ЦИФРОВЫЕ РЕШЕНИЯ

ALTA IDE (версия 1.0)

Руководство пользователя

12.2025
версия 1.0

Содержание

1 О программе

ALTA IDE – это программное обеспечение, которое предоставляет комплекс инструментов для создания, редактирования, отладки и запуска приложений на устройстве (программируемом контроллере - далее ПЛК).

ALTA IDE используется для:

1. Создания и редактирования приложений пользователя на языке ST стандарта МЭК 61131-3.
2. Отладки кода.
3. Загрузки приложения на ПЛК.
4. Мониторинга работы приложения.

Основные принципы ALTA IDE

Нулевой неиспользуемый функционал

- Приложение пользователя собирается из таргета под конкретный прибор и проект.
- Новый функционал приложения наращивается без вмешательства в базовый таргет прибора.

Бизнес-логика отделена от прибора

- Взаимодействие приложения с прибором осуществляется через библиотеки HSAL.
- Исходный код приложения является аппаратно/программно независимым.

Единый инструмент для всех

- Таргет и код пользователя пишется на одном языке IEC.

Нулевой паразитный обмен данными

- Весь обмен данными с пользовательским кодом управляется пользовательским кодом.
- Неиспользуемые данные не опрашиваются.
- Опрос используемых каналов I/O происходит с требуемым темпом.

Удобство разработки

ALTA IDE поддерживает современные средства, упрощающие разработку:

- подсветка синтаксиса;
- автодополнение кода;
- светлая и темная тема интерфейса (в ALTA IDE версии 1.0.0 только темная);
- перекрестные ссылки для вызываемых объектов.

Перечень приборов, для программирования которых может использоваться ALTA IDE, представлен на сайте компании Овен Цифровые решения.

1.1 Системные требования для работы ALTA IDE

Для корректной установки и работы программного обеспечения необходимо соответствие минимальным системным требованиям:

Операционная система:

- Windows 10;
- Linux Ubuntu.20.04;
- или аналогичные версии операционных систем.

Процессор:

- частота центрального процессора 1 ГГц или выше.

Оперативная память (ОЗУ):

- не менее 8 ГБ.

Место на жестком диске:

- не менее 5 ГБ свободного пространства.

Видеоадаптер:

- поддержка DirectX 9 или более поздней версии.

Дисплей:

- разрешение экрана не ниже 800 × 600.

1.2 Используемые аббревиатуры и термины

Drag & drop — механизм перемещения элементов интерфейса, который позволяет перетаскивать нужный элемент, удерживая его кнопкой мыши, из одной области в другую.

Modbus — открытый коммуникационный протокол, основанный на архитектуре ведущий (Master) - ведомый (Slave).

OwenCloud — облачный сервис компании «Овен Цифровые решения», применяемый для удаленного мониторинга, управления и хранения архивов данных приборов, используемых в системах автоматизации. Доступ к сервису осуществляется с помощью web-браузера или мобильного приложения.

Owen Configurator — ПО для настройки и задачи параметров устройствам компании «Овен Цифровые решения».

Persistent — энергонезависимые переменные, которые сохраняют свои значения при загрузке нового приложения на устройство.

POU (Program Organization Units) — программные объекты такие, как функция, функциональный блок и программа.

Retain — энергонезависимые переменные, которые сохраняют свои значения при отключении питания контроллера.

ST (Structured Text) — язык программирования стандарта МЭК 61131-3. Предназначен для программирования промышленных контроллеров и операторских станций.

Битовая маска — это представление битовой строки в виде набора отдельных битов, с которыми можно работать по отдельности.

Глобальные переменные — переменные, константы, внешние или остаточные переменные (Retain и Persistent), которые доступны в рамках всего проекта.

Задача — элемент управления, который позволяет выполнять один или несколько программных объектов на периодической или событийной основе.

Исполняемое приложение — исполняемый файл, содержащий программу или набор инструкций, для записи в прибор и последующего запуска. Результат сборки пользовательского проекта ALTA.

Контекстное меню — элемент графического интерфейса, представляющий собой список команд, вызываемый пользователем для выбора необходимого действия над выбранным объектом. В ALTA IDE вызывается нажатием ПКМ по объекту.

ЛКМ — левая кнопка мыши.

ОЗУ (Оперативное Запоминающее Устройство) — энергозависимая часть памяти прибора, в которой во время работы хранится выполняемый код программы, а также входные, выходные и промежуточные данные, обрабатываемые процессором.

Переменная — поименованная область памяти процесса операционной системы, имя которой можно использовать для осуществления доступа к данным. Может принимать различные значения, но в каждый момент времени только одно.

ПЗУ (Постоянное Запоминающее Устройство) — энергонезависимая память, которая используется для хранения массива неизменяемых данных.

ПК — персональный компьютер.

ПКМ — правая кнопка мыши.

Плагин — динамически подключаемый модуль ALTA IDE содержит информацию о типе компонента и способе отображения компонента в IDE.

ПЛК — программируемый логический контроллер.

ПО — программное обеспечение.

Преобразователь — устройство, через которое прибор подключается к ПК.

Прибор (устройство) — программируемое устройство, например ПЛК110.

Программа — структурированный код, который при выполнении выдает одно или несколько значений. Значения остаются неизменными после выполнения программы и до следующего выполнения. Может содержать функции, функциональные блоки или другие элементы языка.

Программные блоки (объекты) — функция, функциональный блок, программа.

Проект — разработанный пользователем алгоритм работы, созданный в ALTA IDE, включающий в себя комплекс программ и настроек для последующего хранения на ПК.

Сборка проекта — команда, после которой будет выполнено преобразование файлов исходного кода в набор программных артефактов, которые могут быть запущены на приборе.

Сброс — команда переинициализации всех переменных проекта кроме энергонезависимых. После выполнения переменные принимают указанные при их объявлении начальные значения. Если начальные значения не указаны – используются значения по умолчанию (например, 0 для числовых типов).

Сброс заводской — команда удаления из прибора текущего приложения. Все остальные настройки (сетевые настройки и т. д.) не сбрасываются к заводским, а сохраняют текущие значения.

Сброс холодный — команда переинициализации всех переменных проекта, включая Retain переменные.

Слот — разъем в приборе для подключения интерфейса связи или модуля расширения.

Системные события — сообщения, генерируемые средой исполнения прибора во время работы исполняемого приложения. Предоставляют информацию о событиях, происходящих внутри системы.

Таргет — набор шаблонов кода, ресурсов и описаний компонентов для генерации приложения пользователя.

Тулбар — инструментальная панель быстрого доступа в интерфейсе ALTA IDE, с кнопками навигации и управления.

Функциональный блок (ФБ) — структурная единица программы, которая после выполнения выдает одно или более значений. Может быть создано множество поименованных экземпляров (копий) функционального блока.

Функция — структурная единица программы, которая после выполнения выдает только одно значение. Функция не хранит информацию о своем внутреннем состоянии, то есть вызов функции с одними и теми же фактическими параметрами выдает то же значение.

Цикл — время выполнения прибором заданной программы (зависит от количества выполняемых операций в программных цепях).

Шаг внутрь — во время работы в режиме отладки выполнение текущей строки кода. Если в строчке вызывается программный блок, заходит внутрь программного блока.

Шаг наружу — во время работы в режиме отладки продолжение выполнения всего кода текущего программного блока и возврат к строчке кода, откуда произошел переход внутрь программного блока.

Шаг поверх — во время работы в режиме отладки выполнение текущей строки кода. Если в строчке вызывается программный блок, то код программного блока выполняется полностью, без захода внутрь.

2 Установка ПО

Скачать инсталлятор ALTA IDEдля ОС Windows и Linux можно по ссылке.

При установке среды загружаются библиотеки базовой лицензии, при необходимости дополнительные библиотеки можно загрузить на страницеALTA IDE или в web-конфигураторе контроллера во вкладке ПЛК/ Загрузки.

2.1 Установка

1. Скачайте на ПК и запустите файл Alta.exe.
2. В открывшемся окне выберите язык установки, нажмите **OK**.

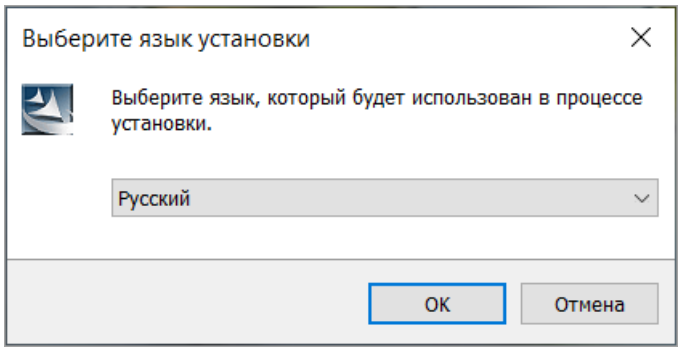


Рисунок 2.1

3. Откроется окно мастера установки. Выберите папку для установки ALTA IDE. Нажмите кнопку **Далее**.

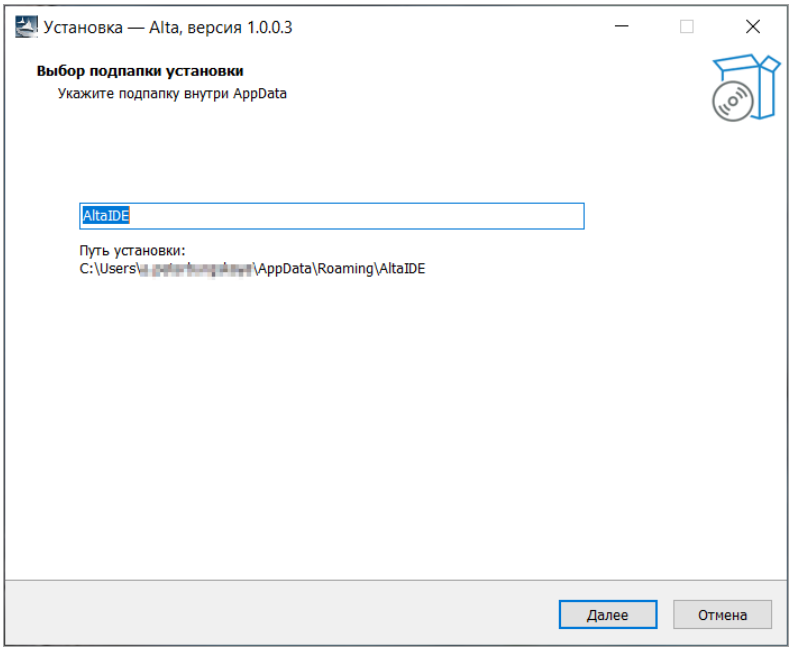


Рисунок 2.2

4. Ознакомьтесь с лицензионным соглашением и, в случае согласия, выберите **Я принимаю условия соглашения**. Нажмите кнопку **Далее**.

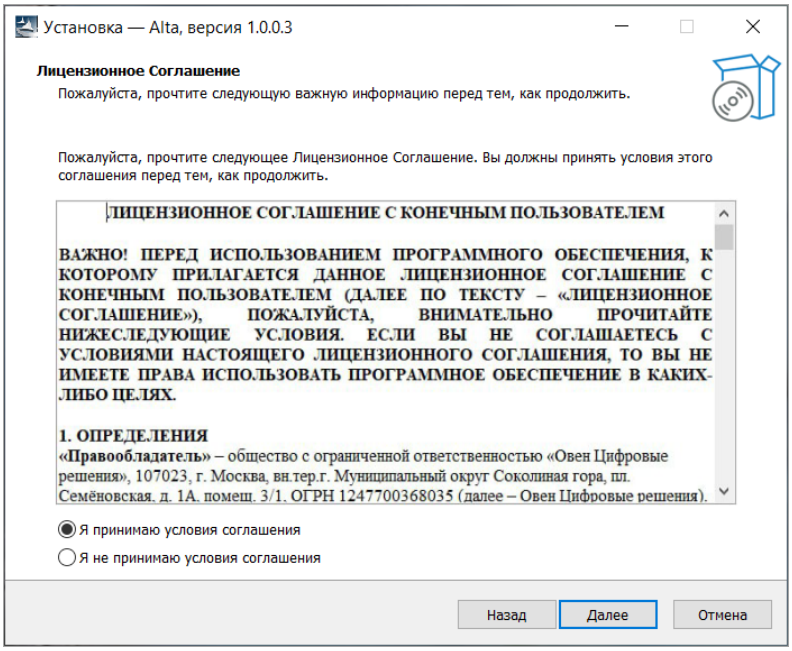


Рисунок 2.3

5. При необходимости создания ярлыка на рабочем столе установите соответствующий чекбокс. Нажмите кнопку **Далее**.

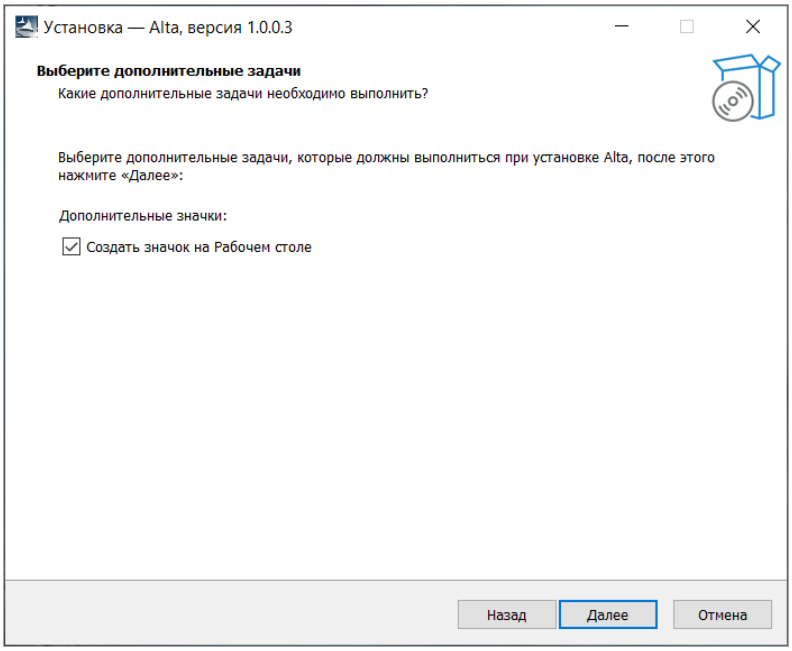


Рисунок 2.4

6. Ознакомьтесь с информацией об установке и нажмите кнопку **Установить**.

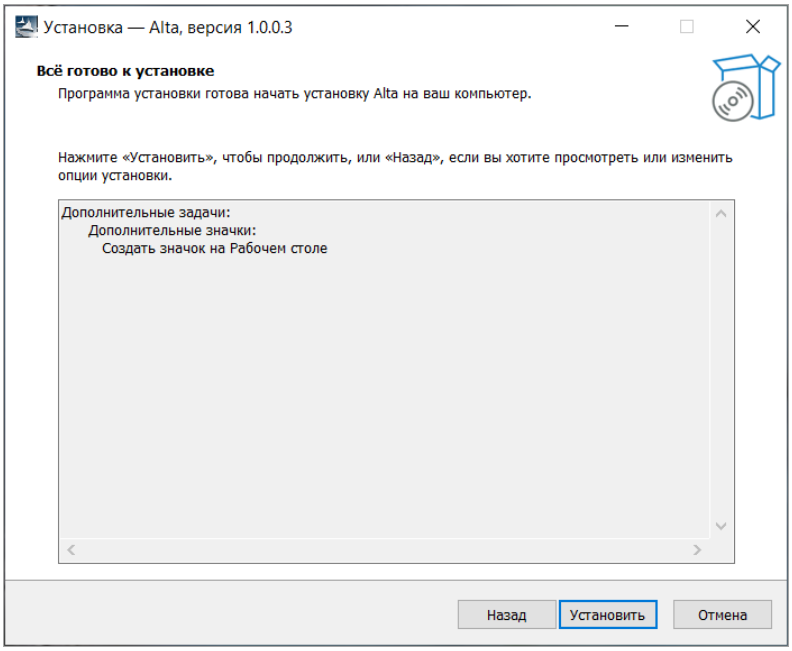


Рисунок 2.5

7. Откроется окно, в котором будет отображаться процесс установки.

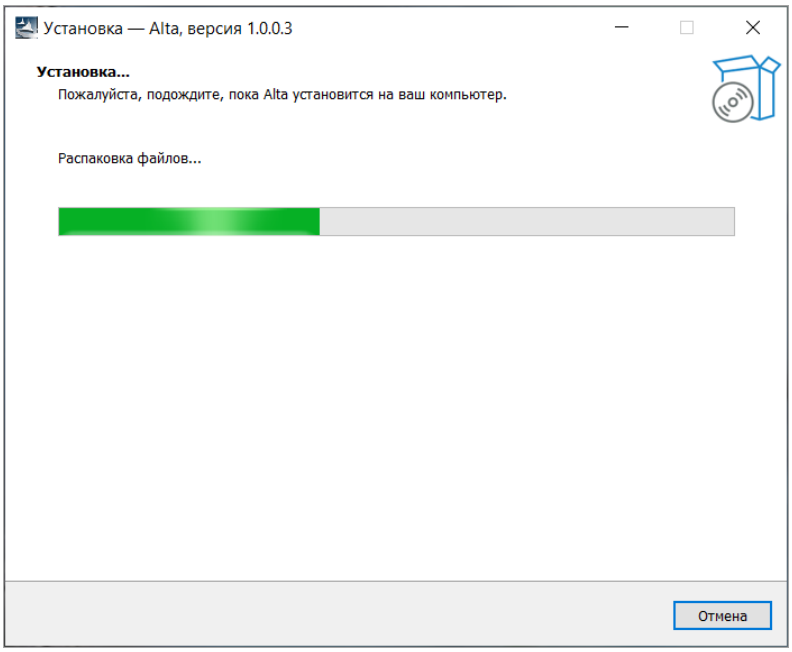


Рисунок 2.6

8. Дождитесь окончания установки. При необходимости установите чекбокс **Запустить Alta** и нажмите **Завершить**.

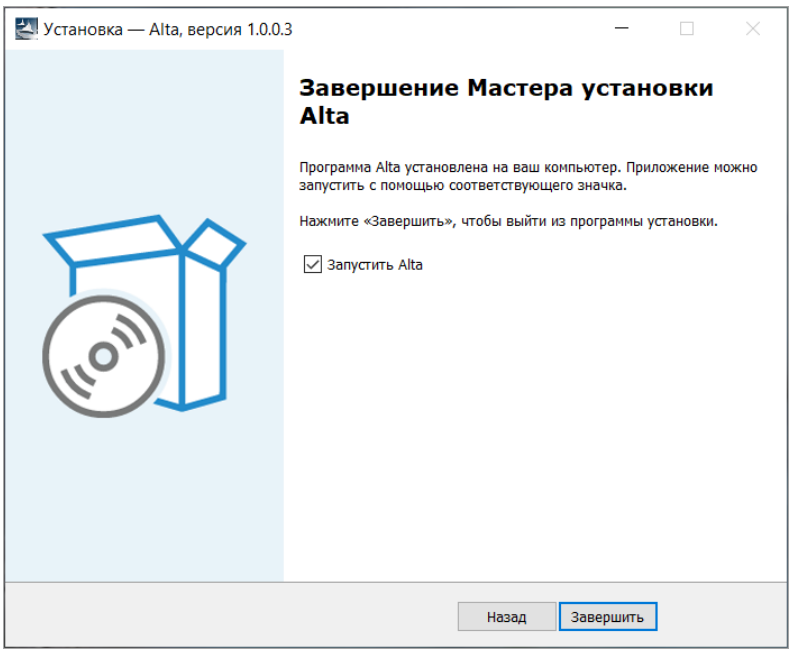


Рисунок 2.7

2.2 Обновление ALTA IDE

В случае наличия обновления при запуске ALTA IDE в правом нижнем углу отобразится информационное окно со списком доступных обновлений и предложением обновить среду до последней версии:

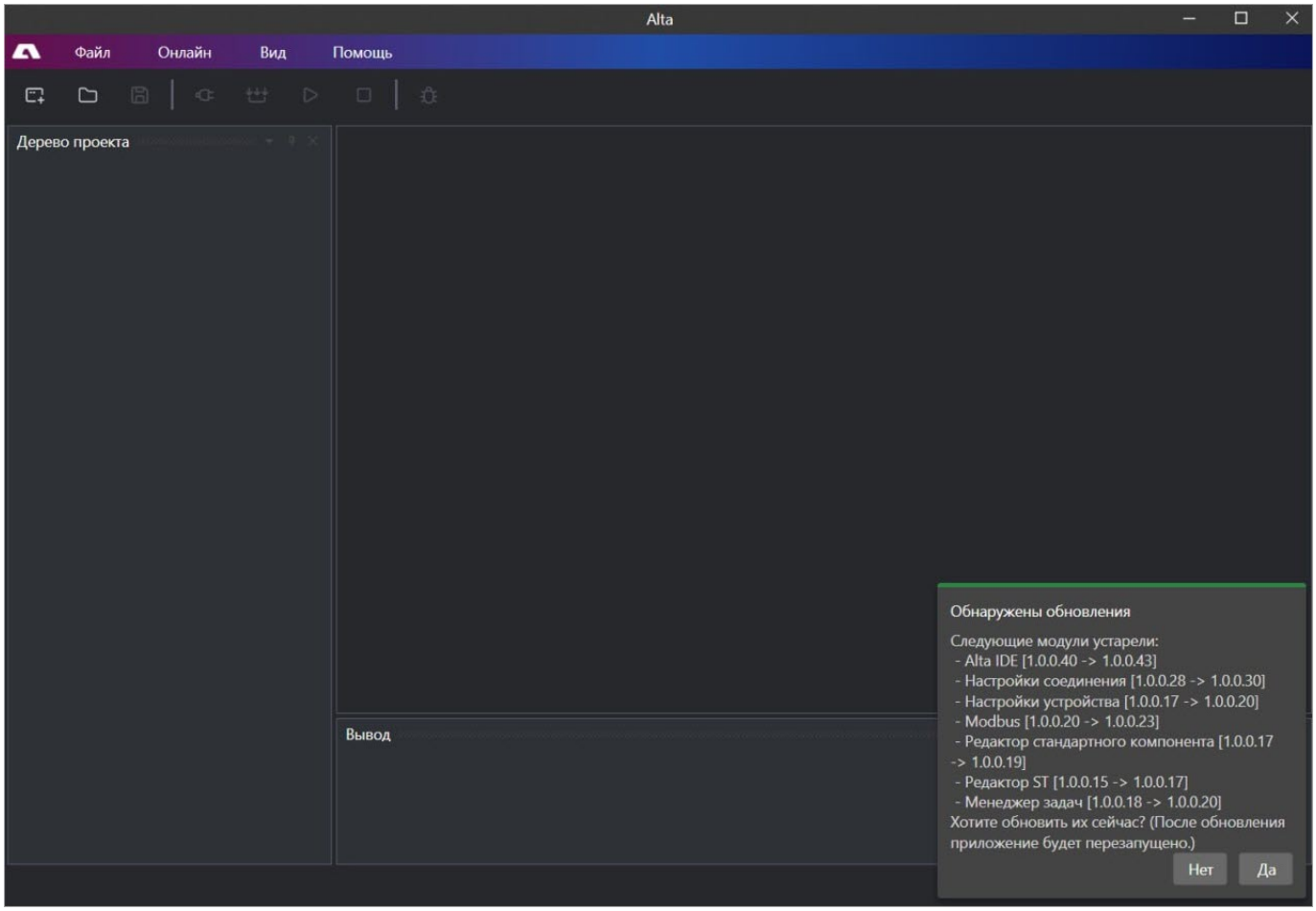


Рисунок 2.8

В случае выбора "Да" ALTA IDE обновится, приложение автоматически перезапустится.

Если выбрать "Нет", либо не выполнить выбор в течение 10 секунд, информационное окно закроется. В строке состояния появится иконка уведомляющая о наличии обновления  , при нажатии на которую снова появится информационное окно со списком доступный обновлений ALTA IDE, и возможностью выбора обновить ПО.

3 Описание интерфейса

ALTA IDE поддерживает работу интерфейса в двух цветовых темах: светлой и темной. Тема интерфейса определяет внешний вид окон, диалогов, кнопок, и всех визуальных элементов пользовательского интерфейса.

ПРИМЕЧАНИЕ
В ALTA IDE версии 1.0.0 поддерживается только темная тема.

После установки и запуска ALTA IDE откроется главное окно:

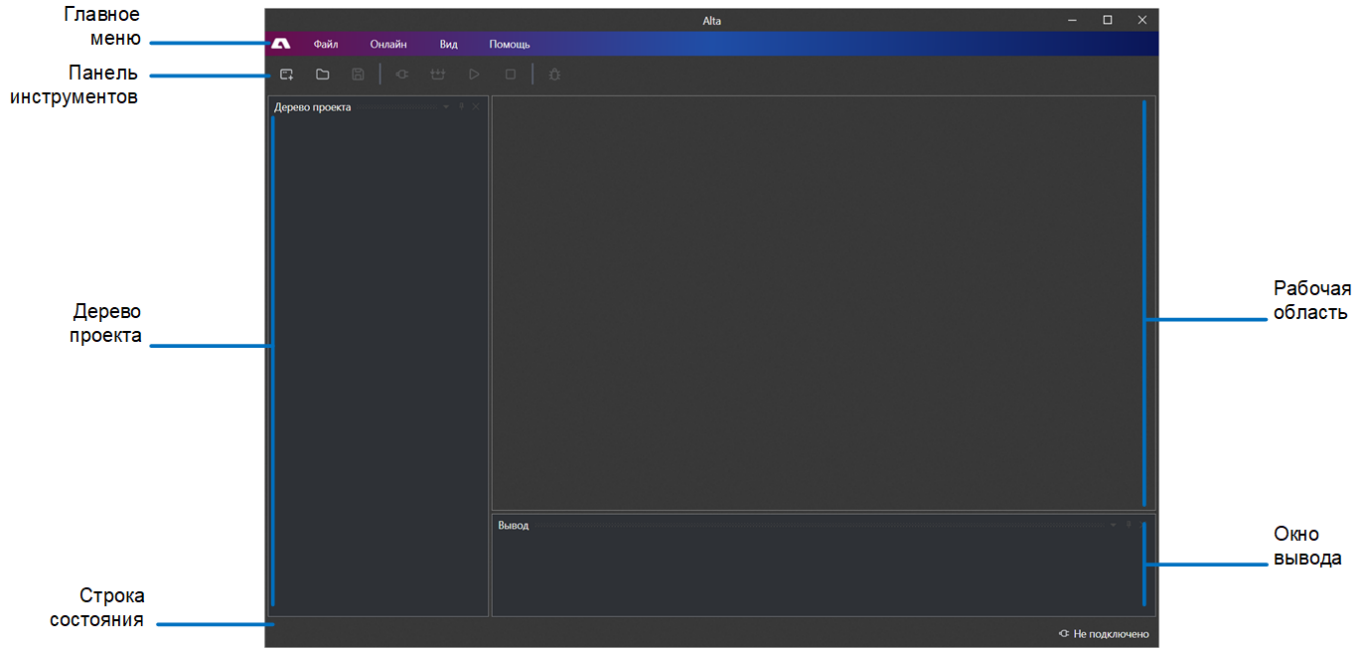


Рисунок 3.1 – Главное окно

Главное окно ALTA IDE содержит:

- главное меню: **Файл**, **Онлайн**, **Вид**, **Помощь**;
- панель инструментов;
- дерево проекта;
- рабочую область проекта;
- окно вывода;
- строку состояния.

3.1 Главное меню

Файл

Создать проект	Создание нового проекта
Открыть проект	Открытие ранее созданного и сохраненного проекта
Сохранить проект	Сохранение текущего проекта

Онлайн

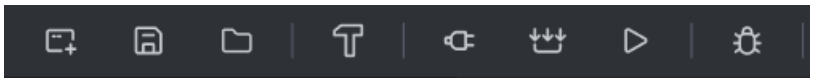
Установить/разорвать соединение	Установка/разрыв соединения с устройством
Настройки соединения	Вызов окна с настройками соединения
Старт/Стоп	Запуск/остановка приложения
Загрузить приложение	Сборка проекта и загрузка приложения в устройство
Выгрузить приложение	Выгрузка приложения с устройства на ПК

Вид

Темы	Выбор цветовой темы интерфейса: темной/светлой **ПРИМЕЧАНИЕ** В ALTA IDE версии 1.0.0 поддерживается только темная тема.

3.2 Панель инструментов

Панель инструментов



	Новый проект	Создание нового проекта. Текущий проект закрывается, перед закрытием будет предложено сохранить проект
	Сохранить проект	Сохранение текущего проекта. При первом сохранении вызывает окно для присвоения имени файлу
	Открыть проект	Открытие ранее созданного и сохраненного проекта
	Сборка проекта	Выполнение компиляции пользовательского кода в файл с загружаемым приложением
	Установить/разорвать соединение	Установка/разрыв соединения с устройством
	Загрузить приложение	Сборка проекта и загрузка приложения в устройство
	Старт/Стоп	Запуск/остановка выполнения приложения
	Онлайн	Запуск режима онлайн

Панель отладки отображается на панели инструментов в случае запуска режима онлайн (значок онлайн становится зеленым):

	Продолжить выполнение	Запуск/пауза выполнения программы
	Шаг внутрь	Выполнение текущей строки кода. Если в строчке вызывается программный блок, заходит внутрь программного блока
	Шаг наружу	Продолжает выполнение всего кода текущего программного блока и возврат к строчке кода, откуда произошел переход внутрь программного блока
	Шаг поверх	Выполнение текущей строки кода. Если в строчке вызывается программный блок, то код программного блока выполняется полностью, без захода внутрь
	Перейти к месту остановки	Переход к месту остановки выполнения программного кода
	Записать все значения	Запись подготовленных значений переменных
	Освободить все значения	Освобождение значений переменных от фиксации. Значения переменных доступны для изменения.

	Отключить все точки останова	Отключение всех точек останова в проекте
	Включить все точки останова	Включение всех точек останова в проекте
	Удалить все точки останова	Удаление всех точек останова в проекте

3.3 Дерево проекта

Проект в ALTA IDE имеет иерархическую структуру, которая представлена в виде Дерева проекта. Область Дерева проекта располагается слева в главном окне ALTA IDE. В Дереве проекта отображаются компоненты проекта – устройства, системные папки, программные объекты, задачи, узлы протоколов обмена:

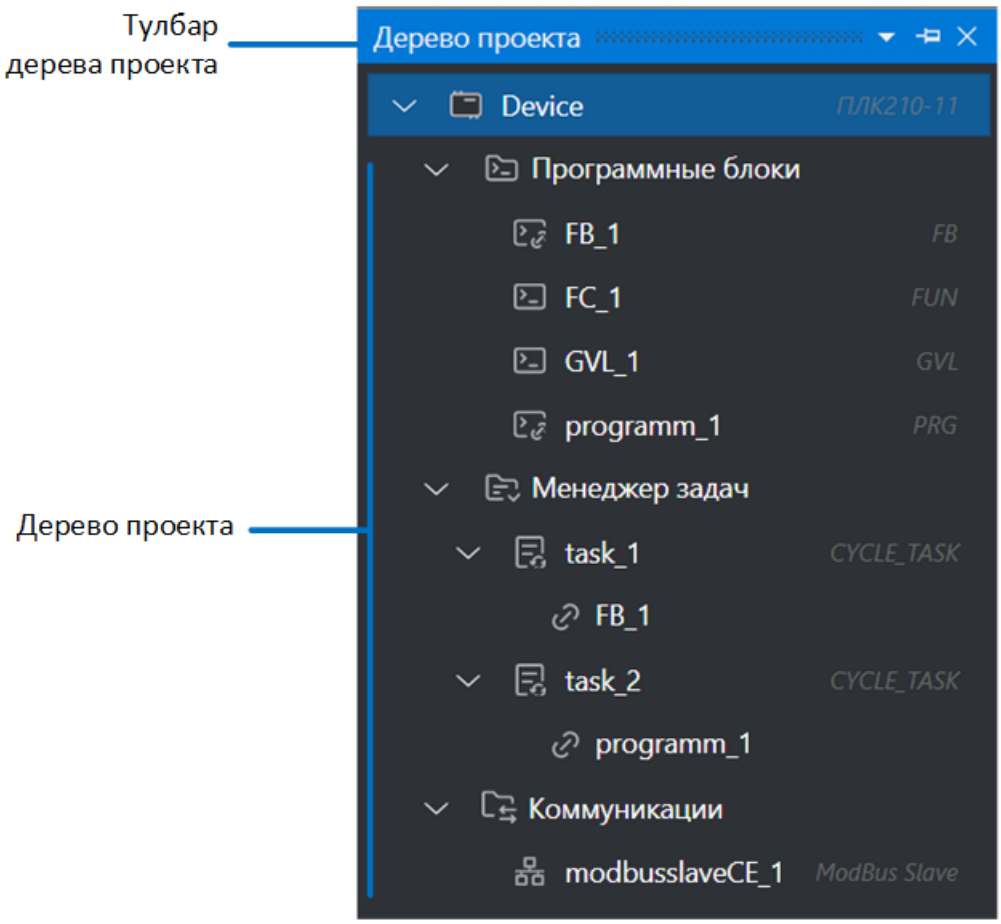


Рисунок 3.2 – Дерево проекта

Дерево проекта содержит обязательные системные папки:

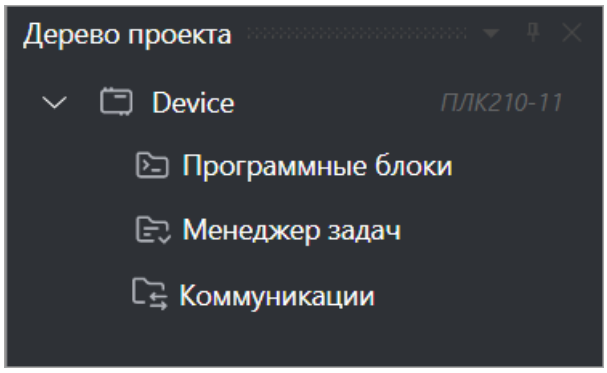


Рисунок 3.3 – Системные папки дерева проекта

	Устройство	Содержит аппаратные настройки, возможности прибора
	Программные блоки	Содержит созданные программы, функции и функциональные блоки
	Менеджер задач	Содержит задачи, созданные в рамках проекта
	Коммуникации	Содержит добавленные протоколы и подключенные по ним устройства

3.4 Рабочая область

В рабочей области отображаются вкладки, в которых происходит разработка программы, настройка параметров, ввод данных и т.п. Для открытия вкладки необходимо два раза нажать ЛКМ на элемент в дереве проекта, либо выбрать нужный пункт в Главном меню. Работа в каждой вкладке различается в зависимости от выбранного компонента.

В ALTA IDE возможно управление расположением вкладок в рабочей области. С помощью drag&drop можно изменять порядок вкладок, а также откреплять их от Главного окна для размещения в отдельном окне и при необходимости закреплять обратно.

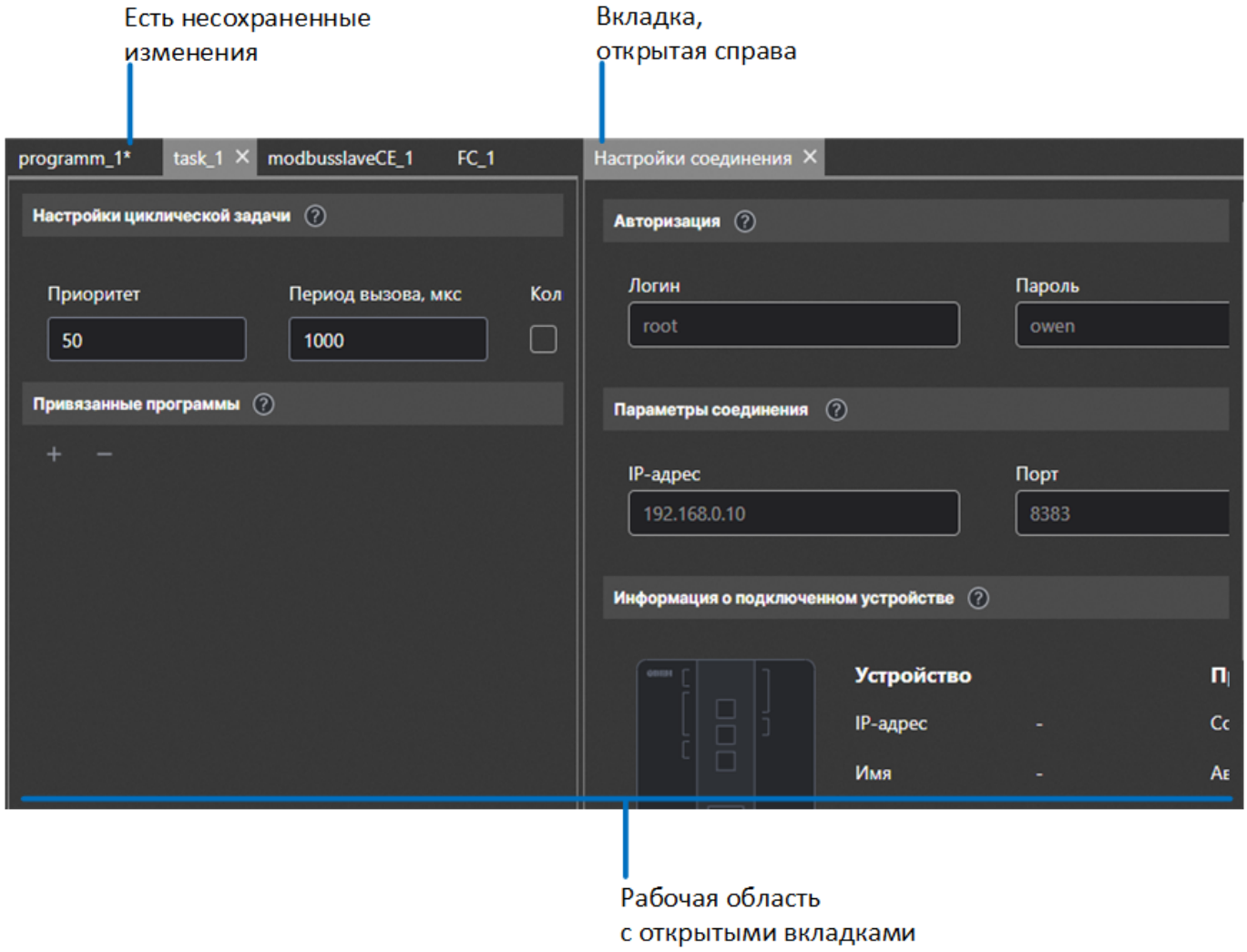


Рисунок 3.4 – Рабочая область ALTA IDE

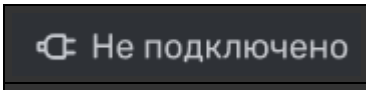
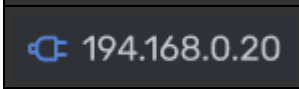
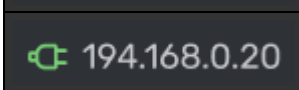
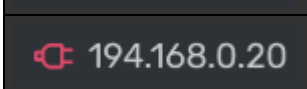
Значок * рядом с названием вкладки говорит о существующих несохраненных изменениях во вкладке. При попытке закрыть такую вкладку отобразится информационное окно с предложением сохранить внесенные изменения. В случае выбора "Да" информация, содержащаяся во вкладке сохранится в проекте, при выборе "Нет" данные сохранены не будут.

3.5 Окно вывода

В окне вывода отображаются ошибки проекта и ошибки сборки (в том числе возникшие при компиляции).

3.6 Строка состояния

Строка состояния располагается в нижней части главного окна ALTA IDE и содержит информацию о статусе и IP-адресе подключенного устройства, а также наличии обновлений:

	не подключено
	установка соединения
	соединение установлено
	ошибка соединения
	доступно обновление ALTA IDE

При загрузке проекта в прибор и запуске сборки проекта в строке состояния отображается шкала с индикацией выполнения:

Рисунок 3.5

4 Разработка проекта и порядок работы

Для успешной разработки проекта, загрузки на устройство и мониторинга работы приложения в ALTA IDE следует выполнить определенный ряд действий:

- создание проекта;
- разработка программы;
- сборка проекта;
- подключение устройства к ПК;
- запись приложения в прибор;
- режим онлайн и отладка программы.

Для облегчения понимания порядка работы приложения на ПЛК ниже приведены примеры порядка обработки задач на ПЛК.

Пример 1

Многопоточное выполнение задач. Достаточно ресурсов процессора для одновременного выполнения задач:

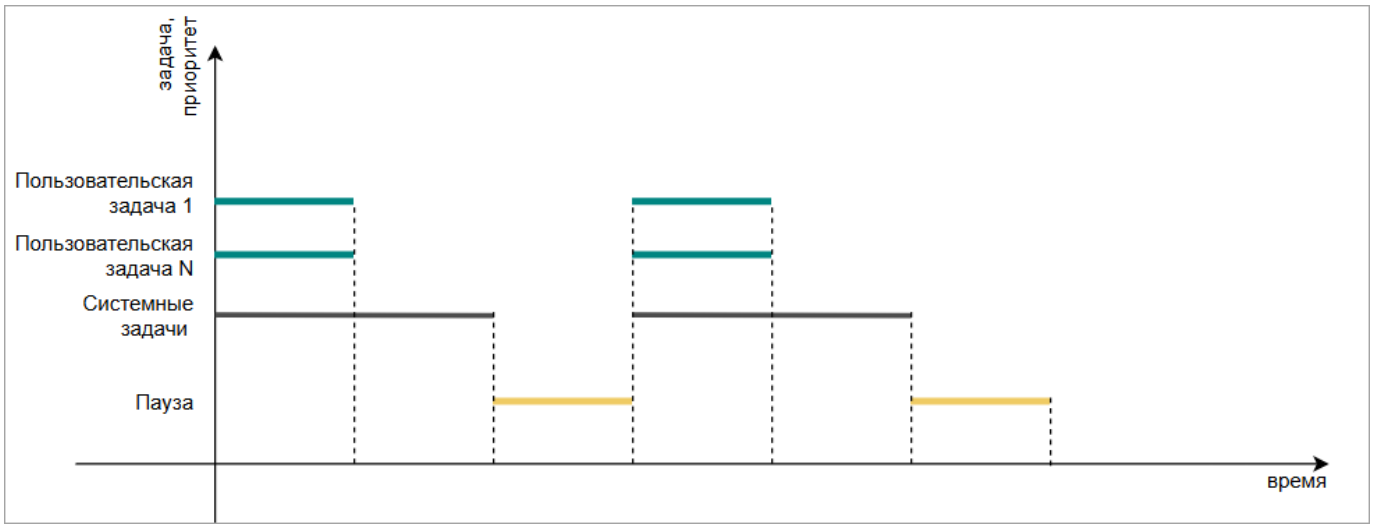


Рисунок 4.1 – Пример 1

Пример 2

Выполнение задач в режиме дефицита ресурсов процессора. Последовательное выполнение задач, согласно приоритету:

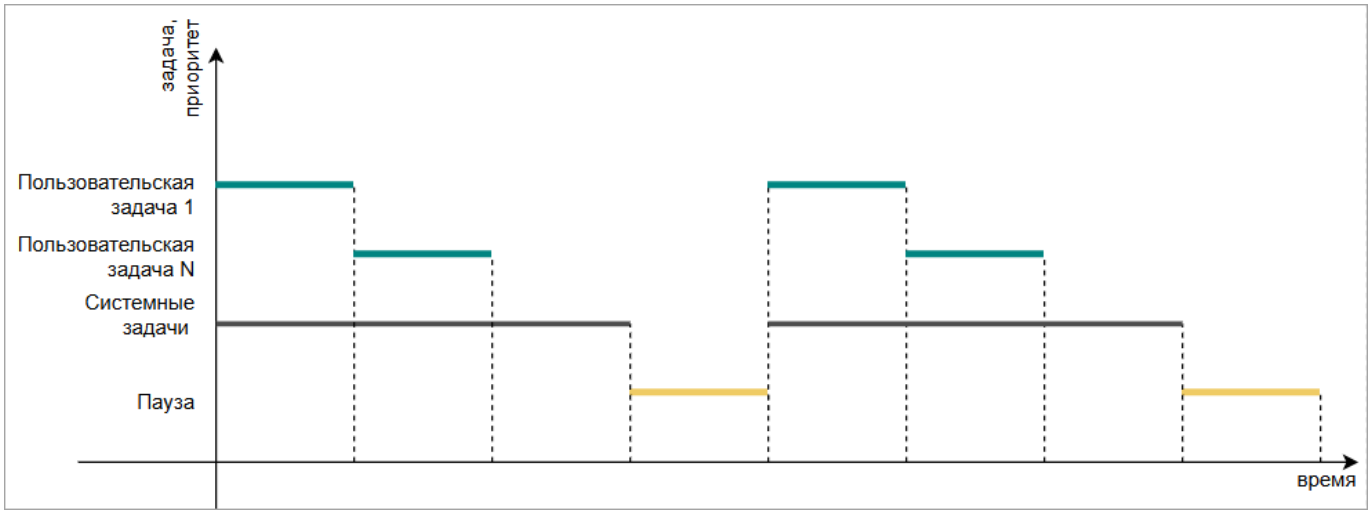


Рисунок 4.2 – Пример 2

Выполнение пользовательской задачи на ПЛК:

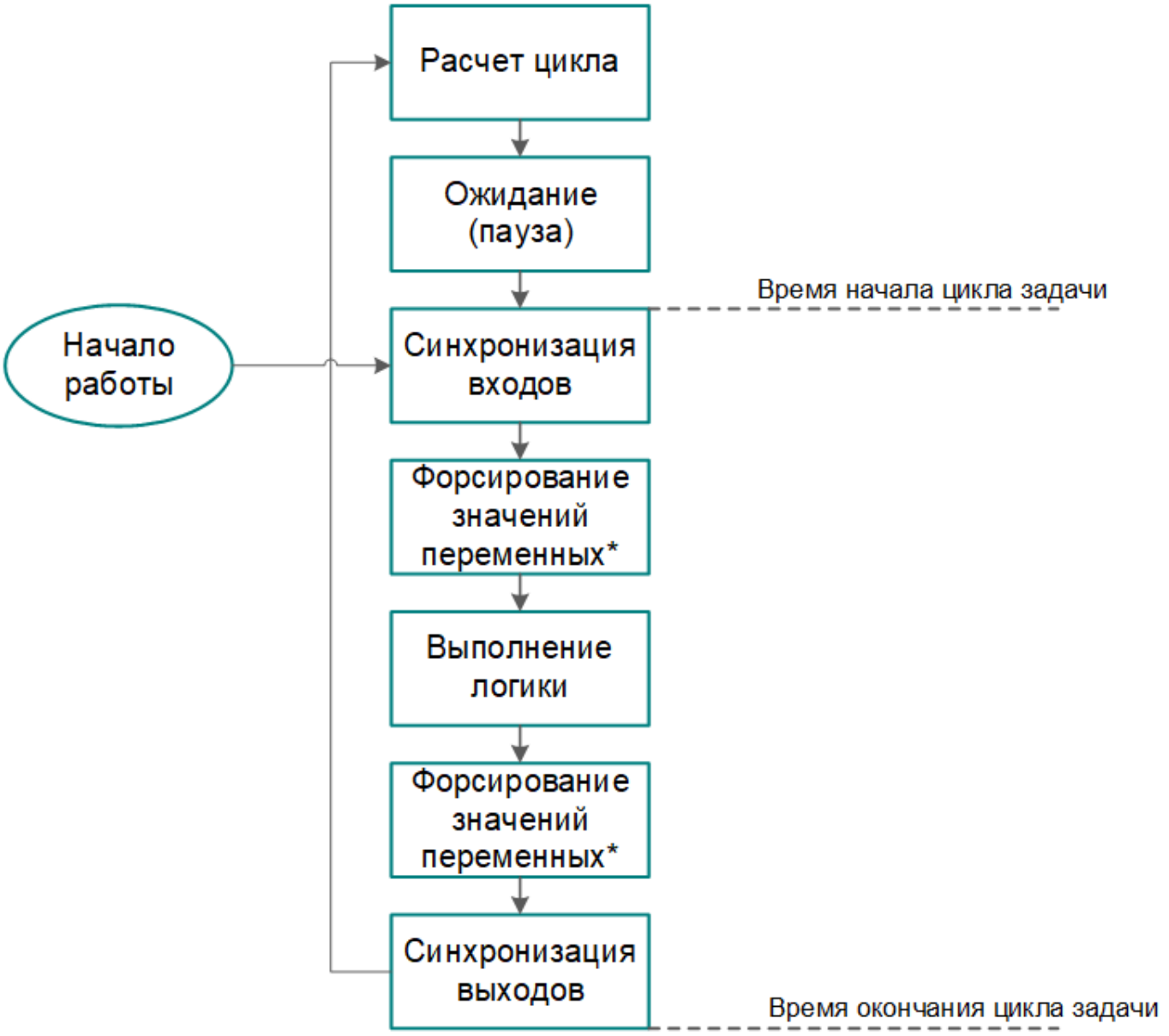


Рисунок 4.3

Блоки, отмеченные звёздочкой (*), выполняются только при наличии заданных пользователем форсированных значений переменных. Если форсированные значения отсутствуют, данные блоки исключаются из цикла.

4.1 Создание проекта

После установки и запуска откроется главное окно, в котором доступно создание проекта . Для этого воспользуйтесь кнопкой **Создать проект** на панели инструментов, или пунктом **Главное меню** → **Файл** → **Создать проект**.

В открывшемся окне введите данные:

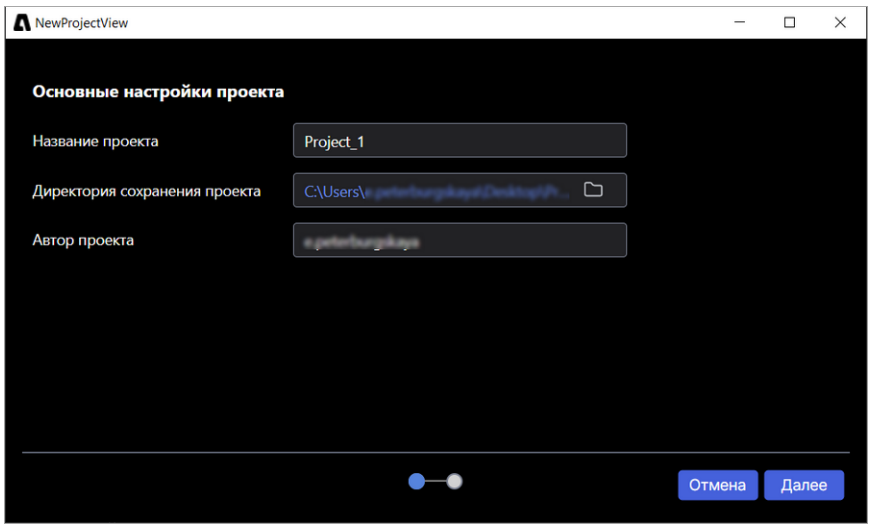


Рисунок 4.4

Название проекта - название создаваемого проекта;

ПРИМЕЧАНИЕ
При именовании проекта допустимо использование латиницы, кириллицы, а также цифр и символов, исключая специальные символы, запрещённые файловыми системами. Ограничение по длине имени 255 символов. При этом следует учитывать ограничение на путь к файлу в 259 символов. Регистр символов значения не имеет.

Директория сохранения проекта - расположение папки с файлами проекта;

Автор проекта - по умолчанию системное имя пользователя.

После заполнения всех строк нажмите кнопку **Далее**.

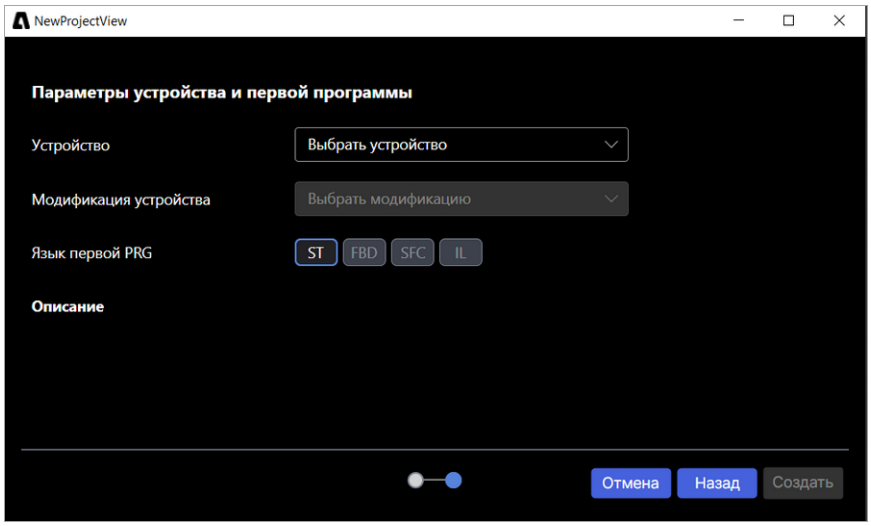


Рисунок 4.5

Заполните параметры устройства и выберите язык для написания первой программы:

Устройство - из выпадающего списка выберите устройство, для которого разрабатывается проект;

Модификация устройства - из выпадающего списка выберите модификацию устройства;

Язык первой PRG - выберите язык первой планируемой к написанию программы. Язык всегда можно поменять в настройках;

ПРИМЕЧАНИЕ
Для ALTA IDE версии 1.0.0 написание программы доступно только на языке ST.

Описание - после выбора устройства в области описания отобразится изображение устройства и его описание.

Нажмите кнопку **Создать**. В дереве проекта отобразится модификация устройства и обязательные системные папки:

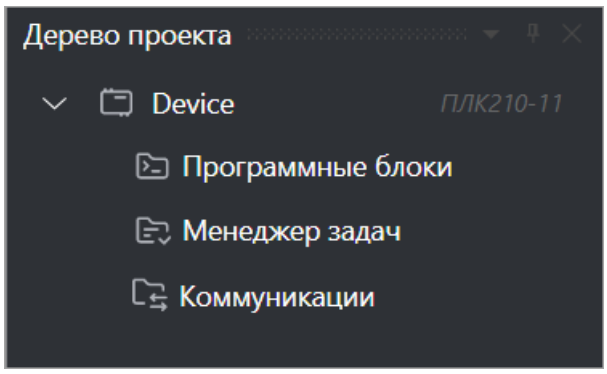


Рисунок 4.6

4.1.1 Открытие проекта

Для открытия ранее сохраненного проекта нажмите на кнопу  на панели инструментов, или воспользуйтесь **Главное меню** → **Файл** → **Открыть проект**. Откроется окно, в котором выберите файл проекта с расширением *.alta* и нажмите **Открыть**.

В дереве проекта отобразятся компоненты проекта, в рабочей области будут доступны просмотр и редактирование проекта.

В случае, если компонент проекта был поврежден (например файл, расположенный в папке проекта был удален), то этот компонент отобразится в дереве проекта с иконкой, оповещающей об ошибке:

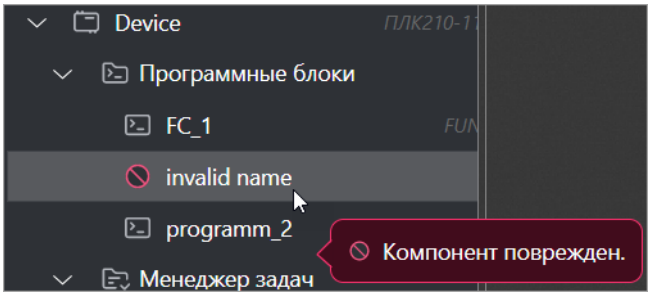


Рисунок 4.7

Для корректной работы приложения рекомендуется удалить такой компонент.

4.2 Разработка программы

Процесс разработки программы рекомендуется начать с этапа планирования. После формирования перечня задач, которые должны выполняться в рамках программы, следует приступить к созданию программных объектов, предназначенных для решения этих задач. Затем созданные программы необходимо привязать к соответствующим задачам, которые добавляются в **Менеджере задач**.

4.2.1 Программные блоки

Компонент Программные блоки (POU) располагается в дереве проекта и позволяет создавать программные объекты: программы, функциональные блоки, функции.

- **Программа** – высокоуровневый программный объект, привязываемый к задачам контроллера. Программы соответствуют крупным фрагментам технологического процесса (например, «управление вентиляцией», «обработка тревог» и т.д.);
- **Функция** – программный объект, возвращающий одно значение;
- **Функциональный блок** – аналог классов из современных языков программирования. Для использования функциональных блоков нужно объявить их экземпляры (инстансы); для каждого экземпляра выделяется своя область памяти. Ключевое отличие функциональных блоков от функций в том, что переменные блоков сохраняют свои значения между вызовами. Функциональные блоки используются для создания счетчиков, таймеров и других объектов, которым требуется сохранение данных.

Тип каждого элемента определяется специальной иконкой:

	Программа
	Функция
	Функциональный блок
	Программа, привязанная к задаче
	Предупреждение. В случае, если рядом с иконкой отображается символ предупреждения, следует навести мышку на иконку - отобразится информационное окно с предупреждением.

4.2.1.1 Действия с программными объектами

Добавление POU

Для добавления программного объекта воспользуйтесь контекстным меню системной папки **Программные блоки** в дереве проекта:

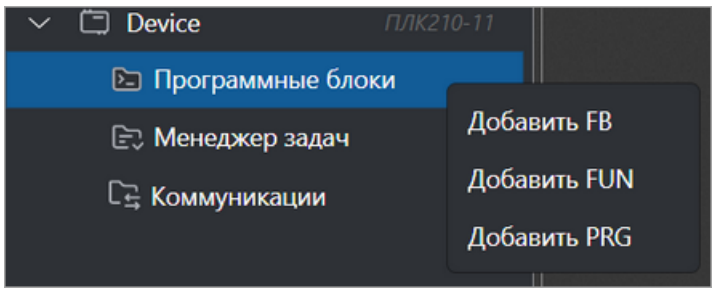


Рисунок 4.8 – Контекстное меню системной папки Программные блоки

- **Добавить FB** — добавление функционального блока;
- **Добавить FUN** — добавление функции;
- **Добавить PRG** — добавление программы.

Добавленный программный объект отобразится в дереве проекта, как ответвление интерфейса:

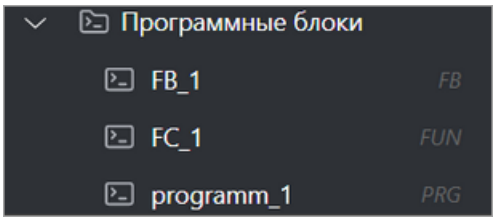


Рисунок 4.9

Созданный программный объект по умолчанию имеет имя *_1 (имя зависит от объекта). Каждому последующему программному блоку будет присваиваться следующий порядковый номер. Переименование доступно в момент создания, либо через контекстное меню программного блока. При задании компоненту имени следует учитывать правила, которые совпадают с правилами именования языка ST, и убедиться, что выбранное имя не дублирует названия других компонентов в дереве проекта.

Контекстное меню программного объекта содержит:

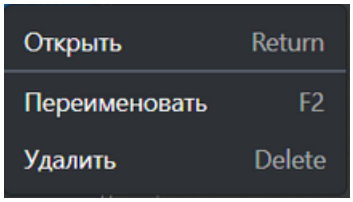


Рисунок 4.10

- **Открыть** — открыть вкладку программного объекта в рабочей области.
- **Переименовать** — введите новое имя программного объекта.

  Если программа была привязана к задаче, то после переименования элемент отобразится в системной папке Программные блоки в дереве проекта и будет иметь статус непривязанной. Программа с прежним именем будет отображаться в дереве проекта и во вкладке Менеджер задач помеченная маркером красного крестика. Задача, к которой выполнена привязка переименованной программы будет подчеркнута красным, при наведении мышки возникнет подсказка с описанием ошибки:

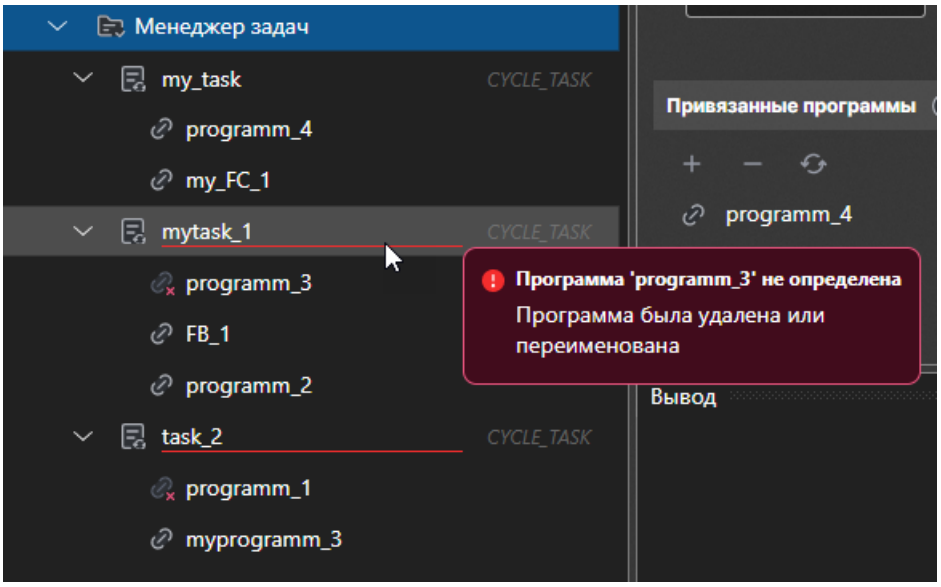


Рисунок 4.11 – Отображение ошибки в дереве проекта

Чтобы устранить ошибку, выберите один из вариантов:

– Создайте новую программу с прежним именем — созданная программа автоматически привяжется к задаче.

– Удалите задачу, связанную с несуществующей программой.

- **Удалить** — удалить выбранный программный объект.

  Отобразится информационное окно с подтверждением удаления объекта. Если программа была привязана к задаче, то в случае удаления она будет удалена из системной папки Программные блоки, но останется в Менеджере задач (будет отображаться с ошибкой). Задача к которой выполнена привязка удаленной программы будет подчеркнута красным, при наведении мышки возникнет подсказка с описанием ошибки и способом ее исправления. Если будет создана программа с именем ранее удаленной привязанной программы, привязка к задаче выполнится автоматически.

Привязка POU

Выполнить привязку созданной программы к задаче можно несколькими способами:

- с помощью контекстного меню элемента **Задача** → **Добавить привязку программы**;
- в окне редактора задачи, в разделе "Привязанные программы" с помощью кнопки  :

  В открывшемся окне привязки программы выберите нужную программу, нажмите кнопку Сохранить. Привязанная программа добавится вниз списка.

Рисунок 4.12 – Окно привязки программ

ПРИМЕЧАНИЕ
Доступна привязка сразу нескольких программ. В окне привязки программы нажмите и удерживайте кнопку Shift и с помощью ЛКМ выделите программы, которые необходимо привязать.

- с помощью drag & drop программы в дереве проекта из вкладки программные блоки во вкладку:
  - **Менеджер задач** → **Задача**: привязанная программа добавляется вниз списка;
  - **Менеджер задач** → **Задача** → **Программа**: привязанная программа добавляется на место выбранной программы, остальные программы сдвигаются вниз.

ПРИМЕЧАНИЕ
В случае, если достигнут лимит привязанных к задаче программ, появится сообщение об ошибке:

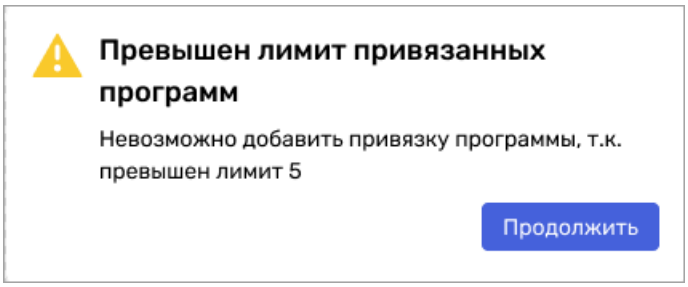


Рисунок 4.13 – Информационное окно

В случае привязки программы, уже привязанной к задаче возникнет логическая ошибка. Программа привяжется, задачи будут подчеркнуты красным цветом, иконка повторно привязанной программы отобразит множественный вызов. При наведении мышки на задачу (программу) появится всплывающее сообщение об ошибке.

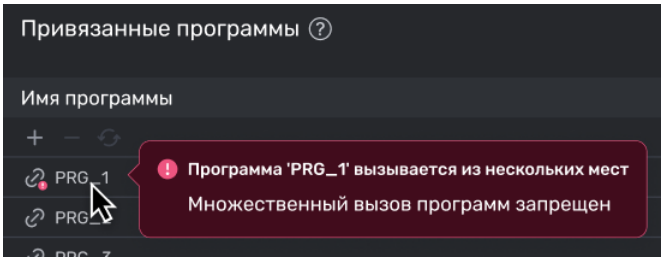


Рисунок 4.14 – Отображение множественного вызова в дереве проекта

Для корректной работы программы требуется удалить привязку к одной из задач, в противном случае выполнение программы может быть некорректно.

Смена порядка вызова привязанных программ

Программы в задаче выполняются в том порядке, в котором они расположены в дереве проекта (перечислены в списке менеджера задач). Порядок выполнения программы в задаче можно сменить методом drag & drop в дереве проекта. Для перетаскивания доступна как одна программа, так и несколько (выбор с помощью удержания кнопки Shift и ЛКМ).

Удаление привязки программ

Для удаления привязки программы воспользуйтесь контекстным меню программы, расположенной в дереве проекта в системной папке Менеджер задач:

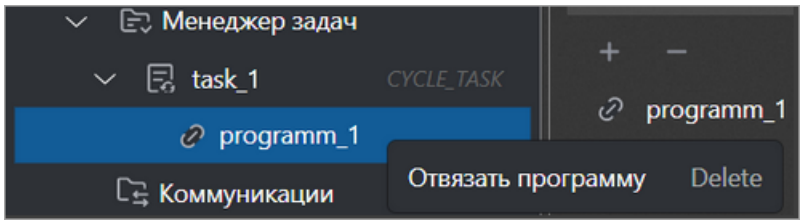


Рисунок 4.15

Выберите Отвязать программу, или воспользуйтесь кнопкой  в окне редактора задачи, в разделе "Привязанные программы". Программный объект перейдет в статус непривязанного.

ПРИМЕЧАНИЕ
Программа не привязанная к задаче не выполняется и не входит в сборку проекта.

4.2.1.2 Редактор ST

Вкладка редактора функции/функционального блока/программы предназначена для написания кода на языке программирования ST. Подробнее правила работы на **языке ST** рассмотрены в следующих разделах.

Чтобы открыть вкладку редактора ST воспользуйтесь контекстным меню программного объекта в дереве проекта, или дважды нажмите ЛКМ:

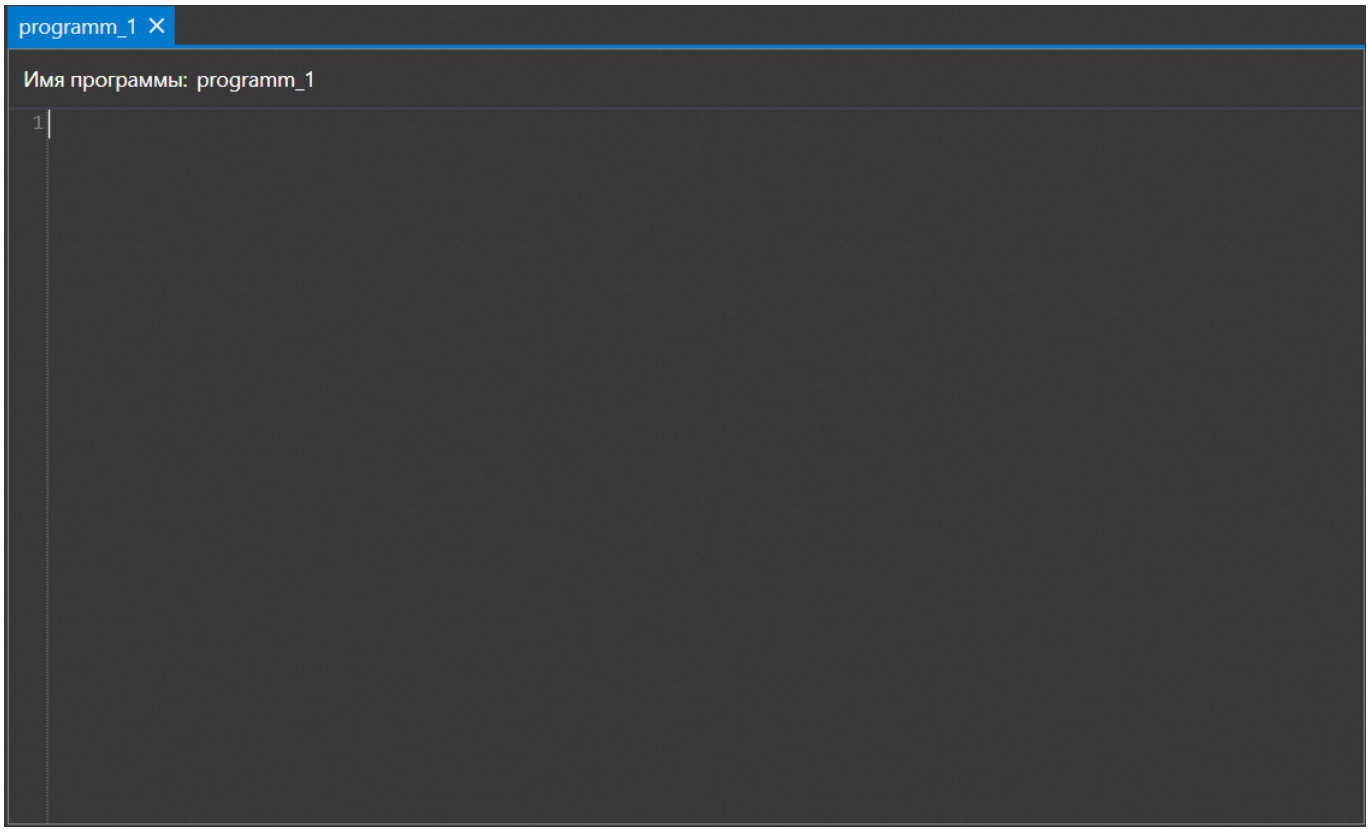


Рисунок 4.16 – Вкладка редактора ST

Вкладка содержит имя программного объекта, а также поле для написания кода. Строки кода автоматически нумеруются в процессе кодирования.

В ALTA IDE поддержана интеллектуальная подсветка кода и автодополнение.

Автодополнение

Автодополнение позволяет получить список возможных вариантов кода на основе текущего контекста и частично введенного текста.

Окно автодополнения открывается автоматически, в процессе написания кода. При необходимости окно автодополнения можно открыть принудительно, для этого нажмите CTRL +SPACE, если место в коде предполагает наличие вариантов автодополнения, то окно откроется.

Из предложенных вариантов с помощью стрелок или мышки выберите нужный, нажмите ЛКМ, либо Enter - слово добавится в код. Если нажать клавишу Tab, то выбранный вариант заменит слово, при написании которого открылось окно автодополнения, включая уже введенные символы.

После добавления слова окно автодополнения закроется автоматически. Также закрыть окно автодополнения можно с помощью клавиши Esc, либо нажатием ЛКМ за пределами окна автодополнения.

4.2.2 Менеджер задач

Компонент Менеджер задач предназначен для создания, мониторинга и управления задачами. Помогает следить за приоритетами задач и периодами выполнения, а также добавлять задачи, которые представлены в иерархии в области дерева проекта:

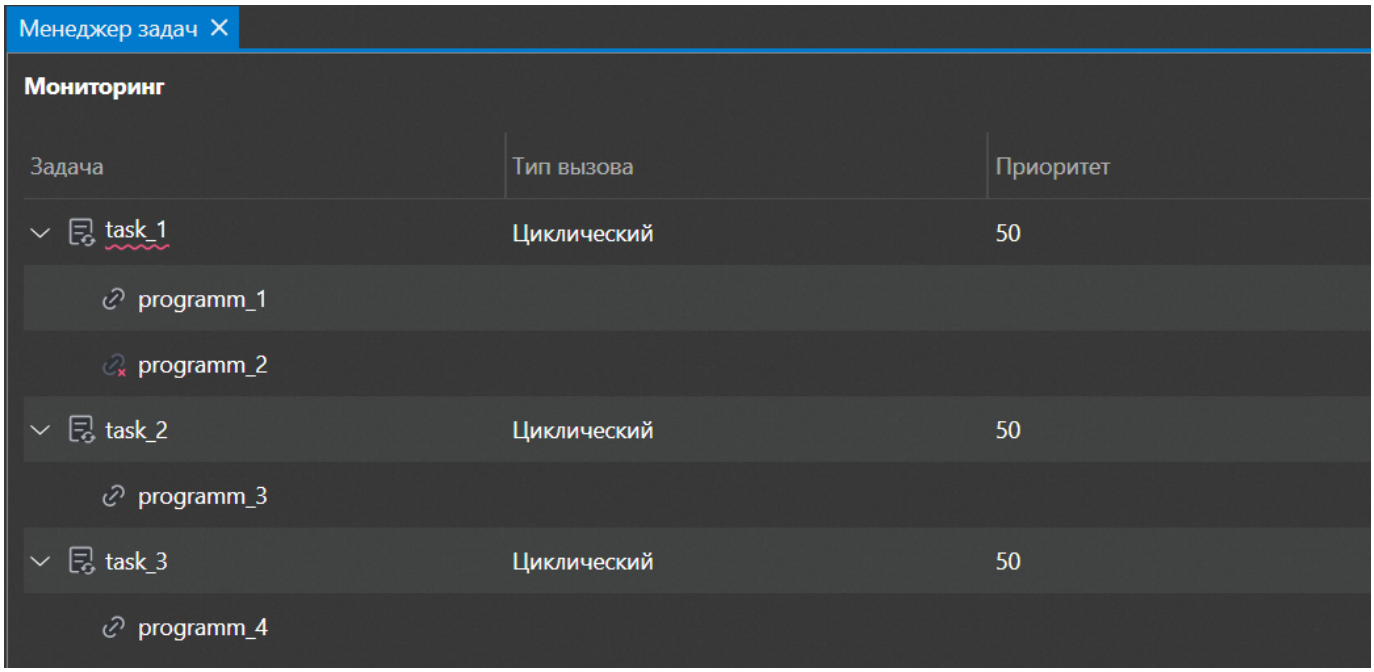


Рисунок 4.17 – Менеджер задач в дереве проекта

К каждой задаче привязывается одна или несколько программ, которые будут выполняться в рамках задачи. Разрешенное количество привязываемых к задаче программ зависит от прибора. Порядок выполнения задач определяется согласно установленному приоритету и условию вызова. Условием вызова задачи может служить время (циклическое выполнение) или событие, внутреннее или внешнее (например, превышение заданного порога глобальной переменной или прерывание в устройстве).

ПРИМЕЧАНИЕ
В ALTA IDE версии 1.0.0 для создания доступны только циклические задачи.

Под каждой задачей раскрывается список привязанных программ (при наличии). Для того, чтобы свернуть/развернуть список привязанных программ нажмите на стрелку, расположенную слева от имени задачи. Если стрелка отсутствует, значит к задаче не привязано ни одной программы.

Тип каждого элемента определяется специальной иконкой:

	Циклическая задача
	Задача по событию **ПРИМЕЧАНИЕ** В ALTA IDE версии 1.0.0 для создания доступны только циклические задачи.
	Программа, привязанная к задаче

В случае возникновения ошибок, иконки помечаются специальными маркерами, обозначающими конкретную ошибку:

	Программный объект, привязанный к нескольким задачам
	Привязанный программный объект не существует (удален или переименован)

При наведении на маркер возникает подсказка с информацией об ошибке и способами исправления.

Задача, содержащая ошибку,в дереве проекта и на вкладке Менеджер задач будет подчеркнута красным, при наведении мышки появится подсказка, содержащая информацию об ошибке и способы исправления:

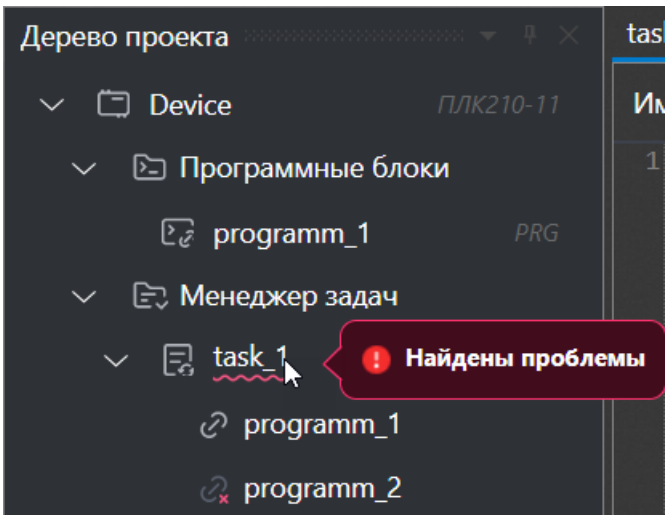


Рисунок 4.18 – Отображение ошибки в задаче

В режиме онлайн задачи могут иметь следующий статус:

	Задача выполняется	Выполнение задачи в режиме онлайн
	Пауза в выполнении задачи	При выполнении задачи привязанная программа достигла точки останова, требуется действие пользователя для продолжения выполнения программы
	Активная задача	Указывает на задачу в дереве проекта, а также на строку в программном коде, выполняющуюся в данный момент

Чтобы открыть вкладку **Менеджер задач** воспользуйтесь контекстным меню системной папки **Менеджер задач** в дереве проекта, или дважды нажмите ЛКМ. Вкладка **Менеджер задач** содержит информацию о созданных задачах, основных настройках, и привязанных программных объектах (при наличии):

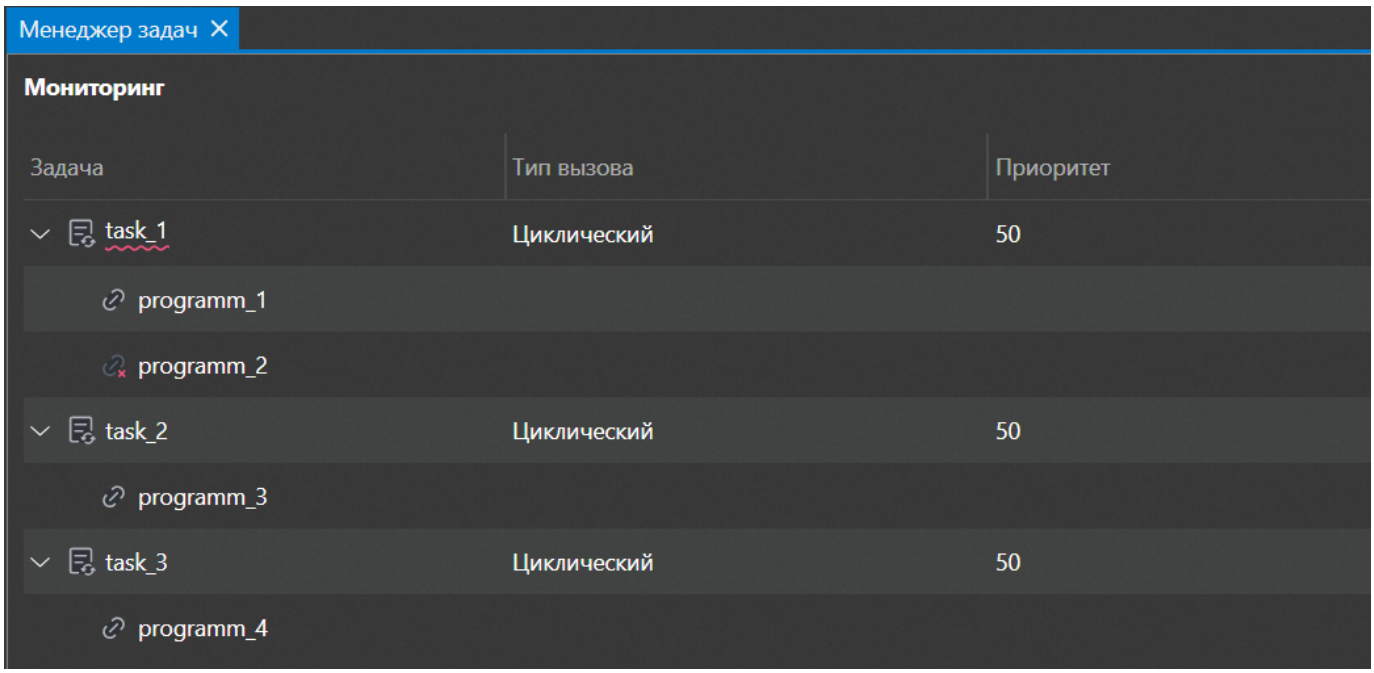


Рисунок 4.19 – Вкладка Менеджер задач

- **Задача** — имя созданной задачи.
- **Тип вызова** — циклическая задача или задача по событию;

  ПРЕДУПРЕЖДЕНИЕ
  В ALTA IDE версии 1.0.0 для создания доступны только циклические задачи.

- **Приоритет** — приоритет выполнения задачи (от 1 до 99, в зависимости от типа задачи, где 1 наивысший приоритет).

Под каждой задачей располагаются привязанные программы (при наличии). Для того, чтобы свернуть/развернуть список привязанных программ нажмите на стрелку, расположенную слева от имени задачи. Если стрелка отсутствует, значит к задаче не привязано ни одной программы.

Контекстное меню

Для вызова контекстного меню **Менеджера задач** нажмите ПКМ на системную папку **Менеджер задач** в дереве проекта:

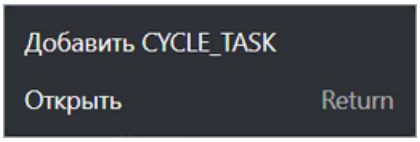


Рисунок 4.20 – Контекстное меню Менеджера задач

- **Добавить CYCLE_TASK** — добавление циклической задачи;
- **Открыть** — открытие вкладки Менеджер задач.

4.2.2.1 Действия с задачами

Задача - элемент управления, который позволяет выполнять один или несколько программных объектов на периодической основе.

При выполнении задач применяются следующие правила:

1. Выполняется та задача, условия выполнения которой истинны, т.е. прошло указанное время.
2. Если у нескольких задач условия выполнены одновременно, то выполняется задача с наивысшим приоритетом.
3. Если у нескольких задач условия выполнены одновременно и одинаковый приоритет, то выполняется та задача, которая имеет большее время ожидания.
4. Программы одной задачи выполняются в том же порядке, в каком они перечислены в списке **Менеджера задач** и расположены в дереве проекта.

5. Перед выполнением привязанных программ в задаче происходит синхронизация входов компонентов с переменными, используемыми в программах задачи. После выполнения программ происходит аналогичная синхронизация выходов.

Добавление задачи

Для добавления задачи воспользуйтесь контекстным меню системной папки **Менеджер задач** в дереве проекта:

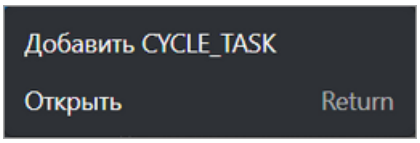


Рисунок 4.21 – Контекстное меню менеджера задач

Добавленная задача отобразится в дереве проекта, как ответвление системной папки **Менеджер задач**:

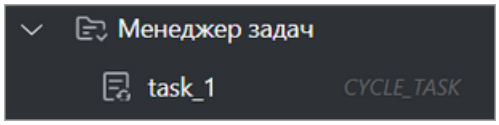


Рисунок 4.22

В случае, если достигнут лимит добавленных задач, появится сообщение об ошибке:

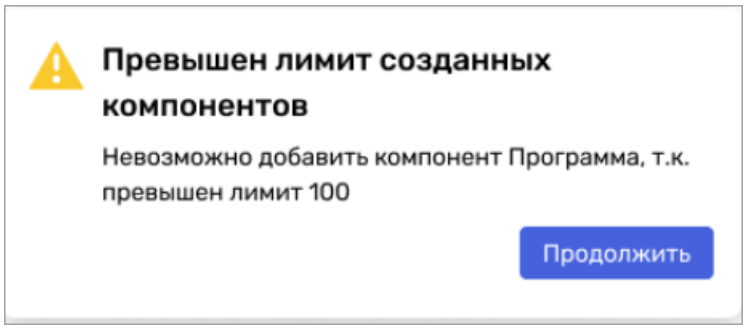


Рисунок 4.23 – Информационное окно

ПРИМЕЧАНИЕ
Максимальное количество добавленных задач зависит от прибора.

Созданная задача по умолчанию имеет имя task_1 (имя созданного компонента зависит от прибора). Каждой последующей задаче будет присваиваться следующий порядковый номер. Переименование доступно в момент создания задачи, либо через контекстное меню задачи. При задании компоненту имени следует учитывать правила, которые совпадают с правилами именования языка ST, и убедиться, что выбранное имя не дублирует названия других компонентов в дереве проекта.

Контекстное меню задачи содержит:

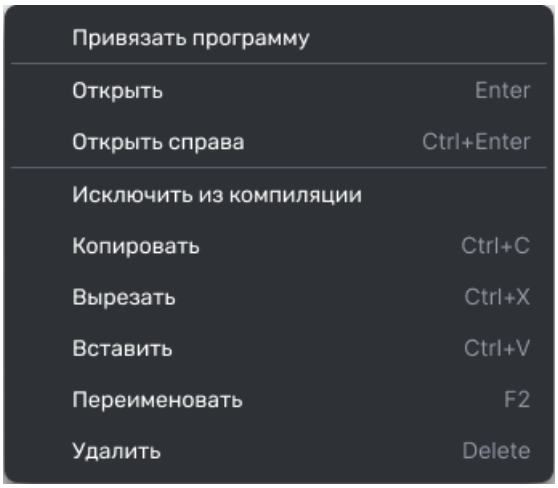


Рисунок 4.24 – Контекстное меню задачи

- **Добавить привязку программы** — привязать программу к задаче.
- **Открыть** — открыть вкладку Задача в рабочей области.
- **Открыть справа** — открыть вкладку Задача в рабочей области справа от открытого окна.
- **Исключить из компиляции** — задача не будет участвовать в сборке проекта, в дереве проекта и во вкладке Менеджер задач будет выделена как неактивный элемент (серым цветом).
- **Копировать/вырезать/вставить** — для копирования/вырезания задачи выберите в контекстном меню необходимую строку. Вставка скопированной/вырезанной задачи возможна либо в пределах папки Менеджер задач в дереве проекта, либо в созданную пользовательскую папку. Для этого наведите мышку на задачу, находящийся внутри папки, вызовите контекстное меню и выберите Вставить. Скопированный объект добавится в выбранную папку.
- **Переименовать задачу** — введите новое имя задачи. Компоненты, в которых использовалась ссылка на переименованную задачу, будут помечены маркером ошибки.
- **Удалить задачу** — удалить выбранную задачу.

ПРИМЕЧАНИЕ
При удалении задачи с привязанными программными объектами удаляется только задача, программные объекты остаются в проекте и переходят в статус непривязанных.

После добавления задачи необходимо настроить условия выполнения и привязать программы.

4.2.2.2 Циклическая задача

Циклическая задача вызывает привязанные программы периодически в соответствии с заданным пользователем интервалом времени. Открыть окно редактора задачи можно с помощью двойного нажатия ЛКМ на элемент Задача или контекстного меню элемента Задача в дереве проекта:

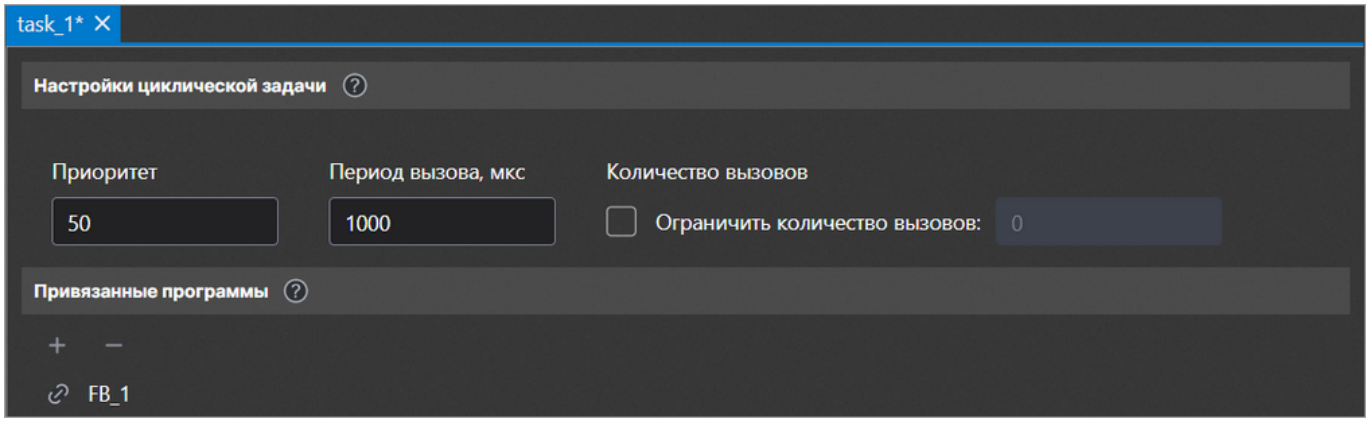


Рисунок 4.25 – Окно редактора циклической задачи

Окно редактора циклической задачи содержит:

- **Приоритет** — приоритет выполнения задачи. Число от 5 до 50 со следующими значениями: 50 - самый высокий, 5 – самый низкий приоритет;
- **Период вызова, мкс** — период времени (в микросекундах), после которого задача должна быть вызвана в очередной раз;
- **Ограничить количество вызовов** — поставьте галочку, и введите число, если требуется ограничение количества вызовов задачи;
- **Привязанные программы** — окно с перечислением привязанных программ в порядке их выполнения.

В случае ввода недопустимого значения любого из параметров окно ввода выделится красным цветом. При наведении мышки отобразится сообщение с текстом ошибки.

4.2.3 Настройка входов/выходов устройства

Для настройки устройства дважды нажмите ЛКМ на элемент **Устройство** в дереве проекта или воспользуйтесь контекстным меню устройства: выберите пункт **Открыть** или **Открыть справа**. Откроется вкладка с настройками, которая состоит из дерева настройки устройства и рабочей области:

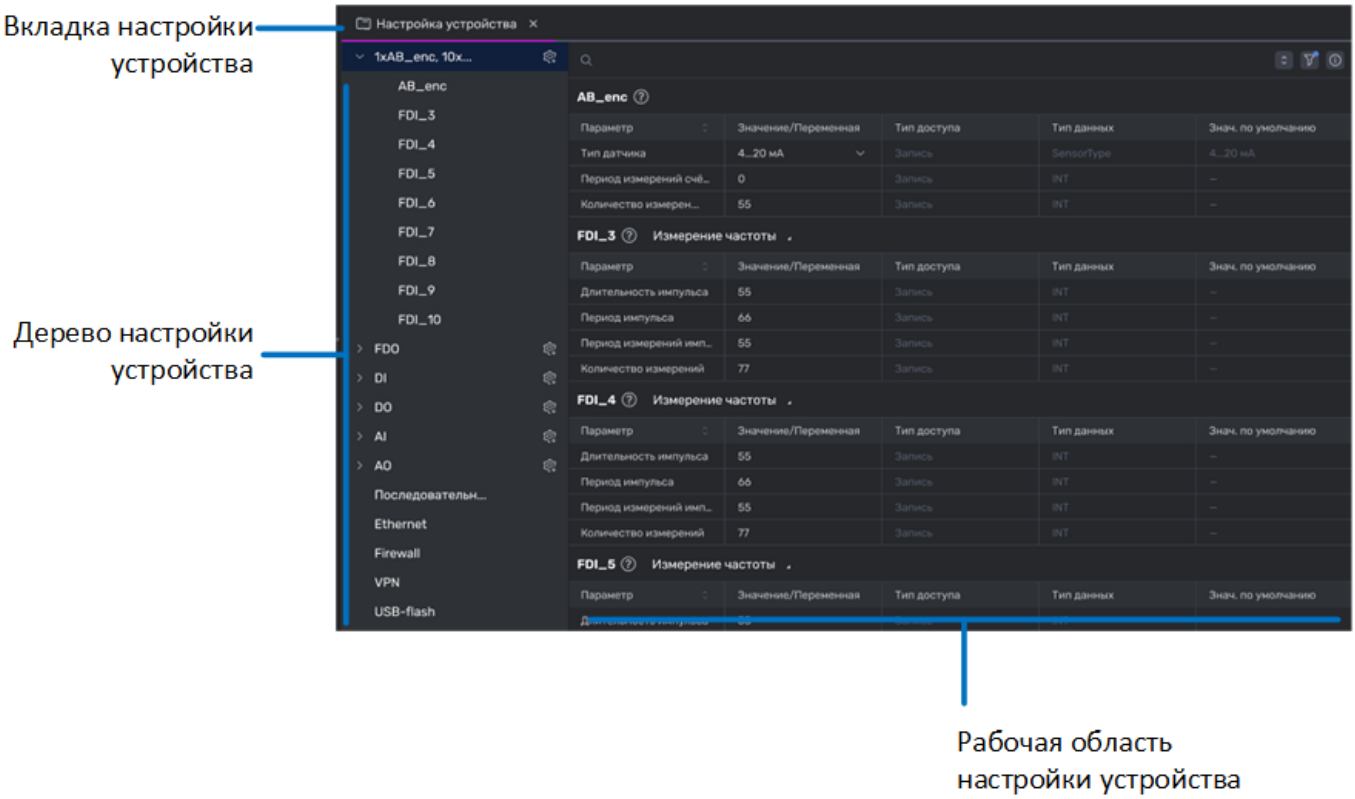


Рисунок 4.26 – Окно Настройки устройства

ПРИМЕЧАНИЕ
В ALTA IDE версии 1.0.0 доступна только настройка входов/выходов.

Настройка входов/выходов

ПРИМЕЧАНИЕ
Наполнение области настройки входов/выходов зависит от устройства.

В дереве настройки устройства отображены группы входов/выходов:

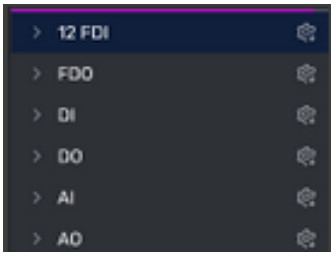


Рисунок 4.27

Для отображения входов/выходов, содержащихся в группе, нажмите на значок  , расположенный перед названием группы.

Если в группе возможен выбор конфигурации входа/выхода, то рядом с названием группы расположен значок  , который позволяет выбрать конфигурацию из предложенного списка:

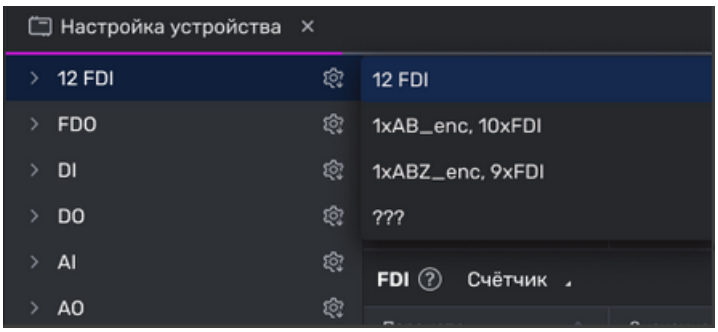


Рисунок 4.28

После выбора конфигурации в рабочей области отобразятся основные настройки входов/выходов для данной конфигурации:

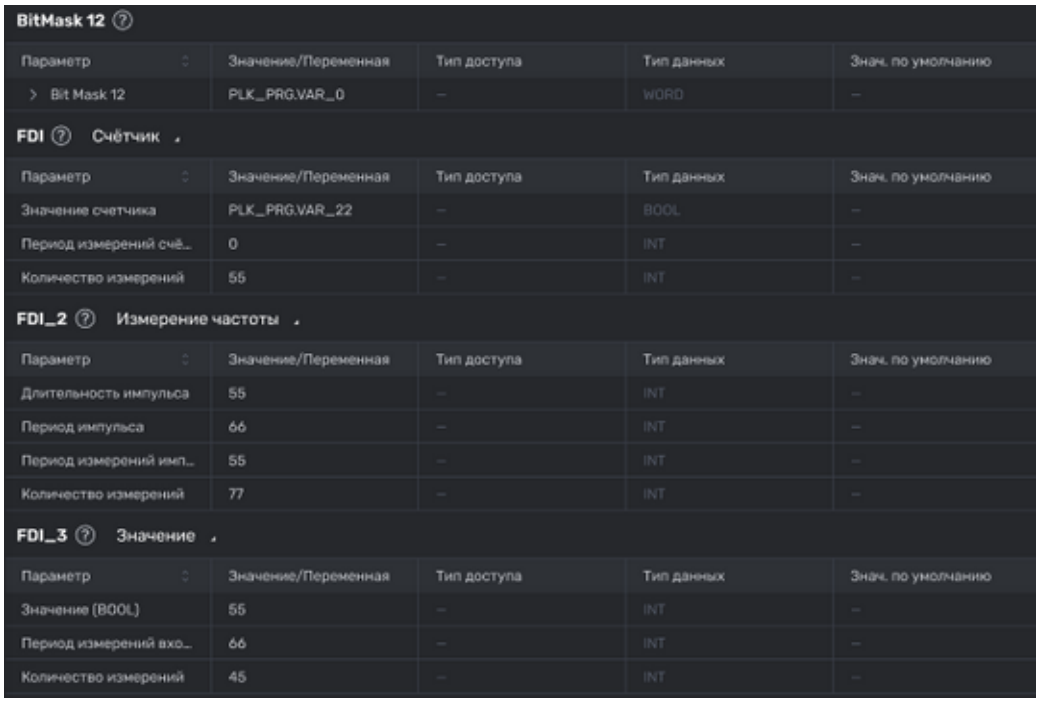
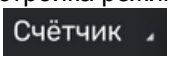


Рисунок 4.29

Если для входа/выхода предусмотрена настройка режима, то рядом с названием входа/выхода отображается название режима и треугольный маркер 

, при нажатии на который откроются доступные для выбора режимы:

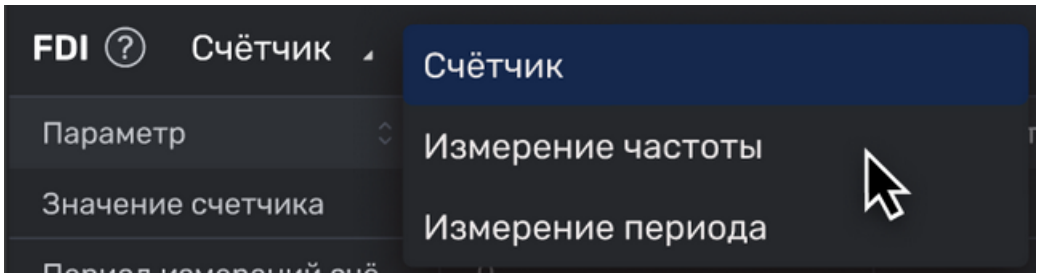


Рисунок 4.30

Если маркер отсутствует, значит выбор режима для данного входа/выхода невозможен.

Каждому входу/выходу соответствуют параметры, которые располагаются в таблице:

- **Параметр** - наименование параметра (не редактируемое поле). При нажатии на значок  осуществляется сортировка по лексикографическому порядку;
- **Значение/Переменная** - ввод значения или привязка переменной к параметру;
- **Тип доступа** - чтение/запись (не редактируемое поле);

- **Тип данных** - тип данных параметра (не редактируемое поле);
- **Знач. по умолчанию** - значение по умолчанию (не редактируемое поле);
- **Мин. значение** - минимальное значение параметра (не редактируемое поле);
- **Макс. значение** - максимальное значение параметра (не редактируемое поле);
- **Комментарий** - пользовательский комментарий.

4.3 Сохранение и сборка проекта

Сохранение проекта

Несохраненные изменения обозначаются в ALTA IDE значком * рядом с названием вкладки:

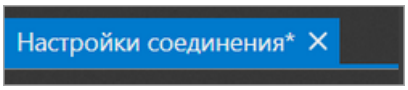


Рисунок 4.31

При попытке закрыть вкладку, содержащую несохраненные изменения, отобразится информационное окно с предложением сохранить изменения. В случае выбора "Да" информация, содержащаяся во вкладке сохранится в проекте.

Сохранить все изменения в проекте можно с помощью пункта **Главное меню→Файл→Сохранить проект**, сочетания клавиш Ctrl+S, или воспользуйтесь кнопкой **Сохранить проект**  на панели инструментов. Информация будет сохранена в папку проекта, по пути, указанному при создании проекта.

Сборка проекта

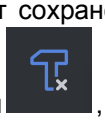
После создания проекта, разработки алгоритма, добавления задач с привязанными к ним программными объектами и ввода настроек, необходимо выполнить сборку проекта. Для этого нажмите кнопку  на панели инструментов. ALTA IDE автоматически выполнит сохранение всех вкладок, в которых произошли изменения. Кнопка перейдет в статус реализации сборки  , в строке состояния отобразится шкала с индикацией процесса компиляции:

Рисунок 4.32 – Шкала сборки проекта

ПРИМЕЧАНИЕ
Программные объекты, не привязанные к задачам, не попадают в файл с исполняемым приложением.

После окончания сборки файл с приложением будет сохранен в папку *...\{Название папки с проектом Alta}/Build/*.

Информационные сообщения, предупреждения, а также возможные ошибки, возникшие в процессе компиляции, будут отображаться в окне вывода, расположенном в нижней части основного окна ALTA IDE.

Процесс сборки проекта можно остановить, для этого нажмите кнопку  на панели инструментов, или нажмите на крестик на шкале сборки проекта, расположенной в строке состояния главного окна ALTA IDE. Процесс сборки проекта будет остановлен.

4.4 Подключение устройства к ПК

Подключите устройство к ПК одним из доступных способов, указанных в Руководстве по эксплуатации прибора.

Подключение к ALTA IDE доступно по

- USB;
- Ethernet.

ВНИМАНИЕ
Модификация и версия прошивки подключенного прибора должна совпадать с загруженным таргетом. В противном случае соединение с устройством не будет установлено.

Для работы в операционной системе Windows следует установить драйвер RNDIS. Драйвер доступен в WEB-конфигураторе на странице **Загрузки** или на сайте www.owen.ru.

Настройка соединения

При создании проекта и первом подключении устройства требуется настройка соединения с устройством. Для этого выберите в главном меню **Онлайн→Настройки соединения**, или нажмите **Настройки соединения** в контекстном меню устройства. Откроется окно настроек соединения:

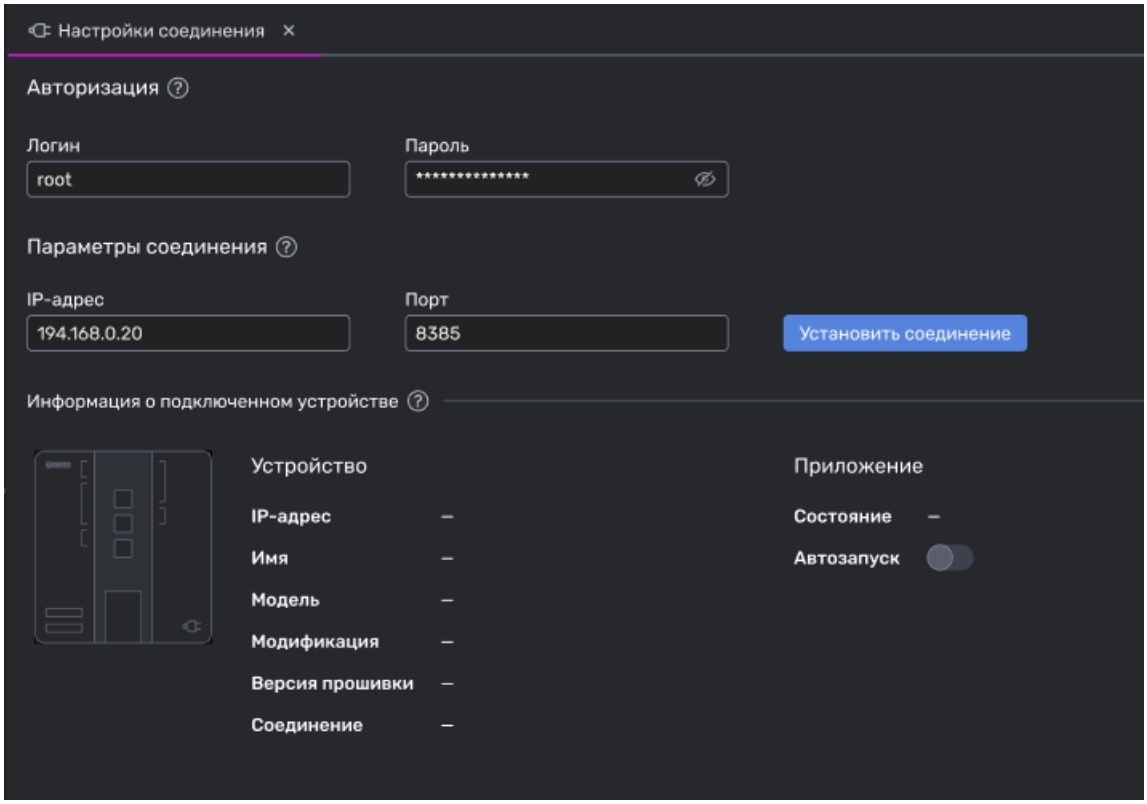


Рисунок 4.33 – Окно Настройки соединения

Введите данные:

- Логин;
- Пароль;
- IP-адрес;
- Порт.

ПРИМЕЧАНИЕ
Предустановленные заводские настройки указаны в Руководстве по эксплуатации прибора. Изменение настроек доступно в web-конфигураторе.

В случае неверно введенных данных (значение отсутствует, либо введен неверный формат) поле ввода будет выделено красным, при наведении мышки появится подсказка, содержащая информацию об ошибке.

В правом нижнем углу рабочей области отобразится уведомление об ошибке:

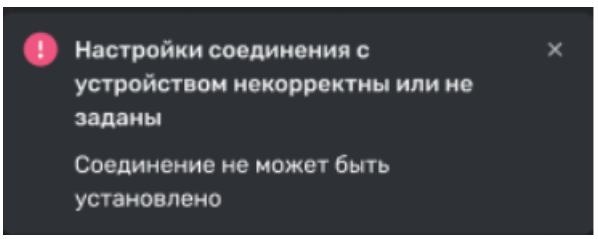


Рисунок 4.34 – Уведомление об ошибке

Введите корректные данные и нажмите **Установить соединение**. Соединение с устройством будет установлено, появится всплывающее сообщение, в окне отобразится информация о подключенном устройстве и работе приложения:

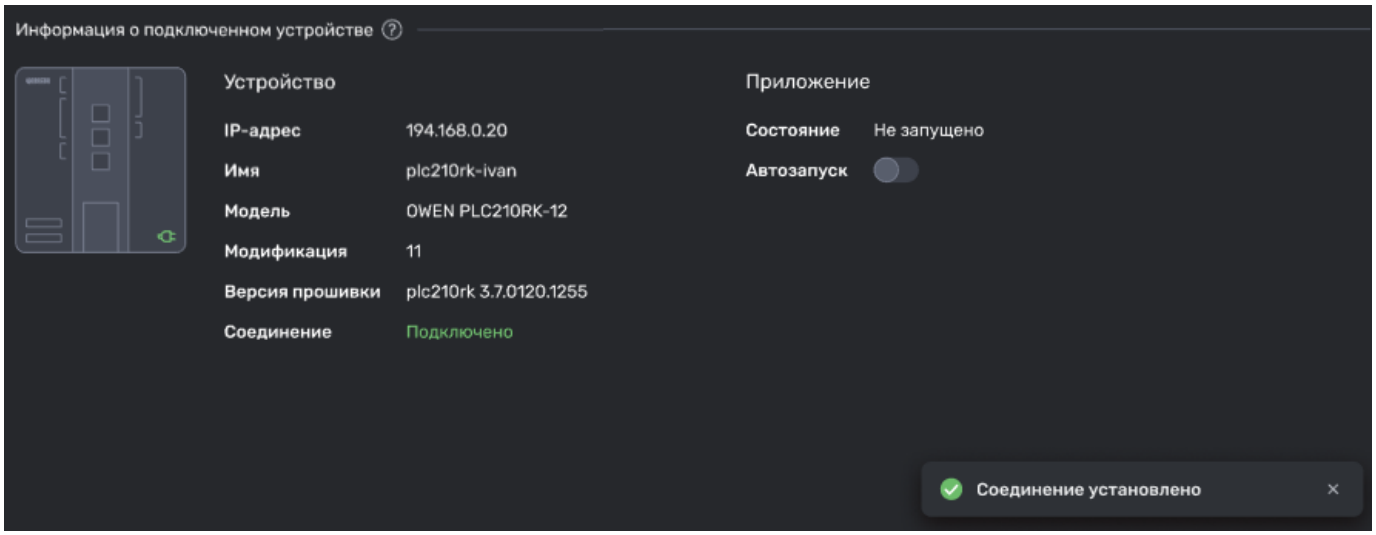


Рисунок 4.35

В окне настроек соединения можно включить возможность автоматического запуска приложения при подключении устройства, для этого переведите выключатель Автозапуск в положение включено.

После сохранения проекта настройки соединения сохранятся, и повторный ввод не потребуется.

В случае потери соединения с устройством отобразится уведомление:

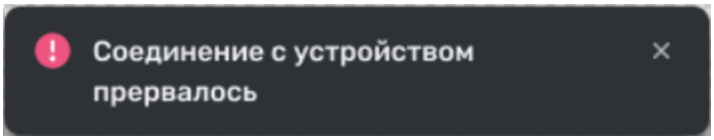


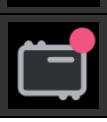
Рисунок 4.36

В строке **Соединение** будет указан статус **Ошибка**.

Установить соединение

В проекте с ранее введенными настройками подключения установить соединение с подключенным устройством можно с помощью кнопки  расположенной на панели инструментов, либо выберите **Установить соединение** в контекстном меню элемента **Устройство**, или **Главное меню→Онлайн→Установить соединение**.

В дереве проекта и строке состояния отобразится статус подключенного устройства:

Дерево проекта	Строка состояния	Статус
		Не подключено
		Установка соединения
		Соединение установлено
		Ошибка соединения

Разорвать соединение

Разорвать соединение с устройством можно с помощью:

- контекстного меню устройства, выбрав **Разорвать соединение**;
- **Главное меню→Онлайн→Разорвать соединение**;
- кнопки **Разорвать соединение** в окне **Настройки соединения**;
- кнопки  расположенной на панели инструментов.

4.5 Запись приложения в прибор

Для записи приложения в прибор устройство должно быть подключено к ПК, соединение установлено. Если на устройстве запущено выполнение приложения, то выполнение будет остановлено.

Нажмите кнопку  на панели инструментов, либо выберите **Загрузить приложение** в контекстном меню элемента Устройство, или **Главное меню→Онлайн→Загрузить приложение**. Перед загрузкой автоматически будет выполнена сборка проекта. После компиляции начнется загрузка приложения в прибор, кнопка перейдет в статус загрузки  , также процесс загрузки отобразится в строке состояния:

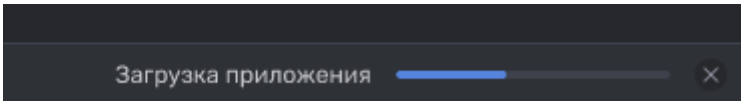


Рисунок 4.37 – Шкала загрузки приложения

Созданное приложение записывается в ПЗУ прибора.

ВНИМАНИЕ
Если в подключенном приборе уже есть ранее записанное приложение, то оно заменяется новым.

После окончания загрузки отобразится уведомление об успешной загрузке приложения.

Если на устройстве запущено выполнение приложения, при попытке записать новое приложение отобразится информационное окно с предложением остановить текущее приложение и выполнить загрузку. В случае согласия выполнение приложения будет остановлено, новое приложение будет загружено на устройство.

Возможные ошибки при загрузке приложения:

- потеряна связь с устройством;
- файл приложения превышает допустимый размер (максимальный размер исполняемого приложения зависит от прибора).

При возникновении ошибки появится всплывающее сообщение с информацией, загрузка приложения будет остановлена.

Процесс загрузки приложения в прибор можно остановить, для этого нажмите кнопку  на панели инструментов, или нажмите на крестик на шкале загрузке приложения, расположенной в строке состояния главного окна ALTA IDE. Процесс загрузки приложения будет остановлен.

4.6 Режим онлайн и отладка программы

Режим онлайн служит для мониторинга программы с реальными значениями со входов прибора. В режиме онлайн можно устанавливать значения переменных, задавать фиксированные значения на входы и выходы прибора. С помощью установки точек останова можно определять место в программе на котором будет остановлено выполнение и после остановки выполнять код по шагам. В режим онлайн можно перейти только при подключенном приборе и после записи приложения в устройство. Выполнение приложения должно быть запущено.

ПРИМЕЧАНИЕ
Проект, из которого выполняется переход в режим онлайн, должен быть идентичен проекту, загруженному в устройство.

В режиме онлайн внесение изменений в проект невозможно.

4.6.1 Переход в режим онлайн

Для запуска режима онлайн запустите выполнение приложения кнопкой  на панели инструментов, а затем нажмите кнопку перехода в режим онлайн  . Если приложение, загруженное в прибор идентично приложению для которого запускается режим онлайн, иконка станет зеленого цвета, на панели инструментов появится панель отладки.

В случае, если приложение загруженное в прибор не совпадает с запущенным приложением, отобразится информационное окно:

Рисунок 4.38

Загрузите актуальное приложение на устройство и выполните запуск режима онлайн повторно.

Если к устройству одновременно подключены несколько пользователей, и устройство уже находится в режиме онлайн, то при попытке перехода в режим онлайн другого пользователя возникнет ошибка, переход выполнен не будет:

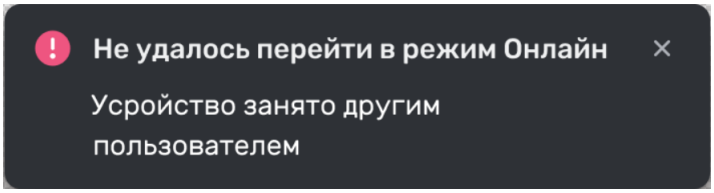


Рисунок 4.39

Для выхода из режима онлайн, например, чтобы внести корректировки в программу, нажмите кнопку  на панели инструментов.

4.6.2 Редактор ST в режиме онлайн. Отладка программы

При переходе в режим онлайн в рабочей области редактора ST возможна отладка программы:

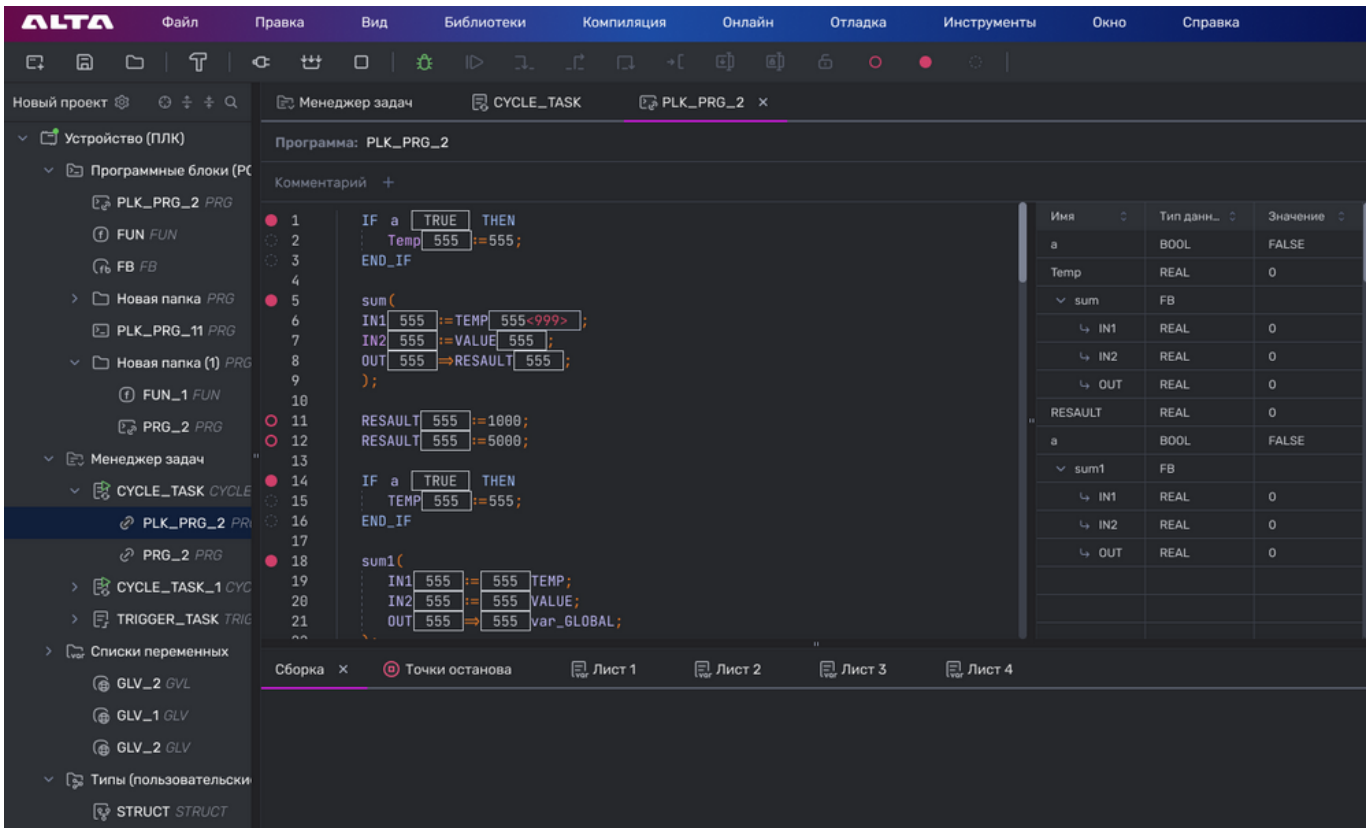


Рисунок 4.40 – Режим онлайн-отладки

- на панели инструментов отображается панель отладки;
- в коде программы отображаются окна просмотра для введения значения переменных и точки останова (при наличии);
- справа от рабочей области располагается таблица локальных переменных;
- в дереве проекта отображаются статусы задач;
- в окне вывода отображается список точек останова (при наличии) и листы просмотра переменных.

Значение переменной

Для ввода значения переменной дважды нажмите ЛКМ:

- в коде программы в окне просмотра, предназначенном для ввода значения переменной;
- в таблице локальных переменных в столбце "Значение";
- в Листе просмотра, расположенном в окне вывода, в столбце "Подготовленное значение".

Появится окно ввода значения переменной:

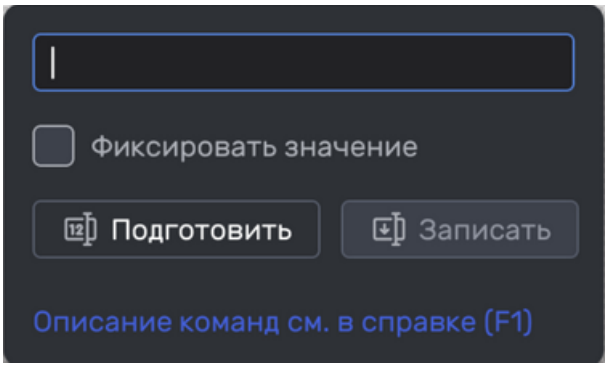


Рисунок 4.41

Фиксировать значение — фиксация значения переменной. Зафиксированное значение переопределяет текущее значение переменной, переменная сохраняет свое значение на границах цикла: до начала выполнения программы и после.

Подготовить — подготовка значения переменной для последующей записи.

Записать — запись значения переменной. Если переменная имеет подготовленное значение, то происходит запись подготовленного значения переменной.

Введите необходимое значение переменной и нажмите **Подготовить** - переменная не примет подготовленное значение до тех пор, пока не будет выполнена команда записи.

Рисунок 4.42 – Отображение подготовленного значения переменной в коде программы

Если ввести недопустимое значение переменной отобразится информационное сообщение об ошибке.

Если необходимо подготовить фиксированное значение переменной, установите чекбокс **Фиксировать значение**, введите значение переменной и нажмите **Подготовить**, подготовленное значение отобразится справа от текущего значения переменной. При записи данное значение запишется с фиксацией:

Рисунок 4.43 – Отображение фиксированного подготовленного значения переменной в коде программы

Можно подготовить значение для зафиксированного значения переменной. В зависимости от установки чекбокса **Фиксировать значение** подготовленное значение переменной будет либо зафиксировано, либо нет:

Рисунок 4.44 – Подготовленное значение переменной не фиксировано

Рисунок 4.45 – Подготовленное значение переменной фиксировано

Для записи ранее подготовленного значения переменной откройте окно ввода значения переменной и нажмите **Записать**, переменная примет подготовленное значение. Также можно записать значение без подготовки, для этого в окне ввода введите необходимое значение и нажмите **Записать**. Переменная примет введенное значение.

Если значение переменной циклически изменяется в самом проекте (например, эта переменная привязана ко входу устройства, или же в программе ей присваивается значение другой переменной), то при мониторинге переменная будет принимать значение полученное в ходе выполнения программы. Чтобы переменная сохраняла введенное значение, при записи значения установите чекбокс **Фиксировать значение**. Значение переменной может измениться в ходе выполнения программы, но после окончания цикла вернется к зафиксированному.

Записать все подготовленные значения переменной можно с помощью кнопки  на панели отладки, или с помощью пункта **Записать все значения** в **Главном меню→Отладка**.

Чтобы снять фиксацию значений у всех переменных в проекте воспользуйтесь кнопкой  на панели отладки, или с помощью пункта **Снять фиксацию у всех переменных** в **Главном меню→Отладка**.

Чтобы закрыть окно ввода переменной нажмите Esc или ЛКМ в любой области вне окна.

Точки останова

Точки останова задаются для остановки выполнения программы на определенной строчке. При достижении точки останова приложение будет автоматически останавливаться. Это позволяет, например, выполнять по шагам программные блоки, или определять какие значения имеют переменные проекта в момент выполнения определенных условий. Точки останова обозначаются слева от нумерации строк программного кода:

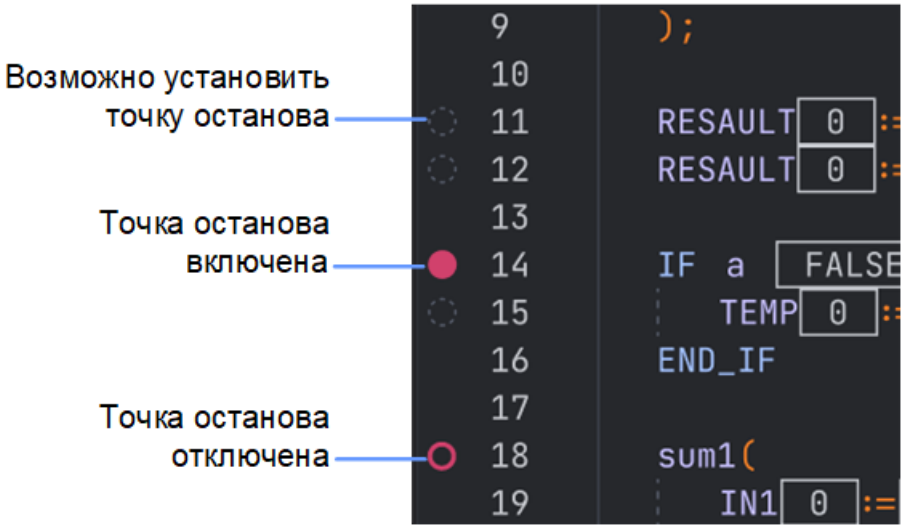


Рисунок 4.46

Для просмотра всех точек останова в программе перейдите во вкладку **Точки останова** в **Окне вывода**:

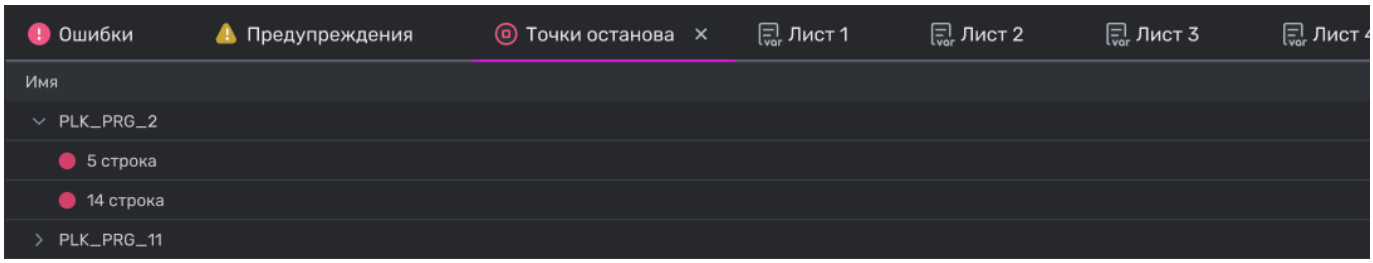


Рисунок 4.47 – Вкладка Точки останова в Окне вывода

Чтобы установить точку останова наведите мышку на маркер возможного места установки  слева от программного кода и нажмите ЛКМ. Точка останова установлена. Удалить включенные или отключенные точки можно повторным нажатием ЛКМ на выбранную точку.

ПРИМЕЧАНИЕ
Точки останова можно установить не во всех местах кода. Возможные места обозначены маркером места установки.

Включить, отключить или удалить точки останова можно также с помощью с кнопок напанели отладки или контекстного меню: наведите мышку на необходимую точку слева от программного кода или во вкладке **Точки останова** в **Окне вывода**, нажмите ПКМ и выберите необходимое действие.

Пошаговая отладка

Когда выполнение программы остановлено в определенной точке, можно просмотреть значения переменных проекта в данный момент времени. Выполняя проект в пошаговом режиме есть возможность проверить логическую корректность программы.

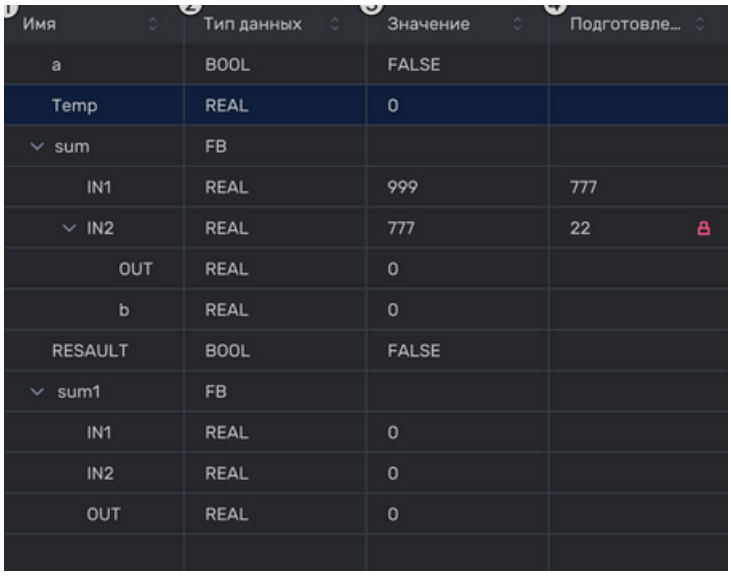
Чтобы перейти к месту остановки выполнения программного кода нажмите кнопку  на панели отладки или в **Главном меню→Отладка**. Строка на которой было приостановлено выполнение программы будет выделена.

Для возобновления выполнения приложения после остановки нажмите кнопку  на панели отладки, или используйте команды для пошагового выполнения программы расположенные на панели отладки, либо в **Главном меню→Отладка**:

	Шаг внутрь	Выполнение текущей строки кода. Если в строчке вызывается программный блок, заходит внутрь программного блока.
	Шаг наружу	Продолжает выполнение всего кода текущего программного блока и возврат к строчке кода, откуда произошел переход внутрь программного блока
	Шаг поверх	Выполнение текущей строки кода. Если в строчке вызывается программный блок, то код программного блока выполняется полностью, без захода внутрь

4.6.3 Таблица локальных переменных

Переменные проекта отображаются в таблице переменных, которая доступна для просмотра только в режиме онлайн, и располагается справа в рабочей области редактора программного объекта:



Имя	Тип данных	Значение	Подготовле...
a	BOOL	FALSE	
Temp	REAL	0	
sum	FB		
IN1	REAL	999	777
IN2	REAL	777	22
OUT	REAL	0	
b	REAL	0	
RESAULT	BOOL	FALSE	
sum1	FB		
IN1	REAL	0	
IN2	REAL	0	
OUT	REAL	0	

Рисунок 4.48 – Таблица локальных переменных

Каждому программному компоненту соответствует своя таблица переменных. Каждая переменная отображается в таблице в порядке объявления, в единственном экземпляре, вне зависимости от количества мест использования в программе.

Таблица переменных содержит:

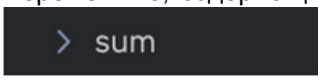
Имя - имя отображения в коде программы;

Тип данных - может быть назначен один из типов: BYTE, WORD, DWORD, INT, DINT, REAL, STRING80, STRING21. См. подробнее раздел **Типы переменных**

Значение - текущее значение переменной. Значение может быть фиксированным, в этом случае переменная сохраняет свое значение на границах цикла. В случае отсутствия фиксации переменная будет принимать значение полученное в ходе выполнения программы;

Подготовленное значение - значение, которое будет присвоено переменной после команды записи переменной.

Переменные, содержащие дочерние переменные, отображаются в таблице со значком  перед именем:



Для того чтобы развернуть отображение дочерних переменных нажмите на значок  , в таблице отобразятся вложенные переменные, которые также могут содержать дочерние переменные.

Чтобы задать значение переменной дважды нажмите ЛКМ в столбце подготовленное значение. Откроется окно ввода данных.

Контекстное меню значения переменной

Для вызова контекстного меню значения переменной нажмите ПКМ:

- в таблице локальных переменных;
- в коде программы в окне просмотра, предназначенном для ввода значения переменной;
- в Листе просмотра, расположенном в окне вывода.

Контекстное меню содержит:

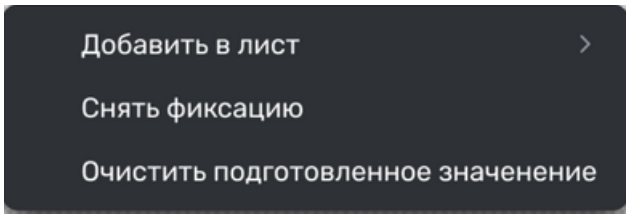


Рисунок 4.49

Добавить в лист - добавление переменной в выбранный лист просмотра переменных.

Переменные, которые содержат вложенные данные (родительские переменные), такие как **структуры**, **массивы** и другие составные типы, автоматически добавляются в лист просмотра переменных вместе со всеми своими вложенными элементами (дочерними переменными).

- **Родительская переменная** — это переменная составного типа, включающая в себя другие переменные.
- **Дочерние переменные** — это отдельные элементы, входящие в состав родительской переменной, например поля структуры или элементы массива.

Если в лист просмотра переменных добавить только дочернюю переменную, то родительская переменная при этом не будет добавлена. Однако к имени такой переменной автоматически будет добавлен префикс с именем родительской переменной.

Например для структуры *Device.Status* дочерняя переменная *ErrorCode* будет отображена как *Device.Status.ErrorCode*.

Снять фиксацию - освобождение значения переменной от фиксации.

Очистить подготовленное значение - удаляет подготовленное значение переменной.

4.6.4 Листы просмотра переменных

Листы просмотра переменных доступны только в режиме онлайн, отображаются в окне вывода и содержат созданные пользователем списки переменных для наблюдения за их значениями (с возможностью изменения при помощи команд Записать и Фиксировать значение в окне ввода значений переменных). Переменные, располагающиеся в одном листе просмотра, могут быть объявлены в разных программных объектах проекта.

Чтобы создать лист просмотра нажмите кнопку  **Добавить лист**, расположенную на тулбаре окна вывода.

Для добавления переменной в лист просмотра воспользуйтесь контекстным меню значения переменной:

- в коде программы в окне просмотра, предназначенном для введения значения переменной;
- в таблице локальных переменных;
- в Листе просмотра, расположенном в окне вывода.

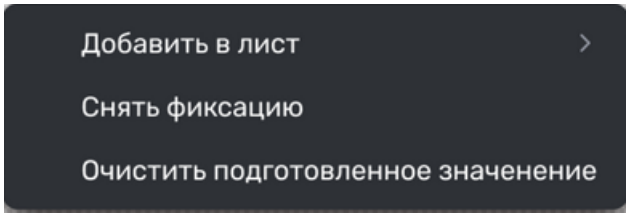


Рисунок 4.50

Выберите пункт **Добавить в лист** и в выпадающем меню выберите лист - переменная будет добавлена в список, расположенный в этом листе.

5 Modbus

Modbus – открытый коммуникационный протокол, основанный на архитектуре Master-Slave (ведущий-ведомый).

Master (ведущее устройство) - является инициатором обмена и может считывать и записывать данные в slave-устройства.

Slave (подчиненное устройство) - отвечает на запросы master-устройства, но не может самостоятельно инициировать обмен.

В ALTA IDE используются две основные реализации протокола:

1. **Modbus Serial** для передачи данных с использованием последовательных интерфейсов. Имеет два режима передачи данных:
   - **Modbus RTU** - передача данных в бинарном виде;
   - **Modbus ASCII** - передача данных в виде ASCII символов;
2. **Modbus TCP** для передачи данных через сети TCP/IP.

Устройства в режиме Slave определяют протокол в зависимости от выбранного интерфейса: UART или TCP. Для устройств в режиме Master необходимо выбрать протокол при добавлении.

Для организации обмена данными в сети через интерфейс связи необходимо устройство в режиме Master для инициации обмена данными.

В случае использования протокола **Modbus Serial** в сети может присутствовать только одно master-устройство и несколько slave-устройств.

В сети **Modbus TCP** на один IP-порт можно подключить до 255 опрашиваемых устройств. Кроме того, устройство может одновременно выполнять функции master и slave.

Запрос master-устройства к slave-устройству содержит:

- **Slave ID** — адрес slave-устройства;
- **Код функции** — применяемый к slave-устройству;
- **Данные** – адрес первого регистра и их количество (в случае записи – также записываемые значения).
- **Контрольную сумму** — для проверки целостности доставленного пакета.

Ответ slave-устройства имеет схожую структуру.

Запрос master-устройства представляет собой обращение к одной из областей памяти slave-устройства с помощью определенной функции. Область памяти характеризуется типом хранящихся в ней значений (биты/регистры) и типом доступа (только чтение или чтение и запись).

Спецификация Modbus определяет 2 варианта размещения данных в памяти slave-устройств:

- независимые области памяти;
- общая область памяти.

В ALTA IDE поддержан второй вариант с общей областью памяти: к одним и тем же данным можно получить доступ с помощью нескольких функций Modbus.

Для конфигурируемых устройств производитель предоставляет карту регистров, в которой содержится информация об адресах и типах параметров устройства. Для программируемых устройств пользователь формирует такую карту самостоятельно с помощью среды разработки. Существуют устройства, в которых сочетаются оба рассмотренных случая – у карты регистров есть фиксированная часть, которую можно дополнить в соответствии со своей задачей (но адреса ячеек не должны пересекаться).

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

Функция определяет операцию (чтение/запись) и область памяти, с которой эта операция будет произведена. Ниже приведен список наиболее часто используемых функций:

Таблица 5.1 – Основные функции протокола Modbus

Код функции	Имя функции	Выполняемая команда
3 (0x03)	Read Holding Registers	Чтение значений из регистров хранения
4 (0x04)	Read Input Registers	Чтение значений из регистров ввода

Продолжение таблицы 5.1

Код функции	Имя функции	Выполняемая команда
6 (0x06)	Write Single Register	Запись значения в один регистр хранения
16 (0x10)	Write Multiple Registers	Запись значений в несколько регистров хранения

Настройка обмена по протоколу Modbus в ALTA IDE представлена на схеме:

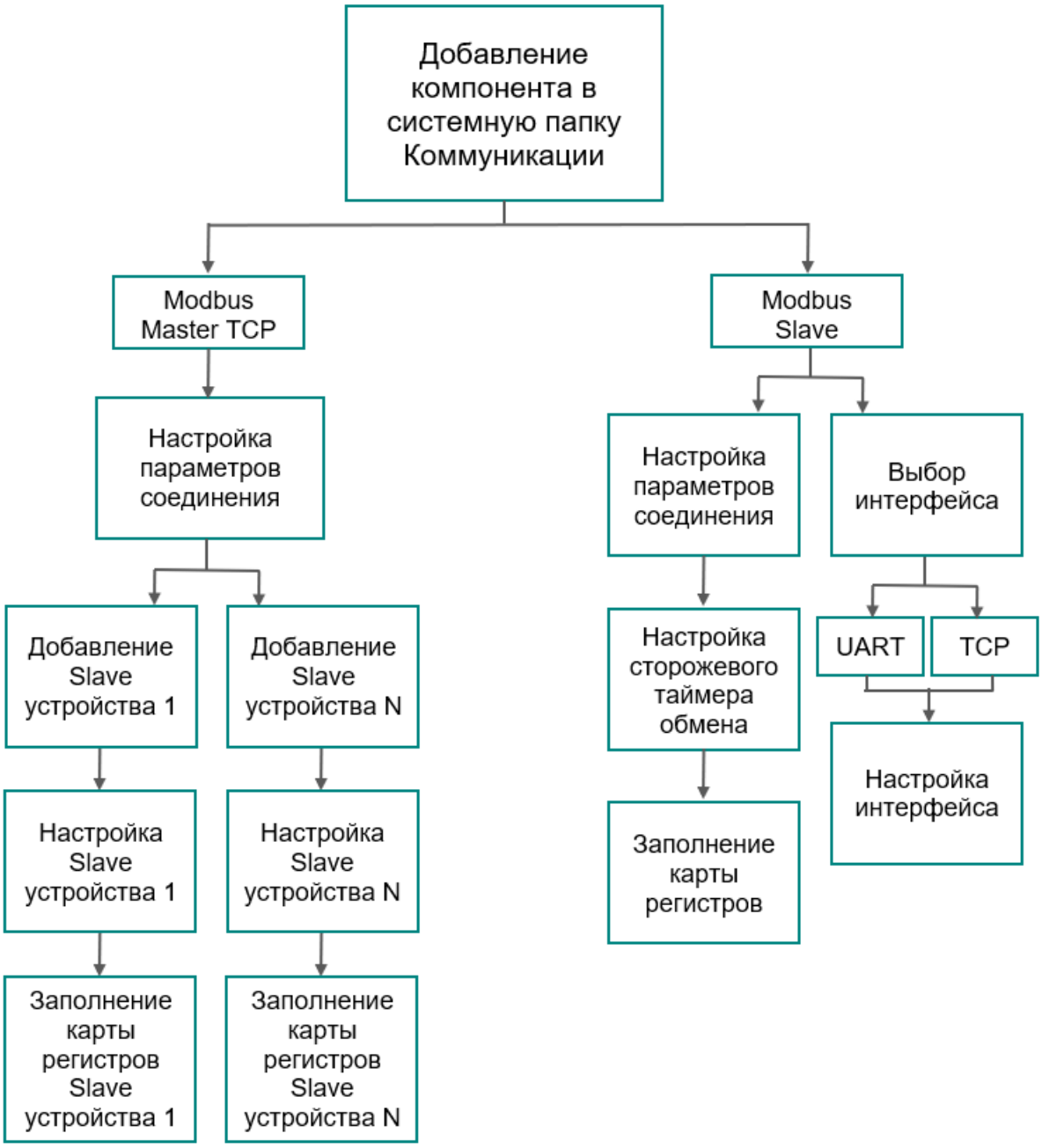


Рисунок 5.1

5.1 Режим Master

Modbus Master — это устройство, которое инициирует обмен данными в сети по протоколу Modbus, отправляя запросы ведомым устройствам (slave) и получая от них ответы. Master выступает в роли ведущего устройства, контролирующего работу остальных устройств в системе.

Для добавления устройства в режиме Master воспользуйтесь контекстным меню системной папки Коммуникации в дереве проекта:

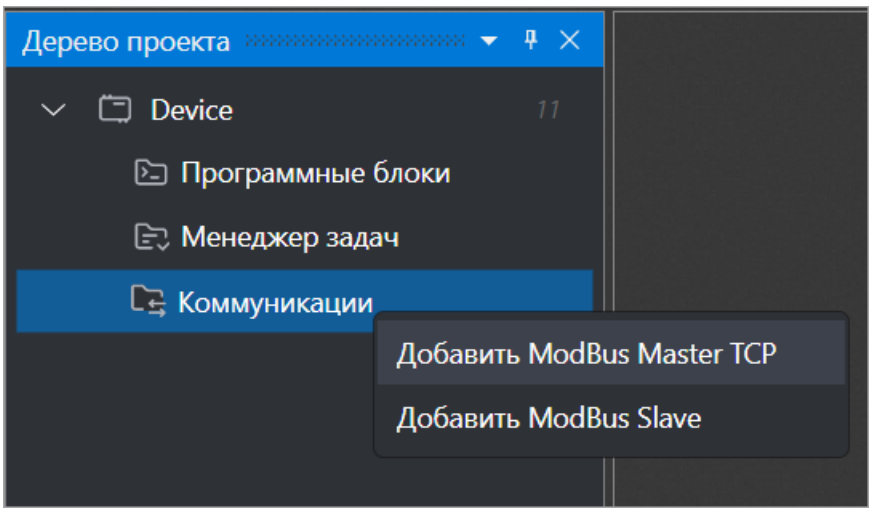


Рисунок 5.2

В ALTA IDE возможно добавление устройства в режиме Master, работающих по одному их двух протоколов: Modbus TCP и Modbus Serial.

ПРИМЕЧАНИЕ
В ALTA IDE версии 1.0.0 поддержана работа по протоколу Modbus TCP.

5.1.1 Modbus TCP Master

Modbus TCP — промышленный протокол передачи данных, используемый для обмена информацией между устройствами, для работы по Ethernet.

После добавления устройство появится в дереве проекта, как ответвление системной папки:

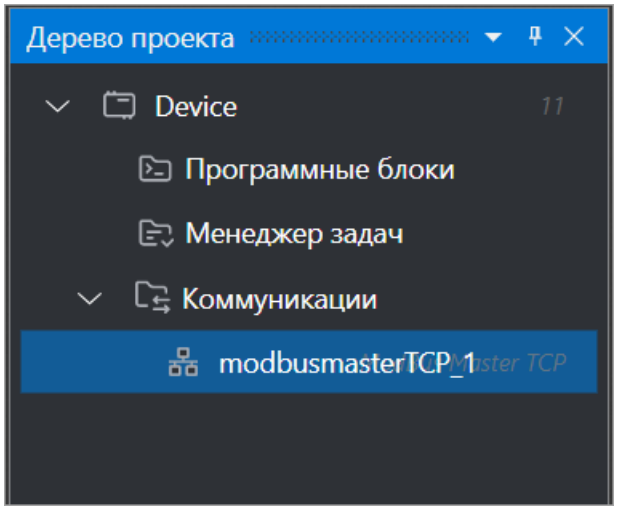


Рисунок 5.3

В открывшейся вкладке редактора Modbus TCP Master настройте параметры соединения:

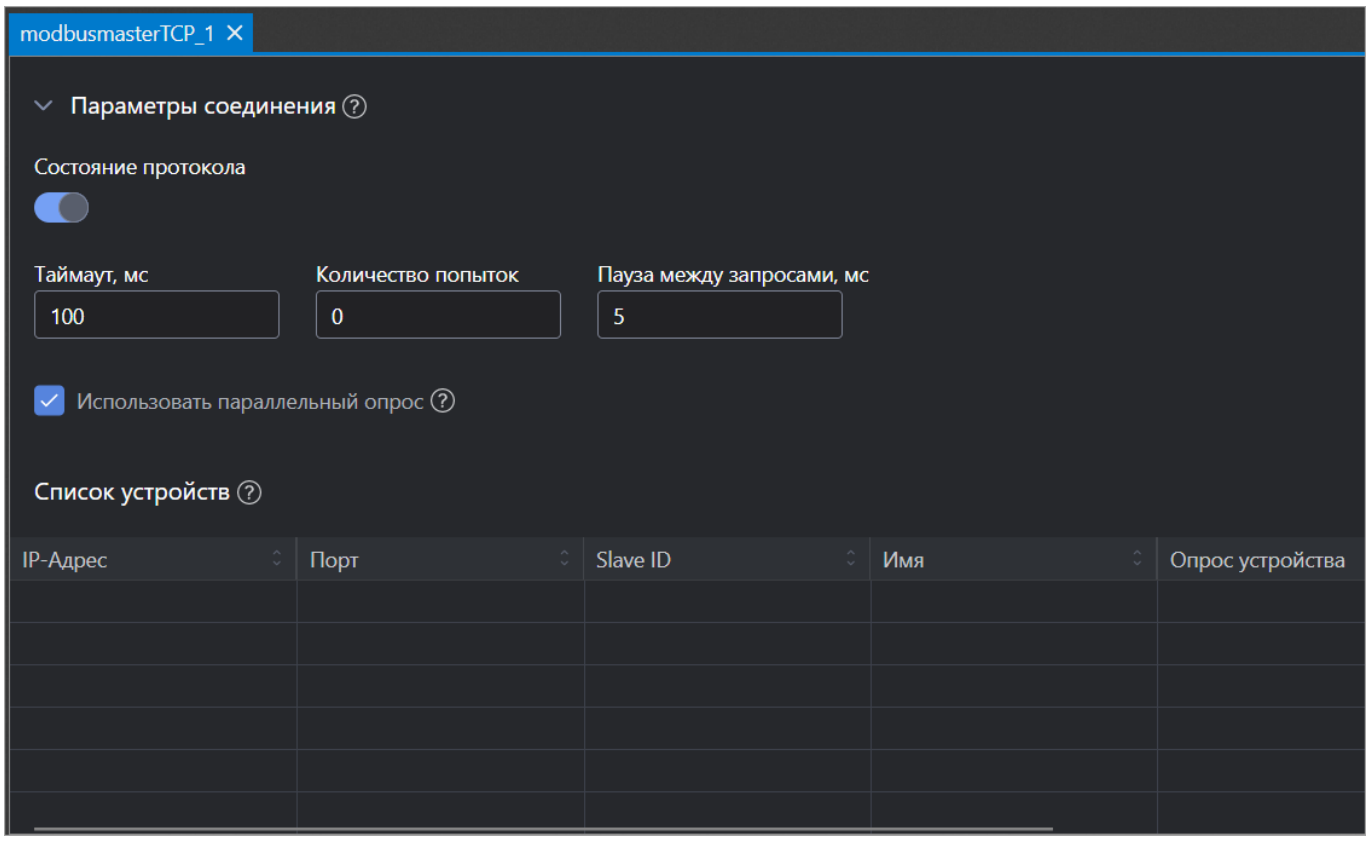


Рисунок 5.4

Состояние протокола — определяет режим работы Master-устройства при запуске приложения. В зависимости от положения переключателя отправляет/не отправляет запросы к slave-устройствам;

Таймаут, мс — время ожидания ответа от slave-устройства, допустимый диапазон от 20 до 6000 мс. Значение, введенное здесь, будет по умолчанию использоваться для всех slave-устройств, если в редакторе Modbus Device TCP не введены индивидуальные значения данного параметра;

Количество попыток - число повторных попыток установить связь со slave-устройством при отсутствии ответа. Значение, введенное здесь, будет по умолчанию использоваться для всех slave-устройств, если в редакторе Modbus Device TCP не введены индивидуальные значения данного параметра;

Пауза между запросами, мс - интервал ожидания перед формированием следующего запроса;

Использовать параллельный опрос - возможность выполнять параллельный опрос нескольких slave-устройств для повышения скорости обмена.

Таблица **список устройств** формируется автоматически при добавлении slave-устройств к master-устройству, и содержит следующие столбцы:

IP-Адрес - сетевой адрес устройства, диапазон значений от 0.0.0.0 до 255.255.255.255;

Порт - порт, используемый для обмена, диапазон значений от 0 до 65535 (по умолчанию - 502);

Slave ID - адрес slave-устройства в сети Modbus, диапазон значений от 0 до 255;

Имя - отображаемое имя устройства в дереве проекта;

Опрос устройства - состояние опроса устройства (включен/отключен), определяемое в настройках соответствующего slave-устройства.

Комментарий - пользовательский комментарий.

При нажатии на значок  в заголовке столбца осуществляется сортировка данных. Редактирование параметров в таблице невозможно. Чтобы внести изменения, перейдите во вкладку соответствующего **Modbus TCP Slave Devise** и задайте необходимые значения. Данные в таблице **Список устройств** обновятся автоматически.

5.1.1.1 Modbus TCP Slave Device

Для добавления Slave-устройства к Modbus TCP Master воспользуйтесь контекстным меню компонента Modbus TCP Master:

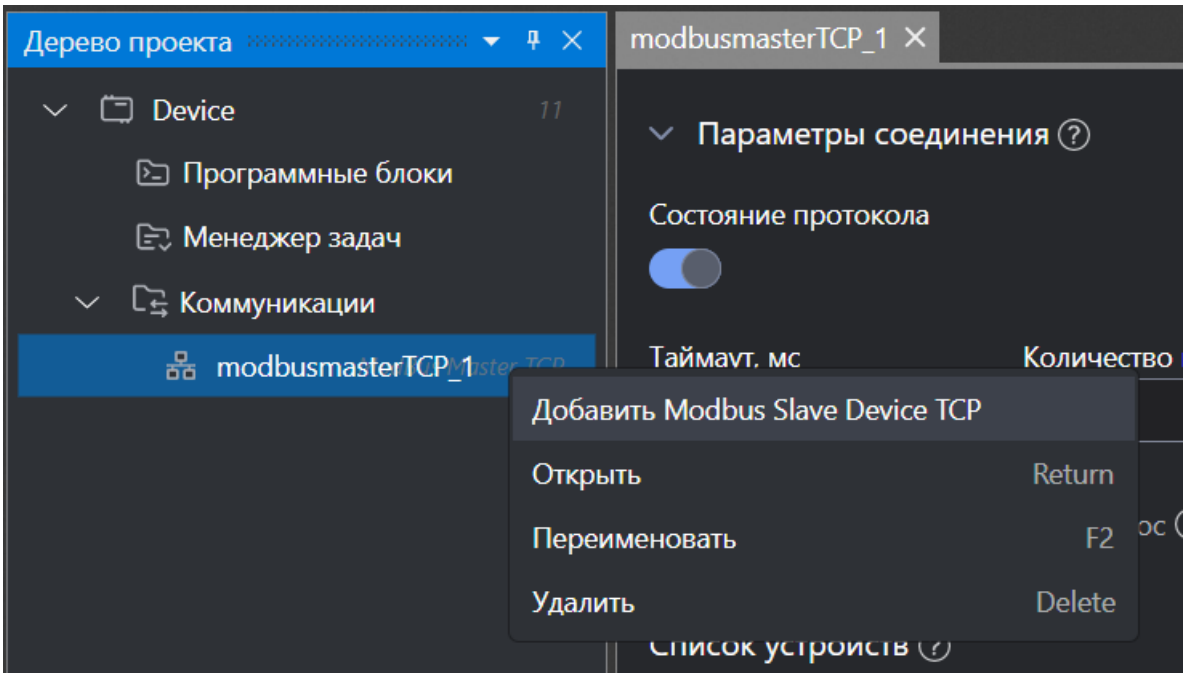


Рисунок 5.5

После добавления устройство появится в дереве проекта, как ответвление протокола Modbus TCP Master:

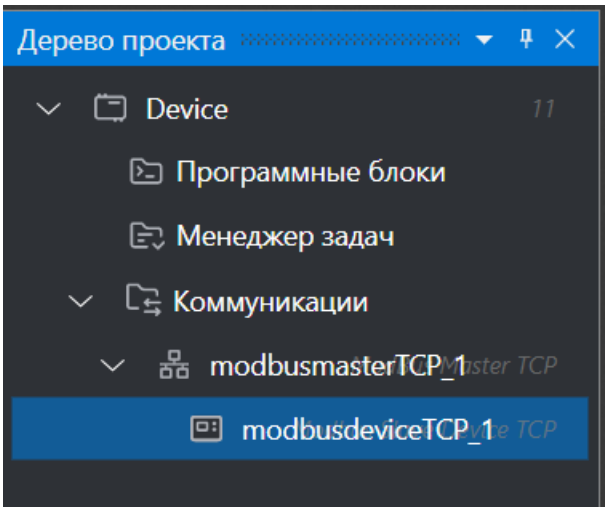


Рисунок 5.6

В открывшейся вкладке редактора Modbus Devise TCP доступна настройка параметров:

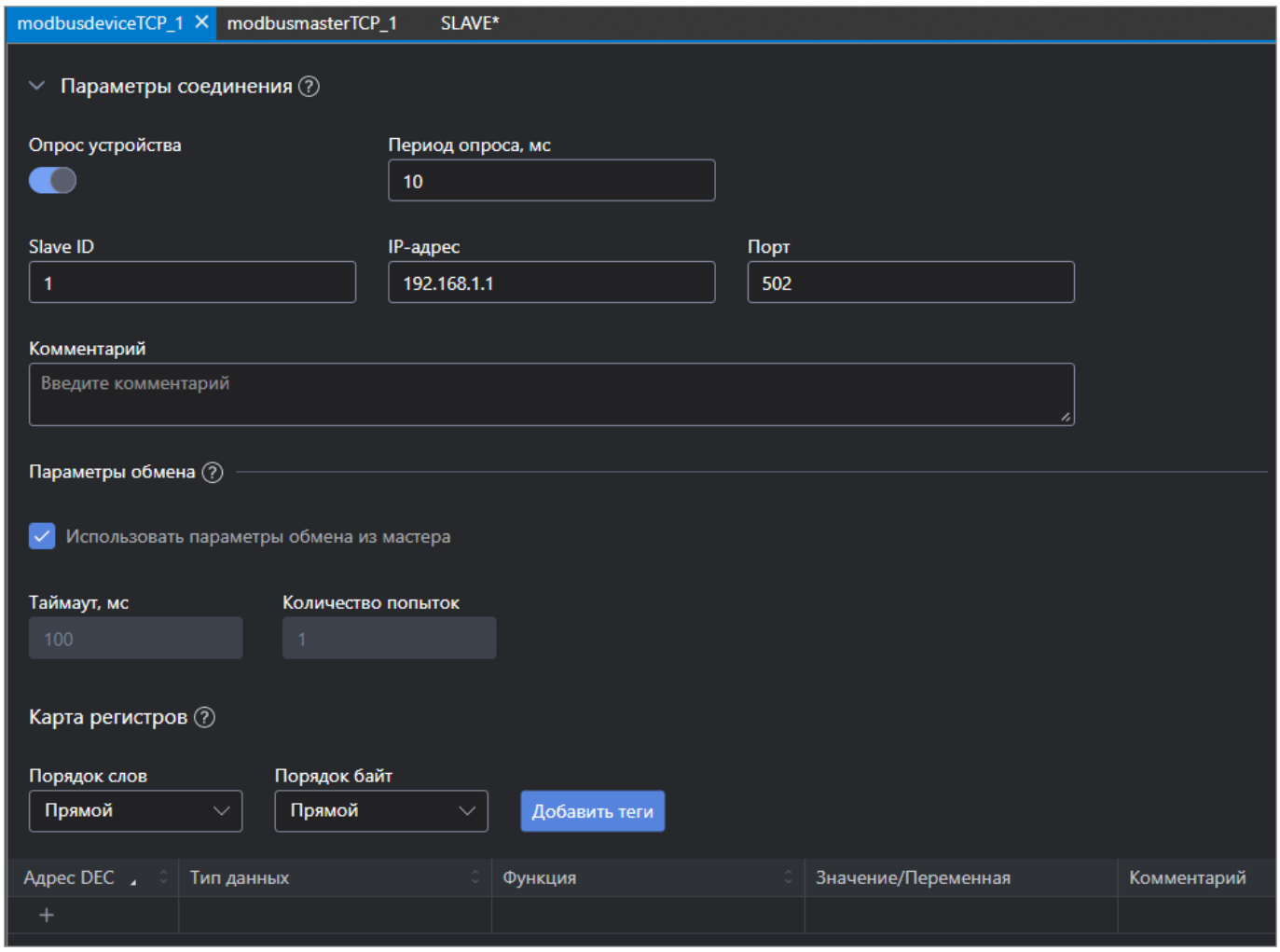


Рисунок 5.7

Параметры соединения:

Опрос устройства — состояние опроса устройства (включен/отключен). При отключенном опросе master-устройство не посылает запросы slave-устройству;

Период опроса — временной интервал, через который повторяется опрос. Допустимый диапазон от 0 до 6000 мс;

Slave ID — адрес slave-устройства в сети Modbus, диапазон значений от 0 до 255;

IP-Адрес — сетевой адрес устройства, диапазон значений от 0.0.0.0 до 255.255.255.255;

Порт — порт, используемый для обмена, диапазон значений от 1 до 65535 (по умолчанию - 502);

Комментарий — пользовательский комментарий. Отображается в таблице **Список устройств** вкладки Modbus TCP Master.

Параметры обмена:

Использовать параметры обмена из мастера - в случае установки чекбокса внесение данных будет недоступно, будут использованы значения, введенные во вкладке редактора Modbus TCP Master. Если чекбокс отключен, для устройства будут использоваться индивидуальные параметры обмена:

Таймаут, мс — время ожидания ответа от slave-устройства, допустимый диапазон от 20 до 6000 мс.

Количество попыток — число повторных попыток установить связь со slave-устройством при отсутствии ответа, допустимый диапазон от 0 до 3.

Настройте общие для всей карты регистров параметры:

Порядок слов/порядок байт — настройка порядка слов и порядка байт для передачи master-устройству.

Добавление тегов

После заполнения параметров необходимо добавить теги в **карту регистров.**

Для добавления тегов нажмите на кнопку **Добавить теги**. Откроется окно создания тегов:

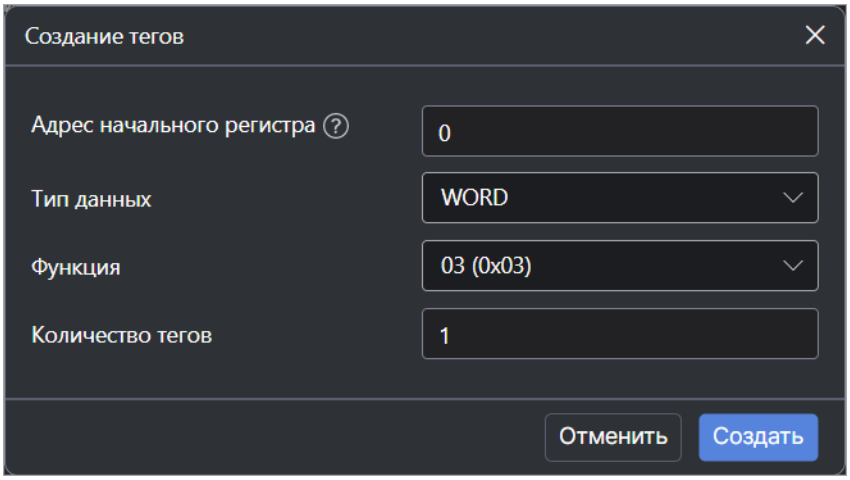


Рисунок 5.8

Адрес начального регистра — задает адрес начального регистра, адреса задаются начиная с указанного. Доступен ввод в десятеричном и шестнадцатеричном формате (с префиксом 16# и 0x);

Тип данных — тип данных тега;

Функция — определяет операцию (чтение/запись) и область памяти, с которой эта операция будет произведена. Список функций, поддерживаемых в ALTA IDE версии 1.0.0 приведен в разделе Modbus

Количество тегов — количество тегов, которое требуется создать.

Нажмите кнопку **Создать**. В карте регистров отобразится информация, согласно введенным данным. Редактирование значений в столбцах **Тип данных** и **Функция** выполняется нажатием ЛКМ по соответствующей строке и выбором нужного значения из выпадающего списка. Значение адреса редактируется двойным нажатием ЛКМ на строку и вводом нового значения. Для переключения формата ввода (**DEX/HEX**) нажмите на заголовок столбца Адрес. При нажатии на значок  в заголовке столбца осуществляется сортировка данных.

Для ввода данных в столбцы **Значение /переменная** и **Комментарий** дважды нажмите ЛКМ на строку:

Значение/переменная — позволяет привязать переменную или ввести её значение вручную.

- привязка переменной-входа для чтения тега в программе;
- привязка переменной-выхода для записи тега из программы;
- ввод значения (константы) для записи тега.

Комментарий - пользовательский комментарий.

Также создание тега доступно с помощью кнопки "+" расположенной в строке **Адрес таблицы регистров**. В этом случае теги добавляются по одному, начиная с адреса, следующего за адресом последнего отредактированного тега.

ПРИМЕЧАНИЕ
Созданные теги будут опрашиваться в порядке добавления в карту регистров, а не по возрастанию адресов.

Для переменных типа BYTE, WORD, DWORD и LWORD доступен просмотр и редактирование битовой маски. Подробно работа с битовой маской описана в разделе **Заполнение карты регистров**.

5.2 Режим Slave

В режиме Slave прибор предоставляет данные для считывания другому master-устройству/устройствам в сети, самостоятельный опрос не ведет.

Для добавления устройства в режиме Slave воспользуйтесь контекстным меню системной папки **Коммуникации** в дереве проекта:

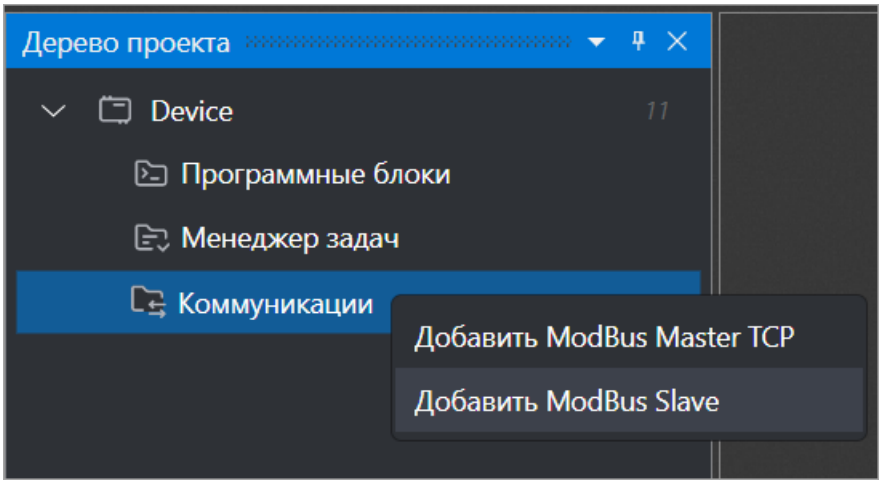


Рисунок 5.9 – Контекстное меню системной папки Коммуникации

Выберите **Добавить ModBus Slave**, устройство появится в дереве проекта, как ответвление системной папки:

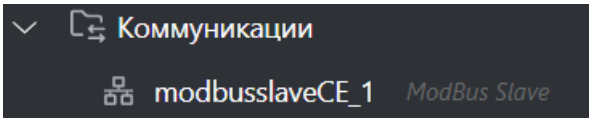


Рисунок 5.10 – Устройство Modbus Slave в дереве проекта

Выбор протокола не требуется - в режиме Slave протокол определяется в зависимости от выбранного интерфейса: UART или TCP.

В ALTA IDE Slave-устройства не поддерживают широковещательную рассылку.

После добавления устройства в режиме Slave откроется вкладка, содержащая параметры, которые необходимо настроить.

5.2.1 Настройка параметров Modbus Slave

Настройка параметров соединения

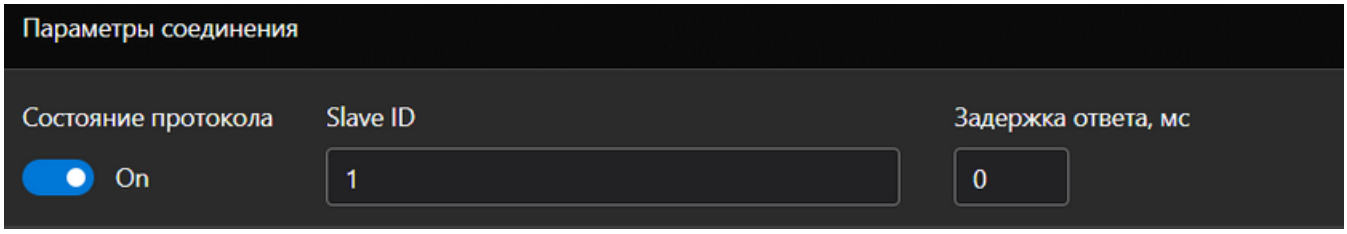


Рисунок 5.11

- **Состояние протокола** — включение/выключение протокола Modbus. В случае выключения протокол не будет обрабатывать входящие запросы.
- **Slave ID** — адрес устройства в сети Modbus. Допустимый диапазон от 0 до 255.
- **Задержка ответа, мс** — пауза перед ответом на запрос от master-устройства.

Настройка сторожевого таймера обмена

Сторожевой таймер обмена позволяет отслеживать наличие запросов от master-устройства.

Рисунок 5.12

- **Состояние таймера** — включение/выключение сторожевого таймера обмена.

- **Таймаут, мс** — время ожидания запроса от master-устройства. В случае отсутствия запроса за указанное время генерируется ошибка, которая отобразится в дереве проекта пиктограммой 

5.2.2 Заполнение карты регистров

Карта регистров используется для обмена данными в архитектуре master-slave. Заполнение карты регистров slave-устройства включает в себя определение назначения регистра, присвоение ему адреса в диапазоне от 0 до 65535 и заполнение данными, которые будут доступны для чтения, записи или чтения/записи master-устройством.

Правила выравнивания

В ALTA IDE происходит автоматическое выравнивание переменных, в зависимости от их типа, при размещении в области памяти.

Каждый тип переменных занимает определенное количество регистров и имеет правила размещения в карте:

Таблица 5.2 – Правила размещения переменных в карте регистров

Тип данных	Размер, байт	Количество занимаемых регистров	Адрес первого регистра
BYTE	1	1	в любом месте карты
WORD	2	1	в любом месте карты
DWORD	4	2	должен быть кратен 2 (в т.ч. 0)
INT	2	1	в любом месте карты
DINT	4	2	адрес первого регистра должен быть кратен 2 (в т.ч. 0)
REAL	4	2	адрес первого регистра должен быть кратен 2 (в т.ч. 0)
STRING80	80	40	в любом месте карты
STRING21	21	11	в любом месте карты

Правила размещения основаны на организации физической памяти таким образом, что переменные размером 8 бит (1 байт) , 2 байта, и 4 байта должны располагаться только по определенным адресам. Адрес 4-байтной переменной должен быть кратен 4, 2-байтной – кратен 2, а однобайтной (или 8 битной) – кратен 1 и может находиться в любой точке пространства памяти.

Если представить область памяти с возрастающими адресами (от 0 до какого-либо числа) и расположить переменные, то, если первая переменная имеет размер 1 байт, она будет расположена по адресу 0x0000, следующая – 0x0001 и т.д. Если дальше идет 4-байтная переменная, она должна располагаться по адресу 0x0004, т. е. кратному 4, и т. д. При этом, если однобайтная переменная заняла место, кратное четырем, следующая 4- байтная переменная занимает следующее кратное четырем место. Порядок задания переменных может быть произвольным, выравнивание же ставит переменные на кратные их длине адреса. Соответственно, при таком порядке размещения переменных неизбежно возникают не занятые пространства памяти, которые нигде не отображаются, не видны в области ввода/вывода, но обязательно должны учитываться пользователем: когда производится опрос прибора извне для получения информации, размещенной по конкретному адресу (регистру). Пользователь должен учитывать особенность выравнивания, чтобы не получить некорректную информацию, причем должен учитывать еще на стадии задания переменных.

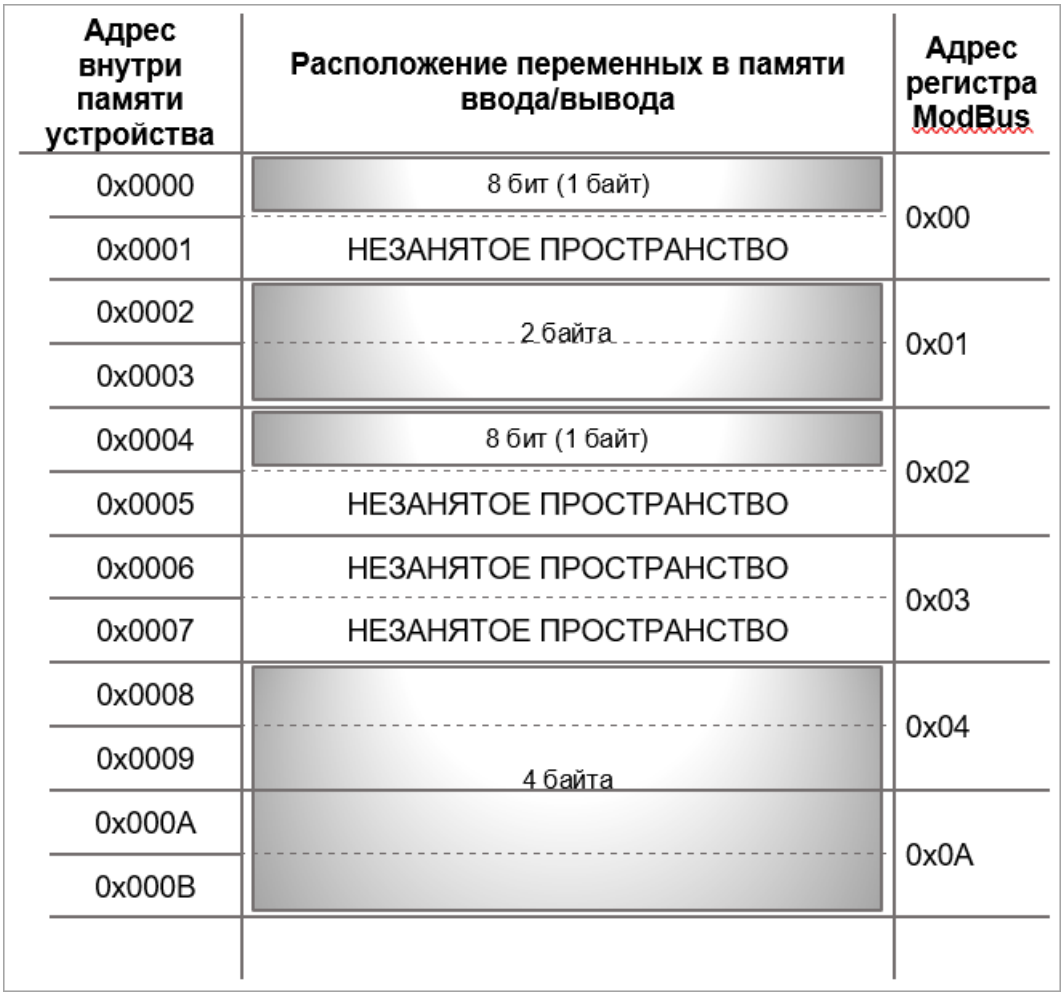


Рисунок 5.13 – Пример расположения переменных в карте регистров с учетом правила выравнивания

Добавление тегов

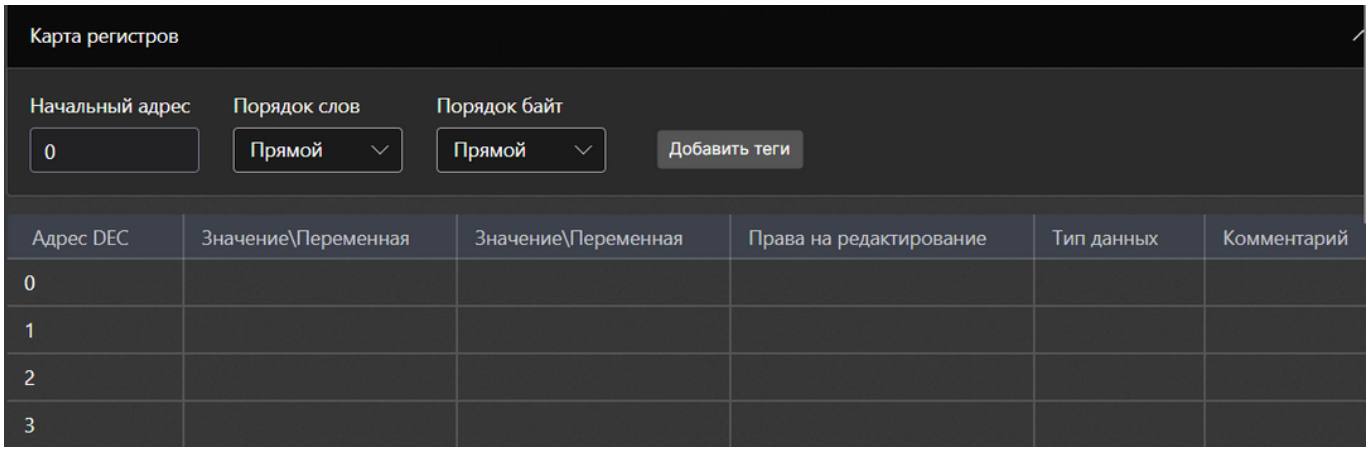


Рисунок 5.14

Заполните общие для всей карты регистров данные:

- **Начальный адрес** — задает начальный адрес регистра для всей карты регистров. Доступен ввод в десятеричном и шестнадцатеричном формате (с префиксом 16# и 0x). В случае ввода начального адреса после заполнения карты регистров, адреса автоматически пересчитаются и будут отображены в таблице согласно введенным данным. Если при вводе нового начального адреса при пересчете будут обнаружены недопустимые адреса регистров, появится сообщение об ошибке.
- **Порядок слов/порядок байт** — настройка порядка слов и порядка байт для передачи master-устройству.

Карта регистров содержит столбцы:

- **Адрес (Dec, Hex)** — адрес регистра. Переключение формата ввода происходит при нажатии на заголовок столбца.
- **Переменная, чтение** — привязка входной переменной для чтения тега.
- **Значение/Переменная, запись** — привязка выходной переменной для записи тега, либо ввод значения.
- **Права на редактирование** - уровень прав доступа, который предоставляется master-устройству для работы с тегом. В ALTA IDE версии 1.1.0 нередактируемый параметр "чтение и запись".
- **Тип данных** — тип данных тега.
- **Комментарий** — пользовательский комментарий.

Для добавления тегов нажмите на кнопку **Добавить теги**. Откроется окно создания тегов:

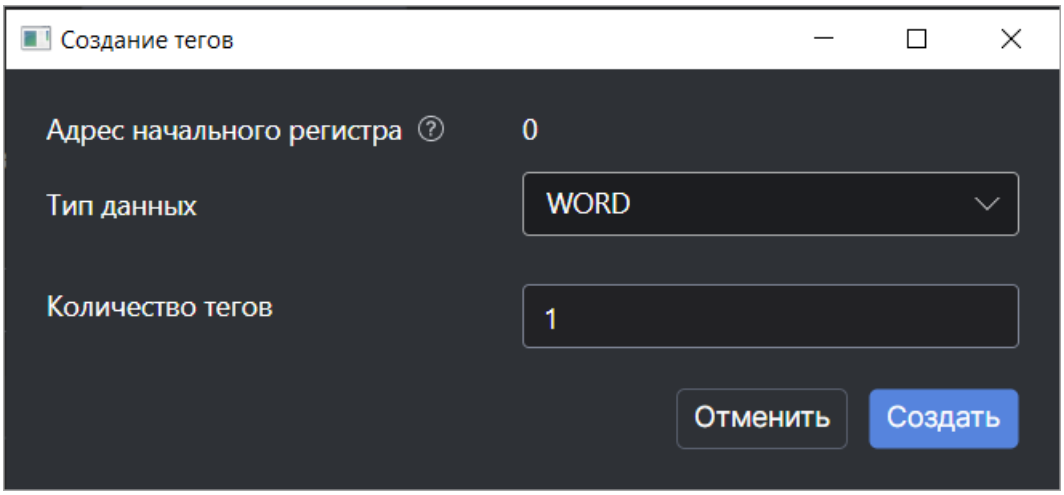


Рисунок 5.15

- **Адрес начального регистра** — не доступен для ввода. Автоматически рассчитывается с учетом ранее добавленных тегов. В случае создания первого тега - соответствует адресу, введенному в поле Начальный адрес (по умолчанию 0).
- **Тип данных** — тип данных тега.
- **Количество тегов** — количество тегов, которое требуется создать.

Нажмите кнопку **Создать**. В таблице отобразится информация, согласно введенным данным.

Созданные тэги располагаются в следующих доступных для размещения регистрах с учетом правил выравнивания.

Добавление тега на адрес, выбранный пользователем, доступно с помощью контекстного меню строки в таблице регистров.

Битовая маска

Для переменных типа BYTE, WORD, DWORD и LWORD доступен просмотр и редактирование битовой маски. Для того, чтобы развернуть битовую маску нажмите ПКМ на строку с тегом в карте регистров и выберите в контекстном меню **Раскрыть битовую маску**. Битовая маска тега отобразится в карте регистров:

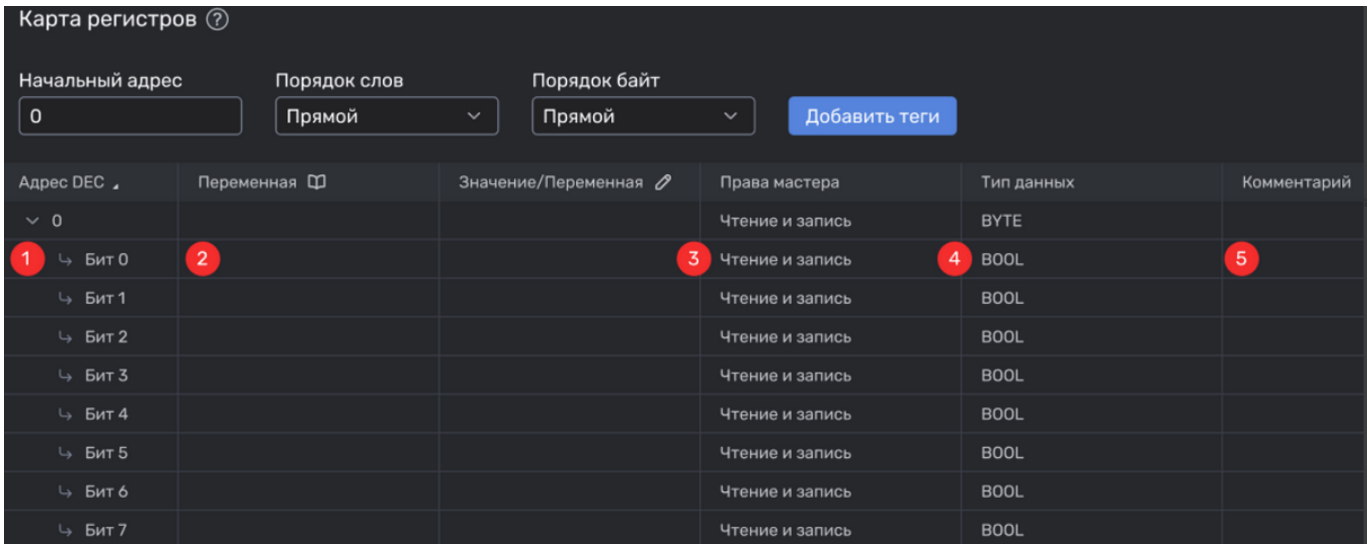


Рисунок 5.16

Если битовая маска раскрыта, то все введенные данные сохраняются для последующего редактирования и сохранения.

Свернуть или развернуть битовую маску можно с помощью значка  , расположенного перед адресом тега.

Чтобы скрыть битовую маску нажмите ПКМ на строку с тегом в таблице регистров и выберите в контекстном меню **Скрыть битовую маску**. Если любое из полей битовой маски было отредактировано, появится окно с предупреждением о том, что введенные данные не сохранятся:

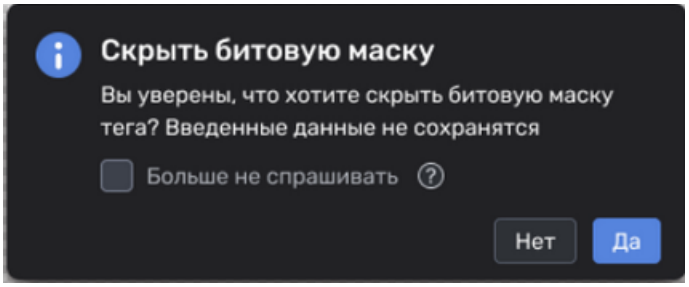


Рисунок 5.17

Если выбрать **Да,** то битовая маска скроется, все внесенные данные будут удалены. В случае выбора **Нет** битовая маска останется раскрыта.

После сохранения проекта в ALTA IDE автоматически формируется файл с картой регистров в формате *.csv*, который располагается в папке проекта *C:User/..../Project_/Communications/*

Address	VariableRe	VariableW	Access	Type	Comment
0	var_1	1	RW	BYTE	
1	var_2	0	RW	BYTE	
2	sens_1	10	RW	INT	
3	sens_2	12	RW	INT	

Рисунок 5.18

5.2.3 Добавление интерфейса подключения UART/TCP

Для получения запросов от master-устройства необходимо добавить интерфейс подключения. Устройство в режиме Slave поддерживает выбор только одного интерфейса подключения.

Для добавления интерфейса воспользуйтесь контекстным меню компонента ModbusSlave в дереве проекта:

- **UART** — добавление аппаратного интерфейса UART (COM)
- **TCP** — добавление аппаратного интерфейса TCP/IP (Ethernet)

После добавления интерфейс отобразится в дереве проекта, как ответвление компонента ModbusSlave системной папки Коммуникации:

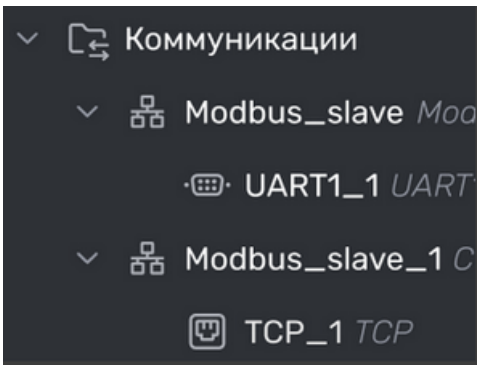


Рисунок 5.19

UART

При добавлении интерфейса UART откроется вкладка, содержащая настройки, доступные для просмотра и редактирования:

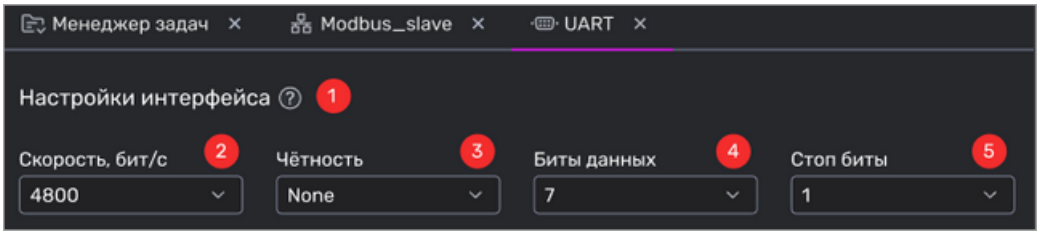


Рисунок 5.20 – Вкладка интерфейса UART

Заполните данные:

- **Скорость, бит/с** — выберите из выпадающего списка скорость, с которой будет осуществляться передача данных. Возможные значения: 300/600/1200/2400/4800/9600/19200/38400/57600/115200.
- **Чётность** — параметр, отображающий использование бита четности в посылке Modbus. Выберите из выпадающего списка: NONE – отсутствует, EVEN – проверка на четность, ODD – проверка на нечетность.
- **Биты данных** — выберите из выпадающего списка число бит данных. Возможные значения: 7 или 8
- **Стоп биты** — параметр отображающий количество стоповых бит в посылке Modbus. Возможные значения: 1, 1.5 или 2.

Каждый из интерфейсов UART может быть добавлен в проект только один раз. В случае добавления интерфейса, уже используемого в проекте, в дереве проекта у повторяющихся интерфейсов отобразится ошибка:

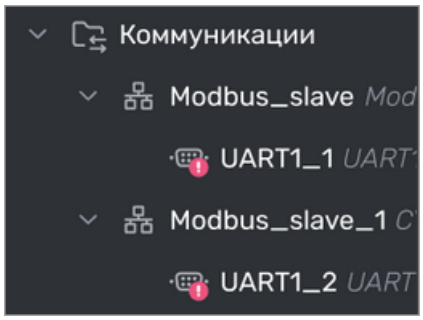


Рисунок 5.21 – Повторное добавление интерфейса

В этом случае один из интерфейсов необходимо удалить, либо заменить на еще неиспользуемый в проекте.

TCP

При добавлении интерфейса TCP откроется вкладка, содержащая настройки, доступные для просмотра и редактирования:

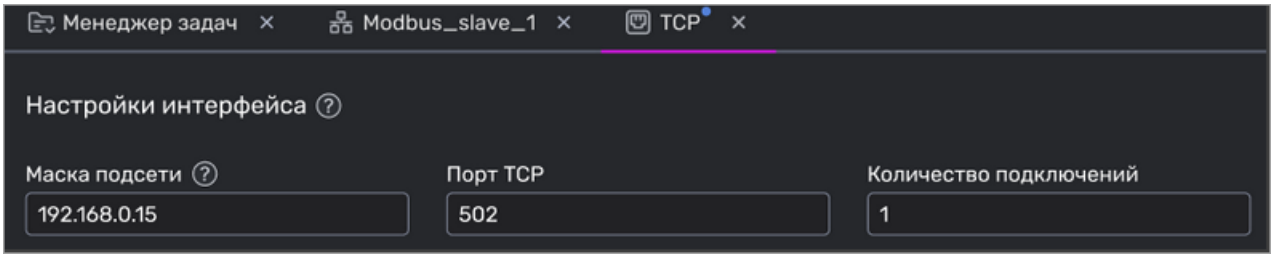


Рисунок 5.22 – Вкладка интерфейса TCP

Заполните данные:

- **Маска подсети** — настройка подсети, в которой идет обмен.
- **Порт TCP** — порт, используемый для обмена.
- **Количество подключений** — позволяет настроить ограничение количества подключений. Если ограничение включено, то интерфейс обрабатывает запросы не более чем от N источников.

Контекстное меню интерфейса подключения

Заменить на UART1 (4,5) — замена выбранного интерфейса на UART1 (4,5) (отсутствует строка с первоначально выбранным интерфейсом). При замене интерфейса ранее введенные настройки сохраняются.

Открыть — открыть вкладку настройки интерфейса.

Открыть справа — открыть справа вкладку настройки интерфейса.

Копировать — копировать интерфейс со всеми настройками.

Вырезать — вырезать интерфейс со всеми настройками.

Вставить — вставить интерфейс со всеми настройками.

Переименовать — переименовать интерфейс.

Удалить — удалить интерфейс.

6 Сочетания клавиш

Сочетание клавиш	Действие
Дерево проекта	
F2	Переименовать
Delete	Удалить выбранный элемент
Enter	Открыть
Ctrl + Enter	Открыть справа
Редактор ST/Таблица локальных переменных	
Ctrl + C	Копировать выделенный элемент(ы) в буфер обмена
Ctrl + V	Вставить из буфера обмена
Ctrl + X	Вырезать выделенный элемент(ы) в буфер обмена
Ctrl + Z	Отменить последнее изменение
Ctrl + Y	Вернуть (восстановить) отмененное действие
Ctrl + A	Выделить все строки
Ctrl + S	Сохранить изменения
Режим онлайн и отладки	
F5	Запустить/продолжить выполнение приложения на устройстве
F7	Фиксировать все значения переменных
F8	Шаг внутрь
F10	Шаг поверх
Ctrl + F7	Записать все значения переменных
Alt + F7	Освободить все значения переменных
Shift + F10	Шаг наружу
Работа с проектом	
Ctrl + N	Создать новый проект
Ctrl + O	Открыть существующий проект
Меню/Помощь	
F1	Вызов справки

7 Язык программирования ST

ST (Structured Text) - это текстовый язык высокого уровня. Является одним из пяти языков, поддерживаемых стандартом МЭК 61131-3.

7.1 Ключевые слова

Ключевые слова — это уникальные комбинации символов, используемых как отдельные синтаксические элементы. Ключевые слова не содержат внутренних пробелов и могут быть введены в символах верхнего и нижнего регистра, регистр символов не учитывается (Например, ключевые слова FUNCTION и function синтаксически эквивалентны).

Ключевые слова не должны использоваться в любых других целях, например, как имена переменных.

Зарезервированные ключевые слова

PROGRAM	VAR	IF
END_PROGRAM	VAR_INPUT	ELSIF
FUNCTION	VAR_GLOBAL	ELSE
END_FUNCTION	VAR_OUTPUT	END_IF
TYPE	END_VAR	FOR
END_TYPE	POINTER TO	END_FOR
STRUCT	REF_TO	WHILE
END_STRUCT	ARRAY	END_WHILE
CONTINUE	OF	EXIT
RETURN		

Встроенные функции/типы/определения

BOOL	SINT	EQ
BYTE	INT	GT
WORD	DINT	SHR
DWORD	LINT	REF
LWORD	REAL	ADR
USINT	LREAL	^
UINT	STRING	SIZEOF
UDINT	WSTRING	TRUE
ULINT	ADD	FALSE

7.2 Правила именования

Имена переменных, функций и функциональных блоков подчиняются следующим правилам:

- Имя не должно содержать пробелов и спецсимволов (например, !, @ и т.д.). Исключение – символ нижнего подчеркивания (_);
- Нижнее подчёркивание _ допускается и считается отдельным символом. (A_BCD и AB_CD - это разные идентификаторы);
- Имя должно начинаться с буквы;
- Имя переменной может содержать только буквы латинского алфавита;
- Имя не должно содержать несколько символов нижнего подчеркивания (_), размещенных подряд (т.е. имя i__Test недопустимо, а имя i_Te_st - допустимо);
- Регистр имен объектов не учитывается (VAR1 и var1 относятся к одной и той же переменной);
- На длину имени не накладывается никаких ограничений;
- Имя не должно совпадать с одним из зарезервированных ключевых слов (например, VAR, INT, и т.д.);

Рекомендации

Рекомендуется использовать венгерскую нотацию для именования переменных. Краткие значащие имена являются предпочтительными. Каждое слово в идентификаторе следует записывать с заглавной буквы. Перед именем переменной указывается строчный префикс, обозначающий её тип.

Тип данных	Префикс
BOOL	x
DWORD	dw

Пример:

```
dwFileSize : DWORD;
xDone  : BOOL;
```

7.3 Выражения

Выражение – это конструкция, возвращающая определенное значение после его вычисления.

Выражение состоит из операторов и операндов. Операндом может быть константа, переменная, функциональный блок или другое выражение.

Вычисление выражений выполняется согласно правилам приоритета. Оператор с самым высоким приоритетом выполняется первым, оператор с более низким приоритетом – вторым и т. д., пока не будут выполнены все операторы.

Операторы с одинаковым приоритетом выполняются слева направо.

В языке поддерживаются следующие виды выражений:

- Литералы;
- Литералы массивов;
- Литералы структур;
- Чтение переменных;
- Индексный доступ;
- Чтение полей структур;
- Унарные выражения;
- Бинарные выражения;
- Вызовы функций;
- Составные выражения;

Примеры:

```
[1,2,3]
(a1 := 1, a2 := 2);
a
a.b
a := 1
+a
a+b
fct(a,b)
```

7.3.1 Литералы

```
<numeric_literal>
```

Порядок вычисления литералов не специфицирован.

Числовые литералы могут записываться в десятичном (0, 1, 100), двоичном (2#0011_0011), восьмеричном (8#127), шестнадцатеричном (16#ABCD) видах.

Числовой литерал может быть типизированным (DWORD#123). Числовой литерал может быть записан в научной нотации с указанием множителя и показателя степени числа 10 (2.5E +1 = 25). В случае положительного показателя степени знак "+" можно опустить.

Литерал вычисляется в соответствующее значение наименьшего подходящего по размеру типа (если тип литерала не указан явно).

7.3.2 Литералы массивов

```
[<элемент_1.1>,<...>,<элемент_1.n>]
[<элемент_1.1>,<...>,<элемент_1.n>,<элемент_2.1>,<...>,<элемент_2.n>,
<элемент_m.1>,<...>,<элемент_m.n>]
[[<элемент_1.1>,<...>,<элемент_1.n>,<элемент_2.1>,<...>,<элемент_2.n>,
<элемент_m.1>,<...>,<элемент_m.n>]]
```

Литералы массивов записываются в квадратных скобках. При этом элементы массива указываются по порядку объявления через запятую.

Пример:

```
VAR
    a : ARRAY [1...4] OF INT;
    b : ARRAY [1...2,1...2] OF INT;
    c : ARRAY [1...2] OF ARRAY [1...2] OF INT;
END-VAR

    a := [1,2,3,4];
    b := [1,2,3,4];
    c := [[1,2],[3,4]];
```

В литералах массивов допускается указывать несколько первых элементов подряд. Неуказанные элементы принимаются равными нулевому значению соответствующего типа. Пропуски элементов ([1,,3]) не допускаются (ошибка компиляции E007).

Пример:

```
VAR
    a : ARRAY [1...3] OF INT;
END-VAR

    a := [1,2,3]; // a = [1,2,3]
    a := [4,5];   // a = [4,5,0]
```

7.3.3 Литералы структур

```
([<идентификатор_поля1>:=<значение>,<...>,<идентификатор_поляN>:=<значение>])
```

Литералы структур записываются в круглых скобках с явным указанием идентификаторов полей и их значений в виде <идентификатор_поля>: =<значение> через запятую. Если имя поля не указано явно, возникнет ошибка компиляции [E071].

Пропущенные значения принимаются равными нулевому значению соответствующего типа.

Пример:

```
TYPE st :
    STRUCT
        a1,a2,a3 : INT;
    END_STRUCT
END_TYPE

VAR
    s : st;
END-VAR

    s := (a1 := 1, a2 := 2, a3 := 3);  // s = (1,2,3)
    s := (a2 := 5);                    // s = (0,5,0)
```

Литералы вложенных структур требуют указания типов для каждого уровня вложенности, кроме самого нижнего. В противном случае возникнет ошибка компиляции [E071].

Пример вложенных структур:

```
TYPE st1 :
    STRUCT
        a1,a2,a3 : INT;
    END_STRUCT
END_TYPE

TYPE st2 :
    STRUCT
        s1,s2 : st1;
    END_STRUCT
END_TYPE

VAR
    s : st2;
END_VAR

s := (st2.s1 := (a1 := 1, a2 := 2, a3 := 3),
      st2.s2 := (a1 := 4, a2 := 5, a3 := 6));
```

Литерал массива структур в текущей реализации работает некорректно.

Пример структуры с массивами:

```
TYPE sta1 :
    STRUCT
        a1,a2: ARRAY [1...2] OF INT;
    END_STRUCT
END_TYPE

VAR
    s : sta1;
END_VAR

    s := (a1 := [1,2], a2 := [3,4]);
```

7.3.4 Чтение переменных

```
<идентефикатор>
```

Для чтения из переменной достаточно указать её идентификатор (a, dwVar, DWVAR).

Чтение из переменной вычисляется в текущее значение переменной и имеет тип, совпадающий с типом этой переменной.

7.3.5 Индексный доступ

```
<выражение>[<индекс1>]
<выражение>[<индекс1>,<индекс2>,<...>,<индексN>]
<выражение>[<индекс1>][<индекс2>]<...>[<индексN>]
```

Доступ к элементу массива по индексу вычисляется соответствующее значение этого элемента.

<Выражение> должно вычисляться в массив. В противном случае возникнет ошибка компиляции [E059]. Порядок вычисления <выражения> и <индекса> не специфицирован.

Попытка доступа к элементу за границами массива (индекс вычисляется на этапе компиляции) приведёт к ошибке компиляции [E058].

Попытка доступа к элементу за границами массива (индекс вычисляется на этапе исполнения) не приводит к ошибке исполнения. Поведение в этом случае не специфицировано.

Индекс может быть представлен целочисленным (в том числе булевым) литералом, целочисленной (в том числе булевой) переменной.

7.3.6 Чтение полей структур

```
<выражение>.идентификатор поля>
```

Чтение поля структуры аналогично чтению переменной. Для доступа к полям структуры после идентификатора структуры используется оператор "точка".

Чтение из поля структуры вычисляется в текущее значение поля указанного экземпляра структуры и имеет тип, совпадающий с типом этого поля.

Вложенный доступ поддерживается. Уровень вложенности неограничен.

Если функция возвращает структуру, недопустимо при вызове функции обращаться к отдельным полям возвращаемого значения. В противном случае произойдёт ошибка компиляции [E002].

ПРИМЕЧАНИЕ
Имеется в виду конструкция вида b := f().a;

7.3.7 Унарные выражения

```
<унарный_оператор> <выражение>
```

Поддерживаются следующие унарные операторы: +.

Все математические операторы возвращают целочисленный тип. Данное выражение сначала вычисляет свое внутреннее выражение, а потом к его результату применяет соответствующее действие.

7.3.8 Бинарные выражения

```
<выражение> <бинарный_оператор> <выражение>
```

Поддерживаются следующие бинарные операторы: +.

Все математические операторы возвращают целочисленный тип. Данное выражение сначала вычисляет свои внутренние выражения (слева направо), а потом применяет к ним соответствующее действие.

Переполнение ведёт себя по принципу арифметики с дополнением до 2.

Пример переполнения:

```
// A : DWORD;

A := 4294967295;
A := A + 1; // Результат: A = 0
```

7.3.9 Вызовы функций

```
<идентификатор_функции>(<аргументы>)
<идентификатор_функции>(<позиционный_аргумент1>,<позиционный_аргумент2>)
<идентификатор_функции>(<именованный_аргумент1> := <выражение>,
<именованный_аргумент2> := <выражение>)
```

Аргументы функции могут передаваться как позиционно, так и именованно, через запятую.

Аргумент может быть структурой, передаётся в функцию аналогично аргументам простого типа.

Аргумент может быть массивом. Такой аргумент не может передаваться в виде литерала (ошибка компиляции [E071]).

ПРЕДУПРЕЖДЕНИЕ
Запрещено совмещать варианты передачи аргументов, в этом случае возникнет ошибка компиляции [E031].

Пример для функции fct(in1,in2,in3):

Вызов fct(in2 := 2, in1 := 1, 3) в CODESYS приведёт к ошибке компиляции.

МЭК 61131-3 описывает только формальный (именованный) и неформальный (позиционный) варианты передачи, но допускает передачу неполного перечня аргументов. Каждая переменная, которой в перечне не

задано значение, имеет начальное значение, присвоенное в объявлении вызванного объекта, или неявное значение соответствующего типа данных.

При вызове функции аргументы вычисляются в порядке указания слева направо, затем происходит их передача по значению в функцию.

Пример:

```
fct(1,2)              //позиционная передача аргументов
fct(in2 := 2,in1 := 1) //именованная передача аргументов
```

Возвращаемый слот функции имеет значение по умолчанию - нулевое значение соответствующего типа.

Функция с результатом может вызываться в выражении или как оператор. Такое выражение вычисляется в возвращаемое значение соответствующего типа.

Функция может возвращать структуру. Начальным значением для возвращаемого слота-структуры будет структура с полями, имеющими соответствующие их типам нулевые значения.

Функция без результата не должна вызываться внутри выражения. В противном случае произойдёт ошибка компиляции [E037].

Пример:

```
dwIn1 := 5;
dwIn2 := 7;
dwRes1 := ADDER(dwIn1, dwIn2); (*результат 12*)
dwRes2 := 3 + ADDER(dwIn1, 7); (*результат 15*)
```

Функция может возвращать структуру (в том числе вложенную). Начальным значением для возвращаемого слота-структуры будет структура с полями, имеющими соответствующие их типам нулевые значения.

Допускается как полная перезапись возвращаемого слота, так и перезапись отдельных полей.

Запрещено обращаться к отдельным полям возвращаемого значения извне функции. В противном случае произойдёт ошибка компиляции [E007].

Пример возврата структуры:

```
TYPE st :
    STRUCT
        dw1,dw2 : DWORD;
    END_STRUCT
END_TYPE

FUNCTION pack : st
    VAR_INPUT
        in1,in2 : DWORD;
    END_VAR
    VAR
        stBuf : st;
    END_VAR

    stBuf.dw1 := in1;
    stBuf.dw2 := in2;
    pack := stBuf;    //полная перезапись возвращаемого слота

    pack.dw1 := in1;  //установка отдельных полей возвращаемого слота
    pack.dw2 := in2;

END_FUNCTION
```

Функция может возвращать массив (в том числе многомерный). Начальным значением для возвращаемого слота-массива будет массив с элементами, имеющими соответствующие типу нулевые значения.

Допускается как полная перезапись возвращаемого слота, так и перезапись отдельных элементов.

ПРЕДУПРЕЖДЕНИЕ
Запрещено обращаться к отдельным элементам возвращаемого значения извне функции. В противном случае произойдёт ошибка компиляции [E007].

Возвращаемое значение-массив может быть проигнорировано.

Пример возврата массива:

```
FUNCTION f : ARRAY [0...1] OF INT
    VAR_INPUT
        a : INT;
    END_VAR
    VAR
        tmp : ARRAY[0...1] OF INT;
    END_VAR

        f[0] := a; //перезапись отдельных элементов
        f[1] := a;

    tmp[0] := a;
    tmp[1] := a;
    f := tmp;     //полная перезапись возвращаемого слота

END_FUNCTION
```

Всё описанное выше справедливо и для массивов структур.

7.3.10 Составные выражения

Составные выражения содержат в качестве элементов другие выражения (необязательно простые). Количество подвыражений и количество уровней вложенности не ограничены.

Вложенные подвыражения вычисляются перед содержащими их выражениями.

Пример:

```
b + f(c)
f(b) + f(-g(x,y),c+d)
a = (c > d)
```

7.3.11 Приоритеты операций

Операции в таблице расположены от высшего приоритета к низшему. Чем выше приоритет операции, тем раньше она выполняется. Операции одного приоритета выполняются слева направо.

Операция	Обозначение
Взятие в скобки	(<Выражение>)
Оператор с передачей операндов в скобках	<Оператор>(<операнды>)
Сложение	+
Сравнение	<, >, <=, >=
Проверка на равенство	=

7.3.12 Операторы

7.3.12.1 Оператор ADD

МЭК-оператор для сложения переменных.

Разрешённые типы данных: WORD, DWORD, DINT.

Пример:

```
a := 1+3;         (*Результат 4*)
c := ADD(1,2);  (*Результат 3*)
```

7.3.12.2 Оператор EQ

МЭК-оператор для проверки операндов на равенство.

Разрешённые типы данных: все элементарные типы.

Возвращает TRUE, если операнды равны, и FALSE в противном случае.

Пример:

```
x1 := 1 = 1;        //Результат: TRUE
x2 := TRUE = FALSE; //Результат: FALSE
x3 := EQ(a,b);
```

7.3.12.3 Оператор GT

МЭК-оператор для проверки утверждения "больше" относительно операндов.

Разрешённые типы данных: WORD, DWORD, DINT.

Возвращает TRUE, если первый операнд больше второго, и FALSE в противном случае.

Пример:

```
xVar1 := 20 > 2;c //Результат: TRUE
xVar2 := 20 > 20; //Результат: FALSE
xVar3 := 20 > 30; //Результат: FALSE
```

7.3.12.4 Оператор REF (ADR)

```
<имя_указателя> := REF(<имя_объекта>);
<имя_указателя> := ADR(<имя_объекта>);
```

МЭК-оператор для возврата указателя на объект.

Взятие указателя реализуется оператором REF() или ADR(). Оператор ADR добавлен для совместимости с CODESYS V3.5. ADR() и REF() семантически эквивалентны.

Пример:

```
VAR
    p1, p2 : REF_TO INT;
    i : INT;
END-VAR

p1 := REF(i);
p2 := ADR(i);
```

7.3.12.5 Оператор разыменования ^

```
<имя_объекта> := <имя_указателя>^;
```

Указатели типизированные. Разыменование вычисляется в значение соответствующего типа по указателю.

При несовпадении типов, где это возможно, выполняется неявное преобразование типа (между целочисленными типами и BOOL). Если тип данных слева меньше, неявное преобразование выполняется с предупреждением компилятора [E067].

Несовпадение типов с невозможностью неявного преобразования приведёт к ошибке компиляции [E037].

Разыменование нулевого указателя приводит к ошибке исполнения (Segmentation fault).

Пример:

```
VAR
    p : REF_TO INT;
    i : INT;
    di : DINT;
END-VAR

p := REF(i);
di := p^;
```

7.3.12.6 Оператор SIZEOF

```
SIZEOF(<выражение>)
```

Возвращает размер объекта в байтах (значение типа ULINT).

Пример:

```
VAR
    i :  INT;
    sz : ULINT;
END-VAR

sz := SIZEOF(i); // sz = 2
```

7.4 Утверждения (инструкции)

Утверждение	Синтаксис
Присвоение переменной	<идентификатор> := <выражение>

Условие	IF <условие1> THEN
	<блок кода1>
	ELSIF <условие2> THEN
	…
	ELSE
	<блок кодаN>
	END_IF

Цикл с предусловием	WHILE <условие> DO … END_WHILE
Цикл FOR	FOR <выражение-счётчик> := <начальное значение> TO <конечное значение> DO … END_FOR
EXIT	WHILE <условие> DO … EXIT; … END_WHILE
CONTINUE	FOR <счётчик> := <начальное значение> TO <конечное значение> BY <шаг> DO <блок кода 1> CONTINUE; <блок кода 2> END_FOR

Утверждение	Синтаксис
RETURN	PROGRAM <…> … RETURN; … END_PROGRAM
Комментарии	// Однострочный комментарий (* многострочный комментарий *)

7.4.1 Присвоение

```
<выражение места>:=<выражение>
```

<Выражение места> (L-Value expression) может быть представлено: именем переменной (в том числе структурой), именем поля переменной-структуры, именем функции (внутри этой функции), указателем.

<Выражение> может быть любым. Порядок вычисления <выражения> и <выражения-места> не определён.

Доступ к элементу массива осуществляется по индексу. Доступ к полям структуры осуществляется через точку. Вложенный доступ поддерживается.

Разрешены присвоения выражений типа BOOL и числовых типов (см. примечание ниже), присвоения структур одного и того же типа. Присвоение переменной простого типа значения структуры (и наоборот), а также структур несовпадающих типов приведёт к ошибке компиляции [E037].

ПРИМЕЧАНИЕ
Если тип идентификатора слева **больше**, чем тип, возвращаемый выражением справа, произойдёт неявное преобразование типа от меньшего к большему. В противном случае произойдёт неявное преобразование от большего типа к меньшему, компилятор выдаст предупреждение [E607].

Пример:

```
a, b, c : DWORD;
ai : ARRAY [0..9] OF INT;

a := 1;
c := a + b + 5;

ai[0] := 1;
ai[a] := 2;
```

Пример доступа к полям структур:

```
TYPE str1 :
STRUCT
a : DINT;
END_STRUCT
END_TYPE

TYPE str2 :
STRUCT
a : DINT;
s : str1;
END_STRUCT
END_TYPE

VAR
s1 : str1;
s2 : str2;
END_VAR

s1.a := 1;
s2.s := s1;
s2.s.a := 2;
```

При присвоении массивов (в том числе многомерных) элементы массива указываются по порядку объявления.

Пример:

```
VAR
a : ARRAY [1..4] OF INT;
b : ARRAY [1..2,1..2] OF INT;
c : ARRAY [1..2] OF ARRAY [1..2] OF INT;
END_VAR

a := [1,2,3,4];
b := [1,2,3,4];
c := [[1,2],[3,4]];
```

В литералах массивов допускается указывать несколько первых элементов подряд. При этом оставшиеся элементы сохранят свои прежние (начальные) значения. Пропуски элементов (например, [1,,3]) не допускаются (ошибка компиляции E007).

ПРИМЕЧАНИЕ
В CODESYS V3.5 присвоение является выражением. То есть допустима конструкция вида a := b := c

7.4.2 IF

Инструкция ветвления в зависимости от условия(-ий).

```
IF <условие> THEN
    <блок кода 1>
ELSE
    <блок кода 2>
END_IF
```

Здесь блоки кода 1, 2 - представляют собой произвольное количество инструкций (от нуля до бесконечности). Блоки кода внутри могут содержать вложенные конструкции IF/FOR/WHILE.

Условия являются выражениями и должны возвращать результат типа BOOL. В противном случае произойдёт неявное преобразование в BOOL, компилятор выдаст предупреждение [E096].

При выполнении сначала вычисляется условие. Если оно вычисляется в значение TRUE, то исполняется блок кода 1, иначе - блок кода 2. После этого выполнение завершается. Блок ELSE может отсутствовать.

Пример:

```
IF dwVar > 10 THEN
x1 := TRUE;
ELSE
x1 := FALSE;
END_IF
```

Также, в данной конструкции может быть использован вариант с ELSIF.

Количество блоков ELSIF не ограничено.

```
IF <условие1> THEN
    ...
ELSIF <условие2> THEN
    ...
ELSIF <условие3> THEN
    ...
ELSE
   ...
END_IF
```

Расширенная конструкция вычисляется следующим образом:

1. Вычисляется условие 1.
2. Если условие1 истинно, выполняется блок кода 1. Далее выполнение конструкции завершается. Если условие 1 ложно, вычисляется условие 2.

3. Если условие 2 истинно, выполняется блок кода 2. Если ложно - вычисляется условие 3.

И так далее по всем имеющимся блокам ELSIF. Если условия IF/ELSIF ложны, выполняется блок кода внутри ELSE. Далее выполнение конструкции завершается.

Расширенная конструкция может быть преобразована к обычной конструкции IF:

```
IF <условие1> THEN
    <блок кода 1>
ELSE
    IF <условие2> THEN
        <блок кода 2>
    ELSIF <условие3> THEN
        ...
    ELSE
        ...
    END_IF
END_IF
```

Здесь первый блок ELSIF преобразован в блок ELSE с вложенным IF с конструкциями ELSIF/ELSE. Операционная семантика расширенной конструкции исходит из операционной семантики базового IF THEN ELSE END_IF.

Пример

```
IF dwVar > 10 THEN
x1 := TRUE;
ELSIF dwVar = 10 THEN
x2 := TRUE;
ELSE
x3 := TRUE;
END_IF
```

7.4.3 WHILE

Инструкция для цикла с предусловием.

```
WHILE <условие> DO
      <блок кода>
END_WHILE
```

Условие должно возвращать результат типа BOOL. В противном случае произойдёт неявное преобразование в BOOL, компилятор выдаст предупреждение [E096].

Блок кода собой произвольное количество инструкций (от нуля до бесконечности). Блок кода внутри может содержать вложенные конструкции IF/FOR/WHILE.

Блок кода выполняется повторно, пока условие истинно, и не выполняется в противном случае. То есть может выполниться от нуля до бесконечности раз в зависимости от условия.

Пример:

```
WHILE xCounting DO
dwCounter := dwCounter + 1;

IF dwCounter = 10 THEN
 xCounting := FALSE;
END_IF
END_WHILE
```

7.4.4 FOR

Оператор FOR указывает, что последовательность операторов выполняется повторно, в то время как переменная управления (счётчик) инкрементируется.

```
FOR <счётчик> := <начальное значение> TO <конечное значение> BY <шаг> DO
    <блок кода>
END_FOR
```

Счётчик, начальное значение и конечное значение должны быть выражениями целого типа (например, DINT). Они не должны изменяться внутри цикла. В противном случае не будет обеспечено заданное количество итераций, но выполнение завершится корректно. Переменная-счётчик должна быть объявлена. При несоответствии типов произойдёт ошибка компиляции [E094].

Блок кода собой произвольное количество инструкций (от нуля до бесконечности). Блок кода внутри может содержать вложенные конструкции IF/FOR/WHILE. По окончании выполнения блока кода счётчик изменяется на заданный шаг.

Блок кода выполняется столько раз, сколько задано начальным и конечным значениями для счётчика.

Конструкция BY может быть опущена. В этом случае изменение счётчика происходит на 1 от начального значения к конечному.

Пример:

```
FOR i := 1 TO 10 DO
dwCounter := dwCounter + 1;
END_FOR
```

7.4.5 EXIT

Утверждение EXIT используется внутри конструкций FOR и WHILE для безусловного выхода из цикла. Блок кода, расположенный после EXIT, выполняться не будет.

Счётчик FOR при выходе из цикла сохраняет своё значение.

При использовании EXIT во вложенном цикле выход происходит только из него. Внешний цикл продолжит выполняться.

```
WHILE <условие> DO
    <блок кода 1>
    EXIT;
    <блок кода 2>
END_WHILE
```

Пример:

```
WHILE xCounting DO
dwCounter := dwCounter + 1;

IF dwCounter = 10 THEN
EXIT;
END_IF

END_WHILE
```

7.4.6 CONTINUE

Утверждение CONTINUE используется внутри конструкций FOR и WHILE для безусловного перехода на следующую итерацию цикла. Блок кода, расположенный после CONTINUE, выполняться не будет.

Счётчик FOR при выполнении CONTINUE сохраняет своё значение в текущей итерации и изменяется на заданный шаг в следующей итерации. Если CONTINUE вызван на последней итерации, счётчик по окончании цикла примет <конечное значение> + <шаг> (как и без использования CONTINUE).

```
FOR <счётчик> := <начальное значение> TO <конечное значение> BY <шаг> DO
    <блок кода 1>
    CONTINUE;
    <блок кода 2>
END_FOR
```

Пример:

```
FOR i := 1 TO 10 DO
IF xCommand AND dwCounter = 5 THEN
CONTINUE;
END_IF
dwCounter := dwCounter + 1;
END_FOR
```

7.4.7 RETURN

Утверждение RETURN используется внутри программ и функций, выполняет безусловный выход из POU и передачу потока управления вызвавшему POU.

```
PROGRAM <...>
...
RETURN;
...
END_PROGRAM
```

Пример:

```
FOR i := 1 TO 10 DO
IF i = 3 AND xCommand
RETURN;
END_IF
END_FOR
```

7.4.8 Комментарии

```
// Однострочный комментарий
 (*многострочный
комментарий*)
```

7.5 Тип данных переменных

Программа представляет собой набор операций, выполняемых над данными, размещенными в памяти контроллера. Для упрощения работы с памятью данные представляются в виде переменных – именованных объектов с определенными характеристиками (в частности – типом). Перед работой с переменными необходимо их объявить.

Сначала указывается имя переменной, затем ее тип и опционально – начальное значение. Имя переменной и тип разделяются символом двоеточия. Тип и начальное значение разделяются оператором присваивания : =. Заканчивается объявление символом «точка с запятой».

Вместе с объявлением переменной можно указать пояснительный комментарий.

ALTA IDEподдерживает однострочные комментарии, начинающиеся с символа //, и многострочные – размещенные между символами (* и *).

Тип данных переменной определяет род информации, диапазон представлений и множество допустимых операций.

В ALTA IDE используются следующие типы переменных:

- булевский (двоичный);
- целочисленный;
- с плавающей точкой;
- строковые типы;
- структура;
- массивы.

Значения от одной переменной к другой могут передаваться только при совпадающих типах переменных.

7.5.1 BOOL

Значения	Память
TRUE (1), FALSE (0)	1 байт

7.5.2 Целочисленные типы

Тип данных	Нижняя граница	Верхняя граница	Память
WORD/UINT	0	65535	16 бит
INT	-32768	32767	16 бит
DWORD/UDINT	0	4294967295	32 бит
DINT	-2147483648	2147483647	32 бит

7.5.3 Типы с плавающей точкой (IEEE 754)

Тип данных	Нижняя граница	Верхняя граница	Память
REAL	1.0E-44	3.402823E+38	32 бит
LREAL	4.94065645841247E-324	1.7976931348623157E+308	64 бит

7.5.4 Строковые типы

Строки являются нуль-терминированными

Тип данных	Кодировка	Память
STRING	UTF-8	1 байт
WSTRING	USC-2	2 байта

7.5.5 Объявление типа

Объявление типа может быть синонимом любому типу, в том числе встроенному.

```
TYPE
    (<имя>: <объявление типа>;)+
END_TYPE
```

Здесь конструкция ()+ означает одно или более повторений.

Объявление типа может быть представлено:

1. Именем другого типа.
2. Ограничением числового типа по диапазону.
3. Объявлением структуры.

ПРЕДУПРЕЖДЕНИЕ
На данный момент в компиляторе присутствуют ошибки: объявление переменной через псевдоним к пользовательскому типу данных приведёт к ошибке исполнения.

7.5.5.1 Имя другого типа

```
<имя_нового_типа> : <имя_типа>;
```

Представляет собой псевдоним другого типа (в том числе встроенного) и может использоваться взаимозаменяемо с именем этого типа.

7.5.5.2 Ограничение диапазона

Представляет собой псевдоним другого целочисленного типа (в том числе встроенного) с ограниченным диапазоном значений.

Попытка присвоить литерал, выходящий за указанный диапазон, должна приводить к ошибке компиляции. Во всех остальных случаях (вычисление на этапе исполнения) ограничение не учитывается.

```
<имя_нового_типа> : <имя_числового_типа>(<нижняя граница>..<верхняя_граница>);
```

ПРЕДУПРЕЖДЕНИЕ
В CODESYS V3.5 допускается ограничение только целочисленных типов. Попытка ограничить другой тип (в том числе REAL) приводит к ошибке компиляции. Нижняя граница не должна быть больше верхней границы. В противном случае произойдёт ошибка компиляции. Ограничение к одному может быть указано только один раз: не допускаются конструкции вида DWORD(0..10)(0..5).

Примеры:

```
TYPE
    DEGREE : DINT;                  // синоним встроенного типа
    SENSOR : MY_SENSOR_TYPE;        // синоним пользовательского типа
END_TYPE

TYPE
    DEGREE_LIMITED : DINT(0..100); // синоним с ограничением диапазона
END_TYPE
```

7.5.6 Структуры

Структура - пользовательский тип данных, содержащий набор именованных полей любых типов данных.

Количество полей от 1 до бесконечности. Попытка объявить структуру без полей приведёт к ошибке компиляции [E028].

Структуры могут быть вложенными. Уровень вложенности неограничен.

Объявление рекурсивной структуры приведёт к ошибке компиляции [E029].

Синтаксис:

```
TYPE <имя структуры> :
    STRUCT
        <блок объявления>
    EDN_STRUCT
END_TYPE
```

Пример объявления:

```
TYPE str1 :
    STRUCT
        a : DINT;
    END_STRUCT
END_TYPE

TYPE str2 :
    STRUCT
        a : DINT;
        s : str1;
    END_STRUCT
END_TYPE
```

Доступ к полям структуры осуществляется через точку. Вложенный доступ поддерживается.

Пример доступа к полям:

```
VAR
    s1 : str1;
    s2 : str2;
END_VAR

s1.a := 1;
s2.s := s1;
s2.s.a := 2;
```

7.5.7 Массивы

Массив - совокупность элементов данных одного типа.

Синтаксис:

```
<имя_переменной> : ARRAY [<размерность>] OF <тип_данных>;
<имя_переменной> : ARRAY [<размерность1>,<размерность2>,<...>,<размерностьN>] OF <тип
<имя_переменной> : ARRAY [<размерность1>] OF ARRAY [<размерность2>] OF <тип_данных>;
```

В случае многомерного массива его размерности указываются через запятую. Число размерностей не ограничено. <Тип_данных> может быть простым типом, объявлением типа, структурой, массивом (в том числе многомерным). Массив массивов **не является** синтаксически и семантически эквивалентным многомерному массиву.

Пример объявления:

```
VAR
    aiArray1 : ARRAY[0..9] OF INT;
    aiArray2 : ARRAY[0..9, 0..9] OF INT;
    aiArray3 : ARRAY[0..9] OF ARRAY [0..9] OF INT;
END_VAR
```

Доступ к элементам массива осуществляется по индексу. Доступ по индексу вычисляется в соответствующее значение элемента массива. Попытка доступа к элементу за границами массива (индекс вычисляется на этапе компиляции) приведёт к ошибке компиляции [E058]. Попытка доступа к элементу за границами массива (индекс вычисляется на этапе исполнения) не приводит к ошибке исполнения.

Синтаксис:

```
<имя_переменной>[<индекс>]
<имя_переменной>[<индекс1>,<индекс2>,<...>,<индексN>]
<имя_переменной>[<индекс1>][<индекс2>]<...>[<индексN>]
```

Примеры доступа по индексу:

```
aiArray1[1] := 255;
aiArray2[1,1] := 255;
aiArray3[1][1] := 255;
```

Доступ по индексу к массиву структур выполняется аналогично.

Пример доступа по индексу к массиву структур:

```
astArray[1].a := 255;
```

7.5.8 Указатели (REF_TO / POINTER TO)

Указатель - переменная, хранящая адрес некоторого объекта в памяти.

Указатель является 64-битным целым числом (ULINT).

ПРИМЕЧАНИЕ
В МЭК61131-3 указатели используются с ключевым словом REF_TO.

Объявление:

```
<имя_указателя> : REF_TO <тип данных>;
<имя_указателя> : POINTER TO <тип данных>;
```

Использование ключевого слова POINTER вызывает предупреждение компилятора (E015), как несоответствие стандарту МЭК61131-3.

Взятие указателя реализуется оператором REF(). Оператор ADR() возвращает числовое значение указателя. В компиляторе реализовано неявное преобразование в обе стороны. Разыменование (взятие значения по указателю) реализуется оператором ^.

Синтаксис:

```
<имя_указателя> := REF(<имя_объекта>);
<имя_указателя> := ADR(<имя_объекта>);
<имя_объекта> := <имя_указателя>^;
```

Пример:

```
VAR
    p : REF_TO INT;
    i : INT;
    di : DINT;
END_VAR

p := REF(i);
di := p^;
```

Указатель на массив

Указатель на массив содержит адрес первого элемента массива.

Указатель на структуру

Указатель на структуру содержит адрес первого поля по порядку указания в объявлении типа структуры.

Указатель на указатель

Допускается использовать указатель на указатель. Глубина "вложенности" неограничена.

Пример:

```
VAR
    i : INT;
    p : REF_TO INT;
    pp : REF_TO REF_TO INT;
END_VAR

p := REF(i);
pp := REF(p);
```

Операция разыменования выглядит следующим образом:

```
i := p^;
i := pp^^;
```

Арифметика указателей

Прибавление/вычитание целочисленного выражения N из указателя вычисляется в прибавление/вычитание N*<размер типа указателя>. Прибавление/вычитание нецелочисленного выражения (REAL, STRING, REF_TO и т.п.) из указателя приведёт к ошибке компиляции [E071].

ПРИМЕЧАНИЕ
В CODESYS V3.5 также допускается сложение и вычитание указателей между собой. Производится как сложение и вычитание LWORD. В случае вычитания результат всегда имеет тип DWORD.

Пример:

```
VAR
    p : REF_TO INT;
    pa : REF_TO ARRAY[1..3] OF INT;
END_VAR

//sizeof(int) = 2
p;   // 0x0
p+1; // 0x2
p+2; // 0x4
p-1; // 0xFE

//sizeof(array[1..3] of int) = 6
pa;      // 0x10
pa+1;    // 0x16
pa+2;    // 0x1C
pa-1;    // 0x0A
```

Неявные преобразования чисел/массивов/указателей

Ниже рассматриваются ситуации, когда выражение типа X оказывается в позиции, где ожидается выражение типа Y (или неявное преобразование X ⇒ Y).

1. Преобразование ARRAY […] OF T ⇒ REF_TO(T) выполняется.
2. Преобразование REF_TO ARRAY […] OF T ⇒ REF_TO(T) выполняется с предупреждением компилятора [E090].
3. Преобразование REF_TO T ⇒ SIZEOF(T) выполняется только если T имеет размер не меньше, чем REF_TO, т.е. LWORD. В противном случае произойдёт ошибка компиляции [E065].
4. Преобразование SIZEOF(T) ⇒ REF_TO T выполняется только если T имеет размер не меньше, чем REF_TO, т.е. LWORD. В противном случае произойдёт ошибка компиляции [E065].

7.6 POU

POU (Programming Organization Unit) - программный компонент, структурная единица проекта, описывающая алгоритм на языке ST.

7.6.1 Функция

Функция - POU, возвращающий одно значение.

ПРИМЕЧАНИЕ
Выходные переменные VAR_OUTPUT в функциях не реализованы на данный момент. Для вывода нескольких значений могут использоваться глобальные переменные VAR_GLOBAL.

В CODESYS V3.5 допускается использовать VAR_OUTPUT в функциях.

Функция не сохраняет значения переменных между вызовами. Вызов функции с одинаковыми параметрами (VAR_INPUT) приводит к одинаковому результату.

ПРИМЕЧАНИЕ
Некоторые системные функции, например, TIME() и RANDOM() могут выдавать различные значения.

Объявление функции

```
FUNCTION <имя_функции> : <возвращаемый_тип>
<объявления переменных и параметров>

<тело реализации>
END_FUNCTION
```

Здесь <имя_функции> - произвольный идентификатор. Возвращаемый тип может не указываться, в этом случае опускается двоеточие.

Объявления переменных и параметров могут содержать любое количество блоков объявлений (в том числе отсутствовать).

Тело реализации содержит любое (в том числе нулевое) количество утверждений.

Пример описания функции:

```
FUNCTION ADDER : DWORD
VAR_INPUT
    dwVar1 : DWORD;
    dwVar2 : DWORD;
END_VAR
VAR

END_VAR

ADDER := dwVar1 + dwVar2;

END_FUNCTION
```

Функция с результатом может вызываться в выражении или как оператор.

Функция без результата **не должна** вызываться внутри выражения.

Пример вызова функции:

```
dwIn1 := 5;
dwIn2 := 7;
dwRes1 := ADDER(dwIn1, dwIn2);   (*Результат 12*)
dwRes2 := 3 + ADDER(dwIn1, 7);   (*Результат 15*)
```

7.6.2 Программа

Программа - POU, возвращающий одно или несколько значений. Программа сохраняет значения локальных переменных между вызовами.

Объявление программы:

```
PROGRAM <имя_программы>
<объявления переменных и параметров>

<тело реализации>
END_PROGRAM
```

Здесь **<имя_программы>** - произвольный идентификатор.

Объявления переменных и параметров могут содержать любое количество блоков объявлений (в том числе отсутствовать).

Тело реализации содержит любое (в том числе нулевое) количество утверждений.

Программа может вызываться из других программ и функций. Рекурсивный вызов допускается. Глубина рекурсии не определена.

ПРИМЕЧАНИЕ
В CODESYS V3.5 программа не может вызываться из функции.

Вызов программы:

```
<имя_программы>(<параметр1> ,<...>,<параметрN>)

<имя_программы>(<параметр1> :=<значение>,<...>,<параметрN>:=<значение>)

<имя_программы>(<выходной_параметр1> =><переменная1>,<...>,<выходной_параметрN>=>
<переменнаяN>)
```

Передача параметров VAR_INPUT в программу может осуществляться как позиционно (при отсутствии VAR_OUTPUT), так и именованно. Запрещено совмещать варианты передачи аргументов, в этом случае возникнет ошибка компиляции [E031]. VAR_OUTPUT могут передаваться только именованно. При наличии в программе блока VAR_OUTPUT все аргументы должны передаваться именованно.

ПРИМЕЧАНИЕ
В CODESYS V3.5 допускается только именованная передача параметров в/из программы.

Извне доступны только переменные VAR_INPUT и VAR_OUTPUT программы. Локальные переменные VAR извне недоступны (ошибка компиляции [E049]).

ПРИМЕЧАНИЕ
В CODESYS V3.5 доступ извне к локальным переменным POU разрешён.

Пример доступа к входным/выходным параметрам программы:

```
var1 := prg.in;
var2 := prg.out;
```

По окончании выполнения программы выходные параметры VAR_OUTPUT по программы передаются по значению в указанные переменные. Поток выполнения передаётся вызвавшему POU. Для принудительного завершения программы используется утверждение RETURN.

Пример вызова программы:

```
prg1();
prg2(param1,param2);
prg3( in:= var1, out => var2);
```

7.6.3 Переменные

В языке имеются следующие виды переменных:

- VAR;
- VAR_INPUT;
- VAR_OUTPUT;
- VAR_GLOBAL.

7.6.3.1 VAR

Локальные переменные объявляются между ключевыми словами VAR и END_VAR в области объявления POU.

Пример:

```
VAR
    dwVar : DWORD;
END_VAR
```

7.6.3.2 VAR_INPUT

Переменные VAR_INPUT являются аргументами для функции/программы. Переменные VAR_INPUT объявляются между ключевыми словами VAR_INPUT и END_VAR в области объявления POU.

Порядок указания переменных в блоке(-ах) VAR_INPUT определяет позиции аргументов POU.

При вызове POU происходит передача по значению.

Пример:

```
VAR_INPUT
    dwIn1 : DWORD; (*Первый аргумент функции*)
    dwIn2 : DWORD; (*Второй аргумент функции*)
END_VAR
```

7.6.3.3 VAR_OUTPUT

Переменные VAR_OUTPUT являются выходными параметрами программы. Переменные VAR_OUTPUT объявляются между ключевыми словами VAR_OUTPUT и END_VAR в области объявления POU. Порядок указания переменных в блоке(-ах) VAR_OUTPUT определяет позиции параметров при вызове POU.

По окончании выполнения POU происходит передача параметров VAR_OUTPUT по значению.

Пример:

```
VAR_OUTPUT
    out1 : INT;
    out2, out3 : BOOL;
END_VAR
```

7.6.3.4 VAR_GLOBAL

Глобальные переменные объявляются между ключевыми словами VAR_GLOBAL и END_VAR вне POU в отдельном списке внутри проекта. Количество списков неограничено. Глобальные переменные видны из всех POU проекта. Пример:

```
VAR_GLOBAL
    a : INT;
END_VAR
```

7.6.4 Примеры объявления

Переменные одного типа могут быть объявлены группами через запятую. Размер группы неограничен.

Пример 1:

```
VAR
    dwVar1, dwVar2, dwVar3 : DWORD
END_VAR
```

Блоки объявления **не могут** быть вложены друг в друга.

Пример 2:

```
FUNCTION fct


VAR_INPUT
in1, in2 : DWORD;
END_VAR

VAR_INPUT
in3 : DWORD;
END_VAR

VAR
local1, local2 : DWORD;
END_VAR


END_FUNCTION
```

7.7 Разрешение имен

7.7.1 Области видимости

Каждый POU имеет локальную область видимости (переменные в разделах VAR, VAR_INPUT). Локальные переменные видны только внутри родительского POU.

В глобальной области видимости могут быть объявлены функции (глобальные), программы и глобальные переменные. Видны из любого POU проекта.

7.7.2 Конфликты

7.7.2.1 Дублирование имен

Дублирование имён в области видимости - объявление переменных и/или функций с одинаковыми именами приведёт к ошибке компиляции [E004].

7.7.2.2 Затенение имен

Если из POU видимы одноимённые **локальная** переменная и **глобальная** переменная/функция, то компиляция и выполнение произойдут корректно.

Выражение f() будет относиться к функции, а f - к локальной переменной.

В CODESYS V3.5 произойдёт ошибка компиляции.

7.7.2.3 Вызов функции/возвращаемый слот функции

Внутри функции не может быть одноименной локальной переменной (см. выше дублирование имен). Выражение f() будет относиться к функции, а f - к возвращаемому слоту.

Рекурсивный вызов функции допускается, глубина рекурсии не определена.

7.7.2.4 Имя типа/Имя функции/переменной

Допускается объявление переменных/функций одноимённых с типами данных.

В CODESYS V3.5 произойдёт ошибка компиляции.

7.8 Размещение данных в памяти

Данные POU размещаются в памяти последовательно в порядке объявления. Для элементарных типов данных занимаемый размер в памяти, выравнивание (кратность адреса) и сдвиг (занимаемый размер в массиве) совпадают. Для массивов выравнивание определяется выравниванием элемента массива.

Тип данных	Размер (size)	Выравнивание (alignment)	Сдвиг (stride)
BOOL	1		
WORD/UINT/INT	2		
DWORD/DINT/UDINT	4		
REAL	4		
LREAL	8		
STRING(len)	len+1	1	len+1
WSTRING(len)	2*len+2	2	2*len+2

7.8.1 Размещение структур

Элементы структуры размещаются в памяти последовательно - по порядку указания в описании типа. Выравнивание элементов выполняется по элементу (простого типа) с максимальным выравниванием.

Аналогичное правило действует для вложенных структур: выравнивание в памяти для надструктуры определяется наибольшим выравниванием элементов подструктуры.

Порядок байт/слов самих элементов структуры при этом не изменяется.

Примеры:

```
TYPE str1 : STRUCT          //size = 4
        a : DINT;           //alignment = 4
END_STRUCT END_TYPE

TYPE str2 : STRUCT
        a : DINT;          //size = 8
        x : BOOL;           //alignment = 4
END_STRUCT END_TYPE       //stride = 8

TYPE str3 : STRUCT
        a : WORD;
        x : BOOL;          //size = 8
        b : DWORD;         //alignment = 4
END_STRUCT END_TYPE      //stride = 8

TYPE str4 : STRUCT
       s1 : str1;         //size = 8
        x : BOOL;          //alignment = 4
END_STRUCT END_TYPE        //stride = 8
```

7.8.2 Размещение массивов

Элементы массива размещаются в памяти последовательно - по порядку от элемента с начальным индексом до элемента с конечным индексом. Выравнивание определяется выравниванием элемента массива. Сдвиг равен размеру в байтах.

Многомерные массивы (в том числе массивы массивов) располагаются в памяти по порядку указания осей. Элементы каждой оси располагаются по порядку от элемента с начальным индексом до элемента с конечным индексом. Пример: [a11, a12, a21, a22].

Порядок байт/слов самих элементов массива при этом не изменяется.

Примеры:

```
x1 : ARRAY [1..2] OF INT;                  //size = 4; al = 2; st = 4
x2_1 : ARRAY [1..2] [1..2] OF INT;          //size = 8; al = 2; st = 8
x2_2 : ARRAY [1..2] OF ARRAY[1..2] OF INT;  //size = 8; al = 2; st = 8
```

ЦИФРОВЫЕ
РЕШЕНИЯ

ООО "Овен Цифровые решения"
Россия, г. Москва, пл. Семёновская, д. 1А, помещ. 3/1
support@owendigital.ru
www.owendigital.ru

рег.:1-RU-1-1.0