Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.Г.ШУХОВА

(БГТУ им. В.Г. Шухова)

Кафедра электроэнергетики и автоматики

**КУРСОВОЙ ПРОЕКТ**

По дисциплине “ Микроконтроллеры в электроприводе”

**«Проектирование системы управления регулируемым электроприводом с помощью частотного преобразователя Delta VFD-B»**

Вариант 9

Выполнил:

студент 4 курса

гр. ЭА-191 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.П. Танчук

Проверил:

доцент, кандидат техн. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Солдатенков

Белгород 2023

СОДЕРЖАНИЕ

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 5](#_Toc130658492)

[ВВЕДЕНИЕ 7](#_Toc130658493)

[1 Исходные данные для проектирования 9](#_Toc130658494)

[2 Управление ПЧ с помощью компьютера на Windows 14](#_Toc130658495)

[2.1 Разработка драйвера последовательного COM порта 19](#_Toc130658496)

[2.2 Разработка драйвера ModbusRTU 31](#_Toc130658497)

[2.3 Разработка драйвера ПЧ 40](#_Toc130658498)

[2.4 Основная программа управления 47](#_Toc130658499)

[2.5 Тестирование работы программы 57](#_Toc130658500)

[3 Управление ПЧ с помощью промышленного контроллера ICP DAS I-7188EX 62](#_Toc130658501)

[3.1 Функции времени на I-7188EX 62](#_Toc130658502)

[3.2 Разработка драйвера UART интерфейса 62](#_Toc130658503)

[3.3 Перенос основной программы 62](#_Toc130658504)

[3.4 Разработка интерфейса взаимодействия с оператором 62](#_Toc130658505)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 63](#_Toc130658506)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

АДКЗ – асинхронный двигатель с короткозамкнутым ротором;

ПЧ – преобразователь частоты;

MSVS – Microsoft Visual Studio 2022;

RS-485 – TIA/EIA-485-A-ENGL 1998 Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems;

DCB – Device Control Block;

RTU – Remote terminal unit;

CRC – сyclic redundancy checking;

ADU – application data unit;

PDU – protocol data unit;

ВВЕДЕНИЕ

Асинхронный двигатель с короткозамкнутым ротором (АДКЗ) являясь самым распространенным типом электрической машины с развитием математической теории машин переменного тока и усовершенствованием IGBT транзисторов стал управляемым приводом. При частотном регулировании скорости вращения АДКЗ составляет конкуренцию электроприводу постоянного тока по диапазону регулирования, плавности и экономичности. А благодаря простоте конструкции и бесконтактности данного типа двигателя снижается его стоимость и появляется возможность использования во взрывоопасных средах.

Преобразователи частоты предназначены для плавной регулировки скорости и момента вращения вала двигателя путем изменения частоты и напряжения переменного тока при изменении какого-либо технологического параметра.

Использование современных средств управления, включая микропроцессорные, позволяет совершенствовать алгоритмы управления системой «преобразователь-двигатель» и создавать надежные системы регулируемых электроприводов. Высокие динамические и эксплуатационные характеристики электроприводов обеспечиваются за счет разработки и использования новых алгоритмов управления с реализацией на современной микроэлектронной базе – сигнальных процессорах (DSP). Точность цифровой регулировки и наличие промышленных сетевых интерфейсов позволяет легко интегрировать ПЧ в систему автоматизации. Модернизированный таким образом частотно-регулируемый электропривод становится звеном архитектуры АСУ ТП производства.

Целью курсовой работы является практическое изучение современных принципов разработки систем управления регулируемым электроприводом в составе АСУ ТП производства с помощью преобразователей частоты и микропроцессорных систем. В курсовой работе должны быть выполнены следующие задачи:

- реализация алгоритма управления для Windows;

- реализация алгоритма управления на промышленном контроллере I7188EX;

- реализация пользовательского интерфейса.

# Исходные данные для проектирования

Задан алгоритм работы двигателя в виде тахограммы (рисунок 1.1). Необходимо разработать программу для реализации заданного алгоритма с поддержкой следующих функций:

- задание параметров связи с ПЧ;

- задание тахограмм вращения;

- обработка ошибок связи;

- измерение параметров двигателя (частота, ток, напряжение, скорость вращения) во время его работы.

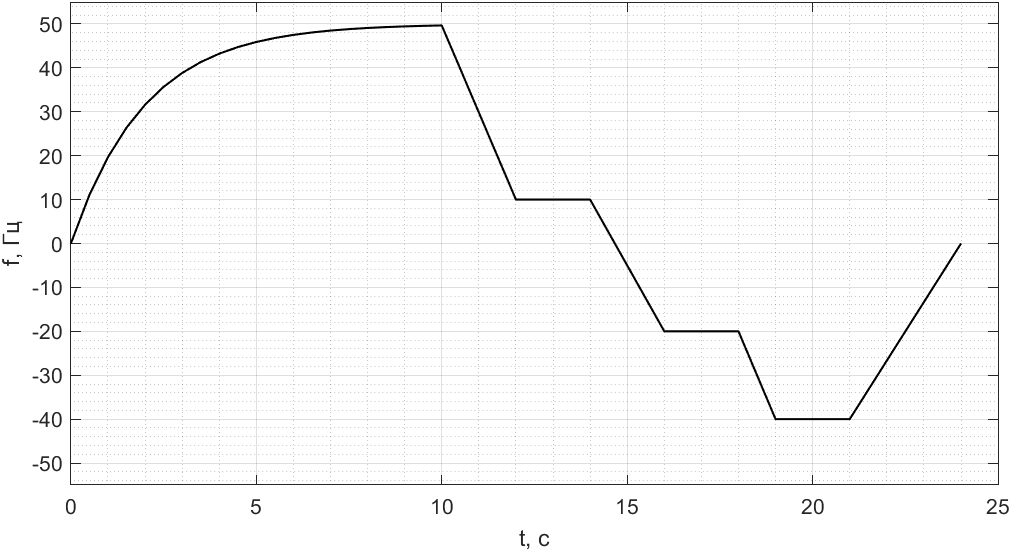


Рисунок 1.1 – Требуемая тахограмма вращения двигателя

Система управления состоит из асинхронного электродвигателя с короткозамкнутым ротором, преобразователя частоты, промышленного контроллера, преобразователей интерфейсов, персонального компьютера. Приведем краткое описание и технические характеристики каждого из элементов системы.

**Электродвигатель A0-90L-6**

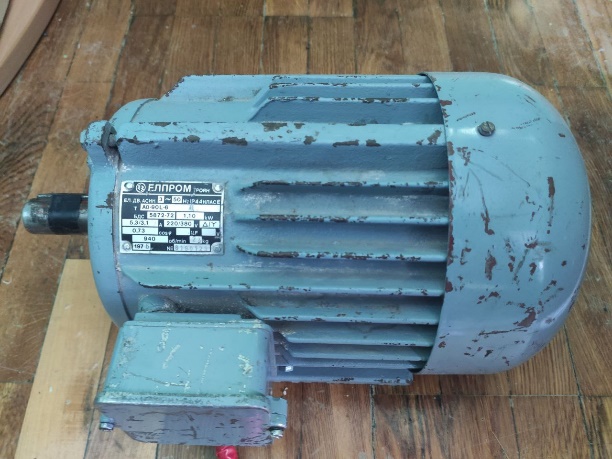


Рисунок 1.2 – Внешний вид двигателя A0-90L-6

Тип: АДКЗ.

Название: A0-90L-6.

Производитель: ЕЛПРОМ троян.

Питание 3Ф~50Гц.

Мощность: 1,1 кВт.

Соединение: Δ/Y.

Напряжение: 220/380 В.

Ток: 5,3/3,1 А.

Коэффициент мощности: 0,73.

Частота вращения: 940 об/мин.

Степень защиты: IP44.

Класс изоляции: E.

**Преобразователь частоты Delta Electronics VFD-015B21A**

Преобразователи частоты (ПЧ) компании Delta Electronics, Inc. серии VFD-В предназначены для управления скоростью вращения трехфазных асинхронных электродвигателей с короткозамкнутым ротором мощностью от 0,4 до 75 кВт в составе такого оборудования как, насосы, вентиляторы, миксеры, экструдеры, транспортирующие и подъемные механизмы и другого.

Мощность: 1,5 кВт.

Питание: 1Ф~220 В 50 Гц.

Выходная частота: 0,1–400 Гц.

Потребляемый ток: 9,4 А.

Функциональные возможности: настраиваемая характеристика V/f и векторное управление, основной и дополнительный источники задания частоты, 15 предустановленных скоростей, автоматическое пошаговое управление, ПИД-регулятор, автоматическая компенсация момента и скольжения, связь по MODBUS через последовательный интерфейс RS-485, векторное управление в разомкнутом и замкнутом контуре, авто тестирование двигателя, управления группой электродвигателей, съемный пульт управления, защиты от многих аварийных ситуаций.



Рисунок 1.3 – Внешний вид ПЧ

**Промышленный контроллер I-7188EX**

Производитель: ICP DAS.

Модель I-7188EX.

Крепление: монтаж на DIN рейку.

Предустановленная операционная система: MiniOS7.

Питание: напряжений 10 В - 30 В и потребление 2 Вт.

Рабочая температура: -25 ~ 75 °C.

Скорость передачи данных: 300 ~ 115200 кБит/сек.

Процессор: AMD 80188-40 или совместимый с тактовой частотой 40 МГц.

Память: 512 кБ (SRAM) + 512 кБ (Flash) + 31Б (NVRAM) с возможностью расширения до 64 Мб путем установки дополнительных SRAM или Flash плат расширения.

Интерфейсы, разъемы и выходы: винтовые клеммы (RS-485, RS-232), RJ-45 (LAN), 10BASE-T, совместимый с NE2000.

Протоколы: TCP/IP (TCP, UDP, IP, ICMP, ARP), Reverse Address Resolution Protocol.



Рисунок 1.4 – Внешний вид контроллера I-7188EX

**Преобразователи интерфейсов**

Конвертер Espada USB/RS-485 на базе микросхемы CH340G и MAX485ESA является преобразователем интерфейсов, предназначенным для перехода с интерфейса USB на порт RS-485. Задача конвертера, подключаемого к USB-порту компьютера или ноутбука, состоит во взаимообратном преобразовании сигналов интерфейсов USB и RS-485. Изделие работает в полудуплексном режиме и поддерживает автоматическое переключение с приема на передачу и обеспечивает подключение контроллеров, измерительных устройств, систем сбора данных к компьютеру с использованием порта USB 2.0.



Рисунок 1.5 – Внешний вид преобразователя интерфейсов

**Персональный компьютер**

Любой с ОС Windows 64bit.

# Управление ПЧ с помощью компьютера на Windows

Схема управления ПЧ (рисунок 2.1) работает следующим образом. При включении автоматического выключателя напряжение однофазной сети подается на ПЧ. ПЧ подключен к компьютеру с помощью интерфейса RS-485 через конвертер RS-485/USB. К выходным клеммам ПЧ подсоединен трехфазный АДКЗ. С помощью протокола Modbus RTU компьютер устанавливает параметры ПЧ (частота вращения, время разгона, направление движения), отправляет команды на запуск и остановку двигателя, считывает параметры тока и напряжения во время работы.

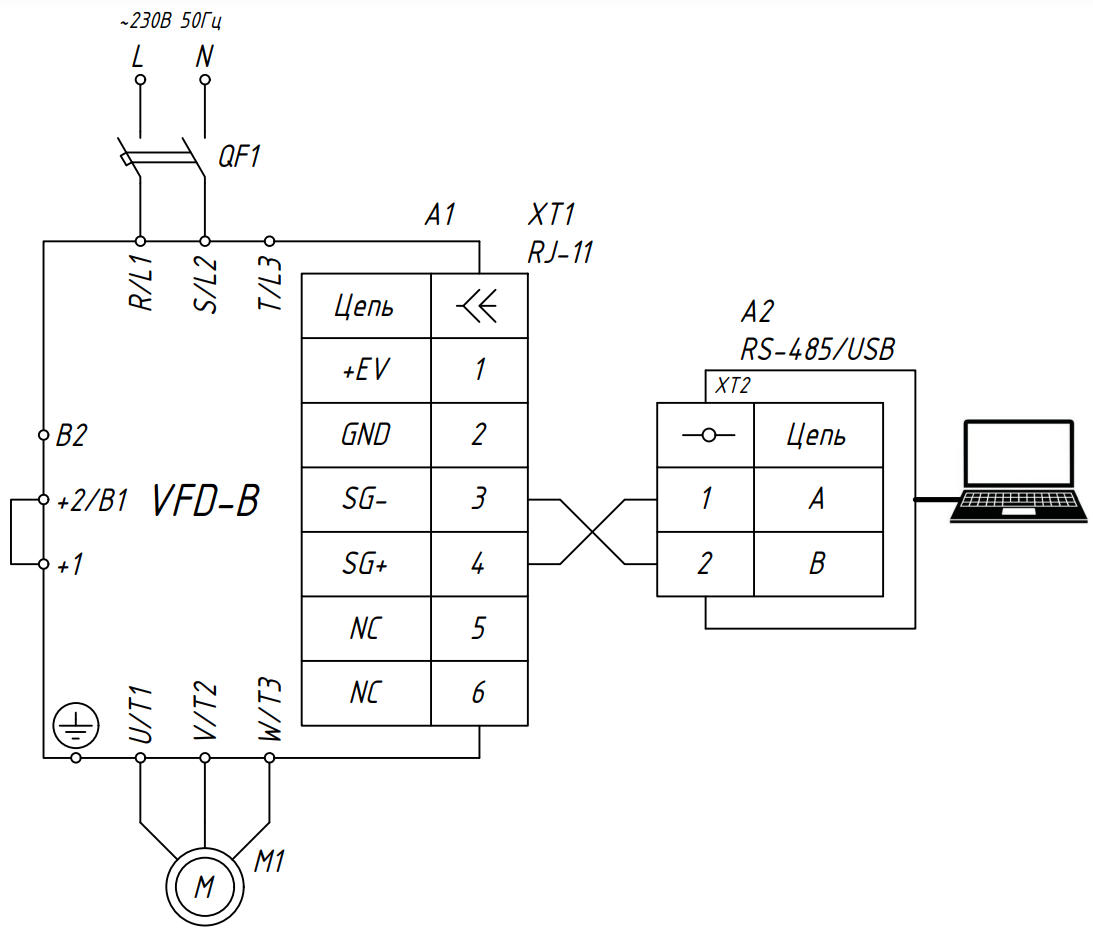


Рисунок 2.1 – Принципиальная схема системы управления двигателем с помощью компьютера

Перед созданием программного проекта нужно продумать его архитектуру. Приложение для управления ПЧ с компьютера (рисунок 2.2) работает следующим образом. С помощью графического интерфейса пользователь задает параметры тахограммы вращения двигателя и отправляет команды запуска или остановки. После чего выбранные пользователем параметры и команды записываются в файл конфигурации и запускается программа управления с помощью командной строки. Эта программа считывает требуемые параметры с файла конфигурации, соответственно им задает режим работы двигателя посредством вызова высокоуровневых методов класса VFD. В классе VFD формируются адреса и значения регистров для доступа к требуемым командам и параметрам ПЧ и отправляются в класс ModbusRTU. В классе ModbusRTU формируется строка запроса Modbus и передается на нижний уровень классу COMPort, который с помощью WinAPI отправляет строку запроса в порт.

Полученный от ПЧ ответ расшифровывается в обратном порядке и также записывается в конфигурационный файл, данные из которого считываются программой графического интерфейса и предоставляются пользователю в виде графиков.

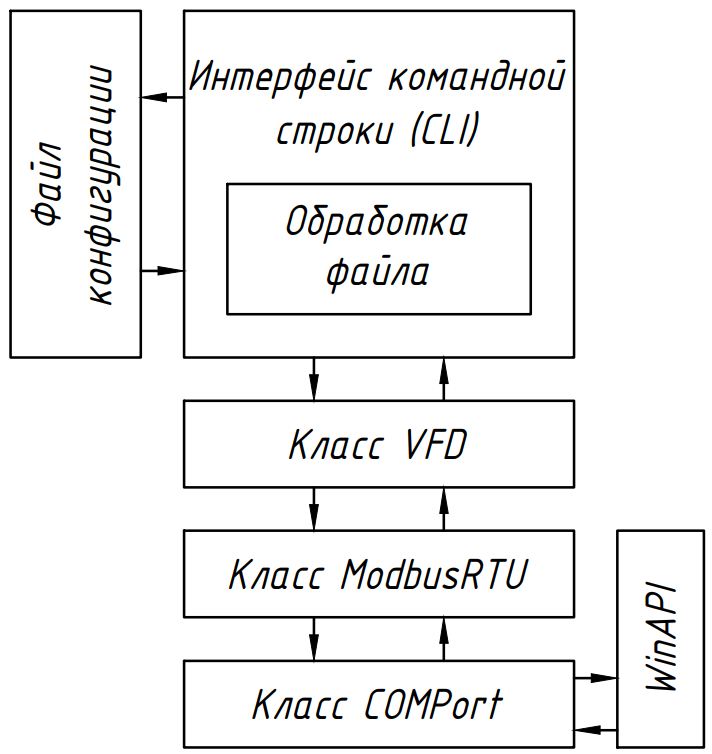


Рисунок 2.2 – Архитектура приложения

Для разработки основной программы управления воспользуемся языком **C/C++** и средой разработки Microsoft Visual Studio 2022 (MSVS). Для этого создадим новый проект на основе шаблона *Empty Project*, к которому постепенно будем подключать дополнительные модули.

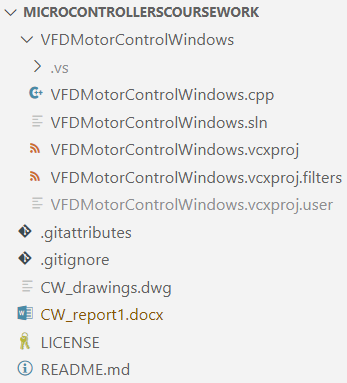
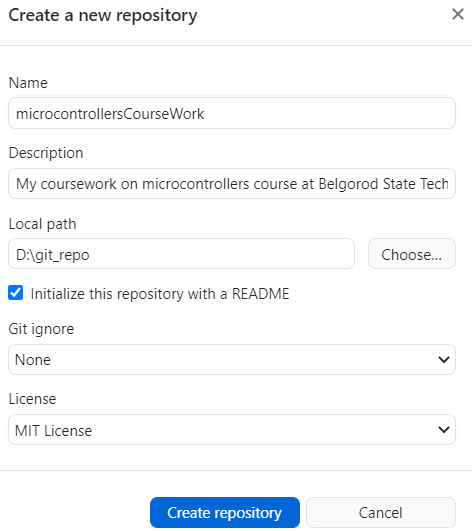
Разработка больших программных проектов требует продуманной организации работы. С этой целью воспользуемся распределенной системой контроля версий Git и Git-хостингом GitHub.

Система контроля версий предназначена для сохранения истории изменений. Как правило, она применяется при разработке программных проектов или набора конфигурационных файлов. История представляет собой снимки проекта, следующие друг за другом в хронологическом порядке. В любой момент можно откатиться к любому состоянию системы в прошлом. Таким образом, можно восстановить поврежденные или случайно удаленные файлы, а также выяснить, кто автор внесенных в код изменений. Также система контроля версий используется при командной работе над проектом [1].

Основная папка проекта Git будет включать в себя как проекты MSVS для Windows и микроконтроллера, так и чертежи, документацию на выбранные элементы системы, описание проекта.

Перед созданием проекта на GitHub зарегистрируемся на сайте <https://github.com/>, установим приложение GitHub Desktop (<https://desktop.github.com/>) и в приложении введем учетные данные от аккаунта GitHub. Git ведет историю изменений всех файлов в выбранной папке. Операционная система Windows и прикладные приложения создают временные файлы при работе, изменения в которых также фиксируются Git. Такое поведение нежелательно, поэтому существует система игнорирования некоторых файлов. В основной папке проекта Git создадим файл *.gitignore*. Для автоматической генерации его содержимого воспользуемся сервисом gitignore.io (<https://www.toptal.com/developers/gitignore/>), на котором зададим ключевые слова: windows, visualstudio, c, c++, microsoftoffice, matlab. В этот же файл добавим шаблоны игнорирования временных файлов AutoCAD (*\*.bak* *\*.dwl* *\*.dwl2*). Благодаря фильтру *.gitignore* в репозиторий попадут только исходные файлы проекта, а получить бинарный *.exe* файл приложения можно из вкладки Release на сайте GitHub или собрать его самостоятельно.

Для создания проекта в программе GitHub Desktop выбираем **File→New Repository…**, вводим название и описание проекта, а также путь к папке, в которой будет располагаться репозиторий. В нашем случае уже имеется папка с файлом *.gitignore* и проектом MSVS, поэтому название репозитория должно совпадать с названием этой папки, а в поле расположения репозитория нужно ввести путь к этой папки без указания названия самой папки (рисунок 1.1, *а*).



*a* *б*

Рисунок 2.3 – Настройки нового Git репозитория (*а*), и его файловая структура (*б*)

Загружаем созданный репозиторий на GitHub командой **Repository→Push**. Репозиторий будет доступен по адресу <https://github.com/TAN4UKmax/microcontrollersCourseWork>). Структура файлов созданного репозитория показана на рисунке 1.1, *б*. Во время работы над проектом фиксация его состояния осуществляется командой **commit**, а отправка на сервер – **push**.

В процессе разработки приложения будем использовать библиотеку **assert** для проверки работоспособности отдельных модулей и поиска возможных ошибок. Ее подключение выглядит следующим образом:

#include <cassert>

Данная библиотека реализована в виде макроопераций и строка для проверки условий имеет вид:

assert(<condition>);

Макрооперация проверяет значение аргумента <condition> и если это значение равно нулю, то выводится диагностическое сообщение, содержащее имя программы, имя файла, номер строки и имя функции в которой была вызвана макрооперация assert. После чего завершается работа программы. В противном случае не производится никаких действий. Также можно добавить строку с дополнительным описанием ошибки.

assert(("description string", <condition>));

Если до подключения библиотеки было определено #define NDEBUG , то проверка assert не производится. Таким образом можно выполнять отладку отдельных модулей. В среде Visual Studio #define NDEBUG добавляется автоматически ко всем файлам при выборе Release версии сборки приложения. Воспользуемся данной особенностью не только для работы assert, но и для вывода дополнительной диагностической информации при выборе Debug версии сборки приложения.

Документирование исходного кода будем выполнять в стиле doxygen, так как это является общепринятым в C++. Суть стиля заключается в помещении специально отформатированного комментария в начале каждого файла для краткого описания этого файла (пример для класса COMPort).

/\*\*

 \* @file COMPort.h

 \* @author TAN4UK (tan4ukmak7@gmail.com)

 \* @brief Class for communication with COM port through WinAPI

 \* @version 0.1

 \* @date 2023-02-13

 \*

 \* @copyright Copyright (c) 2023 TAN4UK

 \*

 \*/

Аналогичный комментарий помещается и перед объявлением каждой функции. MSVS поддерживает данный способ документирования и при наведении курсора на функцию показывает ее описание согласно заданному комментарию (рисунок 2.4).

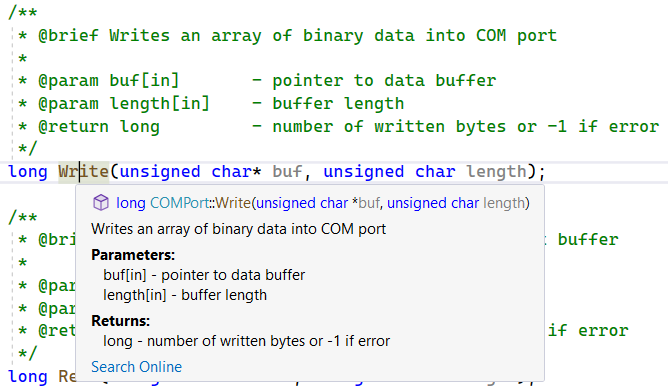


Рисунок 2.4 – Описание функций в стиле doxygen

Doxygen также позволяет генерировать документацию из подготовленного вышеописанным образом исходного кода. Для этого устанавливаем и запускаем приложение Doxywizard, выбираем расположение проекта и задаем параметры генерации [2].

## Разработка драйвера последовательного COM порта

Управлять ПЧ Delta VFD-B можно удаленно по последовательному интерфейсу через разъем RJ-11. Физически интерфейс реализован согласно стандарту EIA/TIA-485(RS-485). Данный стандарт определяет электрические характеристики физического уровня для асинхронного последовательного интерфейса при обмене цифровыми данными с помощью полудуплексной многоточечной дифференциальной линии связи типа «общая шина». Стандарт определяет 3 линии для передачи сигнала: (неинвертирующая), (инвертирующая), (общая). Логические уровни: соответствует логическому «0» (активный), соответствует логическому «1» (неактивный) (рисунок 2.5). Относительно общего провода напряжение и должно находиться в пределах от -7 до 12 вольт, разница в напряжении на дифференциальных выводах и должна составлять не менее ±200 милливольт [3].



Рисунок 2.5 – Логические уровни RS-485

Персональные компьютеры не имеют интерфейса RS-485. Подключение осуществляется с помощью RS-485/USB переходника и отображается в системе Windows как виртуальный последовательный COM порт.

Для доступа к последовательному порту Win32API предоставляет сторонним программам ресурс связи для обмена данными. Ресурс связи – это физическое или логическое устройство, которое предоставляет один двунаправленный асинхронный поток данных. Последовательные порты, параллельные порты, факсы и модемы являются примерами ресурсов связи. Для каждого ресурса связи существует поставщик услуг, состоящий из библиотеки или драйвера, который позволяет приложениям получать доступ к ресурсу [4].

Соединение с последовательным портом осуществляет функция **CreateFile**, которая возвращает дескриптор указанного порта. Дескриптор используется в дальнейшем в качестве идентификатора порта при настройке параметров соединения, а также в функциях чтения и записи.

Класс драйвера последовательного порта **COMPort** необходим для создания интерфейса взаимодействия между приложениями и Win32API функциями управления ресурсом связи. Объявление класса **COMPort**:

#include <Windows.h>

class COMPort

{

private:

    HANDLE          hCOM;

    DWORD           error;

    char            name[9];

    unsigned long   baud;

    unsigned char   dataBit;

    char            parity;

    unsigned char   stopBit;

    unsigned long   tInterval;

    unsigned long   tMultiplier;

    unsigned long   tConstant;

    bool WriteDCB();

    bool TimeoutsSetup();

public:

    COMPort(

        const char\* name = "COM3",

        unsigned long baud = 9600,

        unsigned char dataBit = 8,

        char parity = 'N',

        unsigned char stopBit = 1);

    COMPort(COMPort& other);

    COMPort& operator =(COMPort& other);

    COMPort(COMPort&& other) noexcept;

    COMPort& operator =(COMPort&& other) noexcept;

    ~COMPort();

    bool Open();

    bool isOpen();

    bool Close();

    bool SetConfig(

        unsigned long baud = 9600,

        unsigned char dataBit = 8,

        char parity = 'N',

        unsigned char stopBit = 1);

    bool SetReadTimeouts(

        unsigned long interval = 0,

        unsigned long multiplier = 0,

        unsigned long constant = 1);

    bool ClearReadBuffer();

    long Write(unsigned char\* buf, unsigned char length);

    long Read(unsigned char\* buf, unsigned char length);

};

Подключение Windows.h необходимо для доступа к функциям WinAPI. Рассмотрим члены данного класса:

hCOM – дескриптор открытого порта;

error – последняя ошибка порта;

name – массив символов для хранения имени порта, для виртуального порта имя задается при первом подключении и зависит от разъёма USB, к которому подключен преобразователь интерфейсов;

baud – скорость передачи, бод/с;

dataBit – количество бит данных;

parity – тип проверки четности;

stopBit – количество стоповых битов;

tInterval – максимальный таймаут между приемом двух байт, мс;

tMultiplier – множитель для расчета общего таймаута чтения;

tConstant – константа для расчета общего таймаута чтения, мс.

Класс имеет 2 приватных метода.

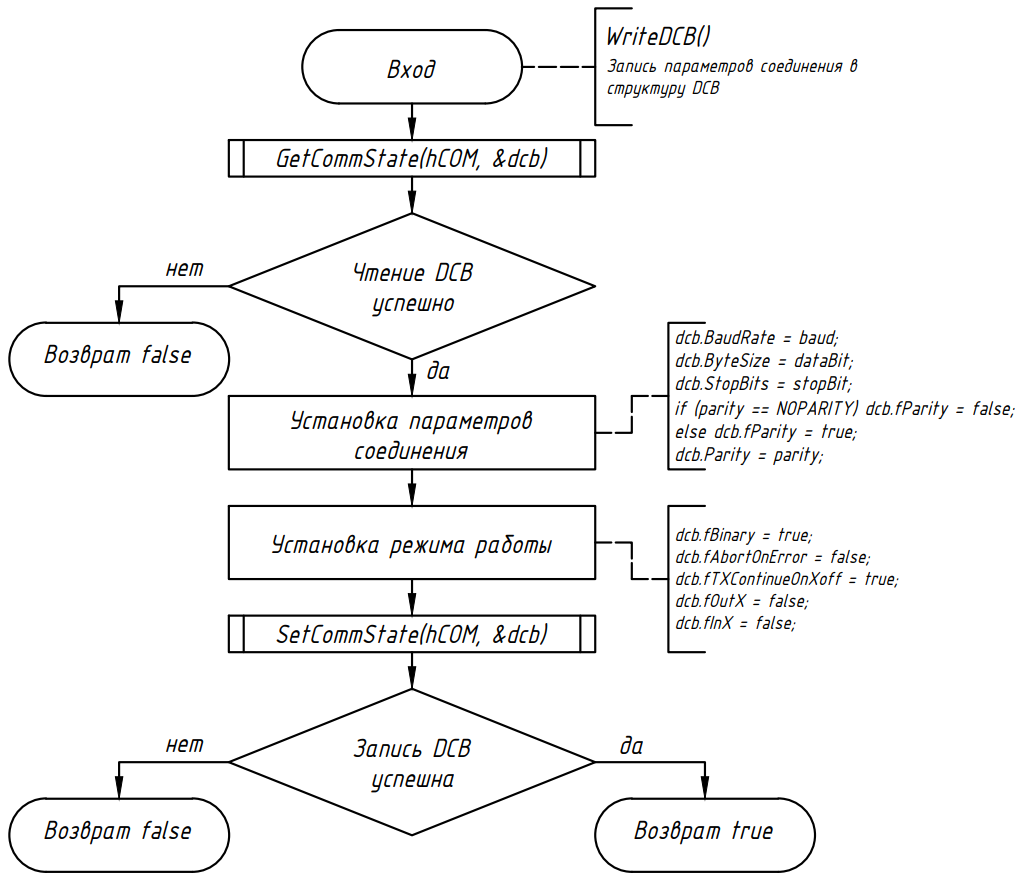


Рисунок 2.6 – Блок-схема алгоритма записи DCB структуры

WriteDCB() – позволяет записывать параметры соединения, которые хранятся в членах класса, во внутреннюю структуру Device Control Block (DCB). Члены структуры DCB определяют такие параметры конфигурации, как скорость, количество битов данных на байт, четность и количество стоп-битов на байт. Другие члены DCB задают специальные символы и обеспечивают проверку четности и управление потоком. Если процессу необходимо изменить только несколько из этих параметров конфигурации, сначала следует вызвать **GetCommState**, чтобы заполнить структуру DCB текущей конфигурацией. Затем можно настроить нужные значения в структуре DCB и перенастроить устройство, вызвав **SetCommState** и указав измененную структуру DCB. Эта процедура гарантирует, что члены структуры DCB, не требующие изменения, содержат неизменные значения (рисунок 2.6).

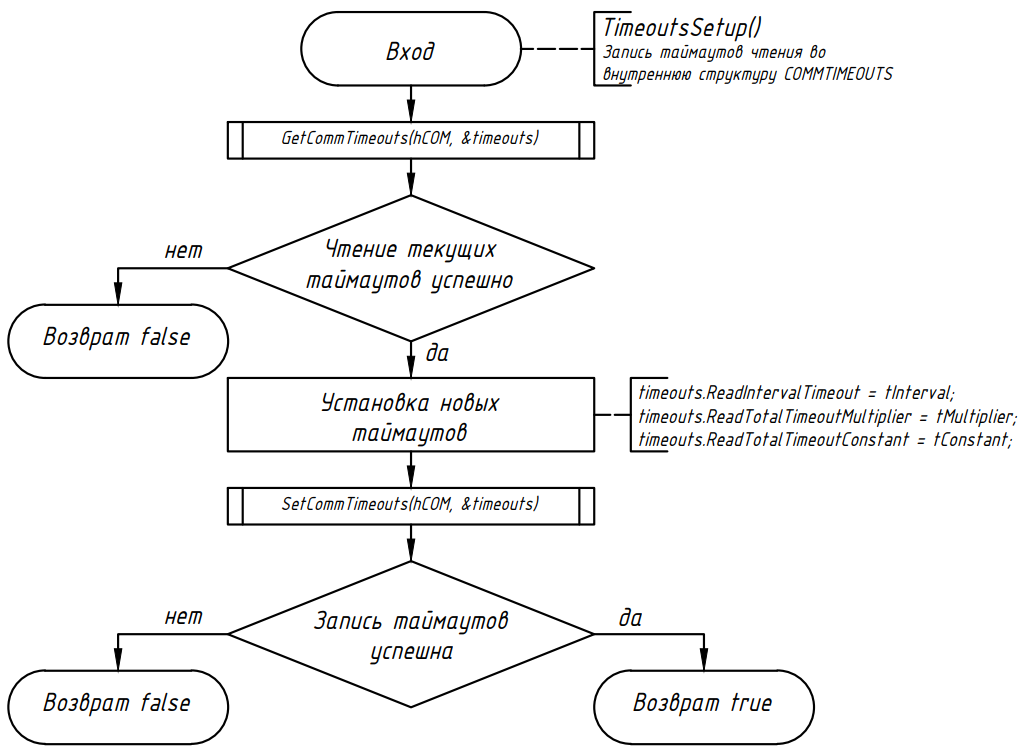


Рисунок 2.7 – Блок-схема алгоритма записи таймаутов

TimeoutsSetup() – записывает значения таймаутов чтения, хранящихся в членах класса, во внутреннюю структуру COMMTIMEOUTS (рисунок 2.7). Для таймаутов операции чтения структура определяет 3 члена:

ReadIntervalTimeout – максимальное время ожидания прибытия следующего байта на линию связи в миллисекундах. Если интервал между поступлением двух байтов превышает это время, операция **ReadFile** завершается и возвращаются все данные из буфера. Значение нуля указывает, что интервал времени ожидания не используется. Значение MAXWORD в сочетании с нулевыми значениями для элементов ReadTotalTimeoutMultiplier и ReadTotalTimeoutConstant указывает, что операция чтения возвращается немедленно с байтами, которые уже были получены, даже если не было принято ни одного байта.

ReadTotalTimeoutMultiplier – множитель, используемый для вычисления общего периода ожидания операций чтения в миллисекундах.

ReadTotalTimeoutConstant – константа, используемая для вычисления общего времени ожидания операций чтения в миллисекундах. Общее время ожидания вычисляется следующим образом:

где – запрашиваемое количество байт для чтения.

Нулевые значения для элементов ReadTotalTimeoutMultiplier и ReadTotalTimeoutConstant указывают, что общее время ожидания не используется для операций чтения.

Класс **COMPort** имеет конструктор по умолчанию, который позволяет при создании нового экземпляра задать имя порта и параметры коммуникации (рисунок 2.8). Если эти параметры не заданы, то берутся значения по умолчанию. Для корректной работы с ресурсом связи доступ к нему должна иметь только одна программа. С этой целью класс содержит конструкторы и операторы для копирования и перемещения (рисунок 2.9). Основная задача этих конструкторов и операторов – закрывать доступ к неиспользуемому дескриптору порта. Таким образом нельзя работать с портом одновременно из нескольких экземпляров класса.

В деструкторе класса вызывается единственная функция **CloseHandle** для закрытия порта.

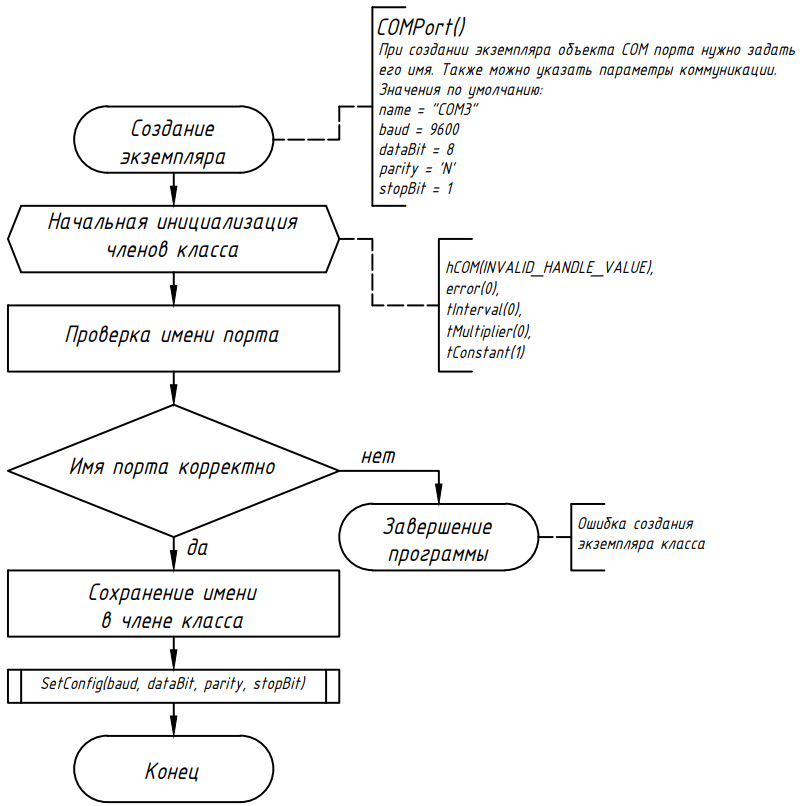


Рисунок 2.8 – Блок-схема алгоритма конструктора по умолчанию класса COMPort

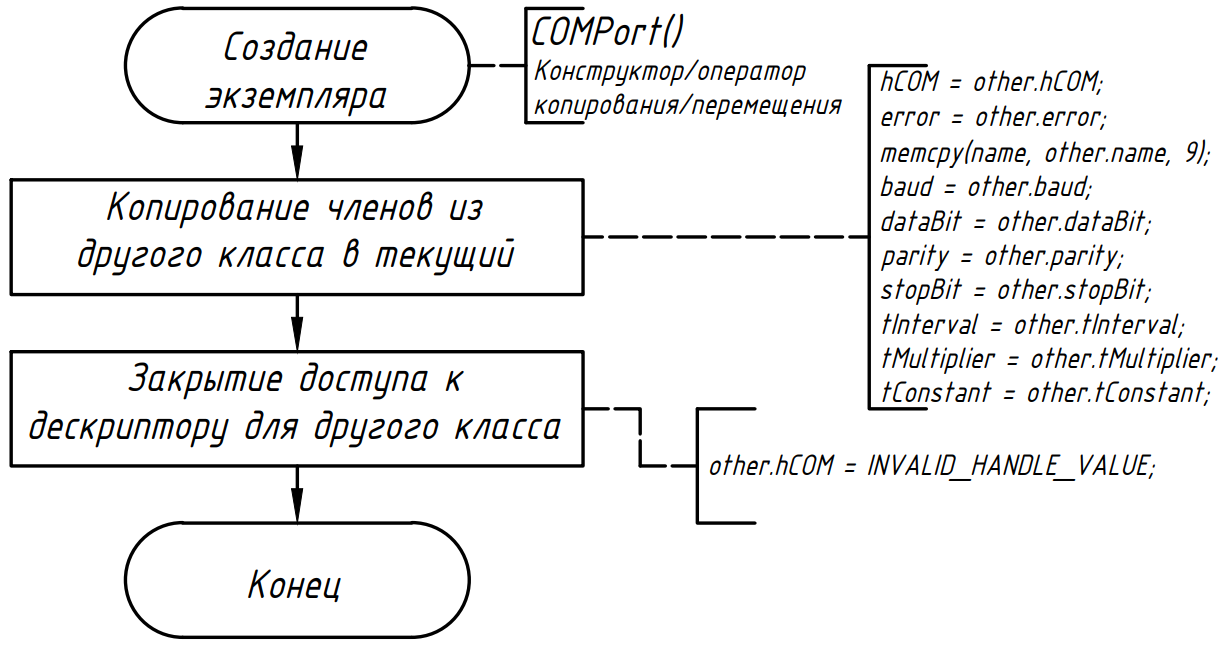


Рисунок 2.9 – Блок-схема алгоритма конструктора/оператора копирования и перемещения класса COMPort

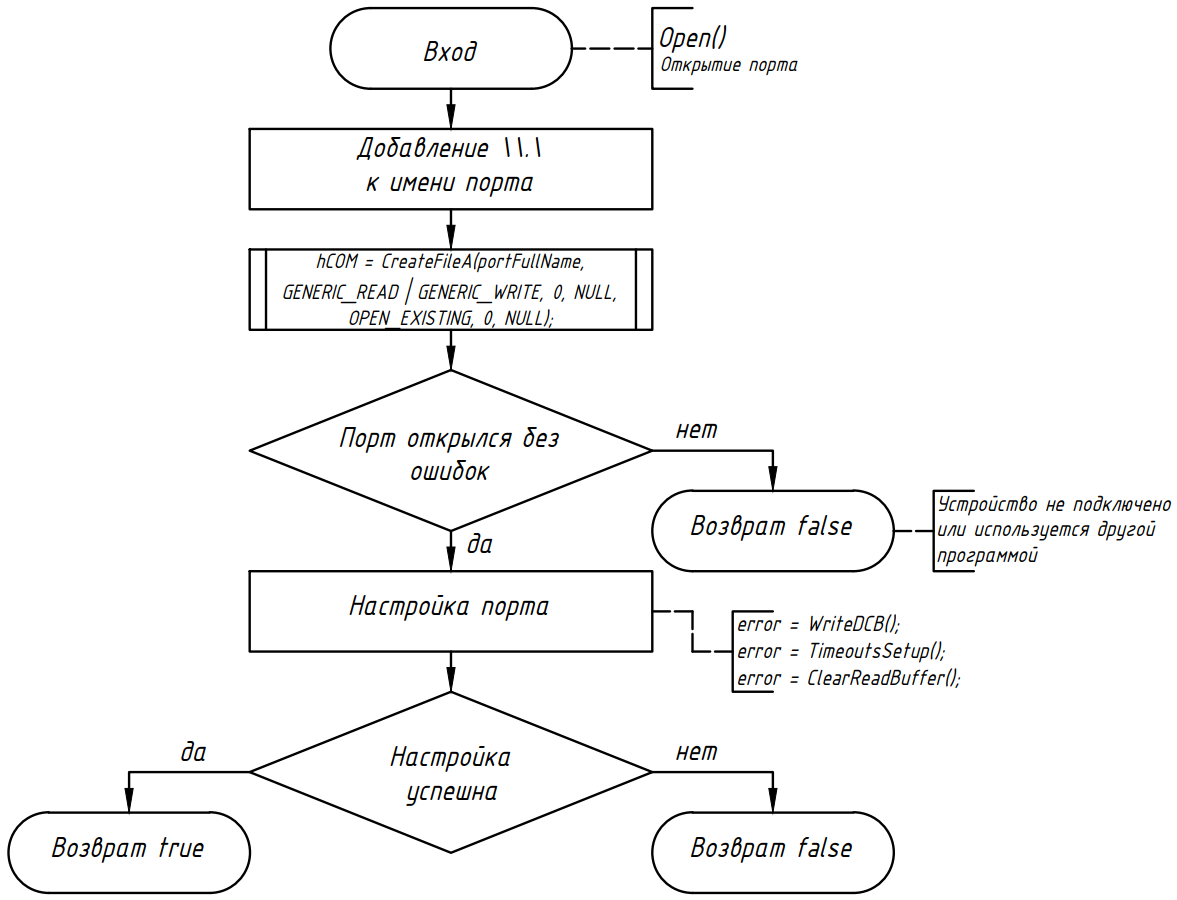


Рисунок 2.10 – Блок-схема алгоритма открытия порта

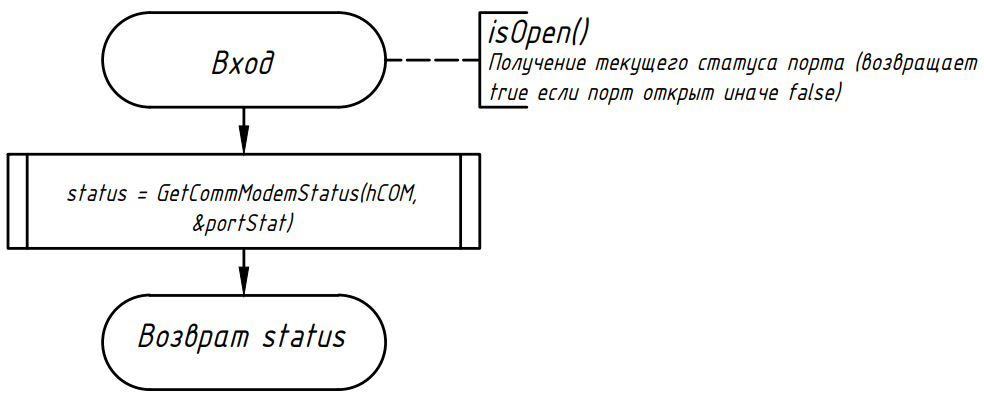


Рисунок 2.11 – Блок-схема алгоритма проверки статуса порта

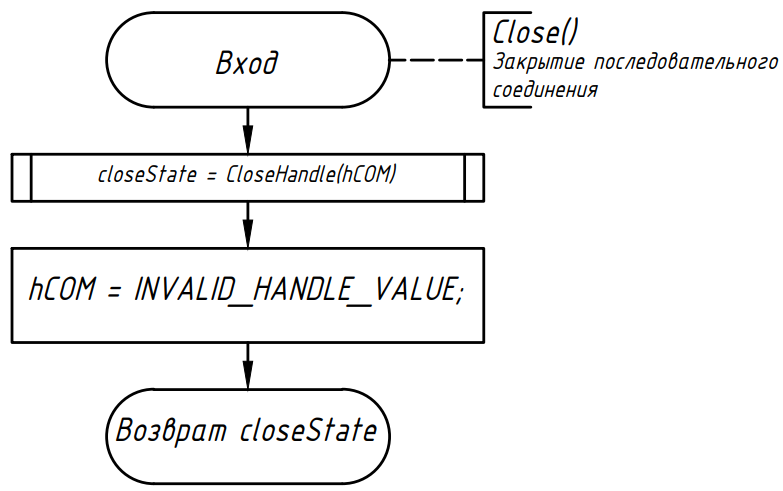


Рисунок 2.12 – Блок-схема алгоритма закрытия порта

Взаимодействие с ресурсом связи в виде последовательного порта происходит аналогично взаимодействию с файлом, из чего следует, что для работы порта его нужно открыть. Метод Open() открывает порт и работает следующим образом: сначала к имени порта добавляется префикс «\\.\» который сообщает системе, что нужно открыть доступ к ресурсу связи, а не к файлу с аналогичным именем. Для открытия вызывается функция **CreateFileA** – версия функции **CreateFile**, которая принимает ASCII строку в качестве имени порта. При успешном открытии функция возвращает дескриптор порта и далее настраиваются параметры соединения порта согласно значениям членов класса (рисунок 2.10). Проверить текущий статус порта можно функцией **GetCommModemStatus** (рисунок 2.11), а закрыть порт – с помощью функции **CloseHandle** (рисунок 2.12).

Изменить параметры коммуникации можно с помощью метода SetConfig(), в котором проверяются заданные пользователем значения и если они корректны то записываются в члены класса, иначе устанавливаются значения по умолчанию. Если порт открыт, то данный метод также записывает параметры в DCB (рисунок 2.13).

SetReadTimeouts() – записывает заданные в качестве аргументов таймауты для синхронной операции чтения в члены класса. Если порт открыт, то таймауты также записываются и во внутреннюю структуру (рисунок 2.15).

Windows поддерживает синхронные и асинхронные (перекрывающиеся) операции ввода-вывода файлов в ресурсах последовательной связи. Перекрывающиеся операции позволяют вызывающей цепочке выполнять другие задачи во время выполнения операции в фоновом режиме. При синхронной операции функция возвращает управление вызывающей цепочке после завершения определенных в этой функции действий или по истечению таймаута. Данный класс поддерживает только синхронные операции.

ClearReadBuffer() очищает входной буфер с помощью функции **PurgeComm** (рисунок 2.14).

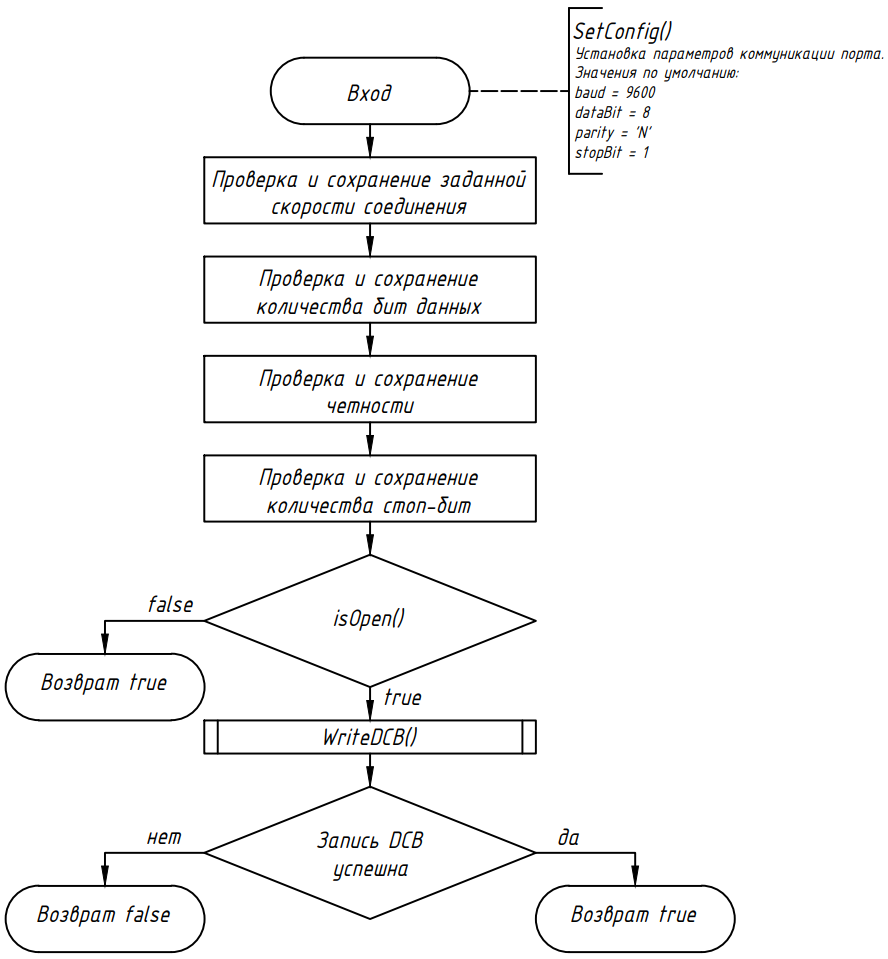


Рисунок 2.13 – Блок-схема алгоритма изменения параметров коммуникации

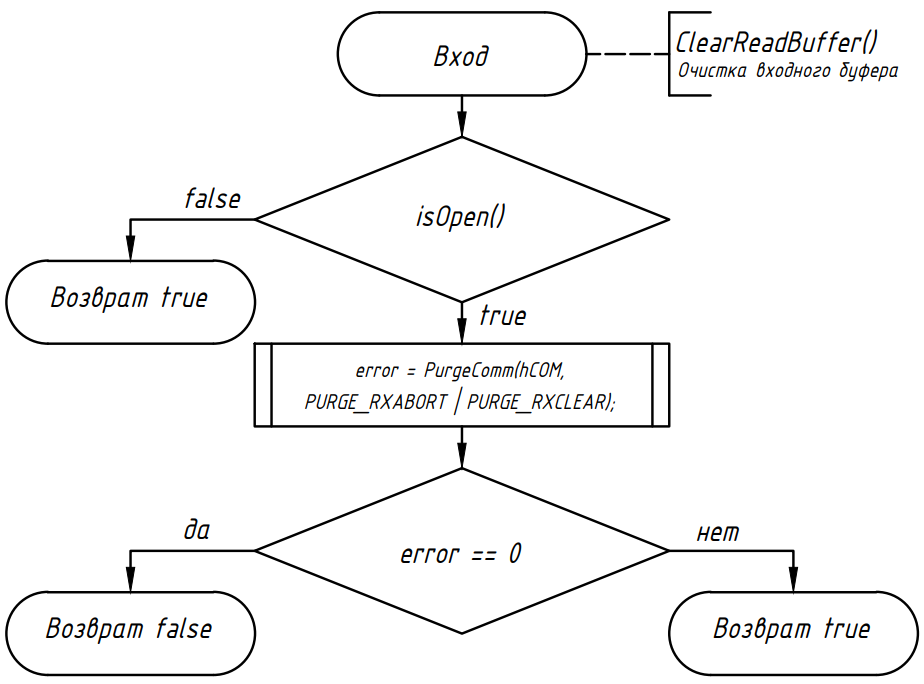


Рисунок 2.14 – Блок-схема алгоритма очистки входного буфера порта



Рисунок 2.15 – Блок-схема алгоритма задания таймаутов функции чтения

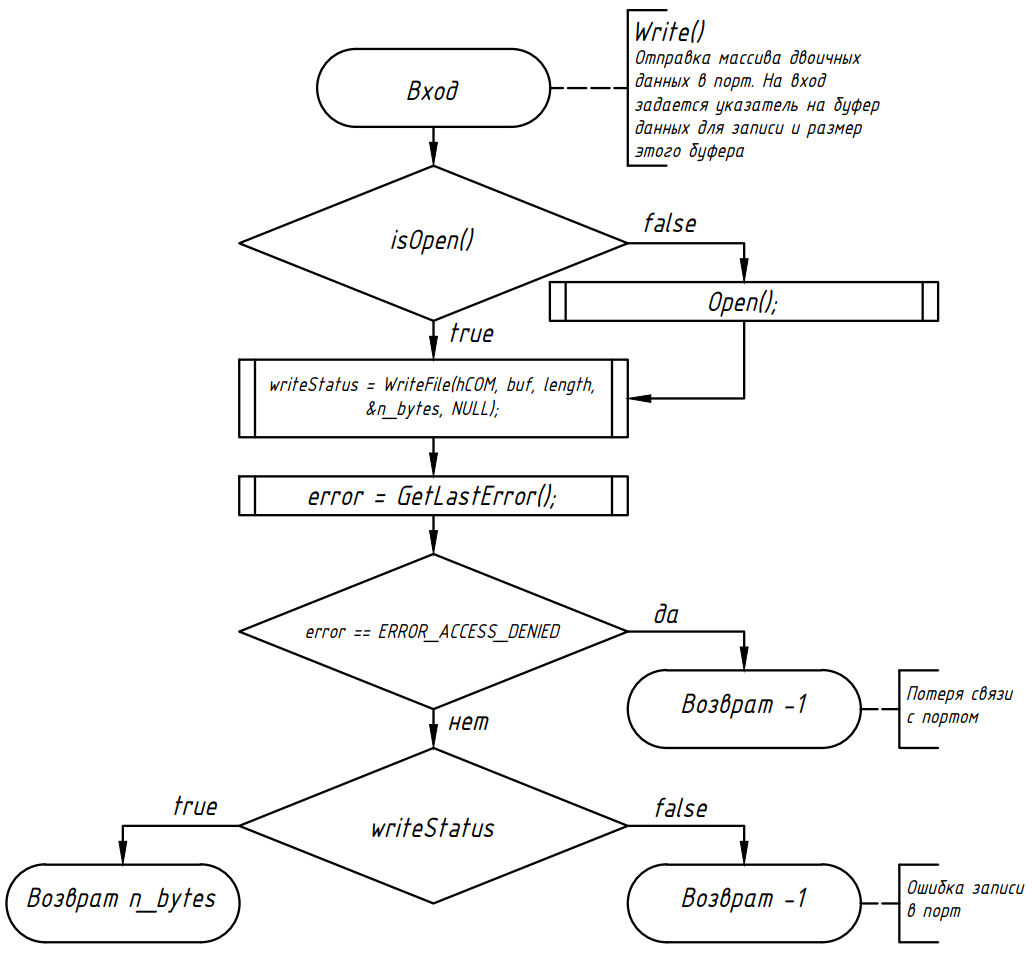


Рисунок 2.16 – Блок-схема алгоритма отправки данных в порт

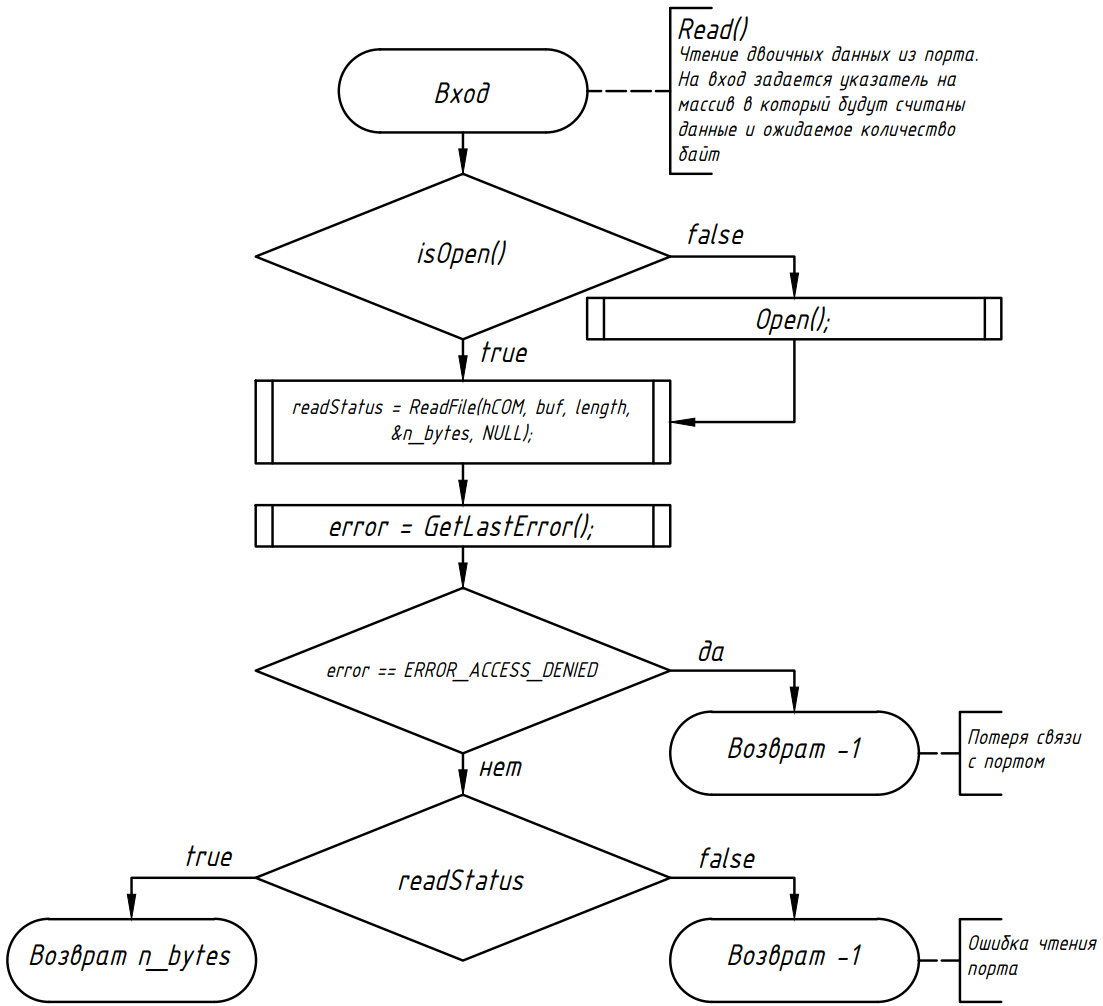


Рисунок 2.17 – Блок-схема алгоритма чтения данных из порта

Основными операциями взаимодействия с портом являются операции записи и чтения. Методы Write() и Read() реализующие данные операции принимают указатель на буфер с бинарными данными и количество байт, которые требуется записать или ожидается принять. Если метод чтения или записи случайно вызван перед открытием порта, то порт будет автоматически открыт. Далее вызывается функция **WriteFile** и **ReadFile** для записи и чтения соответственно, после чего проверяется результат выполнения операции. В случае успеха возвращается количество прочитанных или записанных байт, что обычно равно указанному пользователем количеству байт. При ошибке возвращается -1. При истечении таймаута операция чтения считается успешной и возвращается количество полученных байт которое может быть нулевым.

## Разработка драйвера ModbusRTU

Преобразователь частоты VFD-В может быть настроен для работы в Modbus сетях.

Modbus — открытый коммуникационный протокол, основанный на архитектуре ведущий — ведомый (англ. master-slave; в стандарте Modbus используются термины client-server). Широко применяется в промышленности для организации связи между электронными устройствами. Может использоваться для передачи данных через последовательные линии связи RS-485, RS-422, RS-232 и сети TCP/IP (Modbus TCP). Сеть состоит из одного ведущего устройства и несколько ведомых (до 247), подключенных к одной шине. Обмен данными в виде коммуникационных блоков (рисунок 2.18) начинается с отправки запроса ведущим устройством. На запрос отвечает ведомое устройство, адрес которого указан в запросе. Ответ может содержать подтверждение выполнения операции, запрашиваемые данные или код ошибки [5].

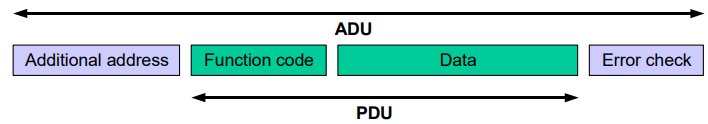


Рисунок 2.18 – Стандартный коммуникационный блок Modbus

Существует 2 режима обмена данными в сетях. ASCII (Американский Стандартный Код для Информационного Обмена) – использует 2 ASCII символа для кодирования одного байта сообщения, которое начинается стартовым символом ‘:’ и заканчивается последовательно ‘CRLF’. RTU (Периферийное устройство) - использует двоичное кодирование и для наглядности может представлять один байт в виде двух 4-битных шестнадцатеричных символов, что повышает плотность данных. RTU режим является предпочтительным и будет использоваться при коммуникации с ПЧ.

В RTU режиме 1 байт данных кодируется 8 битами в двоичном виде и передается 11 битным кадром (рисунок 2.19). Кадр состоит из 1 стартового бита, 8 бит данных (младший бит отправляется первым), 1 бита проверки четности, 1 стопового бита. По умолчанию выполняется проверка на четность (even parity), разрешены также и другие типы проверки четности.

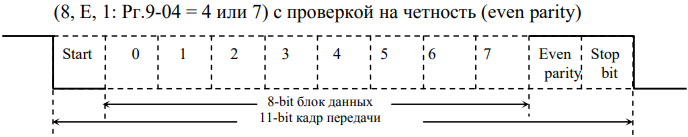


Рисунок 2.19 – 11-bit кадр (для 8-bit блока данных) с проверкой на четность

Все сообщения в Modbus RTU передаются блоками (рисунок 2.20). В каждом блоке может быть до 252 байт данных. Передача блока завершается интервалом тишины не менее 3,5 символов, после чего начинается передача нового блока. Если интервал тишины продолжительностью 3,5 символа возник во время передачи блока, принимающее устройство завершает прием сообщения, сбрасывает принимающий буфер и следующий байт будет воспринят как начало следующего блока. Если новое сообщение начнется раньше 3,5-символьного интервала, принимающее устройство воспримет ею как продолжение предыдущего блока. В этом случае будет установлена ошибка из-за несовпадения контрольных сумм [6].

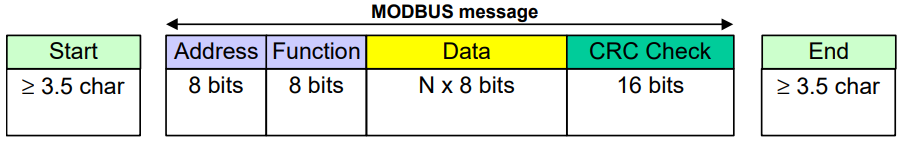


Рисунок 2.20 – Формат коммуникационного блока Modbus RTU

RTU режим также включает проверку на ошибки целого блока с помощью Cyclic redundancy checking (CRC). CRC рассчитывается на основании данных всего блока и состоит из 16-битного значения, которое добавляется в конце каждого блока (сначала младший байт, потом старший).

Класс **ModbusRTUClient** принимает запросы на обмен данными и командами в виде функций описанных в [5] и отвечает за формирование коммуникационных блоков, а также за их отправку, получение и проверку. Объявление класса **ModbusRTUClient**:

// #define FAKE\_PORT

#ifdef FAKE\_PORT

#include "COMPortFake.h"

#else

#include "COMPort.h"

#endif

class ModbusRTUClient

{

private:

#ifdef FAKE\_PORT

    COMPortFake     COM;

#else

    COMPort         COM;

#endif

    unsigned char   devAddress;

    unsigned char   transmitAttempts;

    unsigned char   wBuf[256];

    unsigned char   rBuf[256];

    unsigned int CRC16(unsigned char\* data, unsigned char length);

    bool responseCRCCheck(unsigned char pduBytes);

    bool Transfer(unsigned char wPDUBytes, unsigned char rPDUBytes);

    void PrintException(unsigned int attempt);

public:

#ifdef FAKE\_PORT

    ModbusRTUClient(unsigned char devAddress = 1,

        COMPortFake com = { "COM3", 19200, 8, 'E', 1 });

#else

    ModbusRTUClient(unsigned char devAddress = 1,

        COMPort com = { "COM3", 19200, 8, 'E', 1 });

#endif

    bool ReadHoldingRegisters(

        unsigned short startAddress,

        unsigned char nRegisters,

        unsigned short\* buf);

    bool WriteSingleRegister(

        unsigned short regAddress,

        unsigned short regValue);

    void SetNumberOfTransmitAttempts(unsigned char attempts = 1);

};

Для доступа к последовательному порту подключаем COMPort.h или COMPortFake.h в зависимости от наличия #define FAKE\_PORT. COMPort.h позволяет подключиться к ПЧ с помощью виртуального порта Windows, принцип работы которого описан в предыдущей главе. COMPortFake.h возвращает заранее заданные стандартные ответы на запросы Modbus. Такое решение позволяет отлаживать высокоуровневые библиотеки без физического подключения к ПЧ. Рассмотрим члены данного класса:

COM – экземпляр класса последовательного порта;

devAddress – адрес сервера для подключения;

transmitAttempts – общее количество попыток обмена сообщением. Если данное значение равно 1, то при возникновении ошибки обмена драйвер Modbus вернет вызывающей цепочке сообщение об ошибке. Если значение равно 3, то при ошибке драйвер попытается обменяться сообщением еще 2 раза, прежде чем вернуть ошибку вызывающей цепочке;

wBuf – буфер, хранящий коммуникационный блок, предназначенный для отправки;

rBuf – буфер, хранящий принятый коммуникационный блок.

Класс имеет 4 приватных метода.

CRC16() – рассчитывает CRC по алгоритму (рисунок 2.21), представленному в [7, c. 125].

responseCRCCheck() – сравнивает CRC из принятого сообщения с рассчитанным CRC на основании данных в этом сообщении. Если они равны, то возвращается true (рисунок 2.22).

Transfer() – является основным методом обмена сообщениями между клиентом и сервером. Данный метод принимает размер сформированного вызывающей его функцией PDU запроса и ожидаемый размер PDU ответа. Далее формируется полный ADU путем добавления адреса сервера в буфер для записи и расчета CRC сообщения. Сформированное сообщение запроса отправляется в порт и производится подготовка входного буфера для приема ответа путем очистки этого буфера. Далее в течение заданного таймаута ожидается получение ответа, который сохраняется в буфер для чтения. Результат чтения ответа проверяется на возможные ошибки (получения ответа исключения, несовпадение принятого количества байт с ожидаемым, ошибка CRC) и в случае возникновения таковой увеличивается таймаут и производятся повторные попытки обмена сообщением.

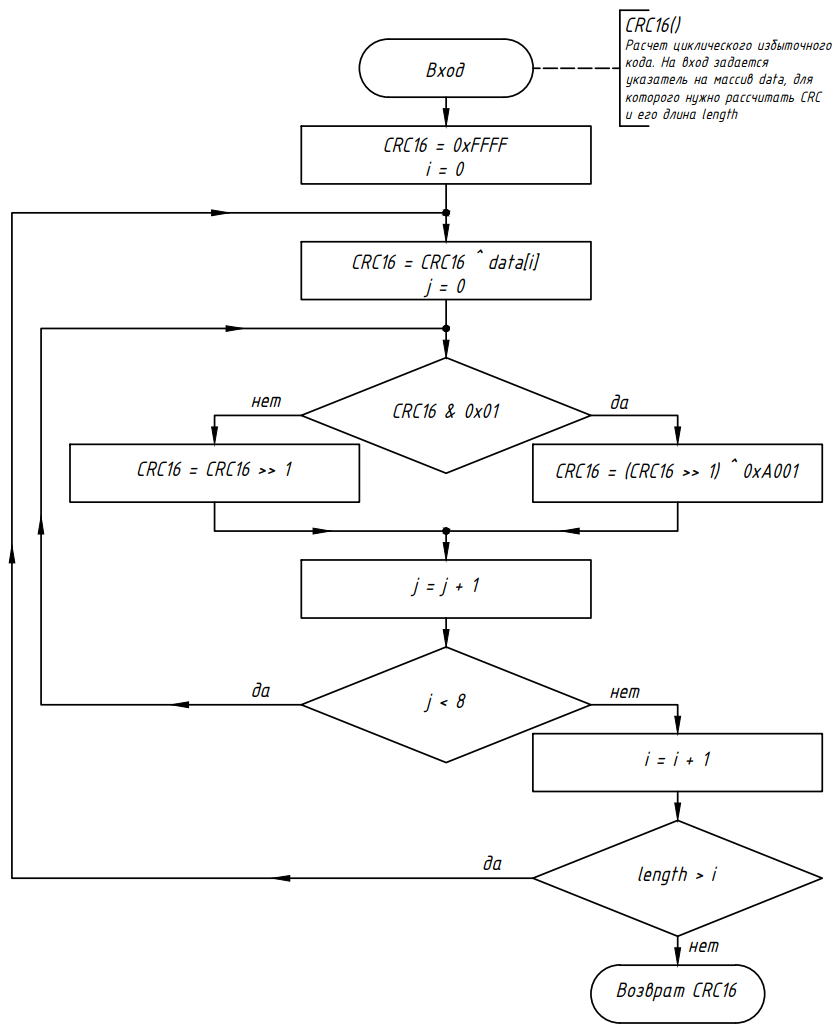


Рисунок 2.21 – Блок-схема алгоритма расчета CRC

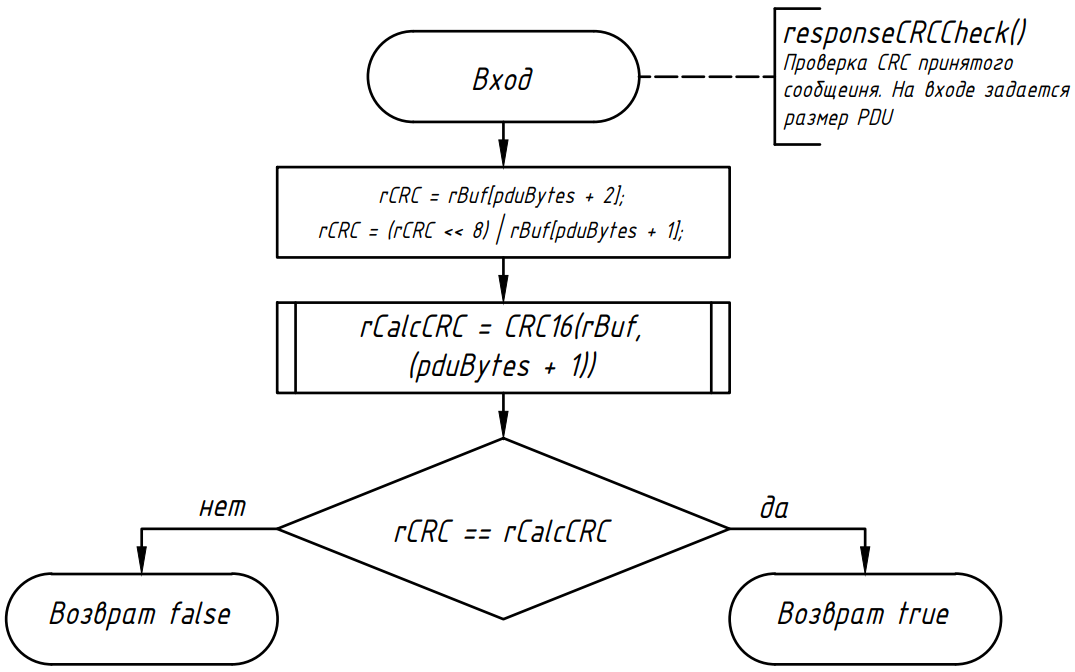


Рисунок 2.22 – Блок-схема алгоритма проверки CRC принятого сообщения



Рисунок 2.23 – Блок-схема алгоритма обмена сообщениями

Класс **ModbusRTUClient** имеет конструктор по умолчанию, который позволяет при создании нового экземпляра задать адрес сервера и параметры последовательного соединения (рисунок 2.24). Если эти параметры не заданы, то берутся значения по умолчанию. Конструктор устанавливает количество попыток обмена равное пяти, проверяет корректность адреса, открывает порт и устанавливает значения таймаутов (1, 0, 1000). При таких таймаутах максимальное время ожидания ответного сообщения равно 1 секунде, а при получении ответного сообщения или исключения функция чтения вернет управление сразу же.

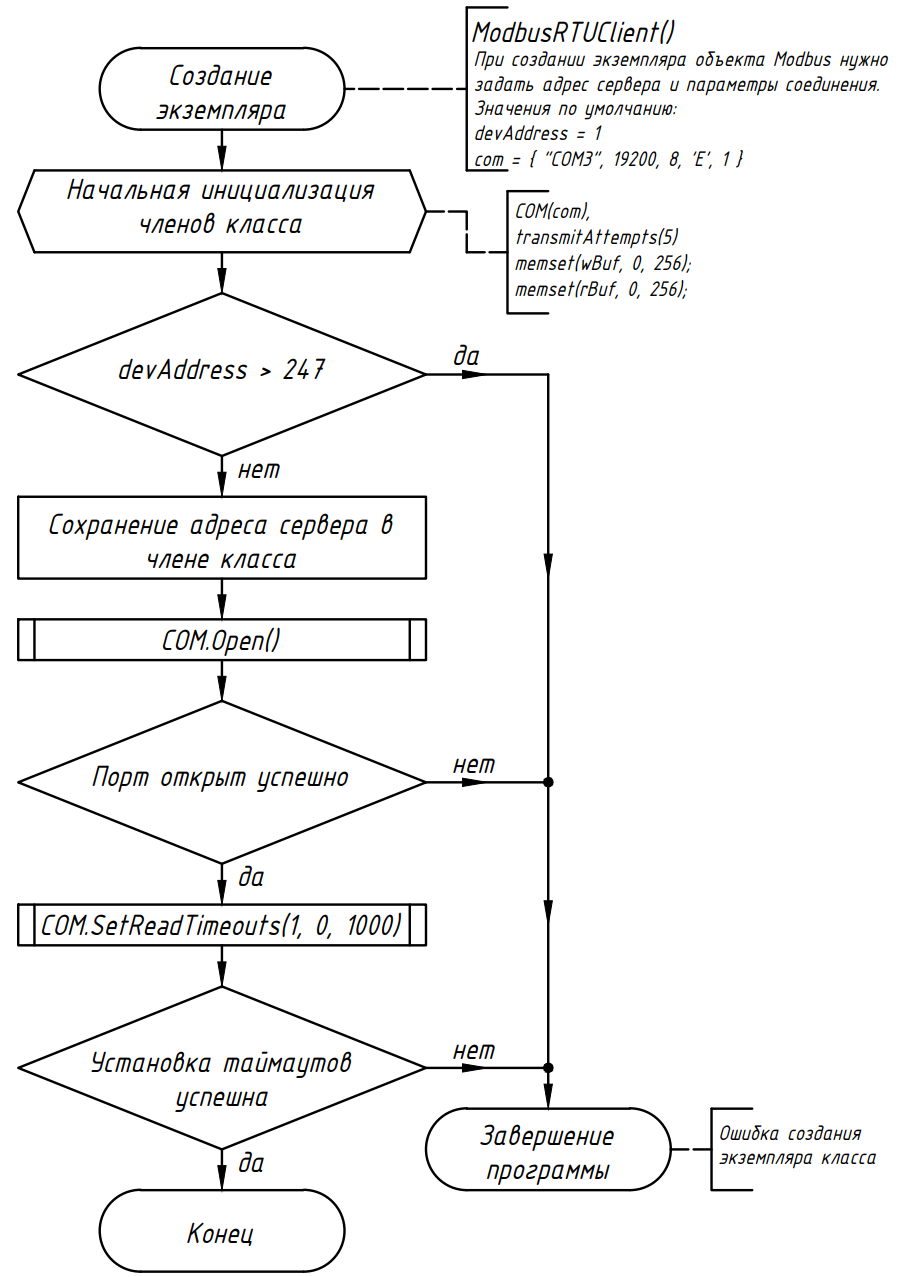


Рисунок 2.24 – Блок-схема алгоритма конструктора по умолчанию класса ModbusRTUClient



Рисунок 2.25 – Блок-схема алгоритма чтения нескольких регистров

Для чтения данных с сервера используется функция с кодом **0x03** (Read Holding Registers). Реализация данной функции (рисунок 2.25) проверяет входные аргументы и разрешает запрос к серверу только тогда, когда количество регистров для чтения не превышает максимальное и все регистры находятся в допустимом диапазоне адресов. Далее формируется PDU запроса и отправляется на сервер. Если в ответном сообщении количество байт данных соответствует запрашиваемому количеству, то эти данные конвертируются в 16 битные значения и записываются в предоставленный буфер.

Для записи одного регистра используется функция с кодом **0x06** (Write Single Register). Функция формирует PDU запроса, отправляет его и проверяет ответное сообщение на равенство запросу (рисунок 2.26).

SetNumberOfTransmitAttempts() устанавливает количество попыток обмена сообщением.

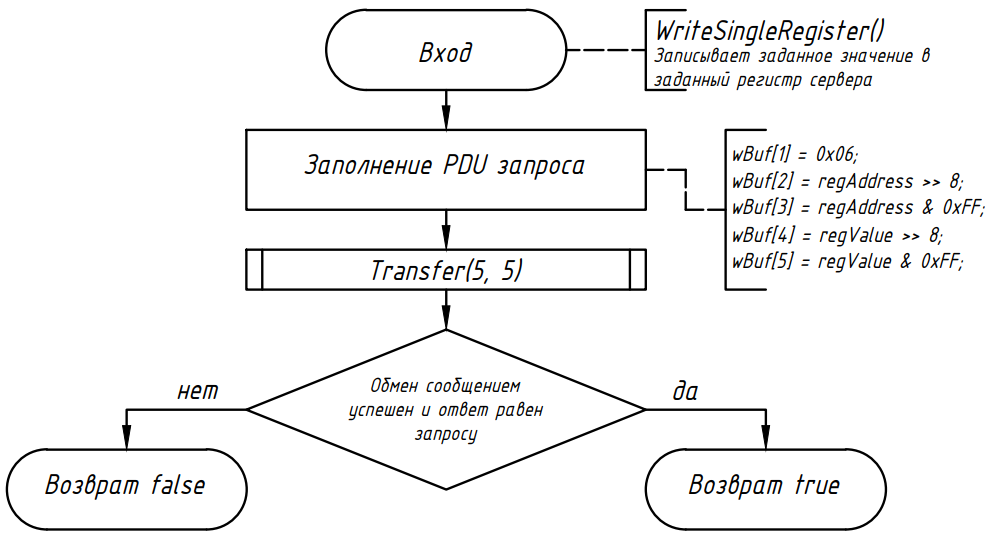


Рисунок 2.26 – Блок-схема алгоритма записи одного регистра

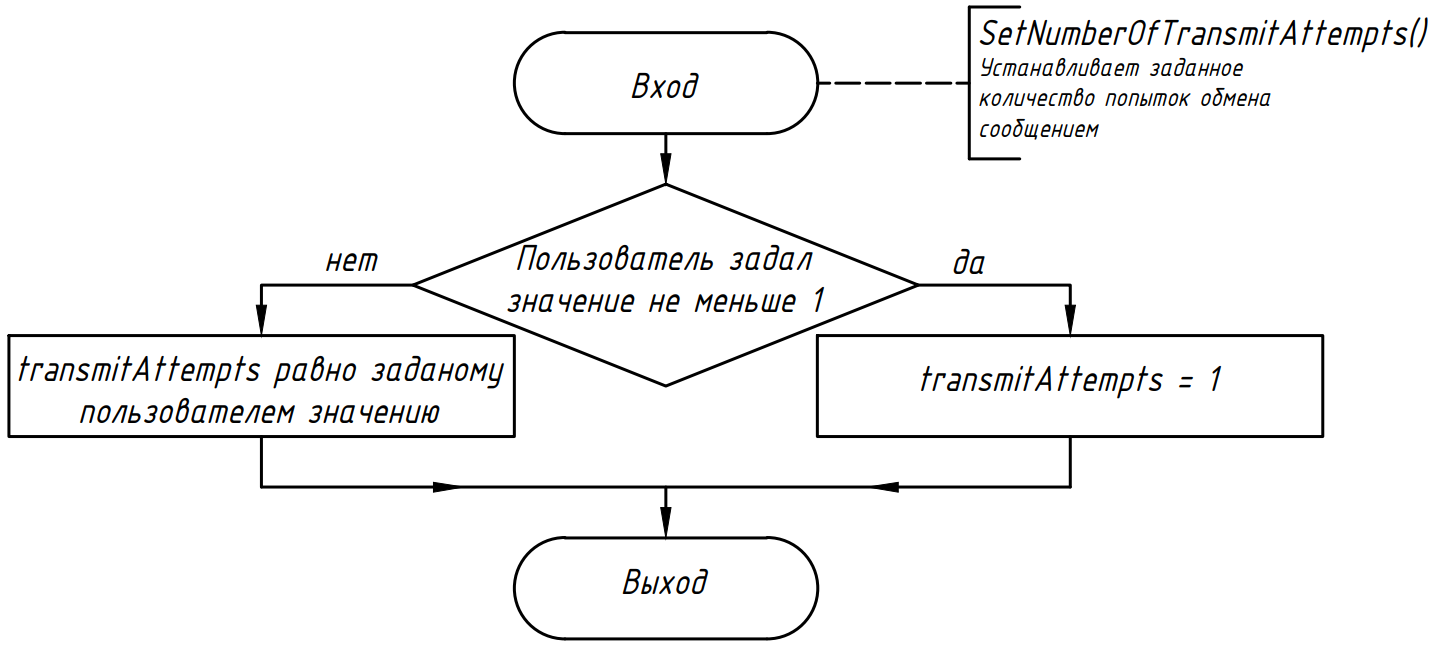


Рисунок 2.27 – Блок-схема алгоритма установки количества попыток обмена

## Разработка драйвера ПЧ

Взаимодействие с ПЧ осуществляется путем записи и чтения внутренних регистров (Holding Registers). Существуют регистры параметров, управления и статуса. Записывать регистры можно по одному с помощью функции **0x06** (Write Single Register), а считывать с помощью функции **0x03** (Read Holding Registers), которая по стандарту поддерживает считывание до 125 регистров. Согласно [7, c. 122] ПЧ VFD-В гарантированно позволяет считать максимум 12 параметров.

Класс **VFD** преобразует команды управления двигателем и чтения параметров в соответствующие адреса и значения регистров [7, c. 125], подходящие для обмена по протоколу Modbus. Объявление класса **VFD**:

#include "ModbusRTUClient.h"

typedef struct VFD\_status {

    struct {

        bool RUN;

        bool STOP;

        bool JOG;

        bool FWD;

        bool REW;

    } LED;

    bool F;

    bool H;

    bool u;

    bool controlFrequencyBySerialInterface;

    bool controlFrequencyByAnalogSignal;

    bool controlVFDBySerialInterface;

    bool parametersBlocked;

    bool VFDCurrentState;

    bool JOGcommand;

} VFD\_status\_t;

typedef struct VFD\_param {

    double FrequencyCommand;// 0x2102

    double OutFrequency;    // 0x2103

    double OutCurrent;      // 0x2104

    double DCVoltage;       // 0x2105

    double OutVoltage;      // 0x2106

    double PowerFactor;     // 0x210A

    double OutTorque;       // 0x210B

    double MotorSpeed;      // 0x210C

} VFD\_param\_t;

class VFD

{

private:

    ModbusRTUClient MB;             // Instance of ModbusRTUClient class

    double          maxFrequency;   // Maximum output motor frequency (01-00 value, default 50Hz)

public:

    VFD(ModbusRTUClient mb = { 1, {"COM3", 9600, 8, 'E', 1} });

    bool Run(unsigned short direction = 0);

    bool Stop();

    bool ChangeFrequency(

        double curFreq = 0,

        double newFreq = 50,

        double changeTime = 1);

    bool ReadParameterRegisters(VFD\_status\_t\* status, VFD\_param\_t\* param);

    bool GetOutPower(double\* power);

    bool GetVFDTemperature(double\* temp);

    bool ReadMaxFrequency(double\* maxFreq = nullptr);

    bool SetFrequency(double freq);

    bool SetAccelerationTime(double time);

    bool SetDecelerationTime(double time);

    bool SetWatchdog(double time = 0);

    bool GetParam(unsigned short addr, unsigned short\* val);

    bool SetParam(unsigned short addr, unsigned short val);

};

Клас содержит подключение ModbusRTUClient.h для доступа к функциям протокола Modbus, а объявление структур VFD\_status\_t и VFD\_param\_t нужно для хранения статуса и измеренных значений ПЧ. Рассмотрим члены данного класса:

MB – экземпляр класса Modbus клиента;

maxFrequency – максимальная выходная частота ПЧ. По умолчанию задано 50 Гц, также есть возможность считать ее как значение параметра **01-00**.

Класс не имеет приватных методов. Конструктор по умолчанию устанавливает значение 50Гц в параметр maxFrequency и позволяет задать параметры Modbus соединения, которые будут установлены по умолчанию, если пользователь их не задал (рисунок 2.28).

Запуск и остановка ПЧ осуществляется с помощью методов Run() (рисунок 2.29) и Stop() (рисунок 2.30). Метод Run() позволяет задать направление вращения при запуске путем задания значения входного параметра (0 – без изменения, 1 – вперед, 2 – назад, 3 – изменить направление).

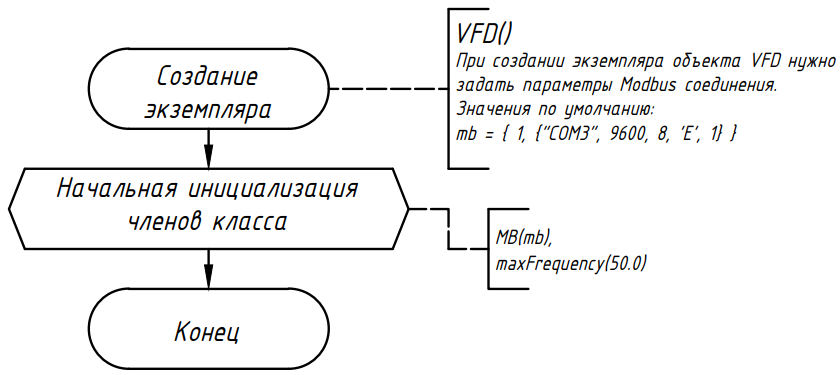


Рисунок 2.28 – Блок-схема алгоритма конструктора по умолчанию класса VFD

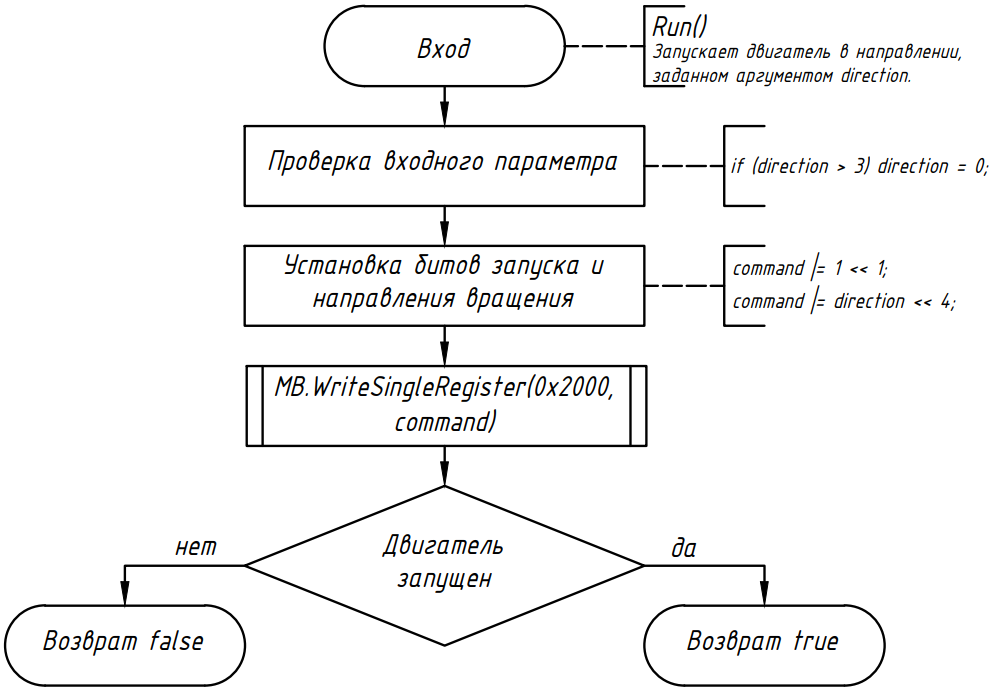


Рисунок 2.29 – Блок-схема алгоритма запуска двигателя

Дополнительно существует метод ChangeFrequency(), позволяющий установить требуемую частоту вращения за указанное время. Алгоритм метода рассчитывает нужное направление вращения и время ускорения/замедления в зависимости от входных параметров (рисунок 2.31).

ReadParameterRegisters() считывает 12 регистров (0x2101-0x210C) текущего статуса и параметров двигателя, записывает их в структуры VFD\_status\_t и VFD\_param\_t соответственно считанным параметрам (рисунок 2.32).

GetOutPower() считывает регистр 0x210F с текущим значением выходной мощности ПЧ и умножает это значение на коэффициент 0.01 для преобразования шестнадцатеричного значения регистра в значение мощности (рисунок 2.33). Аналогичным образом работают методы чтения GetVFDTemperature(), ReadMaxFrequency(), GetParam(), которые записывают значение в переменную через аргумент указатель и возвращают true при успешном чтении. Метод GetParam() позволяет считать нужный параметр ПЧ напрямую и дополнительным аргументом принимает адрес регистра параметра.

SetFrequency() устанавливает частоту вращения двигателя. Метод проверяет корректность введенного пользователем аргумента, умножает его на коэффициент 100 для получения шестнадцатеричного значения и записывает в регистр 0x2001 (рисунок 2.34). Аналогичным образом работают методы записи параметров SetAccelerationTime(), SetDecelerationTime(), SetParam(). Метод SetParam() позволяет записать нужный параметр ПЧ напрямую и дополнительным аргументом принимает адрес регистра параметра.

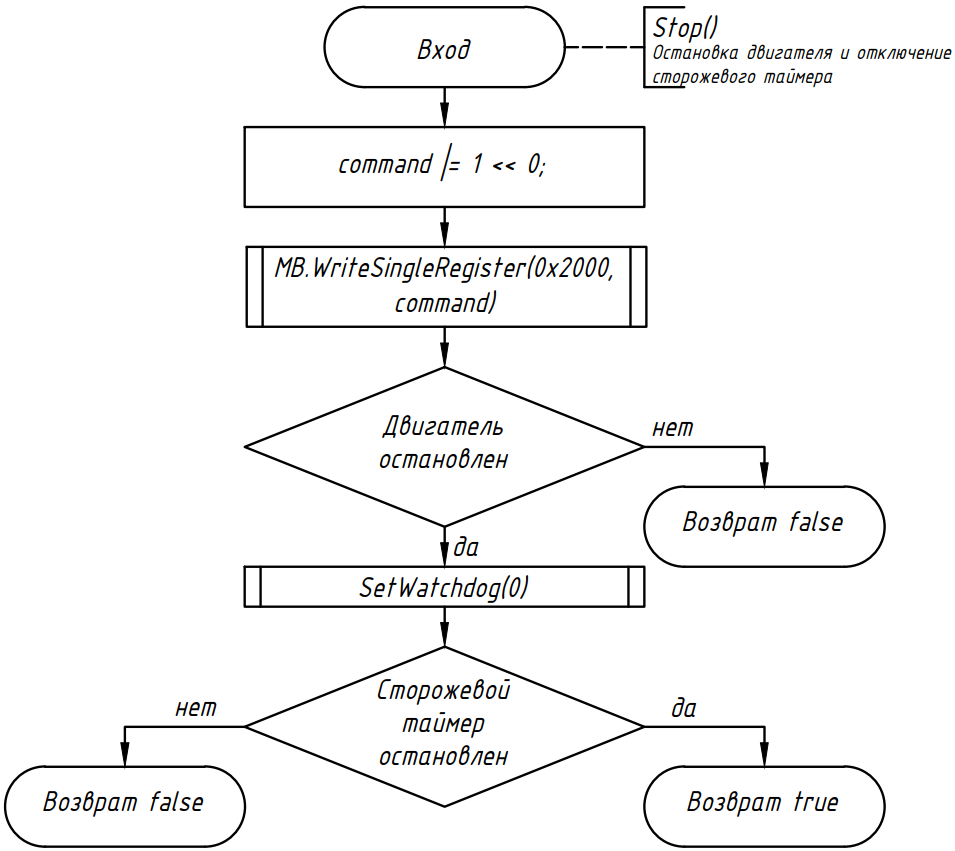


Рисунок 2.30 – Блок-схема алгоритма остановки двигателя

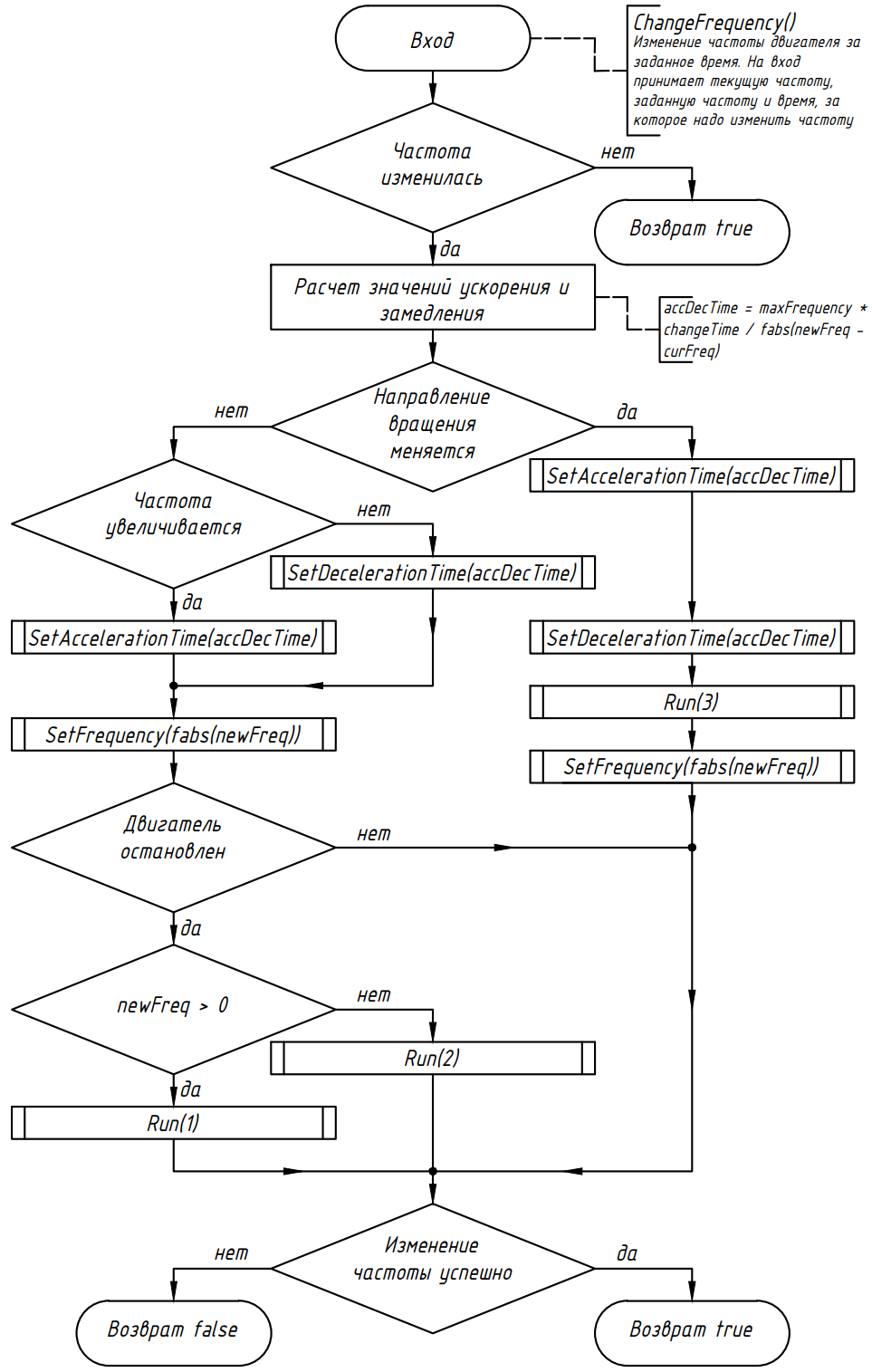


Рисунок 2.31 – Блок-схема алгоритма изменения частоты двигателя

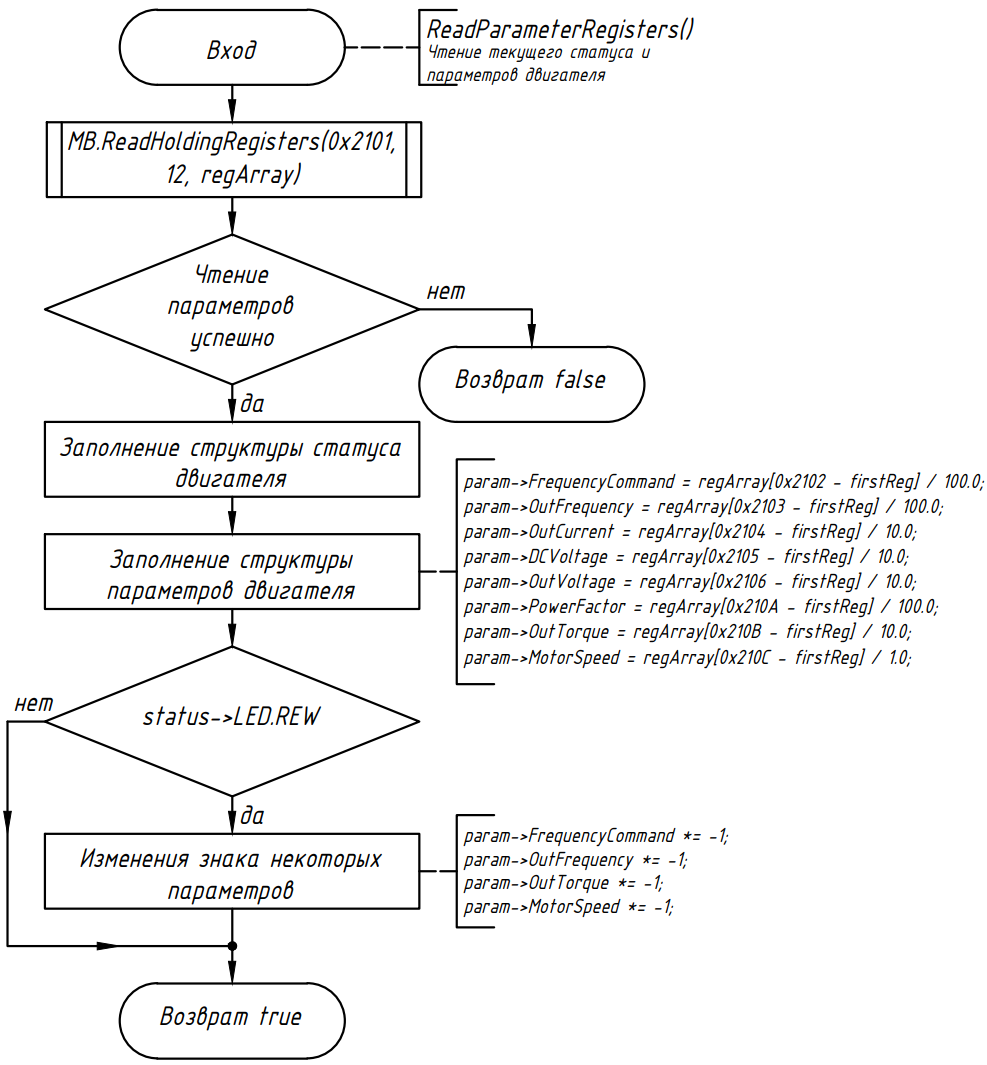


Рисунок 2.32 – Блок-схема алгоритма чтения статуса и параметров двигателя

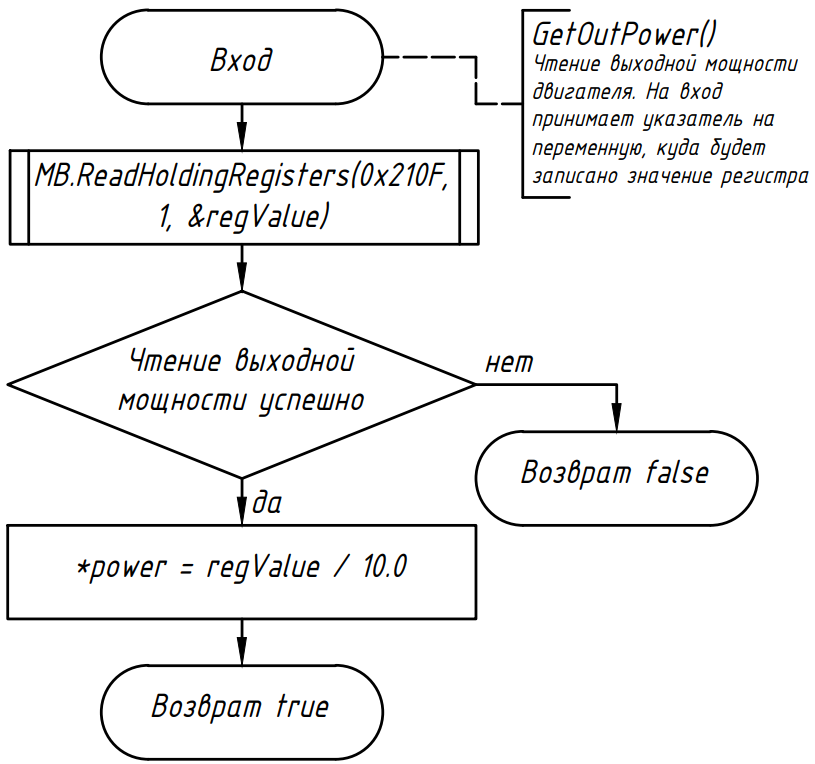


Рисунок 2.33 – Блок-схема алгоритма чтения выходной мощности двигателя

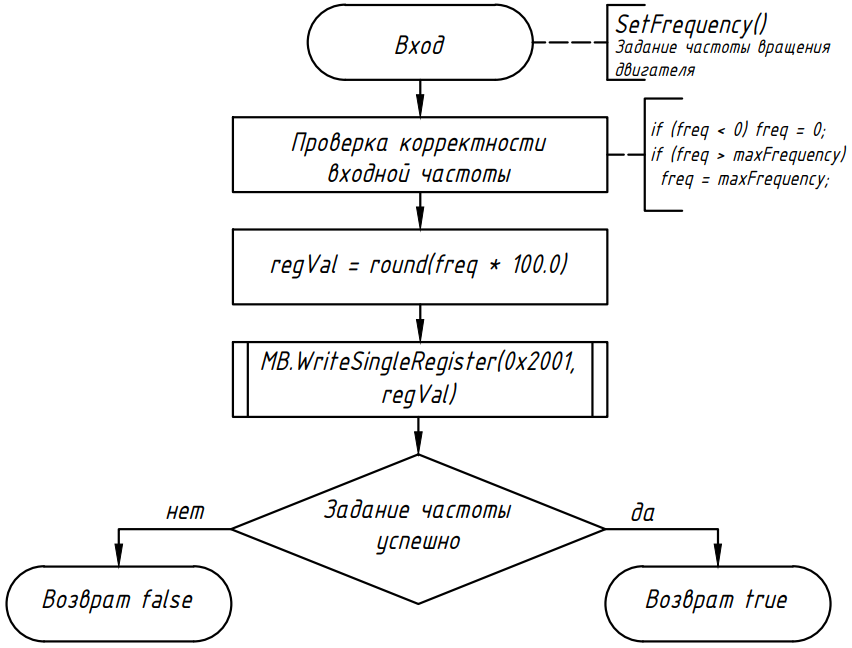


Рисунок 2.34 – Блок-схема алгоритма частоты вращения двигателя

С помощью метода SetWatchdog() задается время ожидания сторожевого таймера, что позволяет останавливать двигатель, если в течение указанного времени нет никаких Modbus запросов от клиента (рисунок 2.35). Установка нулевого значения времени отключает сторожевой таймер.

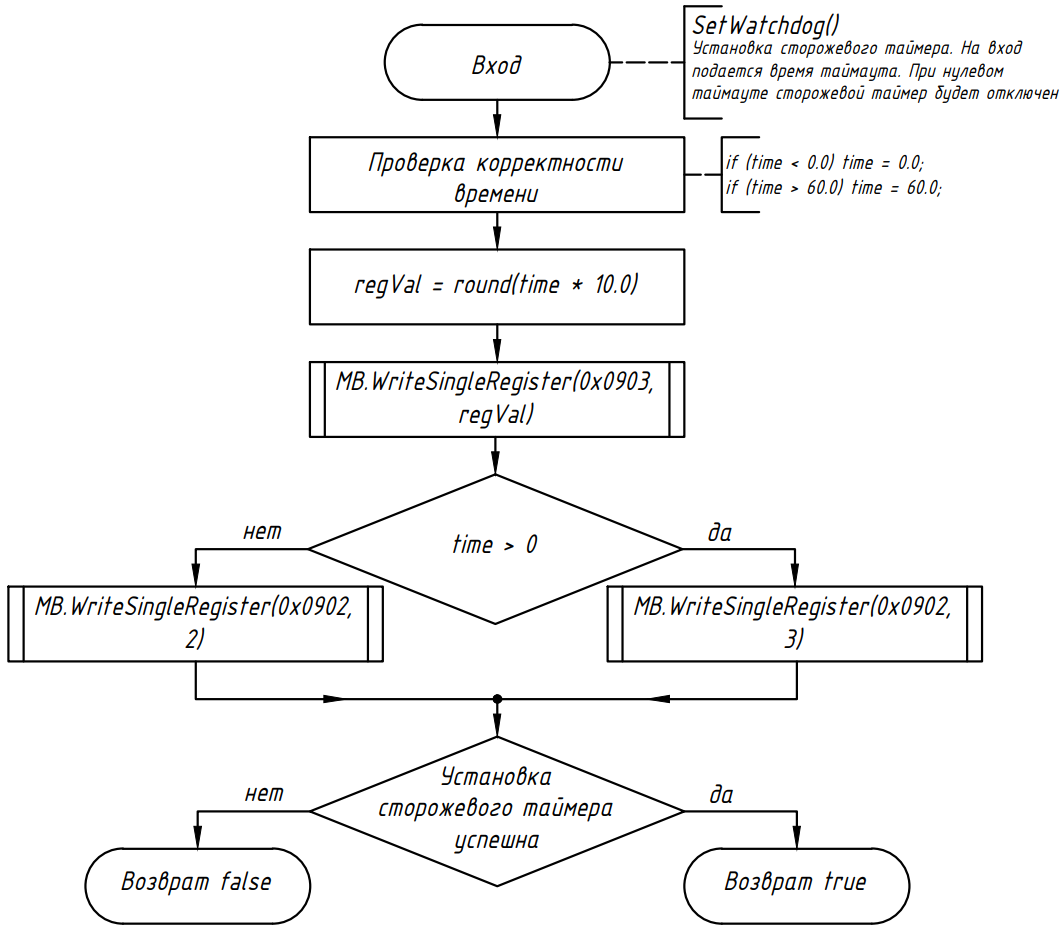


Рисунок 2.35 – Блок-схема алгоритма частоты вращения двигателя

## Основная программа управления

Основная программа предназначена для взаимодействия с пользователем и позволяет задавать параметры запуска двигателя и выводить информацию о состоянии двигателя во время его работы. Интерфейс программы представляет собой консольное приложение, в котором параметры работы задаются в виде аргументов командной строки, а измеренные значения выводятся в окно консоли. Такое решение позволяет работать с программой как с отдельным приложением вручную, так и подключить ее к существующей АСУ ТП.

С помощью аргументов командной строки можно задавать дополнительные параметры при запуске программы, что повышает универсальность программы. Для доступа к введенным пользователем аргументам используется версия функции main() с параметрами:

int main(int argc, char \*argv[])

Первый аргумент функции main() - *argc*, содержит количество аргументов, переданных в программу. Второй аргумент - *argv* массив указателей на символьные строки, каждая из которых соответствует одному параметру командной строки. Нулевой аргумент *argv[0]* всегда содержит название запускаемой программы вместе полным путем расположения этой программы.

В системе Linux существует два вида аргументов: короткие и длинные. Короткие аргументы начинаются с одного дефиса и имеют длину в один символ (-h), их просто и быстро набирать в командной строке. Длинные аргументы начинаются с двух дефисов и могут иметь длинное имя (--help), которое целесообразно использовать в скриптах (чтобы потом можно было вспомнить, что и как происходит). Кроме этого, аргумент может передаваться вместе со значением (--Frequency 50). В разрабатываемой программе будем использовать только длинные аргументы. При запуске программы с аргументом --help выведется справочное сообщение со списком доступных аргументов, которое выглядит следующим образом:

VFDMotorControllWindows --help

Help.

This program can control Delta VFD-B.

Input commandline arguments:

-h | --help Display this help message

--port <COMx> Specify serial port (COM3 default) (--port COM3)

--file <text\_file> Read a file with frequency and time parameters table. (--file coords.txt)

And run motor according to the table.

Text file should contain table with times and frequencies and should look like this:

Time Frequency

0 0

10 30

20 20

40 -10

50 0

(The first column specifies the time to be set.

The second column specifies the frequency that will be reached for the time,

specified in the first column.)

--get <parameter> Read one of the following motor parameters: (--get MotorSpeed)

<FrequencyCommand>

<OutFrequency>

<OutCurrent>

<DCVoltage>

<OutVoltage>

<PowerFactor>

<OutTorque>

<MotorSpeed>

<OutPower>

<VFDTemperature>

<0x(parameter address)> (--get 0x0109)

--set <parameter> <value> Set one of the folloving motor parameters: (--set Frequency 45.5)

<Frequency>

<AccelerationTime>

<DecelerationTime>

<0x(parameter address)> (--set 0x2001 0x1388)

--run <n|f|r|c> Run motor with direction set (no change, forvard, reverse, change) (--run r)

--stop Stop motor

Функция main() реализует основной алгоритм программы управления. При запуске аргументы командной строки передаются в функцию HandleCLIArguments() для обработки. Данная функция сравнивает аргументы командной строки от *argv[0]* до *argv[argc-1]* с допустимыми аргументами. Если программа запущена без аргументов, то выводится сообщение об их необходимости, справочное сообщение со списком доступных аргументов (как будто программа запущена с аргументом --help) и программа завершается. При совпадении аргумента устанавливается соответствующий флаг в структуре CMD. Значения аргументов при их наличии записываются в соответствующие структуры и переменные. Далее поочередно выполняются разные блоки программы в зависимости от указанных пользователем аргументов (рисунок 2.36, 2.37).

При обнаружении аргумента --help будет вызвана функция PrintHelp(), печатающая справочное сообщение. Далее создается экземпляр класса VFD с названием motor, в котором задается адрес Modbus равен 1, указанное пользователем имя порта и параметры последовательного подключения (9600, 8, E, 1).

При обнаружении аргумента --get будут считаны указанные параметры функцией GetMotorParameters(), напечатаны их названия функцией PrintParametersHeader() и значения функцией PrintParameters(). С помощью аргумента --get можно считать следующие параметры: статус VFD-B; ведущая частота, выходная фактическая частота; выходной ток; напряжение на шине DC; выходное напряжение; коэффициент мощности; вычисленный момент; скорость двигателя; выходная мощность; температура радиатора. Также этот аргумент позволяет задать в качестве значения номер параметра в шестнадцатеричном формате и считать любой из доступных параметров ПЧ.

При обнаружении аргумента --set функцией SetMotorParameters() можно установить параметры частоты, времени разгона и замедления. Также этот аргумент позволяет указать номер и значение параметра в шестнадцатеричном формате и записать любой из параметров ПЧ, для которого разрешена запись.

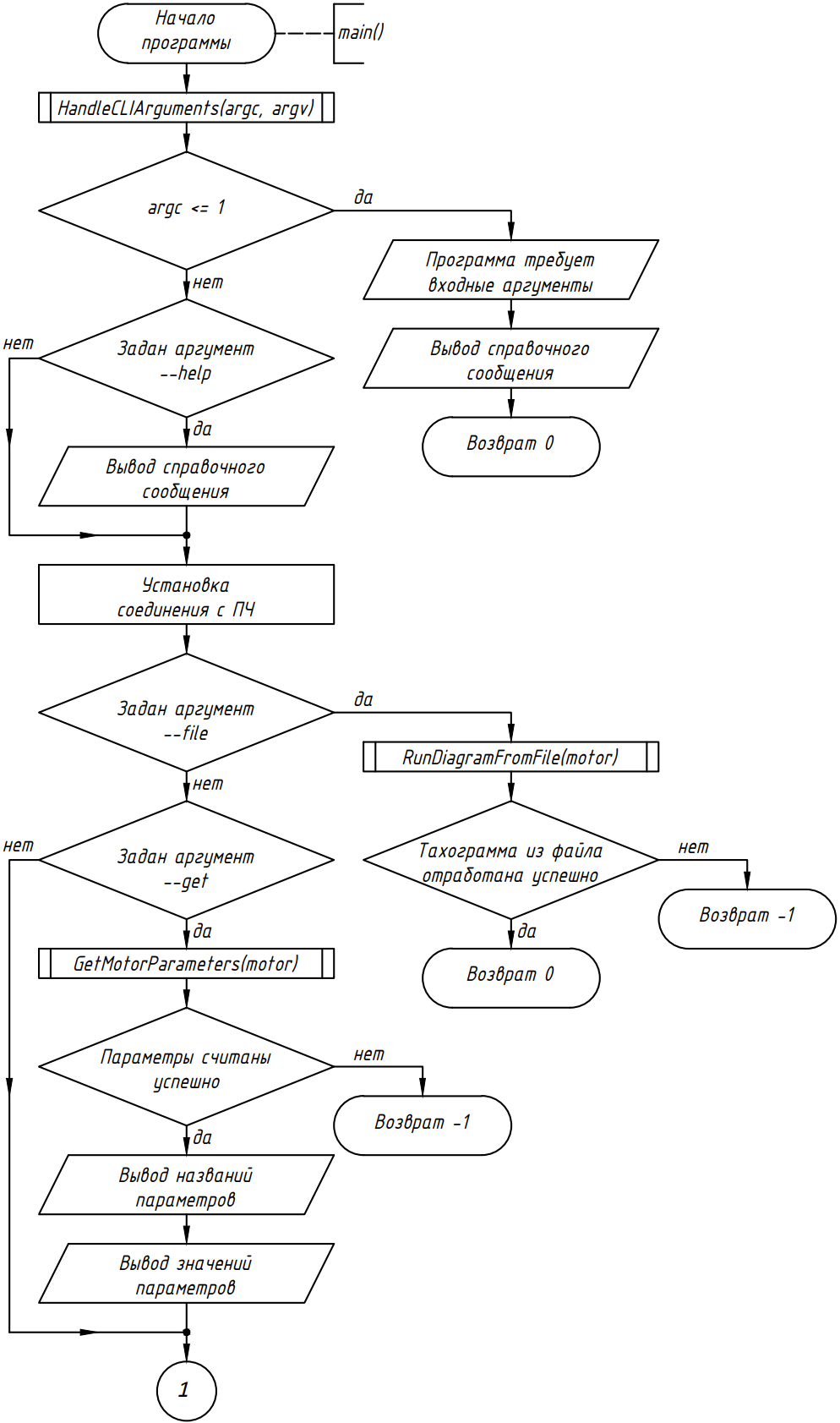


Рисунок 2.36 – Блок-схема алгоритма работы основной программы

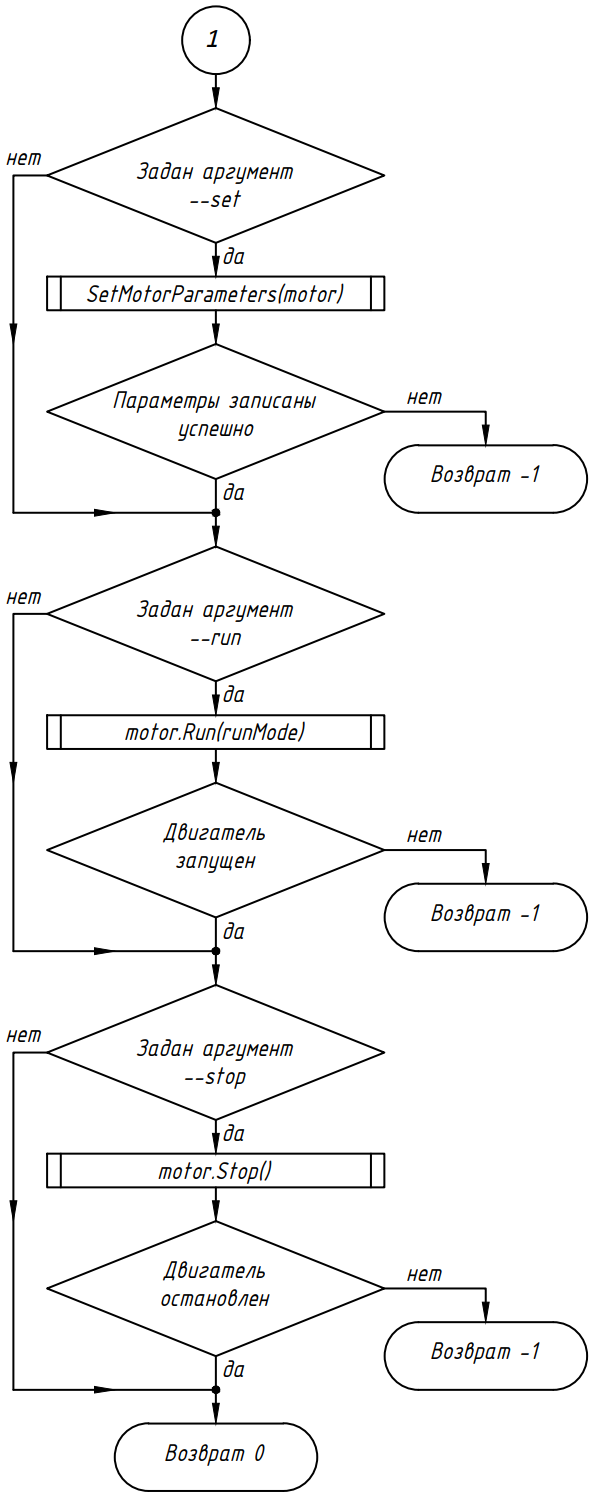


Рисунок 2.37 – Блок-схема алгоритма работы основной программы (продолжение)

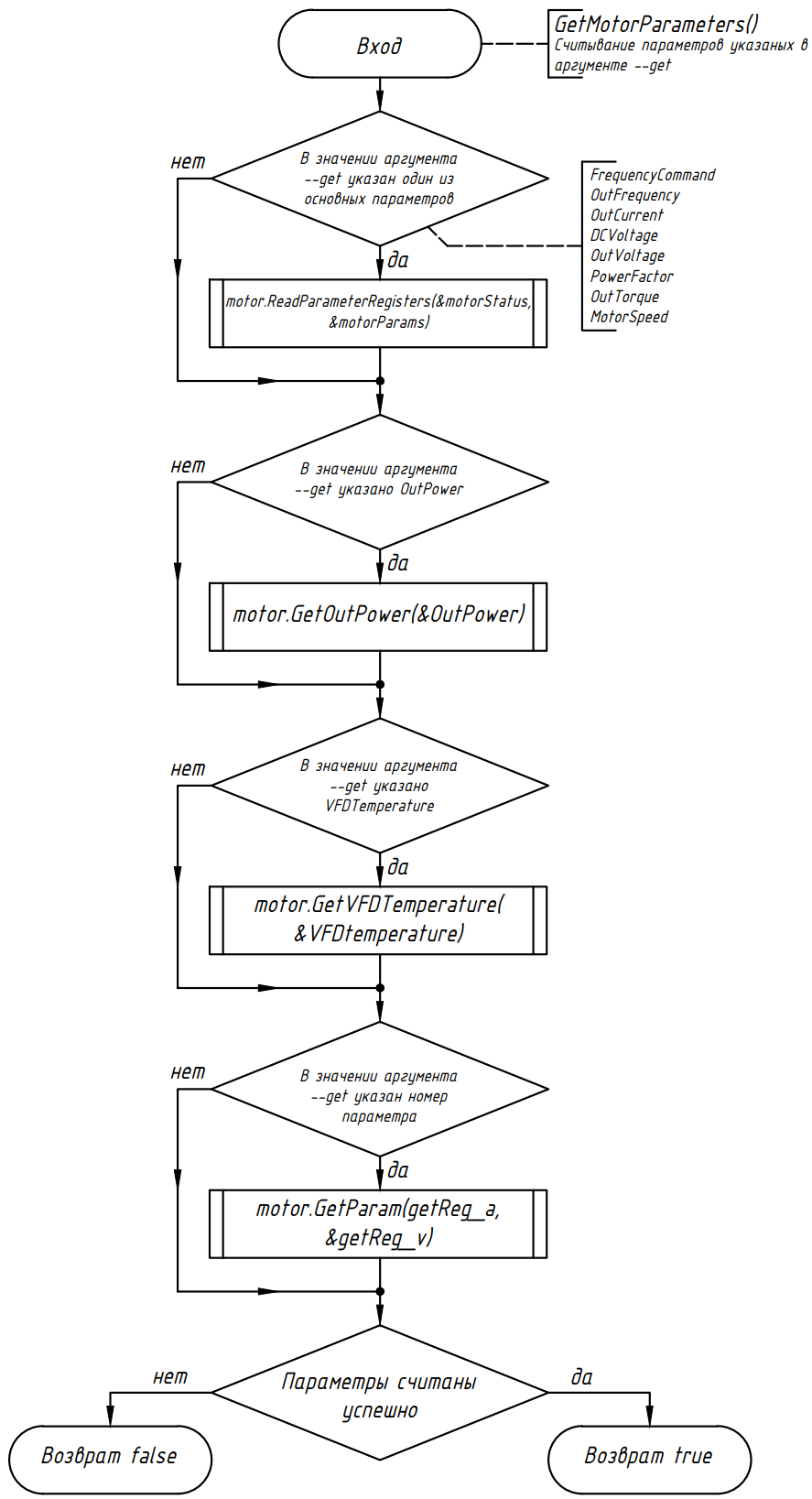


Рисунок 2.38 – Блок-схема алгоритма чтения параметров двигателя

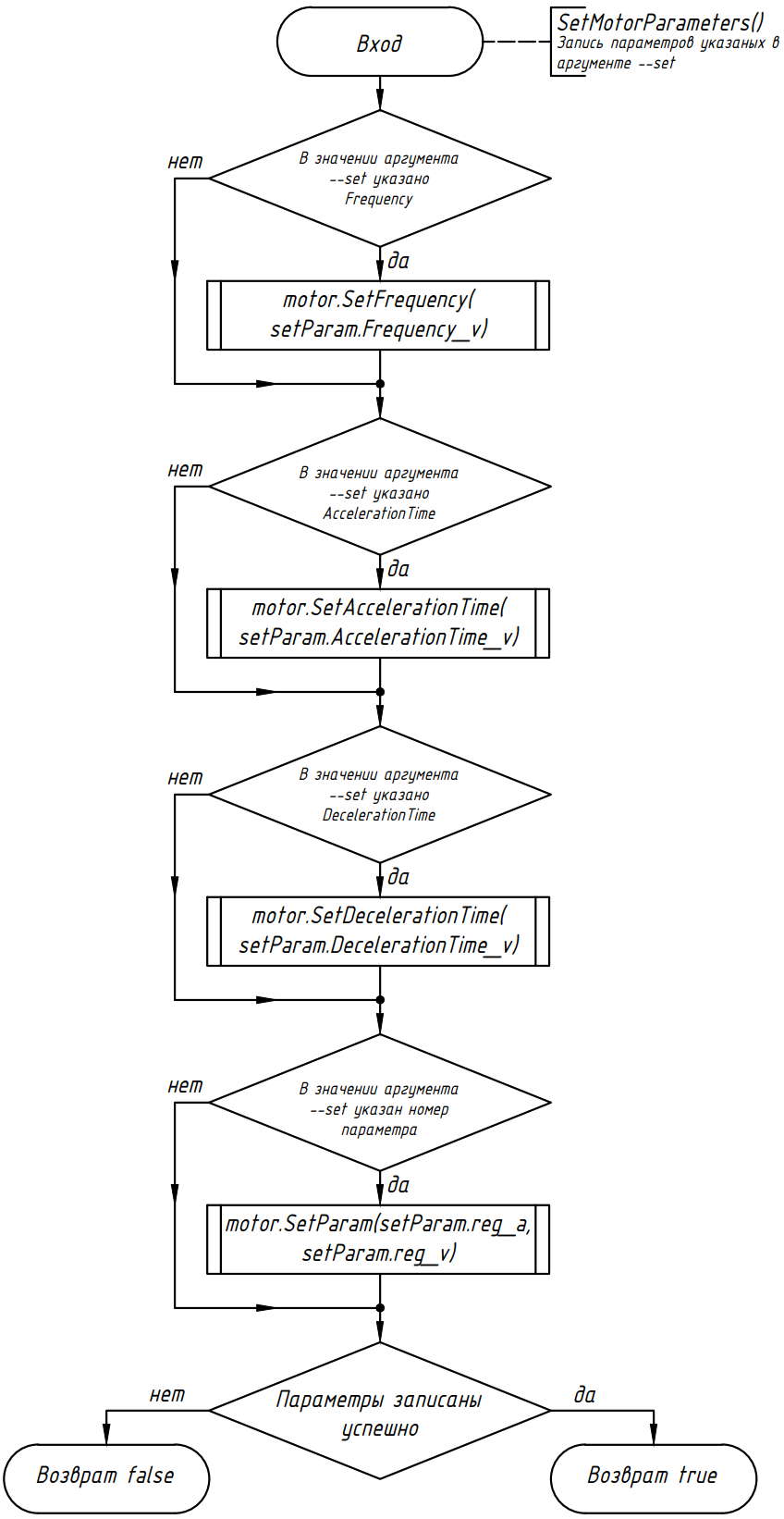


Рисунок 2.39 – Блок-схема алгоритма записи параметров двигателя

При обнаружении аргументов --run и --stop двигатель запускается и останавливается соответственно.

Для реализации вращения двигателя по сложным тахограммам (рисунок 1.1) программа поддерживает работу двигателя согласно координатам, указанным в текстовом файле. Файл должен содержать в каждой строке пары значений в виде времени и частоты, которая должна быть установлена к этому времени. Для корректной работы значения времени должны быть расположены в порядке возрастания и временной интервал между заданиями новой частоты должен быть больше 250 мс. При слишком частом изменении частоты задания некоторые точки тахограммы могут быть пропущены.

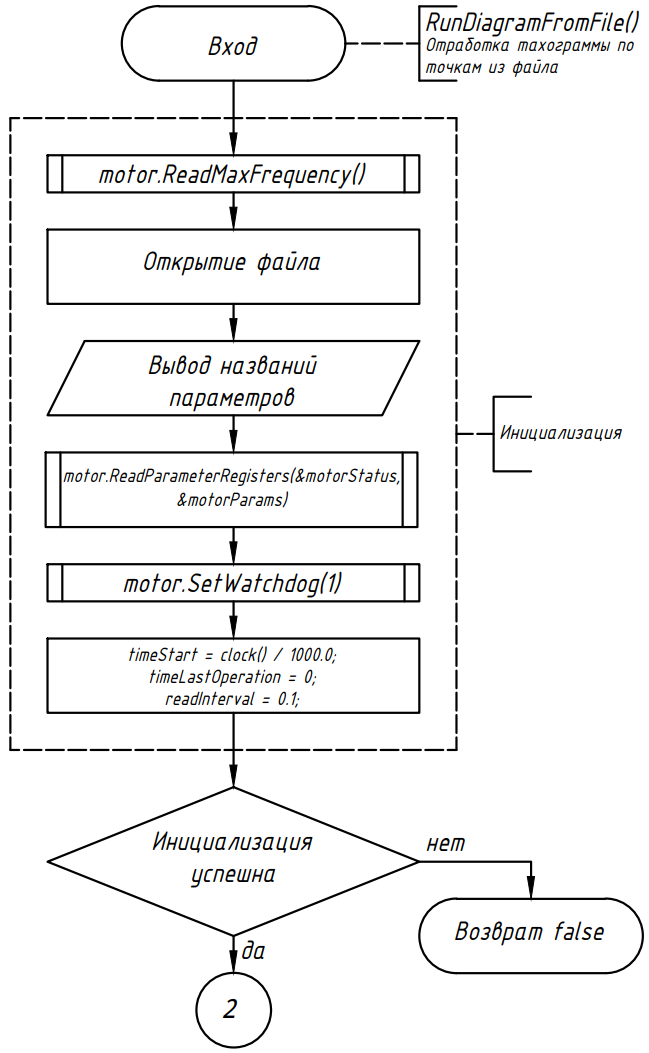


Рисунок 2.40 – Блок-схема алгоритма отработки тахограммы из файла (инициализация)

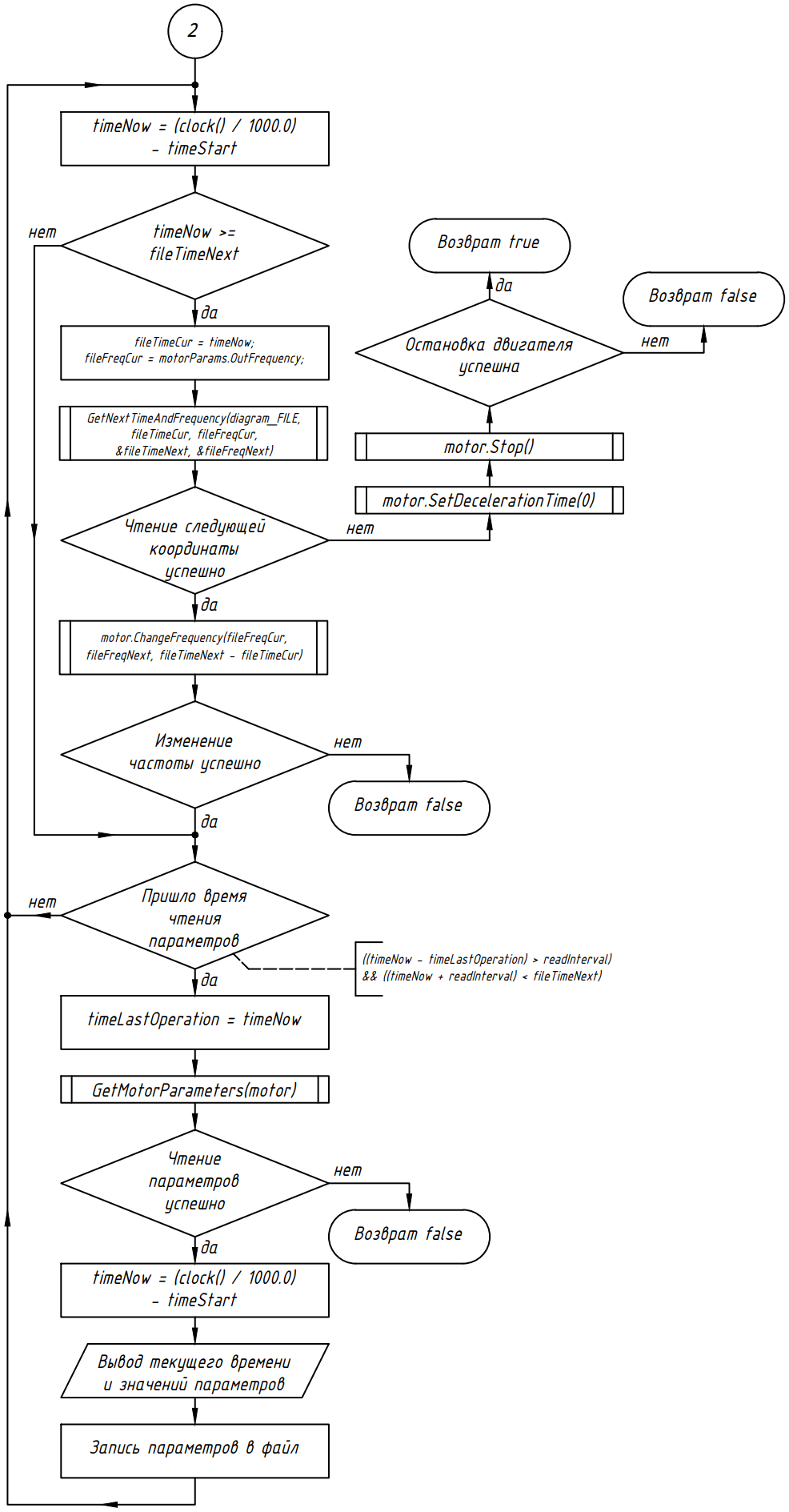


Рисунок 2.41 – Блок-схема алгоритма отработки тахограммы из файла (продолжение)

При обнаружении аргумента --file с именем файла в качестве значения программа считает указанный файл и начнет отработку тахограммы по точкам из файла с помощью функции RunDiagramFromFile() и с выводом времени работы на экран консоли. Дополнительно можно вывести необходимые параметры двигателя указав аргумент --get с требуемыми значениями. Измерение параметров двигателя и вывод их на экран происходит с интервалом 100мс. При большом количестве параметров для вывода данный интервал может увеличиться, а во время изменения частоты задания вывод параметров может быть пропущен.

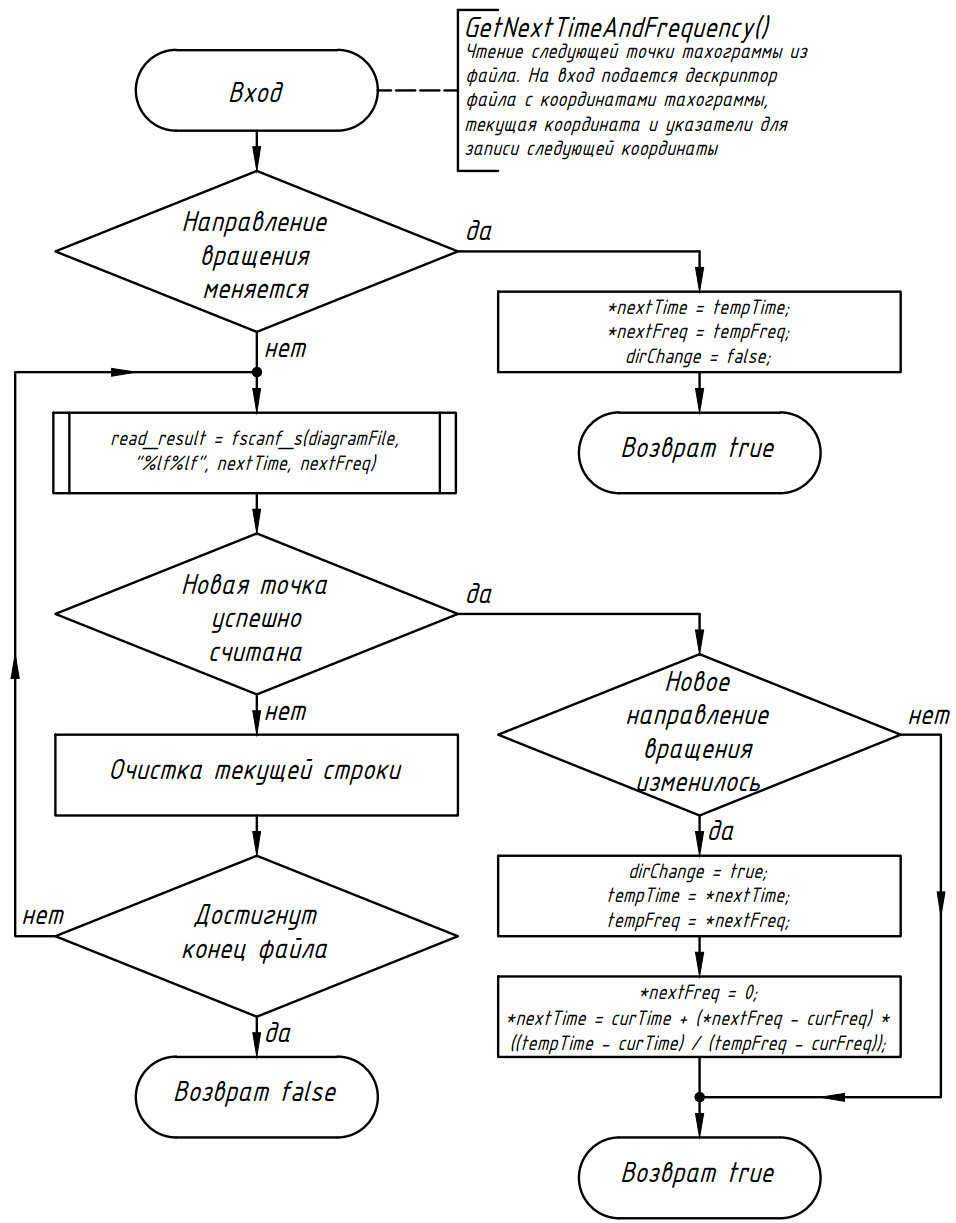


Рисунок 2.42 – Блок-схема алгоритма чтения точки тахограммы из файла

Исходный код для работы с тахограммой вынесен в отдельный файл. Функция RunDiagramFromFile() сначала считывает значение максимальной частоты из ПЧ, которая нужная для расчетов времени разгона и торможения. Далее открывается файл с точками тахограммы, считывается текущий статус ПЧ. Установка сторожевого таймера на одну секунду позволяет реализовать остановку двигателя при потере соединения во время отработки тахограммы. Также инициализируются необходимые таймеры (рисунок 2.40). После инициализации циклически проверяется значение текущего времени, и в нужные моменты считываются новые точки тахограммы из файла и записываются в ПЧ. Также в нужные моменты считываются параметры двигателя, которые выводятся на экран и записываются в файл (рисунок 2.41). При отсутствии новых точек в файле отработка тахограммы заканчивается.

Функция GetNextTimeAndFrequency() читает следующую точку из файла и проверяет ее корректность. При ошибке считывания или некорректности точки будут считываться следующие точки до тех пор, пока не будет считана корректная точка или будет достигнут конец файла. При достижении конца файла функция вернет false, что свидетельствует об отсутствии новых точек (рисунок 2.42).

## Тестирование работы программы

Для вращения двигателя по заданной тахограмме (рисунок 1.1) создадим текстовый файл с точками тахограммы. В данной тахограмме присутствует участок с экспоненциальным изменением частоты вращения. Аппроксимируем данный участок кусочно-линейной функцией, координаты точек которой рассчитываются по формуле:

где – время начала и конца изменения частоты;

– начальная и конечная частоты.

Координаты точек тахограммы выглядят следующим образом:

Time Frequency

0 0

0.5 11.06

1 19.67

1.5 26.38

2 31.61

2.5 35.67

3 38.84

3.5 41.31

4 43.23

4.5 44.73

5 45.9

5.5 46.8

6 47.51

6.5 48.06

7 48.49

7.5 48.82

8 49.08

8.5 49.29

9 49.44

9.5 49.57

10 49.66

12 10

14 10

16 -20

18 -20

19 -40

21 -40

24 0

Сохраним координаты в файл *coords.txt*. При запуске программы командой

VFDMotorControlWindows.exe --port COM9 --file coords.txt --get OutFrequency --get OutCurrent --get OutVoltage --get MotorSpeed > outRes.txt

преобразователь частоты, подключенный к порту COM9, будет вращать двигатель согласно тахограмме из файла *coords.txt*. Во время работы программы будут измерены параметры текущей частоты, тока, напряжения и текущей скорости вращения двигателя. Для удобства дальнейшей обработки вывод программы будет перенаправлен в файл *outRes.txt*, который будет иметь следующий вид:

Time OutFrequency OutCurrent OutVoltage MotorSpeed

0.15 1.58 0.1 6.4 29

0.27 4.02 0.6 17.2 75

.......................................................

По данным из файла *outRes.txt* построим графики измеренных параметров (рисунок 2.43-2.46).

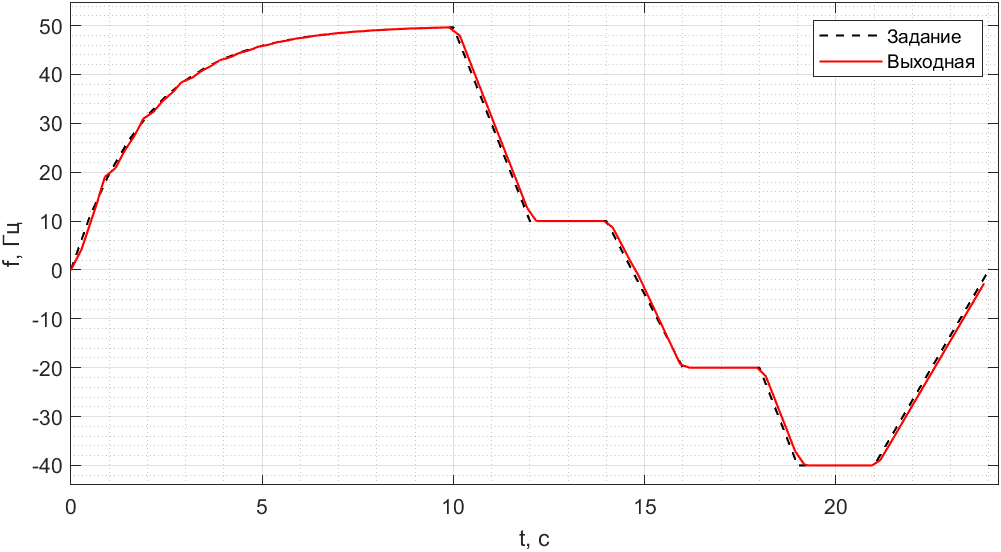


Рисунок 2.43 – Тахограмма вращения двигателя

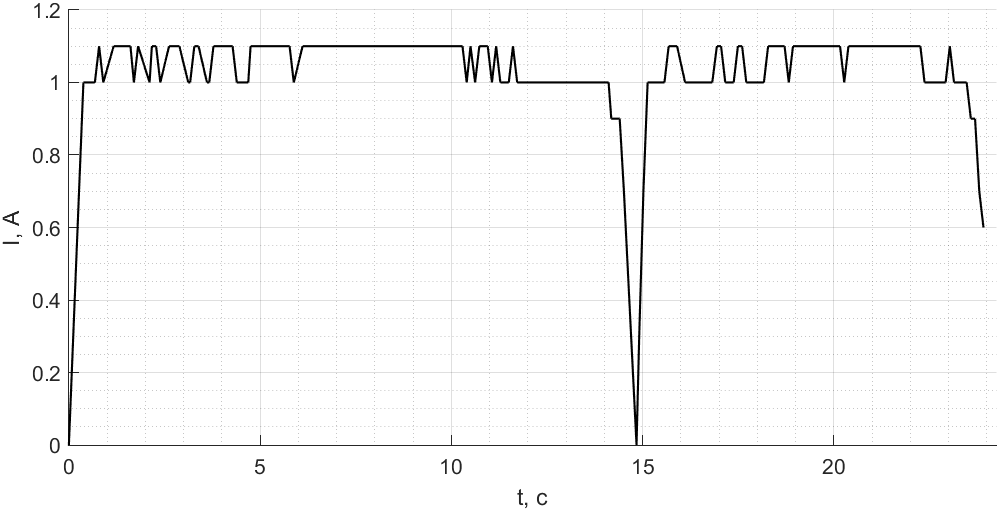


Рисунок . – Ток двигателя

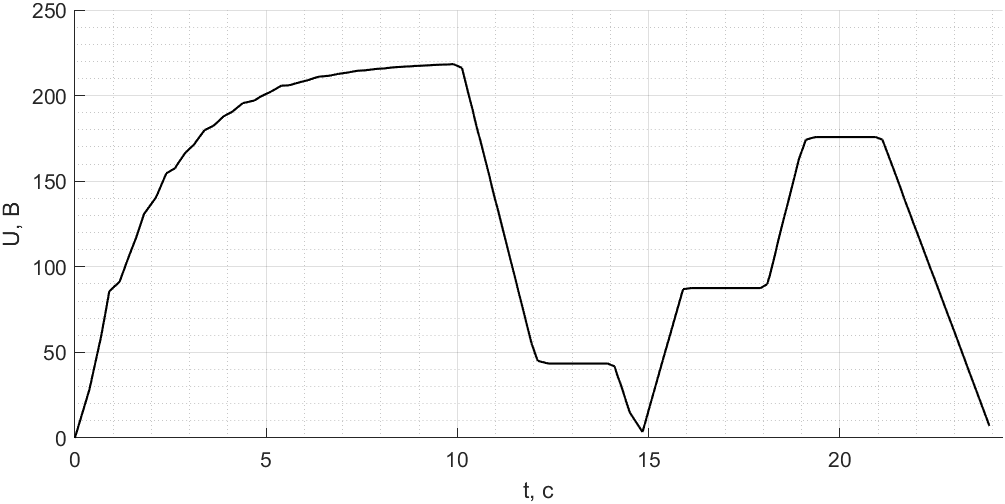


Рисунок . – Напряжение двигателя

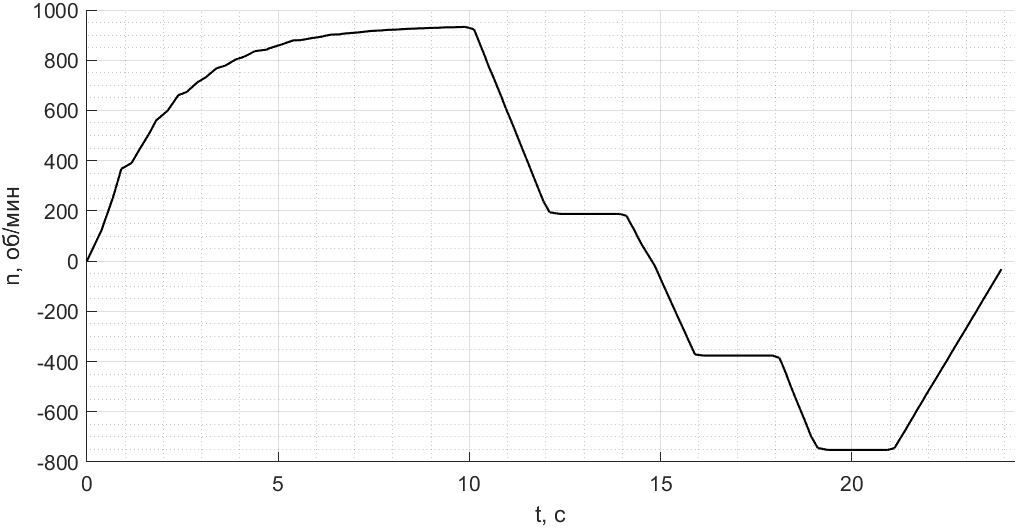


Рисунок . – Скорость вращения двигателя

# Управление ПЧ с помощью промышленного контроллера ICP DAS I-7188EX

Здесь а пришло там как расстраивают сам контроллер его схему подключения там как контролировать хотя не даже не контролировать а какие изменения нужны в проекты скорее всего изменения коснутся комфорта. или далее верно как скомпилировать какими то рамками возможно придется делать мейк файл есть как запускать с помощью программы меня озвучили.

## Функции времени на I-7188EX

## Разработка драйвера UART интерфейса

## Перенос основной программы

## Разработка интерфейса взаимодействия с оператором

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Scott Chacon ProGit. Everything You need to know about Git. [Электронный ресурс] / Scott Chacon, Ben Straub. – Электрон. текстовые данные. – Apress; 2nd ed. edition, 2022. – 538 c. – 978-1484200773. – Режим доступа: <https://git-scm.com/book/ru/v2> – Загл. с экрана.
2. Overview – Doxygen Manual // [Электронный ресурс]. URL: <https://www.doxygen.nl/manual/index.html> (Дата обращения: 22.02.2023).
3. TIA/EIA-485-A-ENGL 1998 Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems // URL: <https://www.mikrocontroller.net/attachment/428561/eia485.pdf> (Дата обращения: 01.03.2023), 1998. – 22 с.
4. Win32 apps: Ресурсы для обмена данными – Microsoft Learn // [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/windows/win32/devio/communications-resources> (Дата обращения: 23.02.2023).
5. MODBUS Application Protocol Specification v1.1b3 – The Modbus organization // [Электронный ресурс]. URL: <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf> (Дата обращения: 24.02.2023).
6. Specification and Implementation Guide for MODBUS over Serial Line – The Modbus organization // [Электронный ресурс]. URL: <https://modbus.org/docs/Modbus_over_serial_line_V1_02.pdf> (Дата обращения: 24.02.2023).
7. Руководство по эксплуатации VFD-B – Delta Electronics Inc. // [Электронный ресурс]. URL: <http://www.deltronics.ru/images/manual/VFD-B_manual_rus.pdf> (Дата обращения: 01.03.2023).

Страна будет добавить к литературу по самому контроллеру там думаю мы 7 книг

<https://www.mikrocontroller.net/attachment/428561/eia485.pdf>

<https://filecenter.deltaww.com/Products/download/06/060101/Manual/DELTA_IA-MDS_VFD-B-P_UM_EN_20090406.pdf>

сайт assert <https://en.cppreference.com/w/cpp/error/assert>

TIA/EIA-485-A-ENGL 1998 Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems // URL: <https://www.mikrocontroller.net/attachment/428561/eia485.pdf> (Дата обращения: 01.03.2023), 1998. – 22 с.

ГОСТ 7746-2015 Трансформаторы тока. Общие технические условия - М.: Стандартинформ, 2016. – 43 с.

Шаньгин Е.С., Управление роботами и робототехническими системами : учеб. пособие для студентов вузов, обучающихся по направлению 652000 "Мехатроника и робототехника" / Е.С. Шаньгин ; Федер. агентство по образованию, Гос. образоват. учреждение высш. проф. образования, Уфим. гос. авиац. техн. ун-т. - Уфа : Уфим. гос. авиац. техн. ун-т, 2005 (Уфа : РИК УГАТУ). - 188 с.

Scott Chacon ProGit. Everything You need to know about Git. [Электронный ресурс] / Scott Chacon, Ben Straub. – Электрон. текстовые данные. – Apress; 2nd ed. edition, 2022. – 538 c. – 978-1484200773. – Режим доступа: <https://git-scm.com/book/ru/v2> – Загл. с экрана.