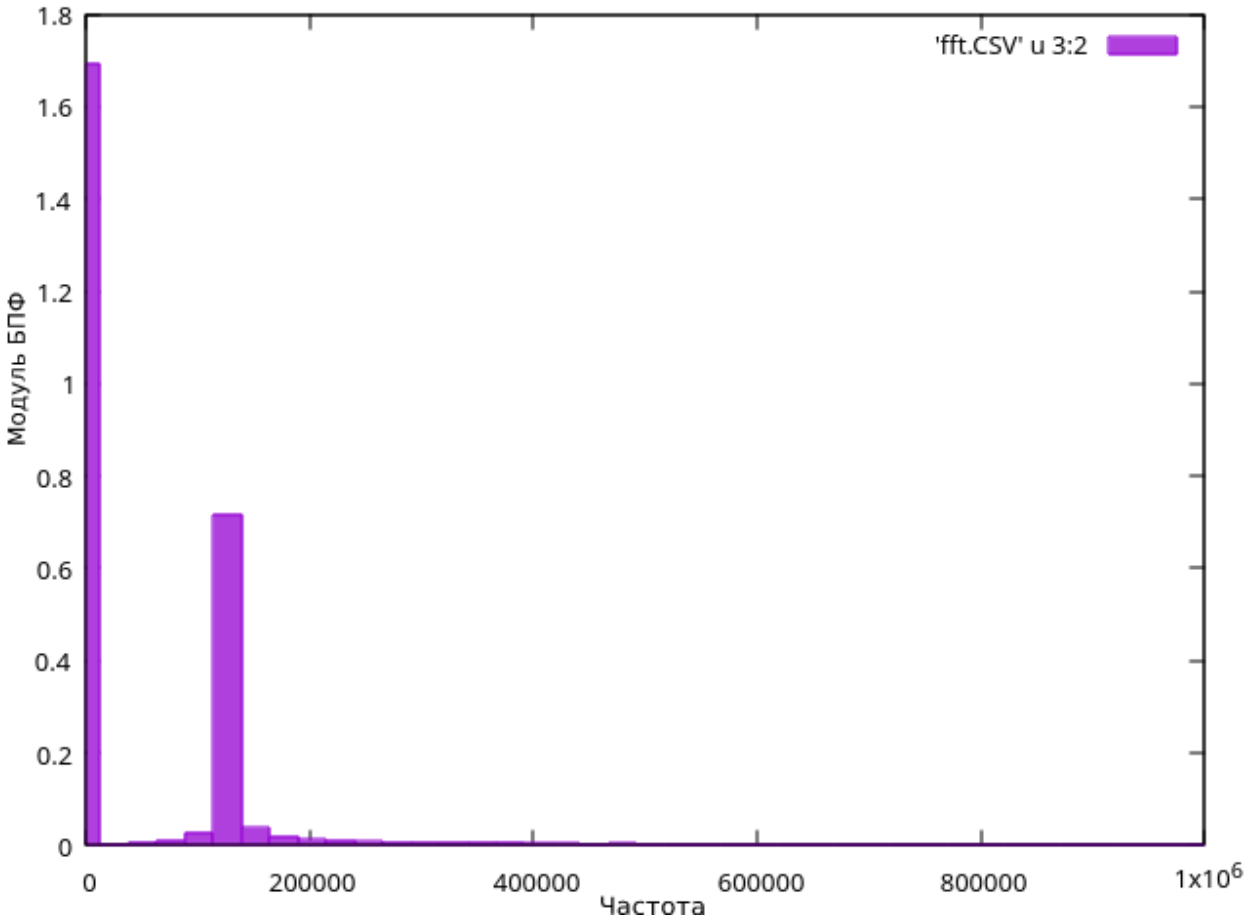


Рис. 3.24 Спектр сигнала.

3.3. Вывод по третьей главе

Таким образом, генератор сигналов был реализован в виде макета. Тестирование показало, что устройство успешно выполняет заданные функции. Были проверены различные формы сигналов, а также возможность регулирования частоты с заданным шагом. Дополнительно был проведен анализ сигнала с помощью сравнения с построенным идеальным и рассмотрением спектра.

В качестве возможных усовершенствований можно рассмотреть:

1. Увеличение разрешения регулирования частоты и диапазона частот;

2. Установка фильтра на выход цифро-аналогового преобразователя;
3. Возможность регулировки амплитуды;
4. Возможность регулировки фазы;
5. Разработка печатной платы и корпуса.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной выпускной квалификационной работы была достигнута поставленная цель — разработан программный генератор сигналов на микроконтроллере STM32F103RCT6, позволяющий генерировать сигналы разной формы, со следующими характеристиками:

- Формы сигналов: синус, треугольник, прямоугольник, пилообразная, обратная пилообразная.
- Частота сигнала: 125 — 50000 Гц.
- Амплитуда: 3 В.
- Шаг по частоте: 125, 250, 500, 1000 Гц.

Помимо микроконтроллера генератор состоит из дисплея с разрешением 128 на 64 пикселя, работающего по интерфейсу I2C, и пяти кнопок управления.

Для достижения поставленной цели были выполнены все задачи, а именно:

1. Выбран метод генерации сигналов;
2. Выбран микроконтроллер;
3. Выбрана среда разработки;
4. Разработана программа;
5. Спроектировано устройство;
6. Протестирован генератор.

Реализованный генератор сигналов отличается простотой, так как использует встроенный цифро-аналоговый преобразователь микроконтроллера и тем самым компактен, а также доступные элементы периферии ввиду этого также его плюсом является невысокая стоимость.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Дьяконов, В. П. Генерация и генераторы сигналов. Москва : ДМК Пресс, 2009. – 384 с.
2. Хоровиц, П. Искусство схемотехники / П. Хоровиц, У. Хилл ; П. Хоровиц, У. Хилл; Пер. с англ. Б.Н. Бронина [и др.]. – 6. изд.. – Москва : Мир, 2003. – 704 с.
3. Исследование способов генерации сигналов [Электронный ресурс] Лаборатория Электронных Средств Обучения (ЛЭСО) СибГУТИ. Режим доступа: http://www.labfor.ru/guidance/fpga-dsp/dds.
4. Аминев, А. В. Основы радиоэлектроники: измерения в телекоммуникационных системах : Учебное пособие / А. В. Аминев, А. В. Блохин. – 1-е изд.. – Москва : Издательство Юрайт, 2018. – 223 с.
5. Vankka J., Halonen K. A. I. Direct digital synthesizers: theory, design and applications. – Springer Science & Business Media, 2001. – Т. 614.
6. Volder J. E. The CORDIC trigonometric computing technique //IRE Transactions on electronic computers. – 1959. – №. 3. – С. 330-334.
7. Беспалов, Н. Н. Применение итерационного метода CORDIC для реализации алгоритма трёхфазного генератора / Н. Н. Беспалов, А. В. Волков, А. Д. Ваничкин // Научно-технический вестник Поволжья. – 2020. – № 7. – С. 43-46.
8. Габдуллин, Р. Б. Проектирование генератора синусоидального тестового сигнала табличным способом на ПЛИС / Р. Б. Габдуллин // Молодежный научный форум : сборник статей по материалам LXIV студенческой международной научно-практической конференции, Москва, 20 ноября 2019 года. Том 34 (64). – Москва: Общество с ограниченной ответственностью ”Международный центр науки и образования”, 2019. – С. 46-50.

9. Ридико Л. DDS: прямой цифровой синтез частоты //Компоненты и технологии. – 2001. – №. 17. – С. 50-56.

10. Cordesses L. Direct digital synthesis: A tool for periodic wave generation (part 1) //IEEE Signal processing magazine. – 2004. – Т. 21. – №. 4. – С. 50-54.

11. Мерфи, Е. Прямой цифровой синтез DDS в тестовом, измерительном и коммуникацинном оборудовании / Е. Мерфи, К. Слэттери, А. Власенко // Компоненты и технологии. – 2006. – № 8(61). – С. 52-55.

12. AWG-4112 Генератор сигналов специальной формы. [Электронный ресурс] Официальный сайт АКТАКОМ. Режим доступа: https://www.aktakom.ru/kio/index.php?SECTION_ID=2087&ELEMENT_ID=11289154.

13. Мерфи, Е. Всё о синтезаторах DDS / Е. Мерфи, А. Власенко // Компоненты и технологии. – 2005. – № 1(45). – С. 28-32.

14. Datasheet - AD9833. [Электронный ресурс] Официальный сайт Analog Devices. Режим доступа: https://www.analog.com/media/en/technical-documentation/data-sheets/ad9833.pdf.

15. Генератор сигналов для DAC. [Электронный ресурс] PROGCONT.RU. Режим доступа: https://progcont.ru/?articles=54&category_articles=ALL.

16. Мокринский, А. А. Микроконтроллеры AVR и STM32 / А. А. Мокринский, А. К. Бойцов // Информационные системы и технологии: теория и практика : Сборник научных трудов / Отв. редактор М.Р. Вагизов. Том Выпуск 15. – Санкт-Петербург : Санкт-Петербургский государственный лесотехнический университет имени С.М. Кирова, 2023. – С. 120-125.

17. Евстифеев, А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ”ATMEL” / А. В. Евстифеев ; А.В. Евстифеев. – Москва : Додэка-XXI, 2004. – 558 с. – (Мировая электроника).

18. Конченков, В. И. Семейство микроконтроллеров STM32. Программирование и применение : учебное пособие / В. И. Конченков, В. Н. Скакунов. – Волгоград : Волгоградский государственный технический университет, 2015. – 78 с.

19. Datasheet - STM32F103xC, STM32F103xD, STM32F103xE. [Электронный ресурс] Официальный сайт STMicroelectronics. Режим доступа: https://www.st.com/resource/en/datasheet/stm32f103rc.pdf.

20. Datasheet - STM32L010F4, STM32L010K4. [Электронный ресурс] Официальный сайт STMicroelectronics. Режим доступа: https://www.st.com/resource/en/datasheet/stm32l010f4.pdf.

21. Datasheet - ATmega32/L. [Электронный ресурс] Официальный сайт Microchip. Режим доступа: https://ww1.microchip.com/downloads/en/DeviceDoc/doc2503.pdf.

22. Datasheet - ATtiny4 / ATtiny5 / ATtiny9 / ATtiny10. [Электронный ресурс] Официальный сайт Microchip. Режим доступа: https://ww1.microchip.com/downloads/en/DeviceDoc/atmel-8127-avr-8-bit-microcontroller-attiny4-attiny5-attiny9-attiny10_datasheet.pdf.

23. STM32CubeIDE [Электронный ресурс] Официальный сайт STMicroelectronics. Режим доступа: https://www.st.com/en/development-tools/stm32cubeide.html.

24. Description of STM32F1 HAL and low-layer drivers [Электронный ресурс] Официальный сайт STMicroelectronics. Режим доступа: https://www.st.com/resource/en/user_manual/um1850-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf.

25. PlatformIO [Электронный ресурс] Официальный сайт PlatformIO. Режим доступа: https://platformio.org/.

26. Visual Studio Code [Электронный ресурс] Официальный сайт Visual Studio Code. Режим доступа: https://code.visualstudio.com/.

27. Docs PlatformIO [Электронный ресурс] Официальный сайт PlatformIO. Режим доступа: https://docs.platformio.org/en/latest/frameworks/index.html.

28. LibOpenCM3 [Электронный ресурс] Официальный сайт LibOpenCM3. Режим доступа: https://libopencm3.org/.

29. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. / Под ред. и с предисл. Вс.С.Штаркмана. - 2-е изд., перераб. и доп. - М.: Финансы и статистика, 1992. – 272 с.

30. Дейтел, Х. М. Как программировать на C / Х. М. Дейтел ; Х.М. Дейтел, П. Дж. Дейтел ; пер. с англ. под ред. В.В. Тимофеева. – 4-е изд.. – Москва : Бином, 2006. – 908 с.

31. STM32: тактирование таймеров от APB1 и APB2. [Электронный ресурс] Microsin — заметки радиолюбителя. Режим доступа: https://microsin.net / programming / arm / stm32 - timers - clocked - from - internal - clock - frequency-apb1-i-apb2.html.

32. Угрюмов Е. П. Цифровая схемотехника: учебное пособие для вузов. 2 е издание //СПб.: БХВ-Петербург. – 2005. – С. 634.

33. ssd1306_libopencm3. [Электронный ресурс] GitHub репозиторий. Режим доступа: https://github.com/StanislavLakhtin/ssd1306_libopencm3/tree/master.

34. STM32F103C8 без HAL и SPL: Работа с монохромными дисплеями STE2007 и SSD1306. [Электронный ресурс] CountZero: Technology, Electronics, Algorithms. Режим доступа: https://count-zero.ru/2023/ste2007/.

35. Audio and waveform generation using the DAC in STM32 products. [Электронный ресурс] Официальный сайт STMicroelectronics. Режим доступа: https://www.st.com/resource/en/application_note/an3126-audio-and-waveform-generation-using-the-dac-in-stm32-products-stmicroelectronics.pdf.

36. Шаталов, Д. А. Применение подтягивающих и заземляющих резисторов при проектировании цифровых устройств / Д. А. Шаталов, В. Е. Драч // Новая наука: От идеи к результату. – 2016. – № 3-1(72). – С. 28-31.

37. GNU Octave. [Электронный ресурс] Официальный сайт GNU Octave. Режим доступа: https://octave.org/.

38. Лайонс, Р. Цифровая обработка сигналов / Р. Лайонс ; Ричард Лайонс ; пер. с англ. под ред. А. А. Бритова. – Москва : БИНОМ, 2007. – 652 с.

ПРИЛОЖЕНИЕ

Программа генератора сигналов.

```
#include <stdio.h>
#include <wchar.h>
#include <libopencm3/stm32/rcc.h>
#include <libopencm3/stm32/flash.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/timer.h>
#include <libopencm3/cm3/nvic.h>
#include <libopencm3/stm32/dac.h>
#include <libopencm3/stm32/i2c.h>
#include <ssd1306_i2c.h>

static void gpio_setup(void);   // установить входы/выходы
static void dac_setup(void);    // настройка цап
static void i2c_setup(void);    // настройка и2ц
static void timers_setup(void); // настройка таймеров
static void nvic_setup(void);   // настройка прерываний
void minus_freq(void);          //
void plus_freq(void);           //
void minus_signal(void);        // функции для кнопок
void plus_signal(void);         //
void step_select(void);         //

uint16_t p_acc = 0;         // аккумулятор фазы
int p_step = 0;             // код частоты 192 - 1khz
uint16_t step = 0;          // размер шага
uint16_t signal[256] = {0}; // буфер для цапа
int8_t num_sig = 0;         // номер сигнала
int8_t num_step = 0;        // номер шага

/*  отсчеты сигналов    */
uint16_t sinus[256] = {2048, 2092, 2136, 2180, 2224, 2268, 2312, 2355, 2399, 2442,
                2485, 2527, 2570, 2612, 2654, 2695, 2736, 2777, 2817, 2857, 2896, 2934, 2973, 3010, 3047,
                3084, 3119, 3155, 3189, 3223, 3256, 3288, 3320, 3351, 3381, 3410, 3439, 3466, 3493, 3519,
                3544, 3568, 3591, 3613, 3635, 3655, 3674, 3693, 3710, 3726, 3742, 3756, 3770, 3782, 3793,
                3803, 3812, 3821, 3828, 3833, 3838, 3842, 3845, 3846, 3847, 3846, 3845, 3842, 3838, 3833,
                3828, 3821, 3812, 3803, 3793, 3782, 3770, 3756, 3742, 3726, 3710, 3693, 3674, 3655, 3635,
                3613, 3591, 3568, 3544, 3519, 3493, 3466, 3439, 3410, 3381, 3351, 3320, 3288, 3256, 3223,
                3189, 3155, 3119, 3084, 3047, 3010, 2973, 2934, 2896, 2857, 2817, 2777, 2736, 2695, 2654,
                2612, 2570, 2527, 2485, 2442, 2399, 2355, 2312, 2268, 2224, 2180, 2136, 2092, 2048, 2003,
                1959, 1915, 1871, 1827, 1783, 1740, 1696, 1653, 1610, 1568, 1525, 1483, 1441, 1400, 1359,
                1318, 1278, 1238, 1199, 1161, 1122, 1085, 1048, 1011, 976, 940, 906, 872, 839, 807, 775,
                744, 714, 685, 656, 629, 602, 576, 551, 527, 504, 482, 460, 440, 421, 402, 385, 369, 353,
                339, 325, 313, 302, 292, 283, 274, 267, 262, 257, 253, 250, 249, 248, 249, 250, 253, 257,
                262, 267, 274, 283, 292, 302, 313, 325, 339, 353, 369, 385, 402, 421, 440, 460, 482, 504,
                527, 551, 576, 602, 629, 656, 685, 714, 744, 775, 807, 839, 872, 906, 940, 976, 1011, 1048,
                1085, 1122, 1161, 1199, 1238, 1278, 1318, 1359, 1400, 1441, 1483, 1525, 1568, 1610, 1653,
                1696, 1740, 1783, 1827, 1871, 1915, 1959, 2003};
/*                  */
```

```
uint16_t square[256] = {3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
                    3847, 3847, 3847, 3847, 3847, 3847, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
                    248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
                    248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
                    248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
                    248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
                    248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
                    248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
                    248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248};
/*                  */
uint16_t triangle[256] = {248, 276, 304, 332, 360, 389, 417, 445, 473, 501, 529, 557, 585, 614, 642, 670, 698,
                    726, 754, 782, 810, 838, 867, 895, 923, 951, 979, 1007, 1035, 1063, 1092, 1120, 1148, 1176, 1204, 1232, 1260,
                    1288, 1316, 1345, 1373, 1401, 1429, 1457, 1485, 1513, 1541, 1570, 1598, 1626, 1654, 1682, 1710, 1738, 1766, 1794,
                    1823, 1851, 1879, 1907, 1935, 1963, 1991, 2019, 2048, 2076, 2104, 2132, 2160, 2188, 2216, 2244, 2272, 2301, 2329,
                    2357, 2385, 2413, 2441, 2469, 2497, 2525, 2554, 2582, 2610, 2638, 2666, 2694, 2722, 2750, 2779, 2807, 2835, 2863,
                    2891, 2919, 2947, 2975, 3003, 3032, 3060, 3088, 3116, 3144, 3172, 3200, 3228, 3257, 3285, 3313, 3341, 3369, 3397,
                    3425, 3453, 3481, 3510, 3538, 3566, 3594, 3622, 3650, 3678, 3706, 3735, 3763, 3791, 3819, 3847, 3819, 3791, 3763,
                    3735, 3706, 3678, 3650, 3622, 3594, 3566, 3538, 3510, 3481, 3453, 3425, 3397, 3369, 3341, 3313, 3285, 3257, 3228,
                    3200, 3172, 3144, 3116, 3088, 3060, 3032, 3003, 2975, 2947, 2919, 2891, 2863, 2835, 2807, 2779, 2750, 2722, 2694,
                    2666, 2638, 2610, 2582, 2554, 2525, 2497, 2469, 2441, 2413, 2385, 2357, 2329, 2301, 2272, 2244, 2216, 2188, 2160,
                    2132, 2104, 2076, 2048, 2019, 1991, 1963, 1935, 1907, 1879, 1851, 1823, 1794, 1766, 1738, 1710, 1682, 1654, 1626,
                    1598, 1570, 1541, 1513, 1485, 1457, 1429, 1401, 1373, 1345, 1316, 1288, 1260, 1232, 1204, 1176, 1148, 1120, 1092,
                    1063, 1035, 1007, 979, 951, 923, 895, 867, 838, 810, 782, 754, 726, 698, 670, 642, 614, 585, 557, 529, 501, 473,
                    445, 417, 389, 360, 332, 304, 276};
/*                  */
uint16_t l_saw[256] = {248, 262, 276, 290, 304, 319, 333, 347, 361, 375, 389, 403, 417, 431, 446, 460, 474,
                    488, 502, 516, 530, 544, 559, 573, 587, 601, 615, 629, 643, 657, 671, 686, 700, 714, 728, 742, 756, 770, 784,
                    798, 813, 827, 841, 855, 869, 883, 897, 911, 925, 940, 954, 968, 982, 996, 1010, 1024, 1038, 1052, 1067, 1081,
                    1095, 1109, 1123, 1137, 1151, 1165, 1180, 1194, 1208, 1222, 1236, 1250, 1264, 1278, 1292, 1307, 1321, 1335, 1349,
                    1363, 1377, 1391, 1405, 1419, 1434, 1448, 1462, 1476, 1490, 1504, 1518, 1532, 1546, 1561, 1575, 1589, 1603, 1617,
                    1631, 1645, 1659, 1673, 1688, 1702, 1716, 1730, 1744, 1758, 1772, 1786, 1801, 1815, 1829, 1843, 1857, 1871, 1885,
                    1899, 1913, 1928, 1942, 1956, 1970, 1984, 1998, 2012, 2026, 2040, 2055, 2069, 2083, 2097, 2111, 2125, 2139, 2153,
                    2167, 2182, 2196, 2210, 2224, 2238, 2252, 2266, 2280, 2294, 2309, 2323, 2337, 2351, 2365, 2379, 2393, 2407, 2422,
                    2436, 2450, 2464, 2478, 2492, 2506, 2520, 2534, 2549, 2563, 2577, 2591, 2605, 2619, 2633, 2647, 2661, 2676, 2690,
                    2704, 2718, 2732, 2746, 2760, 2774, 2788, 2803, 2817, 2831, 2845, 2859, 2873, 2887, 2901, 2915, 2930, 2944, 2958,
                    2972, 2986, 3000, 3014, 3028, 3043, 3057, 3071, 3085, 3099, 3113, 3127, 3141, 3155, 3170, 3184, 3198, 3212, 3226,
                    3240, 3254, 3268, 3282, 3297, 3311, 3325, 3339, 3353, 3367, 3381, 3395, 3409, 3424, 3438, 3452, 3466, 3480, 3494,
                    3508, 3522, 3536, 3551, 3565, 3579, 3593, 3607, 3621, 3635, 3649, 3664, 3678, 3692, 3706, 3720, 3734, 3748, 3762,
                    3776, 3791, 3805, 3819, 3833, 3847};
/*                  */
uint16_t r_saw[256] = {3847, 3833, 3819, 3805, 3791, 3776, 3762, 3748, 3734, 3720, 3706, 3692, 3678, 3664,
                    3649, 3635, 3621, 3607, 3593, 3579, 3565, 3551, 3536, 3522, 3508, 3494, 3480, 3466, 3452, 3438, 3424, 3409,
                    3395, 3381, 3367, 3353, 3339, 3325, 3311, 3297, 3282, 3268, 3254, 3240, 3226, 3212, 3198, 3184, 3170, 3155,
                    3141, 3127, 3113, 3099, 3085, 3071, 3057, 3043, 3028, 3014, 3000, 2986, 2972, 2958, 2944, 2930, 2915, 2901,
                    2887, 2873, 2859, 2845, 2831, 2817, 2803, 2788, 2774, 2760, 2746, 2732, 2718, 2704, 2690, 2676, 2661, 2647,
                    2633, 2619, 2605, 2591, 2577, 2563, 2549, 2534, 2520, 2506, 2492, 2478, 2464, 2450, 2436, 2422, 2407, 2393,
```

```
103                 2379, 2365, 2351, 2337, 2323, 2309, 2294, 2280, 2266, 2252, 2238, 2224, 2210, 2196, 2182, 2167, 2153, 2139,
104                 2125, 2111, 2097, 2083, 2069, 2055, 2040, 2026, 2012, 1998, 1984, 1970, 1956, 1942, 1928, 1913, 1899, 1885,
105                 1871, 1857, 1843, 1829, 1815, 1801, 1786, 1772, 1758, 1744, 1730, 1716, 1702, 1688, 1673, 1659, 1645, 1631,
106                 1617, 1603, 1589, 1575, 1561, 1546, 1532, 1518, 1504, 1490, 1476, 1462, 1448, 1434, 1419, 1405, 1391, 1377,
107                 1363, 1349, 1335, 1321, 1307, 1292, 1278, 1264, 1250, 1236, 1222, 1208, 1194, 1180, 1165, 1151, 1137, 1123,
108                 1109, 1095, 1081, 1067, 1052, 1038, 1024, 1010, 996, 982, 968, 954, 940, 925, 911, 897, 883, 869, 855, 841,
109                 827, 813, 798, 784, 770, 756, 742, 728, 714, 700, 686, 671, 657, 643, 629, 615, 601, 587, 573, 559, 544, 530,
110                 516, 502, 488, 474, 460, 446, 431, 417, 403, 389, 375, 361, 347, 333, 319, 304, 290, 276, 262, 248};
111 /*                */
112 void tim2_isr(void) // обработчик прерывания таймера2 (ЦАП)
113 {
114     dac_load_data_buffer_single(signal[p_acc >> 8], RIGHT12, CHANNEL_2); // загрузка буфера в цап
115     p_acc += p_step;                                    // шаг
116     TIM_SR(TIM2) &= ~TIM_SR_UIF;                                // очистка флага прерывания
117 }
118
119 void tim3_isr(void) // обработчик прерывания таймера3 (обработка кнопок)
120 {
121     minus_freq();
122     plus_freq();
123     minus_signal(); // функции кнопок
124     plus_signal();
125     step_select();
126     TIM_SR(TIM3) &= ~TIM_SR_UIF; // очистка флага прерывания
127 }
128
129 int main(void)
130 {
131     rcc_clock_setup_in_hse_8mhz_out_72mhz(); // установка тактирования
132     gpio_setup();
133     nvic_setup();
134     dac_setup();
135     timers_setup();
136     i2c_setup();
137     ssd1306_init(I2C2, DEFAULT_7bit_OLED_SLAVE_ADDRESS, 128, 64); // инициализация дисплея
138
139     int f = 0;       // переменная частоты
140     wchar_t freq[8]; // буфер для wchar_t строки
141     while (1)
142     {
143         f = p_step / 24 * 125;
144         swprintf(freq, sizeof(freq) / sizeof(wchar_t), L"%d", f); // Использование swprintf для преобразования int в wchar_t*
145         /*  вывод информации на дисплей  */
146         ssd1306_clear();
147         ssd1306_drawWCharStr(0, 0, white, nowrap, L"Форма сигнала:");
148         switch (num_sig)
149         {
150         case 1:
151             ssd1306_drawWCharStr(0, 8, white, nowrap, L"Синус");
152             break;
153         case 2:
154             ssd1306_drawWCharStr(0, 8, white, nowrap, L"Меандр");
155             break;
156         case 3:
```

```
157             ssd1306_drawWCharStr(0, 8, white, nowrap, L"Треугольник");
158             break;
159         case 4:
160             ssd1306_drawWCharStr(0, 8, white, nowrap, L"Пила Левая");
161             break;
162         case 5:
163             ssd1306_drawWCharStr(0, 8, white, nowrap, L"Пила Правая");
164             break;
165         }
166         ssd1306_drawWCharStr(0, 16, white, nowrap, L"Частота(Гц)");
167         ssd1306_drawWCharStr(64, 16, white, nowrap, freq);
168         ssd1306_drawWCharStr(0, 32, white, nowrap, L"Шаг(Гц)");
169         switch (num_step)
170         {
171         case 1:
172             ssd1306_drawWCharStr(64, 32, white, nowrap, L"125");
173             break;
174         case 2:
175             ssd1306_drawWCharStr(64, 32, white, nowrap, L"250");
176             break;
177         case 3:
178             ssd1306_drawWCharStr(64, 32, white, nowrap, L"500");
179             break;
180         case 4:
181             ssd1306_drawWCharStr(64, 32, white, nowrap, L"1000");
182             break;
183         }
184         ssd1306_refresh();
185     }
186 
187     return 0;
188 }
189 
190 static void gpio_setup(void)
191 {
192     // rcc_periph_clock_enable(RCC_GPIOD); // тактирование портов
193     rcc_periph_clock_enable(RCC_GPIOB);
194     gpio_set_mode(GPIOB, GPIO_MODE_INPUT, GPIO_CNF_INPUT_PULL_UPDOWN, GPIO9 | GPIO5 | GPIO6 | GPIO7 | GPIO8); //
      ↪  входы для кнопок, подтянуты к земле
195     // gpio_set_mode(GPIOD, GPIO_MODE_OUTPUT_50_MHZ, GPIO_CNF_OUTPUT_PUSHPULL, GPIO2);
196 }
197 
198 static void dac_setup(void)
199 {
200     rcc_periph_clock_enable(RCC_GPIOA);
201     gpio_set_mode(GPIOA, GPIO_MODE_OUTPUT_2_MHZ, GPIO_CNF_OUTPUT_ALTFN_PUSHPULL, GPIO5);
202     rcc_periph_clock_enable(RCC_DAC); // тактирование цапа и настройка вывода
203     dac_enable(CHANNEL_2);          // включить цап
204 }
205 
206 static void i2c_setup(void){
207     // Включение тактирования периферийного оборудования для I2C2
208     rcc_periph_clock_enable(RCC_I2C2);
209 
```

```
210     /*
211      * Настройка альтернативных функций для пинов SCL и SDA интерфейса I2C2.
212      * Это необходимо для подключения I2C устройств к микроконтроллеру через эти
213  пины.
214      */
215     gpio_set_mode(GPIOB, GPIO_MODE_OUTPUT_50_MHZ,
216                   GPIO_CNF_OUTPUT_ALTFN_OPENDRAIN,
217                   GPIO_I2C2_SCL | GPIO_I2C2_SDA);
218
219     // Отключение I2C перед изменением конфигурации
220     i2c_peripheral_disable(I2C2);
221
222     // Сброс состояния периферийного устройства I2C2
223     i2c_reset(I2C2);
224
225     // Установка стандартного режима работы I2C
226     i2c_set_standard_mode(I2C2);
227
228     // Установка частоты периферии
229     i2c_set_clock_frequency(I2C2, I2C_CR2_FREQ_36MHZ);
230
231     // Настройка тактовой частоты шины;
232     i2c_set_ccr(I2C2, 0xB4);
233
234     // Установка времени подъема сигнала SDA после завершения операции чтения/записи
235     i2c_set_trise(I2C2, 0x25);
236
237     // Включение подтверждения при получении данных от устройства
238     i2c_enable_ack(I2C2);
239
240     // Включение периферийного устройства I2C2
241     i2c_peripheral_enable(I2C2);
242 }
243
244
245 static void timers_setup(void)
246 {
247     rcc_periph_clock_enable(RCC_TIM2);
248     rcc_periph_clock_enable(RCC_TIM3);
249
250     /* Стартовое значение таймера */
251     TIM_CNT(TIM2) = 0;
252     TIM_CNT(TIM3) = 0;
253
254     /* Предделитель 36MHz/36000 => 1000 отсчетов в секунду (счет начинается с 0, поэтому в предделителе и периоде нужно
      ↪  отнимать единичку) */
255     TIM_PSC(TIM2) = 17;
256     TIM_PSC(TIM3) = 35999;
257
258     /* Период таймера */
259     TIM_ARR(TIM2) = 9;
260     TIM_ARR(TIM3) = 249;
261
262     /* Включить прерывания */
```

```
263     TIM_DIER(TIM2) |= TIM_DIER_UIE;
264     TIM_DIER(TIM3) |= TIM_DIER_UIE;
265
266     /* Запустить таймер */
267     TIM_CR1(TIM2) |= TIM_CR1_CEN;
268     TIM_CR1(TIM3) |= TIM_CR1_CEN;
269 }
270
271 static void nvic_setup(void)
272 {
273     /* Активировать прерывания и установить приоритеты */
274     nvic_enable_irq(NVIC_TIM2_IRQ);
275     nvic_set_priority(NVIC_TIM2_IRQ, 2);
276
277     nvic_enable_irq(NVIC_TIM3_IRQ);
278     nvic_set_priority(NVIC_TIM3_IRQ, 1);
279 }
280
281 void minus_freq(void)
282 {
283     bool cur_val = 0;
284     bool prev_val = 0;
285     cur_val = gpio_get(GPIOB, GPIO5);
286     if (cur_val == 1 && prev_val == 0)
287     {
288         p_step -= step;
289     }
290     if (p_step < 0) // ограничение 0
291     {
292         p_step = 0;
293     }
294     prev_val = cur_val;
295 }
296
297 void plus_freq(void)
298 {
299     bool cur_val = 0;
300     bool prev_val = 0;
301     cur_val = gpio_get(GPIOB, GPIO6);
302     if (cur_val == 1 && prev_val == 0)
303     {
304         p_step += step;
305     }
306     if (p_step > 9600) // ограничение 50 кГц
307     {
308         p_step = 9600;
309     }
310     prev_val = cur_val;
311 }
312
313 void minus_signal(void)
314 {
315     bool cur_val = 0;
316     bool prev_val = 0;
```

```
317     cur_val = gpio_get(GPIOB, GPIO7);
318     if (cur_val == 1 && prev_val == 0)
319     {
320         dac_disable(CHANNEL_2);
321         num_sig -= 1;
322         if (num_sig < 1)
323             num_sig = 1;
324         switch (num_sig)
325         {
326         case 1:
327             for (int i = 0; i < 256; i++)
328                 signal[i] = sinus[i];
329             break;
330         case 2:
331             for (int i = 0; i < 256; i++)
332                 signal[i] = square[i];
333             break;
334         case 3:
335             for (int i = 0; i < 256; i++)
336                 signal[i] = triangle[i];
337             break;
338         case 4:
339             for (int i = 0; i < 256; i++)
340                 signal[i] = l_saw[i];
341             break;
342         case 5:
343             for (int i = 0; i < 256; i++)
344                 signal[i] = r_saw[i];
345             break;
346         }
347         dac_enable(CHANNEL_2);
348     }
349     prev_val = cur_val;
350 }
351
352 void plus_signal(void)
353 {
354     bool cur_val = 0;
355     bool prev_val = 0;
356     cur_val = gpio_get(GPIOB, GPIO8);
357     if (cur_val == 1 && prev_val == 0)
358     {
359         dac_disable(CHANNEL_2);
360         num_sig += 1;
361         switch (num_sig)
362         {
363         case 1:
364             for (int i = 0; i < 256; i++)
365                 signal[i] = sinus[i];
366             break;
367         case 2:
368             for (int i = 0; i < 256; i++)
369                 signal[i] = square[i];
370             break;
```

```
371         case 3:
372             for (int i = 0; i < 256; i++)
373                 signal[i] = triangle[i];
374             break;
375         case 4:
376             for (int i = 0; i < 256; i++)
377                 signal[i] = l_saw[i];
378             break;
379         case 5:
380             for (int i = 0; i < 256; i++)
381                 signal[i] = r_saw[i];
382             break;
383         }
384         if (num_sig > 5)
385             num_sig = 5;
386         dac_enable(CHANNEL_2);
387     }
388     prev_val = cur_val;
389 }
390 
391 void step_select(void)
392 {
393     bool cur_val = 0;
394     bool prev_val = 0;
395     cur_val = gpio_get(GPIOB, GPIO9);
396     if (cur_val == 1 && prev_val == 0)
397     {
398         num_step += 1;
399         switch (num_step)
400         {
401         case 1:
402             step = 24; // 125 Гц
403             break;
404         case 2:
405             step = 48; // 250 Гц
406             break;
407         case 3:
408             step = 96; // 500 Гц
409             break;
410         case 4:
411             step = 192; // 1000 Гц
412             break;
413         case 5:
414             step = 24; // 125 Гц
415             num_step = 1;
416             break;
417         }
418     }
419 }
```

ПОСЛЕДНИЙ ЛИСТ ВКР

Выпускная квалификационная работа выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ 2024 г.

_____ Д. С. Вебер