МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра электронных приборов
(полное название кафедры)

Утверждаю

Зав. кафедрой Семенов А.М.

(подпись, инициалы, фамилия)

«___» _____ 202**5** г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Жеребцов Артем Алексеевич
(фамилия, имя, отчество студента – автора работы)

Разработка программной части тестового стенда для аналого-цифрового
(тема работы)

модуля лазерного детектора метана

Факультет радиотехники и электроники
(полное название факультета)

Направление подготовки 11.03.04 — электроника и наноэлектроника
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ	**Руководитель от организации**	**Автор выпускной квалификационной работы**
Чипурнов С.А.	Ивойлов А.Ю.	Жеребцов А.А.
(фамилия, имя, отчество)	(фамилия, имя, отчество)	(фамилия, имя, отчество)
к.т.н., доцент	Рук. отдела программирования	РЭФ, РЭ3-11
(ученая степень, ученое звание)	(ученая степень, ученое звание)	(факультет, группа)
(подпись, дата)	(подпись, дата)	(подпись, дата)

Нормоконтроль:

Чипурнов С.А.	к.т.н., доцент	
(фамилия, имя, отчество)	ученая степень, ученое звание)	(подпись, дата, инициалы, фамилия)

Новосибирск 202**5**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра электронных приборов
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Семенов А.М.
(фамилия, имя, отчество)

(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту Жеребцову Артему Алексеевичу
(фамилия, имя, отчество)

Направление подготовки 11.03.04 — электроника и наноэлектроника
(код и наименование направления подготовки бакалавра)

Факультет радиотехники и электроники
(полное название факультета)

Тема Разработка программной части тестового стенда для аналого-цифрового модуля лазерного детектора метана
(полное название темы выпускной квалификационной работы бакалавра)

Исходные данные (или цель работы) Разработать программное обеспечение для персонального компьютера на языке C++. Написать прошивку на языке C для микроконтроллера STM32 блока измерения и диагностики тестового стенда для проверки работоспособности критических узлов тестируемой платы лазерного детектора метана. Провести тестирование на испытательном стенде.

Структурные части работы Введение
1 Литературный обзор
1.1 Тестовые стенды
1.2 Принципы контактирования с тестируемыми платами
1.3 Программное обеспечение тестовых стендов
1.4 Обзор метода «ложе гвоздей» с точки зрения программного обеспечения
2 Практическая часть

2.1 Лазерный газоанализатор метана

2.2 Методика разработки

2.3 Функционал тестового стенда

2.4 Конфигурация выводов микроконтроллера

2.5 Разработка прошивки микроконтроллера

2.6 Разработка приложения для Windows

2.7 Конструкция тестового стенда

Заключение

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Руководитель от организации**	**Студент**
Чипурнов С.А.	Ивойлов А.Ю.	Жеребцов А.А.
(фамилия, имя, отчество)	(фамилия, имя, отчество)	(фамилия, имя, отчество)
к.т.н., доцент	Рук. отдела программирования	РЭФ, РЭ3-11
(ученая степень, ученое звание)	(ученая степень, ученое звание)	(факультет, группа)
(подпись, дата)	(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № _____ от «____» _____ 202**5** г.

изменена приказом по НГТУ № _____ от «____» _____ 202**5** г.

ВКР сдана в ГЭК № _____, тема сверена с данными приказа

_____
(подпись секретаря государственной экзаменационной комиссии по защите ВКР, дата)

_____
(фамилия, имя, отчество секретаря государственной экзаменационной комиссии по защите ВКР)

РЕФЕРАТ

Пояснительная записка 99 с., включая приложение, 16 рис., 9 табл., 26 ист., 7 прил.

СИСТЕМА ДИАГНОСТИКИ, СИСТЕМА СБОРА ДАННЫХ, СИСТЕМА УПРАВЛЕНИЯ, СИСТЕМА ПЕРЕДАЧИ ДАННЫХ, СИСТЕМА ОБРАБОТКИ

Объектом разработки является программное обеспечение (ПО) на персональный компьютер (ПК) и прошивка для микроконтроллера STM32, являющийся частью платы тестового стенда. Стенд производит проверку ключевых модулей аналого-цифрового модуля лазерного детектора метана.

Цель работы — разработка программы управления для тестового стенда на языке C. Разработка протокола для взаимодействия микроконтроллера стенда с компьютером и исследуемой платой. Разработка ПО на ПК на языке C++, которое будет производить автоматизированную проверку, выводить статус выполнения, строить график.

В процессе работы был исследован алгоритм работы аналого-цифрового модуля лазерного детектора метана. На этой основе был сформирован порядок тестирования ключевых модулей этого устройства, для точного выявления дефектов и брака.

В результате работы был описан алгоритм работы платы тестового стенда. Разработаны модули прошивки микроконтроллера (МК), для работы с платами цифровых входов, цифровых выходов, аналоговых входов. Разработан графический интерфейс для ПК. Проведено тестирование модулей на испытательном стенде, составлена документация.

Содержание

Введение

При разработке электроники важным этапом является тестирование устройства. На данном этапе инженеры-схемотехники, разработавшие электрическую схему, рассчитавшие необходимые компоненты, а также расстановку этих компонентов на печатной плате и трассировку печатной платы, осуществляют проверку критически важных узлов и модулей. Инженеры-программисты, в свою очередь, проверяют работоспособность прошивки, анализируют работу всего прибора, выявляя баги и ошибки на стадии тестирования. Всё это делается для того, чтобы отсеять бракованные платы, не прошедшие установленные проверки. Проверенные приборы затем собираются в корпус и высылаются заказчику.

Часто инженерам приходится тестировать готовые платы в большом объёме перед сдачей заказчику. Время, отведённое на проверку, зависит от сложности разработки платы, а также от специфики используемых компонентов и технологий. Задачу может усложнить и тот факт, что производство данного устройства планируется организовать серийно, что делает оптимизацию этапа проверки особенно актуальной. В таких случаях автоматизация тестирования становится необходимой для снижения затрат времени, затрат мощности и повышения точности выявления дефектов.

Одним из решений данной проблемы является разработка автоматизированного стенда, который проводит проверку плат в автоматическом режиме с минимальным участием человека. Такой стенд способен выполнять последовательность тестовых процедур, фиксировать результаты измерений и сразу же сигнализировать о выявленных ошибках. Это позволяет не только сократить время тестирования, но и значительно уменьшить вероятность человеческой ошибки. Такой подход способствует налаживанию серийного и даже массового производства приборов, что, в свою очередь, повышает прибыль компании за счёт оптимизации производственных процессов и повышения качества выпускаемой продукции.

В данном проекте необходимо разработать прошивку на языке программирования Си для микроконтроллера STM32F411CEU6, который является

центральным компонентом схемы. Микроконтроллер управляет работой тестового стенда, осуществляет сбор, передачу и обработку данных с тестируемой платы аналого-цифрового модуля лазерного детектора метана. Помимо реализации основных функций, прошивка должна обеспечивать:

- надёжное управление периферийными устройствами стенда;
- своевременную обработку сигналов и данных;
- взаимодействие с внешними устройствами через стандартизированные интерфейсы;
- ведение журнала ошибок и событий для дальнейшего анализа и устранения неисправностей.

Необходимо написать ПО с графическим интерфейсом на языке программирования C++ для управления работы прошивки данного МК, для отправки, получения и анализа данных, на основе которых будет в удобном пользователю виде представлен статус выполнения и результат выполнения команд.

Оптимальное распределение задач между аппаратной и программной частями системы позволяет добиться высокой точности тестирования и ускорить процесс проверки, что особенно важно при серийном производстве. Внедрение автоматизированного тестирования способствует сокращению себестоимости конечного продукта и повышению его качества, что в конечном итоге удовлетворяет потребности заказчика и усиливает конкурентные преимущества компании.

Актуальность темы. Тестовый стенд является важным прибором для тестирования аналого-цифрового модуля лазерного детектора метана. Он призван оптимизировать процесс проверки печатных плат газоанализатора, оперативно выявляя бракованные изделия, с указанием модулей, не прошедших проверку. В отсутствие тестового стенда время, затраченное на выпуск приборов, будет кратно выше, что негативно скажется с финансовой точки зрения для предприятия-изготовителя, а значит актуально данной работы – высока.

Новизна темы. Существующие на рынке универсальные тестовые стенды не подходят к применению для плат газоанализатора в связи со спецификой проекта и

его сложностью. Необходима разработка узкоспециализированного тестового стенда, которая учтет все нюансы и тонкости платы детектора метана.

Постановка задачи. Разработать ПО для микроконтроллера STM32 тестового стенда для аналого-цифрового модуля лазерного детектора метана. Разработать ПО с графическим интерфейсом на ПК. Работа включает в себя написание модулей на языке программирования C и C++ для сбора, анализа и передачи состояний модулей исследуемой платы газоанализатора метана на персональный компьютер. Составление документации и отладку написанных модулей на испытательном стенде.

1 Литературный обзор

1.1 Тестовые стенды

Печатные платы (ПП) являются неотъемлемой частью практически всех электронных устройств, от бытовых приборов до высокотехнологичных систем. Эффективность их производства и надежность напрямую зависят от качества изготовления, что, в свою очередь, требует применения различных методов тестирования и проверки ПП на соответствие заявленным характеристикам. Для этого используется множество тестовых стендов, которые различаются по принципам работы, оснащению и сложности. В данном обзоре рассматриваются возможные варианты тестовых стендов для проверки ПП, а также обосновывается выбор одного из направлений работы.

Разработка автоматизированных систем проверки печатных плат зависит от конкретных решаемых задач. Критерии отбраковки могут варьироваться по степени строгости, но в целом их можно свести к трём основным направлениям: а) проверка соответствия монтажных соединений электрической схеме (функциональный контроль); б) анализ параметров монтажа на соответствие установленным нормам (параметрический контроль); в) оценка достаточности надёжности платы для эксплуатации (диагностический контроль).

1.2 Принципы контактирования с тестируемыми платами

Существует несколько методов электрического контактирования печатных плат: ручной, «ложе гвоздей», «летающие щупы» и «летающие матрицы».

1.2.1 Ручной метод

При ручном методе оператор вручную проверяет электрические цепи, используя щупы и измерительные приборы (мультиметр, осциллограф, анализатор

спектра и прочее). Однако из-за человеческого фактора около 25% дефектов остаются незамеченными, что делает этот способ наименее надёжным. Рисунок 1.1.

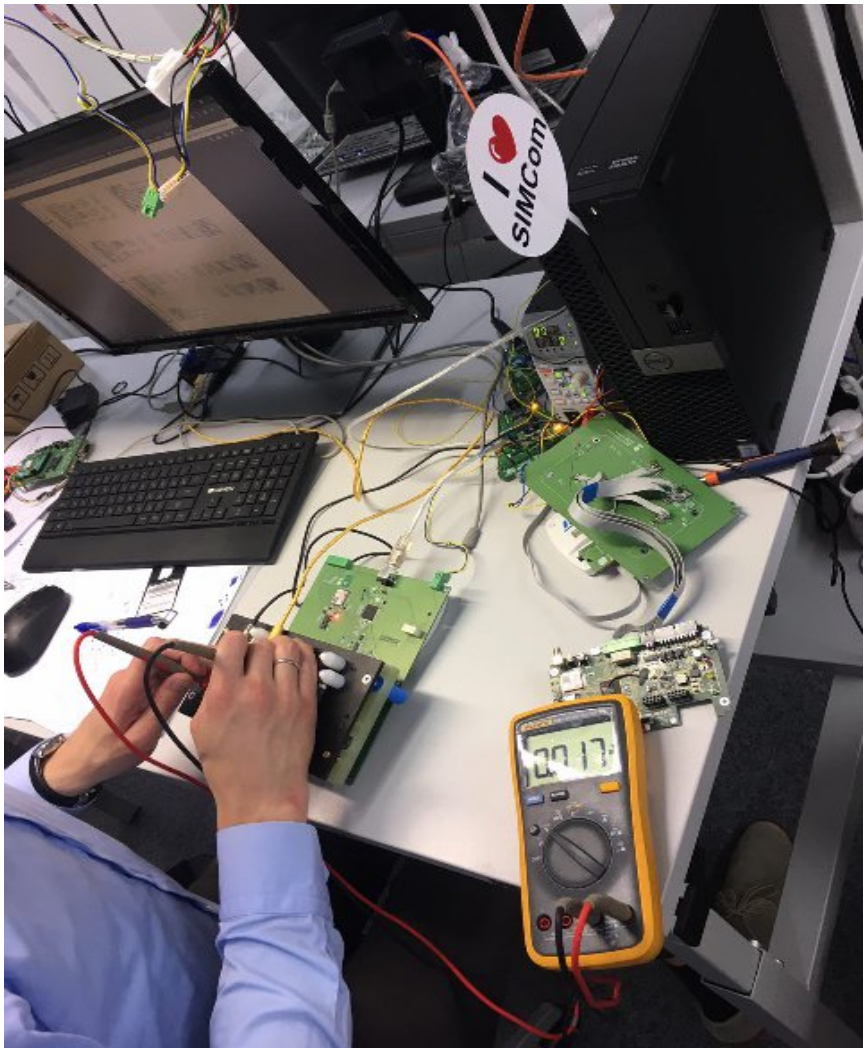

Рисунок 1.1 — Иллюстрация ручного метода [1]

1.2.2 Метод «ложе гвоздей»

Метод «ложе гвоздей» предполагает использование матрицы подпружиненных контактов, размещённых в узлах координатной сетки. Для обеспечения контакта с конкретной платой применяется перфорированная маска или специальный тестовый адаптер с индивидуально расположенными зондами. Все точки платы контактируются одновременно, а скорость проверки определяется быстродействием переключающих элементов. Этот метод обеспечивает высокую производительность, но требует значительных затрат на переналадку при смене типа платы. Рисунок 1.2.

Рисунок 1.2 — Иллюстрация метода «Ложе гвоздей» [1]

1.2.3 Метод «летающие щупы»

Метод «летающих щупов» использует тестовую установку с несколькими подвижными головками, каждая из которых оснащена зондом и может перемещаться по трём осям. Контактирование выполняется последовательно по заданной программе, позволяя подавать сигналы или измерять параметры цепей. Такой подход обеспечивает быструю переналадку благодаря использованию CAD/CAM-данных, но обладает невысокой производительностью из-за последовательного характера тестирования. Рисунок 1.3.

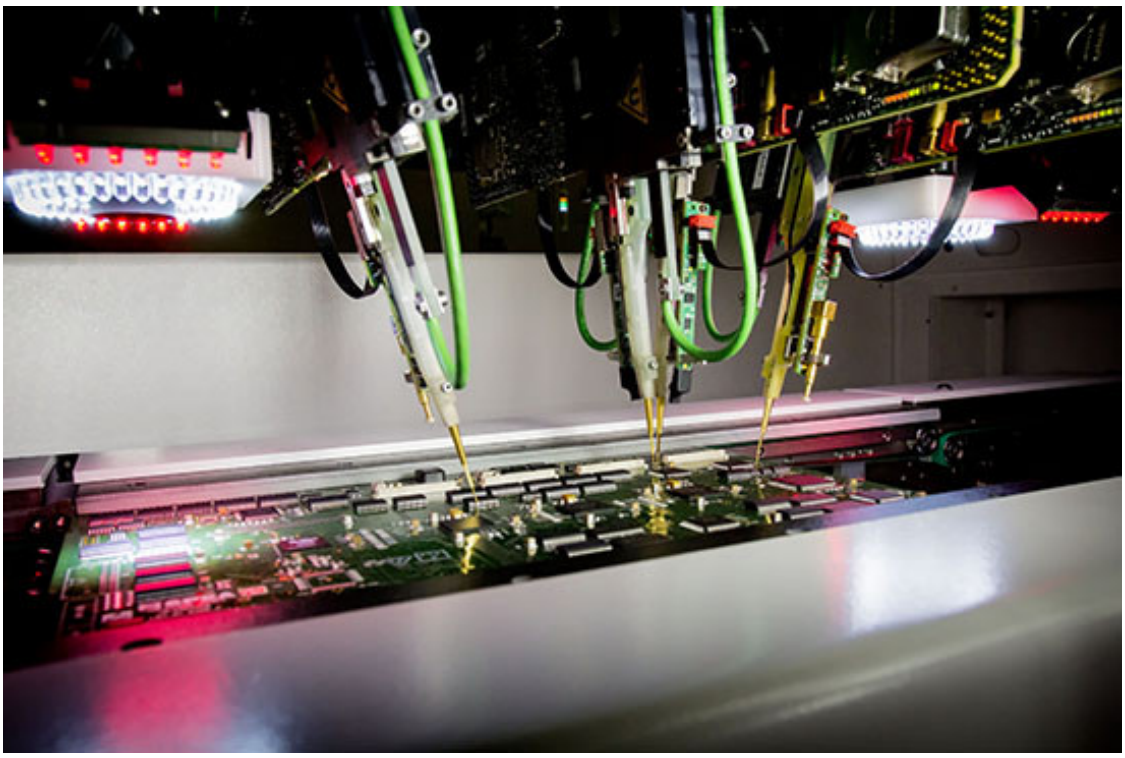

Рисунок 1.3 — Иллюстрация метода «Летающие щупы» [1]

1.2.4 Метод «летающих матриц»

Метод «летающих матриц» подразумевает, что на каждой каретке располагается матрица щупов, причём каждый из них имеет независимый привод по оси. Матрицы перемещаются с высокой скоростью на небольшие расстояния, а наиболее близкий зонд выполняет измерения. Благодаря этому производительность возрастает на порядок по сравнению с методом «летающих щупов».

Разновидностью метода «летающих матриц» является метод «Скорпион», при котором контактные щупы сосредоточены в нескольких модулях, расположенных с двух сторон платы. Эти модули оснащены подвижными щупами с большим рабочим ходом, которые последовательно тестируют зоны платы. Данная технология позволяет проверять трёхмерные печатные платы и электронные модули с высокими компонентами, что делает её незаменимой в таких задачах. Рисунок 1.4.

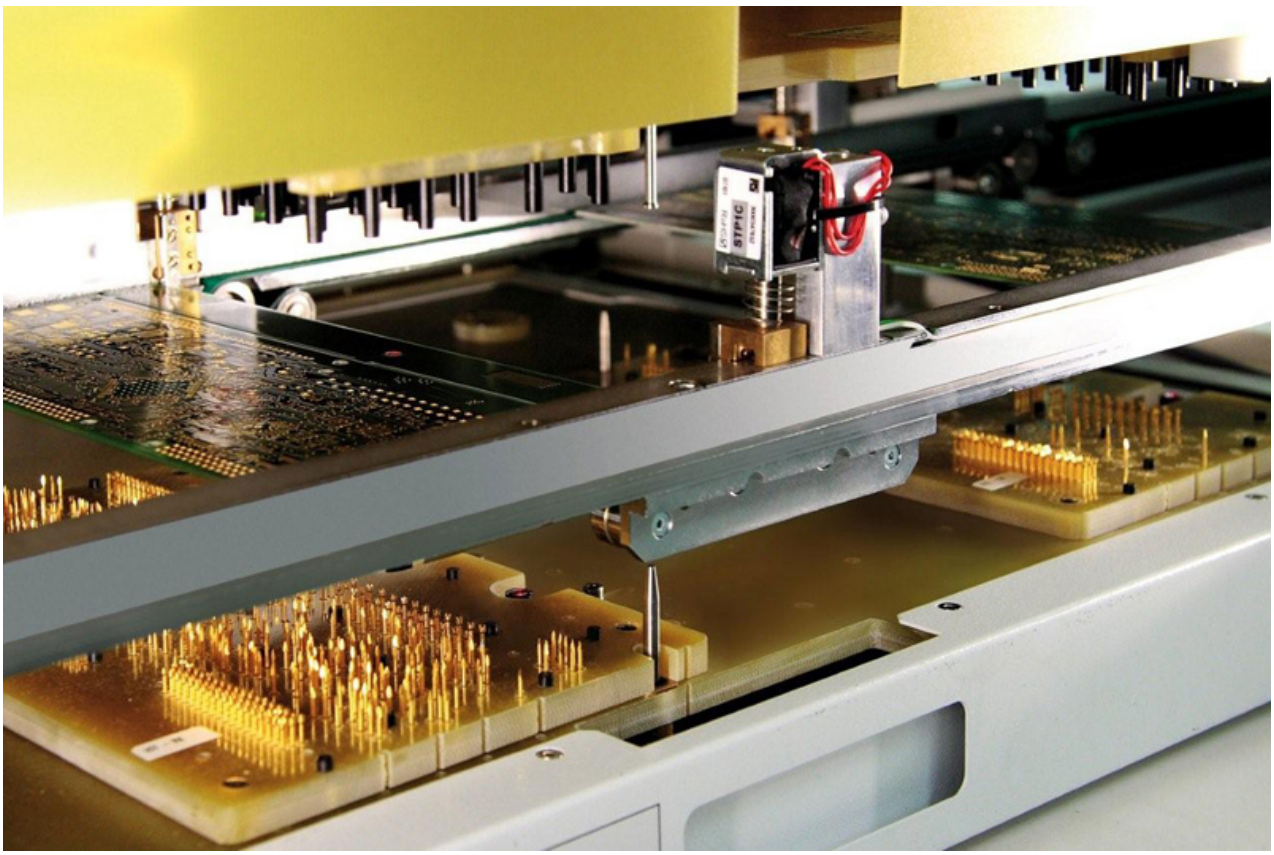

Рисунок 1.4 — Иллюстрация метода «Летающие матрицы» [1]

1.2.5 Анализ методов контактирования

Для наглядности преимуществ и недостатков четырех перечисленных методов, приведена таблица 1.1.

Таблица 1.1 — Преимущества и недостатки методов электрического контактирования

Метод	Вероятность ошибок	Производи-тельность	Разработка изделия	Разработка ПО	Стоимость
Ручной	Высокая	Низкая	Не требуется	Не требуется	Низкая
Ложе гвоздей	Низкая	Высокая	Требуется	Сложная	Средняя
Летающие щупы	Низкая	Средняя	Не требуется	Простая	Высокая
Летающие матрицы	Низкая	Высокая	Требуется	Средняя	Высокая

Выбор оборудования для электрического тестирования должен основываться на нескольких ключевых факторах: требуемой производительности, разнообразии выпускаемой продукции и уровне сложности печатных плат.

Ручной метод, хотя и является наименее затратным с точки зрения капитальных вложений, оправдан в основном для лабораторных исследований и производства прототипов. Однако надёжность такого контроля оставляет желать лучшего.

Оборудование с подвижными зондами выгодно отличается простой переналадки, что делает его оптимальным для мелкосерийного производства с широким ассортиментом плат. Тем не менее, производительность систем с «летающими щупами» недостаточна для массового производства.

Максимальную скорость проверки обеспечивают матричные тестеры, где основной лимитирующий фактор – скорость установки платы на контактное поле и время обработки измерительных сигналов. Такие системы используют коммутаторы на транзисторных ключах, что позволяет ускорить процесс тестирования. Однако применение релейных коммутаторов снижает производительность на порядок, из-за чего этот подход практически вышел из употребления.

Использование матричных тестеров требует значительных затрат на переналадку при смене типа платы, особенно если необходимо изготовление сменных адаптеров из-за несовпадения шагов контактирования с шагом контактного поля. В связи с этим подобные тестеры целесообразно применять в серийном и массовом производстве, где разнообразие выпускаемой продукции невелико.

Оптимальным решением, сочетающим универсальность и высокую производительность, является метод «летающих матриц». Он обеспечивает достаточную скорость проверки и подходит как для мелкосерийного, так и для серийного производства, независимо от ассортимента плат.

При выборе оборудования, помимо производительности и стоимости, следует учитывать тип и износостойкость контактных зондов, точность системы базирования, а также программное обеспечение, входящее в комплект. Именно ПО во многом определяет быстродействие установки и время её переналадки.

1.3 Программное обеспечение тестовых стендов

ПО для прошивки тестового стенда, предназначенного для проверки печатных плат, можно разделить на несколько категорий в зависимости от уровня управления, типа взаимодействия с оборудованием и целей тестирования.

1.3.1 Низкоуровневое программное обеспечение

На низком уровне функционирует встроенное ПО (firmware), которое управляет работой микроконтроллеров и отвечает за перемещение зондов, подачу управляющих сигналов, измерение отклика, коммутацию цепей и базовую обработку данных. Такое ПО обычно разрабатывается на языках C, C++ или Assembly и выполняется на встроенных процессорах тестера. В дополнение к этому работают драйверы оборудования, которые обеспечивают взаимодействие тестового стенда с операционной системой компьютера. Они создают интерфейсы для команд управления и передачи данных, используя API оборудования или стандартные протоколы, такие как USB, RS-232 и Ethernet.

1.3.2 Уровни управления и сбора данных

Следующим уровнем являются программы, предназначенные для настройки и управления процессом тестирования. В эту категорию входят графические интерфейсы оператора (GUI-программы), позволяющие загружать тестовые программы, визуализировать результаты тестирования, настраивать измерительные параметры и формировать отчеты. Такие приложения разрабатываются на C#, Python, Java, а также с использованием C++ и LabVIEW для визуального программирования. Дополнительно могут использоваться скриптовые системы управления тестированием, которые поддерживают автоматическое выполнение сценариев, написанных на Python или TCL.

1.3.3 Уровни обработки и анализа данных

После проведения тестов необходимо обработать и проанализировать полученные данные. Для этого применяются специализированные системы, выполняющие обработку измеренных параметров, сравнение с эталонными значениями и выявление брака. Эти программы могут использовать статистический анализ и формировать отчеты, работая на Python или MATLAB. Помимо анализа, важно обеспечить хранение данных и их интеграцию с производственными процессами. Для этого используются базы данных (SQL и NoSQL).

В современных системах тестирования активно применяется автоматическая генерация тестовых программ на основе CAD- и CAM-данных печатных плат. Такое ПО анализирует схемотехническую и трассировочную информацию, автоматически формируя последовательность тестирования, работая с форматами Gerber, ODB++ и IPC-2581. Некоторые системы используют алгоритмы машинного обучения для оптимизации тестов, сокращая их количество без потери качества.

1.4 Обзор метода «ложе гвоздей» с точки зрения программного обеспечения

Программное обеспечение для микроконтроллера, встроенного в тестовый стенд с методом «ложе гвоздей», выполняет несколько взаимосвязанных функций, обеспечивая передачу, обработку и управление исследуемой печатной платой. Основная задача прошивки МК – организация взаимодействия между контактной матрицей тестового адаптера и исследуемой платой, а также обеспечение точного измерения электрических параметров и своевременной передачи данных внешней системе управления или оператору. При использовании метода «ложе гвоздей» тестовый стенд оборудован матрицей подпружиненных контактов, которая позволяет одновременно контактировать с заданными точками печатной платы, что требует высокой точности управления коммутацией. Для этого программное обеспечение МК реализует алгоритмы управления матрицей контактов, инициируя замыкание

определённых групп контактов посредством цифровых или аналоговых переключателей, что позволяет подавать тестовые сигналы на плату и проводить измерения.

После активации нужной контактной группы происходит подача тестовых сигналов, измерение параметров, таких как напряжение, ток, сопротивление и другие характеристики, с использованием встроенных аналого-цифровых преобразователей (АЦП) и/или внешних измерительных модулей. Полученные данные проходят первичную обработку на микроконтроллере, где осуществляется фильтрация шумов, усреднение значений и сравнение с заранее заданными эталонными параметрами. Программное обеспечение анализирует результаты тестирования, выявляет возможные отклонения от нормы, а затем формирует отчёт, который передаётся во внешнюю систему управления через один из стандартных интерфейсов связи, например, UART, USB, SPI или Ethernet. Такой подход обеспечивает оперативное принятие решения о качестве тестируемой печатной платы.

Кроме того, программная архитектура МК может быть реализована с использованием многозадачности, например, с применением операционных систем реального времени (RTOS), что позволяет одновременно выполнять несколько задач: управление коммутацией, сбор данных, их предварительную обработку и коммуникацию с внешними устройствами.

Для разработки такого ПО используются языки низкого уровня, такие как C, C++ и Assembly, что обеспечивает максимальную производительность в условиях ограниченных вычислительных ресурсов микроконтроллера. Также важным аспектом является гибкость системы: возможность легкой перенастройки и обновления программной прошивки в зависимости от типа тестируемых печатных плат и изменяющихся требований производства. Таким образом, ПО для микроконтроллера тестового стенда с методом «ложе гвоздей» объединяет в себе функции управления контактной матрицей, проведения точных измерений, первичной обработки данных и обмена информацией, что обеспечивает высокую точность и эффективность тестирования печатных плат в автоматизированном режиме.

2 Практическая часть

2.1 Лазерный газоанализатор метана

Разрабатываемый в данном проекте тестовый стенд необходим для тестирования критически важных модулей и контрольных точек ПП лазерного газоанализатора, предназначенного для точного определения концентрации метана в воздухе с использованием специализированного алгоритма.

Внедрение стенда позволит обеспечить стабильное качество производства и сократить время на диагностику плат.

Устройство ПП газоанализатора состоит из двух ключевых модулей — Плату аналого-цифрового модуля (АЦМ) и внешний оптический блок

Оптический блок содержит:

- два канала предусилителей: а) аналитический — регистрирует отраженное лазерное излучение; б) реперный — обеспечивает эталонный сигнал для калибровки;
- лазерный модуль состоящий из: а) лазерный диод (источник излучения); б) элемент Пельтье (термостабилизация); в) термистор (контроль температуры);
- оптические компоненты: объектив, коллиматор, оптоволоконные линии.

АЦМ включает следующие компоненты:

- измерительный блок: а) три 18-битных АЦП для обработки сигналов реперного, аналитического и резервного каналов; б) один 16-битный АЦП, интегрированный в микроконтроллер (для термисторного канала); в) два 16-битных цифро-аналоговых преобразователей (управление током лазера и элемента Пельтье);
- цифровая часть: а) программируемая логическая интегральная схема (ПЛИС) — управляет цифро-аналоговый преобразователь (ЦАП), собирает данные с АЦП и выполняет их обработку; б) МК — рассчитывает концентрацию метана и обеспечивает связь через интерфейсы Ethernet и RS-232;

– интерфейсный модуль: а) разъемы и схемы согласования для подключения внешних устройств; б) индикаторы статуса питания, режима работы, состояния оптического блока и GPS.

2.2 Методика разработки

Объектами разработки является прошивка для МК тестового стенда и ПО с графическим интерфейсом на ПК для управления прошивкой МК. Перед началом проведения работ нужно выбрать микропроцессор, который будет центральным звеном всего проекта. От его выбора зависит большое количество факторов, таких как:

– производительность — тактовая частота, разрядность и архитектура (8, 16, 32 бит) определяют скорость обработки данных;
– объём памяти – важно учитывать объём оперативной (RAM) и флеш-памяти для хранения данных и кода программы;
– наличие и количество периферийных интерфейсов – поддержка UART, SPI, I2C, CAN, USB, Ethernet и других интерфейсов для связи с другими устройствами;
– совместимость с выбранной средой разработки – поддержка компиляторов и отладочных инструментов, например, Keil, IAR, GCC, STM32CubeIDE;
– доступность и стоимость – цена микропроцессора и его наличие на рынке;
– поддержка аналоговых и цифровых входов/выходов – наличие ADC, DAC, GPIO и других встроенных модулей.

Правильный выбор микропроцессора определяет среду разработки, используемые библиотеки, а также надёжность и эффективность работы тестового стенда и удобство его дальнейшей модернизации. Далее идет аргументирование выбора МК, а также описание всех сред разработок, в которых проводились работы.

2.2.1 Выбор 32-разрядного микроконтроллера

Выбор 32-разрядного микроконтроллера (МК) вместо 8- или 16-разрядного обусловлен повышенной производительностью, большим объёмом памяти, поддержкой современных интерфейсов (USB, Ethernet, CAN), возможностью работы с RTOS и энергоэффективностью. Среди 32-битных МК семейство STM32 (STMicroelectronics) выделяется надёжностью, доступностью и универсальностью благодаря ядрам ARM Cortex, статической RAM, флеш-памяти, отладочным интерфейсам (SWD/JTAG) и разнообразной периферии.

Для тестового стенда лазерного детектора метана выбран STM32F411CEU6 из-за совместимости с архитектурой STM32H743VIT (используется в плате газоанализатора), что упрощает разработку ПО.

Серия STM32F4 (Cortex-M4) обеспечивает высокую производительность, но для проекта достаточно компактного STM32F411CEU6. Он сочетает тактовую частоту до 100 МГц, 512 КБ флеш–памяти, 128 КБ RAM, 12-битный АЦП (16 каналов), интерфейсы USART, SPI, I2C, USB OTG, DMA и режимы энергосбережения. Это делает его оптимальным по цене, мощности и функционалу, заменяя устаревшие решения вроде «Blue Pill» (STM32F103C8T6) на современный «Black Pill» с улучшенными характеристиками.

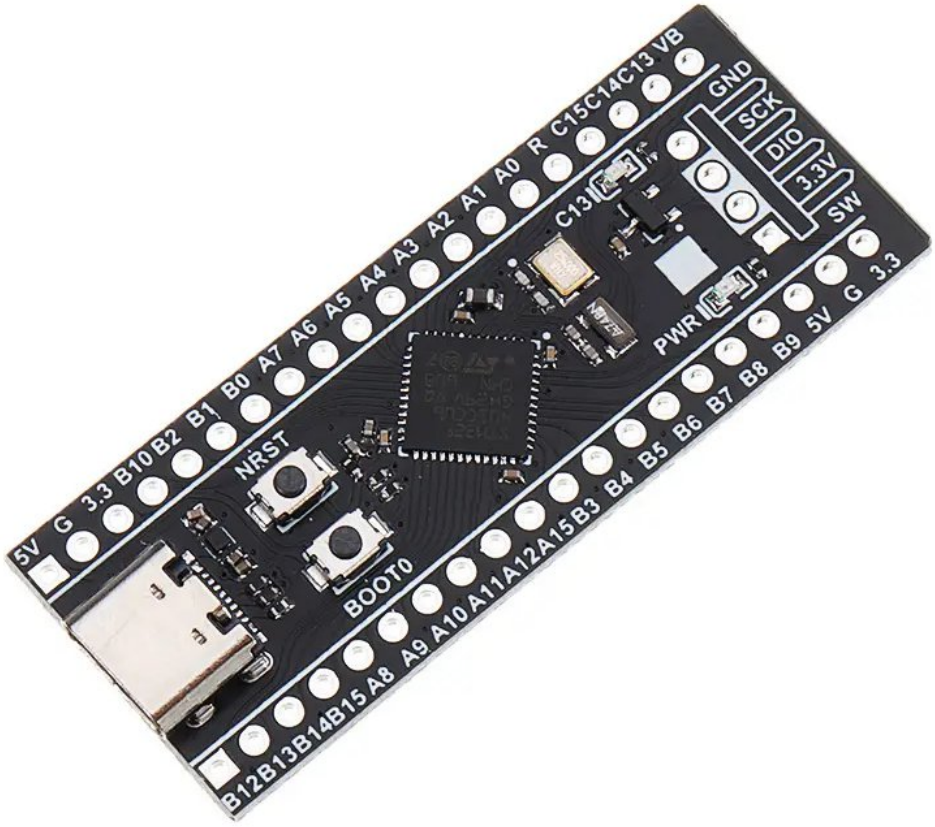

Рисунок 2.1 — Отладочная плата «Black Pill»

Таким образом, отладочная плата «Black Pill» с микроконтроллером STM32F411CEU6 полностью удовлетворяет всем требованиям проекта. Он обеспечивает необходимую производительность, обладает достаточным объемом памяти и поддерживает все необходимые интерфейсы. При этом его стоимость и энергопотребление находятся на оптимальном уровне, что делает его экономически выгодным выбором для реализации поставленных задач.

2.2.2 Среды разработки

До начала проведения работ требуется выбор среды разработки. Он зависит от совместимости с микроконтроллером, удобства работы, доступности инструментов отладки и поддержки необходимых библиотек. Среди доступных решений можно рассмотреть STM32CubeIDE, Keil MDK, IAR, Arduino IDE и другие. Для разработки

графических приложений подходят Visual Studio, Qt creator, CLion и другие. Выбор среды влияет на скорость разработки, гибкость и удобство.

STM32CubeIDE — комплексная платформа для разработки ПО. Она предназначена для создания приложений под микроконтроллеры серии STM32 и объединяет в себе все необходимые инструменты: редактор, компилятор, отладчик и утилиты для тестирования. Решение полностью бесплатно и оптимизировано для проектов, где требуется точная работа с аппаратными компонентами.

Одна из сильных сторон среды — тесная интеграция с STM32CubeMX, утилитой для настройки периферийных модулей микроконтроллера. С её помощью можно графически задавать параметры тактовых сигналов, портов ввода-вывода, интерфейсов связи (UART, SPI, I2C) и других систем, после чего платформа автоматически формирует базовый код на языке C. Это ускоряет настройку проектов и снижает риск ошибок при ручном написании конфигураций.

Для компиляции используется инструментарий GNU (GCC), поддерживающий языки C и C++, что позволяет задействовать современные подходы в программировании. Встроенный отладчик работает с протоколами SWD и JTAG, обеспечивая пошаговое выполнение кода, мониторинг переменных, регистров и памяти.

Преимущества данной платформы — минимальные требования к ручной настройке, доступ к низкоуровневым библиотекам Hardware Abstraction Layer (HAL) и Low Layer (LL), а также возможность быстрого старта за счёт автоматической генерации кода.

Qt Creator — это кроссплатформенная интегрированная среда разработки (IDE), созданная для работы с фреймворком Qt, но также поддерживающая проекты на C++, Python и других языках. Эта среда предоставляет инструменты для полного цикла создания приложений: от написания кода и проектирования интерфейсов до отладки, профилирования и развёртывания на целевых платформах. Qt Creator работает на Windows, macOS и Linux, сохраняя единый интерфейс и функционал во всех операционных системах, что делает её удобной для создания собственного приложения с необходимыми возможностями для данной работы.

Среда предоставляет визуальный редактор Qt Designer, позволяющий «собирать» GUI-приложения из готовых виджетов или QML–компонентов, автоматически генерируя соответствующий код. Поддержка автоматической программной сборки (CMake и QMake) упрощает настройку сборки проектов, а интеграция с системами контроля версий (Git) облегчает работу.

В данном проекте эта среда разработки используется для разработки графического приложения, управляющей тестовым стендом.

2.3 Функционал тестового стенда

Перед написанием кода необходимо определить какие модули и функции необходимо тестировать на ПП газоанализатора, чтобы можно было утверждать, что плата исправна и не имеет дефектов, влияющих на ее работу. Функции можно разделить на два типа. К первому типу относятся функции, которые выполняются прошивкой МК тестового стенда. Ко второму типу относятся функции, которые выполняются приложением на ПК, напрямую передающие команды выполнения плате газоанализатора.

К прошивке МК относятся:

- измерение напряжения в контрольных точках платы: 15 точек, напряжения от -6 до +6В;
- «Замыкание» посадочных мест резисторов R1 и R22;
- подача и снятие питания прибора;
- измерение напряжения и тока питания прибора;
- подключение/отключение эквивалента лазерного модуля (лазерный диод, элемент Пельтье и термистор);
- измерение формы тока через эквивалент лазерного диода с частотой 10 КГц;
- преобразование тока лазера в напряжение для подачи на АЦП прибора;
- измерение тока и напряжения на эквиваленте элемента Пельтье;
- имитация GPS-модуля;
- связь с ПК по интерфейсу USB-UART;

– тестирование связи с прибором по интерфейсу RS232.

К функциям приложения на ПК относятся:

– запуск измерений в тестовом режиме;

– установка формы тока лазера;

– установка температуры в градусах Цельсия.

Таким образом, прошивка для МК и ПО для ПК включают в себя вышеперечисленные функции.

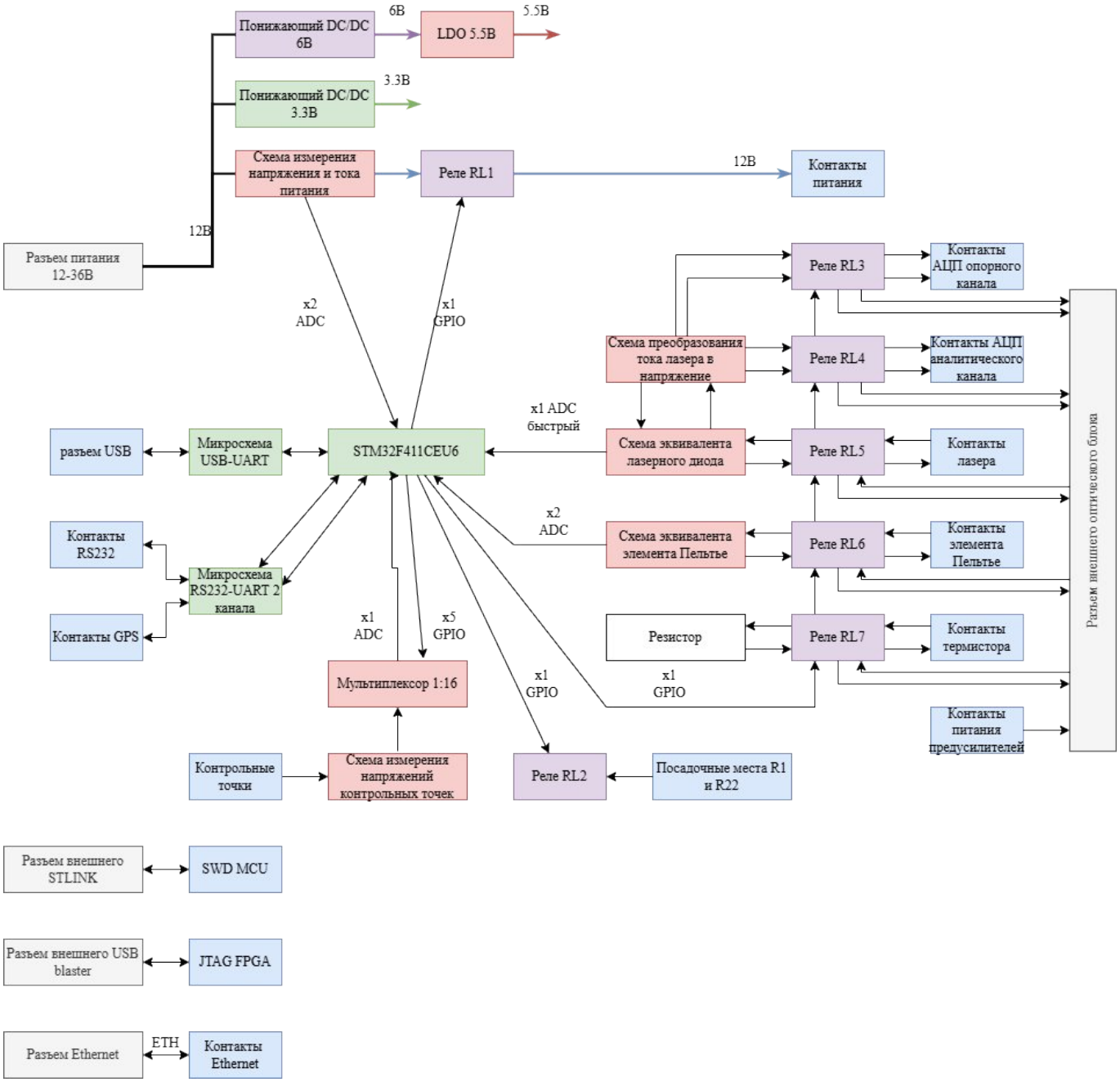

Рисунок 2.2 — Функциональная схема платы тестового стенда

Функциональная схема (рисунок 2.2) тестового стенда для газоанализатора демонстрирует принцип работы, основанный на контроле, измерении и управлении различными модулями платы прибора с помощью микроконтроллера STM32F411CEU6. Питание стенда обеспечивается от внешнего источника с диапазоном напряжений 12–36 В. Это напряжение поступает на понижающий DC/DC-преобразователь, где понижается до 6 В. Далее формируются стабилизированные напряжения 5.5 В (через LDO) и 3.3 В (через дополнительный преобразователь), которые используются для питания логических цепей и самого микроконтроллера.

Подача питания на тестируемое устройство осуществляется через реле RL1, которое управляется через GPIO микроконтроллера. Для контроля состояния прибора реализовано измерение тока и напряжения питания с использованием двух каналов АЦП.

Для измерения напряжений в 15 контрольных точках используется мультиплексор 1:16, подключённый к одному каналу АЦП. Микроконтроллер поочерёдно переключает входы мультиплексора и считывает значения напряжений. Управление реле RL2 позволяет замыкать цепи резисторов R1 и R22, что необходимо для проверки их установки и контактирования. Для симуляции внешних компонентов прибора используются реле RL3–RL7. RL3 подключает опорный канал, RL4 – канал АЦП, RL5 – эквивалент лазерного диода, RL6 – элемент Пельтье, а RL7 – термистор. Все эти цепи позволяют провести полноценную проверку взаимодействия прибора с внешними модулями.

Для преобразования тока лазерного диода в напряжение, пригодное для подачи на АЦП, реализована специализированная схема, подключенная к отдельному каналу АЦП. Коммуникационные интерфейсы представлены USB-UART для связи с ПК, RS232 и разъёмом GPS, имитирующим работу соответствующего модуля. Кроме того, предусмотрена поддержка Ethernet и интерфейсы для программирования и отладки: SWD – для микроконтроллера с использование программатора ST-LINK V2 и JTAG – для ПЛИС.

Таким образом, функциональная схема представляет собой многофункциональный, гибкий и автоматизированный стенд, обеспечивающий полную проверку электрических параметров, работоспособности интерфейсов и симуляции работы устройства в условиях, максимально приближенных к реальной эксплуатации.

2.4 Конфигурация выводов микроконтроллера

Для корректной работы проектируемого устройства необходимо выполнить настройку выводов микроконтроллера STM32F411CEU6 в соответствии с его функциональными задачами, описанными в предыдущем пункте, рисунок 2.3.

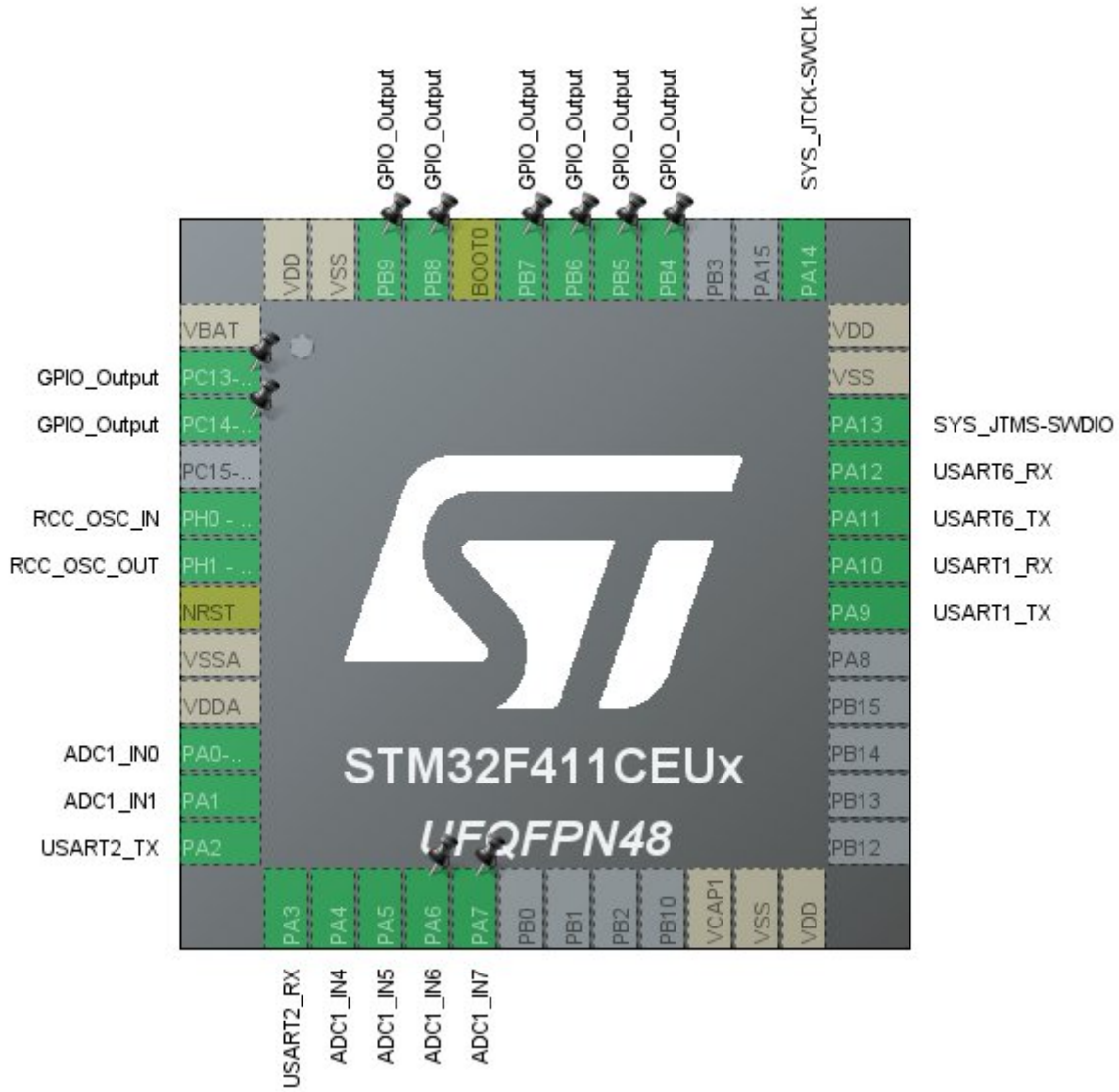

Рисунок 2.3 — Конфигурация выводов микроконтроллера

- цифровые выходы управления (GPIO) — для подачи управляющих сигналов на исполнительные устройства такие как: реле и мультиплексор 1:16. Для этого задействованы следующие цифровые выводы: PB4, PB5, PB6, PB7, PB8, PB9, PC13, PC14 — настроены в режиме GPIO Output Push-Pull. Где PB4-PB8 — используются для управления мультиплексором 1:16. PB9, PC13 и PC14 — управляют состояниями реле.
- аналоговые входы (ADC) — для измерения напряжений и токов на контрольных точках и эквивалентных схемах используется ADC1. Используются следующие выводы с функцией аналогового входа: ADC1_IN0 (PA0), ADC1_IN1 (PA1), ADC1_IN4 (PA4), ADC1_IN5 (PA5), ADC1_IN6 (PA6), ADC1_IN7 (PA7). Где ADC1_IN0 — Для измерения формы тока лазерного диода, ADC1_IN1 — подается на мультиплексор, ADC1_IN4 и ADC1_IN5 — для измерения напряжения и тока питания соответственно, ADC1_IN6 и ADC1_IN7 — для измерения падения напряжения элемента Пельтье.
- интерфейс связи (UART) — для связи микроконтроллера с внешними устройствами через интерфейс UART используется USART1, USART2 и USART6. В данной конфигурации задействованы выводы: PA10 (USART1_RX) и PA9 (USART1_TX) — для UART-RS-232, PA2 (USART2_TX) и PA3 (USART2_RX) — для UART-USB, PA12 (USART6_RX) и PA11 (USART6_TX) — для UART_GPS. Все 3 UART обеспечивают двунаправленную асинхронную последовательную передачу данных.
- подключение отладчика ST-LINK V2 — для программирования и отладки через ST-LINK V2 используется интерфейс SWD (Serial Wire Debug): PA13 — SWDIO (SYS_JTMS-SWDIO) и PA14 — SWCLK (SYS_JTCK-SWCLK)
- тактирование — тактовый генератор микроконтроллера работает от внешнего кварцевого резонатора с максимальной частотой 25 Мгц, установленного на отладочной плате «Black Pill». Резонатор подключается к выводам PH0 (RCC_OSC_IN) и PH1 (RCC_OSC_OUT), обеспечивая стабильную частоту тактирования для всей системы.

Назначение выводов и настройка частоты МК производилось в среде разработки STM32CubeIDE. Схема тактирования представлена на рисунке 2.4.

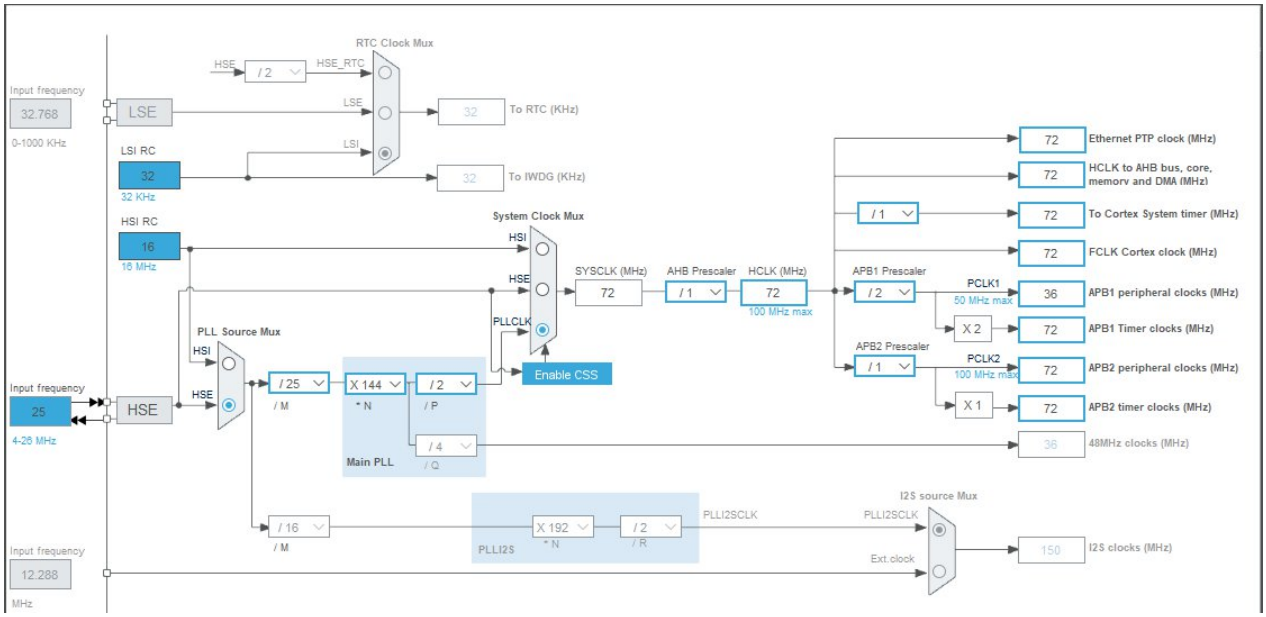

Рисунок 2.4 — Схема тактирования микроконтроллера

2.5 Разработка прошивки микроконтроллера

Прошивка микроконтроллера в рамках данного проекта организована модульно. Такой подход позволяет не только облегчить разработку и отладку системы, но и обеспечивает наглядность структуры программы, её читаемость и удобство масштабирования в будущем.

Структура программы построена на базе HAL библиотеки, а так же были использованы библиотеки CMSIS (Cortex Microcontroller Software Interface Standard) и LL там, где можно оптимизировать код. В проекте использовался компилятор GCC.

Логика работы микроконтроллера разделена на три основных модуля, каждый из которых выполняет строго определённый набор задач:

1) модуль инициализации системы и управления основным циклом выполнения (модуль main);
2) модуль обработки протокольных пакетов и диспетчеризации команд (модуль parser);
3) модуль управления и взаимодействием с периферией (модуль hardware).

Каждый из этих модулей играет ключевую роль в общей архитектуре прошивки, рисунок 2.5. Далее подробный обзор их назначения и алгоритма работы.

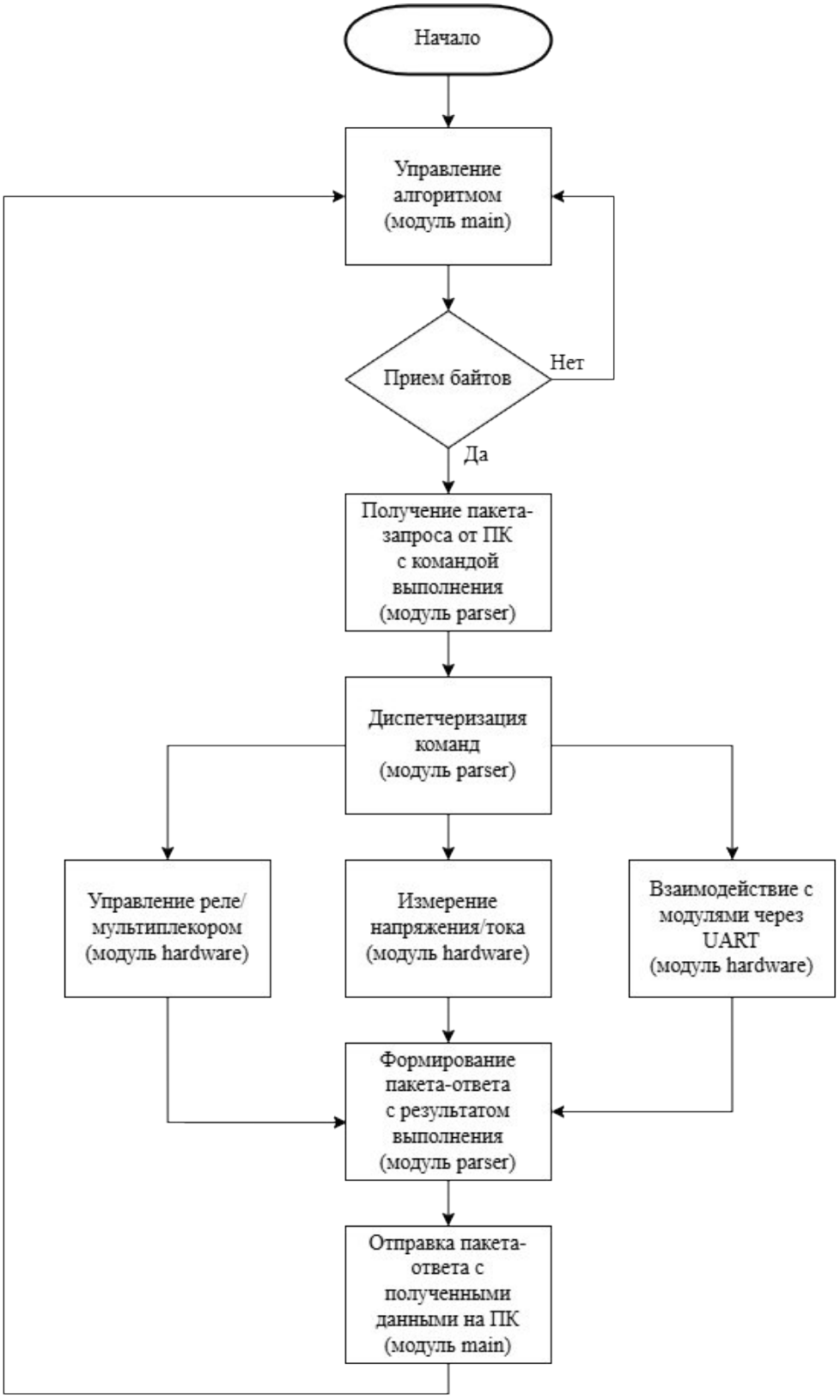

Рисунок 2.5 — Обобщенная блок-схема алгоритма работы прошивки

2.5.1 Модуль инициализации системы и управления основным циклом выполнения

Главный модуль main содержит точку входа программы и реализует базовую инициализацию всех используемых периферийных устройств: GPIO, UART, ADC, системного тактирования. Также он содержит основной бесконечный цикл, в котором происходит приём и обработка входящих UART-сообщений в асинхронном режиме, рисунок 2.6.

Программа начинается с вызова стандартной функции HAL_Init(), которая настраивает базовые системные таймеры и прерывания. Затем вызывается функция SystemClock_Config(), которая отвечает за настройку тактового генератора.

После настройки тактирования осуществляется инициализация всех необходимых периферий: UART (USART1, USART2, USART6), аналогово-цифрового преобразователя (ADC), а также цифровых входов/выходов (GPIO). Все эти шаги позволяют подготовить микроконтроллер к полноценной работе в режиме реального времени.

После завершения этапа инициализации программа входит в бесконечный цикл while(1), в рамках которого происходит основной обмен информацией с внешними устройствами. На каждом цикле вызывается функция HAL_UART_Receive_IT(), инициирующая приём одного байта данных по интерфейсу UART в асинхронном режиме. Как только байт принят, вызывается функция обратного вызова (прерывание UART) HAL_UART_RxCpltCallback(), которая устанавливает флаг status в значение STATUS_OK, сигнализируя основному циклу о готовности к обработке данных. Далее, если флаг установлен, вызывается функция process_rx_byte(), которая реализует поэтапный разбор входящего пакета по определённому протоколу.

Если все байты успешно разобраны и собран полный корректный пакет (функция возвращает PARSER_DONE), вызывается функция choose_command(), которая определяет, какую команду необходимо выполнить. На этом этапе производится вызов соответствующих функций из модуля аппаратной логики, например: включение реле, измерение напряжения, тестирование нагрузки и т.д.

После вызывается функция serialize_reply(), которая формирует окончательный пакет: добавляет заголовки, вычисляет контрольную сумму CRC и записывает всё это в буфер. Полученный пакет передаётся обратно по UART с помощью функции HAL_UART_Transmit_IT().

В случае, если при разборе пакета была обнаружена ошибка (например, некорректная CRC или структура), функция process_rx_byte() возвращает PARSER_ERROR, и в этом случае состояние парсера сбрасывается в STATE_SYNC, чтобы подготовиться к приёму нового пакета. Независимо от результата обработки, в конце цикла значение переменной status принудительно сбрасывается, что предотвращает повторную обработку одного и того же байта.

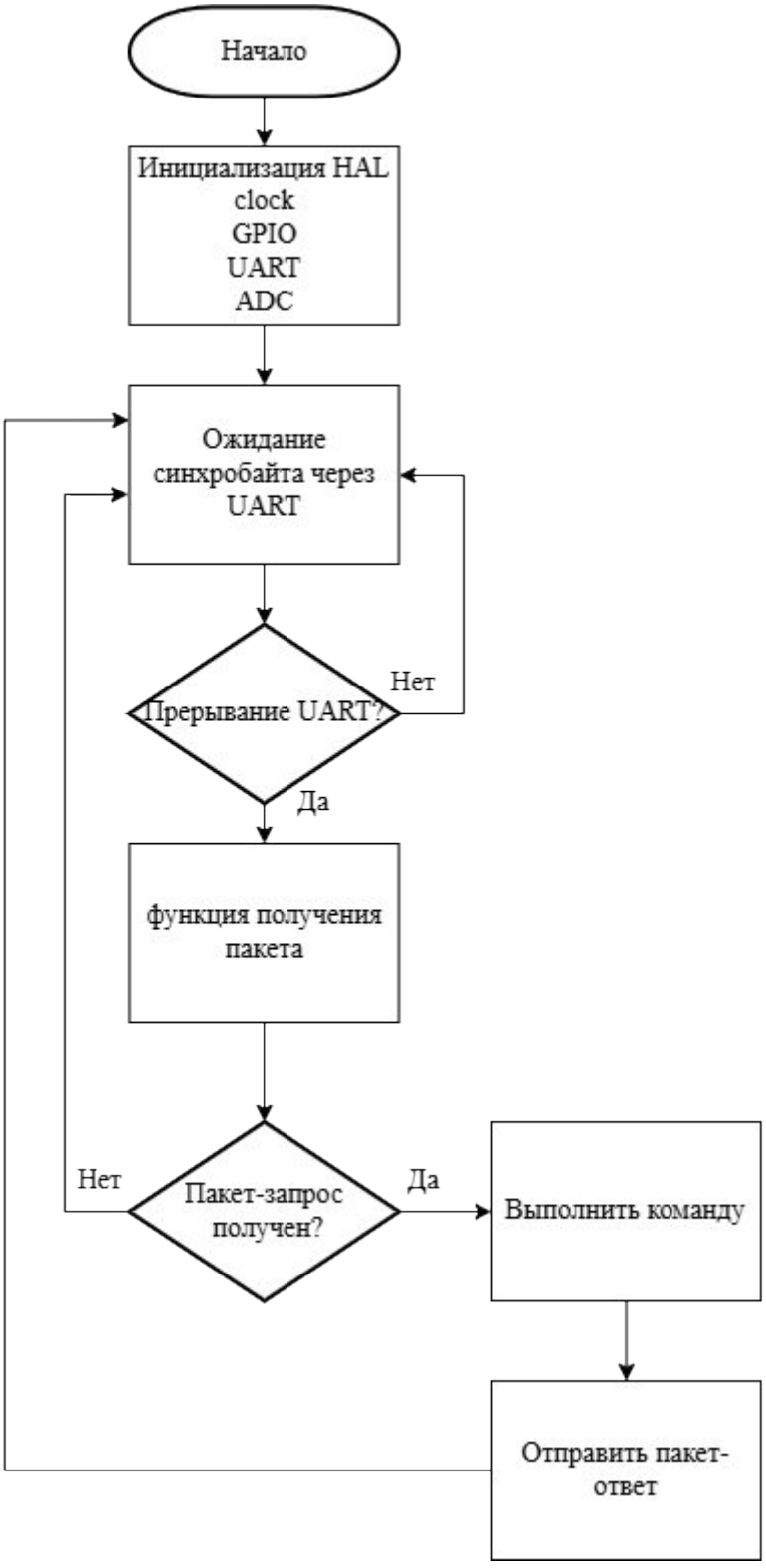

Рисунок 2.6 — Блок-схема модуля инициализации системы и управления основным циклом выполнения

Исходный код модуля main приведен в приложении А.

2.5.2 Модуль обработки протокольных пакетов и диспетчеризации команд

Модуль parser отвечает за обработку принятых байтов, поступивших по UART в микроконтроллер от ПК через переходник UART—USB. Модуль проводит диспетчеризацию команд, а также формирует байты с результатом выполнения команд и измеренными данными для передачи по UART в ПК.

Для корректной обработки и передачи байтов, разработан протокол обмена данными между ПК и МК, представляющий собой простой, но надёжный механизм взаимодействия на основе последовательной передачи данных. Протокол обеспечивает выполнение команд по установке и получению параметров с последующим формированием ответов, а также контроль целостности пакетов при помощи контрольной суммы CRC16. Каждый пакет имеет чётко определённую структуру, состоящую из синхробайта, длины полезной части, кода команды, данных команды и контрольной суммы, таблица 2.1.

Коммуникация начинается с отправки запроса, содержащего один байт синхронизации (0xAA), за которым следуют два байта, обозначающих размер полезной части пакета, включающей код команды, возможные данные и два байта CRC. Далее следует сам код команды, который определяет тип запроса — например, получение значения параметра или его установка. Команда сопровождается данными, специфичными для данного типа запроса. В конце пакета передаётся CRC, рассчитанная на основе всех байтов начиная с кода команды.

Таблица 2.1 — Содержание пакета

Байты	Размер (в байтах)	Описание
0	1	Синхробайт (0xAA)
1 – 2	2	Размер полезных данных: – код команды (1 байт); – данные команды (N байт в диапазоне [0;65529]; – CRC16 сумма (2 байта).
3	1	Код команды
4 – …	N	Данные
(K-2) – (K-1)	2	CRC16 сумма

K — размер пакета, максимум которого 65535 байт.

После получения корректного пакета микроконтроллер анализирует команду и в зависимости от её кода выполняет соответствующее действие, таблица 2.2. Например, при получении команды с кодом 0 осуществляется чтение значения определённого параметра (по его коду), а команда с кодом 1 используется для установки нового значения параметра. По завершении операции микроконтроллер формирует ответный пакет, структура которого аналогична запросу, таблица 2.3. В нём содержится оригинальный код команды, код ошибки (в случае успешного выполнения — 0), и, если необходимо, возвращаемое значение.

Система команд поддерживает как минимальные по объёму запросы (например, только код параметра), так и более расширенные — с установкой значений в виде 4-байтных слов. Размер передаваемых данных может достигать до 65535 байт, что позволяет передавать значительные объёмы информации, если потребуется.

Таблица 2.2 — Набор команд

Код команды	Описание	Данные команды			Размер данных (байт)
		Байты	Размер (байт)	Описание	
0	Получение параметра	0	1	Код параметра	1
1	Установка параметра	0	1	Код параметра	5
		1 – 4	4	Значение параметра	
В ответ на команду МК формирует пакет с кодом принятой команды (X)					
X	Ответ на команду	0	1	Код ошибки	5
		1 – 4	4	Ответное значение	

Таблица 2.3 — Набор ответов

Код команды	Описание	Данные команды			Размер данных (байт)
		Байты	Размер (байт)	Описание	
0	Ответ на запрос параметра	0	1	Код ошибки	5
		1 – 4	4	Значение параметра	

Окончание таблицы 2.3

1	Ответ на установку параметра	0	1	Код ошибки	5
		1 – 4	4	Значение параметра	

Также предусмотрен набор кодов ошибок, передаваемых в ответных пакетах, таблица 2.4. Это позволяет легко диагностировать сбои и реагировать на них в программной логике верхнего уровня.

Таблица 2.4 — Коды ошибок

Код ошибки	Описание
0	Нет ошибки
1	Ошибка выполнения команды
2	Несуществующая команда
3	Превышено время выполнения команды
4	Ошибка размера данных команды

Пример для установки параметра и перечень функций тестового стенда приведен в приложении Б. Блок схема алгоритма получения байтов через UART приведена на рисунке 2.7.

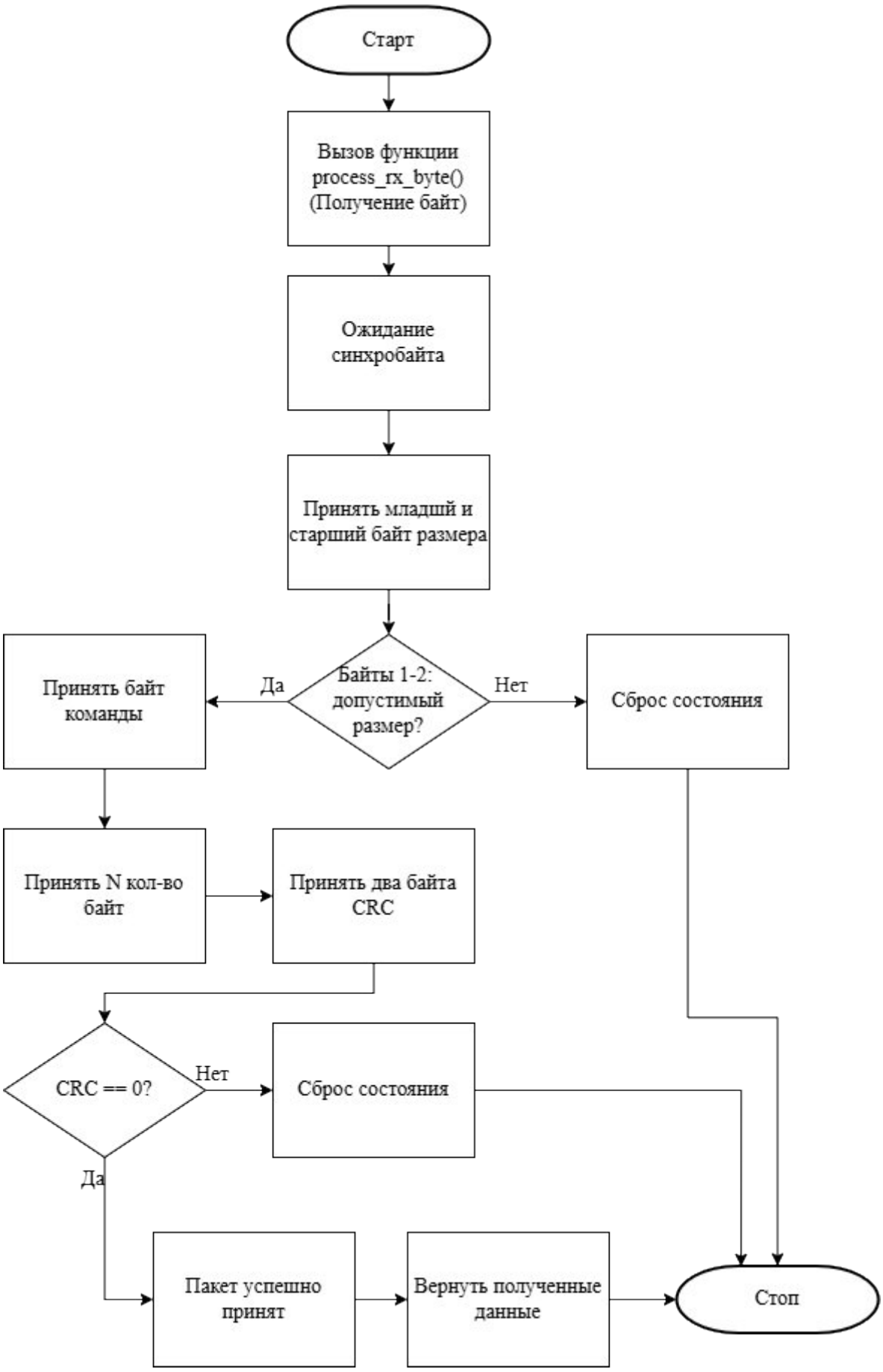

Рисунок 2.7 — Блок схема функции process_rx_byte()

Далее после успешного принятия всего пакета, вызывается функция ветвления choose_command(), рисунок 2.8, в которой в зависимости от принятый байтов кода команды и кода параметра, вызывается модуль hardware, в котором происходят действия, указанные в коде команды и коде параметра. Оттуда возвращается статус выполнения команды, а так же полученный результат, на основе которых функция serialize_reply() формирует-ответ, в который передает байт ошибки и полученные данные. После этого модуль main отправляет сформированный пакет по UART побайтно.

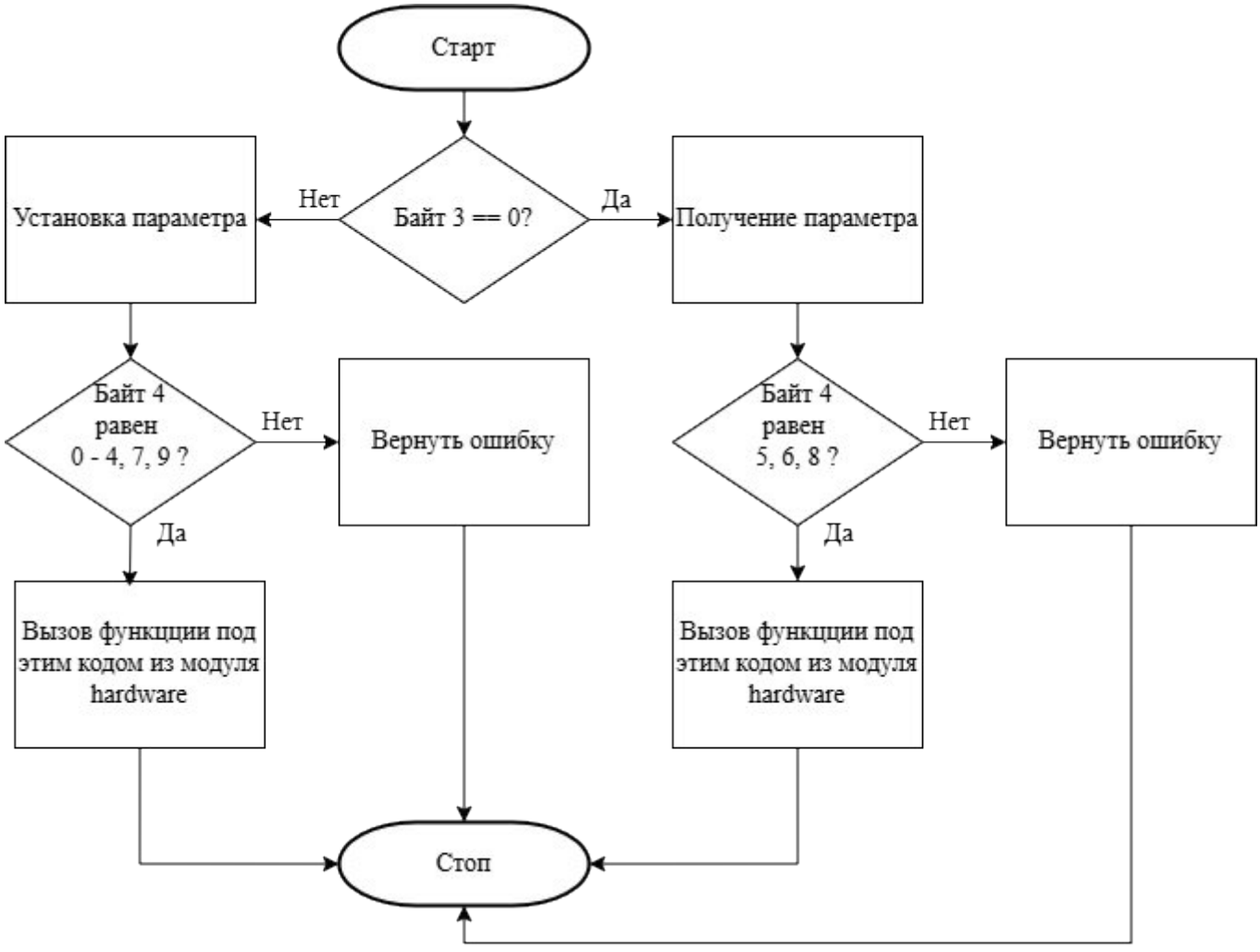

Рисунок 2.8 — Блок-смеха функции choose_command()

Таким образом, протокол построен по принципу запроса-ответа с минимальной избыточностью и высокой гибкостью. Исходный код модуля parser приведен в приложении В.

2.5.3 Модуль управления и взаимодействием с периферией

Модуль hardware реализует взаимодействие микроконтроллера с аппаратной частью устройства. Его основная задача — выполнение команд, полученных по протоколу, путём непосредственного управления реле, считывания аналоговых сигналов (напряжений и токов), а также обмена данными с внешними устройствами через интерфейсы UART.

Одной из ключевых функций модуля является управление реле. Это реализовано через функции apply_voltage_relay_X(), где X — номер реле. В зависимости от команды (открыть или закрыть), происходит установка соответствующего пина GPIO, подключённого к обмотке реле. После активации производится проверка состояния пина, чтобы удостовериться, что реле сработало. Если всё прошло успешно, в выходной буфер возвращается статус выполнения OK, иначе — код ошибки.

Также в модуле реализованы функции измерения напряжений на контрольных точках. Например, функции test_voltage_4_point() и test_voltage_11_point() позволяют выбрать нужную точку путём переключения мультиплексора, после чего производится серия измерений напряжения с использованием АЦП. Усреднённое значение переводится в милливольты и записывается в выходной буфер. В похожем стиле реализовано измерение тока (test_voltage_current()), в котором напряжение на шунте пересчитывается в силу тока.

Отдельно реализована функция test_voltage_peltie(), предназначенная для диагностики тока, проходящего через модуль Пельтье. В ходе работы этой функции осуществляется измерение двух напряжений, рассчитывается разность и на её основе определяется направление и величина тока. Это позволяет отслеживать правильную работу термоэлектрического элемента.

Модуль также содержит вспомогательные функции, такие как set_pins() — для управления четырьмя цифровыми линиями, а также apply_relay() — для подачи управляющего импульса на конкретный пин.

Немаловажной частью является реализация обмена по UART с внешними модулями. Например, функции massage_rs232() и massage_gps() осуществляют передачу и приём строк по соответствующим интерфейсам, после чего производится побайтовое сравнение отправленных и полученных данных. Если ответы совпадают, функция возвращает статус OK, иначе — сообщает об ошибке выполнения.

Таким образом, модуль hardware.c является «исполнителем» команд: он напрямую взаимодействует с физической частью устройства и выполняет всю низкоуровневую работу, оставляя другим модулям прошивки (parser и main) только обработку логики и структуры пакетов. Исходный код модуля parser приведен в приложении Г.

2.6 Разработка приложения для Windows

Представленный алгоритм представляет собой комплексное решение для автоматизированного тестирования печатной платы лазерного детектора метана. Программа реализована на C++ с использованием фреймворка Qt в IDE «qt creator» и предназначена для проверки функциональности всех критически важных узлов устройства: цепей питания, аналоговых и цифровых интерфейсов, работы микроконтроллера и периферийных компонентов. Алгоритм сочетает последовательное выполнение тестовых сценариев с гибкой системой обработки ошибок и интерактивным взаимодействием с пользованием. В приложении есть встроенный график, на который выводится сигнал полученный с платы тестового стенда. Полный алгоритм тестирования ПП газоанализатора приведен в приложении Д. Для более четкого понимания того, как устроен проект, приведен рисунок 2.9.

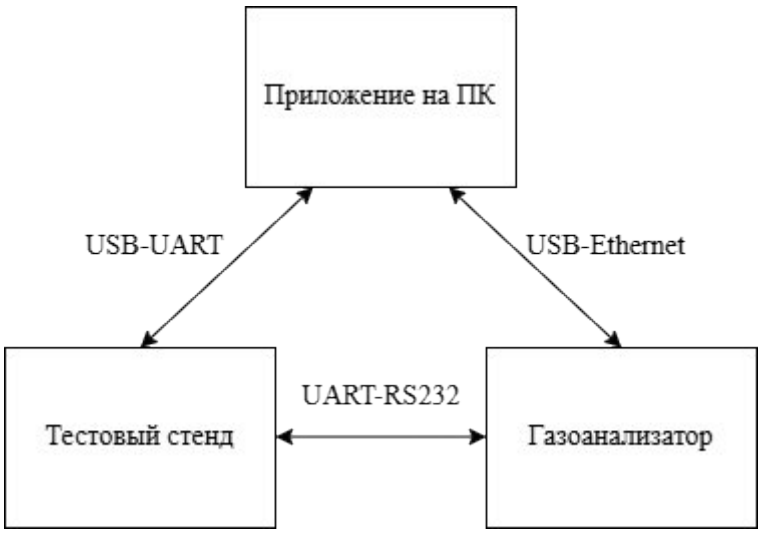

Рисунок 2.9 — Блок-схема подключения

ПО для ПК является главным элементом схемы. Подавая команды на обе платы, приложение получает ответ, со статусом выполнения. Таким образом, прошивка для МК тестового стенда является алгоритмом без возможности самостоятельной работы, так как она только исполняет команды и возвращает измеренные данные, а так же код ошибки. Все остальная логика по оцениваю данных описана в алгоритме работы приложения.

Программа взаимодействует с тестируемой платой через два последовательных порта: USB-UART для отправки команд стенду и USB-Ethernet для коммуникации с платой газоанализатора. В разработке приложения использовались библиотеки «QCustomPlot», для построения графика, а так же библиотека «Func_acm» с описанными функциями для подачи команд на плату газоанализатора. алгоритмы формирования пакетов и получения пакетов идентичны алгоритму в прошивке МК тестового стенда. Исходный код приведен в приложении Е. Исходный код для функций, которые может выполнять плата АЦМ, приведен в приложении Ж, в виде шаблонов функий, потому что конкретная реализация этих функций представляет собой коммерческую тайну. Такия реализация содержит уникальные алгоритмы, оптимизации и технические решения, разработанные компанией для достижения конкурентных преимуществ. Раскрытие полного кода могло бы привести к утечке интеллектуальной собственности, упрощению копирования продукта, что нанесло бы ущерб бизнесу. Поэтому в приложении Ж представлено только интерфейсы и

шаблоны, необходимые для взаимодействия с платой, а сами детали реализации остаются закрытыми и защищёнными от несанкционированного доступа.

Графический интерфейс, на рисунке 2.10, включает:

- элементы управления — выбор COM-порта, настройка скорости передачи (Baudrate), кнопки запуска/остановки теста;
- журнал событий — текстовое поле с поддержкой HTML-форматирования для наглядного отображения статусов;
- график — виджет для визуализации аналоговых сигналов;
- кнопка «Отчистить все» — отчистка журнала событий перед новым тестом.

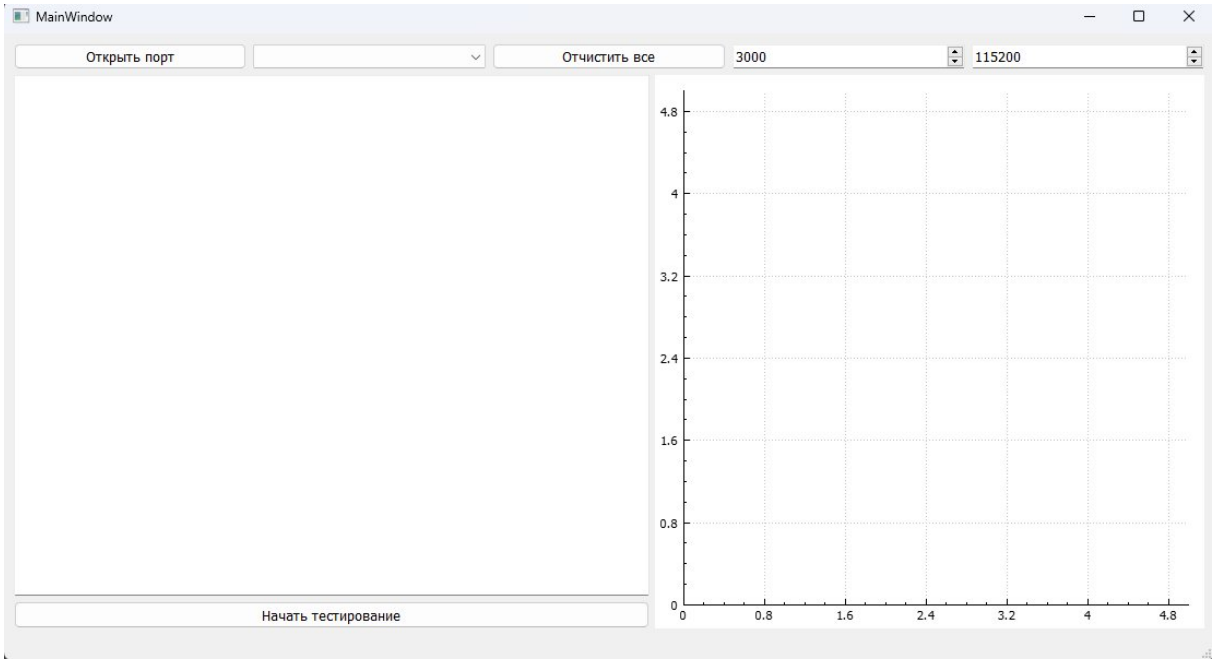

Рисунок 2.10 — Графический интерфейс приложения для Windows

Общую концепция алгоритма приведена на рисунке 2.11.

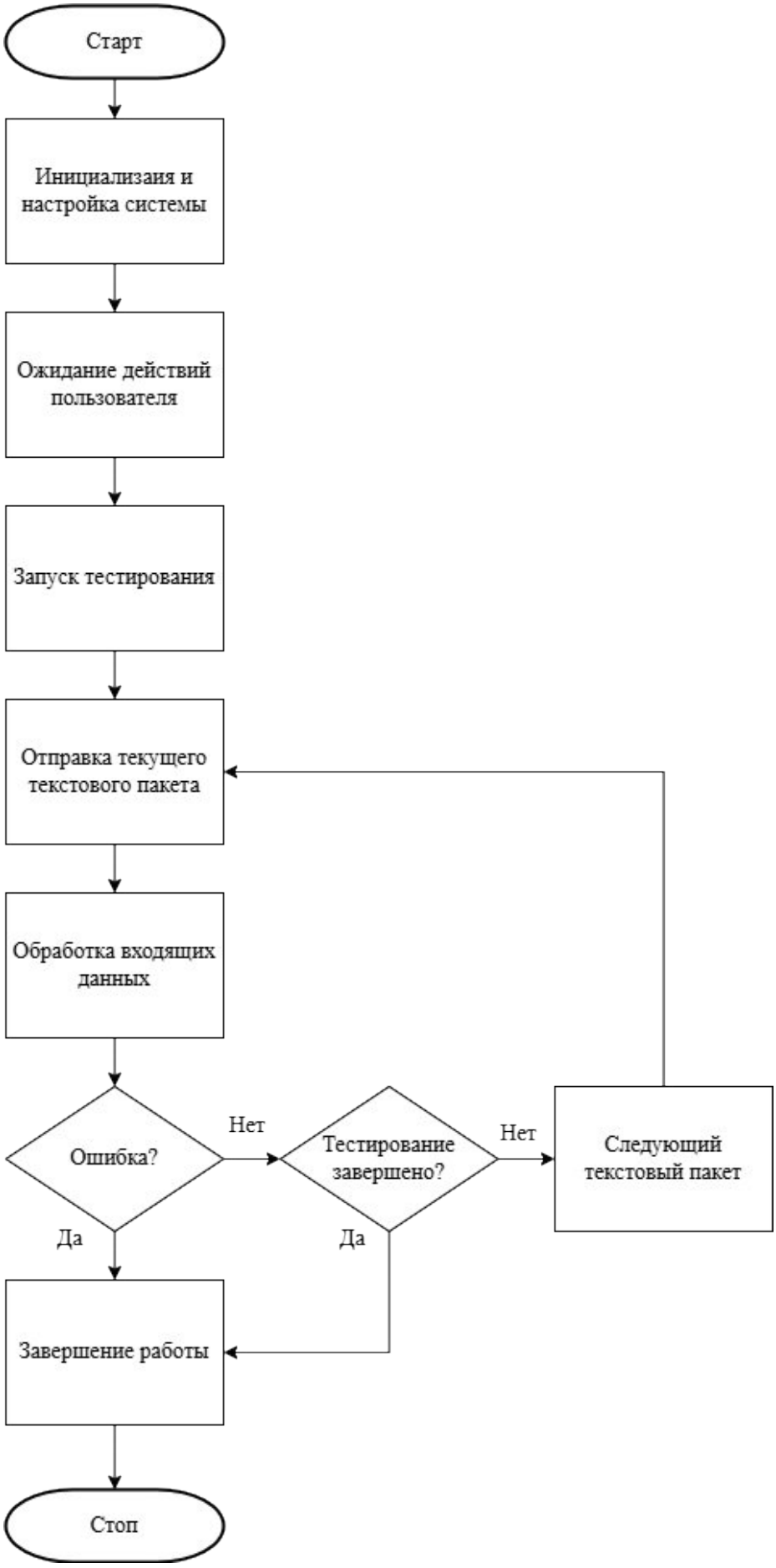

Рисунок 2.11 — Блок-схема алгоритма работы приложения на Windows

Ключевые особенности алгоритма:

- циклическая обработка пакетов — последовательная отправка и проверка каждого тестового случая;
- интерактивные элементы — диалоговые окна для подтверждения критических операций;
- возможность настройки портов — пользователь может указать предпочтительную скорость в бит/секунд для взаимодействия с платой тестового стенда (по умолчанию 115200 бит/секунд), так же пользователь сам может выбрать какой порт использовать для установления соединения с платой газоанализатора (по умолчанию 3000 порт).
- гибкое управление таймерами — контроль времени ожидания ответа и интервалов между командами;
- многоуровневое логирование — цветовая маркировка статусов операций.

Данный алгоритм представляет собой законченное решение для промышленного тестирования электронных устройств. Система успешно решает задачи контроля качества на производстве, существенно снижая вероятность выпуска бракованных изделий. Гибкая архитектура позволяет адаптировать алгоритм для тестирования других электронных устройств с минимальными доработками.

2.7 Конструкция тестового стенда

Общую концепцию тестового стенда можно понять по рисунку 2.12. Тестируемая печатная плата кладется на ложе погопинов (pogo-pin), подключенных к плате тестового стенда, и прижимается с помощью специального прижима.

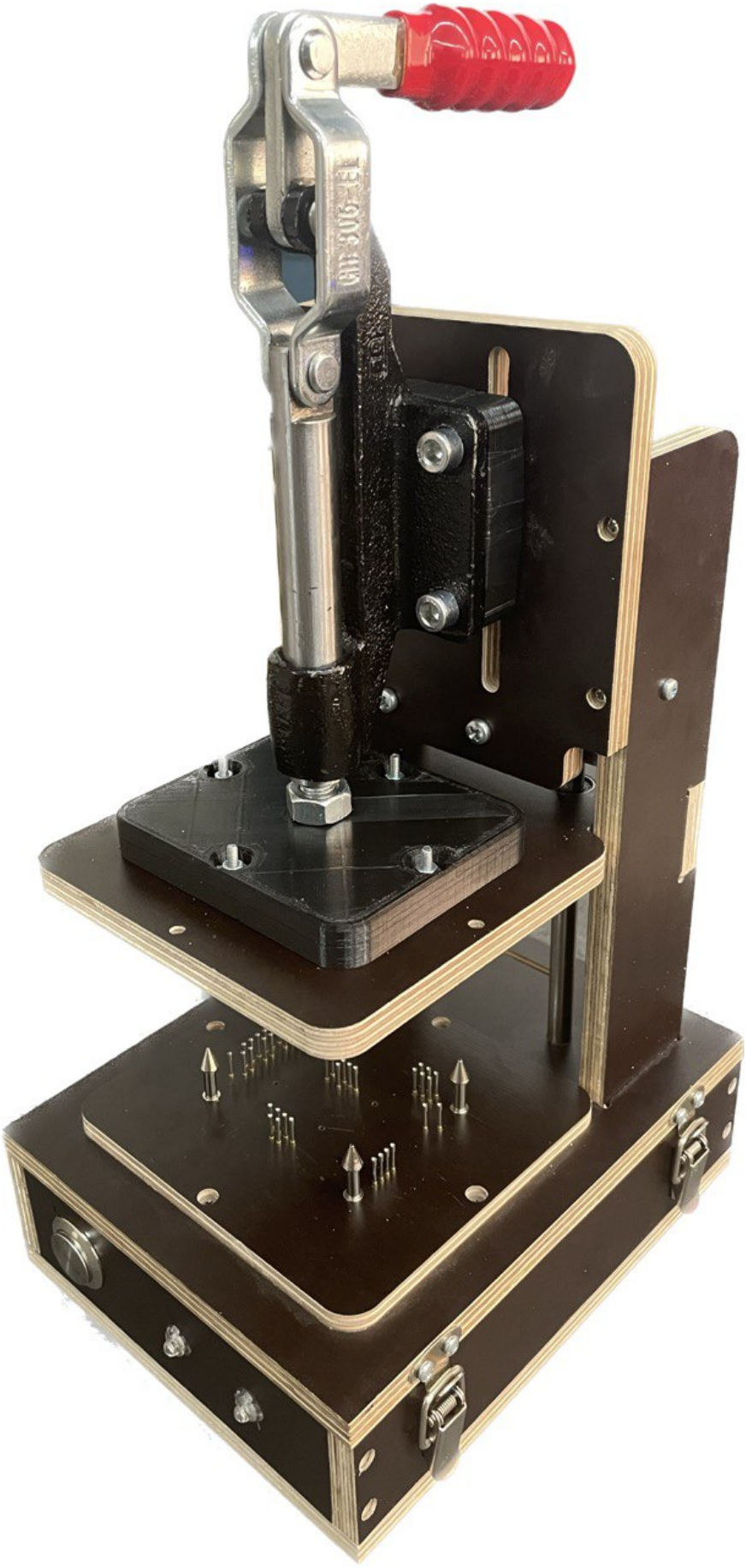

Рисунок 2.12 — Конструкция тестового стенда

Заключение

В результате выполнения выпускной квалификационной работы были достигнуты следующие результаты:

1) разработана прошивка для МК тестового стенда;
2) разработано графическое приложение для ПК;
3) проведено тестирование модулей на испытательном стенде;
4) произведена тестовая проверка печатной платы газоанализатора;
5) составлена документация.

В итоге, была разработана система, которая:

- проводит тестирование ПП газоанализатора, проверяя ключевые модули;
- гибкая настройка скорости и номера портов для подключения платы газоанализатора и платы тестового стенда к ПК;
- выводит удобное отображение статуса выполнения каждой команды в графическое приложение, с понятной цветовой маркировкой, а так же строит график.

Разработанная система:

- значительно увеличивает скорость тестирования ПП газоанализатора, по сравнению с ручным методом тестирования;
- по сравнению с методами тестирования, описанными в разделе «Литературный обзор», дешевле по себестоимости;
- имеет гибкую систему, которую можно быстро адаптировать к возможным изменениям на ПП газоанализатора.

Список использованных источников

1 Стенды функционального тестирования. — [Электронный документ], 2023. — URL: [https://thirdpin.io/stendy_testirovaniya_elektroniki_i_pechatnyh_plat] (19.05.2025).

2 Функциональное тестирование печатной платы. — [Электронный документ], 24 октября 2018. — URL: [https://habr.com/ru/companies/thirdpin/articles/425569/] (19.05.2025).

3 Электрическое тестирование печатных плат. — [Электронный документ], 16 июня 2013. — URL: [https://portal-ed.ru/index.php/testirovanie-i-kontrol/89-elektricheskoe-testirovanie-pechatnykh-plat] (19.05.2025).

4 Городов В. Методы электрического контроля печатных плат. — [Электронный документ], 18 марта 2005. — URL: [https://tech-e.ru/wp-content/uploads/2005_01_68.pdf] (19.05.2025).

5 Карпов С. В. Проблемы контроля многослойных печатных плат. М.: Радиотехника. 2003.

6 Приходько И. Тестирование печатных плат // Электронные компоненты. 2003. № 8

7 Тестирование печатных плат? «Это же просто». — [Электронный документ], 2 июля 2021. — URL: [https://habr.com/ru/companies/ruvds/articles/565526/] (19.05.2025).

8 Методы тестирования ИС, основанные на использовании оптического излучения. — [Электронный документ], 2025. — URL: [file:///C:/Users/user/Downloads/metody-testirovaniya.pdf] (19.05.2025).

9 Проектирование и изготовление тестовых стендов на базе адаптерных систем Ingun. — [Электронный документ], 2023. — URL: [https://thirdpin.io/uslugi/testirovanie-elektroniki/proektirovanie-testovyh-adapterov-ingun] (19.05.2025).

10 Datasheet - STM32F411xC STM32F411xE - Arm® Cortex®-M4 32b MCU+FPU, 125 DMIPS, 512KB Flash, 128KB RAM,

USB OTG FS, 11 TIMs, 1 ADC, 13 comm. interfaces. — [Электронный документ], 29 января 2024. — URL: [https://www.st.com/resource/en/datasheet/stm32f411ce.pdf] (19.05.2025).

11 STM32F411xC/E advanced Arm®-based 32-bit MCUs - Reference manual. — [Электронный документ], 1 ноября 2018. — URL: [https://www.st.com/resource/en/reference_manual/rm0383-stm32f411xce-advanced-armbased-32bit-mcus-stmicroelectronics.pdf] (19.05.2025).

12 Руководство по программированию с помощью фреймворка Qt и языка C++. — [Электронный документ], 15 января 2024. — URL: [https://metanit.com/cpp/qt/] (19.05.2025).

13 Язык программирования Си, Брайан Керниган, Деннис Ритчи. — [Электронный документ], 22 февраля 1978. — URL: [https://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf] (19.05.2025).

14 Программирование: введение в профессию. — Изд. 2-е, испр. и доп. : в 3 томах / А. В. Столяров. — Москва: МАКС Пресс, 2021 — 704 с. (19.05.2025).

15 Новиелло Кармин. Освоение STM32, 17 августа 2018 — 826 с. (19.05.2025).

16 Программирование на языке C++ в среде Qt Creator: / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк,О. В. Чеснокова, А. С. Чмыхало — М. : ALT Linux, 2015. — 448 с. — [Электронный документ], 2015. — URL: [https://www.altlinux.org/Images/4/4b/Book-qtC%2B%2B.pdf] (19.05.2025).

17 Программирование на языке C++ в среде Qt Creator: / Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк,О. В. Чеснокова, А. С. Чмыхало — М. : ALT Linux, 2015. — 448 с. — [Электронный документ], 2015. — URL: [https://www.altlinux.org/Images/4/4b/Book-qtC%2B%2B.pdf] (19.05.2025).

18 Страуструп Б. Программирование. Принципы и практика с использованием C++. — М.: Вильямс, 2021. — 1248 с. (19.05.2025).

19 Серия статей по STM32 (CubeMX, HAL, LL) — [Электронный документ], 2021. — URL: [https://controllerstech.com/] (19.05.2025).

20 Руководство пользователя HAL API — [Электронный документ], 2023. — URL: [https://www.st.com/en/embedded-software/stm32cube-mcu-packages.html] (19.05.2025).

21 Стенд для тестирования плат после монтажа. — [Электронный документ], 2022. — URL: [https://habr.com/ru/articles/686674/] (19.05.2025).

22 STM32 ADC Tutorial + ADC Examples [Ultimate Guide]. — [Электронный документ], 2023. — URL: [https://deepbluembedded.com/stm32-adc-tutorial-complete-guide-with-examples/] (19.05.2025).

23 STM32 UART (USART) Tutorial + Examples (DMA, Interrupt). — [Электронный документ], 2023. — URL: [https://deepbluembedded.com/stm32-usart-uart-tutorial/] (19.05.2025).

24 Git за полчаса: руководство для начинающих. — [Электронный документ], 2023. — URL: [https://proglib.io/p/git-for-half-an-hour] (19.05.2025).

25 Основы цифровой схемотехники. Базовые элементы и схемы. Методы проектирования. - М.: Мир, 2001. - 379 с. — [Электронный документ], 2001 — URL: [https://www.elec.ru/files/2020/02/25/_Novikov_YU.V.__Osnovue_cifrovoi_shemotehniki._Baz.PDF] (19.05.2025).

26 Урок 31. Работа с АЦП через функции библиотеки HAL. — [Электронный документ], 22 января 2021. — URL: [https://mypractic.ru/urok-31-rabota-s-acp-cherez-funkcii-biblioteki-hal.html] (19.05.2025).

Приложение А

(обязательное)

Исходный код модуля main

```
////////////////////////////////////////////////////////////////////////////
// Заголовочный файл main.h модуля инициализации системы и управления
// основным циклом выполнения алгоритма тестирования
// Инициализирует функцию обработчика ошибок, подключает библиотеки и модули
// parser и hardware
// Язык: C
// Среда разработки: STM32CubeIDE
// Компилятор: GCC
// Входные параметры: отсутствуют
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 18 марта 2025 г.
// Версия 0.1.3
////////////////////////////////////////////////////////////////////////////

#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

#include "stm32f4xx_hal.h"
#include "stm32f4xx_it.h"

#include "hardware.h"
#include "parser.h"

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>

void Error_Handler(void);

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */

////////////////////////////////////////////////////////////////////////////
// Основной файл main.c модуля инициализации системы и управления
// основным циклом выполнения алгоритма тестирования
// Инициализирует периферию, подключает библиотеки, взаимодействует с
// модулями обработки (parser) и управления периферией МК (hardware)
// Язык: C
// Среда разработки: STM32CubeIDE
// Компилятор: GCC
// Входные параметры: байты протокола передачи данных с ПК (rx_byte)
// Выходные параметры: массив байт протокола передачи данных на ПК (data.buf)
// Автор: Жеребцов А.А., группа РЭ3-11
```

```
// Модификация: 18 марта 2025 г.
// Версия 0.1.3
////////////////////////////////////////////////////////////////////////////

#include "main.h"

uint8_t rx_byte, status;

// Частные переменные
ADC_HandleTypeDef hadc1;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart6;

// Прототипы функций
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART6_UART_Init(void);
static void MX_ADC1_Init(void);

int main(void)
{
  // Выполняется сброс настроек всех периферийных устройств, инициализируется
интерфейс Flash и системный блок
  HAL_Init();

  // Настройка системных часов
  SystemClock_Config();

  // Инициализация всех периферийных устройств
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  MX_USART6_UART_Init();
  MX_ADC1_Init();

  while (1)
  {
       HAL_UART_Receive_IT(&UART_USB, &rx_byte, 1);
       if (status == STATUS_OK){
            enum parser_result result;
            result = process_rx_byte(&parser, rx_byte);
            if (result == PARSER_DONE) {
                choose_command(parser.buffer, &parser.buffer_length);
                transmission(&data, &parser);
                serialize_reply(&data);
                HAL_UART_Transmit_IT(&UART_USB, data.buf, data.buf_size);
            }
            else if (result == PARSER_ERROR) {
                      parser.state = STATE_SYNC;
            }
            status = STATUS_EXEC_ERROR;
       }
  }
```

```
void SystemClock_Config(void){
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  // Настройка выходного напряжения основного внутреннего регулятора
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  // Инициализация генераторов RCC в соответствии с заданными параметрами
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 144;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
      Error_Handler();
  }

  // Инициализизация тактов процессора и шин AHB и APB
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
      Error_Handler();
  }
}

static void MX_ADC1_Init(void)
{

  ADC_ChannelConfTypeDef sConfig = {0};

  /* Настройка глобальных функций АЦП (тактовая частота, разрешение,
  выравнивание данных и количество преобразований) */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SEQ_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
```

```
      Error_Handler();
  }

  /* Настройка для выбранного обычного канала АЦП его соответствующий ранг во
  время дискретизации */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
      Error_Handler();
  }

}

static void MX_USART1_UART_Init(void)
{
  // Настройка функций UART1
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
      Error_Handler();
  }
}

static void MX_USART2_UART_Init(void)
{
  // Настройка функций UART2
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
      Error_Handler();
  }
}

static void MX_USART6_UART_Init(void)
{
  // Настройка функций UART6
  huart6.Instance = USART6;
  huart6.Init.BaudRate = 115200;
  huart6.Init.WordLength = UART_WORDLENGTH_8B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
```

```
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart6) != HAL_OK)
  {
      Error_Handler();
  }
}

static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  // Включение синхронизации портов GPIO
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  // Настройка уровня выводов GPIO
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13|GPIO_PIN_14, GPIO_PIN_RESET);

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
                          |GPIO_PIN_8|GPIO_PIN_9, GPIO_PIN_RESET);

  // Настройка выводов GPIO: PC13 PC14
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_14;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  // Настройка выводов GPIO: PB4, PB5, PB6, PB7, PB8, PB9
  GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
                          |GPIO_PIN_8|GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  if (huart->Instance == USART2)
      {status = STATUS_OK;}
}

void Error_Handler(void)
{
  __disable_irq();
  while (1)
  {}
}

#ifdef  USE_FULL_ASSERT

void assert_failed(uint8_t *file, uint32_t line){}

#endif /* USE_FULL_ASSERT */
```

Приложение Б
(рекомендуемое)
Протокол передачи данных

Нужно установить в 4 значение параметра с кодом 1. Значение 4 в параметре с кодом 1, является командой измерений контрольной точки +6 вольт. МК тестового стенда провела измерение этой контрольной точки и вернула ответ (в милливольтах), со значением 1770h, что в десятичной системе равно 6000 милливольт, то есть 6 вольт.

Таблица Б.1 — Пример установки параметра

<table>
<tr><th>Байт</th><th>Значение</th><th colspan="2">Описание</th></tr>
<tr><td colspan="4">Запрос (AA 08 00 01 01 04 00 00 00 3D 3A) 11 Байт</td></tr>
<tr><td>0</td><td>0xAA</td><td colspan="2">Синхробайт</td></tr>
<tr><td>1</td><td>0x08</td><td>Младший байт размера полезных данных</td><td rowspan="2">В данном случаем = 8, т.к. Код команды — 1 байт, данные команды — 5 байт, CRC — 2 байта.</td></tr>
<tr><td>2</td><td>0x00</td><td>Старший байт размера полезных данных</td></tr>
<tr><td>3</td><td>0x01</td><td colspan="2">Код команды (1 — установка параметра).</td></tr>
<tr><td>4</td><td>0x01</td><td rowspan="5">Данные команды</td><td>Код параметра</td></tr>
<tr><td>5</td><td>0x04</td><td rowspan="4">Устанавливаемое значение.</td></tr>
<tr><td>6</td><td>0x00</td></tr>
<tr><td>7</td><td>0x00</td></tr>
<tr><td>8</td><td>0x00</td></tr>
<tr><td>9</td><td>0x3D</td><td colspan="2">Младший байт CRC</td></tr>
<tr><td>10</td><td>0x3A</td><td colspan="2">Старший байт CRC</td></tr>
<tr><td colspan="4">Ответ (AA 08 00 01 00 26 03 00 00 FA 82) 11 Байт</td></tr>
<tr><td>0</td><td>0xAA</td><td colspan="2">Синхробайт</td></tr>
</table>

Окончание таблицы Б.1

1	0x08	Младший байт размера полезных данных	В данном случаем = 8, т.к. Код команды — 1 байт, данные команды — 5 байт, CRC — 2 байта.
2	0x00	Старший байт размера полезных данных	
3	0x01	Код команды (1 – ответ на код 1)	
4	0x00	Байт ошибки (0 – ошибок нет)	
5	0x70	Четыре байта возвращаемого значения (от младшего к старшему байту). Если не было ошибок возвращается значение установленного параметра, в данном случае — 1770	
6	0x17		
7	0x00		
8	0x00		
9	0xFA	Младший байт CRC	
10	0x82	Старший байт CRC	

В таблице Б.2 приведен перечень команд, которые способен выполнять тестовый стенд.

Таблица Б.2 — Перечень команд тестового стенда

Код параметра	Описание функций	Диапазон	Устанавливаемый
0	Вкл/откл напряжение питания АЦМ	0…1	+
1	Проверка 4 контрольных точек: +6, -6, +5, +3.3 В	0…3	-
2	Проверка напряжения и тока питания АЦМ	0…1	-
3	Включение/отключение вторичного питания платы АЦМ	0…1	+
4	Проверка 12 контрольных точек: +1.2V, +1.8V, +2.5V, Power GPS (+5.5V), VrefADC (+4.5V), +5.5VA, -5.5VA, +1.8VA, Offset (+2.5V), Laser (+5V), VrefDAC (+2.048V)	0…10	-
5	Измерение формы сигнала лазера	—	-
6	Измерение напряжения элемента Пельтье	—	-
7	Переключить тип входных цепей на эквивалентные схемы/внешний оптический блок	0…1	+
8	Тестирование работы интерфейса RS232	—	-
9	Тестирование работы интерфейса подключения GPS-приемника	0…59	+

Приложение В

(обязательное)

Исходный код модуля parser

```
////////////////////////////////////////////////////////////////////////////
// Заголовочный файл parser.h модуля обработки протокольных пакетов и
// диспетчеризации команд
// Объявляет функции диспетчеризации и функции обработки протокола передачи
// данных. Подключает библиотеки, модули main и hardware, инициализирует
// константы, структуры для протокола передачи данных
// Язык: C
// Среда разработки: STM32CubeIDE
// Компилятор: GCC
// Входные параметры: отсутствуют
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 5 марта 2025 г.
// Версия 0.1.4
////////////////////////////////////////////////////////////////////////////

#ifndef PARSER_H
#define PARSER_H

#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "hardware.h"
#include "main.h"

#define MAX_DATA_SIZE 201 // Максимальный размер буфера.
#define SYNC_BYTE 0xAA // Синхробайт.
#define DATA_SIZE_OFFSET 3 // 2 байта crc + код команды.
#define SIZE_PAKET 7 /* синхробайт + 2 байта полезных данных + cmd + status +
2 CRC. */
#define CRC_INIT 0xffff // для подсчета контрольной суммы CRC.

//коды ошибок
#define STATUS_OK 0 // Ошибок нет.
#define STATUS_EXEC_ERROR 1 // Ошибка выполнения команды.
#define STATUS_INVALID_CMD 2 // Несуществующая команда.
#define STATUS_TIMED_OUT 3 // Превышено время выполнения команды.
#define STATUS_INVALID_SIZE 4 // Ошибка размера данных команды.

//Тестовые команды для отладки протокола
#define APPLY_VOLTAGE_RL1 0 /* команда подать лог. 0 или 1 на RL1 для замыкания
цепи питания 12В. */
#define TEST_VOLTAGE_4_POINT 1 /* команда проверка напр. в 4 контрольных точках
+6 -6 +5 +3.3В. */
#define ANALYSIS_VOLTAGE_CORRENT 2 // команда измерение напр. и тока питания.
#define APPLY_VOLTAGE_RL2 3 /* Команда лог. 0 или 1 на RL2 для замыкания R1 и
R22. */
#define TEST_VOLTAGE_11_POINT 4 // команда проверки напр. в 12 контр. точках.
#define TEST_CORRENT_LASER 5 // Команда измерения формы тока лазерного диода.
```

```
#define TEST_VOLTAGE_PELTIE 6 // Команда измерения напряжения элемента Пельтье.
#define APPLY_VOLTAGE_5_RL 7 /* команда подать лог. 0 или 1 для РАЗМЫКАНИЯ RL3-
RL7. */
#define MASSAGE_RS232 8 // команда отправки заготовленного пакета по RS232.
#define MASSAGE_NMEA 9 /* команда отправки заготовленных пакетов NMEA на GPS
через RS232. */

enum parser_result {
    PARSER_OK,
    PARSER_ERROR,
    PARSER_DONE,
};

struct protocol_parser {
    enum {
        STATE_SYNC,
        STATE_SIZE_L,
        STATE_SIZE_H,
        STATE_CMD,
        STATE_DATA,
        STATE_CRC_L,
        STATE_CRC_H,
    } state;

    uint8_t buffer[MAX_DATA_SIZE]; // Буфер для хранения данных пакета.
    size_t buffer_length;          // Количество принятых байт данных.

   // Размер полезных данных, полученный из пакета с учетом DATA_SIZE_OFFSET.
    uint16_t data_size;

    uint8_t cmd;                   // Команда пакета.
    uint16_t crc;                  // Накопленная контрольная сумма.
};

struct for_transfer
{
    uint8_t* buf;    // Массив с пакетом байтов.
    size_t buf_size; // Размер массива buf.
    uint8_t cmd;     // Код команды.
    uint8_t status;  // Код ошибки.
    uint8_t* value;  // Данные команды.
};

struct value_range {
    uint32_t min;
    uint32_t max;
    uint32_t voltage_4[4];
    uint32_t voltage_11[11];
};

 static const struct value_range VALUE_RANGES[] = {
   [APPLY_VOLTAGE_RL1] = {.min = 0, .max = 1},
   [TEST_VOLTAGE_4_POINT] = {.min = 1, .max = 4, .voltage_4 = {6, 3, 5, 6}},
   [ANALYSIS_VOLTAGE_CORRENT] = {.min = 0, .max = 1},
   [APPLY_VOLTAGE_RL2] = {.min = 0, .max = 1},
```

```
    [TEST_VOLTAGE_11_POINT] = {.min = 0, .max = 10, .voltage_11 = {1200, 1800,
    2500, 5500, 4500, 5500, 5500, 1800, 2500, 5000, 2048}},
    [TEST_CORRENT_LASER] = {1},
    [TEST_VOLTAGE_PELTIE] = {1},
    [APPLY_VOLTAGE_5_RL] = {.min = 0, .max = 1},
    [MASSAGE_RS232] = {1},
    [MASSAGE_NMEA] =  {.min = 0, .max = 59},
};

extern struct protocol_parser parser;
extern struct for_transfer data;

// Функция для получения пакета с командой
enum parser_result process_rx_byte(struct protocol_parser *parser, uint8_t
byte);

// Функция для формирования пакета с данными
void serialize_reply(struct for_transfer* data);

// Функия для выбора команды
void choose_command(uint8_t* buffer, size_t* buffer_length);

// Функция для записи из for_transfer во for_receiving
void transmission(struct for_transfer* data, struct protocol_parser* parser);

#endif /* INC_PARSER_H_ */

////////////////////////////////////////////////////////////////////////////
// Основной файл parser.c модуля обработки протокольных пакетов и
// диспетчеризации команд
// Реализация функций передачи, принятия пакетов протокола передачи данных.
// Реализация функции диспетчеризации команд, инициализация массива для
// корректной работы протокола передачи данных и функции подсчета двух байт
// CRC
// Язык: C
// Среда разработки: STM32CubeIDE
// Компилятор: GCC
// Входные параметры: байты протокола передачи данных с ПК (rx_byte), массив
// полученного пакета (parser.buffer)
// Выходные параметры: массив байт протокола передачи данных на ПК (data.buf)
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 5 марта 2025 г.
// Версия 0.1.4
////////////////////////////////////////////////////////////////////////////

#include "parser.h"

struct protocol_parser parser;
struct for_transfer data;
static const uint16_t crc16_table[256]; /* Инициализая массива для счета
регистра CRC */

// Две функции для обновления значения регистра CRC
static uint16_t update_crc(uint16_t crc, uint8_t byte)
{
  return crc16_table[(crc ^ byte) & 0xFF] ^ (crc >> 8);
}
```

```
static uint16_t calculate_crc(const uint8_t* array, int size) {
  uint16_t crc = CRC_INIT;
  for (int i = 0; i < size; i++) {
       crc = update_crc(crc, array[i]);
  }
  return crc;
}

void serialize_reply(struct for_transfer* data) {
  uint16_t crc;
  static uint16_t PAYLOAD_SIZE;

  PAYLOAD_SIZE = data->buf_size - 6;
  data->buf[0] = SYNC_BYTE;
  data->buf[1] = ((PAYLOAD_SIZE + DATA_SIZE_OFFSET) >> 0) & 0xff;
  data->buf[2] = ((PAYLOAD_SIZE + DATA_SIZE_OFFSET) >> 8) & 0xff;
  data->buf[3] = data->cmd;
  data->buf[4] = data->status;
  int a = 0;
  for (int i = 5; i < data->buf_size - 2; i++)
  {
       data->buf[i] = data->value[a];
       a++;
  }
  crc = calculate_crc(data->buf + 3, PAYLOAD_SIZE + 1);
  data->buf[data->buf_size - 2] = (crc >> 0) & 0xff;
  data->buf[data->buf_size - 1] = (crc >> 8) & 0xff;

}

enum parser_result process_rx_byte(struct protocol_parser *parser, uint8_t
byte) {
  enum parser_result ret = PARSER_OK;

  switch (parser->state) {
      case STATE_SYNC:
          if (byte == SYNC_BYTE) {
              parser->state = STATE_SIZE_L;
              parser->crc = CRC_INIT;
              parser->buffer_length = 0;
          }
          break;
      case STATE_SIZE_L:
          parser->data_size = byte;
          parser->state = STATE_SIZE_H;
          break;
      case STATE_SIZE_H:
          parser->data_size |= ((uint16_t)byte << 8);
          if (parser->data_size >= DATA_SIZE_OFFSET &&
              parser->data_size <= MAX_DATA_SIZE + DATA_SIZE_OFFSET) {
              parser->state = STATE_CMD;
          }
          else{
              parser->state = STATE_SYNC;
              ret = PARSER_ERROR;
          }
          break;
      case STATE_CMD:
```

```
            parser->crc = update_crc(parser->crc, byte);
            parser->cmd = byte;
            parser->state = (parser->data_size != DATA_SIZE_OFFSET) ?
        STATE_DATA :        STATE_CRC_L;
            break;
        case STATE_DATA:
            parser->crc = update_crc(parser->crc, byte);
            parser->buffer[parser->buffer_length++] = byte;
            if (parser->buffer_length + DATA_SIZE_OFFSET >= parser->data_size)
            {
                parser->state = STATE_CRC_L;
            }
            break;
        case STATE_CRC_L:
            parser->crc = update_crc(parser->crc, byte);
            parser->state = STATE_CRC_H;
            break;
        case STATE_CRC_H:
            parser->crc = update_crc(parser->crc, byte);
            parser->state = STATE_SYNC;
            ret = (parser->crc == 0 ? PARSER_DONE : PARSER_ERROR);
            break;
    }
    return ret;
}

void choose_command(uint8_t* buffer, size_t* buffer_length)
{
    switch (buffer[0]){
        case APPLY_VOLTAGE_RL1:
            apply_voltage_relay_1(buffer);
            break;
        case TEST_VOLTAGE_4_POINT:
            test_voltage_4_point(buffer);
            break;
        case ANALYSIS_VOLTAGE_CORRENT:
            test_voltage_current(buffer);
            break;
        case APPLY_VOLTAGE_RL2:
            apply_voltage_relay_2(buffer);
            break;
        case TEST_VOLTAGE_11_POINT:
            test_voltage_11_point(buffer);
            break;
        case TEST_CORRENT_LASER:
            *buffer_length = 201;
            test_corrent_laser(buffer);
            break;
        case TEST_VOLTAGE_PELTIE:
            *buffer_length = 5;
            test_voltage_peltie(buffer);
            break;
        case APPLY_VOLTAGE_5_RL:
            apply_voltage_relay_5(buffer);
            break;
        case MASSAGE_RS232:
            *buffer_length = 5;
            massage_rs232(buffer);
```

```
            break;
        case MASSAGE_NMEA:
            massage_gps(buffer);
            break;
    }
}
void transmission(struct for_transfer* data, struct protocol_parser* parser)
{
    data->buf_size = 6 + parser->buffer_length;
    data->cmd = parser->cmd;
    data->status = parser->buffer[0];
    data->value = (uint8_t*)malloc((parser->buffer_length - 1) *
sizeof(uint8_t));
    if (data->value == NULL)
    {
        return;

    }
    for (size_t i = 0; i < parser->buffer_length - 1; i++)
    {
         data->value[i] = parser->buffer[i + 1];
    }
    data->buf = (uint8_t*)malloc(data->buf_size * sizeof(uint8_t));
    if (data->buf == NULL)
    {
        return;
    }
}
static const uint16_t crc16_table[256] =
{
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
```

```
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040,
};
```

Приложение Г
(обязательное)
Исходный код модуля hardware

```
////////////////////////////////////////////////////////////////////////////
// Заголовочный файл hardware.h модуля управления и взаимодействием с
// периферией МК
// Объявляет функции аппаратного управления переферией МК, подключает
// библиотеки, подключает модули main и parser, устанавливает константы для
// гибкой настройки параметров работы алгоритма
// Язык: C
// Среда разработки: STM32CubeIDE
// Компилятор: GCC
// Входные параметры: отсутствуют
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 15 мая 2025 г.
// Версия 0.2.7
////////////////////////////////////////////////////////////////////////////

#ifndef HARDWARE_H
#define HARDWARE_H

#include "main.h"
#include "parser.h"

#include <stdint.h>
#include <string.h>

#include "stm32f411xe.h"
#include "stm32f4xx_hal.h"

// директивы реле:
#define CLOSE_RELAY 0
#define OPEN_RELAY 1

// директивы 4 контрольных точек (test_voltage_4_point):
#define CHECKPOINT_6V_NEG 1
#define CHECKPOINT_3_3V 2
#define CHECKPOINT_5V 3
#define CHECKPOINT_6V 4

// директивы измерения тока или напряжения (test_voltage_current):
#define SYPPLY_VOLTAGE 0
#define SUPPLY_CURRENT 1

// директивы 11 контрольных точек (test_voltage_11_point):
#define CHECKPOINT_1_2V 0
#define CHECKPOINT_1_8V 1
#define CHECKPOINT_2_5V 2
#define CHECKPOINT_GPS_5_5V 3
#define CHECKPOINT_VREF_ADC_4_5V 4
#define CHECKPOINT_5_5VA 5
#define CHECKPOINT_5_5VA_NEG 6
#define CHECKPOINT_1_8VA 7
```

```
#define CHECKPOINT_OFFSET_2_5V 8
#define CHECKPOINT_LASER_5V 9
#define CHECKPOINT_VREF_DAC_2_048V 10

//директивы настройки параметров

// директивы установки битов GPIO (для реле):
#define RELAY_1_PIN_1 (1 << 14) // PC14
#define RELAY_1_PIN_0 (1 << 30)

#define RELAY_2_PIN_1 (1 << 9) // PB9
#define RELAY_2_PIN_0 (1 << 25)

#define RELAY_5_PIN_1 (1 << 13) // PC13
#define RELAY_5_PIN_0 (1 << 29)

#define MUX_EN (1 << 4) // включение мультиплексора.
#define MUX_DIS (1 << 20) // выключение мультиплексора.

// директивы каналов АЦП
#define ADC_LASER ADC_CHANNEL_0 /* Канал АЦП для измерения формы тока лазерного
диода. */
#define ADC_MUX ADC_CHANNEL_1 // Канал АЦП для мультиплексора 1:16.
#define ADC_SYPPLY_VOLTAGE ADC_CHANNEL_4 /* Канал АЦП для измерения напряжения
питания. */
#define ADC_SUPPLY_CURRENT ADC_CHANNEL_5 /* Канал АЦП для измерения тока
питания. */
#define ADC_PELTIE_1 ADC_CHANNEL_6 /* Канал АЦП для измерения падения
напряжения элемента Пельтье. */
#define ADC_PELTIE_2 ADC_CHANNEL_7 /* Канал АЦП для измерения падения
напряжения элемента Пельтье. */

#define RELAY_PORT_C GPIOC // выбор порта ножек GPIO для реле.
#define RELAY_PORT_B GPIOB // выбор порта ножек GPIO для реле.
#define REFERENCE_VOLTAGE 3300 // опорное напряжение, мВ.
#define ADC_BIT_RATE 4095 // разрешение АЦП.
#define RES_SHUNT_POWER 100 /* номинал шунтирующего резистора для тока питания,
мОм. */
#define RES_SHUNT_PELTIE 3000 /* номинал шунтирующего резистора для тока
Пельте, мОм. */
#define SAMPLES_LASER 100 // кол-во измерений сигнала лазерного диода.
#define SAMPLES 100 // кол-во измерений контрольной точки.
#define TIMEOUT_RX 9000 // время передачи пакета команды, мс.
#define TIME_ADC 9000 // время измерения АЦП, мс.
#define UART_USB huart2 // для выбора uart (выставить huart2).
#define UART_RS_232 huart1 // для выбора uart.
#define UART_GPS huart1 // для выбора uart (выставить huart6).
#define RS_232 7 // размер передаваемого массива функции massage_rs232.
#define GPS_SIZE 49 // размер передаваемого массива функции massage_gps.

// функции подсчета переменных:

// функция для выбора 1 из 16 выходов мультиплексора.
void set_pins( uint8_t a3, uint8_t a2, uint8_t a1, uint8_t a0 );

// функция снятие напряжения с контрольной точки.
```

```
void test_voltage(uint8_t* buf, uint32_t channel);

// функция выставляет лог. 0 или лог. 1 на ножку GPIO (Для реле).
void apply_relay(GPIO_TypeDef *PORT, uint32_t PIN);

// функция для сравнения "эталонного" массива с полученным массивом.
int compare_arrays(uint8_t arr1[], uint8_t arr2[], size_t size);
// функция для передачи/приема байт по uart.
void uart_tx_rx(UART_HandleTypeDef* uart, uint8_t* buf, uint8_t* tx, uint8_t*
rx, size_t size);

// управляющие функции:
void apply_voltage_relay_1(uint8_t* buf);
void test_voltage_4_point(uint8_t* buf);
void test_voltage_current(uint8_t* buf);
void apply_voltage_relay_2(uint8_t* buf);
void test_voltage_11_point(uint8_t* buf);
void test_corrent_laser(uint8_t* buf);
void test_voltage_peltie(uint8_t* buf);
void apply_voltage_relay_5(uint8_t* buf);
void massage_rs232(uint8_t* buf);
void massage_gps(uint8_t* buf);

#endif /* INC_HARDWARE_H_ */

////////////////////////////////////////////////////////////////////////////
// Основной файл hardware.c модуля управления и взаимодействием с
// периферией МК
// Модуль проводящий измерения платы газоанализатора, а так же управлением
// реле и мультиплексора 1:16.
// Язык: C
// Среда разработки: STM32CubeIDE
// Компилятор: GCC
// Входные параметры: массив байт полученного через протокол передачи данных
// (parser.buffer)
// Выходные параметры: массив байт протокола передачи данных на ПК (data.buf)
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 15 мая 2025 г.
// Версия 0.2.7
////////////////////////////////////////////////////////////////////////////

#include "hardware.h"

extern ADC_HandleTypeDef hadc1;
extern UART_HandleTypeDef huart1;
extern UART_HandleTypeDef huart2;
extern UART_HandleTypeDef huart6;

uint16_t vol_raw;
uint32_t vol_average, tok;

// функции подсчета переменных

void set_pins( uint8_t a3, uint8_t a2, uint8_t a1, uint8_t a0 ){
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, a3 ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, a2 ? GPIO_PIN_SET : GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, a1 ? GPIO_PIN_SET : GPIO_PIN_RESET);
```

```
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, a0 ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

void test_voltage(uint8_t* buf, uint32_t channel){
  vol_average = 0;
  ADC_ChannelConfTypeDef sConfig = {0};
  sConfig.Channel = channel;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
      buf[0] = STATUS_EXEC_ERROR;
      return;
  }
  HAL_ADC_Start(&hadc1);
  for (int i = 0; i < SAMPLES; i++) {
       while (!LL_ADC_IsActiveFlag_EOCS(ADC1)) {}
              LL_ADC_ClearFlag_EOCS(ADC1);
              vol_raw = HAL_ADC_GetValue(&hadc1);
              vol_average += vol_raw;
  }
  HAL_ADC_Stop(&hadc1);

  vol_average = vol_average * REFERENCE_VOLTAGE / (ADC_BIT_RATE * SAMPLES);
  buf[1] = (uint8_t)(vol_average & 0xFF);
  buf[2] = (uint8_t)((vol_average >> 8) & 0xFF);
  buf[3] = (uint8_t)((vol_average >> 16) & 0xFF);
  buf[4] = (uint8_t)((vol_average >> 24) & 0xFF);
  buf[0] = STATUS_OK;
  return;
}

void apply_relay(GPIO_TypeDef *PORT, uint32_t PIN){
  return(SET_BIT(PORT->BSRR, PIN));
}

void uart_tx_rx(UART_HandleTypeDef* uart, uint8_t* buf, uint8_t* tx, uint8_t*
rx, size_t size){
  if (HAL_UART_Transmit(uart, tx, size, TIMEOUT_RX) != HAL_OK) {
      buf[0] = STATUS_TIMED_OUT;
       return;
  }
  if (uart->RxState == HAL_UART_STATE_READY) {
      if (HAL_UART_Receive(uart, rx, size, TIMEOUT_RX) != HAL_OK) {
                           buf[0] = STATUS_TIMED_OUT;
                           return; }
  }
  else{
       HAL_UART_AbortReceive(uart);
       if (HAL_UART_Receive(uart, rx, size, TIMEOUT_RX) != HAL_OK) {
      buf[0] = STATUS_TIMED_OUT;
      return;
       }
  }
}

int compare_arrays(uint8_t arr1[], uint8_t arr2[], size_t size){
  for(int i = 0; i < size; i++){
```

```
  if (arr1[i] != arr2[i]){
      return 1;
  }
  }
  return 0;
}

// функции управления

void apply_voltage_relay_1(uint8_t* buf) // PC14
{
  switch (buf[1]) {
      case CLOSE_RELAY:
          apply_relay(RELAY_PORT_C, RELAY_1_PIN_0);
          if (READ_BIT(RELAY_PORT_C->IDR, RELAY_1_PIN_1) != 0){
              buf[0] = STATUS_EXEC_ERROR;
          }
          else {
              buf[0] = STATUS_OK;
          }
          return
      case OPEN_RELAY:
          apply_relay(RELAY_PORT_C, RELAY_1_PIN_1);
          if (READ_BIT(RELAY_PORT_C->IDR, RELAY_1_PIN_1) != 0){
              buf[0] = STATUS_OK;
          }
          else {
              buf[0] = STATUS_EXEC_ERROR;
          }
          return;
      default:
          buf[0] = STATUS_INVALID_CMD;
          return;
  }
}

void test_voltage_4_point(uint8_t* buf)
{
  apply_relay(GPIOB, MUX_EN);
  switch (buf[1]){
      case CHECKPOINT_6V_NEG:
          set_pins(1, 0, 0, 0);
          break;

      case CHECKPOINT_3_3V:
          set_pins(0, 1, 1, 0);
          break;

      case CHECKPOINT_5V:
          set_pins(0, 0, 1, 1);
          break;

      case CHECKPOINT_6V:
          set_pins(0, 0, 0, 0);
          break;
      default:
          buf[0] = STATUS_INVALID_CMD;
          return;
```

```
  }
  test_voltage(buf, ADC_MUX);
  apply_relay(GPIOB, MUX_DIS);
}

void test_voltage_current(uint8_t* buf)
{
  switch (buf[1]){
      case SYPPLY_VOLTAGE:
          uint32_t channel = ADC_SYPPLY_VOLTAGE;
          test_voltage(buf, channel);
          return;

      case SUPPLY_CURRENT:
          vol_average = 0;
          uint32_t res_shunt = RES_SHUNT_POWER; /* шунтирующий резистор для
измерения тока 100 mOm */
          ADC_ChannelConfTypeDef sConfig = {0};
          sConfig.Channel = ADC_SUPPLY_CURRENT;
          sConfig.Rank = 1;
          sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;
          if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
          {
              buf[0] = STATUS_EXEC_ERROR;
              return;
          }

          HAL_ADC_Start(&hadc1);
          for (int i = 0; i < SAMPLES; i++) {
               while (!LL_ADC_IsActiveFlag_EOCS(ADC1)) {}
               LL_ADC_ClearFlag_EOCS(ADC1);
               vol_raw = HAL_ADC_GetValue(&hadc1);
               vol_average += vol_raw;
          }

          HAL_ADC_Stop(&hadc1);
          vol_average = vol_average * REFERENCE_VOLTAGE / (ADC_BIT_RATE *
SAMPLES);

          tok = vol_average / res_shunt;
          buf[1] = (uint8_t)(tok & 0xFF);
          buf[2] = (uint8_t)(tok >> 8 & 0xFF);
          buf[3] = (uint8_t)(tok >> 16 & 0xFF);
          buf[4] = (uint8_t)(tok >> 24 & 0xFF);
          buf[0] = STATUS_OK;
          return;

      default:
          buf[0] = STATUS_INVALID_CMD;
          return;
  }
}

void apply_voltage_relay_2(uint8_t* buf) // PB9
{
  switch (buf[1]){
      case CLOSE_RELAY:
          apply_relay(RELAY_PORT_B, RELAY_2_PIN_0);
```

```
            if(READ_BIT(RELAY_PORT_B->IDR, RELAY_2_PIN_1) != 0){
               buf[0] = STATUS_EXEC_ERROR;
            }
            else{
               buf[0] = STATUS_OK;
            }
            return;
        case OPEN_RELAY:
            apply_relay(RELAY_PORT_B, RELAY_2_PIN_1);
            if (READ_BIT(RELAY_PORT_B->IDR, RELAY_2_PIN_1) != 0){
                buf[0] = STATUS_OK;
            }
            else{
                buf[0] = STATUS_EXEC_ERROR;
            }
            return;
        default:
            buf[0] = STATUS_INVALID_CMD;
            return;
    }
}

void test_voltage_11_point(uint8_t* buf)
{
    apply_relay(GPIOB, MUX_EN);
    switch (buf[1]){
        case CHECKPOINT_1_2V:
            set_pins(1, 0, 1, 1);
            break;

        case CHECKPOINT_1_8V:
            set_pins(1, 1, 0, 0);
            break;

        case CHECKPOINT_2_5V:
            set_pins(1, 1, 1, 0);
            break;

        case CHECKPOINT_GPS_5_5V:
            set_pins(0, 0, 0, 1);
            break;

        case CHECKPOINT_VREF_ADC_4_5V:
            set_pins(0, 1, 0, 0);
            break;

        case CHECKPOINT_5_5VA:
            set_pins(0, 1, 0, 1);
            break;

        case CHECKPOINT_5_5VA_NEG:
            set_pins(1, 0, 0, 1);
            break;

        case CHECKPOINT_1_8VA:
            set_pins(1, 0, 1, 0);
            break;
```

```
        case CHECKPOINT_OFFSET_2_5V:
            set_pins(1, 1, 0, 1);
            break;

        case CHECKPOINT_LASER_5V:
            set_pins(0, 0, 1, 0);
            break;

        case CHECKPOINT_VREF_DAC_2_048V:
            set_pins(1, 1, 1, 1);
            break;

        default:
            buf[0] = STATUS_INVALID_CMD;
            return;
    }
    test_voltage(buf, ADC_MUX);
    apply_relay(GPIOB, MUX_DIS);
}

void test_corrent_laser(uint8_t* buf)
{
    uint16_t adcSamples[SAMPLES_LASER];
    ADC_ChannelConfTypeDef sConfig = {0};
    sConfig.Channel = ADC_LASER;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
    {
        buf[0] = STATUS_EXEC_ERROR;
        return;
    }
    HAL_ADC_Start(&hadc1);
    for (int i = 0; i < SAMPLES_LASER; i++) {
         while (!LL_ADC_IsActiveFlag_EOCS(ADC1)) {}
         LL_ADC_ClearFlag_EOCS(ADC1);
         adcSamples[i] = HAL_ADC_GetValue(&hadc1);
    }
    HAL_ADC_Stop(&hadc1);
    for (int i = 0; i < SAMPLES_LASER; i++){
         vol_average = adcSamples[i] * REFERENCE_VOLTAGE / ADC_BIT_RATE;
         buf[i * 2 + 1] = (uint8_t)(vol_average & 0xFF);
         buf[i * 2 + 2] = (uint8_t)(vol_average >> 8 & 0xFF);
    }
    buf[0] = STATUS_OK;
}

void test_voltage_peltie(uint8_t* buf)
{
    int32_t vol_raw, tok = 0, vol_average_1 = 0, vol_average_2 = 0;
    int32_t res_shunt = RES_SHUNT_PELTIE;
    ADC_ChannelConfTypeDef sConfig = {0};

    sConfig.Channel = ADC_PELTIE_1;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_15CYCLES;

    if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
```

```
  {
      buf[0] = STATUS_EXEC_ERROR;
      return;
  }

  HAL_ADC_Start(&hadc1);
  for (int i = 0; i < SAMPLES; i++) {
       while (!LL_ADC_IsActiveFlag_EOCS(ADC1)) {}
       LL_ADC_ClearFlag_EOCS(ADC1);
       vol_raw = HAL_ADC_GetValue(&hadc1);
       vol_average_1 += vol_raw;
  }

  HAL_ADC_Stop(&hadc1);
  vol_average_1 = vol_average_1 * REFERENCE_VOLTAGE / (ADC_BIT_RATE *
SAMPLES);

  sConfig.Channel = ADC_PELTIE_2;

  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
      buf[0] = STATUS_EXEC_ERROR;
      return;
  }

  HAL_ADC_Start(&hadc1);
  for (int i = 0; i < SAMPLES; i++) {
       while (!LL_ADC_IsActiveFlag_EOCS(ADC1)) {}
       LL_ADC_ClearFlag_EOCS(ADC1);
       vol_raw = HAL_ADC_GetValue(&hadc1);
       vol_average_2 += vol_raw;
  }

  HAL_ADC_Stop(&hadc1);
  vol_average_2 = vol_average_2 * REFERENCE_VOLTAGE / (ADC_BIT_RATE *
SAMPLES);

  tok = vol_average_1 - vol_average_2;
  buf[1] = (uint8_t)(tok & 0xFF);
  buf[2] = (uint8_t)(tok >> 8 & 0xFF);
  if (tok > 0) {
      buf[0] = STATUS_EXEC_ERROR;
      tok = (tok * 1000000) / res_shunt; // мкА
  }
  else{
      buf[0] = STATUS_OK;
      tok = (tok * 1000000) / res_shunt; // мкА
  }
  buf[3] = (uint8_t)(tok >> 16 & 0xFF);
  buf[4] = (uint8_t)(tok >> 24 & 0xFF);
  return;

}

void apply_voltage_relay_5(uint8_t* buf) // PC13
{
  switch (buf[1]) {
      case CLOSE_RELAY:
```

```
            apply_relay(RELAY_PORT_C, RELAY_5_PIN_0);
            if (READ_BIT(RELAY_PORT_C->IDR, RELAY_5_PIN_1) != 0){
                buf[0] = STATUS_EXEC_ERROR;
            }
            else{
                buf[0] = STATUS_OK;
            }
            return;
        case OPEN_RELAY:
            apply_relay(RELAY_PORT_C, RELAY_5_PIN_1);
            if (READ_BIT(RELAY_PORT_C->IDR, RELAY_5_PIN_1) != 0){
                buf[0] = STATUS_OK;
            }
            else{
                buf[0] = STATUS_EXEC_ERROR;
            }
            return;
        default:
            buf[0] = STATUS_INVALID_CMD;
            return;
    }
}

void massage_rs232(uint8_t* buf)
{
    uint8_t rs_232_tx [RS_232] = "RS_232!";
    uint8_t rs_232_rx [RS_232];
    uart_tx_rx(&UART_RS_232, buf, rs_232_tx, rs_232_rx, RS_232);

    if (buf[0] == STATUS_TIMED_OUT){ return; }

    if (compare_arrays(rs_232_tx, rs_232_rx, RS_232) == 0){
        buf[0] = STATUS_OK;
    }
    else{
        buf[0] = STATUS_EXEC_ERROR;
    }
    for (int i = 1; i < 5; i++){
         buf[i] = 0;
    }
}

void massage_gps(uint8_t* buf)
{
    uint8_t gps_tx [GPS_SIZE] =
"$GNGLL,5502.49000,N,08256.07600,E,1235  .000,A,A*"; /* GLL, version 4.1 and
4.2, NMEA 0183 */
    uint8_t gps_rx [GPS_SIZE];
    gps_tx[38] = (buf[1]/ 10) + '0';
    gps_tx[39] = (buf[1] % 10) + '0';
    uart_tx_rx(&UART_GPS, buf, gps_tx, gps_rx, GPS_SIZE);

    if (buf[0] == STATUS_TIMED_OUT){ return; }

    if (compare_arrays(gps_tx, gps_rx, GPS_SIZE) == 0){
        buf[0] = STATUS_OK;
    }
    else{
```

```
        buf[0] = STATUS_EXEC_ERROR;
    }
}
```

Приложение Д

(рекомендуемое)

Алгоритм тестирования платы газоанализатора

1. Снять напряжение питание с платы (отправить стенду команду размыкания RL1).
2. Переключить тип входных цепей на эквивалентные схемы (отправить стенду команду размыкания RL3-RL7).
3. Отключить вторичное питание платы АЦМ (отправить стенду команду размыкания RL2).
4. Подать напряжение 12В на плату газоанализатора (отправить стенду команду замыкания RL1).
5. Измерить напряжение питания и ток потребления платы АЦМ (отправить стенду команду измерения тока и напряжения питания при помощи двух каналов АЦП).
6. Проверить измеренные ток и напряжение питания, если они не соответствуют заданным значениям, завершить тестирование и выполнить алгоритм завершения работы.
7. Провести измерение напряжения в 4 контрольных точках: +6В, -6В, +5В, +3.3В. Для этого отправить тестовому стенду команды измерения напряжений в указанных контрольных точках при помощи одного канала АЦП и мультиплексора 1:16.
8. Проверить соответствие напряжений заданным значениям: отклонение каждого напряжения не должно превышать ±0.3В. Если любое напряжение не проходит проверку, завершить тестирования и выполнить алгоритм завершения работы.
9. Подать вторичное питание на плате АЦМ: отправить стенду команду замыкания реле RL2 для имитации запаивания резисторов-перемычек R1 и R22.
10. Проверить измеренные ток и напряжение питания, если они не соответствуют заданным значениям, завершить тестирование и выполнить алгоритм завершения работы.
11. Провести измерение напряжения в 15 контрольных точках: +6В, -6В, +5В, +3.3В, +1.2V, +1.8V, +2.5V, Power GPS (+5.5V), VrefADC (+4.5V), +5.5VA, -5.5VA, +1.8VA, Offset (+2.5V), Laser (+5V), VrefDAC (+2.048V). Для этого отправить

тестовому стенду команды измерения напряжений в указанных контрольных точках при помощи одного канала АЦП и мультиплексора 1:16.

12. Проверить соответствие напряжений заданным значениям: отклонение каждого напряжения не должно превышать ±0.3В. Если любое напряжение не проходит проверку, завершить тестирование и выполнить алгоритм завершения работы.
13. Загрузить прошивку в микроконтроллер: пользователь должен самостоятельно запустить внешние утилиты ST-Link Utility и Quartus и прошить МК и ПЛИС платы АЦМ, при удачном завершении прошивки нажать на кнопку «Ок». При неудачном завершении прошивки, необходимо нажать «Не удалось прошить», что выполнит алгоритм завершения работы.
14. Запустить на плате АЦМ измерения в тестовом режиме: отправить плате АЦМ по Ethernet команду запуска измерений в тестовом режиме. Если ответ на команду не получен, завершить тестирование и выполнить алгоритм завершения работы.
15. Проверить значение температуры, которое передает плата АЦМ: оно должно быть 25 градусов. Если это не так, завершить тестирование и выполнить алгоритм завершения работы.
16. Измерить ток и напряжение на эквиваленте элемента Пелтье (отправить команду напряжения элемента Пельтье при помощи двух каналов АЦП).
17. Проверить измеренные значения тока и напряжения Пелтье: эти параметры должны быть равны нулю. Если это не так, завершить тестирование и выполнить алгоритм завершения работы.
18. Установить форму тока лазера: отправить плате АЦМ по Ethernet команду установки параметров тестового режима. При успешном выполнении команды сигналы на графиках должны принять форму трапеции.
19. Измерить форму сигнала лазера с помощью платы тестового стенда (отправить плате стенда команду измерения формы тока лазерного диода при помощи одного канала АЦП).
20. Проверить форму тока (должна быть трапеция), если форма тока не соответствует ожидаемой форме, завершить тестирование и выполнить алгоритм завершения работы.

21. Установить температуру 28 градусов: отправить плате АЦМ команду установки параметров тестового режима, в которых установить это значение температуры.
22. Измерить ток и напряжения на эквиваленте элемента Пелтье: отправить команду напряжения элемента Пельтье при помощи двух каналов АЦП.
23. Проверить измеренные значения тока и напряжения Пелтье. Если ток и напряжение не соответствуют ожидаемым, завершить тестирование и выполнить алгоритм завершения работы.
24. Повторить шаги 18, 19, 20 для значений температуры 22, 55 и -5 градусов (9 шагов).
25. Установить температуру 25 градусов: отправить плате АЦМ команду установки параметров тестового режима, в которых установить это значение температуры.
26. Протестировать работу интерфейса RS232: отправить плате стенда команду отправки заготовленного пакета по интерфейсу RS232. Если запрос выполнен с ошибкой, завершить тестирование и выполнить алгоритм завершения работы.
27. Протестировать работу интерфейса подключения GPS-приемника: отправить на плату стенда команду старта отправки заготовленных NMEA пакетов на контакты GPS-модуля через интерфейс RS232 раз в секунду.
28. Проанализировать пакеты результатов работы платы АЦМ: проверить данные GPS, убедиться, что дата, время и координаты соответствуют ожидаемым (тому, что отправляет плата тестового стенда). Если это не так завершить тестирование и выполнить алгоритм завершения работы.
29. Снять напряжение питание с платы (отправить стенду команду размыкания RL1).
30. Переключить тип входных цепей на внешний оптический блок (отправить стенду команду замыкания RL3-RL7).
31. Подать напряжение 12В на плату газоанализатора (отправить стенду команду замыкания RL1).
32. Измерить напряжение питания и ток потребления платы АЦМ (отправить стенду команду измерения тока и напряжения питания при помощи двух каналов АЦП).

33. Проверить измеренные ток и напряжение питания, если они не соответствуют заданным значениям, завершить тестирование и выполнить алгоритм завершения работы.
34. Завершить тестирование и выполнить алгоритм завершения работы.

Алгоритм завершения работы:

1. Снять напряжение питание с платы (отправить стенду команду размыкания RL1).
2. Переключить тип входных цепей на эквивалентные схемы (отправить стенду команду размыкания RL3-RL7).
3. Отключить вторичное питание платы АЦМ (отправить стенду команду размыкания RL2).

Приложение Е

(обязательное)

Исходный код приложения для Windows

```
////////////////////////////////////////////////////////////////////////////
// uart.pro для системы сборки проектов
// Автоматически собирает проект Makefile, подключает модули Qt, генерирует
// нужные вызовы компилятора, учитывает операционную систему пользователя
// Среда разработки: Qt Creator
// Набор инструментов: MinGW 32-bit
// Система сборки проектов: Qmake
// Входные параметры: отсутствуют
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 20 мая 2025 г.
// Версия 0.2.0
////////////////////////////////////////////////////////////////////////////

QT       += core gui serialport printsupport

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

SOURCES += \
    func_acm.cpp \
    main.cpp \
    mainwindow.cpp \
    protocol_parser.cpp \
    qcustomplot.cpp \


HEADERS += \
    customdialog.h \
    func_acm.h \
    mainwindow.h \
    protocol_parser.h \
    qcustomplot.h \

FORMS += \
    mainwindow.ui

qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

////////////////////////////////////////////////////////////////////////////
// Заголовчный файл mainwindow.h основного цикла программы
// Подключает библиотеки, создает главный класс MainWindow, объявляет
// функции-методы и приватные члены класса
// Язык: C++
// Среда разработки: Qt Creator
// Набор инструментов: MinGW 32-bit
// Система сборки проектов: Qmake
// Компилятор: G++
// Входные параметры: наследственный класс QMainWindow
```

```
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 20 мая 2025 г.
// Версия 0.2.0
////////////////////////////////////////////////////////////////////////////

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSerialPort>
#include <QSerialPortInfo>
#include <QVector>
#include <QMessageBox>
#include <QString>
#include <QWidget>
#include <QComboBox>

#include "protocol_parser.h"
#include "customdialog.h"
#include "func_acm.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

QString description (const QVector<uint8_t>& packet);
QString result (const QVector<uint8_t>& packet);

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_port_ready_read();
    void on_pbOpen_clicked();
    void sendNextPacket();
    void on_cleabutt_clicked();
    void stopTesting();
    void on_pushButton_clicked();
    void handleParserError();
    void handleParsedPacket();
    void startTesting();
    void result(uint8_t* packet);
    void handleCaseCommon(uint16_t sample, const QString& labelText);
    void plotAdcData(const QByteArray& byteArray);
    void setupPort();
    void peltie(uint16_t sample, uint16_t bit);
    void setupConnections();
    void onResponseTimeout();
    void closeTest();
    bool Set_RS_232();
```

```
    void logHtml(const QString& message);
    void logPlain(const QString& message);
    void on_rs232_readyRead();
private:
    Ui::MainWindow *ui;
    QSerialPort *port;
    QSerialPort* rs232Port = nullptr;
   QTimer* sendTimer; // таймер для отправки следующих пакетов через задержку
    QTimer* responseTimer; // таймер ожидания ответа от МК
    struct protocol_parser parser;
    QVector<QVector<uint8_t>> testPackets;
    int currentPacketIndex = 0; //индекс текущего пакета в очереди
    bool isTesting = false; // флаг, идет ли сейчас тестирование
    bool emergencyStopTriggered = false;
    void sendPacket(uint8_t cmd, uint8_t status, uint8_t value);
};

class uart : public QComboBox
{
    Q_OBJECT
public:
    explicit uart(QWidget * parent );

signals:
    void clicked();

protected:
    virtual void showPopup();
    void update_port_list();
};

#endif // MAINWINDOW_H

////////////////////////////////////////////////////////////////////////////
// Основной файл mainwindow.cpp с центральным циклом выполнения программы
// Реализует работу графического интерфейса, управляет алгоритмом
// тестирования, подключает библиотеку для создания графиков QCustomPlot
// Язык: C++
// Среда разработки: Qt Creator
// Компилятор: G++
// Набор инструментов: MinGW 32-bit
// Система сборки проектов: Qmake
// Входные параметры: отсутствуют
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 20 мая 2025 г.
// Версия 0.2.0
////////////////////////////////////////////////////////////////////////////

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QTimer>
#include <QVector>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
```

```
    , port(new QSerialPort(this))
    , rs232Port(new QSerialPort(this))
    , sendTimer(new QTimer(this))
    , responseTimer(new QTimer(this))
{
    ui->setupUi(this);
    setupPort();
    setupConnections();
    parser.state = protocol_parser::STATE_SYNC;
}

MainWindow::~MainWindow() {
    delete ui;
    port->close();
}

void MainWindow::setupConnections() {
    connect(sendTimer, &QTimer::timeout, this, &MainWindow::sendNextPacket);
    connect(port,                &QSerialPort::readyRead,                this,
&MainWindow::on_port_ready_read);
    connect(responseTimer,               &QTimer::timeout,               this,
&MainWindow::onResponseTimeout);
    connect(rs232Port,             &QSerialPort::readyRead,             this,
&MainWindow::on_rs232_readyRead);
    responseTimer->setSingleShot(true);
}

void MainWindow::setupPort() {
    port->setDataBits(QSerialPort::Data8);
    port->setParity(QSerialPort::NoParity);
    port->setStopBits(QSerialPort::OneStop);
    port->setFlowControl(QSerialPort::NoFlowControl);
}

void MainWindow::on_pbOpen_clicked() {
    if (port->isOpen()) {
        port->close();
        ui->pbOpen->setText("Открыть порт");
        return;
    }

    port->setBaudRate(ui->sbxBaudrate->value());
    port->setPortName(ui->cmbxComPort->currentText());

    if (port->open(QIODevice::ReadWrite)) {
        ui->pbOpen->setText("Закрыть порт");
        logHtml("<font color='green'>Порт открыт!</font><br>");
    } else {
        logHtml("<font color='red'>Ошибка открытия порта!</font><br>");
    }
}

void MainWindow::onResponseTimeout() {
    logHtml("<font  color='red'>Ошибка:  не  получен  ответ  в  течение  10
секунд</font><br>");
    closeTest();
}

void MainWindow::on_port_ready_read() {
```

```
    const QByteArray data = port->readAll();
    for (const char byte : data) {
        const parser_result res = process_rx_byte(&parser,
static_cast<uint8_t>(byte));
        if (res == PARSER_DONE) handleParsedPacket();
        else if (res == PARSER_ERROR) handleParserError();
    }
}

void MainWindow::on_rs232_readyRead()
{
    QByteArray data = rs232Port->readAll();

    if (!data.isEmpty()) {
      logHtml("<font color='blue'>Получено от RS-232:</font> " + data.toHex('
').toUpper() + "<br>");
        get_settings();
    }
}

void MainWindow::on_pushButton_clicked() {
    if (!port->isOpen()) {
        logHtml("<font color='red'>Порт закрыт!</font><br>");
        return;
    }

    isTesting = !isTesting;
    ui->pushButton->setText(isTesting ? "Остановить" : "Начать");
    isTesting ? startTesting() : closeTest();
}

void MainWindow::logHtml(const QString& message) {
    ui->plainTextEdit->appendHtml(message);
}

void MainWindow::logPlain(const QString& message) {
    ui->plainTextEdit->appendPlainText(message);
}

bool MainWindow::Set_RS_232(){
    QString rs232PortName = "COM" + QString::number(ui->sbxPort->value());
    rs232Port->setPortName(rs232PortName);
    rs232Port->setBaudRate(115200);

    rs232Port->setDataBits(QSerialPort::Data8);
    rs232Port->setParity(QSerialPort::NoParity);
    rs232Port->setStopBits(QSerialPort::OneStop);
    rs232Port->setFlowControl(QSerialPort::NoFlowControl);

    if (!rs232Port->open(QIODevice::ReadWrite)) {
        logHtml("<font color='red'>Ошибка открытия RS-232 на порту " +
rs232PortName + "</font><br>");
        return false;
    } else {
      logHtml("<font color='green'>RS-232 подключен к порту " + rs232PortName
+ "</font><br>");
        return true;
    }
}
```

```
void MainWindow::startTesting()
{
    if (!Set_RS_232()) {
            closeTest();
            return;
    }
    currentPacketIndex = 0; // счетчик элементов массива testPackets
    testPackets = { /* массив с данными и параметрами выполнения для МК
        {0x01, 0x00, 0x00, 0x00, 0x00, 0x00}, {0x01, 0x07, 0x00, 0x00, 0x00,
0x00}, {0x01, 0x03, 0x00, 0x00, 0x00, 0x00},
        {0x01, 0x00, 0x01, 0x00, 0x00, 0x00},
        {0x01, 0x02, 0x00, 0x00, 0x00, 0x00}, {0x01, 0x02, 0x01, 0x00, 0x00,
0x00},
        {0x01, 0x01, 0x01, 0x00, 0x00, 0x00}, {0x01, 0x01, 0x02, 0x00, 0x00,
0x00}, {0x01, 0x01, 0x03, 0x00, 0x00, 0x00}, {0x01, 0x01, 0x04, 0x00, 0x00,
0x00},
        {0x01, 0x03, 0x01, 0x00, 0x00, 0x00},
        {0x01, 0x02, 0x00, 0x00, 0x00, 0x00}, {0x01, 0x02, 0x01, 0x00, 0x00,
0x00},

        {0x01, 0x04, 0x00, 0x00, 0x00, 0x00}, {0x01, 0x04, 0x01, 0x00, 0x00,
0x00}, {0x01, 0x04, 0x02, 0x00, 0x00, 0x00}, {0x01, 0x04, 0x03, 0x00, 0x00,
0x00},
        {0x01, 0x04, 0x04, 0x00, 0x00, 0x00}, {0x01, 0x04, 0x05, 0x00, 0x00,
0x00}, {0x01, 0x04, 0x06, 0x00, 0x00, 0x00}, {0x01, 0x04, 0x07, 0x00, 0x00,
0x00},
        {0x01, 0x04, 0x08, 0x00, 0x00, 0x00}, {0x01, 0x04, 0x09, 0x00, 0x00,
0x00}, {0x01, 0x04, 0x0A, 0x00, 0x00, 0x00},
        {0x00, 0x06},
        {0x00, 0x05},
        {0x00, 0x06},
        {0x00, 0x08},
        {0x01, 0x09, 0x0A, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x0B, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x0C, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x0D, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x0E, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x0F, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x10, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x11, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x12, 0x00, 0x00, 0x00},
        {0x01, 0x09, 0x13, 0x00, 0x00, 0x00},
        {0x01, 0x00, 0x00, 0x00, 0x00, 0x00},
        {0x01, 0x07, 0x01, 0x00, 0x00, 0x00},
        {0x01, 0x00, 0x01, 0x00, 0x00, 0x00},
        {0x01, 0x02, 0x00, 0x00, 0x00, 0x00},
        {0x01, 0x02, 0x01, 0x00, 0x00, 0x00}
    };
    logHtml("<font color='orange'>Тестирование запущено</font><br>");
    sendNextPacket();
}

void MainWindow::sendNextPacket()
{
    if (!isTesting || currentPacketIndex >= testPackets.size()) {
        stopTesting();
        return;
```

```
    }
     const QVector<uint8_t>& packetData = testPackets[currentPacketIndex++];
     ui->plainTextEdit->appendPlainText(description(packetData));
     transfer packet(packetData);
     serialize_reply(&packet);
     int packetSize = 0;
         if (packet.getCmd() == 0) {
             packetSize = 7;
         } else {
             packetSize = 11;
         }

     QByteArray     byteArray(reinterpret_cast<const     char*>(packet.buf),
packetSize);
     port->write(byteArray);
     responseTimer->start(10000);
     sendTimer->stop();
}

QString description (const QVector<uint8_t>& packet){
    uint8_t status = packet[1];

    switch (status){
        case 0x00:
            if (packet[2] == 0x01){ return "Подача напряжение питания 12В на
плату:"; }
            else {return "Снять напряжение питания 12В с платы:"; }
        case 0x01:
            if (packet[2] == 0x01){ return "Измерение напряжение контрольной
точки: -6V:"; }
         else if (packet[2] == 0x02) {return "Измерение напряжение контрольной
точки: +3.3V:"; }
         else if (packet[2] == 0x03) {return "Измерение напряжение контрольной
точки: +5V:"; }
         else if (packet[2] == 0x04) {return "Измерение напряжение контрольной
точки: +6V:"; }
        case 0x02:
         if  (packet[2]  ==  0x00){  return  "Измерение  напряжения  питания
платы:"; }
            else {return "Измерение тока питания платы:"; }
        case 0x03:
         if (packet[2] == 0x01){ return "Включение вторичного питания платы
АЦМ:"; }
            else {return "Отключение вторичного питания платы АЦМ:"; }
        case 0x04:
            switch (packet[2]){
                case 0x00:
                    return "Измерение напряжение контрольной точки: +1.2V:";
                case 0x01:
                    return "Измерение напряжение контрольной точки: +1.8V:";
                case 0x02:
                    return "Измерение напряжение контрольной точки: +2.5V:";
                case 0x03:
                return "Измерение напряжение контрольной точки: +5.5V (Power
GPS):";
                case 0x04:
                    return "Измерение напряжение контрольной точки: +4.5V:";
                case 0x05:
```

```
                    return "Измерение напряжение контрольной точки: +5.5V:";
                case 0x06:
                    return "Измерение напряжение контрольной точки: -5.5V:";
                case 0x07:
                    return "Измерение напряжение контрольной точки: +1.8V:";
                case 0x08:
                    return  "Измерение  напряжение  контрольной  точки:  +2.5
(Offset)V:";
                case 0x09:
               return "Измерение напряжение контрольной точки: +5V (Laser):";
                case 0x0A:
                    return  "Измерение  напряжение  контрольной  точки:  2.048V
(VrefDAC):";
                }
                case 0x05:
                    return "Измерение формы тока лазерного диода:";
                case 0x06:
               return  "Измерение  напряжения  и  тока  на  эквиваленте  элемента
Пельтье:";
                case 0x07:
               if (packet[2] == 0x01){ return "Переключить тип входных цепей
на внешний оптический блок:"; }
               else {return "Переключить тип входных цепей на эквивалентные
схемы:"; }
                case 0x08:
                    return "Тестирование работы интерфейса RS232:";
                case 0x09: {
                    static bool isFirstTime = true;
                    if (isFirstTime) {
                        isFirstTime = false;
                  return  "Тестирование  работы  интерфейса  подключения  GPS-
приемника:";}
                    else {
                        return "";
                    }
            }
    }
}

void MainWindow::peltie(uint16_t sample, uint16_t bit){
    uint16_t volt_raw, tok;
    volt_raw = (parser.buffer[2] << 8) | parser.buffer[1]; // напряжение 0
    tok = (parser.buffer[3] << 16) | parser.buffer[4] << 24; // ток 0
    double volts = volt_raw / 1000.0;
   if ( (volt_raw >= sample - bit && volt_raw <= sample + bit) && (tok >= sample
- bit && tok <= sample + bit) ){
      logHtml(QString("<font color='green'>Измерено: %1 мА — Ток эквивалента
элемента Пельтье допустим</font>").arg(tok));
        logHtml(QString("<font  color='green'>Измерено:  %1  В  —  Напряжение
эквивалента                          элемента                          Пельтье
допустимо</font><br><br>").arg(QString::number(volts, 'f', 3)));
        return;
    }
    else {
       logHtml(QString("<font color='red'>Измерено: %1 мА — Ток эквивалента
элемента Пельтье не допустим</font><br>").arg(tok));
```

```
       logHtml(QString("<font color='red'>Измерено: %1 В — Напряжение
эквивалента элемента Пельтье не
допустимо</font><br><br>").arg(QString::number(volts, 'f', 3)));
        closeTest();
    }
}

void MainWindow::result(uint8_t* packet){
    switch (currentPacketIndex){
        case 1:
        case 2:
        case 3:
        case 4:
        case 11:
            logHtml("<font color='green'>Выполнено!</font><br>");
            return;


        case 5:
        case 12:
            handleCaseCommon(2000, "Питание платы,");
            return;


        case 6:
        case 13:{
            uint16_t sample = 50;
            uint16_t tok = 10;
            uint16_t data = (parser.buffer[2] << 8) | parser.buffer[1];

            if (data >= sample - tok && data <= sample + tok){
            logHtml(QString("<font color='green'>Измерено: %1 мА — Ток питания
платы допустим</font><br>").arg(data));
            }
            else {
           logHtml(QString("<font color='red'>Измерено: %1 мА — Ток питания
платы не допустим</font><br>").arg(data));
               closeTest();
               }
            return; }
        case 7:
            handleCaseCommon(6000, "Контрольная точка -6 В");
            return;
        case 8:
            handleCaseCommon(3300, "Контрольная точка +3.3 В");
            return;
        case 9:
            handleCaseCommon(5000, "Контрольная точка +5 В");
            return;
        case 10:
            handleCaseCommon(6000, "Контрольная точка +6 В");
            return;
        case 14:
            handleCaseCommon(6000, "Контрольная точка +1.2 В");
            return;
        case 15:
            handleCaseCommon(6000, "Контрольная точка +1.8 В");
            return;
        case 16:
```

```
            handleCaseCommon(6000, "Контрольная точка +2.5 В");
            return;
        case 17:
            handleCaseCommon(6000, "Контрольная точка +5.5 В (Power GPS)");
            return;
        case 18:
            handleCaseCommon(6000, "Контрольная точка +4.5 В (VrefADC)");
            return;
        case 19:
            handleCaseCommon(6000, "Контрольная точка +5.5 В");
            return;
        case 20:
            handleCaseCommon(6000, "Контрольная точка -5.5 В");
            return;
        case 21:
            handleCaseCommon(6000, "Контрольная точка +1.8 В");
            return;
        case 22:
            handleCaseCommon(6000, "Контрольная точка +2.5 В (Offset)");
            return;
        case 23:
            handleCaseCommon(6000, "Контрольная точка +5 В (Laser)");
            return;
        case 24:{
            handleCaseCommon(0, "Контрольная точка +2.048 В (VrefDAC)");
            sendTimer->stop();
            responseTimer->stop();
            CustomDialog dialog_1(this,"Выполните условие", "Прошейте МК и
ПЛИС","Ок","Не удалось прошить");
            if (dialog_1.exec()) {
               logHtml("<font color='green'>МК и ПЛИС прошиты. Продолжение
теста...</font><br>");
                } else {
               logHtml("<font color='red'>МК и ПЛИС не прошиты.</font><br>");
                    closeTest();
                    return;
                }

            if (set_test_settings()) {
                    logHtml("<font color='green'></font><br>");
                } else {
                    logHtml("<font   color='red'>Команда   плате   АЦМ   не
отправлена.</font><br>");
                    closeTest();
                    return;
                }
            return; }
        case 25:
            peltie(0,2);
            return;

        case 26: {
            if (set_laser_settings(1)) {
                    logHtml("<font    color='green'>Форма    тока    лазера
установлена.</font>");
                } else {
```

```
                    logHtml("<font color='red'>Форма тока лазера не
установлена.</font><br>");
                    closeTest();
                    return;
                }
            QByteArray rawData(reinterpret_cast<const char*>(parser.buffer),
parser.buffer_length);
            plotAdcData(rawData);
            sendTimer->stop();
            responseTimer->stop();
            CustomDialog dialog_2(this,"Проверка", "Проверьте форму тока
лазера","Корректная форма","Некорреткная форма");
            if (dialog_2.exec()) {
                    logHtml("<font color='green'>Форма тока лазера правильной
формы. Продолжение теста...</font><br>");
                } else {
                logHtml("<font color='red'>Форма тока лазера не правильной
формы.</font><br>");
                    closeTest();
                    return;
                }
            return; }
        case 27: {
            for (size_t i = 0; i < 4; i++){
                std::vector<int> temperatures = {28, 22, 55, -5};
                if (i < temperatures.size()) {
                    int targetTemp = temperatures[i];
                    if (!set_temp_mode(targetTemp)) {
                  logHtml(QString("<font color='red'>Температура %1 градусов
не установлена</font>").arg(targetTemp));
                        closeTest();
                        return;
                    }
              logHtml(QString("<font color='green'>Температура %1 градусов
установлена</font>").arg(targetTemp));
                    peltie(5000,300); // выставить правильно погрешность
                }
             }
            if (!set_temp_mode(25)) {
                logHtml("<font color='red'>Температура 25 градусов не
установлена</font>");
                closeTest();
                return;
            }
            logHtml("<font color='green'>Температура 25 градусов
установлена</font><br>");
            return; }
        case 28:
            logHtml("<font color='green'>Тестирование RS-232 успешно
пройдено</font><br>");
            return;
        case 29:
        case 30:
        case 31:
        case 32:
        case 33:
        case 34:
        case 35:
```

```
        case 36:
        case 37:
        case 38:
            sendTimer->start(1000);
            if (currentPacketIndex == 38){
            logHtml("<font     color='green'>Тестирование     GPS     успешно
пройдено</font><br>"); }
            return;
        case 39:
        case 40:
            logHtml("<font color='green'>Выполнено!</font><br>");
            return;
        case 41:
            logHtml("<font color='green'>Выполнено!</font><br>");
            logHtml("<font         color='green'>Тестирование         успешно
пройдено!</font><br>");
            closeTest();
            return;
    }
}

void MainWindow::plotAdcData(const QByteArray& byteArray) {
    if (byteArray.size() < 201) {
        logHtml("<font   color='red'>Недостаточно   данных   для   построения
графика</font><br>");
        return;
    }
    QVector<double> x(100), y(100);
    int dataStartIndex = 1;

    for (int i = 0; i < 100; ++i) {
         int index = dataStartIndex + i * 2;
         if (index + 1 >= byteArray.size()) break;

         uint8_t low = static_cast<uint8_t>(byteArray[index]);
         uint8_t high = static_cast<uint8_t>(byteArray[index + 1]);
         uint16_t value = (high << 8) | low;
         x[i] = i;
         y[i] = static_cast<double>(value) / 1000;
    }

    ui->customPlot->clearGraphs();
    ui->customPlot->addGraph();
    ui->customPlot->graph(0)->setData(x, y);
    ui->customPlot->xAxis->setLabel("Номер точки (сигнал лазерного диода)");
    ui->customPlot->yAxis->setLabel("Напряжение, В");
    ui->customPlot->xAxis->setRange(0, 99);
    ui->customPlot->yAxis->setRange(0, 3.3);
    ui->customPlot->replot();
    logHtml("<font  color='green'>Снятно  100  точек  напряжений,  построен
график.</font><br>");
}

void MainWindow::handleCaseCommon(uint16_t sample, const QString& labelText)
{
    const uint16_t accuracy = 300;
    uint16_t data = (parser.buffer[2] << 8) | parser.buffer[1];
```

```
    double volts = data / 1000.0;

    if (data >= sample - accuracy && data <= sample + accuracy) {
        logHtml(QString("<font color='green'>Измерено: %1 В — %2 напряжение
допустимо</font><br><br>").arg(QString::number(volts,                    'f',
3)).arg(labelText));
    } else {
        logHtml(QString("<font color='red'>Измерено: %1 В — %2 напряжение
превышает      диапазон</font><br><br>").arg(QString::number(volts,      'f',
3)).arg(labelText));
        closeTest();
    }
}

void MainWindow::handleParsedPacket()
{
    responseTimer->stop();

        switch (parser.buffer[0]){
            case 0x00:
                result(parser.buffer);
                if (isTesting) {
                    sendTimer->start(200);
                    return;
                }
            case 0x01:
            logHtml("<font color='red'>Ошибка выполнения команды (код ошибки:
0x01)</font><br>");
                result(parser.buffer);
                if (isTesting) {
                    closeTest();
                    return; }
            case 0x02:
            logHtml("<font  color='red'>Несуществующая  команда  (код  ошибки:
0x02)</font><br>");
                closeTest();
                break;
            case 0x03:
               logHtml("<font color='red'>Превышено время выполнения команды
(код ошибки: 0x03)</font><br>");
                closeTest();
                break;
            case 0x04:
                logHtml("<font color='red'>Ошибка размера данных команды (код
ошибки: 0x04)</font><br>");
                closeTest();
                break;
        }
}

void MainWindow::closeTest(){
    logHtml("<font color='red'>Завершение тестирования...</font>");
    if (!emergencyStopTriggered) {
        emergencyStopTriggered = true;

        testPackets = {
        {0x01, 0x00, 0x00, 0x00, 0x00, 0x00},
        {0x01, 0x07, 0x00, 0x00, 0x00, 0x00},
```

```
        {0x01, 0x03, 0x00, 0x00, 0x00, 0x00}
        };

        currentPacketIndex = 0;
        logHtml("<font    color='orange'>Повтор    команд    для    отключения
питания...</font><br>");
        sendTimer->start(100);
        return;
    }
    stopTesting();
}

void MainWindow::handleParserError()
{
    logHtml("<font color='red'>Ошибка разбора пакета</font><br>");
    parser.state = protocol_parser::STATE_SYNC;
}

void MainWindow::stopTesting()
{
    sendTimer->stop();
    responseTimer->stop();
    testPackets.clear();
    isTesting = false;
    emergencyStopTriggered = false;
    currentPacketIndex = 0;
    if (rs232Port->isOpen()) {
        rs232Port->close();
    }
    ui->pushButton->setText("Начать тестирование");
    logHtml("<font color='orange'><br>Тестирование завершено</font><br>");
}

void MainWindow::on_cleabutt_clicked()
{
    ui->plainTextEdit->clear();
}

uart::uart(QWidget * parent)
    : QComboBox(parent)
{}

void uart::showPopup()
{
    auto lastText = currentText();
    update_port_list();
    setCurrentText(lastText);
    QComboBox::showPopup();
}

void uart::update_port_list()
{
    auto portInfoList = QSerialPortInfo::availablePorts();
    clear();
    for(auto & portInfo : portInfoList)
        if(!portInfo.isBusy())
            addItem(portInfo.portName());
}
```

```
////////////////////////////////////////////////////////////////////////////
// Заголовочный файл protocol_parser.h модуля обработки протокольных пакетов
// и диспетчеризации команд
// Объявляет функции диспетчеризации и функции обработки протокола передачи
// данных. Подключает библиотеки, инициализирует константы, структуры для
// протокола передачи данных
// Язык: C++
// Среда разработки: Qt Creator
// Компилятор: G++
// Набор инструментов: MinGW 32-bit
// Система сборки проектов: Qmake
// Входные параметры: отсутствуют
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 18 марта 2025 г.
// Версия 0.1.1
////////////////////////////////////////////////////////////////////////////

#ifndef PROTOCOL_PARSER_H
#define PROTOCOL_PARSER_H
#include <stdint.h>
#include <stddef.h>
#include <stdio.h>
#include <cstdlib>
#include <QVector>

#define MAX_DATA_SIZE 201 // Максимальный размер буфера.
#define SYNC_BYTE 0xAA // Синхробайт.
#define DATA_SIZE_OFFSET 3 // 2 байта crc + код команды.
#define SIZE_PAKET 7 // синхробайт + 2 байта полезных данных + cmd + status +
2 CRC.
#define CRC_INIT 0xffff // для подсчета контрольной суммы CRC.

enum parser_result {
    PARSER_OK,
    PARSER_ERROR,
    PARSER_DONE,
};

struct protocol_parser {
    enum {
        STATE_SYNC,
        STATE_SIZE_L,
        STATE_SIZE_H,
        STATE_CMD,
        STATE_DATA,
        STATE_CRC_L,
        STATE_CRC_H,
    }state;

    uint8_t buffer[MAX_DATA_SIZE];  // Буфер для хранения данных пакета.
    size_t buffer_length;           // Количество принятых байт данных.
    uint16_t data_size;             // Размер полезных данных, полученный из
пакета с учетом DATA_SIZE_OFFSET.
    uint8_t cmd;                    // Команда пакета.
    uint16_t crc;                   // Накопленная контрольная сумма.
};
```

```
class transfer{
public:
    uint8_t* buf;
private:
    uint8_t cmd;
    uint8_t status;
    uint8_t* value;
public:
    transfer(const QVector<uint8_t>& input){
        this->cmd = input[0];
        this->status = input[1];

        if(this->cmd == 1) {
        this->value = new uint8_t[4];
        if (this->value != nullptr){
            for ( int i = 0; i < 4; i++ ){
                this->value[i] = input[i + 2]; }
        }
        this->buf = new uint8_t[11];
        if (this->buf == nullptr){
            std::exit(EXIT_FAILURE);
        }
    }
        else {
            this->value = nullptr;
            this->buf = new uint8_t[7];
                if (this->buf == nullptr){
                    std::exit(EXIT_FAILURE); }
        }
}

    ~transfer(){
     delete[] value;
     delete[] buf;
    }

    uint8_t getCmd()const { return cmd; }
    uint8_t getStatus()const { return status; }
    uint8_t* getValue()const { return value; }

};

//static uint16_t update_crc(uint16_t crc, uint8_t byte);
enum parser_result process_rx_byte(struct protocol_parser *parser, uint8_t
byte);
void serialize_reply(transfer* data);

static const uint16_t crc16_table[256] =
{
    0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280, 0xC241,
    0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481, 0x0440,
    0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81, 0x0E40,
    0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880, 0xC841,
    0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81, 0x1A40,
    0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80, 0xDC41,
    0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680, 0xD641,
    0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081, 0x1040,
    0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281, 0x3240,
```

```
    0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480, 0xF441,
    0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80, 0xFE41,
    0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881, 0x3840,
    0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80, 0xEA41,
    0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81, 0x2C40,
    0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681, 0x2640,
    0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080, 0xE041,
    0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281, 0x6240,
    0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480, 0xA441,
    0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80, 0xAE41,
    0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881, 0x6840,
    0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80, 0xBA41,
    0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81, 0x7C40,
    0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681, 0x7640,
    0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080, 0xB041,
    0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280, 0x9241,
    0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481, 0x5440,
    0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81, 0x5E40,
    0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880, 0x9841,
    0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81, 0x4A40,
    0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80, 0x8C41,
    0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680, 0x8641,
    0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081, 0x4040,
};

#endif // PROTOCOL_PARSER_H

////////////////////////////////////////////////////////////////////////////
// Основной файл protocol_parser.cpp модуля обработки протокольных пакетов
// и диспетчеризации команд
// Реализует функции для протокола передачи данных, счетчик регистра CRC
// Язык: C++
// Среда разработки: Qt Creator
// Компилятор: G++
// Набор инструментов: MinGW 32-bit
// Система сборки проектов: Qmake
// Входные параметры: байты протокола передачи данных с ПК (rx_byte)
// Выходные параметры: массив байт протокола передачи данных на ПК (data.buf)
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 18 марта 2025 г.
// Версия 0.1.1
////////////////////////////////////////////////////////////////////////////

#include "protocol_parser.h"

struct protocol_parser parser;

static uint16_t update_crc(uint16_t crc, uint8_t byte)
{
   return crc16_table[(crc ^ byte) & 0xFF] ^ (crc >> 8);
}

static uint16_t calculate_crc(const uint8_t* array, int size) {
   uint16_t crc = CRC_INIT; // #define CRC_INIT 0xffff
   int i;
   for (i = 0; i < size; i++) {
       crc = update_crc(crc, array[i]);
   }
```

```
   return crc;
}

void serialize_reply(transfer* data) {
    if (data == nullptr) return;

    uint16_t crc;
    uint16_t PAYLOAD_SIZE;

    if (data->getCmd() == 0) {
        PAYLOAD_SIZE = 1;
        data->buf[0] = SYNC_BYTE;
        data->buf[1] = ((PAYLOAD_SIZE + DATA_SIZE_OFFSET) >> 0) & 0xff;
        data->buf[2] = ((PAYLOAD_SIZE + DATA_SIZE_OFFSET) >> 8) & 0xff;
        data->buf[3] = data->getCmd();
        data->buf[4] = data->getStatus();
        crc = calculate_crc(data->buf + 3, PAYLOAD_SIZE + 1);
        data->buf[5] = (crc >> 0) & 0xff;
        data->buf[6] = (crc >> 8) & 0xff;
        return;
    } else {
        PAYLOAD_SIZE = 5;
        data->buf[0] = SYNC_BYTE;
        data->buf[1] = ((PAYLOAD_SIZE + DATA_SIZE_OFFSET) >> 0) & 0xff;
        data->buf[2] = ((PAYLOAD_SIZE + DATA_SIZE_OFFSET) >> 8) & 0xff;
        data->buf[3] = data->getCmd();
        data->buf[4] = data->getStatus();
        uint8_t* value = data->getValue();
        for (size_t i = 5; i < 9; i++)
        {
            data->buf[i] = value[i - 5];
        }
        crc = calculate_crc(data->buf + 3, PAYLOAD_SIZE + 1);
        data->buf[9] = (crc >> 0) & 0xff;
        data->buf[10] = (crc >> 8) & 0xff;
    }

}

enum parser_result process_rx_byte(struct protocol_parser *parser, uint8_t
byte) {
    enum parser_result ret = PARSER_OK;

    switch (parser->state) {
        case protocol_parser::STATE_SYNC:
            if (byte == SYNC_BYTE) {
                parser->state = protocol_parser::STATE_SIZE_L;
                parser->crc = CRC_INIT;
                parser->buffer_length = 0;
            }
            break;
        case protocol_parser::STATE_SIZE_L:
            parser->data_size = byte;
            parser->state = protocol_parser::STATE_SIZE_H;
            break;
        case protocol_parser::STATE_SIZE_H:
            parser->data_size |= ((uint16_t)byte << 8);
            if (parser->data_size >= DATA_SIZE_OFFSET &&
```

```
                parser->data_size <= MAX_DATA_SIZE + DATA_SIZE_OFFSET) {
                parser->state = protocol_parser::STATE_CMD;
            } else {
                parser->state = protocol_parser::STATE_SYNC;
                ret = PARSER_ERROR;
            }
            break;
        case protocol_parser::STATE_CMD:
            parser->crc = update_crc(parser->crc, byte);
            parser->cmd = byte;
            parser->state = (parser->data_size != DATA_SIZE_OFFSET) ?
protocol_parser::STATE_DATA : protocol_parser::STATE_CRC_L;
            break;
        case protocol_parser::STATE_DATA:
            parser->crc = update_crc(parser->crc, byte);
            parser->buffer[parser->buffer_length++] = byte;
            if (parser->buffer_length + DATA_SIZE_OFFSET >=
parser->data_size) {
                parser->state = protocol_parser::STATE_CRC_L;
            }
            break;
        case protocol_parser::STATE_CRC_L:
            parser->crc = update_crc(parser->crc, byte);
            parser->state = protocol_parser::STATE_CRC_H;
            break;
        case protocol_parser::STATE_CRC_H:
            parser->crc = update_crc(parser->crc, byte);
            parser->state = protocol_parser::STATE_SYNC;
            ret = (parser->crc == 0 ? PARSER_DONE : PARSER_ERROR);
            break;
        default:
            ret = PARSER_ERROR;
    }
    return ret;
}
```

Приложение Ж

(обязательное)

Исходный текст функций платы газоанализатора

```
////////////////////////////////////////////////////////////////////////////
// Заголовочный файл func_acm.h модуля взаимодействия с платой АЦМ
// Объявляет возможные функции для работы с платой АЦМ, реализация функций
// скрыта
// Язык: C
// Среда разработки: Visual Studio Code
// Компилятор: GCC
// Входные параметры: отсутствуют
// Выходные параметры: отсутствуют
// Автор: Жеребцов А.А., группа РЭ3-11
// Модификация: 2 февраля 2025 г.
// Версия 0.0.3
////////////////////////////////////////////////////////////////////////////

#ifndef FUNC_ACM_H
#define FUNC_ACM_H

#include <QByteArray>
#include <QtGlobal>

// Устройство
void get_device_param(uint8_t par);

// Алгоритм
void exec_cmd(uint8_t cmd);
void exec_cmd_adc_test(uint8_t cmd_data);
void get_alg_param(uint8_t par);
void set_leak_level(uint8_t leakLevel);
int  set_temp_mode(uint8_t tempMode);
void set_calc_mode(uint8_t calcMode);

// Настройки
void set_alg_settings();
void set_work_settings();
int  set_test_settings();
void set_link_settings();
int  set_laser_settings(uint8_t temp);
void set_dist_settings();
void set_gps_settings();

// Чтение настроек
void get_settings();
void exec_settings_cmd(uint8_t cmd, uint8_t type);

// Служебные функции
void make_dev_get_param_packet(uint8_t par);
void make_alg_cmd_packet(uint8_t cmd);
void make_alg_cmd_packet_ext(uint8_t cmd, uint8_t cmd_data);
void make_alg_set_leak_level_packet();
void make_alg_set_temp_mode_packet();
void make_alg_set_calc_mode_packet();
void make_alg_get_param_packet(uint8_t par);
```

```
void make_set_settings_packet();
void make_get_settings_packet(uint8_t par);
void make_settings_cmd_packet(uint8_t cmd, uint8_t par);

#endif // FUNC_ACM_H
```