Тема: **Разработка модуля обработки групповых запросов Modbus в среде Owen Logic**

**СОДЕРЖАНИЕ**

[Список сокращений 3](#_Toc183186190)

[Введение 3](#_Toc183186191)

1 [Обзорная часть 3](#_Toc183186191)

[1.1 Подход к построению среды разработки 5](#_Toc183186191)

[1.1.1 Принципы DDD 5](#_Toc183186192)

[1.1.2 Структура и компоненты на основе DDD 6](#_Toc183186193)

[1.1.3 Преимущества применения DDD 7](#_Toc183186193)

[1.1.4 Применение DDD для расширяемости 7](#_Toc183186193)

1.[2 Анализ сред разработки 8](#_Toc183186196)

1.[2.1 Codesys 8](#_Toc183186197)

1.[2.2 TIA Portal (Siemens) 9](#_Toc183186198)

1.[2.3 Trace Mode 9](#_Toc183186199)

1.[2.4 Owen Logic 10](#_Toc183186199)

1.[2.5 Сравнительная таблица сред разработки 11](#_Toc183186199)

2 [Расчетно-конструкторская часть -](#_Toc183186191)

[2.1 Переменные в среде Owen Logic как основа для группировки -](#_Toc183186191)

[2.1.1 Классификация и типы данных переменных -](#_Toc183186192)

[2.1.2 Сетевые переменные и их роль в Modbus-обмене -](#_Toc183186192)

[2.2 Учет ограничений при группировке запросов -](#_Toc183186191)

[2.2.1 Ограничения протокола Modbus -](#_Toc183186192)

[2.2.2 Аппаратные и программные ограничения -](#_Toc183186192)

[2.2.2.1 Ограничения контроллера КС1 -](#_Toc105421953)

[2.2.2.2 Временные характеристики опроса -](#_Toc105421953)

[2.2.3 Ограничения среды Owen Logic -](#_Toc183186192)

[2.3 Функциональные требования к модулю -](#_Toc183186191)

[2.3.1 Основные задачи и цели разработки -](#_Toc183186192)

[2.3.2 Требования к группировке переменных -](#_Toc183186192)

[2.3.2.1 Функции и области данных -](#_Toc105421953)

[2.3.2.2 Основные фукнции протокола Modbus -](#_Toc105421953)

[2.3.2.3 Опрос Slave-устройства -](#_Toc105421953)

[2.3.2.4 Условия формирования групповых запросов -](#_Toc105421953)

[2.3.3 Пример формирования групповых запросов -](#_Toc183186192)

[2.3.4 Требования к пользовательскому интерфейсу -](#_Toc183186192)

[2.4 Архитектура модуля обработки групповых запросов -](#_Toc183186191)

[2.4.1 Диаграмма компонентов -](#_Toc183186192)

[2.4.2 Основные компоненты и их ответственность -](#_Toc183186192)

[2.5 Алгоритмы работы модуля -](#_Toc183186191)

[2.5.1 Алгоритм группировки переменных -](#_Toc183186192)

[2.5.1.1 Общие принципы работы -](#_Toc105421953)

[2.5.1.2 Пошаговое описание алгоритма -](#_Toc105421953)

[2.5.2 Алгоритм проверки совместимости переменных -](#_Toc183186192)

[2.5.2.1 Общие принципы работы -](#_Toc105421953)

[2.5.2.2 Пошаговое описание алгоритма -](#_Toc105421953)

[2.5.3 Алгоритм сортировки и упорядочивания переменных -](#_Toc183186192)

[2.5.3.1 Общие принципы работы -](#_Toc105421953)

[2.5.3.2 Пошаговое описание алгоритма -](#_Toc105421953)

[2.5.4 Алгоритм слияния совместных групп -](#_Toc183186192)

[2.5.4.1 Общие принципы работы -](#_Toc105421953)

[2.5.4.2 Пошаговое описание алгоритма -](#_Toc105421953)

[2.6 Обработка ограничений протоколов Modbus -](#_Toc183186191)

[2.6.1 Ограничения для протоколов RTU и ASCII -](#_Toc183186192)

[2.6.2 Расчет максимального размера запроса -](#_Toc183186192)

[2.7 Анализ эффективности группировки запросов -](#_Toc183186191)

[2.7.1 Теоретический потенциал группировки -](#_Toc183186192)

[2.7.2 Практические результаты тестовых данных -](#_Toc183186192)

[2.6 Разработка тестовых сценариев -](#_Toc183186191)

3 [Экспериментальная часть -](#_Toc183186191)

[3.1 Реализация логики группировки -](#_Toc183186191)

[3.1.1 Класс ModbusRequestInfo -](#_Toc183186192)

[3.1.2 Класс CreateModbusRequestServicel -](#_Toc183186192)

[3.1.3 Класс ModbusRequestGroupBuilder -](#_Toc183186192)

[3.1.4 Структура ModbusRequestGroup -](#_Toc183186192)

[3.2 Реализация обработки ограничений протокола -](#_Toc183186191)

[3.2.1 Интерфейс IModbusRegisterLimitByProtocol -](#_Toc183186192)

[3.2.2 Реализация ограничения ModbusRegisterLimitByRtuProtocol -](#_Toc183186192)

[3.2.2 Реализация ограничения ModbusRegisterLimitByAsciiProtocol -](#_Toc183186192)

[3.3 Интеграция с пользовательским интерфейсом -](#_Toc183186191)

[3.3.1 Взаимодействие с GlobalVariableDictionary и IGeneratorFacade -](#_Toc183186192)

[3.3.2 Механизм анализа и сборки бинарных структур Bingo -](#_Toc183186192)

[3.4 Валидация запросов и проверка корректности переменных -](#_Toc183186191)

[3.4.1 Валидатор регистров Rs485RegistersPerRequestCountValidator -](#_Toc183186192)

[3.4.2 Цепочка валидаторов ValidatorChainProxy/ValidatorChainBaseProxy -](#_Toc183186192)

[3.4.3 Презентер устройства Rs485DevicePresenter -](#_Toc183186192)

4 [Тестирование модуля -](#_Toc183186191)

[4.1 Тестирование алгоритмов группировки -](#_Toc183186191)

[4.1.1 Тестирование сортировки и упорядочивания переменных -](#_Toc183186192)

[4.1.2 Тестирование обработки ограничений количества регистров -](#_Toc183186192)

[4.1.3 Тестирование группировки по последовательности адресов -](#_Toc183186192)

[4.2 Тестирование группировки по типам данных -](#_Toc183186191)

[4.2.1 Тестирование группировки различных типов переменных -](#_Toc183186192)

[4.2.2 Тестирование группировки битовых переменных -](#_Toc183186192)

[4.3 Тестирование группировки по функциям Modbus -](#_Toc183186191)

[4.3.1 Тестирование группировки по функциям чтения/записи -](#_Toc183186192)

[4.3.2 Тестирование группировки с учетом протоколов RTU/ASCII -](#_Toc183186192)

[4.4 Тестирование расширенных сценариев группировки -](#_Toc183186191)

[4.4.1 Тестирование группировки по статусу изменения переменных -](#_Toc183186192)

[4.4.2 Тестирование группировки по дескрипторам статуса и команд -](#_Toc183186192)

[4.5 Тестирование обработки конфликтов -](#_Toc183186191)

[4.5.1 Тестирование обработки конфликтов порядка переменных -](#_Toc183186192)

[4.5.2 Тестирование обработки переменных с одинаковыми адресами -](#_Toc183186192)

[4.6 Тестирование валидаторов RS-485 -](#_Toc183186191)

[4.6.1 Тестирование валидатора количества регистров в запросе -](#_Toc183186192)

[Заключение -](#_Toc183186191)

[Список использованных источников -](#_Toc183186191)

[ПРИЛОЖЕНИЕ А. Исходный код модуля группового опроса Modbus-устройств -](#_Toc183186191)

[ПРИЛОЖЕНИЕ Б. Руководство пользователя модуля группового опроса Modbus -](#_Toc183186191)

**Список сокращений**

**OL** - **OWEN Logic**

**FBD** - Function Block Diagram

**ST** - Structured Text

**DDD –** Domain-Driven Design

**ПЛК** - Программируемый Логический Контроллер

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](https://owen.ru/documents)], 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**

# **Подход к построению срeды 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**: Всё программное обеспечение строится с учётом требований и особенностей бизнес-логики, связанной с автоматизацией технологических процессов. Все компоненты системы проектируются в тесном взаимодействии с экспертами в предметной области, чтобы учесть все аспекты работы с программируемыми реле и панелями компании «ОВЕН».
* **Единый язык (Ubiquitous Language)**: В проектировании системы важно использовать общий язык, понятный как техническим специалистам, так и экспертам в области автоматизации. Это означает, что терминология и концепции, используемые в коде, документации и при общении с пользователями, должны быть одинаковыми, чтобы избежать недоразумений и упростить взаимодействие между всеми участниками разработки.
* **Разделение на контексты (Bounded Contexts)**: Система разделяется на различные **контексты**, каждый из которых охватывает определённую часть бизнес-логики. Это позволяет избежать излишней сложности и обеспечивать независимость различных компонентов. Например, контекст для работы с алгоритмами управления может быть отделён от контекста работы с базами данных или внешними системами. Это помогает лучше управлять зависимостями и облегчает 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)**: Этот слой включает в себя бизнес-логику и модели, которые отвечают за работу с программируемыми реле и панелями. В нем определяются основные сущности, такие как алгоритмы управления, параметры устройств и другие элементы.
* **Слой приложения (Application Layer)**: Этот слой взаимодействует с пользователем или другими приложениями, предоставляя интерфейсы для работы с бизнес-логикой. В нем находятся сервисы, которые управляют запросами, получают данные и передают их в слой домена
* **Инфраструктурный слой (Infrastructure Layer)**: Этот слой отвечает 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)**: Это взаимодействие с пользователем, другими приложениями или внешними сервисами, например, через графический интерфейс или командную строку.

Схематическое представление архитектуры DDD, представлено на рисунке 1.1.

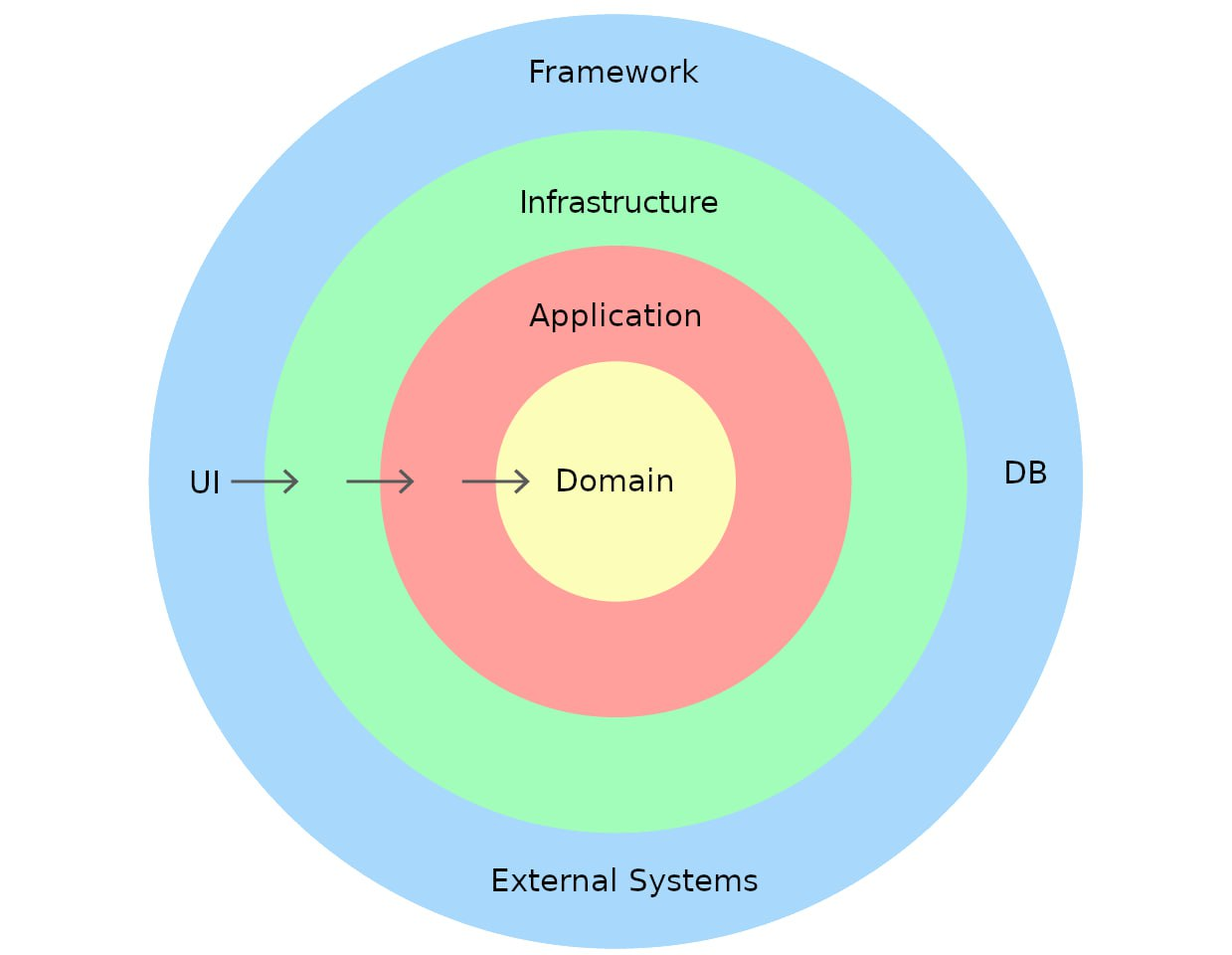


Рисунок 1.1 – Схематическое представление архитектуры DDD

### 1.1.3 Преимущества применения DDD

Использование **DDD** в **OL** позволяет достичь нескольких ключевых целей:

* **Модульность и независимость компонентов**: Разделение системы на контексты и использование чётких интерфейсов между ними позволяет создавать независимые компоненты, которые можно легко изменять, заменять или масштабировать.
* **Гибкость и масштабируемость**: Архитектура, основанная на DDD, позволяет гибко адаптировать систему под изменяющиеся требования. Это особенно важно для среды программирования, которая должна поддерживать разнообразие задач, связанных с управлением технологическими процессами.
* **Упрощение взаимодействия между разработчиками и бизнес-экспертами**: Единый язык, используемый в DDD, способствует лучшему пониманию предметной области и уменьшению недопонимания между участниками разработки. Это позволяет быстрее разрабатывать и внедрять новые функции и решать возникающие проблемы.

### 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.

# **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.

### 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.

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 компанийи.

### 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, что соответствует стандарту IEC 61131-3.
* **Интуитивно понятный интерфейс:** Удобный графический интерфейс для создания и отладки программ.
* **Возможности расширения:** Программа позволяет создавать собственные функциональные блоки (макросы) и использовать их в других проектах.
* **Бесплатное использование:** Доступна бесплатно для пользователей, что делает её привлекательной для небольших и средних предприятий.
* **Интеграция с системами управления и мониторинга:** Возможность интеграции с внешними SCADA-системами для визуализации.

**Недостатки:**

* **Ограниченная совместимость:** Программное обеспечение предназначено исключительно для работы с оборудованием компании «ОВЕН».
* **Ограниченные возможности визуализации:** Отсутствие 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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** – используются в алгоритмах управления и визуализации.
* **Сервисные переменные** – обеспечивают внутренние функции прибора и не изменяются пользователем.
* **Сетевые переменные** – применяются для обмена данными между устройствами через интерфейсы связи (например, RS-485, Ethernet, Modbus).

**Каждая переменная имеет следующие основные параметры**:

* **Имя переменной** – уникальный идентификатор для отображения на схеме проекта.
* **Тип переменной** – определяет формат данных (булевский, целочисленный, с плавающей запятой).
* **Энергонезависимость** – при включении параметра значение сохраняется в энергонезависимой памяти (ПЗУ) устройства.
* **Значение по умолчанию** – используется при инициализации или при отсутствии связи с устройством.
* **Использование в проекте** – отображает, привязана ли переменная к блокам схемы.
* **Комментарий** – описание назначения переменной для удобства разработки.

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

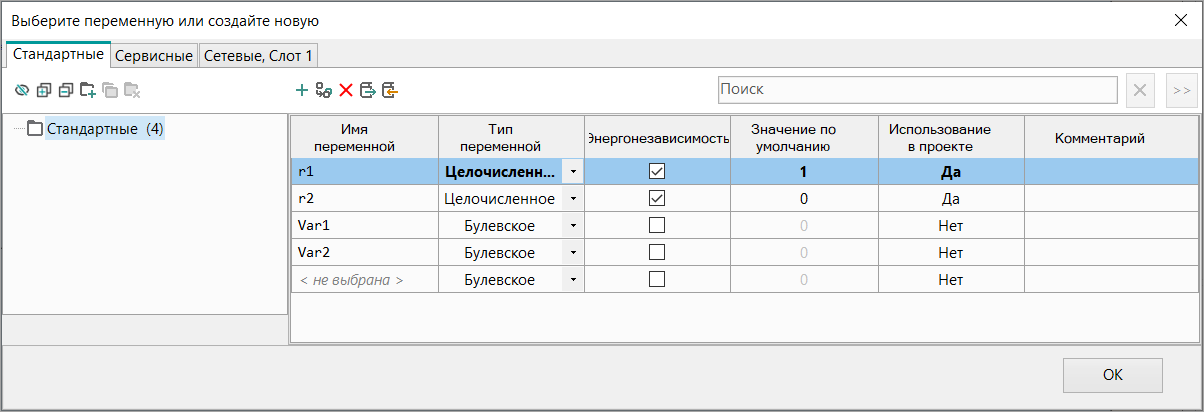
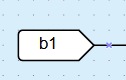


Рисунок 2.1 – Таблица переменных в среде **OL**

**Типы переменных**:

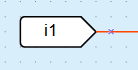
* **Булевский (Bool)** – принимает значения 0 (False) или 1 (True). Используется для логических условий и дискретных сигналов.

На рисунке **2.2** приведен пример отображения булевских переменных на схеме проекта.

  
Рисунок 2.2 – Булевские переменные в **OL**

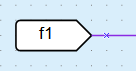
* **Целочисленный (Int/UInt)** – хранит значения целых чисел. Применяется для счетчиков, адресов, параметров, не требующих дробной части.

На рисунке **2.3** показан пример использования целочисленных переменных.

  
Рисунок 2.3 – Целочисленные переменные в **OL**

* **С плавающей запятой (Float/Real)** – хранит вещественные значения, используется для аналоговых сигналов, коэффициентов и вычислений.

На рисунке **2.4** представлен пример отображения переменных с плавающей запятой.

  
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-обменепе

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).

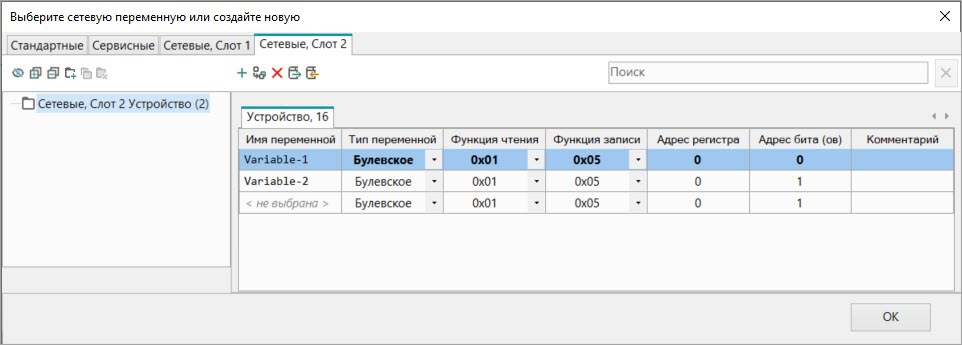


Рисунок 2.5 – Вкладка сетевых переменных в режиме Master

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

* **Тип регистра** – определяет область памяти Slave-устройства (Coil, Discrete Input, Holding Register, Input Register).
* **Адрес регистра** – определяет позицию данных в адресном пространстве устройства.
* **Функция чтения/записи** – определяет операцию, выполняемую с регистром (0x03, 0x04, 0x10 и другие).
* **Направление обмена** – указывает, 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.

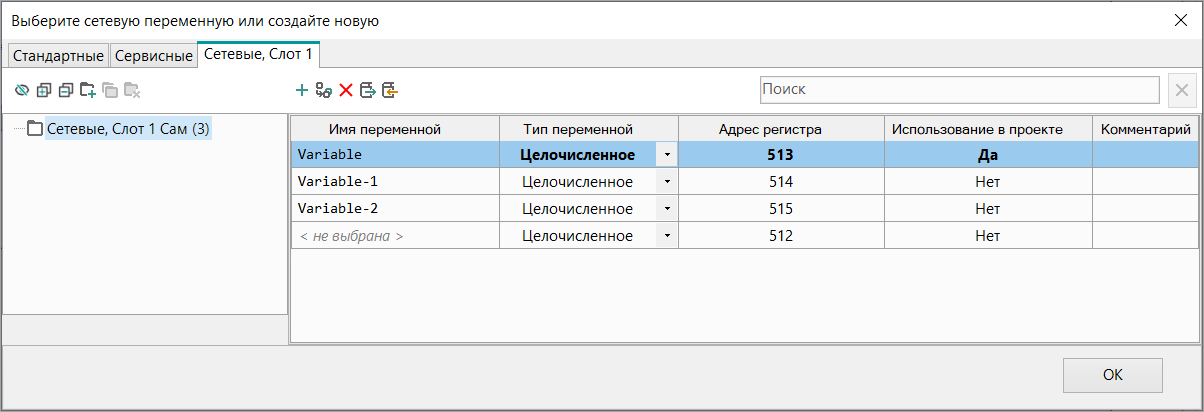


Рисунок 2.6 – Вкладка сетевых переменных в режиме Slave

### 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 Modbus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ 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** — 256 байт на кадр
* **Максимальная скорость обмена по интерфейсу RS-485** — 115200 бит/с
* **Ограничение по времени обработки прерываний** — не более 12 мс между запросами
* **Объем оперативной памяти для хранения очереди запросов** — 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.

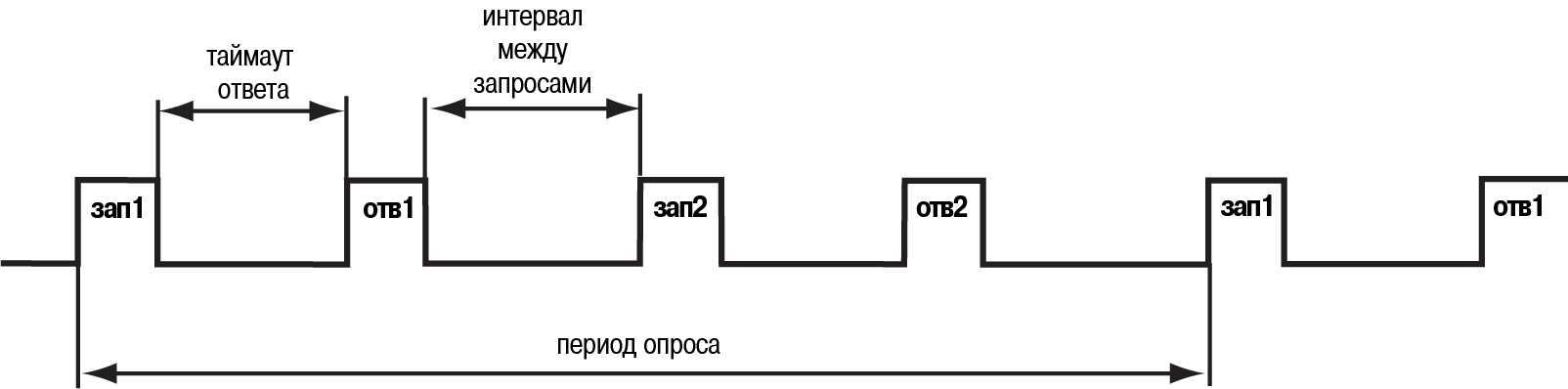


Рисунок 2.7 – Временная диаграмма опроса устройства по протоколу Modbus

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.

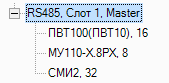


Рисунок 2.8 – Последовательность опроса устройств по Modbus

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 Modbus 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/предупреждений 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, что бы | 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).

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.

### 2.3.2.1 Основные функции протокола Modbus

**Функция** определяет операцию (чтение/запись) и область памяти, с которой эта операция будет произведена.

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:

* Одиночным способом – каждая переменная считывается отдельной командой
* Групповым способом – 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.**  
   Переменные типов Real и Int32 могут объединяться в один групповой запрос, так как оба типа 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](https://confluence.owen.ru/pages/viewpage.action?pageId=27624886#id-%D0%93%D1%80%D1%83%D0%BF%D0%BF%D0%BE%D0%B2%D0%BE%D0%B9%D0%BE%D0%BF%D1%80%D0%BE%D1%81%D0%9A%D0%A11-%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B01). 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 |

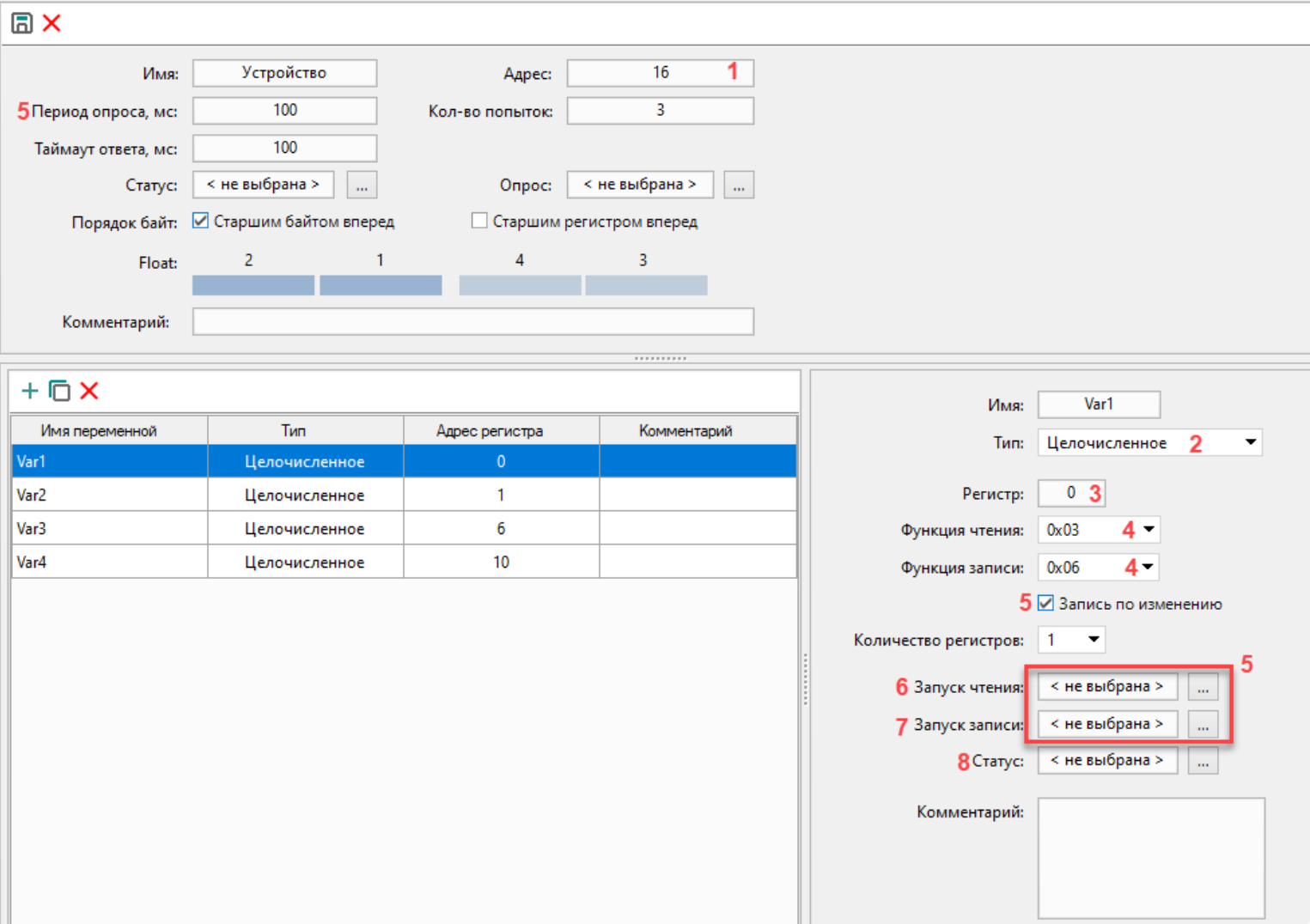


Рисунок 2.9 – Окно настроек опрашиваемого устройства

### 2.3.3 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.

**Входные данные**

Исходные параметры переменных представлены в таблице 2.6.

Таблица 2.6 - Исходные параметры переменных для примера группировки

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Переменная** | **Тип** | **Адрес**  **переменной** | **Функция** | **Период** | **По команде** |
| Var-1 | int  1 регистр | 500 | 0х03 | 100 мс | Адрес переменной  типа bool: 512 |
| Var-2 | int  1 регистр | 501 | 0х03 | 100 мс | Адрес переменной  типа bool: 512 |
| Var-3 | int  1 регистр | 506 | 0х04 | 50 мс | Адрес переменной  типа bool: 600 |

**Дополнительное условие**: Максимальное количество регистров в запросе = 16

**Алгоритм обработки переменных**

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 0х03 - 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 объеденять 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 2.10 представлена диаграмма формирования групповых запросов для данного примера.

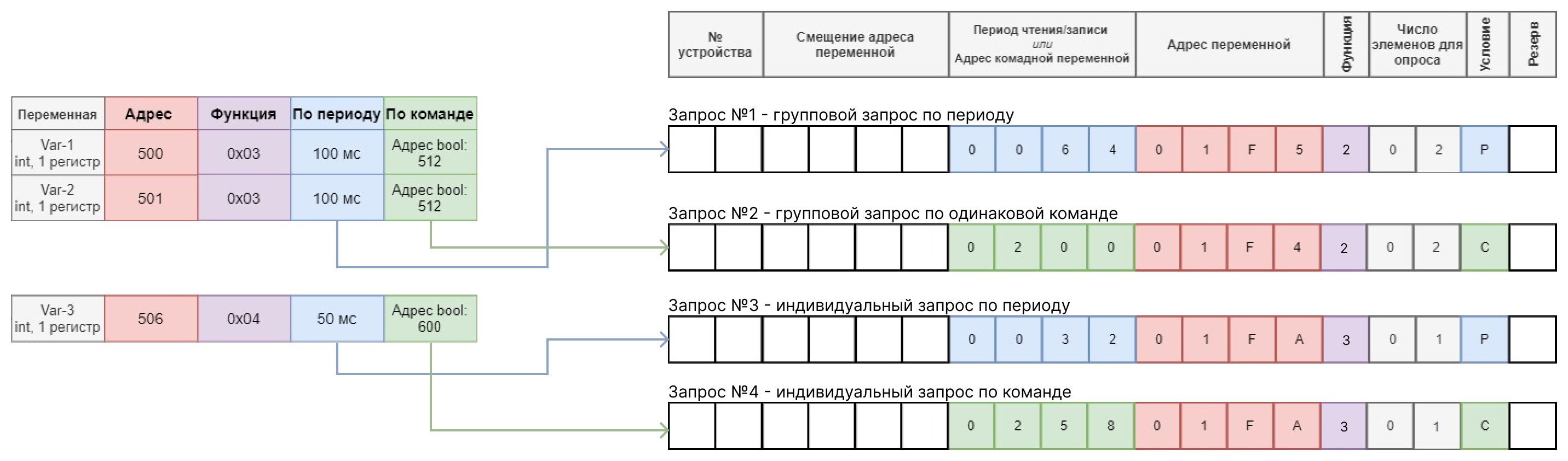


Рисунок 2.10 – Диаграмма формирования групповых запросов для переменных

**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 2.11) показывает снижение количества запросов на 33%

Рисунок 2.11 – 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.

### 2.3.4 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. 2.8).

В окно должны быть добавлены следующие элементы:

1. **Параметр «Протокол»**  
   Тип элемента: выпадающий список.  
   Возможные значения: **Modbus RTU / Modbus ASCII**.  
   При выборе протокола выполняется динамическое обновление допустимого диапазона параметра *«Количество регистров в запросе»* в соответствии с ограничениями выбранного протокола.  
   Значение по умолчанию: **Modbus RTU** (рис. 2.9).
2. **Параметр «Группировать запросы»**  
   Тип элемента: выпадающий список (Да / Нет).  
   Значение по умолчанию: **«Нет»**.  
   При выборе значения **«Да»** активируется возможность задания дополнительных параметров группировки (рис. 2.10).  
   Если значение параметра установлено в **«Нет»**, параметр *«Количество регистров в запросе»* становится **недоступным для редактирования** (рис. 2.11).
3. **Параметр «Количество регистров в запросе»**  
   Диапазон допустимых значений:  
   – для протокола Modbus RTU – от 1 до 125;  
   – для протокола Modbus ASCII – от 1 до 61.  
   Значение по умолчанию: 16.  
   При вводе некорректного значения отображается пиктограмма с восклицательным знаком и тултип с сообщением о допустимом диапазоне значений:  
   – сообщение о валидном диапазоне количества регистров в групповом запросе для протокола RTU (рис. 2.12).

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).

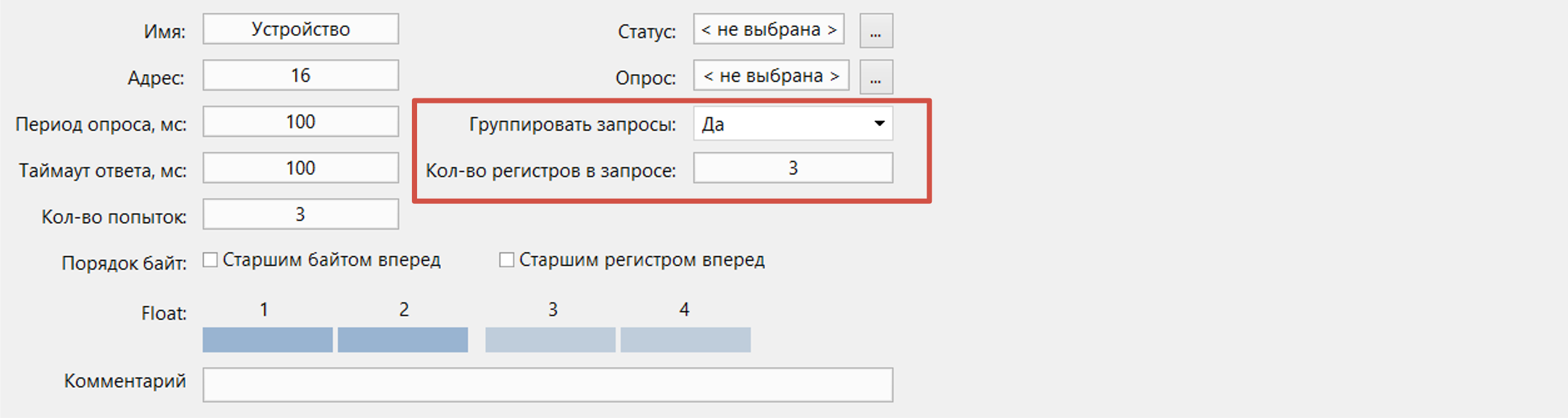
****

Рисунок 2.8 – Окно настроек опрашиваемого устройства с добавленными параметрами

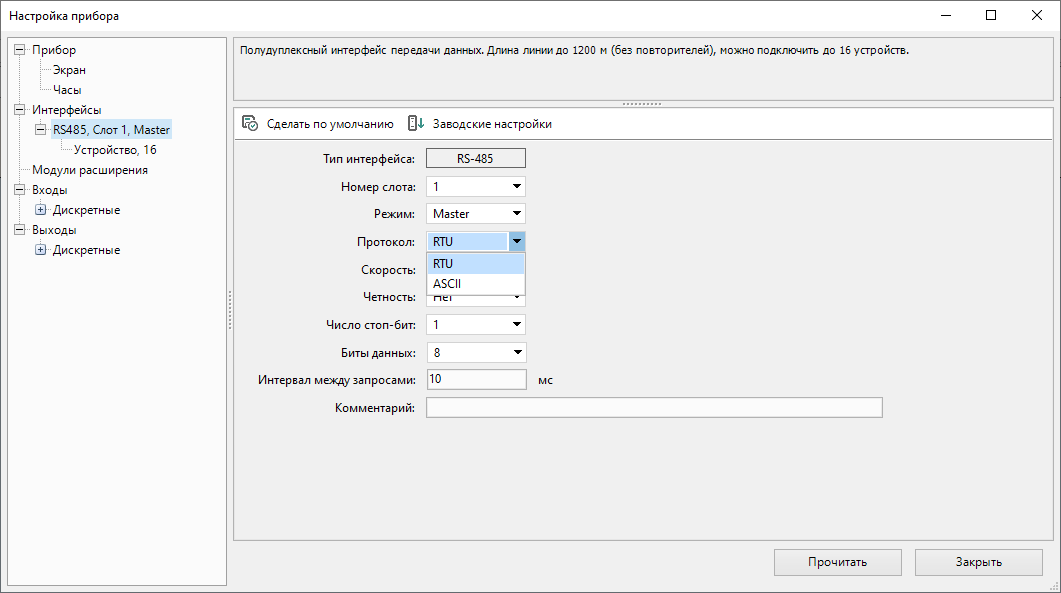


Рисунок 2.9 – Выбор протокола Modbus RTU/ASCII

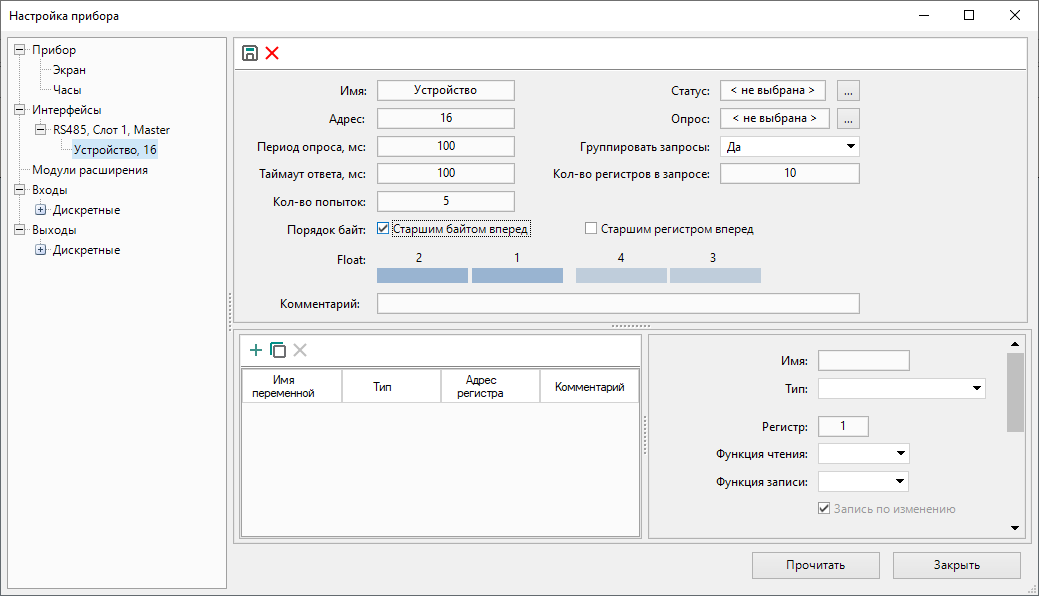


Рисунок 2.10 – Параметр «Группировать запросы»

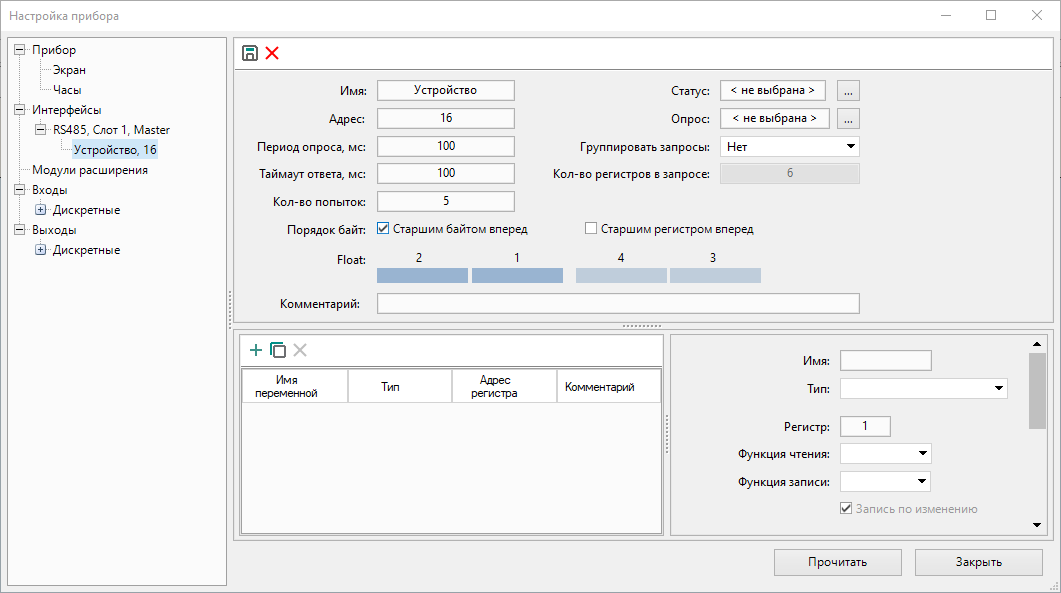


Рисунок 2.11 – Неактивное состояние параметра «Количество регистров в запросе» при значении «Группировать запросы = Нет»

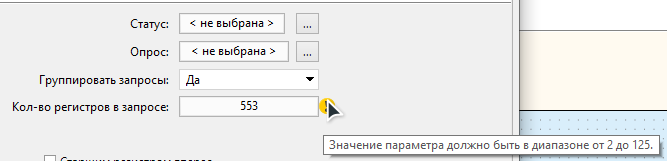


Рисунок 2.12 – Сообщение о недопустимом диапазоне значений для протокола Modbus

### 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 2.13 представлена UML-диаграмма классов, отражающая основные связи и зависимости внутри модуля.

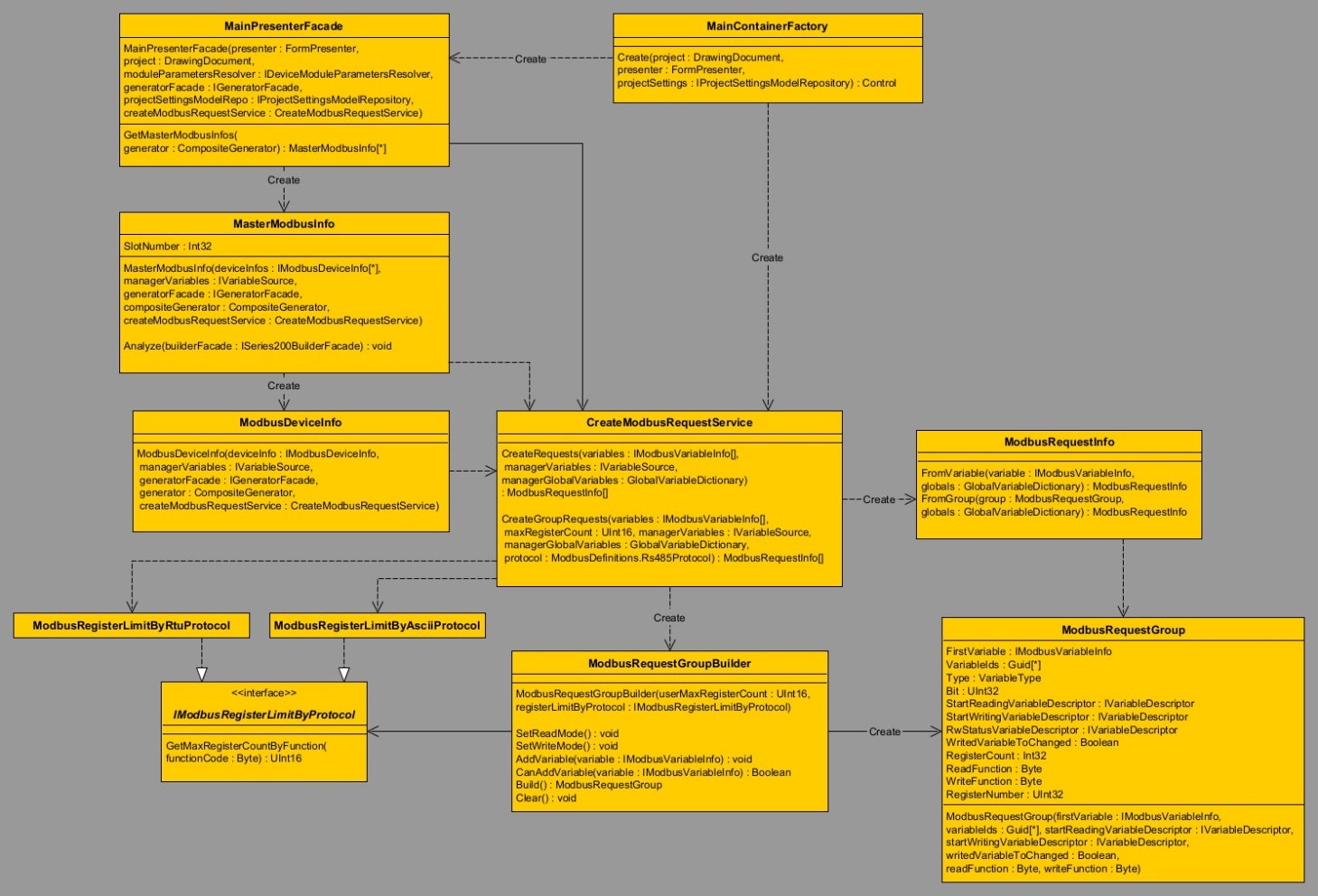


Рисунок 2.13 – UML-диаграмма классов

### 2.4.2 Основные компоненты и их ответственность

Для понимания структуры разрабатываемого модуля в таблице 2.7 приведены основные компоненты и их функциональное назначение.

Таблица 2.7 – Основные компоненты и их ответственность

|  |  |
| --- | --- |
| **Компонент** | **Назначение** |
| ModbusRequestGroup | Представляет агрегированную группу переменных, объединённых в один Modbus-запрос. Хранит параметры, описывающие первую переменную, список идентификаторов, тип данных, функции чтения/записи. |
| ModbusRequestGroupBuilder | Реализует пошаговую сборку группы Modbus-запроса. Отвечает за проверку совместимости переменных, соблюдение ограничений протокола и создание экземпляров ModbusRequestGroup. |
| CreateModbusRequestService | Управляет формированием всех запросов. Делит переменные на группы чтения и записи, сортирует, объединяет совместимые группы и создаёт итоговые структуры ModbusRequestInfo. |
| IModbusRegisterLimitByProtocol | Определяет интерфейс для получения предельного числа регистров в зависимости от используемого протокола (RTU или ASCII). |
| ModbusRegisterLimitByRtuProtocol | Реализует ограничения количества регистров для протокола Modbus RTU в зависимости от кода функции. |
| ModbusRegisterLimitByAsciiProtocol | Реализует ограничения количества регистров для протокола Modbus ASCII в зависимости от кода функции. |
| ModbusRequestInfo | Итоговая структура запроса, готовая к сериализации в бинарный формат Modbus. |
| ModbusDeviceInfo | Инкапсулирует данные одного Modbus-устройства. Хранит список сформированных запросов, переменные разрешения и статуса, обеспечивает их анализ и сборку бинарных данных для программы. |
| MasterModbusInfo | Представляет Master-устройство с набором Modbus-устройств. Формирует и хранит коллекцию ModbusDeviceInfo, обеспечивает анализ и сборку данных для передачи. |
| MainPresenterFacade | 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. |
| MainContainerFactory | 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 Owen 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**  
   Перед началом работы алгоритм переводится в один из режимов:
   * **режим чтения (SetReadMode)** — для формирования групп запросов чтения;
   * **режим записи (SetWriteMode)** — для формирования групп запросов записи.  
     В зависимости от выбранного режима задаются различные правила совместимости переменных.
2. **Инициализация состояния группы**  
   При добавлении первой переменной создаётся новая группа, в которой запоминаются:
   * функция Modbus (readFunction или writeFunction);
   * стартовый дескриптор (чтения или записи);
   * идентификатор первой переменной;
   * текущее количество регистров и начальный адрес.
3. **Проверка возможности добавления переменной**  
   Для каждой следующей переменной вызывается метод CanAddVariable(), который определяет, может ли она быть добавлена в текущую группу.  
   Переменная может быть объединена с предыдущими, если выполняются условия:
   * не превышено ограничение по количеству регистров (учитываются пользовательский лимит и ограничения протокола RTU/ASCII);
   * адреса регистров следуют последовательно без пропусков;
   * совпадают функции Modbus (чтения или записи);
   * совпадают типы данных переменных или их допустимые комбинации (например, Float и Long);
   * совпадают дескрипторы состояния и признаков «запись по изменению».

Если хотя бы одно условие нарушается — текущая группа считается завершённой, создаётся новый запрос, и начинается формирование новой группы.

1. **Формирование запроса**  
   Когда текущая группа готова, вызывается метод Build(), создающий объект ModbusRequestGroup, содержащий:
   * первую переменную группы;
   * список идентификаторов всех переменных в группе;
   * функции Modbus для чтения и записи;
   * диапазон регистров (начальный адрес и количество);
   * признаки дескрипторов и тип данных.  
     После этого состояние строителя очищается методом Clear() для подготовки к созданию следующей группы.
2. **Завершение и объединение совместимых групп**  
   После построения всех групп запросов чтения и записи производится дополнительная обработка, выполняемая сервисом CreateModbusRequestService.  
   Если для одной и той же переменной были сформированы отдельные группы чтения и записи, они объединяются в один запрос (при совпадении 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 переменная совместима с текущей группой — добавить в группу;
      * иначе — сохранить текущую 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 ModbusRequestInfo 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 ModbusRequestGroupBuilder (метод CanAddVariable()). Алгоритм оценивает ряд параметров, которые должны совпадать для корректного объединения переменных в общий запрос.

Основные критерии совместимости:

1. **Количество регистров.**  
   Общее число регистров в группе не должно превышать лимиты протокола (ModbusRegisterLimitByRtuProtocol, ModbusRegisterLimitByAsciiProtocol) и пользовательские настройки.
2. **Последовательность адресов.**  
   Регистры переменных должны идти подряд, без пропусков.
3. **Дескрипторы состояния.**  
   Переменные должны иметь одинаковый дескриптор состояния (RwStatusVariableDescriptor).
4. **Тип данных.**  
   Допускаются одинаковые типы или совместимые пары (например, Float и Long).
5. **Функция Modbus и режим.**  
   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 CreateModbusRequestService.

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

Основные критерии сортировки:

1. **Функция Modbus.**  
   Сначала группируются переменные с одинаковыми кодами функций (например, 03, 04, 16).
2. **Адрес регистра.**  
   Далее сортировка идёт по возрастанию адресов.
3. **Битовое смещение.**  
   Для битовых переменных дополнительно учитывается значение поля Bit.

После сортировки каждой переменной присваивается порядковый номер (Order) для контроля последовательности при формировании групп.

### 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 CreateModbusRequestService. 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:
   * если есть две группы (чтение и запись) — объединить их;
   * если одна — оставить без изменений.
4. Вернуть список объединённых групп для формирования объектов ModbusRequestInfo.

# **3 Экспериментальная часть**

### 3.1.4 Структура ModbusRequestGroup

Класс ModbusRequestGroup 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 3.1.4 – Класс ModbusRequestGroup

**public** **sealed** class ModbusRequestGroup **:** ValueObject**<**ModbusRequestGroup**>**

**{**

/// <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** ModbusRequestGroup**(**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** - базовая переменная, определяющая параметры группы
* **VariableIds** - идентификаторы всех переменных, объединенных в запрос
* **Type, Bit** - тип данных и битовое смещение (наследуются от первой переменной)
* **RegisterNumber, RegisterCount** - начальный адрес и количество регистров
* **ReadFunction, WriteFunction** - коды Modbus-функций для чтения и записи
* **StartReadingVariableDescriptor, StartWritingVariableDescriptor** - дескрипторы для управления опросом

Обоснование архитектурных решений:

1. **Value Object паттерн** - обеспечивает иммутабельность и сравнение по значению
2. **Композиция на основе FirstVariable** - уменьшает дублирование данных
3. **IReadOnlyList для VariableIds** - гарантирует неизменяемость коллекции
4. Полная инкапсуляция - 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 IModbusRegisterLimitByProtocol

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

Листинг 3.2.1 – Интерфейс IModbusRegisterLimitByProtocol

/// <summary>

/// Определяет контракт для получения ограничения количества регистров

/// в зависимости от протокола передачи данных Modbus.

/// </summary>

**internal** **interface** IModbusRegisterLimitByProtocol

**{**

/// <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 ModbusRegisterLimitByRtuProtocol

Класс ModbusRegisterLimitByRtuProtocol 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 3.2.2 – Класс ModbusRegisterLimitByRtuProtocol

/// <summary>

/// Реализует ограничения количества регистров для протокола Modbus RTU.

/// </summary>

**internal** **sealed** class ModbusRegisterLimitByRtuProtocol **:** IModbusRegisterLimitByProtocol

**{**

**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 ModbusRegisterLimitByAsciiProtocol

Класс ModbusRegisterLimitByAsciiProtocol 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 3.2.2 – Класс ModbusRegisterLimitByAsciiProtocol

/// <summary>

/// Реализует ограничения количества регистров для протокола Modbus ASCII.

/// </summary>

**internal** **sealed** class ModbusRegisterLimitByAsciiProtocol **:** IModbusRegisterLimitByProtocol

**{**

**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.

### 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 обработки 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 обработки 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 невалидными

• Некорректные типы данных (строки, дробные числа) отклоняются

• Null-значения вызывают исключение

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

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] URL: <https://owen.ru/documents>

[2] URL: https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/

[3] URL: <https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice>

[4] URL: https://www.codesys.com/

[5] URL: <https://www.siemens-pro.ru/soft/tia-portal.html>

[6] URL: https://ru.wikipedia.org/wiki/Trace\_mode