Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ
КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

На правах рукописи
УДК 004.415.2

РОЛДУГИН ВЛАДИМИР ДМИТРИЕВИЧ

РАЗРАБОТКА МОДУЛЯ ОБРАБОТКИ ГРУППОВЫХ ЗАПРОСОВ MODBUS В СРЕДЕ OWEN LOGIC

Выпускная квалификационная работа бакалавра

Направление подготовки 09.03.01 Информатика и вычислительная техника

Выпускная квалификационная
работа защищена
«___»_____2026 г.
Оценка _____
Секретарь ГЭК _____

г. Москва
2026

Студент-дипломник _____ / Ролдугин В.Д. /

Руководитель работы _____ / Новиков Г.Г. /

Рецензент _____ / Шишов П.Н. /

Заведующий кафедрой №12 _____ / Иванов М.А. /

АННОТАЦИЯ

Выпускная квалификационная работа на тему: «Разработка модуля обработки групповых запросов Modbus в среде Owen Logic».

Пояснительная записка содержит 103 страницы, 4 части, 24 рисунков, 14 таблиц, 25 источников и 2 приложения.

Ключевые слова: MODBUS, OWEN LOGIC, RS-485, КС1, ПРОТОКОЛ, ГРУППИРОВКА ЗАПРОСОВ, RTU, ASCII, ПАРАМЕТРЫ УСТРОЙСТВ, ОПРОС, АЛГОРИТМЫ, ВАЛИДАЦИЯ.

Дипломный проект посвящен решению задачи оптимизации обмена данными по протоколу Modbus для контроллеров КС1 за счет разработки и реализации программного модуля группировки запросов.

Пояснительная записка состоит из четырех частей, описывающих проделанную работу.

В первой части выполнен обзор подходов к построению систем разработки и анализа инструментов, применяемых для конфигурирования логики контроллеров. Рассмотрены принципы организации программных решений на основе DDD, приведено сравнение сред разработки Codesys, TIA Portal, Trace Mode и Owen Logic. На основании сравнительного анализа обоснован выбор среды Owen Logic как основной платформы, предоставляющей доступ к параметрам устройств и позволяющей реализовать расширяемый модуль группировки.

Во второй части описаны проектные решения по построению модуля формирования групповых запросов. Представлены требования к обработке переменных и условия объединения параметров в общие запросы, рассмотрены ограничения протоколов Modbus RTU и ASCII, аппаратные ограничения контроллеров серии КС1 и требования среды Owen Logic. Изложена архитектура модуля, структура основных компонентов, принципы построения алгоритмов сортировки, проверки совместимости и формирования итоговых запросов. Показаны схемы обмена и взаимодействия между элементами системы.

В третьей части представлена реализация разработанных алгоритмов в среде Owen Logic. Приведены ключевые классы, отвечающие за построение групповых запросов, обработку протокольных ограничений и валидацию параметров RS-485 устройств. Рассмотрены примеры формирования реальных групповых запросов, демонстрирующие работу алгоритмов на практических конфигурациях.

В четвёртой части проведено тестирование разработанного модуля. Проверена корректность работы алгоритмов группировки, обработка последовательности адресов,

сортировка переменных, контроль ограничений протокола и поддержка различных типов данных. Описаны сценарии проверки конфликтов, тестирование валидаторов и разбор расширенных случаев формирования запросов. По результатам испытаний подтверждено соответствие модуля функциональным требованиям и корректность его работы в составе системы.

Разработанный модуль обеспечивает автоматизированное формирование групповых Modbus-запросов в среде Owen Logic, учитывая все ограничения протоколов и конфигурации устройств, что повышает производительность опроса и снижает нагрузку на каналы связи.

СОДЕРЖАНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ СОКРАЩЕНИЙ

OL (OWEN Logic) – среда разработки программируемых реле ОВЕН

FBD (Function Block Diagram) – язык программирования функциональными блоками

ST (Structured Text) – язык программирования структурированным текстом

DDD (Domain-Driven Design) – предметно-ориентированное проектирование

ПЛК – программируемый логический контроллер

SCADA – система диспетчерского управления и сбора данных

HMI – человеко–машинный интерфейс

КС1 – серия программируемых контроллеров ОВЕН

RS–485 – интерфейс связи

Modbus – протокол промышленной сети

DI (Dependency Inversion) – принцип инверсии зависимостей

Mock – имитационный объект, используемый при модульном тестировании

Moq – библиотека для создания имитационных объектов в среде .NET

NUnit – фреймворк для модульного тестирования приложений на платформе .NET

ВВЕДЕНИЕ

В условиях современного промышленного производства автоматизация технологических процессов становится ключевым фактором, обеспечивающим повышение производительности, снижение затрат и улучшение качества работы. Одним из важнейших инструментов для решения этих задач является специализированное программное обеспечение для разработки алгоритмов управления различными устройствами. Одним из таких продуктов является OL — среда разработки, предназначенная для автоматизации управления устройствами компании «ОВЕН».

OL представляет собой мощный инструмент для проектирования и реализации алгоритмов управления с помощью графического языка программирования FBD, который соответствует международному стандарту МЭК 61131-3. Этот стандарт описывает методы программирования устройств автоматизации, таких как программируемые логические контроллеры (PLC), что позволяет создавать надежные и эффективные решения для управления различными устройствами и процессами.

Среда OL поддерживает широкий спектр функциональных возможностей, включая разработку алгоритмов для программируемых реле серий ПР100, ПР200, ПР205, а также интеграцию с панелями оператора и другими устройствами. Для взаимодействия с внешними системами OL использует такие протоколы связи, как Modbus, что позволяет seamlessly интегрировать систему в более сложные автоматизированные комплексы и сети. Кроме того, OL включает в себя инструменты для симуляции работы алгоритмов, что даёт возможность протестировать разрабатываемые решения до их внедрения на реальных устройствах, существенно ускоряя процесс разработки и устранения ошибок.

Документация к OL, опубликованная на официальном сайте компании «ОВЕН» [1], подробно описывает её функциональные возможности, включая работу с программируемыми реле серий ПР100, ПР200 и ПР205, а также с панелями оператора. Эти материалы являются ценным ресурсом для пользователей, желающих эффективно использовать возможности OL и разрабатывать решения для автоматизации технологических процессов.

Современные системы автоматизации требуют не только высокой функциональности, но и удобства в использовании, что позволяет специалистам быстро и эффективно решать задачи. В этой связи OL представляет собой инструмент, ориентированный как на опытных разработчиков, так и на тех, кто только начинает работать в сфере автоматизации. Простота в освоении, гибкость и расширяемость системы делают OL удобным выбором для проектирования решений в самых различных отраслях, от промышленности до сельского хозяйства.

Вместе с тем, с учетом быстрого развития технологий и появления новых потребностей пользователей, OL требует постоянного совершенствования. Усовершенствование функционала данной среды и добавление новых возможностей позволяет расширить её область применения, улучшить качество решений и ускорить процесс разработки. В рамках дипломной работы будет предложен анализ существующих функциональных возможностей OL с целью выявления направлений для дальнейшего улучшения и расширения её функционала.

1 Обзорная часть

1.1 Подход к построению среды разработки

Архитектура программного обеспечения играет ключевую роль в разработке и функционировании информационных систем. В случае с OL используется подход DDD, который ориентирован на создание гибких и масштабируемых систем, где бизнес-логика и предметная область играют главную роль в проектировании.

1.1.1 Принципы DDD

DDD — это подход к проектированию программных систем, который ставит в центр внимания предметную область и бизнес-логику [2]. Внутри OL DDD используется для упрощения разработки и возможного расширения функционала в будущем. Основные принципы DDD, которые применяются в системе, включают:

– моделирование предметной области, при котором всё программное обеспечение строится с учётом требований и особенностей бизнес-логики автоматизации технологических процессов, а компоненты системы проектируются в тесном взаимодействии с экспертами для учёта всех аспектов работы с программируемыми реле и панелями компании «ОВЕН»;

– использование единого языка (Ubiquitous Language), где в проектировании применяется общая терминология, понятная как техническим специалистам, так и экспертам в области автоматизации, что обеспечивает согласованность в коде, документации и коммуникации для предотвращения недоразумений и упрощения взаимодействия;

– разделение на контексты (Bounded Contexts), при котором система делится на отдельные области, каждая из которых охватывает определённую часть бизнес-логики, что позволяет избегать излишней сложности, обеспечивать независимость компонентов, улучшать управление зависимостями и облегчать поддержку и расширение системы.

1.1.2 Структура и компоненты на основе DDD

Для более детального описания архитектуры на основе принципов DDD, можно выделить несколько важных аспектов, которые помогут лучше понять роль каждого слоя и компонента. [3] Основные слои, которые составляют эту архитектуру:

– слой домена (Domain Layer), содержащий бизнес-логику и модели, отвечающие за работу с программируемыми реле и панелями, в котором определяются основные сущности, такие как алгоритмы управления, параметры устройств и другие элементы;

– слой приложения (Application Layer), взаимодействующий с пользователем или другими приложениями и предоставляющий интерфейсы для работы с бизнес-логикой, в котором находятся сервисы, управляющие запросами, получающие данные и передающие их в слой домена;

– инфраструктурный слой (Infrastructure Layer), отвечающий за взаимодействие с внешними системами и технологическими компонентами, такими как базы данных, внешние устройства, системы мониторинга и т.д.;

– интерфейсы и внешние компоненты (Interfaces and External Components), обеспечивающие взаимодействие с пользователем, другими приложениями или внешними сервисами, например, через графический интерфейс или командную строку.

Схематическое представление архитектуры DDD, представлено на рисунке 1.1.

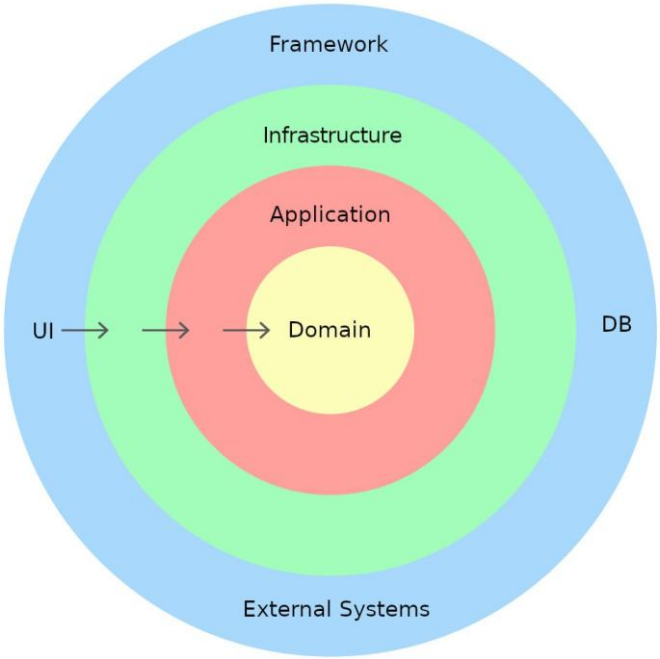

Рисунок 1.1 – Схематическое представление архитектуры DDD

1.1.3 Преимущества применения DDD

Использование DDD в OL позволяет достичь нескольких ключевых целей:

– модульность и независимость компонентов благодаря разделению системы на контексты и использованию чётких интерфейсов между ними, что позволяет легко изменять, заменять или масштабировать отдельные части;

– гибкость и масштабируемость архитектуры, основанной на DDD, что обеспечивает адаптацию системы под изменяющиеся требования, особенно важную для среды программирования, поддерживающей разнообразие задач управления технологическими процессами;

– упрощение взаимодействия между разработчиками и бизнес-экспертами через единый язык DDD, способствующее лучшему пониманию предметной области, уменьшению недопонимания и ускорению разработки и внедрения новых функций.

1.1.4 Применение DDD для расширяемости

С помощью DDD можно легко добавлять новый функционал и расширять программу. Это может быть особенно полезно в будущем, когда появятся новые типы устройств или технологии, которые потребуют интеграции в существующую систему. Благодаря ясному разделению на слои и контексты, расширение системы не будет нарушать её стабильность, а новые функции можно будет добавить с минимальными затратами.

1.2 Анализ сред разработки

Для выбора наиболее подходящей среды разработки для автоматизации процессов в промышленности важно учитывать несколько факторов: совместимость с оборудованием, поддерживаемые языки программирования, возможности интеграции с другими системами, а также требования по визуализации и отладке. В контексте автоматизации задач с использованием ПЛК и панелей управления, различные среды могут иметь свои преимущества в зависимости от потребностей проекта.

1.2.1 Codesys

Codesys является платформой для разработки программного обеспечения для программируемых логических контроллеров (ПЛК). Она поддерживает широкий спектр оборудования от различных производителей и предоставляет многоязыковую среду разработки, соответствующую стандарту IEC 61131-3 [4].

Ключевой функционал:

– поддержка языков программирования FBD, LD, ST, IL и SFC;

– мощные средства отладки, включая пошаговое выполнение и точечные остановки (breakpoints);

– расширенная работа с коммуникациями: поддержка различных протоколов (Modbus, CANopen, BACnet и т.д.);

– интеграция SCADA-систем для визуализации и мониторинга процессов;

– широкая библиотека готовых функциональных блоков;

– возможность управления системой с мобильных устройств.

Недостатки:

– высокая сложность освоения для начинающих пользователей;

– необходимость приобретения коммерческих лицензий для полного функционала и промышленного применения.

Codesys подходит для работы с многозадачными проектами и интеграции с разными производителями, но требует больше времени и ресурсов на настройку и изучение. Она также обладает высокой масштабируемостью и поддерживает интеграцию с современными технологиями, такими как IoT и Industry 4.0, что делает её подходящей для сложных и крупных проектов. Однако её сложность и потребность в обучении могут быть барьером для небольших компаний.

1.2.2 TIA Portal (Siemens)

TIA Portal — это комплексная среда разработки от Siemens, предназначенная для программирования и конфигурирования устройств из линейки SIMATIC, включая ПЛК, HMI и SCADA [5].

Ключевой функционал:

– единая среда разработки для всего оборудования Siemens;

– поддержка языков стандарта IEC 61131-3, включая FBD, LD и ST;

– визуализация процессов с помощью SCADA-систем;

– расширенные возможности анализа производительности оборудования;

– интеграция с системами управления энергопотреблением.

Недостатки:

– ограниченная совместимость с оборудованием других производителей;

– отсутствие встроенных симуляторов для быстрой отладки без подключения устройства.

TIA Portal является мощным инструментом для крупных предприятий, использующих оборудование Siemens, однако избыточна для более узких задач автоматизации.

1.2.3 Trace Mode

Trace Mode — это интегрированная система для автоматизации управления, мониторинга и сбора данных (SCADA/HMI), разработанная российской компанией AdAstrA [6].

Ключевой функционал:

– создание человеко-машинных интерфейсов (HMI) и SCADA-систем;

– поддержка большого количества протоколов связи, включая Modbus, OPC и BACnet;

– интеграция с внешними базами данных для хранения параметров и результатов работы систем;

– расширенные функции визуализации, включая 3D-графику;

– инструменты аналитики для оценки эффективности работы системы.

Недостатки:

– высокие системные требования;

– сложность настройки для пользователей, не знакомых с системами SCADA.

Trace Mode ориентирован на создание SCADA-систем, а не на программирование логики работы реле и панелей. Его возможности визуализации и аналитики подходят для сложных систем мониторинга и управления.

1.2.4 Owen Logic

OWEN Logic — это специализированная среда разработки, предназначенная для программирования реле и панелей управления компании «ОВЕН». Программное обеспечение поддерживает два языка программирования — FBD и ST, что позволяет эффективно решать задачи автоматизации для оборудования этой компании.

Ключевой функционал:

– поддержка языков программирования FBD и ST, соответствующая стандарту IEC 61131;

– графический интерфейс с визуальным редактором для создания и отладки программ;

– возможность расширения за счёт создания собственных функциональных блоков (макросов) и их повторного использования в других проектах;

– бесплатное использование, что делает программу привлекательной для небольших и средних предприятий;

– интеграция с системами управления и мониторинга через возможность подключения к внешним SCADA-системам для визуализации.

Недостатки:

– ограниченная совместимость, поскольку программное обеспечение предназначено исключительно для работы с оборудованием компании «ОВЕН»;

– ограниченные возможности визуализации из-за отсутствия встроенных инструментов для создания сложных SCADA-интерфейсов.

OL подходит для автоматизации задач с использованием оборудования компании «ОВЕН», но ограничена в плане совместимости с другими производителями и возможностями визуализации. Для небольших предприятий это решение является оптимальным благодаря бесплатному доступу и простоте использования, но для более сложных проектов с интеграцией внешних систем или создания сложных SCADA-интерфейсов могут потребоваться дополнительные инструменты визуализации.

1.2.5 Сравнительная таблица сред разработки

В таблице 1.1 приведено сравнение сред разработки для автоматизации. Указаны поддерживаемые языки программирования, совместимость с оборудованием, стоимость лицензии, поддержка протоколов связи и доступные средства отладки.

Таблица 1.1 – Сравнение сред разработки

Среда разработки	Поддержка языков	Совместимость с оборудованием	Стоимость лицензии	Поддержка протоколов связи	Отладочные средства
OWEN Logic	FBD, ST	ОВЕН (ПР100, ПР102, ПР200, ПР205, ИПП120)	Бесплатно	Modbus, CANopen	Пошаговая отладка
Codesys	FBD, LD, ST, IL, SFC	Широкая (множество производителей)	Платная	Modbus, CANopen, OPC, BACnet	Пошаговая отладка, точки остановки
TIA Portal	FBD, LD, ST	Siemens (SIMATIC)	Платная	Modbus, Profinet, OPC UA	Пошаговая отладка, точки остановки
Trace Mode	FBD, ST	Широкая (множество производителей)	Платная	Modbus, OPC, BACnet	Пошаговая отладка

2 РАСЧЕТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ

2.1 Переменные в среде Owen Logic как основа для группировки

Переменные в среде OL являются центральным элементом, служащим для хранения, передачи и обработки данных внутри проекта, а также для взаимодействия с внешними устройствами. Они представляют собой основной строительный блок для создания схем управления, формирования Modbus-запросов и программирования интерфейсов визуализации. Понимание типов, свойств и, что наиболее важно, атрибутов переменных, связанных с Modbus-обменом, является фундаментальным для разработки алгоритма их группировки.

2.1.1 Классификация и типы данных переменных

Создание и управление переменными выполняется через Таблицу переменных, которая включает следующие категории:

– стандартные переменные, используемые в алгоритмах управления и визуализации;

– сервисные переменные, обеспечивающие внутренние функции прибора и недоступные для изменения пользователем;

– сетевые переменные, применяемые для обмена данными между устройствами через интерфейсы связи (например, RS-485, Ethernet, Modbus).

Каждая переменная имеет следующие основные параметры:

– имя переменной в качестве уникального идентификатора для отображения на схеме проекта;

– тип переменной, определяющий формат данных (булевский, целочисленный, с плавающей запятой);

– энергонезависимость: при включении параметра значение сохраняется в энергонезависимой памяти (ПЗУ) устройства;

– значение по умолчанию, используемое при инициализации или при отсутствии связи с устройством;

– использование в проекте, отображающее, привязана ли переменная к блокам схемы;

– комментарий в качестве описания назначения переменной для удобства разработки.

На рисунке 2.1 представлена таблица переменных среды OL, отображающая структуру и категории переменных проекта.

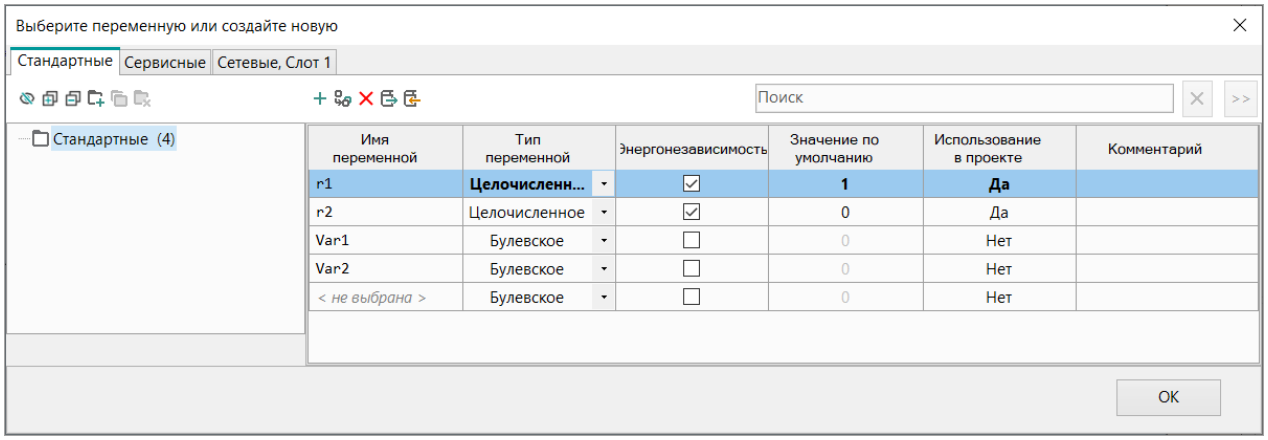

Рисунок 2.1 – Таблица переменных в среде OL

Типы переменных:

– булевский (Bool), принимающий значения 0 (False) или 1 (True) и используемый для логических условий и дискретных сигналов;

– целочисленный (Int/UInt), хранящий значения целых чисел и применяемый для счетчиков, адресов и параметров, не требующих дробной части;

– с плавающей запятой (Float/Real), хранящий вещественные значения и используемый для аналоговых сигналов, коэффициентов и вычислений.

Примеры отображения данных типов переменных в среде OL представлены на рисунках 2.2–2.4.

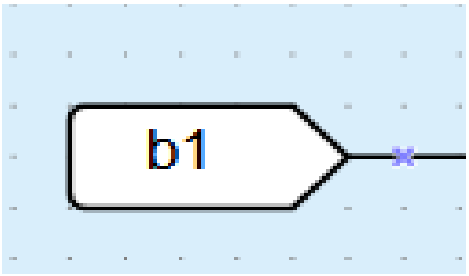

Рисунок 2.2 – Булевские переменные в OL

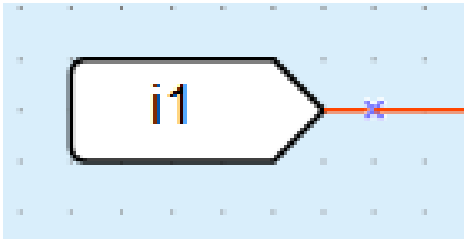

Рисунок 2.3 – Целочисленные переменные в OL

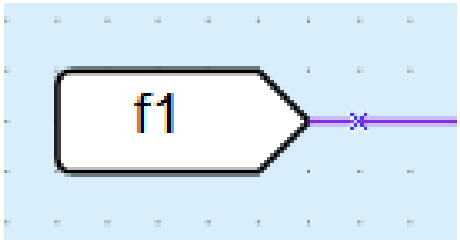

Рисунок 2.4 – Переменные с плавающей запятой в OL

На схеме OL линии, соединяющие переменные разных типов, имеют различный цвет:

– черный для булевских;

– красный для целочисленных;

– фиолетовый для вещественных.

Переменные могут быть связаны с регистрами внешних Modbus-устройств. В этом случае в таблице указывается тип регистра (Holding, Input, Coil, Discrete) и его адрес, что позволяет модулю обмена данными формировать соответствующие запросы чтения и записи.

2.2.2 Сетевые переменные и их роль в Modbus-обмене

Для организации обмена данными по протоколу Modbus используются сетевые переменные. В таблице переменных для каждого физического интерфейса связи (например, RS-485) создается отдельная вкладка. Поведение и структура этих вкладок различаются в зависимости от режима работы устройства (Master или Slave).

Режим Master

В режиме Master (ведущий) контроллер инициирует запросы к подчиненным устройствам. Таблица переменных в этом режиме содержит отдельные вкладки для каждого опрашиваемого Slave-устройства (Рисунок 2.5).

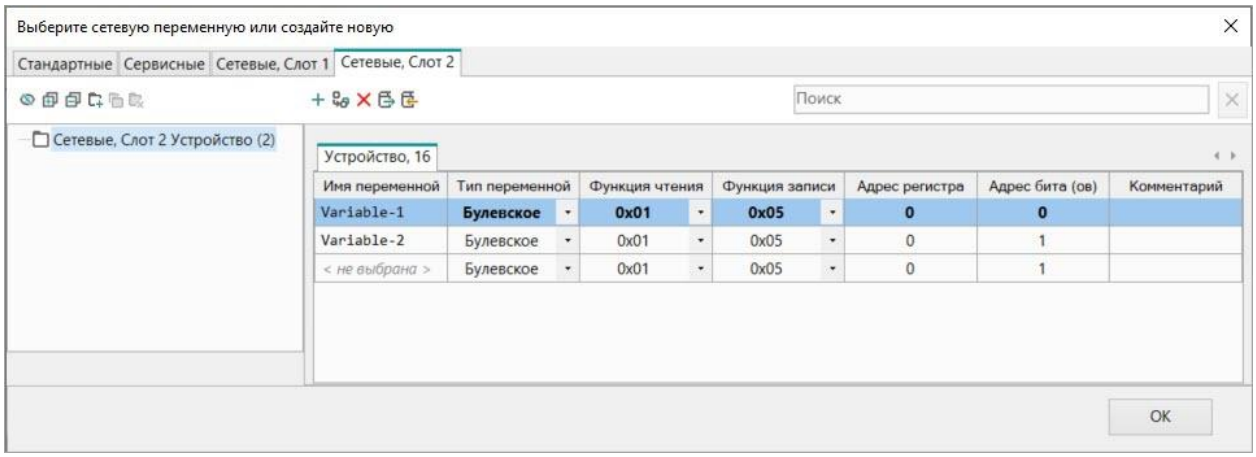

Рисунок 2.5 – Вкладка сетевых переменных в режиме Master

Для каждой сетевой переменной в этом режиме пользователь задает параметры Modbus-связи, которые являются исходными данными для алгоритма группировки:

– тип регистра, определяющий область памяти Slave-устройства (Coil, Discrete Input, Holding Register, Input Register);

– адрес регистра, определяющий позицию данных в адресном пространстве устройства;

– функция чтения/записи, определяющая операцию, выполняемую с регистром (0x03, 0x04, 0x10 и другие);

– направление обмена, указывающее, осуществляется ли чтение данных из устройства или запись данных в устройство.

Эти параметры используются модулем группировки при формировании оптимальных запросов к Slave-устройствам.

Режим Slave

В режиме Slave (подчиненный) контроллер предоставляет свои данные по запросу внешнего Master-устройства. Таблица переменных в этом режиме отображает переменные, значения которых могут быть считаны или записаны извне (Рисунок 2.6). Такие переменные связаны с внутренними параметрами проекта и автоматически обновляются при изменении значений на схеме. Это позволяет другим устройствам считывать актуальные данные в реальном времени. Разрабатываемый модуль группировки не обрабатывает переменные в режиме Slave, так как он активен только на стороне Master.

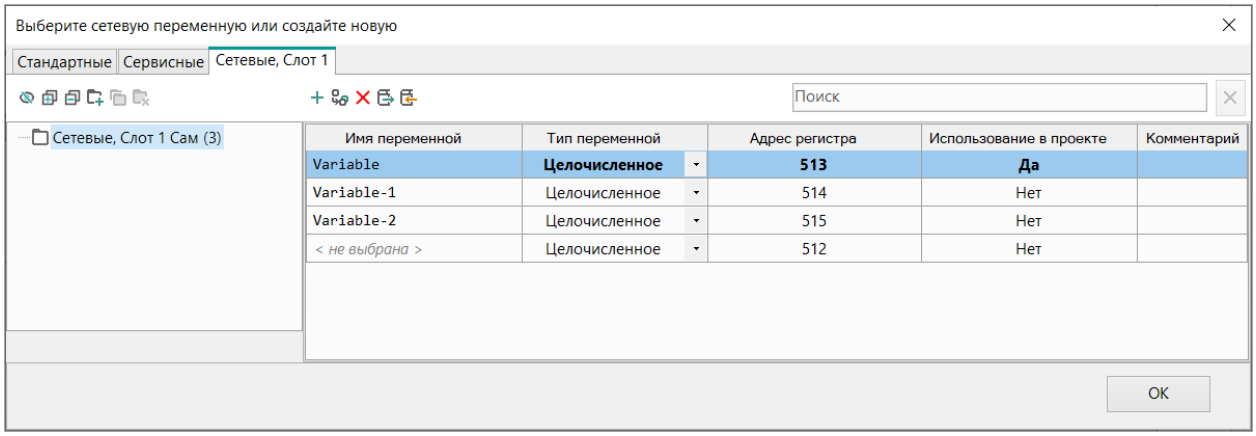

Рисунок 2.6 – Вкладка сетевых переменных в режиме Slave

2.2 Учет ограничений при группировке запросов

При формировании групповых Modbus-запросов модуль должен учитывать комплекс ограничений, накладываемых спецификацией протокола, аппаратными возможностями контроллера и средой разработки OL. Соблюдение этих ограничений гарантирует корректность обмена данными и оптимальную производительность системы.

2.2.1 Ограничение протокола Modbus

При формировании групповых запросов Modbus необходимо учитывать ряд ограничений, определяемых спецификацией протокола и особенностями аппаратной платформы КС1. Основным фактором, ограничивающим количество регистров в одном запросе, является размер кадра (буфера) передачи данных, который для устройств серии КС1 составляет 256 байт. Кроме того, пределы на количество регистров зависят от типа протокола (RTU или ASCII) и номера функции чтения/записи.

Максимальное количество регистров в запросе для различных комбинаций параметров приведено в таблице 2.1.

Таблица 2.1 – Ограничения на максимальное количество регистров в запросах

№ п/п	Тип протокола	Номер функции	макс. кол-во регистров	Примечание
1	RTU	0x03, 0x04	125	Ограничение на ввод параметра кол-во регистров в UI - 125.
2	RTU	0x10	123	Ограничение на ввод параметра кол-во регистров в UI - 125. Таким образом, если пользователь ввел кол-во регистров 125, то необходимо будет сформировать 2 посылки (структуры MB_Request): на 123 регистра и на 2 регистра.
3	RTU	0x01, 0x02	125	Максимальное количество бит/койлов в групповом запросе согласно спецификации Modbus 2000. Но у нас принято общее ограничение на ввод параметра кол-во регистров в UI - 125.
4	RTU	0x0F	125	Максимальное количество бит/койлов в групповом запросе согласно спецификации Modbus 1968. Но у нас принято общее ограничение на ввод параметра кол-во регистров в UI - 125.
5	ASCII	0x03, 0x04	61	Ограничение на ввод параметра кол-во регистров в UI - 125. Таким образом, если пользователь ввел кол-во регистров 125, то необходимо будет сформировать 3 посылки (структуры MB_Request): две посылки на 61 регистр и одну на 3 регистра.
6	ASCII	0x10	59	Ограничение на ввод параметра кол-во регистров в UI - 125. Таким образом, если пользователь ввел кол-во регистров 125, то необходимо будет сформировать 3 посылки (структуры MB_Request): две по 59 регистров и одну на 7 регистров.
7	ASCII	0x01, 0x02	125	Максимальное количество бит/койлов в групповом запросе согласно спецификации Modbus 984. Но у нас принято общее ограничение на ввод параметра кол-во регистров в UI - 125.
8	ASCII	0x0F	125	Максимальное количество бит/койлов в групповом запросе согласно спецификации Modbus 944. Но у нас принято общее ограничение на ввод параметра кол-во регистров в UI - 125.

2.2.2 Аппаратные и программные ограничения

2.2.2.1 Ограничения контроллера КС1

Контроллер КС1 имеет следующие аппаратные ограничения, влияющие на формирование групповых запросов:

– размер буфера передачи данных составляет 256 байт на кадр;

– максимальная скорость обмена по интерфейсу RS-485 составляет 115200 бит/с;

– ограничение по времени обработки прерываний не превышает 12 мс между запросами;

– объём оперативной памяти для хранения очереди запросов составляет 1024 записи.

2.2.2.2 Период опроса и правила его использования

Прибор все запросы ставит в очередь. Если очередь короткая, то прибор выполнит все циклы запроса-ответа и остановится в ожидании пока не подойдет к концу заданный период. Если очередь длинная и не укладывается в заданный период, то прибор будет опрашивать все необходимые параметры с максимально возможным периодом, но этот период будет больше заданного в настройках.

На рисунке 2.7 представлена временная диаграмма опроса устройства по протоколу Modbus, отражающая последовательность формирования и обработки запросов в зависимости от скорости передачи и объёма данных.

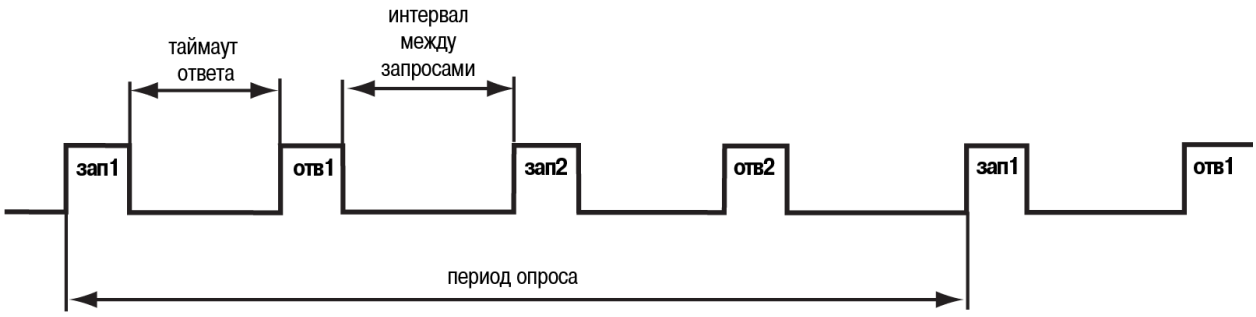

Рисунок 2.7 – Временная диаграмма опроса устройства по протоколу **Modbus**

Для максимальной скорости 115200 бит/с опрос двух переменных типа FLOAT в «идеальных» условиях (короткая линия связи, отсутствие помех) у одного подчиненного устройства следует установить:

– время ответа от начала запроса – 2,64 мс;

– следующий запрос начинается не раньше, чем через 12 мс;

– общее время на запрос 1 регистра FLOAT – 3,4 мс.

В вышеперечисленных условиях, прибор сможет отправлять 83 запроса в секунду. Данное значение справедливо и для других приборов с подобными временными

характеристиками. В процессе разработки алгоритма, когда логика усложняется, то увеличивается время цикла и количество запросов за секунду будет снижаться.

Значение периода опроса зависит от алгоритма, как часто и какие параметры надо опрашивать. Рекомендуется выставлять период опроса равным 1 с. При таком периоде устройство способно опросить до 50 переменных.

2.2.3 Ограничения среды Owen Logic

Среда OL накладывает ограничения на структуру проектов, которые необходимо учитывать при настройке сетей Modbus. Основные из них:

– максимальное количество устройств Modbus в проекте — 32;

– ограничение на количество сетевых переменных — до 1024 на проект;

– поддерживаются только протоколы Modbus RTU и ASCII (TCP не поддерживается);

– ограничение по размеру буфера обмена контроллера — 256 байт на кадр.

При генерации групповых запросов модуль учитывает эти ограничения: диапазоны обрезаются до допустимого объёма, а устройства сверх лимита игнорируются с предупреждением в журнале.

Устройства в сети опрашиваются согласно сформированной очереди, при этом порядок опроса определяется адресами устройств: сначала опрашиваются устройства с наименьшими адресами, постепенно переходя к устройствам с большими адресами. На рисунке 2.8 показан пример: первым опрашивается устройство с адресом 8, последним — устройство с адресом 32. Система позволяет задавать индивидуальный период опроса для каждого подчинённого устройства, что обеспечивает гибкость конфигурации под конкретные задачи проекта.

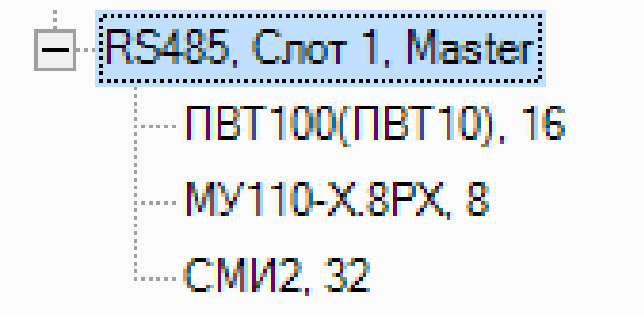

Рисунок 2.8 – Последовательность опроса устройств по Modbus

Система позволяет задавать индивидуальный период опроса для каждого подчинённого устройства, что обеспечивает гибкость конфигурации под конкретные задачи проекта.

Дополнительно, для корректного формирования запросов необходимо учитывать расчёт адреса и бита при считывании булевых переменных в режиме Master, что позволяет правильно объединять данные в групповые Modbus-запросы и поддерживать согласованность информации в сети.

2.3 Функциональные требования к модулю

Разрабатываемый модуль предназначен для формирования и управления групповыми Modbus-запросами (чтение/запись) для устройств платформы КС1 в среде OL. Модуль должен:

– объединять параметры в групповые запросы для повышения производительности взаимодействия с устройствами;

– учитывать ограничения протоколов Modbus RTU и ASCII (максимальный размер запроса, ограничения по функциям);

– поддерживать режимы: групповой (объединение переменных) и одиночный (без объединения);

– предоставлять API для интеграции в существующую архитектуру OL и UI-компоненты для настройки и отображения сформированных запросов;

– обеспечивать валидацию переменных и визуальную выдачу ошибок в UI;

– поддерживать гибкую политику разбиения больших групп на несколько физических сообщений (MB_Request) в соответствии с ограничениями протокола и пользовательскими настройками (макс. количество регистров).

2.3.1 Основные задачи и цели разработки

Для обоснования функциональных требований к модулю группировки запросов целесообразно рассмотреть сценарии использования со стороны конечного пользователя среды OL.

В таблице 2.2 приведены ключевые User Story, отражающие ожидания разработчика проектов при работе с модулем.

Таблица 2.2 – User Story для модуля группировки запросов

Как создатель программы в OL хочу	Для того, чтобы	Примечание
Опрашивать Slave-прибор групповыми запросами	Сократить время опроса устройств	Много опрашиваемых регистров (переменных) приводят к тому, что время опроса увеличивается
Отключить групповые запросы для Slave-прибора	Опрашивать прибор, не поддерживающий групповые запросы	Например, если карта регистров имеет разрывы или устройство не поддерживает чтение блоками
Что бы Logic сам формировал запросы из переменных	Не тратить время на ручное формирование запроса	Ручное формирование запросов занимает много времени; автоматизация повышает эффективность
Ограничить количество элементов (регистров) в опросе	Что бы получать информацию от приборов с небольшим количеством памяти	Например, ПР110 не способен отвечать на длинные запросы, поэтому требуется ограничение длины

2.3.2 Требования к группировке переменных

Перед формированием групповых запросов необходимо учитывать особенности протокола Modbus, в частности — структуру его областей данных и допустимые функции обмена.

2.3.2.1 Функции и области данных

При запросе Master обращается к одной из областей памяти Slave с помощью функции. Область памяти характеризуется типом хранимых значений (биты или регистры) и типом доступа (чтение или чтение/запись). В таблице 2.3 приведены области данных протокола Modbus с указанием их обозначений, типов данных и типов доступа.

Таблица 2.3 – Области данных протокола Modbus

Область данных	Обозначение	Тип данных	Тип доступа
Coils (Регистры флагов)	0x	Булевый	Чтение/запись
Discrete Inputs (Дискретные входы)	1x	Булевый	Только чтение
Input Registers (Регистры ввода)	3x	Целочисленный	Только чтение
Holding Registers (Регистры хранения)	4x	Целочисленный	Чтение/запись

Каждая область памяти состоит из определенного (зависящего от конкретного устройства) количества ячеек. Каждая ячейка имеет уникальный адрес. Для конфигурируемых устройств производитель предоставляет карту регистров, в которой содержится информация о соответствии параметров устройства и их адресов. Для программируемых устройств пользователь формирует такую карту самостоятельно с помощью среды программирования. Существуют устройства, в которых сочетаются оба рассмотренных случая – у их карты регистров есть фиксированная часть и часть, которую пользователь может дополнить в соответствии со своей задачей.

В некоторых устройствах области памяти наложены друг на друга (например, 0x и 4x) – т. е. пользователь сможет обращаться разными функциями к одним и тем же регистрам.

2.3.2.2 Основные функции протокола Modbus

Функция определяет операцию (чтение/запись) и область памяти, с которой эта операция будет произведена. В таблице 2.4 приведены основные функции протокола Modbus с указанием кода, имени функции и выполняемой команды.

Таблица 2.4 – Основные функции Modbus

Код функции	Имя функции	Выполняемая команда
1 (0x01)	Read Coil Status	Чтение значений из нескольких регистров флагов
2 (0x02)	Read Discrete Inputs	Чтение значений из нескольких дискретных входов
3 (0x03)	Read Holding Registers	Чтение значений из нескольких регистров хранения
4 (0x04)	Read Input Registers	Чтение значений из нескольких регистров ввода
5 (0x05)	Force Single Coil	Запись значения в один регистр флага
6 (0x06)	Preset Single Register	Запись значения в один регистр хранения
15 (0x0F)	Force Multiple Coils	Запись значений в несколько регистров флагов
16 (0x10)	Preset Multiple Registers	Запись значений в несколько регистров хранения

2.3.2.3 Опрос Slave-устройства

Опрос устройства может выполняться двумя способами:

– одиночный способ, при котором каждая переменная считывается отдельной командой;

– групповой способ, при котором несколько переменных считываются одной командой при условии, что их адреса расположены последовательно без разрывов.

Групповой опрос позволяет снизить трафик в сети и ускорить обмен, но его применение ограничено типом данных и функцией Modbus.

Особенности формирования групповых запросов:

1. допустимые функции Modbus для групповых запросов. Групповые запросы могут формироваться только для функций чтения 0x01, 0x02, 0x03, 0x04 и функций записи 0x0F, 0x10. Для функций записи одиночных битов или регистров (0x05, 0x06) группировка невозможна, так как данные функции работают только с одним адресом за операцию.

2. особенности опроса при режиме «Запись по изменению». Если в группе переменных задано условие «запись по изменению», то при изменении значения хотя бы одной переменной из этой группы модуль формирует и отправляет единый запрос для всей группы. Такой подход обеспечивает консистентность данных в Slave-устройстве, но может приводить к увеличению частоты передачи сообщений при высокой изменчивости входных данных.

3. совместимость типов Real и Int32. Эти типы данных могут объединяться в один групповой запрос, так как оба занимают по два Modbus-регистра (32 бита). Это позволяет повысить эффективность передачи данных, сохраняя корректное выравнивание адресов. Однако смешивание типов с разным количеством регистров (например, Int16 и Real) не допускается, поскольку нарушает непрерывность адресного пространства.

2.3.2.4 Условия формирования группового запроса

Для формирования группового запроса должны быть соблюдены все условия, перечисленные в таблице 2.5. Окно настроек опрашиваемого устройства представлено на рисунке.2.9.

Таблица 2.5 – Условия и примеры группировки переменных в Modbus-запросы

№	Условие формирования группового запроса	Пример валидного условия	Пример невалидного условия
1	Группировка только для переменных одного устройства	устройство адрес 16, var1, регистр 0; var2, регистр 2	разные адреса устройств (16 и 17)
2	Тип данных переменных должен быть одинаковым	var1:real (0), var2:real (2), var3:real (4)	var1:real, var2:int16
3	Адреса регистров/битов должны идти подряд без разрывов	var1:рег.0, var2:рег.2, var3:рег.4	var1:рег.0, var2:рег.3
4	Функции чтения/записи должны совпадать	все var1,var2: чтение 0x03	var1: чтение 0x03, var2: чтение 0x04
5	Условия опроса (период, команды) должны совпадать	var1,var2: период 100мс	var1: 100мс, var2: 200мс
6	Для опроса по команде «Запуск чтение» – одна командная переменная	var1,var2: чтение по команде var_b1	var1: var_b1, var2: var_b2
7	Для опроса по команде «Запуск записи» – одна командная переменная	var1,var2: запись по команде var_b1	var1: var_b1, var2: var_b2
8	Переменные статуса должны быть одинаковы или не заданы	var1,var2: статус var_i1	var1: статус var_i1, var2: не выбран

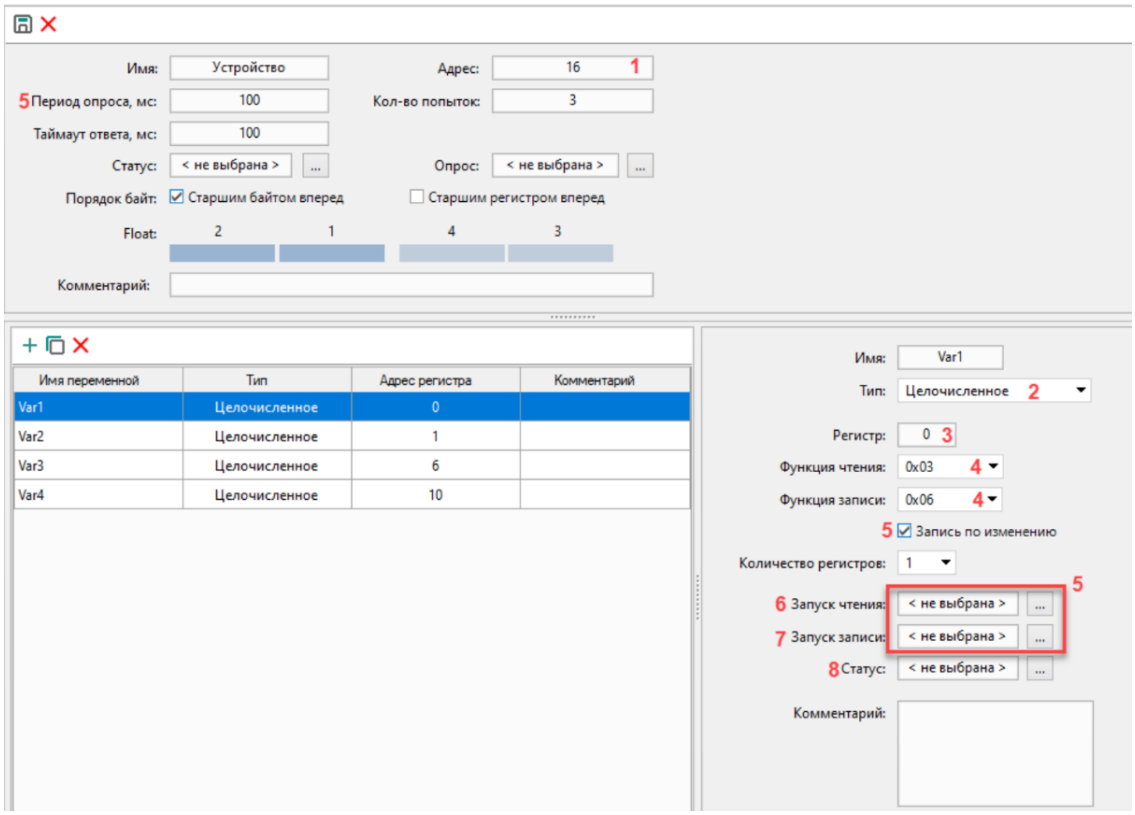

Рисунок 2.9 – Окно настроек опрашиваемого устройства

2.3.3 Требования к пользовательскому интерфейсу

Для реализации функционала групповых Modbus-запросов в среде OL необходимо расширить окно настроек опрашиваемого устройства платформы КС1 за счёт добавления новых параметров (рис. 2.10).

В окно должны быть добавлены следующие элементы:

1. параметр «Протокол» типа выпадающего списка с возможными значениями «Modbus RTU» и «Modbus ASCII», где при выборе протокола выполняется динамическое обновление допустимого диапазона параметра «Количество регистров в запросе» в соответствии с ограничениями выбранного протокола, а значением по умолчанию является «Modbus RTU» (рис. 2.11);

2. параметр «Группировать запросы» типа выпадающего списка со значениями «Да» и «Нет», где значением по умолчанию является «Нет», при выборе значения «Да» активируется возможность задания дополнительных параметров группировки (рис. 2.12), а если значение параметра установлено в «Нет», параметр «Количество регистров в запросе» становится недоступным для редактирования (рис. 2.13);

3. параметр «Количество регистров в запросе» с диапазоном допустимых значений от 1 до 125 для протокола Modbus RTU и от 1 до 61 для протокола Modbus ASCII, где значением по умолчанию является 16, а при вводе некорректного значения отображается пиктограмма с восклицательным знаком и тултип с сообщением о допустимом диапазоне значений, например, сообщение о валидном диапазоне количества регистров в групповом запросе для протокола RTU (рис. 2.14).

Параметр «Количество регистров в запросе» напрямую связан с ограничениями протокола Modbus, приведёнными в таблице 2.2 (см. раздел 2.1.2). Изменение значения данного параметра влияет на разбиение групп переменных на отдельные физические запросы (MB_Request).

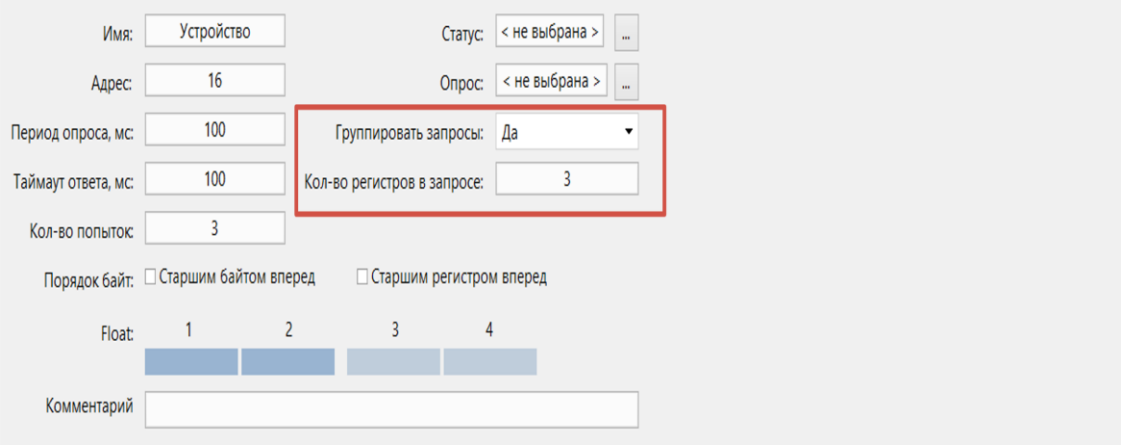

Рисунок 2.10 – Окно настроек опрашиваемого устройства с добавленными параметрами

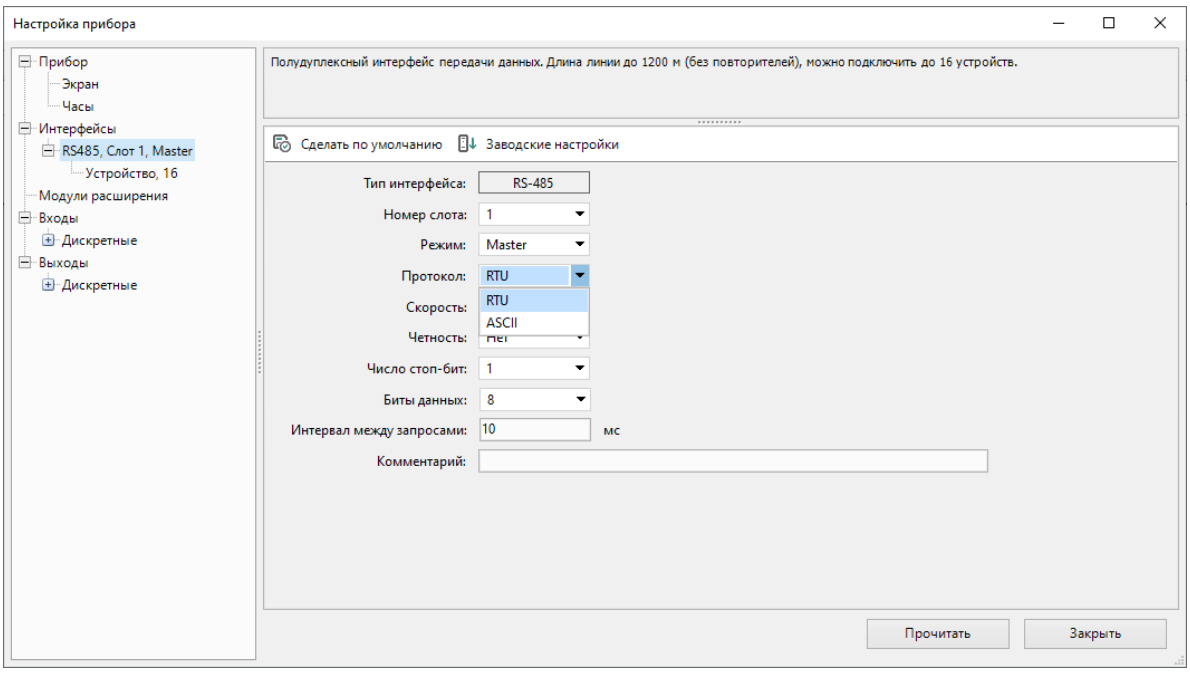

Рисунок 2.11 – Выбор протокола Modbus RTU/ASCII

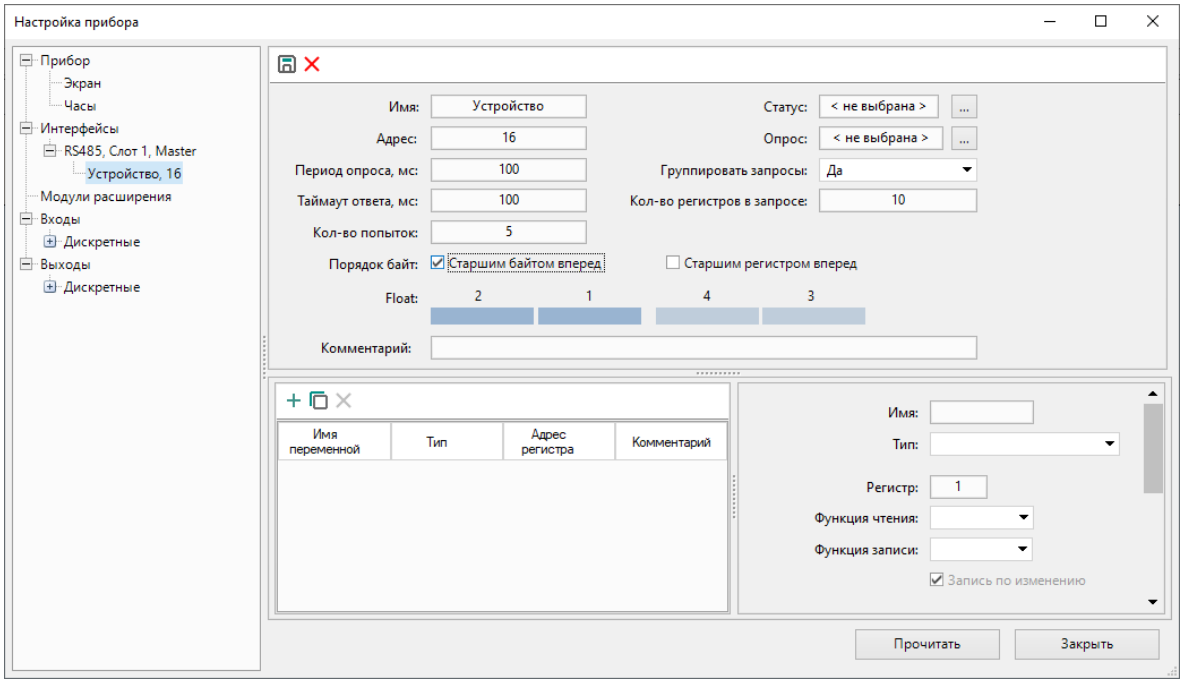

Рисунок 2.12– Параметр «Группировать запросы»

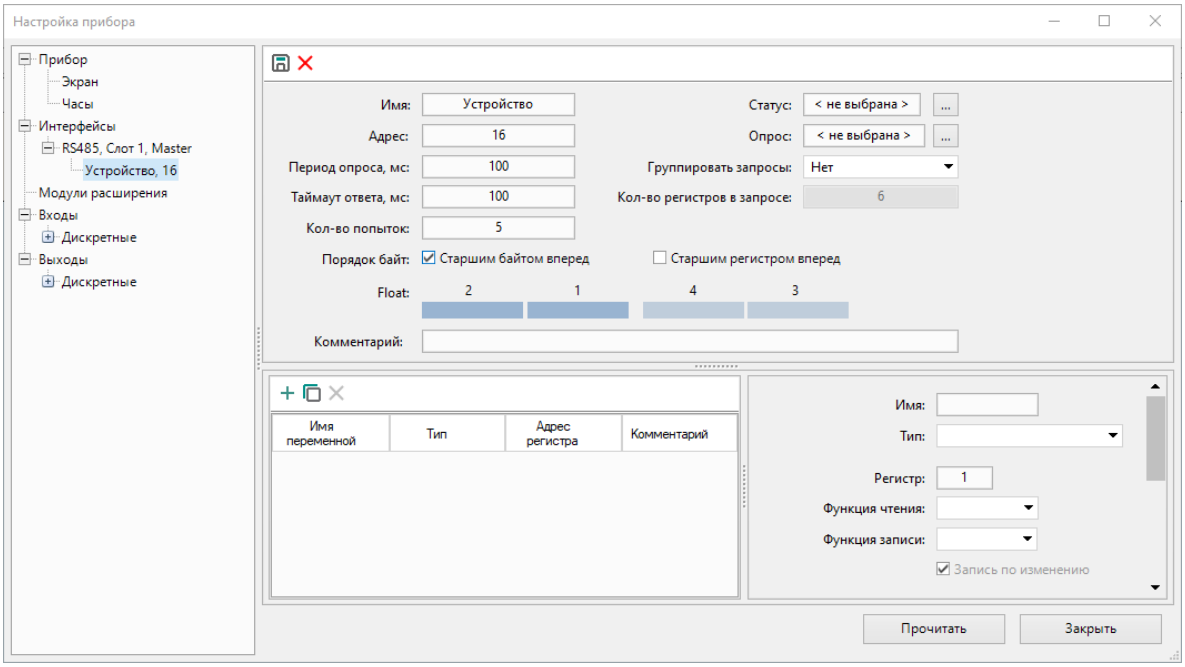

Рисунок 2.13 – Неактивное состояние параметра «Количество регистров в запросе» при значении «Группировать запросы = Нет»

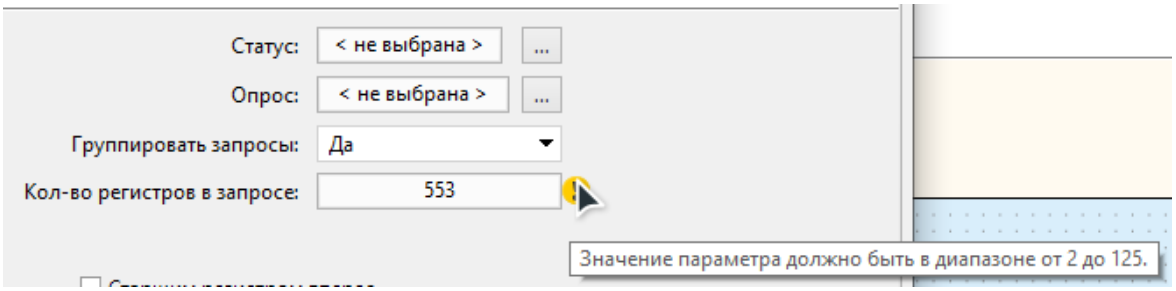

Рисунок 2.14 – Сообщение о недопустимом диапазоне значений для протокола **Modbus**

2.4 Архитектура модуля обработки групповых запросов

Модуль реализует логику построения, объединения и валидации групповых запросов Modbus на основе переменных OL. Архитектура построена по принципам DDD и разделена на изолированные компоненты, каждый из которых отвечает за конкретную часть бизнес-логики.

2.4.1 Диаграмма компонентов

На рисунке 2.15 представлена UML-диаграмма классов, отражающая основные связи и зависимости внутри модуля.

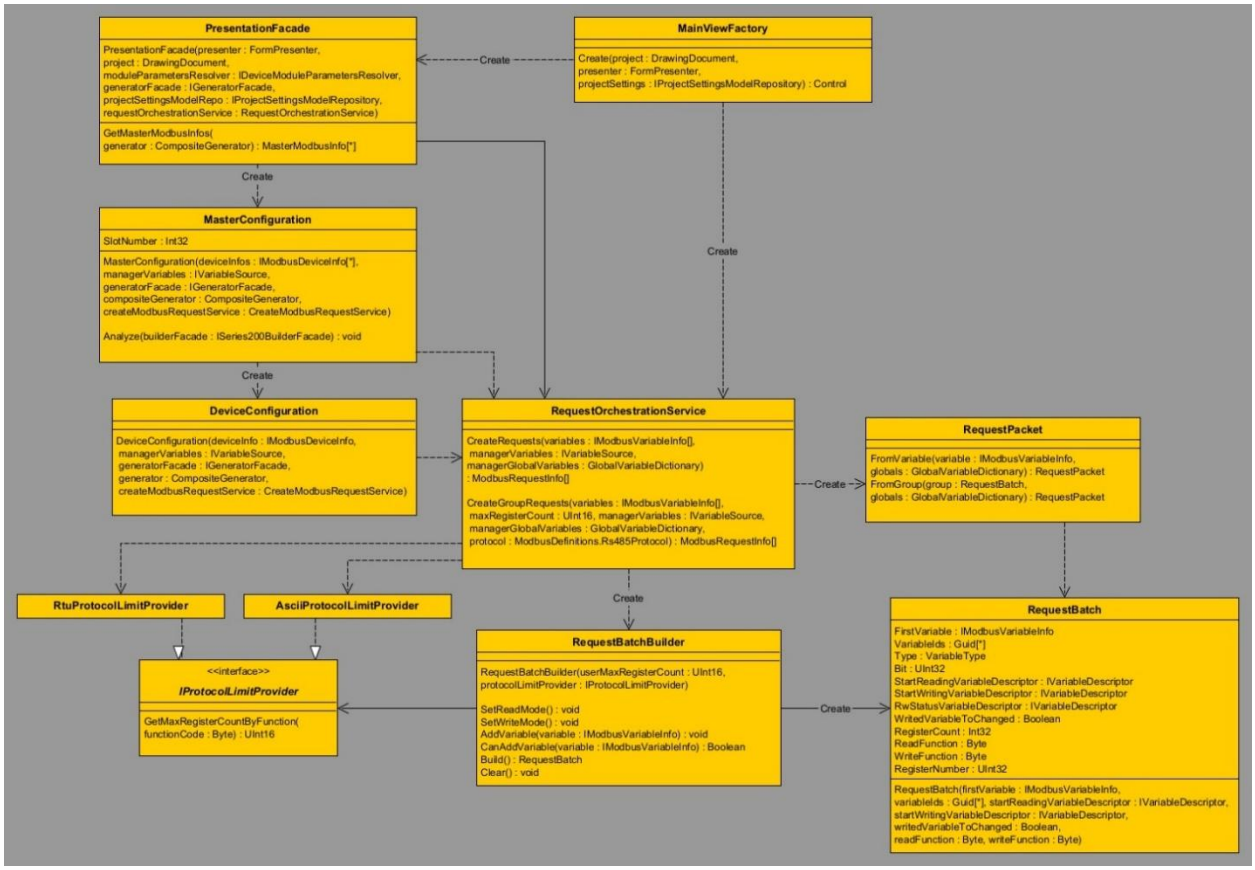

Рисунок 2.15 – UML-диаграмма классов

2.4.2 Основные компоненты и их ответственность

Для понимания структуры разрабатываемого модуля в таблице 2.6 приведены основные компоненты и их функциональное назначение.

Таблица 2.6 – Основные компоненты и их ответственность

Компонент	Назначение
RequestBatch	Представляет агрегированную группу переменных, объединённых в один Modbus-запрос. Хранит параметры, описывающие первую переменную, список идентификаторов, тип данных, функции чтения/записи.
RequestBatchBuilder	Реализует пошаговую сборку группы Modbus-запроса. Отвечает за проверку совместимости переменных, соблюдение ограничений протокола и создание экземпляров RequestBatch.
RequestOrchestrationService	Управляет формированием всех запросов. Делит переменные на группы чтения и записи, сортирует, объединяет совместимые группы и создаёт итоговые структуры RequestPacket.
IProtocolLimitProvider	Определяет интерфейс для получения предельного числа регистров в зависимости от используемого протокола (RTU или ASCII).
RtuProtocolLimitProvider	Реализует ограничения количества регистров для протокола Modbus RTU в зависимости от кода функции.
AsciiProtocolLimitProvider	Реализует ограничения количества регистров для протокола Modbus ASCII в зависимости от кода функции.
RequestPacket	Итоговая структура запроса, готовая к сериализации в бинарный формат Modbus.
DeviceConfiguration	Инкапсулирует данные одного Modbus-устройства. Хранит список сформированных запросов, переменные разрешения и статуса, обеспечивает их анализ и сборку бинарных данных для программы.
MasterConfiguration	Представляет Master-устройство с набором Modbus-устройств. Формирует и хранит коллекцию DeviceConfiguration, обеспечивает анализ и сборку данных для передачи.
PresentationFacade	Фасад для работы с Presenter и проектом. Предоставляет интерфейсы для получения информации о модулях расширения и Master Modbus-устройствах. Скрывает детали внутренней логики создания запросов.
MainViewFactory	Фабрика основного контейнера визуального представления. Создаёт основное окно, настраивает сервисы, Presenter фасады и компоненты визуализации, объединяет их в конечный Control для UI.

2.5 Алгоритмы работы модуля

В данном разделе рассмотрены основные алгоритмы, реализованные в модуле обработки групповых запросов. Каждый из алгоритмов отвечает за отдельный этап формирования и оптимизации Modbus-обмена: группировку переменных, проверку их совместимости, сортировку по адресам регистров, а также слияние совместимых групп. Совместная работа этих алгоритмов обеспечивает минимизацию количества запросов, сокращение времени опроса и повышение эффективности обмена данными между контроллером и устройствами по протоколу Modbus.

2.5.1 Алгоритм группировки переменных

Для оптимизации обмена данными по протоколу Modbus в программируемом реле серии **ПР200** реализован алгоритм группировки переменных в составные запросы. Целью данного алгоритма является уменьшение количества передаваемых запросов за счёт объединения переменных с одинаковыми характеристиками в один Modbus-запрос, при сохранении корректности данных и соблюдении ограничений протокола.

2.5.1.1 Общие принципы работы

Каждая переменная, участвующая в обмене, описывается структурой IModbusVariableInfo, содержащей адрес регистра, количество регистров, тип данных, функции чтения/записи, а также дескрипторы чтения и записи.

Алгоритм последовательно обрабатывает список переменных, отобранных по признаку использования (IsUsed), и выполняет следующие действия:

1. определение режима группировки начинается с перевода алгоритма в один из режимов: режим чтения (SetReadMode) для формирования групп запросов чтения или режим записи (SetWriteMode) для формирования групп запросов записи, после чего в зависимости от выбранного режима задаются различные правила совместимости переменных;

2. инициализация состояния группы выполняется при добавлении первой переменной, когда создаётся новая группа, в которой запоминаются функция Modbus (readFunction или writeFunction), стартовый дескриптор (чтения или записи), идентификатор первой переменной, а также текущее количество регистров и начальный адрес;

3. проверка возможности добавления переменной осуществляется для каждой следующей переменной вызовом метода CanAddVariable(), который определяет, может ли она быть добавлена в текущую группу, при этом переменная может быть объединена с

предыдущими, если выполняются условия: не превышено ограничение по количеству регистров (учтены пользовательский лимит и ограничения протокола RTU/ASCII), адреса регистров следуют последовательно без пропусков, совпадают функции Modbus (чтения или записи), типы данных переменных идентичны или являются допустимыми комбинациями (например, Float и Long), а также совпадают дескрипторы состояния и признак «запись по изменению»; если хотя бы одно условие нарушается, текущая группа считается завершённой, создаётся новый запрос и начинается формирование новой группы;

4. формирование запроса происходит, когда текущая группа готова, вызовом метода Build(), создающего объект RequestBatch, который содержит первую переменную группы, список идентификаторов всех переменных в группе, функции Modbus для чтения и записи, диапазон регистров (начальный адрес и количество), а также признаки дескрипторов и тип данных, после чего состояние строителя очищается методом Clear() для подготовки к созданию следующей группы;

5. завершение и объединение совместимых групп выполняется после построения всех групп запросов чтения и записи дополнительной обработкой сервисом RequestOrchestrationService, где если для одной и той же переменной были сформированы отдельные группы чтения и записи, они объединяются в один запрос при условии совпадения диапазона регистров и идентификаторов переменных.

2.5.1.2 Пошаговое описание алгоритма

Ниже представлена обобщённая последовательность действий алгоритма:

1. Получить список используемых переменных.
2. Разделить их на группы по функциям чтения и записи.
3. Для каждой группы:
    1. Отсортировать переменные по адресу регистра.
    2. Инициализировать новую группу с первой переменной.
    3. Для каждой следующей переменной:
    − если совместима, добавить в группу;
    − иначе сохранить группу и начать новую.
    4. После окончания обработки — зафиксировать последнюю группу.
4. После завершения группировки объединить совместимые группы чтения и записи.
5. Преобразовать сформированные группы в объекты RequestPacket для последующей отправки.

2.5.2 Алгоритм проверки совместимости переменных

Алгоритм проверки совместимости переменных используется при формировании групп Modbus-запросов. Он определяет, можно ли объединить несколько переменных в одну группу без нарушения ограничений протокола и структуры данных. Проверка выполняется при добавлении каждой новой переменной к текущей группе.

2.5.2.1 Общие принципы работы

Проверка совместимости реализована в классе RequestBatchBuilder (метод CanAddVariable()). Алгоритм оценивает ряд параметров, которые должны совпадать для корректного объединения переменных в общий запрос.

Основные критерии совместимости:

1. количество регистров, где общее число регистров в группе не должно превышать лимиты протокола (RtuProtocolLimitProvider, AsciiProtocolLimitProvider) и пользовательские настройки;

2. последовательность адресов, требующая, чтобы регистры переменных шли подряд, без пропусков;

3. дескрипторы состояния, которые должны быть одинаковыми для всех переменных в группе (RwStatusVariableDescriptor);

4. тип данных, допускающий одинаковые типы или совместимые пары (например, Float и Long);

5. функция Modbus и режим, при которых должны совпадать функции чтения/записи и, при необходимости, признак «запись по изменению» (WritedVariableToChanged).

Если хотя бы одно условие нарушено, переменная не добавляется в текущую группу - создаётся новая.

2.5.2.2 Пошаговое описание алгоритма

1. определить номер и количество регистров текущей переменной;

2. сравнить параметры с первой переменной активной группы;

3. проверить:

− не превышен лимит по количеству регистров;

− адреса следуют подряд;

− типы данных и дескрипторы совпадают;

− функции Modbus идентичны.

4. если все условия выполняются — переменная добавляется в группу;

5. иначе текущая группа фиксируется, создаётся новая.

2.5.3 Алгоритм сортировки и упорядочивания переменных

Перед формированием групп переменные проходят сортировку. Это необходимо для правильного порядка обработки и исключения конфликтов при объединении.

2.5.3.1 Общие принципы работы

Сортировка выполняется в методах GetReadVariables() и GetWriteVariables() класса RequestOrchestrationService.

Цель - обеспечить логическую и адресную последовательность переменных при формировании запросов.

Основные критерии сортировки:

1. функция Modbus, по которой сначала группируются переменные с одинаковыми кодами функций (например, 0x03, 0x04, 0x10);

2. адрес регистра, используемый для дальнейшей сортировки по возрастанию адресов;

3. битовое смещение, учитываемое для битовых переменных через значение поля Bit, после чего каждой переменной присваивается порядковый номер (Order).

2.5.3.2 Пошаговое описание алгоритма

1. исключить из списка неиспользуемые переменные (IsUsed == false);
2. разделить их на две категории — для чтения и записи;
3. для каждой категории выполнить сортировку по:

– функции Modbus;

– адресу регистра;

– битовому смещению (если есть).

4. назначить каждой переменной номер Order;
5. использовать отсортированные списки для дальнейшей группировки.

2.5.4 Алгоритм слияния совместных групп

После группировки чтения и записи может оказаться, что одна и та же переменная входит в обе категории. Чтобы избежать дублирования запросов, выполняется слияние совместимых групп.

2.5.4.1 Общие принципы работы

Слияние реализовано в методе MergeCompatibleGroups() класса RequestOrchestrationService. Алгоритм объединяет группы, если они относятся к одной переменной и имеют совпадающий диапазон регистров.

Объединённая группа получает общие параметры — диапазон, функции чтения и записи, а также оба дескриптора. Это позволяет сократить количество запросов и повысить эффективность обмена.

2.5.4.2 Пошаговое описание алгоритма

1. получить все сформированные группы чтения и записи;
2. сгруппировать их по идентификатору переменной и количеству регистров;
3. для каждой пары при наличии двух групп (чтение и запись) необходимо их объединить, а если группа одна, её следует оставить без изменений;
4. вернуть список объединённых групп для формирования объектов RequestPacket.

3 Экспериментальная часть

В этой части работы рассматривается разработка и исследование механизмов формирования **Modbus**-запросов для одиночных и групповых переменных. Приводятся ключевые классы и структуры, обеспечивающие корректную адресацию, типизацию данных, сортировку и группировку переменных, а также соблюдение ограничений протоколов RTU и ASCII. Для наглядности приводится практический пример формирования групповых запросов, демонстрирующий работу алгоритмов на конкретных переменных.

3.1 Реализация логики группировки

Данный раздел описывает центральные компоненты разрабатываемого модуля - алгоритмы и классы, отвечающие за объединение одиночных запросов к переменным в групповые Modbus-запросы. Архитектура построена вокруг нескольких ключевых компонентов, которые взаимодействуют по следующей схеме:

– подготовка данных, при которой сервис RequestOrchestrationService получает список всех переменных, фильтрует используемые и разделяет их на списки для операций чтения и записи;

– сортировка и упорядочивание, в ходе которых переменные внутри каждого списка сортируются по типу функции Modbus, адресу регистра и битовому смещению, при этом вспомогательные структуры VariableSorter и OrderedVariable обеспечивают корректный порядок обработки;

– формирование групп, выполняемое классом RequestBatchBuilder, который последовательно обходит отсортированный список переменных, проверяя для каждой возможность быть добавленной в текущую группу с учетом ограничений протокола и пользовательских настроек, а при нарушении условий совместимости фиксирует текущую группу и начинает формирование новой;

– слияние совместимых групп, где на финальном этапе группы, содержащие одни и те же переменные для операций чтения и записи, объединяются в единый запрос, что дополнительно оптимизирует трафик;

– создание итоговых структур, в результате которого готовые группы переменных превращаются в финальный объект RequestPacket, содержащий всю информацию для создания бинарного Modbus-запроса, понятного среде OL.

3.1.1 Класс RequestPacket

Класс RequestPacket инкапсулирует информацию для формирования одного или нескольких Modbus-запросов. Используется как для одиночных переменных, так и для групп переменных. Класс управляет типом данных, адресацией, кодами функций Modbus и связью с глобальными переменными.

Листинг 3.1 – Класс RequestPacket

```
public sealed class RequestPacket
    {
        private byte _varTypeAndBit;
        private byte _readFunction;
        private byte _writeFunction;
        private byte _timer;
        private int _data;
        private ushort _quantity;
        private ushort _startRegister;
        private RequestPacket(IModbusVariableInfo variableInfo,
            GlobalVariableDictionary globalVariables, IReadOnlyList<Guid>
variableIds)
        {
            _variableInfo = variableInfo;
            _globalVariables = globalVariables; _
             variableIds = variableIds;
        }

        public static RequestPacket FromVariable(IModbusVariableInfo
variable,
            GlobalVariableDictionary globals)
        {
            return new RequestPacket(variable, globals, new []{
variable.VariableId })
                .Initialize(variable.Type, variable.GetRegisterCount(),
variable.GetReadFunction(),
                    variable.GetWriteFunction(),
variable.WritedVariableToChanged,
                    variable.GetRegisterNumber(), variable.Bit);
        }

        public static RequestPacket FromGroup(RequestBatch group,
            GlobalVariableDictionary globals)
        {
            return new RequestPacket(group.FirstVariable, globals,
group.VariableIds)
                .Initialize(group.Type, group.RegisterCount,
group.ReadFunction,
                    group.WriteFunction, group.WritedVariableToChanged,
                    group.RegisterNumber, group.Bit);
        }

        private RequestPacket Initialize(VariableType type, int
registerCount,
            byte readFunc, byte writeFunc, bool writeOnChange,
            uint regNumber, uint bit)
        {
            if (type == VariableType.Long && registerCount == 1)
                _varTypeAndBit = (byte)DataVariableType.Word;
            else if (type == VariableType.Float || (type == VariableType.Long
&& registerCount == 2))
                _varTypeAndBit = (byte)DataVariableType.Dword;
```

```
            else if (type == VariableType.Boolean)
                _varTypeAndBit = (byte)((uint)DataVariableType.Bit | bit <<
4);
            Else
      throw new
NotSupportedException(string.Format(Properties.Resources.TypeNotDefined,
type));

            _readFunction = readFunc;
            _writeFunction = writeFunc;
            _timer = (byte)((readFunc > 0 ? 0x01 : 0) |
                            (writeFunc > 0 ?
(writeOnChange ? 0x04 : 0) : 0));  // запись по изменению, если есть функция
записи!
            _startRegister = (ushort)regNumber;
            _quantity = (ushort)_variableIds.Count;

            return this;
        }


        internal void Analyze(ISeries200BuilderFacade builderFacade)
        {
            var readingDescriptor =
_variableInfo.StartReadingVariableDescriptor;

            if (readingDescriptor != null)
                _evReadVariable =
builderFacade.GetVariable(readingDescriptor.UniqueId);
            var writingDescriptor =
_variableInfo.StartWritingVariableDescriptor;

            if (writingDescriptor != null)
                _evWriteVariable =
builderFacade.GetVariable(writingDescriptor.UniqueId);
            var statusDescriptor = _variableInfo.RwStatusVariableDescriptor;

            if (statusDescriptor != null)
                _statusVariable =
builderFacade.GetVariable(statusDescriptor.UniqueId);

            _dataVariable =
builderFacade.GetVariable(_variableInfo.VariableId);

            if (_dataVariable == null)
                throw new
InvalidOperationException(string.Format(Properties.Resources.VariableNotDefin
ed, _variableInfo));

            if (_variableIds.Count == 1)
                return;

            // NOTE: Резервируем память под все переменные (если она ещё не
зарезервирована), участвующие в запросе в правильном порядке
            foreach (var variableId in _variableIds)
                builderFacade.GetVariable(variableId);
        }

        internal void Build(Stream stream, ref int address, ProgramCode
program)
        {
            if (_dataVariable != null)
                _data =
_globalVariables.GetAddressInGlobalBuffer(_dataVariable);
```

```
            if (_evReadVariable != null)
                _evRead =
_globalVariables.GetAddressInGlobalBuffer(_evReadVariable);

            if (_evWriteVariable != null)
                _evWrite =
_globalVariables.GetAddressInGlobalBuffer(_evWriteVariable);

            if (_statusVariable != null)
                _status =
_globalVariables.GetAddressInGlobalBuffer(_statusVariable);
 }
}
```

В таблице 3.1 приведено назначение основных свойств класса RequestPacket, используемых для хранения информации о переменной или группе переменных и подготовки данных для формирования Modbus-запроса.

Таблица 3.1 – Назначение основных свойств класса RequestPacket

Свойство	Тип	Назначение
_varTypeAndBit	byte	Тип переменной Modbus (бит, слово, двойное слово) и смещение бита
_readFunction	byte	Код функции чтения Modbus
_writeFunction	byte	Код функции записи Modbus
_timer	byte	Флаги: чтение/запись по изменению
_evRead	int	Указатель на событие чтения в глобальном буфере
_evWrite	int	Указатель на событие записи в глобальном буфере
_status	int	Указатель на статус переменной в глобальном буфере
_data	int	Указатель на данные переменной в глобальном буфере
_quantity	ushort	Количество переменных в запросе
_startRegister	ushort	Начальный регистр Modbus
_startRegister	ushort	Начальный регистр Modbus
_globalVariables	GlobalVariable Dictionary	Словарь глобальных переменных
_variableInfo	IModbusVariableInfo	Ссылка на объект переменной Modbus
_variableIds	IReadOnlyList	Список идентификаторов переменных, участвующих в запросе
_evReadVariable, _evWriteVariable, _statusVariable	IRealVariable	Связанные объекты для записи/чтения данных и событий

Принцип работы:

1. инициализация, при которой методы FromVariable и FromGroup создают экземпляр класса, связывая переменные с порядком, типом, функциями и регистрами;

2. определение типа переменной: в методе Initialize определяется тип Modbus (Bit, Word, Dword) в зависимости от типа данных переменной (Boolean, Long, Float) и количества регистров;

3. установка функций чтения/записи, где флаг _timer указывает, участвует ли переменная в операции чтения или записи, включая запись по изменению;

4. анализ переменных, выполняемый методом Analyze, который получает доступ к глобальным переменным через ISeries200BuilderFacade и проверяет корректность ссылок;

5. формирование бинарного запроса методом Build, который создаёт указатели на память и готовит данные для компиляции в бинарный Modbus-запрос.

Роль в системе:

Хранит всю информацию о переменной или группе переменных и подготавливает её для формирования корректного Modbus-запроса в бинарном виде.

3.1.2 Класс RequestOrchestrationService

Класс RequestOrchestrationService отвечает за формирование итогового набора Modbus-запросов, объединяя переменные по признакам использования, режимам (чтение/запись), а также совместимости адресов и функций.

Листинг 3.2 – Класс RequestOrchestrationService

```
public sealed class RequestOrchestrationService
  {
      internal IEnumerable<RequestPacket>
CreateRequests(IEnumerable<IModbusVariableInfo> variables,
           IVariableSource managerVariables, GlobalVariableDictionary
managerGlobalVariables) => variables
           .Where(item => managerVariables.IsUsed(item.VariableId))
           .Select(item => RequestPacket.FromVariable(item,
managerGlobalVariables));

      internal IEnumerable<RequestPacket>
CreateGroupRequests(IEnumerable<IModbusVariableInfo> variables,
          ushort maxRegisterCount, IVariableSource managerVariables,
          GlobalVariableDictionary managerGlobalVariables,
ModbusDefinitions.Rs485Protocol protocol)
      {
          var orderGenerator = new VariableSorter();

          var filteredSortedVariables = GetVariablesWithUsage();
          var readVars = GetReadVariables();
          var writeVars = GetWriteVariables();

          if (readVars.Count == 0 && writeVars.Count == 0)
              return Enumerable.Empty<RequestPacket>();

          var builder = new RequestBatchBuilder(
              maxRegisterCount,
              protocol == ModbusDefinitions.Rs485Protocol.Rtu
                  ? new RtuProtocolLimitProvider()
                  : new AsciiProtocolLimitProvider());
          var requestGroups = new List<RequestBatch>();

          BuildReadGroups();
          BuildWriteGroups();

          return MergeCompatibleGroups()
              .Select(group => RequestPacket.FromGroup(group,
managerGlobalVariables));

          List<IModbusVariableInfo> GetVariablesWithUsage()
              => variables.Where(v =>
managerVariables.IsUsed(v.VariableId)).ToList();

          List<OrderedVariable> GetReadVariables()
          {
              return filteredSortedVariables
                  .Where(v => v.GetReadFunction() != 0)
                  .OrderBy(v => v.GetReadFunction())
                  .ThenBy(v => v.GetRegisterNumber())
                  .ThenBy(v => v.Bit)
                  .Select(item => new OrderedVariable(item,
orderGenerator.GetNextOrderNumber()))
                  .ToList();
          }
```

```
        List<OrderedVariable> GetWriteVariables()
        {
            return filteredSortedVariables
                .Where(v => v.GetWriteFunction() != 0 &&
                            (v.WritedVariableToChanged ||
v.StartWritingVariableDescriptor != null))
                .OrderBy(v => v.GetWriteFunction())
                .ThenBy(v => v.GetRegisterNumber())
                .ThenBy(v => v.Bit)
                .Select(item => new OrderedVariable(item,
GetOrderNumber(item)))
                .ToList();

            uint GetOrderNumber(IModbusVariableInfo variableInfo)
            {
                var readVariable = readVars
                    .FirstOrDefault(item => item.VariableInfo.VariableId ==
variableInfo.VariableId);
                return readVariable.VariableInfo is null
                    ? orderGenerator.GetNextOrderNumber()
                    : readVariable.Order;
            }
        }

        void BuildReadGroups()
        {
            if (readVars.Count == 0) return;

            builder.SetReadMode();
            BuildGroupFromVariables(readVars);
        }

        void BuildWriteGroups()
        {
            if (writeVars.Count == 0) return;

            builder.SetWriteMode();
            BuildGroupFromVariables(writeVars);
        }

        void BuildGroupFromVariables(List<OrderedVariable> vars)
        {
            OrderedVariable previousVariable = default;
            foreach (var variable in vars)
            {
                if (!builder.CanAddVariable(variable.VariableInfo) ||
IsOrderConflict(previousVariable, variable))
                {
                    requestGroups.Add(builder.Build());
                    builder.Clear();
                }

                builder.AddVariable(variable.VariableInfo);
                previousVariable = variable;}

            requestGroups.Add(builder.Build());
            builder.Clear();}

        bool IsOrderConflict(OrderedVariable previousVariable,
OrderedVariable variable)
        {
            if (previousVariable.VariableInfo == default)
                return false;
```

```
            return variable.Order != previousVariable.Order + 1;
        }

        IEnumerable<RequestBatch> MergeCompatibleGroups()
        {
            if (requestGroups.Count == 0) return
Enumerable.Empty<RequestBatch>();

            return requestGroups
                .GroupBy(g => (g.FirstVariable.VariableId,
g.VariableIds.Count))
                .SelectMany(group => ProcessGroup(group.ToList()));

            IEnumerable<RequestBatch> ProcessGroup(List<RequestBatch>
items)
            {
                return items.Count == 2
                    ? new[] { CreateMergedGroup(
                            items.First(g => g.ReadFunction != 0),
                            items.First(g => g.WriteFunction != 0))}
                    : items;}

            RequestBatch CreateMergedGroup(RequestBatch read, RequestBatch
write) =>
                new (firstVariable: read.FirstVariable, variableIds:
read.VariableIds,
                    startReadingVariableDescriptor:
read.StartReadingVariableDescriptor,
                    startWritingVariableDescriptor:
write.StartWritingVariableDescriptor,
                    writedVariableToChanged: write.WritedVariableToChanged,
                    readFunction: read.ReadFunction, writeFunction:
write.WriteFunction);}}}
```

Обоснование выбора решения:

– инкапсуляция сложной логики группировки, при которой класс скрывает внутренние проверки и сортировки, предоставляя простой интерфейс;

– разделение ответственности: RequestBatchBuilder отвечает за детали формирования группы, а RequestOrchestrationService управляет общим процессом;

– оптимизация производительности за счёт однократного выполнения фильтрации и сортировки переменных, что снижает нагрузку при большом количестве тегов;

– расширяемость, позволяющая легко адаптировать реализацию под разные протоколы (RTU/ASCII), задаваемые через интерфейс IProtocolLimitProvider.

В таблице 3.2 приведены основные функции класса RequestOrchestrationService и логика их работы.

Таблица 3.2 – Принцип работы класса RequestOrchestrationService

Функция	Назначение	Логика работы
CreateRequests	Создание одиночных запросов	Формирует запросы для каждой переменной без группировки
CreateGroupRequests	Формирование групповых запросов	Строит оптимальные группы на основе фильтрации, сортировки и проверки ограничений
BuildReadGroups	Построение групп чтения	Настраивает билдер на режим чтения и объединяет переменные по адресам
BuildWriteGroups	Построение групп записи	Настраивает билдер на режим записи и объединяет переменные с возможностью изменения
MergeCompatibleGroups	Слияние совместимых групп	Объединяет пары чтение/запись в один запрос
BuildGroupFromVariables	Универсальный метод построения группы	Добавляет переменные последовательно, создавая новую группу при несоответствии условий
IsOrderConflict	Проверка нарушения порядка	Определяет конфликт последовательности между переменными
VariableSorter	Генератор порядковых номеров	Присваивает каждой переменной уникальный номер для отслеживания порядка
OrderedVariable	Структура переменной с порядком	Объединяет данные переменной и её индекс в одну сущность

Принцип работы:

1. фильтрация переменных для отбора только тех, которые реально используются в программе;

2. сортировка, осуществляющая упорядочивание по функциям, номерам регистров и битам;

3. разделение по режимам с целью формирования отдельных списков переменных для чтения и записи;

4. группировка, реализующая последовательное объединение переменных с помощью RequestBatchBuilder;

5. слияние совместимых групп, объединяющее пар запросов (чтение/запись) с одинаковыми идентификаторами переменных;

6. создание итоговых RequestPacket для подготовки структур, передаваемых в компилятор OL.

3.1.3 Вспомогательные структуры класса RequestOrchestrationService

Для корректной работы алгоритма группировки в классе RequestOrchestrationService используются две вспомогательные структуры:

– variableSorter выполняет роль генератора порядковых номеров для переменных;

– orderedVariable является структурой, которая связывает переменную с её порядковым номером.

Обе структуры инкапсулируют служебную логику и не предназначены для использования вне сервиса.

3.1.3.1 Класс VariableSorter

Класс VariableSorter генерирует последовательные уникальные порядковые номера для переменных, обеспечивая корректное сопоставление между переменными чтения и записи при группировке.

Листинг 3.3– Класс VariableSorter

```
private class VariableSorter
    {
        private uint nextNumber = 0;

        public uint GetNextOrderNumber() => ++nextNumber;
    }
```

Назначение основных свойств:

– nextNumber служит внутренним счётчиком порядковых номеров;

– метод GetNextOrderNumber() предназначен для возврата следующего значения счётчика, начиная с 1.

Принцип работы:

Каждый вызов метода GetNextOrderNumber() увеличивает счётчик на единицу и возвращает новое значение, обеспечивая уникальность номера в рамках текущего процесса построения групп.

Роль в системе:

Используется при сортировке и объединении переменных, гарантируя согласованность порядка между объектами чтения и записи.

3.1.3.2 Класс OrderedVariable

Класс OrderedVariable хранит переменную Modbus вместе с её порядковым номером, который присваивается генератором. Используется для отслеживания порядка добавления переменных при формировании групповых запросов.

Листинг 3.4 – Класс OrderedVariable

```
private readonly struct OrderedVariable
    {
        public IModbusVariableInfo VariableInfo { get; }

        public uint Order { get; }

        public OrderedVariable(IModbusVariableInfo variableInfo, uint order)
        {
            VariableInfo = variableInfo;
            Order = order;
        }
    }
```

Назначение основных свойств:

– variableInfo содержит ссылку на объект переменной Modbus;

– order хранит порядковый номер, присвоенный при добавлении переменной.

Принцип работы:

При формировании группы каждая переменная связывается с порядковым номером, что позволяет контролировать последовательность и определять конфликты порядка при объединении.

Роль в системе:

Используется в методах проверки совместимости переменных, обеспечивая корректное слияние групп Modbus-запросов.

3.1.4 Класс RequestBatchBuilder

Класс RequestBatchBuilder формирует группы Modbus-запросов, объединяя переменные в оптимальные группы с учетом ограничений протокола.

Листинг 3.5 – Класс RequestBatchBuilder

```
internal sealed class RequestBatchBuilder
    {
        private readonly IProtocolLimitProvider _registerLimitByProtocol;
        private readonly ushort _userMaxRegisterCount;
        private readonly List<Guid> _variableIds;

        private IModbusVariableInfo _firstVariable;
        private int _currentRegisterCount;
        private uint? _previousRegisterOrBit;

        private Func<IModbusVariableInfo, bool> _isNewGroupNeeded;
        private Func<RequestBatch> _buildGroup;
        private Func<IModbusVariableInfo, byte> _getCurrentFunction;

        private bool HasVariables => _variableIds.Count > 0;

        internal RequestBatchBuilder(ushort userMaxRegisterCount,
            IProtocolLimitProvider registerLimitByProtocol)
        {
            _userMaxRegisterCount = userMaxRegisterCount;
            _registerLimitByProtocol = registerLimitByProtocol;
            _variableIds = new List<Guid>();
        }

        internal void SetReadMode()
        {
            _getCurrentFunction = v => v.GetReadFunction();

            _isNewGroupNeeded = variable =>
                HasReadDescriptorChanged(variable) ||
                HasReadFunctionChanged(variable);

            _buildGroup = () => CreateGroup(
                startReadingDescriptor:
_firstVariable.StartReadingVariableDescriptor,
                startWritingDescriptor: null, writedVariableToChanged: false,
                readFunction: _firstVariable.GetReadFunction(),
writeFunction: 0
            );

            return;

            bool HasReadDescriptorChanged(IModbusVariableInfo v) =>

HasDescriptorChanged(_firstVariable.StartReadingVariableDescriptor,
v.StartReadingVariableDescriptor);

            bool HasReadFunctionChanged(IModbusVariableInfo v)
                => _firstVariable.GetReadFunction() != 0 &&
                   _firstVariable.GetReadFunction() != v.GetReadFunction();
        }

        internal void SetWriteMode()
        {
            _getCurrentFunction = v => v.GetWriteFunction();
```

```
            _isNewGroupNeeded = variable =>
                HasWriteDescriptorChanged(variable) ||
                HasWriteFunctionChanged(variable) ||
                IsSingleWriteFunction(variable) ||
                IsWriteFlagChanged(variable);

            _buildGroup = () => CreateGroup(
                startReadingDescriptor: null,
                startWritingDescriptor:
_firstVariable.StartWritingVariableDescriptor,
                writedVariableToChanged:
_firstVariable.WritedVariableToChanged,
                readFunction: 0, writeFunction:
_firstVariable.GetWriteFunction()
            );

            return;

            bool HasWriteDescriptorChanged(IModbusVariableInfo v) =>

HasDescriptorChanged(_firstVariable.StartWritingVariableDescriptor,
v.StartWritingVariableDescriptor);

            bool HasWriteFunctionChanged(IModbusVariableInfo v)
                => _firstVariable.GetWriteFunction() != 0 &&
                   _firstVariable.GetWriteFunction() != v.GetWriteFunction();

            bool IsSingleWriteFunction(IModbusVariableInfo v) =>
v.GetWriteFunction() is 5 or 6;

            bool IsWriteFlagChanged(IModbusVariableInfo v)
                => _firstVariable.WritedVariableToChanged !=
v.WritedVariableToChanged;
        }

        internal void AddVariable(IModbusVariableInfo variable)
        {
            if (!HasVariables)
                _firstVariable = variable;

            var regCount = variable.GetRegisterCount();
            var regNumber = variable.GetRegisterNumber();

            UpdateState();
            _variableIds.Add(variable.VariableId);

            return;

            void UpdateState()
            {
                _currentRegisterCount += regCount;
                _previousRegisterOrBit = regNumber;
            }
        }

        internal bool CanAddVariable(IModbusVariableInfo variable)
        {
            if (!HasVariables) return true;

            var regCount = variable.GetRegisterCount();
            var regNumber = variable.GetRegisterNumber();

            return !(ExceedsMaxRegisterCount() ||
                     HasGap() ||
```

```
                    HasStatusDescriptorChanged() ||
                    TypesAreIncompatible() ||
                    _isNewGroupNeeded(variable));

            bool ExceedsMaxRegisterCount() =>
                _currentRegisterCount + regCount > Math.Min(
                    GetMaxRegistersCountByProtocol(),
                    _userMaxRegisterCount);

            ushort GetMaxRegistersCountByProtocol() =>

_registerLimitByProtocol.GetMaxRegisterCountByFunction(_getCurrentFunction(_f
irstVariable));

            bool HasGap()
                => _previousRegisterOrBit is not null &&
                   regNumber - _previousRegisterOrBit != regCount;

            bool HasStatusDescriptorChanged() =>

HasDescriptorChanged(_firstVariable.RwStatusVariableDescriptor,
variable.RwStatusVariableDescriptor);

            bool TypesAreIncompatible()
            {
                if (_firstVariable.GetRegisterCount() != regCount)
                    return true;

                return _firstVariable.Type != variable.Type
                       && (_firstVariable.Type, variable.Type) is not
(VariableType.Long, VariableType.Float)
                       and not (VariableType.Float, VariableType.Long);
            }
        }

        internal RequestBatch Build()
        {
            if (_firstVariable == null)throw new InvalidOperationException();

            return _buildGroup();
        }

        internal void Clear()
        {
            _currentRegisterCount = 0; _previousRegisterOrBit = null;
            _firstVariable = null; _variableIds.Clear();
        }

        private RequestBatch CreateGroup(IVariableDescriptor
startReadingDescriptor,
            IVariableDescriptor startWritingDescriptor, bool
writedVariableToChanged,
            byte readFunction, byte writeFunction)
        {
            return new RequestBatch(
                firstVariable: _firstVariable,
                variableIds: _variableIds.ToArray(),
                startReadingVariableDescriptor: startReadingDescriptor,
                startWritingVariableDescriptor: startWritingDescriptor,
                writedVariableToChanged: writedVariableToChanged,
                readFunction: readFunction, writeFunction: writeFunction);
        }
```

```
        private static bool HasDescriptorChanged(IVariableDescriptor
firstVarDescriptor,
            IVariableDescriptor nextVarDescriptor)
        {
            if (IsEmpty(firstVarDescriptor) && IsEmpty(nextVarDescriptor))
                return false;

            return !Equals(firstVarDescriptor?.UniqueId,
nextVarDescriptor?.UniqueId);

            bool IsEmpty(IVariableDescriptor d) => d == null || d.IsEmpty;
        }}
```

Обоснование выбора решения:

– разделение режимов для реализации независимой логики для чтения и записи;

– проверка ограничений с целью учёта лимитов протокола и пользовательских настроек;

– оптимизация групп путём объединения переменных по критериям совместимости.

В таблице 3.3 представлены основные функции класса RequestBatchBuilder и логика их работы.

Таблица 3.3 – Принцип работы класса RequestBatchBuilder

Функция	Назначение	Логика работы
SetReadMode	Настройка режима чтения	Устанавливает проверки для дескрипторов и функций чтения
SetWriteMode	Настройка режима записи	Устанавливает проверки для дескрипторов, функций записи и флагов
CanAddVariable	Проверка возможности добавления переменной	Проверяет лимиты регистров, разрывы адресов, совместимость типов
AddVariable	Добавление переменной в группу	Обновляет счетчик регистров, сохраняет адрес, добавляет ID
Build	Создание финальной группы	Проверяет наличие переменных, формирует RequestBatch
Clear	Сброс состояния билдера	Обнуляет счетчики, очищает коллекции переменных

3.1.5 Структура RequestBatch

Класс RequestBatch представляет собой неизменяемую структуру данных, которая инкапсулирует сгруппированный Modbus-запрос, содержащий несколько переменных. Данный класс реализован как Value Object в соответствии с принципами DDD.

Листинг 3.6 – Класс RequestBatch

```
public sealed class RequestBatch : ValueObject<RequestBatch>
    {
        /// <summary>
        /// Описатель первой переменной из запроса
        /// </summary>
        public IModbusVariableInfo FirstVariable { get; }
        /// <summary>
        /// Идентификаторы всех переменных, сгруппированных в запрос
        /// </summary>
        public IReadOnlyList<Guid> VariableIds { get; }
        public VariableType Type { get; }
        public uint Bit { get; }
        public IVariableDescriptor StartReadingVariableDescriptor { get; }
        public IVariableDescriptor StartWritingVariableDescriptor { get; }
        public IVariableDescriptor RwStatusVariableDescriptor { get; }
        public bool WritedVariableToChanged { get; }
        public int RegisterCount { get; }
        public byte ReadFunction { get; }
        public byte WriteFunction { get; }
        public uint RegisterNumber { get; }

        public RequestBatch(IModbusVariableInfo firstVariable,
            IReadOnlyList<Guid> variableIds,
            IVariableDescriptor startReadingVariableDescriptor,
            IVariableDescriptor startWritingVariableDescriptor,
            bool writedVariableToChanged, byte readFunction, byte
writeFunction)
        {
            FirstVariable = firstVariable ?? throw new
ArgumentNullException(nameof(firstVariable));
            VariableIds = variableIds;
            Type = firstVariable.Type;
            Bit = firstVariable.Bit;
            RwStatusVariableDescriptor =
firstVariable.RwStatusVariableDescriptor;
            RegisterCount = firstVariable.GetRegisterCount();
            RegisterNumber = firstVariable.GetRegisterNumber();

            ReadFunction = readFunction;
            StartReadingVariableDescriptor = startReadingVariableDescriptor;
            WriteFunction = writeFunction;
            WritedVariableToChanged = writedVariableToChanged;
            StartWritingVariableDescriptor = startWritingVariableDescriptor;
        }

        protected override IEnumerable<object> GetEqualityAttributes()
        {
            yield return FirstVariable;
            yield return Type;
            yield return Bit;
            yield return StartReadingVariableDescriptor;
            yield return StartWritingVariableDescriptor;
            yield return RwStatusVariableDescriptor;
            yield return WritedVariableToChanged;
```

```
            yield return RegisterCount;
            yield return ReadFunction;
            yield return WriteFunction;
            yield return RegisterNumber;

            foreach (var variableId in VariableIds)
                yield return variableId;
        }
    }
```

Назначение основных свойств:

– firstVariable в качестве переменной, определяющей параметры группы;

– variableIds в качестве идентификаторов всех переменных, объединенных в запрос;

– type и Bit хранят тип данных и битовое смещение, наследуемые от первой переменной;

– registerNumber и RegisterCount для хранения адреса и количества регистров;

– readFunction и WriteFunction в качестве кодов Modbus-функций для чтения и записи;

– startReadingVariableDescriptor и StartWritingVariableDescriptor в качестве дескрипторов для управления опросом.

– Обоснование архитектурных решений:

– использование паттерна Value Object для обеспечения неизменяемости и сравнения по значению;

– применение композиции на основе FirstVariable с целью уменьшения дублирования данных;

– использование интерфейса IReadOnlyList для VariableIds в качестве гарантии неизменяемости коллекции;

– реализация полной инкапсуляции, при которой все параметры устанавливаются только в конструкторе.

Роль в системе:

Данный класс является конечным результатом работы алгоритмов группировки и служит основой для генерации бинарных Modbus-запросов, которые передаются в компилятор Logic.

3.2 Реализация обработки ограничений протокола

Данный подраздел описывает реализацию механизма контроля максимального размера Modbus-запроса в зависимости от используемого протокола передачи данных (RTU или ASCII) и кода функции. Учет этих ограничений является критически важным для обеспечения корректности формируемых запросов и их успешной обработки Slave-устройствами.

3.2.1 Интерфейс IProtocolLimitProvider

Для обеспечения гибкости и соблюдения принципа инверсии зависимостей (Dependency Inversion Principle) был разработан абстрактный интерфейс IProtocolLimitProvider. Данный интерфейс инкапсулирует логику определения максимально допустимого количества регистров для конкретной функции Modbus, абстрагируя основной алгоритм группировки от деталей реализации протокола.

Листинг 3.7 – Интерфейс IProtocolLimitProvider

```
/// <summary>
/// Определяет контракт для получения ограничения количества регистров
/// в зависимости от протокола передачи данных Modbus.
/// </summary>
internal interface IProtocolLimitProvider
{
    /// <summary>
    /// Возвращает максимальное количество регистров для указанной функции
Modbus.
    /// </summary>
    /// <param name="functionCode">Код функции Modbus (например, 0x03,
0x10).</param>
    /// <returns>Максимально допустимое количество регистров.</returns>
    ushort GetMaxRegisterCountByFunction(byte functionCode);
}
```

Обоснование выбора решения:

– легко добавлять поддержку новых протоколов;

– упростить модульное тестирование с помочью mock-объектов;

– внедрять зависимости через DI-контейнер;

– соответствовать принципам чисто архитектуру и DDD.

3.2.2 Реализация ограничения RtuProtocolLimitProvider

Класс RtuProtocolLimitProvider предоставляет конкретную реализацию ограничений для протокола Modbus RTU. Ограничения в данном протоколе в первую очередь связаны с размером CRC-контрольной суммы (2 байта) и общей рекомендацией по длине фрейма для обеспечения надежной работы в реальном времени.

Листинг 3.8 – Класс RtuProtocolLimitProvider

```
/// <summary>
/// Реализует ограничения количества регистров для протокола Modbus RTU.
/// </summary>
internal sealed class RtuProtocolLimitProvider : IProtocolLimitProvider
{
    public ushort GetMaxRegisterCountByFunction(byte functionCode)
    {
        return functionCode switch
        {
            // Функция 0x10 (Write Multiple Registers) имеет более строгое
ограничение
            0x10 => 123,
            // Для функций чтения (0x03, 0x04) и других используется
стандартный лимит
            _ => 125
        };
    }
}
```

Анализ и обоснование расчетов:

– базовый лимит (125 регистров): для функций чтения 0x03 и 0x04 запрос содержит минимум служебных данных (адрес устройства, код функции, адрес регистра, количество регистров, CRC). Ответ же содержит сами данные. Ограничение в 125 регистров (250 байт данных) является де-факто стандартом, предотвращающим превышение максимального размера фрейма и потерю производительности;

– лимит для функции 0x10 (123 регистра): функция записи нескольких регистров 0x10 в своем запросе помимо служебных полей передает еще и байт с количеством байт данных и сами данные. Это увеличивает размер запроса, поэтому максимальное количество регистров slightly уменьшено до 123, чтобы итоговый размер фрейма оставался в безопасных пределах.

3.2.3 Реализация ограничения AsciiProtocolLimitProvider

Класс AsciiProtocolLimitProvider реализует более строгие ограничения, характерные для протокола Modbus ASCII. Данный протокол использует ASCII-кодировку, где каждый байт данных передается двумя ASCII-символами, что вдвое увеличивает размер фрейма по сравнению с RTU. Дополнительно налагаются ограничения из-за использования стартового и стопового символов, а также LRC-контрольной суммы.

Листинг 3.9 – Класс AsciiProtocolLimitProvider

```
/// <summary>
/// Реализует ограничения количества регистров для протокола Modbus ASCII.
/// </summary>
internal sealed class AsciiProtocolLimitProvider : IProtocolLimitProvider
{
    public ushort GetMaxRegisterCountByFunction(byte functionCode)
    {
        return functionCode switch
        {
            // Функции чтения регистров хранения (0x03) и входных регистров
(0x04)
            0x03 or 0x04 => 61,
            // Функция записи множества регистров (0x10)
            0x10 => 59,
            // Для прочих функций используется консервативное ограничение
            _ => 125
        };
    }
}
```

Анализ и обоснование расчетов:

– функции 0x03 и 0x04 (61 регистр): каждый регистр — это 2 байта. 61 регистр = 122 байта данных. В формате ASCII это превращается в 244 символа только на данные. С учетом служебной информации (адрес, функция, CRC в виде LRC и т.д.) общая длина фрейма приближается к верхнему допустимому пределу в 513 символов, установленному спецификацией Modbus;

– функция 0x10 (59 регистров): как и в случае с RTU, запрос на запись содержит больше служебных данных (включая байт размера). Поэтому лимит для этой функции еще более строгий — 59 регистров (118 байт данных), что гарантирует формирование валидного запроса, не превышающего максимальную длину фрейма Modbus ASCII.

3.3 Валидация параметров устройств RS-485

Данный подраздел описывает реализацию системы валидации параметров конфигурации Modbus-устройств. Система обеспечивает проверку корректности настроек устройств и переменных перед формированием запросов, что предотвращает ошибки на этапе выполнения Modbus-запросов.

3.3.1 Цепочка валидаторов ValidationChainBase

Класс ValidationChainBase реализует механизм построения цепочек валидаторов для комплексной проверки параметров устройств.

Листинг 3.10 – Класс RegisterCountPerRequestValidator

```
public abstract class ValidationChainBase : IValidatorChain
{
    [NonSerialized]
    protected BaseValidator<IRs485VariableInfo> _variableValidator;
    [NonSerialized]
    protected BaseValidator<IRs485DeviceInfo> _deviceValidator;

    public abstract BaseValidator<IRs485VariableInfo> GetVariableValidator();

    public BaseValidator<IRs485DeviceInfo> GetDeviceValidator()
    {
        return _deviceValidator ??=
            new Rs485DeviceNameValidator(
                new DeviceAddressValidator(
                    new Rs485DeviceAttemptsValidator(
                        new PollingIntervalValidator(
                            new Rs485DeviceTimeoutValidator(
                                new RegisterCountPerRequestValidator()))))));
    }
}
```

Обоснование выбора решения:

– абстрактный базовый класс как единая точка создания цепочек валидаторов;

– композиция валидаторов для последовательной проверки всех параметров устройства;

– lazy-инициализация для оптимизации производительности.

Принцип работы:

1. создает цепочку из шести валидаторов для проверки устройства;

2. последовательно проверяет имя, адрес, попытки, опрос, таймаут и регистры;

3. возвращает готовую цепочку для использования в UI.

3.3.2 Валидатор периода опроса PollingIntervalValidator

Класс PollingIntervalValidator проверяет корректность периода опроса устройства.

Листинг 3.11 – Класс RegisterPollingIntervalValidator

```
public sealed class PollingIntervalValidator :
BaseValidator<IRs485DeviceInfo>
{
    private const int MinValue = 0;
    private const int MaxValue = 65535;

    public PollingIntervalValidator(BaseValidator<IRs485DeviceInfo> next =
null) : base(next)
    {
```

```
        _request = new Rs485Request(Rs485ValidatingParameter.DevicePolling);
    }

    protected override bool IsValidValue(object newValue, IRs485DeviceInfo
owner, Action<string> errorCallback)
    {
        if (newValue == null)
            throw new ArgumentException();

        var newPollingStr = newValue.ToString();
        uint polling;
        bool result = uint.TryParse(newPollingStr, out polling) && polling <=
MaxValue;
        if (!result && errorCallback != null)
            errorCallback(string.Format(Resources.RangeError, MinValue,
MaxValue));

        return result;
    }
}
```

Обоснование выбора решения:

– наследование от BaseValidator обеспечивает единый интерфейс для всех валидаторов системы;

– паттерн Chain of Responsibility позволяет создавать цепочки валидаторов для комплексной проверки;

– nullable контекст повышает безопасность кода за счет явной обработки null-значений.

Принцип работы:

1. проверяет период опроса на соответствие диапазону 0-65535 мс;

2. обеспечивает корректную частоту опроса устройства.

3.3.3 Валидатор регистров RegisterCountPerRequestValidator

Класс RegisterCountPerRequestValidator проверяет количество регистров в Modbus-запросе. Наследуется от BaseValidator<IRs485DeviceInfo> и работает по принципу Chain of Responsibility.

Листинг 3.12 – Класс RegisterCountPerRequestValidator

```
    public sealed class RegisterCountPerRequestValidator :
BaseValidator<IRs485DeviceInfo>{
        private const int MinValue = 2; private const int MaxValue = 125;

        public
RegisterCountPerRequestValidator(BaseValidator<IRs485DeviceInfo>? next =
null) : base(next) => _request = new
Rs485Request(Rs485ValidatingParameter.RegistersPerRequestCount);

        protected override bool IsValidValue(object newValue,
IRs485DeviceInfo owner, Action<string>? errorCallback = null)
        {
```

```
            if (newValue == null)
                throw new ArgumentException(nameof(newValue));

            if (uint.TryParse(newValue.ToString(), out var count) && count is
>= MinValue and <= MaxValue) return true;
            errorCallback?.Invoke(string.Format(Resources.RangeError,
MinValue, MaxValue));
            return false;}}
```

Обоснование выбора решения:

– наследование от BaseValidator обеспечивает единый интерфейс для всех валидаторов системы;

– паттерн Chain of Responsibility позволяет создавать цепочки валидаторов для комплексной проверки;

– nullable контекст повышает безопасность кода за счет явной обработки null-значений.

Принцип работы:

1. проверяет, что значение находится в допустимом диапазоне 2-125 регистров;

2. преобразует входное значение в числовой формат с проверкой корректности;

3. генерирует локализованные сообщения об ошибках при несоответствии критериям;

4. передает управление следующему валидатору в цепочке при успешной проверке;

5. поддерживает опциональные callback-функции для передачи ошибок в UI.

3.3.4 Валидатор адреса устройства DeviceAddressValidator

Класс DeviceAddressValidator проверяет корректность Modbus-адреса устройства.

Листинг 3.13 – Класс RegisterCountPerRequestValidator

```
public sealed class DeviceAddressValidator : BaseValidator<IRs485DeviceInfo>
{
    private const int MinValue = 1; private const int MaxValue = 254;

    public DeviceAddressValidator(BaseValidator<IRs485DeviceInfo> next =
null) : base(next){
        _request = new Rs485Request(Rs485ValidatingParameter.DeviceAddress);
    }

    protected override bool IsValidValue(object newValue, IRs485DeviceInfo
owner, Action<string> errorCallback)
    {
        if (newValue == null) throw new ArgumentException();

        var newAddrStr = newValue.ToString();
        uint addr;
        bool result = false;
        if(uint.TryParse(newAddrStr, out addr))
            result = !(addr < MinValue || addr > MaxValue);
```

```
        if (!result && errorCallback != null)
            errorCallback(string.Format(Resources.RangeError, MinValue,
MaxValue));
        return result;
    }}
```

Обоснование выбора решения:

– наследование от BaseValidator обеспечивает единый интерфейс для всех валидаторов системы;

– паттерн Chain of Responsibility позволяет создавать цепочки валидаторов для комплексной проверки;

– nullable контекст повышает безопасность кода за счет явной обработки null-значений.

Принцип работы:

1. проверяет адрес устройства на соответствие диапазону 1-254;

2. обеспечивает корректность адресации в Modbus-сети.

3.4 Пример формирования групповых запросов

Для демонстрации работы алгоритма группировки рассмотрим практический пример формирования запросов для трех переменных с различными условиями чтения/записи. В примере задействуются все основные правила формирования групповых запросов.

Исходные параметры переменных представлены в таблице 3.4.

Таблица 3.4 - Исходные параметры переменных для примера группировки

Переменная	Тип	Адрес переменной	Функция	Период	По команде
Var-1	int 1 регистр	500	0x03	100 мс	Адрес переменной типа bool: 512
Var-2	int 1 регистр	501	0x03	100 мс	Адрес переменной типа bool: 512
Var-3	int 1 регистр	506	0x04	50 мс	Адрес переменной типа bool: 600

Дополнительное условие: максимальное количество регистров в запросе 16

Процесс формирования групповых запросов выполняется по следующему алгоритму:

1. Для переменной "Var-1" с младшим регистром и для переменной "Var-2" на следующем по порядку регистре формируем запросы:

– между адресами регистров переменных "Var-1" и "Var-2" нет разрыва - переменные можно объеденять в групповые запросы;

– у переменных Var-1 и Var-2 одинаковая функция 0x03 - переменные можно объеденять в групповые запросы;

– у переменных Var-1 и Var-2 одинаковое условие "Период" - переменные можно объеденить в "Запрос №1 - групповой запрос по периоду";

– у переменных Var-1 и Var-2 одинаковое условие "По команде" (одинаковая командная переменная) - переменные можно объеденить в "Запрос №2 - групповой запрос по одинаковой команде".

2. Для переменной "Var-3" на следующем НЕ по порядку регистре формируем запросы:

– между адресами регистров переменных "Var-2" и "Var-3" есть разрыв - переменные нельзя объединять в групповые запросы, соответственно для переменной "Var-3" формируем индивидуальные запросы:

– "Запрос №3 - индивидуальный запрос по периоду;

– "Запрос №4 - индивидуальный запрос по команде.

Визуализация процесса группировки

На рисунке 3.1 представлена диаграмма формирования групповых запросов для данного примера.

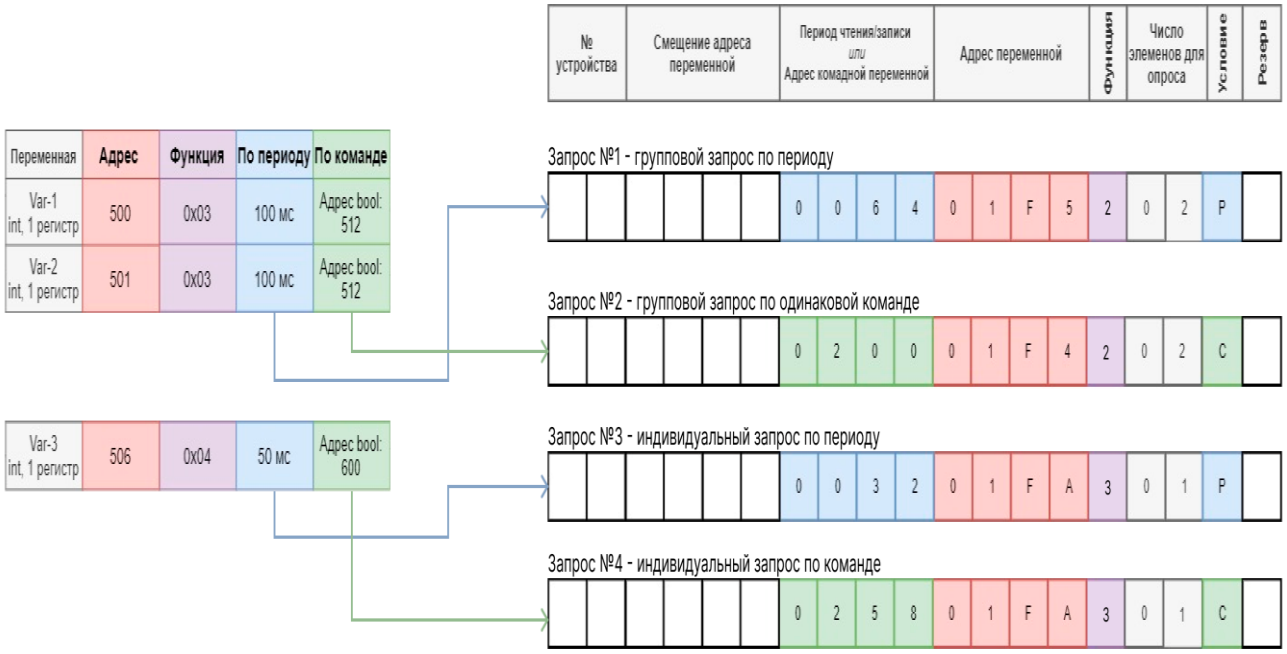

Рисунок 3.1 – Диаграмма формирования групповых запросов для переменных

Выходные данные и анализ эффективности

В результате работы алгоритма группировки сформировано 4 запроса вместо потенциальных 6 запросов (по одному запросу на каждое условие для каждой переменной при отсутствии группировки).

Коэффициент эффективности группировки:

– с группировкой: 4 запроса;

– без группировки: 6 запросов.

Анализ эффективности группировки (рисунок 3.2) показывает снижение количества запросов на 33%

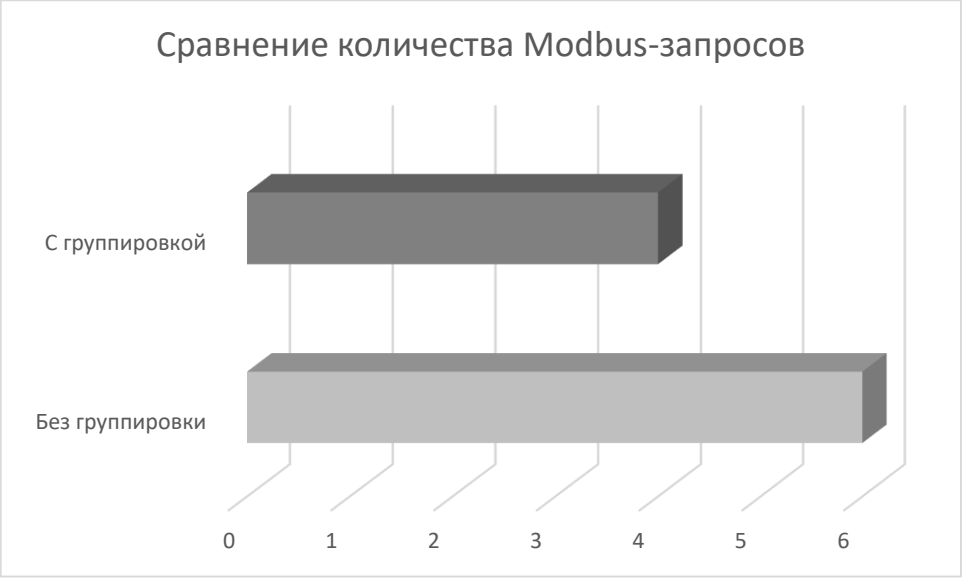

Рисунок 3.2 – Сравнение количества Modbus запросов при использовании группировки

Причины исключения Var-3 из групповых запросов №1 и №2:

– наличие разрыва в адресации регистров (501→506);

– использование другой функции Modbus (0x04 вместо 0x03);

– различные условия опроса (период 50 мс вместо 100 мс);

– разные командные переменные (600 вместо 512).

Данный пример наглядно демонстрирует работу алгоритма группировки и его эффективность при обработке переменных со смешанными параметрами.

4 Тестирование модуля

В данном разделе представлены результаты юнит-тестирования алгоритмов группировки Modbus-запросов. Тестирование проводилось с использованием фреймворка NUnit и библиотеки Moq для создания mock-объектов. Все тесты демонстрируют корректную работу алгоритмов группировки в различных сценариях. Для обеспечения комплексного покрытия функциональности были разработаны тестовые сценарии, охватывающие основные аспекты работы модуля, включая базовые алгоритмы группировки, обработку различных типов данных, работу с функциями Modbus, расширенные сценарии и обработку конфликтных ситуаций.

4.1 Тестирование алгоритмов группировки

Цель: проверить корректность работы алгоритмов группировки переменных в Modbus-запросы.

4.1.1 Тестирование сортировки и упорядочивания переменных

Методика: создаются переменные с произвольным порядком адресов регистров для проверки корректности сортировки.

Результаты:

– переменные с адресами [2, 8, 6, 4, 10] после сортировки группируются в 1 запрос;

– частично непрерывные адреса [0, 2, 6, 8] при ограничении в 5 регистров создают 2 запроса.

Вывод: алгоритм обеспечивает оптимальное упорядочивание переменных по возрастанию адресов регистров.

4.1.2 Тестирование обработки ограничений количества регистров

Методика: создаются наборы переменных, суммарное количество регистров которых превышает заданный лимит.

Результаты:

– 5 переменных по 1 регистру при лимите 3 регистра разделяются на 2 запроса;

– 3 переменных по 2 регистра при лимите 4 регистра разделяются на 2 запроса;

– переменные, укладывающиеся в лимит, создают один запрос.

Вывод: алгоритм корректно разделяет переменные при превышении лимита регистров.

4.1.3 Тестирование группировки по последовательности адресов

Методика: создаются переменные с непрерывными и разрывными адресами регистров.

Результаты:

– непрерывные адреса регистров группируются в один запрос;

– наличие разрыва в адресах приводит к созданию отдельных запросов;

– алгоритм корректно обрабатывает как 1-регистровые, так и 2-регистровые переменные.

Вывод: алгоритм эффективно объединяет переменные с непрерывными адресами регистров.

4.2 Тестирование группировки по типам данных

Цель: проверить влияние типа переменных на процесс группировки.

4.2.1 Тестирование группировки различных типов переменных

Методика: создаются переменные различных типов с разным количеством занимаемых регистров.

Результаты:

– переменные одного типа с разрывом в адресах создают несколько запросов;

– типы Float и Long с одинаковым количеством регистров группируются вместе;

– переменные трех разных типов создают отдельные запросы.

Вывод: Тип переменной влияет на группировку только через количество занимаемых регистров.

4.2.2 Тестирование группировки битовых переменных

Методика: создаются битовые переменные в пределах одного и разных регистров.

Результаты:

– биты в пределах одного регистра группируются в один запрос;

– биты из разных регистров создают отдельные запросы;

– корректно обрабатываются биты с различных позиций (0-15).

Вывод: для битовых переменных группировка происходит на уровне регистров, а не отдельных битов.

4.3 Тестирование группировки по функциям Modbus

Цель: проверить влияние функций Modbus на процесс группировки.

4.3.1 Тестирование группировки по функциям чтения/записи

Методика: создаются переменные с различными комбинациями функций чтения и записи.

Результаты:

– переменные с одинаковыми функциями чтения группируются вместе;

– переменные с одинаковыми функциями записи группируются вместе;

– различные функции чтения/записи создают отдельные запросы;

– совмещение чтения и записи в одном запросе возможно при совпадении функций.

Вывод: функции Modbus являются ключевым фактором при группировке переменных.

4.3.2 Тестирование группировки с учетом протоколов RTU/ASCII

Методика: создаются тестовые наборы переменных, превышающие максимально допустимое количество регистров для каждого протокола.

Результаты:

– для протокола ASCII при чтении 31 переменной типа Long (функция 3) создается 2 групповых запроса;

– для протокола ASCII при чтении 62 переменных типа Long (функция 4) создается 2 групповых запроса;

– для протокола ASCII при записи 30 переменных типа Long (функция 16) создается 2 групповых запроса;

– для протокола RTU при записи 62 переменных типа Long (функция 16) создается 2 групповых запроса.

Вывод: алгоритм корректно учитывает ограничения протоколов и разделяет переменные на оптимальное количество групповых запросов.

4.4 Тестирование расширенных сценариев группировки

Цель: проверить работу алгоритмов в сложных сценариях с дополнительными атрибутами переменных.

4.4.1 Тестирование группировки по статусу изменения переменных

Методика: создаются переменные с различными значениями флага WritedVariableToChanged.

Результаты:

– переменные с одинаковым статусом изменения группируются вместе;

– смешанные статусы создают отдельные запросы;

– флаг изменения влияет только на группировку запросов записи.

Вывод: статус изменения переменных учитывается при группировке запросов записи.

4.4.2 Тестирование группировки по дескрипторам статуса и команд

Методика: создаются переменные с общими и различными дескрипторами статуса и команд.

Результаты:

– общие дескрипторы статуса позволяют группировать переменные;

– различные дескрипторы создают отдельные запросы;

– пустые дескрипторы считаются эквивалентными.

Вывод: дескрипторы статуса и команд являются дополнительным критерием группировки.

4.5 Тестирование обработки конфликтов

Цель: проверить обработку конфликтных ситуаций с переменными.

4.5.1 Тестирование конфликтов порядка переменных

Методика: создаются переменные с пересекающимися адресами регистров.

Результаты:

– конфликты в середине, начале и конце запросов корректно обрабатываются;

– отсутствие конфликтов позволяет объединять запросы;

– алгоритм предотвращает некорректные группировки.

Вывод: Алгоритм надежно обрабатывает конфликтующие ситуации с переменными.

4.5.2 Тестирование переменных с одинаковыми адресами

Методика: создаются переменные с одинаковыми адресами регистров, но разными функциями Modbus.

Результаты:

– различные функции чтения для одинаковых адресов создают отдельные запросы;

– совмещение чтения и записи для одинаковых адресов возможно;

– битовые переменные с разными функциями обрабатываются корректно.

Вывод: переменные с одинаковыми адресами могут группироваться вместе только при совместимых функциях.

4.6 Тестирование валидаторов

Цель: проверить корректность работы валидаторов входных параметров.

4.6.1 Тестирование валидатора количества регистров в запросе

Методика: проверяются различные граничные значения количества регистров.

Результаты:

– значения в диапазоне 2-125 считаются валидными;

– значения 0, 1, 126 и более считаются не валидными;

– некорректные типы данных (строки, дробные числа) отклоняются;

– null-значения вызывают исключение.

Вывод: валидатор корректно проверяет допустимые значения количества регистров в запросе.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы разработан модуль обработки групповых Modbus-запросов для среды Owen Logic (OL).

Проведен анализ существующих методов оптимизации обмена данными по протоколам Modbus RTU и ASCII, рассмотрены ограничения аппаратных средств серии КС1 и особенности среды OL, а также выявлены ограничения существующих подходов к группировке запросов, что позволило определить рациональный метод реализации группового опроса устройств.

Разработана архитектура модуля, основанная на принципах модульности и разделения ответственности, обеспечивающая расширяемость и удобство сопровождения. Спроектированы ключевые компоненты: RequestPacket, RequestOrchestrationService и RequestBatchBuilder, формирующие одиночные и групповые Modbus-запросы с учётом протокольных ограничений, совместимости переменных и правил их объединения.

Реализованы алгоритмы фильтрации и сортировки переменных, проверки совместимости типов и адресов, формирования оптимальных групп запросов и их слияния для повышения эффективности обмена данными. Модуль включает валидацию всех входных параметров, что исключает ошибочные комбинации и обеспечивает надёжность работы с устройствами RS-485.

Тестирование подтвердило корректность работы алгоритмов, обработку ограничений по количеству регистров, последовательности адресов, совместимости функций и типов данных. Результаты показали, что модуль обеспечивает автоматическое формирование оптимальных групповых запросов, повышая производительность обмена и снижая нагрузку на интерфейс связи.

Отмечена научно-практическая значимость работы. Её результаты были представлены совместно с научным руководителем на профильной конференции «КИБ-2025», что подтверждает актуальность темы.

Дальнейшее развитие проекта может быть связано с расширением функциональности модуля, адаптацией под новые типы устройств и протоколов, а также интеграцией дополнительных алгоритмов оптимизации Modbus-обмена.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация Овен [Электронный ресурс] // Owen: [сайт]. [2025]. URL: https://owen.ru/documents (дата обращения: 23.11.2025).
2. Domain Driven Design: что это такое и зачем нужно [Электронный ресурс] // Хабр: [сайт]. [2025]. URL: https://habr.com/ru/articles/443770/ (дата обращения: 23.11.2025).
3. Микросервисы.NET: шаблоны DDD и CQRS [Электронный ресурс] // Microsoft Learn: [сайт]. [2025]. URL: https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ (дата обращения: 23.11.2025).
4. CODESYS - Инструмент разработки IEC 61131-3 [Электронный ресурс] // CODESYS: [сайт]. [2024]. URL: URL: https://www.codesys.com/ (дата обращения: 23.11.2025).
5. TIA Portal – единая среда для всех задач автоматизации [Электронный ресурс] // Siemens PRO: [сайт]. [2024]. URL: https://www.siemens-pro.ru/soft/tia-portal.html (дата обращения: 23.11.2025).
6. Trace Mode [Электронный ресурс] // Википедия: [сайт]. [2024]. URL: https://ru.wikipedia.org/wiki/Trace_mode (дата обращения: 23.11.2025).
7. Протокол Modbus: применение в промышленных сетях [Электронный ресурс] // Control Engineering Russia: [сайт]. [2025]. URL: https://www.controlengrussia.com/protokol-modbus-primenenie-v-promyshlennyh-setyah/ (дата обращения: 23.11.2025).
8. В. Х. Принципы юнит-тестирования. Санкт-Петербург: Питер, 2023. 320 pp.
9. В. А.С. Протокол Modbus: практическое применение в системах АСУ ТП. Москва: Горячая линия – Телеком, 2020. 288 pp.
10. Й. В. Введение в Modbus. Москва: АйСи Групп, 2016. 112 pp.
11. Документация по языку C# [Электронный ресурс] // Microsoft Learn: [сайт]. [2025]. URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 23.11.2025).
12. yEd Graph Editor - Samples [Электронный ресурс] // yWorks: [сайт]. [2025]. URL: https://www.yworks.com/products/yed#samples (дата обращения: 23.11.2025).
13. Микросервис, ориентированный на DDD [Электронный ресурс] // Microsoft Learn: [сайт]. [2025]. URL: URL: https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice (дата обращения: 23.11.2025).
14. Портал уроков по программированию [Электронный ресурс] // ITProger: [сайт]. [2025]. URL: https://itproger.com (дата обращения: 23.11.2025).

15. Форум программистов [Электронный ресурс] // CyberForum: [сайт]. [2025]. URL: https://cyberforum.ru/ (дата обращения: 23.11.2025).
16. Форум программистов [Электронный ресурс] // Overcoder: [сайт]. [2025]. URL: https://overcoder.net/ (дата обращения: 23.11.2025).
17. Дж. Р. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#. Санкт-Петербург: Питер, 2019. 896 pp.
18. Дж. О. C# 9.0. Справочник. Полное описание языка. Санкт-Петербург: Питер, 2022. 976 pp.
19. Э. Т. Джепикс Ф. C# 9.0 и платформа.NET 5. Москва: Диалектика, 2021. 1088 pp.
20. Бек К. Экстремальное программирование: разработка через тестирование. Санкт-Петербург: Питер, 2018. 224 pp.
21. Мартин Р. Чистый код. Создание, анализ и рефакторинг. Бостон: Pearson Education, Inc, 2009. 462 pp.
22. SCADA-системы: взгляд изнутри [Электронный ресурс] // Хабр: [сайт]. [2019]. URL: https://habr.com/ru/companies/advantech/articles/450234/ (дата обращения: 23.11.2025).
23. Диаграммы UML: виды, назначение и примеры использования [Электронный ресурс] // Weeek: [сайт]. [2024]. URL: https://weeek.net/ru/blog/uml-diagrams (дата обращения: 23.11.2025).
24. UML: от теории к практике [Электронный ресурс] // Хабр: [сайт]. [2020]. URL: https://habr.com/ru/articles/508710/ (дата обращения: 23.11.2025).
25. Domain-Driven Design для разработчиков [Электронный ресурс] // Microarch: [сайт]. [2025]. URL: https://microarch.ru/blog/domain-driven-design-dlya-razrabotchikov (дата обращения: 23.11.2025).

ПРИЛОЖЕНИЕ А

Листинги программы

A.1 Основные классы логики группировки

A.1.1 Класс RequestOrchestrationService

```
  public sealed class RequestOrchestrationService
  {
     internal IEnumerable<RequestPacket>
CreateRequests(IEnumerable<IModbusVariableInfo> variables,
         IVariableSource managerVariables, GlobalVariableDictionary managerGlobalVariables)
=> variables
         .Where(item => managerVariables.IsUsed(item.VariableId))
         .Select(item => RequestPacket.FromVariable(item, managerGlobalVariables));

     internal IEnumerable<RequestPacket>
CreateGroupRequests(IEnumerable<IModbusVariableInfo> variables,
         ushort maxRegisterCount, IVariableSource managerVariables,
         GlobalVariableDictionary managerGlobalVariables, ModbusDefinitions.Rs485Protocol
protocol)
     {
         var orderGenerator = new VariableSorter();

         var filteredSortedVariables = GetVariablesWithUsage();
         var readVars = GetReadVariables();
         var writeVars = GetWriteVariables();

         if (readVars.Count == 0 && writeVars.Count == 0)
            return Enumerable.Empty<RequestPacket>();

         var builder = new RequestBatchBuilder(
            maxRegisterCount,
            protocol == ModbusDefinitions.Rs485Protocol.Rtu
               ? new RtuProtocolLimitProvider()
               : new AsciiProtocolLimitProvider());
         var requestGroups = new List<RequestBatch>();

         BuildReadGroups();
         BuildWriteGroups();

         return MergeCompatibleGroups()
            .Select(group => RequestPacket.FromGroup(group, managerGlobalVariables));

         List<IModbusVariableInfo> GetVariablesWithUsage()
            => variables.Where(v => managerVariables.IsUsed(v.VariableId)).ToList();

         List<OrderedVariable> GetReadVariables()
         {
            return filteredSortedVariables
               .Where(v => v.GetReadFunction() != 0)
               .OrderBy(v => v.GetReadFunction())
               .ThenBy(v => v.GetRegisterNumber())
               .ThenBy(v => v.Bit)
               .Select(item => new OrderedVariable(item,
orderGenerator.GetNextOrderNumber()))
```

```
            .ToList();
        }

        List<OrderedVariable> GetWriteVariables()
        {
            return filteredSortedVariables
                .Where(v => v.GetWriteFunction() != 0 &&
                        (v.WritedVariableToChanged || v.StartWritingVariableDescriptor != null))
                .OrderBy(v => v.GetWriteFunction())
                .ThenBy(v => v.GetRegisterNumber())
                .ThenBy(v => v.Bit)
                .Select(item => new OrderedVariable(item, GetOrderNumber(item)))
                .ToList();

            uint GetOrderNumber(IModbusVariableInfo variableInfo)
            {
                var readVariable = readVars
                    .FirstOrDefault(item => item.VariableInfo.VariableId == variableInfo.VariableId);
                return readVariable.VariableInfo is null
                    ? orderGenerator.GetNextOrderNumber()
                    : readVariable.Order;
            }
        }

        void BuildReadGroups()
        {
            if (readVars.Count == 0) return;

            builder.SetReadMode();
            BuildGroupFromVariables(readVars);
        }

        void BuildWriteGroups()
        {
            if (writeVars.Count == 0) return;

            builder.SetWriteMode();
            BuildGroupFromVariables(writeVars);
        }

        void BuildGroupFromVariables(List<OrderedVariable> vars)
        {
            OrderedVariable previousVariable = default;
            foreach (var variable in vars)
            {
                if (!builder.CanAddVariable(variable.VariableInfo) ||
IsOrderConflict(previousVariable, variable))
                {
                    requestGroups.Add(builder.Build());
                    builder.Clear();
                }
```

```
            builder.AddVariable(variable.VariableInfo);
            previousVariable = variable;
        }

        requestGroups.Add(builder.Build());
        builder.Clear();
    }

    bool IsOrderConflict(OrderedVariable previousVariable, OrderedVariable variable)
    {
        if (previousVariable.VariableInfo == default)
            return false;

        return variable.Order != previousVariable.Order + 1;
    }

    IEnumerable<RequestBatch> MergeCompatibleGroups()
    {
        if (requestGroups.Count == 0) return Enumerable.Empty<RequestBatch>();

        return requestGroups
            .GroupBy(g => (g.FirstVariable.VariableId, g.VariableIds.Count))
            .SelectMany(group => ProcessGroup(group.ToList()));

        IEnumerable<RequestBatch> ProcessGroup(List<RequestBatch> items)
        {
            return items.Count == 2
                ? new[] { CreateMergedGroup(
                      items.First(g => g.ReadFunction != 0),
                      items.First(g => g.WriteFunction != 0))}
                : items;
        }

        RequestBatch CreateMergedGroup(RequestBatch read, RequestBatch write) =>
            new (firstVariable: read.FirstVariable, variableIds: read.VariableIds,
                startReadingVariableDescriptor: read.StartReadingVariableDescriptor,
                startWritingVariableDescriptor: write.StartWritingVariableDescriptor,
                writedVariableToChanged: write.WritedVariableToChanged,
                readFunction: read.ReadFunction, writeFunction: write.WriteFunction);
    }
}

private class VariableSorter
{
    private uint _nextNumber = 0;

    public uint GetNextOrderNumber() => ++_nextNumber;
}

private readonly struct OrderedVariable
{
    public IModbusVariableInfo VariableInfo { get; }
```

```
        public uint Order { get; }

        public OrderedVariable(IModbusVariableInfo variableInfo, uint order)
        {
            VariableInfo = variableInfo;
            Order = order;
        }
    }
```

A.1.2 Класс RequestBatchBuilder

```
internal sealed class RequestBatchBuilder
{
    private readonly IProtocolLimitProvider _registerLimitByProtocol;
    private readonly ushort _userMaxRegisterCount;
    private readonly List<Guid> _variableIds;

    private IModbusVariableInfo _firstVariable;
    private int _currentRegisterCount;
    private uint? _previousRegisterOrBit;

    private Func<IModbusVariableInfo, bool> _isNewGroupNeeded;
    private Func<RequestBatch> _buildGroup;
    private Func<IModbusVariableInfo, byte> _getCurrentFunction;

    private bool HasVariables => _variableIds.Count > 0;

    internal RequestBatchBuilder(ushort userMaxRegisterCount,
        IProtocolLimitProvider registerLimitByProtocol)
    {
        _userMaxRegisterCount = userMaxRegisterCount;
        _registerLimitByProtocol = registerLimitByProtocol;
        _variableIds = new List<Guid>();
    }

    internal void SetReadMode()
    {
        _getCurrentFunction = v => v.GetReadFunction();

        _isNewGroupNeeded = variable =>
            HasReadDescriptorChanged(variable) ||
            HasReadFunctionChanged(variable);

        _buildGroup = () => CreateGroup(
            startReadingDescriptor: _firstVariable.StartReadingVariableDescriptor,
            startWritingDescriptor: null, writedVariableToChanged: false,
            readFunction: _firstVariable.GetReadFunction(), writeFunction: 0
        );

        return;

        bool HasReadDescriptorChanged(IModbusVariableInfo v) =>
```

```
            HasDescriptorChanged(_firstVariable.StartReadingVariableDescriptor,
v.StartReadingVariableDescriptor);

        bool HasReadFunctionChanged(IModbusVariableInfo v)
            => _firstVariable.GetReadFunction() != 0 &&
               _firstVariable.GetReadFunction() != v.GetReadFunction();
    }

    internal void SetWriteMode()
    {
        _getCurrentFunction = v => v.GetWriteFunction();

        _isNewGroupNeeded = variable =>
            HasWriteDescriptorChanged(variable) ||
            HasWriteFunctionChanged(variable) ||
            IsSingleWriteFunction(variable) ||
            IsWriteFlagChanged(variable);

        _buildGroup = () => CreateGroup(
            startReadingDescriptor: null,
            startWritingDescriptor: _firstVariable.StartWritingVariableDescriptor,
            writedVariableToChanged: _firstVariable.WritedVariableToChanged,
            readFunction: 0, writeFunction: _firstVariable.GetWriteFunction()
        );

        return;

        bool HasWriteDescriptorChanged(IModbusVariableInfo v) =>
            HasDescriptorChanged(_firstVariable.StartWritingVariableDescriptor,
v.StartWritingVariableDescriptor);

        bool HasWriteFunctionChanged(IModbusVariableInfo v)
            => _firstVariable.GetWriteFunction() != 0 &&
               _firstVariable.GetWriteFunction() != v.GetWriteFunction();

        bool IsSingleWriteFunction(IModbusVariableInfo v) => v.GetWriteFunction() is 5 or 6;

        bool IsWriteFlagChanged(IModbusVariableInfo v)
            => _firstVariable.WritedVariableToChanged != v.WritedVariableToChanged;
    }

    internal void AddVariable(IModbusVariableInfo variable)
    {
        if (!HasVariables)
            _firstVariable = variable;

        var regCount = variable.GetRegisterCount();
        var regNumber = variable.GetRegisterNumber();

        UpdateState();
        _variableIds.Add(variable.VariableId);
```

```
        return;

        void UpdateState()
        {
            _currentRegisterCount += regCount;
            _previousRegisterOrBit = regNumber;
        }
    }

    internal bool CanAddVariable(IModbusVariableInfo variable)
    {
        if (!HasVariables) return true;

        var regCount = variable.GetRegisterCount();
        var regNumber = variable.GetRegisterNumber();

        return !(ExceedsMaxRegisterCount() ||
                 HasGap() ||
                 HasStatusDescriptorChanged() ||
                 TypesAreIncompatible() ||
                 _isNewGroupNeeded(variable));

        bool ExceedsMaxRegisterCount() =>
            _currentRegisterCount + regCount > Math.Min(
                GetMaxRegistersCountByProtocol(),
                _userMaxRegisterCount);

        ushort GetMaxRegistersCountByProtocol() =>

_registerLimitByProtocol.GetMaxRegisterCountByFunction(_getCurrentFunction(_firstVariable
));

        bool HasGap()
            => _previousRegisterOrBit is not null &&
               regNumber - _previousRegisterOrBit != regCount;

        bool HasStatusDescriptorChanged() =>
            HasDescriptorChanged(_firstVariable.RwStatusVariableDescriptor,
variable.RwStatusVariableDescriptor);

        bool TypesAreIncompatible()
        {
            if (_firstVariable.GetRegisterCount() != regCount)
                return true;

            return _firstVariable.Type != variable.Type
                   && (_firstVariable.Type, variable.Type) is not (VariableType.Long,
VariableType.Float)
                   and not (VariableType.Float, VariableType.Long);
        }
    }
```

```
internal RequestBatch Build()
{
   if (_firstVariable == null)
      throw new InvalidOperationException();

   return _buildGroup();
}

internal void Clear()
{
   _currentRegisterCount = 0;
   _previousRegisterOrBit = null;
   _firstVariable = null;
   _variableIds.Clear();
}

private RequestBatch CreateGroup(IVariableDescriptor startReadingDescriptor,
   IVariableDescriptor startWritingDescriptor, bool writedVariableToChanged,
   byte readFunction, byte writeFunction)
{
   return new RequestBatch(
      firstVariable: _firstVariable,
      variableIds: _variableIds.ToArray(),
      startReadingVariableDescriptor: startReadingDescriptor,
      startWritingVariableDescriptor: startWritingDescriptor,
      writedVariableToChanged: writedVariableToChanged,
      readFunction: readFunction, writeFunction: writeFunction);
}

private static bool HasDescriptorChanged(IVariableDescriptor firstVarDescriptor,
   IVariableDescriptor nextVarDescriptor)
{
   if (IsEmpty(firstVarDescriptor) && IsEmpty(nextVarDescriptor))
      return false;

   return !Equals(firstVarDescriptor?.UniqueId, nextVarDescriptor?.UniqueId);

   bool IsEmpty(IVariableDescriptor d) => d == null || d.IsEmpty;
}}
```

A.1.3 - Структура RequestBatch

```
public sealed class RequestBatch: ValueObject<RequestBatch>
{
    /// <summary>
    /// Описатель первой переменной из запроса
    /// </summary>
    public IModbusVariableInfo FirstVariable { get; }
    /// <summary>
    /// Идентификаторы всех переменных, сгруппированных в запрос
    /// </summary>
    public IReadOnlyList<Guid> VariableIds { get; }
    public VariableType Type { get; }
    public uint Bit { get; }
    public IVariableDescriptor StartReadingVariableDescriptor { get; }
    public IVariableDescriptor StartWritingVariableDescriptor { get; }
    public IVariableDescriptor RwStatusVariableDescriptor { get; }
    public bool WritedVariableToChanged { get; }
    public int RegisterCount { get; }
    public byte ReadFunction { get; }
    public byte WriteFunction { get; }
    public uint RegisterNumber { get; }

    public RequestBatch (IModbusVariableInfo firstVariable,
        IReadOnlyList<Guid> variableIds,
        IVariableDescriptor startReadingVariableDescriptor,
        IVariableDescriptor startWritingVariableDescriptor,
        bool writedVariableToChanged, byte readFunction, byte writeFunction)
    {
        FirstVariable = firstVariable ?? throw new
ArgumentNullException(nameof(firstVariable));
        VariableIds = variableIds;
        Type = firstVariable.Type;
        Bit = firstVariable.Bit;
        RwStatusVariableDescriptor = firstVariable.RwStatusVariableDescriptor;
        RegisterCount = firstVariable.GetRegisterCount();
        RegisterNumber = firstVariable.GetRegisterNumber();

        ReadFunction = readFunction;
        StartReadingVariableDescriptor = startReadingVariableDescriptor;
        WriteFunction = writeFunction;
        WritedVariableToChanged = writedVariableToChanged;
        StartWritingVariableDescriptor = startWritingVariableDescriptor;
    }

    protected override IEnumerable<object> GetEqualityAttributes()
    {
        yield return FirstVariable;
        yield return Type;
        yield return Bit;
        yield return StartReadingVariableDescriptor;
        yield return StartWritingVariableDescriptor;
        yield return RwStatusVariableDescriptor;
```

```
        yield return WritedVariableToChanged;
        yield return RegisterCount;
        yield return ReadFunction;
        yield return WriteFunction;
        yield return RegisterNumber;

        foreach (var variableId in VariableIds)
            yield return variableId;
    }}
```

A.2 Классы обработки ограничений протокола

A.2.1 - Интерфейс IProtocolLimitProvider

```
internal interface IProtocolLimitProvider
{
    ushort GetMaxRegisterCountByFunction(byte functionCode);
}
```

A.2.2 - Класс RtuProtocolLimitProvider

```
  public ushort GetMaxRegisterCountByFunction(byte functionCode)
  {
      return functionCode switch
      {
          0x10 => 123,
          _ => 125

      };}
```

A.2.3 - Класс AsciiProtocolLimitProvider

```
  public ushort GetMaxRegisterCountByFunction(byte functionCode)
 {
     return functionCode switch
     {
         0x03 or 0x04 => 61,
         0x10 => 59,
         _ => 125
     };}
```

A.3 Система валидации параметров

A.3.1 - Базовый класс ValidationChainBase

```
public abstract class ValidationChainBase : IValidatorChain
{
    [NonSerialized]
    protected BaseValidator<IRs485VariableInfo> _variableValidator;
    [NonSerialized]
    protected BaseValidator<IRs485DeviceInfo> _deviceValidator;


    public abstract BaseValidator<IRs485VariableInfo> GetVariableValidator();

    /// <summary>
    /// Возвращает цепочку валидаторов устройств
    /// </summary>
    /// <returns></returns>
    public BaseValidator<IRs485DeviceInfo> GetDeviceValidator()
    {
        return _deviceValidator ??=
            new Rs485DeviceNameValidator(
                new DeviceAddressValidator(
                    new Rs485DeviceAttemptsValidator(
                        new PollingIntervalValidator(
                            new Rs485DeviceTimeoutValidator(
                                new RegisterCountPerRequestValidator())))));
    }}
```

A.3.2 - Класс ValidatorChainProxy

```
internal sealed class ValidatorChainProxy : ValidationChainBase
{
    private readonly List<StorageDescription> _networkStorages;

    public ValidatorChainProxy(IEnumerable<StorageDescription> networkStorages)
    {
        _networkStorages = networkStorages.ToList();
    }

    /// <summary>
    /// Возвращает цепочку валидаторов переменных
    /// </summary>
    /// <returns></returns>
    public override BaseValidator<IRs485VariableInfo> GetVariableValidator()
    {
        return _variableValidator ??
            (_variableValidator = new NetworkVariableAddressValidator(
                new
ModbusRegisterRangeValidator(_networkStorages.GetModbusRegisterRanges(),
                    new NetworkVariableBitValidator())));
    }}
```

A.3.3 - Валидатор Rs485DeviceNameValidator

```
public sealed class Rs485DeviceNameValidator: BaseValidator<IRs485DeviceInfo>
{
    public const int MinLength = 1;
    public const int MaxLength = 14;

    public Rs485DeviceNameValidator(BaseValidator<IRs485DeviceInfo> next = null)
        : base(next)
    {
        _request = new Rs485Request(Rs485ValidatingParameter.DeviceName);
    }

    protected override bool IsValidValue(object newDeviceName, IRs485DeviceInfo owner,
        Action<string> errorCallback)
    {
        if (newDeviceName == null)
            throw new ArgumentException();

        var deviceName = (string) newDeviceName;

        var result = !string.IsNullOrEmpty(deviceName) && deviceName.Length >= MinLength
                && deviceName.Length <= MaxLength;

        if (!result && errorCallback != null)
            errorCallback(string.Format(Resources.DeviceNameError, MinLength, MaxLength));

        return result;
    }}
```

A.3.4 - Валидатор DeviceAddressValidator

```
public sealed class DeviceAddressValidator : BaseValidator<IRs485DeviceInfo>
{
    private const int MinValue = 1;
    private const int MaxValue = 254;

    public DeviceAddressValidator(BaseValidator<IRs485DeviceInfo> next = null) : base(next)
    {
        _request = new Rs485Request(Rs485ValidatingParameter.DeviceAddress);
    }

    protected override bool IsValidValue(object newValue, IRs485DeviceInfo owner,
Action<string> errorCallback)
    {
        if (newValue == null)
            throw new ArgumentException();

        var newAddrStr = newValue.ToString();
        uint addr;
        bool result = false;
        if(uint.TryParse(newAddrStr, out addr))
            result = !(addr < MinValue || addr > MaxValue);
```

```
        if (!result && errorCallback != null)
            errorCallback(string.Format(Resources.RangeError, MinValue, MaxValue));
        return result;
    }
}
```

A.3.5 - Валидатор Rs485DeviceAttemptsValidator

```
public sealed class Rs485DeviceAttemptsValidator : BaseValidator<IRs485DeviceInfo>
{
    private const int MinValue = 0;
    private const int MaxValue = 255;

    public Rs485DeviceAttemptsValidator(BaseValidator<IRs485DeviceInfo> next = null) :
base(next)
    {
        _request = new Rs485Request(Rs485ValidatingParameter.DeviceAttempts);
    }

    protected override bool IsValidValue(object newValue, IRs485DeviceInfo owner,
Action<string> errorCallback)
    {
        if (newValue == null)
            throw new ArgumentException();

        var newAttemptsStr = newValue.ToString();
        uint attempts;
        bool result = false;
        if (uint.TryParse(newAttemptsStr, out attempts))
            result = attempts <= MaxValue;

        if (!result && errorCallback != null)
            errorCallback(string.Format(Resources.RangeError, MinValue, MaxValue));
        return result;
    }
}
```

A.3.6 - Валидатор PollingIntervalValidator

```
public sealed class PollingIntervalValidator : BaseValidator<IRs485DeviceInfo>
{
    private const int MinValue = 0;
    private const int MaxValue = 65535;

    public PollingIntervalValidator(BaseValidator<IRs485DeviceInfo> next = null) : base(next)
    {
        _request = new Rs485Request(Rs485ValidatingParameter.DevicePolling);
    }

    protected override bool IsValidValue(object newValue, IRs485DeviceInfo owner,
Action<string> errorCallback)
    {
        if (newValue == null)
            throw new ArgumentException();

        var newPollingStr = newValue.ToString();
        uint polling;
        bool result = uint.TryParse(newPollingStr, out polling) && polling <= MaxValue;
        if (!result && errorCallback != null)
            errorCallback(string.Format(Resources.RangeError, MinValue, MaxValue));

        return result;
    }
}
```

A.3.7 - Валидатор Rs485DeviceTimeoutValidator

```
public sealed class Rs485DeviceTimeoutValidator : BaseValidator<IRs485DeviceInfo>
{
    private const int MinValue = 0;
    private const int MaxValue = 65535;

    public Rs485DeviceTimeoutValidator(BaseValidator<IRs485DeviceInfo> next = null) :
base(next)
    {
        _request = new Rs485Request(Rs485ValidatingParameter.DeviceTimeout);
    }

    protected override bool IsValidValue(object newValue, IRs485DeviceInfo owner,
Action<string> errorCallback)
    {
        if (newValue == null)
            throw new ArgumentException();

        var newTimeoutStr = newValue.ToString();
        uint timeout;
        bool result = false;
        if(uint.TryParse(newTimeoutStr, out timeout))
            result = timeout <= MaxValue;
```

```
        if (!result && errorCallback != null)
            errorCallback(string.Format(Resources.RangeError, MinValue, MaxValue));

        return result;
    }}
```

A.3.8 - Валидатор RegisterCountPerRequestValidator

```
public sealed class RegisterCountPerRequestValidator : BaseValidator<IRs485DeviceInfo>
{
    private const int MinValue = 2;
    private const int MaxValue = 48;

    public RegisterCountPerRequestValidator(BaseValidator<IRs485DeviceInfo>? next = null) :
base(next)
        => _request = new Rs485Request(Rs485ValidatingParameter.RegistersPerRequestCount);

    protected override bool IsValidValue(object newValue, IRs485DeviceInfo owner,
Action<string>? errorCallback = null)
    {
        if (newValue == null)
            throw new ArgumentException(nameof(newValue));

        if (uint.TryParse(newValue.ToString(), out var count) && count is >= MinValue and <=
MaxValue) return true;
        errorCallback?.Invoke(string.Format(Resources.RangeError, MinValue, MaxValue));
        return false;
    }
}
```

A.4 Классы представления запросов

A.4.1 - Класс RequestPacket

```
public sealed class RequestPacket
{
    private byte _varTypeAndBit;
    private byte _readFunction;
    private byte _writeFunction;
    private byte _timer;
    private int _data;
    private ushort _quantity;
    private ushort _startRegister;

    private RequestPacket(IModbusVariableInfo variableInfo,
        GlobalVariableDictionary globalVariables, IReadOnlyList<Guid> variableIds)
    {
        _variableInfo = variableInfo;
        _globalVariables = globalVariables;
        _variableIds = variableIds;
    }

    public static RequestPacket FromVariable(IModbusVariableInfo variable,
```

```
            GlobalVariableDictionary globals)
        {
            return new RequestPacket(variable, globals, new []{ variable.VariableId })
                .Initialize(variable.Type, variable.GetRegisterCount(), variable.GetReadFunction(),
                    variable.GetWriteFunction(), variable.WritedVariableToChanged,
                    variable.GetRegisterNumber(), variable.Bit);
        }

        public static RequestPacket FromGroup(RequestBatch group,
            GlobalVariableDictionary globals)
        {
            return new RequestPacket(group.FirstVariable, globals, group.VariableIds)
                .Initialize(group.Type, group.RegisterCount, group.ReadFunction,
                    group.WriteFunction, group.WritedVariableToChanged,
                    group.RegisterNumber, group.Bit);
        }

        private RequestPacket Initialize(VariableType type, int registerCount,
            byte readFunc, byte writeFunc, bool writeOnChange,
            uint regNumber, uint bit)
        {
            if (type == VariableType.Long && registerCount == 1)
                _varTypeAndBit = (byte)DataVariableType.Word;
            else if (type == VariableType.Float || (type == VariableType.Long && registerCount ==
2))
                _varTypeAndBit = (byte)DataVariableType.Dword;
            else if (type == VariableType.Boolean)
                _varTypeAndBit = (byte)((uint)DataVariableType.Bit | bit << 4);
            else
                throw new
NotSupportedException(string.Format(Properties.Resources.TypeNotDefined, type));

            _readFunction = readFunc;
            _writeFunction = writeFunc;
            _timer = (byte)((readFunc > 0 ? 0x01 : 0) |
                        (writeFunc > 0 ? (writeOnChange ? 0x04 : 0) : 0));  // запись по изменению,
если есть функция записи!
            _startRegister = (ushort)regNumber;
            _quantity = (ushort)_variableIds.Count;

            return this;
        }

        internal void Analyze(ISeries200BuilderFacade builderFacade)
        {
            var readingDescriptor = _variableInfo.StartReadingVariableDescriptor;
            if (readingDescriptor != null)
                _evReadVariable = builderFacade.GetVariable(readingDescriptor.UniqueId);
            var writingDescriptor = _variableInfo.StartWritingVariableDescriptor;
            if (writingDescriptor != null)
                _evWriteVariable = builderFacade.GetVariable(writingDescriptor.UniqueId);
            var statusDescriptor = _variableInfo.RwStatusVariableDescriptor;
```

```
            if (statusDescriptor != null)
                _statusVariable = builderFacade.GetVariable(statusDescriptor.UniqueId);

            _dataVariable = builderFacade.GetVariable(_variableInfo.VariableId);
            if (_dataVariable == null)
                throw new
InvalidOperationException(string.Format(Properties.Resources.VariableNotDefined,
_variableInfo));

            if (_variableIds.Count == 1)
                return;
            // NOTE: Резервируем память под все переменные (если она ещё не
зарезервирована), участвующие в запросе в правильном порядке
            foreach (var variableId in _variableIds)
                builderFacade.GetVariable(variableId);}

        internal void Build(Stream stream, ref int address, ProgramCode program)
        {
            if (_dataVariable != null)
                _data = _globalVariables.GetAddressInGlobalBuffer(_dataVariable);

            if (_evReadVariable != null)
                _evRead = _globalVariables.GetAddressInGlobalBuffer(_evReadVariable);
            if (_evWriteVariable != null)
                _evWrite = _globalVariables.GetAddressInGlobalBuffer(_evWriteVariable);
            if (_statusVariable != null)
                _status = _globalVariables.GetAddressInGlobalBuffer(_statusVariable);
        }}
```

A.4.2 - Перечисление DataVariableType

```
internal enum DataVariableType
 {
    Bit,
    Word,
    Dword
 }
```

ПРИЛОЖЕНИЕ Б

Результаты выполнения модуля

Б.1 Сценарий 1: Идеальная группировка

В таблице Б.1 приведены параметры переменных для сценария 1, используемого для демонстрации идеальной группировки запросов в модуле.

Таблица Б.1 - Параметры переменных для сценария 1

Переменная	Тип данных	Адрес	Функция	Период опроса
Var-1	float	0	0x03	100 мс
Var-2	float	2	0x03	100 мс
Var-3	float	4	0x03	100 мс
Var-4	float	6	0x03	100 мс
Var-5	float	8	0x03	100 мс
Var-6	float	10	0x03	100 мс
Var-7	float	12	0x03	100 мс
Var-8	float	14	0x03	100 мс
Var-9	float	16	0x03	100 мс
Var-10	float	18	0x03	100 мс

Ограничения: максимальное количество регистров в запросе 125 (Modbus RTU)

Результаты работы:

– запросов без группировки: 10 (каждая переменная отдельно);

– запросов с группировкой: 1 (все переменные в одном запросе);

– сокращение запросов: 90%.

На рисунке Б.1 представлен интерфейс среды Owen Logic с конфигурацией переменных для данного сценария данных.

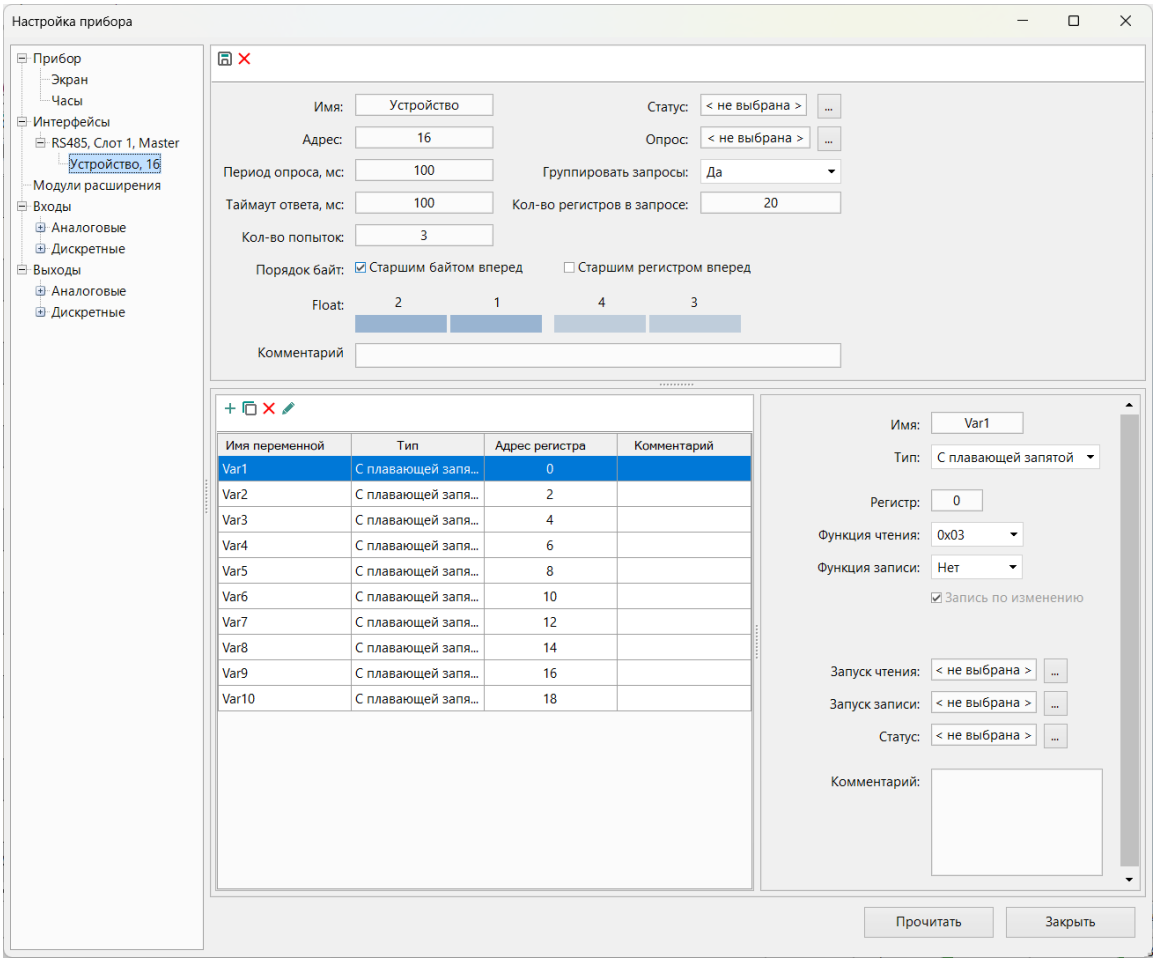

Рисунок Б.1 – Конфигурация переменных в среде Owen Logic (сценарий 1)

На рисунке Б.2 показан результат объединения 10 переменных в один запрос.

Рисунок Б.2 – Результаты формирования групповых запросов (сценарий 1)

Б.2 Сценарий 1: Группировка отключена

Исходные данные: используется набор из 10 переменных типа float, параметры которых приведены в таблице Б.1.

Результаты работы:

– запросов без группировки: 10;

– запросов с группировкой: 10 (группировка отключена);

– сокращение запросов: 0%.

На рисунке Б.3 показано отключение функции группировки в настройках модуля Owen Logic.

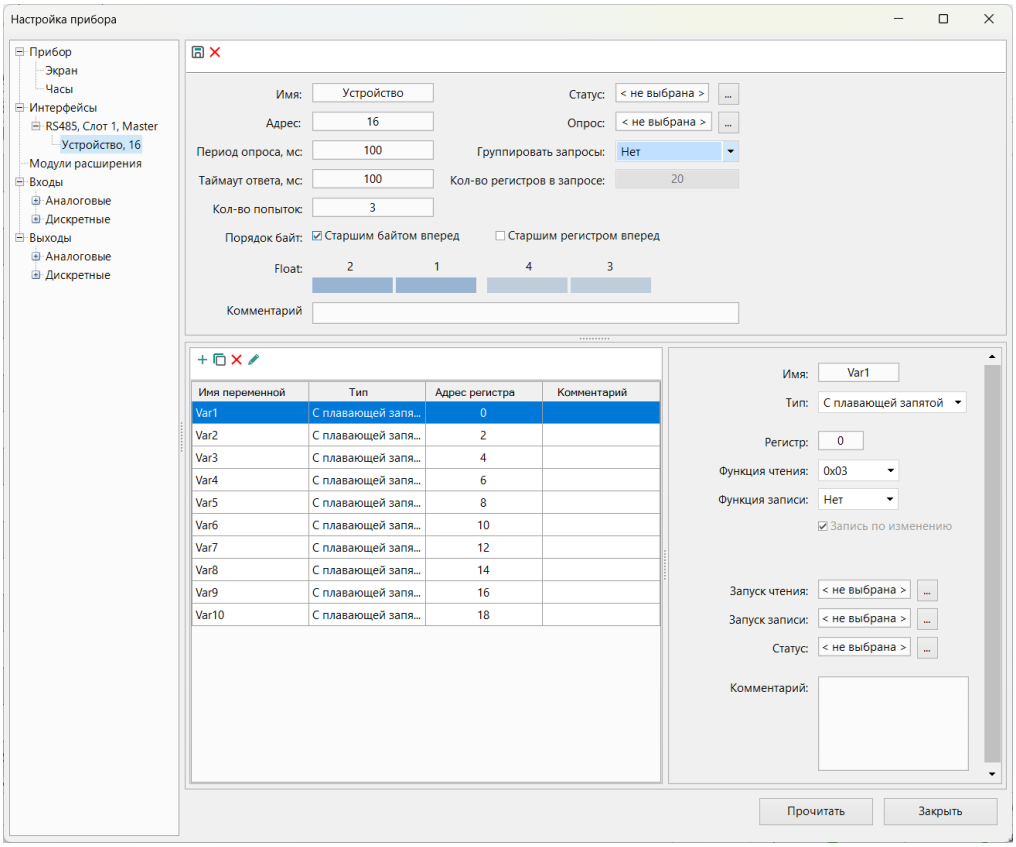

Рисунок Б.3 – Отключение группировки в настройках модуля

На рисунке Б.4 показано формирование 10 индивидуальных запросов (по одному на каждую переменную).

```
[16-12-2025 19:50:05.326] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 00 00 FB 32
[16-12-2025 19:50:05.326] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 12 00 02 67 4F
[16-12-2025 19:50:05.279] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 00 00 FB 32
[16-12-2025 19:50:05.279] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 10 00 02 C6 8F
[16-12-2025 19:50:05.232] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 00 00 FB 32
[16-12-2025 19:50:05.232] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 0E 00 02 A6 89
[16-12-2025 19:50:05.185] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 00 00 FB 32
[16-12-2025 19:50:05.185] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 0C 00 02 07 49
[16-12-2025 19:50:05.138] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 00 00 FB 32
[16-12-2025 19:50:05.138] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 0A 00 02 E7 48
[16-12-2025 19:50:05.091] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 40 A0 CA 8A
[16-12-2025 19:50:05.091] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 08 00 02 46 88
[16-12-2025 19:50:05.044] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 40 80 CB 52
[16-12-2025 19:50:05.044] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 06 00 02 27 4B
[16-12-2025 19:50:04.998] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 40 40 CB 02
[16-12-2025 19:50:04.998] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 04 00 02 86 8B
[16-12-2025 19:50:04.953] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 40 00 CA F2
[16-12-2025 19:50:04.953] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 02 00 02 66 8A
[16-12-2025 19:50:04.906] TRA : Node1::Device1:(COM4) Tx: [0009] 10 03 04 00 00 3F 80 EB 62
[16-12-2025 19:50:04.906] TRA : Node1::Device1:(COM4) Rx: [0008] 10 03 00 00 00 02 C7 4A
```

Рисунок Б.4 – Результаты формирования индивидуальных запросов (сценарий 2)

Б.3 Сценарий 3: Смешанные условия

В таблице Б.2 приведены параметры переменных для сценария 3, в котором рассматриваются смешанные условия группировки.

Таблица Б.2 - Параметры переменных для сценария 3

Переменная	Тип данных	Адрес переменной	Функция	Период	По команде
Var-1	int (1 рег)	500	0x03	100 мс	bool: 512
Var-2	int (1 рег)	501	0x03	100 мс	bool: 512
Var-3	int (1 рег)	506	0x04	50 мс	bool: 600

Ограничения: максимальное количество регистров в запросе 16

Результаты работы:

– запросов без группировки 6(по 2 запроса на каждую переменную);

– запросов с группировкой: 4;

– сокращение запросов: 33%.

На рисунке Б.5 представлена конфигурация переменных для сценария 3 в среде Owen Logic.

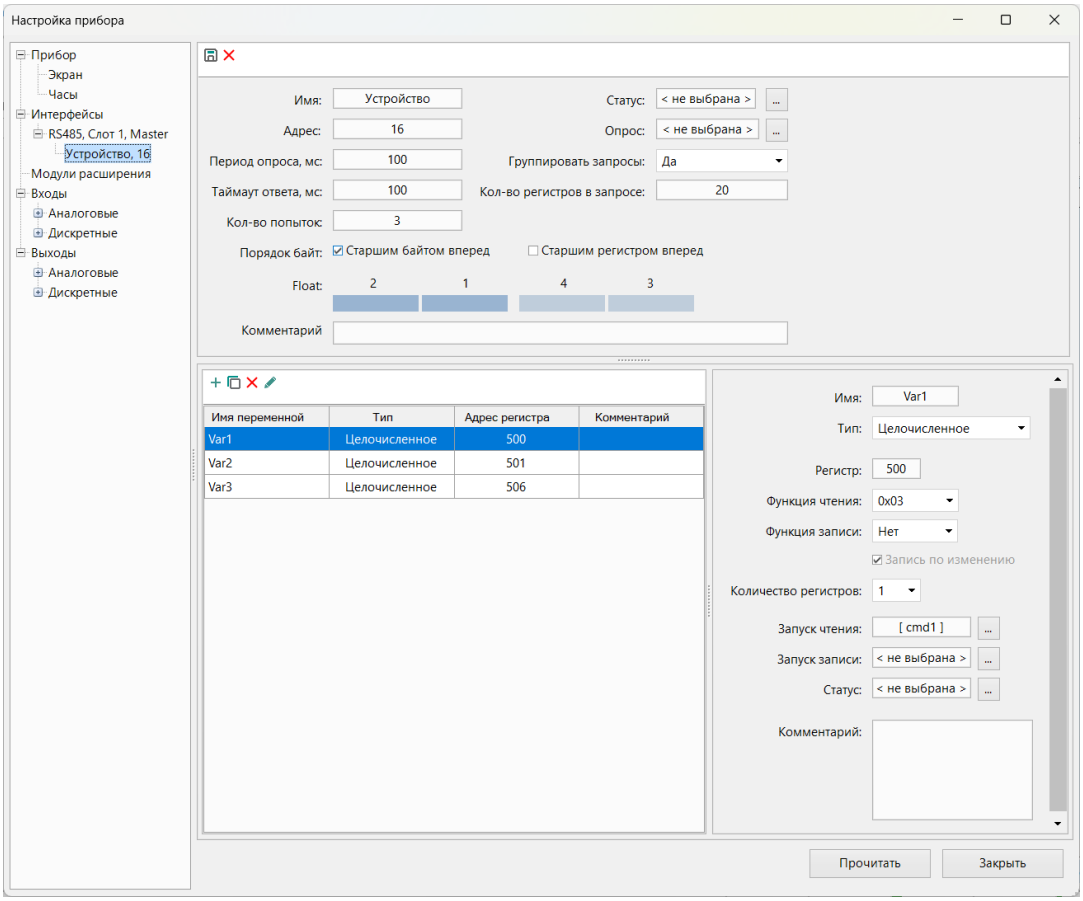

Рисунок Б.5 – Конфигурация переменных (сценарий 3)

На рисунке Б.6 показано формирование 2 групповых запросов для переменных Var-1 и Var-2 и 2 индивидуальных запросов для переменной Var-3.

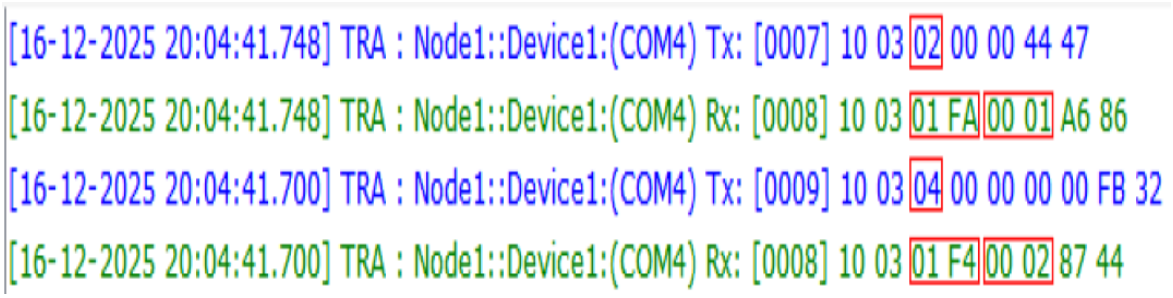

Рисунок Б.6 – Результат формирования запросов (сценарий 3)

Б.4 Итоговая таблица результатов

В таблице Б.3 представлены сводные результаты тестирования модуля группировки по всем сценариям.

Таблица Б.3 - Сводные результаты тестирования модуля группировки

№ сценария	Тип сценария	Кол-во переменных	Запросов (без группы)	Запросов (с группой)	Сокращение запросов	Ключевой вывод
1	Идеальная группировка	10	10	1	90%	Максимальная эффективность при полной совместимости
2	Группировка отключена	10	10	10	0%	Корректная работа при отключенной оптимизации
3	Смешанные условия	3	6	4	33%	Частичная оптимизация в реальных условиях