ЦИФРОВЫЕ РЕШЕНИЯ

ALTA IDE (версия 1.0)

Руководство пользователя

03.2026
версия 1.19

Содержание

Предисловие

ALTA IDE — это современная среда разработки, которая в настоящий момент находится в стадии активного создания и совершенствования. Функциональные возможности среды постоянно расширяются и дорабатываются. На текущем этапе не все функции могут работать так, как задумано изначально, однако мы ежедневно работаем над улучшением стабильности, удобства и производительности ALTA IDE.

Если в процессе работы вы обнаружили ошибку, некорректное поведение или несоответствие ожидаемому результату, пожалуйста, сообщите об этом по электронной почте: **support@owen.ru** или **alta@owen.ru**. Ваша помощь позволяет сделать ALTA IDE лучше с каждым днем.

Спасибо, что используете ALTA IDE и участвуете в её развитии.

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,позволяющие структурировать проект, повышать модульность и переиспользование кода.

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

ST (Structured Text) — текстовый язык программирования стандарта МЭК 61131-3. Предназначен для программирования промышленных контроллеров и операторских станций. По сравнению с графическими языками занимает меньше места, логику программы, написанной на языке ST легче декодировать и понимать.

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

Глобальные переменные — переменные, включая 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 (далее по тексту — 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.

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 в низкоуровневый код, который исполняется 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.

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

Таргет — это ключевой компонент, описывающий конкретное устройство, его ресурсы, периферию и специфические функции, позволяя среде ALTA IDE взаимодействовать с ним, загружать программу и управлять аппаратным обеспечением. Таргеты предоставляются производителями устройств.

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

Устройство — программируемое устройство, например ПЛК110.

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.

1 О программе

ALTA IDE — это среда программирования, которая предоставляет комплекс инструментов для разработки программ для промышленных ПЛК.

Функции ALTA IDE:

- создание приложения пользователя на языке ST стандарта МЭК 61131-3;
- отладка созданной программы на подключенном устройстве;
- загрузка приложения на устройство;
- настройка сетевого обмена для режимов Master и Slave для протоколов Mobus Serial и Mobus TCP;
- мониторинг работы программы.

ALTA IDE поддерживает современные средства, упрощающие разработку:

- подсветка синтаксиса;
- автодополнение кода;
- 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.

2 Установка ПО

При установке среды загружаются базовые библиотеки, при необходимости дополнительные библиотеки можно загрузить через Менеджер библиотек, или в web-конфигураторе контроллера во вкладке ПЛК/ Загрузки.

ПРИМЕЧАНИЕ
Менеджер библиотек не доступен в ALTA IDE версии 1.0.

2.1 Установка

1. Запустите файл *AltaSetup.exe*.
2. Откроется окно мастера установки. Выберите папку для установки ALTA IDE. Нажмите кнопку **Далее**.

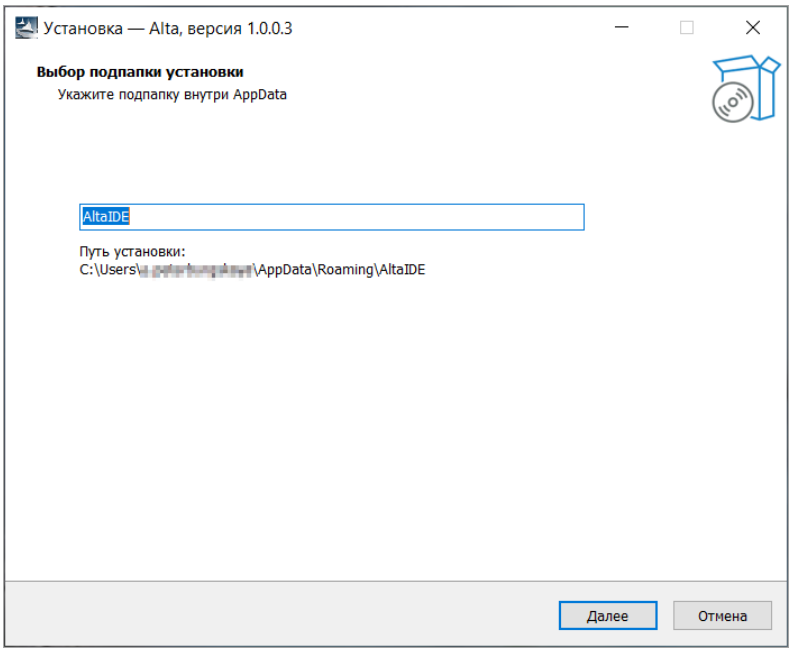


Рисунок 2.1

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

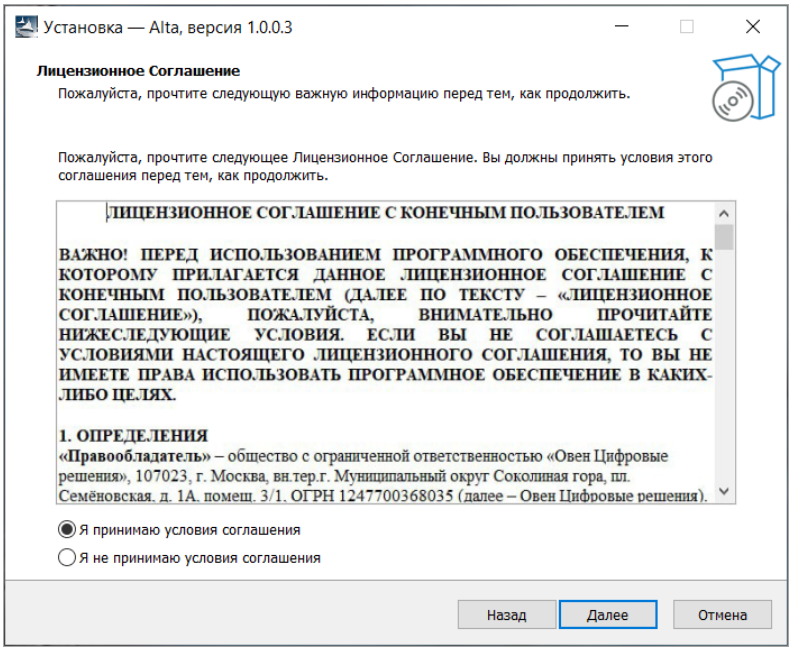


Рисунок 2.2

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

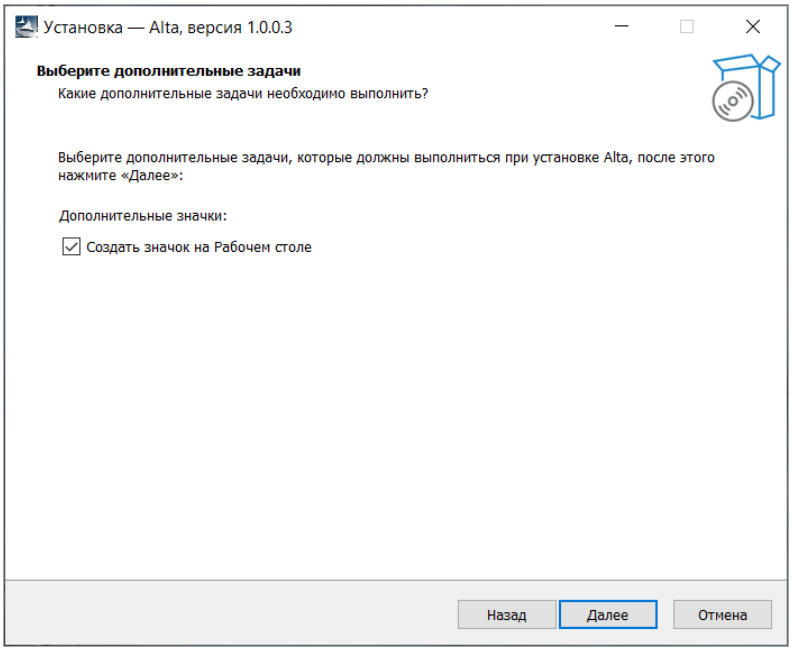


Рисунок 2.3

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

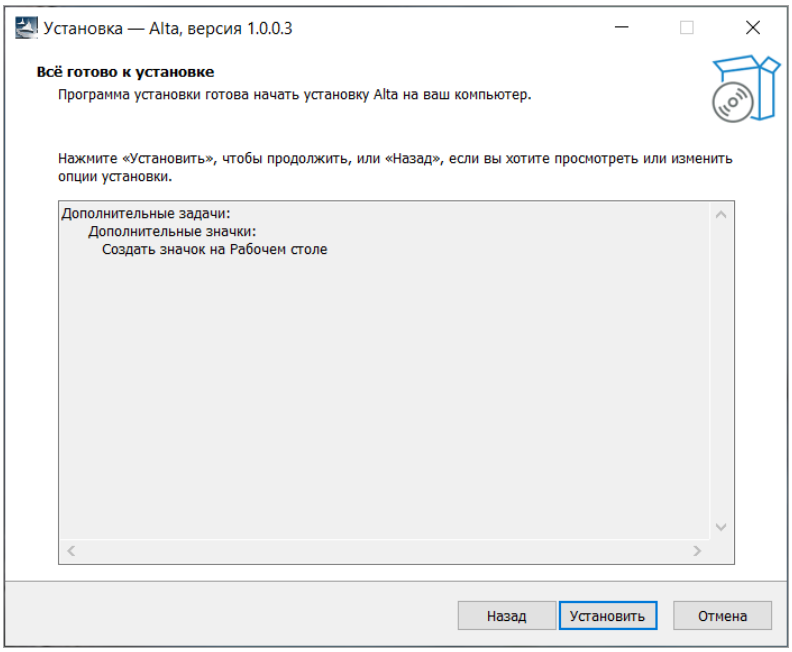


Рисунок 2.4

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

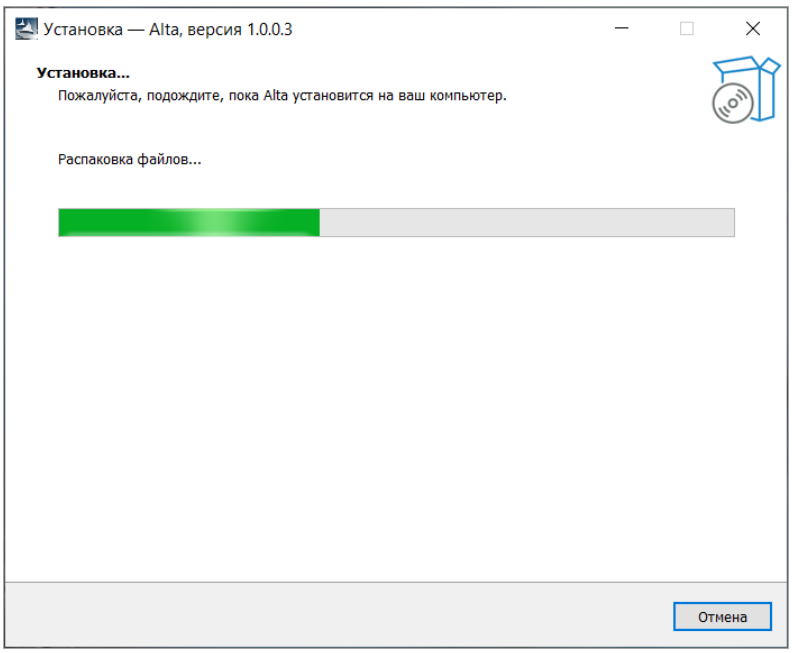


Рисунок 2.5

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

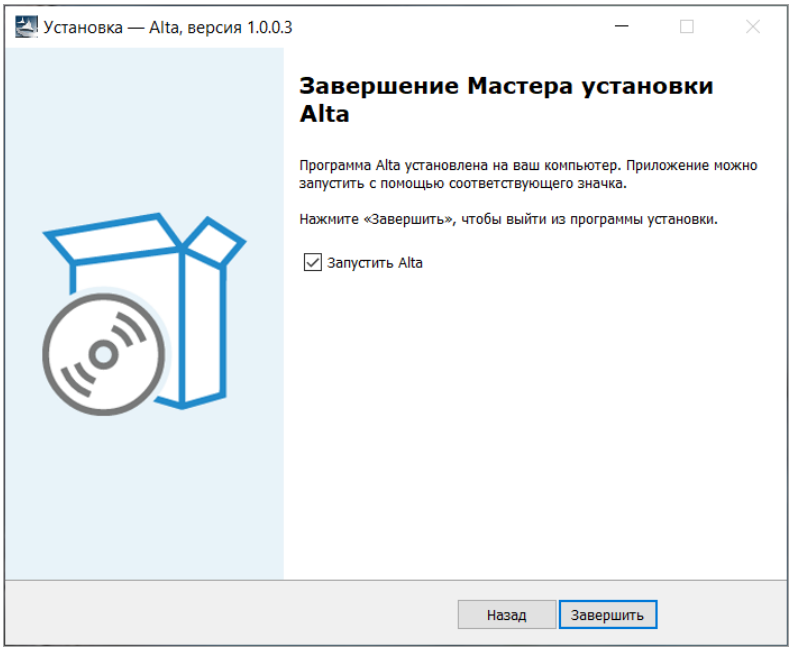


Рисунок 2.6

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:

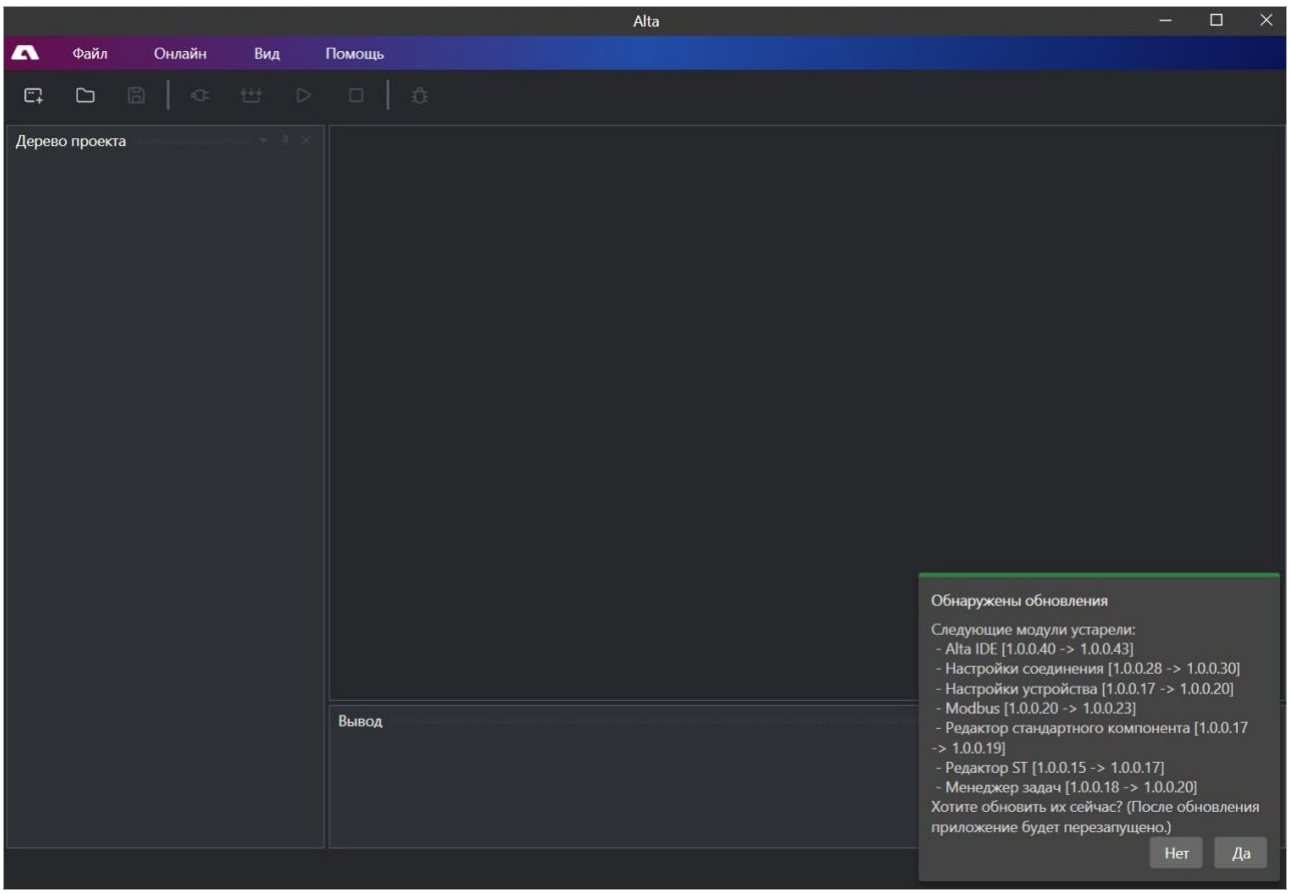


Рисунок 2.7

Для обновления среды выберите **Да** — ALTA IDE обновится и автоматически перезапустится.

Если выбрать **Нет**, либо не сделать выбор в течение 10 секунд, информационное окно закроется. В строке состояния появится иконка сигнализирующая о наличии обновления  , при нажатии на которую откроется информационное окно со списком доступных обновлений ALTA IDE, и возможностью выбора обновления ПО.

3 Описание интерфейса

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

ПРИМЕЧАНИЕ
В ALTA IDE версии 1.0 поддерживается только темная тема.

После установки и запуска ALTA IDE откроется главное окно:

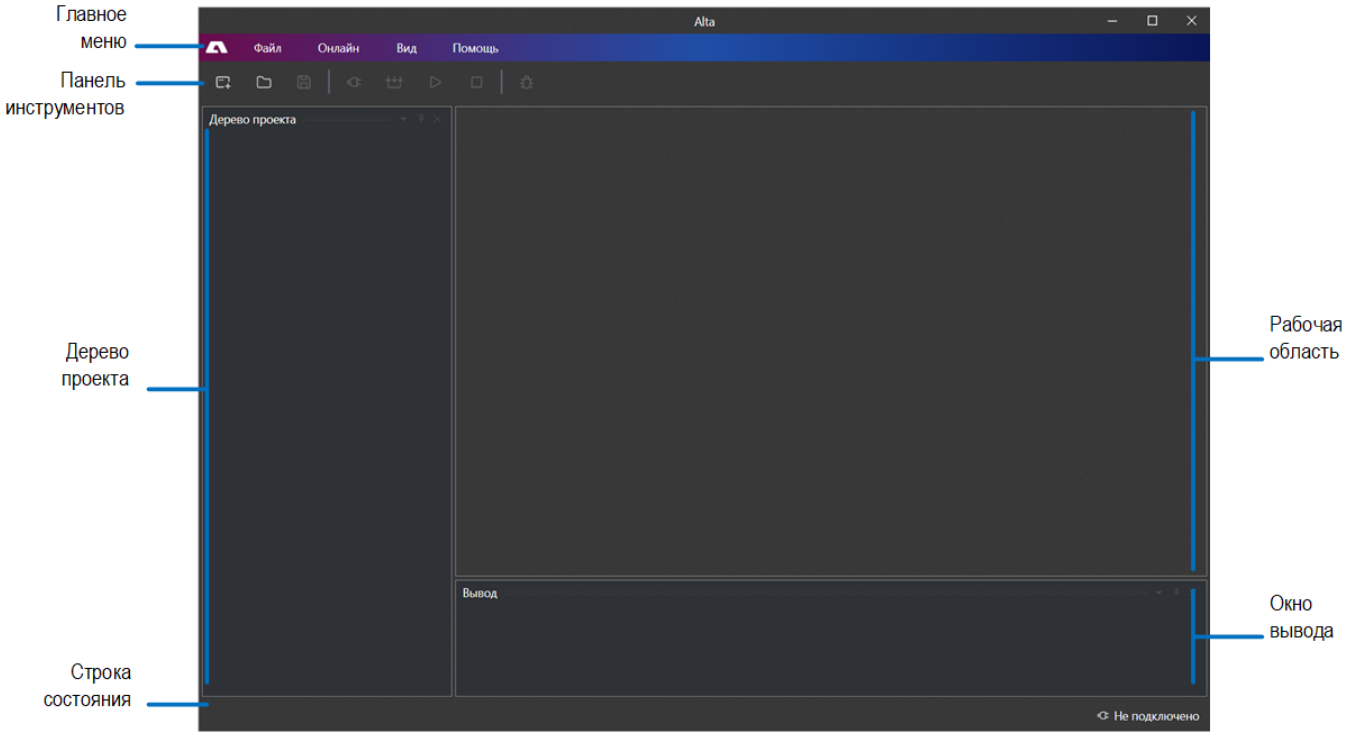


Рисунок 3.1 – Главное окно

Главное окно ALTA IDE содержит:

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

Интерфейс ALTA IDE можно настраивать: рабочие окна среды открепляются от главного окна и могут перемещаться независимо от основного окна ALTA IDE.

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

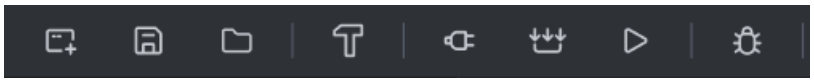
Темы	Выбор цветовой темы интерфейса: темной/светлой **ПРИМЕЧАНИЕ** В ALTA IDE версии 1.0 поддерживается только темная тема.

Помощь

Справка	Вызов руководства пользователя ALTA IDE
О программе	Вызов окна информации о текущей версии ALTA IDE и компонентах среды

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:

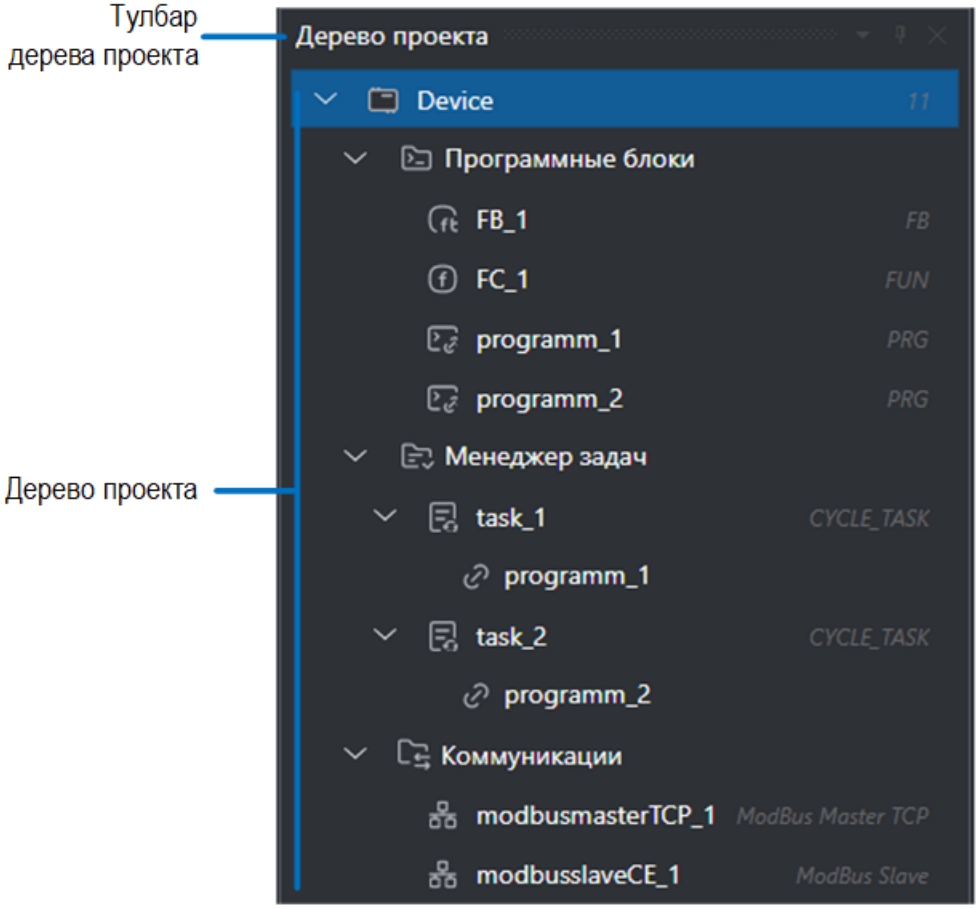


Рисунок 3.2 – Дерево проекта

В дереве проекта отображается наименование устройства (с модификацией) для которого создан проект, а также обязательные системные папки:

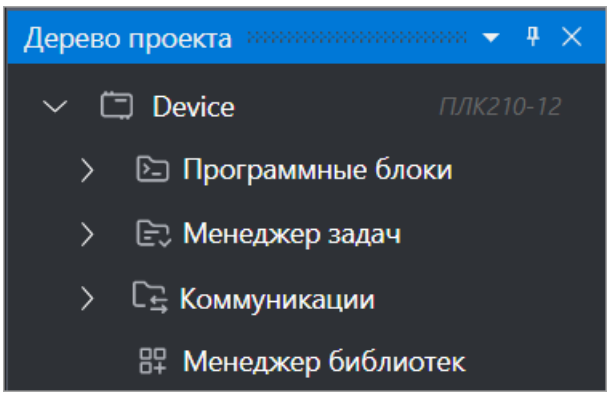


Рисунок 3.3 – Системные папки дерева проекта

	Устройство	Содержит аппаратные настройки, возможности устройства
	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
	Менеджер библиотек	Содержит перечень всех доступных библиотек, с возможностью просмотра информации о библиотеке и управления подключением к проекту

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, выносить на другой экран ПК и при необходимости закреплять обратно.

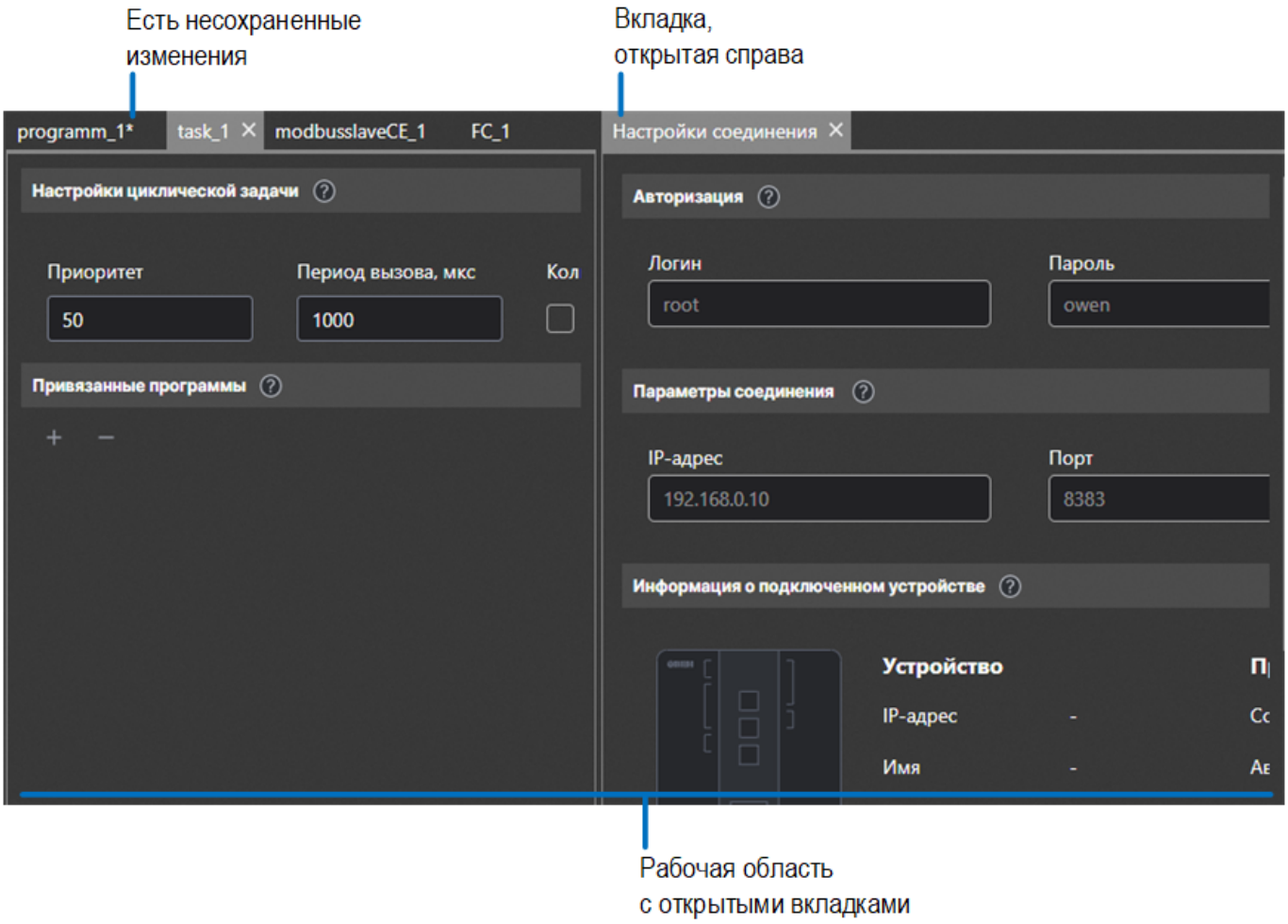


Рисунок 3.4 – Рабочая область ALTA IDE

Значок * рядом с названием вкладки говорит о 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	не подключено
194.168.0.20	установка соединения
194.168.0.20	соединение установлено
194.168.0.20	ошибка соединения
	доступно обновление ALTA IDE

При выполнении некоторых задач в проекте, например при загрузке проекта в прибор, или запуске сборки проекта, в строке состояния отображается шкала с индикацией выполнения:

Рисунок 3.5

3.7 Сочетания клавиш

Сочетание клавиш	Действие
Дерево проекта	
F2	Переименовать
Delete	Удалить выбранный элемент
Enter	Открыть
Ctrl + Enter	Открыть справа
Работа с проектом	
Ctrl + N	Создать новый проект
Ctrl + O	Открыть существующий проект
F11	Выполнить сборку проекта
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
Ctrl + Space	Вызвать окно автодополнения
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
Меню/Помощь	
F1	Вызов справки

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;
- мониторинг работы приложения в режиме онлайн и отладка программы.

4.1 Создание проекта

После установки и запуска откроется главное окно, в котором доступно создание проекта.

1. **Создать проект** можно одним из способов:
   - воспользуйтесь кнопкой **Создать проект** на панели инструментов;
   - или пунктом **Главное меню** → **Файл** → **Создать проект**.
2. В открывшемся окне введите данные:

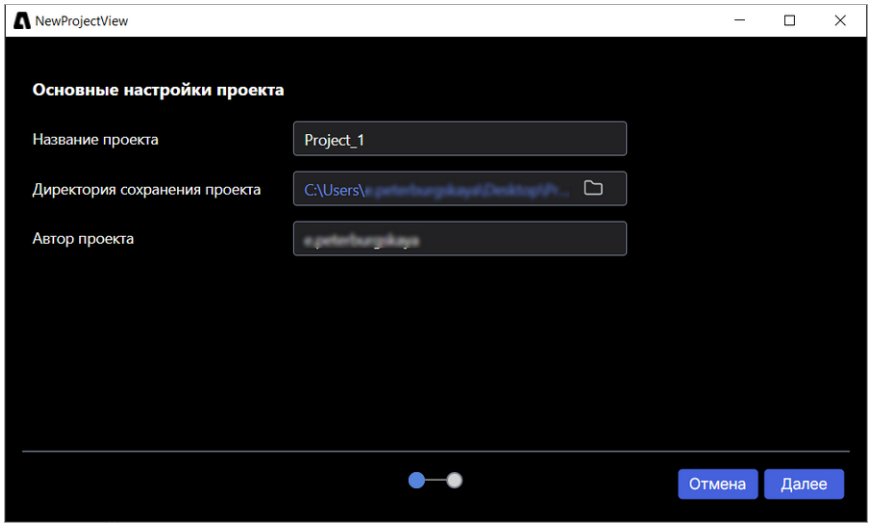


Рисунок 4.1

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**.

3. Выберите устройство и модификацию:

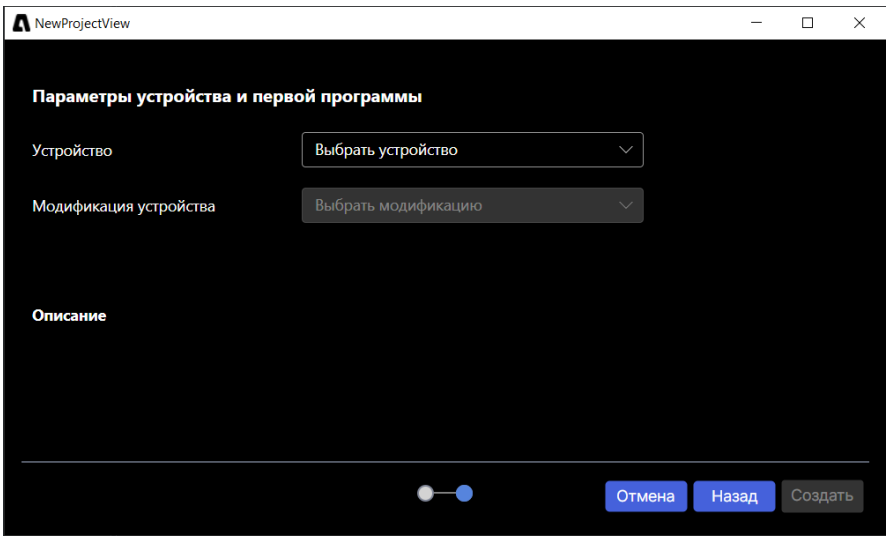


Рисунок 4.2

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

Модификация устройства — из выпадающего списка выберите модификацию устройства;

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

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

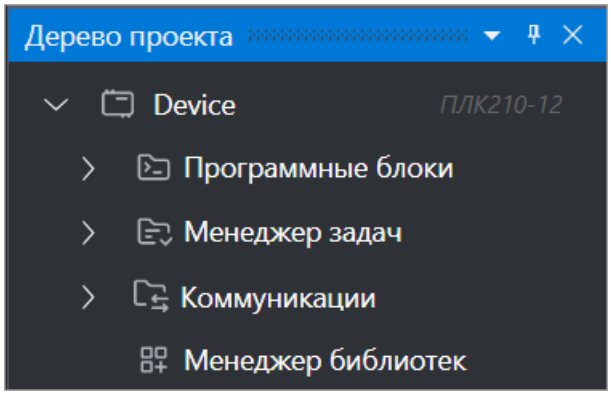


Рисунок 4.3

4.1.1 Открытие проекта

1. **Открыть** ранее сохраненный проект можно одним из способов:

   - воспользуйтесь кнопкой **Открыть проект**  на панели инструментов;
   - или пунктом **Главное меню** → **Файл** → **Открыть проект**.

2. В открывшемся окне выберите файл проекта с расширением *.alta* и нажмите **Открыть**.

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

Возможные ошибки

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:

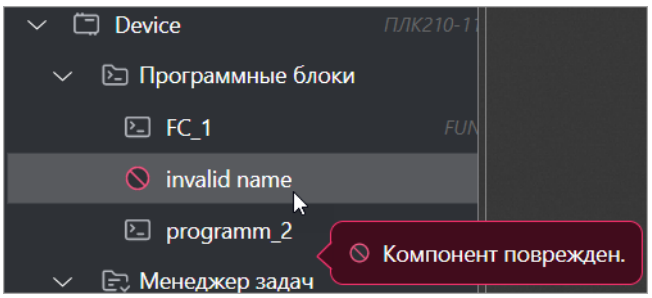


Рисунок 4.4

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

Системная папка **Программные блоки** располагается в дереве проекта и позволяет создавать программные объекты:

- **Программа** — высокоуровневый программный объект, привязываемый к задачам устройства. 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. 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:

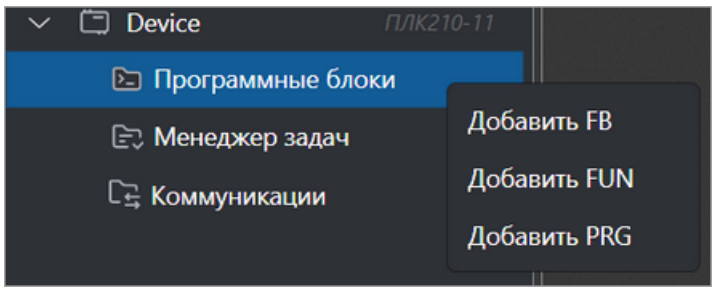


Рисунок 4.5 – Контекстное меню системной папки Программные блоки

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

Добавленный программный объект отобразится в дереве проекта, как ответвление системной папки Программные блоки:

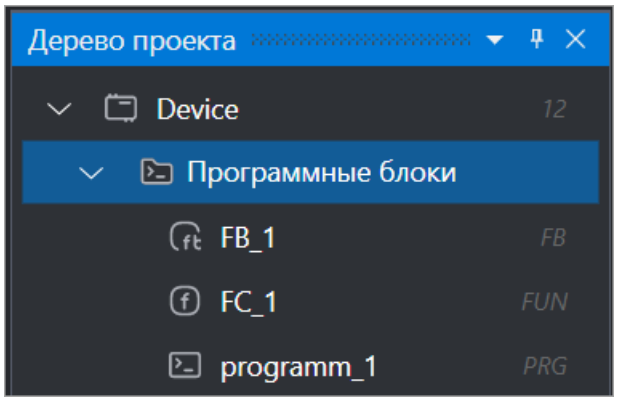


Рисунок 4.6

Созданный программный объект по умолчанию имеет имя *_1 (имя зависит от типа 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 именования языка ST, и убедиться, что выбранное имя не дублирует названия других компонентов в дереве проекта - создание компонентов с одинаковыми именами не допускается.

Контекстное меню

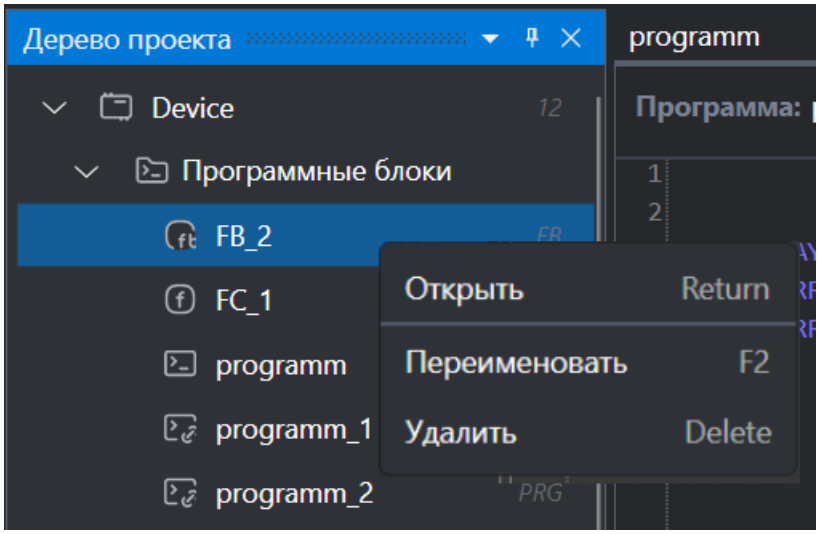


Рисунок 4.7

С помощью контекстного меню программы, функции и функционального блока можно выполнить следующие действия:

- **Открыть** — открыть вкладку редактора программного объекта в рабочей области.
- **Переименовать** — изменить имя программного объекта.

Если изменить имя программы, привязанной к задаче, то после переименования программа 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:

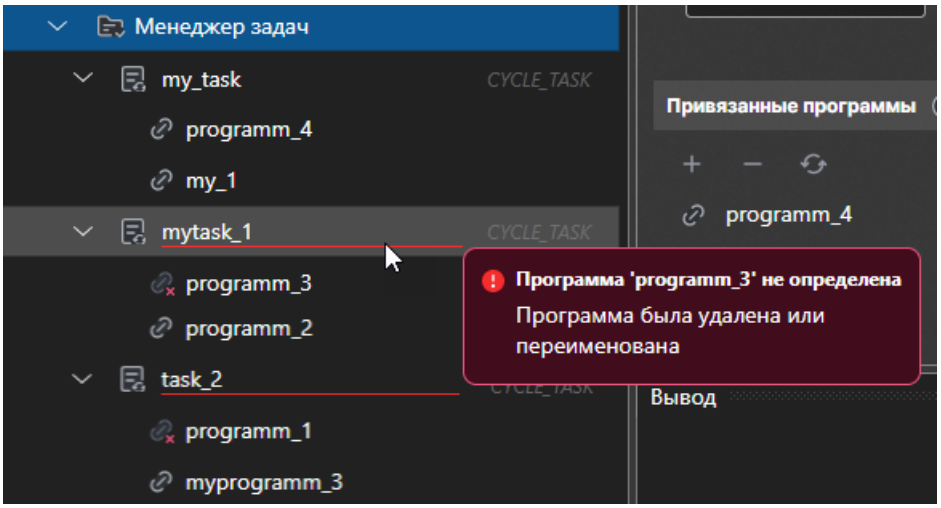


Рисунок 4.8 – Отображение ошибки в дереве проекта

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.

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 далее.

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

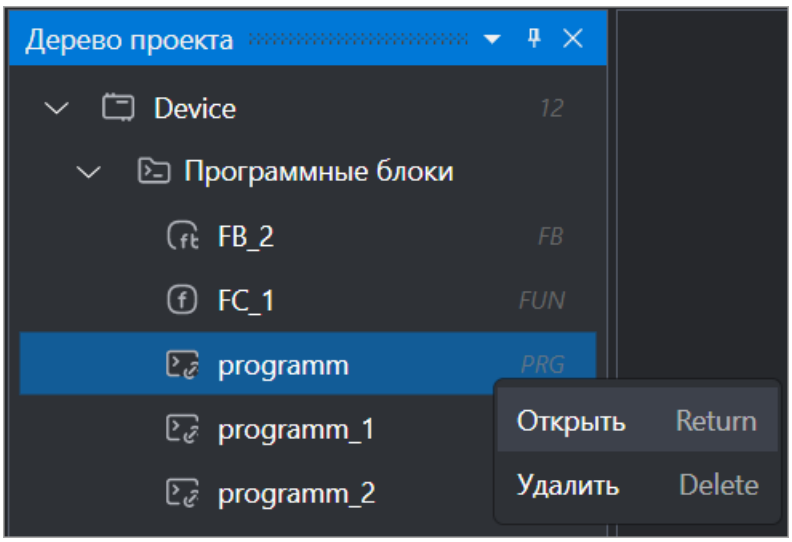
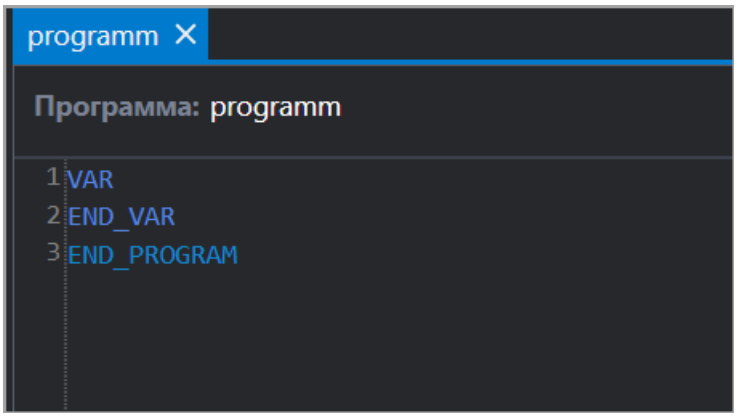
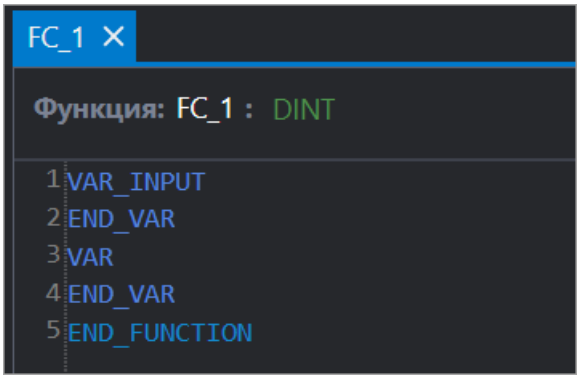
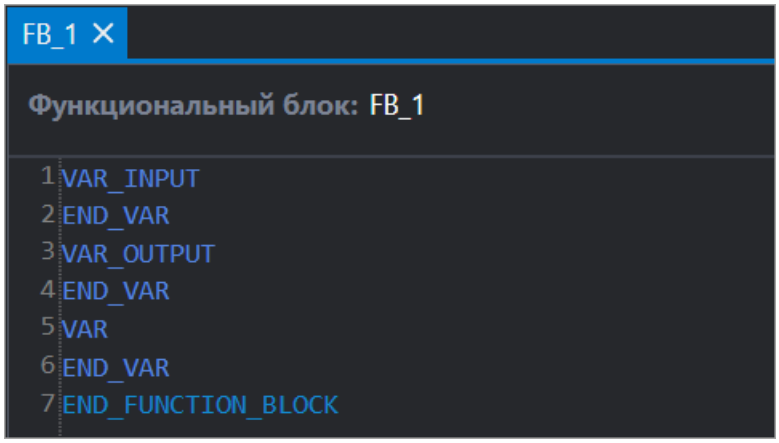


Рисунок 4.9 – Контекстное меню POU

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

Вкладка редактора ST программы	 **Рисунок 4.10** Содержит нередактируемый заголовок **Программа: <имя_программы>**, который является объявлением программы (дополнительно объявлять программу в коде не требуется). Строка *END_PROGRAM* является обязательной завершающей частью конструкции объявления компонента. В теле программы расположен блок объявления локальных переменных в текстовом виде.

Вкладка редактора ST функции	 **Рисунок 4.11** Содержит заголовок **Функция: <имя_функции>**, который является объявлением функции (дополнительно объявлять функцию в коде не требуется). Строка *END_FUNCTION* является обязательной завершающей частью конструкции объявления компонента. В теле функции расположен блок объявления локальных переменных в текстовом виде. Тип возвращаемого значения можно задать в редактируемой части заголовка. Сменить тип возвращаемого значения можно в любой момент, для этого начните вводить новое значение — автодополнение предложит возможные варианты.
Вкладка редактора ST функционального блока	 **Рисунок 4.12** Содержит нередактируемый заголовок **Функциональный блок: <имя_фб>**, который является объявлением функционального блока (дополнительно объявлять функциональный блок в коде не требуется). Строка *END_FUNCTION_BLOCK* является обязательной завершающей частью конструкции объявления компонента. В теле программы расположен блок объявления локальных входных и выходных переменных в текстовом виде.

ПРИМЕЧАНИЕ
В одном POU доступно объявление только одного программного объекта. В случае объявления нескольких программных объектов в одном POU возможны ошибки компиляции.

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.

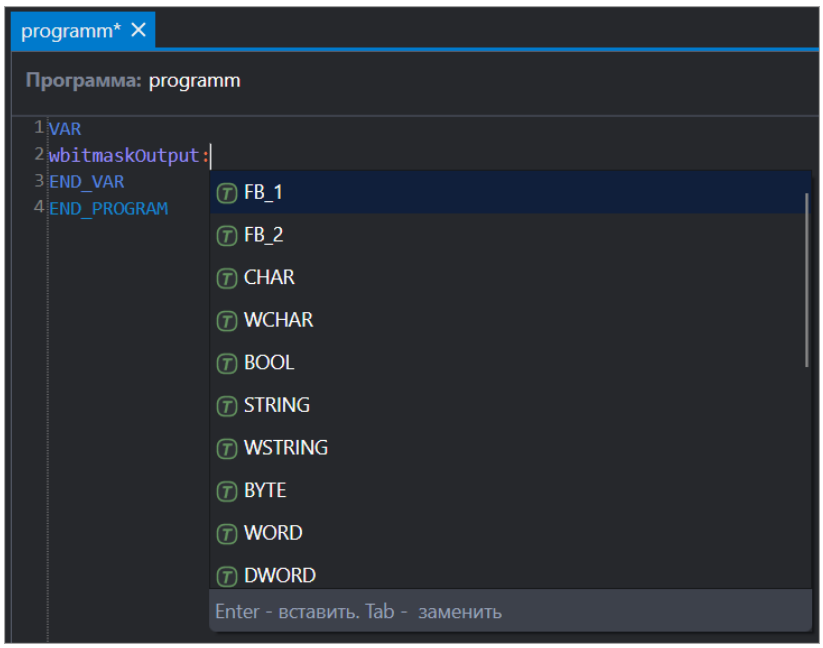


Рисунок 4.13

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.

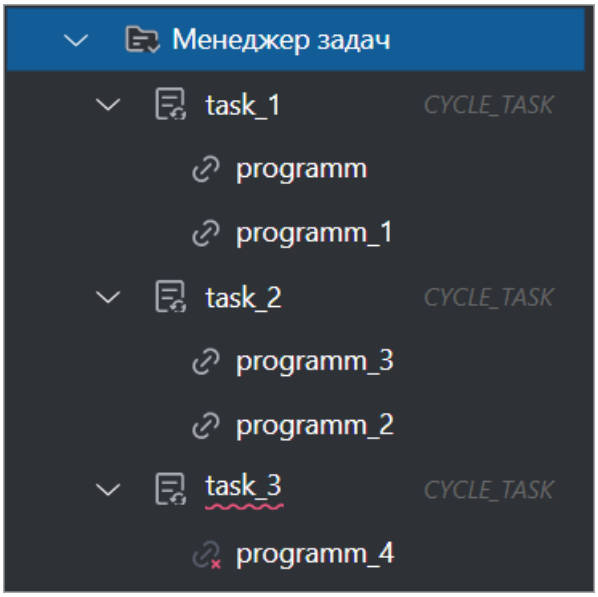


Рисунок 4.14 – Менеджер задач в дереве проекта

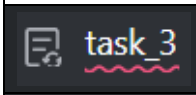
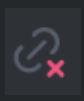
Под каждой задачей располагается список привязанных программ (при наличии), которые будут выполняться в рамках задачи. Разрешенное количество привязываемых к задаче программ зависит от прибора. Для того, чтобы свернуть/развернуть список привязанных программ нажмите на стрелку, расположенную слева от имени задачи. Если стрелка отсутствует, значит к задаче не привязано ни одной программы.

Тип задач и вложенных элементов можно определить по иконке в дереве проекта:

	Циклическая задача
	Задача по событию **ПРИМЕЧАНИЕ** В ALTA IDE версии 1.0 для создания доступны только циклические задачи.
	Программа, привязанная к задаче

В случае возникновения ошибок, иконки помечаются маркерами:

Таблица 4.1 – Возможные ошибки

Графическое отображение	Тип ошибки	Варианты устранения ошибки
	Задача содержит ошибку	Наведите мышку для получения подсказки
	Программа привязана к нескольким задачам	Удалите повторные привязки программы - для корректной работы программа может быть привязана к задаче только один раз
	Привязанная программа не существует (удалена или переименована)	• Создайте программу с именем удаленной/переименованной программы, она автоматически привяжется к задаче. • Удалите привязку удаленной/ переименованной программы к задаче

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

Чтобы открыть вкладку **Менеджер задач** дважды нажмите ЛКМ на системную папку **Менеджер задач** в дереве проекта, или выберите в контекстном меню пункт **Открыть**:

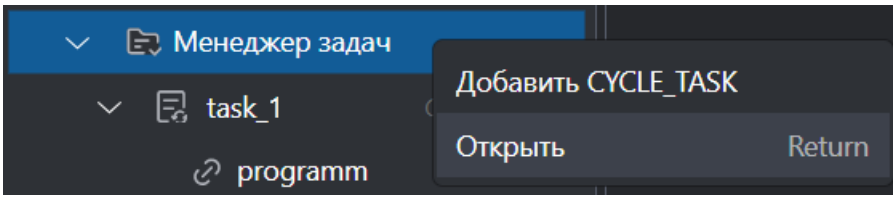


Рисунок 4.15

Вкладка **Менеджер задач** содержит информацию о созданных задачах, основных настройках, и привязанных программах (при наличии):

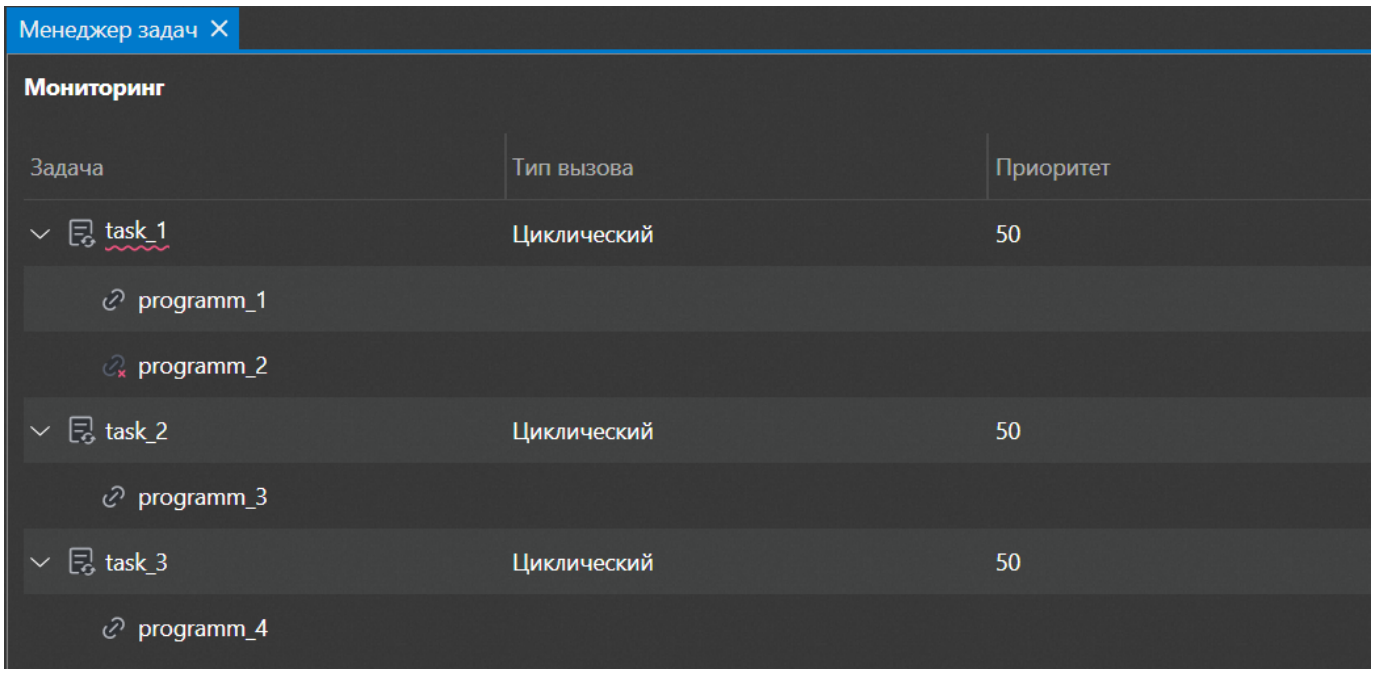


Рисунок 4.16 – Вкладка Менеджер задач

- **Задача** — имя созданной задачи и имена программ, привязанных к задаче (при наличии).
- **Тип вызова** — циклическая задача или задача по событию;

  ПРИМЕЧАНИЕ
  В ALTA IDE версии 1.0 для создания доступны только циклические задачи.

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

Порядок выполнения задач определяется согласно установленному приоритету и условию вызова. Условием вызова задачи может служить:

- время (циклическое выполнение);
- событие внутреннее или внешнее (например, превышение заданного порога глобальной переменной или прерывание в устройстве).

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. Программы одной задачи выполняются в том же порядке, в каком они перечислены в списке **Менеджера задач** и расположены в дереве проекта.
4. Перед выполнением привязанных программ в задаче происходит синхронизация входов компонентов с переменными, используемыми в программах задачи. После выполнения программ происходит аналогичная синхронизация выходов. При каждом старте программы происходит фиксация значений всех переменных проекта, которые были зафиксированы, независимо от того используются ли переменные в выполняемой программе или нет.

Для упрощения понимания логики работы приложения на устройстве ниже приведен пример последовательности выполнения задач проекта.

Пример

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

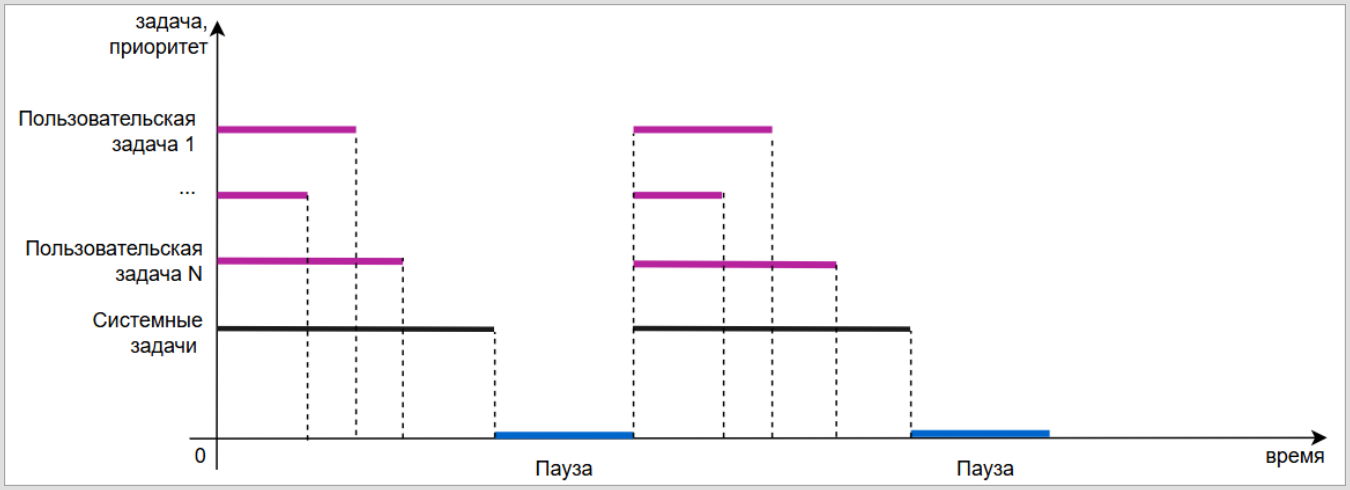


Рисунок 4.17

4.2.2.1 Действия с задачами

Добавление задачи

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

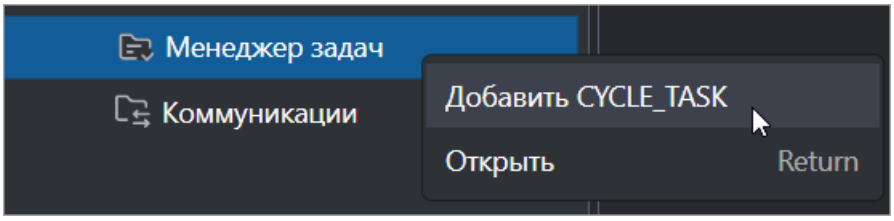


Рисунок 4.18 – Контекстное меню системной папки Менеджер задач

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

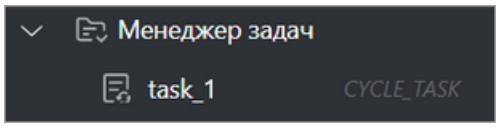


Рисунок 4.19

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, либо через контекстное меню задачи. При задании компоненту имени следует учитывать правила именования языка ST, и убедиться, что выбранное имя не дублирует название других компонентов в дереве проекта - создание компонентов с одинаковыми именами не допускается.

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

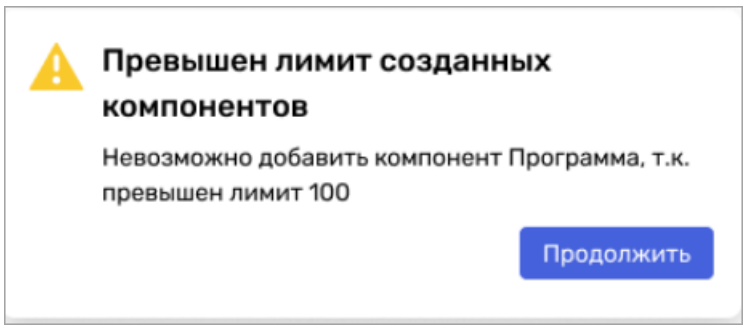


Рисунок 4.20 – Информационное окно

ПРИМЕЧАНИЕ
Максимальное количество добавленных задач зависит от прибора.

Контекстное меню

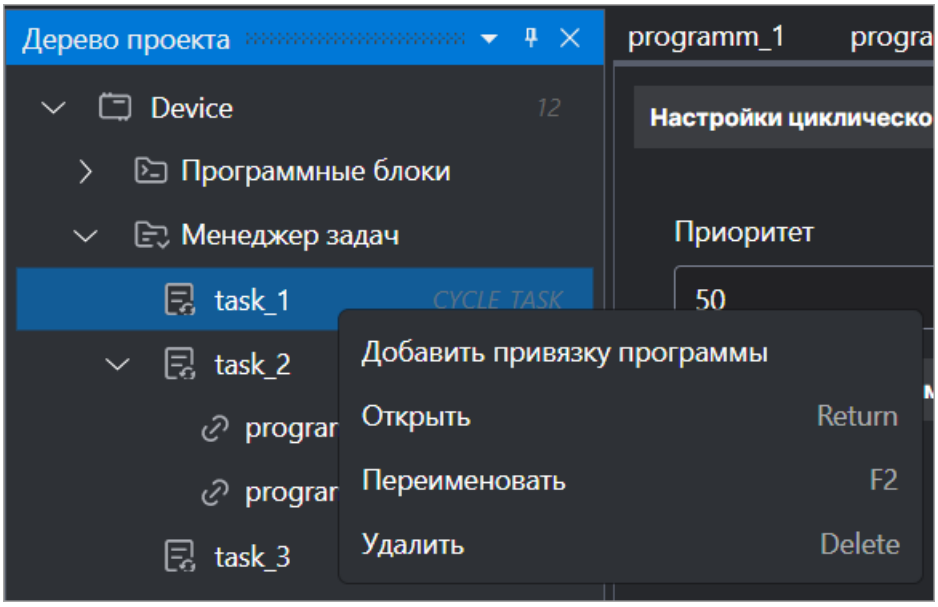


Рисунок 4.21 – Контекстное меню задачи

С помощью контекстного меню задачи можно выполнить следующие действия:

- **Добавить привязку программы** — привязать программу к задаче.
- **Открыть** — открыть вкладку редактора задачи в рабочей области.
- **Переименовать** — изменить имя задачи. Компоненты, в которых использовалась ссылка на переименованную задачу, будут помечены маркером ошибки.
- **Удалить** — удалить выбранную задачу.

ПРИМЕЧАНИЕ
При удалении задачи с привязанными программами удаляется только задача, программы остаются в проекте и переходят в статус непривязанных.

После добавления задачи необходимо настроить условия выполнения и привязать программы.

4.2.2.2 Привязка программы

Привязать программу к задаче можно несколькими способами:

1. С помощью drag&drop программы в дереве проекта из папки программные блоки в папку:
   - **Менеджер задач** → **Задача**: привязанная программа добавляется вниз списка;
   - **Менеджер задач** → **Задача** → **Программа**: привязанная программа добавляется на место выбранной программы, остальные программы сдвигаются вниз.
2. С помощью контекстного меню элемента **Задача** в дереве проекта, выберите пункт **Добавить привязку программы**:

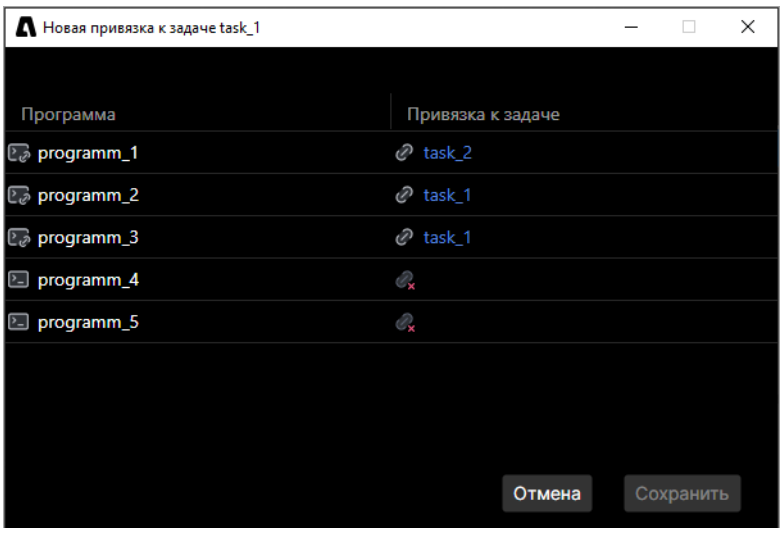


Рисунок 4.22 – Окно привязки программ

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

Для привязки нескольких программ к задаче нажмите и удерживайте кнопку Shift и с помощью ЛКМ выделите программы, которые требуется привязать. Нажмите **Сохранить** - выделенные программы будут привязаны к задаче.

3. В окне редактора задачи, в разделе "Привязанные программы" с помощью кнопки  :

Возможные ошибки привязки программы

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

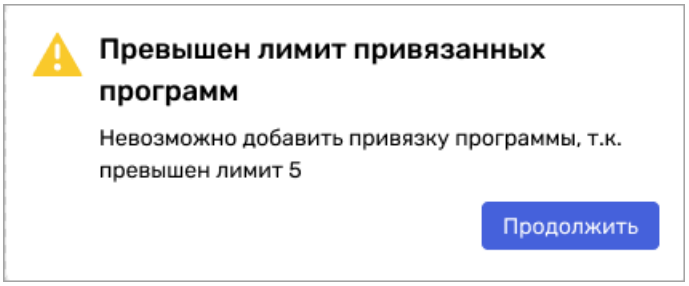


Рисунок 4.23 – Информационное окно

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.

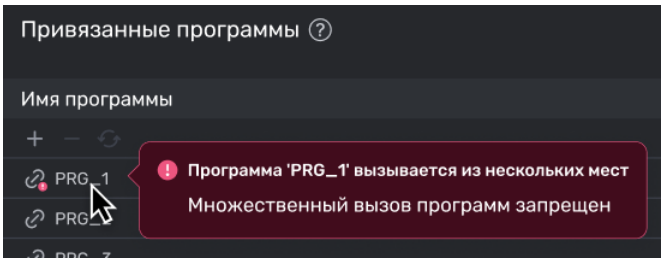


Рисунок 4.24 – Отображение множественного вызова в дереве проекта

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:

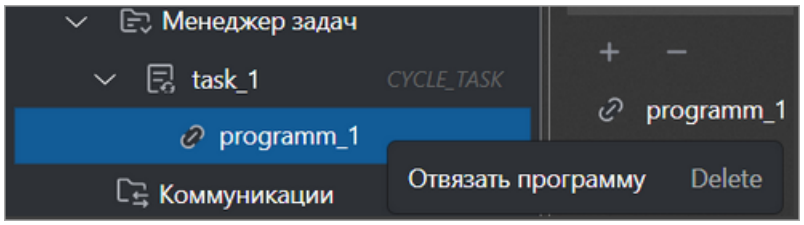


Рисунок 4.25

Выберите **Отвязать программу**, или воспользуйтесь кнопкой  в окне редактора задачи, в разделе "Привязанные программы". Программа перейдет в статус непривязанной.

4.2.2.3 Циклическая задача

Циклическая задача вызывает привязанные программы периодически в соответствии с заданным пользователем интервалом времени.

Схема выполнения циклической задачи на ПЛК приведена ниже:

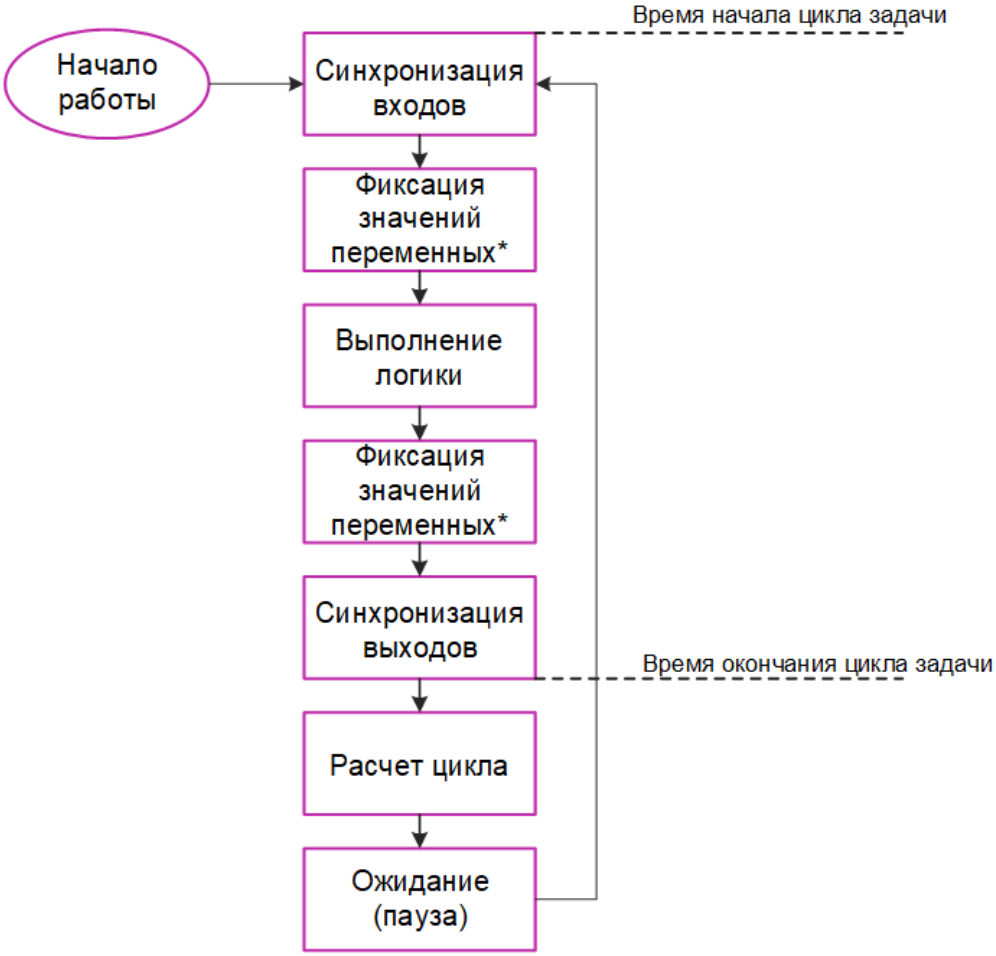


Рисунок 4.26

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

После создания задачи необходимо задать настройки. Для этого **откройте** окно редактора задачи одним из следующих способов:

- дважды щёлкните ЛКМ по элементу **Задача** в дереве проекта;
- выберите пункт **Открыть** в контекстном меню элемента **Задача** в дереве проекта.

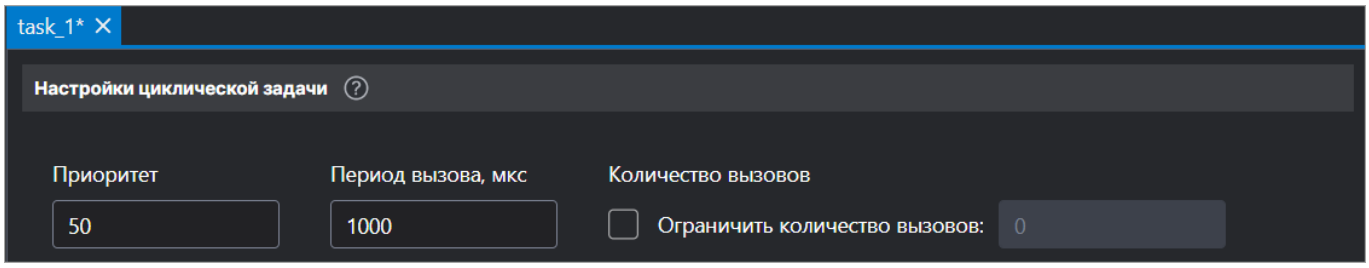


Рисунок 4.27 – Настройки циклической задачи

Заполните:

- **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;

В случае недопустимого значения любого из параметров окно ввода будет выделено красным цветом:

- поле параметра не может быть пустым;
- введенное значение не соответствует типу данных, допустимый формат ввода целочисленный;
- введенное значение вне диапазона.

При наведении курсора мыши отобразится сообщение с текстом ошибки:

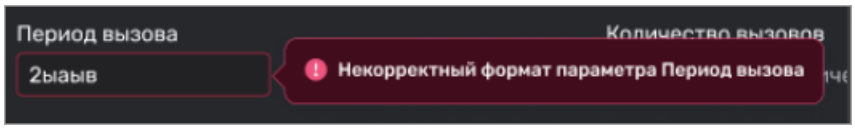


Рисунок 4.28

Раздел редактора циклической задачи **Привязанные программы** содержит список привязанных программ в порядке их выполнения.

Рисунок 4.29

Нажмите кнопку  для привязки программы к задаче.

Чтобы удалить привязку программы нажмите кнопку  .

4.3 Сохранение и сборка проекта

Сохранение проекта

Сохранить все изменения в проекте можно одним из следующих способов:

1. Через **Главное меню→Файл→Сохранить проект**.
2. Нажатием кнопки **Сохранить проект**  на панели инструментов.
3. С помощью сочетания клавиш *Ctrl+S*.

Информация будет сохранена в папку проекта, по пути, указанному при создании проекта.

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

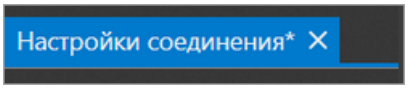


Рисунок 4.30

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. При выборе **Нет** вкладка закроется без сохранения изменений.

Сборка проекта

Сборка проекта — это процесс подготовки проекта к запуску на устройстве. В ходе сборки все файлы проекта обрабатываются и преобразуются в исполняемые файлы с приложением, которые содержат пользовательскую программу и могут быть загружены и выполнены на устройстве.

Для **Сборки проекта** 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 сборки:

Рисунок 4.31 – Шкала сборки проекта

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

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

Процесс сборки проекта можно остановить одним из следующих способов:

- нажмите кнопку  на панели инструментов;
- нажмите на значок  на шкале сборки проекта, расположенной в строке состояния главного окна ALTA IDE.

Процесс сборки проекта будет остановлен.

5 Работа с устройством

Для настройки устройства выполните двойной клик ЛКМ по элементу **Устройство** в дереве проекта, либо воспользуйтесь контекстным меню **Устройства**: выберите пункт **Открыть** или **Открыть справа**. Откроется вкладка с настройками, которая состоит из дерева настройки устройства и рабочей области:

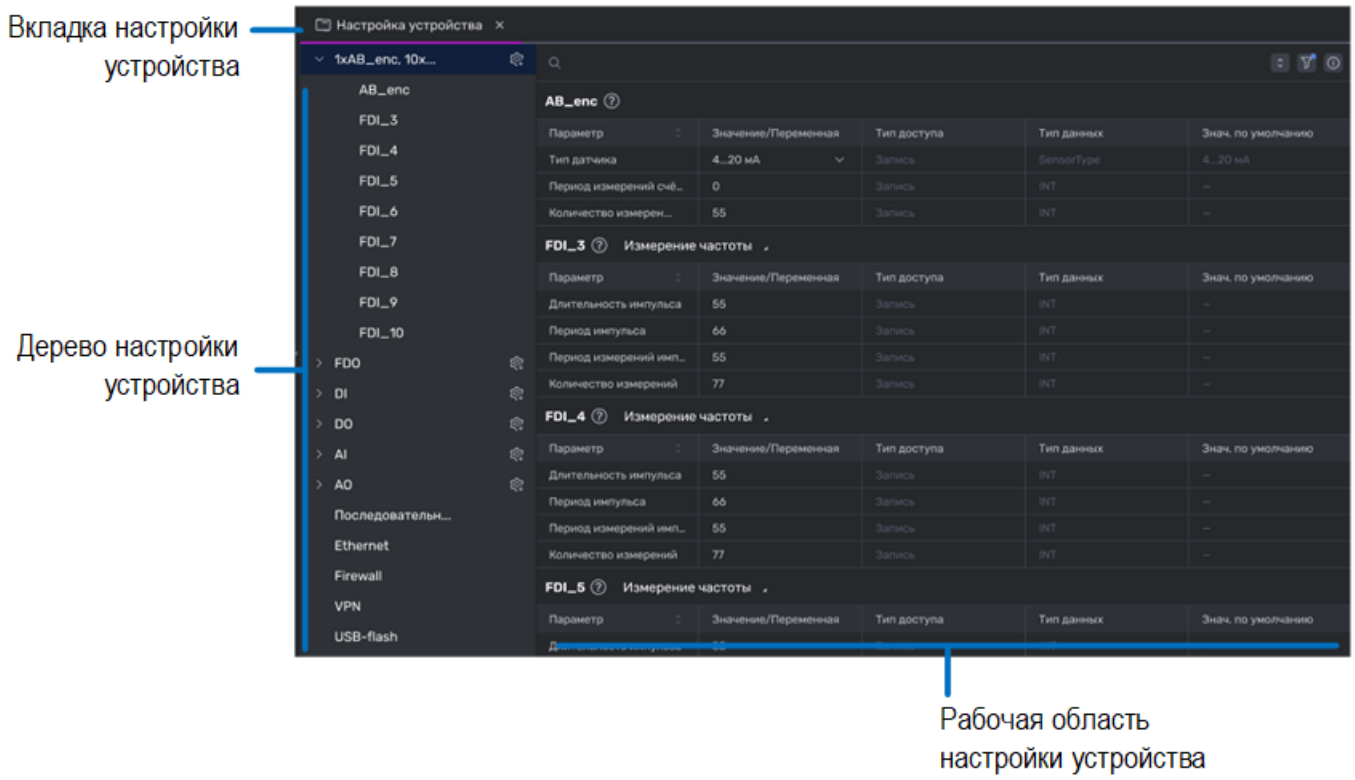


Рисунок 5.1 – Окно Настройка устройства

Наполнение **Дерева настройки устройства** зависит от устройства.

С помощью контекстного меню элемента **Устройство** в **Дереве проекта** можно выполнить ряд действий:

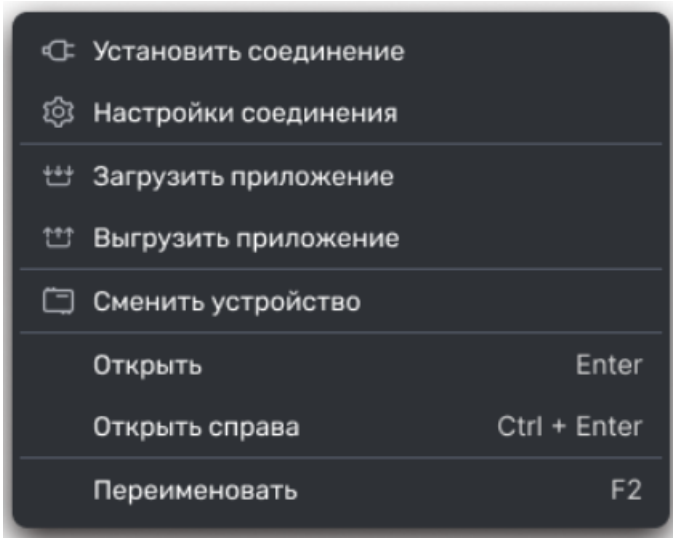


Рисунок 5.2

- **Установить соединение** — установка соединения с устройством;
- **Настройки соединения** — настройка параметров соединения;
- **Загрузить приложение** — импорт приложения на устройство;
- **Выгрузить приложение** — экспорт приложения с устройства;
- **Открыть** — открытие окна настроек устройства;
- **Открыть справа** — открытые окна настроек устройства в рабочей области справа;
- **Переименовать** — переименование устройства.

5.1 Настройка входов/выходов устройства

Откройте окно **Настройки устройства** с помощью двойного клика ЛКМ на элемент **Устройство** в дереве проекта, или контекстного меню элемента **Устройство** — выберите пункт **Открыть** или **Открыть справа**. В **Дереве настройки устройства** входы/выходы сгруппированы по типам:

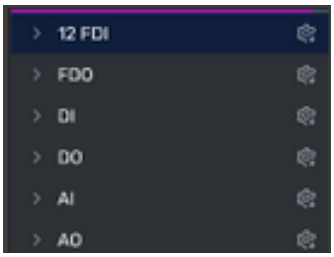


Рисунок 5.3

ПРИМЕЧАНИЕ
Наполнение области настройки входов/выходов зависит от устройства.

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:

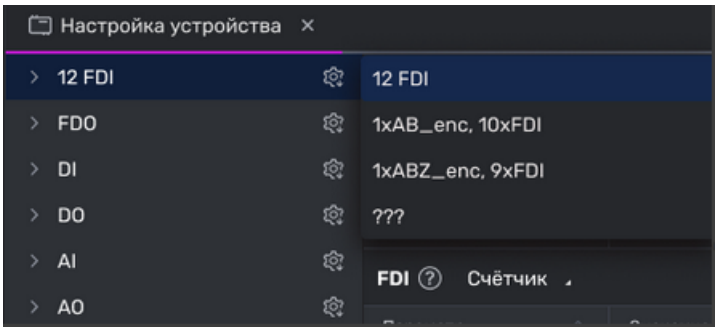


Рисунок 5.4

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

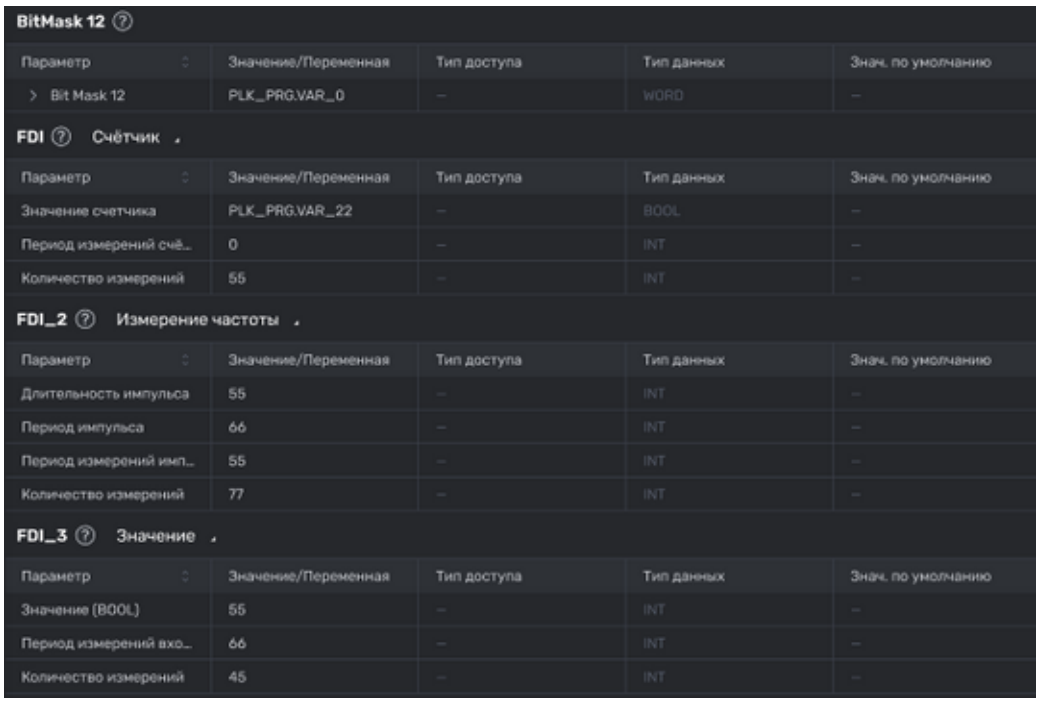
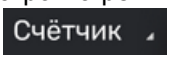


Рисунок 5.5

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:

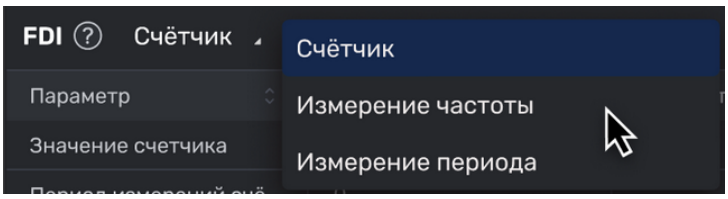


Рисунок 5.6

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

Выполните настройку входов/выходов: у каждого входа/выхода есть параметры, которые располагаются в таблице.

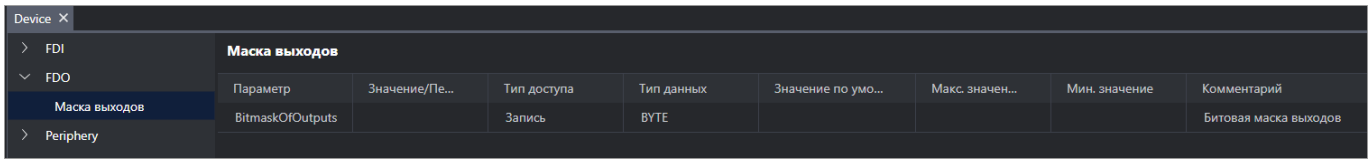


Рисунок 5.7

Параметры, доступные для ввода:

- **Значение/Переменная** — введите значение или привяжите переменную к параметру. При привязке переменной доступно автодополнение;
- **Комментарий** — пользовательский комментарий.

Параметры, заполненные автоматически и недоступные для редактирования:

- **Параметр** — наименование параметра;
- **Тип доступа** — чтение/запись;
- **Тип данных** — тип данных параметра;
- **Значение по умолчанию** — значение по умолчанию;
- **Макс. значение** — максимальное значение параметра;
- **Мин. значение** — минимальное значение параметра.

5.2 Подключение устройства к ПК

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

Подключение к ALTA IDE доступно по интерфейсам:

- USB;
- Ethernet.

ВНИМАНИЕ
Модификация и версия прошивки подключенного прибора должна совпадать с загруженным таргетом. В противном случае соединение с устройством не будет установлено. Эти параметры доступны для просмотра в web-конфигураторе устройства, вкладка Состояние → Обзор.

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 первом подключении устройства необходимо настроить соединение с прибором. Для этого:

1. Выберите:
   - **Главное меню → Онлайн → Настройки соединения**;
   - или нажмите **Настройки соединения** в контекстном меню устройства.

   Откроется окно настроек соединения:

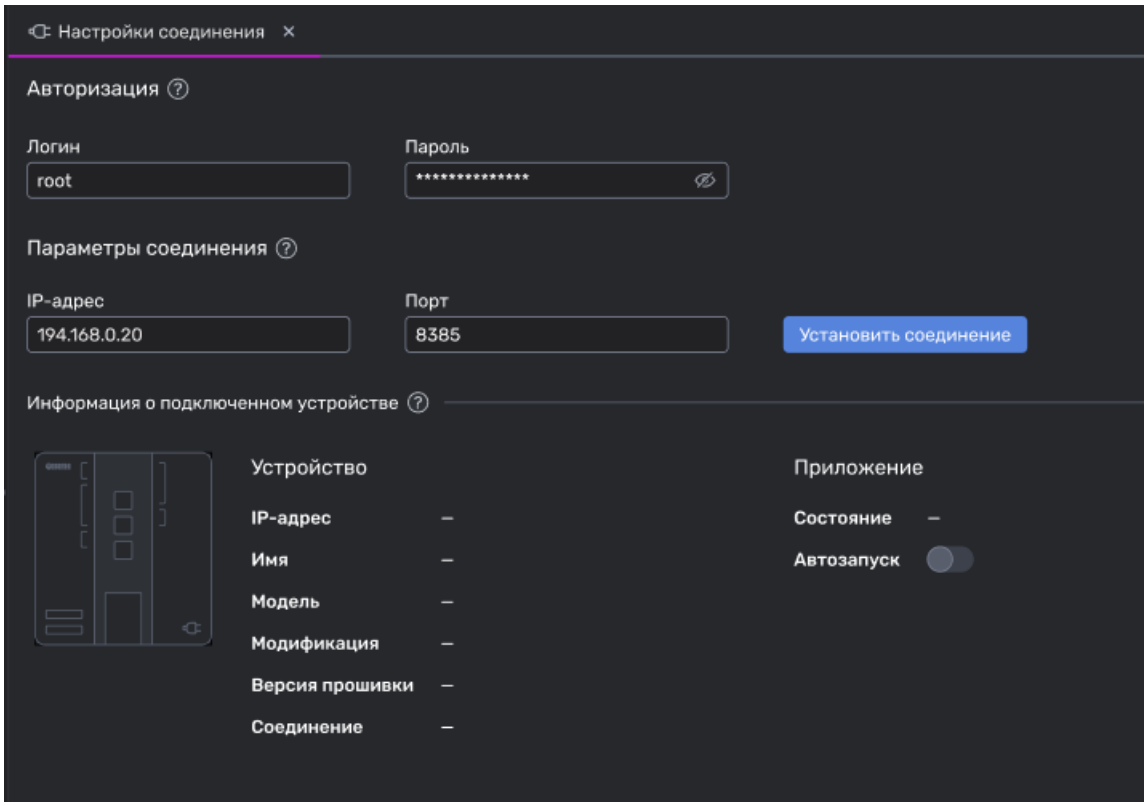


Рисунок 5.8 – Окно Настройки соединения

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

2. Введите данные:
   - Логин;
   - Пароль;
   - IP-адрес;
   - Порт.
3. Нажмите **Установить соединение.**

Если данные введены корректно, то соединение будет установлено, появится всплывающее сообщение об успешной установке соединения, в окне отобразится информация о подключенном устройстве и работе приложения:

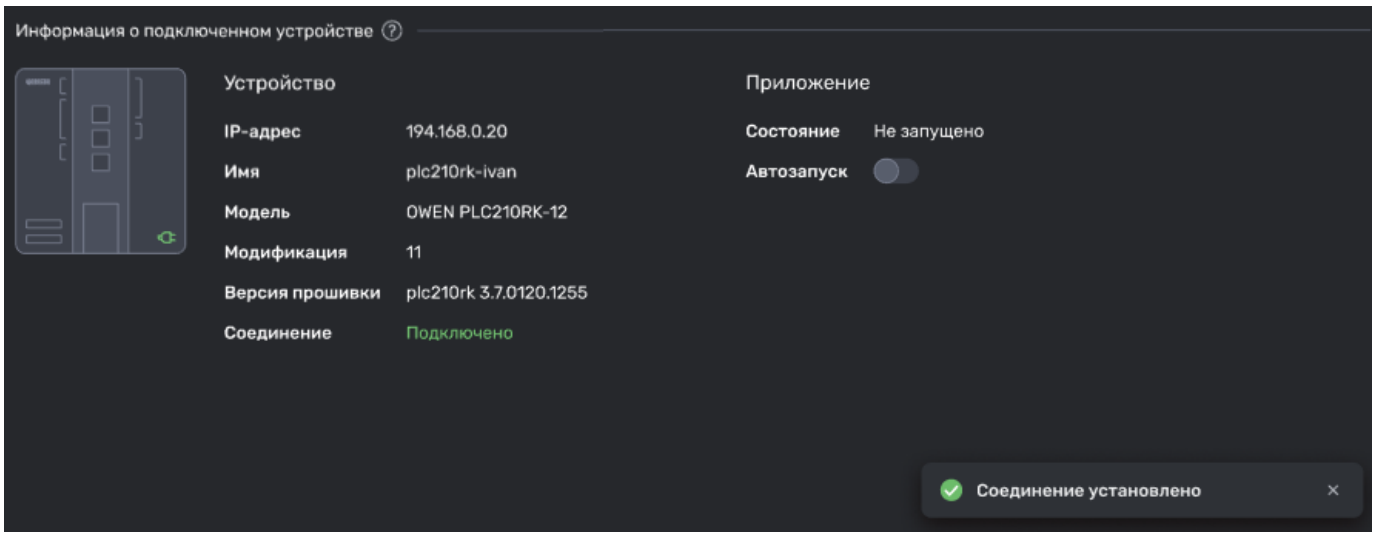


Рисунок 5.9

В противном случае (значение отсутствует, либо введен неверный формат) поле ввода будет выделено красным, при наведении курсора мыши появится подсказка, содержащая информацию об ошибке. В правом нижнем углу рабочей области отобразится уведомление об ошибке:

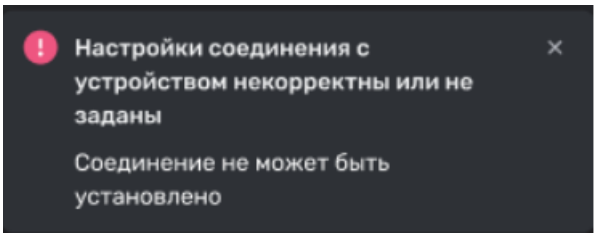


Рисунок 5.10 – Уведомление об ошибке

Включите автозапуск приложения на устройстве, если необходимо автоматически запускать исполняемое приложение при включении устройства (например после перезагрузки). Для этого переведите выключатель **Автозапуск** в положение включено.

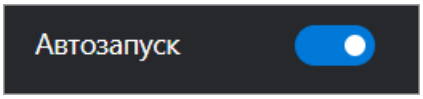


Рисунок 5.11

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

В случае потери соединения с устройством отобразится уведомление:

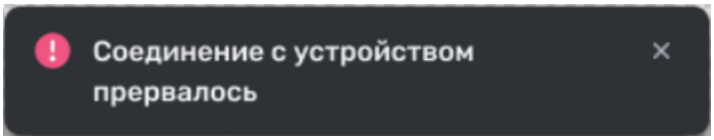


Рисунок 5.12

Включение и отключение соединения с прибором

В проекте с ранее введенными настройками подключения **установить соединение** с подключенным устройством можно одним из следующих способов:

- нажатием кнопки **Установить соединение**  на панели инструментов;
- через контекстное меню элемента **Устройство,** выбрав пункт **Установить соединение**;
- с помощью **Главное меню** → **Онлайн** → **Установить соединение**.

Разорвать соединение с устройством можно с помощью:

- кнопки  расположенной на панели инструментов;
- контекстного меню элемента **Устройство**, выбрав пункт **Разорвать соединение**;

- **Главное меню → Онлайн → Разорвать соединение**;
- кнопки **Разорвать соединение** в окне **Настройки соединения**.

В дереве проекта и строке состояния отображается статус подключенного устройства:

Таблица 5.1 – Отображение статуса устройства в ALTA IDE

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

5.3 Запись приложения в прибор

Для записи приложения в прибор необходимо убедиться, что:

- устройство подключено к ПК;
- соединение установлено.

Если на устройстве запущено выполнение приложения, то выполнение будет остановлено.

Записать приложение можно одним из способов:

- с помощью кнопки  на панели инструментов;
- через контекстное меню элемента **Устройство** в **Дереве проекта**;
- **Главное меню → Онлайн → Загрузить приложение**.

Перед загрузкой автоматически будет выполнена сборка проекта, в случае отсутствия ошибок начнется загрузка приложения в прибор, кнопка перейдет в статус загрузки  . Также процесс загрузки отобразится в строке состояния:

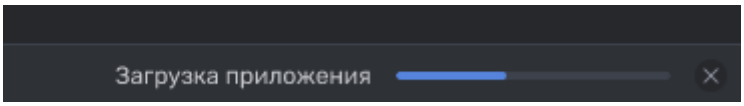


Рисунок 5.13 – Шкала загрузки приложения

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.

Возможные ошибки при загрузке приложения

- потеря связи с устройством;
- файл приложения превышает допустимый размер (максимальный размер исполняемого приложения зависит от доступной памяти в устройстве).

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

Процесс загрузки приложения в прибор можно остановить одним из следующих способов:

- нажмите кнопку  на панели инструментов;
- нажмите на значок  на шкале загрузки приложения, расположенной в строке состояния главного окна ALTA IDE.

5.4 Режим онлайн и отладка программы

Режим онлайн служит для мониторинга работы 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.

5.4.1 Переход в режим онлайн

Для запуска режима онлайн:

1. Запустите выполнение приложения кнопкой  на панели инструментов.
2. Нажмите кнопку перехода в режим онлайн  .

В режиме онлайн иконка  станет зеленого цвета, на панели инструментов появится панель отладки. В дереве проекта отобразятся иконки, обозначающие статусы задач:

	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

Возможные ошибки при переходе в онлайн

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

Рисунок 5.14

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

2. Если к устройству одновременно подключены несколько пользователей, и устройство уже находится в режиме онлайн, то переход выполнен не будет:

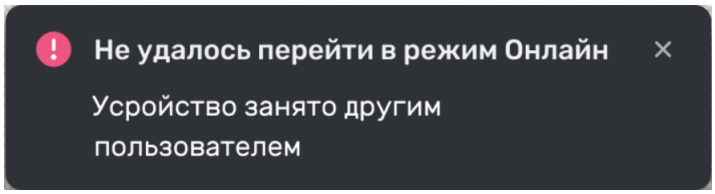


Рисунок 5.15

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

5.4.2 Редактор ST в режиме онлайн. Отладка программы

При переходе **в режим онлайн** интерфейс вкладки редактора ST приобретает ряд отличительных особенностей, а в рабочей области отображаются элементы, доступные только в режиме онлайн:

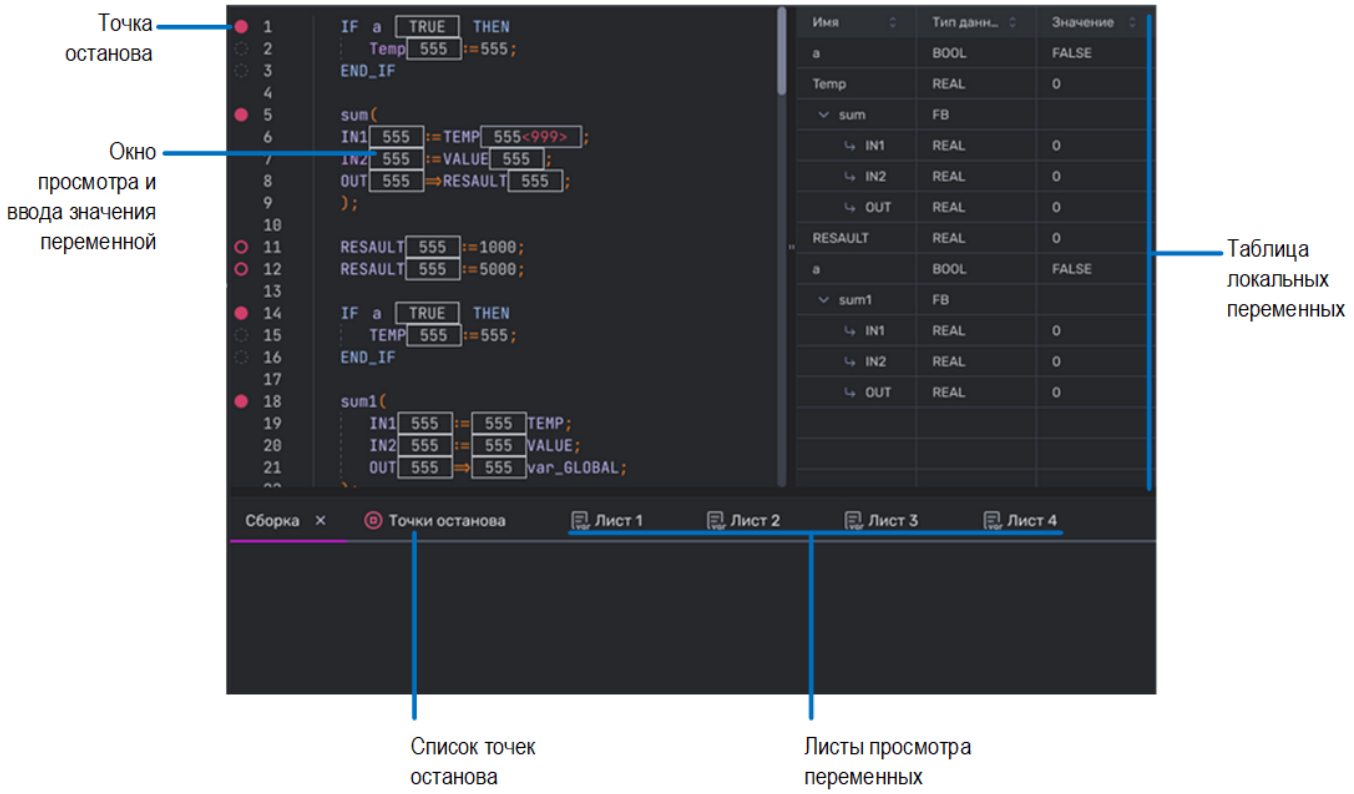


Рисунок 5.16 – Редактор ST в режиме онлайн

Ввод и фиксация значения переменной

1. Для ввода значения переменной дважды нажмите ЛКМ в одном из следующих элементов интерфейса:
   - в коде программы в окне просмотра, предназначенном для ввода значения переменной;
   - в таблице локальных переменных в столбце "Подготовленное значение";
   - в Листе просмотра, расположенном в окне вывода, в столбце "Подготовленное значение".

   Откроется окно ввода значения переменной:

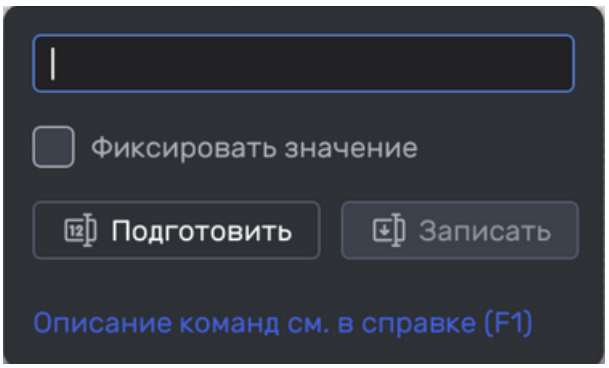


Рисунок 5.17

2. Введите необходимое значение и нажмите:
   - **Подготовить** - переменная будет подготовлена для последующей записи:

Рисунок 5.18 – Отображение подготовленного значения переменной в коде программы

- **Записать** — введенное значение будет записано. Также с помощью кнопки **Записать** можно записать ранее подготовленное значение переменной.
- Установите чекбокс **Фиксировать значение** если необходима фиксация значения переменной. Зафиксированное значение переопределяет текущее значение переменной, переменная сохраняет свое значение на границах цикла: до начала выполнения программы и после.

  Фиксированное значение можно как подготовить так и записать:

  –

Рисунок 5.19 – Отображение зафиксированного подготовленного значения переменной в коде программы

  При последующей записи подготовленное фиксированное значение также запишется с фиксацией;

  –

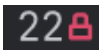


Рисунок 5.20 – Отображение зафиксированного значения в коде программы

Если ввести недопустимое значение переменной строка ввода будет выделена красным, отобразится информационное сообщение об ошибке.

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

Чтобы **очистить подготовленное значение** или **снять фиксацию** переменной воспользуйтесь контекстным меню переменной:

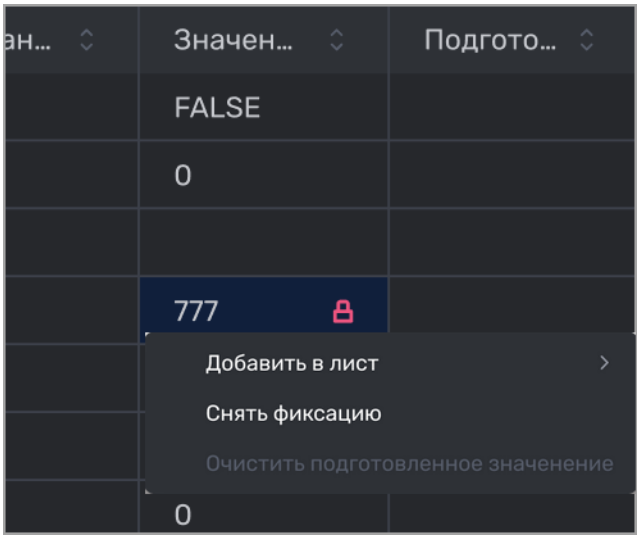


Рисунок 5.21

Записать **все** подготовленные значения переменной можно:

- с помощью кнопки  на панели отладки;
- **Главное меню → Отладка → Записать все значения**.

Снять фиксацию значений у **всех** переменных в проекте:

- с помощью кнопки  на панели отладки;
- **Главное меню → Отладка → Снять фиксацию у всех переменных**.

Точки останова

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

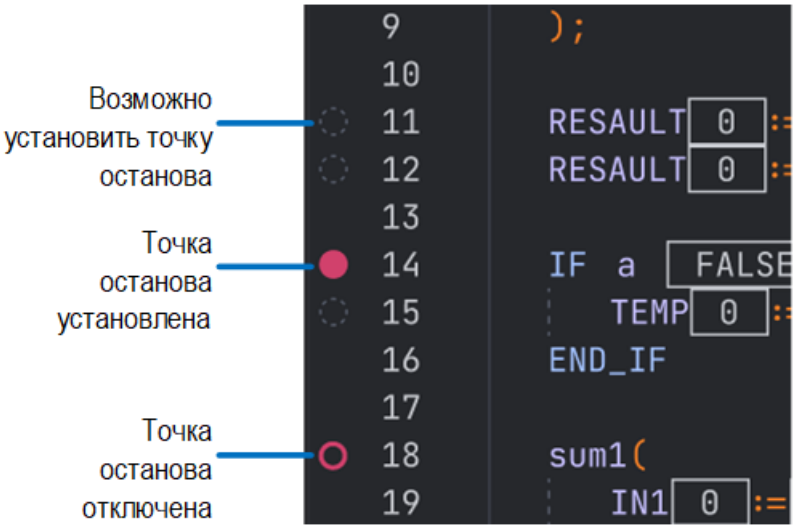


Рисунок 5.22

Установить точку останова можно одним из следующих способов:

- наведите курсор мыши на маркер возможного места установки слева от программного кода и нажмите ЛКМ;
- нажмите кнопку на панели отладки — **все** точки останова в программе будут установлены;
- откройте контекстное меню Точки останова слева от программного кода или в Окне вывода и выберите **Установить точку останова**.

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

Удалить включенные или отключенные точки останова можно одним из следующих способов:

- наведите курсор мыши на точку останова слева от программного кода и нажмите ЛКМ;
- нажмите на кнопку на панели отладки — **все** точки останова в программе будут удалены;
- откройте контекстное меню Точки останова слева от программного кода или в Окне вывода и выберите **Удалить точку останова**.

Отключить установленные точки останова можно одним из следующих способов:

- нажмите на кнопку на панели отладки — **все** точки останова в программе будут отключены;
- откройте контекстное меню Точки останова слева от программного кода или в Окне вывода и выберите **Отключить точку останова**.

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

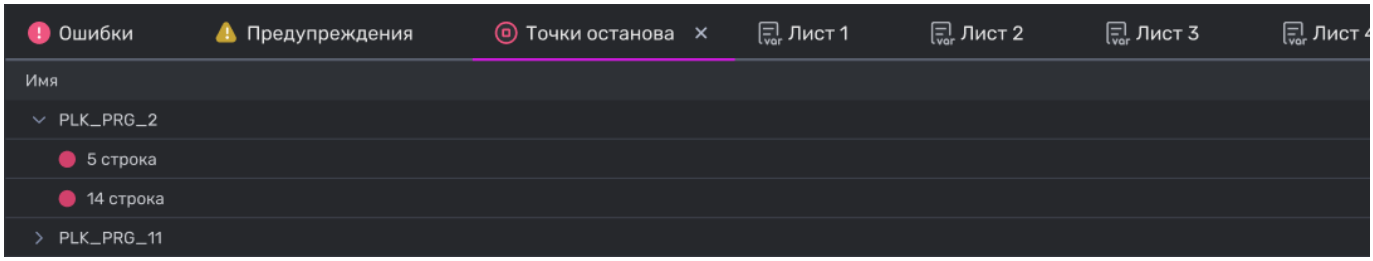


Рисунок 5.23 – Вкладка Точки останова в Окне вывода

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.

1. Чтобы перейти к месту остановки выполнения программного кода:
   - нажмите кнопку  на панели отладки;
   - или **Главное меню** → **Отладка** → **Перейти к месту остановки**.

   Строка на которой было приостановлено выполнение программы будет выделена.

2. 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

5.4.3 Таблица локальных переменных

Переменные проекта отображаются в таблице переменных, которая доступна для просмотра только в **режиме онлайн**:

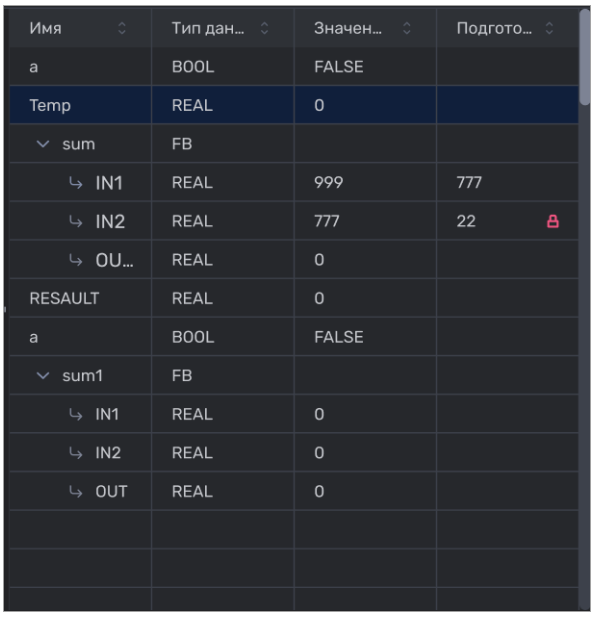


Рисунок 5.24 – Таблица локальных переменных

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 переменной в коде программы;

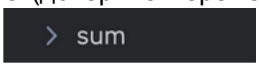
Тип данных — один из типов: BYTE, WORD, DWORD, INT, DINT, REAL, STRING80, STRING21. См. подробнее раздел **Типы переменных**

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.

Чтобы **задать значение переменной** дважды нажмите ЛКМ в столбце подготовленное значение. Откроется окно ввода данных. Подробно работа в окне ввода данных описана в разделе Редактор ST в режиме онлайн. Отладка программы.

Отображение вложенных переменных

Переменные типа **структур** и **массивов** и другие составные типы (родительские переменные), которые содержат вложенные данные (дочерние переменные) , отображаются в таблице переменных с маркером вложенности перед именем: 

- **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.

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

Контекстное меню переменной

Чтобы открыть контекстное меню переменной нажмите ПКМ на любую ячейку в таблице переменных:

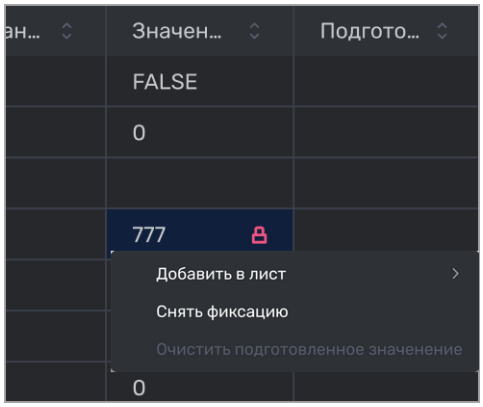


Рисунок 5.25

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

Снять фиксацию - освобождение значения переменной от фиксации.

Очистить подготовленное значение - удаляет подготовленное значение переменной.

5.4.4 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. 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.

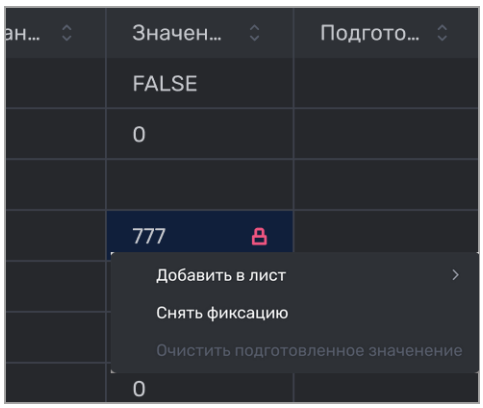


Рисунок 5.26

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

Родительские переменные добавляются в лист просмотра переменных вместе с вложенными элементами (дочерними переменными).

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.

Пример

Для структуры *Device.Status* дочерняя переменная *ErrorCode* будет отображена как *Device.Status.ErrorCode*.

Чтобы **задать значение переменной** дважды нажмите ЛКМ в столбце подготовленное значение. Откроется окно ввода данных. Подробно работа в окне ввода данных описана в разделе Редактор ST в режиме онлайн. Отладка программы.

5.5 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 (самый распространенный RS-485). 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 (чаще всего используется Ethernet).

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 ALTA IDE Slave-устройства не поддерживают широковещательную рассылку.

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 следующие варианты размещения данных в памяти slave-устройств:

- независимые области памяти;
- общая область памяти.

В ALTA IDE поддержан 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 5.2 – 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
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 ниже:

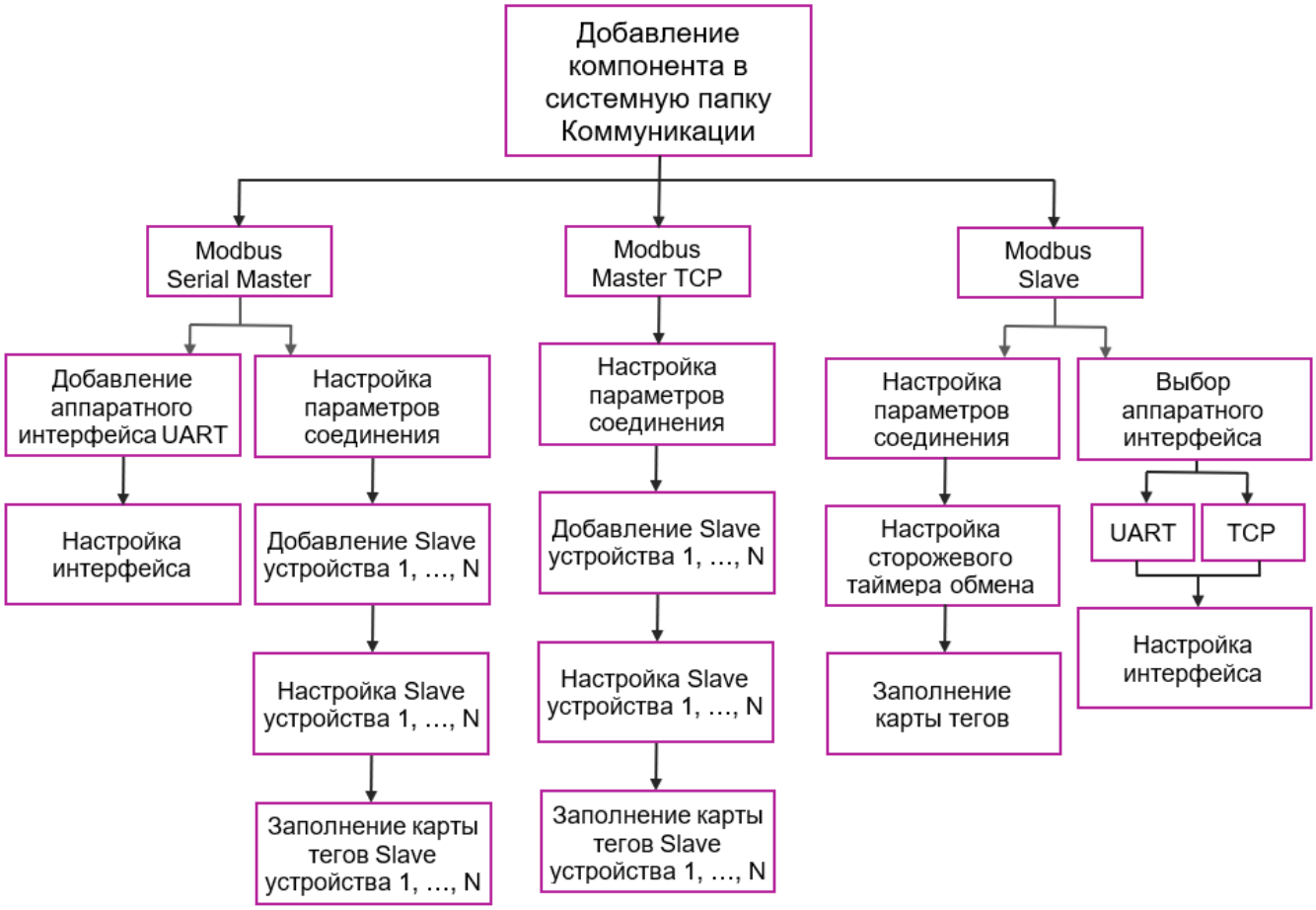


Рисунок 5.27

5.5.1 Modbus Serial/TCP

Ниже приведены общие разделы, описывающие работу с протоколами **Modbus Serial** и **Modbus TCP**.

Работа по протоколам **Modbus Serial** и **Modbus TCP** во многом схожа и отличается лишь незначительными особенностями, поэтому в руководстве они описываются в рамках общих разделов. Дальнейшие инструкции применимы одновременно к **Modbus Serial** и **Modbus TCP**, за исключением случаев, отмеченных как специфичные для одного протокола.

5.5.1.1 Modbus Serial/TCP Master

Для добавления устройства в режиме **Master**:

1. Выберите **Добавить Modbus Master TCP/Добавить Modbus Serial Master** в контекстном меню системной папки **Коммуникации** в дереве проекта:

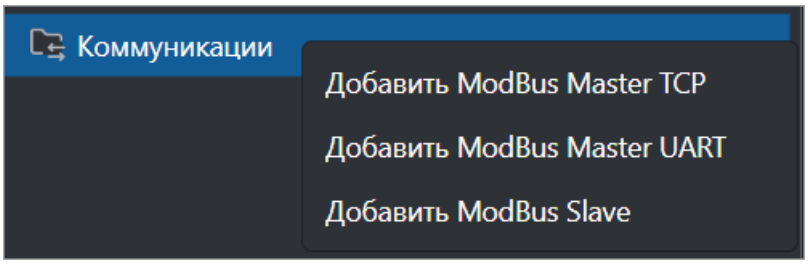


Рисунок 5.28 – Добавление Modbus Master

Добавленное устройство по умолчанию имеет имя modbusmaster*_1 (имя зависит от выбранного протокола). Каждому последующему устройству будет присваиваться следующий порядковый номер. **Переименование** доступно в момент создания, либо через контекстное меню компонента. При задании устройству имени следует учитывать правила именования языка ST, и убедиться, что

выбранное имя не дублирует название других компонентов в дереве проекта - создание компонентов с одинаковыми именами не допускается.

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

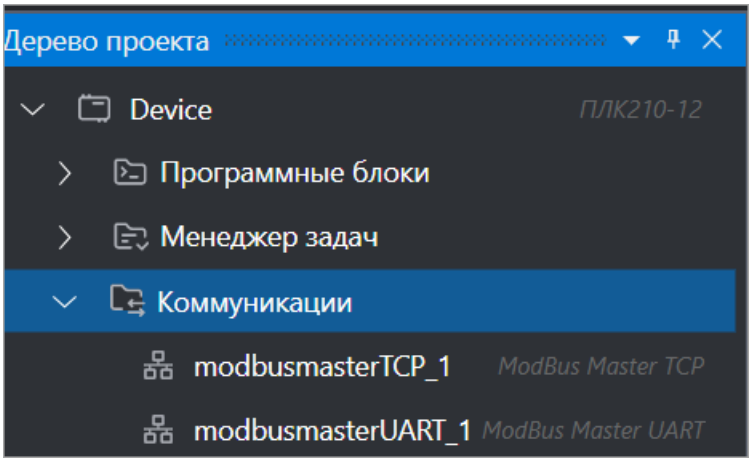


Рисунок 5.29 – Modbus Master в дереве проекта

2. Откройте вкладку редактора Modbus Master:
   - двойным кликом ЛКМ на элемент Modbus Master в дереве проекта;
   - выберите пункт **Открыть** в контекстном меню элемента Modbus Master в дереве проекта.
3. Настройте параметры соединения во вкладке редактора Modbus Master:

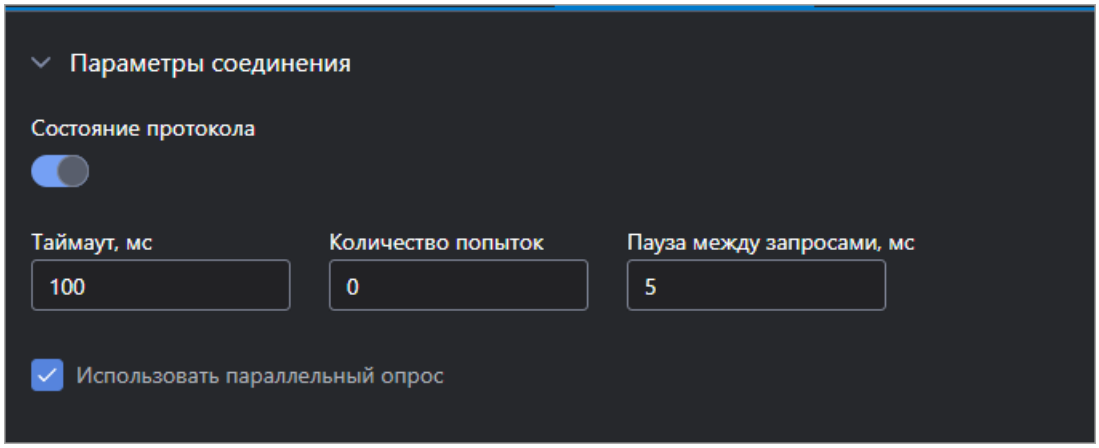


Рисунок 5.30 – Настройка параметров соединения

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 Slave Device 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 Slave Device не введены индивидуальные значения данного параметра;

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

Использовать параллельный опрос (только для **Modbus Master TCP**) — возможность выполнять параллельный опрос нескольких slave-устройств для повышения скорости обмена.

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

Список устройств

IP-Адрес	Порт	Slave ID	Имя	Опрос устройства	Комментарий
192.168.1.1	502	1	modbusdeviceTCP_1	Включен	

Рисунок 5.31 – Таблица список устройств Modbus Master

IP-Адрес (только для **Modbus Master TCP**) — сетевой адрес устройства, диапазон значений от 0.0.0.0 до 255.255.255.255;

Порт (только для **Modbus Master TCP**) — 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.

Для сортировки данных нажмите на значок  в заголовке столбца.

Редактирование параметров в таблице невозможно. Чтобы внести изменения, перейдите во вкладку соответствующего **Modbus Slave Device** и задайте необходимые значения. Данные в таблице **Список устройств** обновятся автоматически.

Добавление интерфейса подключения UART для Modbus Serial Master

Для устройства **Modbus Serial Master** необходимо выбрать и добавить аппаратный интерфейс.

ВНИМАНИЕ
Нельзя добавлять один и тот же интерфейс UART одновременно к компоненту Serial Master и Slave. Такое назначение приведет к отсутствию обмена по сети у одного из устройств.

1. **Добавьте интерфейс** с помощью контекстного меню компонента **Modbus Serial Master** в дереве проекта:

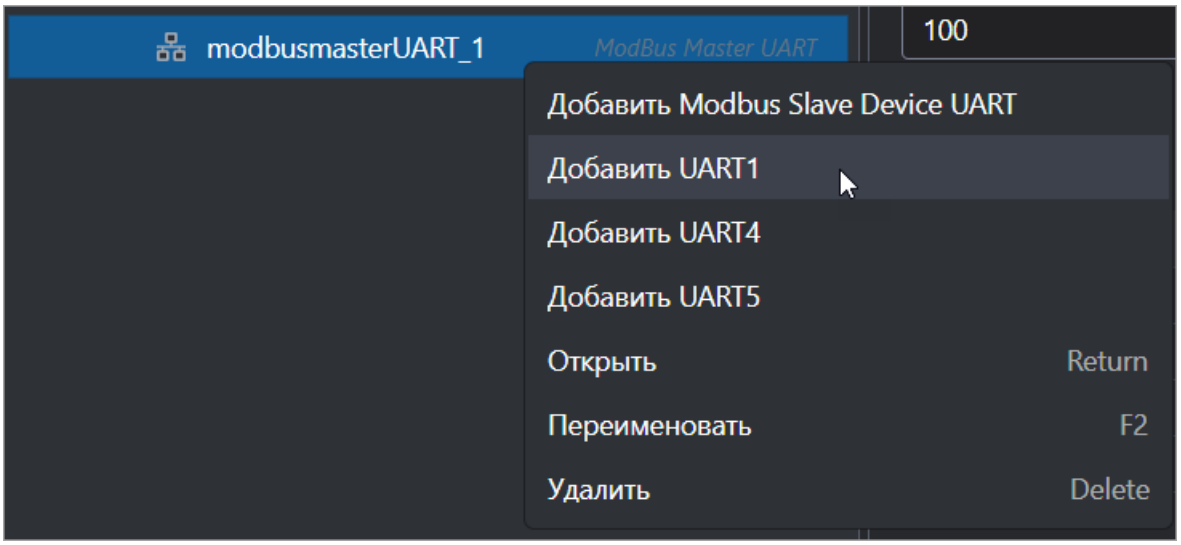


Рисунок 5.32 – Добавление интерфейса UART

После добавления интерфейс отобразится в дереве проекта, как ответвление компонента **Modbus Serial Master**:

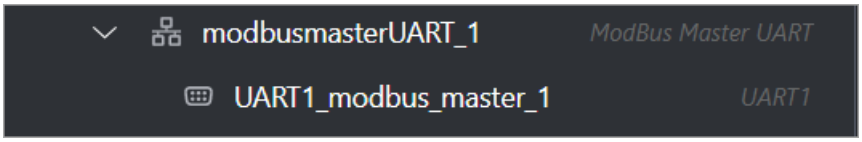


Рисунок 5.33 – Интерфейс UART в дереве проекта

2. **Откройте** вкладку редактора UART:
   - двойным кликом ЛКМ на элемент UART в дереве проекта;
   - выберите пункт **Открыть** в контекстном меню элемента UART в дереве проекта.
3. **Настройте** интерфейс во вкладке редактора UART:

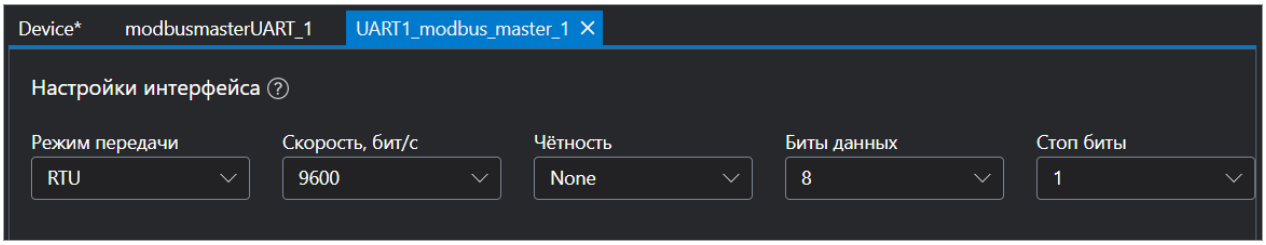


Рисунок 5.34 – Настройки интерфейса UART

Режим передачи — выберите из выпадающего списка режим передачи данных RTU/ASCII;

Скорость, бит/с — выберите из выпадающего списка скорость, с которой будет осуществляться передача данных.

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 — выберите из выпадающего списка количество стоповых бит в посылке Modbus. Возможные значения: 1, 1.5 или 2.

К устройству **Modbus Serial Master** может быть добавлен только один интерфейс UART. В случае попытки добавить еще один интерфейс появится всплывающее окно с информацией об ошибке. Для замены интерфейса удалите установленный UART и добавьте новый.

Удалить и **Переименовать** интерфейс возможно с помощью контекстного меню элемента UART в дереве проекта.

5.5.1.2 Modbus Slave Device Serial/TCP

Для добавления slave-устройства к master-устройству:

1. Откройте контекстное меню master-устройства в дереве проекта и выберите **Добавить Modbus Slave Device TCP/Добавить Modbus Serial Slave Device**:

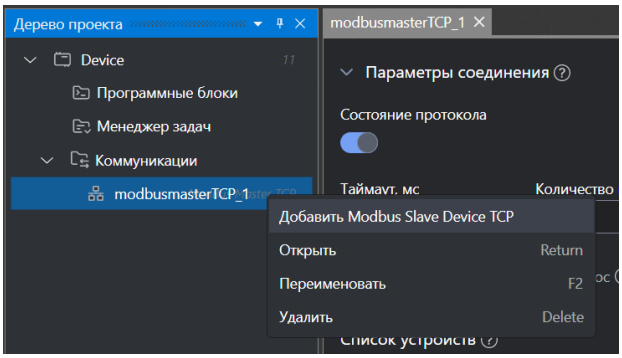


Рисунок 5.35 – Добавление Modbus Slave Device TCP

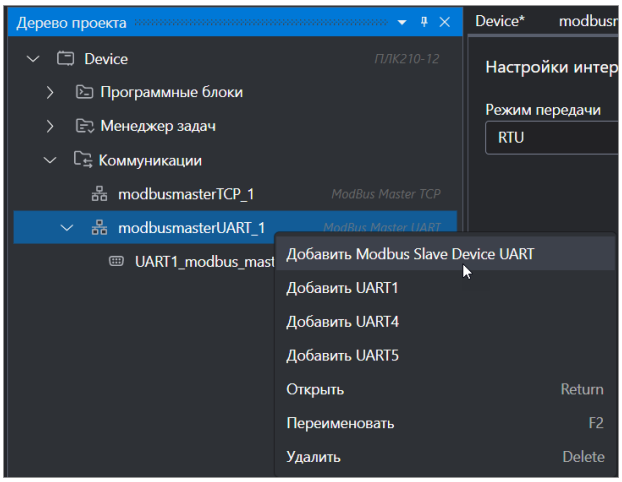


Рисунок 5.36 – Добавление Modbus Serial Slave Device

Добавленное устройство по умолчанию имеет имя modbusdevice*_1 (имя зависит от выбранного протокола). Каждому последующему устройству будет присваиваться следующий порядковый номер.

Переименование доступно в момент создания, либо через контекстное меню компонента. При задании устройству имени следует учитывать правила именования языка ST, и убедиться, что выбранное имя не дублирует название других компонентов в дереве проекта - создание компонентов с одинаковыми именами не допускается.

После добавления slave-устройство отобразится в дереве проекта, как ответвление компонента Modbus Master:

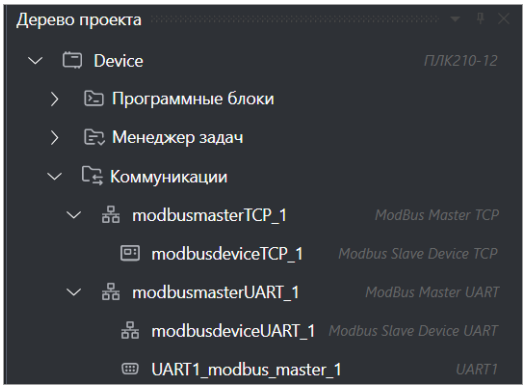


Рисунок 5.37 – Modbus Device в дереве проекта

2. Откройте вкладку редактора Modbus Slave Device:
   - двойным кликом ЛКМ на элемент Modbus Slave Device в дереве проекта;
   - выберите пункт **Открыть** в контекстном меню элемента Modbus Slave Device в дереве проекта.
3. Настройте параметры соединения и параметры обмена во вкладке редактора Modbus Slave Device:

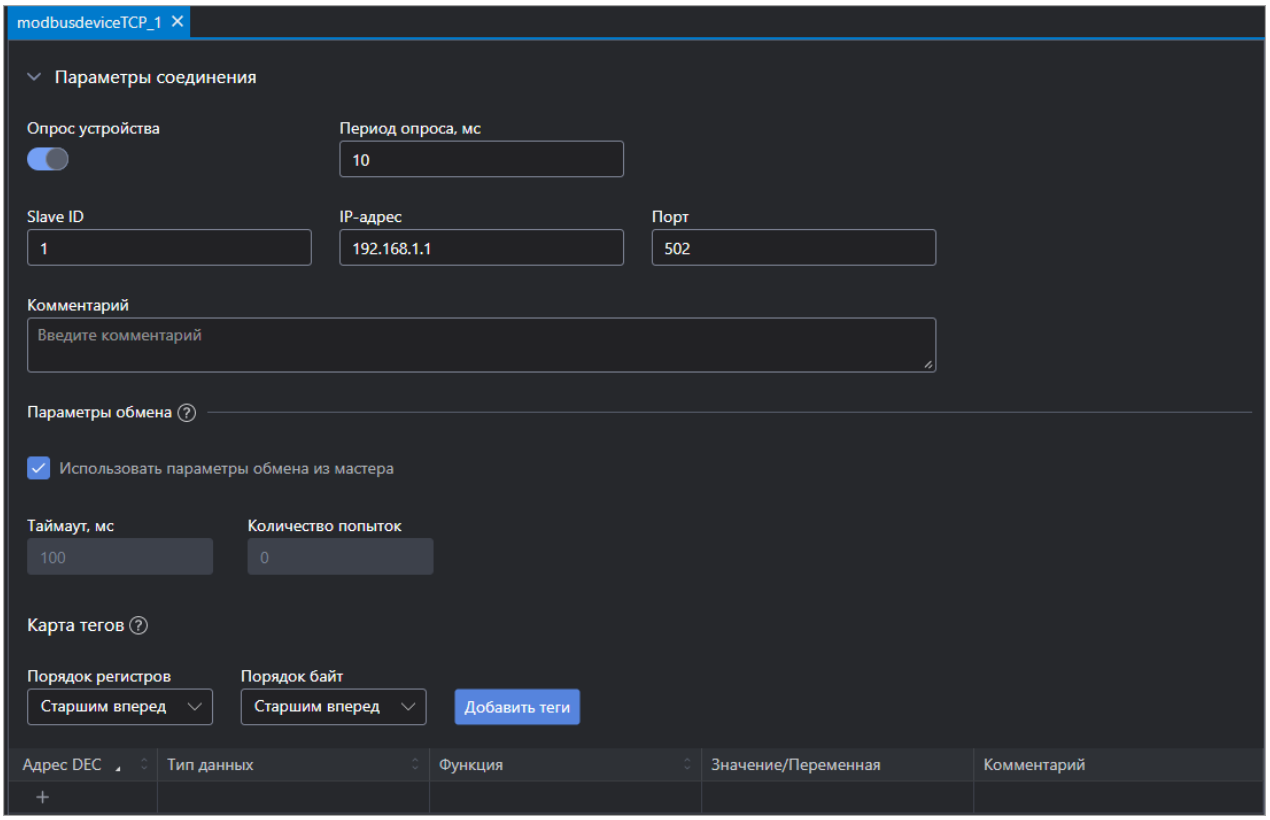


Рисунок 5.38 – Вкладка редактора Modbus Slave Device TCP

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 (только для **Modbus Slave Device TCP**) — сетевой адрес устройства, диапазон значений от 0.0.0.0 до 255.255.255.255;

Порт (только для **Modbus Slave Device TCP**) — порт, используемый для обмена, диапазон значений от 1 до 65535 (по умолчанию - 502);

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

Параметры обмена:

Использовать параметры обмена из мастера — в случае установки чекбокса внесение данных будет недоступно, будут использованы значения, введенные во вкладке редактора Modbus 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.

4. Заполните карту тегов.

   Карта тегов сопоставляет переменные с регистрами Modbus (адресами). Один тег может включать несколько регистров.

   Настройте общие для всей карты тегов параметры:

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

   Добавить теги можно одним из способов:

   a. Нажмите кнопку **Добавить теги**. Откроется окно создания тегов:

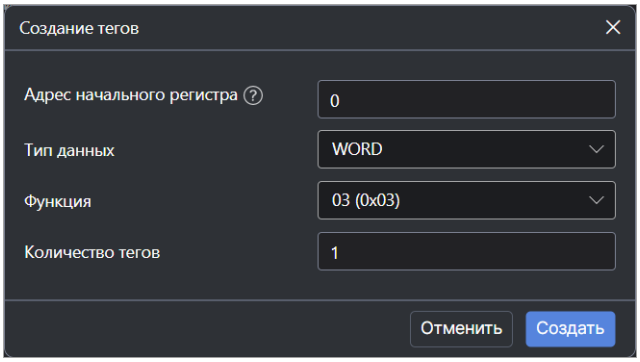


Рисунок 5.39

- **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 приведен в разделе Modbus;
- **Количество тегов** — количество тегов, которое требуется создать.

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

   b. Нажмите кнопку "+" расположенную в столбце **Адрес** таблицы тегов.

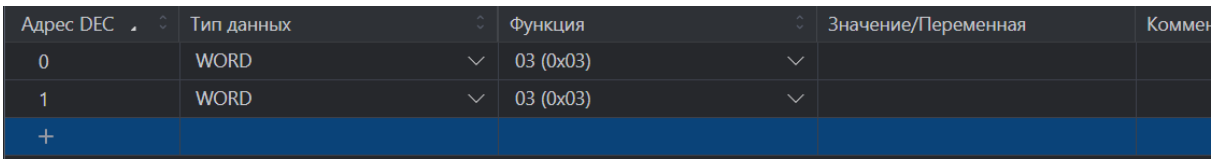
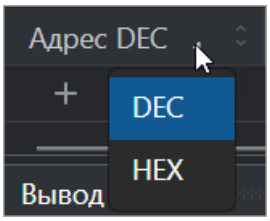
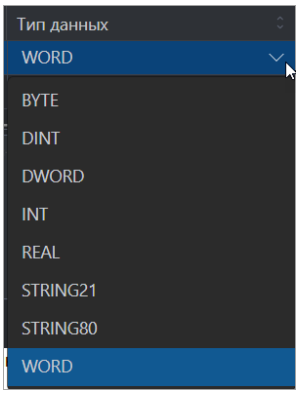
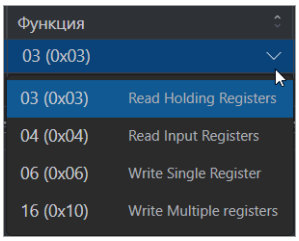
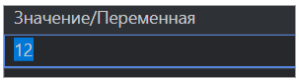
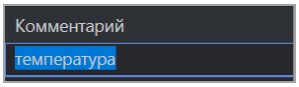


Рисунок 5.40

В этом случае теги добавляются по одному, начиная с адреса, следующего за адресом последнего отредактированного тега.

5. При необходимости **отредактируйте** значения в таблице тегов

Адрес		В поле адрес указывается диапазон регистров, которые занимает созданный тег. **Для переключения формата ввода (DEC/HEX)** — нажмите на заголовок столбца Адрес и выберите нужное значение из выпадающего списка. **Для редактирования значения** — дважды нажмите ЛКМ по ячейке и введите новое значение
Тип данных		Кликните ЛКМ по ячейке и выберите из выпадающего списка нужное значение
Функция		Кликните ЛКМ по ячейке и выберите из выпадающего списка нужное значение Список функций, поддерживаемых в ALTA IDE приведен в разделе Modbus
Значение/ Переменная		Дважды нажмите ЛКМ по ячейке и введите новое значение или привяжите переменную: • привязка переменной-входа для чтения регистра в программе; • привязка переменной-выхода для записи регистра из программы; • ввод значения (константы) для записи регистра
Комментарий		Дважды нажмите ЛКМ по ячейке и введите пользовательский комментарий

Для сортировки данных нажмите на значок  в заголовке столбца.

6. **Удалить** тег можно с помощью контекстного меню тега в карте тегов:

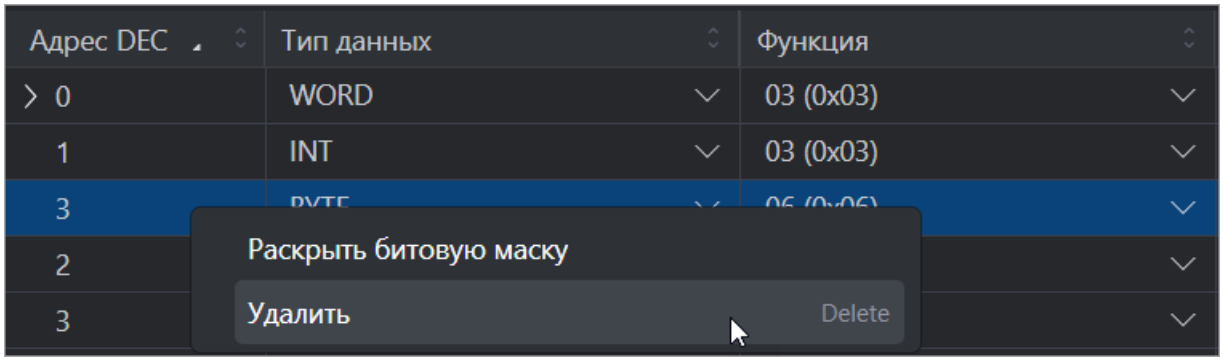


Рисунок 5.41

ПРИМЕЧАНИЕ
Созданные теги будут опрашиваться в порядке добавления в карту тегов, 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 Битовая маска.

5.5.2 Режим Slave

Добавить устройства в режиме Slave можно с помощью контекстного меню системной папки **Коммуникации** в дереве проекта:

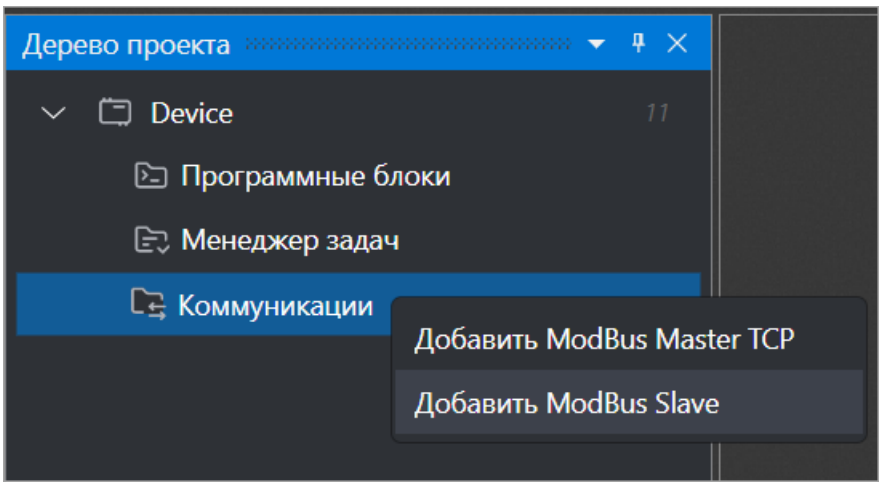


Рисунок 5.42 – Контекстное меню системной папки Коммуникации

Добавленное устройство по умолчанию имеет имя modbusslaveCE_1. Каждому последующему устройству будет присваиваться следующий порядковый номер. Переименование доступно в момент создания, либо через контекстное меню устройства. При задании компоненту имени следует учитывать правила именования языка ST, и убедиться, что выбранное имя не дублирует название других компонентов в дереве проекта - создание компонентов с одинаковыми именами не допускается.

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

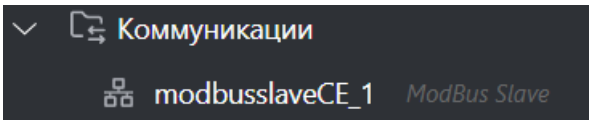


Рисунок 5.43 – Устройство Modbus Slave в дереве проекта

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

5.5.2.1 Настройка параметров Modbus Slave

1. Откройте вкладку редактора Modbus Slave:
   - двойным кликом ЛКМ на элемент ModbusSlave в дереве проекта;
   - выберите пункт **Открыть** в контекстном меню элемента Modbus Slave в дереве проекта.
2. Введите настройки:

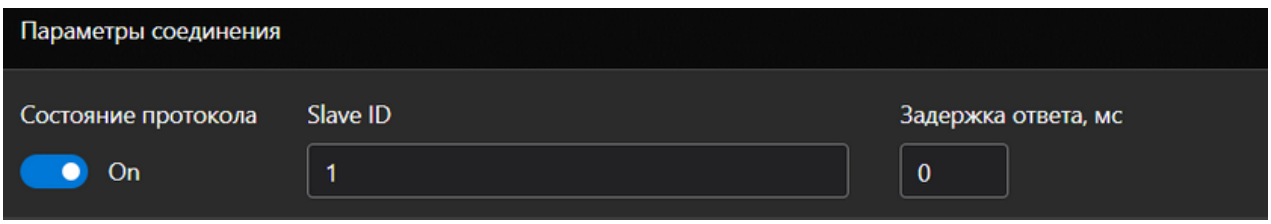


Рисунок 5.44

Настройка параметров соединения

- **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. В случае отсутствия запроса от master-устройства в течение указанного времени генерируется ошибка, которая отобразится в дереве проекта пиктограммой  .

Рисунок 5.45

- **Состояние таймера** — включение/выключение сторожевого таймера обмена.
- **Таймаут, мс** — время ожидания запроса от master-устройства.

3. Заполните карту тегов.

5.5.2.2 Карта тегов

Тег — это переменная в проекте, которая привязывается к одному или нескольким **регистрам Modbus** (ячейкам памяти с конкретным адресом) и имеет заданный тип данных, правила упаковки (порядок слов/байт) и назначение (чтение/запись).

Карта тегов сопоставляет переменные с регистрами Modbus (адресами).

Правила выравнивания

В ALTA IDE происходит автоматическое выравнивание переменных, в зависимости от их типа, при размещении в области памяти устройства, согласно правилам приведенным в таблице:

Таблица 5.3 – Правила размещения переменных в карте тегов

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	41	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 области 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.

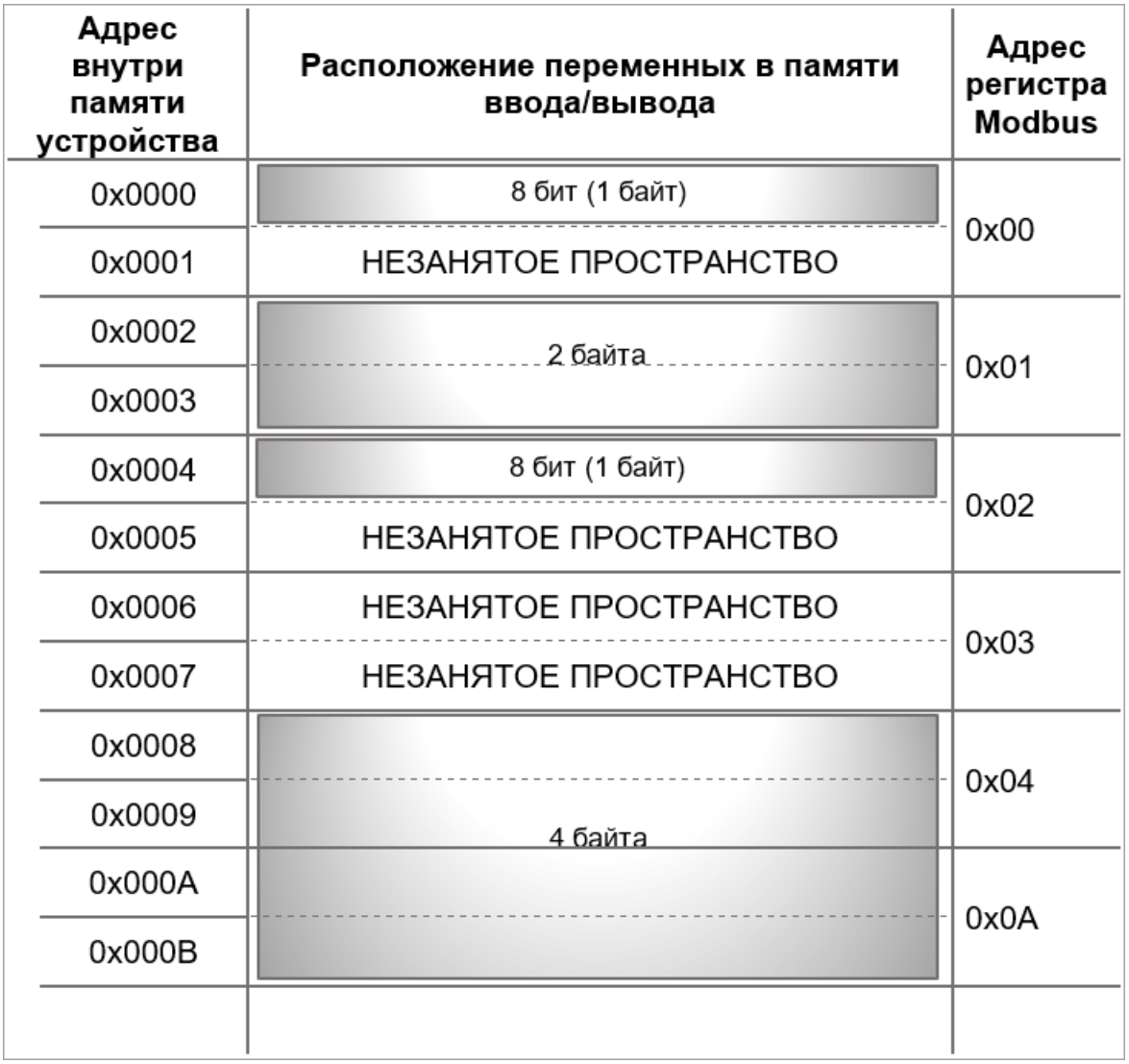


Рисунок 5.46 – Пример расположения переменных в карте тегов с учетом правила выравнивания

Заполнение карты тегов

Заполнение карты тегов slave-устройства включает в себя определение назначения тега, присвоение ему адреса и заполнение данными, которые будут доступны для чтения, записи или чтения/записи master-устройством.

1. Заполните общие для всей карты тегов данные:
   - **Начальный адрес** — задает начальный адрес регистра для всей карты. Доступен ввод в десятеричном и шестнадцатеричном формате (с префиксом 16# и 0x). В случае ввода начального адреса после заполнения карты тегов, 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 регистров** — настройка порядка регистров для передачи master-устройству.
   - **Порядок байт** — настройка порядка байт для передачи master-устройству.
2. Нажмите кнопку **Добавить теги**, откроется окно создания тегов:

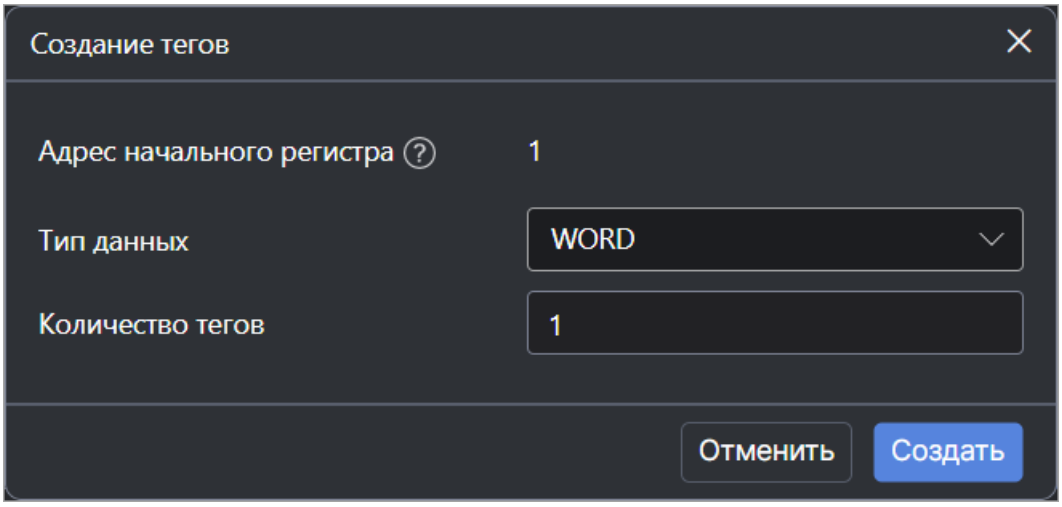


Рисунок 5.47

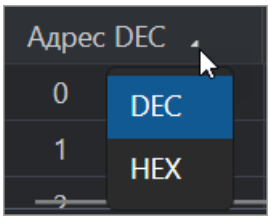
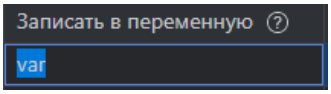
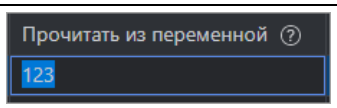
- **Адрес начального регистра** — не доступен для ввода. Автоматически рассчитывается с учетом ранее добавленных тегов и правила выравнивания. В случае создания первого тега - соответствует адресу, введенному в поле Начальный адрес (по умолчанию 0).
- **Тип данных** — тип данных тега.
- **Количество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

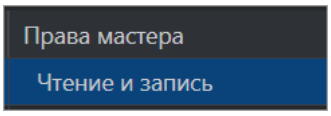
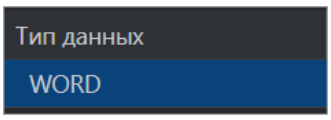
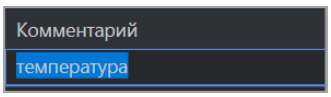
3. Нажмите кнопку **Создать**. В таблице отобразится информация, согласно введенным данным. Созданные теги располагаются в следующих доступных для размещения регистрах с учетом правил выравнивания.

Адрес DEC	Записать в переменную	Прочитать из переменной	Права мастера	Тип данных	Комментарий
0			Чтение и запись	WORD	
1					
2..3			Чтение и запись	REAL	
4..14			Чтение и запись	STRING21	
15..25			Чтение и запись	STRING21	
26..36			Чтение и запись	STRING21	
37					

Рисунок 5.48 – Пример заполнения карты тегов

4. **Отредактируйте** значения в таблице тегов:

Адрес		В поле адрес указывается диапазон регистров, которые занимает созданный тег. **Для переключения формата ввода (DEC/HEX)** - нажмите на заголовок столбца **Адрес** и выберите нужное значение из выпадающего списка. **Редактирование** адреса недоступно
Записать в переменную		Дважды нажмите ЛКМ по ячейке и привяжите переменную для записи тега. Синхронизация переменной и тега происходит при синхронизации входов задачи
Значение/ Переменная		Дважды нажмите ЛКМ по ячейке и введите новое значение или привяжите переменную: • привязка переменной для чтения тега; • ввод значения (константы). Синхронизация переменной/значения и тега происходит при синхронизации выходов задачи

Права мастера		Уровень прав доступа, который предоставляется master-устройству для работы с тегом. Нередактируемое поле Список функций, поддерживаемых в ALTA IDE приведенModbus
Тип данных		Нередактируемое поле
Комментарий		Дважды нажмите ЛКМ по ячейке и введите пользовательский комментарий

5. **Добавить теги** на определенный адрес можно с помощью контекстного меню строки в таблице тегов:

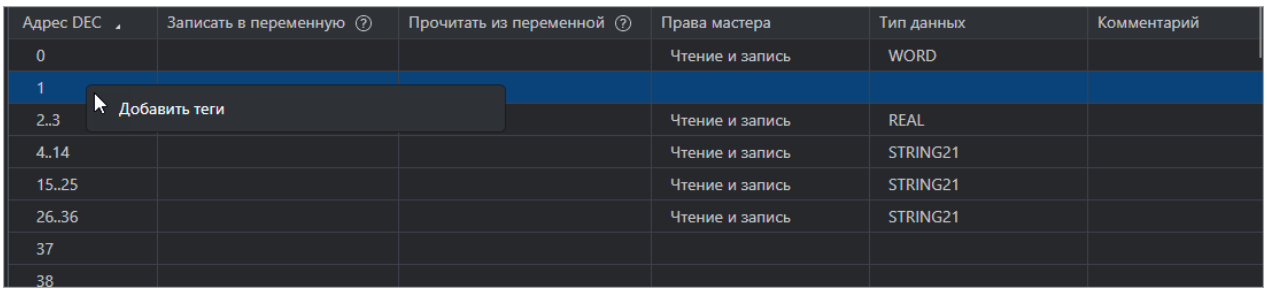


Рисунок 5.49

Если на выбранном адресе невозможно создать тег, то в окне создания тега появится соответствующее предупреждение:

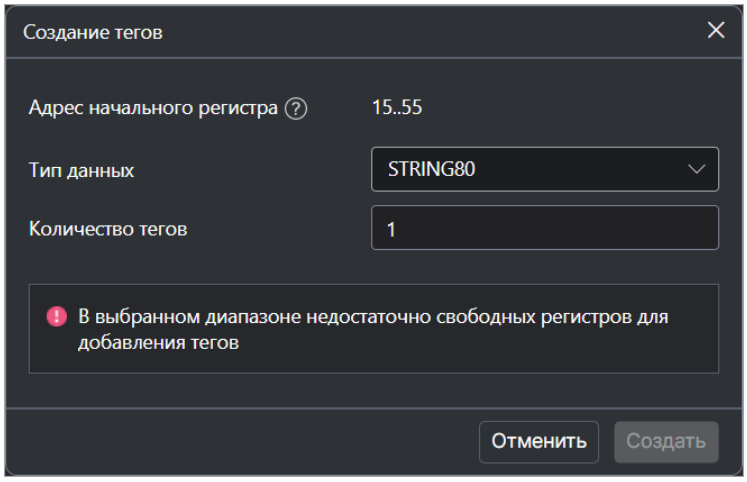


Рисунок 5.50

6. **Удалить** тег можно с помощью контекстного меню тега в карте тегов:

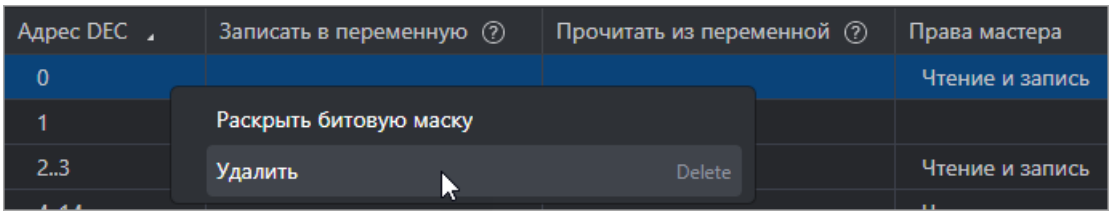


Рисунок 5.51

Для переменных типа BYTE, WORD, DWORD и LWORD доступен просмотр и редактирование битовой маски. Подробно работа с битовой маской описана в разделе Битовая маска.

5.5.2.3 Битовая маска

Для переменных типа BYTE, WORD, DWORD и LWORD доступен просмотр и редактирование битовой маски.

1. Чтобы **раскрыть битовую маску** нажмите ПКМ на строку с регистром в карте тегов и выберите в контекстном меню **Раскрыть битовую маску**. Битовая маска тега отобразится в карте тегов:

Адрес DEC	Переменная	Значение/Переменная	Права мастера	Тип данных	Комментарий
20			Чтение и запись	BYTE	
Бит 0			Чтение и запись	BOOL	
Бит 1			Чтение и запись	BOOL	
Бит 2			Чтение и запись	BOOL	
Бит 3			Чтение и запись	BOOL	
Бит 4			Чтение и запись	BOOL	
Бит 5			Чтение и запись	BOOL	
Бит 6			Чтение и запись	BOOL	
Бит 7			Чтение и запись	BOOL	

Рисунок 5.52

2. **Отредактируйте** данные аналогично редактированию значений в таблице тегов для Modbus Slave Device Serial/TCP и Modbus Slave

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

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:

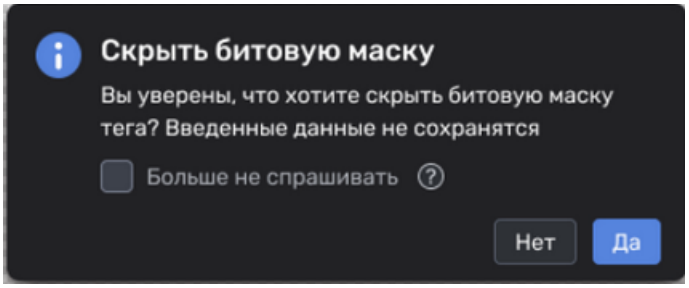


Рисунок 5.53

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

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

После сохранения проекта в ALTA IDE автоматически формируется файл с картой тегов 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 5.54

5.5.2.4 Добавление интерфейса подключения UART/TCP

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

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

Добавьте интерфейс с помощью контекстного меню компонента ModbusSlave в дереве проекта:

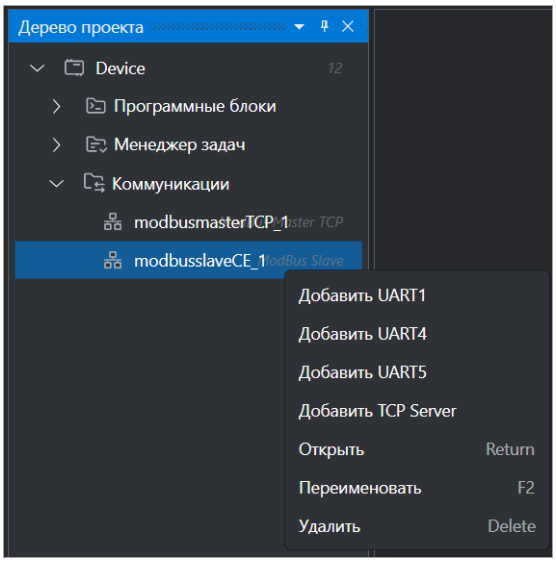


Рисунок 5.55

ВНИМАНИЕ
Нельзя добавлять один и тот же интерфейс UART одновременно к компоненту Serial Master и Slave. Такое назначение приведет к отсутствию обмена по сети у одного из устройств.

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

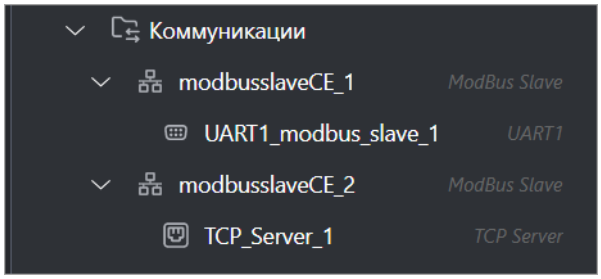


Рисунок 5.56

UART

Выполните настройку интерфейса:

1. **Откройте** вкладку редактора UART:
   - двойным кликом ЛКМ на элемент UART_modbus_slave в дереве проекта;
   - выберите пункт **Открыть** в контекстном меню элемента UART_modbus_slave в дереве проекта.
2. **Настройте** интерфейс во вкладке редактора UART:

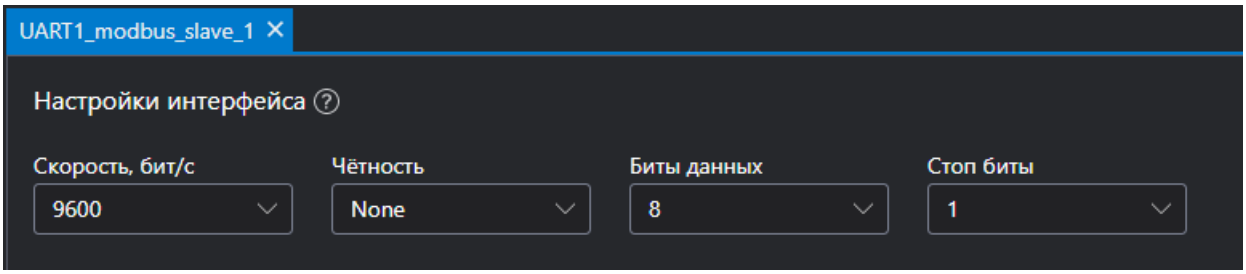


Рисунок 5.57 – Вкладка интерфейса UART

- **Скорость, бит/с** — выберите из выпадающего списка скорость, с которой будет осуществляться передача данных.
- **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:

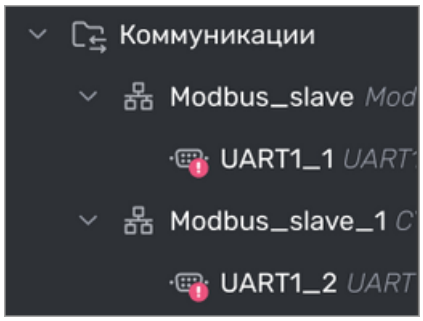


Рисунок 5.58 – Повторное добавление интерфейса

В этом случае один из интерфейсов следует удалить.

Удалить и **Переименовать** интерфейс возможно с помощью контекстного меню элемента UART в дереве проекта.

TCP

Выполните настройку интерфейса:

1. Откройте вкладку редактора TCP:
   - двойным кликом ЛКМ на элемент TCP_Server в дереве проекта;
   - выберите пункт **Открыть** в контекстном меню элемента TCP_Server в дереве проекта.
2. **Настройте** интерфейс во вкладке редактора TCP:

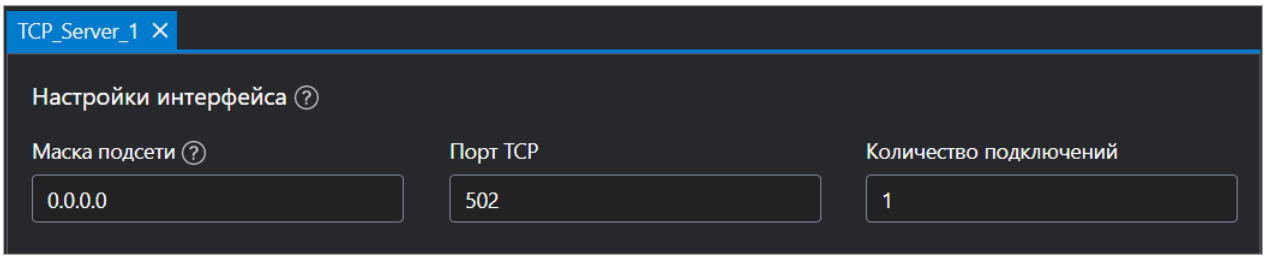


Рисунок 5.59

- **Маска подсети** — настройка подсети, в которой идет обмен. Адрес **0.0.0.0** означает, что сервер принимает подключения на всех сетевых интерфейсах (со всех доступных адресов).
- **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.

В случае если значение отсутствует, либо введен неверный формат поле ввода будет выделено красным. При наведении курсора мыши появится подсказка, содержащая информацию об ошибке.

Удалить и **Переименовать** интерфейс возможно с помощью контекстного меню элемента TCP в дереве проекта.

6 Расширения

Расширения используются для наиболее эффективного создания проектов или их интеграции с другими сервисами компании "Овен Цифровые решения".

Расширения в ALTA IDE:

- Менеджер библиотек 6.1.

ПРИМЕЧАНИЕ
Список расширений постоянно дополняется

6.1 Менеджер библиотек

Компонент **Менеджер библиотек** предназначен для добавления, удаления и управления библиотеками проекта. Помогает поддерживать актуальный состав библиотек, контролировать их подключение к проекту, а также просматривать содержимое библиотек.

Чтобы открыть окно **Менеджер библиотек** дважды нажмите ЛКМ на системную папку **Менеджер библиотек** в дереве проекта, или выберите в контекстном меню пункт **Открыть**:

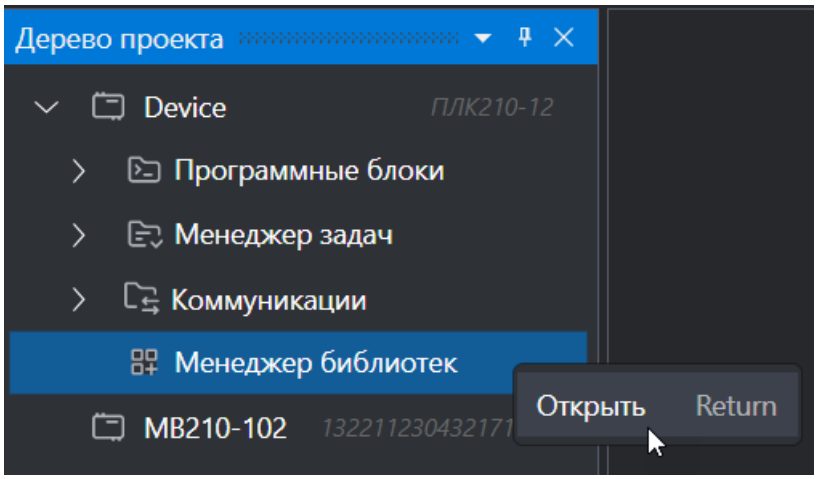


Рисунок 6.1 – Менеджер библиотек в дереве проекта

Окно **Менеджер библиотек** содержит две вкладки:

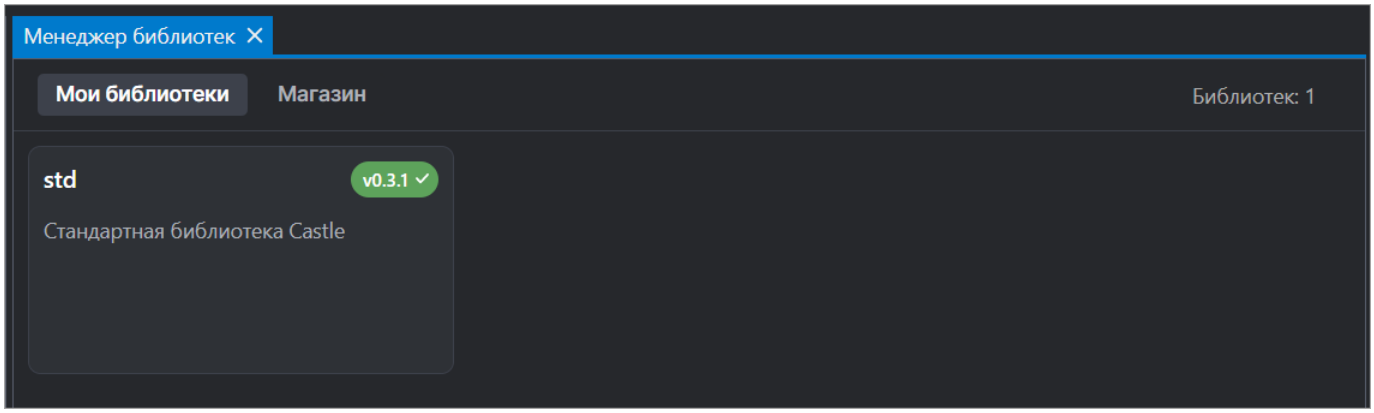


Рисунок 6.2 – Окно Менеджер библиотек

1. **Мои библиотеки** — содержит информацию о библиотеках, подключенных к проекту.

   ПРИМЕЧАНИЕ
   В ALTA IDE по умолчанию присутствует обязательная библиотека **Standard** . Она подключается автоматически и не может быть удалена или скрыта из проекта.

2. **Магазин** — предназначена для просмотра каталога библиотек с возможностью их подключения к текущему проекту.

Информация о библиотеке

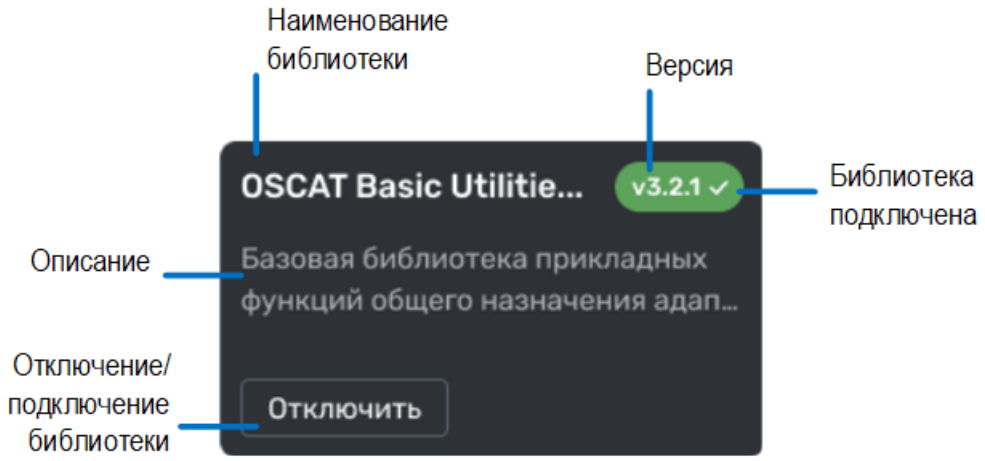


Рисунок 6.3 – Карточка библиотеки

Для просмотра информации о библиотеке нажмите ЛКМ на карточку библиотеки, расположенную в менеджере библиотек. Карточка содержит вкладки:

- **README** — краткое описание содержания библиотеки;
- **Информация** — содержание данной вкладки зависит от конкретной библиотеки. Вкладка может содержать информацию об авторах, лицензии, сайте и репозитории библиотеки, а также описание библиотеки и ссылку на документацию библиотеки.

Подключение библиотеки

Чтобы подключить библиотеку к проекту:

1. Откройте вкладку **Магазин** в редакторе **Менеджер библиотек**.
2. Выберите нужную библиотеку и нажмите кнопку **Подключить** на карточке библиотеки.
3. Библиотека добавится в проект и отобразится на вкладке **Мои библиотеки** в редакторе **Менеджер библиотек**.

Отключение библиотеки

Чтобы отключить библиотеку:

1. Откройте вкладку **Мои библиотеки** в редакторе **Менеджер библиотек**.
2. Выберите нужную библиотеку и нажмите кнопку **Отключить** на карточке библиотеки.
3. Библиотека будет удалена из проекта. После отключения библиотека останется доступной на вкладке **Магазин** в редакторе **Менеджер библиотек**, где её можно подключить повторно.

7 Язык программирования ST

7.1 Ключевые слова

Ключевые слова — это уникальные комбинации символов, используемых как отдельные синтаксические элементы. Ключевые слова не должны содержать 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 (!, @ 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, 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: имена переменных предваряются заранее оговоренными префиксами, состоящими из одного или нескольких символов. 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
```

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 записывается 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]. 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

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

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

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>);
```

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 ^

Разыменование (взятие значения по указателю) реализуется оператором ^.

```
<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
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
```

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 представляет 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 представляет 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 устройства. 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 IDE поддерживает 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	Размер
TRUE (1), FALSE (0)	1 байт

7.5.2 Целочисленные типы

Тип данных	Нижняя граница	Верхняя граница	Размер
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	Размер
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	Размер
STRING	UTF-8	1 байт
WSTRING	USC-2	2 байта

7.5.5 Временные типы

ВНИМАНИЕ
В ближайших версиях ALTA IDE планируется изменение реализации временных типов данных. Существует вероятность, что временные переменные, созданные в проектах текущей версии ALTA IDE, могут работать некорректно при открытии этих проектов в более поздних версиях программного обеспечения. Рекомендуется учитывать данную информацию при обновлении среды разработки и переносе проектов.

Тип данных	Описание	Формат, разрешение наносекунды (ns)	Нижняя граница	Верхняя граница	Память
TIME (T), LTIME (LT)	длительность интервалов времени	T#xDxHxMxSx-MSxUSxNS	0	213503d23h34m33-s709ms551us615ns	64 бит
TIME_OF_DAY (TOD), LTOD	значение местного времени, начиная с 0 часов текущих суток	TOD#hh:mm:ss:ms	00:00:00	23:59:59.999_999_999	64 бит
DATE (D), LDATE (LD)	значение календарной даты, вычисляется относительно UNIX эпохи	D#yyyy-mm-dd	1677-09-21	2262-04-11	64 бит
DATE_AND_TIME (DT), LDT	значение календарной даты и местного времени, вычисляется относительно UNIX эпохи	DT#yyyy-mm-dd-hh:mm:ss.ms	1677-09-21 00:12:43.145-224192	2262-04-11 23:47:16.854775807	64 бит

7.5.6 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.

7.5.6.1 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.

7.5.6.2 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
```

7.5.7 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>
    END_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;
```

7.5.8 Массивы

Массив - совокупность элементов данных одного типа.

Синтаксис:

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

7.5.9 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_обьекта> := <имя_указателя>^;
```

Пример:

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

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

Указатель на массив	Содержит адрес первого элемента массива.
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.

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 видны из всех POU проекта (и доступны для чтения и записи). Глобальные переменные объявляются между ключевыми словами VAR_GLOBAL и END_VAR вне POU в отдельном списке внутри проекта. Количество списков неограничено.

Для объявления глобальной переменной необходимо создать программный объект (программу, функцию или функциональный блок), и объявить список глобальных переменных после завершающей части конструкции объявления компонента (END_PROGRAM END_FUNCTION или END_FUNCTION_BLOCK):

Пример:

Программа: programm_1

```
<объявления переменных и параметров>
    <тело реализации>
END_PROGRAM

VAR_GLOBAL
    a : INT;
END_VAR
```

Глобальную переменную нельзя напрямую привязать к параметрам устройства. Глобальная переменная должна быть присвоена переменной программы, которую, в свою очередь, можно привязать к параметрам устройства.

Пример:

Программа: programm_1

```
VAR
    LocalMask : WORD;       // Локальная копия битовой маски
END_VAR

// Присваивание глобальной переменной
LocalMask := G_DeviceMask;
END_PROGRAM

VAR_GLOBAL
    G_DeviceMask : WORD;    // Битовая маска канала устройства
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 (offset)
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
```

Приложение A. Библиотека Standard

.1 Цель документа

Библиотека Standard — это базовый набор функциональных блоков и функций, соответствующих стандарту МЭК 61131-3, обеспечивающих реализацию универсальных операций и типовых задач при программирования ПЛК.

В данном документе описан алгоритм работы блоков, входные/выходные параметры и типы данных каждого блока.

.2 Установка библиотеки

Библиотека Standard установлена в ALTA IDE по умолчанию. Удаление данной библиотеки из проекта невозможно.

.3 Описание библиотеки Standard

Библиотека Standard содержит:

- таймеры;
- счетчики;
- логические функциональные блоки (триггеры);
- детекторы импульсов;
- операторы сдвига;
- строковые функции;
- математические функции;
- операторы выборки;
- валидаторы;
- операции с временными типами данных;
- преобразователи.

.3.1 Таймеры

.3.1.1 TP

Функциональный блок TP (Pulse Timer) — повторитель импульсов.

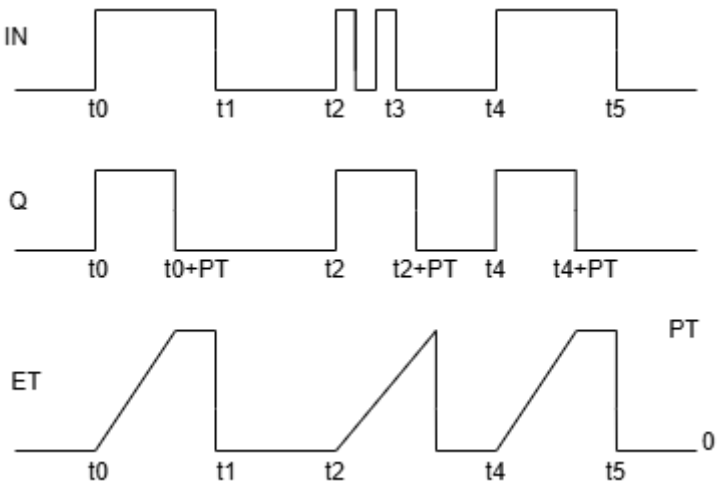


Рисунок A.1

Имя переменной	Тип данных	Описание
Входные переменные		
IN	BOOL	Вход импульса
PT	TIME	Заданная длительность импульса
Выходные переменные		

Имя переменной	Тип данных	Описание
Q	BOOL	Выход импульса
ET	TIME	Внутреннее время

Используется для генерирования импульса с заданной продолжительностью. Если IN становится «TRUE», Q становится «TRUE», и начинается отсчет внутреннего времени (ET). Если внутреннее время достигает значения PT, Q становится «FALSE» (независимо от IN). Отсчет внутреннего времени останавливается/сбрасывается, если IN становится «FALSE». Если внутреннее время не достигло значения PT, импульс IN не влияет на внутреннее время. Если внутреннее время достигло значения PT, и IN равен «FALSE», отсчет внутреннего времени останавливается/сбрасывается, и Q становится «FALSE».

Пример:

```
VAR
    TPInst:   TP;   //экземпляр TP
    Btn:      BOOL; //кнопка-сигнал
    PulseOut: BOOL; //импульсный выход
END_VAR

TPInst (IN := Btn, PT := T#200ms);

PulseOut := TPInst.Q;
```

.3.1.2 TON

Функциональный блок TON (On Delay Timer) — таймер с задержкой включения.

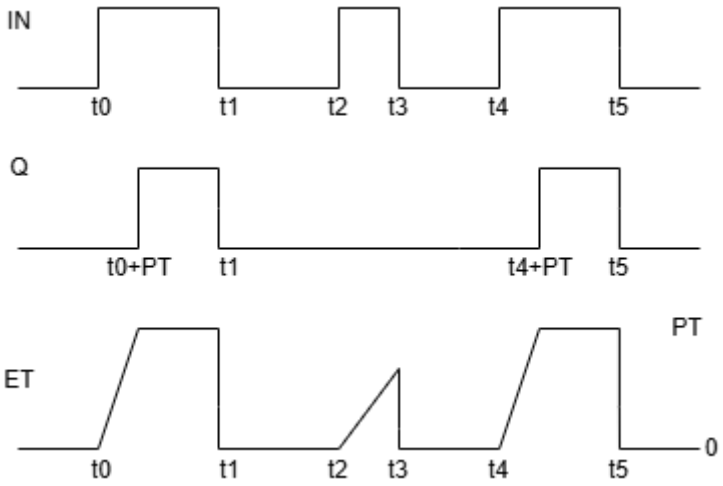


Рисунок А.2

Имя переменной	Тип данных	Описание
Входные переменные		
IN	BOOL	Вход таймера
PT	TIME	Заданное время задержки включения
Выходные переменные		
Q	BOOL	Выход таймера
ET	TIME	Внутреннее время

Если IN становится «TRUE», запускается отсчет внутреннего времени (ET). Если внутреннее время достигает значения PT, Q становится «TRUE». Если IN становится «FALSE», Q становится «FALSE», а подсчет внутреннего времени останавливается/сбрасывается. Если IN становится «FALSE» до того, как внутреннее время достигло значения PT, подсчет внутреннего времени останавливается/сбрасывается, а выход Q не устанавливается в «FALSE».

Пример:

```
VAR
    TOn:   TON;  //экземпляр таймера TON
    Start: BOOL; //входной сигнал
    Motor: BOOL; //выход
END_VAR

//вызов таймера каждый цикл
TOn (IN := Start, PT := T#3s);
//выход активируется после выдержки
Motor := TOn.Q;
```

.3.1.3 TOF

Функциональный блок TOF (off-delay timer) — таймер с задержкой отключения.

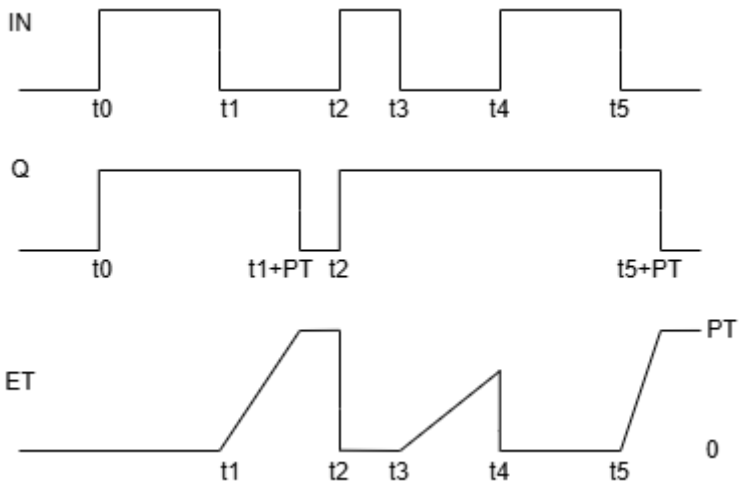


Рисунок А.3

Имя переменной	Тип данных	Описание
Входные переменные		
IN	BOOL	Вход таймера
PT	TIME	Заданное время задержки выключения
Выходные переменные		
Q	BOOL	Выход таймера
ET	TIME	Внутреннее время

Если IN становится «TRUE», Q становится «TRUE».

Если IN становится «FALSE», запускается отсчет внутреннего времени (ET).

Если внутреннее время достигает значения PT, Q становится «FALSE».

Если IN становится «TRUE», Q становится «TRUE», а подсчет внутреннего времени останавливается/сбрасывается.

Если IN становится «TRUE» до того, как внутреннее время достигло значения PT, подсчет внутреннего времени останавливается/сбрасывается, а выход Q не устанавливается в «FALSE».

Пример:

```
VAR
    TOff:   TOF;  //экземпляр таймера TOF
    RunCmd: BOOL; //команда "работать"
    Fan:    BOOL; //выход на вентилятор
END_VAR

TOff (IN := RunCmd, PT := T#10s);

Fan := TOff.Q;
```

.3.2 Счетчики

.3.2.1 CTU

Функциональный блок CTU (Up Counter) — инкрементный счётчик.

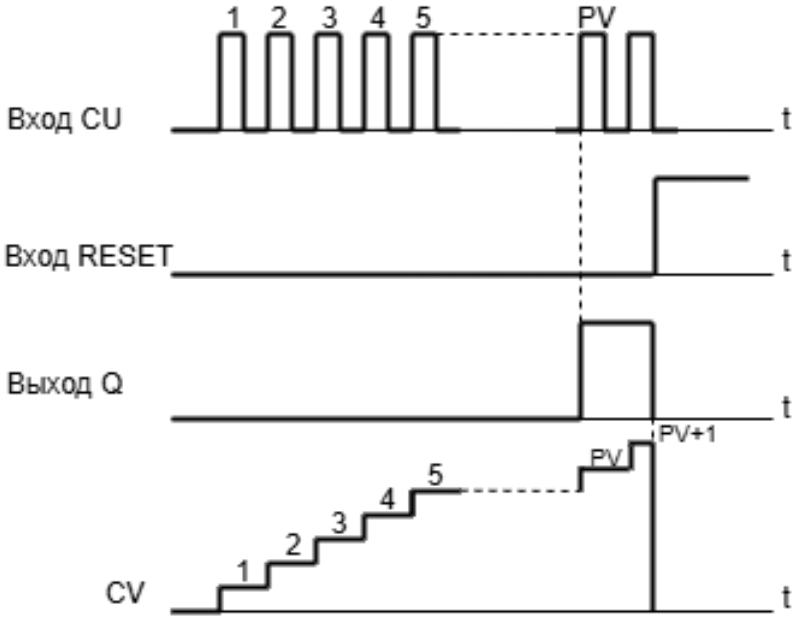


Рисунок А.4

Имя переменной	Тип данных	Описание
Входные переменные		
CU	BOOL	Вход счета - увеличение на 1 при срабатывании
RESET	BOOL	Сброс счётчика в 0
PV	WORD	Предустановленное значение (уставка)
Выходные переменные		
Q	BOOL	Признак достижения уставки
CV	WORD	Текущее значение счетчика

По каждому фронту на входе CU (переход из FALSE в TRUE) выход CV увеличивается на 1. Выход Q устанавливается в TRUE, когда счетчик достигнет значения заданного PV. Счетчик CV сбрасывается в 0 по входу RESET = TRUE.

Пример:

```
VAR
    Cnt:      CTU;  //экземпляр счетчика CTU
    Pulse:    BOOL; //импульсы счета
    ResetBtn: BOOL; //сброс
    Done:     BOOL;
    Value:    WORD;
END_VAR

Cnt (CU := Pulse, RESET := ResetBtn, PV := 10); //считать до 10

Done := Cnt.Q;   //TRUE, когда CV >= 10
Value := Cnt.CV; //текущее значение счетчика
```

.3.2.2 CTD

Функциональный блок CTD (Counter Down) — декрементный счётчик.

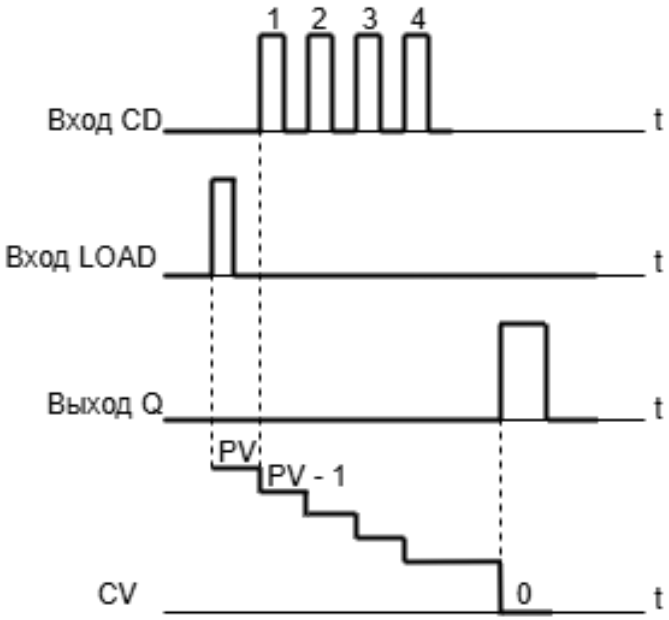


Рисунок А.5

Имя переменной	Тип данных	Описание
Входные переменные		
CD	BOOL	Вход счета - уменьшение на 1 при срабатывании
LOAD	BOOL	Вход загрузки предустановленного значения
PV	WORD	Предустановленное значение (уставка)
Выходные переменные		
Q	BOOL	Признак достижения 0
CV	WORD	Текущее значение счетчика

По каждому фронту на входе CD (переход из FALSE в TRUE) выход CV уменьшается на 1. Когда счетчик достигнет 0, счет останавливается, выход Q переключается в TRUE. Счетчик CV загружается начальным значением, равным PV по входу LOAD = TRUE.

Пример:

```
VAR
    CntD:      CTD;  //экземпляр счетчика CTD
    DecPulse: BOOL; //импульсы декремента
    LoadBtn:   BOOL; //загрузить начальное значение
    Zero:      BOOL;
    Value:     WORD;
END_VAR

CntD (CD := DecPulse, LOAD := LoadBtn, PV := 10); //загрузка 10, затем считать
 вниз
Zero := CntD.Q;   //TRUE, когда CV = 0
Value := CntD.CV; //текущее значение счетчика
```

.3.2.3 CTUD

Функциональный блок CTUD (Count Up/Down) — инкрементный/декрементный счетчик.

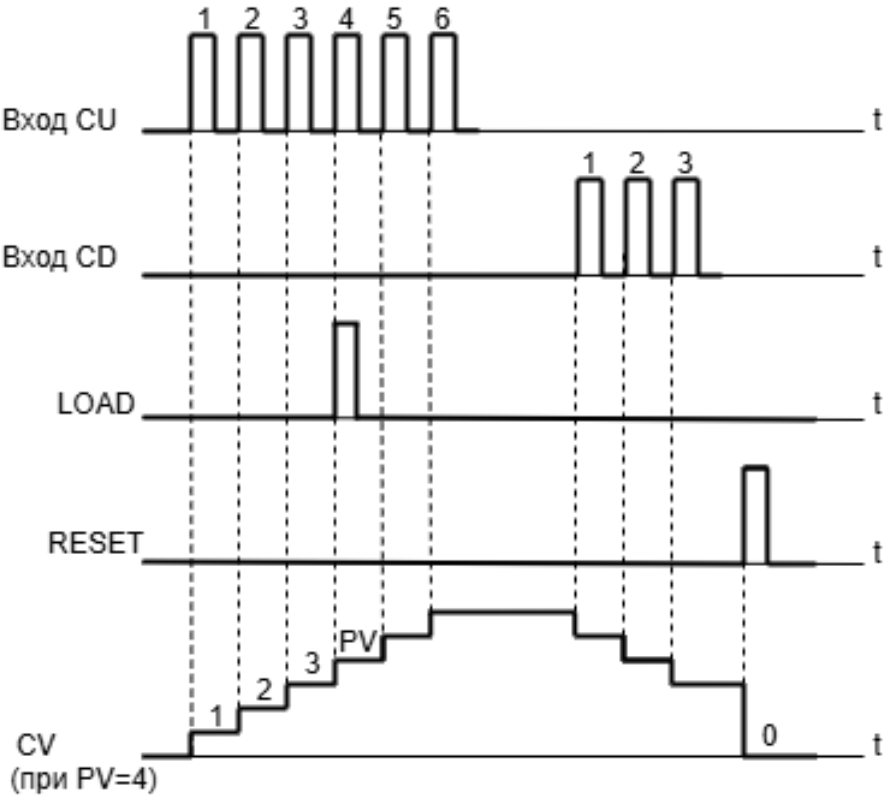


Рисунок А.6

Имя переменной	Тип данных	Описание
Входные переменные		
CU	BOOL	Вход счета - увеличение на 1 при срабатывании
CD	BOOL	Вход счета - уменьшение на 1 при срабатывании
RESET	BOOL	Сброс счетчика в 0
LOAD	BOOL	Вход загрузки предустановленного значения
PV	WORD	Предустановленное значение (уставка)
Выходные переменные		
QU	BOOL	Признак достижения уставки

Имя переменной	Тип данных	Описание
QD	BOOL	Признак достижения 0
CV	WORD	Текущее значение счетчика

По входу RESET счетчик CV сбрасывается в 0, по входу LOAD загружается значением PV.

По фронту на входе CU счетчик увеличивается на 1. По фронту на входе CD счетчик уменьшается на 1 (до 0).

QU устанавливается в TRUE, когда CV больше или равен PV.

QD устанавливается в TRUE, когда CV равен 0.

Пример:

```
VAR
    Cnt:      CTUD; //экземпляр счетчика CTUD
    UpPulse:  BOOL; //импульсы счета вверх
    DnPulse:  BOOL; //импульсы счета вниз
    ResetBtn: BOOL; //сброс
    LoadBtn:  BOOL; //загрузка PV и CV
    QU_reach: BOOL;
    QD_zero:  BOOL;
    Value:    WORD;
END_VAR

Cnt (CU := UpPulse, CD := DnPulse, RESET := ResetBtn, LOAD := LoadBtn, PV := 10);

QU_reach := Cnt.QU;  //TRUE, когда CV >= 10
QD_zero := Cnt.QD;   //TRUE, когда CV = 0
Value := Cnt.CV;     //текущее значение счетчика
```

.3.3 Логические функциональные блоки (триггеры)

.3.3.1 SR

Функциональный блок SR (Set-Reset) — переключатель с доминантой включения.

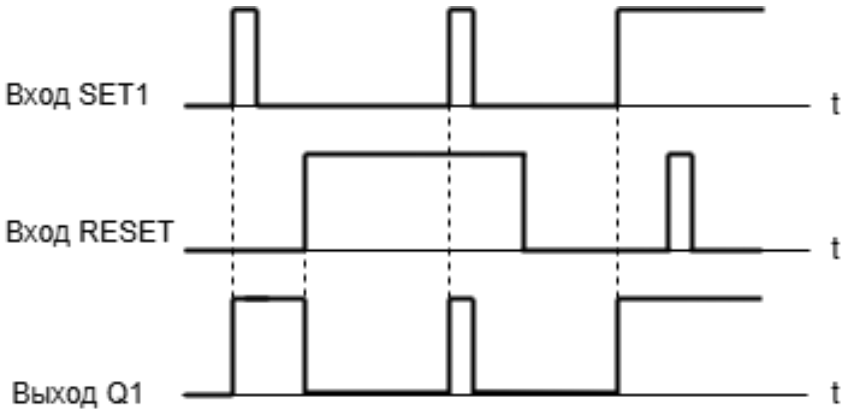


Рисунок А.7

Имя переменной	Тип данных	Описание
Входные переменные		
SET1	BOOL	Вход установки
RESET	BOOL	Вход сброса
Выходные переменные		
Q1	BOOL	Запоминаемый выход (состояние триггера)

Функциональный блок с доминирующим входом SET1. Выход Q1 становится «TRUE», когда вход SET1 становится «TRUE». Это состояние сохраняется, даже если SET1 возвращается обратно в «FALSE». Выход Q1 возвращается в «FALSE», когда вход RESET становится «TRUE».

Если входы SET1 и RESET находятся в «TRUE» одновременно, доминирующий вход SET1 установит выход Q1 в «TRUE». Когда функциональный блок вызывается первый раз, начальное состояние Q1 это «FALSE».

Пример:

```
VAR
    Latch:   SR;   //экземпляр переключателя SR
    StartPB: BOOL; //установить
    StopPB:  BOOL; //сбросить
    MotorOn: BOOL;
END_VAR

Latch (SET1 := StartPB, RESET := StopPB);

MotorOn := Latch.Q1;
```

.3.3.2 RS

Функциональный блок RS (Reset-Set) — переключатель с доминантой выключения.

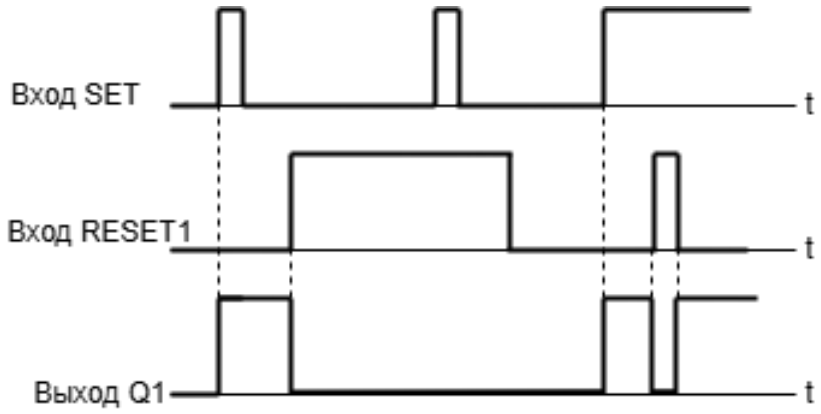


Рисунок А.8

Имя переменной	Тип данных	Описание
Входные переменные		
SET	BOOL	Вход установки
RESET1	BOOL	Вход сброса
Выходные переменные		
Q1	BOOL	Запоминаемый выход (состояние триггера)

Функциональный блок с доминирующим входом RESET1. Выход Q1 становится «TRUE», когда вход SET становится «TRUE». Это состояние сохраняется, даже если SET возвращается обратно в «FALSE». Выход Q1 возвращается в «FALSE», когда вход RESET1 становится «TRUE».

Если входы SET и RESET1 находятся в «TRUE» одновременно, доминирующий вход RESET1 установит выход Q1 в «FALSE». Когда функциональный блок вызывается первый раз, начальное состояние Q1 это «FALSE».

Пример:

```
VAR
    Latch:   RS;   //экземпляр переключателя RS
    StartPB: BOOL; //установить
    StopPB:  BOOL; //сбросить, доминирует
    MotorOn: BOOL;
END_VAR

Latch (SET := StartPB, RESET1 := StopPB);

MotorOn := Latch.Q1;
```

.3.4 Детекторы импульсов

.3.4.1 R_TRIG

Функциональный блок R_TRIG (Rising Trigger) — детектор импульсов по переднему фронту.

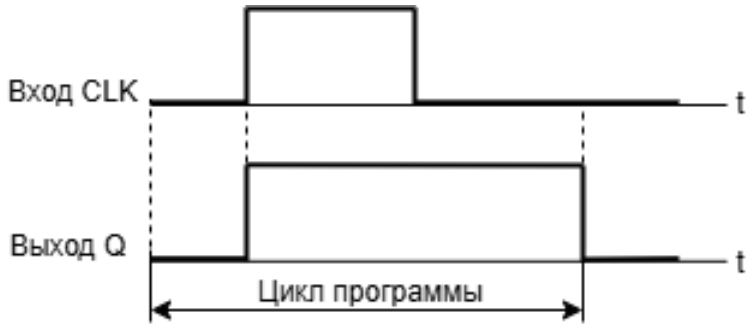


Рисунок А.9

Имя переменной	Тип данных	Описание
Входные переменные		
CLK	BOOL	Вход сигнала, по переднему фронту которого выполняется детектирование
Выходные переменные		
Q	BOOL	Выход импульса

Индикатор нарастания фронта, который генерирует на выходе одиночный импульс при нарастании фронта сигнала. Выход Q становится «TRUE», если происходит переход из «FALSE» в «TRUE» на входе CLK . Выход остается в состоянии «TRUE» от одного выполнения блока до следующего (один цикл); затем выход возвращается в «FALSE».

Пример:

```
VAR
    rt:     R_TRIG; //экземпляр R_TRIG
    Button: BOOL;   //входной сигнал
    Pulse:  BOOL;   //импульс 1 цикл
END_VAR

rt (CLK := Button);

Pulse := rt.Q; //TRUE только при переходе Button FALSE -> TRUE
```

.3.4.2 F_TRIG

Функциональный блок F_TRIG (Falling Trigger) — детектор импульсов по заднему фронту.

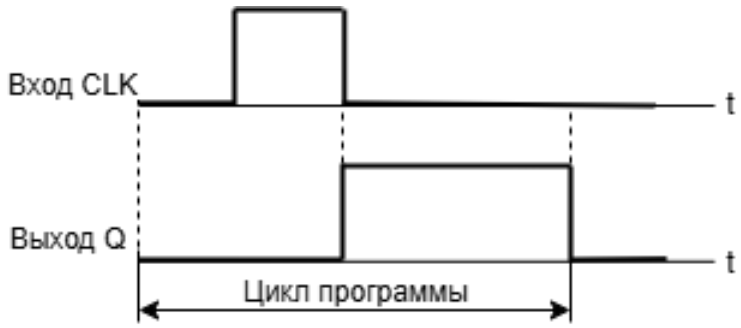


Рисунок А.10

Имя переменной	Тип данных	Описание
Входные переменные		
CLK	BOOL	Вход сигнала, по заднему фронту которого выполняется детектирование
Выходные переменные		
Q	BOOL	Выход импульса

Индикатор спада фронта, который генерирует на выходе одиночный импульс при спаде фронта сигнала. Выход Q становится «TRUE», если происходит переход из «TRUE» в «FALSE» на входе CLK . Выход будет оставаться в состоянии «TRUE» от одного выполнения блока до следующего; затем выход возвращается в «FALSE».

Пример:

```
VAR
    ft:     F_TRIG; //экземпляр F_TRIG
    Sensor: BOOL;   //входной сигнал
    Pulse:  BOOL;   //импульс 1 цикл
END_VAR

Ft (CLK := Sensor);

Pulse := ft.Q; //TRUE только при переходе Sensor FALSE -> TRUE
```

.3.5 Операторы сдвига

.3.5.1 SHL

Функция SHL (Shift Left) — побитовый сдвиг влево.

Возвращает в OUT побитный сдвиг операнда IN на N бит влево с заполнением битов справа нулями.
`OUT:=SHL(in,n)`

Входные переменные и результат типа BOOL, BYTE, WORD, DWORD или LWORD.

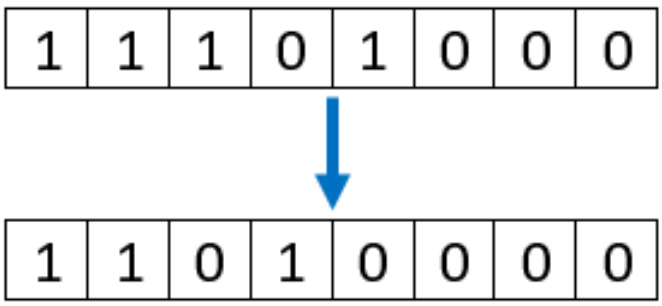


Рисунок А.11 – Сдвиг влево, n=1

В следующем примере представлены различные результаты out_byte и out_word в зависимости от типа входной переменной (BYTE и WORD), хотя их числовые значения равны.

Пример:

```
VAR
    in_byte:  BYTE := 16#45;
    in_word:  WORD := 16#45;
    out_byte: BYTE;
    out_word: WORD;
    n:        BYTE := 2;
END_VAR

out_byte := SHL (in_byte, n); //Результат 16#14
out_word := SHL (in_word, n); //Результат 16#0114
```

.3.5.2 SHR

Функция SHR (Shift Right) — побитовый сдвиг вправо.

Возвращает в OUT побитный сдвиг операнда IN на N бит вправо с заполнением битов слева нулями.
`OUT:=SHR(in,n)`

Входные переменные и результат типа BOOL, BYTE, WORD, DWORD или LWORD.

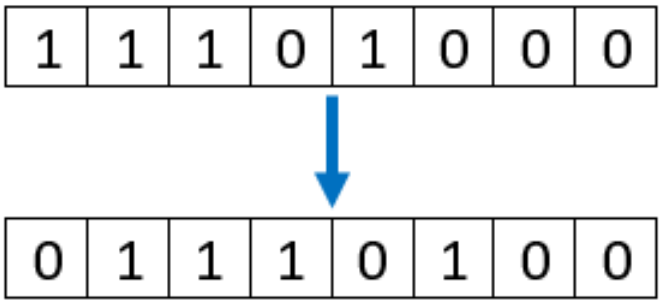


Рисунок А.12 – Сдвиг вправо, n=1

Следующий пример подчеркивает зависимость результата от типа входной переменной.

Пример:

```
VAR
    in_byte:  BYTE := 16#45;
    in_word:  WORD := 16#45;
    out_byte: BYTE;
    out_word: WORD;
    n:        BYTE := 2;
END_VAR

out_byte := SHR (in_byte, n); //Результат 16#11
out_word := SHR (in_word, n); //Результат 16#0011
```

.3.5.3 ROL

Функция ROL (Rotate Left) — циклический сдвиг влево.

Возвращает в OUT циклический сдвиг аргумента IN на N бит влево, младшие биты последовательно заменяются старшими.
`OUT:=ROL(in,n)`

Входные переменные и результат типа BOOL, BYTE, WORD, DWORD или LWORD.

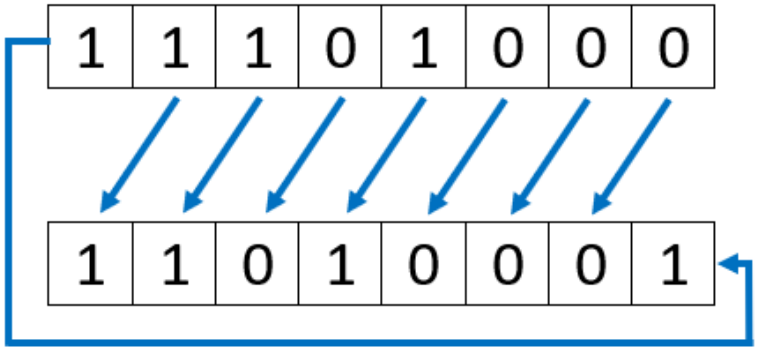


Рисунок А.13 – Циклический сдвиг влево, n=1

В следующем примере представлены различные результаты OUT_byte и OUT_word в зависимости от типа входной переменной (BYTE и WORD), хотя числовые их значения равны.

Пример:

```
VAR
    in_byte:  BYTE := 16#45;
    in_word:  WORD := 16#45;
    out_byte: BYTE;
    out_word: WORD;
    n:        BYTE := 2;
END_VAR

out_byte := ROL (in_byte, n); //Результат 16#15
out_word := ROL (in_word, n); //Результат 16#0114
```

.3.5.4 ROR

Функция ROR (Rotate Right) — циклический сдвиг вправо.

Возвращает в OUT циклический сдвиг аргумента IN на N бит вправо, старшие биты последовательно заменяются младшими.
`OUT:=ROR(in,n)`

Входные переменные и результат типа BOOL, BYTE, WORD, DWORD или LWORD.

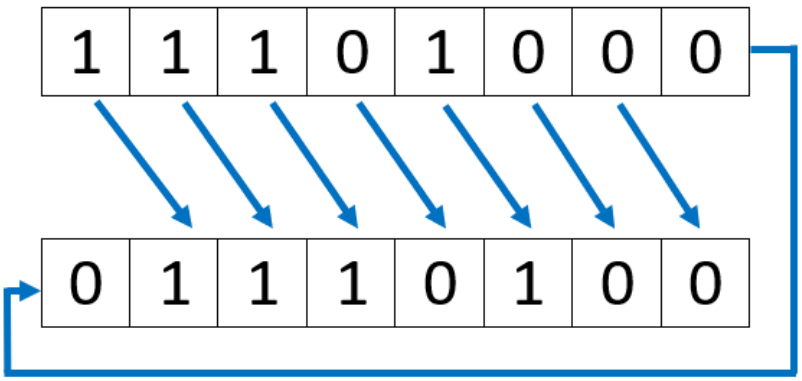


Рисунок А.14 – Циклический сдвиг вправо, n=1

Следующий пример подчеркивает зависимость результата от типа входной переменной.

Пример:

```
VAR
    in_byte:  BYTE := 16#45;
    in_word:  WORD := 16#45;
    out_byte: BYTE;
    out_word: WORD;
    n:        BYTE := 2;
END_VAR

out_byte := ROR (in_byte, n); //Результат 16#51
out_word := ROR (in_word, n); //Результат 16#4011
```

.3.5.5 TO_BIG_ENDIAN, TO_LITTLE_ENDIAN

Функции интерпретации порядка байт при обмене данными.

- TO_BIG_ENDIAN — старший байт передаётся первым.
- TO_LITTLE_ENDIAN — младший байт передаётся первым

Входные переменные и результат любого типа, для которого имеет смысл побайтовое представление.

Пример:

```
VAR
    w:   WORD := 16#1234;
    q:   WORD := 16#1234;
    wBE: WORD;
    qLE: WORD;
END_VAR

wBE := TO_BIG_ENDIAN (w);    //Возвращаемое значение 16#3412
qLE := TO_LITTLE_ENDIAN (q); //Возвращаемое значение 16#1234
```

.3.5.6 FROM_BIG_ENDIAN, FROM_LITTLE_ENDIAN

Функция интерпретации порядка байт при обмене данными.

- FROM_BIG_ENDIAN — интерпретирует входное значение как записанное в big-endian, и преобразует его в порядок байт, принятых в системе.
- FROM_LITTLE_ENDIAN — интерпретирует входное значение как записанное в little-endian, и преобразует его в порядок байт, принятых в системе.

Входные переменные и результат любого типа, для которого имеет смысл побайтовое представление.

Пример:

```
VAR
    wFBE:  WORD := 16#3412; (*байты 34 12, как если бы пришло от big-endian
значение 16#1234*)
    wFLE:  WORD := 16#1234; (*байты 12 34, как если бы пришло от little-endian
значение 16#1234*)
    valBE: WORD;
    valLE: WORD;
END_VAR

valBE := FROM_BIG_ENDIAN (wFBE);    (*на little-endian CPU возвращаемое значение
16#1234*)
valLE := FROM_LITTLE_ENDIAN (wFLE); (*на little-endian CPU возвращаемое значение
16#1234*)
```

.3.6 Строковые функции

.3.6.1 LEN

Функция, возвращающая длину строки.

Аргумент STR типа STRING, возвращаемое значение типа INT.

Пример:

```
VarSTRING1 := LEN ('ALTA'); //возвращаемое значение 4
```

.3.6.2 LEFT

Функция, возвращающая левую часть строки заданной длины.

Входная строка STR типа STRING, размер SIZE типа INT, возвращаемое значение STRING.

Пример:

```
VarSTRING1 := LEFT ('ALTA',3); //возвращаемое значение 'ALT'
```

.3.6.3 RIGHT

Функция, возвращающая правую часть строки заданной длины.

Входная строка STR типа STRING, размер SIZE типа INT, возвращаемое значение STRING.

Пример:

```
VarSTRING1 := RIGHT ('ALTA',3); //возвращаемое значение 'LTA'
```

.3.6.4 MID

Функция, возвращающая часть строки заданной длины с заданной позиции.

Входная строка STR типа STRING, размер LEN и POS типа INT, возвращаемое значение STRING.

`MID (STR, LEN, POS)` означает: вырезать LEN символов из STR строки, начиная с POS.

Пример:

```
VarSTRING1 := MID ('ALTA',2,2); //возвращаемое значение 'LT'
```

.3.6.5 CONCAT

Конкатенация (объединение) двух строк.

Обе входных строки STR1 и STR2 и результат типа STRING.

Пример:

```
VarSTRING1 := CONCAT ('ALTA','IDE'); //возвращаемое значение 'ALTAIDE'
```

.3.6.6 INSERT

Функция, предназначенная для вставки одной строки в заданную позицию другой строки.

Входные переменные STR1 и STR2 - типа STRING, POS - типа INT, возвращаемое значение - строка STRING.

`INSERT(STR1, STR2, POS)` означает: вставить STR2 в STR1 в позиции POS.

Пример:

```
VarSTRING1 := INSERT ('ALTA','IDE',2); //возвращаемое значение 'ALIDETA'
```

.3.6.7 DELETE

Функция, предназначенная для удаления части строки с заданной позиции.

Входные переменные STR типа STRING, LEN и POS типа INT, возвращаемое значение строка STRING.

`DELETE(STR, LEN, POS)` означает: удалить LEN символов из STR, начиная с позиции POS.

Пример:

```
VarSTRING1 := DELETE ('ALTAIDE',2,3); //возвращаемое значение 'ALTDE'
```

.3.6.8 REPLACE

Функция, предназначенная для замены части строки другой строкой с заданной позиции указанной длины.

Входные переменные STR1 и STR2 типа STRING, LEN и POS типа INT, возвращаемое значение строка STRING.

`REPLACE(STR1, STR2, LEN, POS)` означает: заменить LEN символов строки STR1 на STR2 начиная с позиции POS.

Пример:

```
VarSTRING1 := REPLACE ('ALTA','IDE',2,2); //возвращаемое значение 'AIDEA'
```

.3.6.9 FIND

Функция, предназначенная для поиска позиции заданного контекста в строке.

Входные переменные STR1 и STR2 типа STRING, возвращаемое значение INT.

`FIND(STR1, STR2)` означает: найти позицию в строке STR1, где впервые встречается подстрока STR2.

Нумерация позиций в строке начинается с 1. Если STR2 не найдена, STR1 возвращает 0.

Пример:

```
VarINT1 := FIND ('ALTA IDE','ID'); //возвращаемое значение 6
```

.3.6.10 STRING_EQUAL, WSTRING_EQUAL

Функция, предназначенная для сравнения входных переменных и возврата значения TRUE при совпадении значений.

Входные переменные могут быть типа STRING и WSTRING, возвращаемое значение типа BOOL.

Пример:

```
OUT := STRING_EQUAL ('ALTA','IDE');    //возвращаемое значение 'FALSE'
OUT := STRING_EQUAL ('ALTA','ALTA');   //возвращаемое значение 'TRUE'
OUT := WSTRING_EQUAL ("ALTA","IDE");   //возвращаемое значение 'FALSE'
OUT := WSTRING_EQUAL ("ALTA","ALTA");  //возвращаемое значение 'TRUE'
```

.3.6.11 STRING_GREATER, WSTRING_GREATER

Функция, предназначенная для сравнения входных переменных по лексикографическому признаку и возврата значения TRUE если значение первой переменной больше значения второй.

Входные переменные могут быть типа STRING и WSTRING, возвращаемое значение типа BOOL.

Пример:

```
VAR
    IN1: STRING := 'ALTA';
    IN2: STRING := 'IDE';
    IN3: WSTRING := "IDE";
    IN4: WSTRING := "ALTA";
    GS:  BOOL;
    GW:  BOOL;
END_VAR

GS := STRING_GREATER (IN1,IN2); (*возвращаемое значение FALSE, т.к. 'A'
лексикографически меньше 'I'*)
GW := WSTRING_GREATER (IN3,IN4); (*возвращаемое значение TRUE, т.к. 'I'
лексикографически больше 'A'*)
```

.3.6.12 STRING_LESS, WSTRING_LESS

Функция, предназначенная для сравнения входных переменных по лексикографическому признаку и возврата значения TRUE если значение первой переменной меньше значения второй.

Входные переменные могут быть типа STRING и WSTRING, возвращаемое значение типа BOOL.

Пример:

```
VAR
    IN1: STRING := 'ALTA';
    IN2: STRING := 'IDE';
    IN3: WSTRING := "IDE";
    IN4: WSTRING := "ALTA";
    GS:  BOOL;
    GW:  BOOL;
END_VAR

GS := STRING_LESS (IN1,IN2); (*возвращаемое значение TRUE, т.к. 'A'
лексикографически меньше 'I'*)
GW := WSTRING_LESS (IN3,IN4); (*возвращаемое значение FALSE, т.к. 'I'
лексикографически больше 'A'*)
```

.3.7 Математические функции

.3.7.1 ABS

Функция, предназначенная для возврата модуля входного числа.

Входная переменная может быть любого числового типа, с соответствующим возвращаемым значением.

Пример:

```
q := ABS (-5); //возвращаемое значение 5
```

.3.7.2 SQRT

Функция, предназначенная для возврата квадратного корня входного числа.

Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.

Пример:

```
q := SQRT (16.0); //возвращаемое значение 4.0
```

.3.7.3 LN

Функция, предназначенная для возврата натурального логарифма входного числа.

Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.
Пример:

```
q := LN (36.6); //возвращаемое значение 3.6000482404073
```

.3.7.4 LOG

Функция, предназначенная для возврата десятичного логарифма входного числа.
Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.
Пример:

```
q := LOG (36.6); //возвращаемое значение 1.5634810853944
```

.3.7.5 EXP

Функция, предназначенная для возврата значения экспоненты, возведенной в степень входного числа.
Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.
Пример:

```
q := EXP (2.0); //возвращаемое значение 7.3890561
```

.3.7.6 SIN

Функция, предназначенная для возврата значения sin (синуса) входного числа.
Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.
Пример:

```
q := SIN (0.5); //возвращаемое значение 0.4794255386
```

.3.7.7 COS

Функция, предназначенная для возврата значения cos (косинуса) входного числа.
Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.
Пример:

```
q := COS (0.7); //возвращаемое значение 0.7648421873
```

.3.7.8 TAN

Функция, предназначенная для возврата значения tg (тангенса) входного числа.
Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.
Пример:

```
q := TAN (0.33); //возвращаемое значение 0.342654129
```

.3.7.9 ASIN

Функция, предназначенная для возврата значения arcsin (арксинуса) входного числа.
Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.

Пример:

```
q := ASIN (0.5); //возвращаемое значение 0.5235987756
```

.3.7.10 ACOS

Функция, предназначенная для возврата значения arccos (арккосинуса) входного числа.

Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.

Пример:

```
q := ACOS (0.7); //возвращаемое значение 0.7953988302
```

.3.7.11 ATAN

Функция, предназначенная для возврата значения arctg (арктангенса) входного числа.

Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.

Пример:

```
q := ATAN (0.33); //возвращаемое значение 0.3187475604
```

.3.7.12 EXPT

Функция, предназначенная для возврата значения входного числа IN1 в степени IN2.

Входная переменная IN1 может быть типа REAL и LREAL, IN2 любого числового типа, возвращаемое значение типа REAL и LREAL.

Пример:

```
var1 := EXPT (7.1,2); //возвращаемое значение 50.41
```

.3.7.13 TRUNC

Функция, предназначенная для возврата целой части входного числа.

Входная переменная может быть типа REAL и LREAL, возвращаемое значение типа LINT.

Пример:

```
x := TRUNC (7.9);  //возвращаемое значение 7
y := TRUNC (-0.9); //возвращаемое значение 0
```

.3.7.14 ROUND

Функция, предназначенная для округления входного числа до ближайшего целого.

Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.

Пример:

```
q := ROUND (2.3); //возвращаемое значение 2.0
```

.3.7.15 ATAN2

Функция, предназначенная для возврата значения угла направления вектора.

Входная переменная может быть типа REAL и LREAL, с соответствующим возвращаемым значением.

Пример:

```
q := ATAN2 (0.0,-1.0); //возвращаемое значение 3.141592653589793
```

.3.8 Операторы выборки

.3.8.1 MAX

Функция, предназначенная для возврата наибольшего значения из двух входных переменных.

Входные переменные и возвращаемое значение могут быть любого числового типа данных.

Пример:

```
OUT := MAX (10,12); //возвращаемое значение 12
```

.3.8.2 MIN

Функция, предназначенная для возврата наименьшего значения из двух входных переменных.

Входные переменные и возвращаемое значение могут быть любого числового типа данных.

Пример:

```
OUT := MIN (10,12); //возвращаемое значение 10
```

.3.8.3 LIMIT

Функция, предназначенная для возврата ограниченного значения входной переменной.

Входные переменные и возвращаемое значение могут быть любого числового типа данных.

```
OUT := LIMIT(Min, IN, Max);
```

MIN и MAX задают нижнюю и верхнюю границы значений.

Пример:

```
OUT := LIMIT (0,5,10);   //возвращаемое значение 5
OUT := LIMIT (0,-2,10);  //возвращаемое значение 0
OUT := LIMIT (0,-15,10); //возвращаемое значение 10
```

.3.9 Валидаторы

.3.9.1 IS_VALID

Функция, предназначенная для проверки корректности значения: не содержит NaN (Not-a-Number, «не число») и Inf (Infinity, «бесконечность»). Возвращает значение TRUE, если значение корректно.

Входные переменные могут быть типа REAL и LREAL, возвращаемое значение типа BOOL.

Пример:

```
VAR
    x:   LREAL := 12.5;
    OUT: BOOL;
END_VAR

OUT := IS_VALID (x); //возвращаемое значение TRUE, т.к. x — корректное число
```

.3.9.2 IS_VALID_BCD

Функция, предназначенная для проверки корректности значения двоично-десятичного кода и возврата значения TRUE, если значение корректно.

Входные переменные могут быть типа BOOL, BYTE, WORD, DWORD, LWORD, возвращаемое значение типа BOOL.

Пример:

```
VAR
    bcdVal: WORD := 16#1234;  //BCD: 1-2-3-4
    OUT:    BOOL;
END_VAR

OUT := IS_VALID_BCD (bcdVal); //возвращаемое значение TRUE
```

.3.10 Операции с временными типами данных

.3.10.1 CONCAT_DATE, _TOD, _LTOD, _DT, _LDT

Функция, предназначенная для конкатенации (объединения) целочисленных переменных с возвратом значения во временном формате.

Входные переменные могут быть

- любого целочисленного типа, возвращаемое значение DATE, TOD, LTOD, DT, LDT;
- типа DATE и TOD, возвращаемое значение DT;
- типа DATE и LTOD, возвращаемое значение DT.

Пример:

```
VAR
    y:      INT := 2026;
    m:      INT := 11;
    d:      INT := 15;
    myDate: DATE;
END_VAR

myDate := CONCAT_DATE (y, m, d); //возвращаемое значение 2026-11-15
```

.3.10.2 SPLIT_DATE, _TOD, _LTOD, _DT, _LDT

Функция, предназначенная для разделения временных переменных.

Входные переменные могут быть типов DATE, TOD, LTOD, DT, LDT, возвращаемое значение INT.

ПРИМЕЧАНИЕ
В данной реализации возвращаемое значение всегда будет равно 0. Функцию можно использовать для разделения входной временной переменной на выходные переменные целочисленного типа.

Пример:

```
VAR

    myDate: DATE := D#2026-02-20;
    y:      INT;
    m:      INT;
    d:      INT;
    f:      INT;
END_VAR

f := SPLIT_DATE (myDate, y, m, d); //выходные переменные y=2026, m=02, d=20
```

.3.10.3 ADD_TIME, _LTIME, _TOD, _LTOD, _DT, _LDT

Функции, предназначенные для сложения переменных временных типов.

Наименование функции	Входные переменные	Выход функции	Пример
ADD_TIME	TIME	TIME	T#1s + T#500ms = T#1s500ms
ADD_LTIME	LTIME	LTIME	LTIME#1s + LTIME#500ms = LTIME#1s500ms
ADD_TOD_TIME	TIME, TOD	TOD	Прибавляет временной интервал к времени суток, время суток смещается вперед на интервал: T#10s + TOD#12:00:00 = TOD#12:00:10
ADD_LTOD_LTIME	LTOD, LTIME	LTOD	LT#15s + LTOD#15:00:00 = LTOD#15:00:15
ADD_DT_TIME	DT, TIME	DT	Прибавляет временной интервал к дате-времени, время суток смещается вперед на интервал: T#1h + DT#2026-02-20-12:00:00 = DT#2026-02-20-13:00:00

.3.10.4 SUB_TIME, _LTIME, _DATE, _LDATE, _TOD, _LTOD, _DT, _LDT

Функции, предназначенные для вычитания переменных временных типов.

Наименование функции	Входные переменные	Выход функции	Пример
SUB_TIME	TIME	TIME	T#15s - T#8s = T#7s
SUB_LTIME	LTIME	LTIME	LTIME#5s - LTIME#2s = LTIME#3s
SUB_DATE_DATE	DATE	TIME	D#2026-02-20 - D#2026-02-18 = T#48h
SUB_LDATE_ LDATE	LDATE	LTIME	LDATE#2026-02-20 - LDATE#2026-02-19 = LTIME#24h
SUB_TOD_TIME	TOD, TIME	TOD	TOD#12:00:10 - T#5s = TOD#12:00:05
SUB_LTOD_LTIME	LTOD, LTIME	LTOD	LTOD#12:00:10 - LTIME#3s = LTOD#12:00:07
SUB_TOD_TOD	TOD	TIME	TOD#12:00:10 - TOD#12:00:03 = TIME#7s
SUB_LTOD_LTOD	LTOD	LTIME	LTOD#12:00:10 - LTOD#12:00:05 = LTIME#5s
SUB_DT_TIME	DT, TIME	DT	DT#2026-02-20-12:00:10 - T#5s = DT#2026-02-20-12:00:05
SUB_LDT_TIME	LDT, LTIME	LDT	LDT#2026-02-20-12:00:10 - LTIME#5s = LDT#2026-02-20-12:00:05
SUB_DT_DT	DT	TIME	DT#2026-02-20-12:00:10 - DT#2026-02-20-12:00:03 = T#7s
SUB_LDT_LDT	LDT	LTIME	LDT#2026-02-20-12:00:10 - LDT#2026-02-20-12:00:03 = LTIME#7s

.3.10.5 MUL_TIME, _LTIME

Функция, предназначенная для умножения временной переменной на число.

Входные переменные TIME или LTIME, и переменная любого числового типа данных, возвращаемое значение типа TIME и LTIME соответственно.

Пример:

```
t := MUL_TIME (T#2s, 3);//возвращаемое значение T#6s
```

.3.10.6 DIV_TIME, _LTIME

Функция, предназначенная для деления временной переменной на число.

Входные переменные TIME или LTIME, и переменная любого числового типа данных, возвращаемое значение типа TIME и LTIME соответственно.

Пример:

```
t1 := DIV_TIME (T#10s, 4);//возвращаемое значение T#2s500ms
```

.3.10.7 TIME

Функция, предназначенная для возврата значения местного времени, начиная с 00.00 текущих суток.

Возвращаемое значение типа TIME.

Пример:

```
VAR
    CurTime := TIME;
END_VAR

CurTime := TIME (); //возвращаемое значение - текущее время, например T#6h41m30s
```

.3.11 Преобразователи

Преобразование типов происходит с помощью функций-преобразователей.

Пример:

```
VAR
    wVal: WORD := 16#00FA;  //250
    iVal: INT;
END_VAR

iVal := WORD_TO_INT (wVal); //iVal = 250
```

Тип данных	**Может быть преобразован**
LWORD	DWORD, WORD, BYTE, BOOL, LREAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT, STRING, WSTRING, DATE, LDATE, DT, LDT, TOD, LTOD, TIME, LTIME
DWORD	LWORD, WORD, BYTE, BOOL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT, STRING, WSTRING
WORD	LWORD, DWORD, BYTE, BOOL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT
BYTE	LWORD, DWORD, WORD, BOOL, CHAR, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT, STRING, WSTRING
BOOL	LWORD, DWORD, WORD, BYTE, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT
CHAR	BYTE, WORD, DWORD, LWORD, STRING, WCHAR
WCHAR	WORD, DWORD, LWORD, WSTRING, CHAR
REAL	DWORD, STRING, WSTRING, LREAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT
LREAL	LWORD, STRING, WSTRING, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT, TIME, LTIME
SINT	LWORD, DWORD, WORD, BYTE, LREAL, REAL, LINT, DINT, INT, ULINT, UDINT, UINT, USINT
INT	LWORD, DWORD, WORD, BYTE, LREAL, REAL, LINT, DINT, SINT, ULINT, UDINT, UINT, USINT
DINT	LWORD, DWORD, WORD, BYTE, BOOL, LREAL, REAL, LINT, INT, SINT, ULINT, UDINT, UINT, USINT

Тип данных	**Может быть преобразован**
LINT	LWORD, DWORD, WORD, BYTE, STRING, WSTRING, LREAL, REAL, DINT, INT, SINT, ULINT, UDINT, UINT, USINT, TIME, LTIME, TOD, LTOD, DATE, LDATE, BOOL
USINT	LWORD, DWORD, WORD, BYTE, STRING, LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, UINT
UINT	LWORD, DWORD, WORD, BYTE, STRING, LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UDINT, USINT
UDINT	LWORD, DWORD, WORD, BYTE, STRING, LREAL, REAL, LINT, DINT, INT, SINT, ULINT, UINT, USINT
ULINT	LWORD, DWORD, WORD, BYTE, STRING, LREAL, REAL, LINT, DINT, INT, SINT, UDINT, UINT, USINT, DATE, LDATE, DT, LDT, TOD, LTOD, TIME, LTIME
STRING	LINT, DINT, REAL, LREAL, WSTRING, CHAR
WSTRING	LINT, DINT, REAL, LREAL, WCHAR
TIME	STRING, WSTRING, LWORD, LINT, ULINT, LREAL, LTIME
LTIME	TIME, LWORD, LINT, ULINT, LREAL
DATE	STRING, WSTRING, LWORD, LINT, ULINT
LDATE	LWORD, LINT, ULINT
TOD	STRING, WSTRING, LWORD, LINT, ULINT, LTOD
LTOD	TOD, LWORD, LINT, ULINT
DT	STRING, WSTRING, LWORD, LINT, ULINT, DATE, TOD, LTOD, LDT
LDT	LWORD, LINT, ULINT, DT, DATE, TOD, LTOD, LTOD

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-154848-1.19