

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

ЦИФРОВОЙ 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

Консультанты: _____

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

Барнаул 2024

РЕФЕРАТ

Объем работы листов	#
Количество рисунков	#
Количество используемых источников	#
Количество таблиц	#

ГЕНЕРАТОР СИГНАЛОВ, МИКРОКОНТРОЛЛЕР.

В первой главе были рассмотрены семейства микроконтроллеров, различные среды разработки и изучены методы программной генерации сигналов.

Во второй главе был спроектирован генератор. Проведено моделирование выбранного метода генерации, разработан алгоритм работы и создана схема электрическая принципиальная.

В третьей главе была произведена сборка макета, написана и протестирована результирующая программа.

Дипломная работа оформлена с помощью системы компьютерной вёрстки $\text{T}_{\text{E}}\text{X}$ и его расширения $\text{X}_{\text{E}}\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ из дистрибутива *TeX Live*.

СОДЕРЖАНИЕ

Введение	5
1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	7
1.1. Развитие генераторов сигналов	7
1.2. Основные типы сигналов	7
1.2.1. Синусоидальные сигналы	8
1.2.2. Линейно-меняющийся сигнал	9
1.2.3. Треугольный сигнал	10
1.2.4. Шум	10
1.2.5. Прямоугольный сигнал	11
1.2.6. Импульсы	11
1.2.7. Скачки и пики	12
1.3. Виды генераторов	12
1.3.1. Генераторы синусоидальных сигналов	13
1.3.2. Функциональные генераторы	13
1.3.3. Генераторы сигналов произвольной формы	14
1.3.4. Генераторы импульсов	14
1.4. Методы цифровой генерации сигнала	15
1.4.1. Метод аппроксимации	15
1.4.2. CORDIC	16
1.4.3. Табличный метод	18
1.4.4. Метод DDS	19
1.5. Вывод из первой главы	21
2. ПРОГРАММНАЯ ЧАСТЬ	22
2.1. Моделирование DDS	22
2.2. Обзор микроконтроллеров	24
2.2.1. AVR	24
2.2.2. STM32	25
2.3. Сравнение семейств AVR и STM32	26

2.4. Среды разработки для STM32	27
2.4.1. STM32CubeIDE	27
2.4.2. PlatformIO	28
2.5. Алгоритм работы	30
2.6. Вывод из второй главы	35
3. АППАРАТНАЯ ЧАСТЬ	36
3.1. Разработка схемы	36
3.2. Вывод из третьей главы	36
ЗАКЛЮЧЕНИЕ	37
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	38
Приложение 1	40

ВВЕДЕНИЕ

В ходе эксплуатации электронных устройств регулярно возникает необходимость в настройке или ремонте. Для калибровки и отладки приборов необходимы колебания разных форм и периодов. Формирование требуемых электрических колебаний 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 цифрового 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. Сконструировать макет.
6. Протестировать генератор.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

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.

1.2.1. Синусоидальные сигналы

Именно синусоидальные сигналы мы извлекаем из розетки. Математическое выражение, описывающее синусоидальное напряжение, имеет вид:

$$U = A \sin 2\pi f t, \quad (1.1)$$

где A — амплитуда сигнала,

f — частота в герцах.

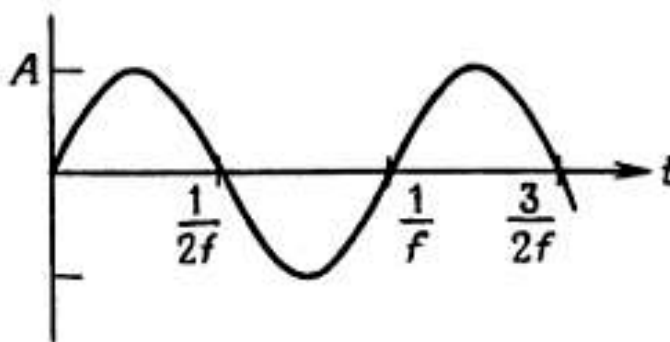


Рис. 1.1 Синусоидальный сигнал.

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:

$$U = A \sin 2\pi f t + \theta \quad (1.2)$$

Синусоидальные сигналы характеризуются тремя параметрами:

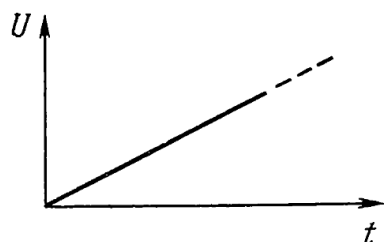
- U_M или I_M — амплитуда переменного напряжения или тока;
- f — частота (период);

- θ — фазовый сдвиг.

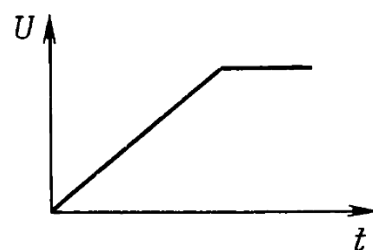
У синусоиды есть своё достоинство в том, что функция данной формы сигнала является решением многих дифференциальных уравнений, которые описывают как физические явления, так и свойства линейных цепей [2]. На практике поведение схемы оценивают по её амплитудно-частотной характеристике (АЧХ), которая показывает, 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.

1.2.2. Линейно-меняющийся сигнал

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 становится пилообразным (рис. 1.3).

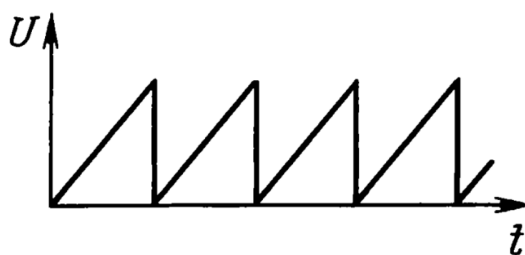


Рис. 1.3 Пилообразный сигнал.

1.2.3. Треугольный сигнал

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

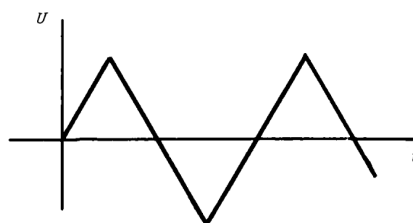


Рис. 1.4 Треугольный сигнал.

1.2.4. Шум

Также существует потребность в генерации шумов для анализа реакции на такой сигнал схемы. Характеризуется частотным спектром (произведение мощности на частоту в герцах) [2]. Самое распространённое шумовое напряжение — белый шум с распределением Гаусса.

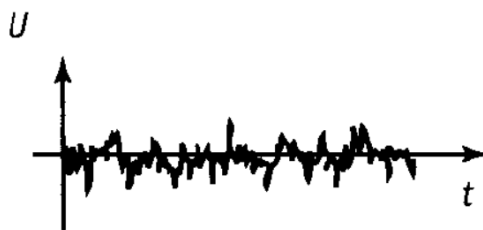


Рис. 1.5 Сигнал шума.

1.2.5. Прямоугольный сигнал

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

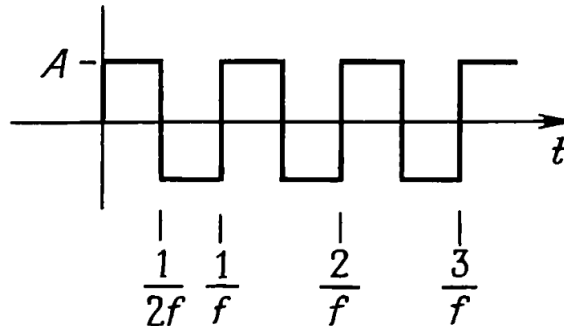


Рис. 1.6 Прямоугольный сигнал.

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].

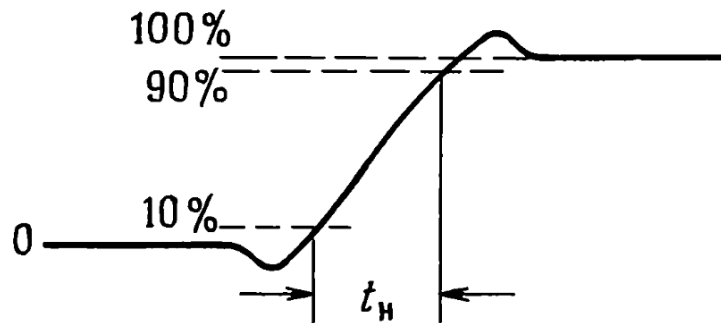


Рис. 1.7 Время нарастания скачка прямоугольного сигнала.

На рисунке 1.7 изображено как обычно выглядит скачок сигнала прямоугольника. Время когда сигнал нарастёт определяется в промежутке от 10 до 90% максимальной амплитуды сигнала.

1.2.6. Импульсы

Сигналы в виде импульса изображены на рисунке 1.8.

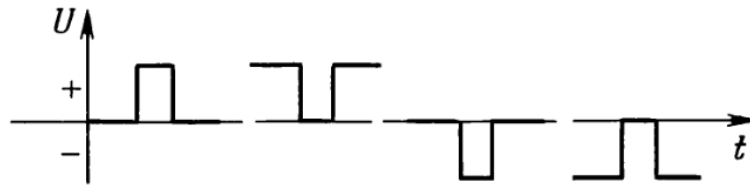


Рис. 1.8 Импульсы.

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.

1.2.7. Скачки и пики

Часто можно слышать о сигналах в виде скачков и пиков, но на самом деле широкого применения они не находят. С помощью них обычно описывают работу схемы. Данный вид сигналов изображён на рисунке 1.9.

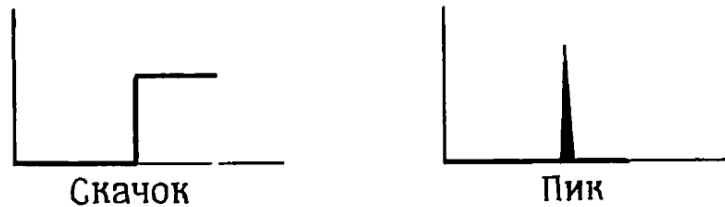


Рис. 1.9 Сигнал в виде скачка и пика.

Скачок представляет из себя отдельную часть прямоугольного сигнала, в то время как пик представляет собой два скачка, разделенных очень коротким промежутком.

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 синтезом 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 интегральные 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 цифровые. 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 включает 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 $p = \frac{T}{\Delta t}$.

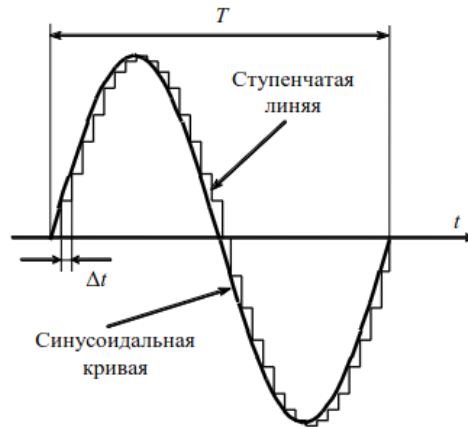


Рис. 1.10 Ступенчатая аппроксимация.

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]. 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 $y(x)$ [6]. Математической моделью в данном методе является единичная окружность с парой векторов, исходящих из центра.

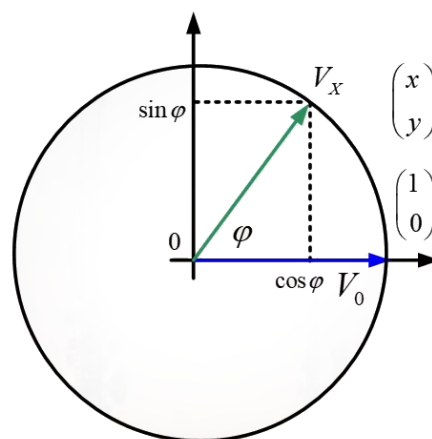


Рис. 1.11 Математическая модель CORDIC.

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:

$$x = x_0 * \cos \phi - y_0 * \sin \phi, \quad (1.3)$$

$$y = x_0 * \sin \phi + y_0 * \cos \phi. \quad (1.4)$$

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

$$x = \cos \phi(x_0 - y_0 * \tan \phi), \quad (1.5)$$

$$y = \cos \phi(y_0 + x_0 * \tan \phi). \quad (1.6)$$

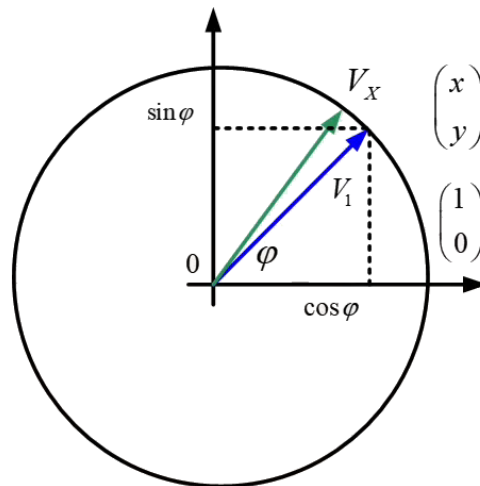


Рис. 1.12 Поворот вектора.

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:

$$\phi_i = \pm atan2^{-i}, \quad (1.7)$$

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).

$$\sigma_i = sign(z_i) \quad (1.8)$$

$$x_{i+1} = x_i - \sigma_i * y_i * 2^{-i} \quad (1.9)$$

$$y_{i+1} = y_i + \sigma_i * x_i * 2^{-i} \quad (1.10)$$

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

Данный алгоритм применим для генерации синуса и его применение целесообразно только при необходимости быстрогодействия 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. 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.

$$f_{out} = \frac{f_{clk}}{n} \quad (1.12)$$

где n — количество отсчётов (длина таблицы).

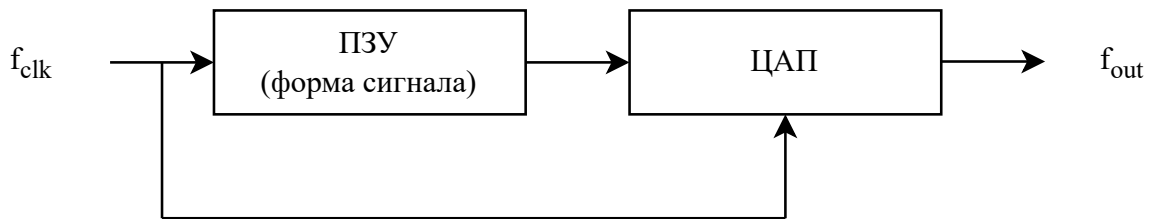


Рис. 1.13 Функциональная схема табличного метода.

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 циф-

ровой последовательности отсчётов и последующего преобразования этих отсчетов из цифрового вида в аналоговый с помощью ЦАП [3].

На рисунке 1.14 изображена структурная схема DDS с аккумулятором фазы.

Частота сигнала в этой архитектуре определяется следующей формулой:

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

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. 1.14 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. 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 1.14.

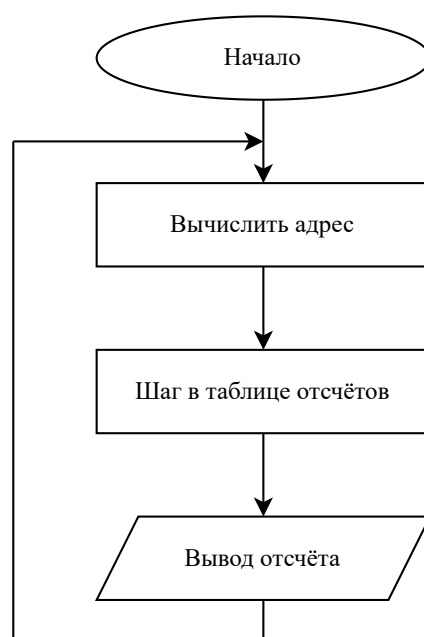


Рис. 1.15 Алгоритм метода DDS.

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 [5].

1.5. Вывод из первой главы

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. В качестве метода генерации сигнала был выбран метод DDS за его простоту реализации и гибкость.

2. ПРОГРАММНАЯ ЧАСТЬ

2.1. Моделирование DDS

Для начала потребуется таблица отсчётов, чтобы её вычислить используем готовый инструмент [7].

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.

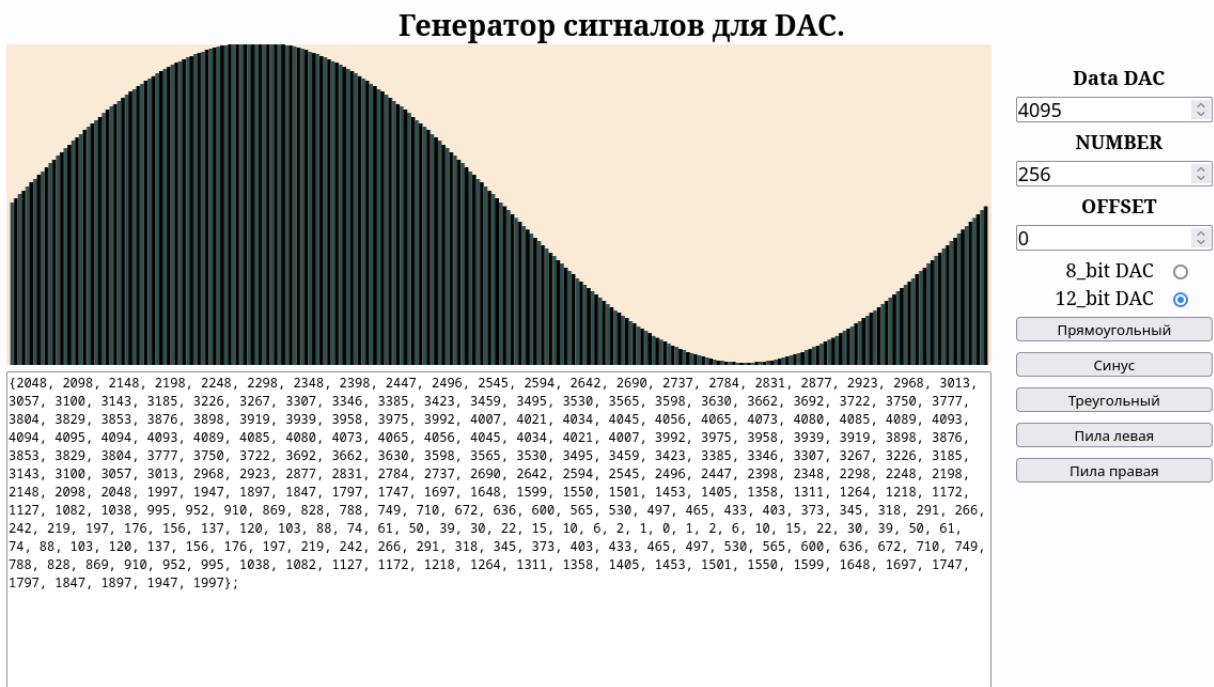


Рис. 2.1 Вычисление таблицы сигнала.

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

```

1 int main() {
2     uint16_t p_acc, p_step;
3     uint8_t addr = 0; // адрес ячейки
4
5     p_acc = 0; // аккумулятор фазы
6     p_step = 128; // код частоты
7
8     while(1) {
9         addr = p_acc >> 8; // выделение старшей части аккумулятора фазы
10        p_acc += p_step; // шаг
11        printf("%d 0x%X\n", addr, sinus[addr]); // вывод отсчёта
12    }
13
14    return 0;
15 }

```

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.



```
kenny@desktop:~/workspace/vkr/dds
> gcc dds.c -o dds && ./dds
0 0x800
0 0x800
1 0x82C
1 0x82C
2 0x858
2 0x858
```

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 [8]. 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 [8].

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 [9]. 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).

Таблица 2.4

Параметры микроконтроллеров

Параметр	ATtiny1	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. Среда 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 [10]. 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

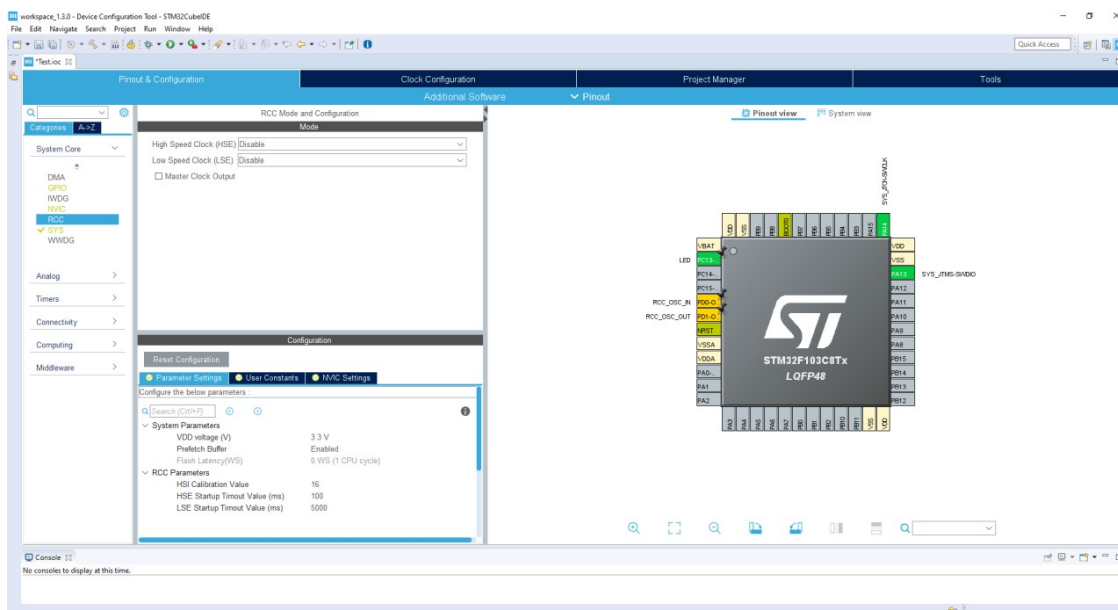


Рис. 2.5 Интерфейс STM32CubeIDE.

функции для инициализации и настройки периферийного устройства, управления передачей данных, обработки прерываний или DMA [11].

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 [12].

Данная среда разработки является расширением для текстового редактора Visual Studio Code. VS Code — это легкий, но мощный редактор кода, имеющий богатую экосистему расширений [13]. Доступен для Windows,

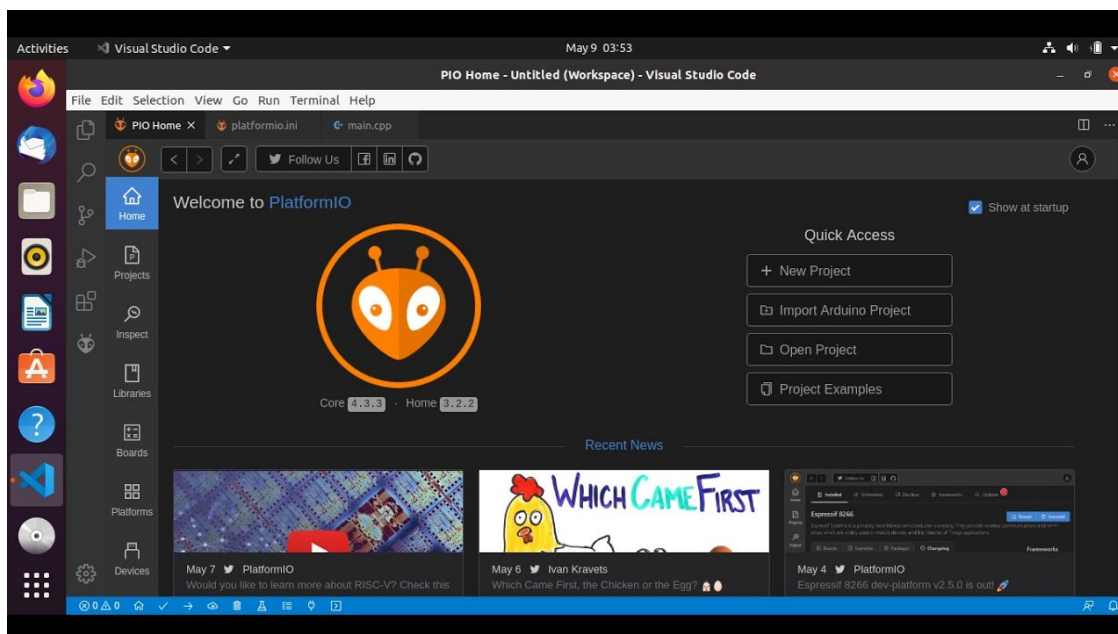


Рис. 2.6 Интерфейс PlatformIO.

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. 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 libopenstm3. Проект libopenstm3 (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 [14].

Достоинства:

- Поддержка различных ОС.
- Быстрая компиляция.
- Поддержка GitHub.
- Возможность работать с разными фреймворками 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`.

2.5. Алгоритм работы

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.

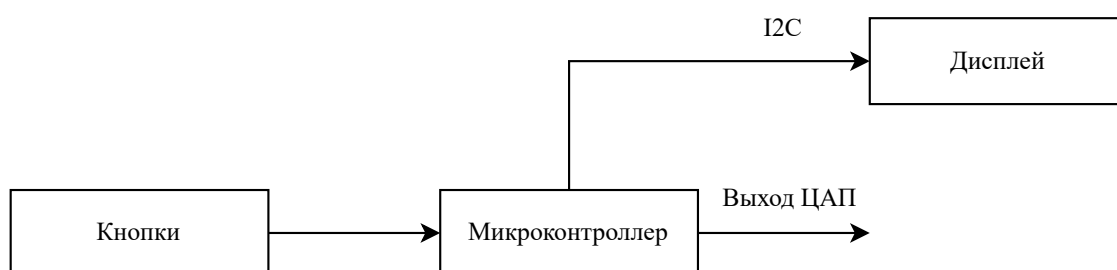


Рис. 2.7 Структурная схема генератора сигналов.

Программа должна выполнять три действия:

1. Вывод отсчёта в ЦАП.
2. Обработка кнопок.
3. Вывод информации на дисплей.

Для цифро-аналогового преобразователя и кнопок выделим два таймера общего назначения, а работа с дисплеем будет идти в главном цикле программы. Применяв такой подход, удастся добиться синхронного выполнения программы.

Таким образом, для подпрограммы генерации сигнала будет следующая блок-схема.

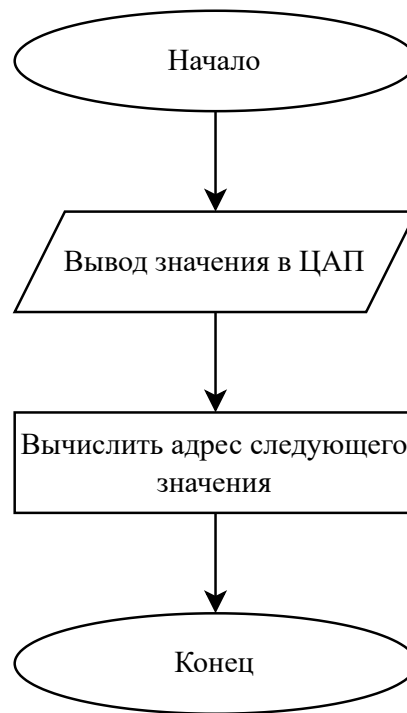


Рис. 2.8 Блок-схема функции генерации сигнала.

По созданной блок-схеме напишем код, который будет выполняться по прерыванию таймера номер 2.

Листинг 2.2

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 }
  
```

Обработка кнопок представлена следующей блок-схемой.

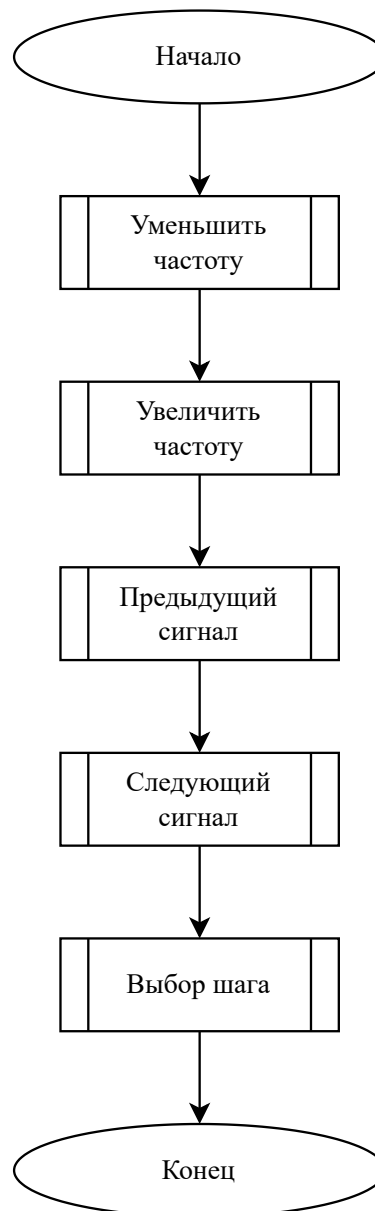


Рис. 2.9 Блок-схема функции кнопок.

Подпрограмма обработки кнопок находится в обработчике прерывания таймера номер 3. Таймер настроен на период 250 миллисекунд. Благодаря такой организации, решается проблема дребезга кнопок. Не приходится делать программную или аппаратную задержку для ожидания установки состояния 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  }

```

Главный блок программы будет содержать в себе основные настройки периферии и вывод информации на дисплей.



Рис. 2.10 Блок-схема главной функции.

Листинг 2.4

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

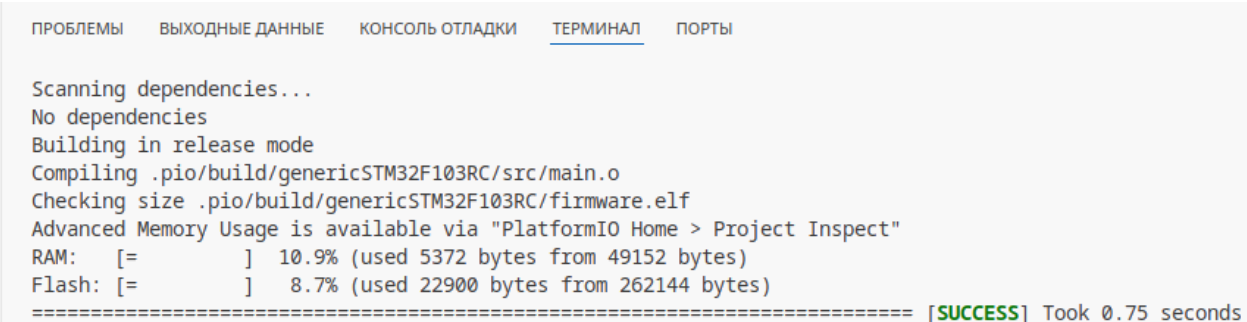
```

```

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          ...
23      }
24      ssd1306_drawWCharStr(0, 16, white, nowrap, L"Частота(Гц)");
25      ssd1306_drawWCharStr(64, 16, white, nowrap, freq);
26      ssd1306_drawWCharStr(0, 32, white, nowrap, L"Шар(Гц)");
27      switch (num_step)
28      {
29          ...
30      }
31      ssd1306_refresh();
32  }
33
34  return 0;
35  }

```

Полный код программы содержится в приложении. После написания программы произведём сборку проекта и получим сообщение об успешной компиляции (рис. 2.12).



```

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. 2.11 Компиляция проекта.

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

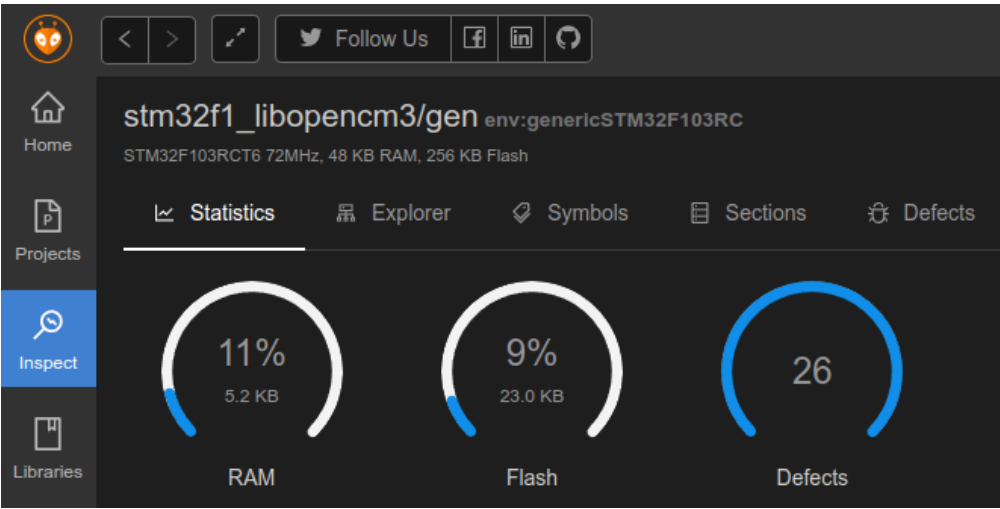


Рис. 2.12 Проверка проекта.

Среда разработки описала занимаемую память и нашла дефекты в проекте.

Defects Summary				Top Defects	
Component	High	Medium	Low	Level	Message
src	0	0	26	LOW	The comparison 'prev_val == 0' is always true.
				LOW	The comparison 'prev_val == 0' is always true.
				LOW	The comparison 'prev_val == 0' is always true.
Total	0	0	26		

Рис. 2.13 Дефекты проекта.

Дефекты оказались незначительные и на работу программы влияния не оказывают.

2.6. Вывод из второй главы

Таким образом, для реализации генератора сигналов был выбран микроконтроллер STM32F103RCT6 с использованием среды разработки PlatformIO и библиотеки libopenm3. 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. АППАРАТНАЯ ЧАСТЬ

3.1. Разработка схемы

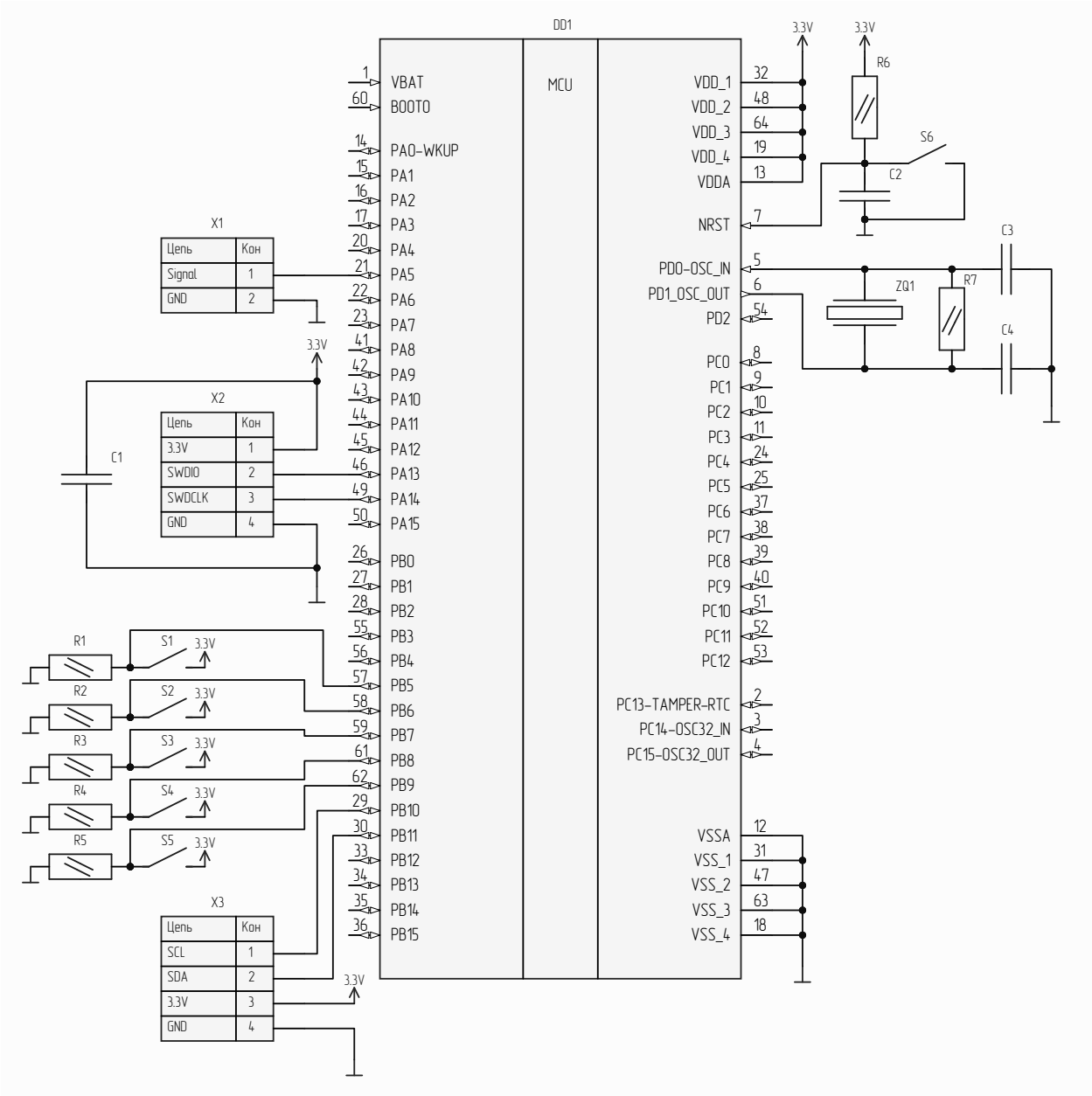


Рис. 3.1 Схема электрическая принципиальная.

3.2. Вывод из третьей главы

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. Спроектирован генератор.
5. Сконструирован макет.
6. Разработана и протестирована программа.

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. Брони́на [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. 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.
7. Генератор сигналов для DAC. [Электронный ресурс] PROGCONT.RU. Режим доступа: https://progcont.ru/?articles=54&category_articles=ALL.
8. 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). – ISBN 5-94120-081-1.
9. 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. – ISBN 978-5-9948-2007-0.

10. STM32CubeIDE [Электронный ресурс] Официальный сайт STMicroelectronics. Режим доступа: <https://www.st.com/en/development-tools/stm32cubeide.html>.
11. 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.
12. PlatformIO [Электронный ресурс] Официальный сайт PlatformIO. Режим доступа: <https://platformio.org/>.
13. Docs PlatformIO [Электронный ресурс] Официальный сайт PlatformIO. Режим доступа: <https://docs.platformio.org/en/latest/integration/ide/pioide.html#platformio-for-vscode>.
14. LibOpenCM3 [Электронный ресурс] Официальный сайт LibOpenCM3. Режим доступа: <https://libopencm3.org/>.

Программа генератора сигналов.

```

1  #include <stdio.h>
2  #include <wchar.h>
3  #include <libopencm3/stm32/rcc.h>
4  #include <libopencm3/stm32/flash.h>
5  #include <libopencm3/stm32/gpio.h>
6  #include <libopencm3/stm32/timer.h>
7  #include <libopencm3/cm3/nvic.h>
8  #include <libopencm3/stm32/dac.h>
9  #include <libopencm3/stm32/i2c.h>
10 #include <ssd1306_i2c.h>
11
12 static void gpio_setup(void); // установить входы/выходы
13 static void dac_setup(void); // настройка цап
14 static void i2c_setup(void); // настройка и2ц
15 static void timers_setup(void); // настройка таймеров
16 static void nvic_setup(void); // настройка прерываний
17 void minus_freq(void); //
18 void plus_freq(void); //
19 void minus_signal(void); // функции для кнопок
20 void plus_signal(void); //
21 void step_select(void); //
22
23 uint16_t p_acc = 0; // аккумулятор фазы
24 int p_step = 0; // код частоты 192 - 1khz
25 uint16_t step = 0; // размер шага
26 uint16_t signal[256] = {0}; // буфер для цапа
27 int8_t num_sig = 0; // номер сигнала
28 int8_t num_step = 0; // номер шага
29
30 /* отсчеты сигналов */
31 uint16_t sinus[256] = {2048, 2092, 2136, 2180, 2224, 2268, 2312, 2355, 2399, 2442,
32 2485, 2527, 2570, 2612, 2654, 2695, 2736, 2777, 2817, 2857, 2896, 2934, 2973, 3010, 3047,
33 3084, 3119, 3155, 3189, 3223, 3256, 3288, 3320, 3351, 3381, 3410, 3439, 3466, 3493, 3519,
34 3544, 3568, 3591, 3613, 3635, 3655, 3674, 3693, 3710, 3726, 3742, 3756, 3770, 3782, 3793,
35 3803, 3812, 3821, 3828, 3833, 3838, 3842, 3845, 3846, 3847, 3846, 3845, 3842, 3838, 3833,
36 3828, 3821, 3812, 3803, 3793, 3782, 3770, 3756, 3742, 3726, 3710, 3693, 3674, 3655, 3635,
37 3613, 3591, 3568, 3544, 3519, 3493, 3466, 3439, 3410, 3381, 3351, 3320, 3288, 3256, 3223,
38 3189, 3155, 3119, 3084, 3047, 3010, 2973, 2934, 2896, 2857, 2817, 2777, 2736, 2695, 2654,
39 2612, 2570, 2527, 2485, 2442, 2399, 2355, 2312, 2268, 2224, 2180, 2136, 2092, 2048, 2003,
40 1959, 1915, 1871, 1827, 1783, 1740, 1696, 1653, 1610, 1568, 1525, 1483, 1441, 1400, 1359,
41 1318, 1278, 1238, 1199, 1161, 1122, 1085, 1048, 1011, 976, 940, 906, 872, 839, 807, 775,
42 744, 714, 685, 656, 629, 602, 576, 551, 527, 504, 482, 460, 440, 421, 402, 385, 369, 353,
43 339, 325, 313, 302, 292, 283, 274, 267, 262, 257, 253, 250, 249, 248, 249, 250, 253, 257,
44 262, 267, 274, 283, 292, 302, 313, 325, 339, 353, 369, 385, 402, 421, 440, 460, 482, 504,
45 527, 551, 576, 602, 629, 656, 685, 714, 744, 775, 807, 839, 872, 906, 940, 976, 1011, 1048,
46 1085, 1122, 1161, 1199, 1238, 1278, 1318, 1359, 1400, 1441, 1483, 1525, 1568, 1610, 1653,
47 1696, 1740, 1783, 1827, 1871, 1915, 1959, 2003};
48 /* */

```



```

49 uint16_t square[256] = {3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
50     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
51     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
52     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
53     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
54     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
55     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
56     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
57     3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847, 3847,
58     3847, 3847, 3847, 3847, 3847, 3847, 248, 248, 248, 248, 248, 248, 248, 248, 248,
59     248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
60     248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
61     248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
62     248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
63     248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
64     248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248,
65     248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248, 248};
66 /* */
67 uint16_t triangle[256] = {248, 276, 304, 332, 360, 389, 417, 445, 473, 501, 529, 557, 585, 614, 642, 670, 698,
68     726, 754, 782, 810, 838, 867, 895, 923, 951, 979, 1007, 1035, 1063, 1092, 1120, 1148, 1176, 1204, 1232, 1260,
69     1288, 1316, 1345, 1373, 1401, 1429, 1457, 1485, 1513, 1541, 1570, 1598, 1626, 1654, 1682, 1710, 1738, 1766, 1794,
70     1823, 1851, 1879, 1907, 1935, 1963, 1991, 2019, 2048, 2076, 2104, 2132, 2160, 2188, 2216, 2244, 2272, 2301, 2329,
71     2357, 2385, 2413, 2441, 2469, 2497, 2525, 2554, 2582, 2610, 2638, 2666, 2694, 2722, 2750, 2779, 2807, 2835, 2863,
72     2891, 2919, 2947, 2975, 3003, 3032, 3060, 3088, 3116, 3144, 3172, 3200, 3228, 3257, 3285, 3313, 3341, 3369, 3397,
73     3425, 3453, 3481, 3510, 3538, 3566, 3594, 3622, 3650, 3678, 3706, 3735, 3763, 3791, 3819, 3847, 3819, 3791, 3763,
74     3735, 3706, 3678, 3650, 3622, 3594, 3566, 3538, 3510, 3481, 3453, 3425, 3397, 3369, 3341, 3313, 3285, 3257, 3228,
75     3200, 3172, 3144, 3116, 3088, 3060, 3032, 3003, 2975, 2947, 2919, 2891, 2863, 2835, 2807, 2779, 2750, 2722, 2694,
76     2666, 2638, 2610, 2582, 2554, 2525, 2497, 2469, 2441, 2413, 2385, 2357, 2329, 2301, 2272, 2244, 2216, 2188, 2160,
77     2132, 2104, 2076, 2048, 2019, 1991, 1963, 1935, 1907, 1879, 1851, 1823, 1794, 1766, 1738, 1710, 1682, 1654, 1626,
78     1598, 1570, 1541, 1513, 1485, 1457, 1429, 1401, 1373, 1345, 1316, 1288, 1260, 1232, 1204, 1176, 1148, 1120, 1092,
79     1063, 1035, 1007, 979, 951, 923, 895, 867, 838, 810, 782, 754, 726, 698, 670, 642, 614, 585, 557, 529, 501, 473,
80     445, 417, 389, 360, 332, 304, 276};
81 /* */
82 uint16_t l_saw[256] = {248, 262, 276, 290, 304, 319, 333, 347, 361, 375, 389, 403, 417, 431, 446, 460, 474,
83     488, 502, 516, 530, 544, 559, 573, 587, 601, 615, 629, 643, 657, 671, 686, 700, 714, 728, 742, 756, 770, 784,
84     798, 813, 827, 841, 855, 869, 883, 897, 911, 925, 940, 954, 968, 982, 996, 1010, 1024, 1038, 1052, 1067, 1081,
85     1095, 1109, 1123, 1137, 1151, 1165, 1180, 1194, 1208, 1222, 1236, 1250, 1264, 1278, 1292, 1307, 1321, 1335, 1349,
86     1363, 1377, 1391, 1405, 1419, 1434, 1448, 1462, 1476, 1490, 1504, 1518, 1532, 1546, 1561, 1575, 1589, 1603, 1617,
87     1631, 1645, 1659, 1673, 1688, 1702, 1716, 1730, 1744, 1758, 1772, 1786, 1801, 1815, 1829, 1843, 1857, 1871, 1885,
88     1899, 1913, 1928, 1942, 1956, 1970, 1984, 1998, 2012, 2026, 2040, 2055, 2069, 2083, 2097, 2111, 2125, 2139, 2153,
89     2167, 2182, 2196, 2210, 2224, 2238, 2252, 2266, 2280, 2294, 2309, 2323, 2337, 2351, 2365, 2379, 2393, 2407, 2422,
90     2436, 2450, 2464, 2478, 2492, 2506, 2520, 2534, 2549, 2563, 2577, 2591, 2605, 2619, 2633, 2647, 2661, 2676, 2690,
91     2704, 2718, 2732, 2746, 2760, 2774, 2788, 2803, 2817, 2831, 2845, 2859, 2873, 2887, 2901, 2915, 2930, 2944, 2958,
92     2972, 2986, 3000, 3014, 3028, 3043, 3057, 3071, 3085, 3099, 3113, 3127, 3141, 3155, 3170, 3184, 3198, 3212, 3226,
93     3240, 3254, 3268, 3282, 3297, 3311, 3325, 3339, 3353, 3367, 3381, 3395, 3409, 3424, 3438, 3452, 3466, 3480, 3494,
94     3508, 3522, 3536, 3551, 3565, 3579, 3593, 3607, 3621, 3635, 3649, 3664, 3678, 3692, 3706, 3720, 3734, 3748, 3762,
95     3776, 3791, 3805, 3819, 3833, 3847};
96 /* */
97 uint16_t r_saw[256] = {3847, 3833, 3819, 3805, 3791, 3776, 3762, 3748, 3734, 3720, 3706, 3692, 3678, 3664,
98     3649, 3635, 3621, 3607, 3593, 3579, 3565, 3551, 3536, 3522, 3508, 3494, 3480, 3466, 3452, 3438, 3424, 3409,
99     3395, 3381, 3367, 3353, 3339, 3325, 3311, 3297, 3282, 3268, 3254, 3240, 3226, 3212, 3198, 3184, 3170, 3155,
100    3141, 3127, 3113, 3099, 3085, 3071, 3057, 3043, 3028, 3014, 3000, 2986, 2972, 2958, 2944, 2930, 2915, 2901,
101    2887, 2873, 2859, 2845, 2831, 2817, 2803, 2788, 2774, 2760, 2746, 2732, 2718, 2704, 2690, 2676, 2661, 2647,
102    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"Шар(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 return 0;
187 }
188
189 static void gpio_setup(void)
190 {
191     // rcc_periph_clock_enable(RCC_GPIOB); // тактирование портов
192     rcc_periph_clock_enable(RCC_GPIOB);
193     gpio_set_mode(GPIOB, GPIO_MODE_INPUT, GPIO_CNF_INPUT_PULL_UPDOWN, GPIO9 | GPIO5 | GPIO6 | GPIO7 | GPIO8); //
194     ↔ входы для кнопок, подняты к земле
195     // gpio_set_mode(GPIOB, 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 {
208     rcc_periph_clock_enable(RCC_I2C2);
209     /* Set alternate functions for the SCL and SDA pins of I2C2. */
210     gpio_set_mode(GPIOB, GPIO_MODE_OUTPUT_50_MHZ,

```

```

210         GPIO_CNF_OUTPUT_ALTFN_OPENDRAIN,
211         GPIO_I2C2_SCL | GPIO_I2C2_SDA);
212
213     /* Disable the I2C before changing any configuration. */
214     i2c_peripheral_disable(I2C2);
215
216     /* APB1 is running at 36MHz. */
217     i2c_set_clock_frequency(I2C2, I2C_CR2_FREQ_36MHZ);
218
219     /* 400KHz - I2C Fast Mode */
220     i2c_set_fast_mode(I2C2);
221
222     /*
223      * fclock for I2C is 36MHz APB2 -> cycle time 28ns, low time at 400kHz
224      * incl trise -> Thigh = 1600ns; CCR = tlow/tcycle = 0x1C,9;
225      * Datasheet suggests 0x1e.
226      */
227     i2c_set_ccr(I2C2, 0x1e);
228
229     /*
230      * fclock for I2C is 36MHz -> cycle time 28ns, rise time for
231      * 400kHz => 300ns and 100kHz => 1000ns; 300ns/28ns = 10;
232      * Incremented by 1 -> 11.
233      */
234     i2c_set_trise(I2C2, 0x0b);
235
236     /*
237      * Enable ACK on I2C
238      */
239     i2c_enable_ack(I2C2);
240
241     /*
242      * This is our slave address - needed only if we want to receive from
243      * other masters.
244      */
245     i2c_set_own_7bit_slave_address(I2C2, 0x32);
246
247     /* If everything is configured -> enable the peripheral. */
248     i2c_peripheral_enable(I2C2);
249 }
250
251 static void timers_setup(void)
252 {
253     rcc_periph_clock_enable(RCC_TIM2);
254     rcc_periph_clock_enable(RCC_TIM3);
255
256     /* Стартовое значение таймера */
257     TIM_CNT(TIM2) = 1;
258     TIM_CNT(TIM3) = 1;
259
260     /* Предделитель 36MHz/36000 => 1000 отсчетов в секунду */
261     TIM_PSC(TIM2) = 18;
262     TIM_PSC(TIM3) = 36000;
263

```

```

264  /* Период таймера */
265  TIM_ARR(TIM2) = 10;
266  TIM_ARR(TIM3) = 250;
267
268  /* Включить прерывания */
269  TIM_DIER(TIM2) |= TIM_DIER_UIE;
270  TIM_DIER(TIM3) |= TIM_DIER_UIE;
271
272  /* Запустить таймер */
273  TIM_CR1(TIM2) |= TIM_CR1_CEN;
274  TIM_CR1(TIM3) |= TIM_CR1_CEN;
275  }
276
277  static void nvic_setup(void)
278  {
279      /* Активировать прерывания и установить приоритеты */
280      nvic_enable_irq(NVIC_TIM2_IRQ);
281      nvic_set_priority(NVIC_TIM2_IRQ, 2);
282      nvic_enable_irq(NVIC_TIM3_IRQ);
283      nvic_set_priority(NVIC_TIM3_IRQ, 1);
284  }
285
286  void minus_freq(void)
287  {
288      bool cur_val = 0;
289      bool prev_val = 0;
290      cur_val = gpio_get(GPIOB, GPIO5);
291      if (cur_val == 1 && prev_val == 0)
292      {
293          p_step -= step;
294      }
295      if (p_step < 0) // ограничение 0
296      {
297          p_step = 0;
298      }
299      prev_val = cur_val;
300  }
301
302  void plus_freq(void)
303  {
304      bool cur_val = 0;
305      bool prev_val = 0;
306      cur_val = gpio_get(GPIOB, GPIO6);
307      if (cur_val == 1 && prev_val == 0)
308      {
309          p_step += step;
310      }
311      if (p_step > 9600) // ограничение 50 кГц
312      {
313          p_step = 9600;
314      }
315      prev_val = cur_val;
316  }
317

```

```

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

```

```

372     case 2:
373         for (int i = 0; i < 256; i++)
374             signal[i] = square[i];
375         break;
376     case 3:
377         for (int i = 0; i < 256; i++)
378             signal[i] = triangle[i];
379         break;
380     case 4:
381         for (int i = 0; i < 256; i++)
382             signal[i] = l_saw[i];
383         break;
384     case 5:
385         for (int i = 0; i < 256; i++)
386             signal[i] = r_saw[i];
387         break;
388     }
389     if (num_sig > 5)
390         num_sig = 5;
391     dac_enable(CHANNEL_2);
392 }
393 prev_val = cur_val;
394 }
395
396 void step_select(void)
397 {
398     bool cur_val = 0;
399     bool prev_val = 0;
400     cur_val = gpio_get(GPIOB, GPIO9);
401     if (cur_val == 1 && prev_val == 0)
402     {
403         num_step += 1;
404         switch (num_step)
405         {
406             case 1:
407                 step = 24; // 125 Гц
408                 break;
409             case 2:
410                 step = 48; // 250 Гц
411                 break;
412             case 3:
413                 step = 96; // 500 Гц
414                 break;
415             case 4:
416                 step = 192; // 1000 Гц
417                 break;
418             case 5:
419                 step = 24; // 125 Гц
420                 num_step = 1;
421                 break;
422         }
423     }
424 }

```

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