

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

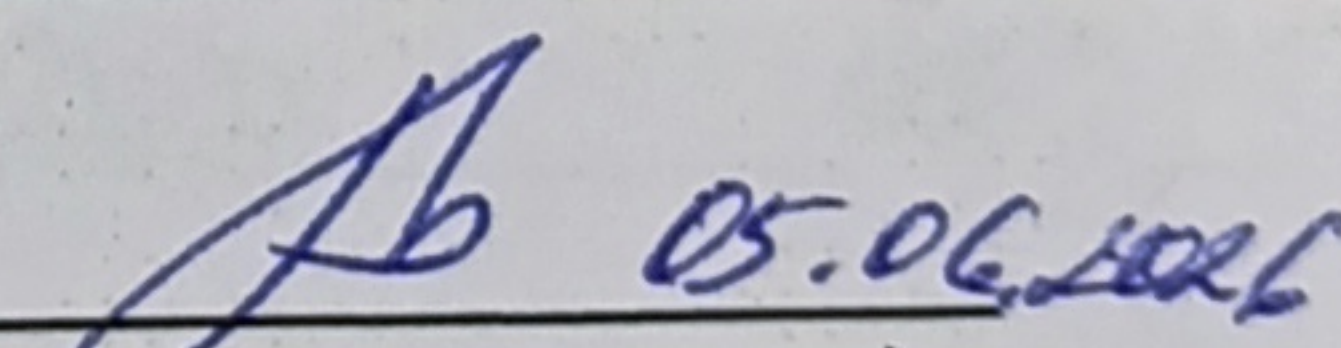
Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

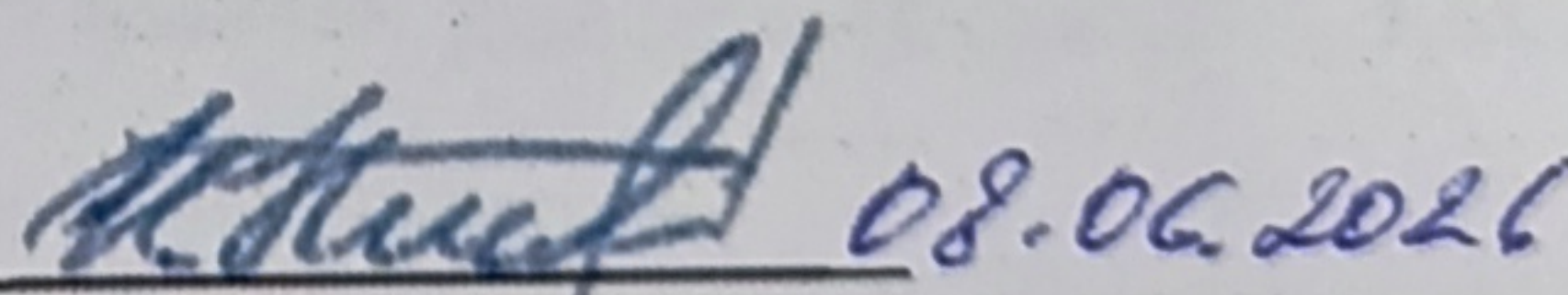
ИНТЕЛЛЕКТУАЛЬНАЯ
МИКРОКОНТРОЛЛЕРНАЯ СИСТЕМА
УПРАВЛЕНИЯ
ДЛЯ ОХРАННОГО КОМПЛЕКСА

Студент


(подпись, дата)

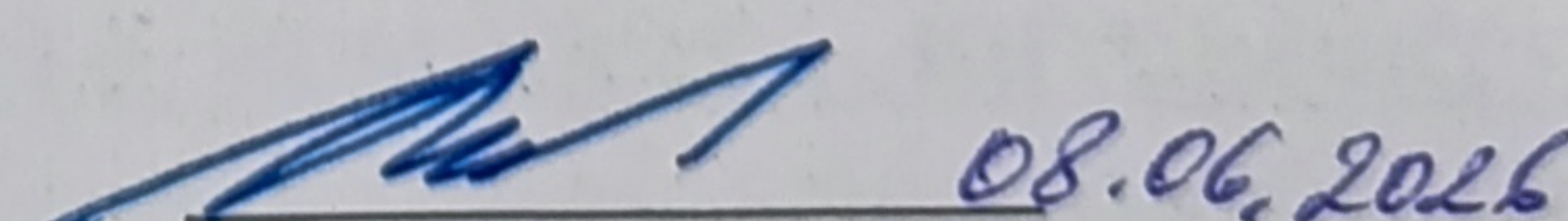
А.С. Грошев
(ФИО полностью)

Руководитель


(подпись, дата)

К.И. Никишин
(фамилия, инициалы)

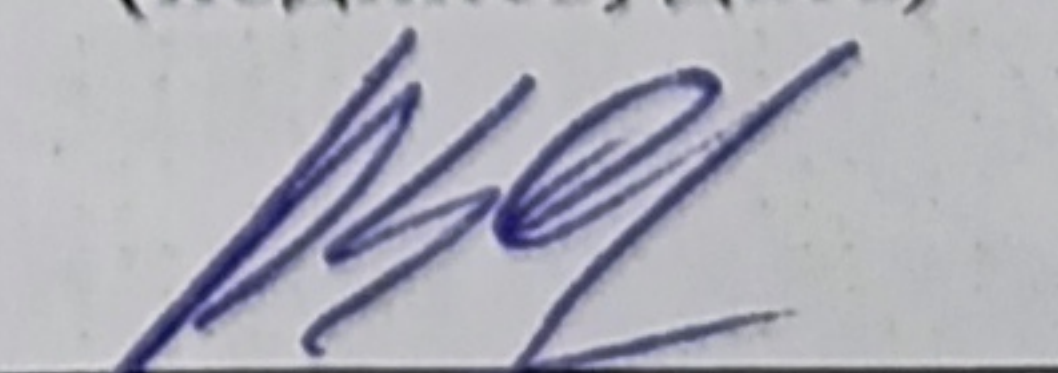
Нормоконтролёр

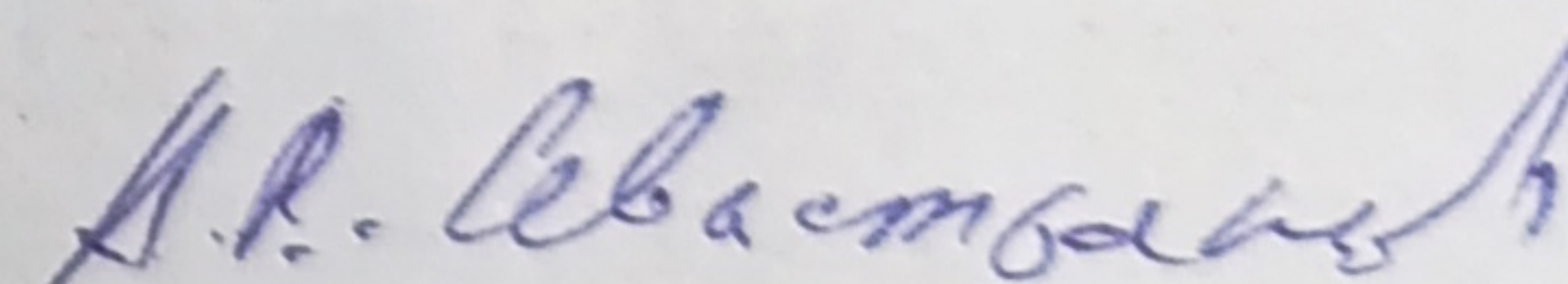

(подпись, дата)

А.С. Бычков
(фамилия, инициалы)

Рецензент

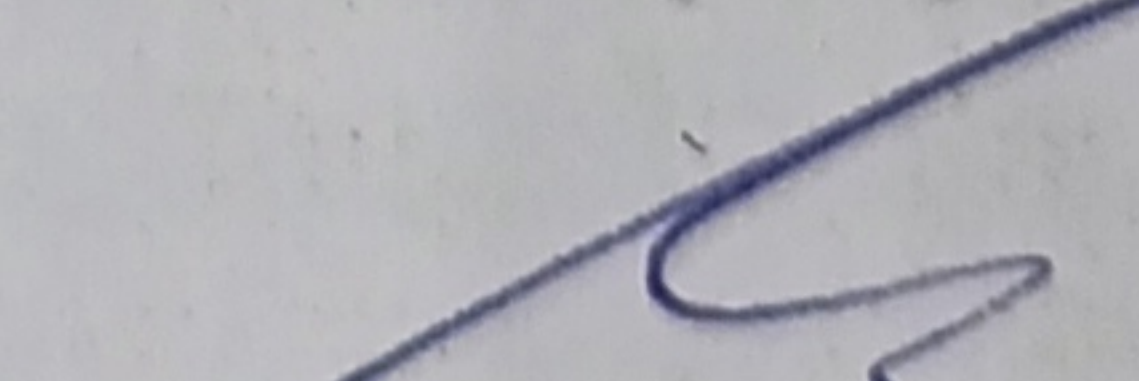
(должность, место работы)


(подпись, дата)

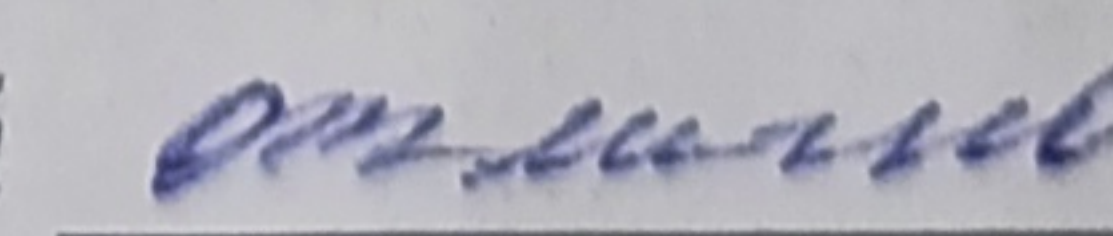

(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 № 14)

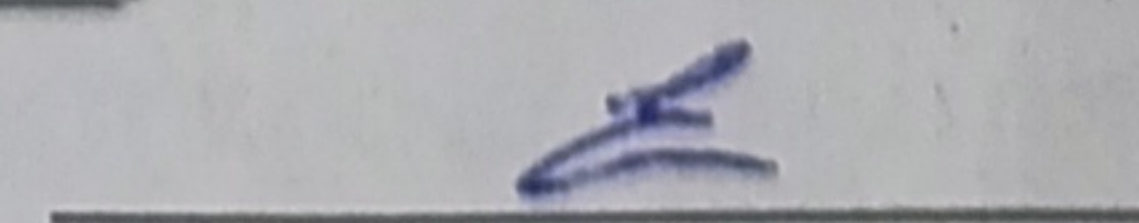
Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от

09 июня 2026г. № 23)
Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 20 » 09 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Грошев Александр Сергеевич гр. 24ВВИм2 факультета ВТ

2. Тема работы Интеллектуальная микроконтроллерная система управления для охранного комплекса

Тема утверждена приказом ПГУ № 920/о от " 20 " 09 2024 г.

3. Руководитель магистерской диссертации к.т.н, доцент Никишин К.И.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Создание программно-аппаратного комплекса интеллектуального управления охранной сигнализацией на базе распределённых микроконтроллерных ведомых узлов, и подсистемы удалённого доступа с применением средств искусственного интеллекта для обработки запросов оператора и исследование влияния архитектурных решений на надёжность обмена и оперативность управления охранными зонами.

Создание ведомого микроконтроллерного модуля, программного мастера опроса, приложения мониторинга и подсистемы удалённого доступа; исследование согласованного протокола обмена и интеллектуального взаимодействия с оператором.

Объекты и подсистемы разработки:

ведомый узел на микроконтроллере STM32F103 с конечным автоматом охранности и локальными датчиками;

Telegram-бот с локальной языковой моделью для оркестрации команд управления.

Исходные данные: техническое задание на магистерскую диссертацию; материалы анализа предметной области; документация на микроконтроллер STM32F103; описание учебно-лабораторного стенда охранного комплекса; отчёты по научно-исследовательской работе.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области, требований к охранным комплексам с распределёнными микроконтроллерными модулями и существующих подходов к

централизованному управлению и удалённому доступу.

2) Анализ и выбор элементной базы, средств разработки прошивки и прикладного программного обеспечения (STM32CubeMX, STM32CubeIDE, Python).

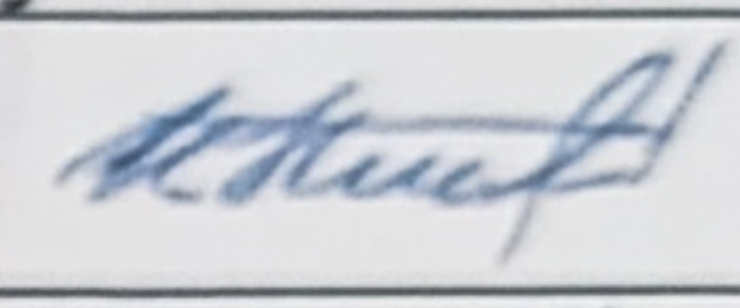
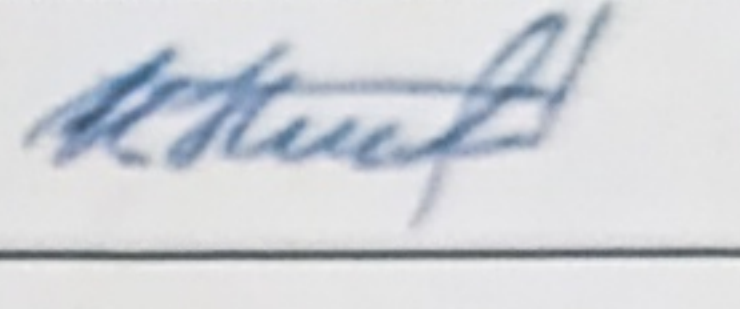
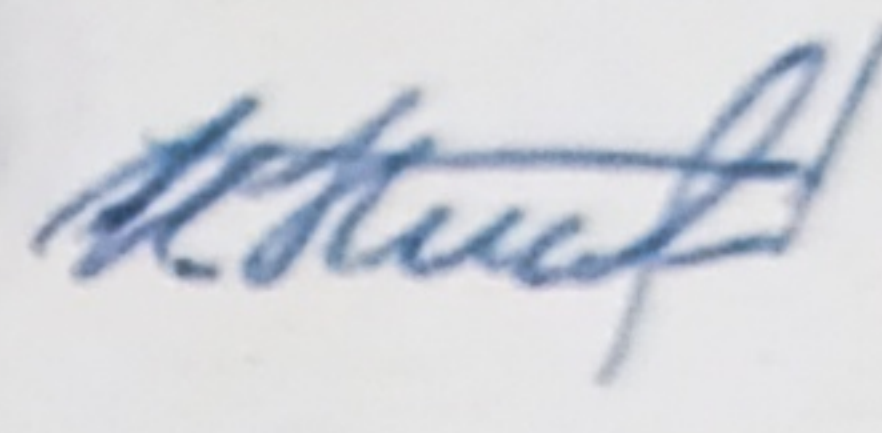
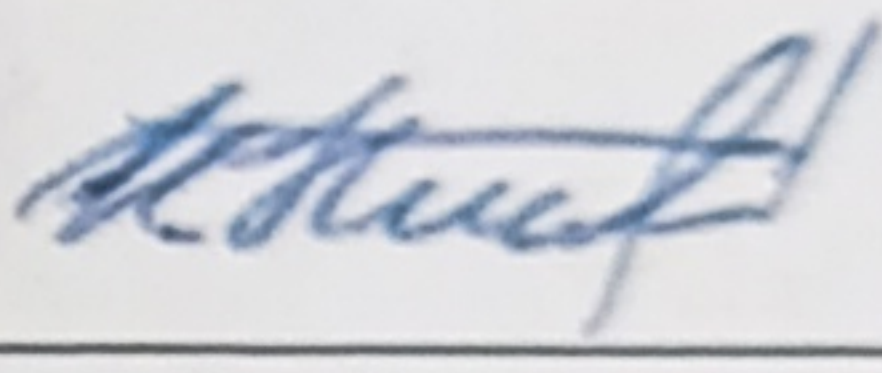
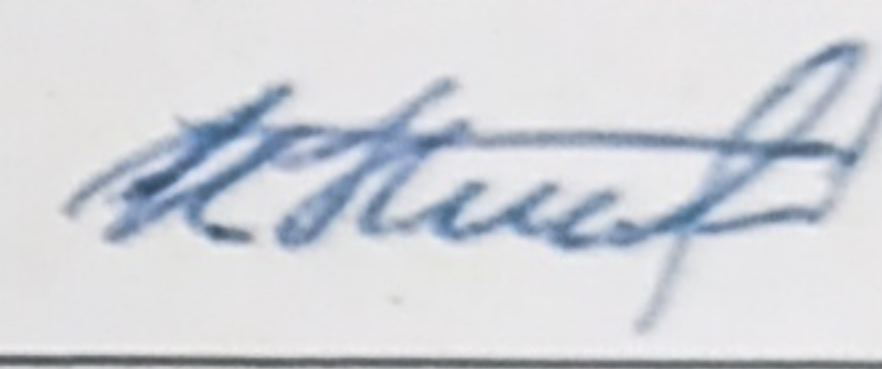
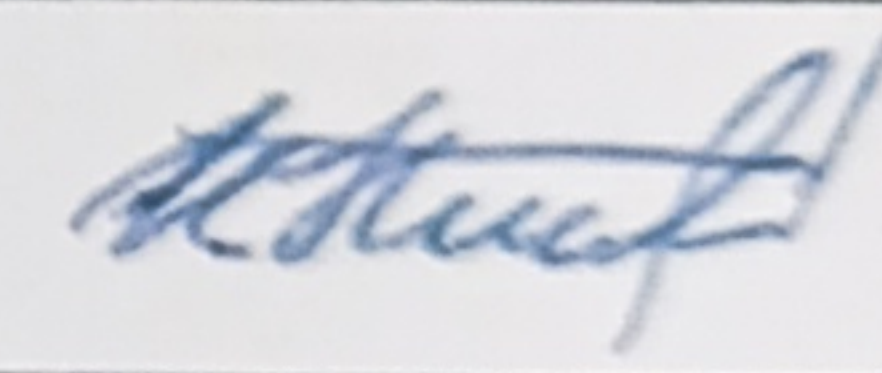
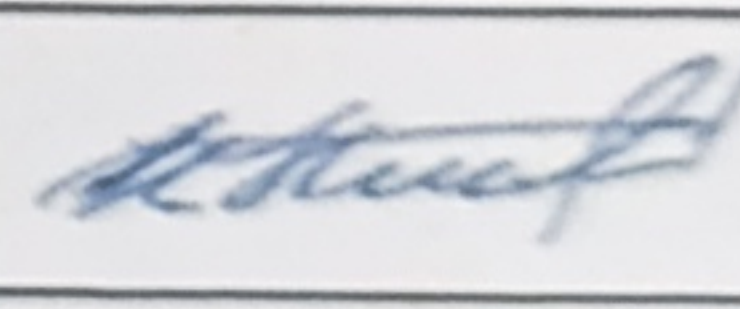
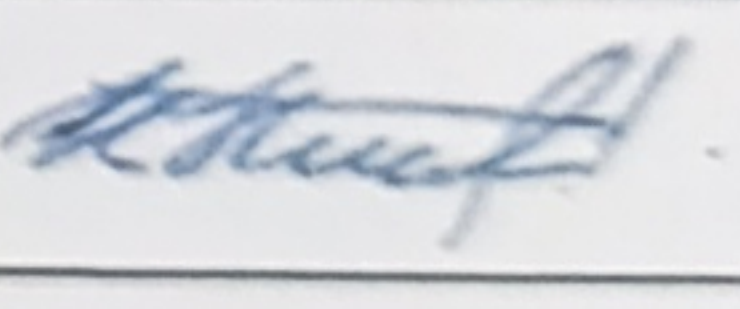
3) Создание прошивки ведомого модуля с обработкой сигналов датчиков, формированием байта статуса охранных зон и выполнением команд постановки и снятия с охраны.

4) Разработка программного мастера циклического опроса, журналирования событий и HTTP-интерфейса на персональном компьютере.

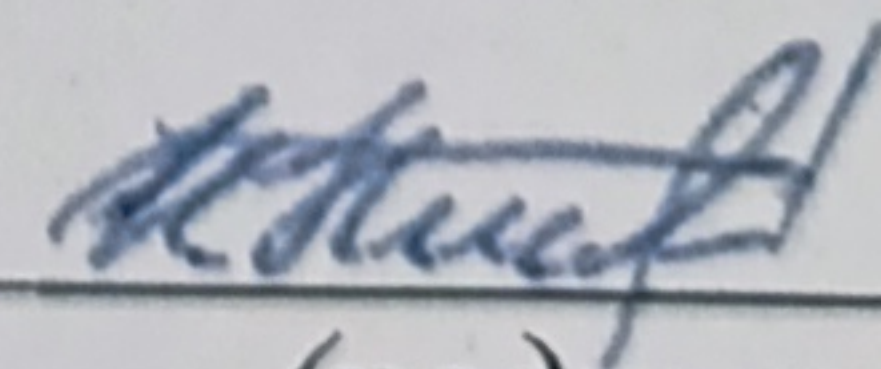
5) Создание приложения мониторинга и подсистемы удалённого доступа через Telegram с локальной языковой моделью для обработки запросов оператора на естественном языке.

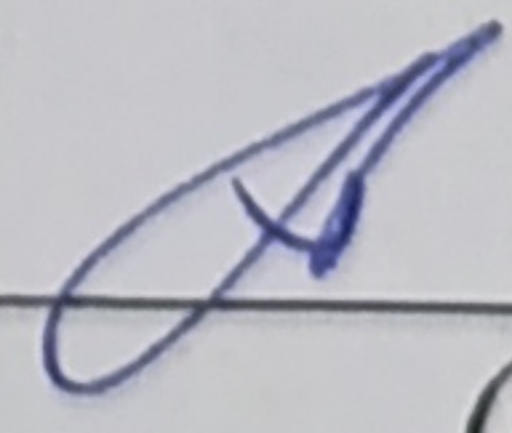
Тестирование и отладка комплекса при различных сценариях работы охранных зон, потери связи и удалённого управления.

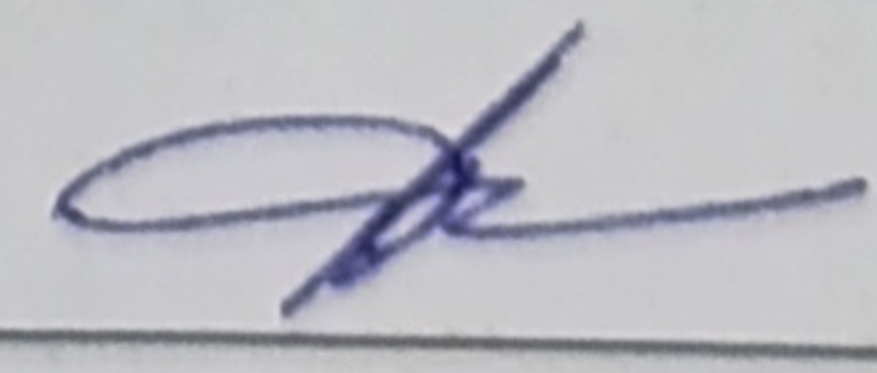
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объём работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	11%	29.11.2024	
2	Выбор средства моделирования	11%	29.01.2025	
3	Разработка архитектуры программно-аппаратного комплекса и протокола обмена	22%	30.03.2025	
4	Создание прошивки ведомого узла на STM32F103	27%	30.05.2025	
5	Разработка программного мастера опроса и HTTP-интерфейса	11%	01.12.2025	
6	Создание приложения мониторинга и подсистемы Telegram-бота	6%	10.02.2026	
7	Исследование полученных моделей	6%	01.04.2026	
8	Оформление пояснительной записки	6%	30.04.2026	

Дата выдачи задания "20" "09" 2024 г.

Руководитель магистерской диссертации  Никишин К.И.
(подпись, дата)

Задание к исполнению принял студент  Грошев А.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л. Р.
(подпись, дата)
09.06.2024

Реферат

Пояснительная записка содержит 75 листов, 10 таблиц, 18 рисунков.

STM32F103, ОХРАННЫЙ КОМПЛЕКС, МИКРОКОНТРОЛЛЕР, ПРОТОКОЛ ОБМЕНА, TELEGRAM-БОТ, ПРИКЛАДНОЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

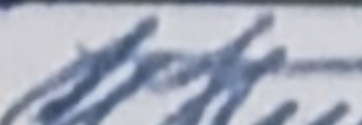
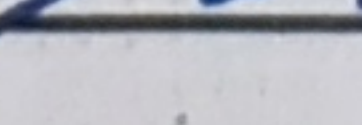
Объектом исследования являются распределённые системы сбора охранной информации поверх аппаратного UART с адресацией ведомых устройств.

Предметом исследования являются архитектуры программного обеспечения и алгоритмы синхронизации обменов на фоне возможных ошибок синхронизации и битового мусора.

Цель данной работы состояла в разработке и описании прототипа интеллектуальной микроконтроллерной системы управления для охранного комплекса, включающего ведомый узел на STM32F103, программный мастер циклического опроса на персональном компьютере, приложение мониторинга и подсистему удалённого доступа через мессенджер с применением локальной языковой модели для интерпретации пользовательских запросов.

В результате проделанной работы разработана и испытана прошивка ведомого узла; реализован программный мастер; создано приложение операторского мониторинга; построена подсистема Telegram-бота с локальной языковой моделью Qwen 2.5-3B для оркестрации вызовов инструментов управления.

Научная новизна состоит в том, что предложенная архитектура комплекса обеспечивает интеграцию средств прикладного искусственного интеллекта в задачу интуитивно понятного пользовательского интерфейса к локальному управляющему устройству.

					ПГУ 1.090401.02.001 ПЗ			
		№ докум.	Подпись	Дата	Интеллектуальная микроконтроллерная система управления для охранного комплекса	Лит.	Лист	Листов
Разраб.	Грошев А.С.		15.06.20				2	75
Провер.	Никишин К.И.		18.06.20					
Н. Контр	Бычков А.С.		18.06.20					
Утверд.	Митрохин М.А.		18.06.20			ФВТ, каф. ВТ гр. 24ВВИм2		

СОДЕРЖАНИЕ

<u>ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ</u>	4
<u>ВВЕДЕНИЕ</u>	5
<u>1 Анализ предметной области и обоснование технических решений</u>	11
<u>2 Элементная база и архитектура комплекса</u>	32
<u>3 Программная реализация</u>	55
<u>3.1 Программная реализация ведомого узла (STM32F103)</u>	55
<u>3.2 Программная реализация мастера опроса на ПК</u>	58
<u>3.3. Telegram-бот и взаимодействие с языковой моделью</u>	60
<u>3.4 Выводы</u>	62
<u>4 Результаты работы</u>	63
<u>4.1 Стенд</u>	63
<u>4.2 Сценарии испытаний</u>	64
<u>4.3 Результаты</u>	64
<u>4.4 Практическая польза от средств разработки</u>	68
<u>4.5 Сравнение с решениями на рынке</u>	69
<u>4.6 Ограничения прототипа и направления развития</u>	70
<u>4.7 Выводы</u>	70
<u>ЗАКЛЮЧЕНИЕ</u>	71
<u>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</u>	73
<u>ПРИЛОЖЕНИЕ А Листинг кода приложения микроконтроллера</u>	76
<u>ПРИЛОЖЕНИЕ Б Листинг кода приложения Telegram-бота</u>	90
<u>ПРИЛОЖЕНИЕ В Листинг кода приложения мастера опроса</u>	108

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

АРМ – автоматизированное рабочее место

АЦП – аналого-цифровой преобразователь

ИМКС – интеллектуальная микроконтроллерная система управления

МК – микроконтроллер

МКС – микроконтроллерная система

ПК – персональный компьютер

СУБД – система управления базами данных

API – программный интерфейс приложения

ARM – команда постановки на охрану (в протоколе обмена)

COM – последовательный порт операционной системы COM-порт

CRC16 – 16-разрядная циклическая избыточная сумма

DISARM – команда снятия с охраны

HTTP – протокол передачи гипертекста

LLM – большая языковая модель (Large Language Model)

PIN – персональный идентификационный номер (код доступа на узле)

RS-485 – промышленный последовательный интерфейс связи

UART / USART – универсальный асинхронный приёмопередатчик

USB – универсальная последовательная шина

ВВЕДЕНИЕ

Актуальность темы обусловлена потребностью в контролируемом по шине RS-485 распределённом охранном контуре с ясно определённым протоколом и открытой архитектурой прошивки на стороне ведомого устройства. Распространённые готовые решения часто имеют закрытый протокол обмена; Обзор типовых решений — по [2]. Отсутствие интеграции с другими подобными решениями и закрытое программное обеспечение затрудняет проверки и широкое внедрение устройств в низкобюджетный сегмент, а также недоступность демонстрации таких решений в магистерской работе. Использование отладочной платы на базе ядра Cortex-M3 в составе STM32F103 и единого бинарного протокола на микроконтроллере и на ПК позволяет добиться прозрачности и повторяемости измерений на линии.

Компоненты, которые включает любая система охранной сигнализации, представлены на рисунке 1.1.



Рисунок 1.1 – структура системы охранной сигнализации

Объект исследования – распределённые системы сбора охранной информации на базе микроконтроллерных ведомых узлов, связанных с централизованным мастером по последовательному интерфейсу RS-485.

Предмет исследования – архитектуры программно-аппаратного комплекса, протоколы обмена кадрами с контролем целостности, алгоритмы синхронизации и согласования состояния охранных зон, а также методы интеллектуального взаимодействия оператора с системой через локальный HTTP-интерфейс мастера и Telegram-бот с применением языковой модели.

Цель данной работы – разработка и исследование интеллектуальной микроконтроллерной системы управления для охранного комплекса, обеспечивающей централизованный опрос ведомых узлов, локальное и удалённое управление режимами охраны, Оформление работы — по [1], журналирование событий и обработку запросов оператора на естественном языке без нарушения единства источника достоверных данных о состоянии объекта.

Для достижения поставленной цели в магистерской работе решаются следующие задачи:

- анализ предметной области, требований к охранным комплексам,
- распределёнными микроконтроллерными модулями и существующих подходов к централизованному управлению и удалённому доступу,
- обоснование архитектуры программно-аппаратного комплекса, выбор элементной базы и средств разработки прошивки и прикладного программного обеспечения,
- разработка прошивки ведомого узла на микроконтроллере STM32F103 с обработкой сигналов датчиков и формированием байта статуса охранных зон,

- разработка согласованного протокола обмена с контролем целостности кадра по правилам CRC16 на стороне ведомого и мастера,
- создание программного модуля циклического опроса на персональном компьютере с журналированием событий и HTTP-интерфейсом для внешних клиентов,
- разработка приложения мониторинга и подсистемы удалённого доступа через Telegram-бот с локальной языковой моделью для оркестрации вызовов инструментов управления,
- проведение функциональных испытаний комплекса на учебно-лабораторном стенде при различных сценариях работы охранных зон, потери связи и удалённого управления.

Методы исследований. Для решения поставленных задач в магистерской работе применены следующие методы:

- системный анализ и сравнительный анализ предметной области,
- методы проектирования встраиваемых систем и многоуровневой программной архитектуры,
- методы формализации протоколов обмена и программной верификации целостности кадров,
- методы объектно-ориентированного и событийно-ориентированного проектирования прикладного программного обеспечения
- методы экспериментальной отладки и функционального тестирования программно-аппаратного прототипа,
- методы прикладного использования локальной языковой модели для маршрутизации пользовательских запросов к ограниченному набору API-операций мастера.

Научная новизна работы состоит в следующем:

- предложена архитектура интеллектуального охранного комплекса с единственным владельцем последовательного порта на стороне ПК, централизованной очередью команд и согласованным представлением состояния ведомых узлов для локального и удалённого операторских интерфейсов,

- разработан согласованный протокол обмена по RS-485 с симметричной обработкой кадров на ведомом узле и в программном мастере, обеспечивающий единообразную диагностику ошибок линии и сохранение последнего достоверного статуса узла,

- обоснован и реализован подход интеграции локальной языковой модели в подсистему удалённого доступа как оркестратора вызовов инструментов управления без передачи наружу непосредственного доступа к линии RS-485 и без подмены машинных данных протокола текстовыми утверждениями модели.

Внедрение интеллектуальной охранной сигнализации на базе микроконтроллера позволит добиться результатов, представленных на рисунке 1.2. Подходы к применению нейросетевых моделей в прикладных задачах обобщены в [20].



Рисунок 1.2 – Блок-схема результатов внедрения интеллектуальной охранной системы управления на базе микроконтроллера

Практическая значимость работы. Разработанный программно-аппаратный прототип может быть использован в составе учебно-лабораторного стенда кафедры для демонстрации принципов построения распределённых охранных систем, отладки протоколов последовательного обмена и интеграции средств прикладного искусственного интеллекта в локальные системы управления. Созданные модули прошивки, мастера опроса, приложения мониторинга и Telegram-бота применимы в лабораторных работах по дисциплинам «Вычислительные системы», «Микроконтроллеры и встраиваемые системы» и «Прикладной искусственный интеллект».

Достоверность результатов работы подтверждается согласованностью реализации протокола на микроконтроллере и на стороне мастера, воспроизводимостью экспериментов на учебно-лабораторном стенде, результатами функциональных испытаний сценариев опроса, постановки и снятия с охраны, а также проверкой эквивалентности локального и удалённого управления через HTTP-интерфейс мастера.

На защиту выносятся:

- архитектура интеллектуальной микроконтроллерной системы управления охранним комплексом с распределёнными ведомыми узлами, централизованным мастером на ПК и подсистемой удалённого доступа,
- протокол обмена с контролем целостности кадра и алгоритмы формирования и интерпретации байта статуса охранных зон,
- программно-аппаратный комплекс, включающий прошивку ведомого узла на STM32F103, программный мастер опроса, приложение мониторинга, HTTP-интерфейс и Telegram-бот с локальной языковой моделью Qwen 2.5-3B,
- результаты функциональных испытаний прототипа при опросе ведомых, выполнении команд управления и удалённом взаимодействии с оператором.

Результаты комплексных исследований подтверждают работоспособность предложенной архитектуры, возможность централизованного управления охранными зонами и целесообразность применения локальной языковой модели для упрощения взаимодействия оператора с системой при сохранении достоверности данных, получаемых от программного мастера.

1 Анализ предметной области и обоснование технических решений

В настоящее время обеспечение безопасности объектов различного назначения остаётся одной из важнейших задач для промышленных предприятий, коммерческих организаций и частных пользователей. Систематизация сведений о рынке охранных комплексов опирается на [2]. Рост числа угроз, связанных с несанкционированным проникновением на охраняемые территории, требует применения современных технических средств, способных обеспечивать непрерывный контроль состояния объекта, своевременное обнаружение тревожных событий и оперативное информирование пользователей.

Традиционные охранные системы строятся на основе датчиков обнаружения, контрольных панелей и средств оповещения. Несмотря на высокую надёжность таких решений, большинство из них выполняют преимущественно функции регистрации и передачи тревожных сигналов, не обеспечивая расширенной поддержки оператора при анализе поступающей информации и принятии управленческих решений.

Развитие микроконтроллерной техники и средств цифровой связи создаёт предпосылки для построения более гибких охранных комплексов с централизованным мониторингом и возможностью удалённого доступа к сведениям о состоянии объекта. Это обосновывает необходимость анализа существующих подходов к построению охранных систем и формулирования требований к разрабатываемой интеллектуальной микроконтроллерной системе управления охранным комплексом, что и составляет цель настоящей работы.

1.1 Анализ предметной области и требований к комплексу охраны

На практике охранную сигнализацию ставят, когда необходимо сочетание ранней индикации проникновения и сохранении работоспособности при временных помехах. В прототипе реализовано простое разделение: сложные

правила исполняются на стороне ПК, а обязанности микроконтроллера: быстрый отклик на контуре датчиков и прозрачное поведение линии RS-485.

Современные комплексы часто строятся по иерархии «шлейфы датчиков – приёмно-контрольный прибор – канал связи – пульт централизованного наблюдения». Состав систем охранной сигнализации рассмотрен по [2]. В рассматриваемой работе такая иерархия упрощена до «ведомый контроллер зоны – физическая шина – мастер ПК – человеко-машинный интерфейс и при необходимости мессенджер». Иерархия «шлейфы – ПКН – пульт» и особенности малых систем описаны в [3]. Такой сдвиг целесообразен там, где важна прозрачность формата кадра и скорость отладки.

Типовые извещатели для периметра включают магнитоконтактные пары на дверях и окнах, акустические и стеклобойные датчики. Классификация технических средств охранно-пожарной сигнализации приведена в [4]. В магистерской работе допущена имитация части событий по таймеру чтобы наглядно показать изменение битов статуса даже при отсутствии физического нарушения по линии.

Требования к устойчивости асинхронного обмена в учебном комплексе сводятся к однозначному выделению границ кадра и проверке целостности поля данных. В проекте используется фиксированный префикс синхронизации и CRC16, вычисляемая по одному и тому же правилу на стороне ведомого микроконтроллера и на стороне мастера ПК на согласованной области полей кадра. Такой подход обеспечивает переносимость реализации: корректный приём возможен при программном вычислении контрольной суммы без привязки к конкретному «железному» CRC-блоку, при условии идентичности полинома, порядка байт и области охвата. С архитектурной точки зрения ошибка контрольной суммы или несогласованная структура кадра классифицируются как повреждение транспортного уровня: ведомый отбраковывает неверный запрос, а

мастер переводит соответствующий адрес в диагностическое состояние наряду с таймаутом ответа, что поддерживает наблюдаемость линии при отладке и эксплуатации стенда. Нормативная классификация систем охраны — по [5]; порядок проектирования — по [6].

1.2 Особенности распределённого контура наблюдения

Наличие единственного мастера на линии снимает столкновения передачи в полудуплексном сценарии характерном для преобразователей интерфейса RS-232 к RS-485 на стороне ПК, где приёмник и передатчик не работают одновременно на одной паре условным полудуплексом. Особенности полудуплексного обмена по RS-485 рассмотрены в [8].

Требование задержки ответов микроконтроллера определяется выбором времени опроса в программе ПК если линии не находятся в аварии. Таймаут ожидания ответа и период циклических опросов являются параметрами настройки в пользовательском интерфейсе.

Для обеспечения преемственности между анализом предметной области и дальнейшим инженерным описанием комплекса в работе фиксируется набор архитектурных допущений, определяющих состав функций и распределение данных. Ведомый узел обеспечивает первичную фиксацию состояния охранных контуров и локальные сценарии взаимодействия с оператором на объекте, а обмен по линии RS-485 выполняется в режиме единственного мастера, инициирующего обмен со стороны персональной рабочей станции. Сводное состояние участника линии передаётся в компактном виде байтом (полем) статуса, смысл которого согласован между программами ведомого и мастера; расшифровки для человека-оператора формируются на стороне ПК.

Журналирование событий, требующее длительного хранения и удобного просмотра, отнесено к ресурсам рабочей станции, без дублирования одинаковой политики записи в нескольких независимых компонентах. Удалённые команды наблюдения реализуются через локальный интерфейс доступа к мастеру

(прикладной HTTP-сервис процесса мастера), поэтому внешние клиенты (в том числе мессенджер-бот) не имеют права напрямую захватывать последовательный порт шины и не становятся параллельным источником управления каналом.

Перечисленные решения задают архитектурную «рамку», внутри которой далее обоснованы технические элементы построения микроконтроллерной системы, протокол обмена и программное обеспечение стороны ПК.

1.3 Связность с задачами дальнейших разделов

Анализ из этого раздела даёт отправные точные определения зон охранения и понятие адреса ведомого на шине, которая далее переходит по протокольным кодам состояния между подсистемами системы наблюдая за одной и той же битовой раскладкой байта статуса.

С момента появления первых однокристальных микроконтроллеров в 20 веке, был произведен качественный скачок в архитектуре ядра МК и системе команд. Появление энергонезависимой памяти на кристалле, развитие интерфейсов связи и разработка конвейера производства, позволили широко применить МК в технически сложных решениях с небольшими затратами как на стадии разработки проекта, так и на стадии поддержания актуальности уже существующих систем. Разработаны десятки компиляторов, среди которых поддерживается не только программирование на низком уровне, что позволило привлечь в эту отрасль много новых специалистов ЯП высокого уровня. Современные МК, благодаря такому тесному сотрудничеству производителей и разработчиков, выполняют большинство команд за один такт, что ещё 15 лет назад было невозможно. На данный момент наиболее широкое распространение получили 32-разрядные для технически сложных решений, к которым предъявляют высокие требования по точности и отказоустойчивости.

Общие принципы построения МКС

Проектирование микроконтроллерной системы (МКС) для охранного комплекса состоит из трех этапов. Этапы проектирования МКС согласованы с [7].:

- Выбор элементов и составление структурной схемы.
- Оценка занимаемой памяти и составления протокола обмена, с соответствующими кодами и адресами устройств.
- Составление и отладка программы.

Основными данными для проектирования являются: описание алгоритма решаемой задачи, характеристики и уровни сигналов, требования к помехоустойчивости, быстродействие, энергопотребление и надежность. Во многих случаях указываются рекомендации по применению определенной элементной базы, которые в процессе проектирования проверяются на соответствие требований и вносятся коррективы при анализе системы.

На первом этапе выбирается микроконтроллер (МК). Его выбор обуславливает специфика алгоритмов и протокол обмена.

Поскольку в системах охранного комплекса важнейшим фактором является простота сопряжения с объектами управления, особых требований к тактовой частоте не предъявляется, однако, электромеханические и температурные диапазоны должны быть достаточно высоки. В таких системах во многих случаях достаточно использовать 32-разрядные МК семейства STM32 серий F103, которые позволяют строить простые и экономичные системы. Не маловажным фактором является имеющаяся возможность использовать интегрированную среду разработки (IDE) от производителя, что ускоряет процесс создания системы, а обширная библиотека документации даёт возможность оптимизировать код.

На первом этапе так же выбираются модули, обеспечивающие связь МК с внешними устройствами. Производится проверка уровней сигнала,

сопротивления и оценка канала связи. Затем составляется функциональная и принципиальная схема.

На втором этапе выделяется область памяти для хранения основной программы, констант, макроопределений и функций. Выделяется память стека для очередей обмена, а также конфигурации системы.

На третьем этапе составляется программа работы МКС. При программировании используется язык С, как наиболее универсальное средство достижения оптимального быстродействия и взаимодействия с существующими библиотеками для разработки ПО.

Эффективным средством разработки МКС является метод декомпозиции. Декомпозиция встроенной системы на модули соответствует подходу [16], при котором вся задача разделяется на небольшие функциональные модули, каждый из которых можно анализировать, отлаживать и оптимизировать независимо от других. При выполнении программы, МК передает управление от одного модуля к другому, реализуя завершённую структуру, образуя общий алгоритм работы.

При разработке ПО для МК могут быть использованы два способа организации: монолитный и модульный. При монолитном, вся программа разрабатывается как единое целое, при модульном – строится из отдельных блоков. Отдельные фрагменты оформляются в виде подпрограмм и вызываются по мере необходимости. Подпрограмма должна обладать следующими свойствами: выполнять законченную процедуру обработки данных и иметь только один вход и один выход.

Использование процедур позволяет сократить время разработки, отладки и требуемой емкости памяти, что, как правило, сокращает время разработки и срок поддержания актуальности системы.

В рамках магистерской работы разрабатывается интеллектуальная микроконтроллерная система управления (ИМКС) охранным комплексом; обобщённая структура приведена на рисунке 1. В состав входят

микроконтроллерный модуль охранной зоны, средства связи между модулем и персональным компьютером (при необходимости через преобразователь интерфейса типа UART/RS-485), а также аппаратно-программные средства на стороне ПК для циклического опроса ведомых устройств, ведения журнала событий и операторского наблюдения. Микроконтроллер с заданным периодом обрабатывает сигналы на входных линиях, формирует выходные и по запросам мастера участвует в обмене данными по согласованному протоколу. Через пульт управления оператор задаёт режим работы узла охраны и получает информацию о его состоянии; при этом допускается и удалённый доступ к сведениям об объекте и к командам постановки/снятия охраны через телеграм-бот, обращённый не к последовательному порту напрямую, а к сервису управления уже запущенным на ПК мастером опроса.

1.4 Обоснование элементной базы ведомого узла

Элементная база ведомого узла формируется как согласованный набор компонентов, обеспечивающих одновременно работоспособность по заданному протоколу, устойчивость к типовым помехам на линии и воспроизводимость результата на стенде. Выбор конкретного набора устройств и простейшей обвязки учебного ведомого узла, с которым удаётся повторить результат на стенде: воспроизвести обмен по протоколу, позволяет наблюдать типовые сбои линии, помехи, обрывы, временные пропуски ответов, и при этом понимать, к какому узлу схемы это относится. По линии RS-485 действует уже принятое решение архитектуры комплекса: один мастер, и связь выполняется по очереди. Передача и приём на одной паре проводников, поэтому выбор элементов «вокруг» UART и трансивера должен быть согласован с программными задержками переключения направления передачи. Ниже перечислены типовые части узла и линии — питание, преобразователь RS-485, цепочки входов и индикации, кабельное соединение с ПК через USB–UART-адаптер — в том виде,

как это нужно именно для отладки и приёмки учебной постановки, без обзора устройств, которые в данной реализации не использовались.

Питание и устойчивость по электропитанию. Практическая реализация ведомого модуля предполагает питание от стабилизированного источника постоянного напряжения, соответствующего допустимому диапазону микроконтроллера и приёмопередатчика интерфейса RS-485. Целесообразно разделять цепи «силовой/внешней» части и «чувствительной» аналоговой цепи канала «стекло» по правилам разводки: общая точка заземления, локальные блокировочные конденсаторы керамики у выводов питания кристалла, отсутствие длинных «петлёр» возврата тока рядом с входами АЦП. Для учебного стенда допускается линейный стабилизатор с малым уровнем пульсаций; при необходимости наращивания устойчивости возможна двухступенчатая схема (предварительное понижение/грубая фильтрация — точный стабилизатор возле МК).

Цепочка RS-485: приёмопередатчик и управление направлением. Физический уровень выполняется внешним приёмопередатчиком, совместимым по напряжению логики с используемыми выводами UART микроконтроллера. Выбор конкретной микросхемы (или её функциональных аналогов) определяется допустимой скоростью, режимом полудуплекса и требованиями к защите. Управление направлением передачи (линии типа DE/RE) должно быть согласовано с программной моделью: перевод в режим передачи непосредственно перед выдачей кадра и возвращение в приём по завершении передачи с учётом задержки «хвоста» бита и времени включения передатчика. На стенде, как правило, не требуется гальванической развязки портов, однако при переносе решения на объект с длинными линиями и разными «землями» целесообразно отдельно обосновывать изоляцию и ограничения по топологии.

Разделение, согласование линии и помехоустойчивость. Рекомендации по согласованию линии RS-485 уточнены по [8], [9]. Для отрезка линии,

используемого в лабораторной постановке, применяется классическая рекомендация: согласующие резисторы на концах сегмента шины и контроль волновых отражений при выбранной скорости. Наличие ответвлений и несогласованных отрезков ухудшает форму фронтов и повышает вероятность битовых ошибок, что косвенно проявляется в таймаутах ответа на стороне мастера. Для снижения риска выбросов на длинной кабельной линии на стенде допускается применение ограничительных цепочек (например, TVS-защита на линии A/B в рамках допустимых уровней), при этом важно не нарушить согласование и не внести чрезмерную ёмкостную нагрузку.

Дискретные входы охранных зон. Согласование уровней дискретных сигналов — см. [17]. Контактные шлейфы и дискретные датчики подключаются к GPIO с явно заданной политикой внутренней подтяжки и фиксированным уровнем срабатывания. В учебной конфигурации допускается программная защита от повторных срабатываний и-фильтрация дребезга контактов; при необходимости для подавления ВЧ-помех на входе используются RC-цепи. Важное архитектурное требование: локальное состояние зон сохраняется на микроконтроллере как первичная «аппаратная истина» до момента успешной доставки сводной информации по протоколу.

Органы местного ввода и индикации: матричная клавиатура и индикатор относятся к элементной базе как к подсистеме человеко-машинного интерфейса на объекте. Сканирование клавиатуры реализуется распределением линий «строб/опрос», при этом количество задействованных GPIO должно быть согласовано с корпусом микроконтроллера и ограничениями по объёму программы. Цепочки индикации (сегментные линии, токоограничивающие резисторы или драйверы при необходимости) задают нагрузку на порты и влияют на разводку общего провода: для стенда допустима упрощённая схема при условии соблюдения допустимых токов по выводам.

Аналоговая часть: использование встроенного АЦП микроконтроллера предполагает согласование динамического диапазона сигнала с допустимым входным диапазоном АЦП, а также выбор частоты дискретизации и окна усреднения/пороговой обработки на прикладном уровне. Элементная база здесь включает не только «сенсорный» узел (в учебной постановке — имитатор/пьезоэлемент согласно схеме стенда), но и цепи согласования: делители, фильтры нижних частот при необходимости, защитные ограничители, не допускающие выход за пределы «штырьков» МК. Заявленный программный порог должен быть трактуем как инженерный компромисс между чувствительностью и ложными срабатываниями на фоне шумов питания и наводок.

Связь рабочей станции с линией. На стороне персонального узла применяется преобразователь USB–UART и далее — отдельный преобразователь уровня/интерфейса к RS-485 (либо комбинированный адаптер), что определяет элементную базу «последней мили» до шины. С точки зрения архитектуры комплекса критично не столько конкретное название адаптера, сколько единоличное владение COM-портом процессом мастера: параллельное открытие порта другим приложением приводит к коллизиям и «рваным» кадрам, что при проектировании стенда должно быть отражено в инструкции по эксплуатации и составе ПО.

Итоговое обоснование объёма перечня. Расширенное описание элементной базы необходимо не для «увеличения количества компонентов», а для демонстрации трассируемости: каждый класс узлов (питание, RS-485, дискретные входы, аналоговый канал, ПК-адаптер) явно привязан к уровням архитектуры комплекса и к требованиям протокола и прошивки. Такой подход соответствует инженерной практике оформления пояснительной записки, где состав аппаратуры фиксируется с достаточной детализацией для

воспроизведения результата и последующей приёмки по функциональным критериям.

1.5 Выбор микроконтроллера ведомого узла

В контексте ведомого охранного узла под выбором микроконтроллера понимается не только маркировка кристалла, а закрепление минимально достаточного набора технических параметров изделия STM32F103, который одновременно обеспечивает: аппаратный UART под внешний RS-485 трансивер; требуемое число дискретных линий под зоны и сопряжение с матричной клавиатурой/индикатором в учебной конфигурации; наличие встроенного АЦП для канала имитации акустического события по стеклу; достаточный объём SRAM/FLASH для буфера приёмника, машины состояний охраны, вычисления CRC16 по правилу, принятому в протоколе; возможность воспроизводимой отладки по SWD средствами ST-Link без смены философии разработки. Архитектура и периферия микроконтроллера — по [10]; назначение выводов USART — в [13]. Такая формулировка нужна затем, чтобы выбор мог быть проверен при приёмке: «проверено по перечню периферии и ограничениям проекта», а не «выбрано по общему ощущению рынка».

Связь выбора кристалла с принятыми архитектурными решениями. Архитектура комплекса закрепляет единственного мастера на линии и передачу сводного статуса кадрами с фиксированной структурой. Следовательно, микроконтроллер ведомого должен устойчиво обрабатывать приём с выделением синхронизации, разбор адреса и команды, формирование ответа и возврат линии в режим приёма согласованно с DE/RE внешнего трансивера. Это накладывает требования не «к максимальной частоте», а к детерминированности обработки прерываний UART, к предсказуемой длительности критических участков кода и к запасу по тактам на операции 16- и 32-битной арифметики при CRC. В учебной постановке запас по частоте ядра Arm Cortex-M3 на практике проявляется как

запас на усложнение фильтрации по «стеклу», расширение числа зон при сохранении корректных таймингов линии.

Корпус, выводы и учебная воспроизводимость. Для стенда предпочтительны варианты с числом выводов, позволяющим развести линии зон, клавиатуры, индикации, UART и управления направлением RS-485 без «ломки» схемы при повторении работы другим исполнителем. Указание на типовой корпус LQFP64, в связке с конкретной моделью вроде STM32F103C8T6 служит инженерным якорем: такой конкретный выбор позволяет однозначно заказать компонент и получить ожидаемую разводку в STM32CubeMX. При этом отдельно подчёркивается, что не все ресурсы кристалла обязаны быть задействованы: запас по периферии и памяти допустим и соответствует нормальной практике прототипирования.

Память программы и данных как часть элементной базы. В элементной базе целесообразно учитывать не только объём FLASH под код HAL-инициализацию и прикладную логику, но и SRAM под рабочие буферы UART, промежуточные структуры состояния и таблицы/константы, если они используются при разборе кадра и журнализации локальных событий до выдачи на шину. Для прототипа важно явно указать: требование к памяти выводится из слоя транспортного кадра и слоя состояний охраны, а не из «желания поставить больше библиотек». Наличие аппаратного умножителя у Cortex-M3 снижает стоимость программной реализации контроля целостности по сравнению с упрощёнными 8-битными ядрами без быстрой 32-битной арифметики - без перехода к сравнительному обзору «всех возможных производителей».

Электрическая совместимость и ограничения ввода-вывода. Частью обоснования является согласование уровней: логика трансивера RS-485 должна находиться в одном номинале с доменом GPIO/UART контроллера. Это влияет на выбор конкретной модификации платы/кристалла и на класс внешних преобразователей. Дополнительно фиксируется, что дискретные входы и линии

сканирования клавиатуры должны быть укладывается в допустимые по документации токи подтяжек и конфигурации «вход с подтяжкой/без».

Легкая поддерживаемость, как дополнительный фактор выбора. Для результата существенна не только схема, но и воспроизводимая цепочка: генерация карты периферии в CubeMX и сборка/отладка в STM32CubeIDE с GCC и ST-Link. Выбор семейства STM32F103 в связке ST-tooling является частью элементной базы в расширенном смысле: это снижает риск «уникальной» среды, которую нельзя восстановить на другом ПК. Переход на иной МК возможен только при переоценке драйверов трансивера, таймерных зависимостей и ресурсоёмкости кадрового парсера - то есть решение связано уже с содержательной переработкой протокола и прошивки.

Таким образом, STM32F103 выступает не просто как «современное ядро», а как техническое решение, удовлетворяющее согласованному набору ограничений линии RS-485, периферийного набора охранного узла и воспроизводимости инструментальной цепочки разработки, что является достаточным обоснованием элементной базы микроконтроллера для целей настоящего дипломного проекта.

Для стенда, энергопотребление микроконтроллера не является доминирующим критерием выбора кристалла: узел может питаться от стабилизированного источника лабораторного типа или от штатного блока питания, а контролируемые интервалы опроса на линии RS-485 задаются протокольным ритмом мастера, а не задачами сверх длительной автономии.

Тем не менее в составе элементной базы разумно зафиксировать типовые режимы выполнения:

- при активной охране преобладают событийно-управляемые переходы (готовность UART к приёму кадра, локальное обслуживание дискретных входов и клавиатуры, опрос канала АЦП в соответствии с выбранной политикой фильтрации),

- переходы в младшие режимы останова либо сна в рамках дипломного эталона рассматриваются как опциональное усложнение: оно может сократить среднее потребление, но должно быть согласовано с сохранением обнаружения прихода синхробайта на линию и времени выхода периферии UART в работоспособное состояние.

С практической точки зрения приемлемо разделить требование «доступности шины для мастера» и требование «минимизации тока»: на стенде приоритет остаётся за предсказуемостью таймерных характеристик и отладки, а параметры экономии задаются в ограниченном диапазоне, не противоречащем времени реакции на протокольные команды.

Критерии выбора сфокусированы на соответствии архитектуре комплекса, а не на обзоре всего рынка микроконтроллеров:

- наличие аппаратного UART для сопряжения с внешним преобразователем RS-485 при заданной скорости обмена;
- достаточное число линий ввода-вывода для дискретных зон, индикации и матричной клавиатуры;
- встроенный АЦП для реализации канала «стекло» без внешнего оцифровывающего узла в учебной постановке;
- запас вычислительной мощности для буферизации приёма, разбора кадров и вычисления CRC16 на прикладном уровне.

По сравнению с рядом 8-разрядных ядер [11] без аппаратного умножения реализация контрольных сумм и работы с 16-битными полями кадра на STM32 оказывается проще за счёт 32-разрядной арифметики; при этом не все ресурсы кристалла обязаны быть задействованы в учебной конфигурации.

Надёжность ведомого узла в учебной постановке опирается на типовые аппаратные механизмы кристалла и на программное восстановление после сбоев. К аппаратной части относятся корректный «холодный» сброс по цепочке NRST,

штатный сценарий Power-on reset, целесообразность использования внутреннего контроля понижения питания (при наличии варианта кристалла/настройки) для предотвращения «защёлкивания» периферии на неопределённых уровнях при просадках.

Сторожевой таймер, будучи активированным в финальной сборке, должен восприниматься как противодействие зависаниям главного цикла при помехах по шине или ошибках указателей при разборе кадра при условии, что период перезагрузки согласован с безопасной политикой индикации и не провоцирует ложного ощущения «снятия» охраны на время переинициализации. Выбор времени сторожевого таймера и точек перезапуска таймера фиксируется на уровне прошивки и испытаний стенда.

Категория «защёлка» при сбое питания относится к поведению выходов и линий направления RS-485. При проектировании элементной базы учитываются подтягивающие состояния, начальное состояние портов до выполнения кода инициализации через настройку в генераторе кода, и, возможное, кратковременное неверное включение передатчика при старте. Для стенда эти явления ограничиваются как штатной последовательностью инициализации трансивера, так и кабельным подключением только к эталонной линии.

Полное резервирование микроконтроллера в классе промышленных систем в проект явно не вводится: обоснованием является учебный масштаб и одиночная линейная топология под единственного мастера; при этом перечень механизмов в настоящем подразделе фиксирует границу между «демонстрационным» уровнем и инженерно-осмысленной проработкой устойчивости узла.

1.6 Средства разработки и отладки прошивки

Разработка и отладка прошивки ведётся официальной цепочкой STMicroelectronics. Практика разработки на STM32 в CubeMX/HAL изложена в [12]. Конфигурирование тактирования, мультиплексирование выводов и первичную инициализацию периферии по шаблону HAL формируют в утилите

STM32CubeMX; интегрированная среда STM32CubeIDE объединяет редактирование исходного текста, сборку GCC for ARM из поставки инструментария производителя, управление сборочными конфигурациями и отладку через ST-Link по интерфейсу SWD.

Применение STM32CubeMX в работе упрощается тем, что настраиваются «один раз и документируются» параметры тактирования системы от внешнего кварца или встроенных источников, распределение ножек порта между UART, входами охранных зон, строками матрицы клавиатуры и сигналами управления приёмопередатчиком RS-485. Сгенерированный каркас HAL-инициализации задаёт воспроизводимое начальное состояние аппаратуры и снимает типовую ошибку ручной периферии — когда часть параметров задаётся в коде приложения случайными константами. Поэтому утилита рассматривается здесь как инженерный инструмент фиксации конфигурации проекта.

На этапе создания проекта в STM32CubeMX выбирается микроконтроллер STM32F103, задаётся тактирование и назначение выводов под периферию ведомого узла. Аппаратная конфигурация задаётся до написания прикладной логики охраны и может быть воспроизведена на другом рабочем месте по тому же файлу проекта CubeMX. На рисунке 1.6 показан фрагмент интерфейса утилиты: окно выбора семейства/модели МК и начальная конфигурация проекта (или экран Pinout & Configuration с включённым USART и линиями, используемыми в стенде).

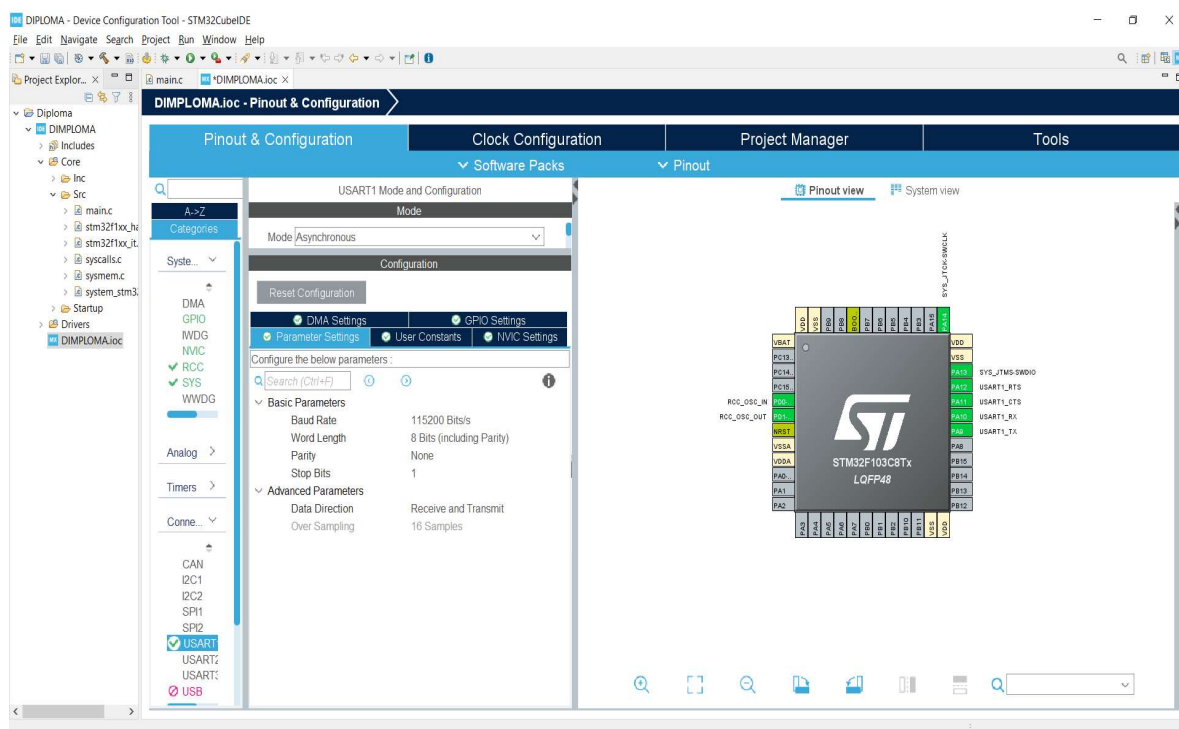


Рисунок 1.6 – Настройка микроконтроллера и периферии в STM32CubeMX

После генерации кода инициализации проект переносится в STM32CubeIDE, где выполняются редактирование модулей прошивки, сборка и загрузка образа в память микроконтроллера. На рисунке 1.6а приведён типовой вид интегрированной среды: дерево проекта с каталогами Core/Src, Core/Inc, открытый исходный файл и панель сборки/отладки. Наличие единого окна IDE подтверждает, что цепочка «конфигурация → код HAL → прикладные модули → бинарный образ» не разорвана между несколькими несогласованными инструментами.

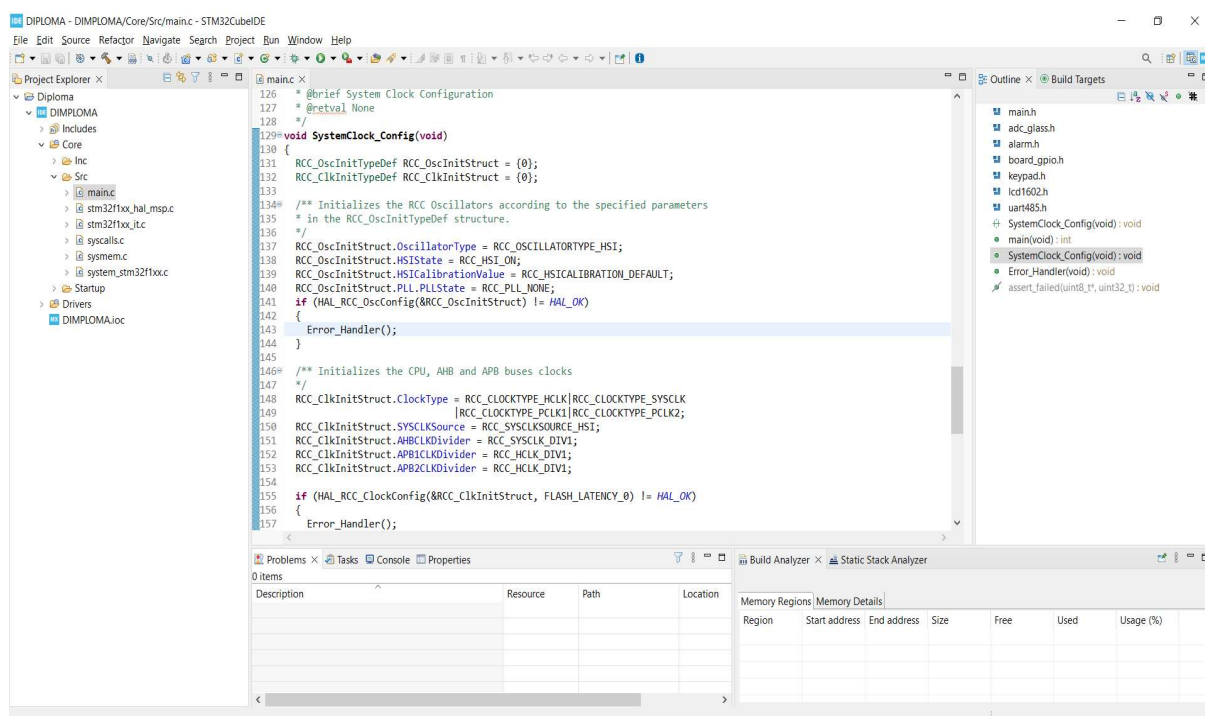


Рисунок 1.6а – Интерфейс STM32CubeIDE при работе с проектом прошивки ведомого узла

STM32CubeIDE является практической точкой сборки. Модель программирования ядра Cortex-M3 — по [14]; программирование на C — [15].: в одном рабочем окне доступны сборка «toolchain GCC for ARM», подключённые библиотеки абстрагирования, формирование бинарного образа загрузки и пошаговая отладка с просмотром регистров и памяти. Отладчик ST-Link через SWD позволяет проверять сценарии «первый запуск после сброса», корректность настройки приёмника UART в связке с внешним трансивером RS-485, а также временные параметры переходных режимов охраны без постоянной перепрошивки. Для учебной практики важное следствие — повторяемость: восстановить сборку можно на другом персональном компьютере при установке той же ветви инструментов ST, без перевода проекта на «уникальный» закрытый компилятор.

Отдельно стоит подчеркнуть, что выбор данной связки связан именно со строением проекта как STM32-ориентированной реализации и не должен

пониматься как утверждение абсолютной незаменимости среды. При переходе на другого производителя сохранится потребность пересмотреть карту проектных файлов, HAL-слой и иногда настройку линковщика под другой официальный поддерживаемый пакет.

1.7 Обоснование выбора программных средств прикладного уровня

Разработка прикладного уровня дипломного комплекса — программного мастера опроса, модулей журналирования, локального HTTP-интерфейса и Telegram-бота – выполняется на языке Python и требует среды, в которой удобно вести несколько связанных модулей одного проекта, быстро запускать и отлаживать приложение, контролировать зависимости и согласованность настроек. Реализация прикладного ПО на Python опирается на [19]; структурирование модулей – [18]. В отличие от средств прошивки ведомого узла, здесь решаются задачи прикладной интеграции компонентов на персональном компьютере; поэтому выбор IDE определяется прежде всего удобством работы с многофайловым Python-проектом, а не с низкоуровневой отладкой микроконтроллера.

При сравнении вариантов учитывались следующие критерии: поддержка Python и виртуальных окружений; подсветка синтаксиса и навигация по каталогам `pc_master/` и `chatbot/`; встроенный терминал для запуска `python -m pc_master` и `telegram_bot.py`; возможность поиска по проекту при согласовании констант протокола и параметров конфигурации; наличие средств контроля версий и расширений для работы с `.env`, JSON и HTTP-запросами; доступность и распространённость в учебной среде кафедры.

В качестве основной среды разработки прикладного ПО выбрана Visual Studio Code. Выбор IDE для Python-компонентов согласован с возможностями, описанными в [19]. Такой выбор обоснован тем, что среда бесплатна для учебного использования, не требует тяжёлой установки, поддерживает

расширение Python и позволяет в одном окне одновременно редактировать модули мастера и бота, запускать их из встроенного терминала и оперативно локализовать ошибки конфигурации (порт, адреса опроса, переменные окружения) на этапе отладки стенда. Использование IDE снижает риск несогласованности между частями прикладного комплекса и ускоряет итерации при доработке HTTP-слоя и Telegram-канала без изменения прошивки ведомого узла.

На рисунке 1.7 приведён типовой вид выбранной среды разработки при открытом проекте комплекса: дерево каталогов репозитория с модулями pc_master и chatbot, окно редактора исходного файла и панель терминала для запуска прикладных компонентов.

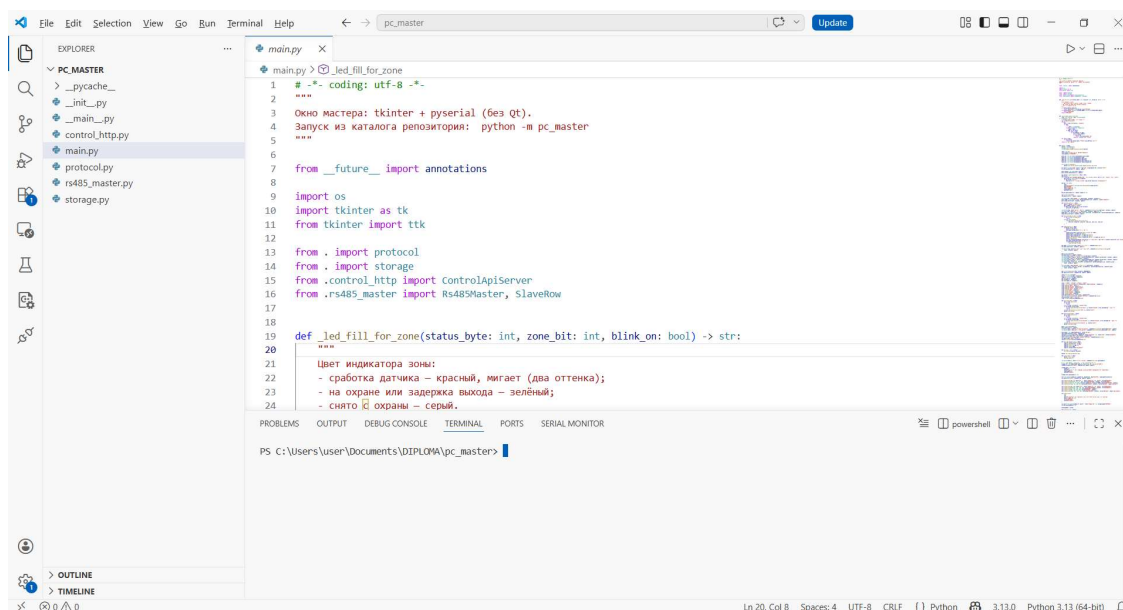


Рисунок 1.7 – Интерфейс интегрированной среды разработки

Таким образом, для прикладного уровня зафиксирован выбор Visual Studio Code как основной IDE, обеспечивающей разработку, запуск и отладку Python-компонентов комплекса.

1.8 Выводы

В результате анализа предметной области охранных комплексов с распределёнными микроконтроллерными ведомыми узлами были

сформулированы требования к разрабатываемой интеллектуальной микроконтроллерной системе управления. Для учебно-исследовательского прототипа целесообразна архитектура с централизованным опросом со стороны персонального компьютера и разделением функций между полевым узлом (быстрый отклик на контуре датчиков) и хост-уровнем (журналирование, мониторинг и удалённый доступ оператора). На этой основе зафиксированы функциональные и эксплуатационные требования к комплексу, которые определяют состав подсистем и границы ответственности программных компонентов в последующих разделах.

По результатам анализа обоснован выбор элементной базы и технических средств разработки. Для ведомого узла приняты микроконтроллер STM32F103, интерфейс RS-485 и периферия, необходимая для реализации охранных зон на стенде; для прошивки — официальная цепочка STM32CubeMX и STM32CubeIDE со сборкой GCC for ARM и отладкой через ST-Link (SWD). Для прикладного уровня на ПК выбраны Python и интегрированная среда разработки, обеспечивающая сопровождение модулей мастера опроса, журналирования, локального HTTP-интерфейса и Telegram-бота. Таким образом, на этапе анализа предметной области зафиксирован согласованный набор аппаратных и программных решений.

2 Элементная база и архитектура комплекса

В этом разделе комплекс рассматривается не как набор аппаратных компонентов и программных средств, а как целостная модель взаимодействия ведомого узла, последовательной линии связи, программного мастера на персональном компьютере и каналов операторского доступа. В разделе последовательно вводятся многоуровневая архитектура комплекса, логическое распределение функций между подсистемами, формальная спецификация протокола обмена кадрами с контролем целостности, семантическая модель байта статуса охранных зон, алгоритмы конечного автомата режимов охраны ведомого узла, циклического опроса со стороны мастера и обобщённый алгоритм работы ИМКС в целом.

Этот раздел отделяет проектное и алгоритмическое описание от программной реализации. На этом уровне фиксируются состав и смысл полей протокола, правила переходов между режимами охраны, порядок опроса адресов и реакция комплекса на типовые ошибки линии и граничные ситуации эксплуатации.

В этом разделе описывается, из каких частей собран охранный комплекс и как эти части между собой согласованы. Основное внимание уделяется разделению функций: на объекте остаются задачи локального наблюдения за входными цепями и безопасного поведения исполнительных элементов узла при смене режимов охраны; на линии связи задаётся порядок обмена «один мастер и несколько ведомых»; на персональной рабочей станции сосредоточены сбор и отображение сводной информации, ведение журнала длительного хранения и единственный доступ к аппаратному последовательному каналу, через который выполняются опросы и команды управления.

Такое разделение облегчает дальнейшее описание протокола, программы мастера и программ интеграции: читатель заранее понимает, какие свойства считаются «естественными» для узла охраны, где проходит граница между

полевым устройством и компьютером, и почему внешнее подключение (например, через мессенджер-бот или вспомогательные средства человеко-машинного диалога) не должно обходить эту границу и конкурировать за ресурс связи напрямую.

2.1 Многоуровневая архитектура комплекса

Разработку интеллектуальной микроконтроллерной системы управления охранным комплексом целесообразно рассматривать как совокупность технологических уровней, согласованных по информационным потокам и распределению функций между полевым ведомым узлом и персональным компьютером оператора. Такой подход позволяет отделить аппаратную привязку к объекту охраны от правил обмена по линии связи, прикладной логики режимов охраны и каналов представления данных пользователю, что соответствует требованиям, сформулированным в разделе 1, и создаёт основу для последующего описания протокола и алгоритмов. Многоуровневое представление комплекса согласовано с [16].

Физический уровень охватывает полевые охранные модули, непосредственно связанные с объектом защиты: контактные и аналоговые каналы датчиков, органы локального управления и индикации на узле, а также аппаратные средства сопряжения с линией связи. На этом уровне формируется первичная информация о состоянии охраняемых зон до её упаковки в протокольные структуры.

Канальный уровень задаёт топологию «общая двухпроводная последовательная шина RS-485». Промышленные сети и интерфейс RS-485 – по [8] и полудуплексный режим обмена при единственном мастере, инициирующем передачу кадров со стороны персонального компьютера. Здесь фиксируются ограничения среды: одновременная активная передача от нескольких участников линии не допускается, а доступность ведомого узла определяется фактом корректного ответа в отведённый интервал времени.

Транспортный уровень определяет машинную форму кадра: синхронизирующий префикс, адрес ведомого, поле команды, длина и содержание полезной нагрузки, а также контрольная сумма CRC16, вычисляемая по одному правилу на микроконтроллере и на стороне мастера. Ошибки этого уровня (повреждение кадра, несовпадение CRC) относятся к достоверности доставки данных, а не к смыслу режимов охраны.

Прикладной уровень разделён на две функциональные части. На ведомом узле реализуются конечный автомат охранности, формирование сводного байта статуса и выполнение команд постановки на охрану и снятия после разбора принятого транспортного кадра. На хостовом узле (персональный компьютер) сосредоточены единственное владение открытым COM-портом через USB-преобразователь, циклический опрос диапазона адресов ведомых, контроль таймаутов ответа, приоритетная выдача команд управления от оператора или удалённого клиента и накопление табличной модели наблюдений по состоянию линии.

Уровень представления включает настольное приложение мониторинга (АРМ), работающее в том же процессе, что и объект мастера, а также локальный HTTP-сервис для внешних клиентов (Telegram-бот, инструменты языковой модели) без открытия последовательного порта вторым процессом. На этом уровне формируются человекочитаемые представления статуса и журнала; достоверные поля режима и связи восходят к данным прикладного уровня мастера, а не подменяются текстовыми утверждениями внешних интерфейсов. Совокупность перечисленных технологических уровней и направление основных информационных потоков между ними представлены на рисунке 2.1.

Представленное разделение используется далее при формализации протокола обмена, описании семантики байта статуса и алгоритмов функционирования ИМКС.



Рисунок 2.1 – Многоуровневая архитектура комплекса охранной сигнализации

2.2 Логическая структура комплекса и распределение ролей

Если п. 2.1 фиксирует технологические уровни обработки информации, то в настоящем пункте задаётся логическая модель комплекса: какие подсистемы существуют, какие функции каждая из них выполняет и какие границы ответственности между ними не должны нарушаться при эксплуатации. Такое описание необходимо для однозначного понимания того, где формируется «аппаратная истина» о состоянии охранных зон, где принимаются решения о доступности узла на линии и через какие интерфейсы оператор получает команды управления и сведения о событиях. Обоснование выбора аппаратных и программных средств приводилось ранее; здесь рассматривается только распределение ролей в принятой архитектуре.

Архитектурную целостность комплекса обеспечивает явная фиксация авторитетности и границ доверия между подсистемами. Ни одна внешняя

клиентская оболочка не должна обходить мастер опроса при доступе к последовательному каналу; ни один текстовый канал (в том числе с участием языковой модели) не должен подменять собой машинные данные протокола и журнала.

Микроконтроллерный модуль охранной зоны (ведомый узел) выступает первичным источником аппаратной информации на объекте: состояние контактных и аналого-цифровых каналов датчиков, локально заданные временные ограничения переходных режимов (в том числе задержка на выход), сценарии взаимодействия с оператором через пульт и PIN-доступ там, где это предусмотрено проектом. Ведомый узел отвечает на адресованные кадры по линии RS-485 и формирует сводный байт статуса; он не инициирует опрос шины и не ведёт централизованный журнал длительного хранения.

Программный модуль мастера опроса на персональном компьютере является единственным активным участником последовательного обмена со стороны хоста: владеет СОМ-портом, выполняет циклический опрос диапазона адресов, контролирует таймауты ответа, ставит в очередь команды постановки на охрану и снятия. Мастер определяет, считается ли ведомый «онлайн», фиксирует время последней успешной транзакции и формирует человекочитаемые расшифровки байта статуса для операторских интерфейсов. Состояние линии и таблица ведомых на стороне ПК являются авторитетными для мониторинга и удалённого доступа при исправной работе мастера.

Подсистема журналирования, связанная с мастером, при включённом режиме учёта обеспечивает долговременное хранение строковых сообщений о событиях на линии (опрос, команды, таймауты, диагностика). Запись выполняется по единой политике через мастер; Telegram-бот и приложение мониторинга читают журнал, но не создают параллельный несогласованный источник событий.

Приложение мониторинга (АРМ) работает в том же процессе, что и объект мастера, и предназначено для локального оператора: отображение таблицы ведомых, индикация зон, выдача команд ARM/DISARM, просмотр текущего журнала. АРМ не открывает СОМ-порт самостоятельно и не дублирует низкоуровневый обмен кадрами.

Клиент удалённого доступа (Telegram-бот) не обращается к последовательному порту напрямую. Он вызывает локальный HTTP-сервис процесса мастера на рабочей станции оператора и тем самым получает статус, инициирует разрешённые команды управления и запросы к журналу. Такая схема сохраняет правило единственного владельца шины и согласует локальное и удалённое управление через один программный контур.

Генеративная языковая модель (при использовании в подсистеме бота) ограничена ролью оркестратора вызовов инструментов и формулировки текстового ответа пользователю. Она не является вторичным источником фактической картины режима охранности: достоверные поля статуса и записи журнала восходят к ответам HTTP-инструментов мастера, а не к «додуманному» тексту модели.

Совокупность перечисленных подсистем, направление основных потоков данных и команд, а также границы доступа к последовательному каналу представлены на рисунке 2.2.

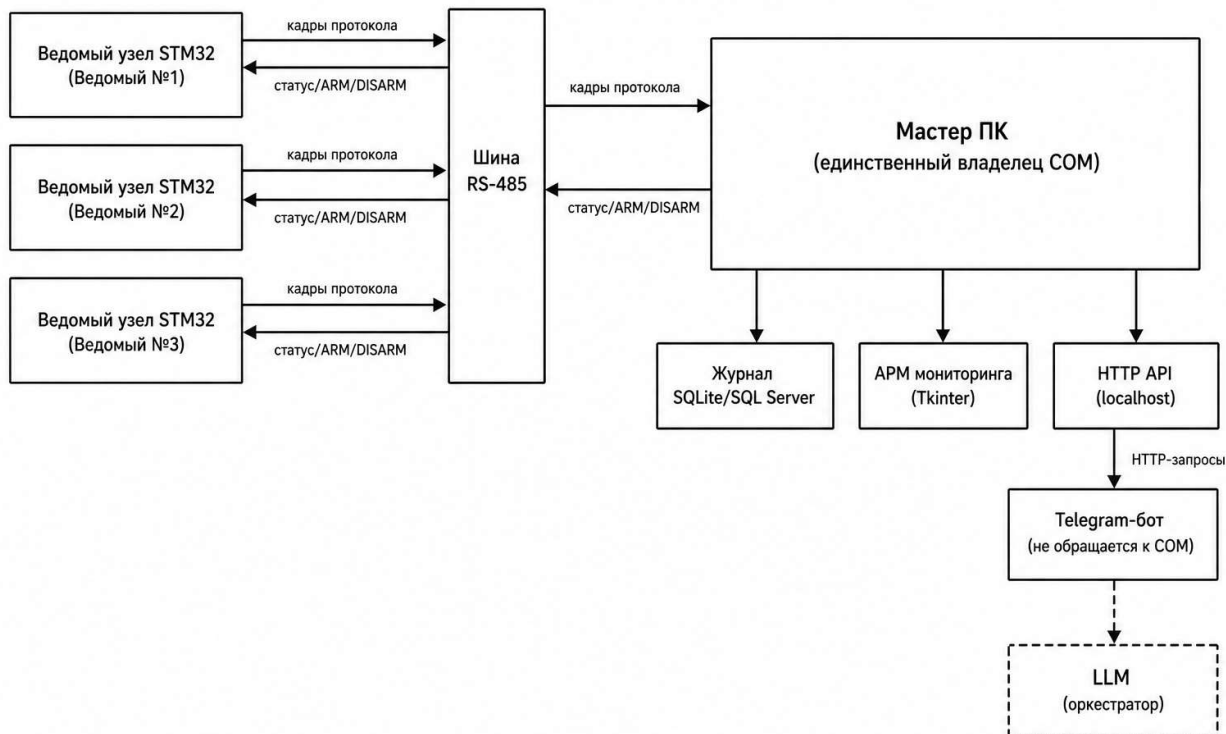


Рисунок 2.2 – Логическая структура комплекса и распределение ролей подсистем

Структурная топология комплекса и привязка подсистем к физическим каналам связи рассматриваются далее.

2.3 Конфигурация аппаратного узла охраны и линии связи

Комплекс описан через технологические уровни и логическое распределение ролей программных подсистем. Настоящий пункт дополняет эти модели структурно-функциональным представлением: как физически и топологически соединены ведомые узлы, линия RS-485 и персональный компьютер оператора, какие каналы информации проходят между объектом охраны и рабочей станцией.

Интеллектуальная микроконтроллерная система управления охранным комплексом (ИМКС) в рамках магистерской работы строится по схеме «ведомые узлы охранной зоны — общая полевая шина — мастер на ПК — каналы оператора». В состав структурной модели входят: один или несколько ведомых

модулей на объекте (микроконтроллерный узел с датчиками, локальным пультом и сопряжением с RS-485); двухпроводная линия RS-485 как общий канал связи; персональный компьютер оператора с USB–UART-адаптером и программным мастером опроса; локальное приложение мониторинга и подсистема удалённого доступа (Telegram-бот через HTTP-службу мастера), исполняемые на том же хосте без второго процесса, владеющего COM-портом.

Полевая часть представлена ведомым узлом охранной зоны. На структурной схеме комплекса узел отображается как совокупность функциональных блоков: контур дискретных датчиков (дверь, окно и аналогичные зоны), канал контроля через аналого-цифровой вход, органы локального управления и индикации, блок UART-сопряжения с внешним приёмопередатчиком RS-485. Ведомый узел не содержит централизованного сервера и не выполняет функций мастера шины: он реагирует на адресованные кадры и поддерживает локальные сценарии взаимодействия с оператором на объекте.

Канал связи реализован как последовательная шина RS-485 с полудуплексным обменом. Топология и режим полудуплекса — см. [8], [9]. Топологически для стенда принята конфигурация: все ведомые подключены к одной паре линий A/B; направление передачи на стороне каждого узла и на стороне ПК определяется сигналами драйвера приёмопередатчика. Структурная модель предполагает единственного инициатора опроса — программный мастер на ПК; это согласуется с логической схемой и исключает конфликт одновременной передачи от нескольких хостов.

Хостовая часть включает персональный компьютер, USB–преобразователь в виртуальный COM-порт и программный комплекс мастера. Структурно на ПК выделяются: модуль циклического опроса и управления линией; подсистема журналирования; приложение мониторинга; локальный HTTP-интерфейс для внешних клиентов. Физически все эти компоненты относятся к одному

вычислительному узлу стенда; удалённый оператор в Telegram взаимодействует с комплексом через сеть мессенджера, но управляющие команды на шину RS-485 по-прежнему проходят только через процесс мастера на ПК.

Для учебной эксплуатации структурная модель допускает расширение числа ведомых адресов на той же линии без изменения топологии «одна шина — один мастер»; изменяются лишь параметры опроса и таблица наблюдения на хосте. Ограничения по длине линии, помехоустойчивости и резервированию канала в рамках данной работы не рассматриваются как предмет структурного проектирования и относятся к разделу испытаний.

Структурно-функциональная схема ИМКС и топология связи элементов учебно-лабораторного стенда представлены на рисунке 2.3.

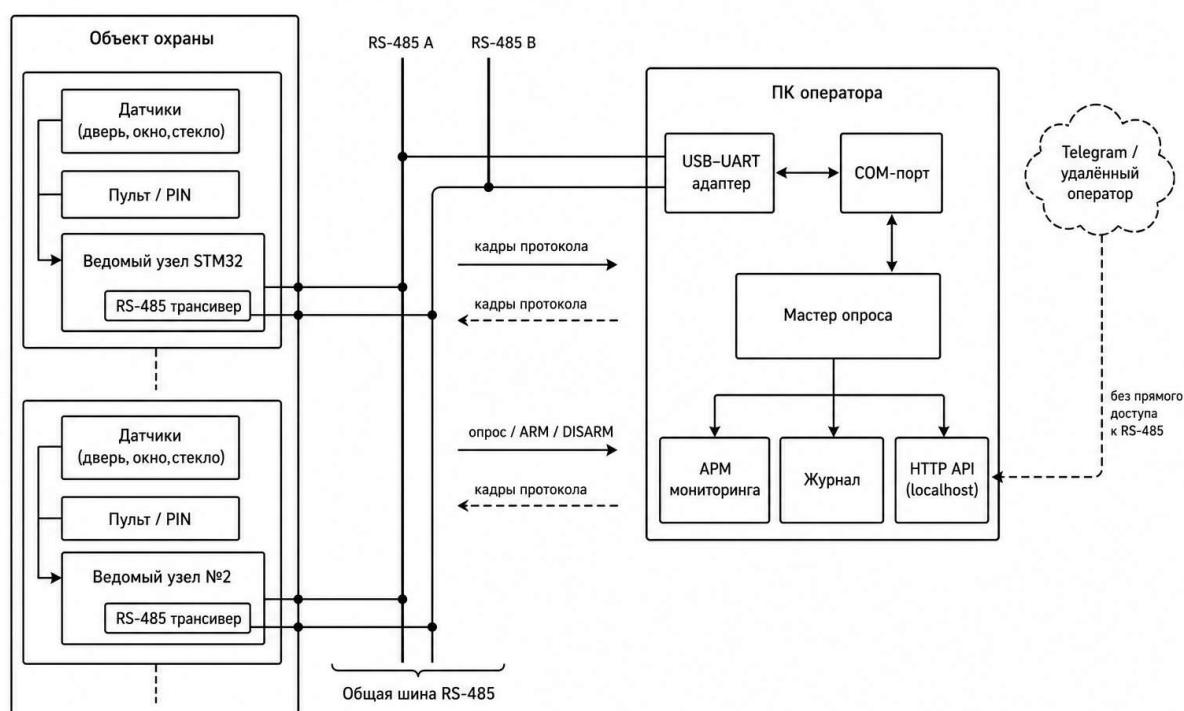


Рисунок 2.3 – Структурно-функциональная схема ИМКС и топология связи учебно-лабораторного стенда

Таким образом, структурная модель комплекса является основой для формального описания протокола обмена и алгоритмов функционирования.

2.4 Формальная модель протокола информационного обмена

На транспортном и прикладном уровнях обмен между мастером и ведомыми узлами задаётся единым кадровым протоколом, одинаково трактуемым на микроконтроллере и на персональном компьютере. Цель формализации — зафиксировать минимальный набор правил, достаточный для опроса состояния и выдачи команд управления режимом охраны, без привязки к конкретным фрагментам исходного кода. Формализация кадра и согласование приёмников – в духе [9].

Кадр включает: байт синхронизации 0xAA, адрес ведомого (1...247), код команды, длину полезной нагрузки, поле данных (для ответа на чтение статуса — один байт состояния) и CRC16, вычисляемую по области «синхробайт + адрес + команда + длина + данные» по правилу, совместимому с Modbus-CRC16. Минимальная длина кадра без полезной нагрузки — 6 байт. Неверный префикс, несовпадение CRC или несогласованная длина трактуются как повреждение транспортного уровня; кадр отбрасывается, прикладная команда не выполняется.

Команды прикладного уровня в рамках прототипа ограничены тремя типами, представленными в таблице 1.

Таблица 1 – Назначение команд протокола обмена

Код	Назначение	Направление	Полезная нагрузка
0x01	Чтение статуса	Мастер- ведомый; ответ - ведомый - мастер	в ответе — байт статуса
0x02	Постановка на охрану (ARM)	мастер → ведомый	пустая
0x03	Снятие с охраны (DISARM)	мастер → ведомый	пустая

Семантика байта статуса (биты режима охраны и зон) задаётся в далее; здесь важно лишь, что единственным содержательным полем ответа на опрос является один сводный байт, согласованный между прошивкой и мастером.

Режим работы линии. Обмен организован по схеме «запрос мастера — ответ ведомого» в полудуплексном режиме RS-485: инициатором всегда выступает мастер на ПК; ведомый обрабатывает только кадры с совпадающим адресом и корректной CRC. Циклический опрос адресов, таймаут ожидания ответа и обновление таблицы состояния на хосте относятся к алгоритму мастера; реактивный разбор кадра и выполнение команд — к алгоритму ведомого.

Обобщённые алгоритмы обработки протокола на стороне ведомого узла и на стороне мастера ПК представлены на рисунке 2.4.

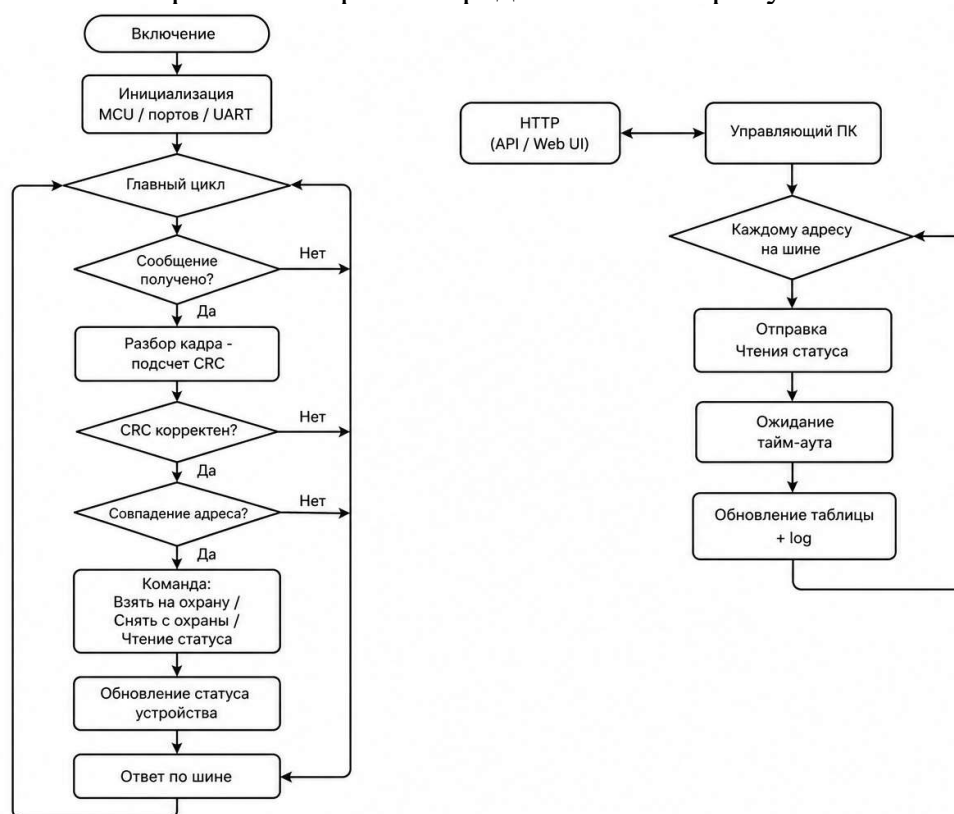


Рисунок 2.4 - Алгоритмы обработки кадров протокола на ведомом узле и на стороне мастера ПК

Левая часть рисунка отражает цикл прошивки: приём кадра, проверка CRC и адреса, выполнение команды (взять на охрану / снять / чтение статуса),

обновление состояния и ответ по шине. Правая часть отражает цикл мастера: последовательный опрос адресов на шине, ожидание ответа в пределах таймаута, обновление таблицы ведомых и журнала; связь с уровнем представления (НТТР-интерфейс, приложение мониторинга) показана как внешний контур относительно опроса линии, без прямого доступа клиентов к СОМ-порту.

Таким образом, протокол описан как согласованная формальная модель кадра, команд и циклов «запрос — ответ». Далее раскрывается содержательный смысл байта статуса охранных зон.

2.5 Семантическая модель байта статуса охранных зон

Ответ ведомого узла на команду чтения статуса содержит один байт полезной нагрузки, который в комплексе используется как сводное машинное представление состояния охранной зоны. Этот байт формируется на микроконтроллере по результатам работы конечного автомата охранности и опроса датчиков, передаётся мастеру в кадре ответа и далее интерпретируется на персональном компьютере в табличном виде и в текстовых расшифровках для оператора. Описание назначения битов байта статуса представлены в таблице 2.

Таблица 2 – Назначение битов байта статуса

Маска (hex)	Бит	Смысл при установленном бите
0x01	0	Узел на охране (режим ARMED)
0x10	4	Действует задержка на выход (переход к постановке после команды ARM или локального сценария)

Байт статуса трактуется как битовая маска: отдельные биты (или их комбинации) отражают режим охраны узла и признаки активности охранных зон. Незаданные в прототипе биты зарезервированы и должны игнорироваться принимающей стороной при разборе, если не оговорено иное в расширении протокола.

Режимные биты описывают логическое состояние автомата охраны ведомого узла:

Режим «снято с охраны» соответствует отсутствию битов 0x01 и 0x10 (при этом биты зон могут отражать физическое состояние датчиков независимо от режима — в зависимости от политики прошивки; в реализации прототипа зоны учитываются при формировании байта всегда, а смысл тревоги по зонам анализируется оператором в контексте режима).

Зональные биты отражают активность соответствующих охранных контуров на момент формирования ответа и представлены в таблице 3:

Таблица 3 – Семантика битов байта статуса ведомого узла

Маска (hex)	Бит	Охранный зона / канал
0x02	1	Дверь (магнитно-контактный контур)
0x04	2	Окно (магнитно-контактный контур)
0x08	3	Стекло (канал по порогу АЦП)

Установленный бит зоны означает нарушение / срабатывание соответствующего контура в логике прошивки (для учебного стенда допускается демонстрационная подстановка зон по таймеру после постановки на охрану — без изменения формата байта). Комбинация нескольких зональных битов допустима и отображает одновременное срабатывание нескольких контуров.

Правила согласованности. Формирование байта выполняется только на ведомом узле; мастер и клиентские интерфейсы не «дополняют» маску произвольными значениями. Мастер сохраняет последний достоверно принятый байт статуса для каждого адреса вместе с признаком «узел online/offline»; при таймауте опроса содержательность статуса не обновляется. Текстовые подписи

(«на охране», «дверь», «стекло» и т.п.) на ПК и в Telegram-канале являются расшифровкой маски, а не самостоятельным источником данных.

Структура байта статуса и соответствие битов охранным зонам и режимам представлены на рисунке 2.5.

	Бит 0	Бит 1	Бит 2	Бит 3	Бит 4	Бит 5-7
Назначение	На охране	Дверь	Окно	Стекло	Задержка выхода	Резерв
Маска	0x01	0x02	0x04	0x08	0x10	-

Рисунок 2.5 - Семантическая модель байта статуса охранных зон

Таким образом, байт статуса определён как компактная семантическая модель состояния узла для протокола. Алгоритм переходов между режимами охраны (снято, задержка, на охране) и связь с битами статуса, циклический опрос и отображение статуса на мастере рассматриваются далее.

2.6 Алгоритм конечного автомата режимов охраны ведомого узла

Байт статуса отражает не только мгновенное состояние датчиков, но и режим охраны ведомого узла. Этот режим задаётся конечным автоматом, работающим на микроконтроллере независимо от цикла опроса мастера: мастер инициирует переходы командами ARM и DISARM, а также периодически считывает результат через READ_STATUS. Настоящий пункт формализует состояния и переходы автомата на уровне алгоритма; программная реализация находится в файле прошивки alarm.c, связь с UART и клавиатурой.

Множество состояний в рамках прототипа включает три основных режима представленных в таблице 4:

Таблица 4 – Основные режимы работы ведомого узла

Состояние	Смысл	Бит статуса
Снято с охраны	Охранная логика не блокирует объект; допускается локальное управление	бит 0x01 и 0x10 сброшены

Продолжение таблицы 4

Состояние	Смысл	Бит статуса
Задержка на выход	После команды постановки выдерживается интервал, позволяющий оператору покинуть зону	установлен бит 0x10
На охране	Объект считается поставленным; зональные биты отражают срабатывание контуров	установлен бит 0x01

Состояния «снято» и «на охране» вместе с «задержкой» взаимоисключающи по режимным битам: одновременно 0x01 и 0x10 в корректном ответе не устанавливаются. Биты зон (0x02, 0x04, 0x08) формируются дополнительно по результатам опроса датчиков и могут присутствовать в байте статуса независимо от режима; их интерпретация как «тревога» на стороне оператора выполняется с учётом того, находится ли узел на охране.

События, инициирующие переходы:

- Команда ARM по линии RS-485 (мастер): переход из «снято» в «задержка на выход» с запуском таймера; по истечении интервала (в учебной конфигурации порядка 30 с) — автоматический переход в «на охране»,
- Команда DISARM по линии: переход в «снято» независимо от предыдущего режима (если команда принята и адрес совпал),

– Локальный PIN на пульте узла: при верном коде — переключение «снято ↔ задержка/снятие» по той же логике, что и удалённая команда (постановка через задержку, снятие — немедленно),

– Тактовое обновление (alarm_tick): единственный автоматический переход «задержка → на охране» по таймеру без участия мастера.

Команда READ_STATUS не меняет состояние автомата; она лишь инициирует формирование текущего байта статуса и ответ по шине.

Связь с датчиками. В состоянии «на охране» периодически (в главном цикле прошивки) опрашиваются контуры двери, окна и канал «стекло»; при нарушении соответствующие зональные биты устанавливаются в байте статуса. На учебном стенде допускается демонстрационная подстановка зон по таймеру после входа в «на охране» — без изменения структуры автомата.

Диаграмма состояний конечного автомата режимов охраны ведомого узла представлена на рисунке 2.6.

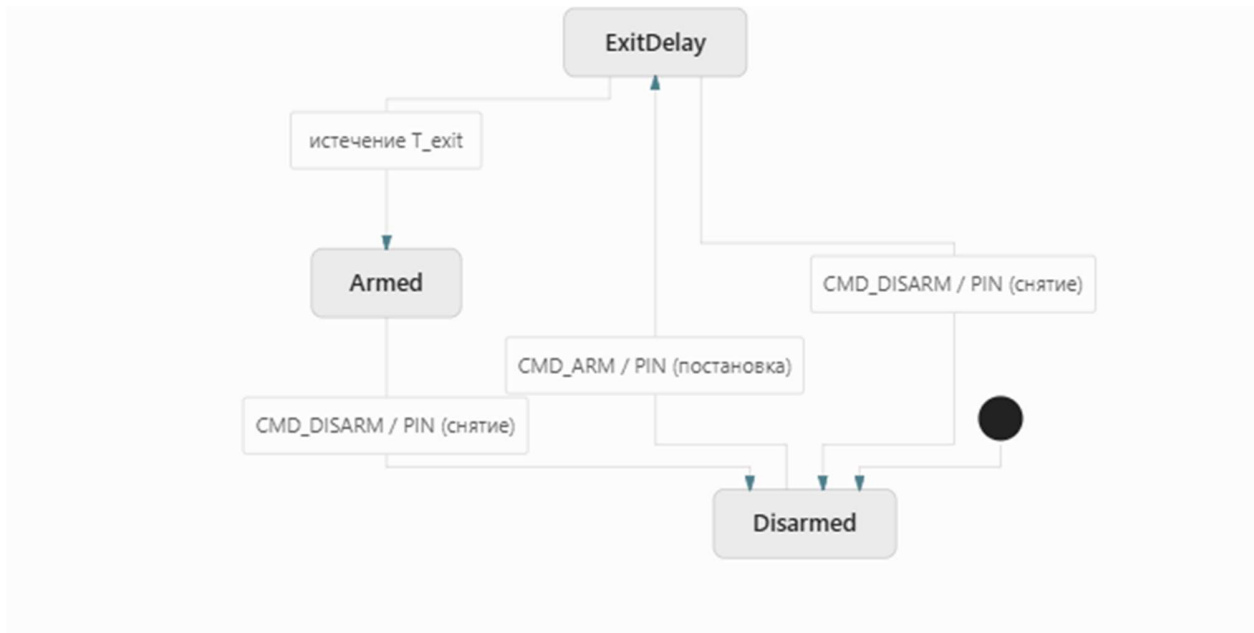


Рисунок 2.6 – Диаграмма состояний конечного автомата режимов охраны ведомого узла

Таким образом, режимы охраны задают алгоритмическую модель охраны ведомого узла, согласованную с семантикой байта статуса и протоколом.

2.7 Алгоритм циклического опроса и управления со стороны мастера

На хостовой стороне комплекса функции опроса линии RS-485 и выдачи команд постановки/снятия сосредоточены в программном мастере. В отличие от ведомого узла, работающего по реактивной схеме «кадр — ответ», мастер реализует активный циклический алгоритм: он последовательно обходит заданный диапазон адресов, инициирует чтение статуса, фиксирует успех или таймаут и обновляет сводную модель наблюдений для приложения мониторинга и HTTP-клиентов. Настоящий пункт описывает алгоритм на уровне проектирования; реализация в модуле `rs485_master.py`.

Предпосылки алгоритма. Мастер является единственным процессом, открывающим виртуальный COM-порт USB-UART; параллельное использование порта другими приложениями не допускается. Обмен с линией выполняется в фоновом рабочем потоке, чтобы интерфейс оператора и HTTP-обработчики не блокировали ожидание ответа от ведомого. Команды ARM и DISARM, поступившие от АРМ или удалённого клиента, не создают второй канал доступа к линии, а помещаются в очередь передачи и выдаются на шину в паузах опроса (очередь обслуживается перед отправкой очередного READ_STATUS и внутри интервала ожидания ответа).

Основной цикл опроса при открытом порте повторяется до остановки мастера:

- обход адресов от `addr_min` до `addr_max` (параметры конфигурации стенда),
- для каждого адреса:
 - 1 формирование кадра READ_STATUS;
 - 2 отправка в последовательный порт;
 - 3 ожидание ответа в пределах `reply_timeout_ms` с накоплением потока байт и выделением кадра (синхробайт, CRC, адрес, команда ответа);

4 при успешном разборе - обновление записи ведомого: «online», байт статуса, текстовая расшифровка, время последнего ответа;

5 при таймауте - перевод адреса в «offline» и запись события в журнал без подмены статуса «успешными» данными.

- после прохода по всем адресам – пауза `poll_interval_ms` до следующего цикла; в паузе также обслуживается очередь команд.

Управляющие команды. Запросы ARM и DISARM формируют кадры протокола и ставятся в очередь передачи. Для оператора результат команды определяется последующими циклами `READ_STATUS`, а не немедленным «успехом» на уровне HTTP.

Представление для клиентов. Таблица ведомых обновляется в рабочем потоке; для GUI и HTTP-API выдаётся снимок (snapshot) - единый источник данных для локального и удалённого доступа.

Журналирование. События опроса (порт, таймауты, ARM/DISARM) передаются в подсистему учёта через функцию обратного вызова мастера.

Параметры (`addr_min`, `addr_max`, `poll_interval_ms`, `reply_timeout_ms`, скорость UART) задаются при настройке стенда и влияют на компромисс «оперативность — нагрузка на линию — устойчивость к задержкам».

Блок-схема алгоритма циклического опроса и обслуживания очереди команд мастера представлена на рисунке 2.7.

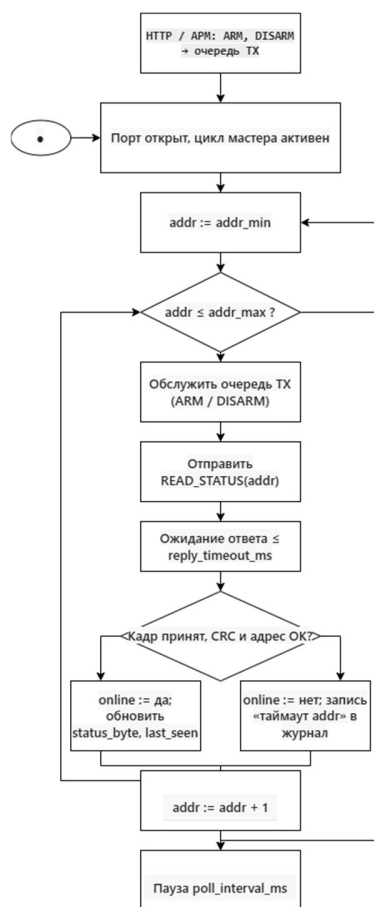


Рисунок 2.7 – Алгоритм циклического опроса ведомых и управления линией RS-485 на стороне мастера ПК

2.8 Обобщённый алгоритм функционирования интеллектуальной микроконтроллерной системы управления

Ранее были формализованы протокол обмена, семантика байта статуса, конечный автомат охраны ведомого узла и алгоритм циклического опроса мастера. Настоящий пункт объединяет эти модели в единое описание работы ИМКС как распределённой системы «ведомые узлы — шина RS-485 — мастер на ПК — каналы оператора», фиксируется порядок взаимодействия подсистем во времени и при типовых сценариях эксплуатации.

Комплекс строится по схеме «ведущий на ПК — шина связи — ведомые узлы охранной зоны». Единственный активный участник последовательного канала со стороны компьютера периодически опрашивает адреса в заданном

диапазоне и принимает ответы. Ведомый микроконтроллер реактивно отвечает на адресованные кадры и по командам изменяет режим охраны. Локальный оператор на АРМ наблюдает таблицу узлов и журнал; удалённый пользователь через Telegram запрашивает состояние и отдаёт команды через HTTP-контур мастера, не открывая COM-порт другим процессом.

На ведомом узле: инициализация микроконтроллера и периферии (GPIO датчиков, UART/RS-485, при необходимости АЦП, клавиатура, индикация), установка начального состояния автомата охраны «снято с охраны», включение приёма кадров по линии.

На ПК: запуск приложения мастера, инициализация таблицы ведомых и параметров опроса, при необходимости — подсистемы журналирования; открытие COM-порта и старт фоновых потока циклического опроса.

Параллельно выполняются два контура:

- Мастер — цикл опроса адресов, READ_STATUS, обновление online/offline и байта статуса, обслуживание очереди ARM/DISARM,
- Ведомый — главный цикл: опрос датчиков, alarm_tick (переход «задержка → на охране»), приём и разбор кадров, ответ READ_STATUS, выполнение ARM/DISARM.

Согласованность состояния достигается тем, что актуальный статус-байт доставляется мастеру только в успешном ответе на опрос.

Оператор на месте может ввести PIN на клавиатуре узла и переключить режим охраны (постановка через задержку или снятие) без участия ПК. Изменение режима отражается в байте статуса и становится видимым мастеру при следующем успешном READ_STATUS. Конфликт с удалённой командой разрешается очередью мастера и правилом: источником аппаратной истины на узле остаётся прошивка ведомого, мастер синхронизирует модель опросом.

Оператор на рабочей станции через АРМ просматривает таблицу ведомых, журнал и инициирует ARM/DISARM для выбранного адреса. Команды

передаются тому же объекту мастера, что ведёт опрос; СОМ-порт повторно не открывается. Отображаемые данные — снимок состояния мастера на момент последнего цикла опроса.

Пользователь отправляет запрос или команду в мессенджере; бот обращается к локальному HTTP-API мастера. Статус и команды управления эквивалентны сценариям АРМ при работоспособном мастере. Языковая модель, если используется, маршрутизирует вызовы инструментов к HTTP и не подменяет байт статуса текстом.

- Возможны ошибки и граничные случаи при работе приложений: Повреждённый кадр / неверная CRC — отбой на ведомом или мастере, переход к следующему адресу/ожиданию нового кадра.

- Таймаут ответа — адрес помечается offline, событие в журнале; последний успешный статус сохраняется как устаревший снимок.

- Мастер остановлен или порт закрыт — АРМ и бот получают отказ связи с понятным сообщением.

- Журнал недоступен — бот сообщает о невозможности выдать выборку за период.

Закрытие СОМ-порта на ПК останавливает цикл опроса. Ведомые остаются в последнем логическом состоянии до отключения питания или локального сброса/снятия с охраны.

Обобщённая блок-схема алгоритма функционирования ИМКС представлена на рисунке 2.8.

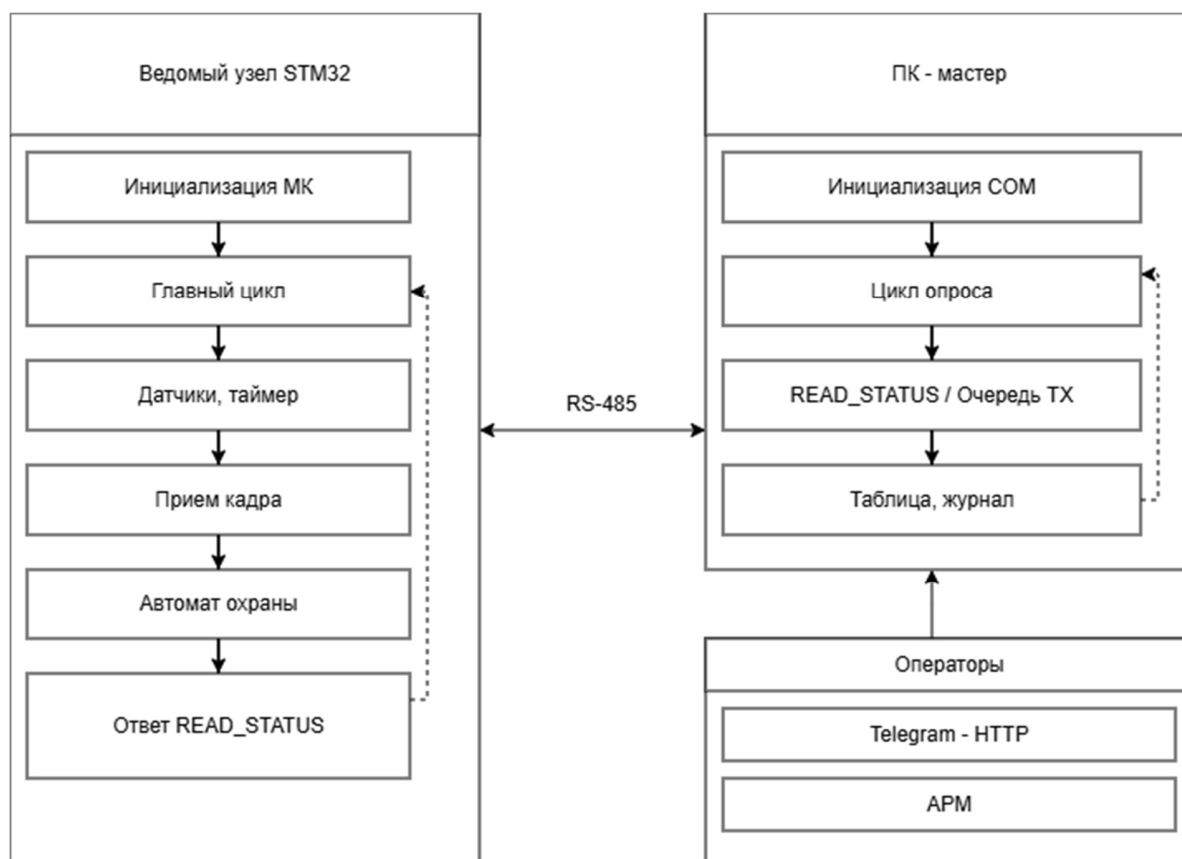


Рисунок 2.8 – Обобщённая блок-схема алгоритма функционирования ИМКС

На рисунке 2.8 показаны параллельные контуры ведомого узла и мастера ПК, связанные шиной RS-485, и каналы оператора (APM, Telegram через HTTP).

2.9 Выводы

В текущем разделе построена архитектурная и алгоритмическая модель интеллектуальной микроконтроллерной системы управления охранным комплексом. Описаны уровни комплекса (полевой узел, канал RS-485, мастер на ПК, APM и удалённый доступ), распределение ролей и правил согласованности состояния между ведомым узлом, линией связи и клиентами управления. Зафиксирована структурная модель ИМКС и формализован прикладной протокол обмена с единым правилом CRC16, семантикой байта статуса, конечным автоматом охраны ведомого узла и алгоритмом циклического опроса мастера. Обобщённый алгоритм функционирования связывает эти модели в

единый сценарий работы от инициализации до остановки с участием локального оператора, АРМ и Telegram-контура.

Полученные решения задают однозначные спецификации для программной реализации: состав кадров и команд, интерпретацию битов статуса, переходы режимов «снято с охраны — задержка на выход — на охране», политику online/offline и очередь команд ARM/DISARM на стороне мастера. Архитектурно подтверждено единственное владение COM-портом процессом мастера и доступ удалённых клиентов только через его HTTP-API, что исключает параллельные потоки на шине RS-485 и задаёт границы доверия при интеграции языковой модели.

3 Программная реализация

3.1 Программная реализация ведомого узла (STM32F103)

3.1.1 Модульная структура прошивки

Реализация ведомого узла на STM32F103 выполнена в виде набора прикладных модулей с разделением транспортного уровня, логики охраны и периферии. Исходные файлы прошивки размещены в каталоге /Core/ (подкаталоги Src/ и Inc/). Список основных файлов прошивки содержится в таблице 5; Модульная структура прошивки реализует декомпозицию по [16]; периферия — [13], [14].

Таблица 5 – Основные файлы проекта прошивки микроконтроллера

Модуль	Назначение
board.h, board_gpio.c	Адрес RS-485, выводы GPIO, пороги АЦП, PIN, задержка на выход
proto.c / proto.h	Кадр и CRC16 — совместимо с pc_master/protocol.py
uart485.c	USART, кольцевой буфер, прерывание приёма
alarm.c	Автомат охраны, датчики, ответы мастеру
keypad.c, lcd1602.c, adc_glass.c	Локальный оператор и каналы зон
main.c	Инициализация и главный цикл

3.1.2 Протокольный уровень

Константы команд и масок статуса совпадают. Функции proto_build_frame и proto_try_parse реализуют симметричный разбор потока байт (возврат 0 – ждём данных, -1 – сдвиг на 1 байт). Симметричный разбор кадра согласован с транспортной моделью п. 2.4 и [9].

Фрагмент 3.1 – Определения протокола (proto.h)

```

#define PROTO_SYNC 0xAAU          /* синхробайт кадра */
#define CMD_READ_STATUS 0x01U     /* запрос status_byte */
#define CMD_ARM          0x02U     /* постановка (задержка на выход)
*/
#define CMD_DISARM       0x03U     /* снятие с охраны */
#define STATUS_ARMED     0x01U     /* «на охране» */
#define STATUS_DOOR      0x02U     /* зона «дверь» */
#define STATUS_WINDOW    0x04U     /* зона «окно» */
#define STATUS_GLASS      0x08U     /* зона «стекло» */
#define STATUS_EXIT_DELAY 0x10U     /* задержка на выход */

```

3.1.3 Приём кадров по RS-485

В файле `uart485.c` приём по USART2 соответствует описанию периферии в [13] и прерывание `USART2_IRQHandler`; байты складываются в кольцевой буфер, в главном цикле `uart485_poll()` выделяется кадр и при совпадении адреса вызывается `alarm_on_rs485_command`.

Фрагмент 3.2 – Опрос линии (`uart485.c`)

```

void uart485_poll(void)
{
    /* перенос байт из кольца в поток разбора */
    while (rx_pop(&b))
        s_rx_stream[s_rx_stream_len++] = b;
    while (s_rx_stream_len >= 6U) {
        int n = proto_try_parse(s_rx_stream, s_rx_stream_len, ...);
        if (n > 0) {
            if (addr == MODULE_RS485_ADDR)
                alarm_on_rs485_command(cmd, payload, plen);
            /* удалить обработанный кадр */
            continue;
        }
        if (n == -1) { /* resync: сдвиг на 1 байт */ continue; }
        break;
    }
}

```

```
}
```

3.1.4 Логика охраны

Модуль `alarm.c` реализует автомат п. 2.6. Функция `alarm_status_byte()` формирует байт для `READ_STATUS`; `alarm_tick()` завершает задержку на выход; `alarm_on_keypad_char()` обрабатывает локальный PIN.

Фрагмент 3.3 – Команды и статус (`alarm.c`)

```
void alarm_on_rs485_command(uint8_t cmd, ...)
{
    if (cmd == CMD_READ_STATUS) {
        uint8_t st = alarm_status_byte();
        proto_build_frame(MODULE_RS485_ADDR, CMD_READ_STATUS, &st,
1U, buf, ...);
        uart485_tx(buf, n);
    } else if (cmd == CMD_ARM) {
        s_state = ALARM_EXIT_DELAY;
        s_exit_deadline_ms = HAL_GetTick() + EXIT_DELAY_MS;
    } else if (cmd == CMD_DISARM) {
        s_state = ALARM_DISARMED;
    }
}
```

3.1.5 Главный цикл

Фрагмент 3.4 – `main.c`

```
while (1) {
    uart485_poll();           /* приоритет – приём кадров
мастера */
    alarm_tick();             /* таймер задержки на выход */
    char c = keypad_poll();
    if (c != 0) alarm_on_keypad_char(c);
    /* ЖКИ обновляется раз в 200 мс */
    lcd1602_update(alarm_get_state_raw(), alarm_status_byte());
}
```

3.2 Программная реализация мастера опроса на ПК

3.2.1 Назначение и состав

Приложение `pc_master` - единственный владелец COM-порта. Мастер реализован на Python с `pyserial` по [19]; архитектура модулей — [18] с использованием `pyserial`, `SQLite` (журнал) и встроенного HTTP-сервера для внешних клиентов. Основные модули представлены в таблице 6:

Таблица 6 – Основные файлы проекта мастера опроса на ПК

Модуль	Файл	Функция
Протокол	<code>protocol.py</code>	<code>build_frame</code> , <code>try_parse</code> , <code>CRC16</code>
Мастер	<code>rs485_master.py</code>	Класс <code>Rs485Master</code> , фоновый опрос, очередь TX
HTTP API	<code>control_http.py</code>	<code>/status</code> , <code>/arm</code> , <code>/disarm</code> , <code>/journal</code>
Хранение	<code>storage.py</code>	Журнал событий в <code>SQLite</code>
Интерфейс	<code>main.py</code>	Tkinter-АРМ оператора

3.2.2 Класс `Rs485Master` и цикл опроса

Объект `Rs485Master` хранит таблицу ведомых устройств (`SlaveRow`: адрес, `online`, `status_byte`, расшифровка). При `open_port()` запускается фоновый поток `worker_loop`: обход адресов от `addr_min` до `addr_max` включительно, `READ_STATUS`, таймаут → `offline`. Циклический опрос и таймауты соответствуют алгоритму п. 2.7 и [8].

Фрагмент 3.5 – Цикл опроса (`rs485_master.py`)

```
def _worker_loop(self) -> None:
    while self._running and self._ser and self._ser.is_open:
        self._drain_tx_queue()          # сначала команды
        ARM/DISARM из очереди
```

```

self.poll_busy = True
for addr in range(self.addr_min, self.addr_max + 1):
    self._drain_tx_queue()
    frame = protocol.build_frame(addr,
protocol.CMD_READ_STATUS, b"")
    self._ser.write(frame)
    rx_stream = bytearray()
    deadline = time.monotonic() + self.reply_timeout_ms /
1000.0

    got = False
    while time.monotonic() < deadline:
        chunk = self._ser.read(256)
        if chunk:
            rx_stream.extend(chunk)
            if self._process_rx_stream(rx_stream, addr):
                got = True
                break
        if not got:
            self._mark_offline(addr)    # таймаут – узел offline
self.poll_busy = False
time.sleep(self.poll_interval_ms / 1000.0)

```

Команды постановки/снятия не блокируют интерфейс: `send_arm` / `send_disarm` кладут кадр в `tx_queue`, поток опроса вызывает `_drain_tx_queue()` между адресами.

3.2.3 HTTP API для Telegram и других клиентов

Модуль `control_http.py` поднимает сервер на 127.0.0.1 (порт `_CONTROL_API_PORT`, по умолчанию 18765). Эндпоинты отдают снимок таблицы мастера и принимают команды без прямого доступа к COM:

Фрагмент 3.6 – Обработка запроса статуса (`control_http.py`)

```

if path == "/status":

```

```

rows = master.get_slaves_snapshot()
data = [{
    "address": r.address,
    "online": r.online,
    "status_byte": r.status_byte,
    "status_text": r.status_text,
    "last_seen": r.last_seen,
} for r in rows]
_json_response(self, 200, {"ok": True, "slaves": data})

```

Так реализовано архитектурное правило: второй процесс не открывает COM, а обращается к мастеру по HTTP.

3.2.4 Журналирование

При событиях на линии (таймаут, ARM, DISARM, изменение статуса) мастер вызывает `_log()`; при подключённом `journal_hook` строки сохраняются через [storage.py](#) в SQLite — единый источник для APM и бота.

3.3 Telegram-бот и взаимодействие с языковой моделью

3.3.1 Разделение модулей

Подсистема удалённого доступа на базе Telegram реализована отдельными файлами каталога `chatbot` с разграничением ролей «бот — HTTP-клиент — агент — LLM», что приведено в таблице 8.

Таблица 8 – Программные модули подсистемы удалённого доступа

Модуль	Файл	Роль
Точка входа	<code>telegram_bot.py</code>	Регистрация команд, фильтр пользователей
HTTP-клиент	<code>security_api_client.py</code>	Запросы к API мастера
Агент	<code>telegram_agent.py</code>	Инструменты + Ollama / эвристический fallback

Модуль	Файл	Роль
LLM	llm_qwen.py	Вызов Qwen 2.5-3B через Ollama

Бот не использует pyserial: все факты о линии приходят из HTTP-ответов мастера.

3.3.2 Клиент API мастера

Фрагмент 3.7 – Запрос статуса (security_api_client.py)

```
def get_status_text() -> str:
    d = get_status_raw()    # GET http://127.0.0.1:18765/status
    for s in d.get("slaves", []):
        lines.append(
            f"узел {s['address']}: "
            f"{'онлайн' if s['online'] else 'офлайн'},
{s['status_text']}"
        )
    return "\n".join(lines)
```

Команды ARM/DISARM — POST /arm, POST /disarm с телом {"addr": N}.

3.3.3 Инструменты и оркестрация LLM

В telegram_agent.py описан набор tools для Ollama: get_alarm_status, arm_slave, disarm_slave, get_journal. Роль LLM как оркестратора инструментов согласована с [20]; модель не подменяет status_byte, она выбирает инструмент, Python выполняет HTTP-вызов и возвращает строку фактов; модель формулирует ответ пользователю.

Фрагмент 3.8 – Описание инструмента (telegram_agent.py)

```
OLLAMA_TOOLS = [{
    "type": "function",
    "function": {
        "name": "get_alarm_status",
        "description": "Узнать состояние ведомых: связь, охрана,
зоны",
```

```
        "parameters": {"type": "object", "properties": {}},  
    },  
    }, ...]
```

При недоступности Ollama агент переходит на эвристический разбор русскоязычных фраз и прямой вызов тех же функций API.

3.3.4 Безопасность

Удалённый доступ повышает удобство эксплуатации, но расширяет поверхность атаки, поэтому в прототипе заложены ограничения на круг пользователей бота и на доступ к управляющему HTTP-интерфейсу мастера. Основные меры реализуются через переменные окружения и привязку API к локальному интерфейсу:

- TELEGRAM_ALLOWED_USER_IDS - белый список пользователей,
- CONTROL_API_TOKEN - опциональная авторизация HTTP,
- API слушает только localhost.

3.4 Выводы

В этом разделе описана программная реализация трёх уровней ИМКС. Прошивка STM32 обеспечивает автономную охранную логику и ответы по RS-485; мастер на ПК инкапсулирует владение COM-портом, циклический опрос и журнал; Telegram-бот и языковая модель Qwen 2.5-3B добавляют удалённый текстовый интерфейс без нарушения архитектурной границы «один владелец шины». Сквозная совместимость протокола (proto.c ↔ protocol.py) и HTTP-контур (control_http ↔ security_api_client) подтверждает согласованность решений из раздела 2 с кодом. Результаты работы прототипа, иллюстрации и сравнительный анализ приведены в разделе 4.

4 Результаты работы

Раздел 4 фиксирует результаты разработки и опытной эксплуатации прототипа ИМКС: состав стенда, подтверждение сценариев, иллюстративные материалы и оценка практической целесообразности принятых технических и программных решений в сопоставлении с типовыми коммерческими системами охранной сигнализации.

4.1 Стенд

Испытания прототипа выполнялись на учебно-лабораторном стенде, в состав которого входят ведомый узел на базе платы STM32F103 с дисплеем 1602 и клавиатурой, имитаторы охранных зон, сегмент линии RS-485 с преобразователем USB–RS-485, персональный компьютер с приложением мастера опроса rs_master и локально развёрнутой средой Ollama для работы Telegram-бота с языковой моделью. Общий вид стенда и соединений показан на рисунке 4.1.

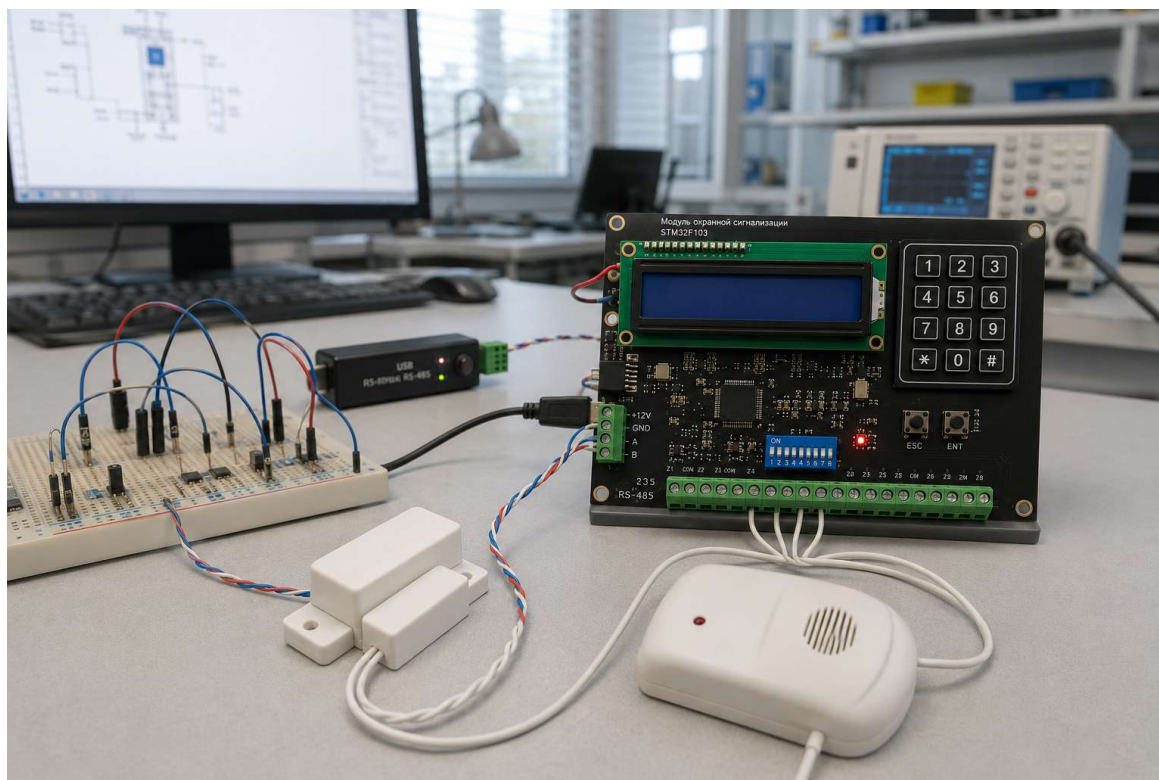


Рисунок 4.1 – Учебно-лабораторный стенд прототипа ИМКС

4.2 Сценарии испытаний

Испытания прототипа ИМКС выполнялись по заранее заданным сценариям, охватывающим обмен по RS-485, управление режимом охраны, отображение состояния на АРМ, журналирование и удалённый доступ через Telegram-бот. Ниже приведён перечень проверяемых ситуаций:

- Циклический опрос адресов, online при подключённом ведомом,
- Таймаут и offline при отключении узла/обрыве,
- ARM/DISARM с APM и через Telegram — изменение битов статуса,
- Локальный PIN на клавиатуре, синхронизация при READ_STATUS,
- Журнал SQLite — выборка через бот,
- Ollama недоступна — работа эвристическим методом.

4.3 Результаты

Испытания проводились на стенде, где фиксировалось поведение системы, формировалась таблица ведомых узлов и журнал событий. По сценариям подтверждена работоспособность основных функций прототипа. Ниже приведены иллюстрации интерфейсов. Состояние ведомых узлов, журнал событий и индикация тревоги по зоне «окно» на приложении мастера опроса приведены на рисунке 4.3.

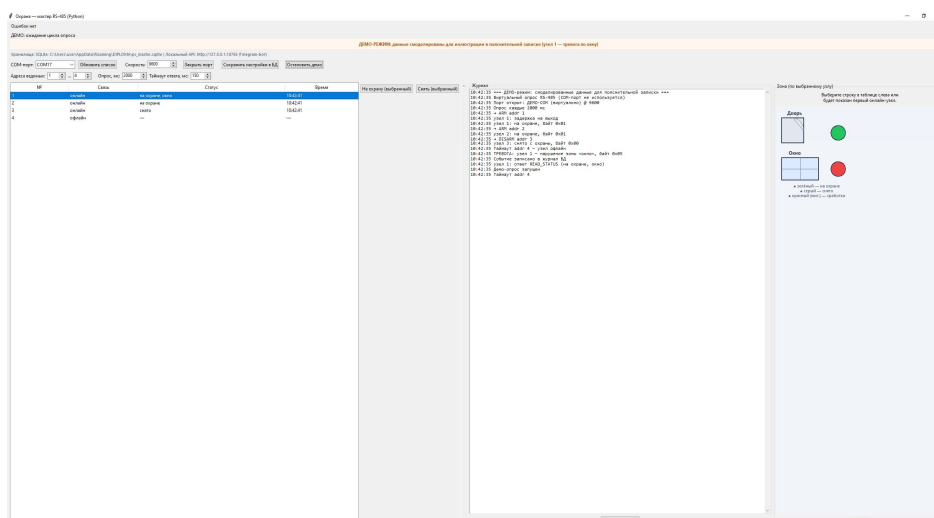


Рисунок 4.3 – Окно приложения мастера: таблица ведомых и журнал событий

Удалённый запрос статуса и отображение ответа на русском языке в диалоге с Telegram-ботом показаны на рисунке 4.4.

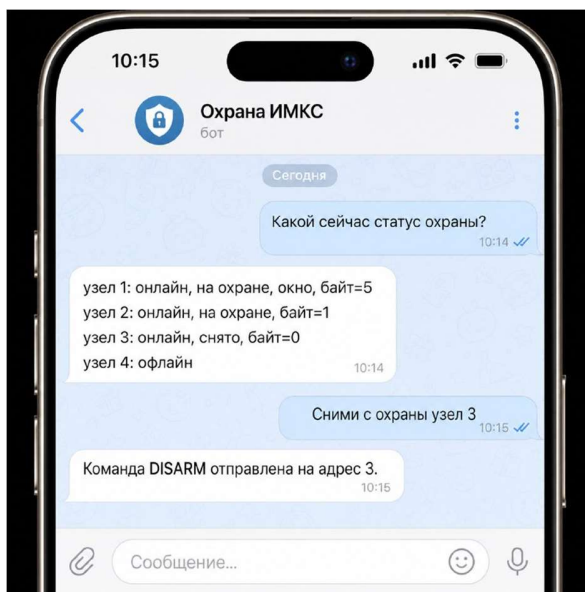


Рисунок 4.4 – Диалог Telegram-бота: запрос статуса и команда снятия с охраны

Локальная индикация режимов охраны на дисплее 1602 ведомого узла (снято с охраны представлена на рисунке 4.5.

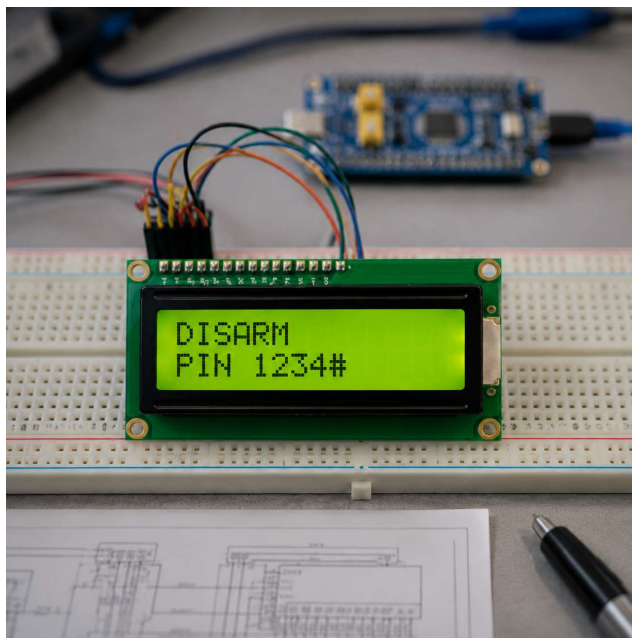


Рисунок 4.5 – ЖКИ ведомого узла в режиме «снято с охраны»

Результаты функциональных испытаний сведены в таблицу 9.

Таблица 9 – Результаты функциональных испытаний

Сценарий испытания	Результат
Запуск мастера опроса, открытие СОМ-порта, опрос адресов 1...4	В таблице ведомых отображаются строки адресов; при ответе ведомого — статус «онлайн», расшифровка режима и зон; в журнале — строки опроса
Отключение ведомого или обрыв линии	По истечении таймаута адрес переводится в «офлайн»; в журнале фиксируется «Таймаут addr N»
Команда ARM с АРМ мастера для выбранного адреса	В линию уходит кадр ARM; на ведомом — задержка на выход, затем «на охране»; в ответе READ_STATUS установлены биты ARMED / EXIT_DELAY согласно автомату
Команда DISARM с АРМ мастера	Ведомый переходит в «снято с охраны»; бит ARMED сброшен в байте статуса
Локальная постановка/снятие по PIN на клавиатуре узла	Без участия ПК режим меняется на ведомом; при следующем успешном READ_STATUS мастер получает обновлённый байт статуса
Имитация нарушения зоны «окно»	В байте статуса устанавливается бит WINDOW; в таблице мастера — «окно» в тексте статуса; на схеме зон — индикация тревоги
Имитация нарушений «дверь», «стекло»	Устанавливаются биты DOOR / GLASS; отображение в таблице и журнале согласовано с proto.h

Продолжение таблицы 9

Сценарий испытания	Результат
Запрос статуса через HTTP API мастера	JSON/текст со списком узлов: online, status_byte, расшифровка; без открытия COM вторым процессом
Команды ARM/DISARM через Telegram-бот	Запрос уходит в API мастера; в линии — кадр управления; подтверждение в ответе бота; изменение статуса при опросе
Запрос журнала через бот за N минут	Выборка событий из БД мастера (SQLite); ответ на русском языке без обращения к COM из бота
Работа бота при недоступном мастере или закрытом порте	Сообщение об ошибке связи с мастером; команды на шину не выполняются
Согласованность CRC16 и разбора кадра (повреждённый/обрезанный ответ)	Неверный кадр не принимается; мастер выполняет resync; адрес не переходит в ложный «онлайн» с некорректным статусом
Обработка произвольного запроса на русском (Ollama + инструменты или эвристика)	Бот вызывает get_alarm_status / arm_slave / disarm_slave; ответ пользователю на русском; факты статуса совпадают с API мастера

Зависимость средней задержки появления обновлённого статуса в таблице мастера от интервала циклического опроса poll_interval_ms при фиксированном таймауте ответа и диапазоне адресов 1–4 приведена на рисунке 4.6.

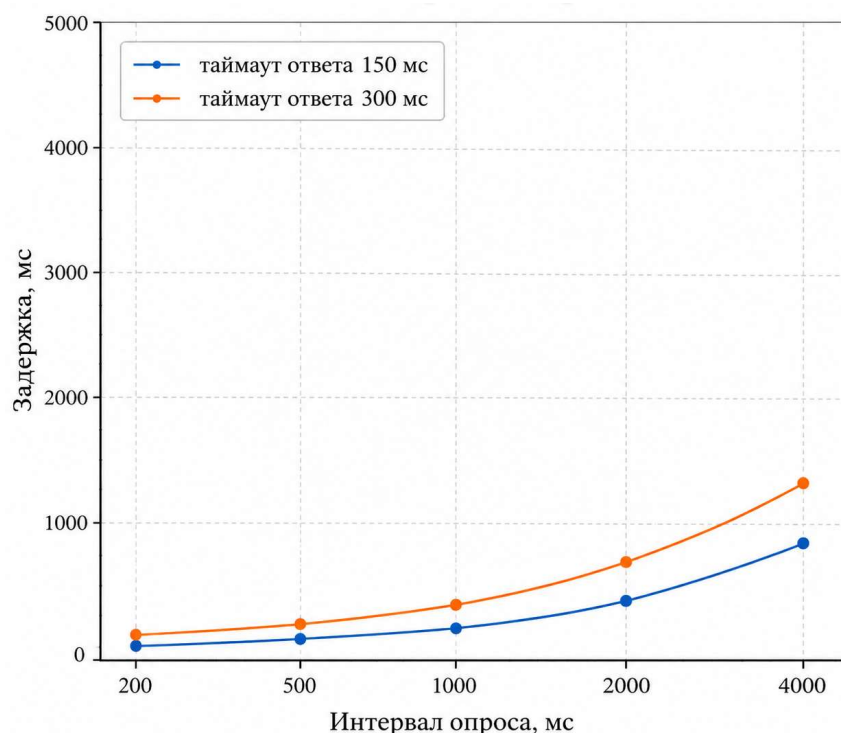


Рисунок 4.6 – Зависимость задержки обновления статуса устройства

4.4 Практическая польза от средств разработки

При разработке ведомого узла цепочка STM32CubeMX + STM32CubeIDE + GCC + ST-Link сократила время настройки тактирования и периферии и обеспечила воспроизводимую отладку по SWD. Модульная прошивка (разделение proto, uart485, alarm) упростила поэтапную отладку: сначала линия и CRC, затем автомат охраны.

На стороне ПК Python позволил быстро реализовать мастер с фоновым потоком, очередью команд и HTTP без привязки к конкретному GUI; SQLite дал единый журнал для АРМ и бота. Ollama с моделью Qwen 2.5-3B обеспечила локальную обработку естественного языка без облачного API и с механизмом function calling: модель маршрутизирует запросы к HTTP-инструментам, а достоверность состояния охраны определяется ответами мастера, а не

«додумыванием» LLM. Это соответствует требованию раздела 2 о разграничении ролей NLP и протокольной истины.

4.5 Сравнение с решениями на рынке

Для оценки разработанного прототипа среди распространённых охранных систем Сопоставление с типовыми комплексами выполнено по [2]; нормативные требования к проектированию – [6], по ключевым признакам (назначение, протокол, удалённый доступ, масштаб и сопровождение) сведено в таблицу 10.

Таблица 10 – Сравнительные характеристики

Критерий	Типовая коммерческая ОПС	Разработанный прототип ИМКС
Протокол шины	Закрытый, проприетарный	Открытый кадр с CRC16, документирован в записке
Мастер	Облако / фирменная станция	Локальный ПК, один процесс – владелец COM
Удалённый доступ	Фирменное приложение	Telegram + локальная LLM + HTTP API
Масштаб	Десятки–сотни адресов	Малый диапазон адресов (учебный)
Стоимость владения	Лицензии, абонент	STM32 + ПК + open-source стек
Модификация	Ограничена производителем	Полный доступ к прошивке и мастеру

Прототип не претендует на замену сертифицированных систем, но демонстрирует интеграцию распределённой шины RS-485, централизованного

мастера и интеллектуального удалённого интерфейса на базе локальной LLM — с явным разделением «машинной правды» (`status_byte`, журнал) и текстового слоя для пользователя. Преимуществом является: прозрачность протокола, воспроизводимость и возможность наращивания (новые зоны, инструменты бота) без смены вендора.

4.6 Ограничения прототипа и направления развития

Разработанный программно-аппаратный комплекс относится к учебно-исследовательскому прототипу и предназначен для демонстрации архитектуры, протокола и интеграции удалённого доступа, а не для прямой замены сертифицированных охранных систем на объекте. Ниже перечислены основные ограничения, выявленные при проектировании, реализации и испытаниях на стенде:

- один мастер на сегмент RS-485,
- нет промышленной гальванической развязки и резервирования,
- LLM — вспомогательный слой, не SIL-уровень,
- демо-режим `DEMO_MODE` без полного комплекта датчиков.

4.7 Выводы

На учебно-лабораторном стенде подтверждены сценарии опроса, управления охраной и удалённого доступа через Telegram. Ниша прототипа — прозрачный протокол и локальная LLM [20] на фоне закрытых систем [2]. Иллюстративные материалы подтверждают работоспособность архитектуры и реализации. Сравнение с рыночными системами показывает нишу прототипа: прозрачная инженерная база и интеграция локальной языковой модели; коммерческие системы сильнее в масштабе, сертификации и готовой эксплуатации на объекте.

ЗАКЛЮЧЕНИЕ

Выпускная работа была направлена на разработку и интеграцию программно-аппаратного комплекса охранной сигнализации, в составе которого микроконтроллерный ведомый узел на базе STM32F103 выполняющий функции сбора данных датчиков и локальной индикации/управления, а обмен организован по линии RS-485 по согласованному кадровому протоколу с контролем ошибок передачи. На стороне персонального компьютера функции единственного владельца последовательного интерфейса и циклического опроса реализованы программным мастером, обеспечивающим сбор актуальных состояний ведомых, выдачу команд постановки на охрану и снятия с охраны, локальную визуализацию режимов и ведение единого журнала событий в связке с подсистемой хранения данных.

Дополнительно спроектирован и реализован канал удалённого взаимодействия на базе Telegram-бота, который использует локальный управляющий HTTP-слой мастера и тем самым сохраняет целостность протокола и единство источников информации. Для упрощения диалогового формата обращений применена локальная компактная языковая модель как вспомогательный элемент маршрутизации пользовательских запросов к ограниченному набору разрешённых операций запроса статуса и управления без подмены ими результатов протокола и журнальных записей.

Практический результат выпускной работы заключается в прототипе, на котором подтверждена связность ключевых сценариев: опрос ведомых, выполнение команд ARM/DISARM, сопоставление статус-информации с ожиданиями различных режимов, согласованность отображения и журналирования на ПК и в удалённом канале при корректной конфигурации среды. При этом решение не претендует к полному перечню нормативных требований к промышленным охранным изделиям и требует дальнейшего

проектирования надёжности, информационной безопасности и развития модели для реального применения на объектах.

Таким образом поставленная цель создания интеллектуальной микроконтроллерной системы охранной сигнализации считается достигнутой в объёме, определённом задачами проекта и ограничениям по результатам функциональных испытаний. Перспективы развития связаны с ужесточением требований к журналированию и аутентификации, улучшением помехоустойчивости линии, возможным резервированием и альтернативными транспортом удалённой связи в зависимости от профиля заказчика.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Методические указания кафедры «Вычислительная техника» Пензенского государственного университета по выполнению и оформлению магистерской диссертации (пояснительной записки): учеб.-метод. материалы / ПГУ. – Пенза, 2024–2026. – URL: elib.pnzgu.ru/resource_type/78 электронный архив кафедры (дата обращения: 12.02.2026).
- 2 Синилов, В. Г. Системы охранной, пожарной и охранно-пожарной сигнализации: учебник / В. Г. Синилов. – 6-е изд., стер. – М.: Академия, 2011. – 510 с.
- 3 Назаров, В. И. Охранные и пожарные системы сигнализаций / В. И. Назаров, В. И. Рыженко. – М.: Оникс, 2007. – 32 с.
- 4 Чибунин, В. М. Технические средства охранно-пожарной сигнализации, используемые подразделениями органов внутренних дел: учеб.-нагляд. пособие / В. М. Чибунин, В. Н. Шапочанский. – Иркутск: ВСИ МВД России, 2016. – 90 с.
- 5 ГОСТ Р 59639–2021. Системы охраны и охранно-пожарной сигнализации. Классификация. – М.: Стандартинформ, 2021.
- 6 ГОСТ Р 53325–2012. Системы охраны и охранно-пожарной сигнализации. Порядок проектирования. – М.: Стандартинформ, 2013.
- 7 Сигорский, Ю. Я. Основы микропроцессорной техники: учеб. для вузов / Ю. Я. Сигорский, В. А. Шевцов, Н. П. Маштакова; под ред. Ю. Я. Сигорского. – 11-е изд. – М.: Горячая линия – Телеком, 2019. – 400 с.
- 8 Куликов, В. П. Промышленные сети и интерфейсы: учеб. пособие / В. П. Куликов. – М.: Горячая линия – Телеком, 2013. – 272 с.
- 9 Рыжиков, В. И. Интерфейсы в электронных устройствах: учеб. пособие / В. И. Рыжиков. – СПб.: Корона-Век, 2012. – 304 с.

- 10 Юров, В. А. Микроконтроллеры: архитектура, программирование, интерфейс / В. А. Юров. – СПб.: БХВ-Петербург, 2016. – 416 с.
- 11 Баранов, А. А. Архитектура 8-битных микроконтроллеров: учеб. пособие / А. А. Баранов. – СПб.: НИУ ИТМО, 2015. — 208 с.
- 12 Петин, Е. А. Практическое руководство по разработке устройств на микроконтроллерах STM32 с использованием среды разработки CubeMX и HAL-библиотек / Е. А. Петин. – СПб.: БХВ-Петербург, 2017. – 368 с.
- 13 Руководство пользователя по семейству микроконтроллеров STM32F103xx средней плотности: [пер. с англ.] / STMicroelectronics. — 2010. – URL: <https://www.st.com> (дата обращения: 04.03.2025).
- 14 Руководство по программированию микроконтроллеров STM32F100xx, STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx и STM32F107xx (Cortex-M3): [пер. с англ.] / STMicroelectronics. 2014. – URL: <https://www.st.com> (дата обращения: 04.03.2025).
- 15 Шевцов, С. И. Микроконтроллеры ARM Cortex-M3-M4-M7. Программирование на языке C / С. И. Шевцов. – СПб.: БХВ-Петербург, 2019. – 368 с.
- 16 Лысенко, А. В. Встроенные системы на основе микроконтроллеров / А. В. Лысенко. – М.: Горячая линия – Телеком, 2014. – 272 с.
- 17 Угрюмов, Е. П. Цифровая схемотехника: учеб. пособие / Е. П. Угрюмов. – СПб.: БХВ-Петербург, 2010. – 800 с.
- 18 Платунов, И. А. Высокоуровневые методы программирования: учеб. пособие / И. А. Платунов. – СПб.: СПбГУ, 2014. – 120 с.
- 19 Лутц, М. Изучаем Python / М. Лутц; пер. с англ. — 5-е изд. – СПб.: Символ-Плюс, 2019. – 1280 с.
- 20 Головкин, М. В. Нейронные сети: основные положения: учеб. пособие / М. В. Головкин. – СПб.: Лань, 2018. – 192 с.

21 ГОСТ 19.201–78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению. – М.: Стандартиформ, 2010.

22 ГОСТ 7.32–2017. Отчёт о научно-исследовательской работе. Структура и правила оформления. М.: Стандартиформ, 2017.

23 ГОСТ 2.105–2019. Единая система конструкторской документации. Общие требования к текстовым документам. – М.: Стандартиформ, 2019.

24 ГОСТ 7.0.100–2018. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: Стандартиформ, 2018.

ПРИЛОЖЕНИЕ А
(обязательное)

Интеллектуальная микроконтроллерная система управления охранного
комплекса

Листинг кода приложения микроконтроллера

```

// --- main.c -- точка входа и главный цикл ---
/* USER CODE BEGIN Header */
/**

*****
*****
* @file           : main.c
* @brief          : Main program body

*****
*****
* @attention
*
* Copyright (c) 2026 STMicroelectronics.
* All rights reserved.
*
* This software is licensed under terms that can be found in the
LICENSE file
* in the root directory of this software component.
* If no LICENSE file comes with this software, it is provided AS-
IS.
*

*****
*****
*/
/* USER CODE END Header */
/* Includes -----
-----*/
#include "main.h"

/* Private includes -----
-----*/
/* USER CODE BEGIN Includes */
#include "adc_glass.h"
#include "alarm.h"
#include "board_gpio.h"
#include "keypad.h"
#include "lcd1602.h"
#include "uart485.h"
/* USER CODE END Includes */

/* Private typedef -----
-----*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----
-----*/

```

```

-----*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -----
-----*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----
-----*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----
-----*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----
-----*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----
    -----*/

    /* Reset of all peripherals, Initializes the Flash interface and
    the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

```

```

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
/* USER CODE BEGIN 2 */
board_gpio_init();
adc_glass_init();
keypad_init();
lcd1602_init();
uart485_init();
alarm_init();
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    uart485_poll();
    alarm_tick();
    {
        char c = keypad_poll();
        if (c != 0)
            alarm_on_keypad_char(c);
    }
    {
        static uint32_t t_lcd;
        uint32_t now = HAL_GetTick();
        if ((now - t_lcd) >= 200U) {
            t_lcd = now;
            lcd1602_update(alarm_get_state_raw(), alarm_status_byte());
        }
    }
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */

```

```

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

    /** Initializes the RCC Oscillators according to the specified
    parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
    RCC_OscInitStruct.HSISState = RCC_HSI_ON;
    RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) !=
HAL_OK)
    {
        Error_Handler();
    }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error

```

```

return state */
__disable_irq();
while (1)
{
}
/* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line
number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name
 * @param line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name
and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

// --- proto.c - кадр протокола и CRC16 ---
#include "proto.h"

uint16_t proto_crc16_modbus(const uint8_t *data, size_t len)
{
    uint16_t crc = 0xFFFFU;
    for (size_t i = 0; i < len; ++i) {
        crc ^= (uint16_t)data[i];
        for (int b = 0; b < 8; ++b) {
            if (crc & 1U)
                crc = (uint16_t)((crc >> 1) ^ 0xA001U);
            else
                crc >>= 1;
        }
    }
    return crc;
}

size_t proto_build_frame(uint8_t addr, uint8_t cmd, const uint8_t
*payload, size_t len_payload,
                        uint8_t *buf, size_t buf_size)

```

```

{
    if (len_payload > 255U)
        return 0U;
    const size_t need = 4U + len_payload + 2U;
    if (need > buf_size)
        return 0U;
    buf[0] = PROTO_SYNC;
    buf[1] = addr;
    buf[2] = cmd;
    buf[3] = (uint8_t)len_payload;
    for (size_t i = 0; i < len_payload; ++i)
        buf[4U + i] = payload[i];
    uint16_t c = proto_crc16_modbus(buf, 4U + len_payload);
    buf[4U + len_payload] = (uint8_t)(c & 0xFFU);
    buf[4U + len_payload + 1U] = (uint8_t)((c >> 8) & 0xFFU);
    return need;
}

int proto_try_parse(const uint8_t *data, size_t len, uint8_t *addr,
uint8_t *cmd,
uint8_t *payload, size_t *payload_len, size_t
payload_cap)
{
    if (len < 6U)
        return 0;
    if (data[0] != PROTO_SYNC)
        return -1;

    *addr = data[1];
    *cmd = data[2];
    size_t plen = data[3];
    size_t total = 4U + plen + 2U;
    if (plen > payload_cap)
        return -1;
    if (len < total)
        return 0;

    uint16_t got = (uint16_t)data[4U + plen] | ((uint16_t)data[4U +
plen + 1U] << 8);
    uint16_t exp = proto_crc16_modbus(data, 4U + plen);
    if (got != exp)
        return -1;

    for (size_t k = 0; k < plen; ++k)
        payload[k] = data[4U + k];
    *payload_len = plen;
    return (int)total;
}

```

```

// --- uart485.c — USART2, приём и разбор кадров ---
/**
 * @file uart485.c
 * @brief USART2 (PA2/PA3) для RS-485: приём по прерыванию, разбор
кадров, ответ мастеру.
 * Реализация без HAL UART — только регистры, чтобы не подключать
отсутствующие в проекте hal_uart.c.
 */

#include "uart485.h"
#include "board.h"
#include "proto.h"
#include "alarm.h"

#include "stm32f1xx.h"
#include "stm32f1xx_hal.h"

#define RX_RING_SIZE 128U

static uint8_t s_rx_ring[RX_RING_SIZE];
static volatile uint32_t s_rx_w;
static volatile uint32_t s_rx_r;

static uint8_t s_rx_stream[192];
static size_t s_rx_stream_len;

static void uart485_gpio_init(void)
{
    __HAL_RCC_GPIOA_CLK_ENABLE();
    GPIO_InitTypeDef g = {0};
    g.Pin = GPIO_PIN_2;
    g.Mode = GPIO_MODE_AF_PP;
    g.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOA, &g);
    g.Pin = GPIO_PIN_3;
    g.Mode = GPIO_MODE_INPUT;
    g.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &g);
}

static void uart485_periph_init(void)
{
    __HAL_RCC_USART2_CLK_ENABLE();
    USART2->CR1 = 0;
    USART2->CR2 = 0;
    USART2->CR3 = 0;
    uint32_t pclk1 = HAL_RCC_GetPCLK1Freq();
    USART2->BRR = (uint16_t)(pclk1 / RS485_UART_BAUD);
    USART2->CR1 = USART_CR1_UE | USART_CR1_TE | USART_CR1_RE |

```

```

USART_CR1_RXNEIE;
    HAL_NVIC_SetPriority(USART2_IRQn, 6, 0);
    HAL_NVIC_EnableIRQ(USART2_IRQn);
}

void uart485_init(void)
{
    s_rx_w = s_rx_r = 0U;
    s_rx_stream_len = 0U;
    uart485_gpio_init();
    uart485_periph_init();
}

static void rx_push(uint8_t b)
{
    uint32_t next = (s_rx_w + 1U) % RX_RING_SIZE;
    if (next != s_rx_r) {
        s_rx_ring[s_rx_w] = b;
        s_rx_w = next;
    }
}

static int rx_pop(uint8_t *b)
{
    if (s_rx_w == s_rx_r)
        return 0;
    *b = s_rx_ring[s_rx_r];
    s_rx_r = (s_rx_r + 1U) % RX_RING_SIZE;
    return 1;
}

void USART2_IRQHandler(void)
{
    if (USART2->SR & USART_SR_RXNE) {
        uint8_t v = (uint8_t)(USART2->DR & 0xFFU);
        rx_push(v);
    }
}

void uart485_tx(const uint8_t *data, size_t len)
{
    for (size_t i = 0; i < len; ++i) {
        while ((USART2->SR & USART_SR_TXE) == 0U) {
        }
        USART2->DR = data[i];
    }
    while ((USART2->SR & USART_SR_TC) == 0U) {
    }
}

```

```

void uart485_poll(void)
{
    uint8_t b;
    while (rx_pop(&b)) {
        if (s_rx_stream_len < sizeof(s_rx_stream))
            s_rx_stream[s_rx_stream_len++] = b;
    }

    while (s_rx_stream_len >= 6U) {
        uint8_t addr, cmd, payload[32];
        size_t plen = 0U;
        int n = proto_try_parse(s_rx_stream, s_rx_stream_len,
                                &addr, &cmd, payload, &plen,
                                sizeof(payload));
        if (n > 0) {
            if (addr == MODULE_RS485_ADDR)
                alarm_on_rs485_command(cmd, payload, plen);
            memmove(s_rx_stream, s_rx_stream + (size_t)n,
s_rx_stream_len - (size_t)n);
            s_rx_stream_len -= (size_t)n;
            continue;
        }
        if (n == -1) {
            memmove(s_rx_stream, s_rx_stream + 1U, s_rx_stream_len
- 1U);
            s_rx_stream_len -= 1U;
            continue;
        }
        break;
    }
}

// --- alarm.c — логика охраны и ответ мастеру ---
/**
 * @file alarm.c
 * @brief Логика постановки/снятия, задержки на выход, формирование
байта статуса для ответа мастеру.
 *
 * При DEMO_MODE (board.h) биты дверь/окно/стекло в ответе
CMD_READ_STATUS подставляются по таймеру
 * после входа в «на охране» — см. demo.c (цикл 4×60 с без реальных
датчиков).
 */

#include "alarm.h"
#include "adc_glass.h"
#include "board.h"
#include "demo.h"

```

```

#include "proto.h"
#include "uart485.h"

#include "stm32f1xx_hal.h"

typedef enum {
    ALARM_DISARMED = 0,
    ALARM_EXIT_DELAY,
    ALARM_ARMED
} alarm_state_t;

static alarm_state_t s_state = ALARM_DISARMED;
/** HAL_GetTick() в момент перехода в ALARM_ARMED — база для демо-
цикла (demo_apply). */
static uint32_t s_armed_mark_ms;
static uint32_t s_exit_deadline_ms;
static char s_pin_buf[8];
static uint8_t s_pin_len;

static uint8_t s_pin_ok(void)
{
    if (s_pin_len != 4U)
        return 0U;
    return (s_pin_buf[0] == (char)('0' + ALARM_PIN_0) &&
s_pin_buf[1] == (char)('0' + ALARM_PIN_1)
        && s_pin_buf[2] == (char)('0' + ALARM_PIN_2) &&
s_pin_buf[3] == (char)('0' + ALARM_PIN_3))
        ? 1U
        : 0U;
}

static void s_pin_reset(void)
{
    s_pin_len = 0U;
}

static uint8_t door_violation(void)
{
    return (HAL_GPIO_ReadPin(REED_DOOR_PORT, REED_DOOR_PIN) ==
GPIO_PIN_SET) ? 1U : 0U;
}

static uint8_t window_violation(void)
{
    return (HAL_GPIO_ReadPin(REED_WINDOW_PORT, REED_WINDOW_PIN) ==
GPIO_PIN_SET) ? 1U : 0U;
}

static uint8_t glass_violation(void)

```

```

{
    uint16_t v = (uint16_t)adc_glass_raw();
    return (v >= GLASS_ADC_THRESHOLD) ? 1U : 0U;
}

uint8_t alarm_status_byte(void)
{
    uint8_t s = 0U;
    if (s_state == ALARM_ARMED)
        s |= STATUS_ARMED;
    if (s_state == ALARM_EXIT_DELAY)
        s |= STATUS_EXIT_DELAY;

    uint8_t d = door_violation();
    uint8_t w = window_violation();
    uint8_t g = glass_violation();
#ifdef DEMO_MODE
    if (s_state == ALARM_ARMED) {
        demo_apply(1U, s_armed_mark_ms, &d, &w, &g);
    }
#endif
    if (d != 0U)
        s |= STATUS_DOOR;
    if (w != 0U)
        s |= STATUS_WINDOW;
    if (g != 0U)
        s |= STATUS_GLASS;
    return s;
}

static void send_status_response(void)
{
    uint8_t st = alarm_status_byte();
    uint8_t buf[16];
    size_t n = proto_build_frame((uint8_t)MODULE_RS485_ADDR,
    CMD_READ_STATUS, &st, 1U, buf, sizeof(buf));
    if (n > 0U)
        uart485_tx(buf, n);
}

void alarm_on_rs485_command(uint8_t cmd, const uint8_t *payload,
size_t len)
{
    (void)payload;
    (void)len;
    if (cmd == CMD_READ_STATUS) {
        send_status_response();
        return;
    }
}

```

```

    if (cmd == CMD_ARM) {
        s_state = ALARM_EXIT_DELAY;
        s_exit_deadline_ms = HAL_GetTick() + EXIT_DELAY_MS;
        s_pin_reset();
        return;
    }
    if (cmd == CMD_DISARM) {
        s_state = ALARM_DISARMED;
        s_armed_mark_ms = 0U;
        s_pin_reset();
    }
}

void alarm_on_keypad_char(char c)
{
    if (c == '#') {
        if (s_pin_ok()) {
            if (s_state == ALARM_DISARMED) {
                s_state = ALARM_EXIT_DELAY;
                s_exit_deadline_ms = HAL_GetTick() + EXIT_DELAY_MS;
            } else {
                s_state = ALARM_DISARMED;
                s_armed_mark_ms = 0U;
            }
        }
        s_pin_reset();
        return;
    }
    if (c >= '0' && c <= '9' && s_pin_len < sizeof(s_pin_buf)) {
        s_pin_buf[s_pin_len++] = c;
    }
}

void alarm_tick(void)
{
    if (s_state == ALARM_EXIT_DELAY) {
        if ((int32_t)(HAL_GetTick() - s_exit_deadline_ms) >= 0) {
            s_state = ALARM_ARMED;
            s_armed_mark_ms = HAL_GetTick();
        }
    }
}

uint8_t alarm_get_state_raw(void)
{
    return (uint8_t)s_state;
}

void alarm_init(void)

```

```
{  
    s_state = ALARM_DISARMED;  
    s_armed_mark_ms = 0U;  
    s_pin_reset();  
}
```

ПРИЛОЖЕНИЕ Б
(обязательное)

Интеллектуальная микроконтроллерная система управления охранного
комплекса

Листинг кода приложения Telegram-бота

```

# --- telegram_bot.py — точка входа и обработчики Telegram ---
#!/usr/bin/env python3
"""
Telegram-бот: Qwen2.5-3B + инструменты (статус, ARM/DISARM, журнал
из БД).
Требуется запущенный rc_master с открытым СОМ для команд на шину;
журнал читается из той же БД.
"""

from __future__ import annotations

import argparse
import asyncio
import logging
import os
import sys
from collections import defaultdict
from pathlib import Path

_ROOT = Path(__file__).resolve().parent.parent
if str(_ROOT) not in sys.path:
    sys.path.insert(0, str(_ROOT))

_CHATBOT_DIR = Path(__file__).resolve().parent

def _load_dotenv() -> None:
    env_path = _CHATBOT_DIR / ".env"
    if not env_path.is_file():
        return
    try:
        from dotenv import load_dotenv
    except ImportError:
        return
    load_dotenv(env_path, override=False)

from telegram import Update
from telegram.constants import ChatAction
from telegram.ext import Application, CommandHandler, ContextTypes,
MessageHandler, filters

from llm_qwen import (
    DEFAULT_HF_MODEL,
    DEFAULT_OLLAMA_MODEL,
    HINT_FILE,
    LLMRunner,
    load_system_prompt,
    load_text,

```

```

)
from telegram_agent import build_telegram_system_prompt,
run_agent_turn

import security_api_client as api

logging.basicConfig(
    format="%asctime)s %(levelname)s %(name)s: %(message)s",
    level=logging.INFO,
)
log = logging.getLogger(__name__)

def _parse_allowed_ids() -> set[int] | None:
    raw = os.environ.get("TELEGRAM_ALLOWED_USER_IDS", "").strip()
    if not raw:
        return None
    ids: set[int] = set()
    for part in raw.split(","):
        part = part.strip()
        if part:
            ids.add(int(part))
    return ids

def _resolve_model(backend: str, override: str | None) -> str:
    if override:
        return override
    if backend == "ollama":
        return DEFAULT_OLLAMA_MODEL
    return DEFAULT_HF_MODEL

def _check_allowed(context: ContextTypes.DEFAULT_TYPE, uid: int) -> bool:
    allowed = context.bot_data.get("allowed_ids")
    return allowed is None or uid in allowed

class BotState:
    def __init__(self, runner: LLMRunner, system_prompt: str,
backend: str, model: str) -> None:
        self.runner = runner
        self.system_prompt = system_prompt
        self.backend = backend
        self.model = model
        self.histories: dict[int, list[dict]] = defaultdict(list)

    def reset(self, user_id: int) -> None:

```

```

        self.histories[user_id] = [{"role": "system", "content":
self.system_prompt}]

    def get_messages(self, user_id: int) -> list[dict]:
        if user_id not in self.histories or not
self.histories[user_id]:
            self.reset(user_id)
        return self.histories[user_id]

async def cmd_start(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    st: BotState = context.bot_data["state"]
    uid = update.effective_user.id
    st.reset(uid)
    await update.message.reply_text(
        "Привет. Можно писать обычным языком: статус охраны,
постановка/снятие, журнал за N минут.\n"
        "Команды: /status, /arm [адрес], /disarm [адрес], /clear,
/help."
    )

async def cmd_help(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    hint = load_text(HINT_FILE, "См. файл ПОДСКАЗКА.txt в каталоге
бота.")
    chunk = hint[:4000]
    await update.message.reply_text(chunk)
    if len(hint) > 4000:
        await update.message.reply_text("... (полный текст в
ПОДСКАЗКА.txt на сервере)")

async def cmd_clear(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    st: BotState = context.bot_data["state"]
    st.reset(update.effective_user.id)
    await update.message.reply_text("История диалога очищена.")

async def cmd_status(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    if not _check_allowed(context, update.effective_user.id):
        await update.message.reply_text("Доступ запрещён.")
        return
    await update.message.reply_text(api.get_status_text())

```

```

def _parse_addr(args: list[str]) -> int:
    if args:
        return max(1, min(247, int(args[0])))
    return max(1, min(247,
int(os.environ.get("TELEGRAM_DEFAULT_SLAVE_ADDR", "1"))))

async def cmd_arm(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    if not _check_allowed(context, update.effective_user.id):
        await update.message.reply_text("Доступ запрещён.")
        return
    try:
        addr = _parse_addr(list(context.args or []))
    except ValueError:
        await update.message.reply_text("Неверный адрес. Пример:
/arm 1")
        return
    await update.message.reply_text(api.post_arm(addr))

async def cmd_disarm(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    if not _check_allowed(context, update.effective_user.id):
        await update.message.reply_text("Доступ запрещён.")
        return
    try:
        addr = _parse_addr(list(context.args or []))
    except ValueError:
        await update.message.reply_text("Неверный адрес. Пример:
/disarm 1")
        return
    await update.message.reply_text(api.post_disarm(addr))

async def on_text(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> None:
    if not update.message or not update.message.text:
        return

    uid = update.effective_user.id
    if not _check_allowed(context, uid):
        await update.message.reply_text("Доступ запрещён.")
        log.warning("Отклонён user_id=%s", uid)
        return

    text = update.message.text.strip()
    if not text:
        return

```

```

st: BotState = context.bot_data["state"]
messages = st.get_messages(uid)
messages.append({"role": "user", "content": text})

await context.bot.send_chat_action(
    chat_id=update.effective_chat.id,
    action=ChatAction.TYPING,
)

try:
    reply = await asyncio.to_thread(
        run_agent_turn,
        messages,
        backend=st.backend,
        model=st.model,
        runner_complete=st.runner.complete,
    )
except Exception as e:
    log.exception("Agent error")
    messages.pop()
    await update.message.reply_text(f"Ошибка: {e}")
    return

messages.append({"role": "assistant", "content": reply})
if len(reply) <= 4096:
    await update.message.reply_text(reply)
else:
    for i in range(0, len(reply), 4000):
        await update.message.reply_text(reply[i : i + 4000])

def main() -> None:
    _load_dotenv()

    try:
        from pc_master import storage as _pc_storage

        _pc_storage.init_storage()
    except Exception as e:
        log.warning("pc_master.storage недоступен (журнал в БД может не работать): %s", e)

    token = os.environ.get("TELEGRAM_BOT_TOKEN", "").strip()
    if not token:
        print("Задайте переменную окружения TELEGRAM_BOT_TOKEN",
              file=sys.stderr)
        sys.exit(1)

```

```

    p = argparse.ArgumentParser(description="Telegram-бот c
Qwen2.5-3B")
    p.add_argument(
        "--backend",
        choices=("ollama", "transformers"),
        default=os.environ.get("CHATBOT_BACKEND", "ollama"),
    )
    p.add_argument("--model", default=None)
    args = p.parse_args()

    model = _resolve_model(args.backend, args.model)
    allowed = _parse_allowed_ids()

    print("Загрузка LLM...")
    runner = LLMRunner(args.backend, model)
    base = load_system_prompt()
    system = build_telegram_system_prompt(base)
    state = BotState(runner, system, args.backend, model)

    app = Application.builder().token(token).build()
    app.bot_data["state"] = state
    app.bot_data["allowed_ids"] = allowed

    app.add_handler(CommandHandler("start", cmd_start))
    app.add_handler(CommandHandler("help", cmd_help))
    app.add_handler(CommandHandler("clear", cmd_clear))
    app.add_handler(CommandHandler("status", cmd_status))
    app.add_handler(CommandHandler("arm", cmd_arm))
    app.add_handler(CommandHandler("disarm", cmd_disarm))
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND,
on_text))

    log.info("Бот запущен (backend=%s, model=%s)", args.backend,
model)
    app.run_polling(allowed_updates=Update.ALL_TYPES)

if __name__ == "__main__":
    main()

# --- telegram_agent.py - агент, инструменты и Ollama ---
# -*- coding: utf-8 -*-
"""
Агент для Telegram: вызов инструментов через Ollama (function
calling) и запасной разбор фраз.
"""

from __future__ import annotations

```

```

import json
import os
import re
import sys
from pathlib import Path
from typing import Any, Callable

# Репозиторий: родитель каталога chatbot
_ROOT = Path(__file__).resolve().parent.parent
if str(_ROOT) not in sys.path:
    sys.path.insert(0, str(_ROOT))

from llm_qwen import chat_ollama, chat_ollama_raw

import security_api_client as api

ExecFn = Callable[..., str]

OLLAMA_TOOLS: list[dict[str, Any]] = [
    {
        "type": "function",
        "function": {
            "name": "get_alarm_status",
            "description": "Узнать текущее состояние всех ведомых узлов: связь RS-485, на охране или снято, зоны (дверь/окно/стекло).",
            "parameters": {"type": "object", "properties": {}},
        },
    },
    {
        "type": "function",
        "function": {
            "name": "arm_slave",
            "description": "Поставить выбранный узел на охрану (команда ARM по RS-485).",
            "parameters": {
                "type": "object",
                "properties": {
                    "slave_addr": {
                        "type": "integer",
                        "description": "Адрес ведомого на шине, 1-247",
                    }
                }
            },
            "required": ["slave_addr"],
        },
    },
]

```

```

        "type": "function",
        "function": {
            "name": "disarm_slave",
            "description": "Снять выбранный узел с охраны (команда
DISARM по RS-485).",
            "parameters": {
                "type": "object",
                "properties": {
                    "slave_addr": {
                        "type": "integer",
                        "description": "Адрес ведомого на шине, 1-
247",
                    }
                },
                "required": ["slave_addr"],
            },
        },
    },
    {
        "type": "function",
        "function": {
            "name": "get_journal",
            "description": "Получить журнал событий из локальной БД
за последние N минут (записи мастера ПК).",
            "parameters": {
                "type": "object",
                "properties": {
                    "minutes": {
                        "type": "integer",
                        "description": "Сколько минут назад от
текущего времени, например 5 или 30",
                    }
                },
                "required": ["minutes"],
            },
        },
    },
]

```

```

def _default_slave_addr() -> int:
    try:
        return int(os.environ.get("TELEGRAM_DEFAULT_SLAVE_ADDR",
"1"))
    except ValueError:
        return 1

```

```

def _journal_text(minutes: int) -> str:

```

```

from pc_master import storage

if not storage.is_enabled():
    storage.init_storage()
if not storage.is_enabled():
    return (
        "База данных журнала недоступна (DIPLOMA_DB=None или
ошибка). "
        "Включите SQLite/MSSQL у мастера ПК."
    )
rows = storage.journal_since_minutes(minutes, limit=300)
if not rows:
    return f"За последние {minutes} мин. записей в журнале
нет."
lines = []
for r in rows:
    lines.append(f"{r['ts']} [{r['evt']}] {r['message']}")
text = "\n".join(lines)
if len(text) > 3500:
    text = text[-3500:]
    text = "... (обрезано)\n" + text
return text

def _execute_tool(name: str, args: dict[str, Any], default_addr:
int) -> str:
    try:
        if name == "get_alarm_status":
            return api.get_status_text()
        if name == "arm_slave":
            addr = int(args.get("slave_addr", default_addr))
            return api.post_arm(addr)
        if name == "disarm_slave":
            addr = int(args.get("slave_addr", default_addr))
            return api.post_disarm(addr)
        if name == "get_journal":
            minutes = max(1, min(1440, int(args.get("minutes",
5))))
            return _journal_text(minutes)
    except Exception as e:
        return f"Ошибка инструмента {name}: {e}"
    return f"Неизвестный инструмент: {name}"

def fallback_intent_ru(text: str, default_addr: int) ->
list[tuple[str, dict[str, Any]]]:
    """Простой разбор намерений без LLM (русский)."""
    t = text.lower().strip()
    out: list[tuple[str, dict[str, Any]]] = []

```

```

        if any(x in t for x in ("журнал", "лог", "запис", "события")):
            m = re.search(r"(\d+)\s*(?:минут|мин\.?|минуты|минуту)\b",
t)
            minutes = int(m.group(1)) if m else 5
            out.append(("get_journal", {"minutes": max(1, min(1440,
minutes))}))
            return out

        addr = default_addr
        for pat in (r"узел\s*(\d+)", r"адрес\s*(\d+)", r"№\s*(\d+)",
r"контроллер\s*(\d+)"):
            mm = re.search(pat, t)
            if mm:
                addr = max(1, min(247, int(mm.group(1))))
                break

        disarm_kw = (
            "снять с охраны",
            "сними с охраны",
            "снять",
            "снятие",
            "disarm",
            "отключи охрану",
            "убери с охраны",
            "с охраны",
        )
        arm_kw = (
            "на охрану",
            "поставь на охрану",
            "постановк",
            "arm",
            "включи охрану",
            "поставить",
        )
        status_kw = ("статус", "состояние", "что на охране", "как
узлы", "связь с узлами")

        if any(k in t for k in disarm_kw) and not any(k in t for k in
arm_kw):
            out.append(("disarm_slave", {"slave_addr": addr}))
            return out
        if any(k in t for k in arm_kw):
            out.append(("arm_slave", {"slave_addr": addr}))
            return out
        if any(k in t for k in status_kw):
            out.append(("get_alarm_status", {}))
            return out
        return []

```

```

def _summarize_ollama(user_question: str, facts: str, model: str) -
> str:
    prompt = (
        f"Вопрос пользователя:\n{user_question}\n\n"
        f"Факты от системы охраны:\n{facts}\n\n"
        "Ответь кратко и по-русски пользователю. Не выдумывай
данные."
    )
    return chat_ollama([{"role": "user", "content": prompt}],
model)

def run_agent_turn(
    messages: list[dict],
    *,
    backend: str,
    model: str,
    runner_complete: ExecFn,
) -> str:
    """
    Добавляет в историю ответ пользователю.
    Для ollama: до 6 раундов tool-calling; иначе fallback_intent +
опционально суммаризация.
    """
    default_addr = _default_slave_addr()
    user_text = ""
    for m in reversed(messages):
        if m.get("role") == "user":
            user_text = str(m.get("content", ""))
            break

    if backend == "ollama":
        msgs = [dict(x) for x in messages]
        for _ in range(6):
            data = chat_ollama_raw(msgs, model, tools=OLLAMA_TOOLS)
            msg = data["choices"][0]["message"]
            tool_calls = msg.get("tool_calls")
            if tool_calls:
                msgs.append(msg)
                for tc in tool_calls:
                    fn = tc.get("function", {}).get("name", "")
                    raw = tc.get("function", {}).get("arguments")
or "{}"

                try:
                    args = json.loads(raw) if isinstance(raw,
str) else {}
                except json.JSONDecodeError:

```

```

        args = {}
        out = _execute_tool(fn, args, default_addr)
        msgs.append(
            {
                "role": "tool",
                "tool_call_id": tc.get("id", ""),
                "content": out,
            }
        )
        continue
    text = (msg.get("content") or "").strip()
    if text:
        return text
    break

# Fallback: эвристика + вызов инструментов + короткая
суммаризация (ollama) или только факты
intents = fallback_intent_ru(user_text, default_addr)
if intents:
    parts = []
    for name, args in intents:
        parts.append(_execute_tool(name, args, default_addr))
    combined = "\n---\n".join(parts)
    if backend == "ollama":
        return _summarize_ollama(user_text, combined, model)
    return combined

return runner_complete(messages)

def build_telegram_system_prompt(base: str) -> str:
    extra = (
        "\n\nТы помогаешь управлять охранной системой по RS-485. "
        "У тебя есть инструменты: узнать статус узлов, поставить на
охрану, снять с охраны, "
        "прочитать журнал из БД за последние N минут. "
        "Если пользователь формулирует просьбу естественным языком
(например «сними первый с охраны», "
        "«дай журнал за 10 минут»), вызывай соответствующие
инструменты с подходящим адресом или числом минут. "
        "Если адрес не указан, используй значение по умолчанию из
контекста (часто 1). "
        "Не придумывай состояние оборудования — опирайся на ответы
инструментов."
    )
    return (base or "").strip() + extra

# --- security_api_client.py — HTTP-клиент к API мастера ПК ---
# -*- coding: utf-8 -*-

```

```

"""HTTP-клиент к локальному API мастера ПК
(pc_master.control_http)."""

from __future__ import annotations

import json
import os
from typing import Any

import requests

DEFAULT_BASE = os.environ.get("DIPLOMA_CONTROL_API_URL",
"http://127.0.0.1:18765").rstrip("/")
_TOKEN = os.environ.get("DIPLOMA_CONTROL_API_TOKEN", "").strip()

def _headers() -> dict[str, str]:
    h = {"Content-Type": "application/json"}
    if _TOKEN:
        h["Authorization"] = f"Bearer {_TOKEN}"
    return h

def get_status_raw() -> dict[str, Any]:
    r = requests.get(f"{DEFAULT_BASE}/status", headers=_headers(),
timeout=10)
    r.raise_for_status()
    return r.json()

def get_status_text() -> str:
    try:
        d = get_status_raw()
    except requests.RequestException as e:
        return f"Ошибка связи с мастером ПК: {e}. Запущен ли
pc_master и открыт ли COM?"
    if not d.get("ok"):
        return json.dumps(d, ensure_ascii=False)
    lines = []
    for s in d.get("slaves", []):
        lines.append(
            f"узел {s['address']}: "
            f"{'онлайн' if s['online'] else 'офлайн'},
{s['status_text']}, "
            f"байт статуса={s['status_byte']},
время={s['last_seen']}"
        )
    return "\n".join(lines) if lines else "Нет строк опроса."

```

```

def post_arm(addr: int) -> str:
    return _post_cmd("/arm", addr)

def post_disarm(addr: int) -> str:
    return _post_cmd("/disarm", addr)

def _post_cmd(path: str, addr: int) -> str:
    try:
        r = requests.post(
            f"{DEFAULT_BASE}{path}",
            headers=_headers(),
            json={"addr": addr},
            timeout=15,
        )
        txt = r.text
        try:
            d = r.json()
            txt = json.dumps(d, ensure_ascii=False, indent=0)
        except json.JSONDecodeError:
            pass
        if r.status_code >= 400:
            return f"HTTP {r.status_code}: {txt}"
        return txt
    except requests.RequestException as e:
        return f"Ошибка запроса: {e}"

# --- llm_qwen.py - вызов Qwen через Ollama/transformers ---
"""
Общая логика Qwen2.5-3B: Ollama или transformers. Используется
chat.py и telegram_bot.py.
"""

from __future__ import annotations

import os
from pathlib import Path

import requests

DIR = Path(__file__).resolve().parent
SYSTEM_PROMPT_FILE = DIR / "system_prompt.txt"
HINT_FILE = DIR / "ПОДСКАЗКА.txt"

DEFAULT_OLLAMA = os.environ.get("OLLAMA_HOST",
    "http://127.0.0.1:11434").rstrip("/")
DEFAULT_OLLAMA_MODEL = os.environ.get("QWEN_MODEL", "qwen2.5:3b")

```

```
DEFAULT_HF_MODEL = os.environ.get("HF_MODEL", "Qwen/Qwen2.5-3B-Instruct")
```

```
def load_text(path: Path, fallback: str) -> str:
    if path.is_file():
        return path.read_text(encoding="utf-8").strip()
    return fallback
```

```
def load_system_prompt() -> str:
    return load_text(
        SYSTEM_PROMPT_FILE,
        "Ты полезный ассистент. Отвечай кратко.",
    )
```

```
def chat_ollama(messages: list[dict], model: str, ollama_base: str
| None = None) -> str:
    data = chat_ollama_raw(messages, model, tools=None,
ollama_base=ollama_base)
    msg = data["choices"][0]["message"]
    return (msg.get("content") or "").strip()
```

```
def chat_ollama_raw(
    messages: list[dict],
    model: str,
    tools: list[dict] | None = None,
    ollama_base: str | None = None,
) -> dict:
    """Полный ответ OpenAI-совместимого API (в т.ч. tool_calls)."""
    base = (ollama_base or DEFAULT_OLLAMA).rstrip("/")
    url = f"{base}/v1/chat/completions"
    payload: dict = {
        "model": model,
        "messages": messages,
        "stream": False,
    }
    if tools:
        payload["tools"] = tools
    r = requests.post(url, json=payload, timeout=600)
    r.raise_for_status()
    return r.json()
```

```
def load_transformers(model_id: str):
    try:
        import torch
```

```

        from transformers import AutoModelForCausalLM,
AutoTokenizer
    except ImportError as e:
        raise RuntimeError(
            "Установите: pip install transformers torch accelerate"
        ) from e

    tokenizer = AutoTokenizer.from_pretrained(model_id,
trust_remote_code=True)
    kwargs = {"trust_remote_code": True}
    if torch.cuda.is_available():
        kwargs["torch_dtype"] = torch.float16
        kwargs["device_map"] = "auto"
    else:
        kwargs["torch_dtype"] = torch.float32

    model = AutoModelForCausalLM.from_pretrained(model_id,
**kwargs)
    if not torch.cuda.is_available():
        model = model.to(torch.device("cpu"))
    model.eval()
    return tokenizer, model

def chat_transformers_once(
    tokenizer,
    model,
    messages: list[dict],
) -> str:
    import torch

    text = tokenizer.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True,
    )
    inputs = tokenizer(text, return_tensors="pt")
    dev = next(model.parameters()).device
    inputs = {k: v.to(dev) for k, v in inputs.items()}

    with torch.no_grad():
        out = model.generate(
            **inputs,
            max_new_tokens=512,
            do_sample=True,
            temperature=0.7,
            top_p=0.9,
        )
    gen = out[0, inputs["input_ids"].shape[1] :]

```

```

        return tokenizer.decode(gen, skip_special_tokens=True).strip()

class LLMRunner:
    """Один экземпляр на процесс: для transformers модель
    загружается один раз."""

    def __init__(self, backend: str, model_name: str) -> None:
        self.backend = backend
        self.model_name = model_name
        self._tf_tokenizer = None
        self._tf_model = None
        if backend == "transformers":
            self._tf_tokenizer, self._tf_model =
load_transformers(model_name)

    def complete(self, messages: list[dict]) -> str:
        if self.backend == "ollama":
            return chat_ollama(messages, self.model_name)
        assert self._tf_tokenizer is not None and self._tf_model is
not None
        return chat_transformers_once(self._tf_tokenizer,
self._tf_model, messages)

```

ПРИЛОЖЕНИЕ В
(обязательное)

Интеллектуальная микроконтроллерная система управления охранного
комплекса

Листинг кода приложения мастера опроса

```

# -*- coding: utf-8 -*-
"""
Окно мастера: tkinter + pyserial (без Qt).
Запуск из каталога репозитория: python -m pc_master
"""

from __future__ import annotations

import os
import tkinter as tk
from tkinter import ttk

from . import protocol
from . import storage
from .control_http import ControlApiServer
from .rs485_master import Rs485Master, SlaveRow

def _led_fill_for_zone(status_byte: int, zone_bit: int, blink_on:
bool) -> str:
    """
    Цвет индикатора зоны:
    - сработка датчика — красный, мигает (два оттенка);
    - на охране или задержка выхода — зелёный;
    - снято с охраны — серый.
    """
    if status_byte & zone_bit:
        return "#ef4444" if blink_on else "#7f1d1d"
    if status_byte & (protocol.STATUS_ARMED |
protocol.STATUS_EXIT_DELAY):
        return "#22c55e"
    return "#9ca3af"

def _pick_status_byte_from_tree(
    tree: ttk.Treeview, rows: list[SlaveRow]
) -> tuple[int, str]:
    """(status_byte, подпись источника)."""
    sel = tree.selection()
    if sel:
        vals = tree.item(sel[0], "values")
        if vals:
            try:
                addr = int(vals[0])
            except (ValueError, TypeError):
                addr = None
            if addr is not None:

```

```

        for row in rows:
            if row.address == addr:
                src = f"узел {addr}"
                if row.online:
                    return row.status_byte, src
                return 0, f"{src} (нет связи)"
    for row in rows:
        if row.online:
            return row.status_byte, f"узел {row.address} (авто)"
    return 0, "нет данных"

def main() -> None:
    master = Rs485Master()
    storage.init_storage()
    storage.apply_loaded_config_to_master(master)

    root = tk.Tk()
    root.title("Охрана – мастер RS-485 (Python)")
    root.geometry("1180x580")

    port_var = tk.StringVar(value=master.port_name)
    baud_var = tk.IntVar(value=master.baud)
    amin_var = tk.IntVar(value=master.addr_min)
    amax_var = tk.IntVar(value=master.addr_max)
    poll_var = tk.IntVar(value=master.poll_interval_ms)
    tout_var = tk.IntVar(value=master.reply_timeout_ms)

    if storage.is_enabled():
        master.set_journal_hook(storage.append_journal_log_line)

    err_lbl = tk.Label(root, text="", fg="red", wraplength=1100,
        justify="left")
    err_lbl.pack(anchor="w", padx=8, pady=4)

    poll_status = tk.Label(root, text="")
    poll_status.pack(anchor="w", padx=8)

    demo_banner = tk.Label(
        root,
        text="",
        fg="#b45309",
        bg="#fff7ed",
        font=("Segoe UI", 9, "bold"),
        padx=8,
        pady=4,
    )

```

```

api_server: ControlApiServer | None = None
api_extra = ""
if os.environ.get("DIPLOMA_CONTROL_API", "1").strip().lower()
not in ("0", "false", "no", "off"):
    api_server = ControlApiServer(master)
    if api_server.start():
        api_extra = f" | Локальный API: {api_server.base_url}
(Telegram-бот) "

db_lbl = tk.Label(
    root,
    text=f"Хранилище: {storage.describe_backend()}{api_extra}",
    fg="#475569",
    font=("Segoe UI", 8),
    wraplength=1100,
    justify="left",
)
db_lbl.pack(anchor="w", padx=8, pady=(0, 2))

top = ttk.Frame(root)
top.pack(fill="x", padx=8, pady=4)

ttk.Label(top, text="COM-порт:").grid(row=0, column=0,
sticky="w")
port_combo = ttk.Combobox(top, textvariable=port_var, width=14,
state="readonly")
port_combo.grid(row=0, column=1, padx=4)

def refresh_ports() -> None:
    ports = master.refresh_ports()
    port_combo["values"] = ports
    if ports and port_var.get() not in ports:
        port_var.set(ports[0])

ttk.Button(top, text="Обновить список",
command=refresh_ports).grid(row=0, column=2, padx=4)
ttk.Label(top, text="Скорость:").grid(row=0, column=3,
padx=(16, 0))
baud_spin = ttk.Spinbox(top, from_=1200, to=921600,
increment=1200, textvariable=baud_var, width=10)
baud_spin.grid(row=0, column=4, padx=4)

def save_settings_to_db() -> None:
    if not storage.is_enabled():
        return
    storage.save_config(

```

```

        storage.build_config_dict_from_vars(
            port_var, baud_var, amin_var, amax_var, poll_var,
tout_var
        )
    )

def toggle_demo() -> None:
    if master.demo_mode:
        master.stop_demo()
        btn_demo.config(text="Демо (скриншот)")
        demo_banner.config(text="")
        demo_banner.pack_forget()
    else:
        if master.serial_open and not master.demo_mode:
            master.close_port()
            btn_open.config(text="Открыть порт")
            master.start_demo()
            btn_demo.config(text="Остановить демо")
            demo_banner.config(
                text="ДЕМО-РЕЖИМ: данные смоделированы для
иллюстрации в пояснительной записке "
                "(узел 1 – тревога по окну)"
            )
            demo_banner.pack(after=poll_status, anchor="w",
fill="x", padx=8, pady=(0, 4))
            btn_open.config(text="Закрыть порт")

        def _select_demo_node() -> None:
            if tree.exists("1"):
                tree.selection_set("1")
                tree.focus("1")
                tree.see("1")

        root.after(300, _select_demo_node)

    btn_demo = ttk.Button(top, text="Демо (скриншот)",
command=toggle_demo)
    btn_demo.grid(row=0, column=7, padx=8)

def toggle_port() -> None:
    if master.demo_mode:
        err_lbl.config(text="Остановите ДЕМО-режим перед
работой с COM", fg="red")
        return
    if master.serial_open:
        master.close_port()
        btn_open.config(text="Открыть порт")

```

```

else:
    master.port_name = port_var.get().strip() or "COM1"
    master.baud = int(baud_var.get())
    master.poll_interval_ms = int(poll_var.get())
    master.reply_timeout_ms = int(tout_var.get())
    master.set_address_range(int(amin_var.get()),
int(amax_var.get()))
    master.open_port()
    err_lbl.config(text=master.last_error or "Ошибок нет",
fg=("red" if master.last_error else "black"))
    btn_open.config(text="Закрыть порт")
    if not master.last_error:
        save_settings_to_db()

    btn_open = ttk.Button(top, text="Открыть порт",
command=toggle_port)
    btn_open.grid(row=0, column=5, padx=12)

    ttk.Button(top, text="Сохранить настройки в БД",
command=save_settings_to_db).grid(
        row=0, column=6, padx=8
    )

    mid = ttk.Frame(root)
    mid.pack(fill="x", padx=8, pady=4)
    ttk.Label(mid, text="Адреса ведомых:").grid(row=0, column=0)
    ttk.Spinbox(mid, from_=1, to=247, textvariable=amin_var,
width=5).grid(row=0, column=1, padx=4)
    ttk.Label(mid, text="...").grid(row=0, column=2)
    ttk.Spinbox(mid, from_=1, to=247, textvariable=amax_var,
width=5).grid(row=0, column=3, padx=4)
    ttk.Label(mid, text="Опрос, мс:").grid(row=0, column=4,
padx=(12, 0))
    ttk.Spinbox(mid, from_=200, to=60000, increment=100,
textvariable=poll_var, width=8).grid(
        row=0, column=5, padx=4
    )
    ttk.Label(mid, text="Таймаут ответа, мс:").grid(row=0,
column=6)
    ttk.Spinbox(mid, from_=20, to=2000, increment=10,
textvariable=tout_var, width=6).grid(
        row=0, column=7, padx=4
    )

    pan = ttk.Panedwindow(root, orient=tk.HORIZONTAL)
    pan.pack(fill="both", expand=True, padx=8, pady=4)

```

```

left_fr = ttk.Frame(pan)
mid_fr = ttk.Frame(pan, width=300)
right_fr = ttk.Frame(pan, width=240)
pan.add(left_fr, weight=4)
pan.add(mid_fr, weight=2)
pan.add(right_fr, weight=2)

cols = ("addr", "online", "status", "seen")
tree = ttk.Treeview(left_fr, columns=cols, show="headings",
height=14)
tree.heading("addr", text="№")
tree.heading("online", text="СВЯЗЬ")
tree.heading("status", text="Статус")
tree.heading("seen", text="Время")
tree.column("addr", width=40)
tree.column("online", width=70)
tree.column("status", width=280)
tree.column("seen", width=80)
tree.pack(side="left", fill="both", expand=True)
sb = ttk.Scrollbar(left_fr, orient="vertical",
command=tree.yview)
sb.pack(side="right", fill="y")
tree.configure(yscrollcommand=sb.set)

def arm_selected() -> None:
    sel = tree.selection()
    if not sel:
        return
    a = int(tree.item(sel[0], "values")[0])
    if not master.serial_open and not master.demo_mode:
        storage.add_security_action("arm", a, status="failed",
error_message="порт закрыт")
        return
    storage.add_security_action("arm", a, status="sent")
    master.send_arm(a)

def disarm_selected() -> None:
    sel = tree.selection()
    if not sel:
        return
    a = int(tree.item(sel[0], "values")[0])
    if not master.serial_open and not master.demo_mode:
        storage.add_security_action("disarm", a,
status="failed", error_message="порт закрыт")
        return
    storage.add_security_action("disarm", a, status="sent")
    master.send_disarm(a)

```

```

    btns = ttk.Frame(left_fr)
    btns.pack(fill="x", pady=4)
    ttk.Button(btns, text="На охрану (выбранный)",
command=arm_selected).pack(side="left", padx=4)
    ttk.Button(btns, text="Снять (выбранный)",
command=disarm_selected).pack(side="left", padx=4)

    log_frame = ttk.LabelFrame(mid_fr, text="Журнал")
    log_frame.pack(fill="both", expand=True)
    log_txt = tk.Text(log_frame, height=20, font=("Consolas", 9),
wrap="word", state="disabled")
    log_txt.pack(side="left", fill="both", expand=True)
    lsb = ttk.Scrollbar(log_frame, orient="vertical",
command=log_txt.yview)
    lsb.pack(side="right", fill="y")
    log_txt.configure(yscrollcommand=lsb.set)

    def set_log_text(s: str) -> None:
        log_txt.config(state="normal")
        log_txt.delete("1.0", tk.END)
        log_txt.insert(tk.END, s)
        log_txt.see(tk.END)
        log_txt.config(state="disabled")

    def on_log(_: str) -> None:
        set_log_text(master.log_text)

    master.set_log_callback(on_log)

    def clear_log() -> None:
        master.clear_log()
        set_log_text("")

    ttk.Button(mid_fr, text="Очистить журнал",
command=clear_log).pack(pady=4)

    # --- Правая панель: схема дверь / окно + «светодиоды» ---
    schema_lf = ttk.LabelFrame(right_fr, text="Зона (по выбранному
узлу)")
    schema_lf.pack(fill="both", expand=True, padx=2, pady=2)

    schema_hint = ttk.Label(
        schema_lf,
        text="Выберите строку в таблице слева или\nбудет показан
первый онлайн-узел.",
        wraplength=210,

```

```

        justify="center",
    )
    schema_hint.pack(pady=(4, 2))

    sch = tk.Canvas(schema_lf, width=220, height=280, bg="#f1f5f9",
highlightthickness=0)
    sch.pack(fill="both", expand=True, padx=4, pady=4)

    sch.create_text(50, 18, text="Дверь", font=("Segoe UI", 10,
"bold"), fill="#334155")
    sch.create_rectangle(15, 32, 75, 102, outline="#1e293b",
width=2, fill="#e2e8f0")
    sch.create_arc(15, 32, 75, 102, start=0, extent=90,
style=tk.CHORD, outline="#64748b", width=1)
    sch.create_oval(148, 52, 188, 92, outline="#334155", width=2,
fill="#9ca3af", tags="led_door")

    sch.create_text(50, 128, text="Окно", font=("Segoe UI", 10,
"bold"), fill="#334155")
    sch.create_rectangle(15, 142, 115, 202, outline="#1e293b",
width=2, fill="#dbeafe")
    sch.create_line(15, 172, 115, 172, fill="#64748b", width=1)
    sch.create_line(65, 142, 65, 202, fill="#64748b", width=1)
    sch.create_oval(148, 152, 188, 192, outline="#334155", width=2,
fill="#9ca3af", tags="led_window")

    sch.create_text(
        110,
        230,
        text="● зелёный – на охране\n● серый – снято\n● красный
(миг.) – сработка",
        font=("Segoe UI", 8),
        fill="#475569",
        justify="center",
    )

    src_lbl = ttk.Label(schema_lf, text="", font=("Segoe UI", 8),
foreground="#64748b")
    src_lbl.pack(pady=(0, 4))

    blink_phase = [True]

    def refresh_ui() -> None:
        err_lbl.config(text=master.last_error or "Ошибок нет",
fg=("red" if master.last_error else "black"))
        if master.demo_mode:
            poll_status.config(

```

```

        text="ДЕМО: идёт виртуальный опрос..."
        if master.poll_busy
        else "ДЕМО: ожидание цикла опроса"
    )
else:
    poll_status.config(text="Идёт опрос..." if
master.poll_busy else "Ожидание таймера опроса")
    # Сохранить выделение: при полном delete/insert строки
получают новые iid – выбор пропал.
    sel_keep = tree.selection()
    preserved_iid = sel_keep[0] if sel_keep else None

    for i in tree.get_children():
        tree.delete(i)
    rows = master.get_slaves_snapshot()
    for row in rows:
        iid = str(row.address)
        tree.insert(
            "",
            tk.END,
            iid=iid,
            values=(
                row.address,
                "онлайн" if row.online else "офлайн",
                row.status_text,
                row.last_seen,
            ),
        )
    if preserved_iid is not None and
tree.exists(preserved_iid):
        tree.selection_set(preserved_iid)
        tree.focus(preserved_iid)
        tree.see(preserved_iid)

    set_log_text(master.log_text)

    blink_phase[0] = not blink_phase[0]
    st, src = _pick_status_byte_from_tree(tree, rows)
    src_lbl.config(text=src)

    c_door = _led_fill_for_zone(st, protocol.STATUS_DOOR,
blink_phase[0])
    c_win = _led_fill_for_zone(st, protocol.STATUS_WINDOW,
blink_phase[0])
    sch.itemconfig("led_door", fill=c_door)
    sch.itemconfig("led_window", fill=c_win)

```

```

        root.after(200, refresh_ui)

    refresh_ports()
    if port_combo["values"] and port_var.get() not in
port_combo["values"]:
        port_var.set(port_combo["values"][0])

    if os.environ.get("PC_MASTER_DEMO", "").strip().lower() in
("1", "true", "yes", "on"):
        root.after(400, toggle_demo)

    def on_close() -> None:
        save_settings_to_db()
        if api_server:
            api_server.stop()
        if master.demo_mode:
            master.stop_demo()
        else:
            master.close_port()
        root.destroy()

    root.protocol("WM_DELETE_WINDOW", on_close)

    refresh_ui()
    root.mainloop()

if __name__ == "__main__":
    main()

```