CODESYS V3.5

Первый старт



Руководство пользователя

21.09.2022

версия 3.0

Оглавление

Оглавление 3
1 Введение 7
1.1 Цель документа 7
1.2 Общие сведения о CODESYS V3.5 7
1.3 Версионность CODESYS 7
1.4 Контроллеры ОВЕН, программируемые в CODESYS V3.5 8
2 Установка ПО 9
2.1 Установка CODESYS 9
2.2 Установка таргет-файлов и пакетов 14
2.3 Установка библиотек 18
3 Быстрый старт 19
3.1 Создание проекта 19
3.2 Установка связи с контроллером ОВЕН 26
3.2.1 Подключение по интерфейсу Ethernet 26
3.2.2 Подключение по интерфейсу USB 31
3.3 Подключение к контроллеру ОВЕН в среде CODESYS 37
3.4 Подключение к виртуальному контроллеру в среде CODESYS 40
3.5 Загрузка и запуск проекта. Работа с проектом 44
3.6 Вопросы после «быстрого старта» 49
3.6.1 Приложение не сохраняется после перезагрузки контроллера 49
3.6.2 Как удалить приложение из контроллера? 49
3.6.3 Можно ли выгрузить приложение из контроллера и загрузить в другой контроллер? 50
3.6.4 Как правильно перенести проект на другой ПК с установленным CODESYS? 52
3.6.5 Где посмотреть лог сообщений контроллера? 53
3.6.6 Как понять, что в проекте есть ошибки? 53
3.6.7 С каким периодом выполняется код программы? 55
3.6.8 Как включить сетку в редакторе визуализации? 56
3.6.9 Существует ли ограничение на размер приложения? 57
4 Интерфейс CODESYS 58
4.1 Обзор интерфейса CODESYS 58
4.2 Панель меню 59
4.2.1 Меню «Файл» 59
4.2.2 Меню «Правка» 60
4.2.3 Меню «Вид» 61
4.2.4 Меню «Проект» 62

4.2.5 Меню «Компиляция» 64
4.2.6 Меню «Онлайн» 65
4.2.7 Меню «Отладка» 66
4.2.8 Меню «Инструменты» 68
4.2.9 Меню «Окно» 69
4.2.10 Меню «Справка» 70
4.3 Дерево проекта 71
4.3.1 Узлы контроллера и компонентов 71
4.3.2 Узел «Application» 75
4.3.3 Вкладка «POU» 77
4.3.4 Состав шаблона проекта 79
5 Основы программирования 80
5.1 Языки программирования стандарта МЭК 61131-3 81
5.2 Переменные 86
5.3 Типы данных 87
5.3.1 Тип BOOL 87
5.3.2 Целочисленные типы данных 88
5.3.3 Типы данных с плавающей точкой 89
5.3.4 Строковые типы 90
5.3.5 Типы данных даты и времени 91
5.3.6 Конверсия типов 92
5.3.7 Массивы 93
5.3.8 Пользовательские типы данных 97
5.3.9 Структуры 98
5.3.10 Объединения 100
5.3.11 Перечисления 101
5.3.12 Псевдонимы 103
5.3.13 Специфические типы данных 104
5.4 Области переменных 105
5.4.1 Виды областей переменных 105
5.4.2 Локальные переменные (VAR) 105
5.4.3 Входные переменные (VAR_INPUT) 105
5.4.4 Выходные переменные (VAR_OUTPUT) 105
5.4.5 Переменные области «вход-выход» (VAR_IN_OUT) 106
5.4.6 Глобальные переменные (VAR_GLOBAL) 106
5.4.7 Остальные области 107
5.5 Модификаторы областей переменных 108
5.5.1 Константы (CONSTANT) 108

5.5.2 Энергонезависимые переменные (RETAIN и PERSISTENT)....110
5.6 Операторы....113
5.6.1 Арифметические операторы....113
5.6.2 Логические операторы....114
5.6.3 Операторы сравнения....115
5.6.4 Операторы битового сдвига....116
5.6.5 Математические операторы....117
5.6.6 Тригонометрические операторы....118
5.6.7 Операторы выбора....119
5.6.8 Операторы конверсии типов....120
5.6.9 Оператор присваивания....120
5.6.10 Операторы определения размера....121
5.6.11 Остальные операторы....122
5.6.12 Порядок выполнения операторов....123
5.7 Управляющие операторы....124
5.7.1 Оператор условного выбора IF....124
5.7.2 Оператор множественного выбора CASE....125
5.7.3 Оператор цикла FOR....127
5.7.4 Остальные управляющие операторы....130
5.8 Программные объекты....131
5.8.1 Функции....132
5.8.2 Функциональные блоки....134
5.8.3 Программы....136
5.8.4 Сравнение функций, функциональных блоков и программ....136
5.8.5 Вложенные программные объекты (методы и др.)....137
5.9 Задачи....138
5.10 Библиотеки....140
5.10.1 Установка библиотек....140
5.10.2 Добавление библиотеки в проект CODESYS....141
5.10.3 Использование объектов библиотеки в проекте. Пространства имен....142
5.10.4 Библиотека Standard....145
6 Пример создания проекта диспетчеризации....149
6.1 Формулировка задачи....149
6.2 Настройка модулей Mx210....150
6.3 Эмуляция модулей Mx210 с помощью Modbus Universal MasterOPC Server....152
6.4 Настройка опроса модулей Mx210 с помощью шаблонов....153
6.4.1 Создание нового проекта и подключение к контроллеру....153
6.4.2 Установка пакета шаблонов Mx210....157

6.4.3 Настройка компонентов Modbus 158
6.4.4 Проверка обмена 162
6.5 Разработка программы 169
6.5.1 Создание структур 169
6.5.2 Привязка переменных к шаблонам Mx210 172
6.5.3 Создание функции IsValueOutOfLimits 174
6.5.4 Код программы – первая версия 175
6.5.5 Код программы – вторая версия 177
6.5.6 Создание функционального блока StatusChange 180
6.6 Создание визуализации 183
6.6.1 Создание фрейма 183
6.6.2 Добавление и настройка основных элементов экранов визуализации 190
6.6.3 Настройка экрана visuMainScreen 194
6.6.4 Создание конфигурации тревог 198
6.6.5 Настройка экрана visuAlarmTable 201
6.7 Проверка примера 204
6.8 Настройка архивации 208
6.8.1 Установка пакета архиватора 208
6.8.2 Добавление и настройка архиватора 208
6.8.3 Проверка архивации 212
6.9 Подключение контроллера к SCADA-системе 214
6.10 Подключение контроллера к облачному сервису OwenCloud 220
7 Отличия СПК и ПЛК 227
8 Импорт проектов из старых версий CODESYS 228
9 Дополнительные материалы 236

1 Введение

1.1 Цель документа

Этот документ содержит информацию, необходимую для начала работы в среде CODESYS V3.5. Он рассчитан на начинающих пользователей и не подразумевает наличия у читателей значительного опыта работы с программируемыми контроллерами. В рамках документа излагается только базовый объем информации с акцентом на простоту и доступность подачи учебного материала. Процесс ознакомления со средой CODESYS V3.5 описан применительно к контроллерам ОВЕН, но основная часть документа является универсальной и может быть использована для обучения работе с любыми контроллерами, программируемыми в этой среде. Кроме того, структура документа подразумевает, что у читателя вообще может не быть в наличии контроллера – поэтому отдельно рассматривается процесс запуска проектов на виртуальном контроллере, который устанавливается на ПК вместе со средой программирования.

Документ соответствует версии среды разработки **CODESYS V3.5 SP17 Patch 3**. Если вы планируете работать в более ранних версиях среды – то рекомендуем вам использовать предыдущие версии документа.

1.2 Общие сведения о CODESYS V3.5

CODESYS V3.5 – это среда создания приложений для программируемых контроллеров (ПЛК), разработанная немецкой компанией CODESYS Group. В этой среде программируются контроллеры более 500 производителей – в том числе, современная линейка контроллеров ОВЕН. CODESYS включает в себя редакторы языков программирования стандарта МЭК 61131-3, редактор визуализации, конфигураторы промышленных протоколов, средства отладки и другие компоненты. Среда разработки распространяется бесплатно; ее дистрибутив может быть загружен с сайта компании ОВЕН в разделе CODESYS V3/Среда программирования. Пользовательский интерфейс CODESYS имеет русскоязычную локализацию.

1.3 Версионность CODESYS

CODESYS V3.5 является идейным наследником среды CoDeSys V2.3, поддержка которой прекратилась в конце 2019 года. Если вы имеете опыт программирования в CoDeSys V2.3 – то это поможет вам в процессе освоения V3.5, хотя архитектура и функционал новой версии принципиально отличаются от старой.

Переход с платформы V2.3 на платформу V3.0 начался в 2005 году. В 2011 году вышла версия V3.5, развитие и поддержка которой продолжается и в настоящий момент (2022 г.). Раз в год происходит выпуск крупного обновления среды, называемого сервис-паком (SP). Для каждого сервиса-пака выходит несколько патчей (patch) с исправлениями ошибок. Поэтому полное обозначение версии среды на конкретном примере выглядит следующим образом:

V3.5 SP17 Patch 3

где **V3.5** – версия платформы, **SP17** – номер сервис-пака, **Patch 3** – номер патча.

Часто версия среды записывается в сокращенном виде: **3.5.17.30**. Подобная запись характерна не только для версии среды, но и для входящих в ее состав компонентов – плагинов, библиотек и т. д.

Конкретный контроллер нельзя программировать в произвольной версии CODESYS – нужно использовать именно ту версию среды, которая предусмотрена для этого производителем ПЛК. Если вы работаете с контроллером ОВЕН, то определить нужную версию CODESYS можно с помощью «выбиратора», который доступен на странице CODESYS V3 на сайте ОВЕН. Для использования «выбиратора» необходимо знать версию прошивки контроллера – ее можно посмотреть в web-конфигураторе контроллера (*а для панельного контроллера СПК – версия прошивки также отображается на экране, если в контроллер еще не загружено приложение CODESYS*).

ПРИМЕЧАНИЕ
См. видео по подключению к web-конфигуратору и его использованию: https://youtu.be/ZIRRdIQ0s-U

1.4 Контроллеры ОВЕН, программируемые в CODESYS V3.5

В данный момент (2022 г.) компания ОВЕН выпускает три линейки контроллеров, программируемых в среде CODESYS V3.5.

Таблица 1.1 – Список контроллеров ОВЕН, программируемых в CODESYS V3.5

Линейка контроллеров	Требуемая для программирования версия CODESYS (для заводских прошивок)
ПЛК210	3.5.17.30
ПЛК200	3.5.17.30
СПК1xx [M01]	3.5.17.30

2 Установка ПО

2.1 Установка CODESYS

Дистрибутив CODESYS может быть загружен с сайта ОВЕН в разделе CODESYS V3/Среда программирования. Дистрибутив распространяется в виде .zip-архива. Распакуйте архив в отдельную папку и запустите файл установщика с расширением **.exe** (*важно запустить именно этот файл, а не файл с расширением **.msi** – иначе в процессе установки не будет установлена утилита **CODESYS Installer**, без которой работа в современных версиях CODESYS крайне неудобна*).

ПРИМЕЧАНИЕ
Установка CODESYS требует подключения к интернету, поскольку в процессе установки загружается дополнительное ПО, необходимое для работы среды.

ПРИМЕЧАНИЕ
На время установки CODESYS отключите ваш антивирус.

ПРИМЕЧАНИЕ
Начиная с версии **V3.5 SP17 Patch 3** среда CODESYS не поддерживает операционные системы Windows 7 и 8. Поддерживаемые операционные системы – **Windows 10 / 11**.

В появившемся окне нажмите кнопку **Install** для установки утилиты **CODESYS Installer** – она потребуется для установки пакета таргет-файлов, о которых мы расскажем в следующем пункте.

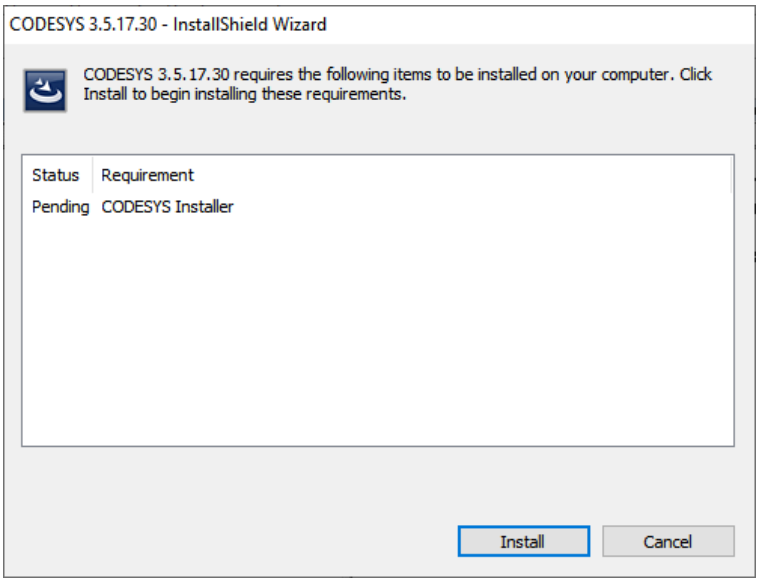


Рисунок 2.1.1 – Окно установки утилиты CODESYS Installer

В следующем окне нажмите кнопку **Next**.

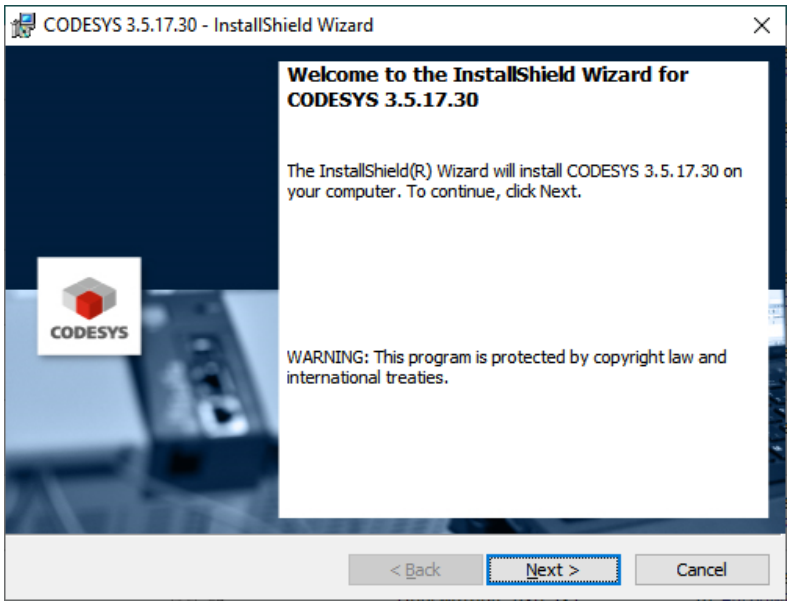


Рисунок 2.1.2 – Стартовое окно установщика

В следующем окне ознакомьтесь с текстом лицензионного соглашения, выберите пункт **I accept the terms in the license agreement** и нажмите кнопку **Next**:

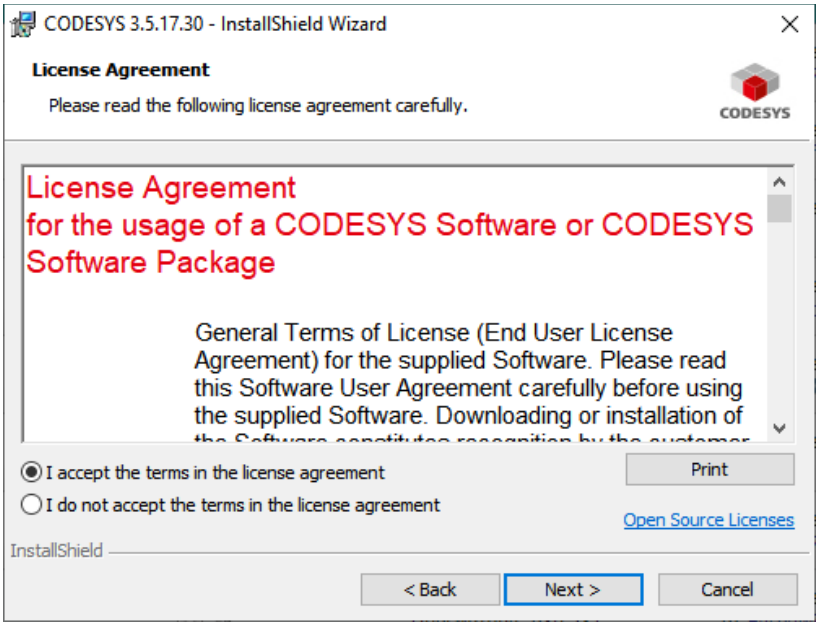


Рисунок 2.1.3 – Окно лицензионного соглашения

В следующем окне ознакомьтесь с информацией об основных изменениях, произошедших в устанавливаемой версии CODESYS, выберите пункт **I have read the information** и нажмите кнопку **Next**:

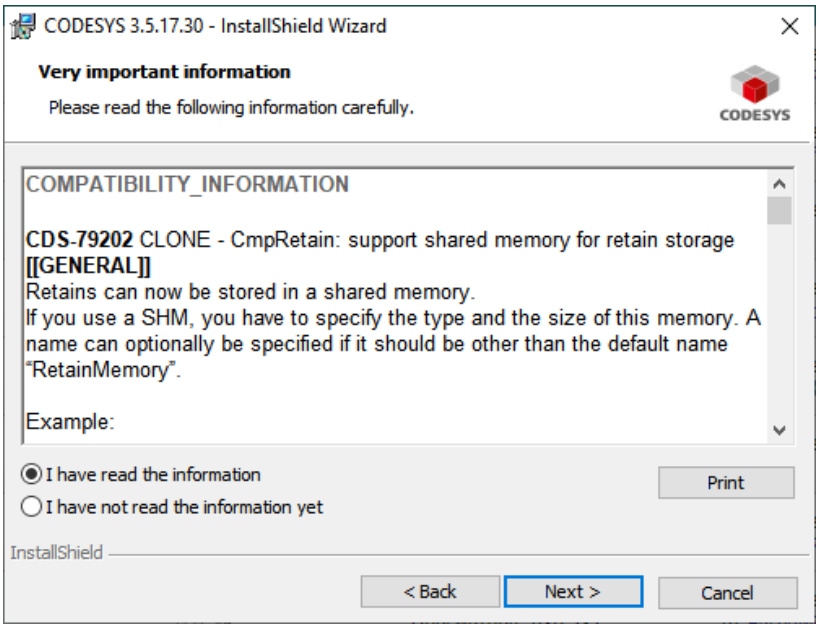


Рисунок 2.1.4 – Окно с информацией об основных изменениях в устанавливаемой версии среды

В следующем окне выберите директорию, в которую будет установлен CODESYS, и нажмите кнопку **Next**. Каждую версию среды следует устанавливать в отдельную директорию.

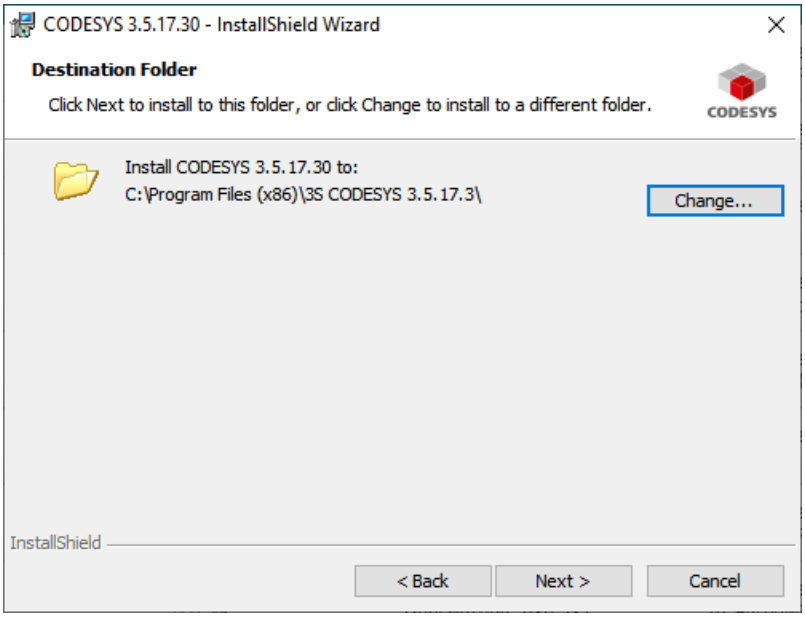


Рисунок 2.1.5 – Окно выбора директории установки

В следующем окне необходимо выбрать режим установки CODESYS. Обязательно выберите режим полной установки (**Complete**), чтобы установить все доступные плагины, и нажмите **Next**.

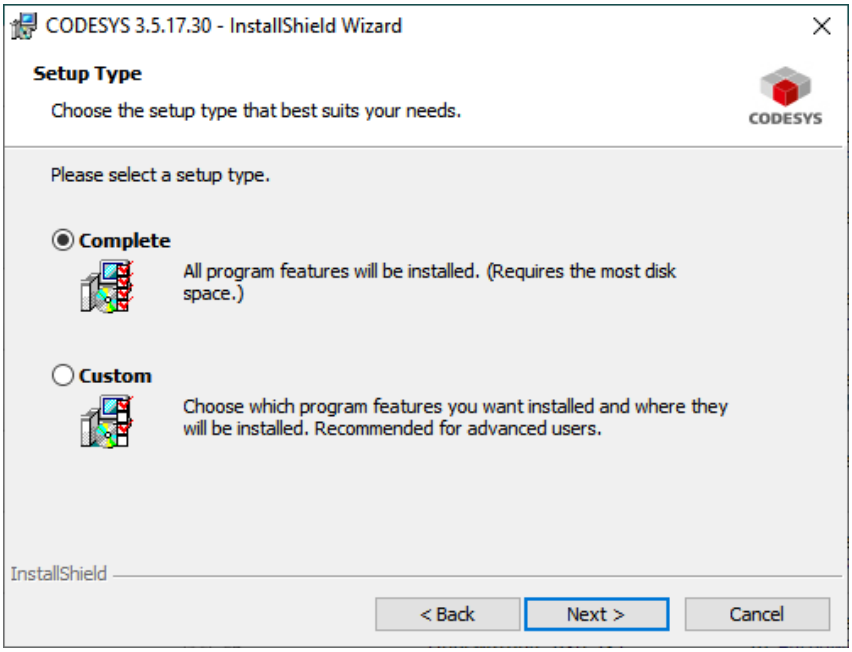


Рисунок 2.1.6 – Окно выбора режима установки CODESYS

В следующем окне нажмите кнопку **Install** для запуска процесса установки.

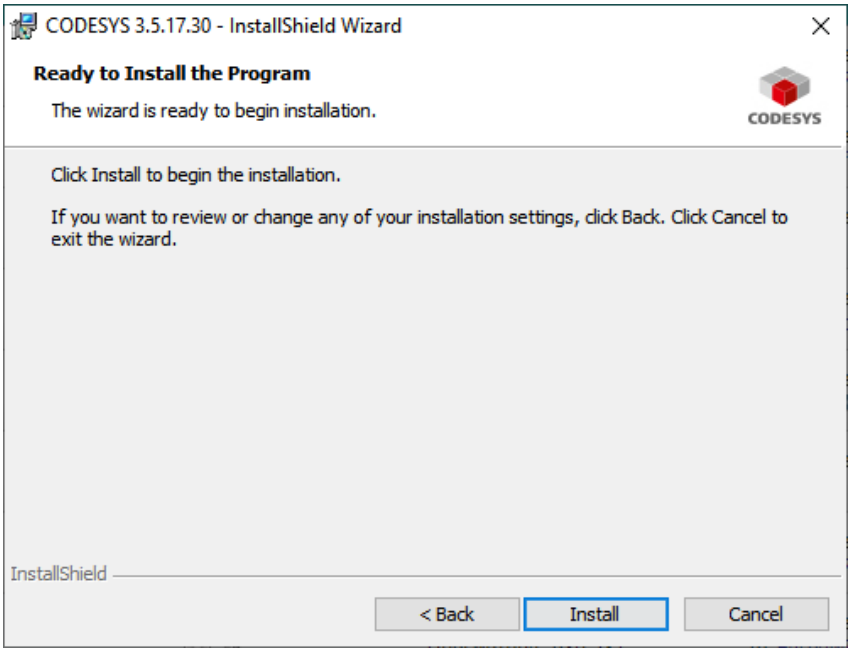


Рисунок 2.1.7 – Окно запуска установки CODESYS

Процесс установки занимает от 10 минут до часа (в зависимости от характеристик ПК). В случае его успешного завершения появится следующее окно:

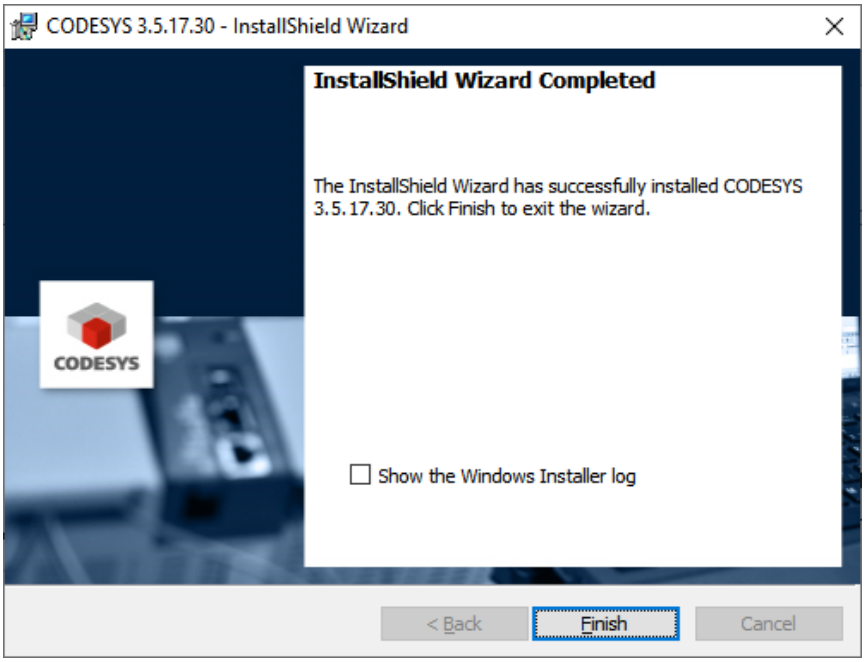


Рисунок 2.1.8 – Окно завершения установки CODESYS

Нажмите кнопку **Finish** для закрытия этого окна.

2.2 Установка таргет-файлов и пакетов

CODESYS используется для программирования множества контроллеров разных производителей. Поскольку у различных контроллеров разные аппаратные и программные возможности, то CODESYS должен «знать» о них для выбора подходящего компилятора, скрытия неподдерживаемых функций и т. д. Для этого используется **таргет-файл**, содержащий всю необходимую информацию. Таргет-файл является неотъемлемой частью проекта CODESYS – без него, в частности, не получится установить соединение с контроллером для загрузки приложения.

Таргет-файлы для всех контроллеров ОВЕН распространяются в виде единого пакета (файла формата **.package**). Его можно загрузить с сайта ОВЕН в разделе CODESYS V3/Сервисное ПО.

Как и используемая версия CODESYS, версия пакета таргет-файлов должна соответствовать версии прошивки контроллера. Если вы работаете с контроллером ОВЕН, то определить нужную версию пакета таргет-файлов можно с помощью «выбиратора», который доступен на странице CODESYS V3 на сайте ОВЕН. Кроме того, можно загрузить требуемую для конкретного контроллера версию пакета таргет-файлов из его web-конфигуратора.

Версионность пакета таргет-файлов соответствует версионности CODESYS: например, в данный момент для контроллеров, программируемых в версии **CODESYS V3.5 SP17 Patch 3** (**3.5.17.30**) используется пакет таргет-файлов версии **3.5.17.31**. Последняя цифра означает порядковый номер пакета – то есть для следующих прошивок могут быть выпущены пакеты 3.5.17.32, 3.5.17.33 и т. д.

В примерах документа мы будем использовать таргет-файлы ОВЕН – так что обязательно установите их пакет.

Начиная с версии CODESYS V3.5 SP17 установка пакетов производится с помощью утилиты CODESYS Installer. Запустить утилиту можно через меню **Пуск** (**CODESYS – CODESYS Installer**) или же из интерфейса CODESYS (**Инструменты – CODESYS Installer**).

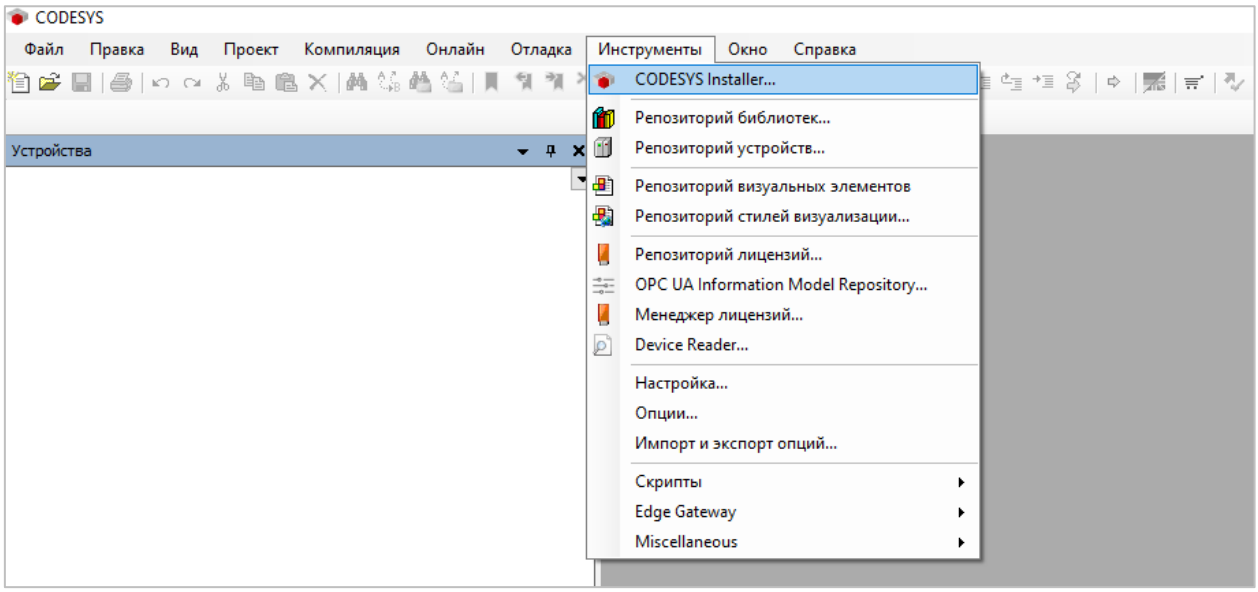


Рисунок 2.2.1 – Запуск утилиты CODESYS Installer из интерфейса среды CODESYS

Если утилита запущена из меню **Пуск**, то сначала будет отображен список установленных на этом ПК **окружений** (см. рис. 2.2.2). Окружение представляет собой конкретную версию CODESYS с конкретным набором плагинов конкретных версий. Создать новое окружение можно прямо в CODESYS Installer с помощью кнопки **Add Installation**. Например, если у вас уже установлено

окружение CODESYS V3.5 SP17 Patch 3, то вы можете создать еще одно окружение с этой же версией среды и обновить в нем плагины до последних доступных версий, чтобы протестировать их функционал. При этом у вас останется ранее установленное «стабильное» окружение с версиями плагинов, входящих в «чистый» дистрибутив CODESYS V3.5 SP17 Patch 3, и эти два окружения никак не будут влиять друг на друга.

Выберите окружение, в котором вы собираетесь установить пакет таргет-файлов, и нажмите кнопку **Change**. Если вы запустили CODESYS Installer из интерфейса CODESYS (см. рис. 2.2.1) – то сразу окажетесь в меню соответствующего окружения, и кнопку **Change** нажимать не потребуется.

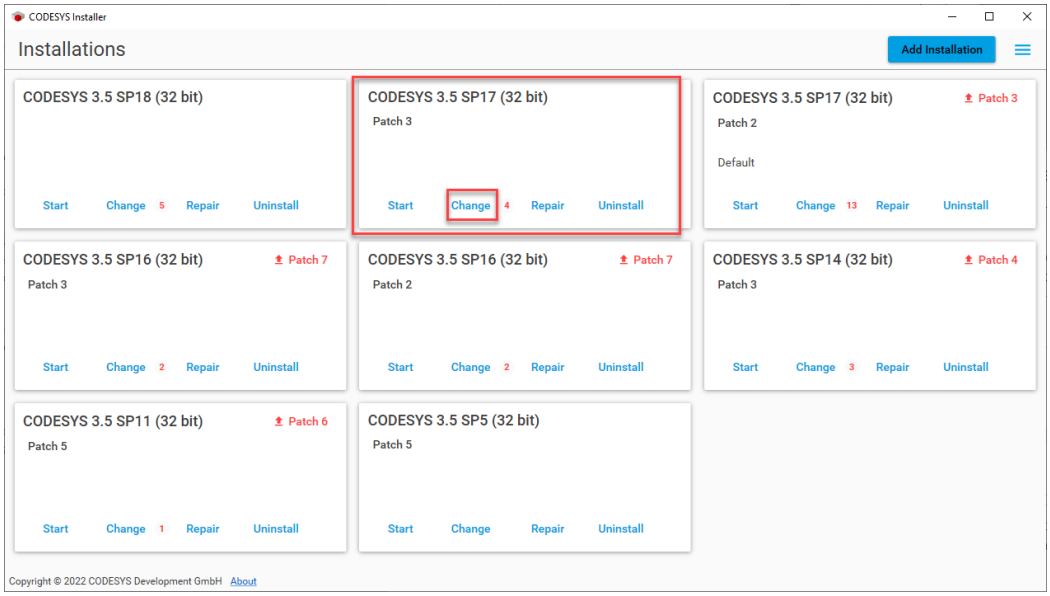


Рисунок 2.2.2 – Список установленных на ПК окружений CODESYS (*на вашем ПК набор окружений будет отличаться*)

В появившемся окне нажмите кнопку **Install file**, выберите пакет таргет-файлов и нажмите **Открыть**. Перед запуском установки потребуется закрыть все открытые экземпляры среды CODESYS.

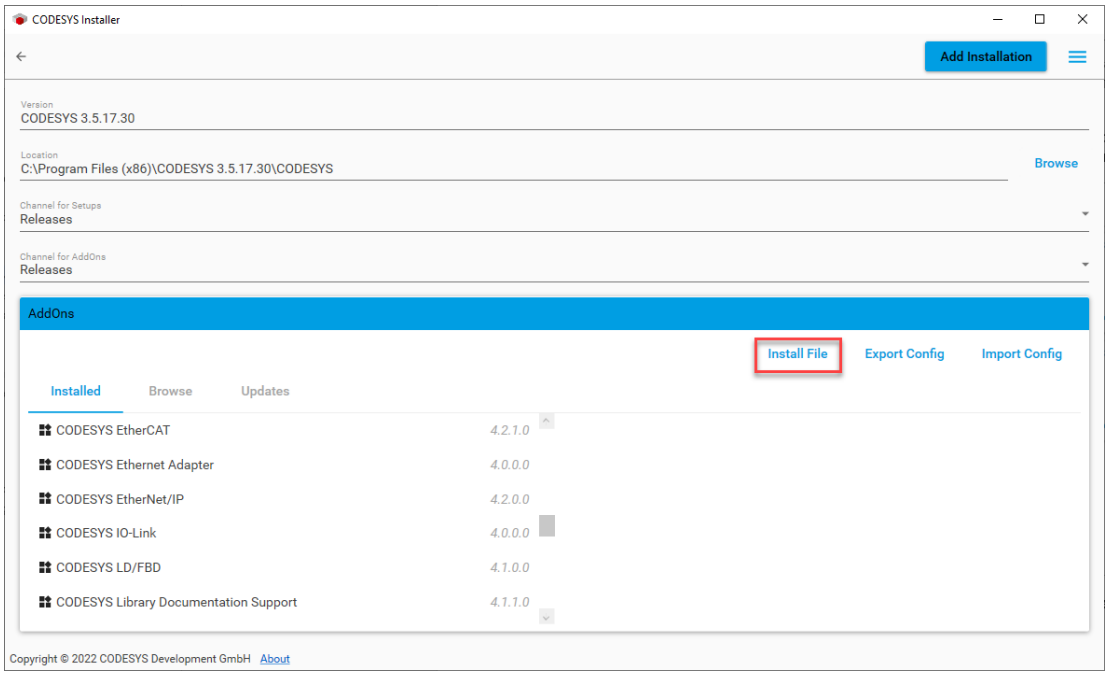


Рисунок 2.2.3 – Кнопка установки пакетов

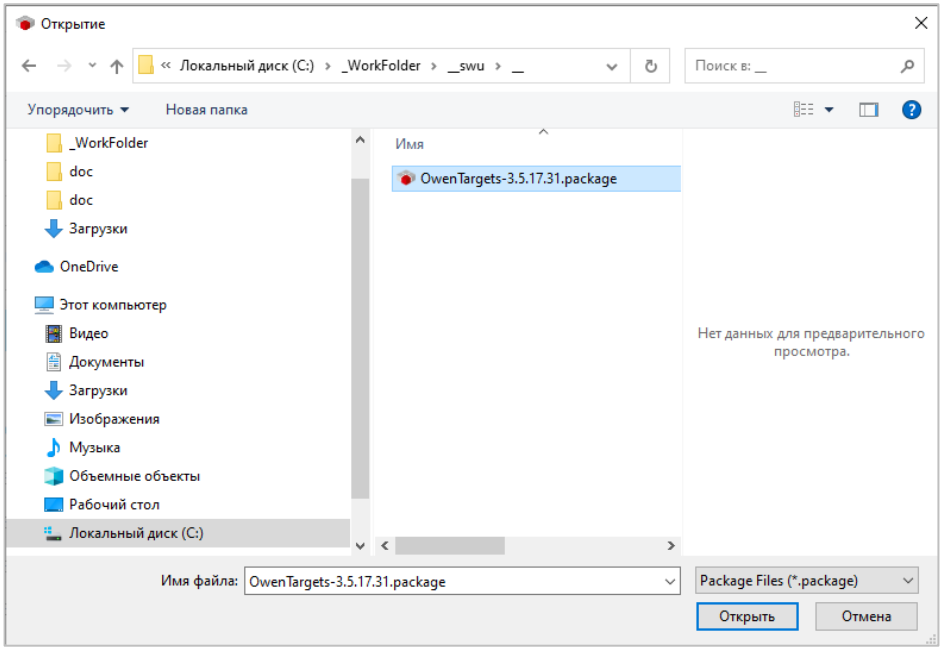


Рисунок 2.2.4 – Выбор пакета таргет-файлов

В появившемся окне нажмите кнопку **OK** для подтверждения установки пакета.

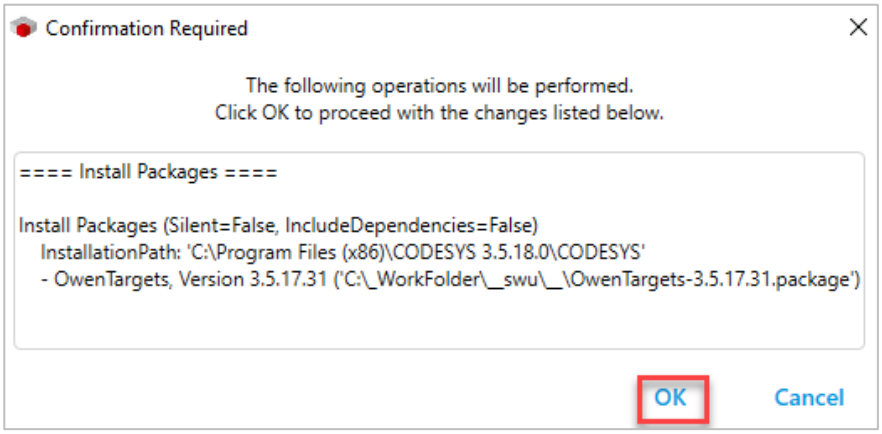


Рисунок 2.2.5 – Подтверждение установки пакета

В появившемся окне установите галочку **I want to continue despite of the missing signature(s)** для подтверждения установки пакета без цифровой подписи и нажмите кнопку **Continue**.

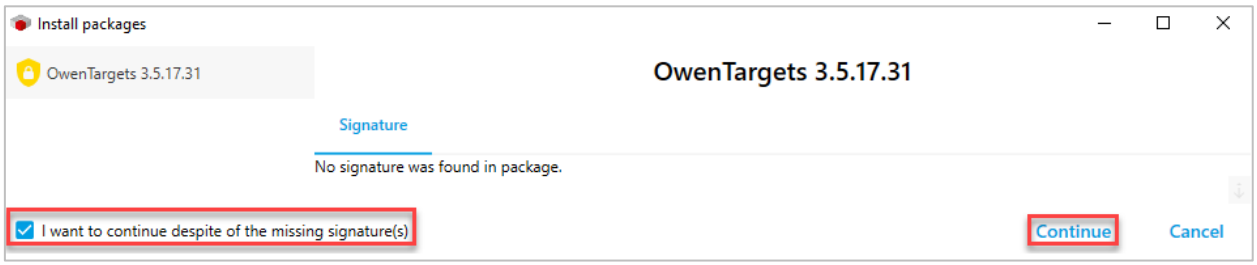


Рисунок 2.2.6 – Подтверждение установки пакета без цифровой подписи

После этого начнется установка пакета таргет-файлов ОВЕН. В процессе установки появится окно мастера установки шрифтов для панельных контроллеров СПК. Нажмите кнопку **Установить**, а в следующем окне – кнопку **Завершить**.

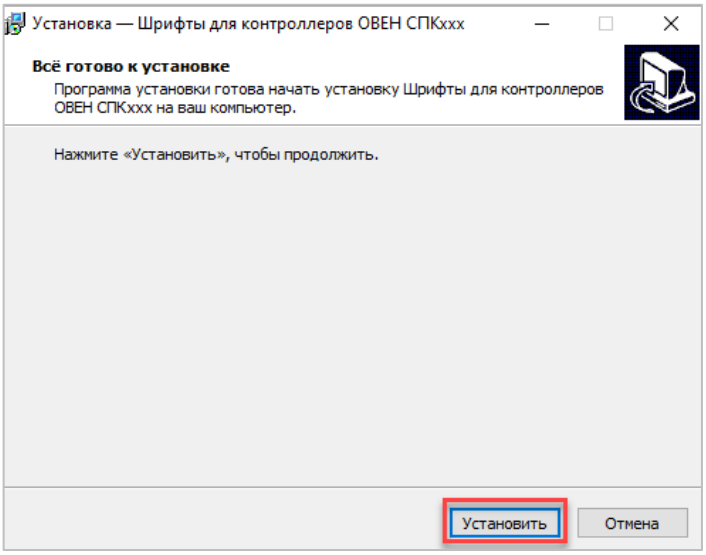


Рисунок 2.2.7 – Окно запуска установки шрифтов

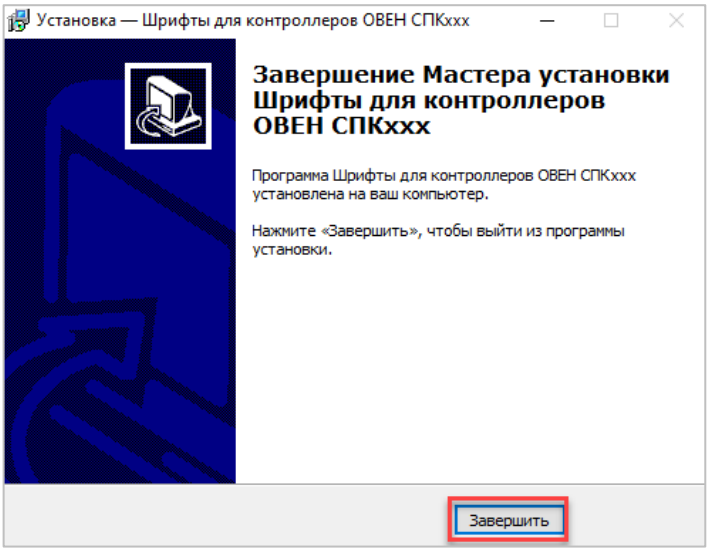


Рисунок 2.2.8 – Окно завершения установки шрифтов

После завершения установки нажмите кнопку **Ok** в последнем появившемся окне.

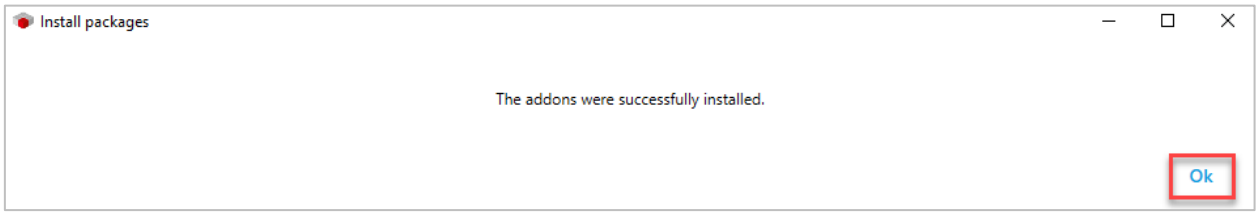


Рисунок 2.2.9 – Окно завершения установки CODESYS

Аналогичным образом устанавливаются другие пакеты – например, пакеты шаблонов опроса устройств ОВЕН.

2.3 Установка библиотек

Для расширения возможностей CODESYS производители контроллеров и пользователи могут разрабатывать собственные объекты (функциональные блоки, диалоги визуализации и т. д.) и распространять их в виде библиотек.

Библиотеки, разработанные компанией ОВЕН, и некоторые свободно распространяемые библиотеки от пользователей CODESYS доступны на сайте ОВЕН в разделе CODESYS V3/Библиотеки и компоненты.

Для установки библиотеки в CODESYS нужно выбрать в меню **Инструменты** пункт **Репозиторий библиотек**, нажать кнопку **Установить** и указать путь к файлу нужной библиотеки.

Библиотеки могут распространяться в виде двух форматов:

- **.compiled-library** – для таких библиотек доступ к исходным кодам запрещен;
- **.library** – такие библиотеки доступны в исходных кодах.

Обратите внимание, что если в окне выбора файла библиотеки указан неподходящий формат – то установить ее не получится. Поэтому рекомендуется переключить фильтр в режим **Все файлы**.

В данный момент никаких библиотек устанавливать не требуется – в рамках примера мы будем использовать только библиотеки, входящие в дистрибутив CODESYS.

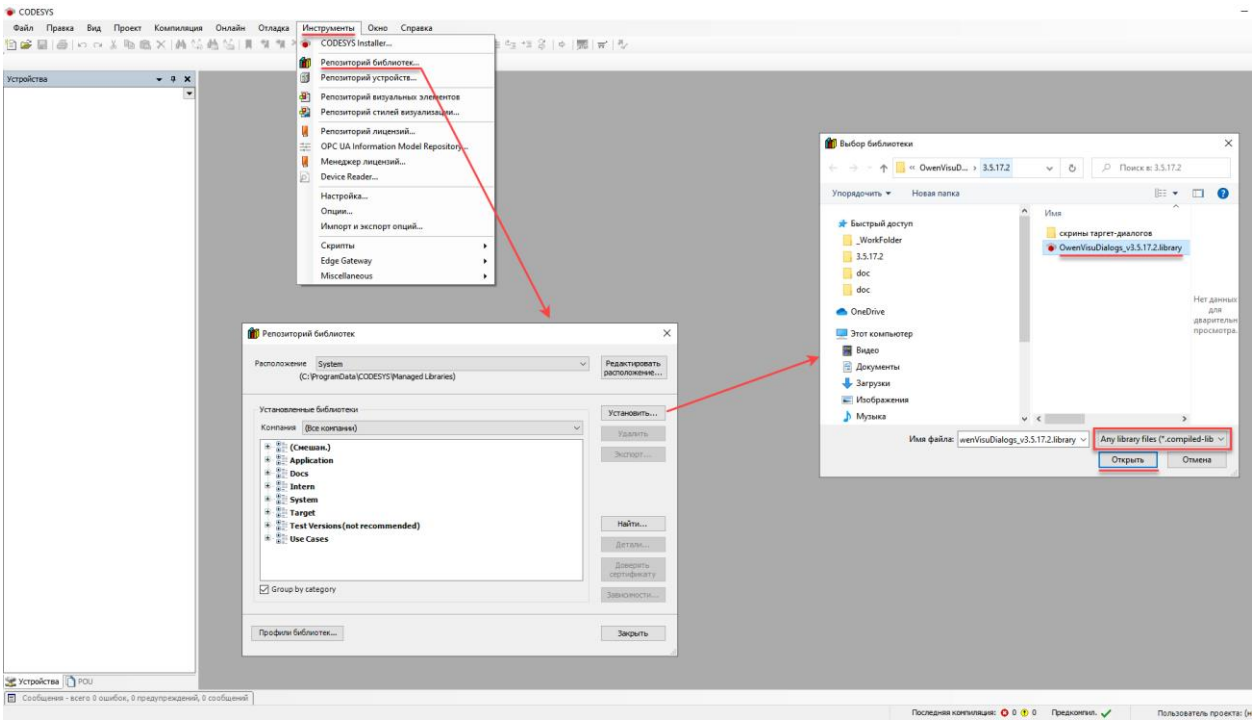


Рисунок 2.3.1 – Установка библиотеки в репозиторий библиотек

Подключение установленных библиотек к проекту CODESYS будет рассмотрено в п. 5.10.2.

3 Быстрый старт

В этом разделе мы создадим простейший проект и загрузим его в контроллер. Цель раздела – как можно быстрее запустить на контроллере готовое приложение, поэтому большинство аспектов создания проекта подробно не поясняются; вся необходимая информация будет приведена в следующих разделах.

3.1 Создание проекта

Для создания нового проекта необходимо выбрать в меню **Файл** пункт **Новый проект**. Если же вы хотите открыть существующий проект – то можно сделать это с помощью команды **Открыть проект** или пункта **Недавние проекты**.

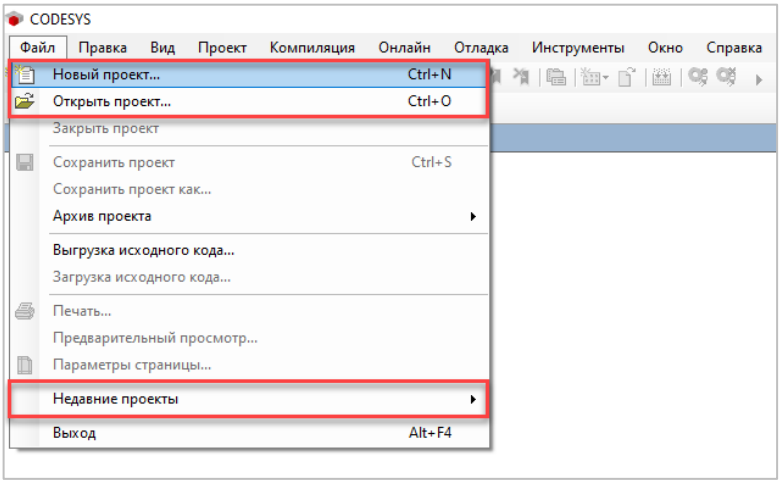


Рисунок 3.1.1 – Создание нового проекта в CODESYS

После этого откроется окно мастера создания проектов (см. рис. 3.1.2).

Если вы уже установили пакет таргет-файлов ОВЕН, то в этом окне отобразятся шаблоны проектов. Если еще нет – то самое время это сделать (см. п. 2.3).

Шаблоны включают в себя ряд преднастроенных компонентов – визуализацию, сетевые интерфейсы контроллера и т. д. Создавать проекты на их основе существенно проще, чем делать это «с нуля». В рамках примеров данного документа будет использоваться панельный контроллер СПК1xx – поэтому мы выберем его шаблон. Если вы работаете с другим контроллером ОВЕН (ПЛК210 или ПЛК200) – то выберите его шаблон. Если у вас в наличии нет реального контроллера, то можете выбрать любой шаблон – в п. 3.4 мы покажем, как работать с виртуальным контроллером.

В нижней части окна мастера происходит выбор имени файла проекта и директории, в которой он будет сохранен.

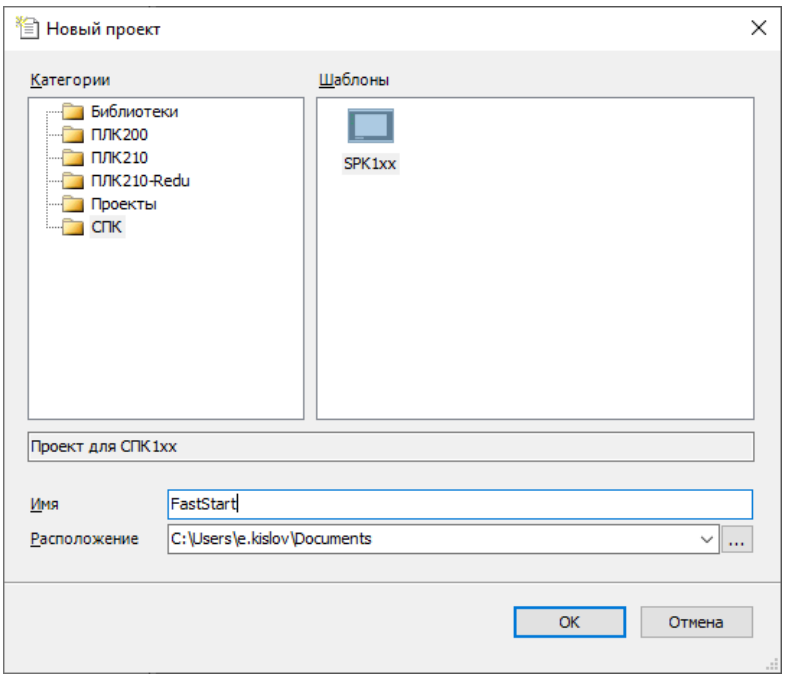


Рисунок 3.1.2 – Выбор шаблона проекта

Интерфейс CODESYS выглядит следующим образом:

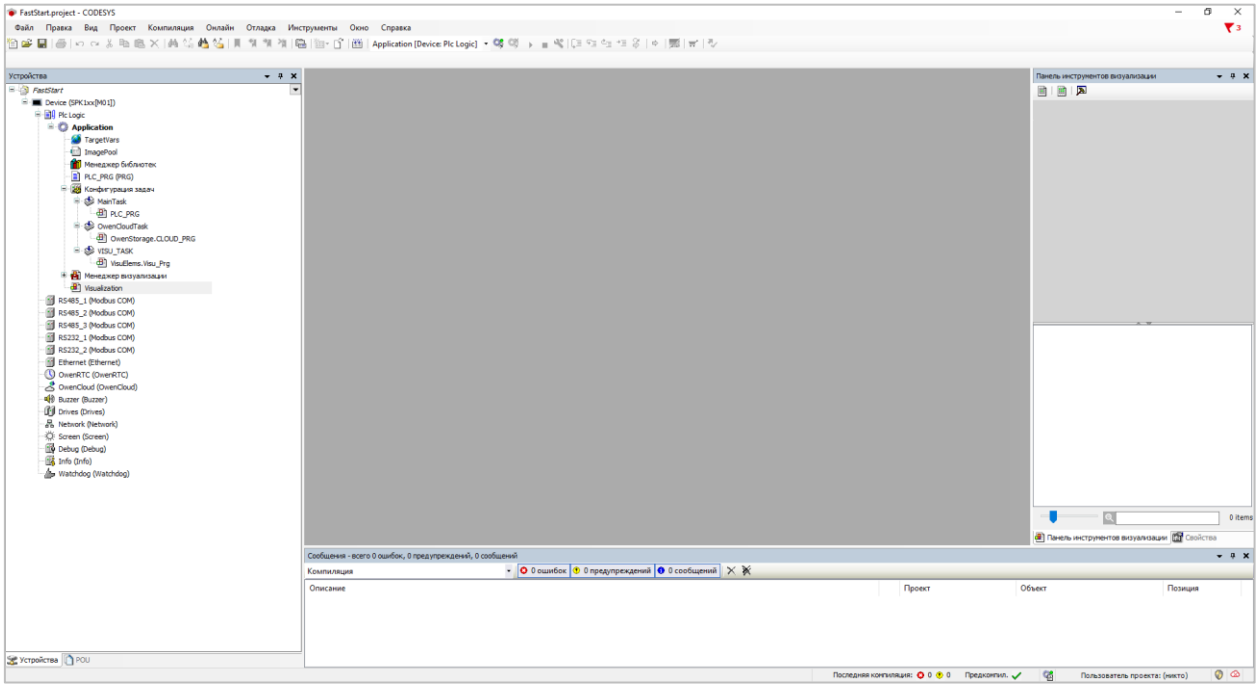


Рисунок 3.1.3 – Интерфейс среды CODESYS

В левой части окна расположено дерево проекта. В нем отображаются компоненты проекта – программы, функциональные блоки, экраны визуализации, узлы протоколов обмена и т. д. В центре отображается текущий открытый редактор – например, редактор языка программирования или редактор визуализации. Справа размещена панель инструментов открытого редактора – например, панель элементов языка CFC или панель элементов визуализации. Снизу отображаются различные панели – панель сообщений компиляции, панель мониторинга и т. д.

Давайте откроем редактор программы **PLC_PRG**. Для этого найдем программу в дереве проекта и два раза нажмем на нее левой кнопкой мыши. Поскольку в шаблоне проекта для этой программы выбран язык ST – то откроется редактор данного языка.

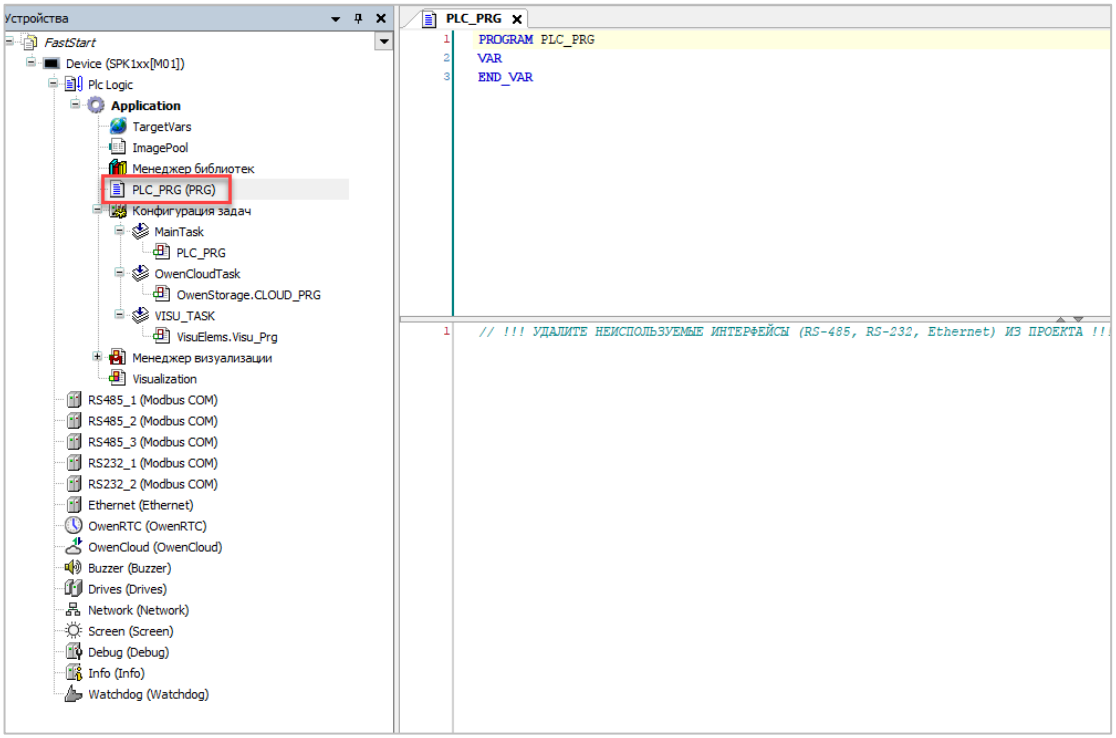


Рисунок 3.1.4 – Редактор языка ST

Окно редактора ST (как и других редакторов программирования в CODESYS) состоит из двух панелей – в верхней происходит объявление переменных, а в нижней – написание кода.

Объявление переменных происходит в рамках какой-либо **области** (например, области входов, выходов и т. д.). В нашей программе **PLC_PRG** присутствует единственная область, выделенная ключевыми словами **VAR** и **END_VAR** – это область локальных (внутренних) переменных программы. Объявление локальных переменных происходит между этими ключевыми словами.

Объявим две переменные:

- переменная **iValue** типа **INT** (целочисленный тип) будет представлять собой значение какого-то параметра (например, таким параметром может являться значение от датчика температуры, считанное с модуля ввода);
- переменная **xAlarm** типа **BOOL** (логический тип) будет являться сигналом тревоги, который сообщает о том, что значение **iValue** превысило допустимый предел (например, **50**).

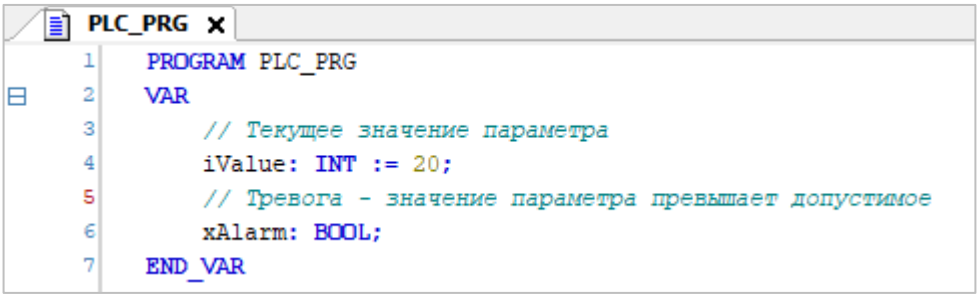


Рисунок 3.1.5 – Объявление переменных

При объявлении переменной обязательно нужно указать ее имя и тип данных. При необходимости можно указать начальное значение – в примере выше мы сделали это для переменной **iValue**. Имя переменной и ее тип разделяется двоеточием (:), а заканчивается строка объявления точкой с запятой (;). Вместе с объявлением переменной удобно добавить комментарий, поясняющий ее назначение. Однострочные комментарии в CODESYS начинаются с символа //.

Добавим в проект код программы – в примере он будет состоять всего из одной строки.

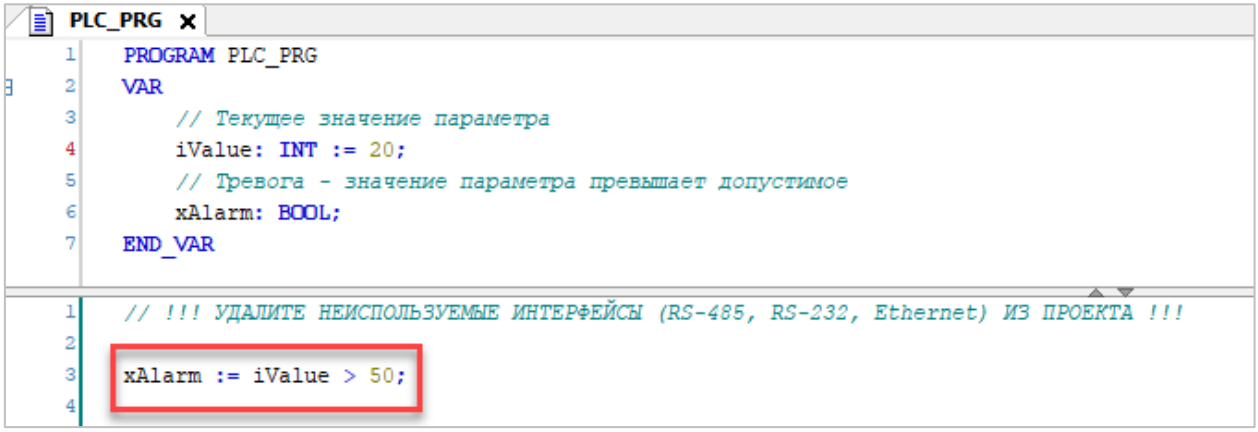


Рисунок 3.1.6 – Добавление кода программы

В этой строке происходит сравнение значения переменной **iValue** с числом **50**. Результат сравнения является логическим значением – если оцениваемое выражение истинно (то есть **iValue** больше **50**), то результатом будет значение **TRUE**, а если нет (то есть **iValue** меньше или равно **50**) – то **FALSE**. Результат сравнения записывается в переменную **xAlarm** с помощью оператора присваивания – в языке ST он обозначается как записанные рядом символы «двоеточие» и «равно» (пробелы между этими символами недопустимы).

В шаблоне проекта в первой строке программы **PLC_PRG** приведен комментарий с рекомендацией удалить из проекта неиспользуемые интерфейсы – иначе они будут бессмысленно расходовать ресурсы контроллера. В нашем примере интерфейсы вообще не используются – поэтому в дереве проекта выделим их все (для этого зажмите левую кнопку мыши и не отпуская обведите все узлы интерфейсов рамкой, после чего отпустите кнопку мыши), нажмем на любой из этих интерфейсов правой кнопкой мыши и выберем в появившемся контекстном меню команду **Удалить** (см. рис. 3.1.7).

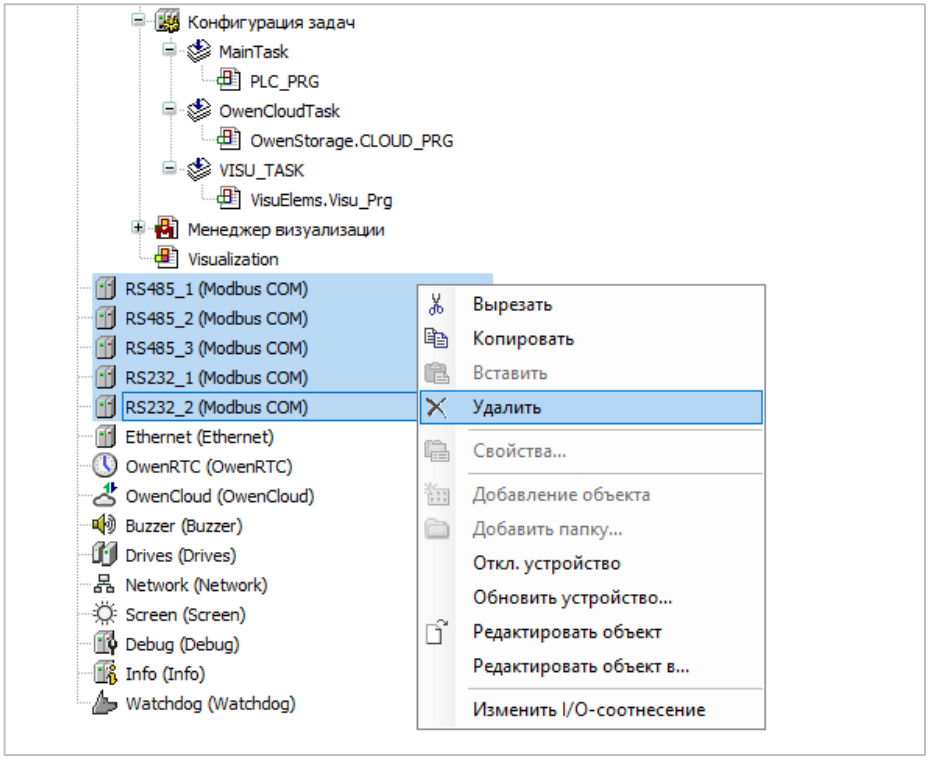


Рисунок 3.1.7 – Удаление неиспользуемых в примере интерфейсов из дерева проекта

Теперь приступим к созданию визуализации. В дереве проекта есть узел **Visualization**, представляющий собой одноименный экран визуализации. Два раза нажмем на него левой кнопкой мыши, чтобы перейти к редактированию.

Чтобы включить в редакторе визуализации сетку – перейдите в меню **Инструменты** и используйте команду **Опции – Визуализация – Сетка – Видимая**.

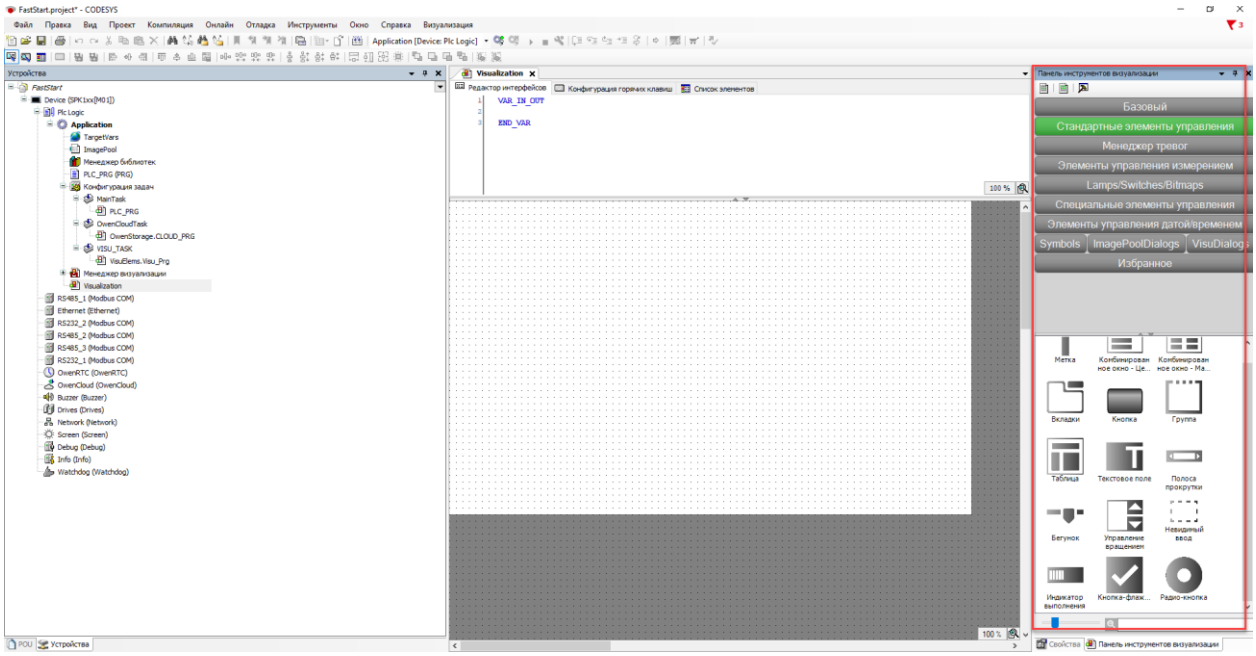


Рисунок 3.1.8 – Внешний вид редактора визуализации и панели элементов визуализации

В центре экрана откроется редактор визуализации, а справа – панель элементов визуализации. Элементы сгруппированы по вкладкам. Выберите вкладку **Стандартные элементы управления** и найдите элемент **Бегунок**. «Перетащите» его на экран визуализации при зажатой левой кнопке мыши или же выделите элемент на панели элементов и один раз нажмите на экран. Аналогичным образом добавьте элемент **Индикатор** из вкладки **Lamps/Switches/Bitmaps**.

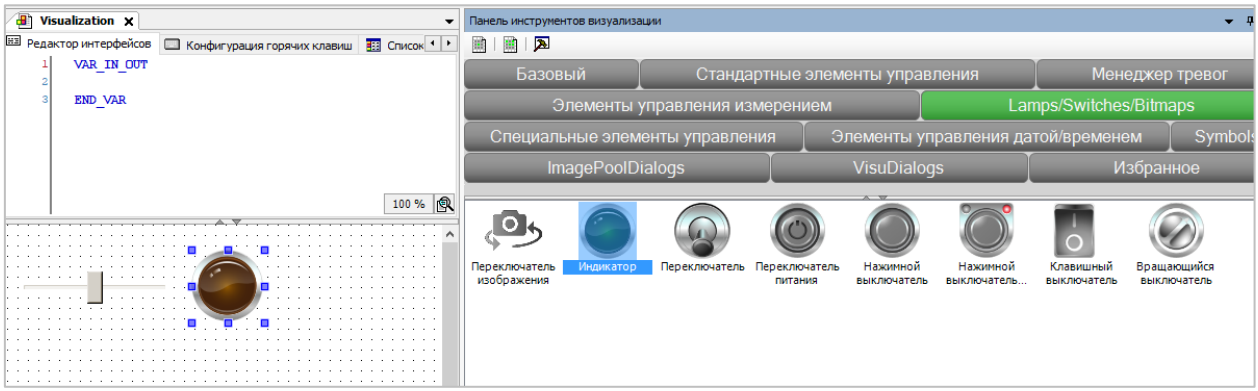


Рисунок 3.1.9 – Размещение элементов Бегунок и Индикатор на экране визуализации

Для перемещения элемента по экрану необходимо выделить его одиночным нажатием левой кнопкой мыши, затем зажать ее, переместить в нужную позицию и отпустить. Для изменения размеров элемента необходимо при зажатой левой кнопке мыши переместить одну из его опорных точек (на рис. 3.1.9 они выделены синим).

После выделения элемента в правой части экрана откроется окно его настроек. Давайте настроим элемент **Бегунок** – зададим ему размеры, включим шкалу делений и привяжем к нему переменную **iValue**. Для привязки переменной нужно два раза нажать левой кнопкой мыши на пустую строку рядом с параметром **Переменная**, после этого нажать на появившуюся справа от нее кнопку «…» и выбрать нужную переменную из списка переменных проекта.

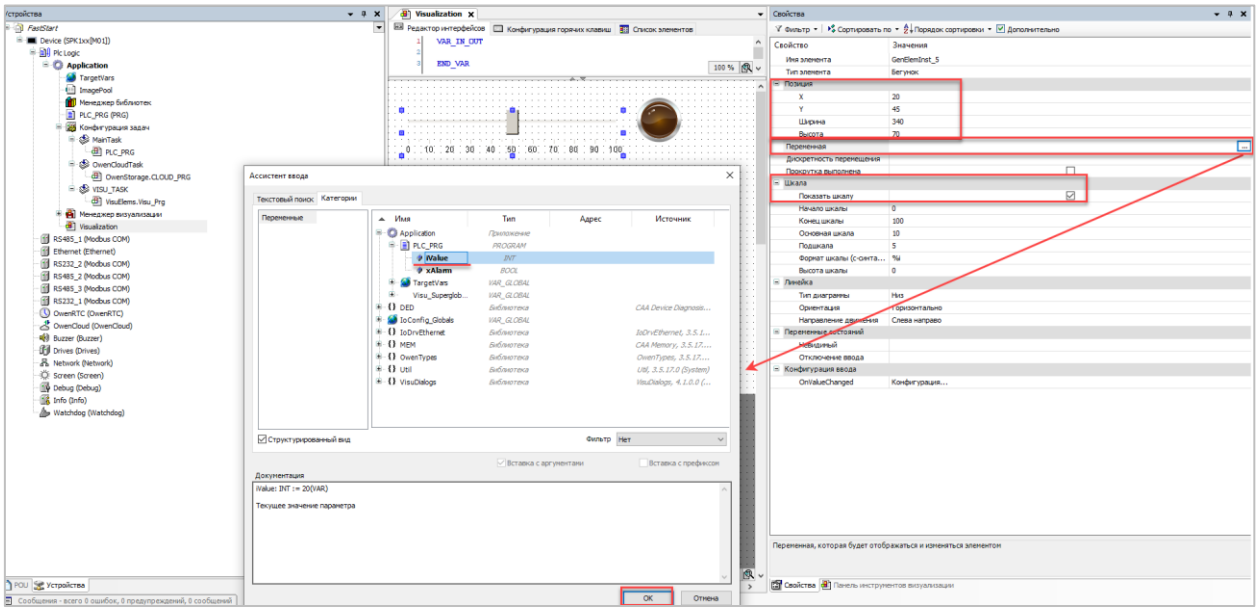


Рисунок 3.1.10 – Настройки элемента Бегунок

Аналогичным образом зададим положение и размеры элемента **Индикатор**, установим для него красный цвет и привяжем переменную **xAlarm**.

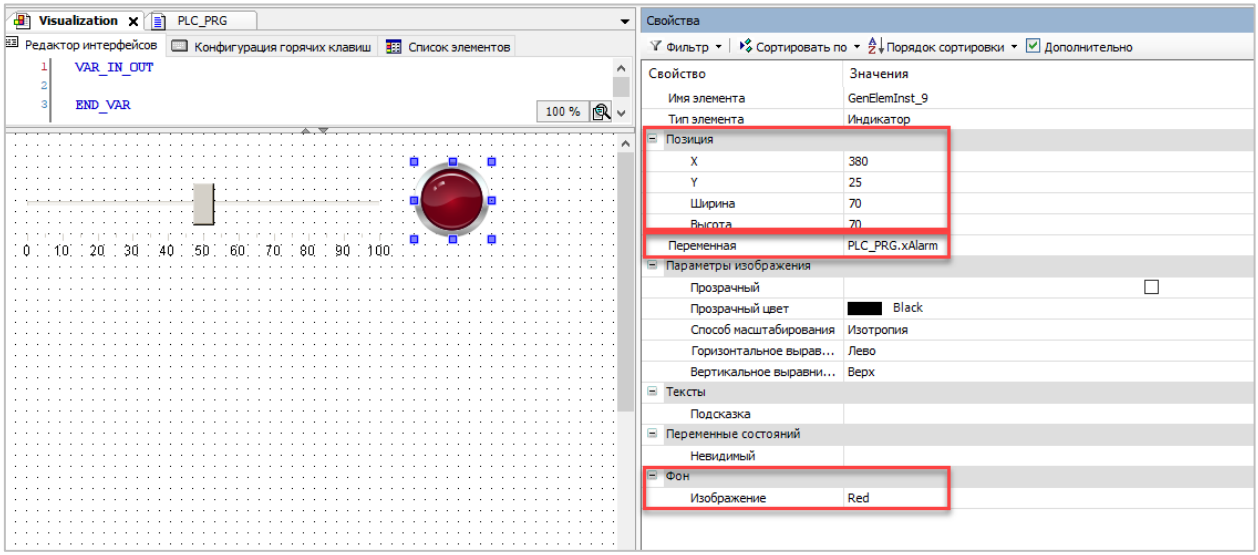


Рисунок 3.1.11 – Настройки элемента Индикатор

Наш простейший проект готов. Теперь нужно загрузить его в контроллер и убедиться, что он работает. Если у вас уже есть в наличии контроллер ОВЕН – то переходите к п. 3.2. Если же контроллера у вас еще нет, то переходите в п. 3.3 – в нем описано, как запустить проект на ПК с помощью виртуального контроллера **CODESYS Control Win V3**.

3.2 Установка связи с контроллером ОВЕН

Для контроллеров ОВЕН подключение из среды CODESYS возможно по интерфейсам Ethernet и USB. В следующих пунктах описана процедура подключения для обоих интерфейсов.

3.2.1 Подключение по интерфейсу Ethernet

Возможны два основных варианта взаимного сетевого расположения контроллера и компьютера:

1. Контроллер и компьютер находятся в **одной локальной сети** (частный случай этого варианта – контроллер подключен напрямую к ПК);
2. Контроллер и компьютер находятся в **разных локальных сетях**, связанных с помощью соответствующих сетевых устройств (маршрутизаторов).

Рассмотрим пример первого варианта подключения – когда порты Ethernet контроллера и компьютера соединяются напрямую с помощью кабеля RJ45-RJ45.

ПРИМЕЧАНИЕ
В локальных сетях не должно возникать конфликта IP-адресов, т. е. разные устройства не должны иметь совпадающие адреса.

Сначала следует определить сетевые параметры контроллера.

По умолчанию сетевые настройки интерфейсов Ethernet контроллера следующие:

Таблица 3.2.1 – Сетевые настройки интерфейса Ethernet по умолчанию

Контроллер	Интерфейс	Режим DHCP Client	IP-адрес	Маска подсети	IP-адрес шлюза
ПЛК210	Ethernet 1–3	Отключен	192.168.0.10	255.255.0.0	192.168.0.1
	Ethernet 4	Включен	-	-	-
ПЛК200	Ethernet 1	Отключен	192.168.0.10	255.255.0.0	192.168.0.1
	Ethernet 2	Включен	-	-	-
СПК1xx [M01]	Ethernet	Отключен	192.168.0.10	255.255.0.0	192.168.0.1

В качестве примера будет рассмотрен процесс подключения к компьютеру контроллера с сетевыми настройками по умолчанию. Для подключения следует использовать интерфейс со статическим IP-адресом; режим DHCP-клиента используется для автоматического получения сетевых настроек при подключении контроллера к локальной сети, в которой есть DHCP-сервер – поскольку мы подключаем контроллер напрямую к ПК, то в большинстве случаев такой вариант не подойдет.

Чтобы изменить сетевые настройки контроллера следует подключиться к web-конфигуратору (*если вы работаете с панельным контроллером СПК – то можно вместо этого настроить контроллер через экранный конфигуратор*). Для запуска web-конфигуратора подключите контроллер к ПК по интерфейсу Ethernet и в web-браузере введите IP-адрес интерфейса. На открывшейся странице аутентификации (см. рис. 3.2.1) введите имя пользователя **root** и пароль (пароль по умолчанию – **owen**).

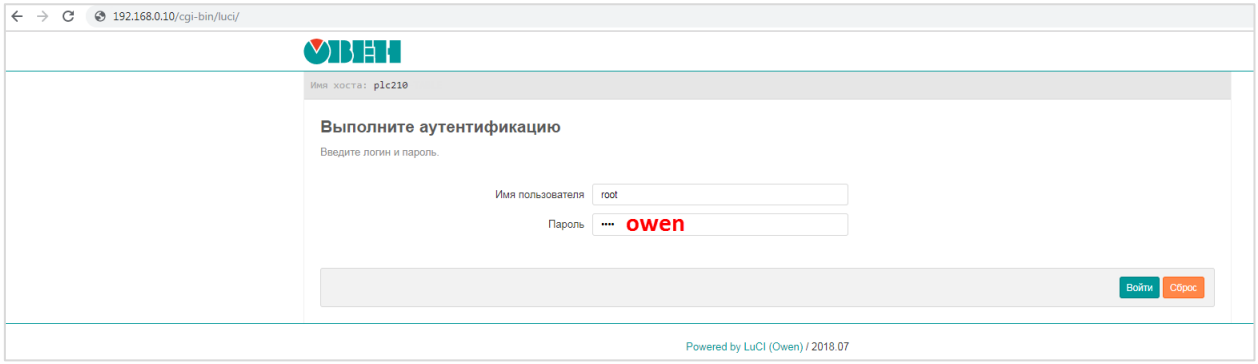


Рисунок 3.2.1 – Страница аутентификации в web-конфигураторе

При первом подключении к web-конфигуратору контроллера ПЛК2xx будет запущен **Мастер настройки**, с помощью которого можно задать сетевые настройки контроллера. В случае необходимости мастер настройки можно запустить повторно на вкладке **Система/Мастер настройки**. Также сетевые настройки можно задать на вкладке **Сеть/Интерфейсы**. На этой вкладке следует выбрать нужный интерфейс и нажать кнопку **Изменить**.

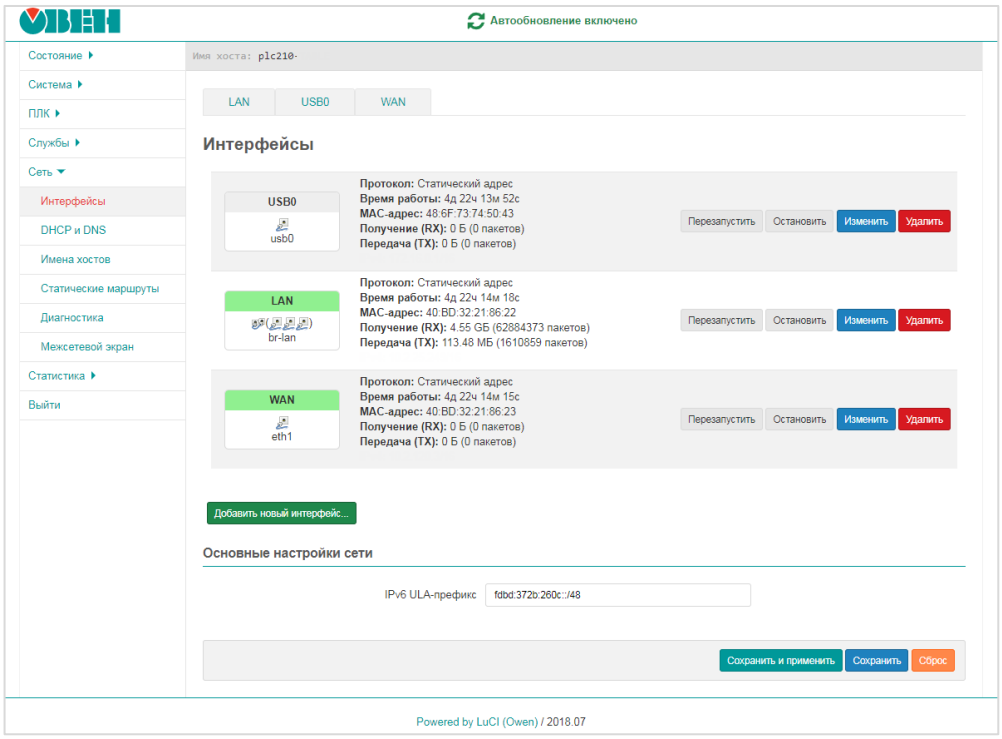


Рисунок 3.2.2 – Список интерфейсов контроллера

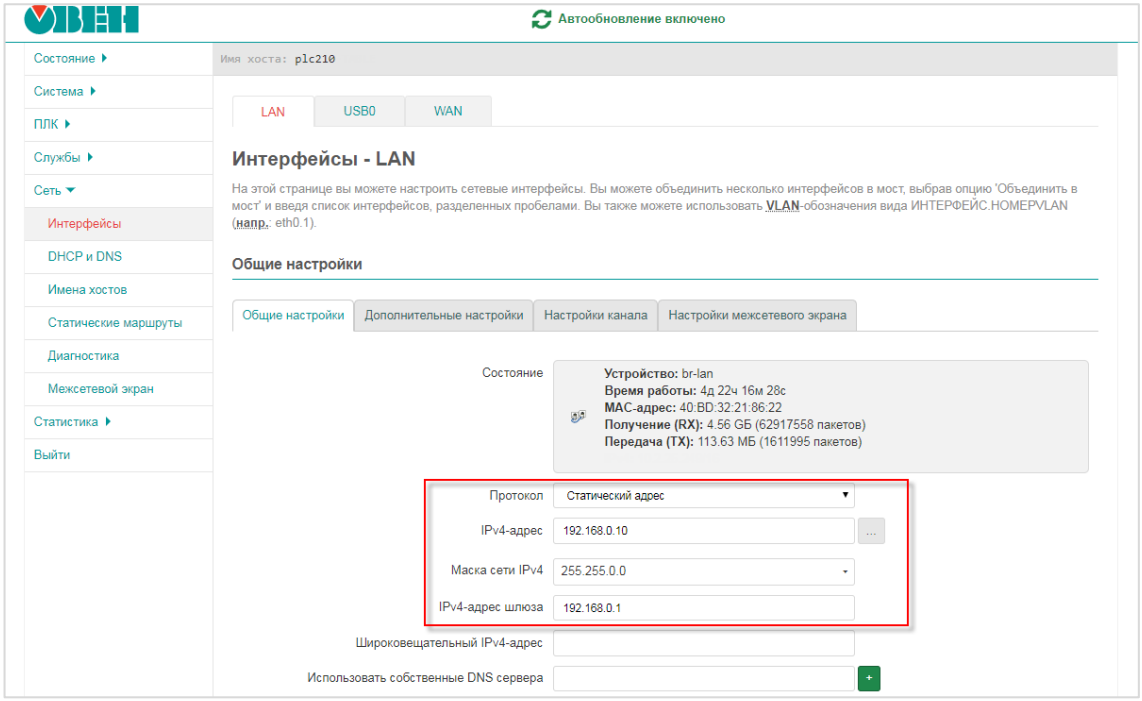


Рисунок 3.2.3 – Сетевые настройки интерфейса

Для применения новых сетевых настроек следует нажать кнопку **Сохранить и применить** (Сохранить и применить). В рамках примера мы оставим сетевые настройки в значениях по умолчанию.

ПРИМЕЧАНИЕ
После изменения сетевых настроек может потребоваться переподключение к web-конфигуратору по новым настройкам для их применения.

ПРИМЕЧАНИЕ
Подробное описание web-конфигуратора приведено в документе **Краткое описание основных функций Web-интерфейса управления контроллеров ОВЕН**

ПРИМЕЧАНИЕ
Описание экранного конфигуратора контроллеров СПК приведено в документе **Экранный конфигуратор СПК1xx. Руководство пользователя**.

Для настройки сетевых параметров ПК следует открыть Центр управления сетями и общим доступом (**Пуск — Панель управления — Центр управления сетями и общим доступом**) и выбрать вкладку **Изменение параметров адаптера**.

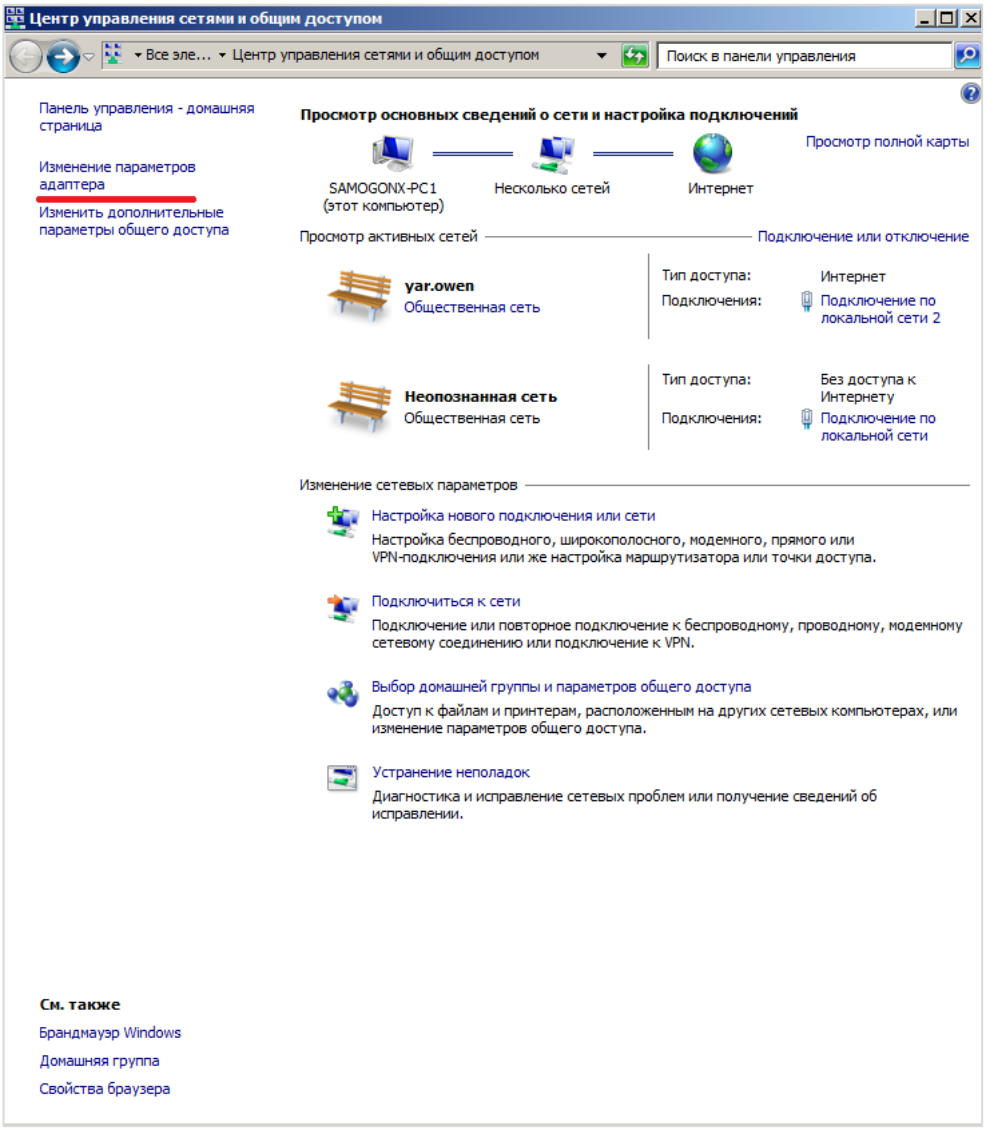


Рисунок 3.2.4 – Окно Центра управления сетями и общим доступом

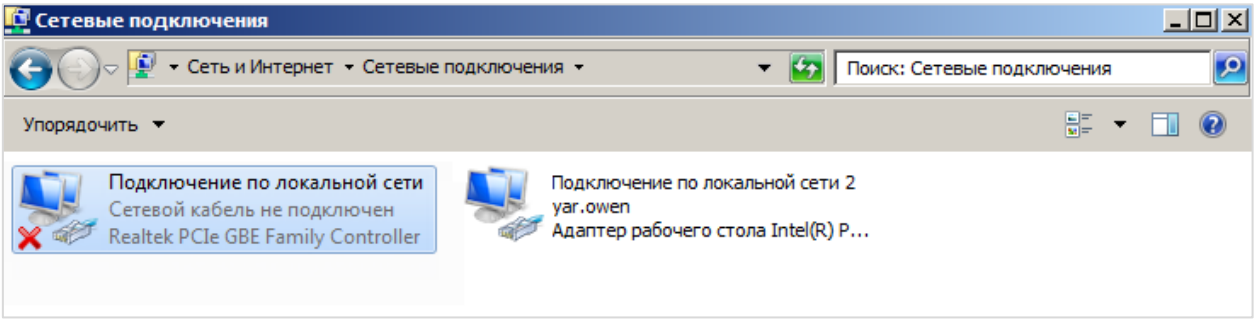


Рисунок 3.2.5 – Вкладка Изменение параметров адаптера (Сетевые подключения)

Затем следует подсоединить один конец Ethernet-кабеля к порту Ethernet **включенного** контроллера, а другой конец – к аналогичному порту ПК. Один из сетевых адаптеров станет «активным» (с его пиктограммы пропадет красный крестик).

Затем следует нажать на «активный» адаптер правой кнопкой мыши и открыть вкладку **Свойства**, выбрать компонент **Протокол Интернета версии 4 (TCP/IPv4)** и открыть его **Свойства**:

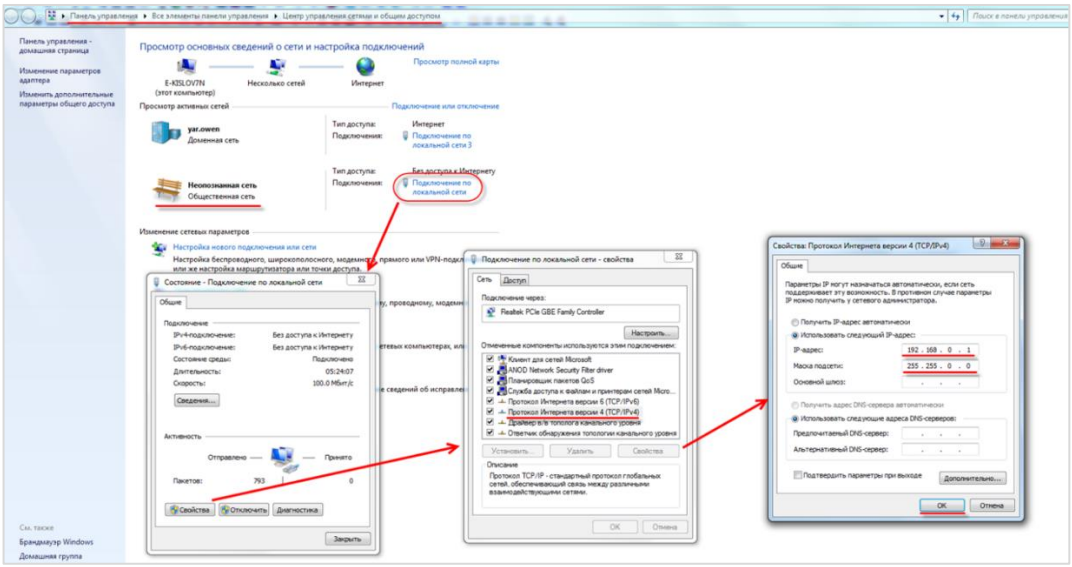


Рисунок 3.2.6 – Установка сетевых настроек ПК

Для ПК в рамках примера задается IP-адрес **192.168.0.1** и маска подсети **255.255.0.0**. Поля остальных настроек остаются пустыми. После нажатия кнопки **ОК** адаптеру потребуется несколько секунд, чтобы применить новые настройки.

Чтобы проверить наличие связи между ПК и контроллером следует открыть **командную строку** (**Пуск** — **Все программы** — **Служебные** — **Командная строка**) и ввести команду

ping 192.168.0.10 -t

Если связь есть, то результат будет следующим:

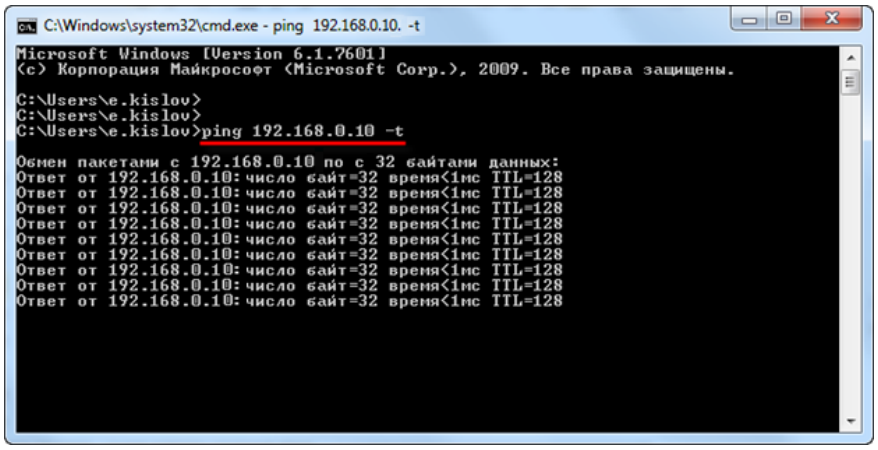


Рисунок 3.2.7 – Результат успешного выполнения команды ping

Закройте окно пинга. После этого можно будет подключиться к контроллеру из среды CODESYS (см. п. 3.3).

3.2.2 Подключение по интерфейсу USB

Для связи контроллера и ПК по USB используется кабель **USB A–B** (для панельных контроллеров СПК) или **MicroUSB** (для контроллеров ПЛК2xx).

По умолчанию сетевые настройки интерфейса USB контроллера следующие:

Таблица 3.2.2 – Сетевые настройки интерфейса USB по умолчанию

Контроллер	Режим DHCP Server	IP-адрес	Маска подсети	IP-адрес шлюза
ПЛК2xx	Включен	172.16.0.1	-	-
СПК1xx [M01]	Включен	10.0.6.10	255.255.255.248	-

Сетевые настройки интерфейса USB могут быть изменены в web-конфигураторе контроллера (*если вы работаете с панельным контроллером СПК – то можно вместо этого настроить контроллер через экранный конфигуратор*) – но в большинстве случаев это не требуется.

Перед первым подключением контроллера к ПК следует выполнить установку драйвера USB. Для этого следует запустить установщик (**SPK1xx. Драйвер USB. Версия 1.5.102.exe),** который можно загрузить на сайте ОВЕН в разделе CODESYS V3/Сервисное ПО. Перед установкой может возникнуть следующее сообщение:

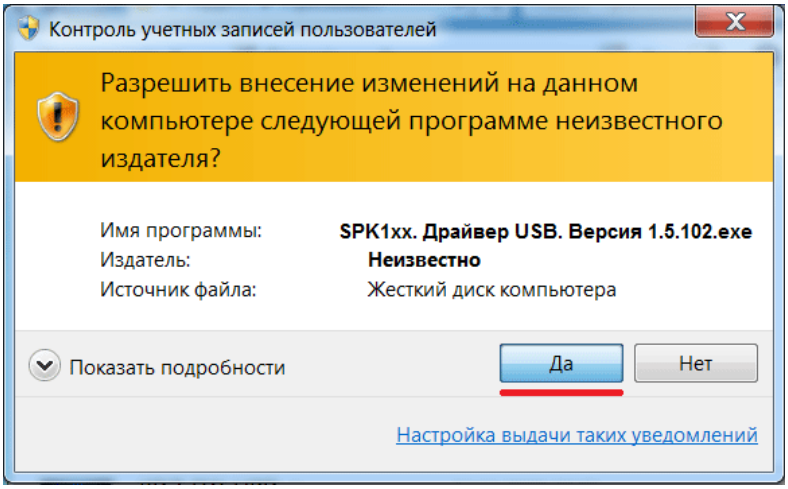


Рисунок 3.2.8 – Информационное сообщение перед установкой драйвера

Следует нажать **Да**.

ПРИМЕЧАНИЕ
Для подключения контроллеров СПК и ПЛК2xx используется один и тот же драйвер USB с названием Owen SPK.

Далее последовательно будут появляться диалоговые окна **Мастера установки драйверов** и **Мастера установки драйверов устройств**, информационное сообщение **брандмауэра** Windows (если он включен) и диалоговые окна завершения установки:

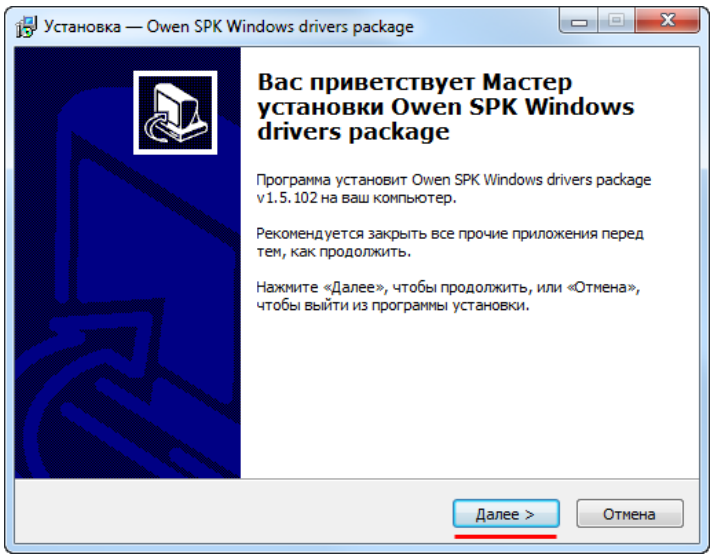


Рисунок 3.2.9 – Диалоговое окно Мастера установки драйверов

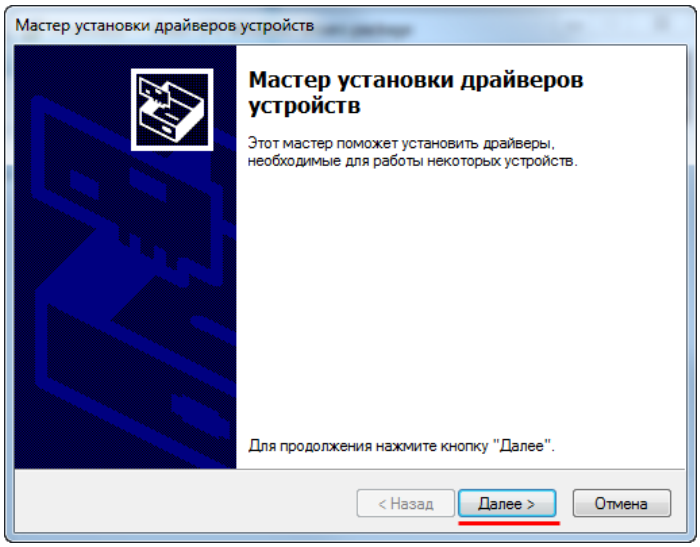


Рисунок 3.2.10 – Диалоговое окно Мастера установки драйверов устройств

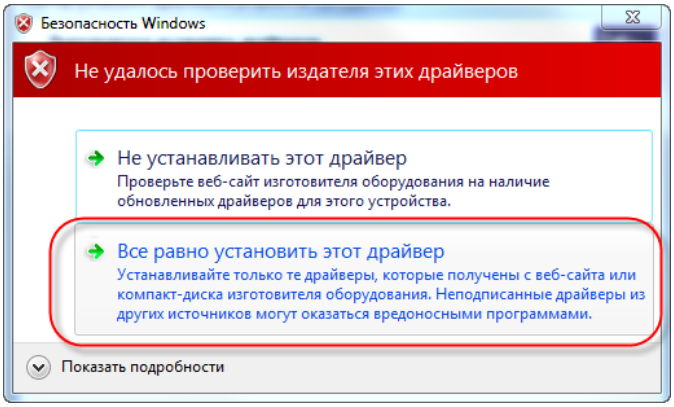


Рисунок 3.2.11 – Информационное сообщение брандмауэра Windows

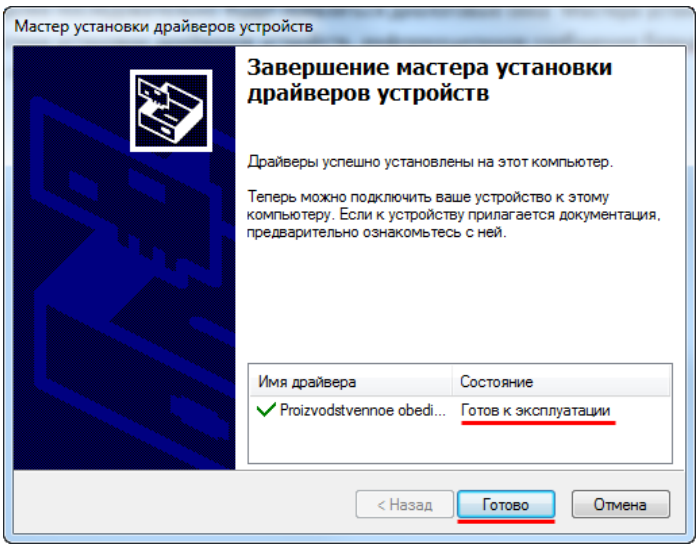


Рисунок 3.2.12 – Диалоговое окно завершения установки драйверов устройств

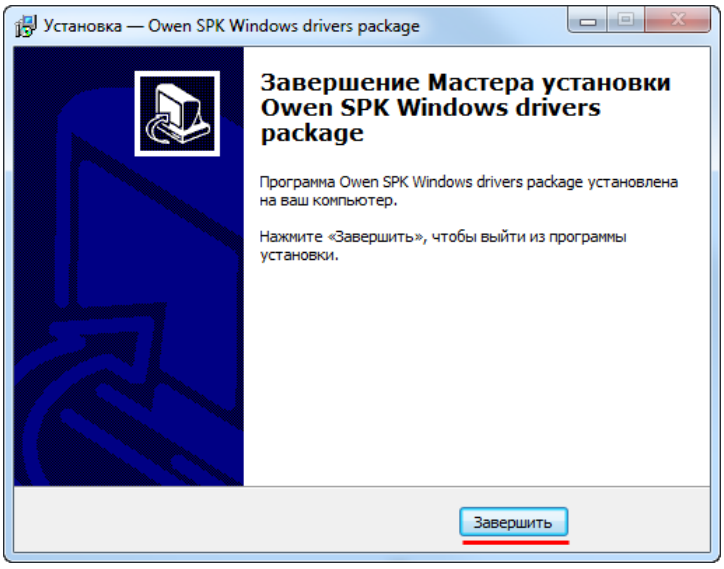


Рисунок 3.2.13 – Диалоговое окно завершения установки драйверов

Затем следует подключить контроллер к ПК с помощью соответствующего кабеля. В диспетчере устройств (**Пуск – Панель управления – Диспетчер устройств**) появится новый сетевой адаптер – **Owen SPK**.

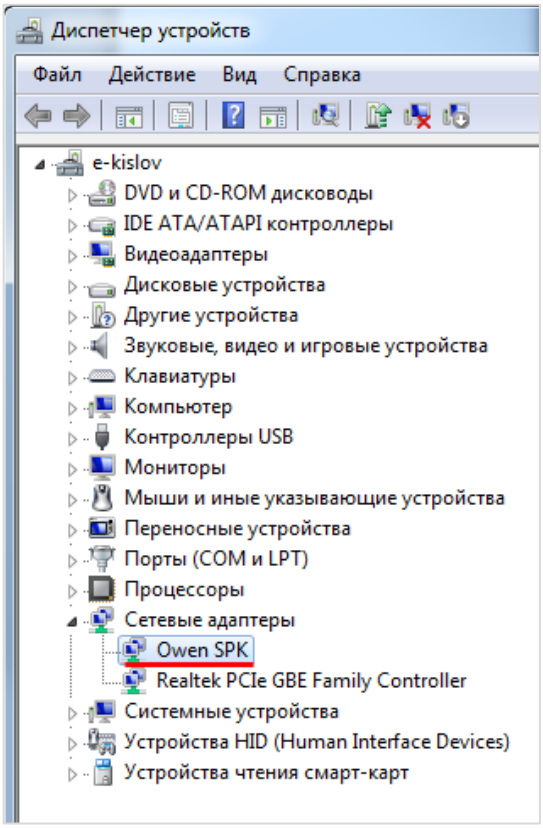


Рисунок 3.2.14 – Окно диспетчера устройств после подключения контроллера по USB

В меню **Изменение параметров адаптера** (**Пуск — Панель управления — Центр управления сетями и общим доступом – Изменение параметров адаптера**) появится новый виртуальный сетевой адаптер:

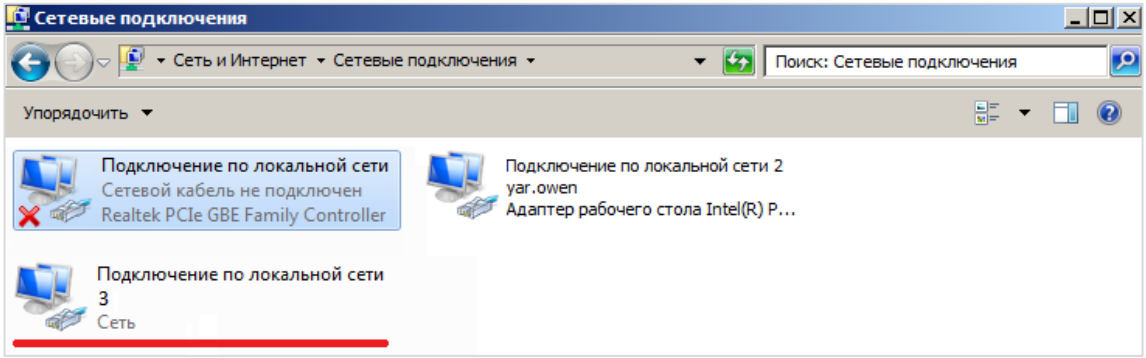


Рисунок 3.2.15 – Виртуальный сетевой адаптер для контроллеров ОВЕН

Настройки сетевого адаптера для подключения к СПК:

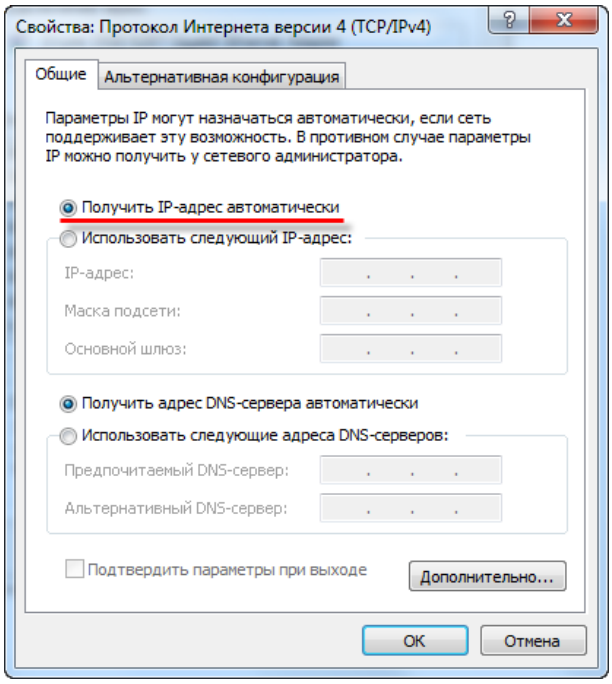


Рисунок 3.2.16 – Настройки виртуального сетевого адаптера для подключения к СПК

Так как интерфейс USB у контроллера СПК1xx [M01] работает в режиме DHCP Server, то сетевой адаптер ПК автоматически получит нужные настройки.

Чтобы проверить наличие связи между ПК и контроллером, следует открыть **командную строку** (**Пуск** — **Все программы** — **Служебные** — **Командная строка**) и ввести команду:

ping 10.0.6.10 -t

Если связь есть, то результат будет следующим:

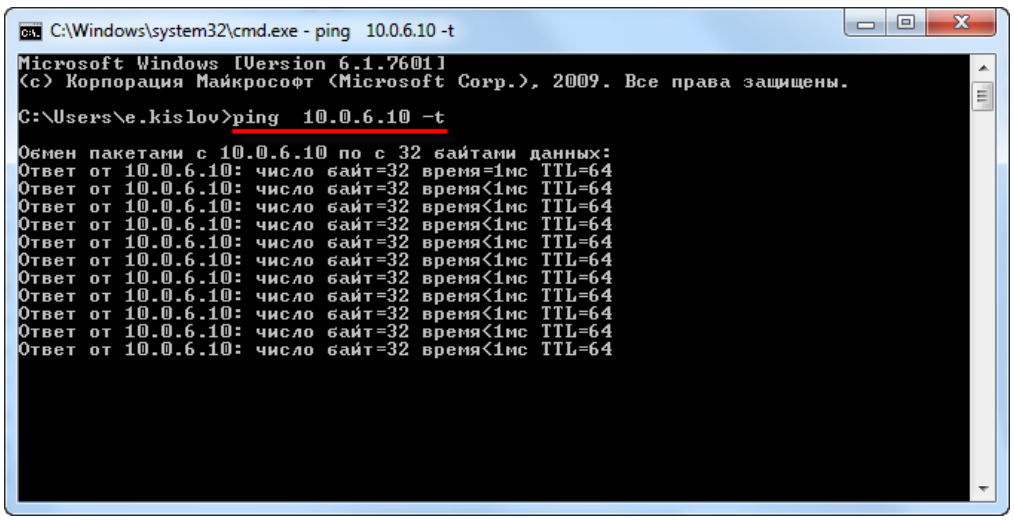


Рисунок 3.2.17 – Результат успешного выполнения команды ping

Закройте окно пинга. После этого можно будет подключиться к контроллеру из среды CODESYS (см. п. 3.3).

Настройки сетевого адаптера для подключения к ПЛК2xx:

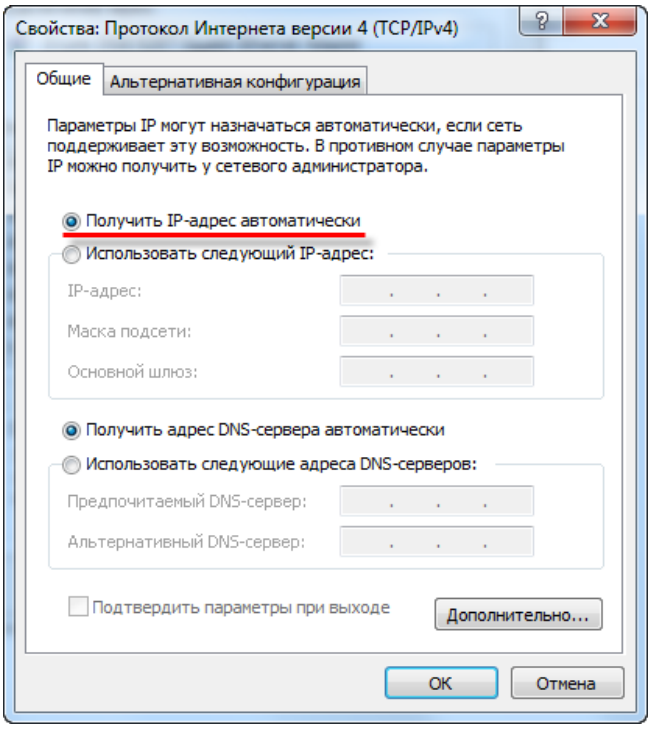


Рисунок 3.2.18 – Настройки виртуального сетевого адаптера для подключения к ПЛК

Так как интерфейс USB у контроллера ПЛК2xx работает в режиме DHCP Server, то сетевой адаптер ПК автоматически получит нужные настройки.

Чтобы проверить наличие связи между ПК и контроллером, следует открыть **командную строку** (**Пуск** — **Все программы** — **Служебные** — **Командная строка**) и ввести команду:

ping 172.16.0.1 -t

Если связь есть, то результат будет следующим:

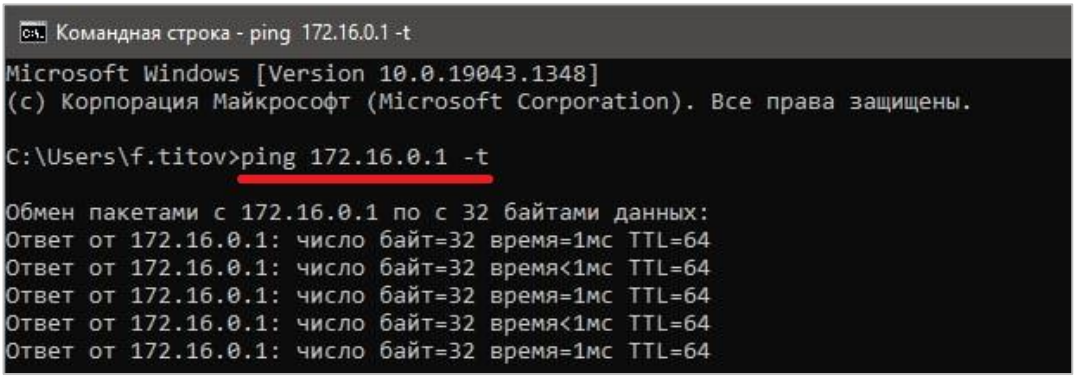


Рисунок 3.2.19 – Результат успешного выполнения команды ping

Закройте окно пинга. После этого можно будет подключиться к контроллеру из среды CODESYS (см. п. 3.3).

3.3 Подключение к контроллеру ОВЕН в среде CODESYS

После настройки сетевых параметров контроллера и компьютера следует установить связь между ними в среде CODESYS.

Узел **Device** (который определяется соответствующим таргет-файлом) должен соответствовать модели подключаемого контроллера. Если был выбран неподходящий тип контроллера – то его можно изменить, нажав в дереве проекта правой кнопкой мыши на узел **Device** и использовав команду **Обновить устройство**:

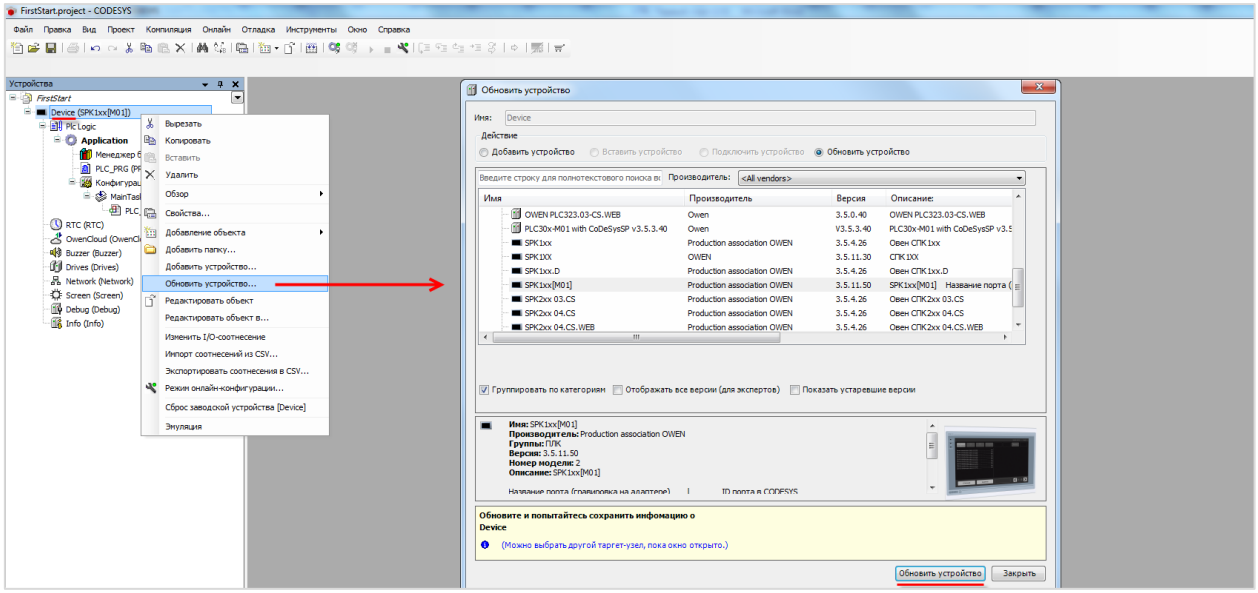


Рисунок 3.3.1 – Внешний вид окна обновления устройства

Затем следует выбрать из списка нужный таргет-файл. Напомним, что его версия (как и версия среды CODESYS) должна соответствовать версии прошивки контроллера (если вы не знаете эти версии – то используйте «выбиратор» на сайте ОВЕН). По умолчанию отображаются только самые новые версии установленных таргет-файлов. Для отображения всех доступных версий установите галочку **Отображать все версии**. После выбора таргет-файла следует нажать кнопку **Обновить устройство** и закрыть окно. В дереве проекта у узла **Device** отобразится название выбранного контроллера.

Затем следует настроить **Gateway** (шлюз). Настройка производится однократно – обычно она требуется только в том случае, если на этом ПК раньше не было установлено ни одной версии CODESYS.

Два раза нажмите левой кнопкой мыши на узел **Device** и перейдите на вкладку **Установки соединения**. Нажмите на кнопку **Gateway** и выберите пункт **Добавить gateway**:

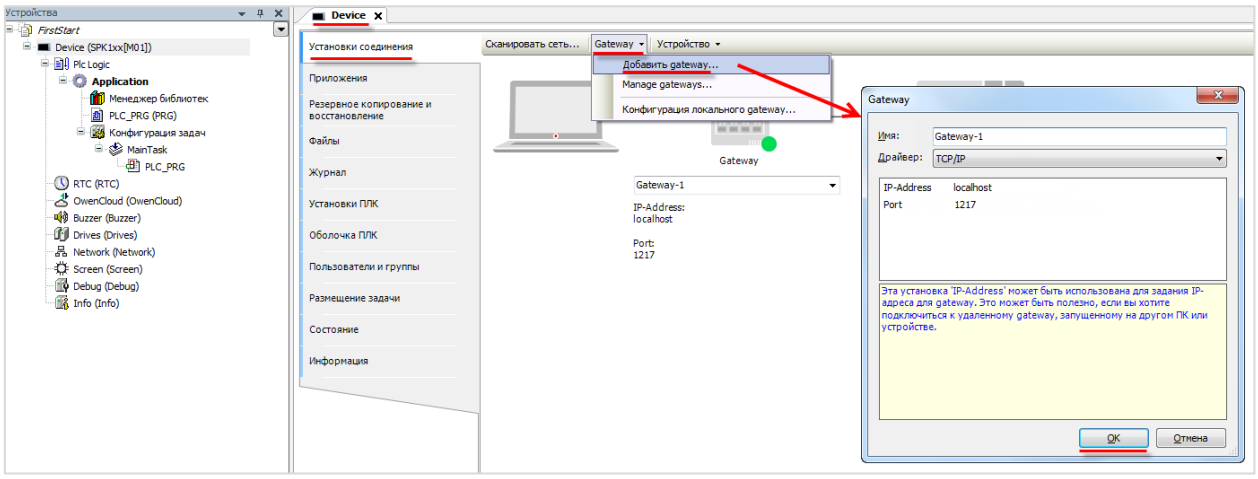


Рисунок 3.3.2 – Создание нового gateway (шлюза)

Настройки рекомендуется оставить по умолчанию (имя – **Gateway-1**, IP-адрес – **localhost**). Закройте окно настроек шлюза и нажмите кнопку **Сканировать сеть**. В появившемся списке следует выбрать нужный контроллер и установить связь, нажав кнопку **ОК**.

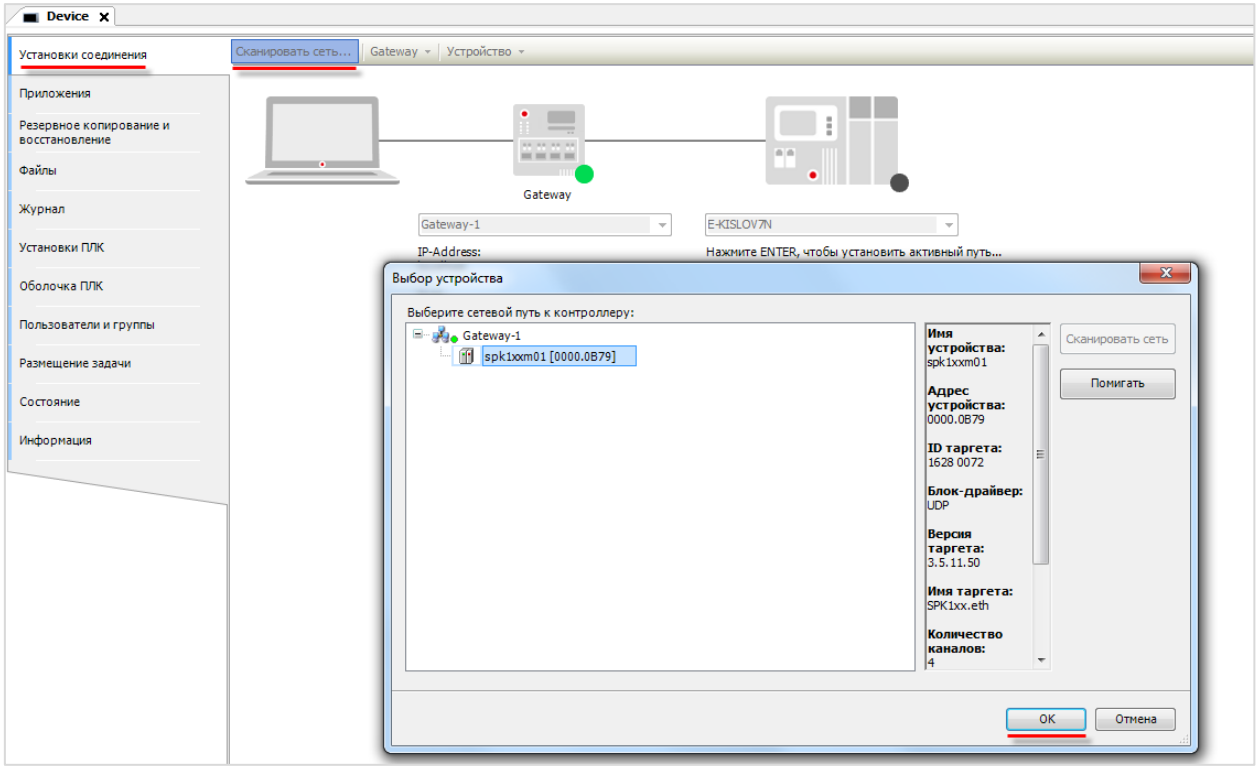


Рисунок 3.3.4 – Окно сканирования сети

В случае успешной установки связи индикаторы шлюза и контроллера загорятся зеленым:

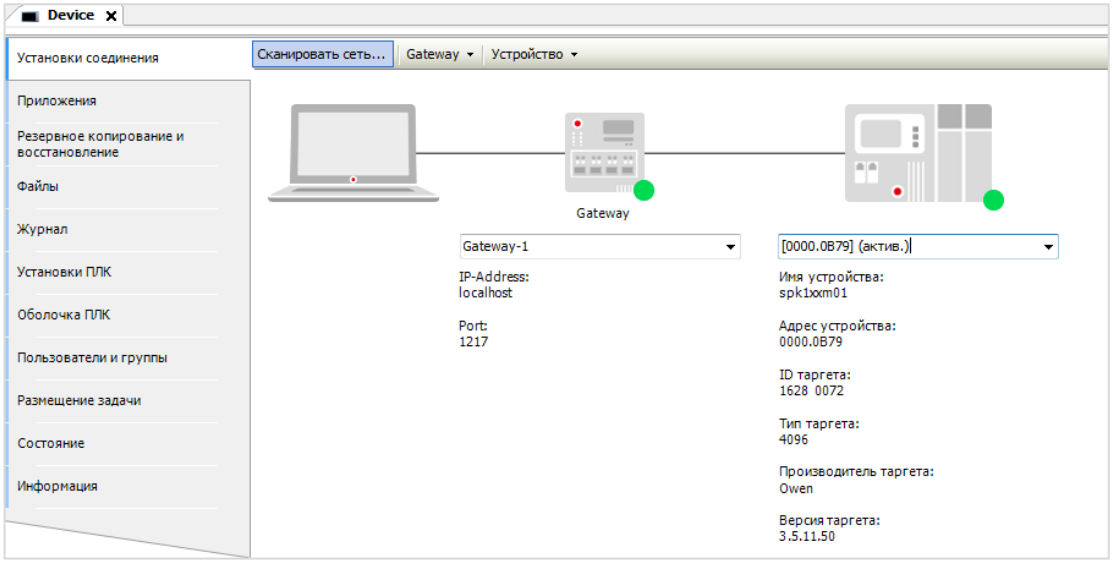


Рисунок 3.3.5 – Результат успешной установки связи

В ряде случаев (например, на ПК запущен антивирус, который блокирует рассылку широковещательных UDP пакетов или получение пакетов по портам 1740-1743) контроллер может не определиться во время сканирования сети. Тогда следует ввести IP-адрес контроллера вручную и нажать **Enter**.

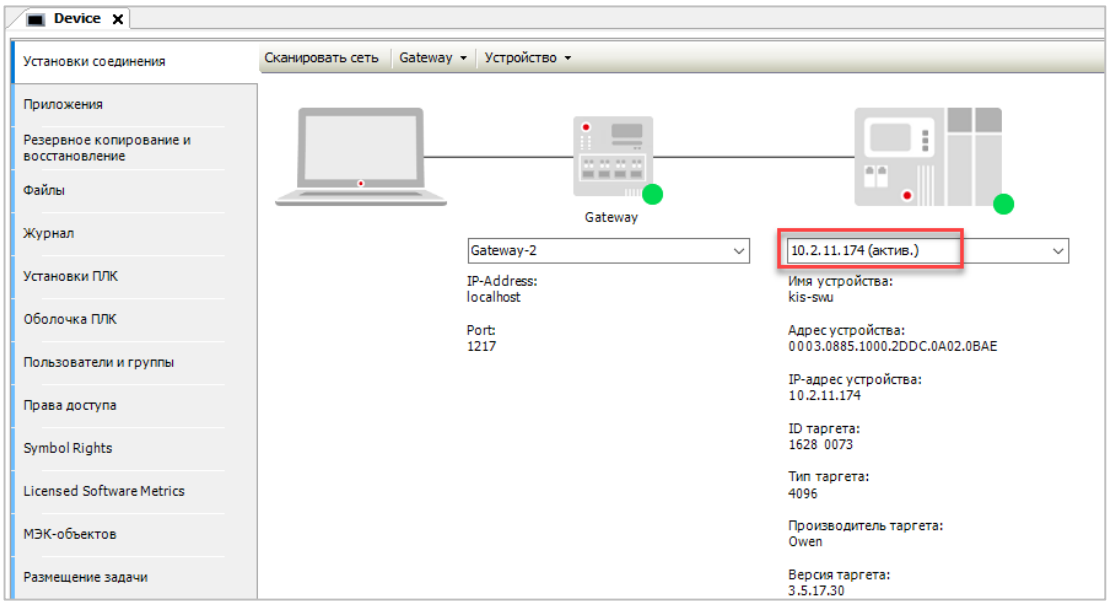


Рисунок 3.3.6 – Ввод IP-адреса контроллера для подключения без сканирования сети

ПРИМЕЧАНИЕ
Подключение к контроллеру СПК возможно только в том случае, если не запущен экранный конфигуратор.

После успешного подключения можно переходить к загрузке проекта – см. п. 3.5. Если же реального контроллера у вас еще нет, то можно использовать виртуальный контроллер – об этом мы расскажем в следующем пункте.

3.4 Подключение к виртуальному контроллеру в среде CODESYS

Виртуальный контроллер является полноценной системой исполнения, запускаемой на ПК. В состав дистрибутива CODESYS входит виртуальный контроллер **CODESYS Control Win V3** с ограничением на время непрерывной работы (2 часа). После истечения этого времени виртуальный контроллер можно перезапустить.

Виртуальный контроллер позволяет проверить все основные функции проекта – в том числе, web-визуализацию, обмен с другими устройствами и работу с файлами. Не получится протестировать только специфичный для конкретного контроллера функционал – например, работу со входами-выходами и внешние библиотеки (например, библиотеку **CmpSysExec**, которая используется для вызова утилит операционной системы контроллера).

Для запуска виртуального контроллера нужно раскрыть системный трей Windows, нажать на пиктограмму виртуального контроллера правой кнопкой мыши и выбрать команду **Start PLC**. Если вы запускаете виртуальный контроллер первый раз – появится информационное окно. Нажмите в нем кнопку **ОК**.

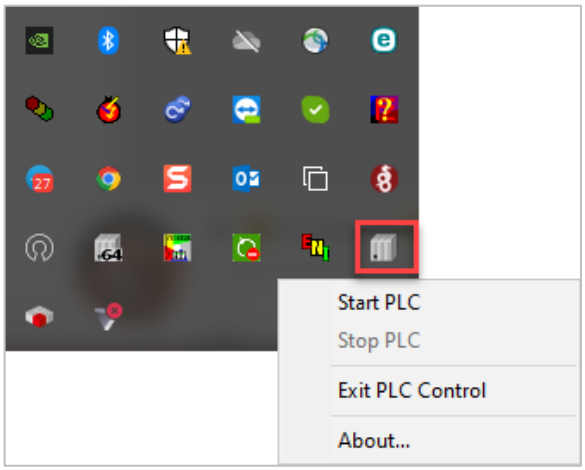


Рисунок 3.4.1 – Запуск виртуального контроллера

Далее нужно заменить таргет-файл контроллера в проекте на таргет-файл виртуального контроллера (см. рис 3.4.2). Для этого нажмите правой кнопкой мыши на узел **Device**, используйте команду **Обновить устройство** и выберите таргет-файл **CODESYS Control Win V3**. Версия таргет-файла должна соответствовать версии среды CODESYS (например, для среды **V3.5 SP17 Patch 3** используйте версию **3.5.17.30**; в случае необходимости установите галочку **Отображать все версии**).

ПРИМЕЧАНИЕ
Если вы установили 64-битную версию среды (на сайте ОВЕН такие версии не выложены), то вам нужно выбрать таргет-файл **CODESYS Control Win V3 x64**.

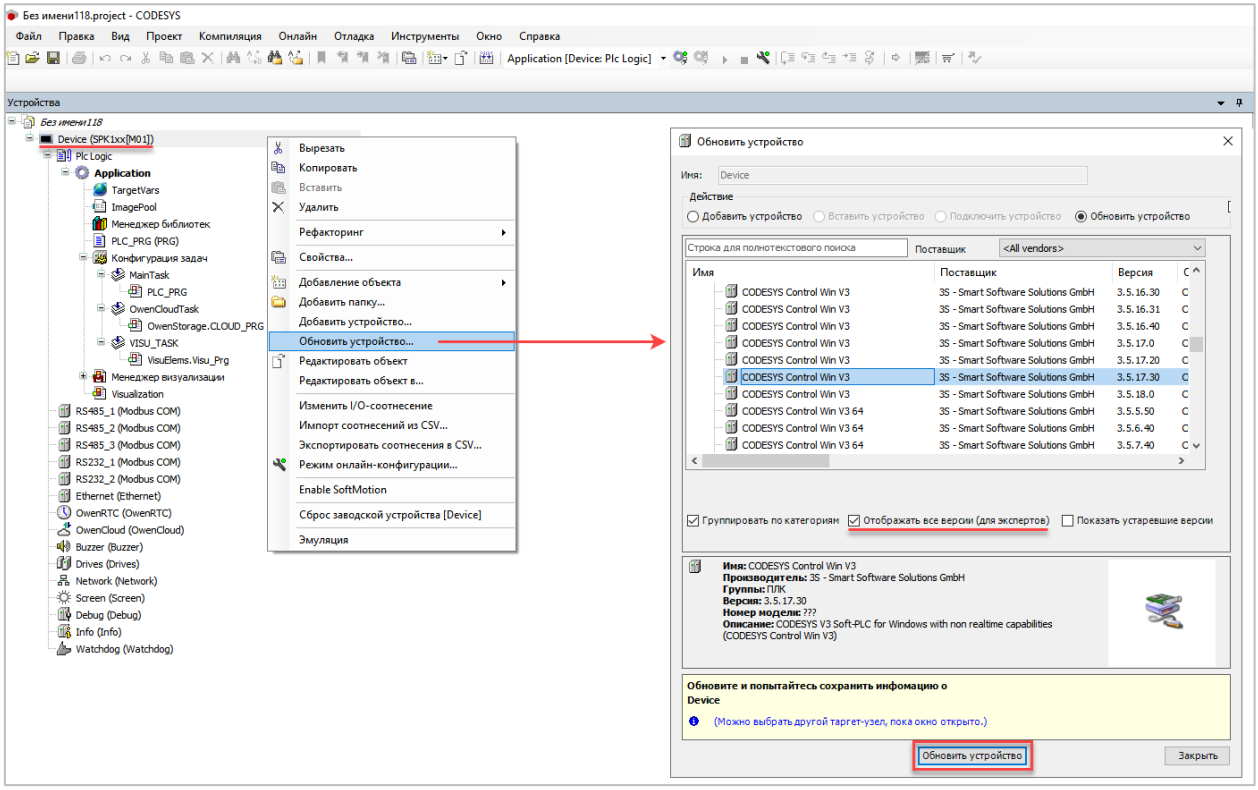


Рисунок 3.4.2 – Выбор таргет-файла виртуального контроллера

Если ваш проект был изначально создан для контроллера ОВЕН, то удалите из дерева проекта задачу **OwenCloudTask** – иначе вы не сможете загрузить проект в виртуальный контроллер.

Затем следует настроить **Gateway** (шлюз). Настройка производится однократно – обычно она требуется только в том случае, если на этом ПК раньше не было установлено ни одной версии CODESYS.

Два раза нажмите левой кнопкой мыши на узел **Device** и перейдите на вкладку **Установки соединения**. Нажмите на кнопку **Gateway** и выберите пункт **Добавить gateway**:

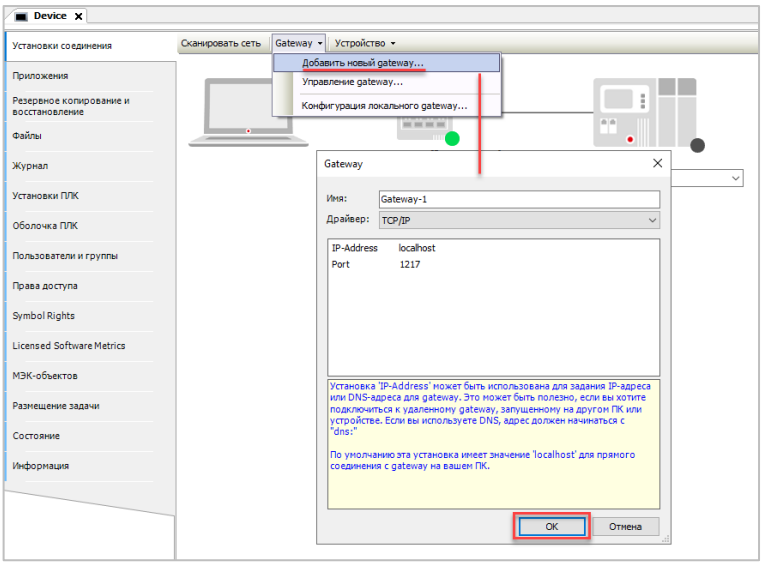


Рисунок. 3.4.3 – Создание нового gateway (шлюза)

Настройки рекомендуется оставить по умолчанию (имя – **Gateway-1**, IP-адрес – **localhost**). Закройте окно настроек шлюза и нажмите кнопку **Сканировать сеть**. В появившемся списке следует выбрать виртуальный контроллер (его имя совпадает с именем ПК).

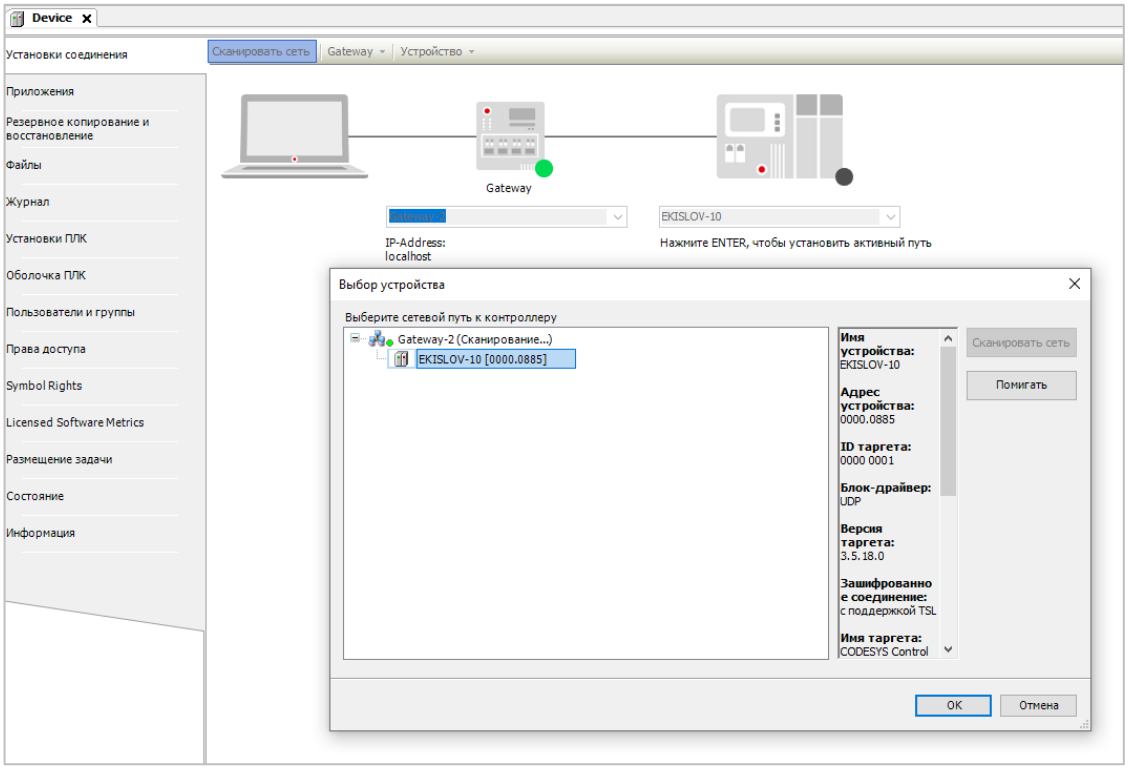


Рисунок 3.4.4 – Окно сканирования сети

Если вы используете среду CODESYS V3.5 SP17 или выше, то после нажатия на кнопку **ОК** появится окно с предложением создать пользователя контроллера и задать ему логин и пароль. Эти логин и пароль потребуется вводить при каждом подключении к виртуальному контроллеру.

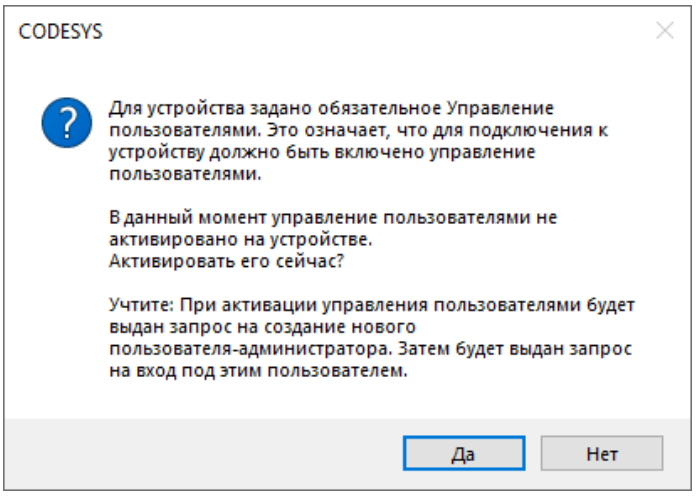


Рисунок 3.4.5 – Предложение создания пользователя при первом подключении к контроллеру

Если вы хотите избежать этого, то обязательное управление пользователями можно отключить. Для этого перейдите в папку **C:\ProgramData\CODESYS\CODESYSControlWinV3\<идентификатор контроллера>**, откройте файл **CODESYSControl.cfg** и в секции [CmpUserMgr] в строке ;SECURITY.UserMgmtEnforce=NO уберите первый символ (т. е. нужно раскомментировать параметр):

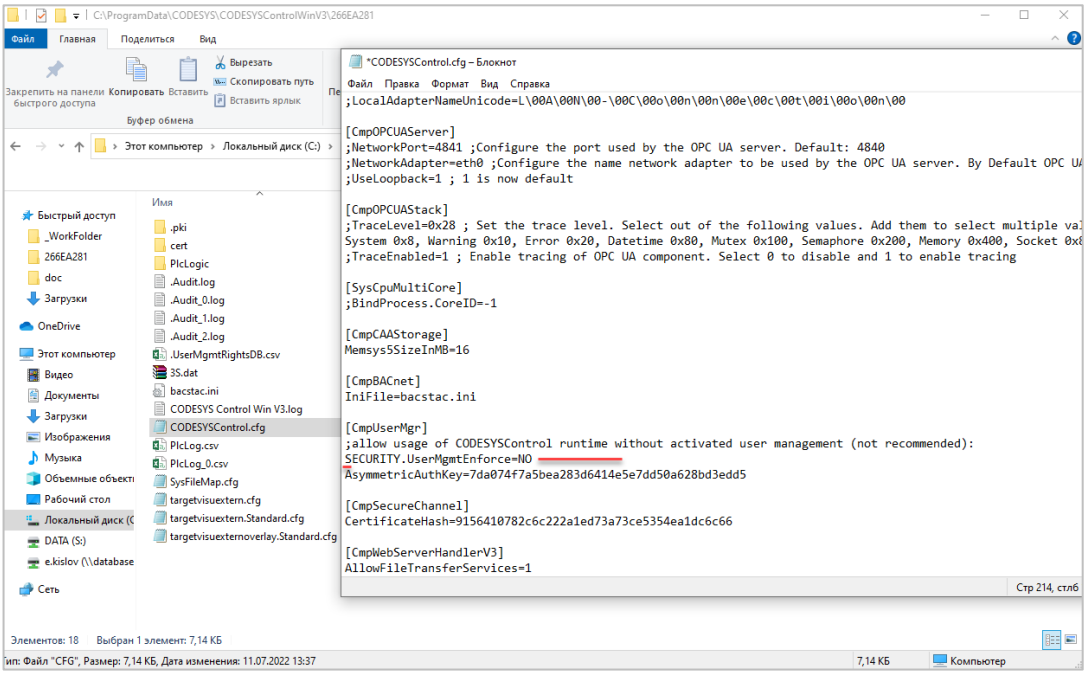


Рисунок 3.4.6 – Окно сканирования сети

Сохраните изменения в файле и перезапустите виртуальный контроллер: нажмите на его иконку в системном трее правой кнопкой мыши, выполните команду **Stop PLC** и после этого повторно выполните команду **Start PLC**. Повторите подключение к контроллеру в CODESYS – в этот раз никаких окон с предложением создать пользователя не возникнет.

В случае успешной установки связи индикаторы шлюза и контроллера загорятся зеленым:

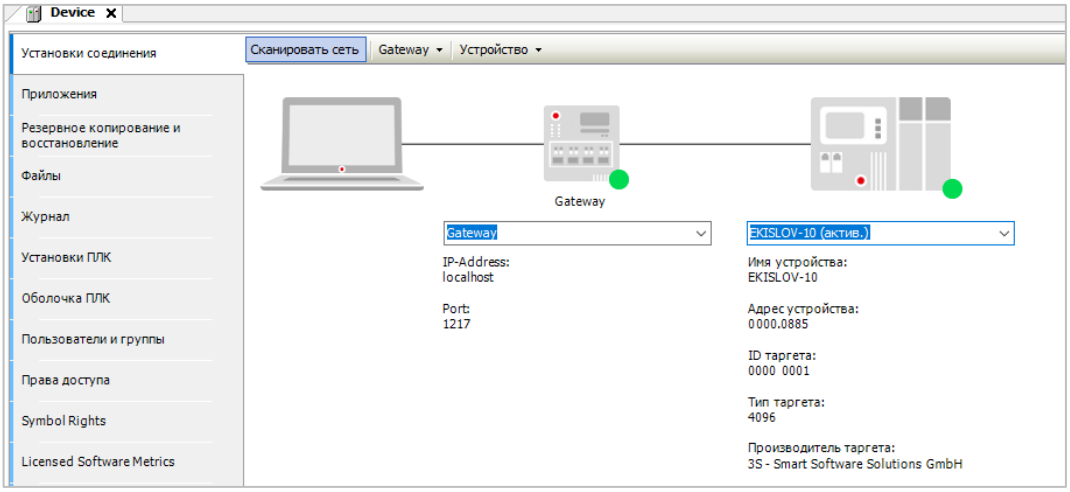


Рисунок 3.4.7 – Результат успешной установки связи

После успешного подключения можно переходить к загрузке проекта – см. следующий пункт.

3.5 Загрузка и запуск проекта. Работа с проектом

Для загрузки проекта используется команда **Логин** из меню **Онлайн** (ее ярлык продублирован на панели инструментов CODESYS).

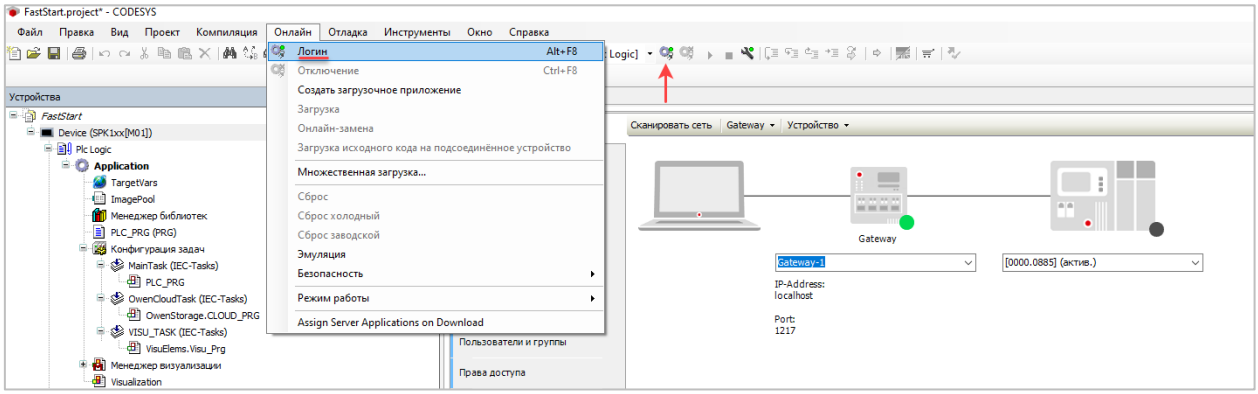


Рисунок 3.5.1 – Команда загрузки проекта

Если в контроллере еще нет проекта, то появится соответствующее информационное сообщение. Нажмите кнопку **Да**.

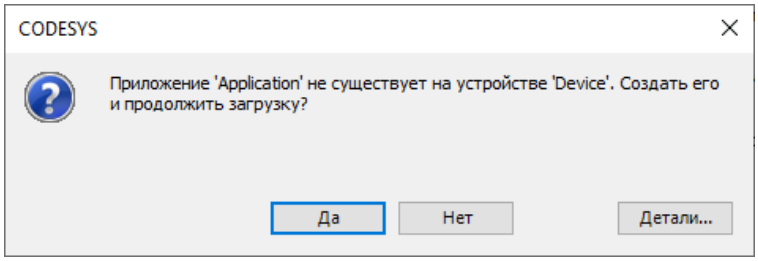


Рисунок 3.5.2 – Информационное окно об отсутствии приложения в контроллере

В результате проект будет сконвертирован в бинарное приложение, и это приложение будет загружено в контроллер, но не запущено на исполнение. Это можно определить по статусу **Стоп** в узле **Application** в дереве проекта, а также по серым неактивным пиктограммам задач и компонентов.

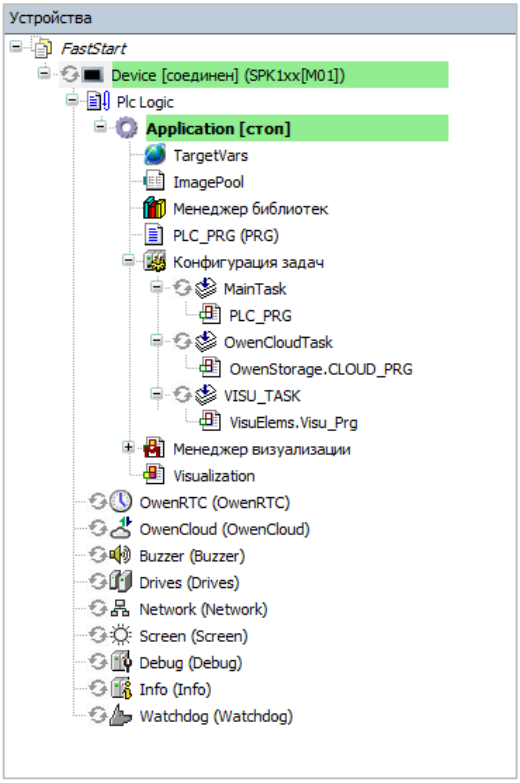


Рисунок 3.5.3 – Отображение статуса приложения (загружено, но не запущено)

Для запуска приложения используется команда **Старт** из меню **Отладка** (ее пиктограмма также продублирована на панели инструментов). Если вы запускаете проект на виртуальном контроллере – то во весь экран ПК откроется таргет-визуализация. Для того чтобы выйти из нее – нажмите на клавиатуре кнопку **Win** или **Alt+Tab**. После этого окно таргет-визуализации можно закрыть.

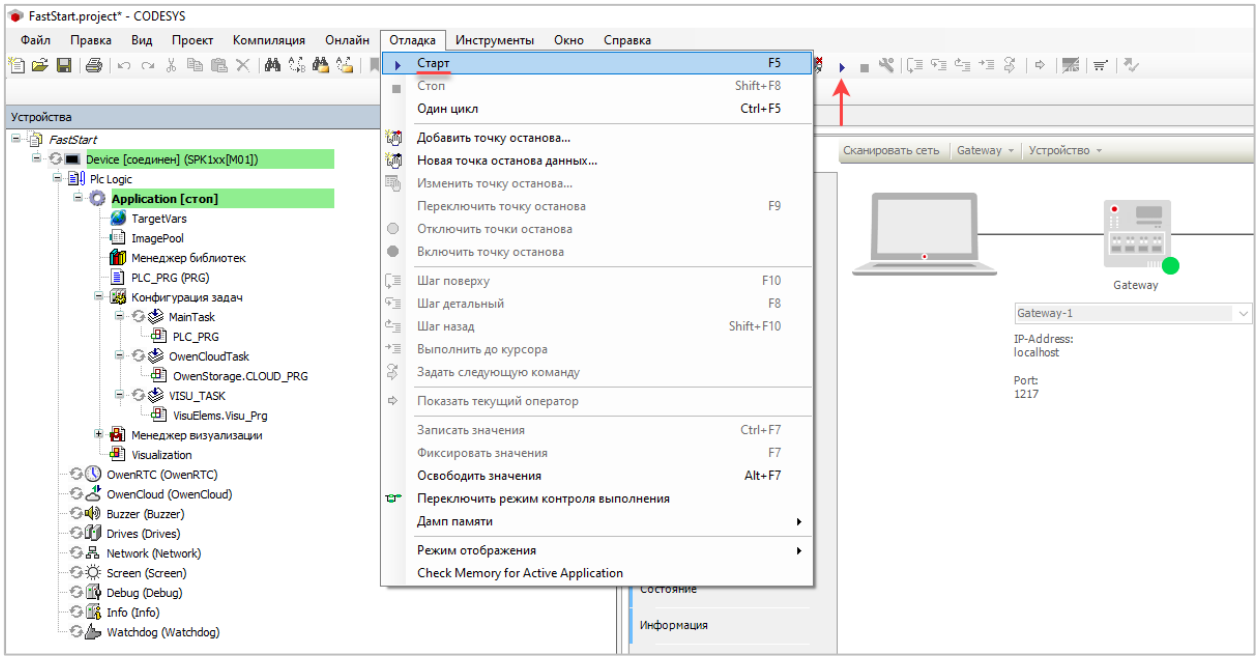


Рисунок 3.5.4 – Команда запуска приложения

Теперь приложение запущено – статус в узле **Application** изменился на «запуск», а пиктограммы задач и компонентов стали зелеными.

Сразу нужно рассказать об очень важном моменте. После предыдущих операций приложение загружено в оперативную память контроллера, но еще не сохранено в его flash-памяти. То есть если сейчас перезагрузить контроллер – то после загрузки в нем уже не будет нашего приложения. Поэтому необходимо сделать его загрузочным с помощью команды **Создать загрузочное приложение** из меню **Онлайн** – тогда оно сохранится во flash-памяти контроллера и будет автоматически запускаться после его перезагрузки.

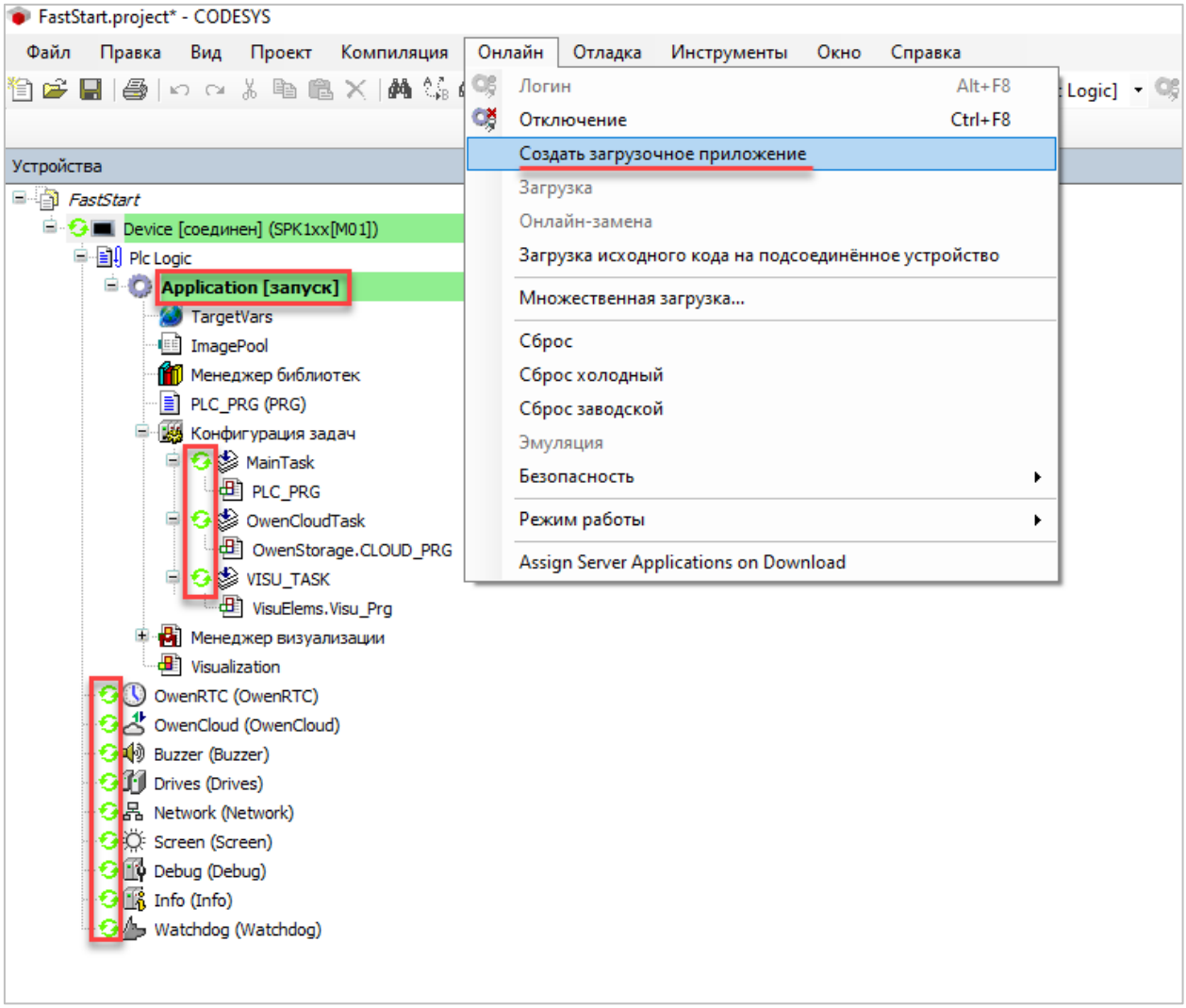


Рисунок 3.5.5 – Команда создания загрузочного приложения и индикация запущенного приложения в дереве проекта

ПРИМЕЧАНИЕ
Выше мы обозначили разницу между терминами «проект» и «приложение». Проект – это файл формата **.project**, с которым вы работаете в среде CODESYS. Когда вы нажимаете кнопку **Логин**, то перед загрузкой выполняется генерация кода (об этом подробнее см. в п. 4.2.5), в результате которой на основе содержимого проекта формируется бинарный файл приложения – именно он и будет загружен в контроллер.

После запуска проекта можно проверить работу web-визуализации. Для этого необходимо в адресной строке браузера ввести IP-адрес контроллера и порт сервера web-визуализации (по умолчанию – **8080**).

Если ваш ПК подключен к контроллеру с заводскими настройками по интерфейсу Ethernet (см. таблицу 3.2.1), то нужно ввести **192.168.0.10:8080**

Если ваш ПК подключен к контроллеру с заводскими настройками по интерфейсу USB (см. таблицу 3.2.2), то нужно ввести **172.16.0.1:8080** для ПЛК2xx или **10.0.6.10:8080** для СПК1xx.

Если вы используете виртуальный контроллер, то нужно ввести **localhost:8080**

После этого в web-браузере будет отображена созданная нами в п. 3.1 страница визуализации (а если вы используете панельный контроллер СПК – то сразу же после старта приложения эта же визуализация отобразится на экране контроллера):

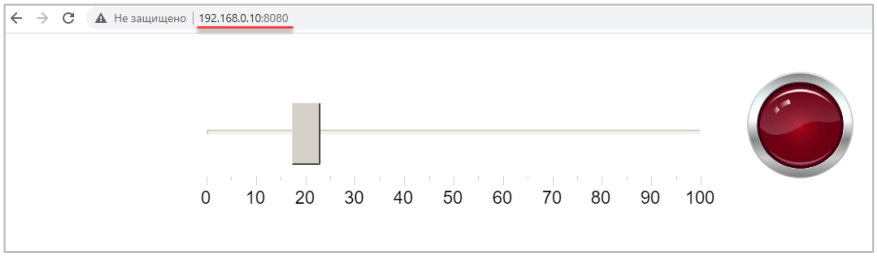


Рисунок 3.5.6 – Отображение web-визуализации в браузере

Измените положение ползунка. Если его значение превысит **50**, то будет включен индикатор тревоги. Если значение станет меньше или равным **50**, то индикатор выключится. Таким образом, программа работает именно так, как и было задумано.

В данный момент мы всё еще подключены к контроллеру в среде CODESYS. Давайте откроем программу **PLC_PRG** и увидим, что рядом с названиями переменных в данный момент отображаются их текущие значения. Довольно часто бывает полезным изменять эти значения «на лету» и смотреть, как реагирует программа (*конечно, в нашем случае это можно сделать и через web-визуализацию – но в реальном проекте могут быть сотни и даже тысячи переменных, и значительная часть из них не будет использоваться в визуализации*). Проще всего сделать это так: в таблице переменных нажмите два раза на ячейку столбца **Подготовленное значение** и введите значение, которое хотите присвоить переменной, и нажмите **Enter**. После этого в меню **Отладка** выполните команду **Записать значение** (или нажмите **Ctrl+F7**).

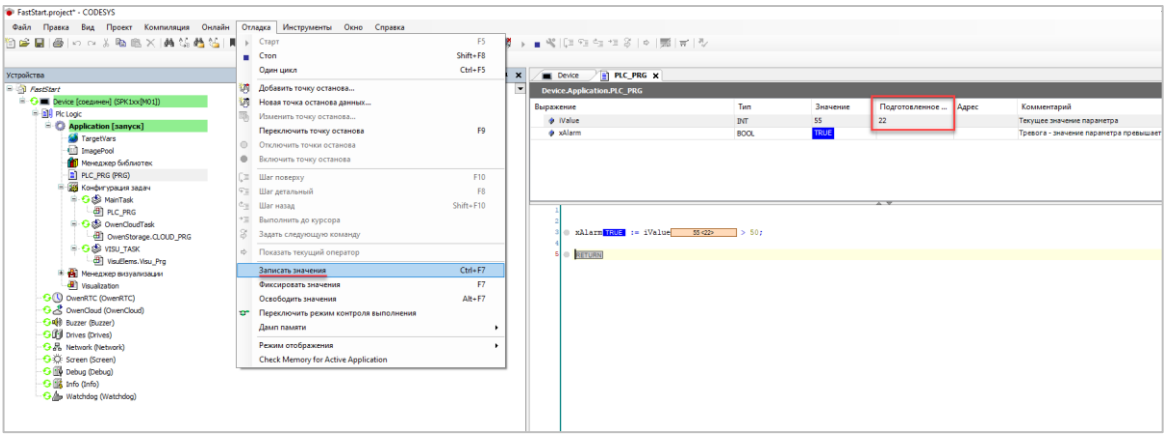


Рисунок 3.5.7 – Изменение значения переменной

После записи можно будет изменить значение переменной с помощью перемещения ползунка в визуализации. А вот если использовать команду **Фиксировать значения** – то переместить ползунок станет невозможным, так как подготовленное значение будет записываться в нашу переменную каждый цикл контроллера.

Для отключения от контроллера и возвращения к редактированию проекта используйте команду **Отключение** из меню **Онлайн**.

Если вы внесете в проект какие-либо изменения (например, поменяйте сейчас начальное значение переменной **iValue** с 20 на 30), то при выполнении команды **Логин** появится следующее окно:

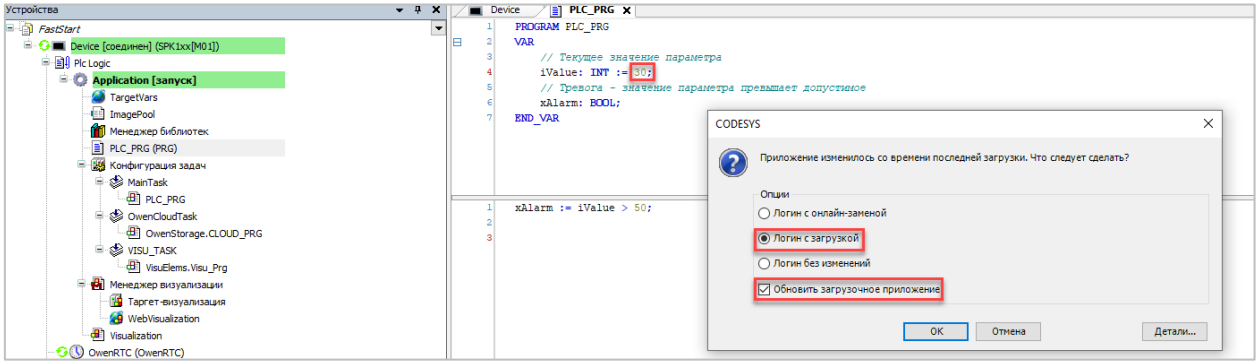


Рисунок 3.5.8 – Окно загрузки приложения после внесения в него изменений

В этом окне предлагается выбрать один из вариантов подключения к контроллеру. **Очень важный момент** – **никогда** не выбирайте вариант **Логин с онлайн-изменением** (он предлагается по умолчанию). Этот вариант подразумевает «горячее обновление» кода приложения, в процессе которого будут загружены только изменившиеся фрагменты кода. На практике во многих случаях эта процедура может работать некорректно – в результате вы можете столкнуться с рядом сложноуловимых ошибок (например, произвольное изменение значений переменных из-за некорректного перераспределения их адресов в памяти). Поэтому всегда используйте для загрузки приложения вариант **Логин с загрузкой** – в этом случае в контроллер будет произведена полная загрузка приложения. Вы можете также установить галочку **Обновить загрузочное приложение** – тогда после загрузки приложения будет автоматически выполнена команда **Создать загрузочное приложение**, и вам не придется выполнять ее вручную.

Итак, в данном разделе мы:

- создали простейший проект с программой на языке ST и визуализацией;
- подключились к контроллеру из среды CODESYS и загрузили в него этот проект;
- поработали с web-визуализацией;
- научились изменять значения переменных при онлайн-подключении к контроллеру;
- узнали, что нельзя использовать вариант загрузки приложения **Логин с онлайн-заменой**.

Если вы раньше уже работали с программируемыми контроллерами, то, вероятно, к этому моменту у вас появился ряд вопросов. В следующем пункте мы постараемся ответить на самые предсказуемые из них.

3.6 Вопросы после «быстрого старта»

3.6.1 Приложение не сохраняется после перезагрузки контроллера

Если после перезагрузки контроллера приложение не сохраняется (то есть вы видите сообщение с рис. 3.5.2) – то это значит, что вы не сделали его загрузочным. См. рис. 3.5.5 и информацию перед этим рисунком.

3.6.2 Как удалить приложение из контроллера?

В п. 3.5 мы разобрались, как загрузить приложение в контроллер. Если его необходимо удалить – то используйте команду **Сброс заводской** из меню **Онлайн**. Выполнение команды приведет к удалению приложения CODESYS, при этом все остальные настройки (сетевые настройки и т. д.) не будут сброшены к заводским – они сохранят свои текущие значения.

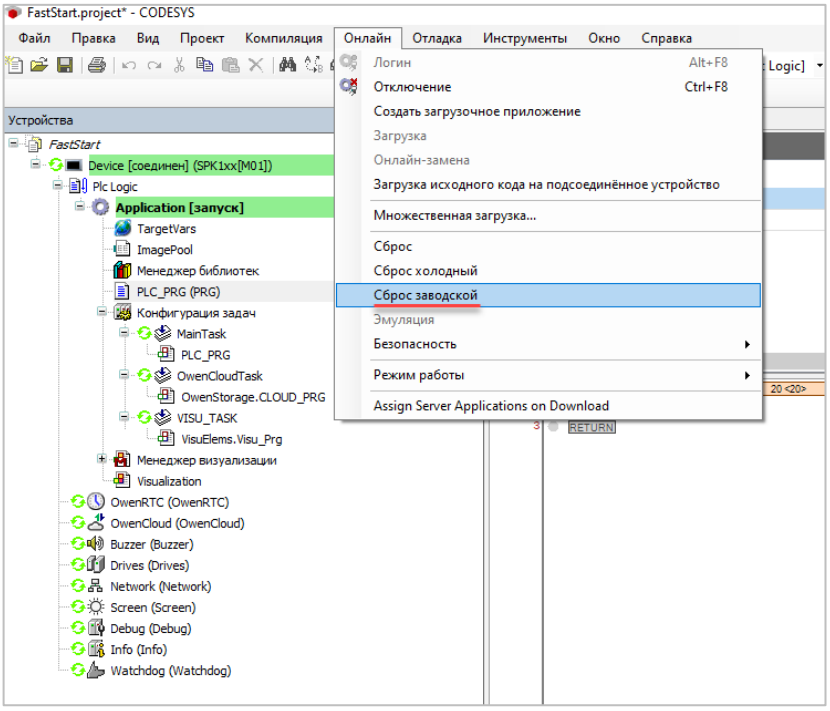


Рисунок 3.6.1 – Команда удаления приложения

3.6.3 Можно ли выгрузить приложение из контроллера и загрузить в другой контроллер?

Этот вопрос обычно имеет две разные интерпретации:

- инженера эксплуатации интересует возможность подготовки «бэкапа» – чтобы в случае выхода контроллера из строя достать из ЗИП запасной контроллер, загрузить в него тот же проект и восстановить работу системы автоматизации;
- инженера-программиста интересует вопрос защиты интеллектуальной собственности – чтобы разработанный им проект не смогли изменять и тиражировать на другие контроллеры без его ведома.

Самые важные моменты, связанные с этим вопросом:

- при выполнении команды **Создать загрузочное приложение** в контроллере сохраняется бинарный файл загрузочного приложения. Получение этого файла не дает доступа к исходникам – восстановить из него файл проекта (.project) не получится;
- файл загрузочного приложения можно перенести на другой контроллер той же модели и с той же версией прошивки, но эту возможность можно отключить или защитить паролем. Кроме того, можно в коде программы считать серийный номер контроллера, сравнить с заданным и, например, не выполнять в программе никаких действий в случае их несовпадения – это делает перенос загрузочного приложения на другой контроллер бессмысленным (серийный номер присваивается контроллеру на этапе производства и не может быть изменен пользователем);
- CODESYS имеет функционал для создания резервной копии приложения и его развертывания. Контроллеры ОВЕН позволяют через web-конфигуратор (и экранный конфигуратор панельных контроллеров СПК) создавать резервные копии, в состав которых входит и приложение CODESYS, и все остальные настройки контроллера (сетевые настройки и т. д.). В обоих случаях эти операции могут быть защищены паролем;
- загрузка в контроллер «исходников» (файла формата .project) может быть выполнена с помощью команды **Загрузка исходного кода на подсоединенное устройство** из меню **Онлайн** (см. рис. 3.6.2). Для выгрузки этого файла в CODESYS нужно использовать команду **Выгрузка исходного кода** из меню **Файл** (см. рис. 3.6.3).

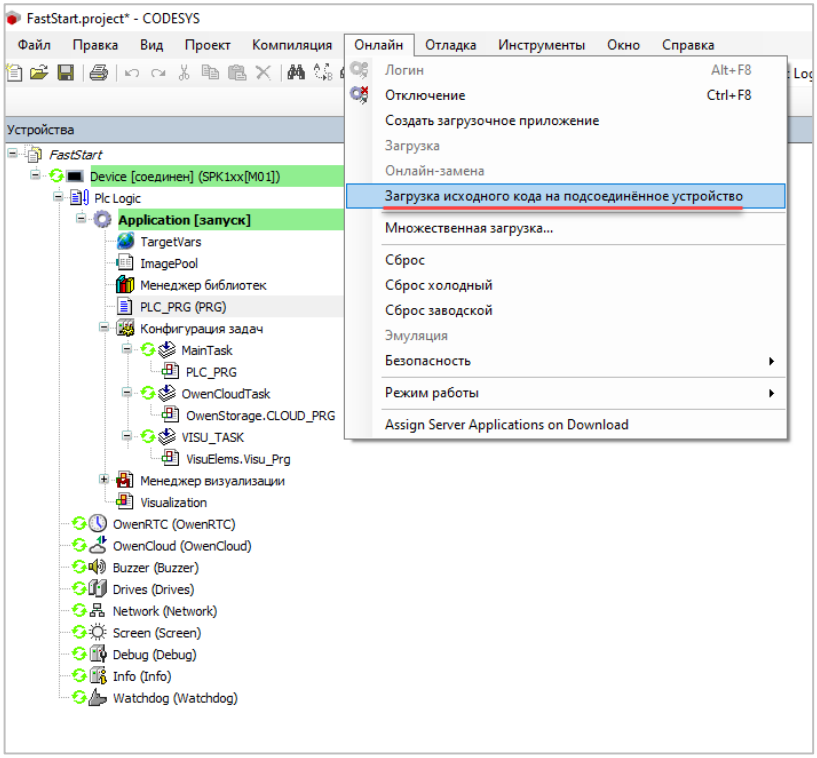


Рисунок 3.6.2 – Команда загрузки исходного кода проекта в контроллер

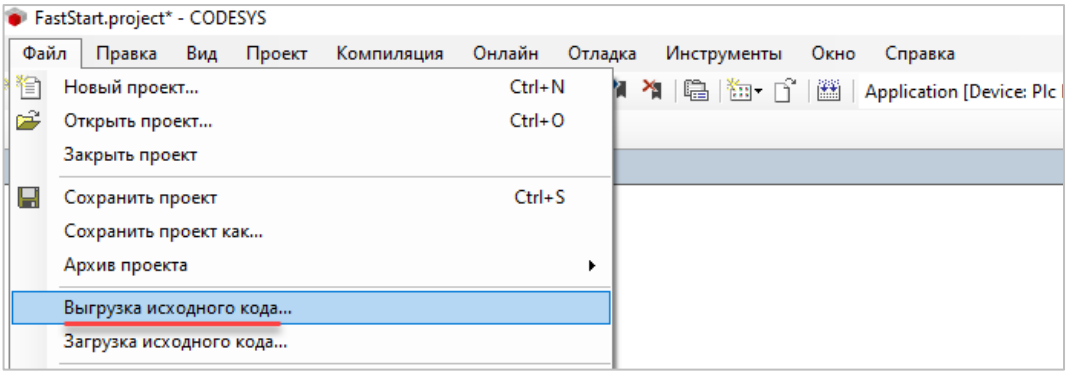


Рисунок 3.6.3 – Команда выгрузки исходного кода проекта из контроллера

3.6.4 Как правильно перенести проект на другой ПК с установленным CODESYS?

Достаточно часто требуется передавать файлы проектов между различными ПК. На всех этих ПК должна быть установлена та версия CODESYS, в которой был создан проект. Но проект может включать в себя дополнительные компоненты, библиотеки и т. д. Чтобы не передавать вместе с проектом все эти файлы – нужно использовать архивы проектов (**.projectarchive**). Для создания архива проекта используется команда **Архив проекта** – **Сохранить/отправить архив** из меню **Файл**. Размер архива проекта существенно больше файла проекта (обычно он составляет не менее 60 Мб), но зато он включает в себя все компоненты и библиотеки, используемые в проекте.

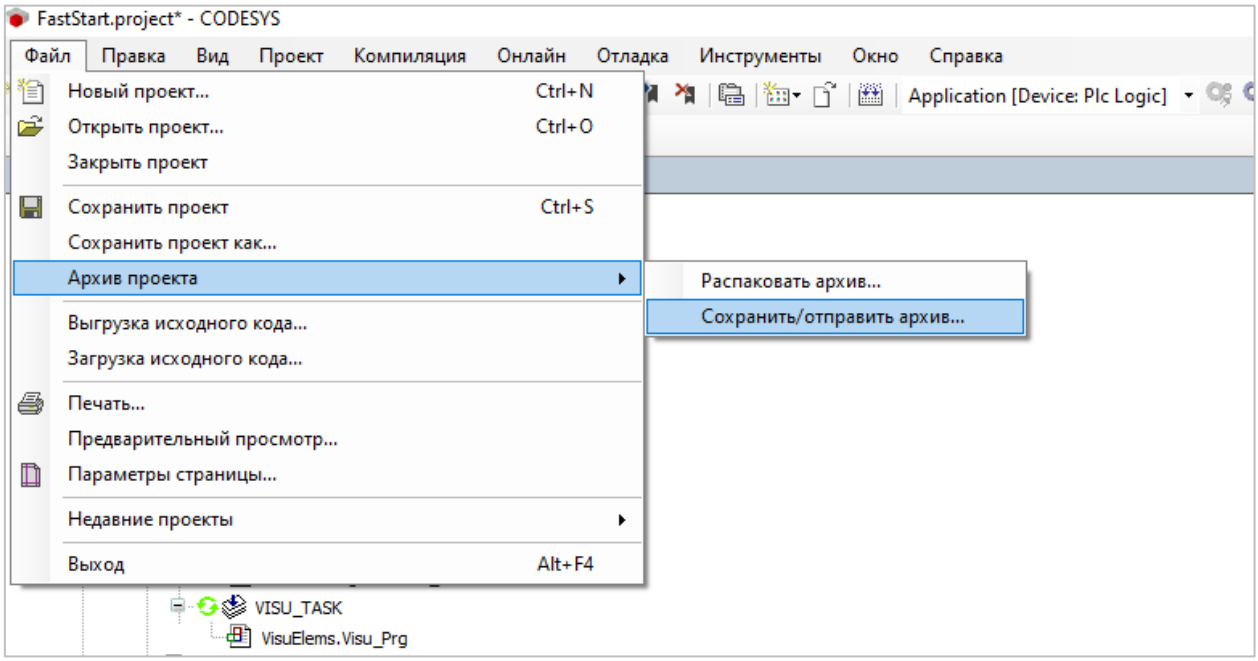


Рисунок 3.6.4 – Команда создания архива проекта

3.6.5 Где посмотреть лог сообщений контроллера?

Лог сообщений контроллера доступен в узле **Device** на вкладке **Журнал**. Также его можно посмотреть в web-конфигураторе контроллера на вкладке **ПЛК/Файлы журналов**.

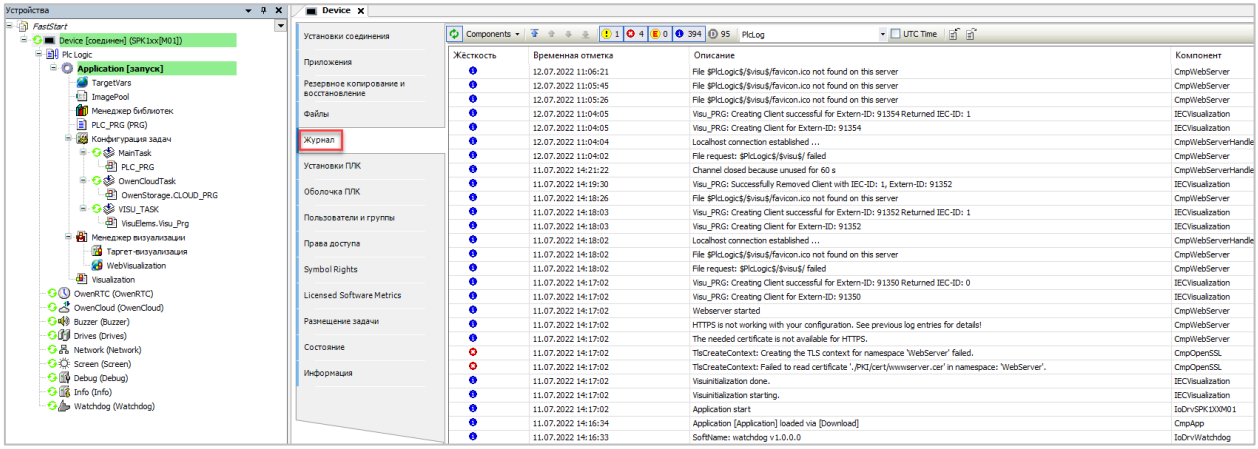


Рисунок 3.6.5 – Отображение лога сообщений контроллера в CODESYS

3.6.6 Как понять, что в проекте есть ошибки?

Если в проекте допущена ошибка, то в большинстве случаев она будет детектирована автоматически. Предположим, что в коде нашего проекта мы забыли поставить точку с запятой в конце выражения. Тогда объект, в котором детектирована ошибка (в нашем случае – это программа **PLC_PRG**) будет подчеркнут в дереве проекта красной волнистой линией, а на панели сообщений компиляции, отображаемой в нижней части экрана, появится сообщение об ошибке с указанием объекта, в котором она произошла, и предполагаемой позиции (в нашем случае отображается номер строки). Ошибка может быть детектирована на разных этапах (предкомпиляция, компиляция и т. д.). Для просмотра сообщений по конкретному этапу нужно выбрать его в выпадающем списке.

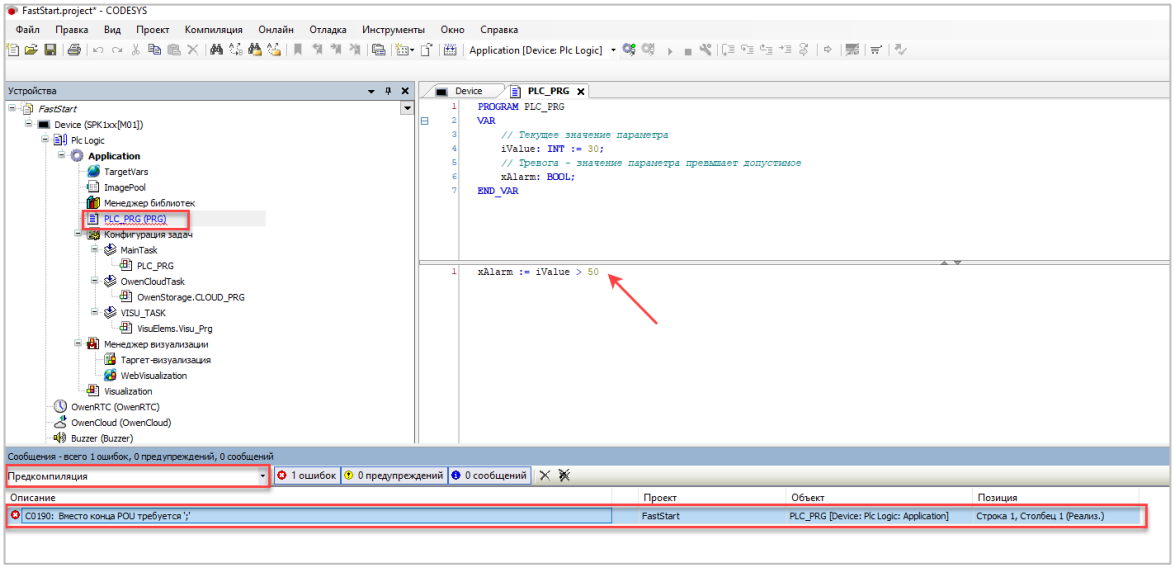


Рисунок 3.6.6 – Информация об ошибке на панели сообщений компиляции

Если вы случайно закрыли панель сообщений, то вернуть ее можно с помощью команды **Сообщения** в меню **Вид**.

В ряде случаев ошибка не может быть обнаружена в процессе автоматических проверок. Поэтому рекомендуется периодически выполнять команду **Генерировать код** из меню **Компиляция**. Эта команда проверяет проект на наличие ошибок, и в случае их отсутствия – создает файл приложения, который будет загружен в контроллер при подключении к нему. Информация о проверках сохраняется во внутренние файлы CODESYS; в ряде случаев эти файлы могут быть повреждены или перезаписаны некорректно (например, в процессе постоянных регулярных проверок). Это, например, может привести к сообщениям об ошибках, которых фактически в проекте нет. Поэтому рекомендуется перед выполнением команды **Генерировать код** выполнить команду **Очистить все**, чтобы удалить информацию о предыдущих проверках проекта.

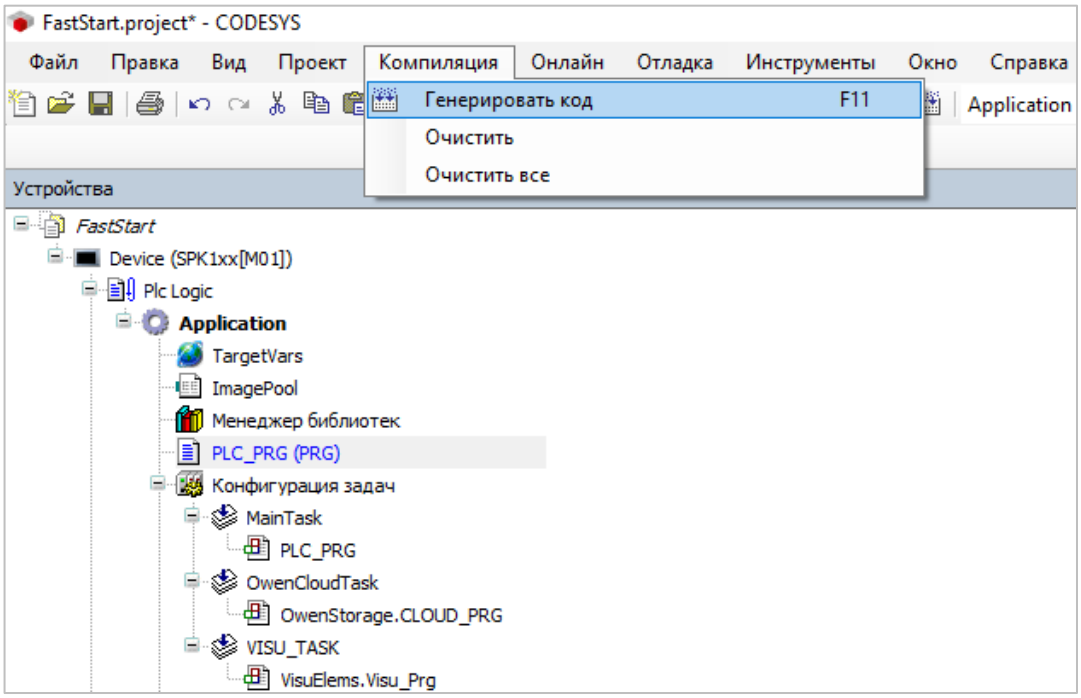


Рисунок 3.6.7 – Команды генерации кода и очистки файлов компиляции

3.6.7 С каким периодом выполняется код программы?

В п. 3.1 мы добавили в программу **PLC_PRG** код сравнения значения переменной **iValue** с заданной величиной (эту величину мы задали равной **50**). Справедливый вопрос, который может возникнуть у внимательного читателя – с какой периодичностью производится эта проверка?

Напомним, что мы создавали проект на основе шаблона. В шаблоне проекта программа **PLC_PRG** привязана к задаче **MainTask** – это можно увидеть в узле дерева проекта, который называется **Конфигурация задач**. Задачи определяют интервалы вызова привязанных к ним программ. Интервалы обычно задаются в миллисекундах. Например, интервал задачи **MainTask** в шаблоне проекта контроллера СПК равен 10 мс (см. рис. ниже) – соответственно, именно с таким периодом вызывается код программы **PLC_PRG**.

Больше информации о задачах приведено в п. 5.9.

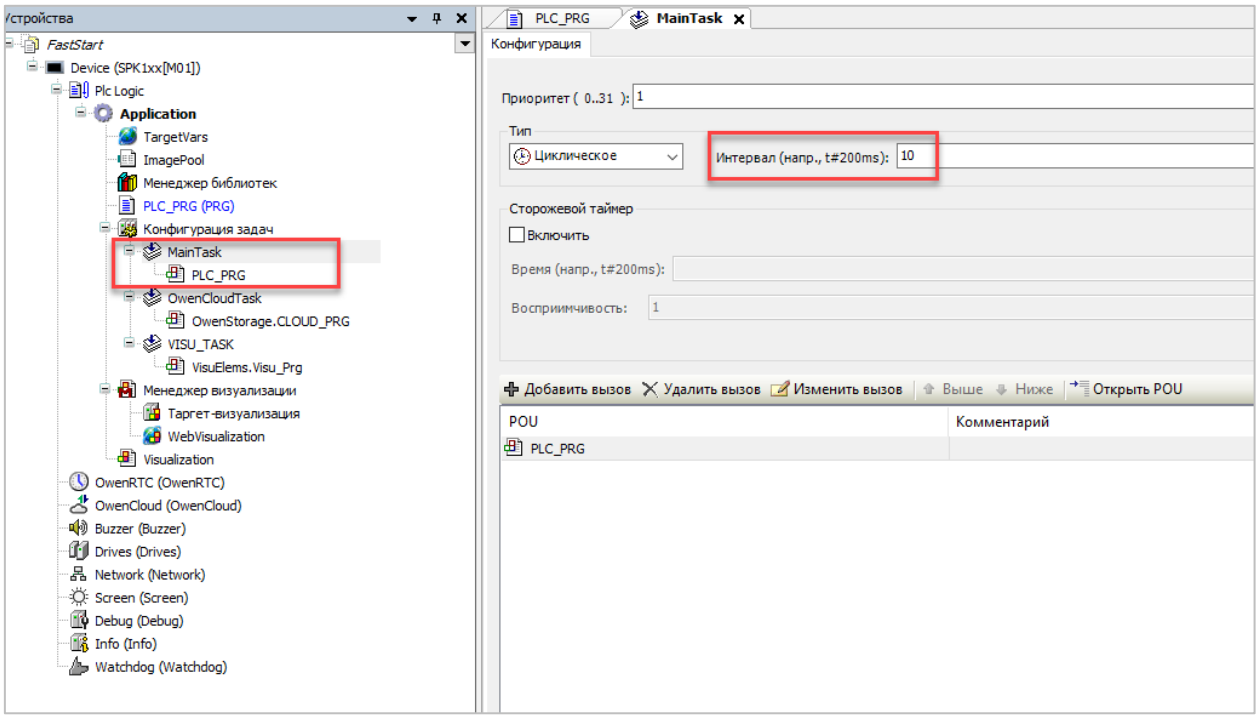


Рисунок 3.6.8 – Настройки задачи MainTask в шаблоне проекта

3.6.8 Как включить сетку в редакторе визуализации?

Для включения сетки в редакторе визуализации перейдите в меню **Инструменты** и используйте команду **Опции – Визуализация – Сетка – Видимая**.

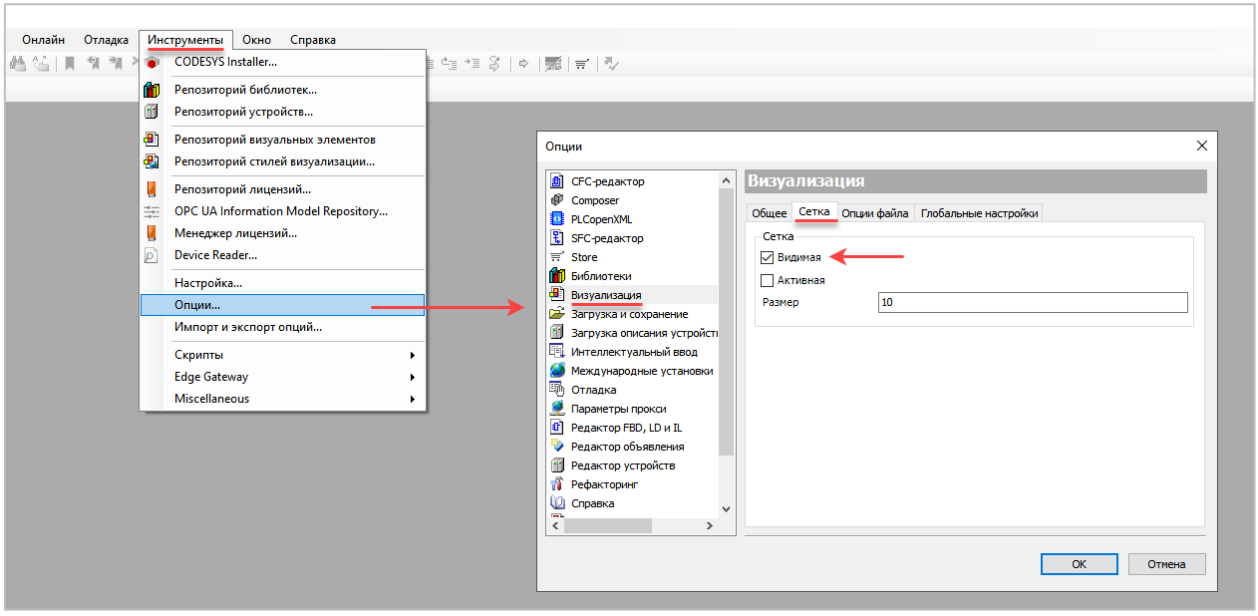


Рисунок 3.6.9 – Включение сетки в редакторе визуализации

3.6.9 Существует ли ограничение на размер приложения?

Явных ограничений на размер приложения нет; есть лишь ограничение, связанное с объемом свободной памяти контроллера. Для контроллеров ПЛК2xx объем доступной пользователю памяти составляет ≈70 Мб, для контроллеров СПК1xx [M01] – 2 Гб.

Посмотреть текущий объем свободной памяти можно при подключении к контроллеру в узле **Drives**, присвоив значение **TRUE** каналу **Enable Drives**. Объем свободной памяти в байтах будет отображен в канале **FS free**.

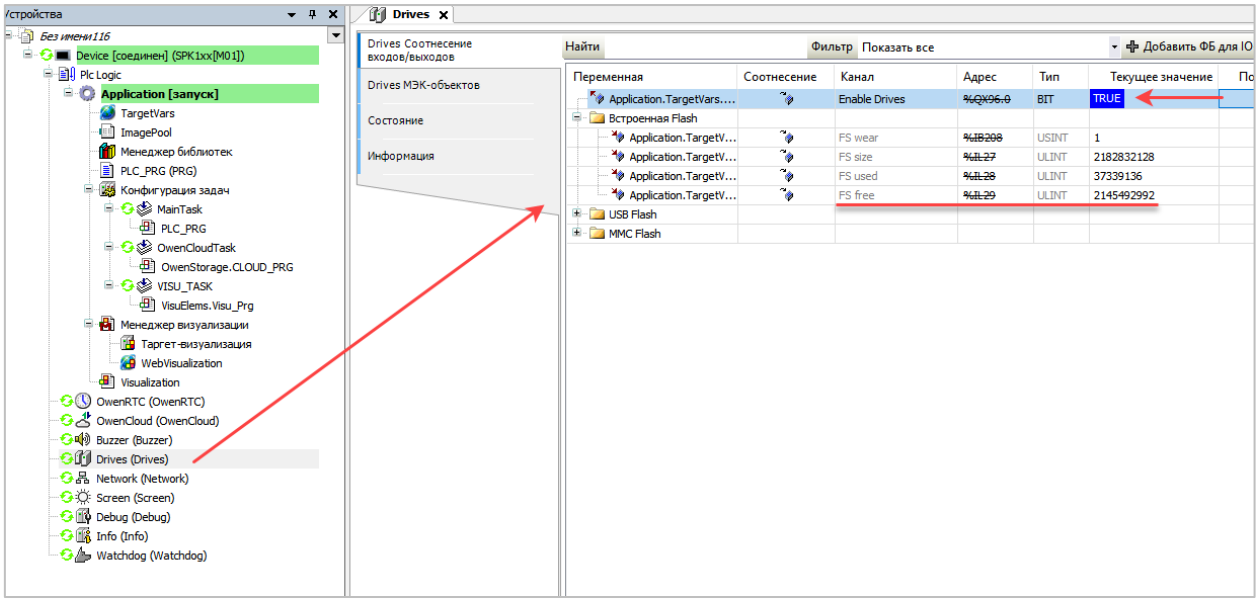


Рисунок 3.6.10 – Отображение объема свободной памяти контроллера в узле Drives

4 Интерфейс CODESYS

В этом разделе мы более подробно рассмотрим интерфейс среды CODESYS и познакомимся с наиболее часто используемыми вкладками и командами.

4.1 Обзор интерфейса CODESYS

На рисунке 4.1.1 отображены основные области интерфейса CODESYS:

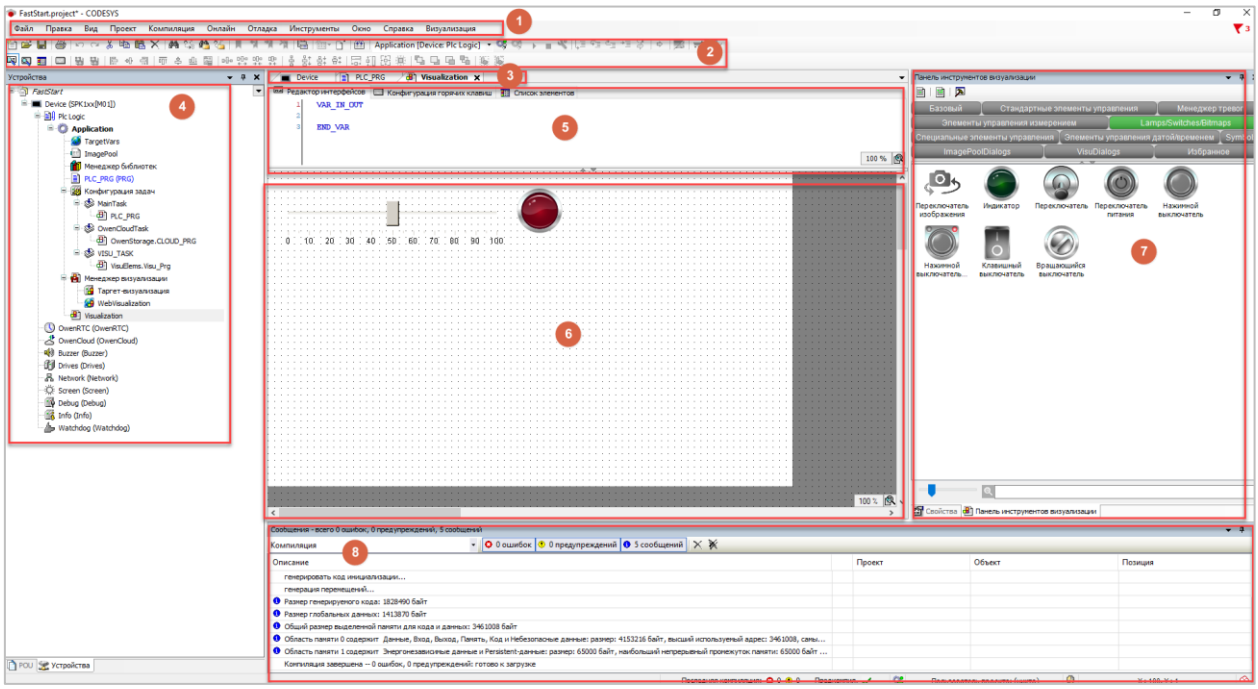


Рисунок 4.1.1 – Основные области интерфейса CODESYS

1 – панель меню. Содержит вкладки с командами, используемыми при работе в CODESYS;
2 – панель инструментов. Содержит ярлыки для наиболее часто используемых команд;
3 – вкладки открытых компонентов CODESYS (редакторов, конфигураторов и т. д.). Новая вкладка открывается с помощью двойного нажатия левой кнопкой мыши на узел в дереве проекта;
4 – дерево проекта. Содержит древовидную структуру используемых в проекте объектов;
5 – область объявления переменных (для редакторов языков программирования и редактора визуализации);
6 – окно открытого редактора;
7 – панель элементов открытого редактора (например, панель элементов языка CFC, панель элементов визуализации и т. д.);
8 – панель сообщений компиляции. Также в этой области открываются другие панели – например, панель мониторинга переменных, панель точек останова и т. д.

Пользователь может перемещать вкладки и панели по экрану – для этого нужно зажать левой кнопкой мыши заголовок вкладки/панели. Набор команд в меню и ярлыков панели инструментов можно изменить в меню **Инструменты – Настройка**.

4.2 Панель меню

4.2.1 Меню «Файл»

Меню **«Файл»** содержит базовые команды для работы с проектом – команды открытия, закрытия и сохранения.

Команда **Архив проекта** позволяет сохранить проект в формате **.projectarchive**. В отличие от файла проекта (**.project**) архив включает в себя используемые в проекте компоненты, библиотеки, файлы описания устройств и т. д. Это очень удобно при переносе проекта с одного ПК на другой – в этом случае не потребуется на втором ПК отдельно устанавливать нужные для конкретного проекта библиотеки и компоненты.

Команды **Загрузка исходного кода** и **Выгрузка исходного кода** позволяют загрузить и выгрузить в контроллер файл проекта (**.project**). Это удобно на стадии эксплуатации контроллера: если потребуется внести в проект доработки, но исходники будут утеряны – то можно будет выгрузить их из контроллера.

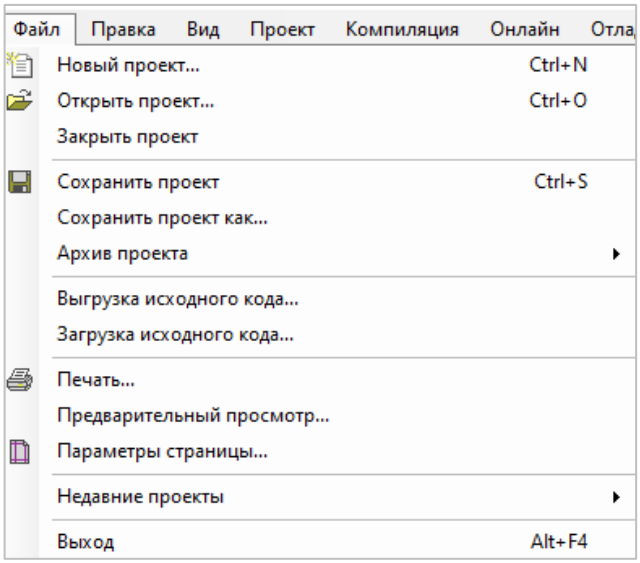


Рисунок 4.2.1 – Команды меню «Файл»

4.2.2 Меню «Правка»

Меню **«Правка»** содержит базовые команды для работы с содержимым проекта – команды копирования/вставки, отмены последней команды (undo), поиска и поиска с заменой по тексту и т. д.

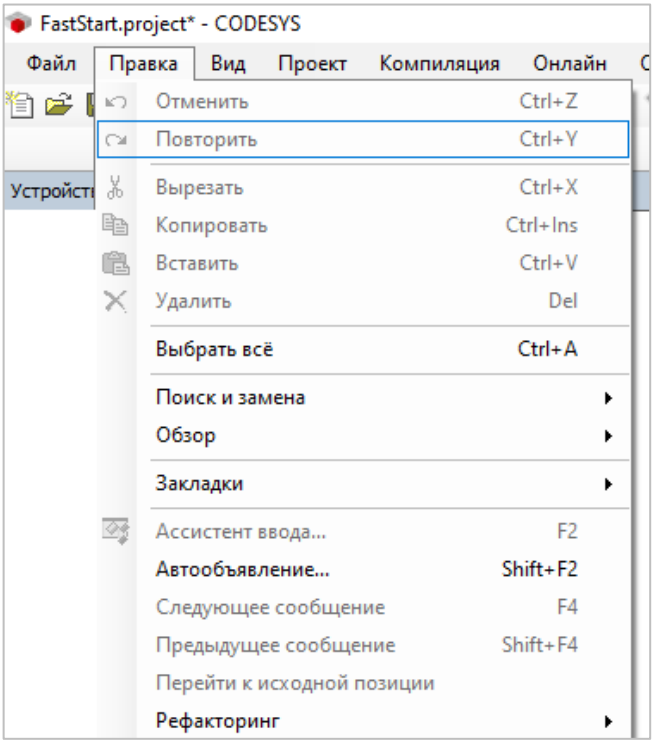


Рисунок 4.2.2 – Команды меню «Правка»

4.2.3 Меню «Вид»

Меню **«Вид»** содержит команды для открытия различных панелей – например, дерева проекта («Устройства»), панели сообщений компиляции («Сообщения»), панели мониторинга («Просмотр») и т. д. Если вы случайно закрыли какие-то нужные вам панели – то сможете повторно открыть их с помощью этой вкладки.

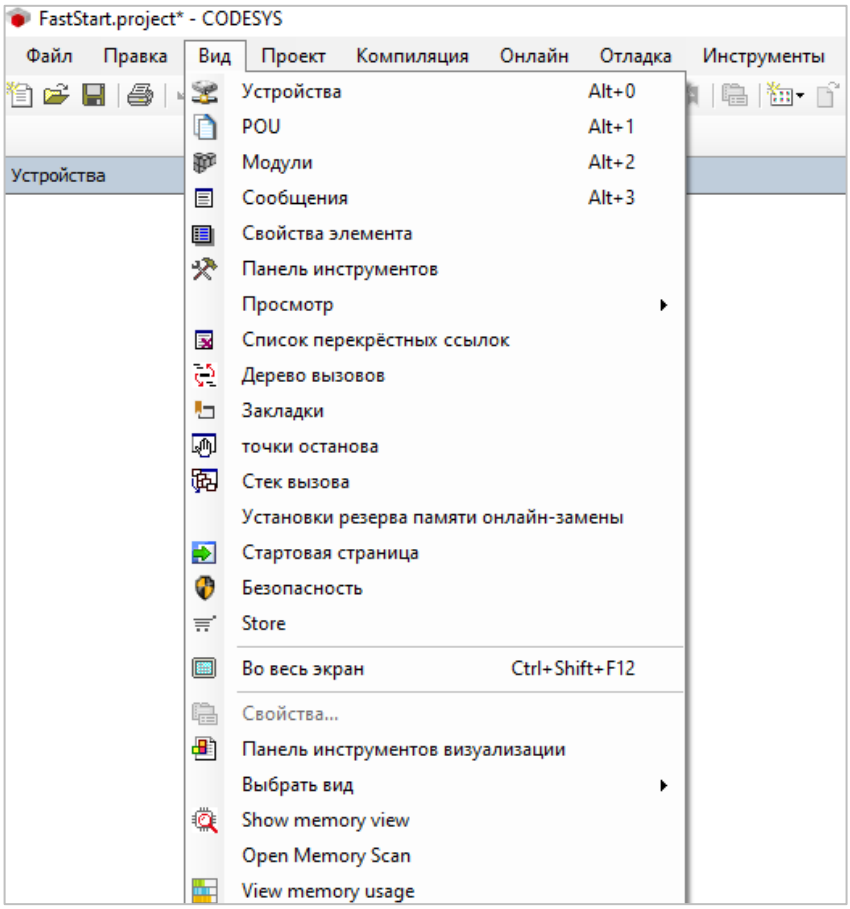


Рисунок 4.2.3 – Команды меню «Вид»

4.2.4 Меню «Проект»

Меню **«Проект»** содержит команды для настройки проекта, а также экспорта/импорта его содержимого в различных форматах. Наиболее часто используемые команды:

- **Установки проекта** – открывает окно настроек конкретного проекта. См. описание наиболее часто требуемых настроек на следующей странице;
- **Экспорт/Импорт** – эти команды позволяют экспортировать/импортировать проект или любые его фрагменты (например, функциональные блоки, экраны визуализации и т. д.) в специфическом для CODESYS xml-подобном формате. Это удобно в тех случаях, когда требуется передать другому пользователю фрагмент вашего проекта без передачи самого проекта;
- **Экспорт PLCopenXML/Импорт PLCopen XML** – эти команды позволяют экспортировать/импортировать программные объекты проекта в формате **PLCopen XML**. Это универсальный формат для передачи программных объектов между различными средами разработки для контроллеров, специфицированный в стандарте МЭК 61131-10;
- **Управление пользователями** – эта команда позволяет создать для конкретного проекта ряд пользователей с различными правами (например, одному пользователю проект будет доступен в режиме «только чтение», другому доступ к конкретным объектам вообще будет закрыт и т. д.). Можно настроить права доступа для каждого объекта в дереве проекта. Для каждого пользователя задается логин и пароль.

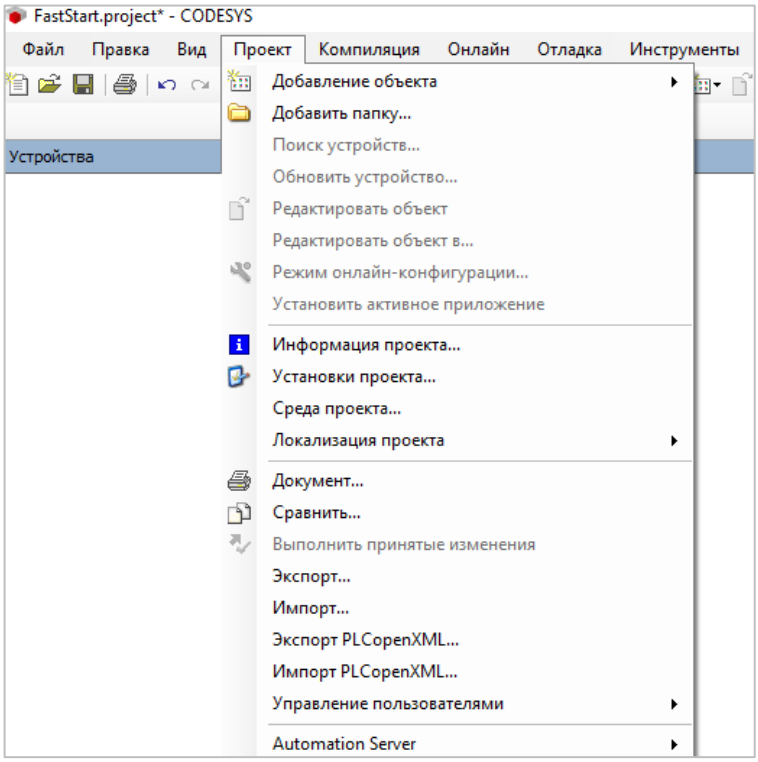


Рисунок 4.2.4 – Команды меню «Проект»

Окно **Установки проекта** выглядит следующим образом:

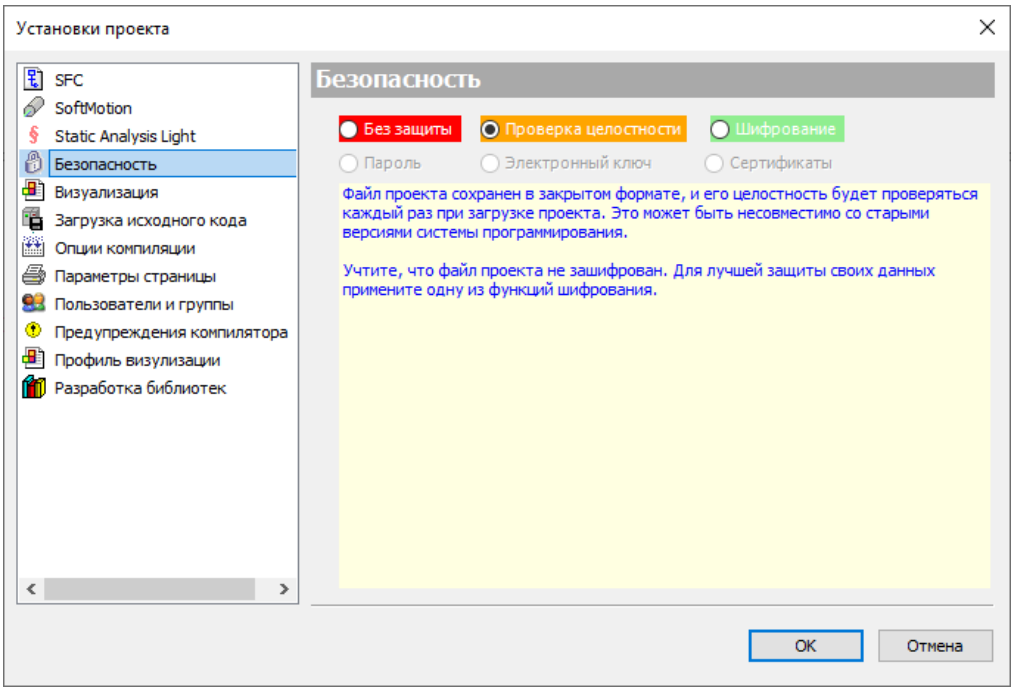


Рисунок 4.2.5 – Внешний вид окна установок проекта

Наиболее часто используется вкладки:

- **Безопасность** – позволяет ограничить доступ к проекту с помощью пароля/сертификата/электронного ключа для защиты интеллектуальной собственности. В отличие от управления пользователями, упомянутого на предыдущей странице, действие данной вкладки распространяется не на конкретные объекты проекта, а на файл проекта в целом;
- **Опции компиляции** и **Профиль визуализации** – настройки этих вкладок должны соответствовать версии таргет-файла контроллера. Изменение этих настроек необходимо при переносе проекта, созданного в старой версии CODESYS, в новую версию.

4.2.5 Меню «Компиляция»

Меню **«Компиляция»** содержит команду **Генерировать код**, которая проверяет проект на наличие ошибок, и в случае их отсутствия – создает файл приложения, который будет загружен в контроллер при подключении к нему. Информация о проверках сохраняется во внутренние файлы CODESYS; в ряде случаев эти файлы могут быть повреждены или перезаписаны некорректно (например, в процессе постоянных регулярных проверок). Это, например, может привести к сообщениям об ошибках, которых фактически в проекте нет. Поэтому рекомендуется перед выполнением команды **Генерировать код** выполнить команду **Очистить все**, чтобы удалить информацию о предыдущих проверках проекта.

В случае обнаружения в процессе генерации кода ошибок – информация о них будет отображена в панели сообщений компиляции (см. п. 3.6.6).

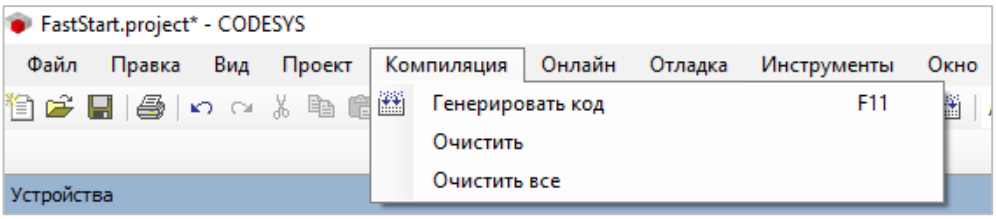


Рисунок 4.2.6 – Команды меню «Компиляция»

4.2.6 Меню «Онлайн»

Меню **«Онлайн»** содержит команды для подключения к контроллеру, загрузки приложения и исходников проектов, а также сброса контроллера. Наиболее часто используемые команды:

- **Логин** – команда подключения к контроллеру. В процессе логина может быть произведена загрузка в контроллер нового приложения. См. более детальную информацию в п. 3.5;
- **Создать загрузочное приложение** – если команда выполняется при подключении к контроллеру, то происходит сохранение загруженного приложения во flash-память контроллера. Если после загрузки приложения не выполнена эта команда – то после перезагрузки контроллера приложение не будет сохранено. См. более детальную информацию в п. 3.6.1. Если команда выполняется в тот момент, когда подключение к контроллеру отсутствует – то загрузочное приложение сохраняется на ПК; в дальнейшем его можно загрузить в контроллер с USB-/SD-накопителя или через конфигуратор;
- **Загрузка исходного кода на подсоединенное устройство** – аналогична команде **Загрузка исходного кода** из меню «Файл», но позволяет загрузить файл проекта сразу в подключенный контроллер (команда **Загрузка исходного кода** открывает список найденных в сети контроллеров для загрузки проекта в конкретный из них);
- **Сброс** – команда переинициализации всех переменных проекта, кроме энергонезависимых. После ее выполнения переменные принимают указанные при их объявлении начальные значения; если начальные значения не указаны – используются значения по умолчанию (например, 0 для числовых типов);
- **Сброс холодный** – команда переинициализации переменных проекта, включая энергонезависимые (RETAIN) переменные;
- **Сброс заводской** – удаление из контроллера текущего приложения и его исходного кода (если он загружен в контроллер). При этом все остальные настройки (сетевые настройки и т. д.) не будут сброшены к заводским – они сохранят свои текущие значения.

Отдельно следует рассказать про команду **Эмуляция**. Она переводит проект в режим эмуляции; для подключения к эмулятору нужно использовать команду **Логин**. Эмуляция имеет серьезные ограничения. В частности, в ней нельзя проверить:

- работу web-визуализации;
- работу с файлами;
- обмен с другими устройствами.

Поэтому рекомендуется вообще не использовать режим эмуляции и вместо этого проверять работу проекта на виртуальном контроллере.

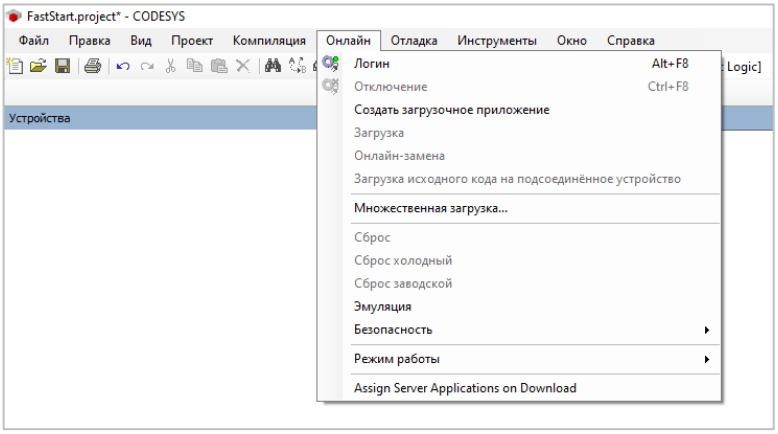


Рисунок 4.2.7 – Команды меню «Онлайн»

4.2.7 Меню «Отладка»

Меню **«Отладка»** содержит команды для анализа ошибок в коде проекта. Большинство команд доступны только при подключении к контроллеру. Наиболее часто используемые команды:

- **Старт** – команда запуска приложения, обычно выполняется пользователем сразу после загрузки нового приложения;
- **Команды, связанные с точками останова** – позволяют установить в редакторах кода проекта точки останова. При достижении этих точек приложение будет автоматически останавливаться. Это позволяет, например, отследить, выполняются ли определенные ветви кода, определить, какие значения имеют переменные проекта в моменты выполнения определенных условий и т. д.;
- **Команды, связанные с шагами** – если приложение остановлено в точке останова, то с помощью этих команд можно выполнять его фрагментарно, отдельно «проходя» через каждую строку кода;
- **Записать значения** – эта команда позволяет присвоить переменным проекта заданные пользователем значения (см. рис. 3.5.7 и пояснение перед ним). Если значение переменной циклически изменяется в самом проекте (например, эта переменная привязана ко входу контроллера или подключенного к контроллеру модуля ввода, или же в программе ей присваивается значение другой переменной), то команда **Записать значения** не сработает – вместо нее нужно использовать команду **Фиксировать значения**;
- **Фиксировать значения** – принудительно циклически записывает в переменную заданное пользователем значение. Пример использования – переменная привязана к аналоговому входу контроллера, к которому еще не подключен датчик, и поэтому она имеет значение 0.0. Чтобы сымитировать получаемое от датчика значение – необходимо использовать команду **Фиксировать значения** (если использовать команду **Записать значения,** то записанное пользователем значение сразу будет сброшено в 0.0, так как опрос датчика происходит циклически);
- **Освободить значения** – эта команда отключает фиксацию значений выбранных переменных;
- **Режим отображения** – переключает режим отображения целочисленных переменных (доступные режимы: BIN/DEC/HEX).

Вместе с командами меню **Отладка** удобно использовать дополнительные панели, которые можно открыть в меню «Вид»:

- **Просмотр – Watch** – позволяет создать список переменных для наблюдения за их значениями (с возможностью изменения при помощи команд **Записать значения** и **Фиксировать значения**). Переменные могут быть объявлены в разных программных объектах проекта;
- **Просмотр – Наблюдение всех фиксаций** – отображает все переменные проекта, находящиеся в состоянии фиксации;
- **Точки останова** – отображает все точки останова проекта.

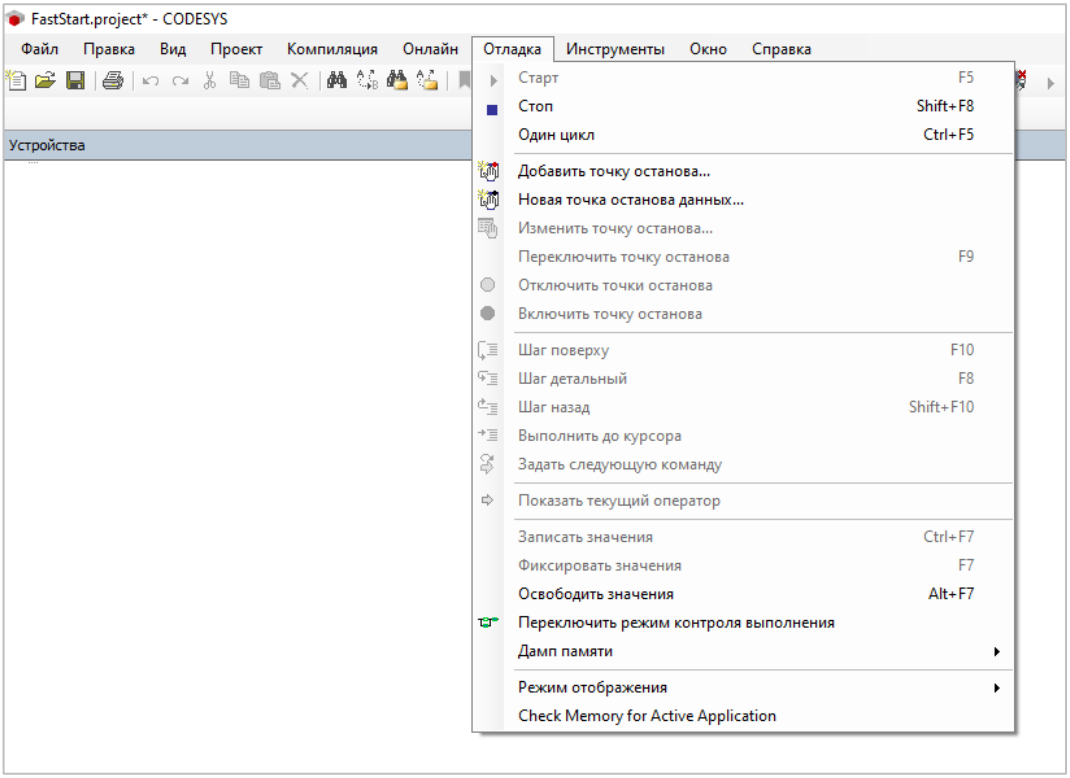
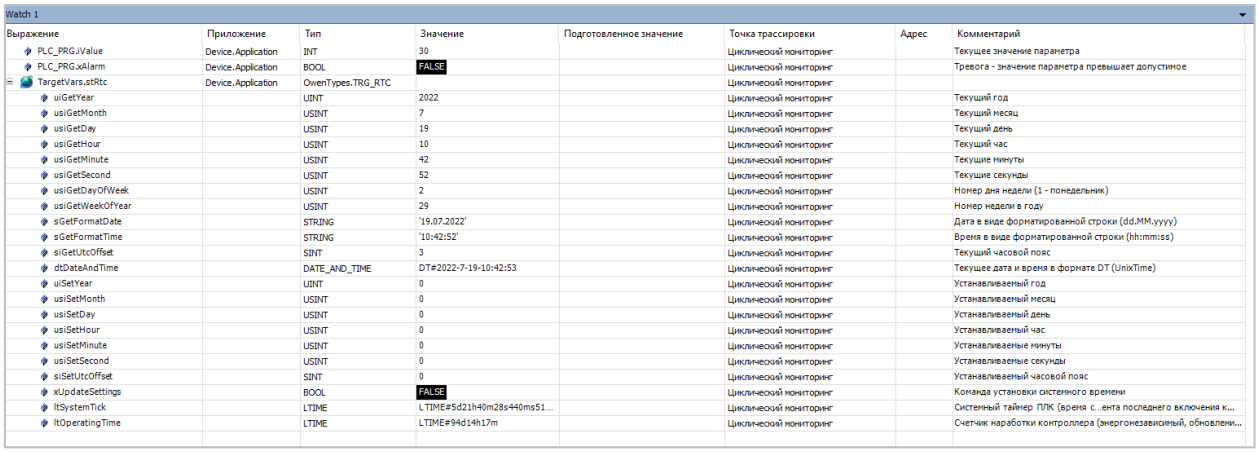


Рисунок 4.2.8 – Команды меню «Отладка»



Выражение	Приложение	Тип	Значение	Подготовленное значение	Точка трассировки	Адрес	Комментарий
PLC_PRG.iValue	Device.Application	INT	30		Циклический мониторинг		Текущее значение параметра
PLC_PRG.xAlarm	Device.Application	BOOL	FALSE		Циклический мониторинг		Тревога - значение параметра превышает допустимое
TargetVars.stRtc	Device.Application	OwenTypes.TRG_RTC			Циклический мониторинг		
uiGetYear		UINT	2022		Циклический мониторинг		Текущий год
usiGetMonth		USINT	7		Циклический мониторинг		Текущий месяц
usiGetDay		USINT	19		Циклический мониторинг		Текущий день
usiGetHour		USINT	10		Циклический мониторинг		Текущий час
usiGetMinute		USINT	42		Циклический мониторинг		Текущие минуты
usiGetSecond		USINT	52		Циклический мониторинг		Текущие секунды
usiGetDayOfWeek		USINT	2		Циклический мониторинг		Номер дня недели (1 - понедельник)
usiGetWeekOfYear		USINT	29		Циклический мониторинг		Номер недели в году
sGetFormatDate		STRING	'19.07.2022'		Циклический мониторинг		Дата в виде форматированной строки (dd.MM.yyyy)
sGetFormatTime		STRING	'10:42:52'		Циклический мониторинг		Время в виде форматированной строки (hh:mm:ss)
siGetUtcOffset		SINT	3		Циклический мониторинг		Текущий часовой пояс
dtDateAndTime		DATE_AND_TIME	DT#2022-7-19-10:42:53		Циклический мониторинг		Текущее дата и время в формате DT (UnixTime)
uiSetYear		UINT	0		Циклический мониторинг		Устанавливаемый год
usiSetMonth		USINT	0		Циклический мониторинг		Устанавливаемый месяц
usiSetDay		USINT	0		Циклический мониторинг		Устанавливаемый день
usiSetHour		USINT	0		Циклический мониторинг		Устанавливаемый час
usiSetMinute		USINT	0		Циклический мониторинг		Устанавливаемые минуты
usiSetSecond		USINT	0		Циклический мониторинг		Устанавливаемые секунды
siSetUtcOffset		SINT	0		Циклический мониторинг		Устанавливаемый часовой пояс
xUpdateSettings		BOOL	FALSE		Циклический мониторинг		Команда установки системного времени
ltSystemTick		LTIME	LTIME#5d21h40m28s440ms51...		Циклический мониторинг		Системный таймер ПЛК (время с...ента последнего включения к...
ltOperatingTime		LTIME	LTIME#94d14h17m		Циклический мониторинг		Счетчик наработки контроллера (энергонезависимый, обновлени...

Рисунок 4.2.9 – Внешний вид панели просмотра (Watch)

4.2.8 Меню «Инструменты»

Меню **«Инструменты»** содержит команды для установки в CODESYS дополнительных компонентов (пакетов, библиотек и т. д.), а также настройки среды разработки. Наиболее часто используемые команды:

- **CODESYS Installer** – используется для установки пакетов (в частности, пакетов таргет-файлов), обновлений CODESYS и создания окружений (см. п. 2.2);
- **Репозиторий библиотек** – используется для установки дополнительных библиотек (см. п. 2.3);
- **Настройка** – используется для настройки внешнего вида среды разработки (набора команд в меню, набора ярлыков на панели инструментов, сочетания «горячих клавиш»);
- **Опции** – используется для настройки CODESYS.

Например, в меню **Опции** можно:

- изменить язык интерфейса CODESYS (вкладка **Международные установки**);
- включить или отключить сетку в редакторе визуализации (вкладка **Визуализация**);
- настроить цвет и шрифт текста в редакторе языка ST (вкладка **Текстовый редактор**);
- и т. п.

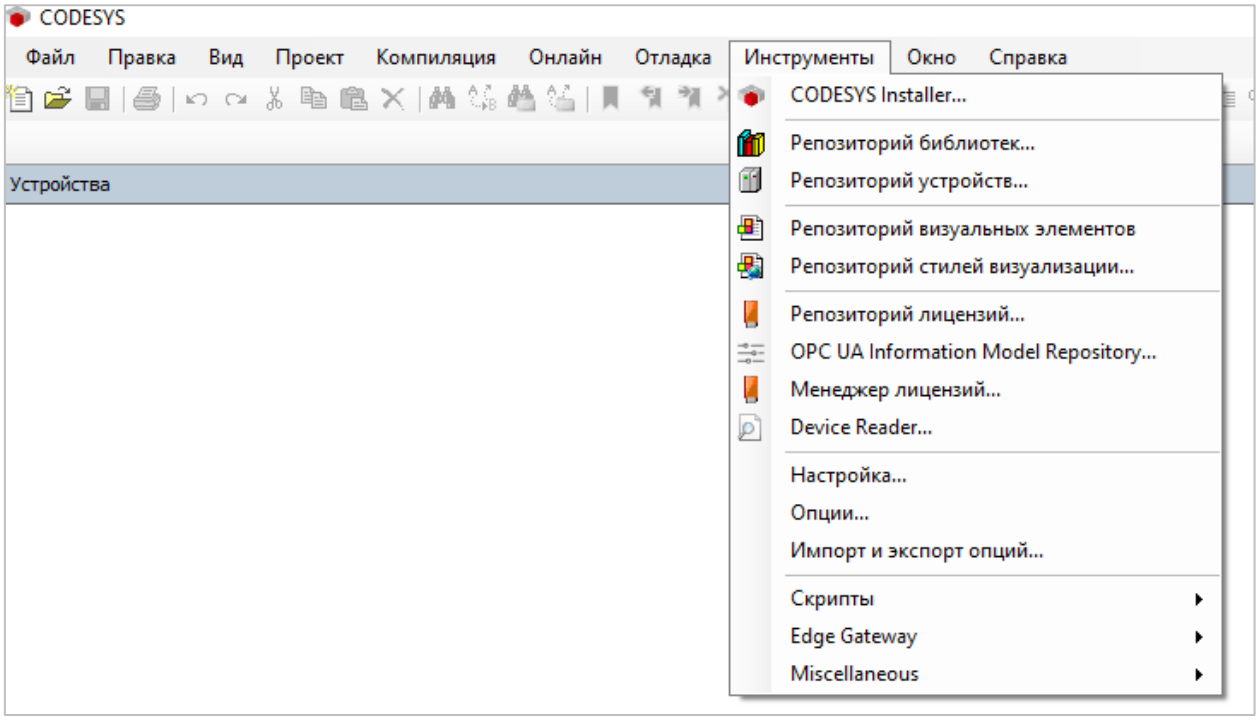


Рисунок 4.2.10 – Команды меню «Инструменты»

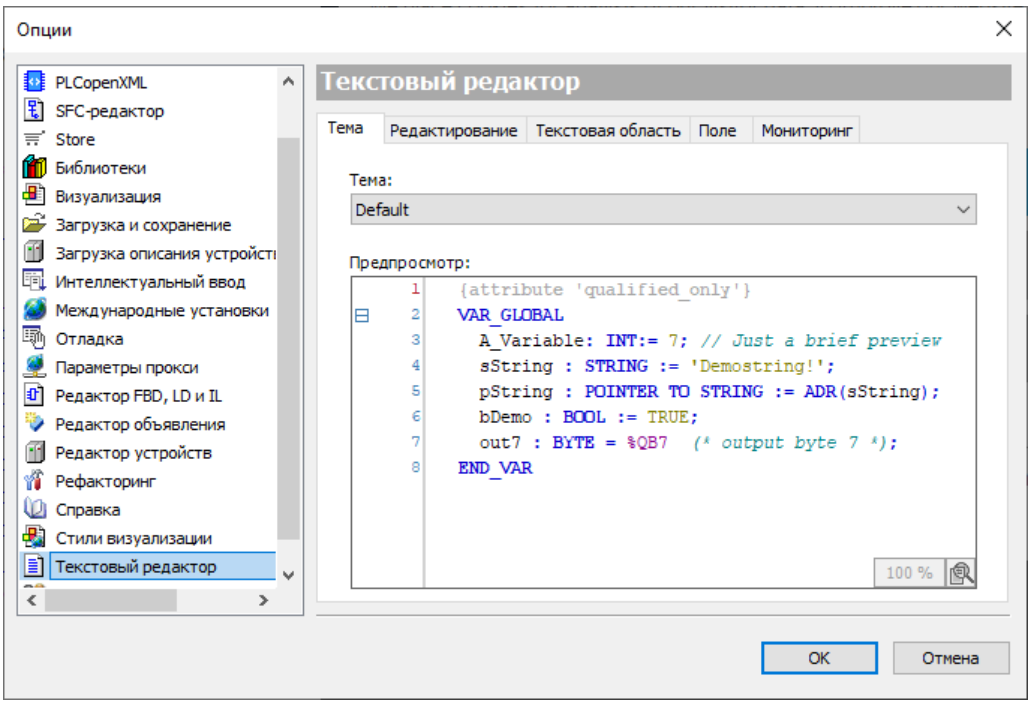


Рисунок 4.2.11 – Внешний вид меню опций

4.2.9 Меню «Окно»

Меню **«Окно»** содержит команды для переключения и закрытия открытых в среде вкладок и панелей.

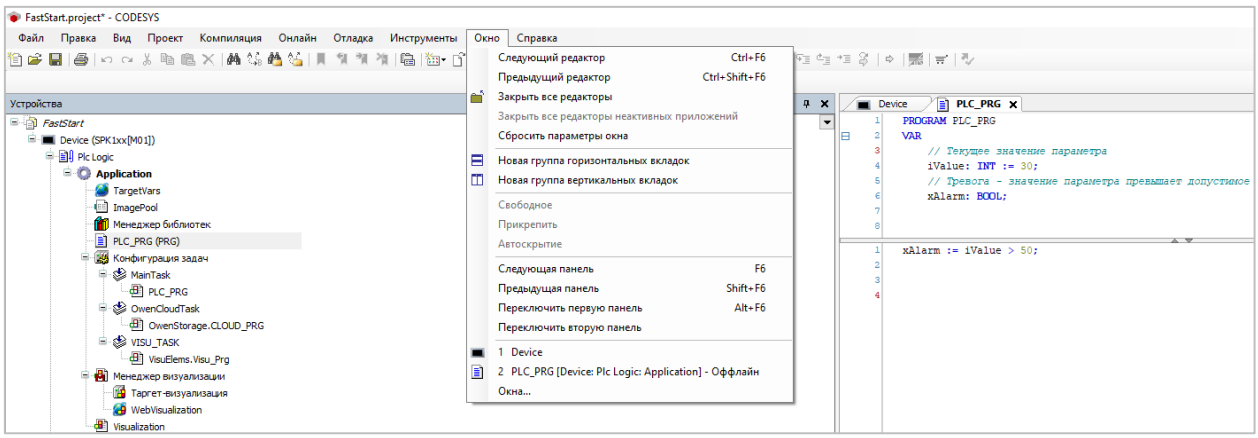


Рисунок 4.2.12 – Команды меню «Окно»

4.2.10 Меню «Справка»

Меню **«Справка»** содержит команды для перехода в онлайн-справку CODESYS (при отсутствии подключения к интернету открывается офлайн-версия справки) и отображения информации об используемой версии среды («О программе»).

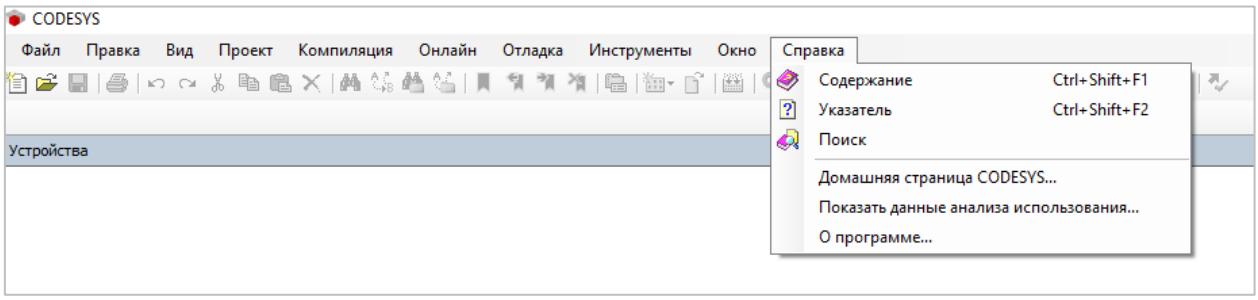


Рисунок 4.2.13 – Команды меню «Справка»

4.3 Дерево проекта

4.3.1 Узлы контроллера и компонентов

Дерево проекта содержит древовидную структуру используемых в проекте объектов. Объектом верхнего уровня является контроллер (**Device**). В проекте может быть несколько контроллеров; CODESYS предоставляет удобные средства настройки обмена между контроллерами проекта, которые в рамках «первого старта» не рассматриваются (см. подробности в документе **CODESYS V3.5. Настройка связи между ПЛК**).

Изначально проект включает в себя один контроллер. Если требуется добавить в проект другие контроллеры – нажмите правой кнопкой мыши на свободную область дерева проекта и в контекстном меню выберите команду **Добавить устройство**.

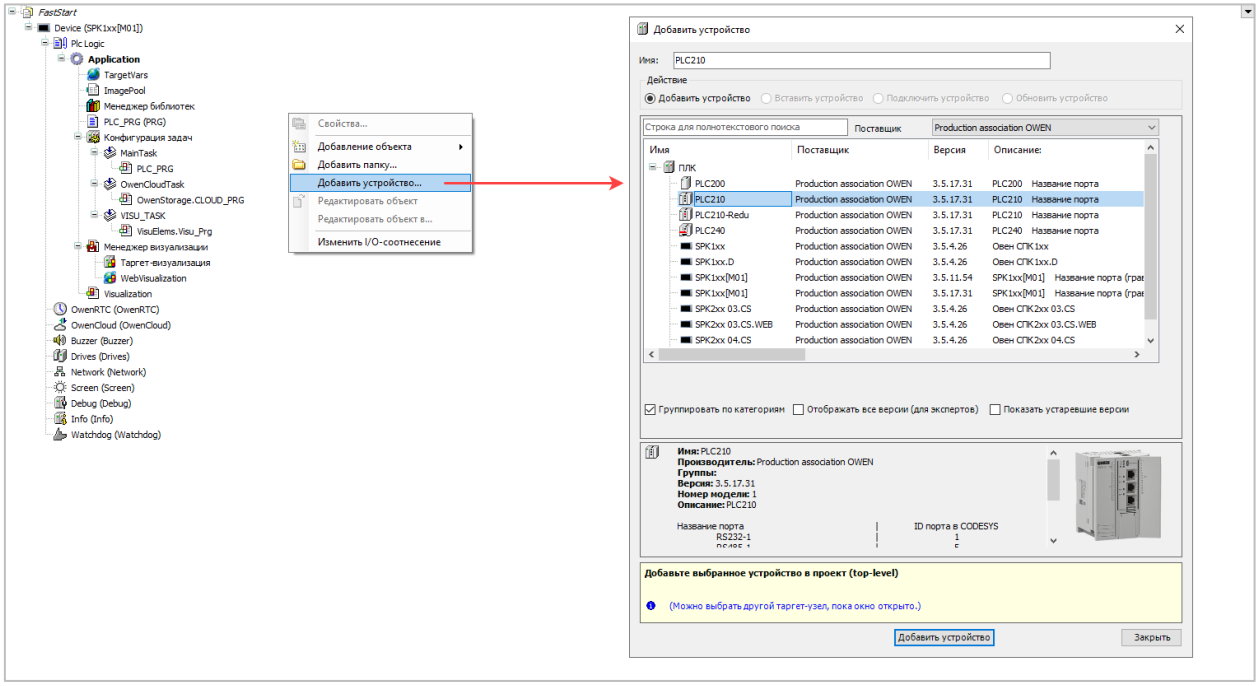


Рисунок 4.3.1 – Добавление еще одного контроллера в проект CODESYS

Узел контроллера включает в себя приложение (**Application**) и дополнительные компоненты (например, узлы коммуникационных драйверов, системные узлы таргет-файла и т. д.). Для добавления компонента нажмите на узел контроллера (по умолчанию он имеет название **Device**) правой кнопкой мыши и в контекстном меню выберите команду **Добавить устройство**:

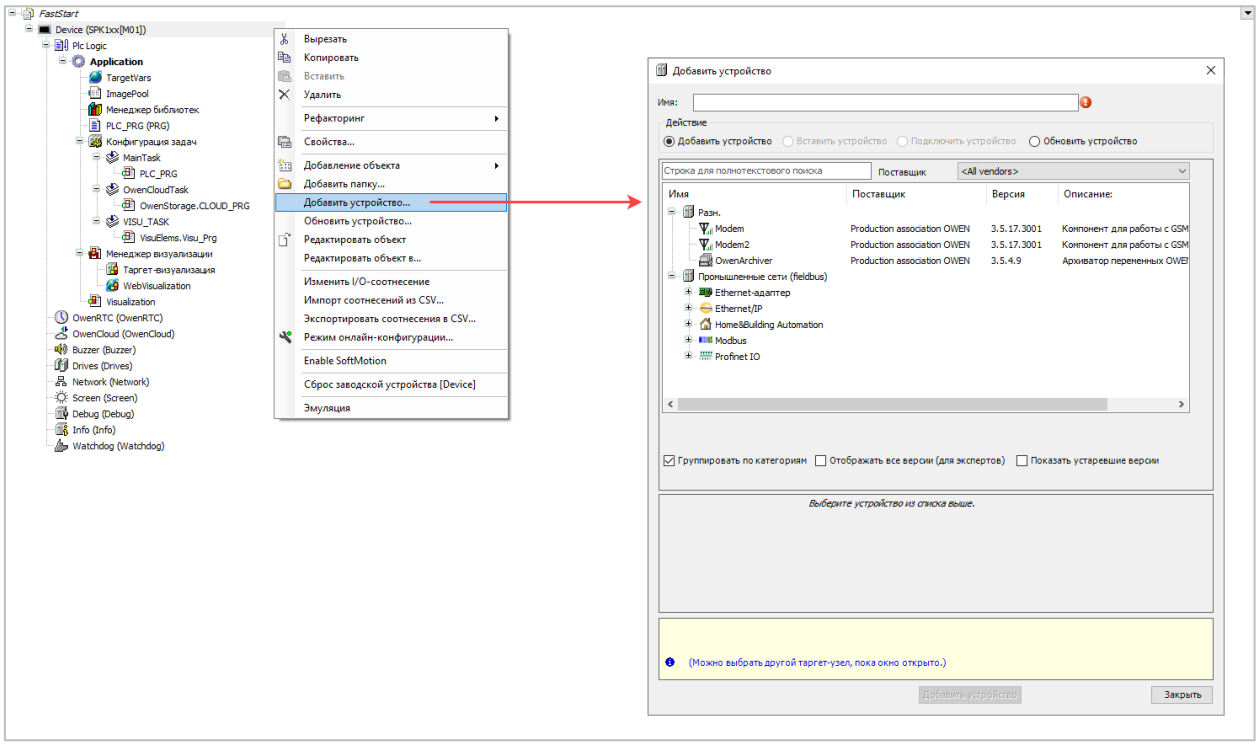


Рисунок 4.3.2 – Добавление компонента

На рис. 4.3.2 отображаются компоненты протоколов, поддержанные в среде CODESYS – Ethernet/IP, Profinet и т. д. Но это не означает, что эти протоколы можно использовать на любом контроллере – для работы с ними требуется наличие лицензии. Контроллеры ОВЕН имеют лицензию на протоколы Modbus Serial, Modbus TCP и OPC UA (режим «сервер»). Возможность активации лицензий для других протоколов в настоящий момент не поддерживается; но следует отметить, что ряд дополнительных протоколов реализован разработчиками ОВЕН в виде библиотек – например, SNMP, протоколы тепло- и электросчетчиков и др.

Некоторые компоненты могут включать в себя дочерние компоненты. Для их добавления нужно нажать на компонент правой кнопкой мыши и использовать команду контекстного меню **Добавить устройство**. На рис. 4.3.3 показано, как в компонент **Modbus COM** можно добавить компонент **Modbus Master** или **Modbus Serial Device** – таким образом определяется режим работы контроллера в рамках выбранного COM-порта (Master или Slave).

С помощью команды **Обновить устройство** можно изменить версию компонента (например, обновить версию таргет-файла контроллера или другого компонента) или заменить один компонент на другой.

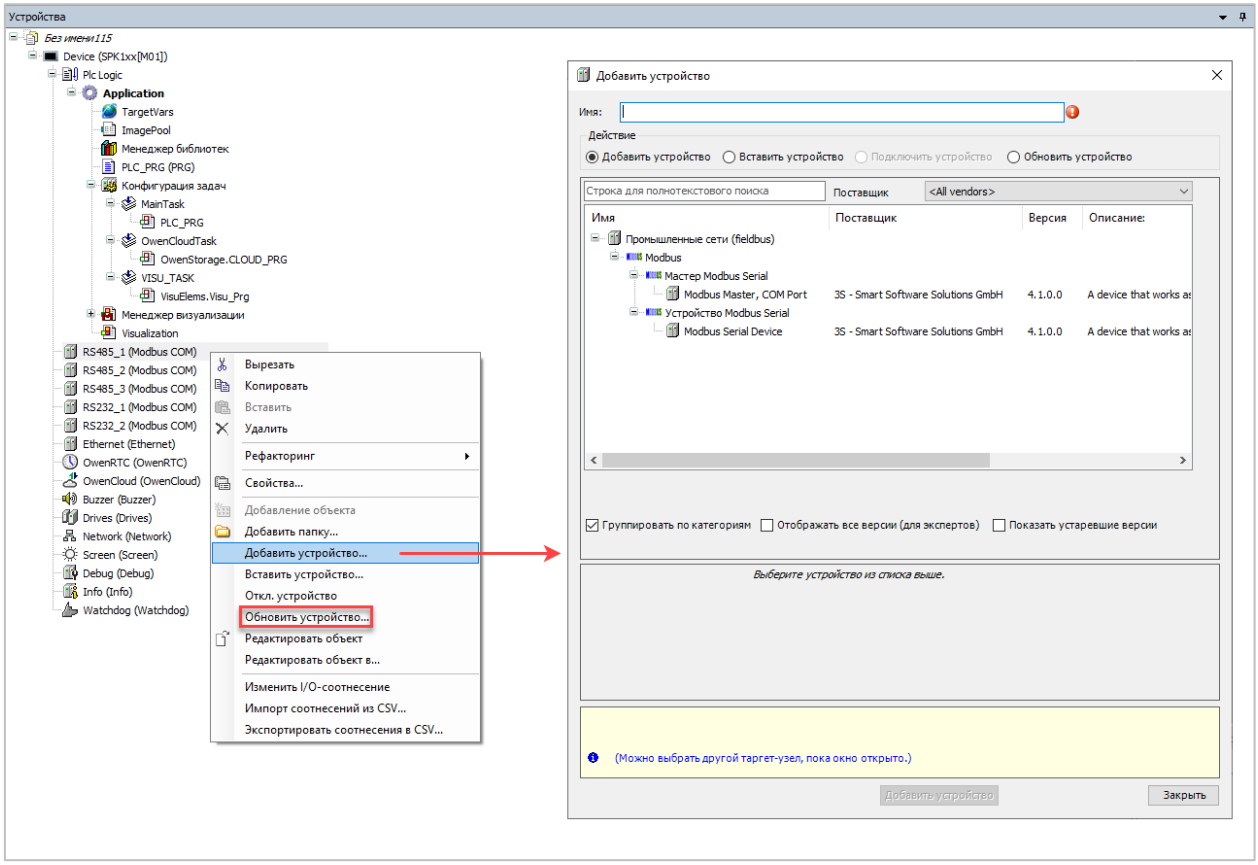


Рисунок 4.3.3 – Добавление в компонент дочернего компонента

Таргет-файлы контроллеров ОВЕН включают ряд системных узлов, каждый из которых упрощает работу с определенным функционалом ПЛК. Эти узлы нельзя удалить из дерева проекта. Ниже приведено их обзорное описание:

- **OwenRTC** – позволяет считать/записать системное время контроллера;
- **OwenCloud** – определяет настройки связи с облачным сервисом OwenCloud. Настройка связи с облачным сервисом рассмотрена в п. 6.10;
- **Buzzer** – позволяет управлять пьезоизлучателем (зуммером) контроллера;
- **Drives** – позволяет считать информацию о памяти контроллера и подключенных накопителей, а также определить, что накопитель примонтирован и демонтировать его;
- **Network** – позволяет считать/записать сетевые настройки контроллера (*этот узел присутствует только у контроллеров СПК*);
- **Screen** – позволяет управлять яркостью подсветки экрана контроллера и настроить параметры «спящего режима» экрана (*этот узел присутствует только у контроллеров СПК*);
- **Debug** – позволяет считать значения отладочных параметров (загрузку процессора и т. д.);
- **Info** – позволяет считать информацию о контроллере и проекте (версию прошивки, серийный номер, название проекта и т. д.);
- **Watchdog** – позволяет получить информацию от сторожевого таймера контроллера (информацию о последнем исключении, счетчики перезагрузок и т. д.) и перезагрузить контроллер из кода программы.

Подробное описание системных узлов таргет-файла приведено в документе **CODESYS V3.5. Описание таргет-файлов**. У контроллеров ПЛК210 и ПЛК200 также присутствуют специальные узлы для работы со входами и выходами.

Все системные узлы имеют вкладку **Соотнесение входов-выходов**, на которой происходит привязка переменных проекта к каналам компонента. В шаблоне проекта ко всем каналам всех узлов переменные уже привязаны – мы расскажем об этом в п. 4.3.4.

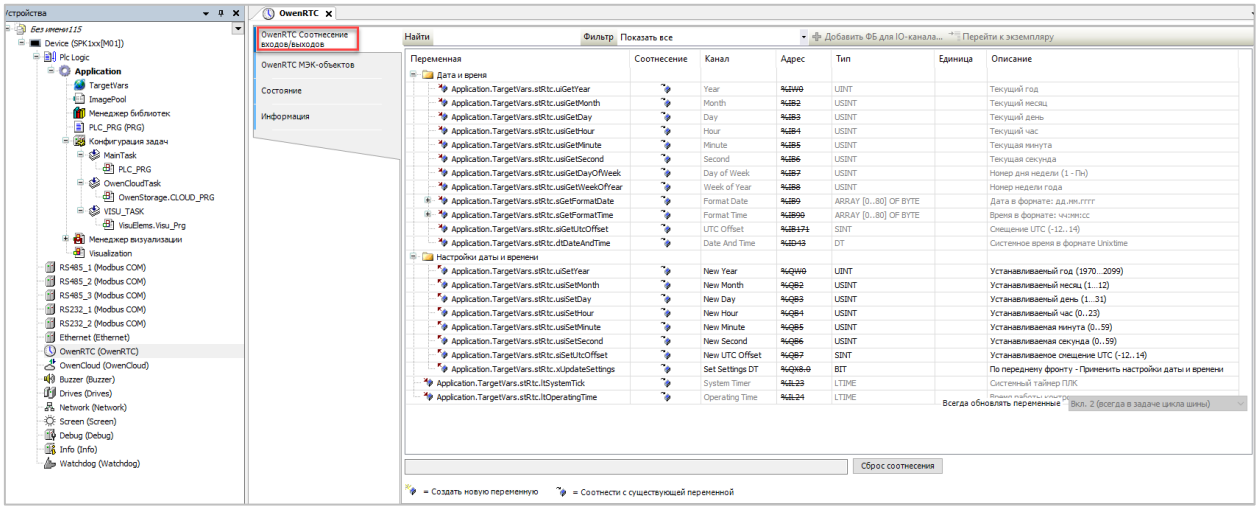


Рисунок 4.3.4 – Привязка переменных к каналам компонента

У некоторых узлов также есть вкладка **Конфигурация**. На ней пользователь задает значения параметров, которые уже нельзя будет изменить в процессе работы приложения (то есть для их изменения потребуется загрузить в контроллер новое приложение).

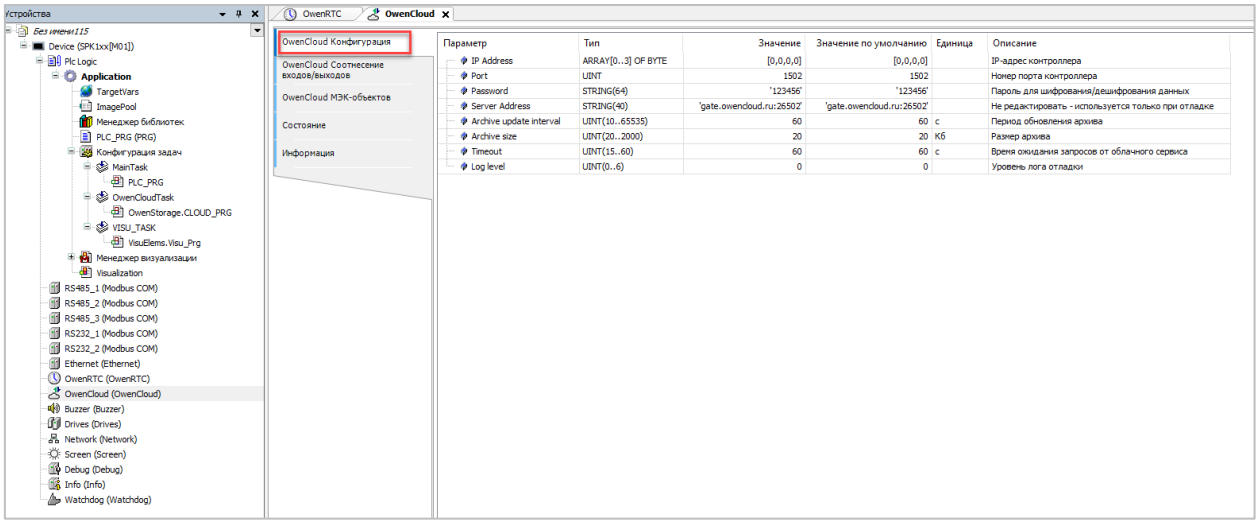


Рисунок 4.3.5 – Настройка конфигурационных параметров компонента

4.3.2 Узел «Application»

Узел **«Application»** включает в себя объекты, входящие в состав приложения контроллера – программы, функциональные блоки, экраны визуализации и др. Проекты для контроллеров ОВЕН содержат только одно приложение с названием **Application** (переименовать его нельзя).

Для добавления в проект какого-либо объекта нужно нажать на узел **Application** правой кнопкой мыши и выбрать команду **Добавление объекта**.

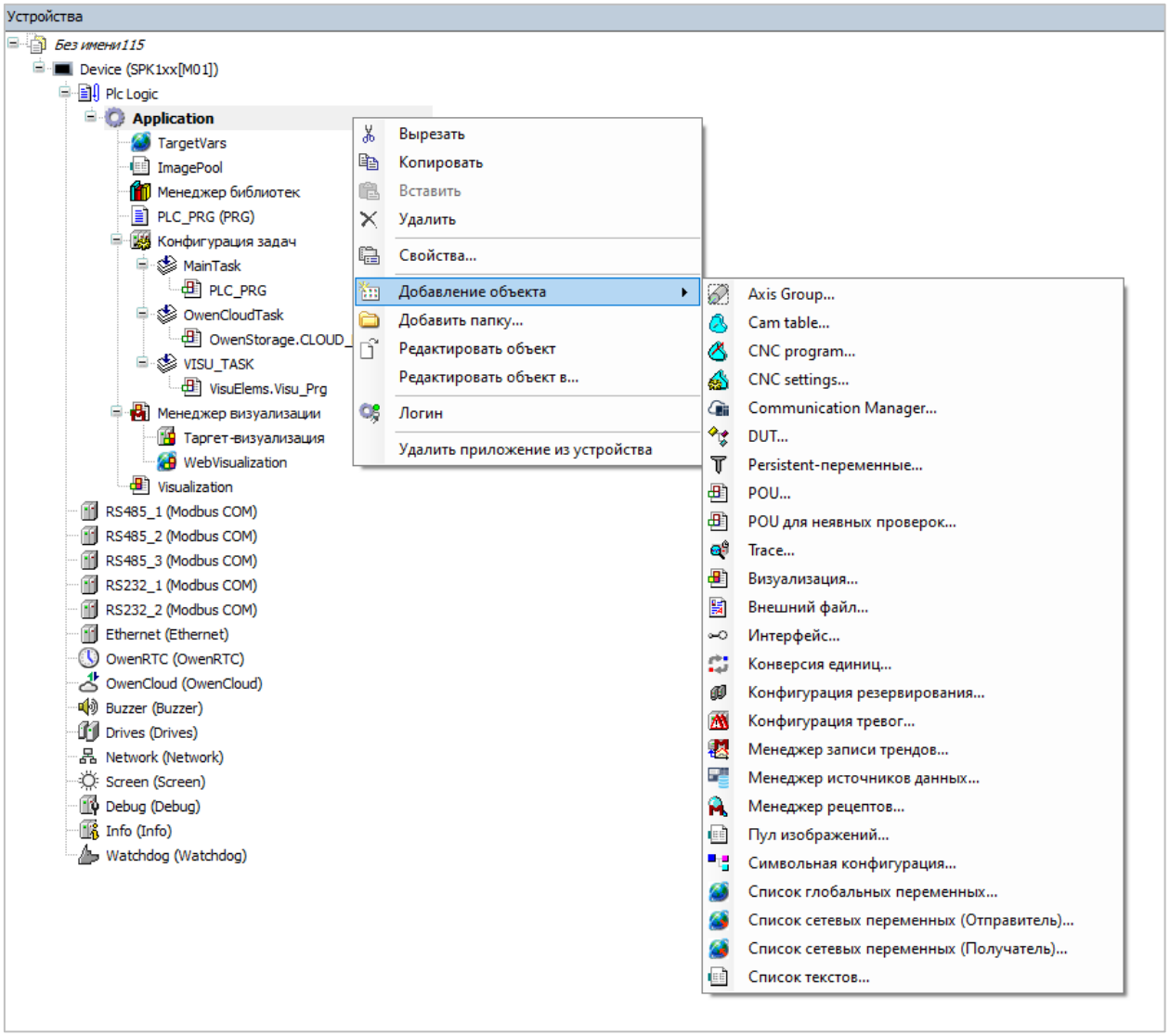


Рисунок 4.3.6 – Добавление объектов в приложение

По структурной принадлежности объекты можно разделить на несколько категорий:

Таблица 4.1 – Структурная принадлежность объектов приложения

Категория	Объекты
Объекты, не поддерживаемые контроллерами ОВЕН	Axis Group, Cam table, CNC Program, CNC Settings, Конфигурация резервирования
Объекты программной части проекта	DUT, Persistent-переменные, POU, POU для неявных проверок, Интерфейс, Список глобальных переменных, Конфигурация задач, Менеджер библиотек
Объекты визуализации	Визуализация, Менеджер визуализации, Конверсия единиц, Конфигурация тревог, Менеджер записи трендов, Менеджер рецептов, Пул изображений, Список текстов
Объекты коммуникации с др. устройствами	Communication Manager, Менеджер источников данных, Символьная конфигурация, Список сетевых переменных
Остальные объекты	Trace, Внешний файл

Некоторые объекты могут включать дочерние объекты (подобъекты) – например, на рис. 4.3.6 в объект **Менеджер визуализации** добавлены дочерние объекты **Таргет-визуализация** и **WebVisualization**.

Часть объектов уже была обзорно рассмотрена в п. 3.1, и еще некоторые мы рассмотрим в п. 6 в процессе создания демо-проекта.

4.3.3 Вкладка «POU»

Как мы уже упоминали в начале п. 4.3.1 – проект CODESYS может включать в себя несколько контроллеров. В таких случаях часто требуется, чтобы некоторые объекты (например – функциональные блоки) могли использоваться в процессе разработки приложений всех контроллеров проекта.

Такие «общедоступные» объекты размещаются на вкладке **POU**. Для переключения на эту вкладку нажмите одноименную кнопку в самом низу дерева проекта или используйте команду **POU** из меню «Вид».

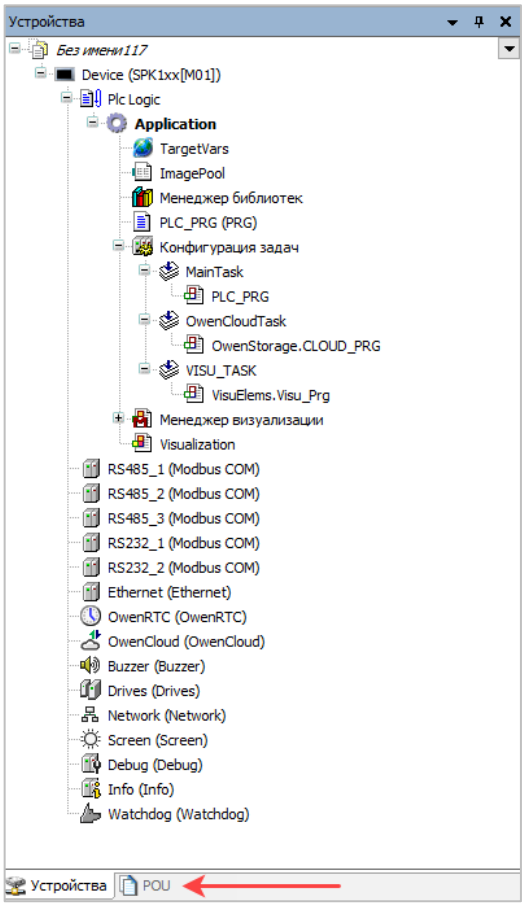


Рисунок 4.3.7 – Переключение на вкладку POU

На этой вкладке по умолчанию уже присутствует несколько объектов:

- **GlobalTextList** – глобальный список текстов, содержащий статические тексты визуализации проекта;
- **Менеджер библиотек** – аналогичен менеджеру библиотек приложения, но добавленные в него библиотеки будут доступны во всех приложениях проекта. Подробнее о библиотеках см. в п. 5.10;
- **Информация о проекте** и **Установки проекта** – кнопки перехода в соответствующие меню, аналогичные одноименным командам меню «Проект».

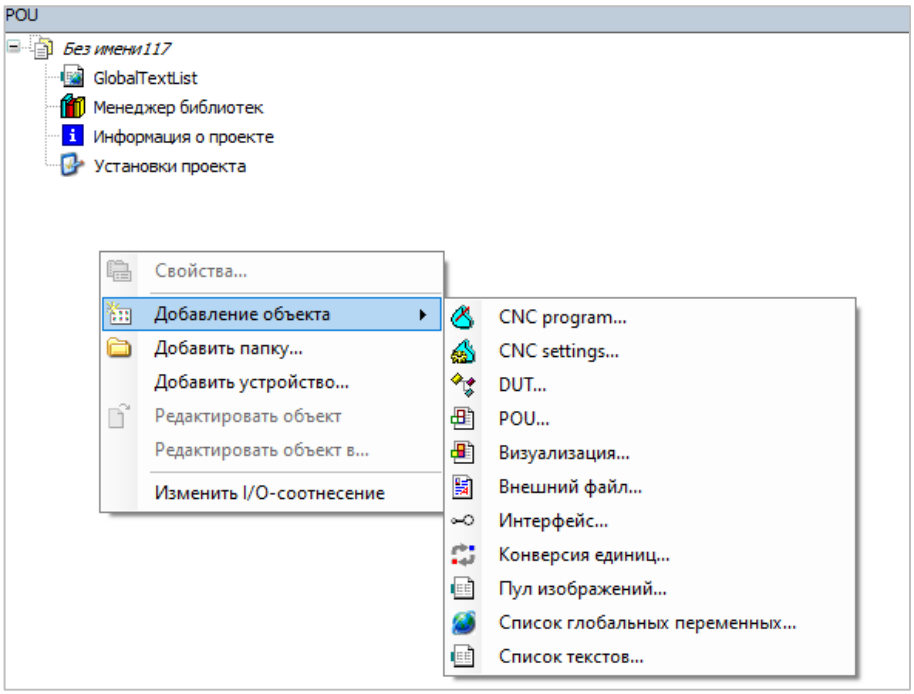


Рисунок 4.3.8 – Добавление объекта на вкладку POU

4.3.4 Состав шаблона проекта

В п. 3.1 мы создали проект на основе шаблона. Шаблоны проектов для контроллеров ОВЕН устанавливаются вместе с пакетом таргет-файлов ОВЕН.

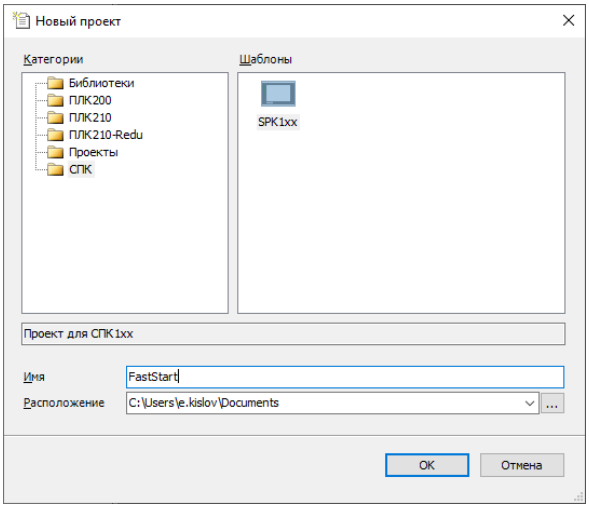


Рисунок 4.3.9 – Выбор шаблона проекта

В отличие от «стандартного» проекта (его можно выбрать в папке **Проекты**) шаблон дополнительно включает в себя:

- список глобальных переменных **TargetVars**. Этот список содержит экземпляры структур, переменные которых привязаны к системным узлам таргет-файла. Мы еще не говорили о структурах и глобальных переменных (расскажем об этом в п. 5.3.8 и п. 5.4.6), но вкратце – это позволяет на уровне шаблона получить обширный набор «системных» переменных – например, для получения системного времени контроллера, чтения его серийного номера, работы с его входами/выходами и т. д.;
- пул изображений **ImagePool**. В этот объект добавляются графические файлы, которые потом можно использовать при создании визуализации;
- менеджер визуализации с настройками, подходящими для конкретной модели контроллера (например, для панельных контроллеров СПК указано корректное разрешение экрана);
- экран визуализации **Visualization**;
- коммуникационные компоненты для интерфейсов RS-485/RS-232 и Ethernet. Для RS-485/RS-232 сразу указаны идентификаторы COM-портов контроллера. Если в рамках проекта какие-то интерфейсы не используются – то их компоненты рекомендуется удалить из дерева проекта.

В менеджер библиотек шаблона добавлен ряд библиотек, используемых в большинстве проектов:

- **OwenTypes** – библиотека структур параметров системных узлов таргет-файлов ОВЕН;
- **Standard64** – библиотека с функциями для работы со строками типа WSTRING, 64-битными счетчиками импульсов и т. д.;
- **Util** – библиотека прикладных функциональных блоков (блок линейного масштабирования, блок гистерезиса, ПИД-регулятор и т. д.);
- **CAA Memory** – библиотека для работы с памятью (копирование блоков памяти, заполнение блоков памяти заданным значением и т. д.).

Файлы шаблонов хранятся в директории установки CODESYS в папке **…/CODESYS/Templates** – так что вы можете отредактировать их «под себя» или создать собственные шаблоны.

5 Основы программирования

В этом разделе мы изучим средства, необходимые для программирования ПЛК на языках стандарта МЭК 61131-3, сделав акцент на языке ST. Рассказывать об этих средствах последовательно, от простых к сложным, довольно тяжело – например, особенности некоторых типов данных невозможно продемонстрировать без конкретных операторов и т. д. Поэтому иногда мы будем «забегать вперед» и использовать термины и определения, еще не получившие раскрытия. Подразумевается, что к концу раздела фрагменты отдельных пунктов сложатся в цельную мозаику. Если этого не случится – попробуйте перечитать раздел во второй раз (уже имея представления обо всех базовых понятиях) или напишите критику по поводу раздела на почту e.kislov@owen.ru – она непременно будет учтена при подготовке следующих версий документа.

Для начала давайте обзорно рассмотрим архитектуру приложения по принципу «сверху вниз»:

- каждое приложение содержит объект **Конфигурация задач**, в котором созданы одна или несколько циклически выполняемых **задач**, определяющих интервалы вызова программ;
- к каждой задаче привязана одна или несколько **программ** – крупных «строительных» блоков приложения, реализующих общий алгоритм системы управления или его отдельную существенную часть;
- программы включают в себя **переменные**, **операторы** (обычные и условные), а также программные объекты – **функции** и **функциональные блоки**, которые позволяют избежать копирования повторяющихся фрагментов кода и повышающих модульность приложения;
- функции и функциональные блоки включают в себя переменные, операторы (обычные и условные) и вызовы функций. Функциональные блоки могут включать в себя вызовы других функциональных блоков, но вот внутри функций почти никогда не производится вызов функциональных блоков;
- операторы представляют собой действия, совершаемые над переменными;
- переменные – это самые простейшие из «строительных блоков», представляющие собой именованные сегменты памяти. Они принадлежат какому-либо типу (например, целочисленному или строковому) и объявлены в некой области (например, области входов, выходов или локальных переменных). Области могут помечаться специальными модификаторами для обеспечения энергонезависимости переменных.

В следующих пунктах мы достаточно детально рассмотрим элементы этой архитектуры, двигаясь по принципу «снизу вверх».

5.1 Языки программирования стандарта МЭК 61131-3

Стандарт МЭК 61131 определяет требования к программируемым логическим контроллерам. Третий раздел стандарта (обычно обозначаемый как МЭК 61131-3) посвящен языкам программирования контроллеров. Первая редакция этого раздела стандарта была выпущена в 1993 году. На данный момент актуальной редакцией является третья, выпущенная в 2013 – в ней, в частности, добавили средства объектно-ориентированного программирования (ООП).

Стандарт МЭК 61131-3 определяет 5 языков программирования контроллеров:

- **LD** – графический язык релейно-контактных схем. Исторически первый язык программирования ПЛК;
- **FBD** – графический язык функциональных блоков;
- **SFC** – графический язык диаграмм состояний. В отличие от других языков стандарта является языком проектирования, а не программирования. SFC позволяет описать структуру программы в виде конечного автомата, состоящего из шагов, переходов между шагами и действий, выполняемых на этих шагах. Действия должны быть закодированы на одном из других языков стандарта;
- **IL** – низкоуровневый текстовый язык, похожий на язык ассемблера. В третьей редакции стандарта признан устаревшим («deprecated»). В свежих версиях CODESYS по умолчанию отключен; включить его поддержку можно в меню **Инструменты – Опции – Редактор FDB, LD и IL – IL**;
- **ST** – высокоуровневый текстовый язык программирования, похожий на Pascal.

CODESYS поддерживает все 5 языков стандарта МЭК 61131-3 и в дополнение к ним – еще один графический язык функциональных блоков, который называется **CFC** (не путайте с SFC). Отличия между FBD и CFC заключаются в следующем:

- FBD подразумевает «жесткое» размещение блоков на «холсте» – каждый новый добавляемый блок должен быть «сцеплен» с одним из существующих. То есть блоки добавляются по одному и сразу привязываются к одному из уже ранее добавленных блоков. Порядок выполнения блоков определяется порядком их размещения;
- CFC поддерживает свободное размещение блоков на «холсте» – пользователь может сначала добавить на «холст» множество блоков, а потом уже «протянуть» связи между ними (в том числе, поддерживаются обратные связи – когда выход блока заводится на вход одного из предыдущих блоков). Порядок выполнения блоков определяется либо автоматически в зависимости от их взаимного расположения (выполнение производится слева направо/сверху вниз), либо вручную задается пользователем для каждого блока.

На рис. 5.1.1 – 5.1.6 приведены примеры кода на различных языках стандарта МЭК 61131-3.

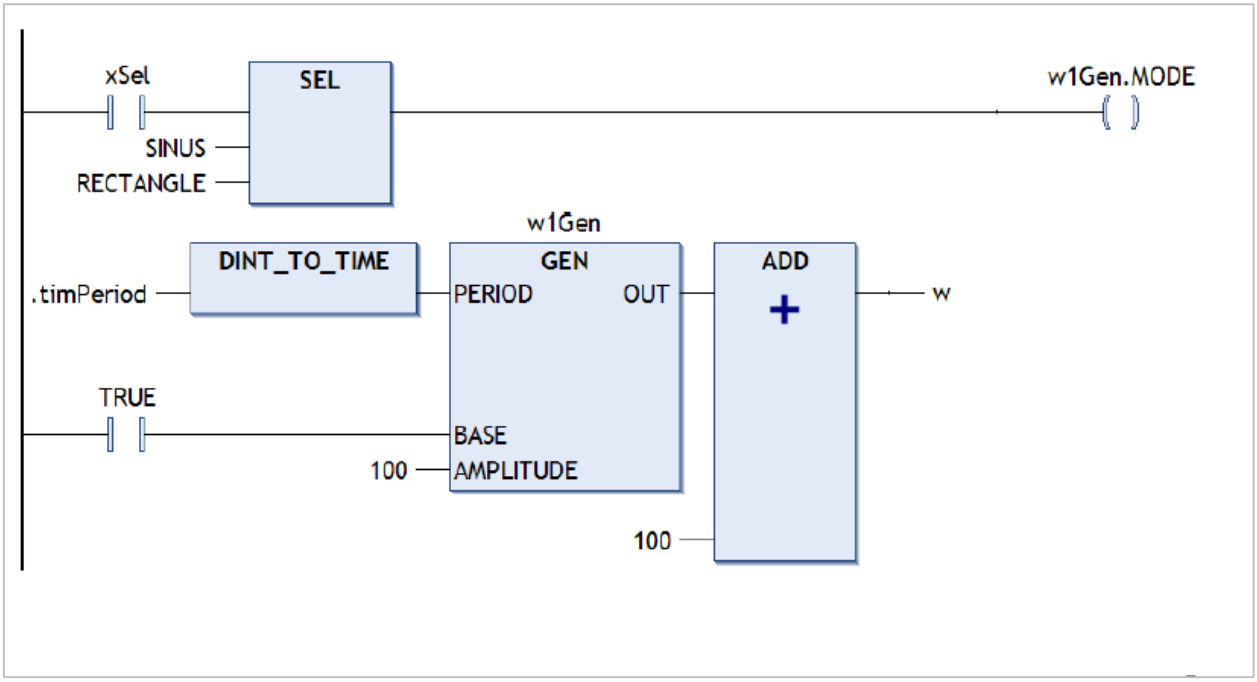


Рисунок 5.1.1 – Пример программного кода на языке LD

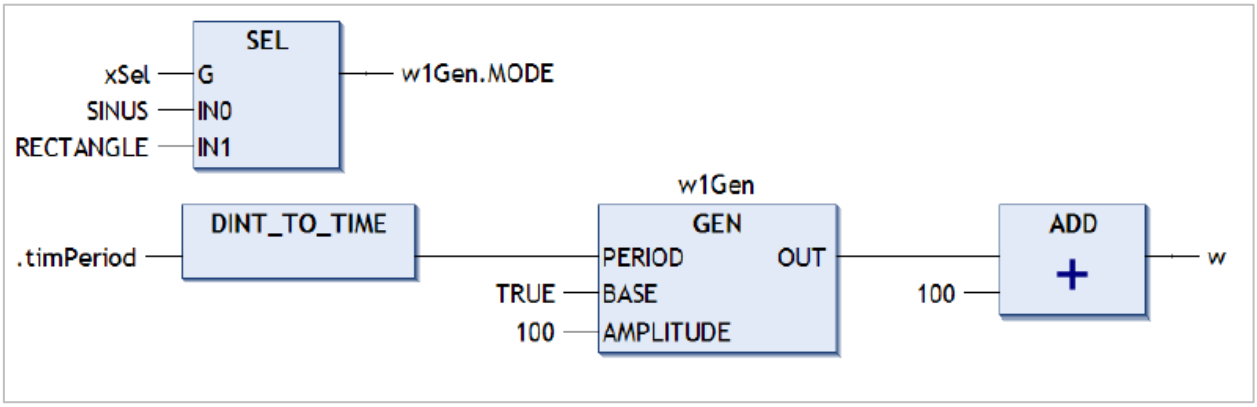


Рисунок 5.1.2 – Пример программного кода на языке FBD

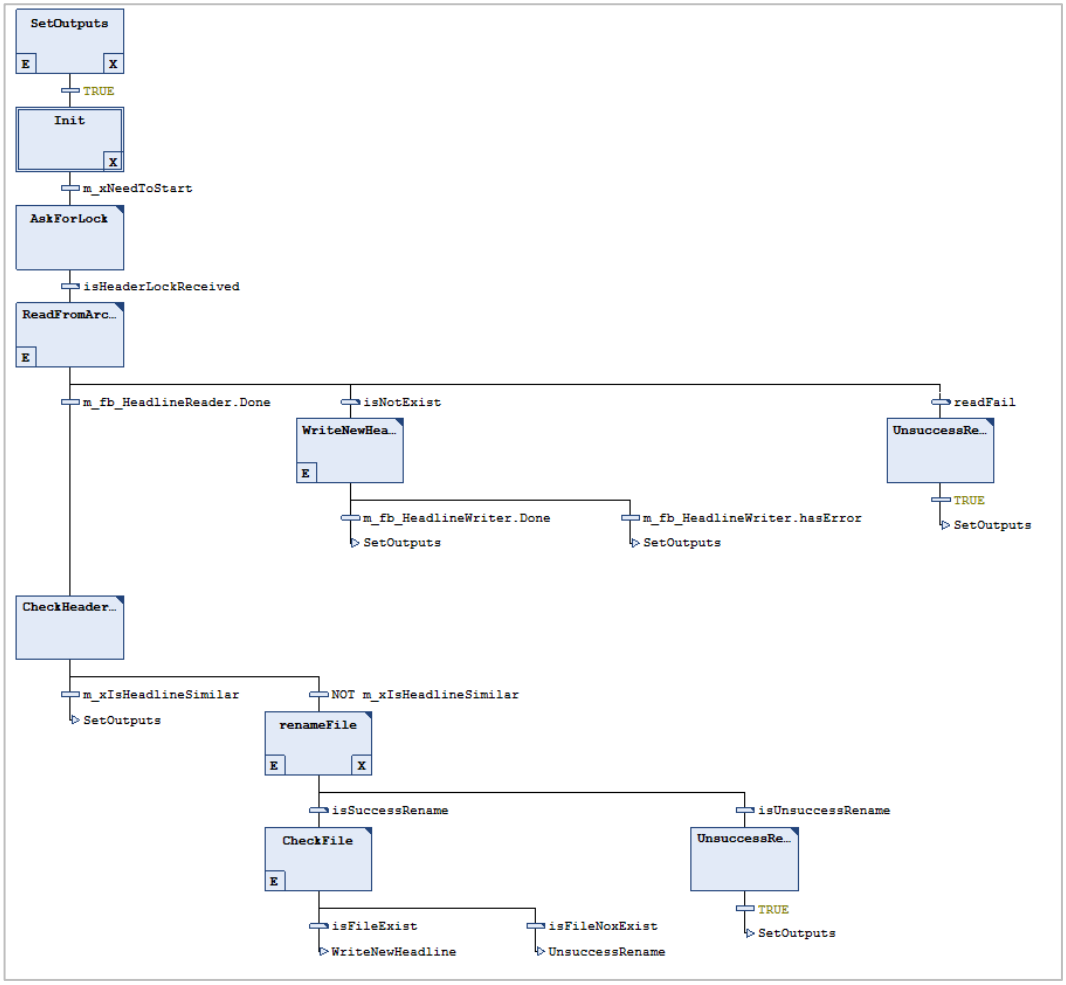


Рисунок 5.1.3 – Пример программного кода на языке SFC

```
LD                  xSel
SEL                 SINUS
                    RECTANGLE
ST                  w1Gen.MODE
LD                  .timPeriod
DINT_TO_TIME
ST                  w1Gen.PERIOD
CAL                 w1Gen(
                BASE:= TRUE,
           AMPLITUDE:= 100)
LD                  w1Gen.OUT
ADD                 100
ST                  w
```

Рисунок 5.1.4 – Пример программного кода на языке IL

```
_uiBufferSize := UDINT_TO_UINT (szBuffer);

Mem.MemMove
(
    pSource         := pBuffer,
    pDestination    := ADR (_abyBuffer),
    uiNumberOfBytes := _uiBufferSize
);

FOR i := 0 TO UINT_TO_INT (_uiBufferSize) - 1 DO
    _abyBuffer [i] := Util.BCD_TO_BYTE (_abyBuffer [i]);
    _usiExponent := INT_TO_USINT (UINT_TO_INT (_uiBufferSize) -1 - i);
    IF _usiExponent > 0 THEN
        _udiValue := _udiValue + LREAL_TO_UDINT(_abyBuffer [i] * EXPT (10, UINT_TO_INT (_uiBufferSize) - i));
    ELSE
        _udiValue := _udiValue +_abyBuffer [i];
    END_IF
END_FOR

IF uiAccuracy > 0 THEN
    _rValue := UDINT_TO_REAL (_udiValue) / uiAccuracy;
ELSE
    _rValue := 0.0;
END_IF

GetRealByHEXBytes := _rValue;
```

Рисунок 5.1.5 – Пример программного кода на языке ST

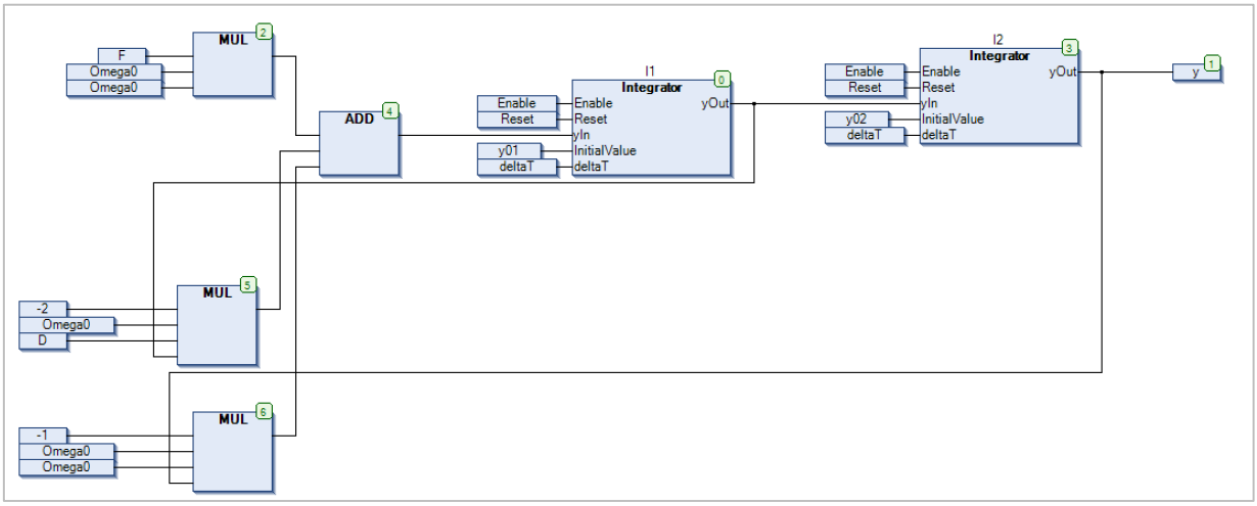


Рисунок 5.1.6 – Пример программного кода на языке CFC

Итак, в CODESYS доступно 6 языков программирования. На практике некоторые из них используются довольно редко:

- язык IL, как уже упоминалось, признан устаревшим. Поскольку CODESYS поддерживает все остальные языки стандарта МЭК 61131-3 – то IL используется предельно редко;
- языки LD и FBD в реализации CODESYS практически идентичны, разница только в отображении логических операторов («И», «ИЛИ» и т. д.) – сравните рис. 5.1.1 и 5.1.2. На практике вместо этих языков обычно используется не включенный в стандарт язык CFC – за счет свободного размещения блоков он оказывается более удобным и гибким;
- язык SFC не получил особого распространения, хотя довольно нагляден для графического описания пошаговых процессов управления с ограниченным числом шагов.

Таким образом, из всех 6 доступных языков большинство пользователей применяет только CFC и ST. Язык ST обычно используют инженеры, у которых уже есть опыт разработки ПО на языках общего назначения (C, Java и т. д.). Язык CFC в основном применяют инженеры без подобного опыта – инженеры КИПиА, инженеры-технологи и т. д.

В рамках этого документа мы будем использовать только язык ST.

Надо отметить, что CODESYS позволяет в рамках одного проекта использовать разные языки программирования – например, одна программа может быть написана на языке ST, другая – на CFC и т. д.

5.2 Переменные

Программа представляет собой набор операций, выполняемых над данными, размещенными в памяти контроллера. Например:

- контроль значения температуры и сигнализация о ее выходе за аварийные пределы;
- ротация насосов по времени наработки;
- включение/отключение освещения в зависимости от времени суток и срабатывания датчиков движения;
- и т. д.

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. В CODESYS объявление переменных производится в верхней области редактора языка программирования. Вспомним, как мы объявляли переменные проекта, который был создан в п. 3:

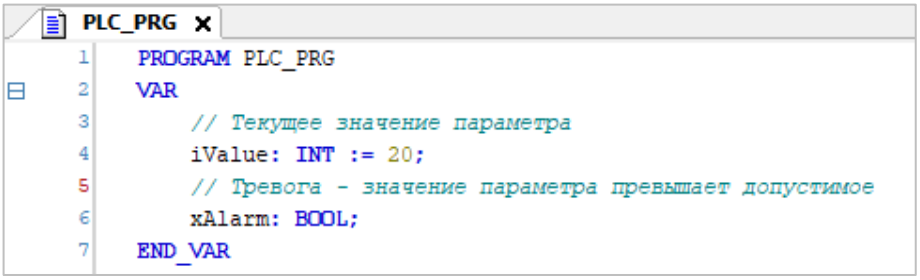


Рисунок 5.2.1 – Объявление переменных

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

```
PROGRAM PLC_PRG
VAR
   // однострочный комментарий
   (*
      многострочный
      комментарий
    *)
   iVar:          INT;
END_VAR
```

Имена переменных (и всех остальных объектов CODESYS) подчиняются следующим правилам:

- имя не должно содержать пробелов и спецсимволов (например, !, @ и т. д.). Исключение – символ нижнего подчеркивания (_);
- имя не должно начинаться с цифры;
- имя не должно содержать несколько символов нижнего подчеркивания (_), размещенных подряд (т.е. имя **i__Test** недопустимо, а имя **_i_Test** – допустимо);

- регистр имен объектов не учитывается (**iTest** и **iTEST** будет интерпретироваться как одно и то же имя);
- на длину имени не накладывается никаких ограничений;
- имя не должно совпадать с одним из зарезервированных ключевых слов (например, VAR, INT и т. д.);
- рекомендуется для имен переменных использовать венгерскую нотацию (вариант для CODESYS) и стиль lowerCamelCase.

По умолчанию CODESYS поддерживает только латиницу в именах переменных. Включить поддержку кириллицы можно (**Проект – Установки проекта – Опции компиляции** – установить галочку **Использование символов Unicode для идентификаторов**), но настоятельно рекомендуем так не делать – названия переменных на русском обычно выглядят чужеродно и получаются более длинными, чем на английском.

Теперь нужно рассмотреть несколько важных понятий, связанных с переменными:

- типы данных;
- области переменных;
- модификаторы областей.

5.3 Типы данных

Тип данных определяет размер переменной, диапазон ее возможных значений и набор допустимых операций.

5.3.1 Тип BOOL

- логический тип данных;
- допустимые значения: **FALSE** (0) или **TRUE** (1);
- размер – 1 байт (именно байт, а не бит; такова реализация этого типа в CODESYS);
- используется для описания бинарных событий и команд (например, отсутствие/наличие тревоги, отключение/включение насоса и т. д.;
- поддерживает логические операции (AND, OR, XOR, NOT).

```
PROGRAM PLC_PRG
VAR
   xVar1:        BOOL;
   xVar2:        BOOL;
   xVar3:        BOOL;
END_VAR

// Область кода

xVar1 := TRUE;

// xVar3 примет значение TRUE, если переменные xVar1 и xVar2 будут иметь значение TRUE
// если xVar1 или xVar2 будет иметь значение FALSE, то xVar3 тоже будет иметь значение
// FALSE
xVar3  :=  xVar1 AND xVar2;
```

5.3.2 Целочисленные типы данных

- могут быть знаковыми и беззнаковыми;
- размер и диапазон определяется конкретным типом;
- поддерживают арифметические операции и операции сравнения;
- поддерживают побитовый доступ.

Таблица 5.3.1 – Характеристики целочисленных типов данных

Тип	Размер, байт	Диапазон значений
USINT / BYTE	1	0…255
SINT	1	-128…127
UINT / WORD	2	0…65535
INT	2	-32768…32767
UDINT / DWORD	4	$0…2^{32}-1$
DINT	4	$-2^{31}…2^{31}-1$
ULINT / LWORD	8	$0…2^{64}-1$
LINT	8	$-2^{63}…2^{63}-1$

Типы USINT / BYTE и остальные подобные пары в реализации CODESYS являются идентичными. «Двойное» обозначение сохранено для совместимости со стандартом МЭК 61131-3, в котором для типов USINT/UINT/UDINT/ULINT определены только арифметические операции, а для типов BYTE/WORD/DWORD/LWORD – определен только побитовый доступ. В CODESYS арифметические операции и побитовый доступ возможны для любого из этих типов.

При выполнении арифметических операций возможен эффект переполнения – например, если переменная типа USINT имеет значение 255 (это верхний предел ее диапазона) и мы прибавим к ней 1, то переменная получит значение 0 (т. е. ее значение станет равным нижней границе диапазона ее значений).

Деление на 0 приведет к исключению в приложении контроллера (приложение при этом будет остановлено).

```
PROGRAM PLC_PRG
VAR
   iVar:          INT;
   siVar1:        SINT;
   siVar2:        SINT;
   wVar:          WORD;
   dwVar:         DWORD;
END_VAR

// Область кода

iVar   := 123 + 2 * siVar1;
siVar1 := 9;

// siVar2 получит значение -120 из-за эффекта переполнения
siVar2 := 127 + siVar1;

// можно задавать значения в 16-ричной системе счисления (HEX)
wVar := 16#00FF;

// побитовый доступ к целочисленным переменным
// в качестве номера бита нельзя использовать переменную
dwVar.0 := TRUE;
dwVar.2 := TRUE;
```

5.3.3 Типы данных с плавающей точкой

- реализация соответствует стандарту IEEE 754-2008;
- размер, диапазон и точность (число значащих цифр после точки) определяется конкретным типом;
- поддерживают арифметические операции и операции сравнения.

Таблица 5.3.2 – Характеристики типов данных с плавающей точкой

Тип	Размер, байт	Диапазон значений	Точность
REAL	4	± (≈ 10^{-38}…≈10^{38})	7-8 знаков после точки
LREAL	8	± (≈ -10^{-308}…≈10^{308})	15-17 знаков после точки

Число с плавающей точкой всегда представляет собой некое приближение к «действительному» значению. При арифметических операциях с такими числами могут возникнуть погрешности округления (см. пример ниже), поэтому некорректно проверять их на строгое равенство.

```
PROGRAM PLC_PRG
VAR
   rVar1:        REAL;
   rVar2:        REAL;
   rVar3:        REAL;

   iVar1:        INT := 10;
END_VAR

// Область кода

rVar1 := 0.6;

// из-за погрешности округления rVar1 будет равен 0.700000048
rVar1 := rVar1 + 0.1;

// rVar2 будет равен 3.0, потому что iVar1 и 3 – это целые числа
// результат деления целых чисел является целым числом
rVar2 := iVar1 / 3;

// чтобы получить «правильный» результат – один из операндов должен быть типа с
// плавающей точкой. Поэтому нужно сделать так:
rVar3 := iVar1 / 3.0;

// или так (см. информацию про операторы конверсии типов в п. 5.3.6):
rVar3 := TO_REAL(iVar1) / 3;
```

5.3.4 Строковые типы

- поддерживается два типа строк – **STRING** (ASCII-подобная кодировка) и **WSTRING** (кодировка UTF-16). Характеристики типов описаны в таблице ниже;
- значения типа STRING задаются в одинарных кавычках ('test'), значения типа WSTRING – в двойных ("test");
- допустимое число символов строки указывается при ее объявлении, при этом сразу выделяется память под строку максимально допустимой длины;
- ограничение на допустимое число символов отсутствует, но стандартные библиотеки (**Standard** и **Standard64**) рассчитаны на работу со строками длиной не более **255** символов;
- являются нуль-терминированными (как в языке C). Память под NULL выделяется автоматически – пользователю не нужно об этом думать;
- поддерживают управляющие последовательности (переход на новую строку и т. д.). См. примеры ниже;
- поддерживают индексный доступ (как массивы; индексация с нуля);
- для работы со строками («склеивание», определение длины и т. д.) используются функции библиотек **Standard** (для STRING) и **Standard64** (для WSTRING);
- для конвертации кодировок используются функции библиотеки **OwenStringUtils**;
- для отображения в визуализации текста на русском используйте только тип **WSTRING**.

Таблица 5.3.3 – Характеристики строковых типов данных

	STRING	WSTRING
Кодировка	8-битная, основанная на ASCII	UCS-2 (подвид UTF-16)
Пример значения	'test'	"test"
Размер символа, байт	1	2
Размер строковой переменной	Допустимая длина строки + 1	(Допустимая длина строки + 1) · 2
Библиотека функций	Standard	Standard64

```
PROGRAM PLC_PRG
VAR
   // строка с допустимой длиной 20 символов
   // размер переменной 21 байт (20 символов + 1 байт для NULL-терминатора)
   sVar1:        STRING(20) := 'test';
   // по умолчанию для строки выделяется 80 символов
   // так что размер этой переменной - 81 байт
   sVar2:        STRING;

   // размер переменной - 42 байта (2 байт на NULL-терминатор)
   // символы управляющей последовательности экранируются знаком $
   // ниже приведен пример переноса строки после hello
   wsVar1:       WSTRING(20) := "hello$n$rworld ";
   // по умолчанию для строки выделяется 80 символов
   // так что размер этой переменной - 162 байта
   wsVar2:       WSTRING;
END_VAR

// Область кода

sVar2  := 'температура';
wsVar2 := "давление";

// пример индексного доступа к строке
// sVar1 получит значение 'rest', т.к. 16#72 - ASCII-код символа 'r'
sVar1[0] := 16#72;
```

5.3.5 Типы данных даты и времени

- используются для описания меток времени и интервалов времени;
- размер и диапазон зависят от конкретного типа;
- для типов DT/LDT/DATE/LDATE в качестве точки отсчета используется 00:00:00 01.01.1970 (как в Unix time);
- имеют специфический синтаксис для указания значений (см. примеры ниже);
- поддерживают некоторые арифметические операции (см. примеры ниже) и операции сравнения.

Таблица 5.3.3 – Характеристики типов данных даты и времени

Тип	Описание	Дискретность	Размер, байт	Диапазон значений
TIME	Интервал времени	Миллисекунды	4	T#0d0h0m0s0ms…T#49d17h2m47s295ms
LTIME		Наносекунды	8	LTIME#0d0h0m0s0ms0us0ns… LTIME#213503d23h34m33s709ms551us615ns
DT	Дата и время	Секунды	4	DT#1970-1-1-0:0:0…DT#2106-02-07-06:28:15
LDT		Наносекунды	8	LDT#1970-1-1-0:0:0… LDT#2554-7-21-23:59:59.99999999
DATE	Дата	Секунды	4	D#1970-01-01…D#2106-02-07
LDATE		Наносекунды	8	LD#1970-01-01…LD#2554-7-21
TOD	Время суток	Миллисекунды	4	TOD#0:0:0…TOD#23:59:59.999
LTOD		Наносекунды	8	LTOD#0:0:0…LTOD#23:59:59.999999999

```
PROGRAM PLC_PRG
VAR
   dtVar:         DT;
   tVar:          TIME;
   // количество часов в виде целочисленной переменной
   uiHourCount:   UINT := 8;
END_VAR

// Область кода
dtVar := DT#2022-7-22-11:55:11;
// tVar получит значение T#8h
tVar := T#1H * uiHourCount;

// Возможные арифметические операции:
// TIME + TIME = TIME
// TIME – TIME = TIME
// TIME · целочисленный тип = TIME
// TIME / целочисленный тип = TIME

// DT + TIME = DT
// DT – TIME = DT
// DT – DT = TIME

// TOD + TIME = TOD
// TOD – TIME = TOD

// аналогично для LTIME/LDT/LTOD
```

5.3.6 Конверсия типов

В предыдущих пунктах мы рассмотрели большинство из поддержанных в CODESYS элементарных типов данных. Все эти типы могут быть преобразованы друг в друга. Например, если вы планируете отображать в визуализации целочисленное значение, дополнив его единицами измерения (и в зависимости от настроек программы должна быть возможность переключить эти единицы), то сначала потребуется преобразовать это число в строку.

Операторы конверсии выглядят следующим образом: TO_<название типа результата конверсии>

Полный список доступных операторов приведен в этой и соседних с ней статьях онлайн-справки CODESYS.

```
PROGRAM PLC_PRG
VAR
   xVar:          BOOL;
   iVar1:         INT;
   iVar2:         INT;
   uiVar1:        UINT;
   uiVar2:        UINT;
   uiVar3:        UINT;
   usiVar:        USINT;
   rVar:          REAL;
   sVar:          STRING;
   dtVar:         DT;
   udiVar:        UDINT;
END_VAR

// Область кода

xVar := TRUE;
// iVar получит значение 1
iVar1 := TO_INT(xVar);
// sVar получит значение 'TRUE'
sVar := TO_STRING(xVar);

iVar2 := -123;
// uiVar получит значение 65413
// так как тип UINT является беззнаковым, то произойдет вот что:
// 65535 (верхняя граница UINT) + 1 - 123 = 65413
uiVar1 := TO_UINT(iVar2);
// sVar получит значение '-123'
sVar := TO_STRING(iVar2);

uiVar2 := 513;
// верхняя граница USINT - 255
// поэтому usiVar получит значение 1 из-за эффекта переполнения
usiVar := TO_USINT(uiVar2);

rVar := 11.22;
// uiVar получит значение 11
uiVar3 := TO_UINT(rVar);

dtVar := DT#2022-7-22-11:55:11;
// в переменную udiVar будет записано число секунд, прошедших между...
// ...1970-1-1-0:0:0 и 2022-7-22-11:55:11
udiVar := TO_UDINT(dtVar);
```

Обратите внимание, что конверсия STRING в WSTRING и WSTRING в STRING не приведет к конвертации кодировок – поэтому если строка содержала, например, символы кириллицы, то после конверсии к другому строковому типу в ней окажутся «кракозябры». Для конвертации кодировок нужно использовать функции из библиотеки **OwenStringUtils**.

5.3.7 Массивы

В предыдущих пунктах мы рассматривали только элементарные типы данных. Переменная такого типа хранит в себе одно конкретное значение. Но даже в средних по размеру проектах количество сигналов, которые требуется обрабатывать, редко меньше нескольких сотен. Кроме того, часто между этими сигналами можно увидеть взаимосвязи – например, температуры 10 одинаковых узлов системы управления должны обрабатываться одинаковым образом.

Для удобной работы с группами объектов одного типа используются массивы. Рассмотрим типичную задачу – необходимо формировать сигнал тревоги, если температура превышает допустимый предел. Предположим, в нашей системе 100 точек, в которых мы контролируем температуру. Без использования массивов код мог бы выглядеть вот так:

```
PROGRAM PLC_PRG
VAR
   rTemp_1:                       REAL;
   rTemp_2:                       REAL;
   // объявление еще 97 переменных
   rTemp_100:                     REAL;

   rTempHighAlarmLimit_1:         REAL := 11.22;
   rTempHighAlarmLimit_2:         REAL := 22.33;
   // объявление еще 97 переменных
   rTempHighAlarmLimit_100:       REAL := 33.44;

   xAlarm_1:                      BOOL;
   xAlarm_2:                      BOOL;
   // объявление еще 97 переменных
   xAlarm_100:                    BOOL;
END_VAR

// Область кода

xAlarm_1   := rTemp_1 > rTempHighAlarmLimit_1;
xAlarm_2   := rTemp_2 > rTempHighAlarmLimit_2;
// ...
// еще 97 строк, отличающихся только индексами
// ...
xAlarm_100 := rTemp_100 > rTempHighAlarmLimit_100;
```

В нашем коде с одной стороны много строк (по строке на каждую температуру – то есть всего будет 100 строк кода), а с другой – все эти строки отличаются лишь номерами в названиях переменных, в которых очень легко допустить ошибку при копировании и переименовании.

С использованием массивов можно сделать так:

```
PROGRAM PLC_PRG
VAR
   arTemp:                 ARRAY [1..100] OF REAL;
   arTempHighAlarmLimit:  ARRAY [1..100] OF REAL := [11.22, 22.33, 97(32.23), 33.44];
   axAlarm:                ARRAY [1..100] OF BOOL;

   i:                     INT;
END_VAR

// Область кода
//
// FOR - оператор цикла. Он будет рассмотрен далее в п. 5.7.3
// [i] - обращение к i-му элементу массива
// В приведенном ниже коде происходит попарное сравнение (проверка на больше)...
// ...элементов массивов arTemp и arTempHighAlarmLimit
// Результаты сравнения сохраняются в массив axAlarm
//
FOR i := 1 TO 100 DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

Кода стало значительно меньше.

Оператор цикла **FOR** мы подробнее рассмотрим в п. 5.7.3.

При объявлении массива указываются его нижний и верхний пределы, которые определяют число элементов массива. Массивы в стандарте МЭК 61131-3 являются статическими, и число их элементов обязательно должно быть известно на этапе объявления переменной. Объявить динамический массив нельзя.

Часто массивы нумеруют с 0, но это совершенно необязательно – границами массива могут быть любые целые числа (в т. ч. и отрицательные), и единственное очевидное ограничение – верхняя граница массива должна быть больше нижней.

При объявлении массива можно указать начальные значения его элементов – в примере выше это сделано для допустимых пределов температур. Можно инициализировать несколько последовательно расположенных элементов массива одним и тем же значением – в примере выше элемент с индексом 1 получит начальное значение 11.22, элемент с индексом 2 – значение 22.33, а расположенные за ним 97 элементов (то есть элементы с индексами 3…99) – значение 32.23.

Для обращения к элементу массива необходимо указать в квадратных скобках его индекс:

```
// запись значения в 5-й элемент массива
arTemp[5] := 55.44;
```

Индексом может быть как число, так и переменная. В случае индексов-переменных система исполнения не проверяет принадлежность текущего значения переменной диапазону границ массива. То есть такой код будет выполнен без каких-либо предупреждений со стороны CODESYS:

```
i := 1000;
// запись значения в несуществующий элемент массива
arTemp[i] := 55.44;
```

Фактически – мы перезапишем какой-то блок памяти нашего приложения. Если эта память используется системой исполнения, то мы можем получить исключение с кодом **AccessViolation**, являющееся признаком ошибки сегментации памяти. Другой возможный вариант развития событий – мы перезапишем память нашего приложения, и в результате одна или нескольких других переменных получат некорректные значения, что нарушит логику работы программы. Поэтому при доступе к массиву требуется тщательно контролировать используемые значения индексов.

Элементы массива расположены в памяти приложения последовательно и непрерывно.

В примере выше мы рассмотрели одномерный массив. Помимо них существуют еще многомерные массивы, но в реальных проектах они используются не слишком часто, поэтому в рамках «первого старта» мы не будем их рассматривать. По этой же причине мы не будем рассматривать массивы переменной длины (они используются в функциях и функциональных блоках, которые должны уметь обрабатывать массивы различных размеров).

Давайте еще раз вернемся к нашему примеру:

```
PROGRAM PLC_PRG
VAR
   arTemp:                 ARRAY [1..100] OF REAL;
   arTempHighAlarmLimit:  ARRAY  [1..100] OF REAL := [11.22, 22.33, 97(32.23), 33.44];
   axAlarm:                ARRAY [1..100] OF BOOL;

   i:                     INT;
END_VAR

// Область кода
FOR i := 1 TO 100 DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

В примере 4 раза используется одно и то же число, описывающее верхнюю границу массива – **100**. Если нам потребуется изменить число элементов нашего массива – то потребуется не забыть сделать это в 4 разных местах. Вместо этого можно объявить константу (подробнее мы поговорим о них в п. 5.5.1):

```
PROGRAM PLC_PRG
VAR
   arTemp:                 ARRAY [1..c_iMaxTempPointCount] OF REAL;
   arTempHighAlarmLimit:  ARRAY  [1..c_iMaxTempPointCount] OF REAL := [11.22, 22.33,
         97(32.23), 33.44];
   axAlarm:                ARRAY [1..c_iMaxTempPointCount] OF BOOL;

   i:                     INT;
END_VAR
VAR_CONSTANT
   c_iMaxTempPointCount:  INT := 100;
END_VAR

// Область кода
FOR i := 1 TO c_iMaxTempPointCount DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

Теперь в случае изменения числа точек измерения температуры нам будет достаточно отредактировать всего лишь одну константу. Мы могли бы объявить константой и нижнюю границу массива, но обычно такой подход не используется – почти всегда массивы нумеруются с 0 или 1, и эти числа в коде интуитивно понятны (в отличие от «произвольных» чисел типа 100).

Вместо константы мы могли бы использовать специальные операторы для определения границ массива (их второй аргумент означает число измерений массива). Поскольку эти операторы возвращают значение типа **DINT**, но нам потребуется изменить тип переменной счетчика цикла.

```
PROGRAM PLC_PRG
VAR
   // объявление массивов, как в предыдущем примере
   ...
   i:    DINT;
END_VAR

// Область кода
FOR i := LOWER_BOUND(arTemp, 1) TO UPPER_BOUND(arTemp, 1) DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

Последнее, что стоит рассказать о массивах – строки тоже являются массивами и поддерживают индексный доступ. Вы можете работать с переменной типа **STRING** как с массивом типа **ARRAY [...] OF BYTE**, а с переменной типа **WSTRING** – как с массивом типа **ARRAY [...] OF WORD**.

Тезисно подведем итоги изучения массивов:

- массивы удобны для представления групп объектов одинакового типа;
- массивы поддерживают индексный доступ;
- обращение к несуществующему элементу массива не контролируется CODESYS и может привести к ошибкам – поэтому принадлежность значения индекса массива диапазону его границ должна контролироваться разработчиком;
- элементы массива расположены в памяти приложения последовательно и непрерывно;
- при объявлении массивов в качестве верхней границы желательно использовать константу;
- операторы **LOWER_BOUND** и **UPPER_BOUND** позволяют определить значения границ массивов в коде проекта;
- строки, как и массивы, поддерживают индексный доступ.

5.3.8 Пользовательские типы данных

Все предыдущие рассмотренные типы являются «встроенными» – то есть объявление переменных таких типов не требует каких-то предварительных действий. Но в некоторых случаях удобно создавать свои типы, подходящие для конкретных задач. Для этого используются пользовательские типы данных, называемые **DUT** (Data Unit Type).

Для создания такого типа нужно нажать правой кнопкой мыши на узел **Application**, использовать команду **Добавление объекта** и выбрать объект **DUT**. В появившемся окне нужно выбрать требуемый пользовательский тип.

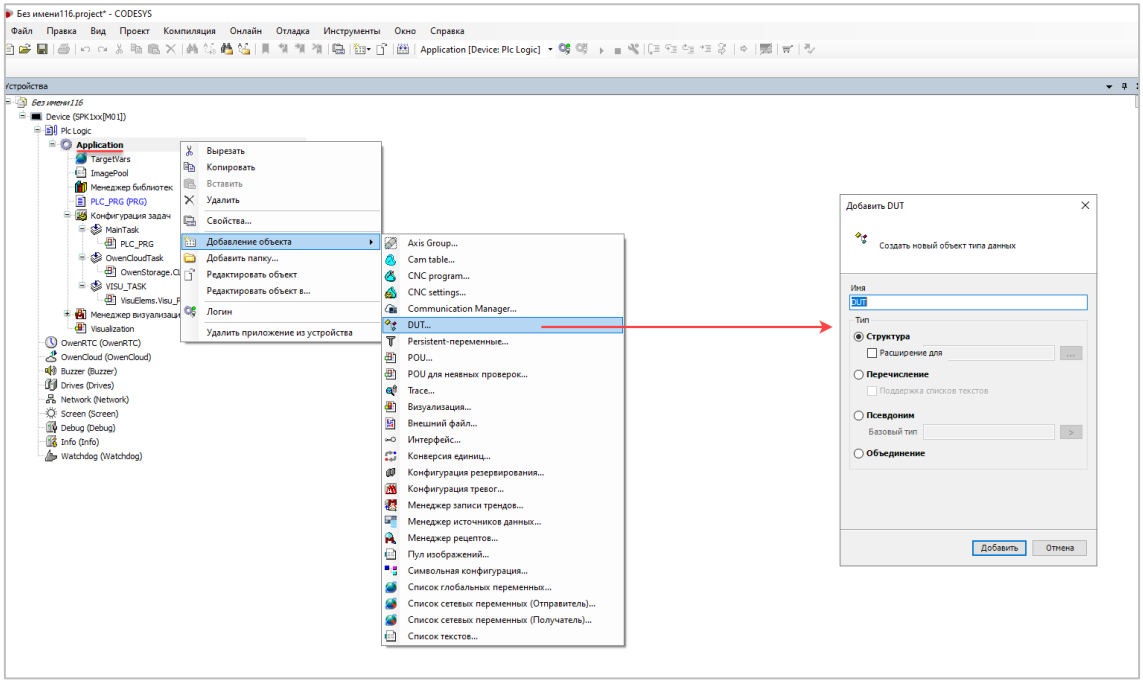


Рисунок 5.3.1 – Создание пользовательского типа

После создания пользовательского типа можно использовать его при объявлении переменных. Всего существует 4 пользовательских типа: структуры, перечисления, псевдонимы и объединения.

5.3.9 Структуры

Структуры позволяют в рамках одного объекта сгруппировать переменные различных типов. Этим они отличаются от массивов, которые используются для группировки переменных одного типа.

Давайте создадим структуру с именем **TEMP_POINT**, описывающую одну точку измерения температуры:

ПРИМЕЧАНИЕ
Указывать в названии постфикс **(STRUCT)**, который виден на скриншоте, не следует – среда добавит его автоматически.

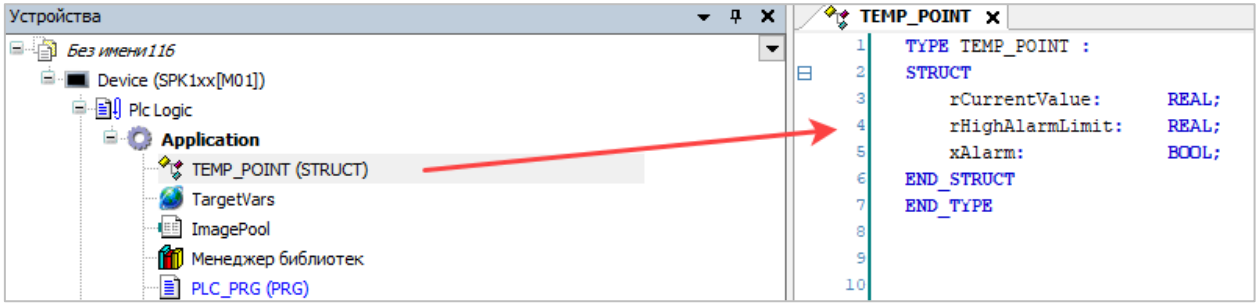


Рисунок 5.3.2 – Создание структуры

```
PROGRAM PLC_PRG
VAR
   stTempPoint: TEMP_POINT := (rHighAlarmLimit := 22.33);
END_VAR

// Область кода
stTempPoint.xAlarm := stTempPoint.rCurrentValue > stTempPoint.rHighAlarmLimit;
```

В примере выше показан синтаксис инициализации структур и обращение к их переменным в коде программы через точку.

Давайте вспомним наш пример из пункта про массивы:

```
PROGRAM PLC_PRG
VAR
   arTemp:                 ARRAY [1..c_iMaxTempPointCount] OF REAL;
   arTempHighAlarmLimit:   ARRAY [1..c_iMaxTempPointCount] OF REAL := [11.22, 22.33,
         97(32.23), 33.44];
   axAlarm:                ARRAY [1..c_iMaxTempPointCount] OF BOOL;

   i:                      INT;
END_VAR
VAR CONSTANT
   c_iMaxTempPointCount:  INT := 100;
END_VAR

// Область кода
FOR i := 1 TO c_iMaxTempPointCount DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

Перепишем его, используя структуру:

```
PROGRAM PLC_PRG
VAR
   astTempPoint:          ARRAY [1..c_iMaxTempPointCount] OF TEMP_POINT;

   i:                     INT;
END_VAR
VAR CONSTANT
   c_iMaxTempPointCount:  INT := 100;
END_VAR

// Область кода
FOR i := 1 TO c_iMaxTempPointCount DO
   astTempPoint[i].xAlarm := astTempPoint[i].rCurrentValue >
         astTempPoint[i].rHighAlarmLimit;
END_FOR
```

Код стал компактнее – вместо трех массивов мы обошлись одним массивом структур.

В последнем примере мы не стали указывать начальные значения допустимых границ – для массива структур синтаксис инициализации довольно сложен (если вам это интересно – см. соответствующую статью в онлайн-справке CODESYS). Вместо этого в подобных случаях присвоение начальных значений часто производят прямо в коде программы:

```
PROGRAM PLC_PRG
VAR
   astTempPoint:          ARRAY [1..c_iMaxTempPointCount] OF TEMP_POINT;

   xInit:                 BOOL;
END_VAR
VAR CONSTANT
   c_iMaxTempPointCount:  INT := 100;
END_VAR

// Область кода

IF NOT(xInit) THEN
   astTempPoint[1].rHighAlarmLimit := 11.22;
   astTempPoint[2].rHighAlarmLimit := 22.33;
   // ...
   astTempPoint[100].rHighAlarmLimit := 33.44;

   xInit := TRUE;
END_IF
```

Приведенный код однократно выполняется при старте приложения: в момент его запуска переменная **xInit** имеет значение **FALSE**, за счет чего выполняется условие оператора **IF** (подробнее об этом операторе мы поговорим в п. 5.7.1). Внутри оператора **IF** происходит инициализация нужных элементов массива структур, после чего переменной **xInit** присваивается значение **TRUE** – и в результате этот блок кода больше не будет выполняться.

Элементы структур располагаются в памяти контроллера последовательно, но не обязательно непрерывно. Из-за этого размер структуры может не совпадать с суммой размеров ее элементов. Вернемся к рис. 5.3.2 – из пунктов 5.3.1 и 5.3.3 мы знаем, что переменная типа **REAL** занимает 4 байта памяти, а переменная типа **BOOL** – 1 байт. Таким образом, суммарный размер элементов этой структуры составляет 9 байт, но реальный размер данной структуры – 12 байт. Это связано с выравниванием памяти – компилятор оптимизирует размещение переменных в памяти, оставляя между ними «разрывы», для ускорения доступа к ним со стороны процессора.

В некоторых случаях требуется, чтобы элементы структуры размещались в памяти без разрывов, «вплотную» друг к другу. В CODESYS для этого используется специальный атрибут **pack_mode**. Более подробная информация об этом атрибуте и других особенностях структур приведена в п. 1.1 данной статьи.

5.3.10 Объединения

Объединения похожи на структуры с одним существенным отличием – все их элементы размещаются в одной и той же области памяти. То есть каждый элемент объединения представляет собой различную интерпретацию одних и тех же данных. Проще пояснить это на примере: предположим, мы считываем с какого-то датчика значение типа **REAL**, в старшем байте которого кодируется код ошибки (ошибкой, например, может быть обрыв датчика, ошибка АЦП и т. д.). Чтобы определить наличие ошибки – нам надо сравнить значение старшего байта значения с заранее известными кодами ошибок. Для этого мы можем создать объединение:

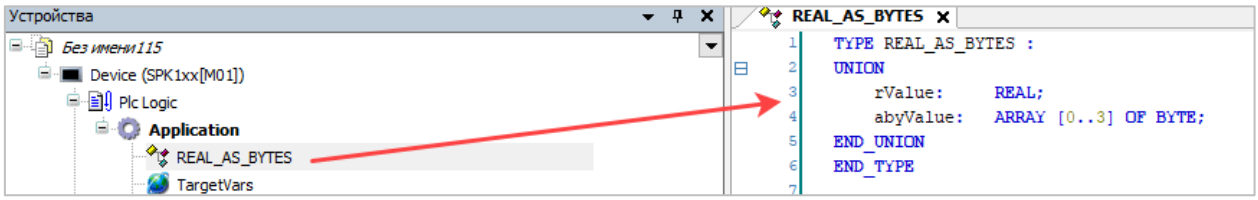


Рисунок 5.3.3 – Создание объединения

Объявим в программе переменную этого типа и напишем вот такой код:

```
PROGRAM PLC_PRG
VAR
   uTemp:          REAL_AS_BYTES;

   xSensorFailed: BOOL;
END_VAR

// Область кода

// для отладки запишем в REAL-переменную структуры какое-то значение
uTemp.rValue := 11.22;

// проверяем старший байт значения (240 - код ошибки «обрыв датчика»)
// в нашем случае ошибки нет, так что xSensorFailed получит значение FALSE
xSensorFailed := (uTemp.abyValue[3] = 240);
```

Как и в случае структур – доступ к элементам объединения осуществляется через точку. Поскольку элементы объединения размещаются в одной и той же области памяти, то проверяя старший байт массива **abyValue** – мы проверяем старший байт переменной **rValue** типа **REAL**.

Более подробная информация об объединениях приведена п. 1.3 данной статьи.

5.3.11 Перечисления

Перечисления позволяют задать для набора конкретных целочисленных значений символьные имена. Это повышает читабельность кода программы. Вернемся к примеру из предыдущего пункта:

```
PROGRAM PLC_PRG
VAR
   uTemp:          REAL_AS_BYTES;

   xSensorFailed: BOOL;
END_VAR

// Область кода

// для отладки запишем в REAL-переменную структуры какое-то значение
uTemp.rValue := 11.22;

// проверяем старший байт значения (240 – код ошибки «обрыв датчика»)
xSensorFailed := (uTemp.abyValue[3] = 240);
```

Здесь в коде используется число **240**, которое означает конкретный код ошибки датчика. Смысл числа поясняется в комментарии, который размещен строкой выше. Но в реальном проекте работа с ошибками датчиков может происходить в нескольких фрагментах программы, и в каждом из них нам придется дублировать этот комментарий. Если выяснится, что в новой прошивке датчика код ошибки изменился – то придется менять его во всех фрагментах кода, где мы с ним работали, а также отредактировать все комментарии. Вполне вероятно, что где-то мы забудем внести исправление, и программа будет работать некорректно. Кроме того, комментарии могут быть потеряны при переносе кода в другой проект и т. п.

Поэтому нагляднее использовать перечисления. Создадим перечисление с кодами ошибок датчика:

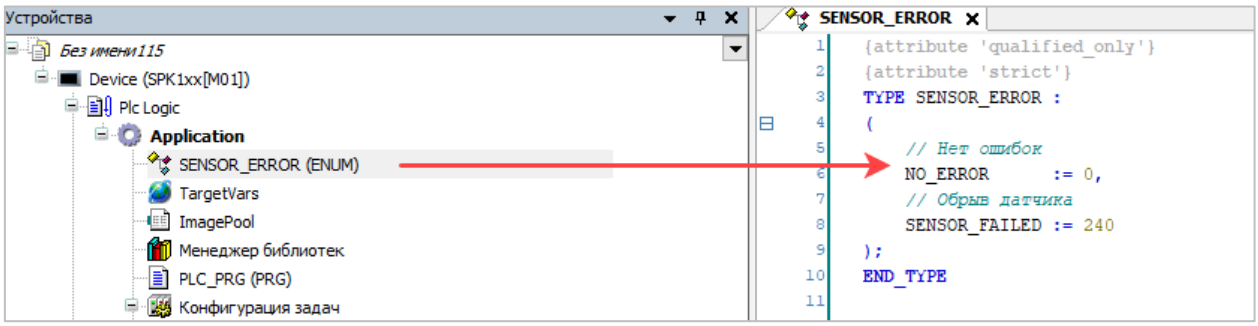


Рисунок 5.3.4 – Создание перечисления

В нашем случае мы объявили в перечислении код нормального состояния датчика и код одной ошибки; вы можете добавить в перечисление столько элементов, сколько вам потребуется.

Теперь можно изменить код программы следующим образом:

```
PROGRAM PLC_PRG
VAR
   uTemp:          REAL_AS_BYTES;

   xSensorFailed: BOOL;
END_VAR

// Область кода

// для отладки запишем в REAL-переменную структуры какое-то значение
uTemp.rValue := 11.22;

// проверяем старший байт значения
xSensorFailed := (uTemp.abyValue[3] = SENSOR_ERROR.SENSOR_FAILED);
```

Мы избавились от не очень понятной стороннему читателю цифры **240**, заменив ее гораздо более понятным символьным именем.

Обычно вместе с созданием перечисления производится объявление переменной соответствующего типа. Это удобно в тех случаях, когда переменная должна описывать код ошибки, режим работы оборудования, выполняемую оборудованием технологическую операцию и т. д. Рассмотрим и такой пример:

```
PROGRAM PLC_PRG
VAR
   rValue:                 REAL;
   eSensorError:           SENSOR_ERROR;
END_VAR
VAR CONSTANT
   rValueLowLimit:         REAL := 4.0;
   rValueHighLimit:        REAL := 20.0;
END_VAR

// Область кода
IF rValue < rValueLowLimit OR rValue > rValueHighLimit THEN
   eSensorError := SENSOR_ERROR.SENSOR_FAILED;
END_IF
```

По умолчанию перечисление занимает 2 байта памяти. В случае необходимости (например, коды ваших ошибок превышают значение 65535 и, соответственно, не могут быть представлены 2-х байтовой целочисленной переменной) – вы можете указать тип перечисления при его создании.

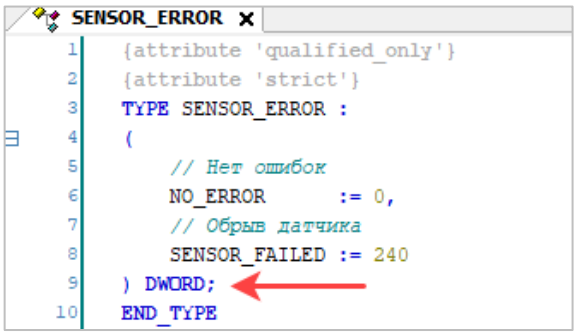


Рисунок 5.3.5 – Указание типа перечисления

Более подробная информация о перечислениях приведена п. 1.2 данной статьи.

5.3.12 Псевдонимы

Псевдонимы позволяют задать для одного из существующих типов данных дополнительное имя. Давайте рассмотрим на конкретном примере, в каких случаях такая возможность может оказаться полезной. Предположим, в нашей программе мы активно работаем со строками. На этапе проектирования было определено, что длина строк не может превышать 100 символов. Соответственно, в нашей программе объявлено множество строковых переменных с указанием их допустимой длины:

```
PROGRAM PLC_PRG
VAR
   sUserId:      STRING(100);
   sVendorId:    STRING(100);
   sCategoryId:  STRING(100);
   sProductId:   STRING(100);
   // и еще множество переменных типа STRING(100)
END_VAR
```

Предположим, в процессе разработки требования изменились, и теперь необходимо, чтобы строки могли включать в себя до 120 символов. Придется редактировать объявление каждой строковой переменной, изменяя ее допустимую длину. В качестве альтернативы можно было создать псевдоним:

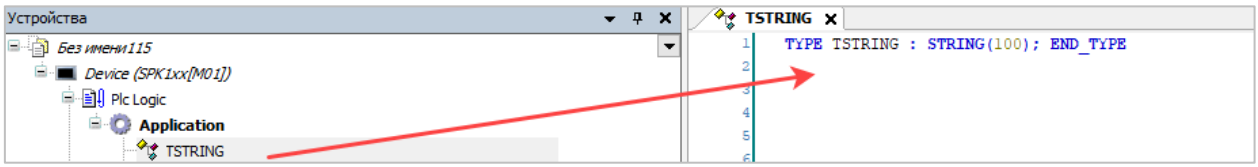


Рисунок 5.3.6 – Создание псевдонима

Теперь мы можем использовать этот псевдоним при объявлении наших строковых переменных:

```
PROGRAM PLC_PRG
VAR
   sUserId:      TSTRING;
   sVendorId:    TSTRING;
   sCategoryId:  TSTRING;
   sProductId:   TSTRING;
END_VAR
```

Изменения требований больше не страшны, ведь нам будет достаточно внести правку только в одном месте проекта – в псевдониме.

Другим вариантом являлось бы использование в качестве длины строки константы.

5.3.13 Специфические типы данных

В прошлых пунктах мы не рассмотрели несколько специфических типов данных, которые иногда используются разработчиками библиотек, но крайне редко – разработчиками приложений. Мы не будем подробно рассматривать их в рамках «первого старта» – приведем лишь краткий обзор:

- **BIT** – аналог типа **BOOL**, который занимает 1 бит (а не 1 байт, как **BOOL**) в памяти контроллера. Переменные этого типа могут быть объявлены только в структурах, объединениях и функциональных блоках. См. подробнее в п. 1.1.3 данной статьи;
- POINTER (указатель) – специфический тип данных, используемый для работы с адресами переменных. Манипуляции с указателями позволяют создавать емкий и эффективный код (например, для копирования больших объемов данных), но в случае ошибок программирования возможна некорректная работа контроллера (вплоть до остановки приложения). Поэтому начинающим разработчикам лучше отказаться от использования указателей;
- REFERENCE (ссылка) – более безопасный, но менее функциональный вариант указателя. Обычно используется совместно со средствами объектно-ориентированного программирования;
- __UXINT, __XINT, __XWORD – платформо-зависимые типы данных, для 32-битных приложений интерпретируемые как **UDINT/DINT/DWORD**, а для 64-битных – как **ULINT/LINT/LWORD**. Используются разработчиками для создания аппаратно-независимых библиотек;
- ANY – специфический тип данных, который может использоваться только при объявлении входных переменных (**VAR_INPUT**). На вход типа ANY можно передать переменную любого типа. Это позволяет создавать универсальные функции и функциональные блоки, которые могут обрабатывать различные типы данных;
- __VECTOR – специфический тип массивов, который может обрабатываться некоторыми процессорами более быстро, чем обычные массивы, с помощью специальных инструкций.

5.4 Области переменных

5.4.1 Виды областей переменных

В прошлых пунктах мы объявляли переменные в программе **PLC_PRG** между ключевыми словами **VAR** и **END_VAR**.

```
PROGRAM PLC_PRG
VAR
   iVar:          INT;
   siVar1:        SINT;
   siVar2:        SINT;
   wVar:          WORD;
   dwVar:         DWORD;
END_VAR
```

Эти ключевые слова определяют область объявления переменных. Область накладывает ограничения на доступ к переменным со стороны этого (в котором объявляется переменная) или других программных объектов. Каждая область начинается с уникального ключевого слова, а заканчивается ключевым словом **END_VAR**. В следующих пунктах описаны основные области переменных. В данном пункте не приводится конкретных примеров – мы рассмотрим их в процессе изучения функций и функциональных блоков.

5.4.2 Локальные переменные (VAR)

Локальные (внутренние) переменные доступны для чтения и записи только внутри того программного объекта, в котором они объявлены. Технически они доступны для чтения со стороны других программных объектов – но не рекомендуется использовать этот метод, потому что это нарушает логику передачи данных внутри приложения. Если требуется, чтобы переменные были доступны для чтения – то сделайте их выходными (VAR_OUTPUT).

5.4.3 Входные переменные (VAR_INPUT)

Входные переменные доступны только для чтения внутри того программного объекта, в котором они объявлены, и доступны для чтения и записи со стороны других программных объектов, вызывающих этот объект. Входные переменные позволяют получить программному объекту данные от других программных объектов.

5.4.4 Выходные переменные (VAR_OUTPUT)

Выходные переменные доступны для чтения и записи внутри того программного объекта, в котором они объявлены, и доступны для чтения со стороны других программных объектов, вызывающих этот объект. Выходные переменные позволяют программному объекту передать данные другим программным объектам.

5.4.5 Переменные области «вход-выход» (VAR_IN_OUT)

Переменные области «вход-выход» доступны для чтения и записи как внутри того программного объекта, в котором они объявлены, так и со стороны других программных объектов, вызывающих этот объект. В большинстве случаев использование этой области не приносит существенной пользы, но затрудняет понимание логики передачи данных внутри приложения. Поэтому мы рекомендуем не использовать эту область в программных объектах (но, например, допустимо использовать ее в объектах визуализации – мы рассмотрим пример такого подхода в п. 6.6.1).

5.4.6 Глобальные переменные (VAR_GLOBAL)

Глобальные переменные доступны для чтения и записи внутри любого программного объекта приложения. В отличие от других областей – глобальные переменные объявляются в отдельном объекте дерева проекта:

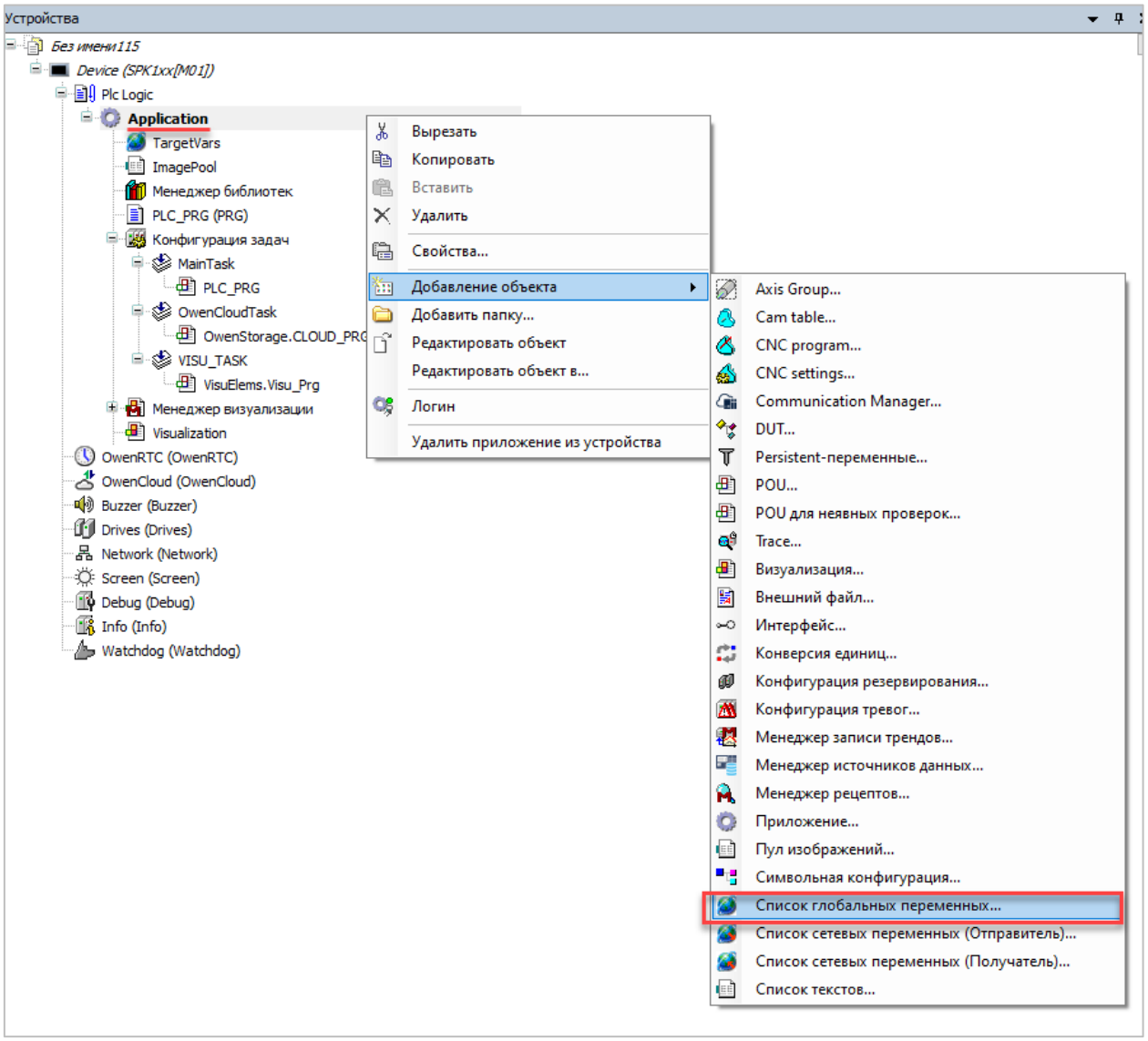


Рисунок 5.4.1 – Создание списка глобальных переменных

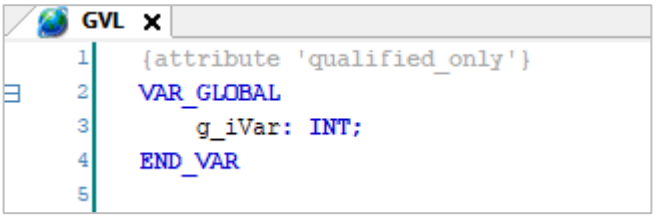


Рисунок 5.4.2 – Объявление переменной в списке глобальных переменных

Для обращения к глобальной переменной необходимо указать имя ее списка:

```
PROGRAM PLC_PRG
VAR
END_VAR

// Область кода
GVL.g_iVar := 11;
```

Глобальные переменные следует использовать осознанно. В состав шаблона проекта входит список глобальных переменных **TargetVars**, переменные которого привязаны к каналам системных узлов таргет-файла (**OwenRTC**, **Buzzer** и т. д.). Переменные этого списка можно воспринимать как системные переменные. Например, значение системного времени и правда может потребоваться в разных программах – так что сделать его глобальным вполне логично.

Но необдуманное использование глобальных переменных (особенно в случае записи одной переменной из нескольких программных объектов) затрудняет понимание логики передачи данных внутри приложения. Поэтому используйте эту область только в тех случаях, когда без нее не получается обойтись.

5.4.7 Остальные области

В CODESYS доступно еще несколько областей переменных – например, **VAR_STAT**, **VAR_EXTERNAL** и **VAR_CONFIG**. Часть из них присутствует только ради сохранения совместимости со стандартом МЭК 61131-3; некоторые области используются только в специфических конкретных случаях, которые нет смысла рассматривать в процессе «первого старта».

В большинстве проектов достаточно трех областей: локальных, входных и выходных переменных.

5.5 Модификаторы областей переменных

Модификаторы областей переменных используются для придания областям определенных свойств. В CODESYS есть два вида модификаторов: для создания констант и энергонезависимых переменных.

5.5.1 Константы (CONSTANT)

Некоторые значения в приложении всегда должны оставаться неизменными. Например, если в системе управления используется 8 насосов, то маловероятно, что в процессе нормальной работы системы это число изменится. Конечно, насосы периодически могут включаться и отключаться – но их общее число останется неизменным. Если же в процессе модернизации системы добавятся еще несколько насосов – то проект для ПЛК, скорее всего, всё равно придется дорабатывать.

Вместо использования в коде конкретных чисел лучше объявить константы. Во-первых, это упростит поддержку кода – ведь если «магическое число» изменится, то придется искать в программе все фрагменты, где оно использовалось, и редактировать их. В случае константы – достаточно будет изменить строку объявления константы. Кроме того, константы защищены от записи – даже в случае ошибки программист не сможет изменить их из кода программы.

Для объявления констант необходимо создать область переменных с добавленным модификатором CONSTANT. В пункте 5.3.7, рассматривая массивы, мы объявили в программе область локальных констант:

```
PROGRAM PLC_PRG
VAR
   arTemp:                 ARRAY [1..c_iMaxTempPointCount] OF REAL;
   arTempHighAlarmLimit:  ARRAY [1..c_iMaxTempPointCount] OF REAL := [11.22, 22.33,
         97(32.23), 33.44];
   axAlarm:                ARRAY [1..c_iMaxTempPointCount] OF BOOL;

   i:                      INT;
END_VAR
VAR_CONSTANT
   c_iMaxTempPointCount:   INT := 100;
END_VAR

// Область кода
FOR i := 1 TO c_iMaxTempPointCount DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

В реальных проектах обычно используются либо локальные, либо глобальные константы.

В языке CFC могут использоваться входные константы (**VAR_INPUT CONSTANT**). С точки зрения обработки в языке CFC они ничем не отличаются от обычных констант, но в редакторе программирования их значения можно изменить с помощью специального меню:

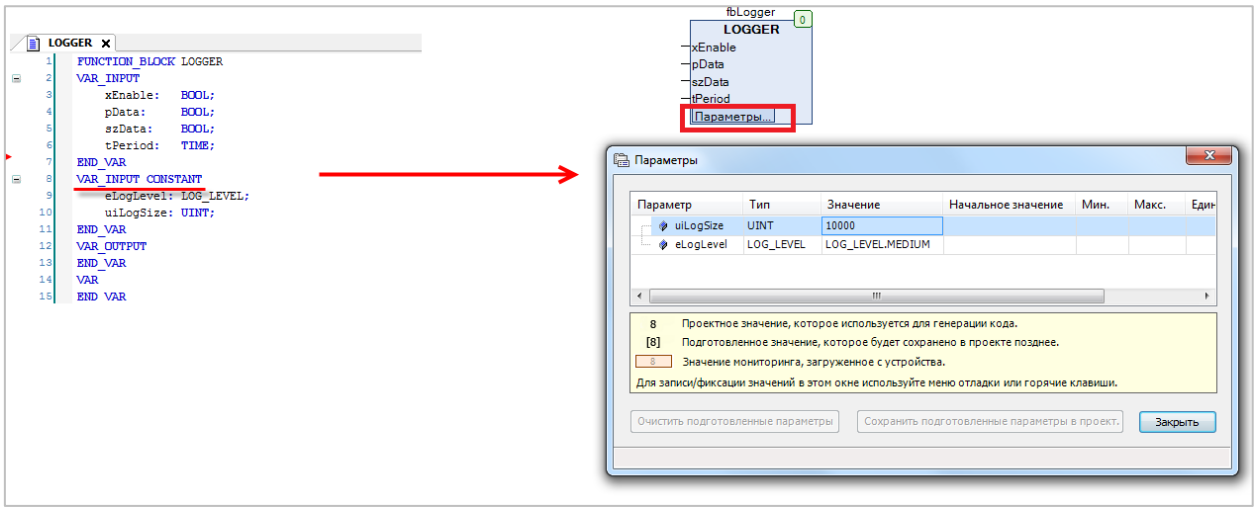


Рисунок 5.5.1 – Работа со входными константами в редакторе языка CFC

В других языках программирования **VAR_INPUT CONSTANT** обрабатываются как обычные входные переменные.

Кроме того, модификатор **CONSTANT** можно добавить к области **VAR_IN_OUT**. Как мы указывали ранее – эта область не рекомендуется к использованию, поэтому мы не будем останавливаться на этом вопросе. При желании вы можете ознакомиться с соответствующей статьей из онлайн-справки CODESYS.

5.5.2 Энергонезависимые переменные (RETAIN и PERSISTENT)

Переменные всех областей, которые мы рассмотрели в п. 5.4, после перезагрузки контроллера принимают значения, указанные при их объявлении (если такие значения не указаны, то используются значения по умолчанию – например, для числовых типов это 0, для строк – пустая строка и т. д.).

В некоторых ситуациях это неприемлемо – например, уставки технологического процесса могут меняться оператором в процессе работы приложения и при этом должны сохраняться после перезагрузки контроллера. В этом случае требуется объявить такие переменные в области энергонезависимых переменных.

Существует два модификатора энергонезависимости – **RETAIN** и **PERSISTENT**. Их отличия заключаются в поведении при загрузке в контроллер нового проекта:

- RETAIN-переменные при загрузке нового проекта переинициализируются;
- PERSISTENT-переменные при загрузке нового проекта сохраняют свои значения – но только те из них, чьи имена и тип не изменились по сравнению с предыдущей версией проекта.

PERSISTENT-переменные удобно использовать в тех случаях, когда заранее известно, что проект будет дорабатываться – например, он длительное время будет находиться в опытной эксплуатации, за время которой обслуживающий персонал будет формулировать пожелания и замечания. В течение этого времени операторы могут задавать уставки, параметры рецептов и т. д. Если число таких параметров превышает 10 (а в реальных проектах оно может составлять несколько сотен и больше) – то вряд ли они будут рады перспективе вводить эти значения заново после каждой загрузки нового проекта. В случае использования PERSISTENT-переменных такой необходимости не возникнет (если только не потребуется изменить названия и типы PERSISTENT-переменных).

Во всех остальных случаях можно использовать RETAIN-переменные.

Объявление RETAIN- и PERSISTENT-переменных имеет существенные отличия.

Для объявления RETAIN-переменных необходимо создать область переменных с добавленным модификатором RETAIN. В реальных проектах используются либо локальные RETAIN-переменные программ, либо глобальные RETAIN-переменные. Объявление RETAIN-переменных в функциях не имеет смысла, а в функциональных блоках – нежелательно из-за чрезмерного расходования памяти (подробнее мы расскажем об этом в п. 5.8.1 и 5.8.2).

Пример объявления RETAIN-переменных:

```
PROGRAM PLC_PRG
VAR
   iNonRetainVar:         INT := 10;
END_VAR
VAR RETAIN
   iRetainVar:            INT := 100;
END_VAR
```

После загрузки приложения переменная **iRetainVar** примет значение **100**. Далее в процессе работы контроллера это значение может изменяться – например, оператор в визуализации задаст переменной значение 200. После перезагрузки контроллера значение RETAIN-переменной сохранится и будет равно 200.

PERSISTENT-переменные объявляются в специальном одноименном объекте, который необходимо добавить в дерево проекта:

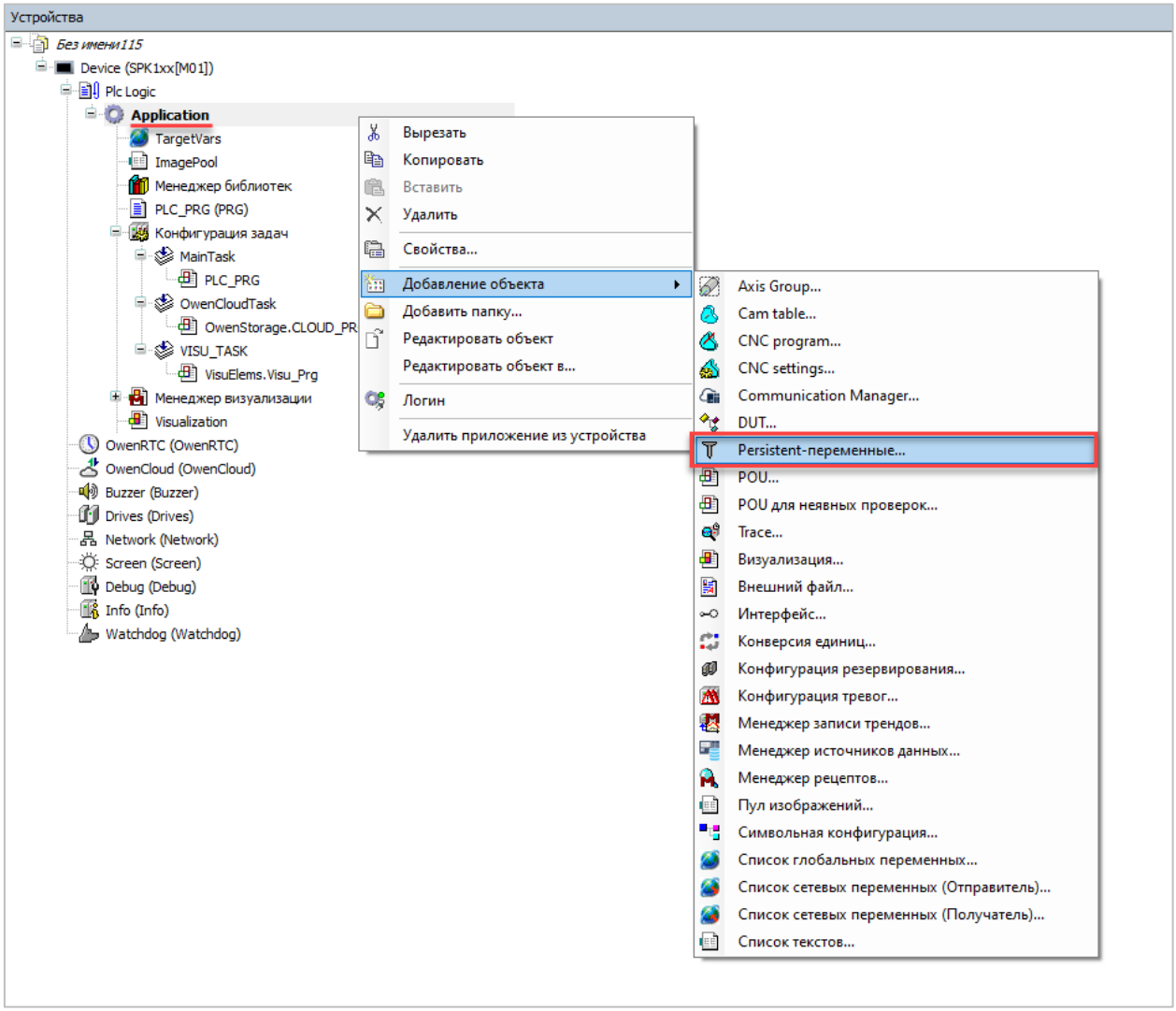


Рисунок 5.5.2 – Создание списка PERSISTENT-переменных

Переменные, объявленные в этом списке, будут глобальными PERSISTENT-переменными. Как и в случае обычных глобальных переменных – для доступа к ним в программе потребуется указывать имя списка.

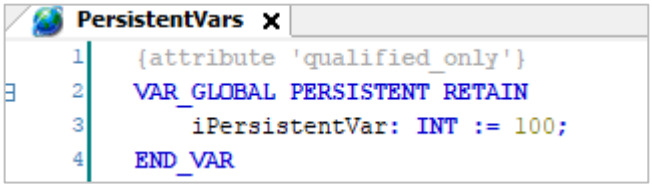


Рисунок 5.5.3 – Объявление PERSISTENT-переменной

```
// где-то в коде программы:
// PersistentVars - имя списка PERSISTENT-переменных
PersistentVars.iPersistentVar := 200;
```

Физически RETAIN- и PERSISTENT-переменные размещаются в одной области памяти. Для их сохранения используется специальная микросхема типа MRAM. Сохранение энергонезависимых переменных выполняется по изменению любой из переменных, но не чаще раза в секунду – при этом вся область MRAM перезаписывается целиком. Число циклов перезаписи MRAM практически бесконечно. MRAM включает в себя две дублирующие друг друга области памяти. При старте контроллер проверяет контрольные суммы этих областей – если для одной области рассчитанная контрольная сумма не соответствует сохраненной (такое может быть, например, при пропадании питания в момент сохранения), то значения переменных восстанавливаются из другой области.

Для выпускаемых в данный момент контроллеров ОВЕН объем энергонезависимой памяти составляет 64 Кб.

В п. 4.2.6 мы рассказывали о разных вариантах команды **Сброс**. Теперь можно пояснить, как разные варианты сброса влияют на значения энергонезависимых переменных:

Таблица 5.5.1 – Поведение переменных при различных типах сброса

	Энерго**зависимые** переменные (локальные и т. д.)	RETAIN	PERSISTENT
Сброс	переинициализация	сохранение	сохранение
Сброс холодный	переинициализация	переинициализация	сохранение
Загрузка приложения	переинициализация	переинициализация	сохранение*

* – только те переменные, чьи имена и тип не изменились по сравнению с предыдущей версией проекта

Таким образом, применительно к командам сброса и загрузки проекта – RETAIN-переменные сохраняют свои значения только в случае команды **Сброс**. PERSISTENT-переменные же сохраняют свои значения и при **Сбросе холодном**, а также при загрузке нового приложения.

5.6 Операторы

В прошлых пунктах мы разобрались с переменными – рассмотрели их типы данных, области объявления и модификаторы. Программа контроллера представляет собой набор действий, выполняемых с переменными. Такие действия называются **операторами**, а переменные и значения, над которыми эти действия производятся – **операндами** (также их называют аргументами). Последовательность связанных между собой операторов называется **выражением**. Поясним это на примере:

```
xAlarm := xSensorFailed AND xAlarmDelayTimeout;
```

AND и «**:=**» – это операторы. Операндами оператора **AND** являются переменные **xSensorFailed** и **xAlarmDelayTimeout**. Операндами оператора «**:=**» являются переменная **xAlarm** и результат выполнения оператора **AND**. Вся строка кода в целом является выражением.

В следующих пунктах мы рассмотрим базовые операторы CODESYS (большая их часть определена в стандарте МЭК 61131-3).

5.6.1 Арифметические операторы

Арифметические операторы используются для выполнения базовых арифметических операций – сложения, вычитания, умножения и деления. Эти операторы применимы к переменным и значениям целочисленного типа, типа с плавающей точкой, а также, с некоторыми ограничениями (описанными по следующей ссылке), к переменным типа даты и времени. Тип возвращаемого значения определяется типами операндов.

```
PROGRAM PLC_PRG
VAR
   iVar1:        INT := 2;
   iVar2:        INT := 3;
   iVar3:        INT;
   iVar4:        INT;
   iVar5:        INT;
   rVar:         REAL;
END_VAR

// Область кода
iVar1 := iVar2 + 5;
iVar3 := 23 - iVar2;
iVar4 := iVar3 * 7;
rVar  := TO_REAL(iVar4) / 3;

// В выражении можно использовать любое сочетание операторов
iVar4 := iVar1 + iVar2 + 3 - iVar3 * 2;
// Получение остатка от деления
iVar5 := iVar2 MOD iVar1;
```

Обратите внимание на пример деления – один из операндов приводится к типу **REAL**, чтобы в результате деления получить значение с плавающей точкой. Без конверсии типов результат деления был бы целым числом (так как происходило бы деление двух целых чисел).

Оператор **MOD** позволяет получить остаток от деления для двух целых чисел.

5.6.2 Логические операторы

Логические операторы используются для выполнения логических операций (И, ИЛИ, НЕ) с переменными и значениями типа BOOL, битами целочисленных переменных и самими целыми числами (т. е. эти операторы являются побитовыми). Тип возвращаемого значения определяется типом операнда.

```
PROGRAM PLC_PRG
VAR
   xVar1:                                                    BOOL;
   xVar2:                                                    BOOL;
   xVar3:                                                    BOOL;
   xResult1, xResult2, xResult3, xResult4, xResult5:         BOOL;
   byBitMask:                                                BYTE;
   byVar1:                                                   BYTE := 1;
   byVar2:                                                   BYTE := 2;
   byResult1, byResult2:                                     BYTE;
END_VAR

// Область кода

// Логическое И
// xResult1 получит значение TRUE только в том случае, если каждый из операндов будет
// иметь значение TRUE. Если хотя бы один из операндов будет иметь значение FALSE – то
// xResult1 получит значение FALSE

xResult1 := xVar1 AND xVar2 AND byBitMask.0;

// Логическое ИЛИ
// xResult2 получит значение TRUE в том случае, если хотя бы один из операндов будет
// иметь значение TRUE. Если все операнды будут иметь значение FALSE – то
// xResult2 получит значение FALSE

xResult2 := xVar1 OR xVar2 OR byBitMask.0;


// Исключающее ИЛИ
// xResult3 получит значение TRUE в том случае, если только один из операндов будет
// иметь значение TRUE. Во всех остальных случаях xResult3 получит значение FALSE

xResult3 := xVar1 XOR xVar2;

// Пример с несколькими XOR в одном выражении.
// Предположим, что все три переменные имеют значение TRUE.
// Тогда выражение принимает вид TRUE XOR TRUE XOR TRUE
// Результат TRUE XOR TRUE – это FALSE
// Подставляем результат в выражение выше, получаем FALSE XOR TRUE
// Результат такого выражения – TRUE (потому что только один из операндов равен TRUE)
xResult4 := xVar1 XOR xVar2 XOR xVar3;


// Оператор логического отрицания
// Если xVar1 равен TRUE, то xResult5 получит значение FALSE
// Если xVar1 равен FALSE, то xResult5 получит значение TRUE
xResult5 := NOT(xVar1);

// при подстановке значений получим 1 OR 2
// byResult1 получит значение 3, потому что логические операторы применяются побитово:
// 2#00000001 OR 2#00000010 -> 2#00000011 (3 в десятичной системе счисления)
byResult1 := byVar1 OR byVar2;

// byResult2 получит значение 254, так как будут инвертированы значений всех бит
// byVar1 равен 1 (2#00000001). После инверсии получится значение 2#11111110
// (254 в десятичной системе)
byResult2 := NOT(byVar1);
```

Операторы **AND** и **OR** всегда производят проверку обоих своих операндов. То есть даже если первый операнд оператора **AND** имеет значение **FALSE** и результат выполнения оператора будет равен **FALSE** независимо от значения второго операнда – то всё равно будет произведена проверка этого значения.

В CODESYS V3.5 в языке ST реализованы специальные операторы **AND_THEN** и **OR_ELSE**, которые выполняют «сокращенную» оценку своих операндов – то есть второй операнд проверяется только в том случае, если это имеет смысл.

5.6.3 Операторы сравнения

Операторы сравнения используются для сравнений переменных и значений целочисленного типа, типа с плавающей точкой и типов даты и времени. Тип возвращаемого значения – BOOL.

```
PROGRAM PLC_PRG
VAR
   iVar1:    INT   := 11;
   iVar2:    INT   := 22;
   rVar:     REAL  := 33.44;
   tVar:     TIME  := T#10m20s;
   xResult1: BOOL;
   xResult2: BOOL;
   xResult3: BOOL;
   xResult4: BOOL;
   xResult5: BOOL;
   xResult6: BOOL;
   xResult7: BOOL;
END_VAR

// Область кода

// Проверка на то, что первый операнд больше второго
xResult1 := iVar2 > iVar1;

// Проверка на то, что первый операнд меньше второго
xResult2 := iVar2 < iVar1;

// Проверка на то, что первый операнд больше или равен второму
xResult3 := rVar >= 0.5;

// Проверка на то, что первый операнд меньше или равен второму
xResult4 := tVar <= T#15m;

// Проверка на принадлежность значения диапазону [10, 20]
xResult5 := iVar1 >= 10 AND iVar1 <= 20;

// Проверка на равенство
xResult6 := iVar1 = 11;

// Проверка на неравенство
xResult7 := iVar1 <> 11;
```

5.6.4 Операторы битового сдвига

Первым аргументом операторов битового сдвига является переменная или значение целочисленного типа, вторым – число бит, на которое выполняется сдвиг. Тип возвращаемого значения определяется типом операнда.

```
PROGRAM PLC_PRG
VAR
   bySource:     BYTE := 2#00110001;
   byResult1:    BYTE;
   byResult2:    BYTE;
   byResult3:    BYTE;
   byResult4:    BYTE;
END_VAR

// Область кода

// Сдвиг влево на 3 бита с дополнением нулями
// byResult1 примет значение 2#10001000
byResult1 := SHL(bySource, 3);


// Сдвиг вправо на 3 бита с дополнением нулями или - для отрицательных чисел –
// единицами
// byResult2 примет значение 2#00000110;
byResult2 := SHR(bySource, 3);

// Циклический сдвиг влево на 3 бита
// byResult3 примет значение 2#10001001
byResult3 := ROL(bySource, 3);


// Циклический сдвиг вправо на 3 бита
// byResult4 примет значение 2#00100110;
byResult4 := ROR(bySource, 3);
```

5.6.5 Математические операторы

Аргументами математических операторов являются переменные или значения целочисленного типа или типа с плавающей точкой. Тип возвращаемого значения – с плавающей точкой (кроме оператора **ABS** – его тип определяется типом аргумента).

```
PROGRAM PLC_PRG
VAR
   iVar1:     INT := -11;
   iVar2:     INT := 2;
   iResult:   INT;
   rResult1:  REAL;
   rResult2:  REAL;
   rResult3:  REAL;
   rResult4:  REAL;
   rResult5:  REAL;
END_VAR

// Область кода

// Получение значения по модулю
// iResult получит значение 11
iResult := ABS(iVar1);

// Извлечение квадратного корня
rResult1 := SQRT(iVar2);

// Экспонента
rResult2 := EXP(iVar2);

// Возведение в степень (второй аргумент – показатель степени)
rResult3 := EXPT(iVar2, 3);

// Натуральный логарифм
rResult4 := LN(iVar2);

// Логарифм по основанию 10
rResult5 := LOG(iVar2);
```

Операции извлечения корня и взятия логарифма имеют смысл только для положительных чисел. Если же аргументом будет отрицательное число, то в результате будет возвращен NaN. NaN – это особое состояние числа с плавающей точкой, которое может быть получено в результате некорректных математических операций. Для проверки на NaN можно использовать функции из библиотеки **FloatingPointUtils**.

5.6.6 Тригонометрические операторы

Аргументами тригонометрических операторов являются переменные или значения целочисленного типа или типа с плавающей точкой, определяющие угол в радианах (для арксинуса, арккосинуса и арктангенса аргументы являются безразмерными). Тип возвращаемого значения – с плавающей точкой (можно использовать как REAL, так и LREAL).

```
PROGRAM PLC_PRG
VAR
   // 0.785398 радиан соответствуют 45°
   rAngleInRadian:    REAL := 0.785398;
   rValue:            REAL := 0.707106;

   rResult1:           REAL;
   rResult2:           REAL;
   rResult3:           REAL;
   rResult4:           REAL;
   rResult5:           REAL;
   rResult6:           REAL;
END_VAR

// Область кода

// Синус
// rResult1 получит значение ≈ 0.707106
rResult1 := SIN(rAngleInRadian);

// Косинус
// rResult2 получит значение ≈ 0.707106
// (значения синуса и косинуса 45° совпадают)
rResult2 := COS(rAngleInRadian);

// Тангенс
// rResult3 получит значение ≈ 1
rResult3 := TAN(rAngleInRadian);

// Арксинус
// rResult4 получит значение ≈ 0.78539
rResult4 := ASIN(rValue);

// Арккосинус
// rResult5 получит значение ≈ 0.78539
rResult5 := ACOS(rValue);

// Арктангенс
// rResult6 получит значение ≈ 0.61548
rResult6 := ATAN(rValue);
```

5.6.7 Операторы выбора

Аргументами операторов выбора являются переменные или значения любого типа. Тип возвращаемого значения определяется типом аргументов.

```
PROGRAM PLC_PRG
VAR
   xSel:          BOOL;
   sSel:          STRING;

   iVar1:         INT := 10;
   iVar2:         INT := 20;
   iVar3:         INT := 30;
   iMax:          INT;
   iMin:          INT;

   rVar:          REAL := 15.25;
   rLimit:        REAL;

   usiIndex:      USINT := 2;
   udiMux1:       UDINT;
   udiMux2:       UDINT;
END_VAR

// Область кода
// SEL - оператор бинарного выбора
// Первый аргумент – типа BOOL
// Если он имеет значение FALSE, то оператор возвращает значение второго аргумента
// Если он имеет значение TRUE, то оператор возвращает значение третьего аргумента
sSel := SEL(xSel, 'Норма', 'Авария');

// Определение максимального значения из произвольного числа аргументов
iMax := MAX(iVar1, iVar2, iVar3);

// Определение минимального значения из произвольного числа аргументов
iMin := MIN(iVar1, iVar2, iVar3);

// LIMIT - оператор ограничения значения
// Первый аргумент – минимально допустимое значение
// Второй аргумент – контролируемое значение
// Третий аргумент - максимально допустимое значение
// Если контролируемое значение меньше минимального, то оператор возвращает
// минимальное значение
// Если контролируемое значение больше максимального, то оператор возвращает
// максимальное значение
// Если контролируемое значение находится в диапазоне между минимальным и максимальным
// (включая границы диапазона), то оператор возвращает контролируемое значение

// rLimit получит значение 12.0 (так как 15.25 > 12.0)
rLimit := LIMIT(10.0, rVar, 12.0);

// MUX - оператор множественного выбора
// Первый аргумент – целочисленный индекс значения (нумерация с 0)
// Остальные аргументы (в произвольном количестве) – доступные значения
// В зависимости от индекса возвращается то или иное значение

// udiMux1 получит значение 30
udiMux1 := MUX(usiIndex, 10, 20, 30);

// udiMux2 получит значение 20 – так как значение с индексом 2 отсутствует,
// то будет использовано последнее значение из доступных
udiMux2 := MUX(usiIndex, 10, 20);

// для строковых переменных использование операторов MIN, MAX и LIMIT не имеет смысла
```

5.6.8 Операторы конверсии типов

Операторы конверсии типов были рассмотрены в п. 5.3.6.

Кроме них существуют еще два специфических оператора конверсии – **TRUNC** и **TRUNC_INT**. Они аналогичны операторам **TO_DINT** и **TO_INT**, примененным к значениям типа **REAL**, но имеют одно отличие: если стандартные операторы конверсии производят округление значения к ближайшему целому, то операторы **TRUNC** и **TRUNC_INT** «отсекают» его дробную часть.

```
PROGRAM PLC_PRG
VAR
   diResult1:    DINT;
   diResult2:    DINT;
   diResult3:    DINT;
   diResult4:    DINT;
END_VAR

// Область кода

// diResult1 получит значение 1
diResult1 := TO_DINT(1.4);

// diResult2 получит значение 2
diResult2 := TO_DINT(1.6);

// diResult3 получит значение 1
diResult3 := TRUNC(1.4);

// diResult4 получит значение 1
diResult4 := TRUNC(1.6);
```

5.6.9 Оператор присваивания

Мы уже множество раз использовали оператор присваивания. Единственное, что можно добавить – допускается групповое присваивание в рамках одного выражения.

```
PROGRAM PLC_PRG
VAR
   iVar1: INT;
   iVar2: INT;
   iVar3: INT;
END_VAR

// Область кода

// Все переменные получат значение 10
iVar1 := iVar2 := iVar3 := 10;
```

5.6.10 Операторы определения размера

Оператор **SIZEOF** определяет размер аргумента в байтах. Аргумент может быть переменной любого типа. Тип возвращаемого значения зависит от размера переменной:

- если размер переменной меньше 256 байт, то тип возвращаемого значения – **USINT**;
- если размер переменной в диапазоне 256…65535 байт, то тип возвращаемого значения – **UINT**;
- если размер переменной в диапазоне 65536…4294967295 байт, то тип возвращаемого значения – **UDINT**;
- если размер переменной больше 4294967295 байт, то тип возвращаемого значения – **ULINT**.

В **CODESYS V3.5 SP16** был добавлен оператор **XSIZEOF**. Этот оператор работает аналогично **SIZEOF**, но тип возвращаемого значения – всегда **UDINT** для 32-битных платформ и **ULINT** для 64-битных платформ. Начиная с этой версии CODESYS рекомендуется использовать только оператор **XSIZEOF**.

```
PROGRAM PLC_PRG
VAR
   udiSize1:     UDINT;
   udiSize2:     UDINT;
   udiSize3:     UDINT;
   udiSize4:     UDINT;

   iVar:         INT;
   rVar:         REAL;
   wsVar:        WSTRING;
   awVar:        ARRAY [0..11] OF WORD;
END_VAR

// Область кода

// udiSize1 получит значение 2
udiSize1 := XSIZEOF(iVar);

// udiSize2 получит значение 4
udiSize2 := XSIZEOF(rVar);

// udiSize3 получит значение 162
udiSize3 := XSIZEOF(wsVar);

// udiSize4 получит значение 24
udiSize4 := XSIZEOF(awVar);
```

5.6.11 Остальные операторы

В прошлых пунктах мы обсудили основные операторы, используемые в большинстве пользовательских проектов. Нерассмотренными остались:

- операторы, используемые для работы с указателями. Указатели – слишком сложная тема для «первого старта», поэтому они остались за пределами данного документа;
- операторы **INDEXOF** и **INI**, сохраненные для совместимости с проектами, созданными в среде **CoDeSys V2.3**. Эти операторы не должны использоваться в проектах, создаваемых в CODESYS V3.5;
- системные операторы. Названия таких операторов начинаются с символа __ (например, __**VARINFO**). Эти операторы используются редко и, в основном, только разработчиками библиотек, поэтому они тоже в данном документе не рассматриваются.

5.6.12 Порядок выполнения операторов

Выражения обычно состоят из нескольких операторов. В некоторых случаях порядок выполнения операторов играют существенную роль. См. пример ниже:

```
PROGRAM PLC_PRG
VAR
   uiVar:   UINT := 200;
   xAlarm:  BOOL;
END_VAR

// Область кода

// xAlarm получит значение FALSE
xAlarm := NOT uiVar < 123;
```

Разработчик мог ожидать следующего поведения: переменная **uiVar** имеет значение **200** – это больше, чем **123**. Поэтому оператор сравнения вернет значение **FALSE**. Далее отработает оператор **NOT**, и переменная **xAlarm** получит значение **TRUE**.

Но в действительности переменная **xAlarm** получит значение **FALSE**. Это связано с тем, что порядок выполнения операторов является иным: сначала оператор **NOT** будет применён к переменной **uiVar**, в результате чего будет получено значение **65335** (потому что оператор **NOT** является побитовым; см. более подробное разъяснение в п. 5.6.2). После этого будет произведено сравнение – так как **65335** больше, чем **123**, то результатом сравнения будет значение **FALSE**.

Таблица приоритетов операторов приведена в онлайн-справке CODESYS.

Операторы с одинаковым приоритетом обрабатываются в порядке «слева направо».

Для управления приоритетом используются скобки – размещенные в них операторы всегда выполняются первыми. Добавим скобки в наш пример, чтобы обеспечить выполнение оператора сравнения первым:

```
// xAlarm получит значение TRUE
xAlarm := NOT(uiVar < 123);
```

5.7 Управляющие операторы

В прошлом разделе мы рассмотрели базовые операторы, выполняющие простейшие действия – например, арифметические и логические операции. Но чтобы реализовать даже самый простой алгоритм – потребуется использовать операторы, управляющие ходом выполнения программы. В этом разделе мы рассмотрим управляющие операторы языка ST. Надо сказать, что все эти операторы могут вызываться внутри друг друга – то есть внутри IF можно использовать CASE, FOR и сам IF, и т. д.

5.7.1 Оператор условного выбора IF

Оператор **IF** – наиболее часто используемый управляющий оператор. Он позволяет проверить одно или несколько условий, и если хотя бы одно из условий является истинным – выполнить заданные для этого условия выражения. После выполнения выражений производится выход из оператора – то есть оставшиеся условия уже не проверяются.

Рассмотрим работу оператора на примере сигнализации выхода значения температуры за допустимые границы:

```
PROGRAM PLC_PRG
VAR
   rRoomTemp:                    REAL;

   xHighTempAlarm:               BOOL;
   xLowTempAlarm:                BOOL;
END_VAR
VAR CONSTANT
   c_rHighTempAlarm:             REAL := 32.5;
   c_rLowTempAlarm:              REAL := 13.0;
END_VAR

// Область кода

IF rRoomTemp > c_rHighTempAlarm THEN
   xHighTempAlarm := TRUE;
ELSIF rRoomTemp < c_rLowTempAlarm THEN
   xLowTempAlarm := TRUE;
ELSE
  xHighTempAlarm := FALSE;
  xHighTempAlarm := FALSE;
END_IF
```

Если условие в операторе **IF** выполняется (значение переменной **rRoomTemp** больше константы **c_rHighTempAlarm**), то переменной **xHighTempAlarm** будет присвоено значение **TRUE** и произойдет выход из оператора (следующее условие проверяться не будет). Если же условие не выполняется, то произойдет проверка следующего условия, размещенного во вложенном операторе **ELSIF**. Если условие в **ELSIF** выполняется (значение переменной **rRoomTemp** меньше константы **c_rLowTempAlarm**), то переменной **xLowTempAlarm** будет присвоено значение **TRUE** и произойдет выход из оператора (следующее условие проверяться не будет). Если ни одно из условий в **IF** и **ELSIF** не выполняется (то есть значение температуры находится в допустимых пределах), то произойдет выполнение выражений, размещенных во вложенном операторе **ELSE** – присвоение переменным **xHighTempAlarm** и **xLowTempAlarm** значения **FALSE**.

Использование вложенных операторов **ELSIF** и **ELSE** является необязательным. Внутри оператора **IF** может быть размещено произвольное число операторов **ELSIF**.

Отдельно рассмотрим конкретный случай, в котором использование оператора **IF** является лишним:

```
PROGRAM PLC_PRG
VAR
   xMotorControlButton:           BOOL;
   xStartMotor:                   BOOL;
END_VAR

// Область кода

IF xMotorControlButton THEN
   xStartMotor := TRUE;
ELSE
   xStartMotor := FALSE;
END_IF

// Более короткий вариант этого же кода без использования оператора IF
xStartMotor := xMotorControlButton;
```

5.7.2 Оператор множественного выбора CASE

Оператор **CASE** позволяет сравнить значение заданной целочисленной переменной (селектора) с набором констант или целочисленных значений (метками), и в случае совпадения выполнить заданные для этой метки выражения. После выполнения выражений производится выход из оператора.

Рассмотрим работу оператора на синтетическом примере:

```
PROGRAM PLC_PRG
VAR
   iSelector:                     INT := 2;

   xOutput1:                      BOOL;
   xOutput2:                      BOOL;
   xOutput3:                      BOOL;
   xOutput4:                      BOOL;
   xOutput5:                      BOOL;
END_VAR

// Область кода
CASE iSelector OF

   0:
      xOutput1 := TRUE;

   1, 5:
      xOutput2 := TRUE;
      xOutput3 := TRUE;

   2..4:

      xOutput4 := TRUE;

ELSE
   xOutput5 := TRUE;
END_CASE
```

Как видно из примера – метка может включать в себя несколько значений, перечисленных через запятую, или диапазон. При этом значения одной из меток не должны совпадать со значениями других. Вложенный оператор **ELSE** является необязательным; размещенные в нём выражения выполняются в том случае, если значение селектора не совпало ни с одной из меток.

```
// Вариант этого же кода с использованием оператора IF

IF iSelector = 0 THEN
   xOutput1 := TRUE;
ELSIF iSelector = 1 OR iSelector = 5 THEN
   xOutput2 := TRUE;
   xOutput3 := TRUE;
ELSIF iSelector >=2 AND iSelector <=4 THEN
   xOutput4 := TRUE;
ELSE
   xOutput5 := TRUE;
END_IF
```

За счёт отсутствия необходимости явно прописывать сравнения вариант кода с оператором **CASE** выглядит очень лаконично.

Часто в качестве меток оператора **CASE** используются элементы перечисления:

```
// Код перечисления
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE WORK_MODE :
(
   // Режим местного управления
   LOCAL  := 0,
   // Режим дистанционного управления
   REMOTE := 1,
   // Режим автоматического управления
   AUTO   := 2
);
END_TYPE


PROGRAM PLC_PRG
VAR
   eWorkMode:   WORK_MODE;
END_VAR

// Область кода

CASE eWorkMode OF

   WORK_MODE.LOCAL:
      // тут будет код с выражениями для режима местного управления

   WORK_MODE.REMOTE:
      // тут будет код с выражениями для режима дистанционного управления

   WORK_MODE.AUTO:
      // тут будет код с выражениями для режима автоматического управления
END_CASE
```

Также оператор **CASE** может использоваться для создания асинхронно выполняемого кода. В рамках «первого старта» этот вопрос не рассматривается; примеры асинхронного кода приведены, например, в документах **CODESYS V3.5. Архивация** и **CODESYS V3.5. Реализация нестандартных протоколов**.

5.7.3 Оператор цикла FOR

Оператор **FOR** применяется для организации цикла с заранее известным числом итераций. Обычно он используется для операций над массивами данных.

В п. 5.3.7, рассматривая массивы, мы уже приводили примеры использования этого цикла. Повторим один из них:

```
PROGRAM PLC_PRG
VAR
   arTemp:                 ARRAY [1..c_iMaxTempPointCount] OF REAL;
   arTempHighAlarmLimit:   ARRAY [1..c_iMaxTempPointCount] OF REAL;
   axAlarm:                ARRAY [1..c_iMaxTempPointCount] OF BOOL;
   i:                      INT;
END_VAR
VAR CONSTANT
   c_iMaxTempPointCount:  INT := 100;
END_VAR

// Область кода
FOR i := 1 TO c_iMaxTempPointCount DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

Переменная **i** называется итератором (счетчиком) цикла. Эта переменная должна принадлежать целочисленному знаковому типу (обычно используется тип **INT** или **DINT**). После каждого выполнения тела цикла происходит изменение значения итератора – по умолчанию на **+1**. Пользователь может сам задать «шаг» итератора с помощью вложенного оператора **BY**. После этого сразу происходит переход к следующей итерации – то есть весь цикл выполняется «от начала и до конца».

```
// i будет принимать значения 1, 3, 5, 7 и т. д.
FOR i := 1 TO c_iMaxTempPointCount BY 2 DO
   axAlarm[i] := arTemp[i] > arTempHighAlarmLimit[i];
END_FOR
```

После завершения последней итерации происходит последнее изменение итератора – то есть после завершения цикла из примера выше переменная **i** получит значение **101**.

Пример ниже напоминает нам о том, что со строками можно работать как с массивами, а также демонстрирует цикл с уменьшением итератора и досрочным завершением цикла после нахождения в строке нужного символа с помощью оператора **EXIT**:

```
PROGRAM PLC_PRG
VAR
   sPouName:          STRING := 'Device.Application.PLC_PRG';
   i:                 INT;
END_VAR
VAR CONSTANT
   c_sCharSeparator: STRING(1) := '.';
END_VAR

// Область кода
FOR i := LEN(sPouName) - 1 TO 0 BY -1 DO
   IF sPouName[i] = c_sCharSeparator[0] THEN
         EXIT;
   END_IF
END_FOR
```

У нас есть строка **sPouName**, содержащая путь к программному объекту в проекте. Предположим, мы хотим вырезать из этого пути имя программного объекта (**PLC_PRG**) для последующего логирования. Для этого нам нужно определить номер символа в строке, с которого это имя начинается.

Для этого мы можем «пройтись» по строке как по массиву от последнего индекса к начальному (**BY -1**). Индексация массивов строк начинается с нуля – так что конечный индекс нам известен. Для определения начального индекса мы используем функцию **LEN** из библиотеки **Standard**, которая возвращает число символов строки. Так как мы работаем со строкой как с массивом, то нам требуется вычесть из длины строки единицу (потому что первому символу строки соответствует элемент массива с номером **0**).

В теле цикла мы проверяем, является ли текущий обрабатываемый символ строки точкой. Если является – то мы досрочно завершаем цикл с помощью оператора **EXIT**, чтобы не выполнять уже ненужные оставшиеся итерации.

В результате выполнения этого примера **i** примет значение **18** (это номер символа – последней точки строки). Зная этот номер, мы можем «вырезать» часть строки, размещенную следом за ним и представляющую собой имя интересующего нас программного объекта.

Еще один доступный вложенный оператор для цикла **FOR** – это **CONTINUE**. Он позволяет прервать текущую итерацию цикла и сразу перейти к следующей. В примере ниже он используется для того, чтобы избежать деления на **0**:

```
PROGRAM PLC_PRG
VAR
   // предположим, что массивы заполняются значениями в другом фрагменте программы
   arDividend:       ARRAY [1..c_iCountOfNumbers] OF REAL;
   aiDivisor:        ARRAY [1..c_iCountOfNumbers] OF INT;
   arResult:         ARRAY [1..c_iCountOfNumbers] OF REAL;
   i:                INT;
END_VAR
VAR CONSTANT
   c_iCountOfNumbers: INT := 5;
END_VAR

// Область кода
FOR i := 1 TO c_iCountOfNumbers DO

   IF aiDivisor[i] = 0 THEN
       CONTINUE;
   ELSE
      arResult[i] := arDividend[i] / aiDivisor[i];
   END_IF

END_FOR
```

В принципе, можно было бы переписать этот фрагмент кода без оператора **CONTINUE**:

```
PROGRAM PLC_PRG
VAR
   // предположим, что массивы заполняются значениями  в другом фрагменте программы
   arDividend:       ARRAY [1..c_iCountOfNumbers] OF REAL;
   aiDivisor:        ARRAY [1..c_iCountOfNumbers] OF INT;
   arResult:         ARRAY [1..c_iCountOfNumbers] OF REAL;
   i:                INT;
END_VAR
VAR CONSTANT
   c_iCountOfNumbers: INT := 5;
END_VAR

// Область кода
FOR i := 1 TO c_iCountOfNumbers DO

   IF aiDivisor[i] <> 0 THEN
       arResult[i] := arDividend[i] / aiDivisor[i];
   END_IF

END_FOR
```

Вообще, в реальных проектах **CONTINUE** применяется достаточно редко.

Поскольку оператор **FOR** обычно используется совместно с массивами – то рекомендуем вам при желании перечитать соответствующий пункт документа.

Последнее, что стоит сказать – не стоит манипулировать значением итератора цикла в теле самого цикла. В большинстве случаев это не приносит никакой пользы, а только создает сложно читаемый и подверженный ошибкам код.

5.7.4 Остальные управляющие операторы

Мы рассмотрели три основных управляющих оператора языка ST – IF, CASE и FOR. Есть еще несколько операторов, которые на практике используются крайне редко, поэтому мы приведем здесь только краткий их обзор без примеров:

- операторы **WHILE** и **REPEAT** используются для создания цикла с заранее неизвестным числом итераций; сигналом окончания цикла служит выполнение заданного условия (для **WHILE** проверка условия выполняется до выполнения выражений, для **REPEAT** – после). На практике в большинстве случаев число итераций заранее известно, так как, например, размер массивов (которые в основном и обрабатываются в циклах) жестко определяется на этапе их создания. Кроме того, программы контроллера обычно привязаны к циклическим задачам – то есть сами по себе выполняются в бесконечном цикле. Наконец, использование операторов **WHILE** и **REPEAT** новичками часто приводит к возникновению в программе бесконечного цикла – в результате чего возникает исключение и приложение контроллера перестает выполняться;
- оператор безусловного перехода **JMP** – аналог оператора **GOTO** из других языков программирования. Обычно использование этого оператора приводит к получению плохо читаемого и сложного в отладке «спагетти-кода», с высокой вероятностью содержащего ошибки;
- оператор выхода из программного объекта **RETURN**. Обычно без использования этого оператора можно легко обойтись, немного изменив код программы. Например, мы разрабатываем большой фрагмент кода, который должен выполняться только в том случае, если оператор ввёл в визуализации корректный пароль. С помощью оператора **RETURN** можно написать такой код:

```
// Область кода
IF sVisuPassword <> sExpectedPassword THEN
   RETURN;
ELSE
  // тут будут выражения, выполняемые при вводе корректного пароля
END_IF
// конец программного объекта
```

Этот фрагмент кода можно легко переписать и без оператора **RETURN**, просто изменив проверяемое условие:

```
// Область кода
IF sVisuPassword = sExpectedPassword THEN
   // тут будут выражения, выполняемые при вводе корректного пароля
END_IF
// конец программного объекта
```

5.8 Программные объекты

В примерах, которые мы разбирали до этого, весь код размещался в программе **PLC_PRG**. Эта программа автоматически создается вместе с новым проектом CODESYS. В принципе, вы можете ограничиться этой программой, добавив в неё весь код своего приложения – но это плохой подход. В результате вы получите одну программу из нескольких сотен (а, возможно, даже тысяч) строк кода, значительная часть из которых наверняка будет создана с помощью операций копирования-вставки. Эту программу будет сложно отлаживать и поддерживать; скорее всего, она будет подвержена ошибкам.

Поэтому более разумным вариантом является разделение программы на изолированные программные объекты, которые можно будет переиспользовать и редактировать независимо друг от друга. Такие объекты в CODESYS называются **POU** (Program Organization Unit).

Для создания программного объекта нужно нажать правой кнопкой мыши на узел **Application**, выбрать команду **Добавление объекта** и добавить объект **POU**.

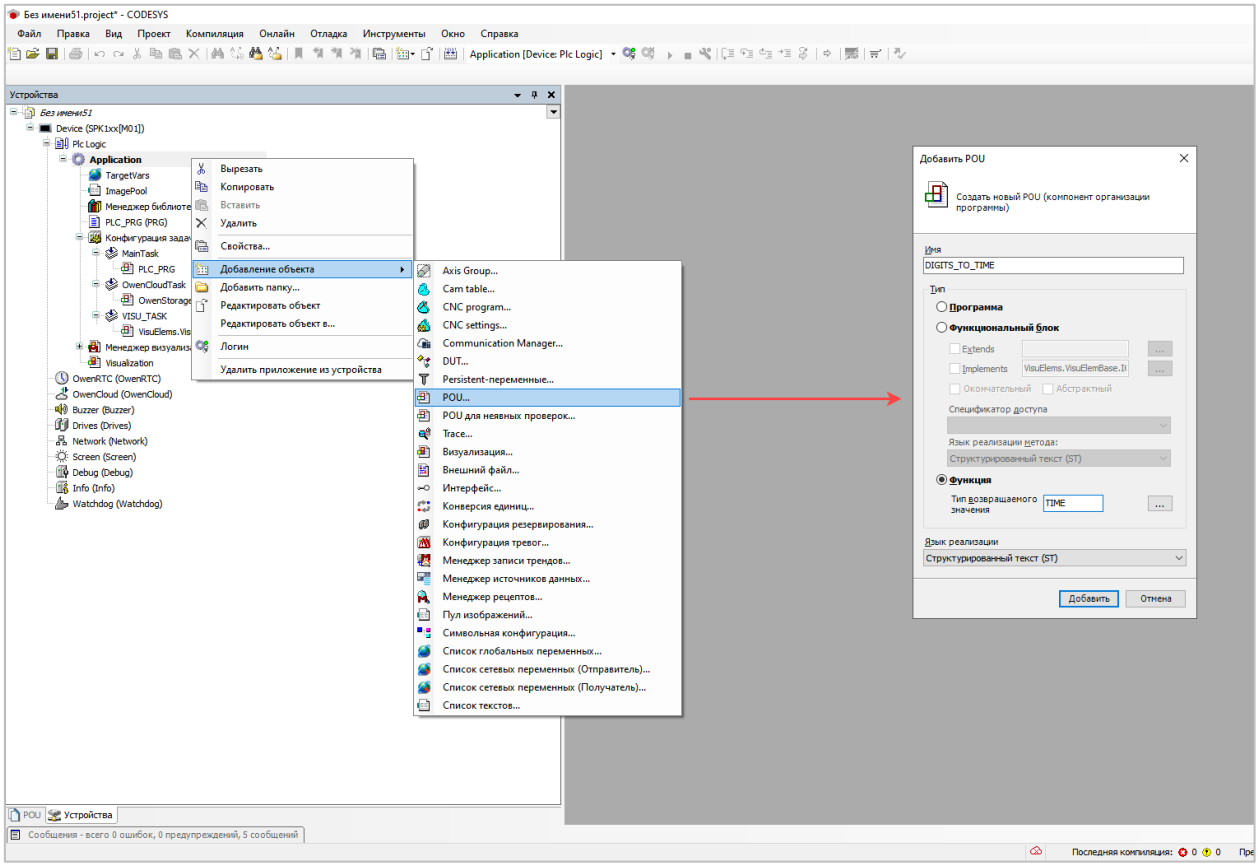


Рисунок 5.8.1 – Добавление POU в проект

Всего существует 3 типа программных объектов – функции, функциональные блоки и программы.

5.8.1 Функции

Функция – это программный объект, не имеющий памяти для сохранения значений своих переменных между вызовами. Функции используются для арифметических вычислений, масштабирования сигналов, конвертации данных из одного формата в другой, различных проверок (например, на принадлежность значения допустимому диапазону) и т. д. Так как функция не имеет памяти – то с ее помощью нельзя организовать счетчик, таймер и другие элементы, рассчитанные на сохранение промежуточных значений своих переменных.

В большинстве случаев внутри функции не объявляются функциональные блоки – потому что обычно этим блокам требуется память для сохранения значений между вызовами.

По этой же причине (отсутствие памяти) внутри функций нельзя объявить энергонезависимые переменные.

При создании функции указывается тип возвращаемого ею значения. Каждая функция имеет неявно создаваемую переменную, имя которой совпадает с именем функции – именно тип этой переменной и указывается при создании функции.

Ниже приведен пример объявления простейшей функции, которая используется для «сборки» переменной типа **TIME** из отдельных разрядов времени.

```
FUNCTION DIGITS_TO_TIME : TIME
VAR_INPUT
   usiHour:      USINT := 0;
   usiMinute:    USINT := 0;
   usiSecond:    USINT := 0;
END_VAR
VAR
END_VAR

// Область кода
DIGITS_TO_TIME := usiHour * T#1H + usiMinute * T#1M + usiSecond * T#1S;
```

Пример вызова этой функции в программе:

```
PROGRAM PLC_PRG
VAR
   tVar1:        TIME;
   tVar2:        TIME;
END_VAR

// Область кода

// tVar1 получит значение T#20H5M3S
tVar1 := DIGITS_TO_TIME(20, 5, 3);

// tVar2 получит значение T#12H7M1S
tVar2 := DIGITS_TO_TIME(usiHour := 12, usiMinute := 7, usiSecond:= 1);
```

При вызове функции можно не указывать имена ее входных переменных – это называется «неформальным вызовом». Порядок передачи значений в этом случае должен совпадать с порядком входных переменных функции. Если при вызове функции указываются имена ее входных переменных, то вызов называется «формальным». Порядок перечисления входов при формальном вызове функции может быть любым.

Если входные переменные функции имеют начальные значения, но они могут быть пропущены при ее вызове (если им не требуется передать какие-то значения). Например, нашу функцию можно вызвать так:

```
// tVar2 получит значение T#7M
tVar2 := DIGITS_TO_TIME(usiMinute := 7);
```

Функция может иметь дополнительные выходы, объявленные в области VAR_OUTPUT. Например, доработаем нашу функцию таким образом, чтобы она возвращала сигнал ошибки и обнуляла свой основной выход в тех случаях, когда введенные значения разрядов времени не соответствуют допустимым значениям для разрядов времени суток:

```
FUNCTION DIGITS_TO_TIME : TIME
VAR_INPUT
   usiHour:      USINT := 0;
   usiMinute:    USINT := 0;
   usiSecond:    USINT := 0;
END_VAR
VAR_OUTPUT
   xError:       BOOL;
END_VAR
VAR
END_VAR

// Область кода
DIGITS_TO_TIME := usiHour * T#1H + usiMinute * T#1M + usiSecond * T#1S;

xError := usiHour > 23 OR usiMinute > 59 OR usiSecond > 59;

IF xError THEN
    DIGITS_TO_TIME := T#0s;
END_IF
```

Пример вызова этой функции в программе с обращением к ее дополнительному выходу (для таких выходов допустим только формальный вызов):

```
VAR
   tVar1:                TIME;
   xWrongParams:         BOOL;
END_VAR

// Область кода

tVar1 := DIGITS_TO_TIME(20, 30, 30, xError => xWrongParams);
```

Значения выходов функции зависят только от значений ее входов. Каждый вызов функции никак не влияет на другие ее вызовы – вы можете вызывать функцию в вашем проекте сколько угодно раз с разными аргументами. При этом каждый вызов функции с одними и теми же аргументами приведет к одним и тем же значениям ее выходов. Это связано с тем, что, как уже упоминалось, переменные функции не сохраняют значения между ее вызовами.

5.8.2 Функциональные блоки

Функциональный блок (ФБ) – это программный объект, значения переменных которого сохраняются между вызовами. Все элементы, которым требуется память – счетчики, таймеры, блоки архивации, блоки обмена и т. д. – представляют собой функциональные блоки.

Ниже приведен пример объявления функционального блока сбора статистики изменения значения: пока вход **xEnable** имеет значение **TRUE**, блок «запоминает» наименьшее и наибольшее значения, которое принимает вход **iValue**, и передает их на выходы **iMin** и **iMax** соответственно. Если вход **xReset** принимает значение **TRUE**, то эти выходы инициализируются значением **iValue** – происходит «сброс» статистики. Выход **xActive** сигнализирует о работе блока: пока вход **xEnable** имеет значение **TRUE**, то этот выход тоже имеет значение **TRUE**.

```
FUNCTION_BLOCK STAT
VAR_INPUT
   xEnable:      BOOL;
   iValue:       INT;
   xReset:       BOOL;
END_VAR
VAR_OUTPUT
   xActive:      BOOL;
   iMin:         INT;
   iMax:         INT;
END_VAR
VAR
END_VAR

// Область кода

// при запуске начинаем вести статистику относительно первого полученного значения
IF (xEnable AND NOT xActive) OR xReset THEN

         iMin := iValue;
         iMax := iValue;

END_IF

// определение минимума и максимума
IF xEnable THEN

   IF iValue < iMin THEN
      iMin := iValue;
   END_IF

   IF iValue > iMax THEN
      iMax := iValue;
   END_IF

END_IF

xActive := xEnable;
```

Функциональные блоки обычно не используются в функциях, потому что блокам нужна память для сохранения своих значений – а у функций памяти нет. Поэтому блоки в основном используются в других блоках и программах. Для работы с функциональным блоком нужно создать его экземпляр (их может быть несколько). Экземпляры представляют собой переменные, тип которых – функциональный блок (фактически при объявлении экземпляра функционального блока происходит объявление структуры его данных).

Пример объявления и вызова созданного выше функционального блока в программе:

```
PROGRAM PLC_PRG
VAR
   fbStat:                STAT;

   xStatEnable:           BOOL;
   xResetStatistics:      BOOL;
   iUserValue:            INT;
   iStatMin:              INT;
   iStatMax:              INT;
END_VAR

// Область кода

fbStat
(
   xEnable := xStatEnable,
   iValue  := iUserValue,
   xReset  := xResetStatistics,
   iMin    => iStatMin,
   iMax    => iStatMax
);
```

Обратите внимание на специфический оператор копирования значения из выходных переменных блока (=>).

Функциональные блоки поддерживают только формальный вызов (с указанием имен входов и выходов), при этом часть входов и выходов может быть пропущена (в нашем примере мы не обращаемся к выходу **xActive**).

Если требуется получать статистику по нескольким переменным – то потребуется объявить соответствующее количество экземпляров функциональных блоков (или массив из них).

```
PROGRAM PLC_PRG
VAR
   fbStat1:               STAT;
   fbStat2:               STAT;
   fbStat3:               STAT;

   afbStat:               ARRAY [1..3] OF STAT;
END_VAR
```

Все экземпляры блоков работают независимо друг от друга и каждый обладает собственной памятью.

Технически в функциональном блоке можно объявить энергонезависимые переменные; но при объявлении хотя бы одной такой переменной все переменные блока будут автоматически сохраняться в энергонезависимой памяти. Поэтому обычно внутри блоков энергонезависимые переменные не объявляются, чтобы не тратить эту память зря. По этой же причине экземпляры функциональных блоков редко объявляются в энергонезависимой памяти, хотя в ряде конкретных случаев это является вполне разумным подходом (например, если блок занимает относительно немного памяти).

При создании функционального блока доступно несколько галочек и опций (например, **EXTENDS** и **IMPLEMENTS**). Все эти опции могут применяться в случае использования объектно-ориентированного подхода к разработке приложения. В рамках данного документа этот подход не рассматривается.

5.8.3 Программы

Программа – это программный объект, значения переменных которого сохраняются между вызовами. Отличия программ от функциональных блоков заключаются в следующем:

- программу можно привязать к задаче;
- для программы невозможно объявить экземпляр;
- в программе допустимо объявление энергонезависимых переменных, и это не приводит к дополнительным тратам энергонезависимой памяти, как в случае с ФБ.

Программы – это высокоуровневые элементы, охватывающие значительные фрагменты проекта. Обычно в среднем проекте разумное число программ не превышает **5**, в крупном – **10**. Число функций и функциональных блоков в проектах существенно больше.

5.8.4 Сравнение функций, функциональных блоков и программ

Ниже приведена сравнительная таблица различных программных объектов:

Таблица 5.1 – Сравнение функций, функциональных блоков и программ

	Функция	**ФБ**	**Программа**
Сохранение значений переменных между вызовами	-	+	+
Использование через экземпляры	-	+	-
Энергонезависимые переменные	-	~	+
Вызов в функциях	+	~	~
Неформальный вызов	+	-	-
Наличие пространства имен	-	+	+

Символом «~» обозначаются случаи, в которых использование функционала является синтаксически корректным, но в большинстве случаев противоречащим логике.

5.8.5 Вложенные программные объекты (методы и др.)

Если нажать на любой из **POU** правой кнопкой мыши и использовать команду **Добавление объекта** – то можно добавить в него вложенные POU:

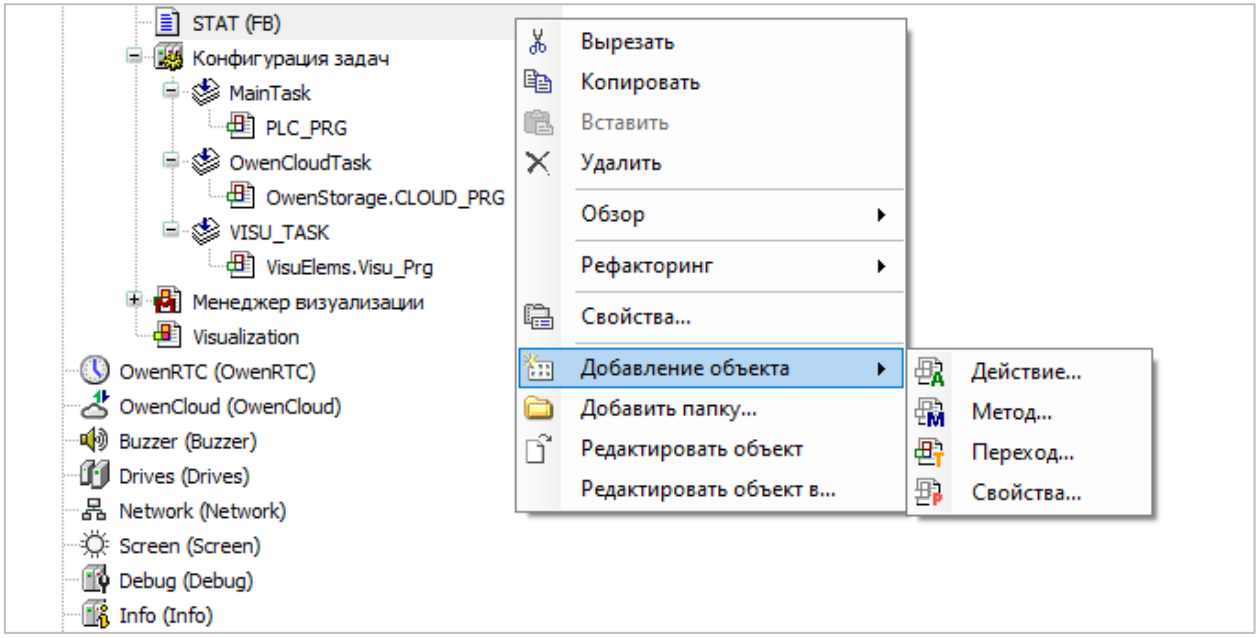


Рисунок 5.8.2 – Добавление вложенного POU

Ниже приведен краткий обзор вложенных программных объектов:

- **действие** позволяет выделить фрагмент POU в обособленный объект с собственным названием. В других языках программирования аналогичные объекты называются процедурами (при этом действие является процедурой без параметров). За исключением языка SFC – всегда предпочтительнее использовать метод, а не действие (его функциональность перекрывает функциональность действия);
- **метод** представляет собой функцию, встраиваемую в ФБ и имеющую доступ к его переменным. В отличие от действия – метод может иметь собственные входные, выходные и локальные переменные (а также константы). Точно также, как POU используются для повышения модульности проектов – методы могут использоваться для повышения модульности конкретного POU (например, функциональный блок записи архивов может включать в себя несколько сотен строк кода, среди которых есть очень похожие друг на друга фрагменты. Такие фрагменты можно оформить в виде методов);
- **переход** является аналогом действия, используемым в языке SFC для описания условий переходов между шагами;
- **свойство** представляет собой переменную, для которой автоматически создаются методы чтения (**Get**) и записи (**Set**).

В реальных проектах действия и переходы обычно применяются только в том случае, если в проекте используется язык **SFC**. Свойства являются одним из элементов объектно-ориентированного подхода и редко используются за его пределами. Методы являются удобным инструментом структурирования кода – они могут использоваться как в рамках объектно-ориентированного подхода, так и без него.

5.9 Задачи

Задачи – это объекты, определяющие условия вызова программ. Благодаря им происходит исполнение кода проекта. Список задач проекта отображается в компоненте **Конфигурация задач**.

В шаблоне проекта для контроллера ОВЕН присутствует 3 задачи:

- **MainTask** – задача пользовательской логики;
- **OwenCloudTask** – задача связи с облачным сервисом OwenCloud;
- **VISU_TASK** – задача визуализации.

Некоторые задачи автоматически создаются при добавлении в проект соответствующих компонентов (архиватора, конфигурации тревог и т. п.). Кроме того, у пользователя есть возможность создавать задачи самостоятельно – но начинающим пользователям мы настоятельно рекомендуем не создавать задачи и не редактировать настройки существующих задач. Это связано с тем, что принцип обработки в CODESYS достаточно сложен (используется механизм вытесняющей многозадачности) и мало документирован. Более подробную информацию о задачах вы можете почерпнуть в этой статье.

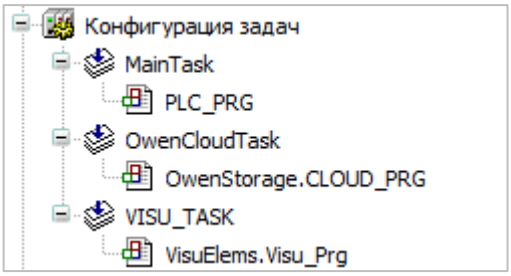


Рисунок 5.9.1 – Структура компонента Конфигурация задач

Задача имеет следующие настройки:

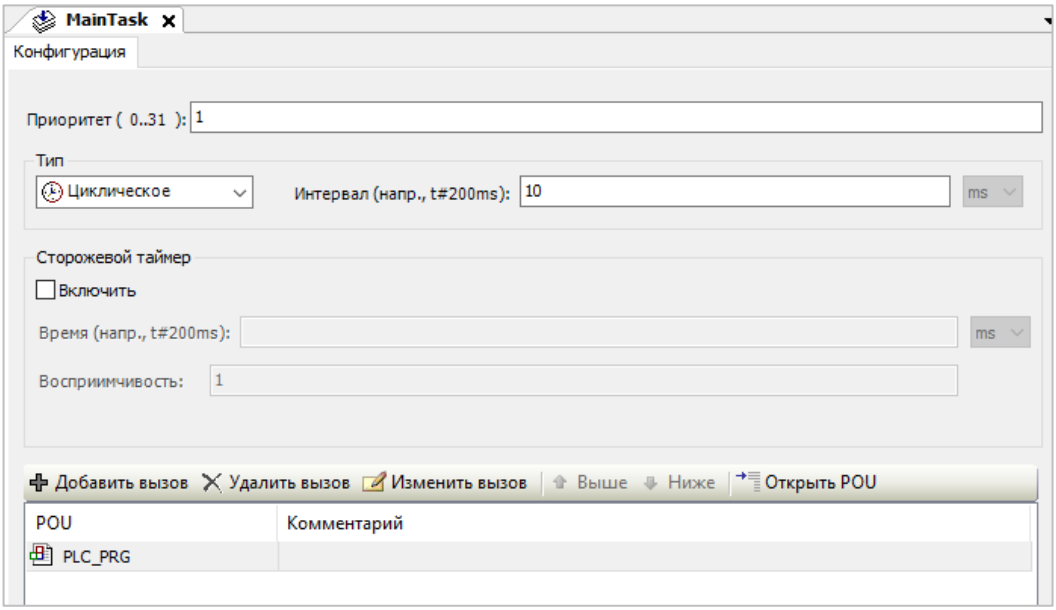


Рисунок 5.9.2 – Настройки задачи

- **приоритет** определяет «важность» задачи. Он имеет значение в тех случаях, когда в процессе работы контроллера наступает момент, в который несколько задач должны быть вызваны одновременно. В этой ситуации начнет выполняться задача с наивысшим приоритетом; запуск менее приоритетных задач будет отложен. В CODESYS наивысшему приоритету соответствует значение **0**, наименьшему – **31**. Задачи с приоритетами **0…15** имеют приоритет реального времени и могут вытеснять некоторые сервисы операционной системы;
- **тип задачи** определяет способ ее вызова. В основном, используются только задачи циклического типа. Другие типы задач не рекомендуются к использованию;
- **интервал вызова** определяет период, с которым вызывается циклическая задача;
- **сторожевой таймер** позволяет прервать выполнение задачи, если реальное время ее выполнения превысит время, заданное для сторожевого таймера. Параметр **Восприимчивость** определяет число последовательных вызовов задачи, в течение которых это условие должно выполняться. Кроме того, выполнение задачи будет прервано, если в пределах одного вызова задачи реальное время выполнения превысит заданное в (восприимчивость) раз. В реальных проектах сторожевые таймеры задач используются редко – на этапе отладки все «медленные» фрагменты кода, требующие много времени на исполнение, должны быть обнаружены и оптимизированы. Сделать это можно с помощью онлайн-мониторинга задач.

Если открыть конфигурацию задач при подключении к контроллеру, то на вкладке **Мониторинг** отобразится информация о реальном времени исполнения задачи. ***Обратите внимание***, что все времена отображаются в **микро**секундах, хотя интервал вызова задачи для контроллеров ОВЕН задается в **милли**секундах.

В момент запуска приложения происходят дополнительные операции, за счет которых первые несколько циклов задач выполняются существенно медленнее, чем последующие – из-за этого в столбце **Макс. время цикла** по умолчанию отображаются довольно большие значения. Поэтому после запуска приложения желательно обнулить статистику с помощью команды **Сброс**, чтобы увидеть достоверные показатели. Наиболее информативным параметром является среднее время цикла. В корректном проекте оно всегда должно быть меньше, чем заданное. Если среднее время цикла больше заданного – то это признак того, что проект нужно оптимизировать.

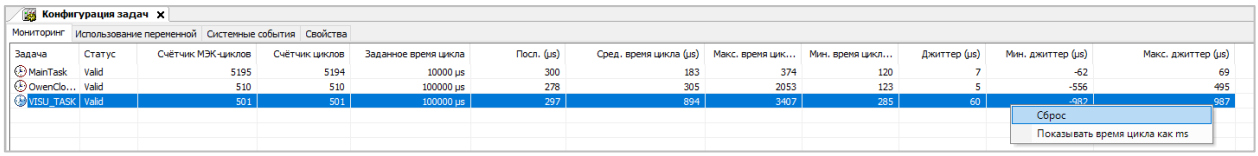


Рисунок 5.9.3 – Онлайн-мониторинг задач

Напоследок еще раз напомним – мы настоятельно рекомендуем начинающим пользователям не создавать задачи вручную и не редактировать настройки существующих задач.

5.10 Библиотеки

Для расширения возможностей CODESYS производители контроллеров и пользователи могут разрабатывать собственные элементы проекта (функциональные блоки, диалоги визуализации и т. д.) и распространять их в виде библиотек.

Библиотеки, разработанные компанией ОВЕН, и некоторые свободно распространяемые библиотеки от пользователей CODESYS доступны на сайте ОВЕН в разделе CODESYS V3/Библиотеки и компоненты.

Для использования библиотеки сначала требуется установить ее в репозиторий библиотек среды CODESYS, а затем добавить в менеджер библиотек конкретного проекта.

5.10.1 Установка библиотек

Для установки библиотеки в CODESYS нужно выбрать в меню «Инструменты» пункт **Репозиторий библиотек**, нажать кнопку **Установить** и указать путь к файлу нужной библиотеки.

Библиотеки могут распространяться в виде двух форматов:

- **.compiled-library** – для таких библиотеки доступ к исходным кодам запрещен;
- **.library** – данные библиотеки доступных в исходных кодах.

Обратите внимание, что если в окне выбора файла библиотеки указан неподходящий формат – то установить ее не получится. Поэтому рекомендуется переключить фильтр в режим **Все файлы**.

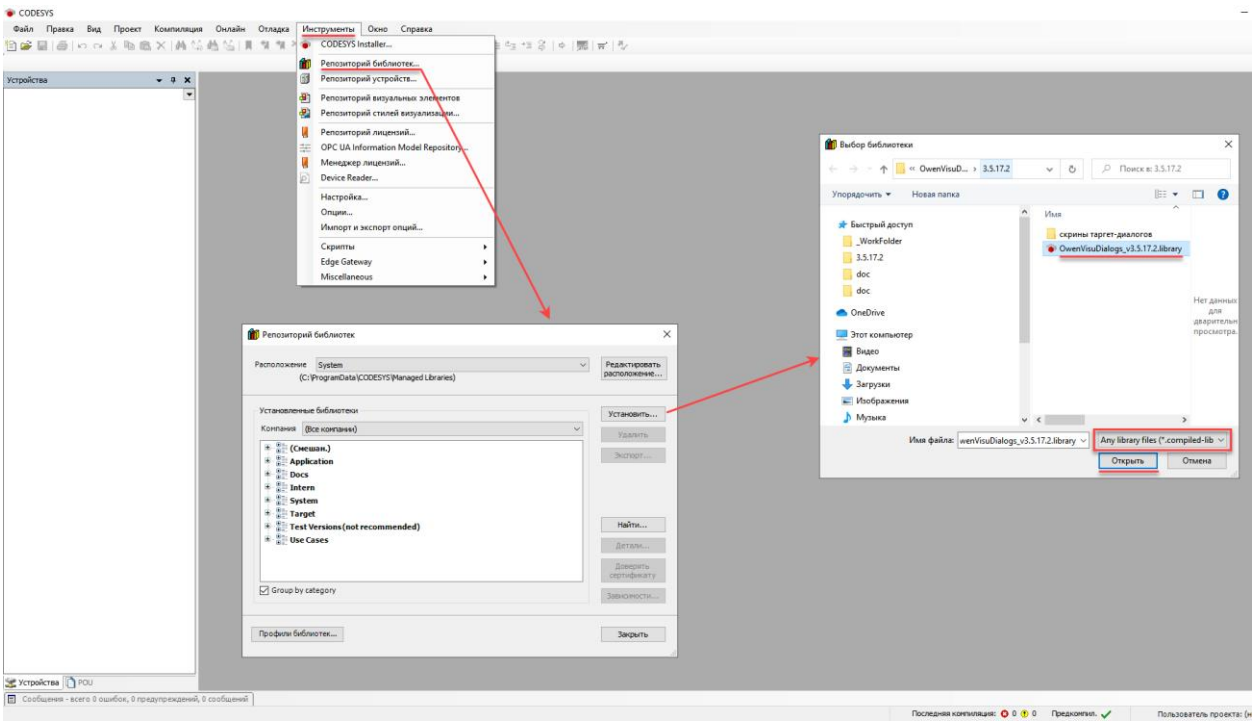


Рисунок 5.10.1 – Установка библиотеки в репозиторий библиотек

5.10.2 Добавление библиотеки в проект CODESYS

Для добавления библиотеки в проект CODESYS нужно перейти в компонент **Менеджер библиотек** и нажать кнопку **Добавить библиотеку**. Появится окно со списком выбора библиотеки – рекомендуется переключить его в режим плоского списка с помощью двух кнопок в правой части окна (по умолчанию список библиотек отображается в виде дерева, и найти в нем нужную библиотеку может быть довольно сложно).

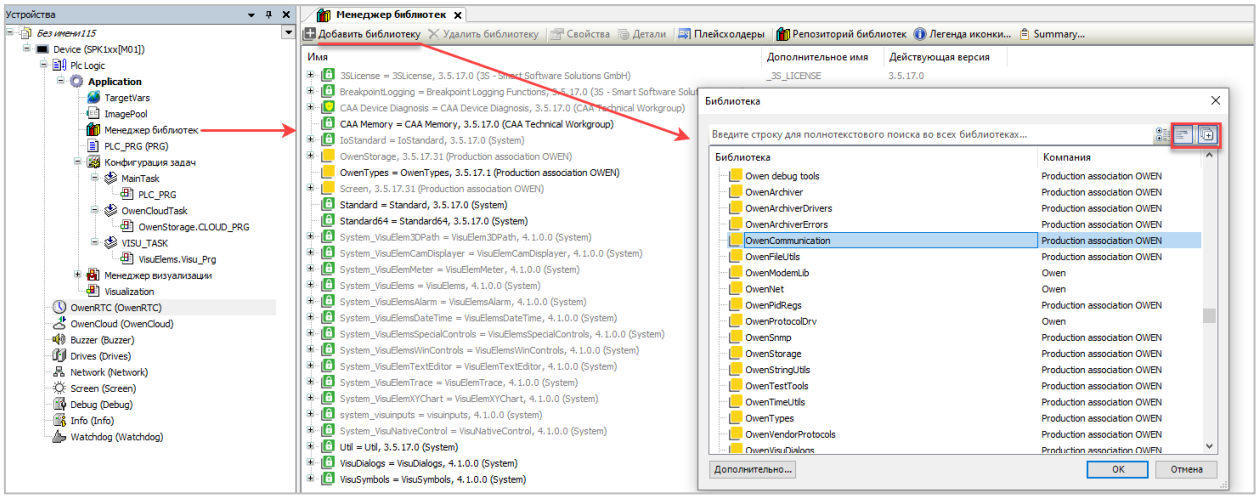


Рисунок 5.10.2 – Добавление библиотеки в менеджер библиотек проекта

5.10.3 Использование объектов библиотеки в проекте. Пространства имен

При выборе в менеджере библиотек какой-либо библиотеки в нижней части окна будет отображаться список объектов этой библиотеки. При выборе конкретного объекта отображается справочная информация по этому объекту.

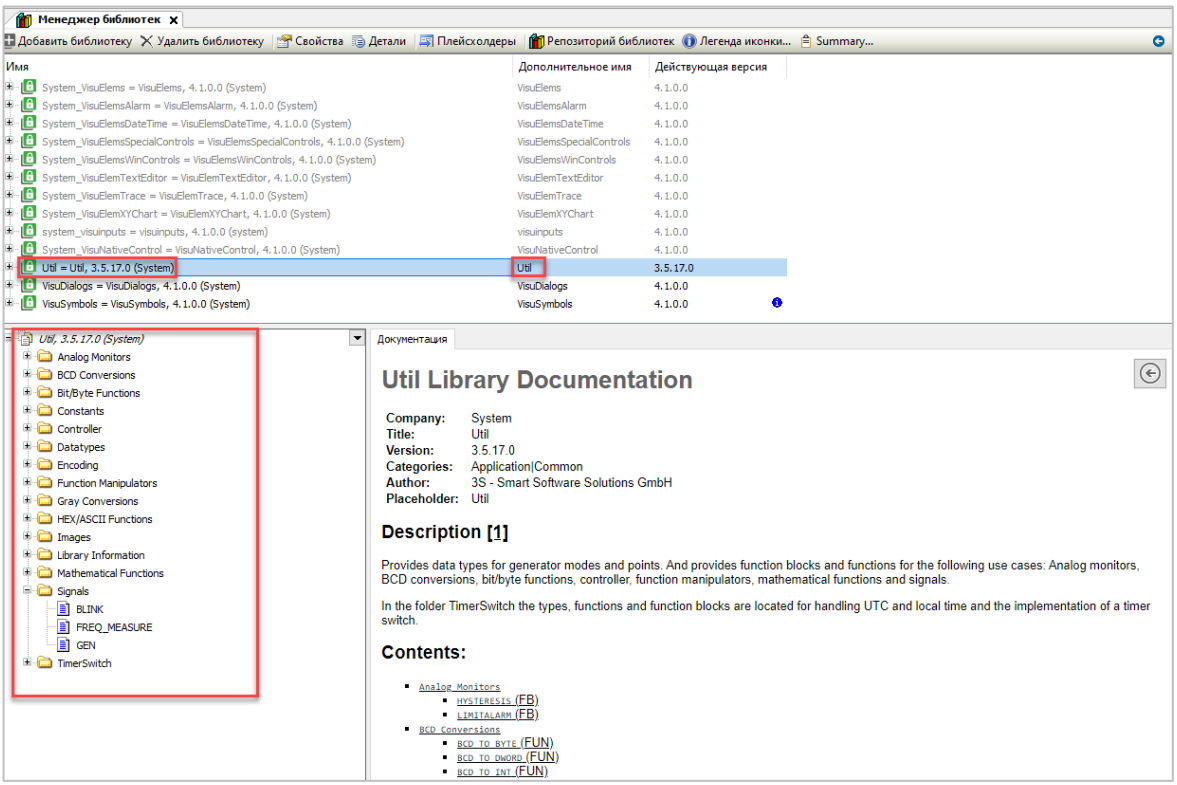


Рисунок 5.10.3 – Отображение объектов библиотеки и встроенной справки в менеджере библиотек

Для использования объекта библиотеки в проекте требуется указать его **пространство имен**. В менеджере библиотек пространства имен отображаются в столбце **Дополнительное имя**. Например, вам нужно объявить в программе экземпляр функционального блока **GEN** из библиотеки **Util**, которая имеет пространство имен **Util** (см. рис. выше). Это делается следующим образом:

```
PROGRAM PLC_PRG
VAR
   fbGen:         Util.GEN;
END_VAR
```

Мы уже раньше видели метод доступа к объектам «через точку» – например, когда рассматривали структуры, объединения и перечисления. Действительно, все эти объекты (а также функциональные блоки, программы, списки глобальных переменных и некоторые другие объекты дерева проекта) имеют собственные пространства имен, скрывающие их вложенные объекты.

Пространства имен позволяют легко определить принадлежность объекта, а также использовать различные объекты с совпадающими именами. Например, во множестве библиотек есть перечисление **ERROR** с кодами ошибок, и для каждой библиотеки эти коды – свои. Использование пространств имен позволяет избежать путаницы – всегда будет понятно, к какой библиотеке относится конкретный экземпляр перечисления.

```
// Пример скомпилируется только при добавлении библиотек CAA File и CAA SerialCom
PROGRAM PLC_PRG
VAR
   // Экземпляр перечисления ERROR из библиотеки CAA File
   eFileError:   FILE.ERROR;
   // Экземпляр перечисления ERROR из библиотеки CAA SerialCom
   eComError:    COM.ERROR;
END_VAR
```

Вернемся к блоку **GEN** из библиотеки **Util**. Этот блок используется для генерации различных сигналов – синусоиды, косинусоиды, «пилы» и т. д. Тип сигнала определяется значением входа **MODE**, который имеет тип **GEN_MODE** – это перечисление из этой же библиотеки.

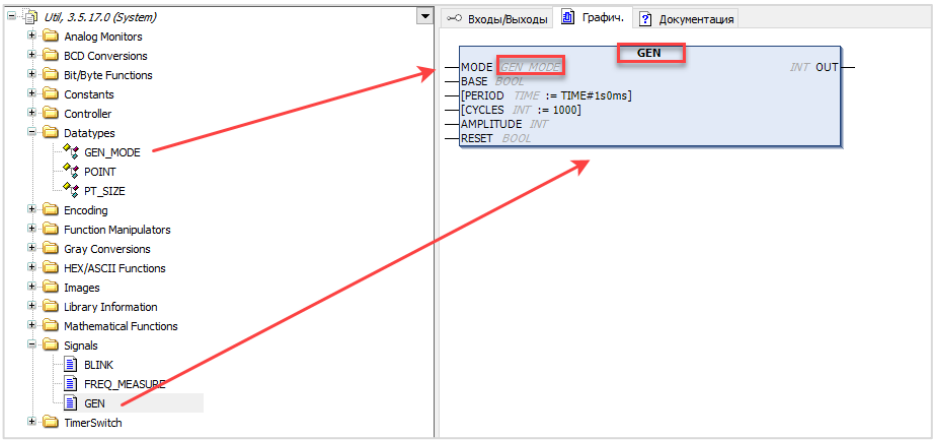


Рисунок 5.10.4 – Перечисление GEN_MODE в библиотеке Util

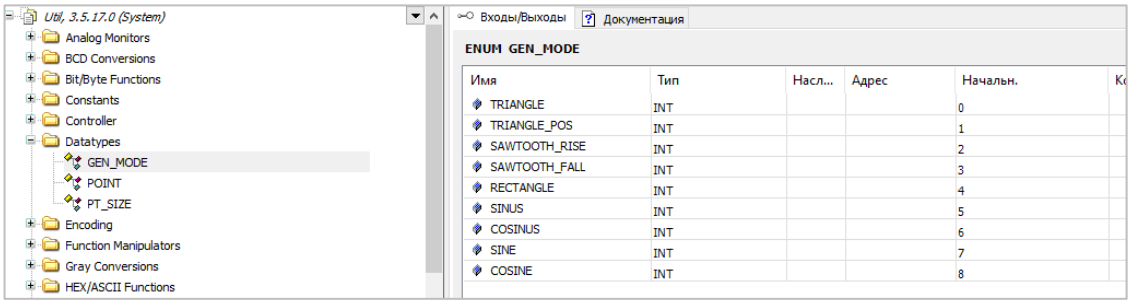


Рисунок 5.10.5 – Элементы перечисления GEN_MODE

При вызове экземпляра блока **GEN** в программе присвоение значения входу **MODE** происходит следующим образом:

```
PROGRAM PLC_PRG
VAR
   fbGen:         Util.GEN;
END_VAR

// Область кода
fbGen
(
   MODE       := Util.GEN_MODE.SINUS,
   BASE       := TRUE,
   PERIOD     := T#5S,
   AMPLITUDE  := 10
);
```

В данном случае при доступе к элементу перечисления библиотеки мы сначала указываем пространство имен библиотеки, а следом – пространство имен перечисления.

Библиотеки могут включать в себя вложенные библиотеки. Для доступа к объектам вложенных библиотек требуется последовательно указать пространства имен всех библиотек цепочки.

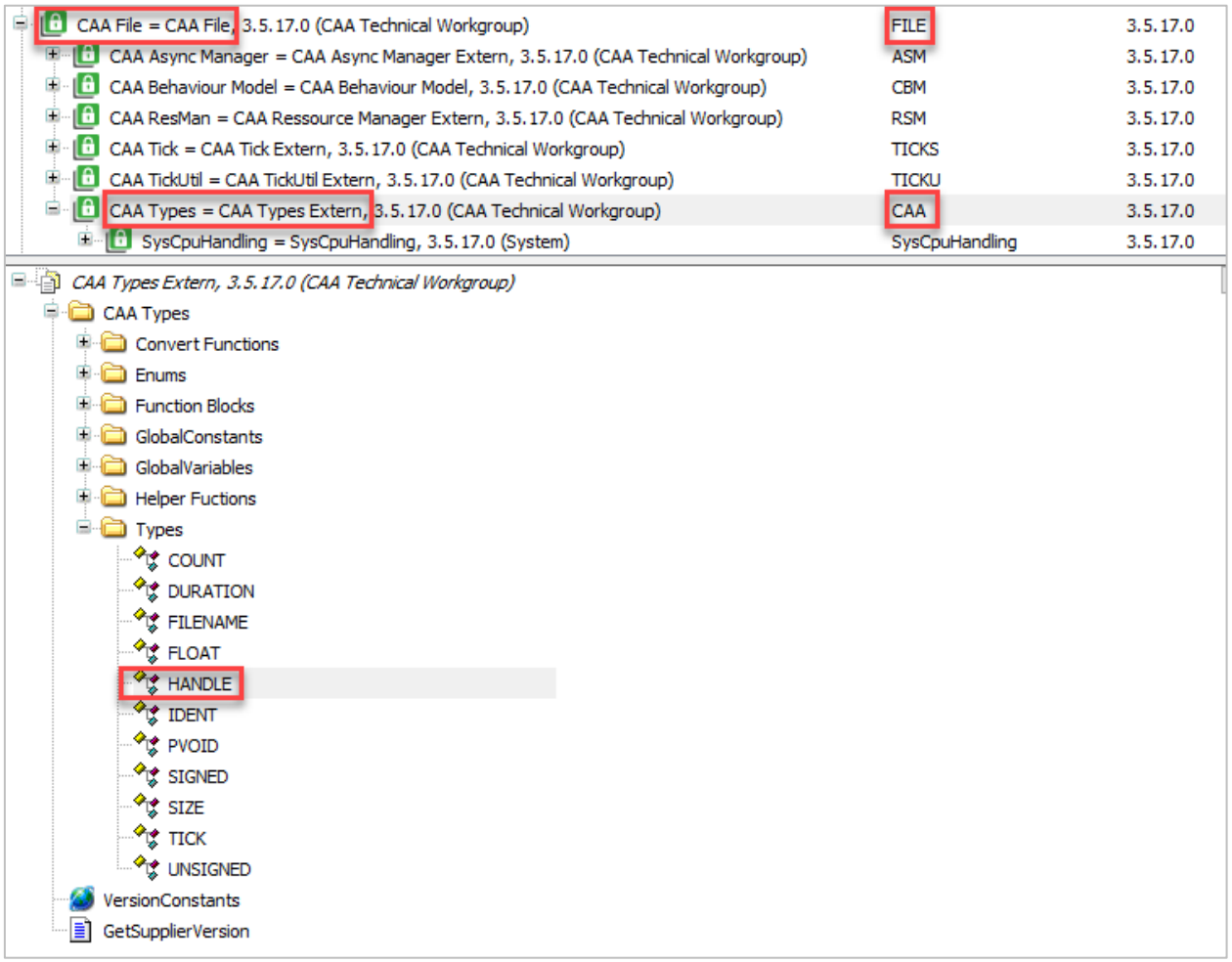


Рисунок 5.10.6 – Псевдоним HANDLE из библиотеки CAA Types, вложенной в библиотеку CAA File

```
PROGRAM PLC_PRG
VAR
   hFile: FILE.CAA.HANDLE;
END_VAR
```

Объекты библиотеки **Standard**, о которой мы поговорим в следующем пункте, обычно используются без пространства имен – поскольку их названия определены в стандарте МЭК 61131-3 и сложно перепутать эти блоки с какими-то другими.

5.10.4 Библиотека Standard

Наиболее часто используемой библиотекой является библиотека **Standard**. Эта библиотека содержит базовые блоки, определенные в стандарте МЭК 61131-3 и применяемые практически в любом проекте – детекторы фронта, счетчики, таймеры и т. д. Ниже приведены примеры использования самых востребованных блоков.

Детекторы фронта

Функциональный блок **R_TRIG** является детектором переднего фронта. Когда его вход **CLK** изменяет свое значение от **FALSE** к **TRUE** – на выходе **Q** генерируется единичный импульс. Блок **F_TRIG** работает аналогичным образом, но отслеживает спад сигнала – единичный импульс на его выходе генерируется тогда, когда вход **CLK** изменяет свое значение от **TRUE** к **FALSE**.

Эти блоки используются в тех случаях, когда требуется однократно выполнить какие-то операции при изменении значения логической переменной – например, отправить SMS при возникновении тревоги.



Рисунок 5.10.7 – Входы и выходы блоков R_TRIG / F_TRIG

```
PROGRAM PLC_PRG
VAR
   xAlarm:      BOOL;
   fbAlarmTrig: R_TRIG;
END_VAR

// Область кода

fbAlarmTrig(CLK := xAlarm);

IF fbAlarmTrig.Q THEN

   // код отправки сообщения о тревоге

END_IF
```

Счетчики импульсов

Библиотека **Standard** включает три блока счетчиков импульсов:

- **CTU** – инкрементный счетчик (его значение увеличивается на 1 при каждом импульсе);
- **CTD** – декрементный счетчик (его значение уменьшается на 1 при каждом импульсе);
- **CTUD** – счетчик, который может работать как в режиме инкремента, так и в режиме декремента.

Рассмотрим принцип работы счетчиков на примере блока **CTU**.

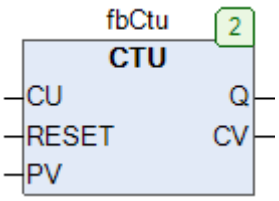


Рисунок 5.10.8 – Входы и выходы блока CTU

CU – это счетный вход, к которому привязывается переменная типа **BOOL**. По переднему фронту этой переменной выход **CV** (типа **WORD**), представляющий собой значение счетчика, увеличивается на единицу. Вход **PV** позволяет задать пороговое значение: когда значение на выходе **CV** сравняется со значением входа **PV**, то выход **Q** (сигнал достижения порога) примет значение **TRUE**. Для обнуления счетчика нужно передать значение **TRUE** на вход **RESET**.

```
PROGRAM PLC_PRG
VAR
   xDi:           BOOL;
   wCounter:      WORD;

   fbCtu:         CTU;
END_VAR

// Область кода

fbCtu
(
   CU := xDi,
   CV => wCounter
);
```

Верхний предел типа WORD – **65535**. После его достижения счет прекратится; для возобновления счета потребуется обнулить счетчик. Если для подсчитываемых вами сигналов требуется больший диапазон – то используйте блок **LCTU** из библиотеки **Standard64**. Его верхний предел составляет $2^{64} - 1$.

Таймеры

Библиотека **Standard** включает три блока таймеров:

- **TP** – генератор сигнала заданной длительности;
- **TON** – таймер задержки включения;
- **TOF** – таймер задержки отключения.

Наиболее часто используемым таймером является блок **TON**.

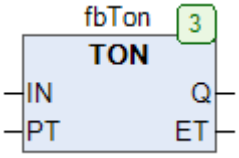


Рисунок 5.10.9 – Входы и выходы блока TON

Вход **PT** типа **TIME** представляет собой уставку таймера. Таймер запускается в работу по переднему фронту на входе **IN**. В процессе работы таймера вход **IN** должен сохранять значение **TRUE** – иначе работа блока будет прекращена, и при следующем запуске отсчет начнется заново. В процессе работы блока выход **ET** отображает время, прошедшее с момента запуска таймера. Когда значение выхода **ET** станет равным значению входа **PT** – то выход **Q** (сигнал срабатывания таймера) примет значение **TRUE.**

Ниже приведен пример использования таймера для организации задержки обработки нажатия на кнопку для игнорирования дребезга контактов.

```
PROGRAM PLC_PRG
VAR
   xButton:      BOOL;
   fbDelay:      TON;
END_VAR

// Область кода

fbDelay
(
   IN := xButton,
   PT := T#200ms
);

IF fbDelay.Q THEN

   // код обработки нажатия на кнопку

END_IF
```

Еще один пример использования таймера **TON** – генерация единичных импульсов. Для этого нужно завести на вход таймера инвертированное значение его выхода:

```
PROGRAM PLC_PRG
VAR
   fbPoll:       TON;
END_VAR

// Область кода

fbPoll
(
   IN := NOT(fbPoll.Q),
   PT := T#3s
);

// Каждые 3 секунды на выходе Q будет генерироваться единичный импульс
IF fbPoll.Q THEN

   // действия, которые нужно выполнять раз в 3 секунды

END_IF
```

Вызов таймеров в CODESYS является неблокирующим – то есть в процессе работы таймера весь код, размещенный после его вызова, будет продолжать выполняться каждый цикл задачи.

Функции для работы со строками

Библиотека **Standard** включает 9 функций для работы со строками:

- **CONCAT** – объединение («склеивание») строк;
- **DELETE** – удаление подстроки из строки;
- **FIND** – поиск первого вхождения подстроки в строку;
- **INSERT** – вставка подстроки в строку;
- **LEFT** – выделение заданного числа символов строки с ее начала;
- **LEN** – определение длины строки;
- **MID** – выделение заданного числа символов строки с заданной позиции;
- **REPLACE** – замена подстроки в строке;
- **RIGHT** – выделение заданного числа символов строки с ее конца.

Все функции рассчитаны на работу со строками типа **STRING(255)**. Для работы со строками типа **WSTRING** используются аналогичные функции из библиотеки **Standard64** с префиксом W (**WCONCAT**, **WDELETE** и т. д.). Для работы со строками, длина которых превышает 255 символов, используются функции библиотеки **StringUtils**.

Несколько примеров использования строковых функций:

```
PROGRAM PLC_PRG
VAR
   sVar1:         STRING := 'Hello, ';
   sVar2:         STRING := 'world';
   sConcat:       STRING;

   iLen:          INT;

   sMid:          STRING;
END_VAR

// Область кода


// sConcat получит значение 'Hello, world'
sConcat := CONCAT(sVar1, sVar2);

// iLen получит значение 12 (столько символов в строке 'Hello, world'
iLen := LEN(sConcat);

// sMid получит значение 'ello’
// (копируем 4 символа строки, начиная со 2-го)
sMid := MID(STR := sConcat, LEN := 4, POS := 2);
```

6 Пример создания проекта диспетчеризации

В прошлых разделах мы рассмотрели интерфейс среды CODESYS и основы программирования на языке ST. Пришло время испытать полученные навыки на практике и создать простой демонстрационный проект. В качестве примера мы рассмотрим проект системы диспетчеризации. Готовый проект, который будет создан в данном разделе, доступен по ссылке.

6.1 Формулировка задачи

В системе имеется 8 точек измерения неких параметров (температуры, давления и т. д.). Необходимо организовать:

- считывание значений этих параметров в контроллере;
- проверку на принадлежность этих значений заданному диапазону с генерацией тревог и включением аварийных индикаторов в случае выхода значения за границы диапазона;
- отображение в визуализации этих значений, сигналов тревог, а также истории тревог;
- архивацию значений в формате .csv;
- передачу значений и сигналов тревог в SCADA-систему;
- передачу значений в облачный сервис OwenCloud.

Для получения аналоговых сигналов мы будем использовать модуль аналогового ввода МВ210-101, а для индикации тревог – модуль дискретного вывода МУ210-401 (к его дискретным выходам можно подключить сигнальные лампы, средства звуковой сигнализации и т. д.). Протокол обмена модулей – **Modbus TCP**.

Если у вас в данный момент еще нет контроллера и модулей – то ничего страшного. Мы рассмотрим, как проверить работу приложения на виртуальном контроллере, эмулируя модули с помощью OPC-сервера.

Подразумевается, что к этому моменту вы уже прошли «быстрый старт» (п. 3) и освоили основные операции, необходимые для создания проекта (добавление компонентов, подключение к контроллеру, загрузка проектов и т. д.).

6.2 Настройка модулей Mx210

Если у вас есть модули MB210-101 и МУ210-401, которые будут использоваться в примере, то необходимо задать им сетевые настройки: IP-адрес, маску подсети и шлюз.

Для начала следует загрузить и установить утилиту OWEN Configurator, которая используется для настройки модулей.

После этого подключите модуль к ПК по интерфейсу **MicroUSB** и запустите **OWEN Configurator.** Нажмите кнопку **Добавить устройства**, выберите интерфейс, в названии которого есть фраза **Virtual COM Port**, и режим **Найти одно устройство** (с адресом 1). Нажмите кнопку **Найти**. После обнаружения модуля выделите его галочкой и нажмите кнопку **Добавить устройства** для перехода к его настройкам.

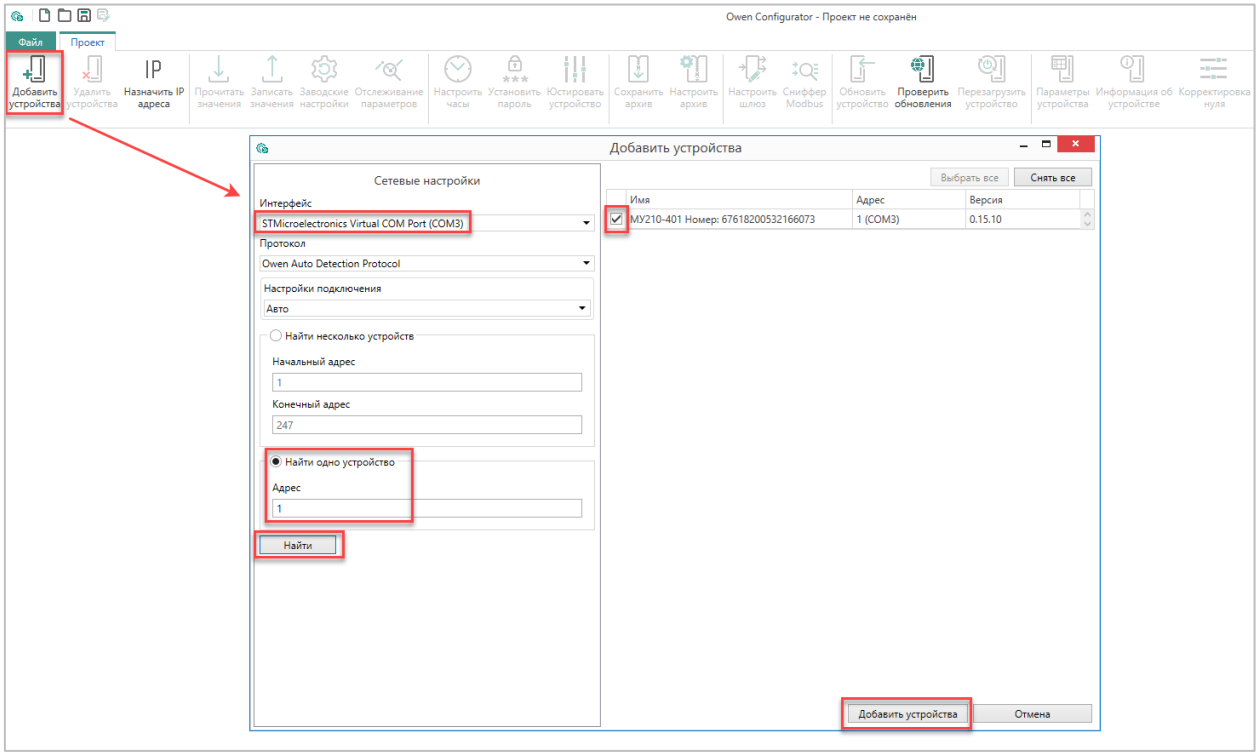


Рисунок 6.2.1 – Подключение к модулю Mx210 в утилите OWEN Configurator

В настройках модуля перейдите в папку **Сетевые настройки/Настройки Ethernet** и введите значения для установки IP-адреса, маски подсети и шлюза. Эти настройки должны соответствовать настройкам сетевого интерфейса контроллера, через который вы будете опрашивать модули. Например, контроллер имеет следующие настройки:

- IP-адрес: 10.2.11.174
- Маска подсети: 255.255.0.0
- Шлюз: 10.2.1.1

Тогда модулям можно задать такие настройки:

- IP-адрес: 10.2.11.120 (MB210-101), 10.2.11.121 (МУ210-401);
- Маска подсети: 255.255.0.0
- Шлюз: 10.2.1.1

То есть маска подсети и шлюз у контроллера и модулей должны совпадать, а IP-адреса – принадлежать одной подсети и быть уникальными в ее пределах. Если вы подключаете модули напрямую к контроллеру, а не к локальной сети, то адрес шлюза в большинстве случаев не имеет значения.

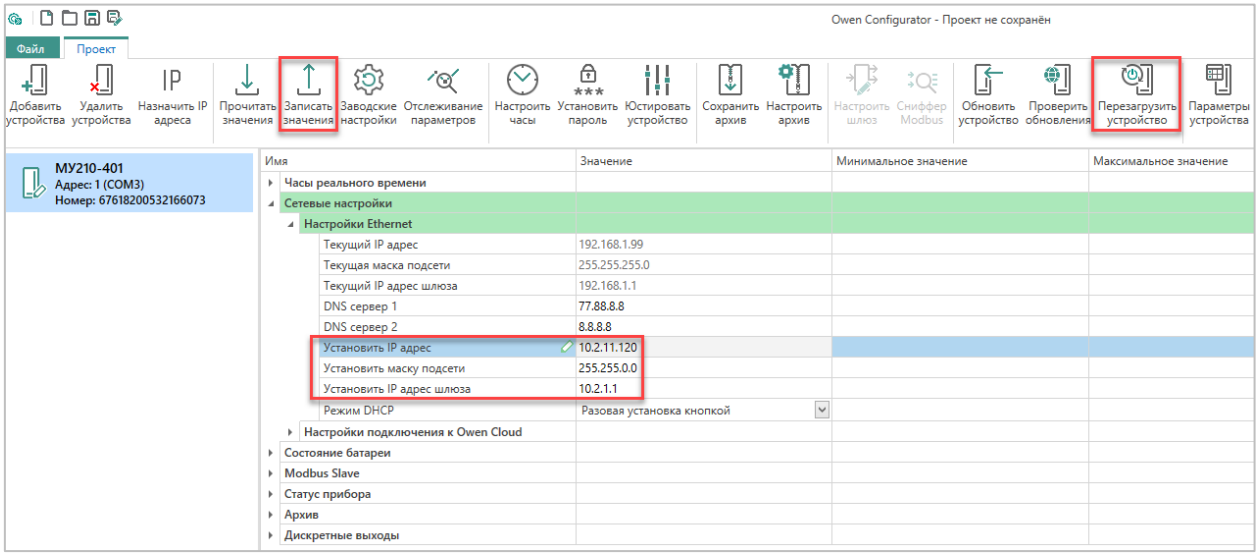


Рисунок 6.2.2 – Настройка модуля Mx210 в утилите OWEN Configurator

Все остальные параметры модулей (например, типы датчиков аналоговых каналов модуля MB210-101 и т. д.) можно сейчас не настраивать – вы сможете задать их в проекте CODESYS при настройке обмена.

После установки сетевых настроек требуется перезагрузить модуль с помощью кнопки **Перезагрузить устройство** или по питанию, чтобы новые значения настроек вступили в силу.

6.3 Эмуляция модулей Mx210 с помощью Modbus Universal MasterOPC Server

Если у вас нет реальных модулей Mx210, то их можно сэмулировать. Для этого:

1. Загрузите и установите Modbus Universal MasterOPC Server («бесплатную Trial версию OPC-сервера с ограничением по времени работы»; подойдет как обычная, так и 64-битная редакция).

2. Загрузите конфигурацию OPC-сервера с эмуляцией модуля примера.

3. Запустите OPC-сервер.

4. Нажмите кнопку **Открыть** и укажите путь к файлу конфигурации.

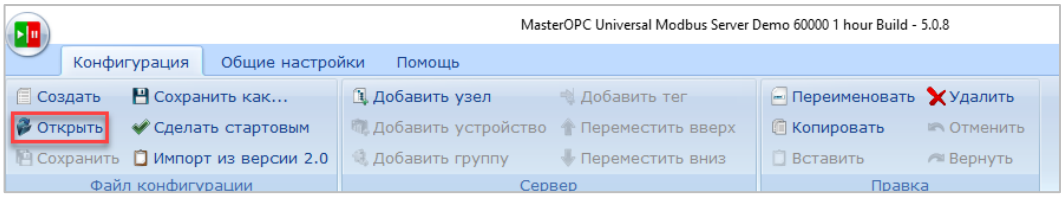


Рисунок 6.3.1 – Панель инструментов Modbus Universal MasterOPC Server

5. В дереве OPC-сервера отобразятся модули и их параметры.

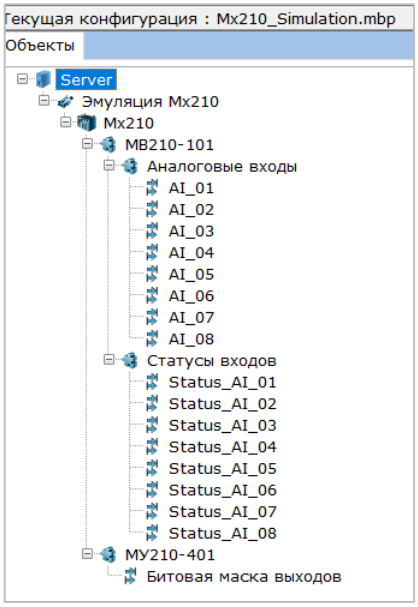


Рисунок 6.3.2 – Дерево тегов Modbus Universal MasterOPC Server

6. Для запуска OPC-сервера используйте команду **Старт**.

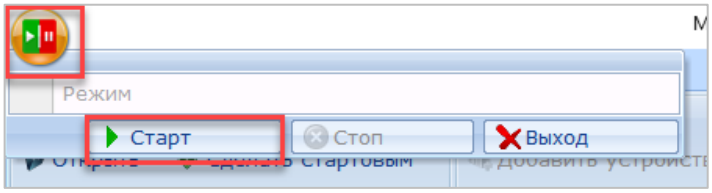


Рисунок 6.3.3 – Запуск OPC-сервера

6.4 Настройка опроса модулей Mx210 с помощью шаблонов

6.4.1 Создание нового проекта и подключение к контроллеру

В прошлых пунктах мы разобрались, как настроить модули Mx210 (если они у вас есть) или как сэмулировать их с помощью **Modbus Universal MasterOPC Server** (если модулей у вас нет).

Теперь пора приступать к созданию проекта. Создайте новый проект CODESYS с названием **MonitoringSystem** на базе шаблона. В рамках примера будет использоваться сенсорный панельный контроллер СПК. Если у вас в наличии другой контроллер – то выберите его шаблон.

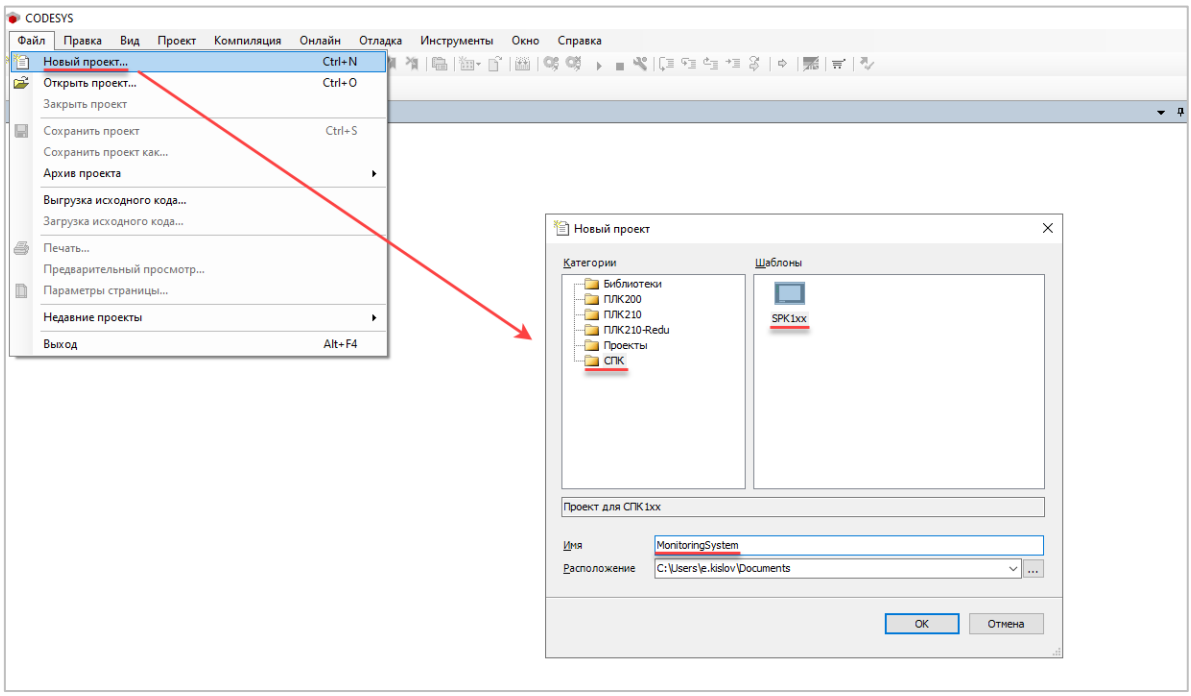


Рисунок 6.4.1 – Создание нового проекта

Удалите из дерева проекта компоненты **Modbus COM** – в рамках примера мы будем опрашивать модули через интерфейс Ethernet, так что эти компоненты нам не потребуются.

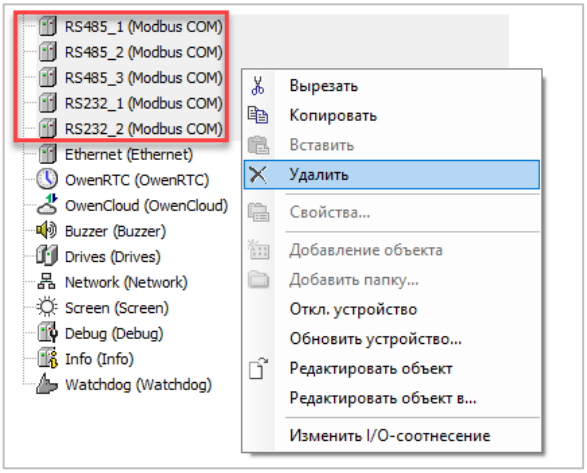


Рисунок 6.4.2 – Удаление компонентов Modbus COM

Перейдите в узел **Device** на вкладку **Установки соединения**. Выполните команду сканирования сети и выберите в списке ваш контроллер.

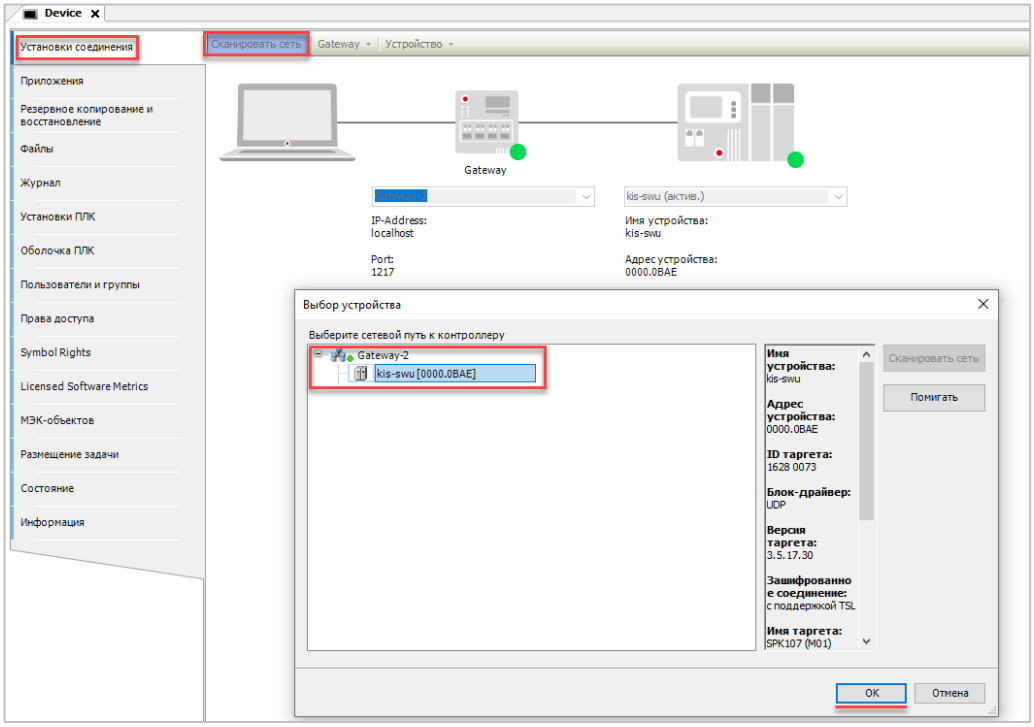


Рисунок 6.4.3 – Поиск контроллера с помощью сканирования сети

В ряде случаев (например, на ПК запущен антивирус, который блокирует рассылку широковещательных UDP пакетов или получение пакетов по портам **1740-1743**) контроллер может не определиться во время сканирования сети. Тогда следует ввести IP-адрес контроллера вручную и нажать **Enter** (надпись **актив.** появится автоматически после обнаружения контроллера – вводить ее не надо).

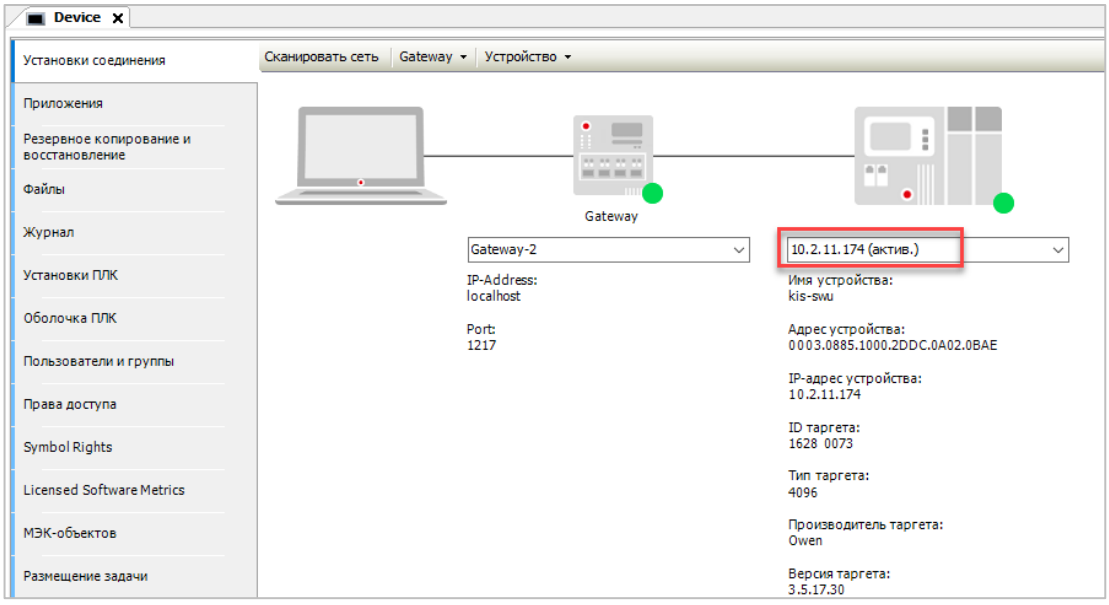


Рисунок 6.4.4 – Ввод IP-адреса контроллера для подключения без сканирования сети

Если у вас нет в наличии реального контроллера, то вы можете использовать виртуальный контроллер. Для этого нажмите на узел **Device** правой кнопкой мыши, используйте команду **Обновить устройство** и выберите таргет-файл **CODESYS Control Win V3**. Версия таргет-файла должна соответствовать используемой вами версии CODESYS (например, для версии среды **V3.5 SP17 Patch 3** используйте версию таргет-файла **3.5.17.30**).

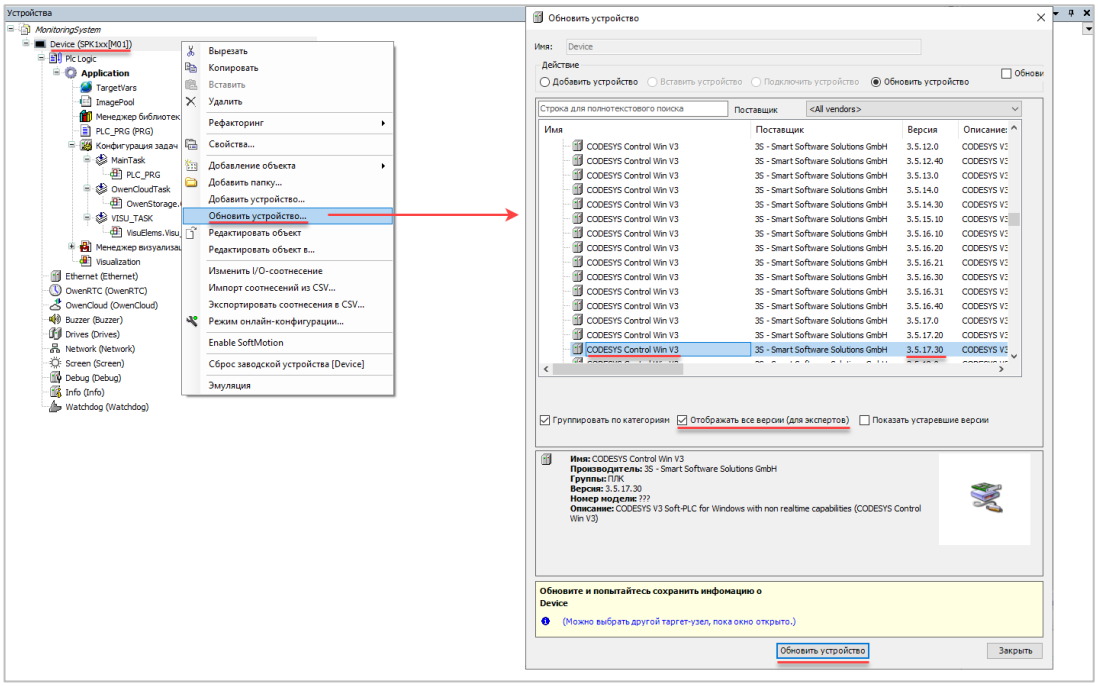


Рисунок 6.4.5 – Выбор таргет-файла виртуального контроллера CODESYS Control Win V3

После этого удалите задачу **OwenCloudTask** – потому что подключить виртуальный контроллер к облачному сервису OwenCloud не получится. Наличие этой задачи в проекте виртуального контроллера приведет к ошибкам компиляции.

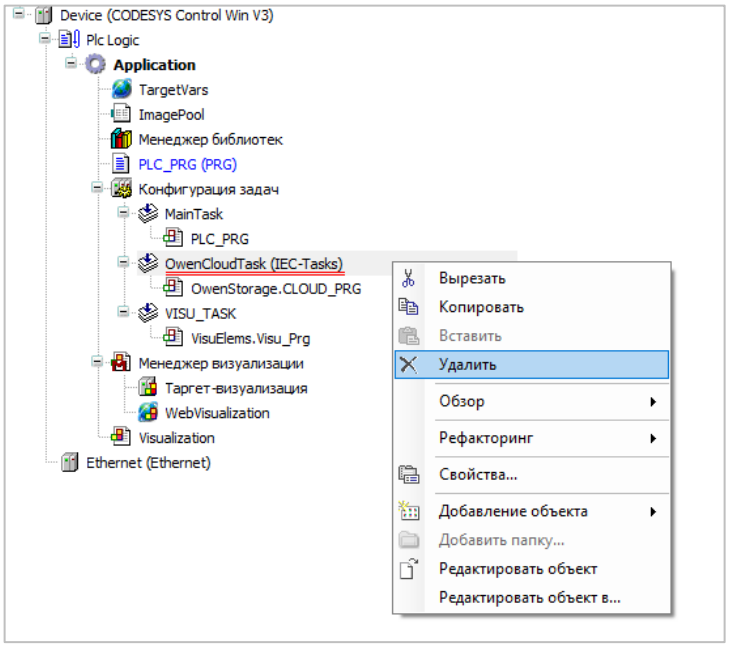


Рисунок 6.4.6 – Удаление задачи OwenCloudTask из проекта виртуального контроллера

Для запуска виртуального контроллера нужно раскрыть системный трей Windows, нажать на пиктограмму виртуального контроллера правой кнопкой мыши и выбрать команду **Start PLC**.

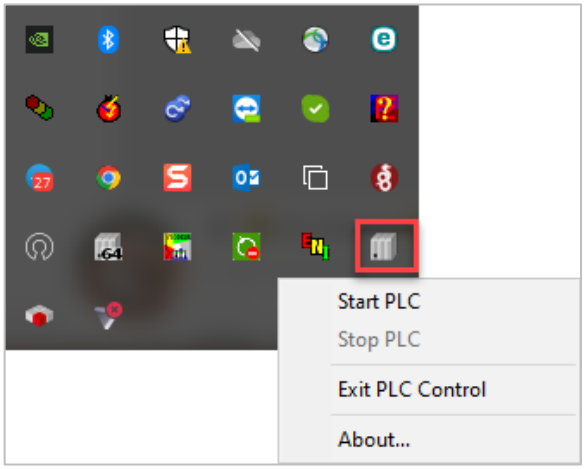


Рисунок 6.4.7 – Запуск виртуального контроллера

Установите соединение с виртуальным контроллером с помощью сканирования сети (см. рис. 6.4.3).

6.4.2 Установка пакета шаблонов Mx210

После установки соединения можно переходить к настройке обмена. Мы будем настраивать обмен с модулями с помощью готовых шаблонов – это значительно упрощает процедуру настройки. Для начала требуется перейти на сайт ОВЕН в раздел CODESYS V3/Библиотеки и компоненты и загрузить пакет шаблонов Mx210.

Установка пакета выполняется с помощью утилиты **CODESYS Installer** (**Инструменты – CODESYS Installer**) – мы уже рассматривали этот вопрос в п. 2.2.

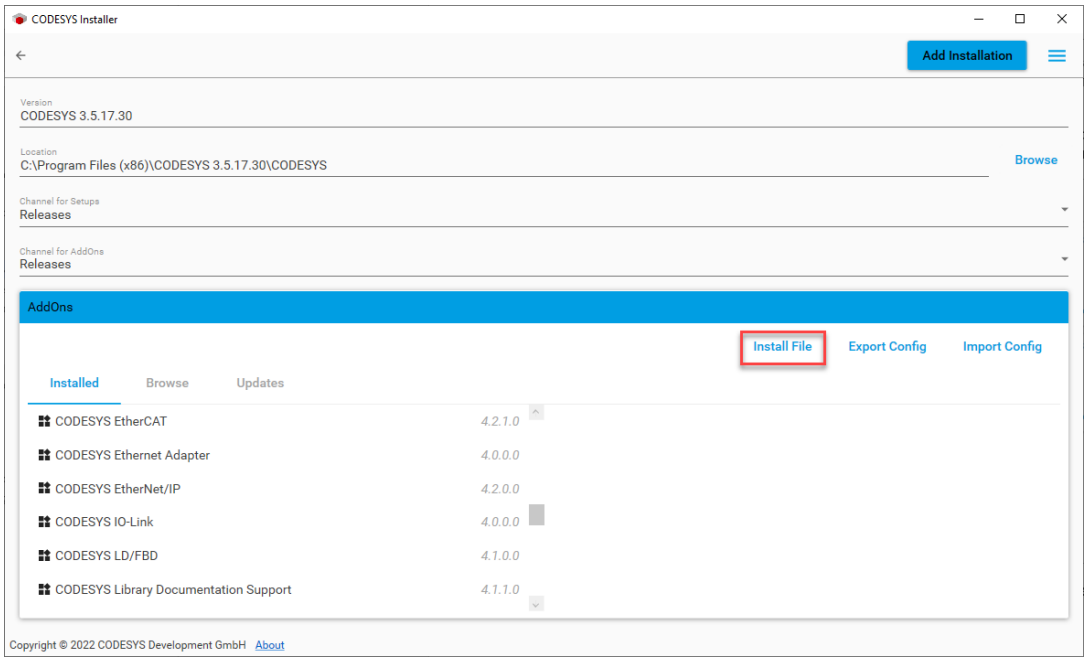


Рисунок 6.4.8 – Кнопка установки пакетов

6.4.3 Настройка компонентов Modbus

Теперь перейдем в компонент **Ethernet** и выберем сетевой интерфейс контроллера, который будет использоваться для связи с модулями Mx210. Собственно, именно ради этого чуть ранее нам потребовалось установить соединение с контроллером – чтобы на этом шаге получить список его интерфейсов.

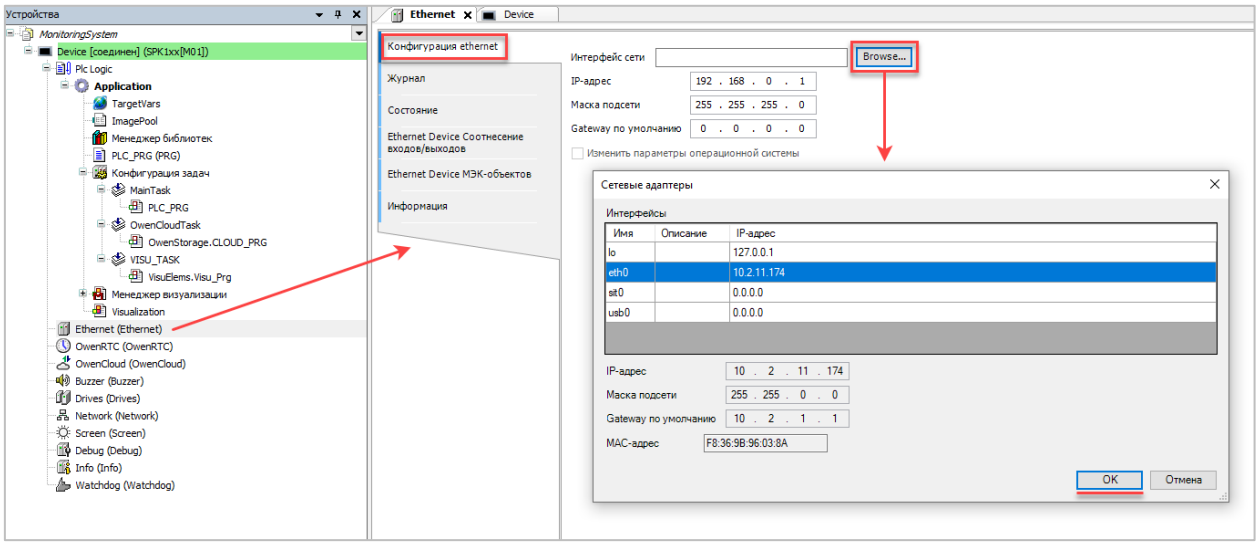


Рисунок 6.4.9 – Выбор сетевого интерфейса в компоненте Ethernet

Нажмите правой кнопкой мыши на компонент **Ethernet**, используйте команду **Добавить устройство** и добавьте в проект компонент **Modbus TCP Master**.

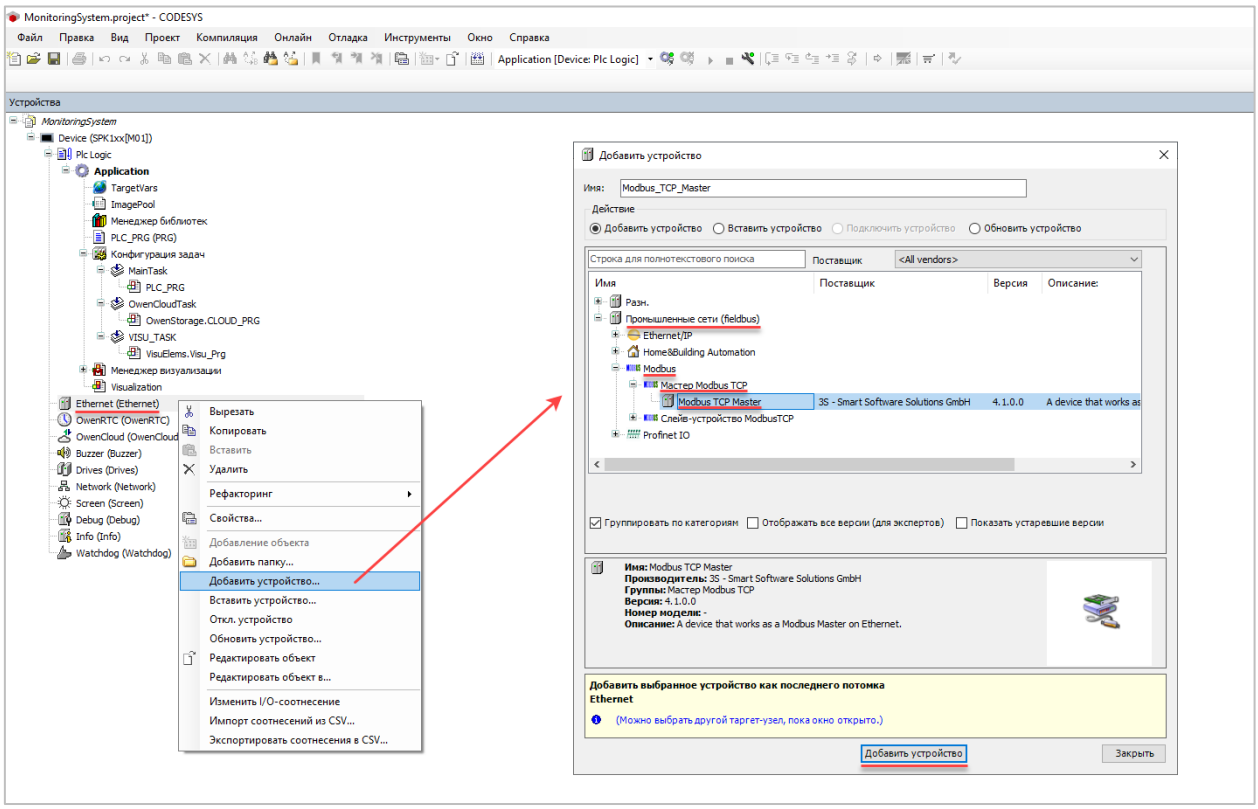


Рисунок 6.4.10 – Добавление компонента Modbus TCP Master

В настройках компонента на вкладке **Общее** установите галочку **Автоподключение**. За счет нее контроллер будет пытаться повторно установить соединение с модулями в случае ошибок обмена, а не прекратит их опрос.

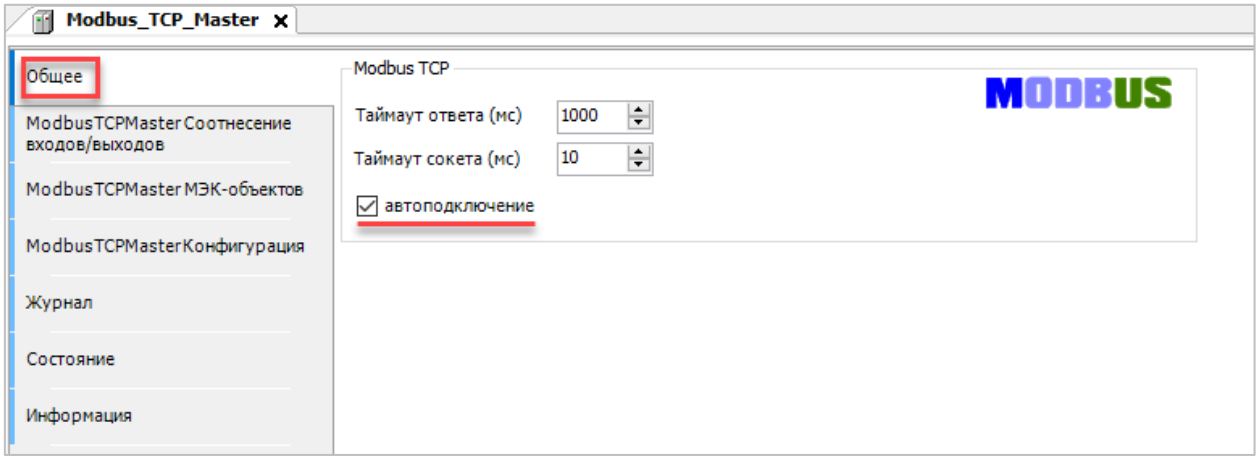


Рисунок 6.4.11 – Настройки компонента Modbus TCP Master

Нажмите правой кнопкой мыши на компонент **Modbus TCP Master**, используйте команду **Добавить устройство** и добавьте в проект шаблоны модулей МВ210-101 и МУ210-401. Если у вас не отображаются шаблоны – значит, вы еще не установили их пакет (см. рис. 6.4.8).

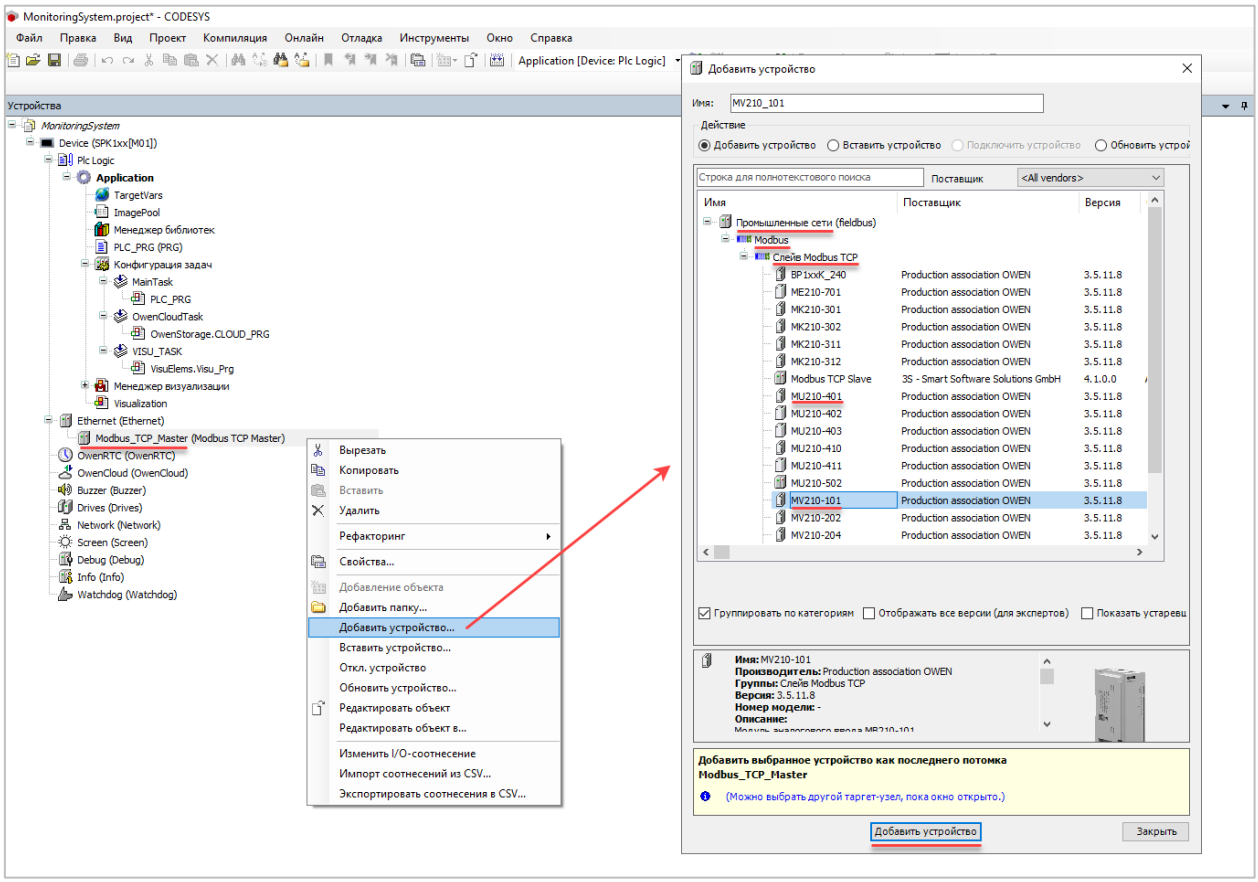


Рисунок 6.4.12 – Добавление шаблонов модулей МВ210-101 и МУ210-401

В настройках шаблонов на вкладке **Общее** необходимо указать IP-адреса модулей. Тут возможны несколько вариантов:

- если вы будете опрашивать реальные модули, то укажите IP-адреса, которые вы задали им в утилите ОВЕН Конфигуратор;
- если вы эмулируете модули и при этом используете реальный контроллер – то укажите в обоих шаблонах IP-адрес сетевого интерфейса ПК, к которому подключен контроллер. Именно этот вариант показан на рис. 6.4.13 (**10.2.8.133** – это адрес сетевого интерфейса ПК; у вас он будет своим);
- если вы эмулируете модули и при этом используете виртуальный контроллер **CODESYS Control Win V3** – то укажите в обоих шаблонах IP-адрес **127.0.0.1** (localhost).

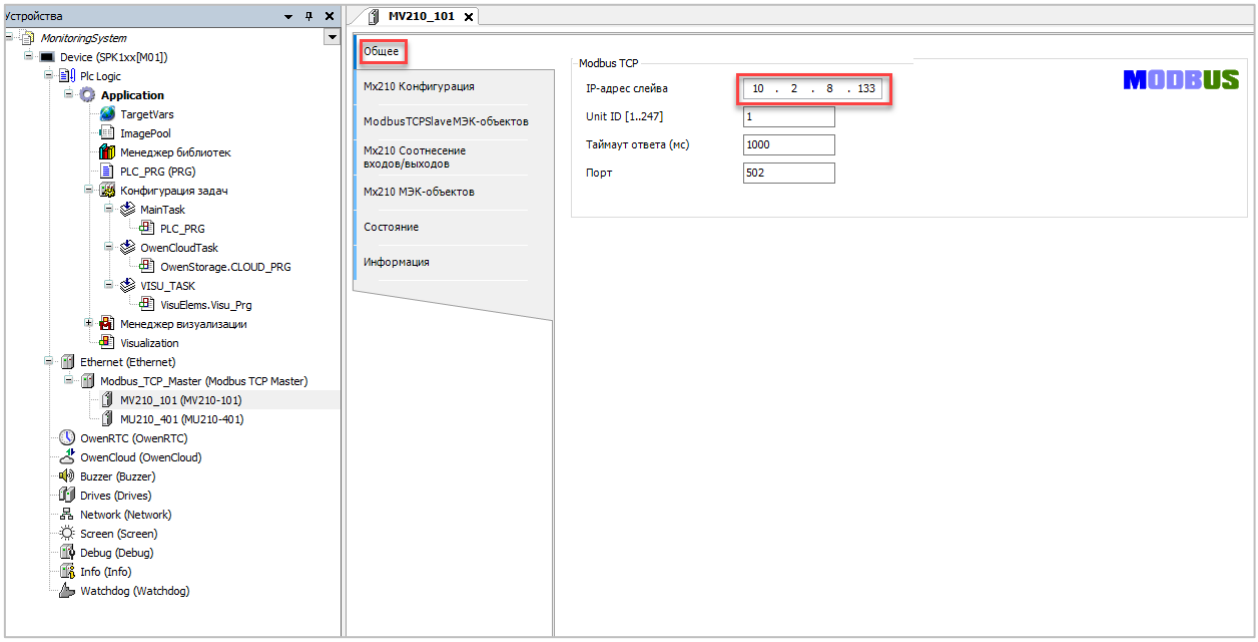


Рисунок 6.4.13 – Указание IP-адреса модуля MB210-101

Шаблон содержит две вкладки, на которых пользователь работает с параметрами модуля – **Конфигурация** и **Соотнесение входов/выходов**. На вкладке **Конфигурация** шаблона MB210-101 задаются настройки аналоговых входов модуля – тип датчика, полоса фильтра и т. д. При загрузке проекта заданные здесь настройки будут записаны в модуль – за счет того, что параметр **Использовать настройки конфигурации** по умолчанию имеет значение **TRUE**. Если вы уже задали все эти настройки через утилиту OWEN Configurator, то задайте параметру **Использовать настройки конфигурации** значение **FALSE**, чтобы ваши настройки не были перезаписаны настройками этой вкладки.

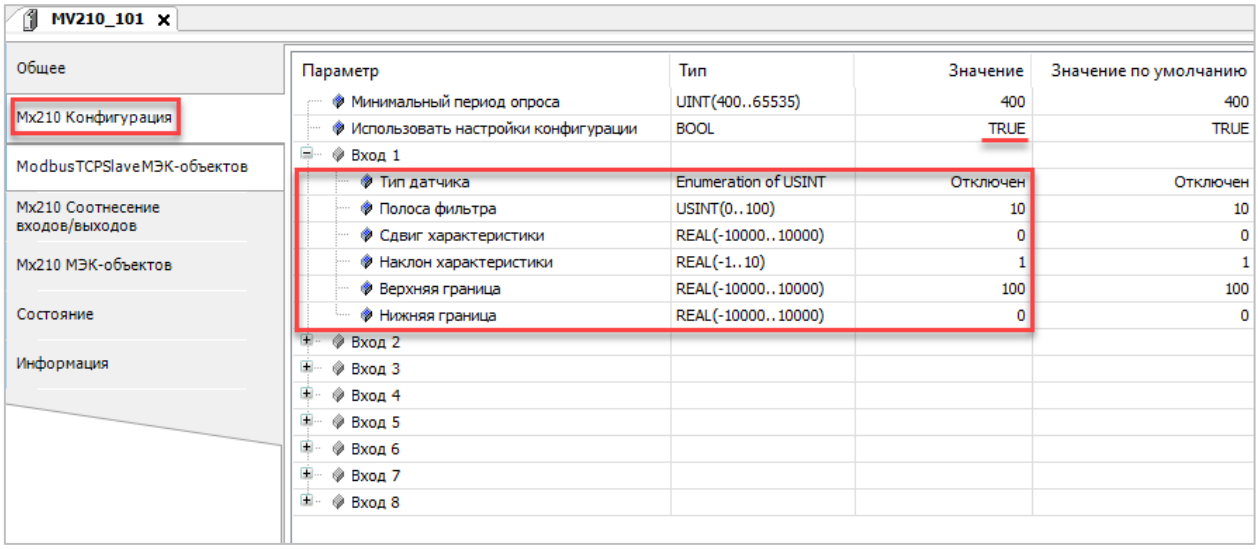


Рисунок 6.4.14 – Параметры вкладки «Конфигурация» шаблона модуля MB210-101

На вкладке **Соотнесение входов-выходов** расположены каналы оперативных параметров модуля – значения аналоговых входов, их статусы и т. д. К этим каналам привязываются переменные, которые используются в коде проекта, визуализации и т. д. В нашем проекте переменные еще не объявлены, так что привязка будет произведена в следующих пунктах.

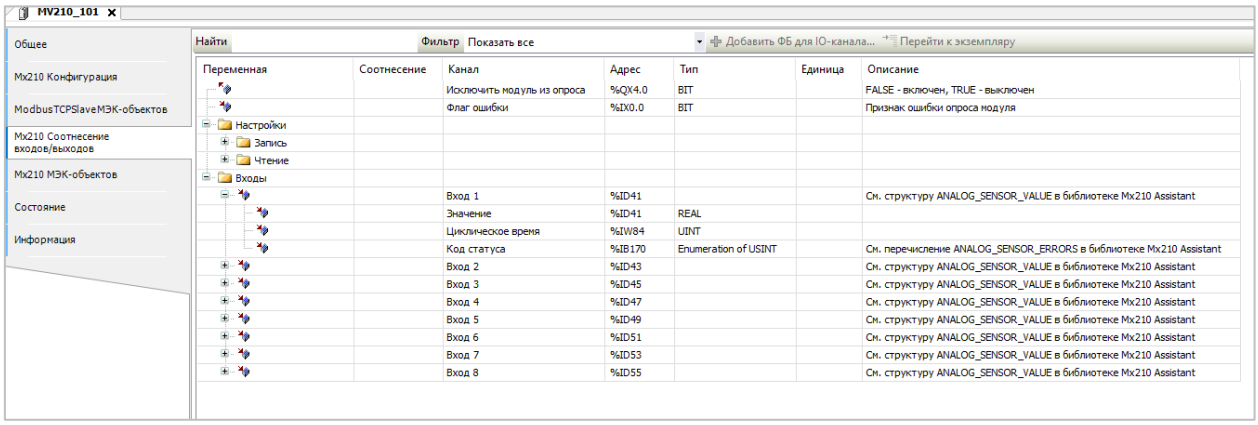


Рисунок 6.4.15 – Вкладка соотнесения входов-выходов шаблона модуля MB210-101

Но уже на данном этапе мы можем проверить обмен с модулями. Если вы эмулируете модули с помощью Modbus Universal MasterOPC Server – то не забудьте сначала его запустить (см. рис. 6.3.2).

6.4.4 Проверка обмена

Загрузите ваше приложение в контроллер и запустите его (используйте для этого ярлыки команд **Логин** и **Старт** на панели инструментов).

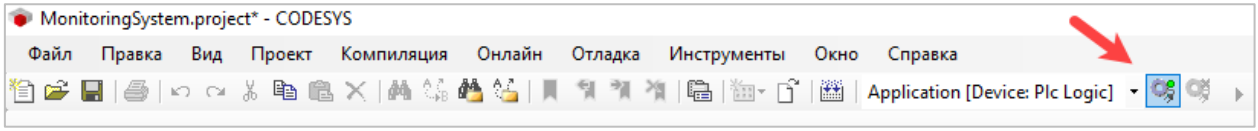


Рисунок 6.4.16 – Загрузка приложения в контроллер

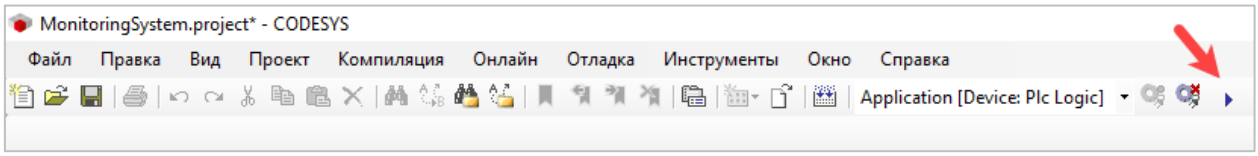


Рисунок 6.4.17 – Запуск приложения

Если обмен настроен без ошибок и физическое подключение контроллера/модулей/ПК (в зависимости от используемого оборудования) произведено корректно, то пиктограммы компонентов Modbus в дереве проекта загорятся зеленым, а в OPC-сервере будет отображаться лог запросов и ответов.

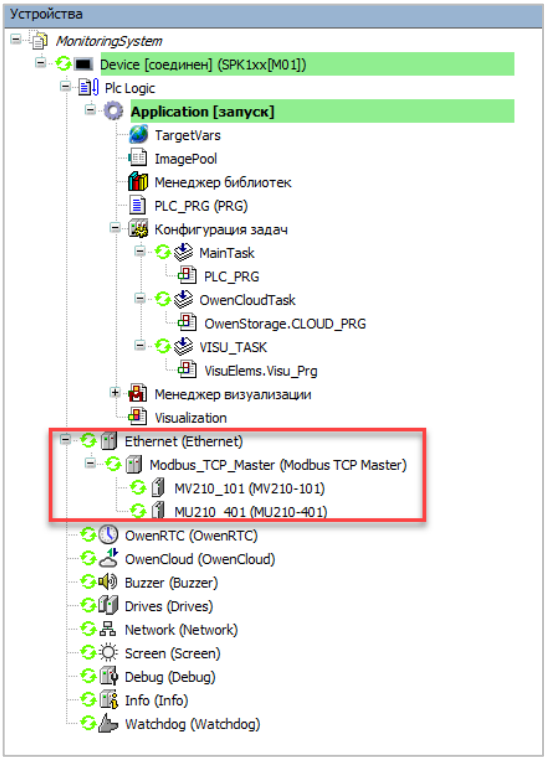


Рисунок 6.4.18 – Индикаторы диагностики успешного обмена

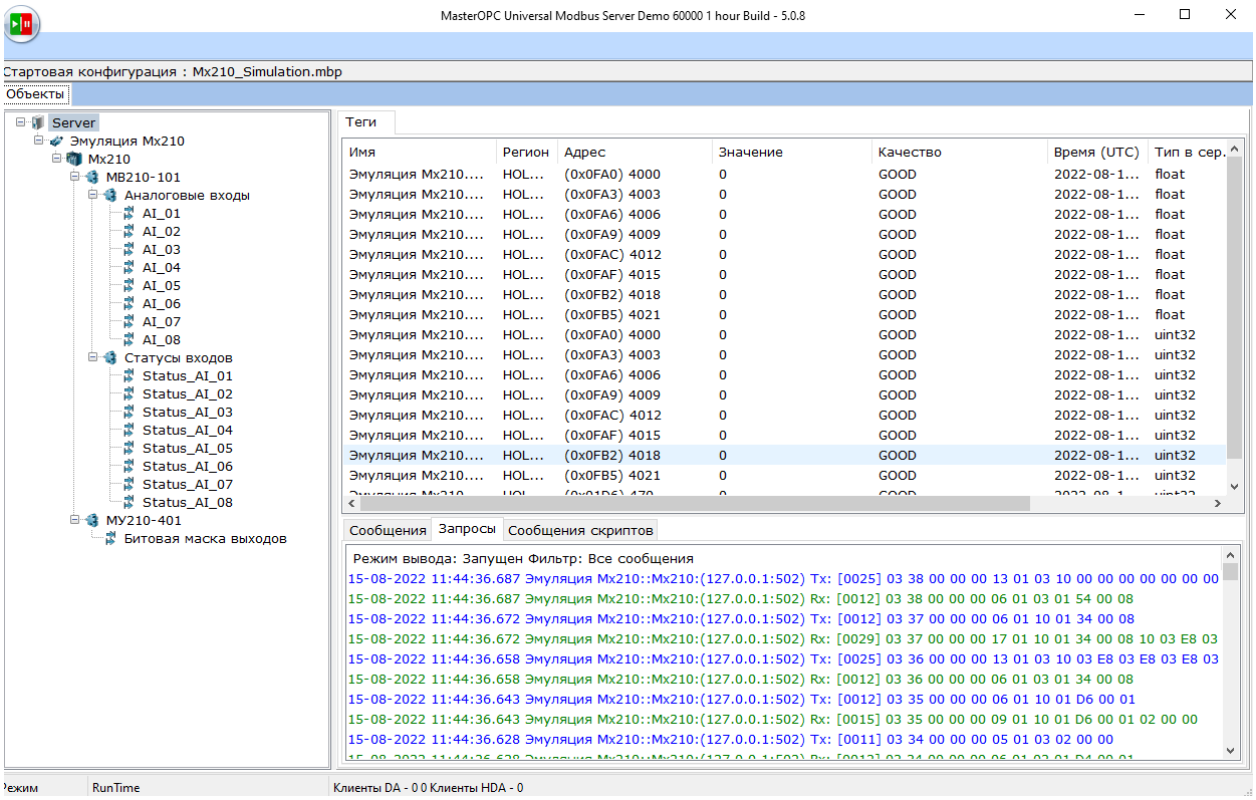


Рисунок 6.4.19 – Лог обмена между контроллером и OPC-сервером

В случае использования реальных модулей – вы получите сигналы от датчиков, которые подключены ко входам модуля. В случае использования OPC-сервера – значения потребуется вводить вручную. Например, выберите параметр **AI_01**, который соответствует первому аналоговому входу, нажмите на ячейку **Значение** и введите новое значение (например, **11.22**).

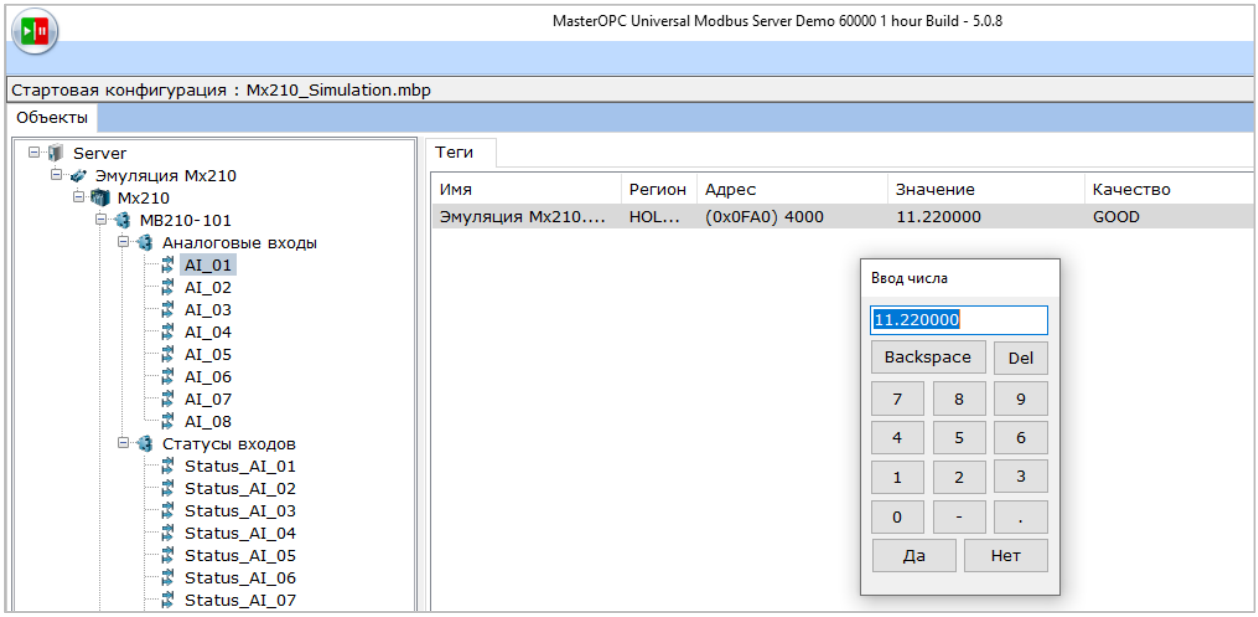


Рисунок 6.4.20 – Ввод значения параметра в OPC-сервере

Теперь в CODESYS перейдите на вкладку **Соотнесение входов-выходов** шаблона модуля MB210-101 – и вы увидите там введенное вами число, которое контроллер считал из OPC-сервера.

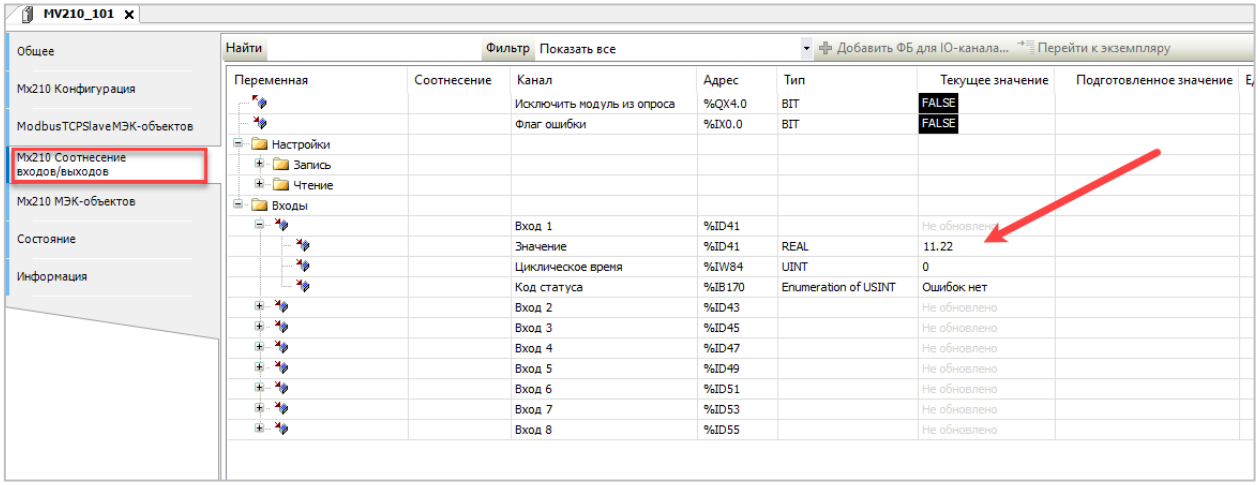


Рисунок 6.4.21 – Отображение считанного из OPC-сервера значения

На рисунке 6.4.21 виден канал **Код статуса** – он используется для отображения ошибок датчиков (например, обрыва датчика, отсутствия связи с АЦП и т. д.). Код ошибки передается в старшем байте измеренного значения. Список кодов ошибок приведен в руководстве эксплуатации на модуль:

Ошибки датчиков

В случае возникновения ошибки на входе во время обмена по протоколу ModBus код ошибки передается в регистр статуса входа и в крайний байт регистра значения (тип float).

Коды ошибок

Характер ошибки	Значение в регистре значения	Индикация
Измерение успешно	Передается результат измерения	Зеленый
Значение заведомо неверно	0xF0	Оранжевый
Данные не готовы. Необходимо дождаться результатов первого измерения после включения модуля	0xF6	Оранжевый
Датчик отключен	0xF7	Выключен
Велика температура свободных концов ТП	0XF8	Оранжевый
Мала температура свободных концов ТП	0XF9	Оранжевый
Измеренное значение слишком велико	0xFA	Оранжевый
Измеренное значение слишком мало	0xFB	Оранжевый
Короткое замыкание датчика	0xFC	Красный
Обрыв датчика	0xFD	Красный
Отсутствие связи с АЦП	0xFE	Красный
Некорректный калибровочный коэффициент	0xFF	Оранжевый

Рисунок 6.4.22 – Список кодов ошибок модуля MB210-101

В случае использования реальных модулей вы можете сымитировать ошибку вручную – например, отключив датчик. Для имитации ошибки в OPC-сервере нужно сделать следующее:

1. Выбрать параметр, начинающийся с префикса **Status** (например, **Status_AI_01** – код статуса первого аналогового входа).

2. Найти в таблице (см. рис. 6.4.22) код ошибки, который вы собираетесь сымитировать (например, **0xFD** – обрыв датчика).

3. Записать значение регистра измерения с этим кодом в шестнадцатеричной системе счисления (HEX): старший байт будет равен коду ошибки, а три младших могут иметь произвольные значения (например, пусть они будут нулями).



Рисунок 6.4.23 – Формирование измеренного значения с кодом ошибки

4. Перевести значение в десятичную систему счисления.

5. Записать полученное число в параметр статуса соответствующего входа в OPC-сервере.

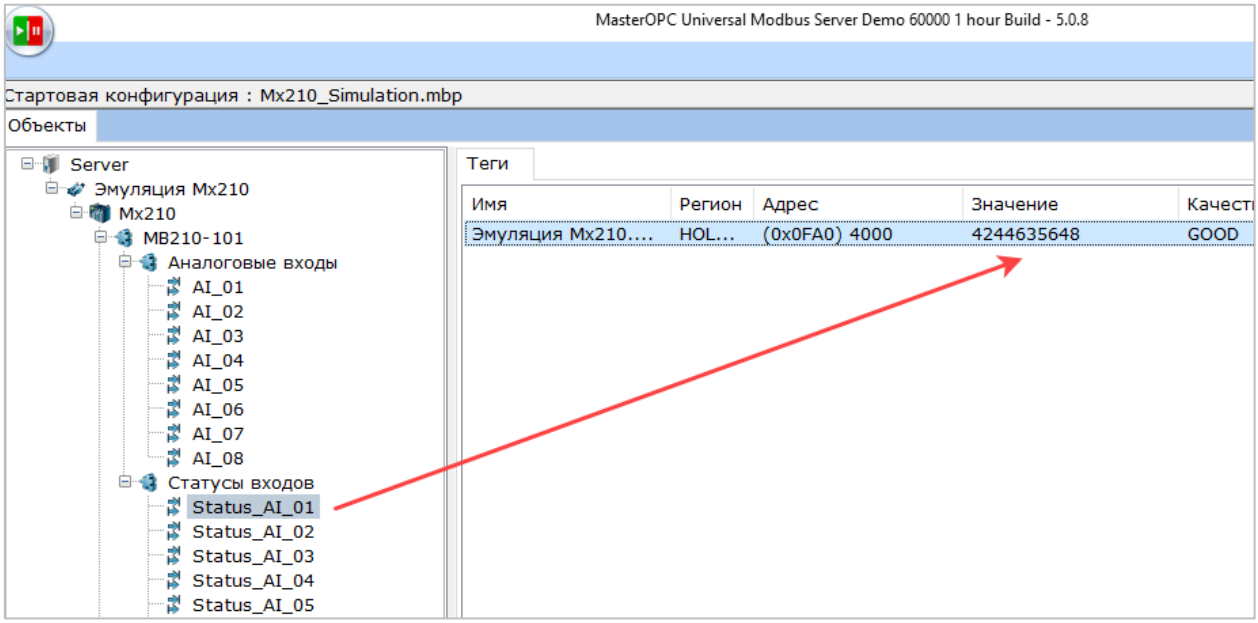


Рисунок 6.4.24 – Запись значения с кодом ошибки в OPC-сервере

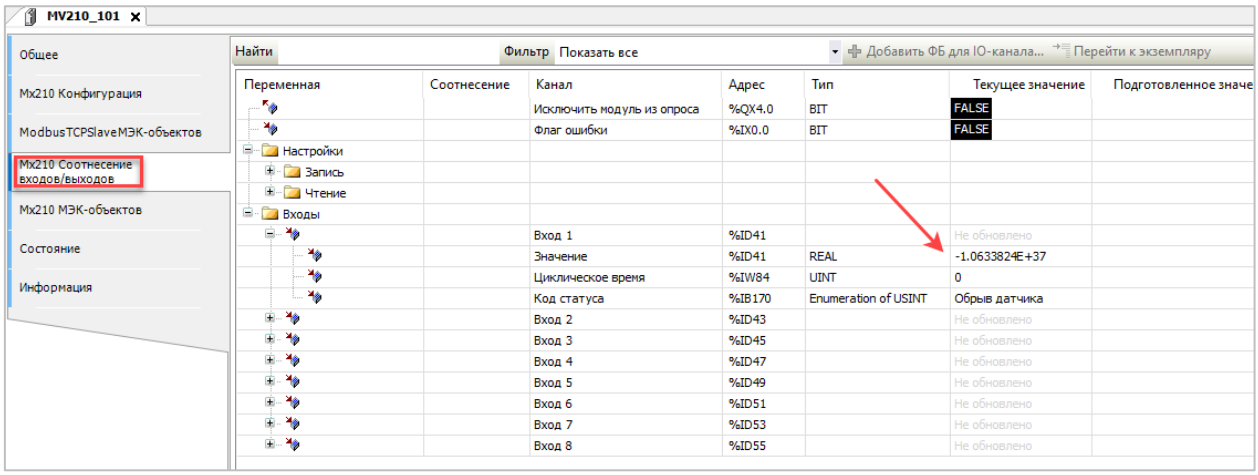


Рисунок 6.4.25 – Отображение кода ошибки в CODESYS

Теперь рассмотрим работу с модулем МУ210-401 в том случае, когда он эмулируется с помощью OPC-сервера. Перейдите на вкладку **Соотнесение входов-выходов** этого шаблона и найдите канал **Битовая маска выходов (запись).** В столбце **Подготовленное значение** присвойте некоторым выходам (например, выходам 1 и 8) значение **TRUE**.

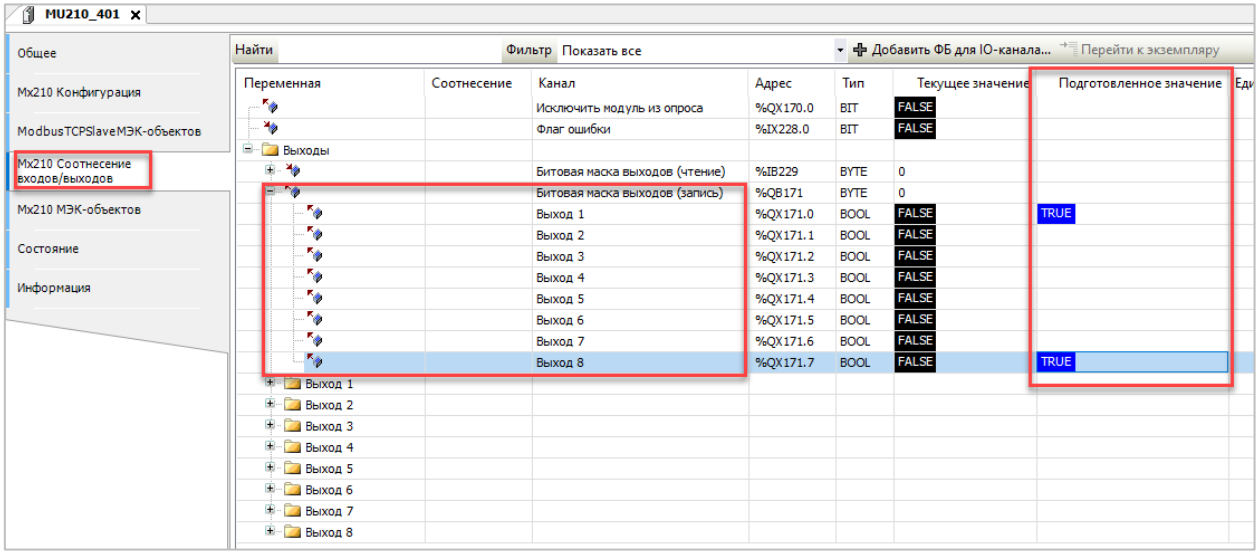


Рисунок 6.4.26 – Подготовка значений дискретных выходов модуля МУ210-401

Затем используйте команду **Отладка – Записать значения** для записи подготовленных значений в каналы шаблона.

В корневом канале битовой маски выходов отобразится число **129**. Это же число можно будет увидеть в OPC-сервере:

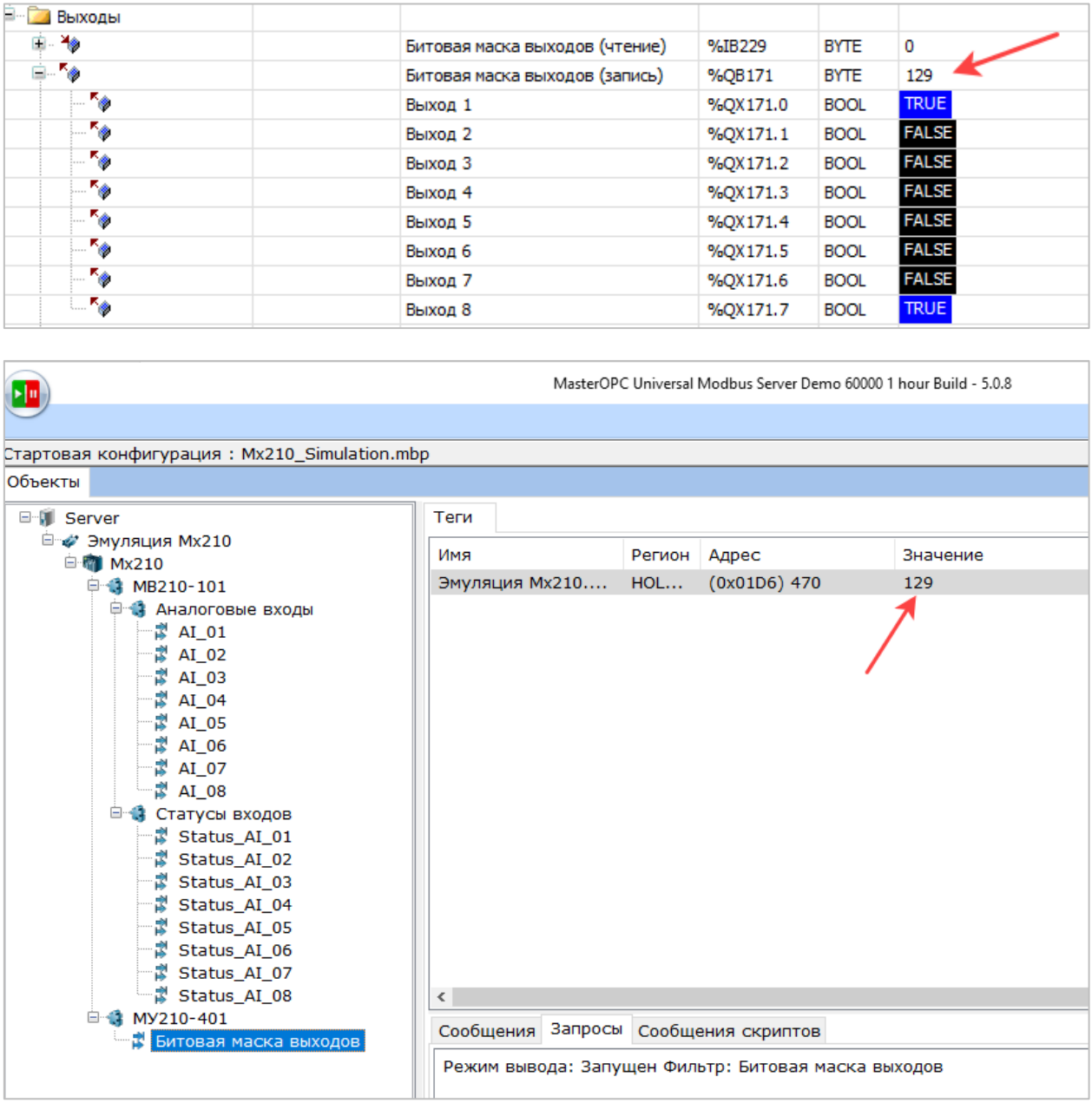


Рисунок 6.4.27 – Отображение маски дискретных выходов модуля МУ210-401 в CODESYS и OPC-сервере

Чтобы понять, как сформировалось это число, нужно представить значение битовой маски в двоичном виде. Мы записали **TRUE** в 0 и 7 биты нашего числа (нумерация бит ведется с 0, так что выходу 1 модуля соответствует бит 0 битовой маски, выходу 2 – бит 1 и т. д.). Если теперь перевести это значение из двоичной системы счисления в десятичную, то результатом как раз будет число 129.

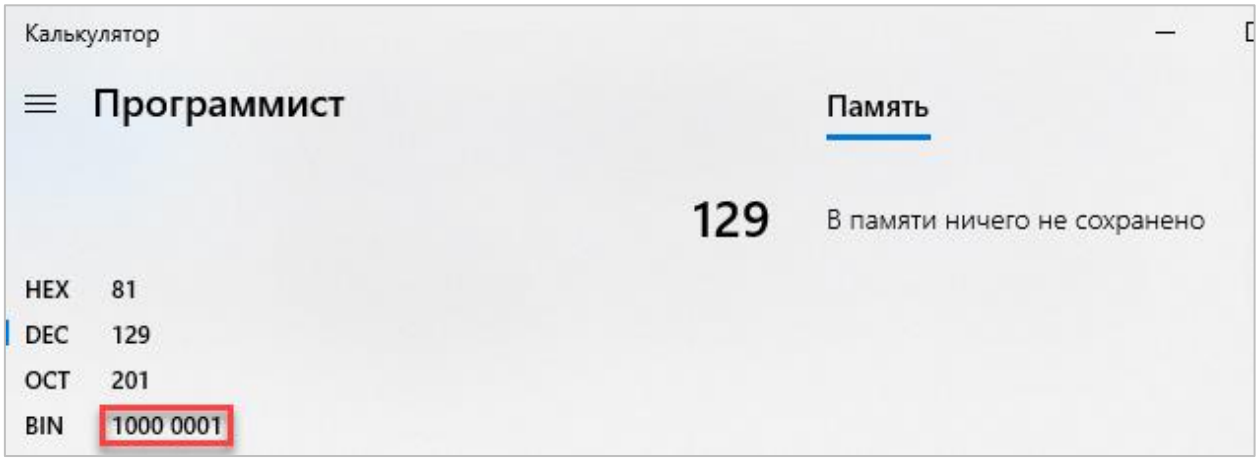


Рисунок 6.4.28 – Расшифровка битовой маски

На данный момент мы установили связь с модулями и научились эмулировать значения их параметров с помощью OPC-сервера. Теперь необходимо написать программный код, который будет обрабатывать значения аналоговых входов модуля МВ210-101 и формировать маску дискретных выходов модуля МУ210-401. Этим мы и займемся в следующем пункте.

6.5 Разработка программы

6.5.1 Создание структур

Перед тем как приступить непосредственно к написанию кода – следует объявить переменные, которые этот код будет использовать. В нашем проекте основными объектами являются точки измерения. Давайте перечислим параметры, которые требуются для описания одной точки:

- название точки измерения, отображаемое в визуализации (например, «Температура ГВС»).
- измеренное значение;
- код ошибки измерения;
- текст ошибки измерения для отображения в визуализации;
- нижний и верхний предел допустимого диапазона значения;
- сигнал тревоги о выходе значения за допустимый диапазон;
- счетчик тревог.

Хорошим решением будет создание структуры, в состав которой войдут переменные, соответствующие всем перечисленным выше параметрам. Но нужно сразу отметить, что ряд параметров должны быть энергонезависимыми – это значения пределов (в нашем примере они будут задаваться оператором в визуализации, так что сделать их константами нельзя) и счетчик тревог. Поэтому разумно создать две структуры – одна будет включать в себя энергозависимые параметры (назовем ее **MEASURE_POINT**), а другая – энергонезависимые (**MEASURE_POINT_MEM**).

Для создания структуры нажмите правой кнопкой мыши на узел **Application** и выберите команду **Добавление объекта – DUT – Структура**.

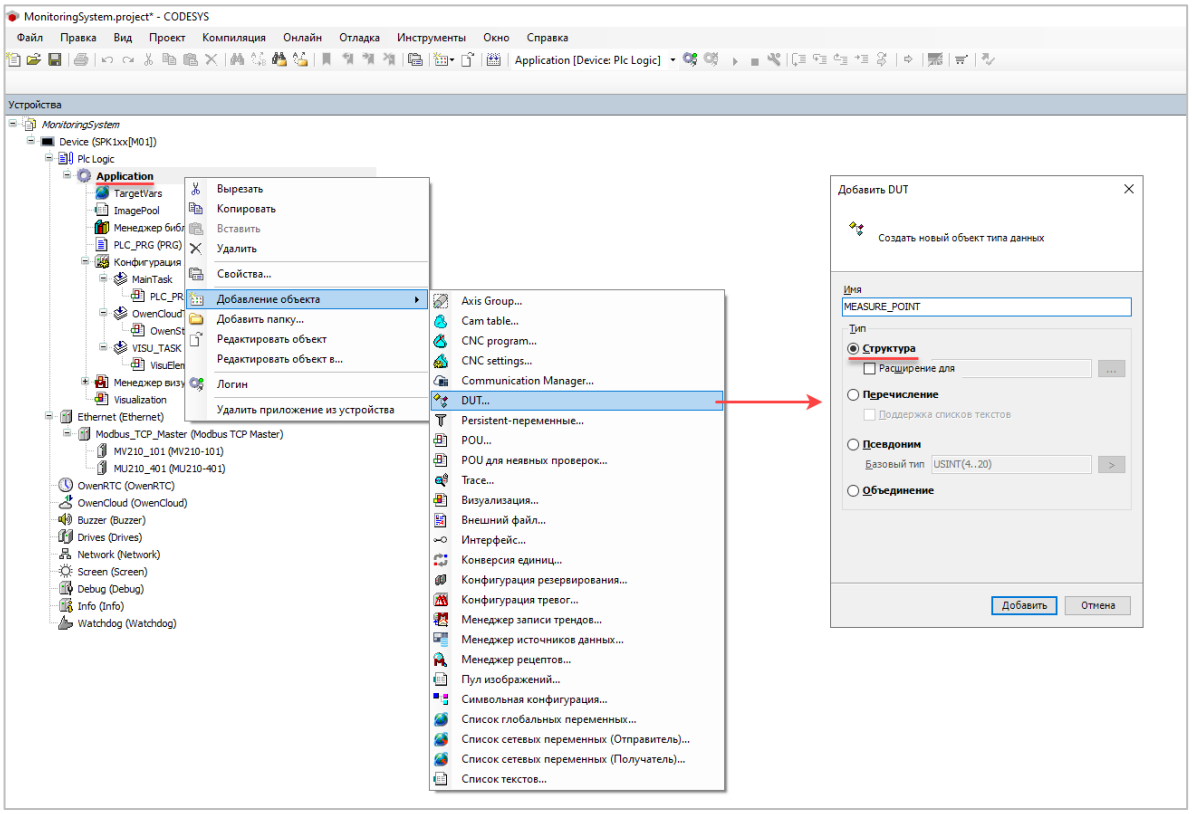


Рисунок 6.5.1 – Создание структуры

```
// Структура оперативных параметров одной точки измерения системы автоматизации
TYPE MEASURE_POINT :
STRUCT
   // Название точки измерения
   wsPointName:  WSTRING(20);
   // Текущее измеренное значение
   rValue:       REAL;
   // Код ошибки аналогового входа
   eStatus:      Mx210Assistant.ANALOG_SENSOR_ERRORS;
   // Текстовое описание ошибки
   wsStatus:     WSTRING;
   // Сигнал выхода значения за допустимый диапазон
   xAlarm:       BOOL;
END_STRUCT
END_TYPE
```

```
// Структура энергонезависимых параметров одной точки измерения системы автоматизации
TYPE MEASURE_POINT_MEM :
STRUCT
   // Нижний допустимый предел значения
   rLowAlarmValue:  REAL := 10.0;
   // Верхний допустимый предел значения
   rHighAlarmValue: REAL := 30.0;
   // Экземпляр счетчика ошибок
   fbAlarmCounter:  CTU;
END_STRUCT
END_TYPE
```

В каждой из структур нужно обсудить по одному моменту.

В структуре **MEASURE_POINT** для кода ошибки аналогового входа мы используем перечисление **ANALOG_SENSOR_ERRORS** из библиотеки **Mx210Assistant**. Эта библиотека была автоматически добавлена в менеджер библиотек при добавлении шаблона модуля MB210-101.

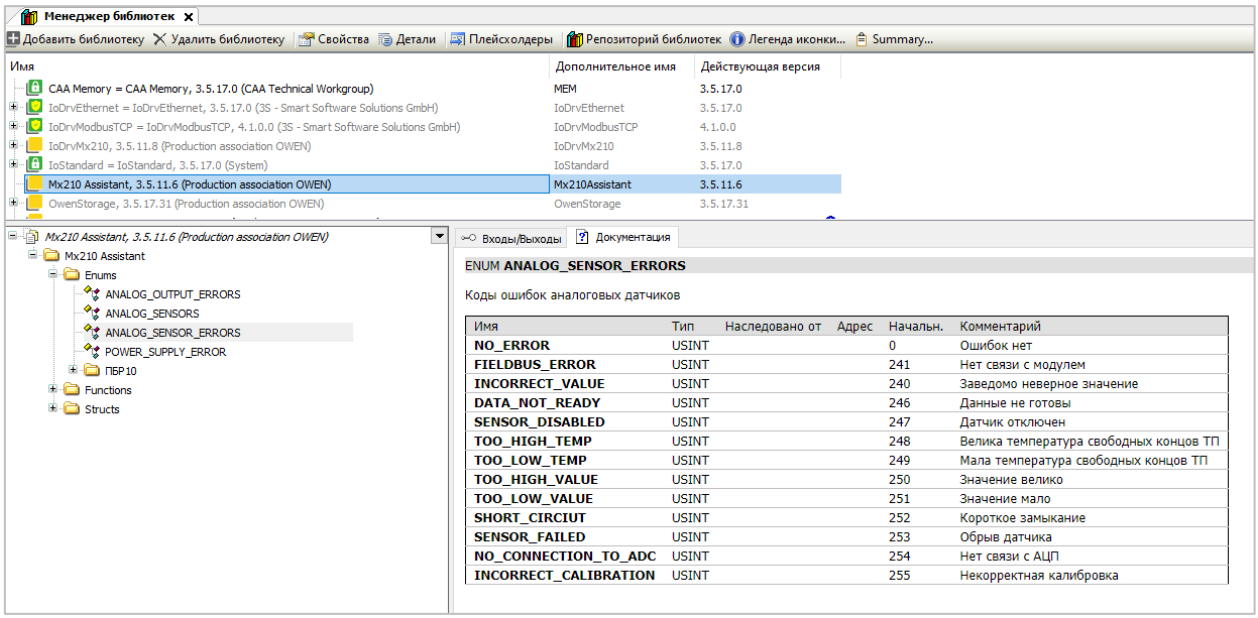


Рисунок 6.5.2 – Список элементов перечисления ANALOG_SENSOR_ERRORS библиотеки Mx210Assistant

В структуре энергонезависимых переменных **MEASURE_POINT_MEM** мы объявили экземпляр функционального блока **CTU** из библиотеки Standard. Вообще, сохранять в энергонезависимой памяти экземпляры функциональных блоков обычно довольно накладно, поскольку они могут содержать в себе множество переменных; разумно определить в блоке именно те переменные, значения которых должны сохраняться после перезагрузки контроллера, объявить «копии» этих переменных в энергонезависимой памяти и написать код, который будет синхронизировать значения этих энергонезависимых «копий» и соответствующих им переменных функционального блока при старте приложения.

Но чтобы упростить наш пример – мы целиком разместим экземпляр блока в энергонезависимой памяти. Кроме того, экземпляр блока **CTU** занимает всего 16 байт памяти (это можно определить с помощью оператора XSIZEOF) – это достаточно немного, и такой подход вполне применим даже в большинстве реальных проектов.

После создания структур можно объявить их экземпляры в программе **PLC_PRG**, которая уже присутствует в нашем проекте. Число точек измерения в различных проектах может быть разным. В нашем примере мы будем считать, что их 8 (по числу входов модуля MB210-101). Поэтому мы сразу объявим массивы наших структур, использовав в качестве верхней границы константу – это позволит легко адаптировать проект в том случае, если число точек изменится. Массивы обычно обрабатываются в цикле FOR – именно так и мы будем делать в нашем примере, так что сразу объявим переменную **i**, которая будет являться счетчиком цикла.

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:         ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                        INT;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:   INT  := 8;
END_VAR
```

6.5.2 Привязка переменных к шаблонам Mx210

В нашей структуре **MEASURE_POINT**, созданной в предыдущем пункте, есть переменные **rValue** (измеренное значение) и **eStatus** (код ошибки аналогового входа) – их значения считываются с модуля MB210-101. Переменная **xAlarm** представляет собой сигнал ошибки – эти сигналы должны записываться в модуль MУ210-401 для управления его дискретными выходами (к ним, например, могут быть подключены приборы световой и звуковой сигнализации).

Эти переменные нужно привязать к каналам шаблонов. Перейдите на вкладку **Соотнесение входов-выходов** шаблона MB210-101, два раза нажмите на ячейку столбца **Переменная** нужного канала и с помощью ассистента ввода выберите соответствующую переменную структуры (для канала **Значение** – **rValue**, для канала **Код статуса** – **eStatus**):

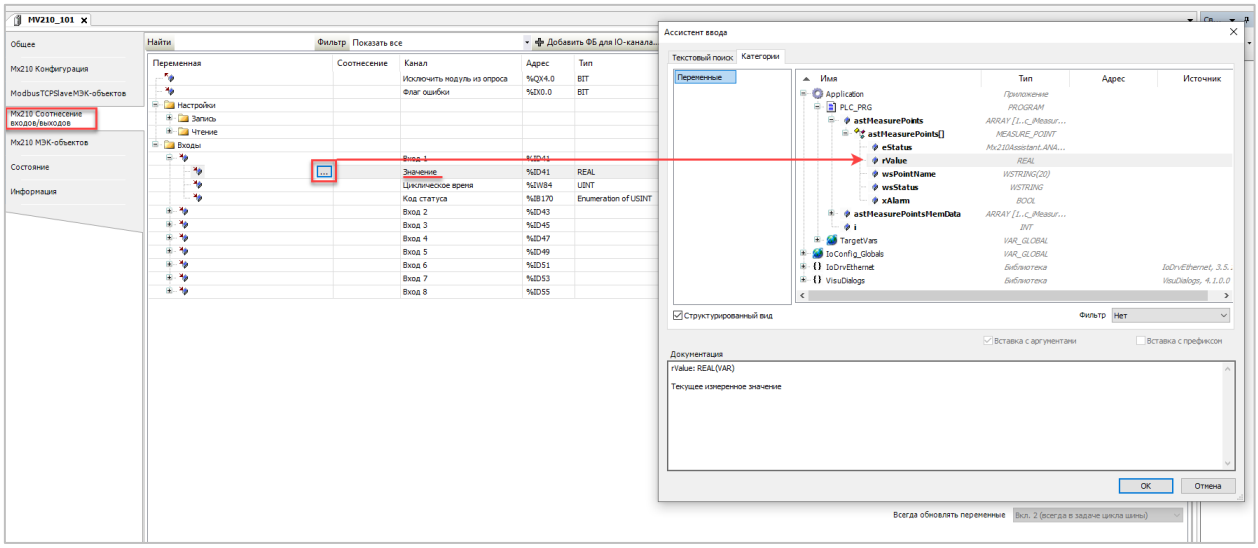


Рисунок 6.5.3 – Выбор переменной для привязки к каналу

После нажатия на кнопку **ОК** в ячейку будет записан путь к переменной (включающий все необходимые пространства имен):

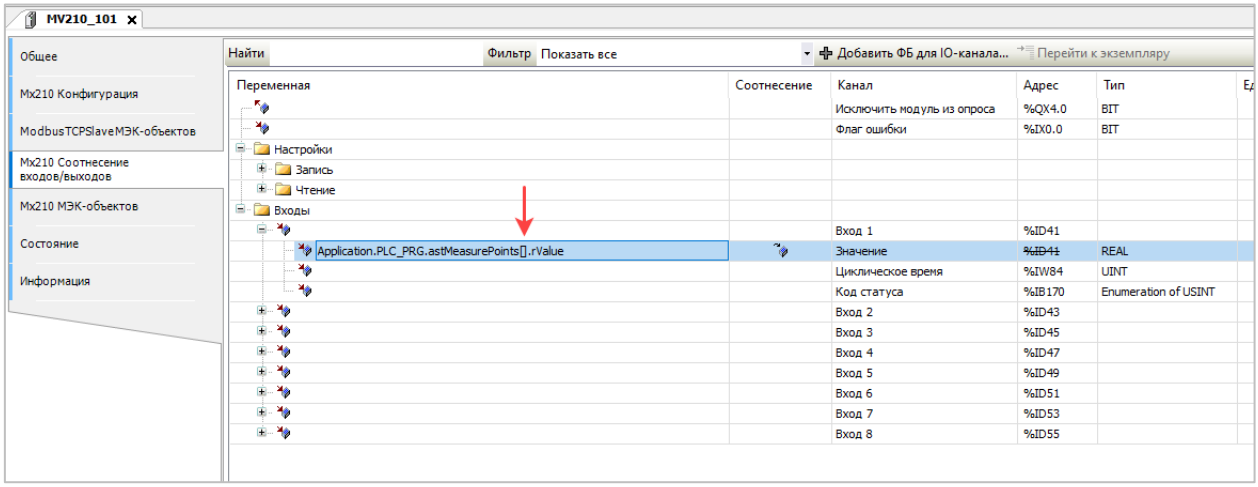


Рисунок 6.5.4 – Привязка переменной. Индекс массива отсутствует

Обратите внимание, что по умолчанию в скобках не указан индекс массива – его необходимо ввести самостоятельно (1 для первого входа, 2 для второго и т. д.):

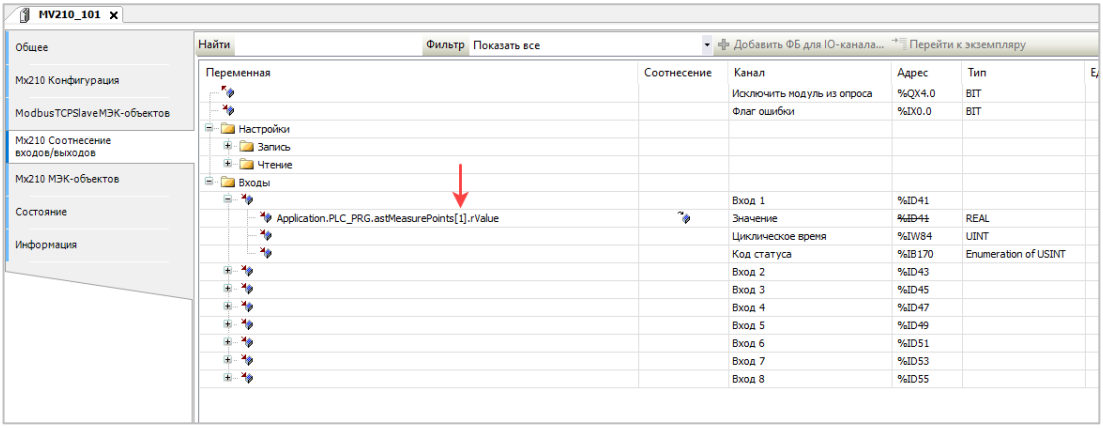


Рисунок 6.5.5 – Привязка переменной. Ручной ввод индекса массива

Привяжите переменные к каналам **Значение** и **Код статуса** всех входов. На рисунке ниже показана привязка переменных к 1 и 2 аналоговому входу; для остальных входов отличие будет только в индексе массива.

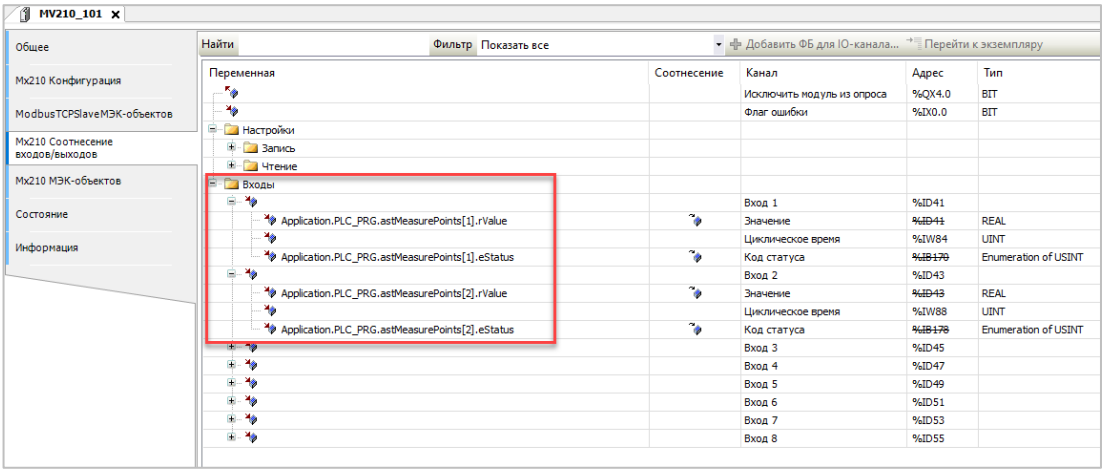


Рисунок 6.5.6 – Привязка переменных к каналам шаблона MB210-101

По аналогии в шаблоне МУ210-401 привяжите переменные **xAlarm** к подканалам канала **Битовая маска выходов (запись)**.

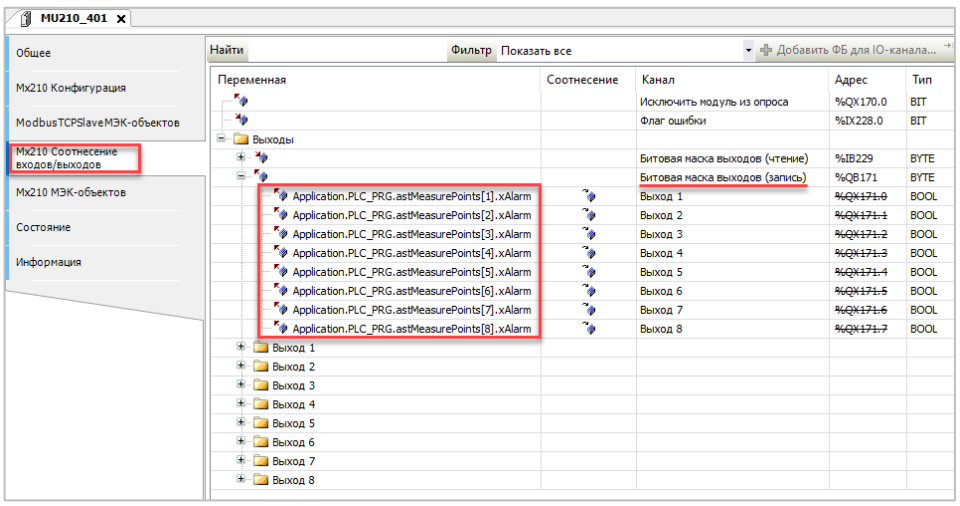


Рисунок 6.5.7 – Привязка переменных к каналам шаблона МУ210-401

6.5.3 Создание функции IsValueOutOfLimits

Одно из действий, которое будет выполняться в нашем коде – генерация сигнала тревоги в том случае, если измеренное значение выходит за допустимый диапазон. Для этого, соответственно, нам нужно контролировать принадлежность значения заданному диапазону. Давайте создадим для этого простейшую функцию с названием **IsValueOutOfLimits**.

Для создания функции нажмите правой кнопкой мыши на узел **Application** и выберите команду **Добавление объекта – POU – Функция**. Тип возвращаемого значения – **BOOL**.

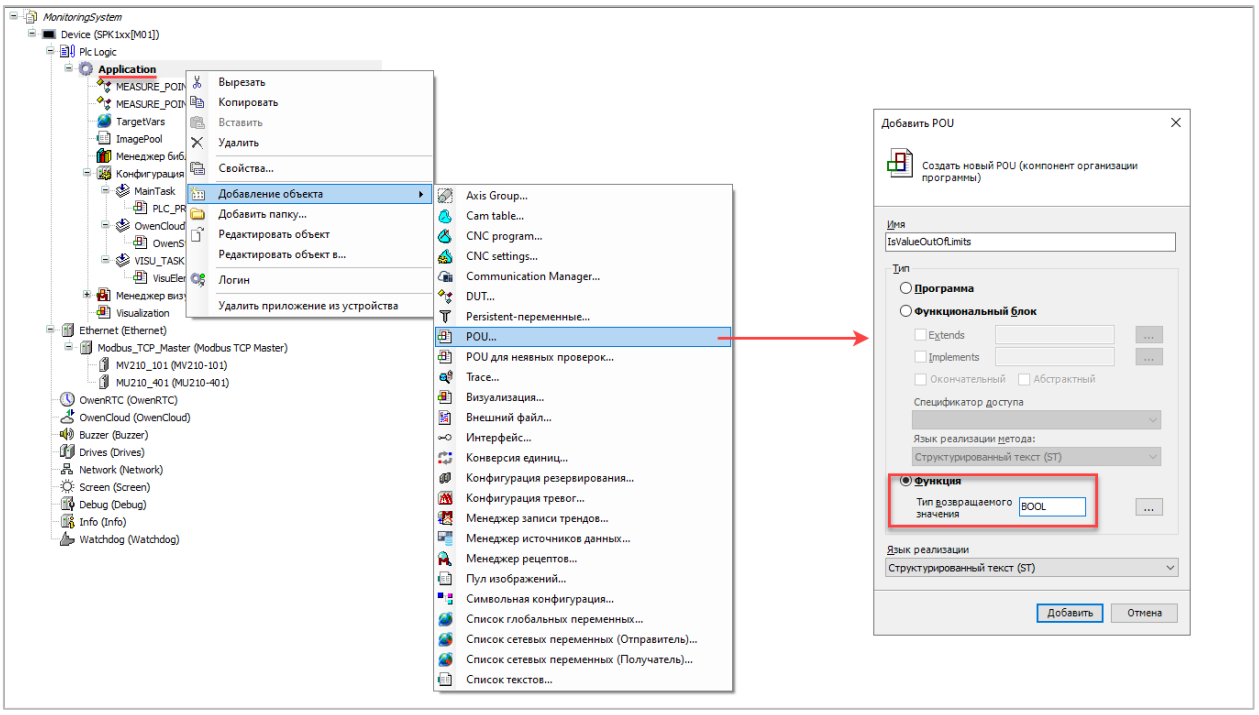


Рисунок 6.5.8 – Создание функции

```
FUNCTION IsValueOutOfLimits : BOOL
VAR_INPUT
   rValue:          REAL;
   rLowAlarmValue:  REAL;
   rHighAlarmValue: REAL;
END_VAR
VAR
END_VAR

// Область кода

IsValueOutOfLimits := rValue < rLowAlarmValue OR rValue > rHighAlarmValue;
```

6.5.4 Код программы – первая версия

Добавим в программу **PLC_PRG** код для генерации сигналов тревог в случае выхода измеренного значения за допустимый диапазон и подсчета количества таких выходов. Кроме того, реализуем конвертацию кода ошибки аналогового входа в строковое представление. Поскольку данные по точкам измерения мы разместили в массивах – то очень просто сделать это в цикле FOR:

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:         ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                        INT;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:   INT  := 8;
END_VAR

// Область кода

FOR i := 1 TO c_iMeasurePointsCount DO

   astMeasurePoints[i].xAlarm := IsValueOutOfLimits(astMeasurePoints[i].rValue,
      astMeasurePointsMemData[i].rLowAlarmValue,
         astMeasurePointsMemData[i].rHighAlarmValue);

   astMeasurePointsMemData[i].fbAlarmCounter(CU := astMeasurePoints[i].xAlarm);

   astMeasurePoints[i].wsStatus :=
      Mx210Assistant.ANALOG_SENSOR_ERROR_TO_WSTRING(astMeasurePoints[i].eStatus);

END_FOR
```

Для конверсии кода ошибки в строку мы используем функцию **ANALOG_SENSOR_ERROR_TO_WSTRING** из библиотеки **Mx210 Assistant**.

Эту версию проекта уже можно загрузить в контроллер и проверить, что она работает корректно. При создании структуры **MEASURE_POINT_MEM** мы указали для нижнего предела допустимого диапазона значение **10**, а для верхнего – **30** (см. п. 6.5.1). Введем в OPC-сервере для аналогового входа 1 значение, которое выходит за этот диапазон – и увидим, что сигнал тревоги получил значение **TRUE**.

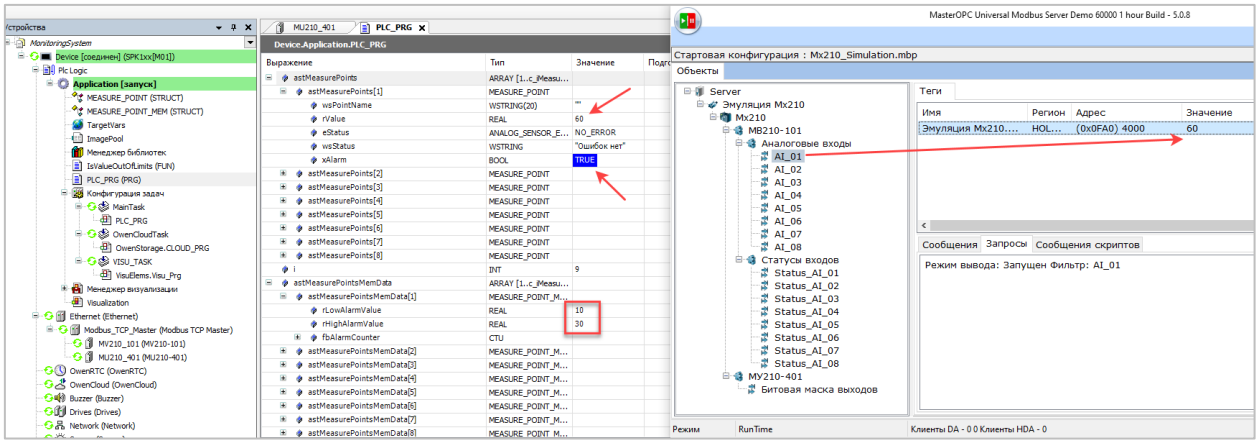


Рисунок 6.5.9 – Проверка работы программы

Если вас смущает текст «Ошибок нет» в переменной **wsStatus**, то напомним, что это сообщение об ошибке измерения модуля – оно никак не связано со значением переменной **xAlarm**, которая сигнализирует о выходе измеренного значения за допустимые границы.

Обсудим отдельно эту строку программы:

```
astMeasurePointsMemData[i].fbAlarmCounter(CU := astMeasurePoints[i].xAlarm);
```

Мы передаем на вход счетчика **CU** значение нашего сигнала тревоги – таким образом, счетчик будет показывать общее количество тревог, случившихся за время эксплуатации контроллера. Но выход счетчика **CV**, который как раз и «отображает» это значение, мы не присваиваем никакой переменной. Это связано с тем, что мы не будем использовать значение счетчика в коде – ограничимся его отображением в визуализации. Поэтому в нашем случае можно обойтись и без промежуточной переменной.

Созданный в данном пункте код вполне работоспособен, но некоторые моменты можно улучшить:

1. В структуре **MEASURE_POINT** есть переменная **wsPointName**, которая определяет название точки измерения. Мы не присвоили этой переменной начальное значение, поэтому в данный момент название каждой из точек является пустой строкой.

2. Мы генерируем сигнал тревоги (**xAlarm**) в момент выхода значения за допустимый диапазон. Но если измеренное значение будет находиться на границе диапазона и постоянно выходить за него и возвращаться обратно, то возможен «дребезг» – сигналы тревоги будут генерироваться очень часто. В реальной жизни это может соответствовать хаотичному миганию аварийных ламп, постоянному включению/отключению звуковой сигнализации и т. д.

3. Конвертация кода статуса в строку выполняется в каждом цикле контроллера. Но разумнее выполнять эту конвертацию только в тот момент, когда происходит изменение кода статуса – то есть перейти от циклического выполнения к событийному.

В следующих пунктах мы оптимизируем нашу программу согласно приведенным замечаниям.

6.5.5 Код программы – вторая версия

Для начала разберемся с названиями точек измерения. Мы могли бы задать им какие-то значения при объявлении массива структур – но, как уже упоминалось в п. 5.3.8, синтаксис инициализации массива структур довольно сложен.

Вместо этого сделаем инициализацию прямо в коде (добавленный код выделен **жирным шрифтом**):

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:         ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                        INT;
   xInit:                    BOOL;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:    INT  := 8;
END_VAR


IF NOT(xInit) THEN

  // размещенный здесь код будет однократно выполнен при старте приложения

  astMeasurePoints[1].wsPointName := "Точка измерения 1";
  astMeasurePoints[2].wsPointName := "Точка измерения 2";
  astMeasurePoints[3].wsPointName := "Точка измерения 3";
  astMeasurePoints[4].wsPointName := "Точка измерения 4";
  astMeasurePoints[5].wsPointName := "Точка измерения 5";
  astMeasurePoints[6].wsPointName := "Точка измерения 6";
  astMeasurePoints[7].wsPointName := "Точка измерения 7";
  astMeasurePoints[8].wsPointName := "Точка измерения 8";

  xInit := TRUE;

END_IF


FOR i := 1 TO c_iMeasurePointsCount DO

   astMeasurePoints[i].xAlarm := IsValueOutOfLimits(astMeasurePoints[i].rValue,
      astMeasurePointsMemData[i].rLowAlarmValue,
         astMeasurePointsMemData[i].rHighAlarmValue);

   astMeasurePointsMemData[i].fbAlarmCounter(CU := astMeasurePoints[i].xAlarm);

   astMeasurePoints[i].wsStatus :=
      Mx210Assistant.ANALOG_SENSOR_ERROR_TO_WSTRING(astMeasurePoints[i].eStatus);

END_FOR
```

Код, размещенный в добавленном нами условном операторе IF, будет однократно выполнен при старте приложения.

Следующий момент, который стоит оптимизировать – исключение «дребезга» сигнала тревоги для случая, когда измеренное значение находится на границе диапазона и постоянно выходит за него и возвращается обратно.

Один из вариантов поведения в подобной ситуации – сделать задержку срабатывания тревоги. То есть сигнал выхода измеренного значения за границы диапазона должен сохраняться в течение некоторого заданного нами времени – и только после этого будет генерироваться тревога.

Соответственно, нам нужно разделить сигнал выхода значения за границы диапазона и сигнал тревоги на две отдельных переменных, а также объявить экземпляр таймера. Внесем изменения (выделены **жирным шрифтом**) в нашу структуру **MEASURE_POINT**:

```
// Структура оперативных параметров одной точки измерения системы автоматизации
TYPE MEASURE_POINT :
STRUCT
   // Название точки измерения
   wsPointName:   WSTRING(20);
   // Текущее измеренное значение
   rValue:        REAL;
   // Код ошибки аналогового входа
   eStatus:       Mx210Assistant.ANALOG_SENSOR_ERRORS;
   // Текстовое описание ошибки
   wsStatus:      WSTRING;
   // Признак выхода значения за допустимые границы
   xOutOfLimits:  BOOL;
   // Сигнал ошибки
   xAlarm:        BOOL;
   // Экземпляр таймера, используемый для задержки генерации ошибки
   // Позволяет отфильтровывать кратковременный выход значения за допустимую границу
   fbAlarmDelay:  TON;
END_STRUCT
END_TYPE
```

Теперь изменим код программы, добавив в нее вызов экземпляра таймера (изменения выделены **жирным шрифтом**):

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:        ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                       INT;
   xInit:                   BOOL;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:   INT  := 8;
   c_tAlarmDelay:           TIME := T#5S;
END_VAR


IF NOT(xInit) THEN

  // размещенный здесь код будет однократно выполнен при старте приложения

  astMeasurePoints[1].wsPointName := "Точка измерения 1";
  astMeasurePoints[2].wsPointName := "Точка измерения 2";
  astMeasurePoints[3].wsPointName := "Точка измерения 3";
  astMeasurePoints[4].wsPointName := "Точка измерения 4";
  astMeasurePoints[5].wsPointName := "Точка измерения 5";
  astMeasurePoints[6].wsPointName := "Точка измерения 6";
  astMeasurePoints[7].wsPointName := "Точка измерения 7";
  astMeasurePoints[8].wsPointName := "Точка измерения 8";

  xInit := TRUE;

END_IF


FOR i := 1 TO c_iMeasurePointsCount DO

   astMeasurePoints[i].xOutOfLimits := IsValueOutOfLimits(astMeasurePoints[i].rValue,
      astMeasurePointsMemData[i].rLowAlarmValue,
         astMeasurePointsMemData[i].rHighAlarmValue);

   astMeasurePoints[i].fbAlarmDelay
   (
      IN := astMeasurePoints[i].xOutOfLimits OR
         astMeasurePoints[i].eStatus <> Mx210Assistant.ANALOG_SENSOR_ERRORS.NO_ERROR,
      PT := c_tAlarmDelay,
      Q => astMeasurePoints[i].xAlarm
   );

   astMeasurePointsMemData[i].fbAlarmCounter(CU := astMeasurePoints[i].xAlarm);

   astMeasurePoints[i].wsStatus :=
      Mx210Assistant.ANALOG_SENSOR_ERROR_TO_WSTRING(astMeasurePoints[i].eStatus);

END_FOR
```

Отметим, что теперь в логике формирования тревоги мы используем не только признак выхода измеренного значения за границы допустимого диапазона, но и код ошибки измерения аналогового датчика (**eStatus**). Если значение **eStatus** отличается от **NO_ERROR** («ошибок нет»), то это также приведет к генерации сигнала тревоги.

6.5.6 Создание функционального блока StatusChange

Остался последний момент, который мы планировали оптимизировать – а именно конвертация кода ошибки в строку. Как мы упоминали в конце п. 6.5.4 – нет смысла делать это каждый цикл задачи; достаточно проводить конвертацию в момент изменения кода ошибки. Для этого нам потребуется определять этот момент изменения – то есть нужно сравнивать код статуса, считанный в текущем цикле задачи, с кодом статуса, считанным в предыдущем цикле задачи.

Давайте создадим для этой цели отдельный функциональный блок. Функция здесь не подойдет, так как нам потребуется сохранять код статуса между вызовами POU – а у функций, как мы помним, памяти нет.

Для создания функционального блока нажмите правой кнопкой мыши на узел **Application** и выберите команду **Добавление объекта – POU – Функциональный блок**. Назовем наш функциональный блок **StatusChange**.

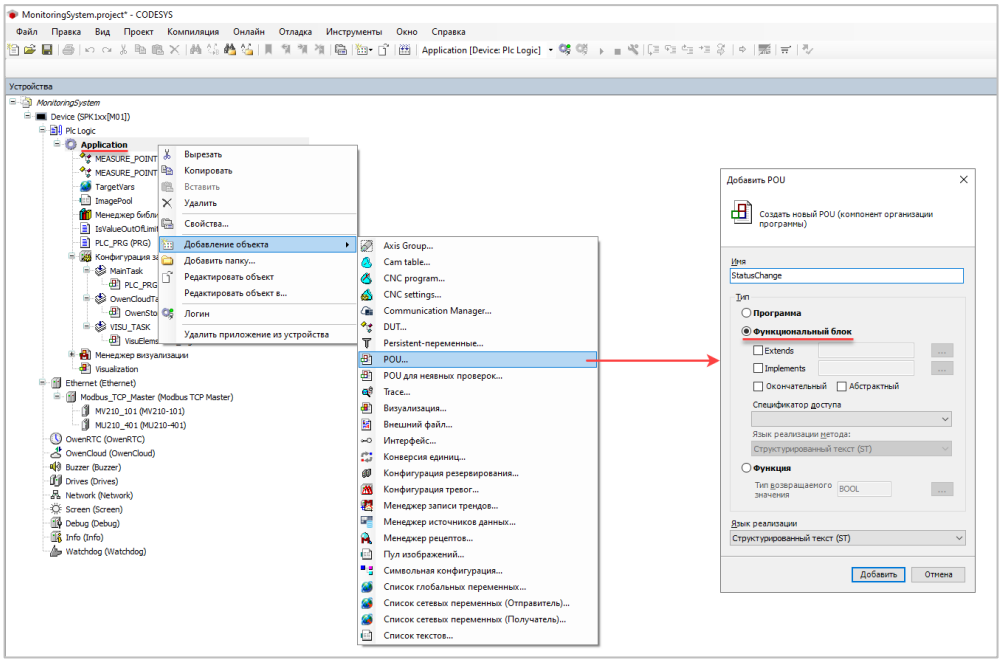


Рисунок 6.5.10 – Создание функционального блока

```
FUNCTION_BLOCK StatusChange
VAR_INPUT
   eStatus:      Mx210Assistant.ANALOG_SENSOR_ERRORS;
END_VAR
VAR_OUTPUT
   Q:            BOOL;
END_VAR
VAR
   ePrevStatus:  Mx210Assistant.ANALOG_SENSOR_ERRORS;
END_VAR

// Область кода

IF eStatus <> ePrevStatus THEN
   Q := TRUE;
   ePrevStatus := eStatus;
ELSE
   Q := FALSE;
END_IF
```

При изменении кода статуса на выходе **Q** генерируется единичный импульс (потому что если код статуса не изменился с предыдущего цикла задачи, то выходу **Q** присваивается значение **FALSE**).

Добавим экземпляр блока в структуру **MEASURE_POINT** (изменения выделены **жирным**):

```
// Структура оперативных параметров одной точки измерения системы автоматизации
TYPE MEASURE_POINT :
STRUCT
   // Название точки измерения
   wsPointName:    WSTRING(20);
   // Текущее измеренное значение
   rValue:         REAL;
   // Код ошибки
   eStatus:        Mx210Assistant.ANALOG_SENSOR_ERRORS;
   // Текстовое описание ошибки
   wsStatus:       WSTRING;
   // Признак выхода значения за допустимые границы
   xOutOfLimits:   BOOL;
   // Сигнал ошибки
   xAlarm:         BOOL;
   // Экземпляр таймера, используемый для задержки генерации ошибки
   // Позволяет отфильтровывать кратковременный выход значения за допустимую границу
   fbAlarmDelay:   TON;
   // Экземпляр блока детектирования изменения кода ошибки
   // Используется для определения момента,
   // в которой нужно сформировать текстовое описание кода ошибки
   fbStatusChange: StatusChange;
END_STRUCT
END_TYPE
```

Теперь организуем вызов экземпляра блока в нашей программе:

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:        ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                       INT;
   xInit:                   BOOL;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:   INT  := 8;
   c_tAlarmDelay:           TIME := T#5S;
END_VAR

// Область кода

IF NOT(xInit) THEN

  // размещенный здесь код будет однократно выполнен при старте приложения

  astMeasurePoints[1].wsPointName := "Точка измерения 1";
  astMeasurePoints[2].wsPointName := "Точка измерения 2";
  astMeasurePoints[3].wsPointName := "Точка измерения 3";
  astMeasurePoints[4].wsPointName := "Точка измерения 4";
  astMeasurePoints[5].wsPointName := "Точка измерения 5";
  astMeasurePoints[6].wsPointName := "Точка измерения 6";
  astMeasurePoints[7].wsPointName := "Точка измерения 7";
  astMeasurePoints[8].wsPointName := "Точка измерения 8";

  xInit := TRUE;

END_IF
```

```
FOR i := 1 TO c_iMeasurePointsCount DO

   astMeasurePoints[i].xOutOfLimits := IsValueOutOfLimits(astMeasurePoints[i].rValue,
      astMeasurePointsMemData[i].rLowAlarmValue,
         astMeasurePointsMemData[i].rHighAlarmValue);

   astMeasurePoints[i].fbAlarmDelay
   (
      IN := astMeasurePoints[i].xOutOfLimits OR
         astMeasurePoints[i].eStatus <> Mx210Assistant.ANALOG_SENSOR_ERRORS.NO_ERROR,
      PT := c_tAlarmDelay,
      Q => astMeasurePoints[i].xAlarm
   );

   astMeasurePointsMemData[i].fbAlarmCounter(CU := astMeasurePoints[i].xAlarm);


   astMeasurePoints[i].fbStatusChange(eStatus := astMeasurePoints[i].eStatus);

   IF astMeasurePoints[i].fbStatusChange.Q THEN
      astMeasurePoints[i].wsStatus :=
         Mx210Assistant.ANALOG_SENSOR_ERROR_TO_WSTRING(astMeasurePoints[i].eStatus);
   END_IF

END_FOR
```

На этом оптимизация программы завершена; в следующих разделах мы объявим в ней еще несколько переменных, но основная часть ее кода останется неизменной.

Теперь можно приступать к созданию визуализации.

6.6 Создание визуализации

Наш проект будет содержать два экрана визуализации:

- основной экран, на котором будет отображаться информация о точках измерения;
- экран с информацией о тревогах.

В нашем примере 8 точек измерения, каждая из которых включает в себя ряд параметров – название точки, значение и т. д. Отобразить их все на одном экране панельного контроллера СПК будет затруднительно – а у оператора будут разбегаться глаза от количества графических элементов.

Поэтому мы будем в каждый момент времени отображать информацию по одной точке, предоставив оператору возможность переключаться между ними. Для этого мы используем **фрейм** – шаблон экрана, с помощью которого можно в одних и тех же графических элементах отображать значения различных переменных.

6.6.1 Создание фрейма

В нашем проекте, созданном на базе шаблона, уже есть экран **Visualization** – мы будем использовать его в качестве основного экрана, на котором будем отображать фрейм.

Добавим в проект экран визуализации с названием **frmMeasurePoint**, из которого в дальнейшем сделаем наш фрейм.

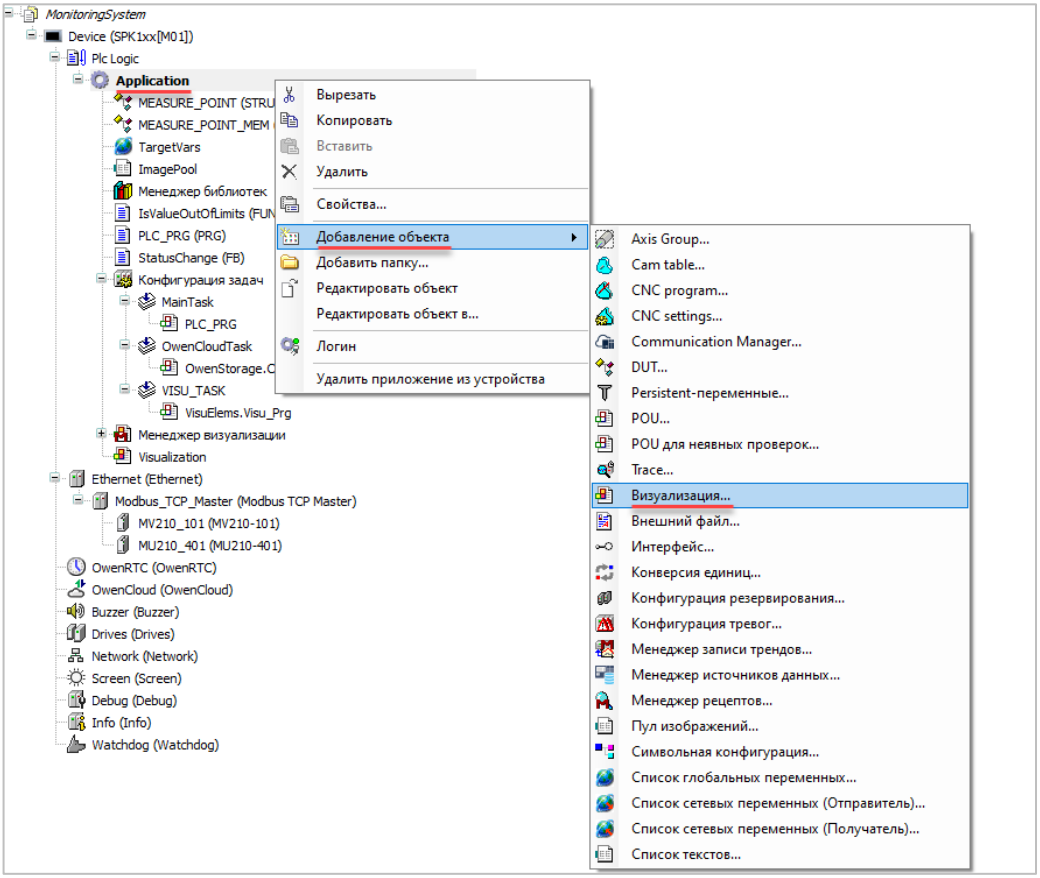


Рисунок 6.6.1 – Создание экрана визуализации

В его свойствах выберем режим **Использовать заданный размер визуализации** и установим размер **400x250** – потому что фрейм будет лишь частью основного экрана визуализации и должен быть меньше его по размерам (размеры экранов панельных контроллеров СПК совпадают с разрешением дисплея контроллера и составляют **800x480** пикселей).

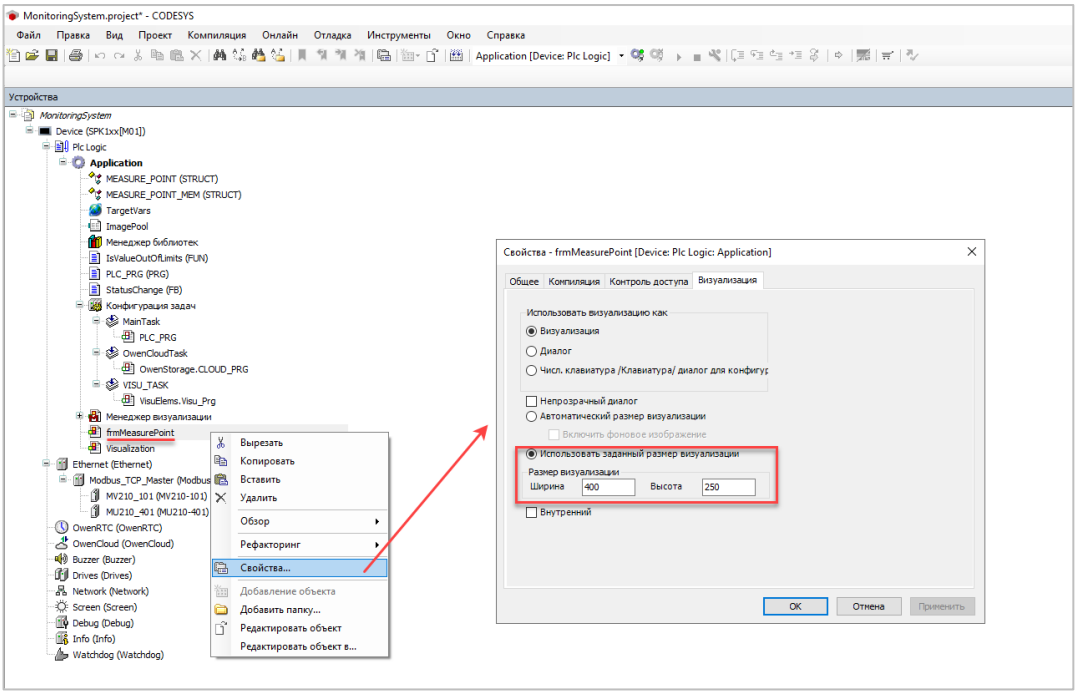


Рисунок 6.6.2 – Свойства экрана визуализации

Теперь перейдем в редактор визуализации, откроем в нем редактор интерфейсов (для этого может потребоваться нажать на маленькую стрелочку, отмеченную на рис. 6.6.3) и добавим в нем область входов-выходов (VAR_IN_OUT), в которой объявим экземпляры структур **MEASURE_POINT** и **MEASURE_POINT_MEM**.

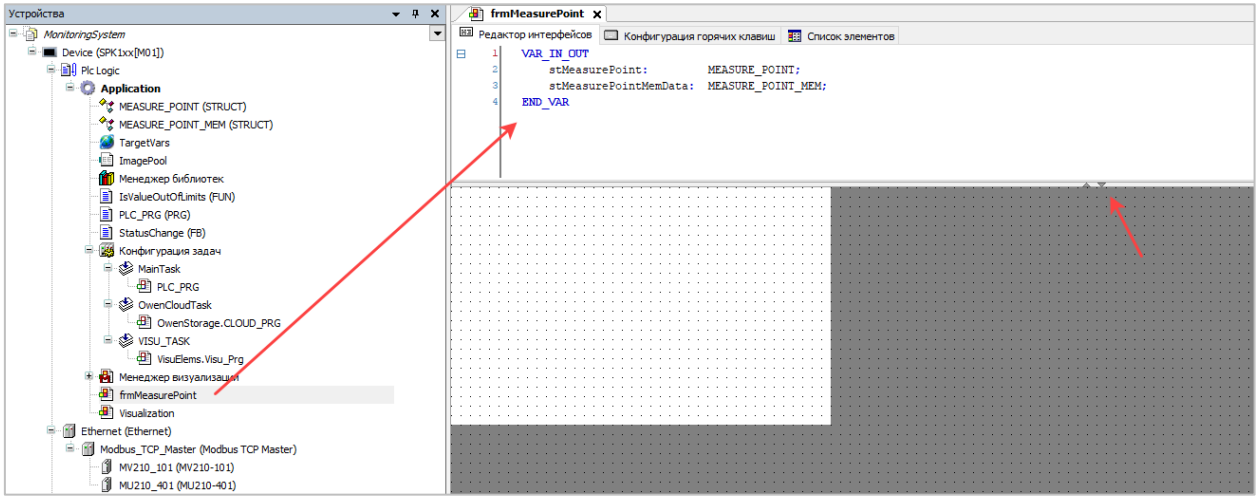


Рисунок 6.6.3 – Создание интерфейса фрейма

```
VAR_IN_OUT
   stMeasurePoint:        MEASURE_POINT;
   stMeasurePointMemData: MEASURE_POINT_MEM;
END_VAR
```

Именно в момент объявления входных переменных (VAR_INPUT) или входов-выходов (**VAR_IN_OUT**) экран визуализации перестает быть экраном и становится фреймом, который не может использоваться самостоятельно – он обязательно должен быть добавлен на один из экранов визуализации проекта с помощью графического элемента **Фрейм** или **Набор вкладок**.

Мы использовали область входов-выходов (**VAR_IN_OUT**) по той причине, что часть параметров (например, название точки измерения и измеренное значение) будут передаваться из программы в визуализацию, а часть (например, границы допустимого диапазона) будет задаваться пользователем визуализации и передаваться в программу. В принципе, все записываемые со стороны визуализации значения расположены в структуре **MEASURE_POINT_MEM** – так что можно было бы оставить в области **VAR_IN_OUT** только ее, а экземпляр структуры **MEASURE_POINT** разместить в области **VAR_INPUT**. Но используемый нами подход, при котором оба экземпляра размещены в области **VAR_IN_OUT** – тоже легитимен.

Теперь откроем панель инструментов визуализации и перетащим на экран фрейма графические элементы:

- 6 прямоугольников (вкладка **Базовые**);
- 2 индикатора (вкладка **Lamps/Switches/Bitmaps**);
- 1 кнопку (вкладка **Стандартные элементы управления**).

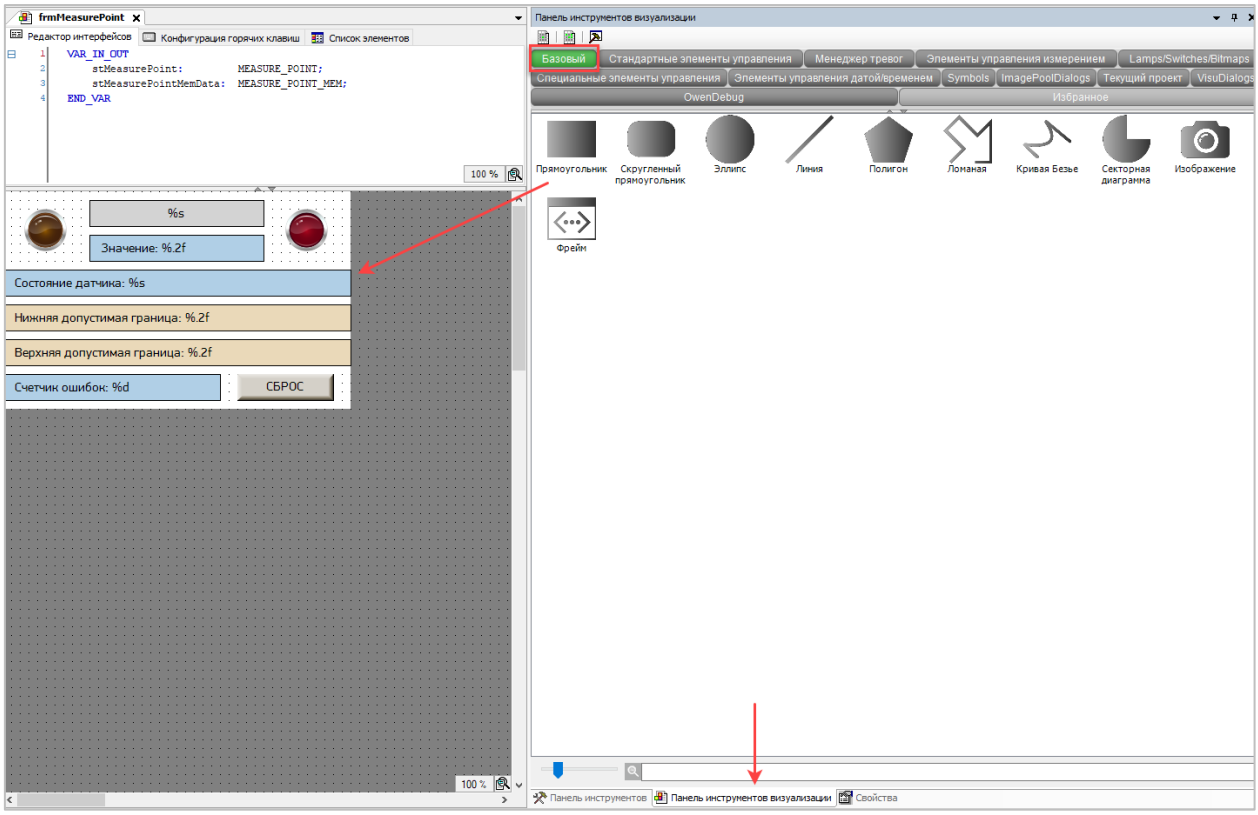


Рисунок 6.6.4 – Добавление элементов на экран фрейма

После выделения элемента нажатием левой кнопки мыши откроется окно его свойств. Для перемещения элемента по экрану нужно нажать на него левой кнопкой мыши и, не отпуская ее, передвинуть элемент в нужное место. Для изменения размера элемента нужно «потянуть» за одну из опорных точек (отображаются синими квадратиками). Кроме того, перемещать элементы по экрану и изменять их размер можно с помощью свойств вкладки **Позиция**.

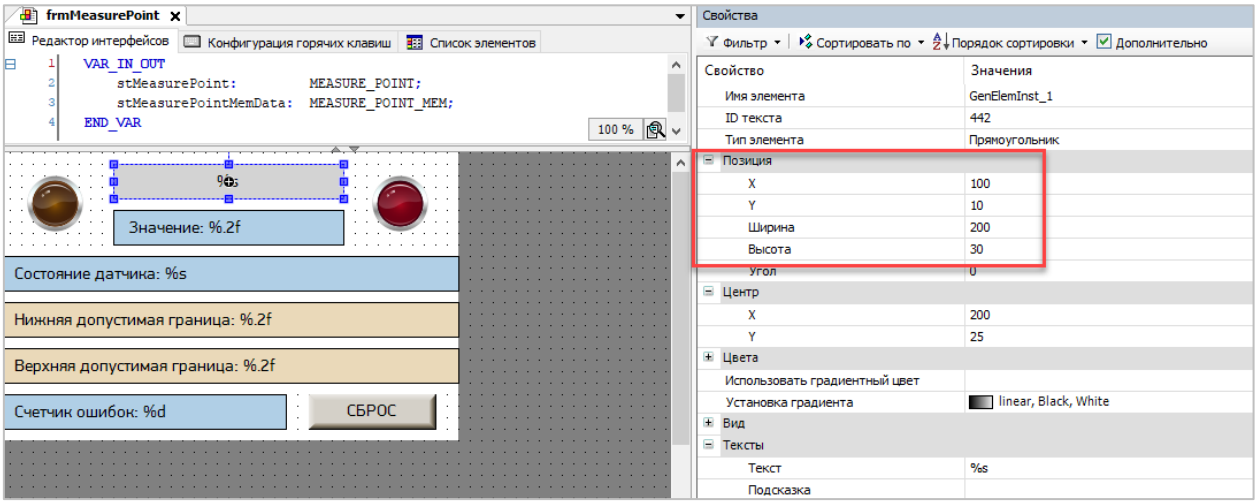


Рисунок 6.6.5 – Свойства элемента визуализации

Цвета элементов задаются в свойстве **Цвета/Нормальное состояние/Цвет заливки**, а тексты – в свойстве **Тексты/Текст**.

В визуализации фрейма используются три цвета:

- серый – для отображения статической информации (названия точки измерения);
- синий – для отображения значений, которые не могут быть изменены пользователем визуализации (измеренные значения, строки состояния датчиков и т. д.);
- песочный – для отображения значений, которые могут быть изменены пользователем визуализации (допустимые границы измеренных значений).

В текстах элементов используются особые спецификаторы, указывающие на тип значения, отображаемого элементом:

- **%s** – для строковых переменных;
- **%.2f** – для переменных с плавающей точкой (2 – это число символов, отображаемых после плавающей точки; вы можете указать другое значение);
- **%d** – для целочисленных переменных.

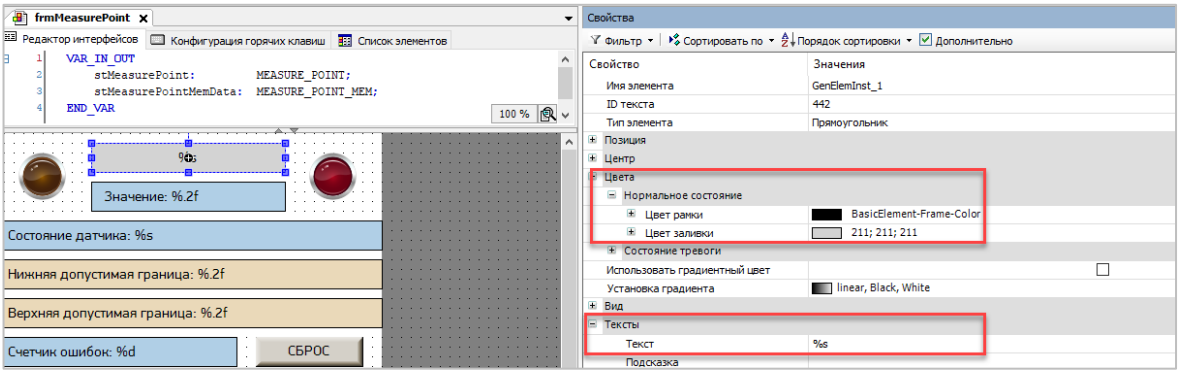


Рисунок 6.6.6 – Установка цвета и текста элемента

Привязка переменной, значение которой будет отображать элемент, осуществляется в свойстве **Текстовые переменные/Текстовая переменная**. В случае фрейма – к этим свойствам привязываются переменные структур, объявленных в интерфейсе фрейма.

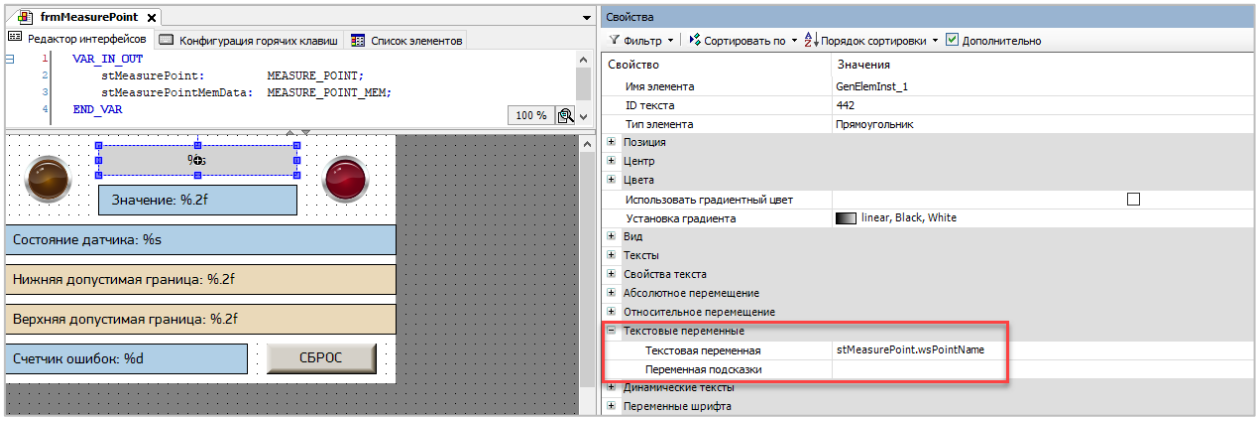


Рисунок 6.6.7 – Установка цвета и привязка переменной к элементу

К прямоугольникам привязаны следующие переменные:

- к первому прямоугольнику – **stMeasurePoint.wsPointName**
- Значение – **stMeasurePoint.rValue**
- Состояние датчика – **stMeasurePoint.wsStatus**
- Нижняя допустимая граница – **stMeasurePointMemData.rLowAlarmValue**
- Верхняя допустимая граница – **stMeasurePointMemData.rHighAlarmValue**
- Счетчик ошибок – **stMeasurePointMemData.fbAlarmCounter.CV**

Обратите внимание: к «счетчику ошибок» мы привязываем выход экземпляра функционального блока. Такой подход вполне допустим, если не требуется использовать выход блока в самой программе – иначе лучше для повышения читабельности кода создать промежуточную переменную.

Параметры «**Нижняя допустимая граница**» и «**Верхняя допустимая граница**» будут записываться пользователем визуализации – это нужно настроить отдельно (см. рис. 6.6.8). В группе свойств **Конфигурация ввода** есть набор событий (нажатие на элемент, отпускание элемента и т. д.), для каждого из которых можно задать набор действий. Выберите событие **OnMouseClick** («клик» – нажатие и отпускание элемента) и настройте для него действие **Записать переменную**. По умолчанию выбран режим **Исп. текстовую выходную переменную** – в этом случае запись будет произведена в переменную, привязанную к свойству **Текстовая переменная**. Нам как раз подходит этот вариант.

С помощью других настроек действия можно выбрать тип ввода (экранная или аппаратная клавиатура), заголовок диалога ввода значения для экранной клавиатуры, нижний и верхний предел вводимого значения и т. д.

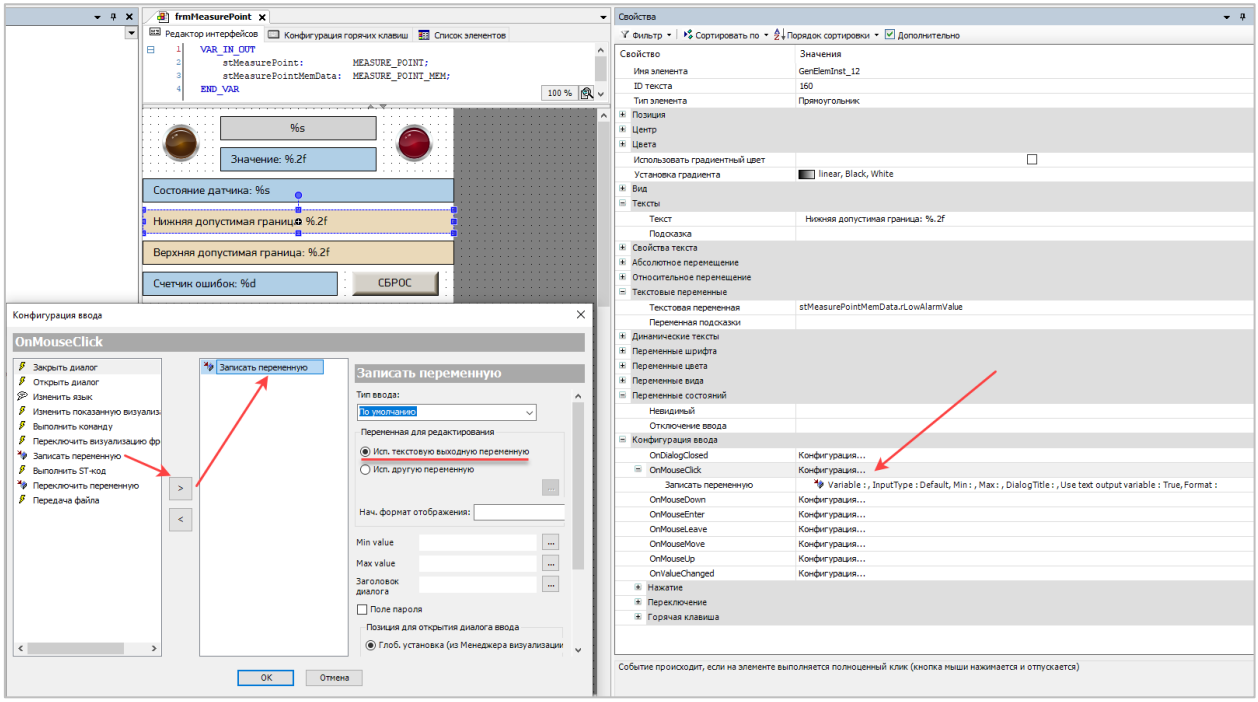


Рисунок 6.6.8 – Настройка конфигурации ввода элемента для записи значения

К желтому индикатору привяжем переменную **stMeasurePoint.xOutOfLimits** (сигнал выхода значения за допустимые границы), а к красному – **stMeasurePoint.xAlarm** (сигнал тревоги). Цвет индикатора выбирается в свойстве **Фон/Изображение**.

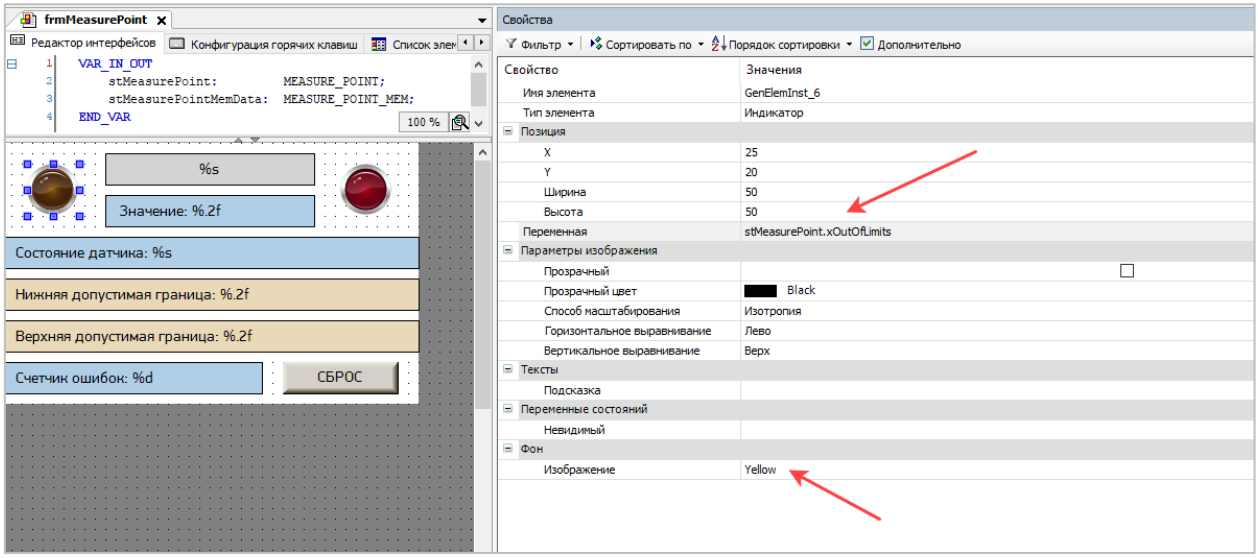


Рисунок 6.6.9 – Привязка переменной к индикатору и настройка его цвета

Осталось привязать переменную к кнопке сброса счетчика тревог. Для этого раскроем группу свойств **Конфигурация ввода** и привяжем к свойству **Нажатие/Переменная** вход блока счетчика RESET (**stMeasurePointMemData.fbAlarmCounter.RESET**).

Переменная, привязанная к этому свойству, принимает значение **TRUE** на время нажатия кнопки. После отпускания кнопки она автоматически будет сброшена в **FALSE**.

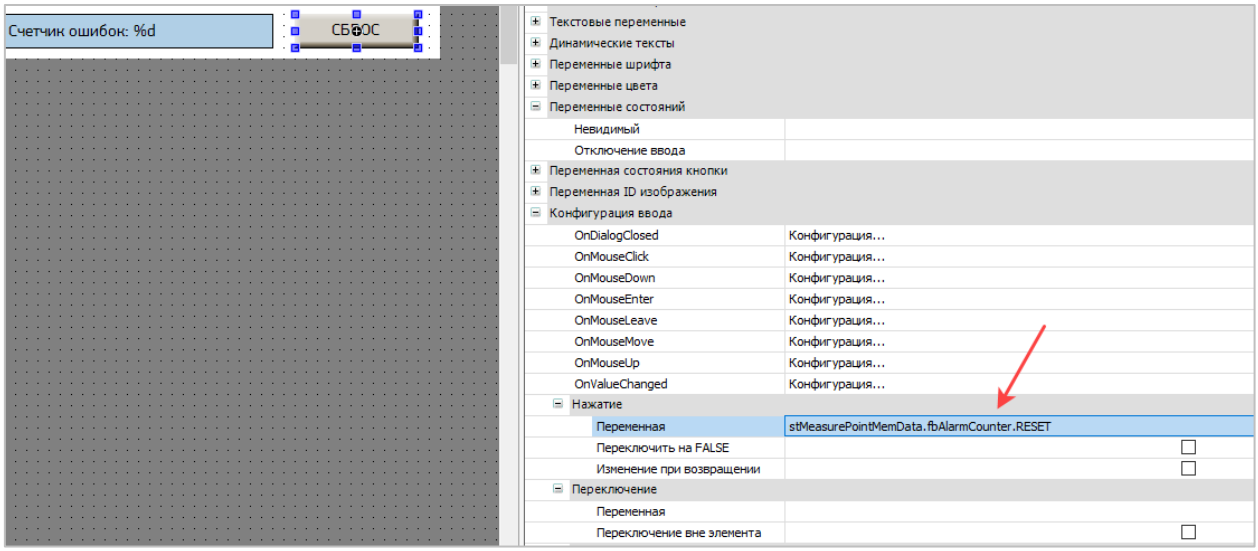


Рисунок 6.6.10 – Привязка переменной к кнопке

Если вам потребуется создать кнопку «с фиксацией», нажатие на которую изменяет значение переменной на противоположное – то используйте свойство **Конфигурация ввода/Переключение/Переменная**.

Размер тени кнопки настраивается с помощью свойства **Высота кнопки** (для нашей кнопки мы использовали значение 5).

Наш фрейм готов – теперь можно разместить его на основном экране визуализации.

6.6.2 Добавление и настройка основных элементов экранов визуализации

В нашем проекте, созданном на базе шаблона, уже присутствует экран **Visualization**. Нажмите два раза левой кнопкой мыши в дереве проекта на его название и переименуйте экран в **VisuMainScreen**. После этого откроются окна переименования и рефакторинга – нажмите в них кнопки **Да** и **ОК**.

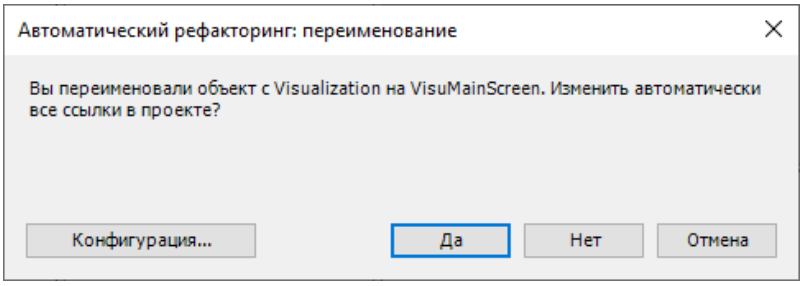


Рисунок 6.6.11 – Окно переименования

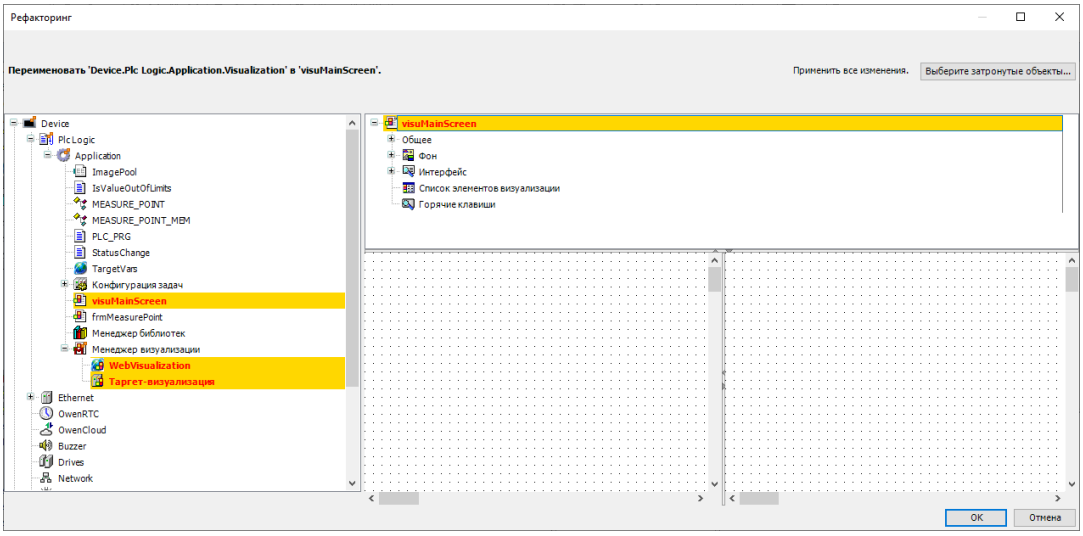


Рисунок 6.6.12 – Окно рефакторинга

Создадим второй экран, который будет использоваться для отображения таблицы тревог. Нажмите на экран **visuMainScreen** в дереве проекта правой кнопкой мыши и выберите команду **Копировать**. Выделите в дереве проекта узел **Application**, нажмите на него правой кнопкой мыши и используйте команду **Вставить**.

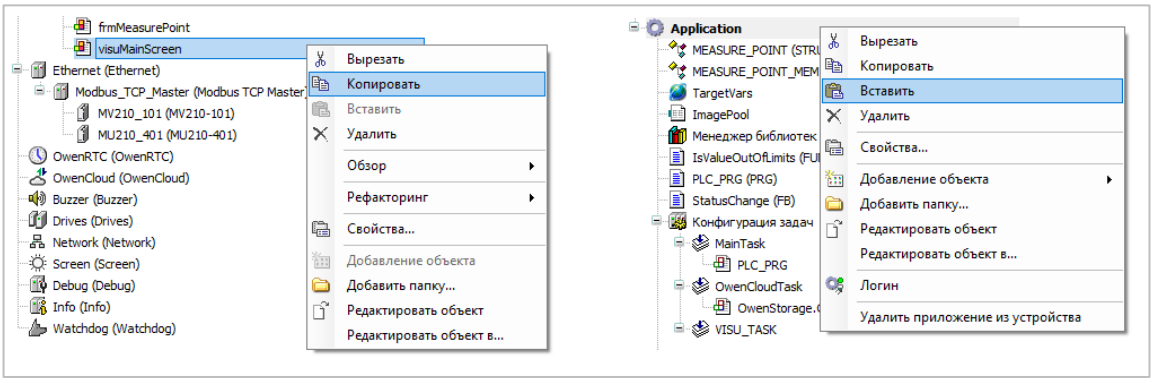


Рисунок 6.6.13 – Копирование экрана визуализации

Переименуйте скопированный экран в **visuAlarmTable**.

Приступим к наполнению экрана **visuMainScreen**.

Для начала разместим на экране несколько элементов:

- элемент **Метка** для отображения названия экрана (вкладка **Стандартные элементы управления**);
- 2 прямоугольника, которые будут использоваться для отображения текущей даты и времени;
- 2 кнопки, которые будут использоваться для перехода между экранами.

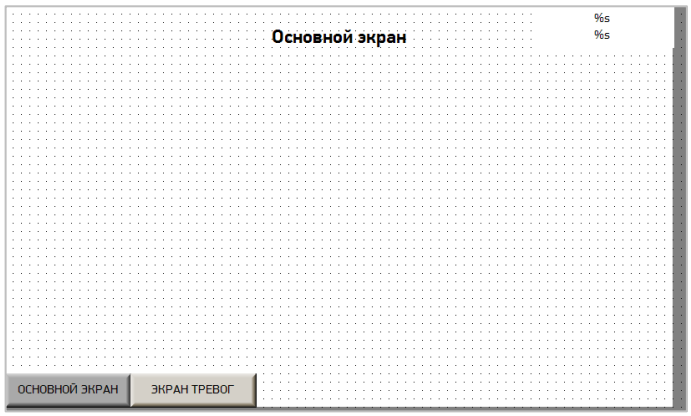


Рисунок 6.6.14 – Размещение элементов на экране visuMainScreen

Текущие дата и время будут отображаться в строковом виде.

В дереве проекта присутствует системный узел **OwenRTC**, в котором есть каналы для работы с системным временем, и список глобальных переменных **TargetVars**, содержащий экземпляры структур, переменные которых уже привязаны к этим каналам.

Привяжите к текстовой переменной верхнего прямоугольника переменную **TargetVars.stRtc.sGetFormatDate**, а к текстовой переменной нижнего – **TargetVars.stRtc.sGetFormatTime**. Напомним, что для отображения строковых переменных в свойстве **Тексты/Текст** нужно указать спецификатор **%s**.

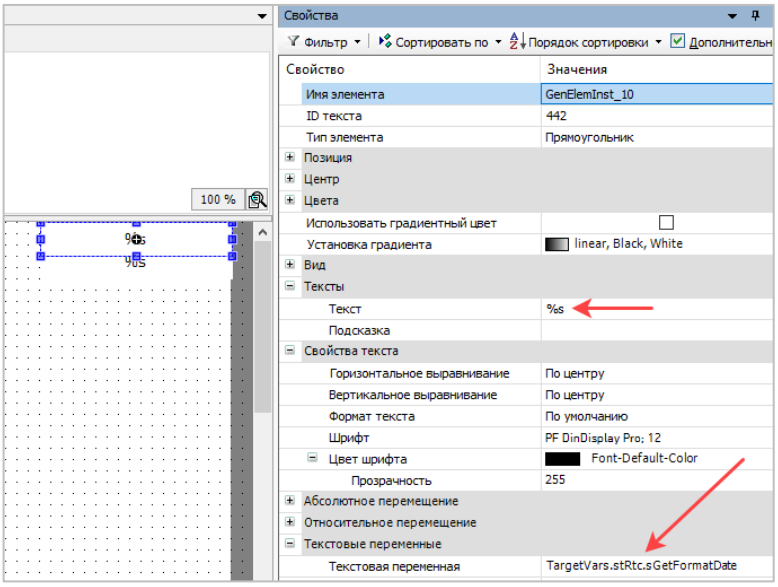


Рисунок 6.6.15 – Настройки отображения системного времени

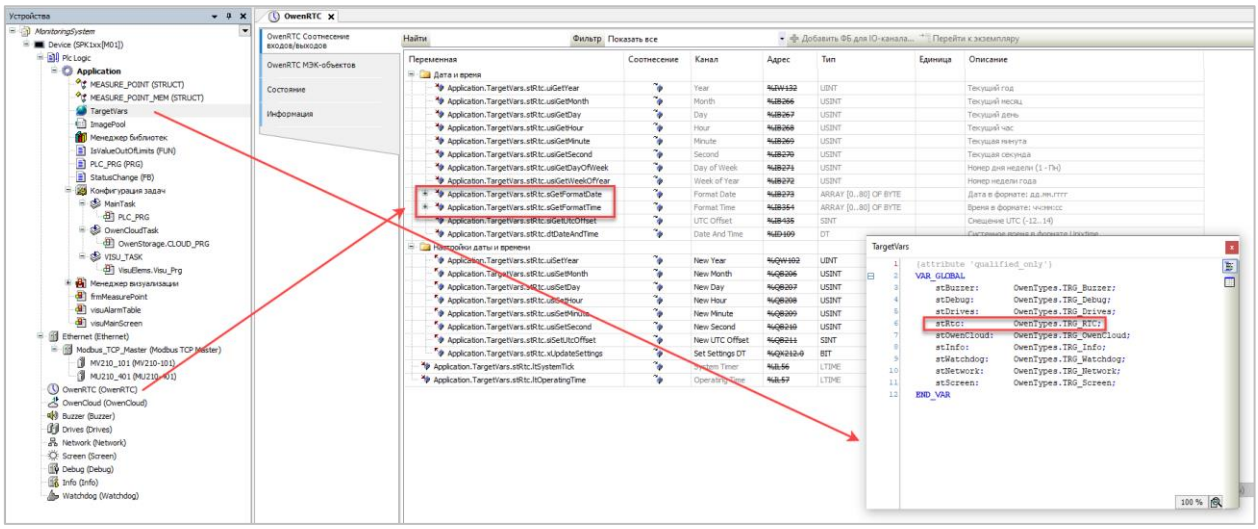


Рисунок 6.6.16 – Переменные экземпляров структур глобального списка переменных TargetVars привязаны к каналам системных узлов дерева проекта

Обратите внимание, что этот функционал не получится проверить на виртуальном контроллере, так как у него в дереве проекта не будет системных узлов – они поддержаны только для контроллеров ОВЕН.

Теперь настроим для кнопок переходы между экранами. В группе свойств **Конфигурация ввода** выберите событие **OnMouseClick**, привяжите к нему действие **Изменить показанную визуализацию** и выберите для кнопки «ОСНОВНОЙ ЭКРАН» визуализацию **visuMainScreen**, а для кнопки «ЭКРАН ТРЕВОГ» – визуализацию **visuAlarmTable**.

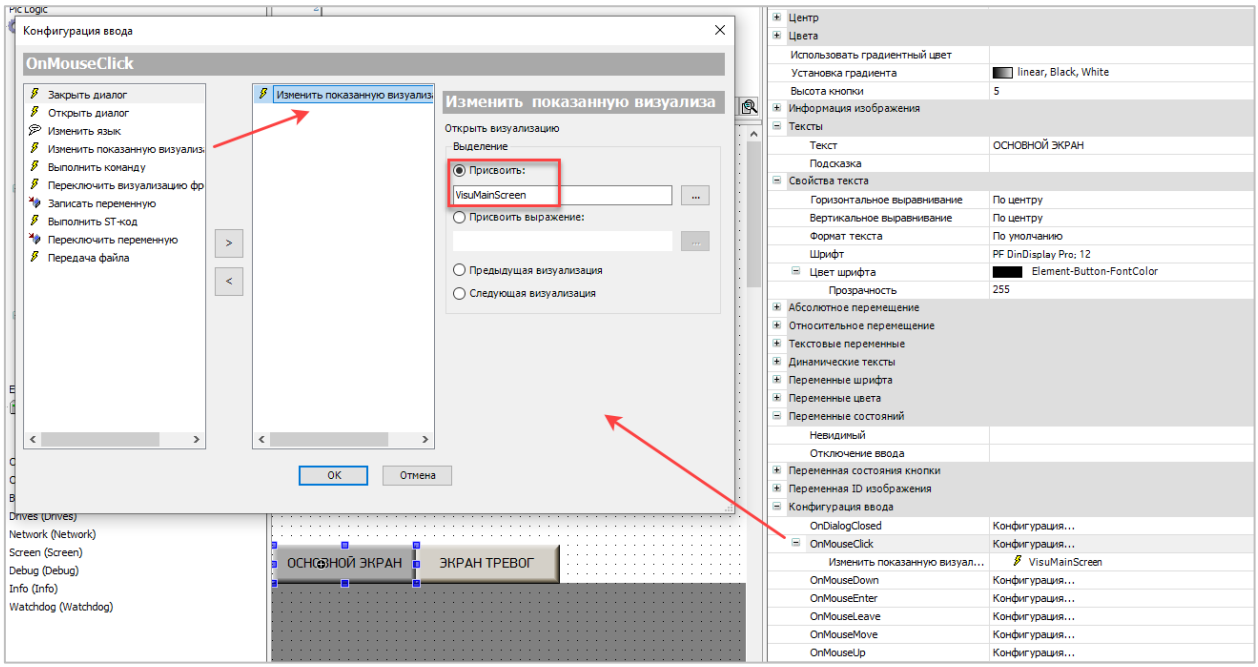


Рисунок 6.6.17 – Настройка действия переключения экранов для кнопки

Скопируйте все эти элементы на экран тревог. Измените название экрана и цвета кнопок (кнопка текущего открытого экрана выделяется отдельным цветом).

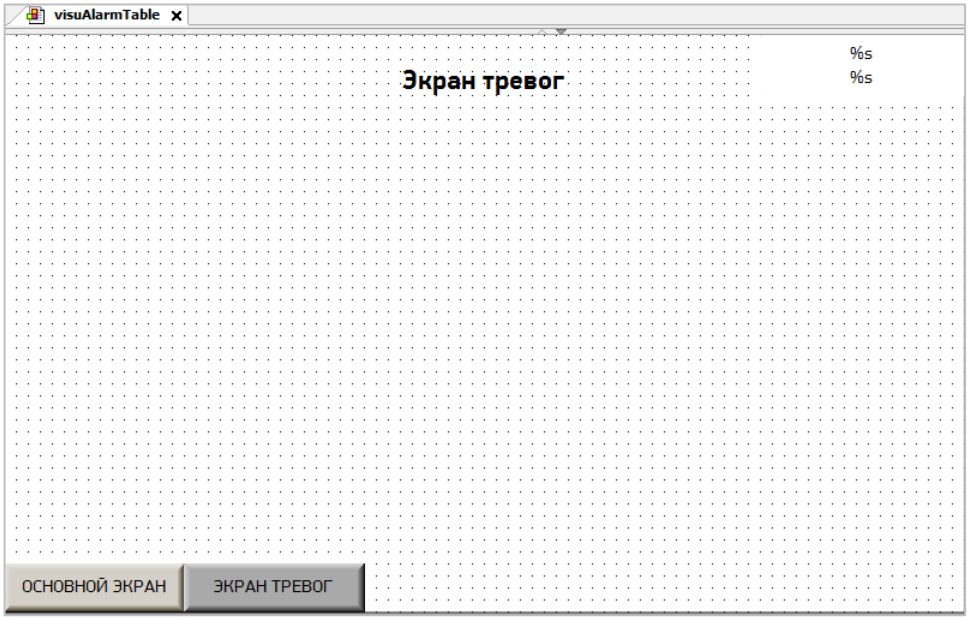


Рисунок 6.6.18 – Размещение элементов на экране visuAlarmTable

Мы настроили общие для обоих экранов элементы.

Теперь перейдем к элементам, специфичным для каждого экрана. Для экрана **visuMainScreen** таким элементом будет **Фрейм**, а для **visuAlarmTable** – **Таблица тревог**.

6.6.3 Настройка экрана visuMainScreen

На экране **visuMainScreen** нужно разместить созданный нами в п. 6.6.1 фрейм. Для этого добавьте на экран элемент **Фрейм** из вкладки **Базовые**. После добавления элемента автоматически откроется окно его настроек. На левой панели выделите фрейм **frmMeasurePoint**, а на правой нажмите кнопку **Добавить** для привязки к элементу экземпляра фрейма.

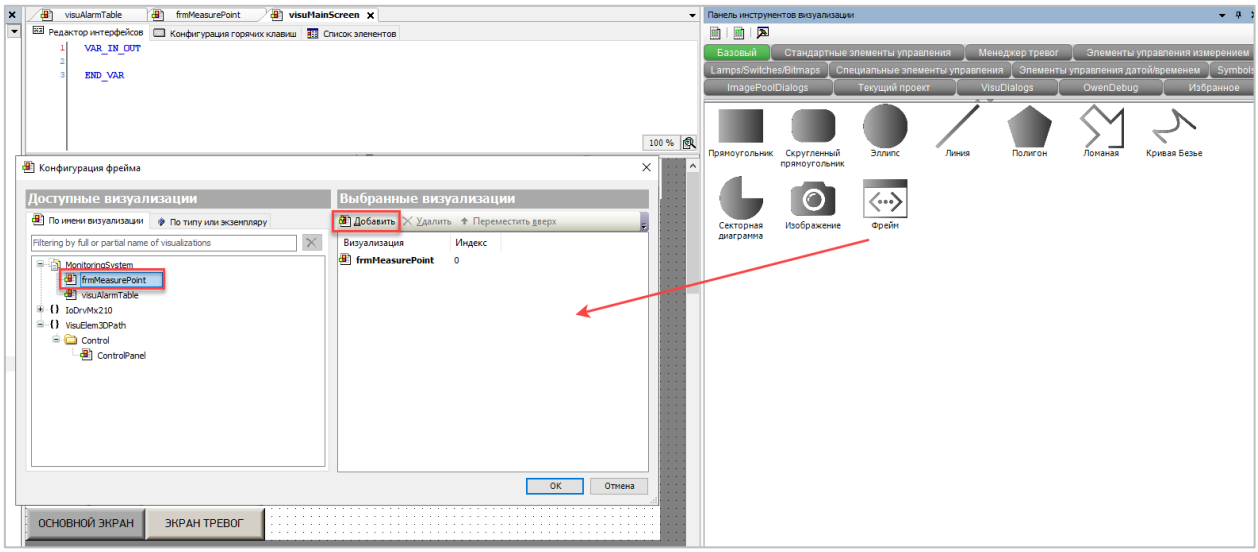


Рисунок 6.6.19 – Добавление элемента Фрейм на экран визуализации

Резонный вопрос – почему мы привязали к элементу **Фрейм** только один экземпляр фрейма, ведь в нашем проекте 8 точек измерения? Дело в том, что в каждый момент времени пользователь визуализации будет просматривать информацию только по одной точке – поэтому мы можем обойтись одним экземпляром фрейма и с помощью кнопок переключения точек «подставлять» в этот экземпляр данные по нужной точке.

Для этого нам потребуется новая переменная, которая будет определять номер текущей точки, отображаемой в визуализации. Объявим ее в программе **PLC_PRG** и присвоим начальное значение 1 (первая точка).

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:         ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                        INT;
   xInit:                    BOOL;
   iCurrentFrameIndex:       INT := 1;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:    INT := 8;
   c_tAlarmDelay:            TIME := T#5S;
END_VAR
```

Теперь нужно привязать к элементу **Фрейм** переменные, значения которых будут в нем отображаться. Для этого в свойствах элемента откройте группу свойств **Ссылки** – в ней будет отображаться экземпляр привязанного фрейма и его интерфейсные переменные.

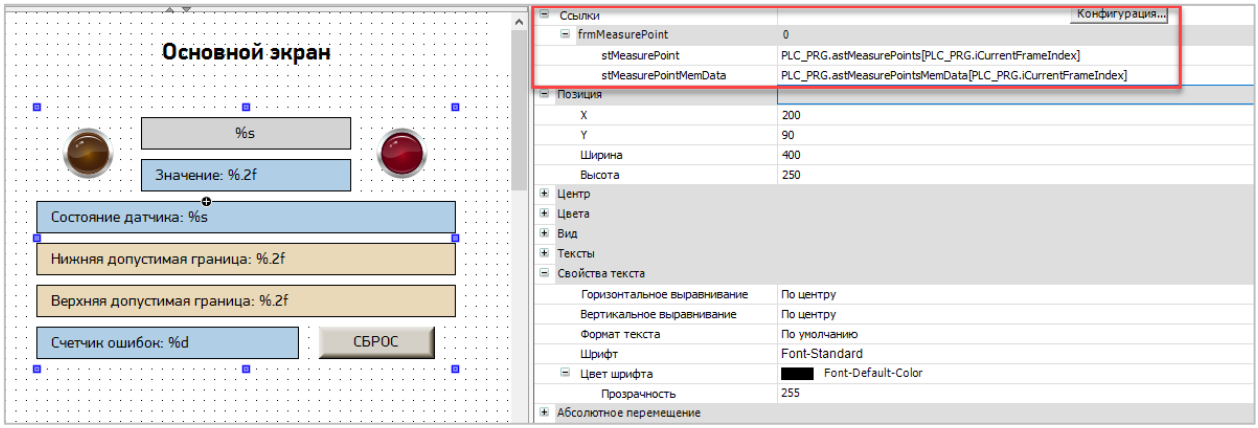


Рисунок 6.6.19 – Привязка переменных к элементу Фрейм

Привяжите к входу-выходу **stMeasurePoint** переменную

PLC_PRG.astMeasurePoints[PLC_PRG.iCurrentFrameIndex]

То есть мы привязываем элемент массива структур, в качестве индекса которого используется переменная. Меняя значение этой переменной – мы будем изменять элемент массива, значение которого в данный момент отображается во фрейме. По аналогии привяжите к входу-выходу **stMeasurePointMemData** переменную

PLC_PRG.astMeasurePointsMemData[PLC_PRG.iCurrentFrameIndex]

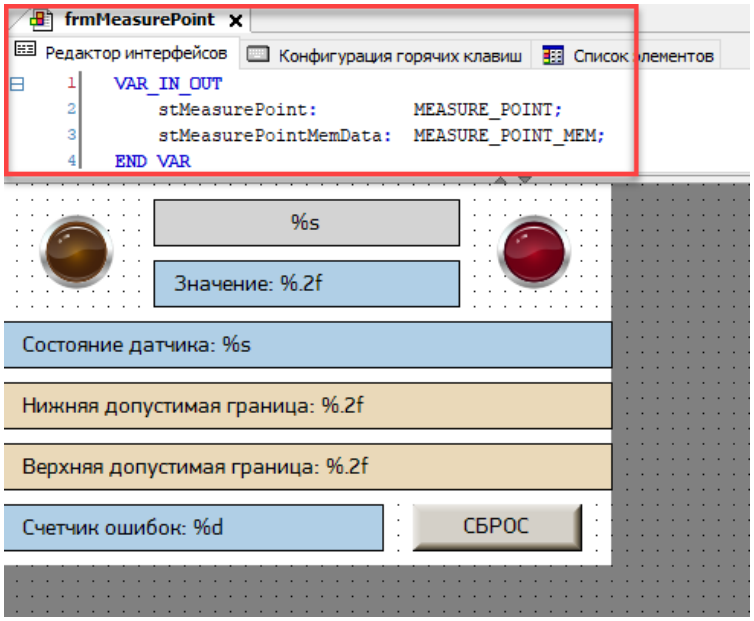


Рисунок 6.6.20 – Интерфейсные переменные, созданные в п. 6.6.1 – именно они отображаются в ссылках элемента Фрейм

Осталось добавить кнопки переключения точек – с их помощью мы будем изменять значение переменной **iCurrentFrameIndex**.

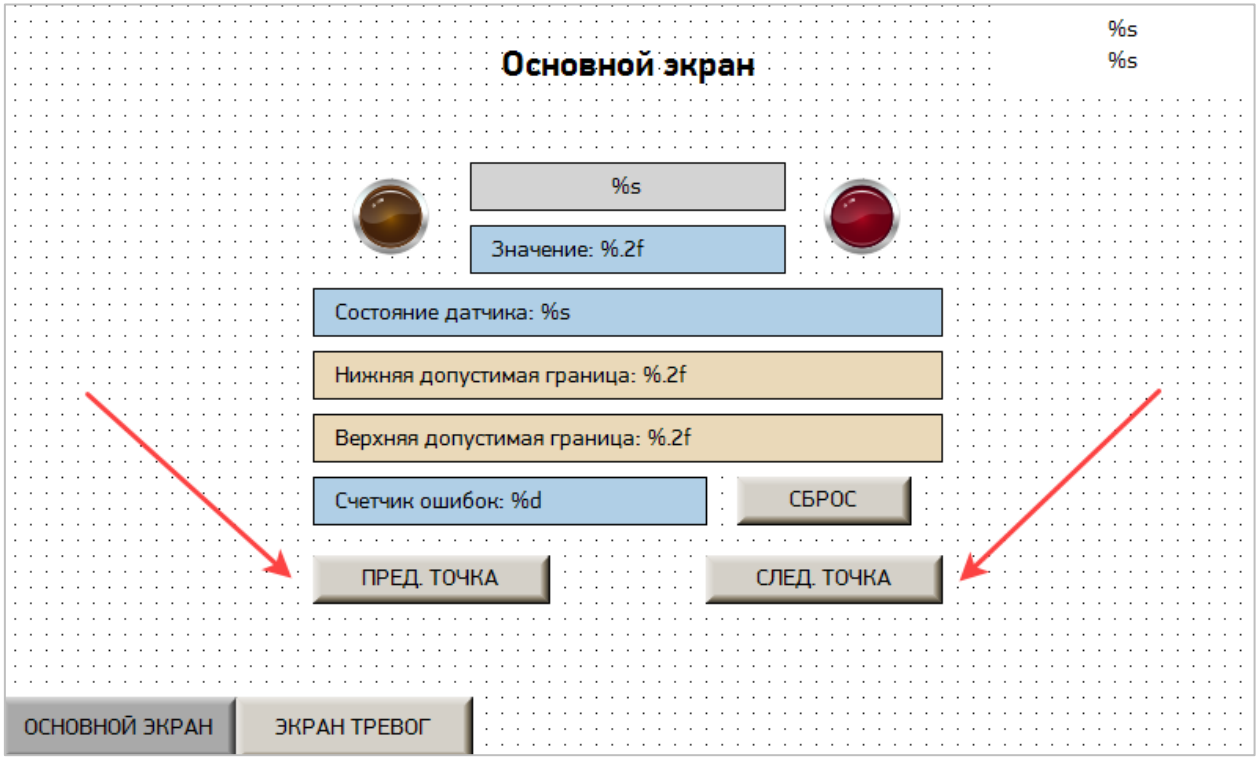


Рисунок 6.6.21 – Добавление кнопок переключения точек

В группе свойств **Конфигурация ввода** добавьте событие **OnMouseClick** и привяжите к нему действие **Выполнить ST-код**. ***Обратите внимание***, что при обращении к переменным программ нужно указывать пространство имен программы – то есть ее название (**PLC_PRG**).

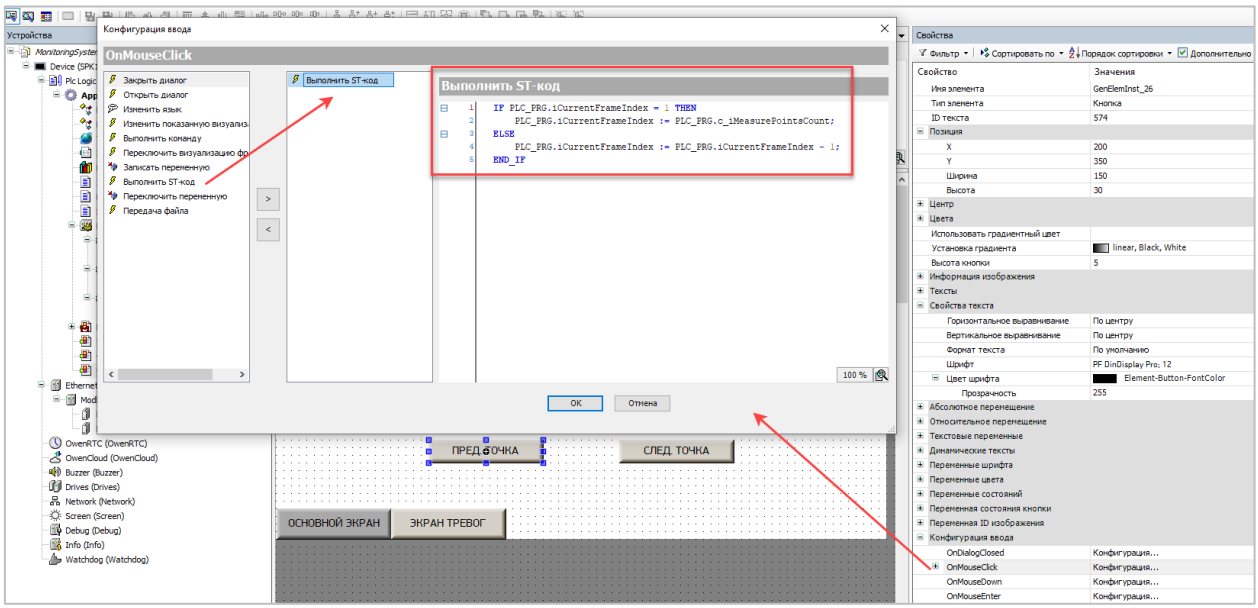


Рисунок 6.6.22 – Настройка выполнения ST-кода при нажатии на кнопку

Код кнопки «ПРЕД. ТОЧКА»:

```
IF PLC_PRG.iCurrentFrameIndex = 1 THEN
   PLC_PRG.iCurrentFrameIndex := PLC_PRG.c_iMeasurePointsCount;
ELSE
   PLC_PRG.iCurrentFrameIndex := PLC_PRG.iCurrentFrameIndex - 1;
END_IF
```

Код прост – при каждом нажатии на кнопку мы уменьшаем индекс массива данных (то есть номер точки) на 1. Если же в данный момент пользователь работает с первой точкой – то при нажатии на кнопку происходит переход к последней точке (ее номер определяется константой **PLC_PRG.c_iMeasurePointsCount**).

Код кнопки «СЛЕД. ТОЧКА» работает «наоборот» – при нажатии на кнопку индекс массива увеличивается на 1, а если пользователь работает с последней точкой, то нажатие на кнопку приводит к переходу на первую точку.

```
IF PLC_PRG.iCurrentFrameIndex = PLC_PRG.c_iMeasurePointsCount THEN
   PLC_PRG.iCurrentFrameIndex := 1;
ELSE
   PLC_PRG.iCurrentFrameIndex := PLC_PRG.iCurrentFrameIndex + 1;
END_IF
```

На этом создание экрана **visuMainScreen** завершено.

Теперь нужно организовать работу с тревогами.

6.6.4 Создание конфигурации тревог

На экране **visuAlarmTable** будет располагаться таблица тревог. Но чтобы использовать этот элемент – сначала нужно настроить тревоги в нашем проекте.

Добавим в проект объект **Конфигурация тревог**:

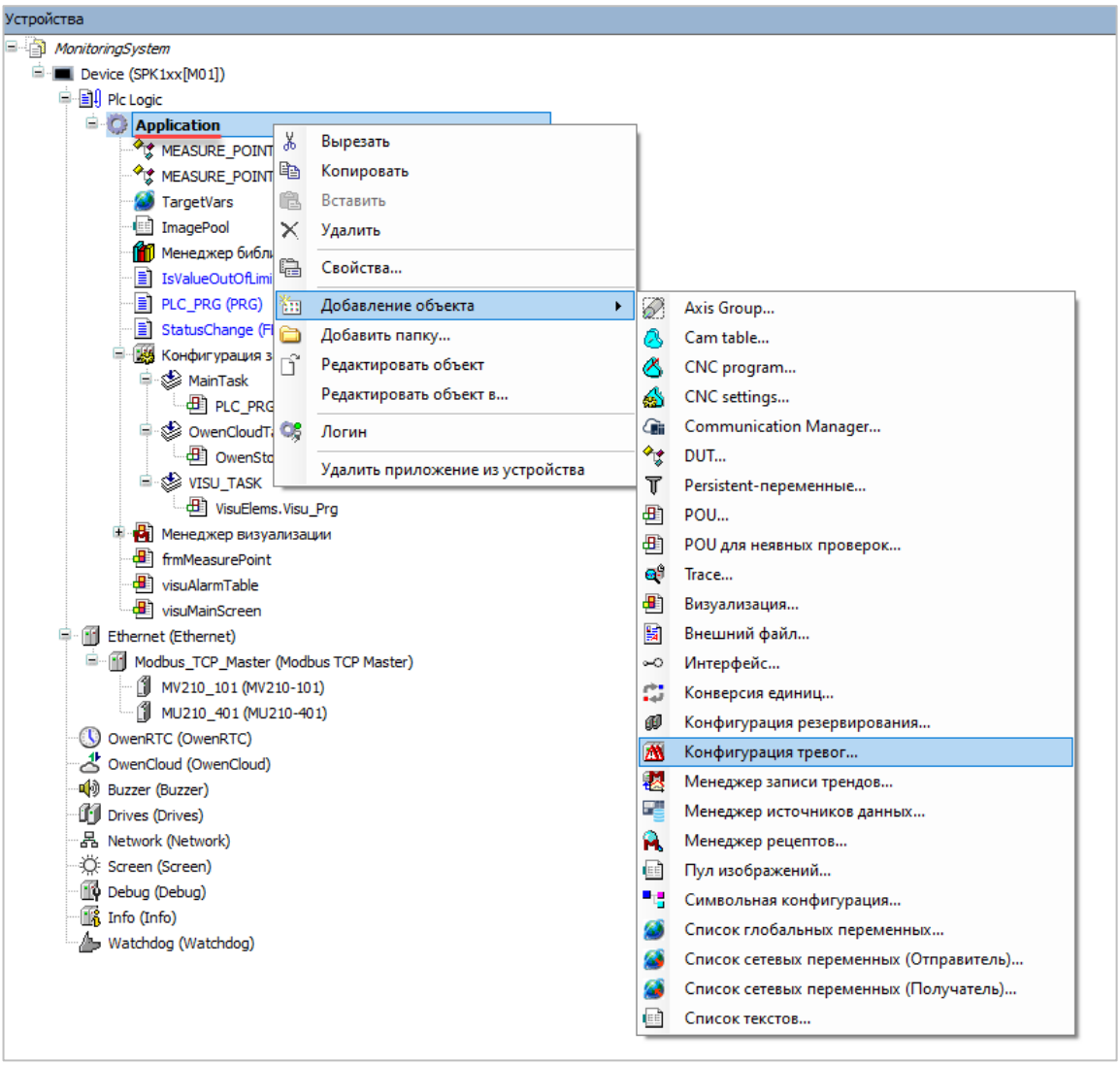


Рисунок 6.6.23 – Добавление в проект конфигурации тревог

Сразу добавим в конфигурацию тревог подобъект **Группа тревог**. Кстати, ***обратите внимание***, что при добавлении в проект конфигурации тревог в конфигурацию задач была автоматически добавлена задача **AlarmManagerTask**. Эту задачу не следует удалять (иначе возникнут ошибки компиляции) и не рекомендуется редактировать (то есть изменять ее приоритет и интервал вызова).

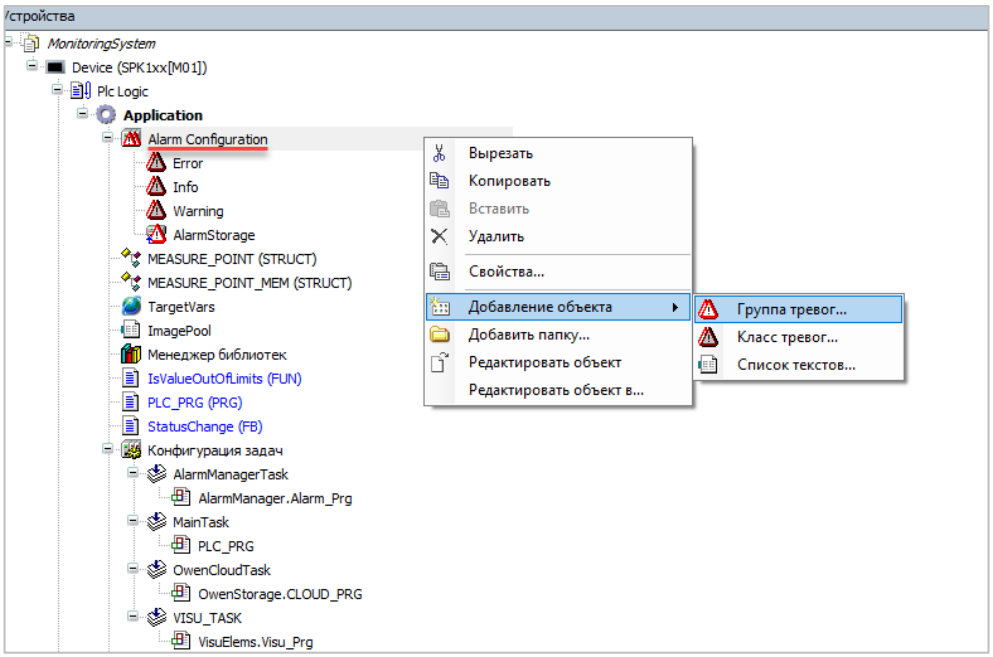


Рисунок 6.6.24 – Добавление группы тревог

Вместе с группой тревог будет автоматически добавлен одноименный список текстов.

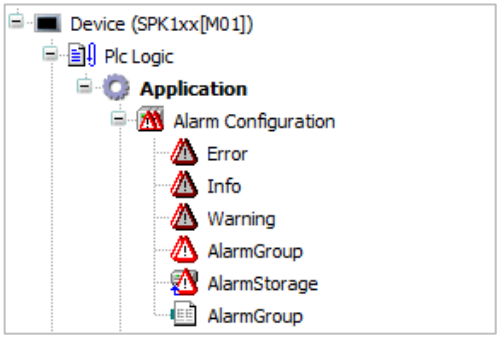


Рисунок 6.6.25 – Подобъекты конфигурации тревог после добавления группы тревог

Теперь конфигурация тревог включает в себя следующие объекты:

- классы тревог **Error**, **Info** и **Warning** – они позволяют настроить тип подтверждения тревоги (если тревога должна подтверждаться – «квитироваться» – пользователем визуализации), цвет сообщения тревоги в таблице тревог и т. д. Каждая тревога принадлежит конкретному классу. Пользователь может создать произвольное количество классов тревог;
- группа тревог **AlarmGroup** – в ней настраиваются сами тревоги (выбираются условия их срабатывания, задаются классы, тексты сообщений и т. д.). В группе может быть произвольное число тревог, и пользователь может создать произвольное число групп. Вместе с группой автоматически создается список текстов, в котором хранятся тексты сообщений, отображаемых в таблице тревог;
- хранилище **AlarmStorage**, определяющее параметры сохранения истории тревог в памяти контроллера.

В нашем примере мы будем использовать единственный класс тревог – **Error**. В его настройках выберем способ подтверждения **REP_ACK** – в этом случае поддерживается квитирование тревог, при этом квитирование возможно только в том случае, если условие тревоги перестало выполняться. Установим галочку **Архивация**, чтобы тревоги сохранялись в памяти контроллера и отображались в истории тревог.

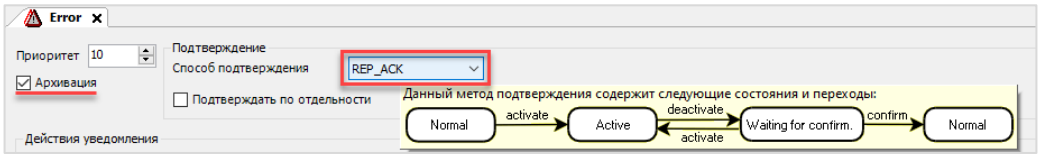


Рисунок 6.6.26 – Настройка класса тревог Error

В настройках группы тревог создадим нужные тревоги. В структуре **MEASURE_POINT** есть переменная **xAlarm**, представляющая сигнал тревоги. Выберем способ наблюдения – **Дискретный** и в столбце **Детали** в качестве условия пропишем эти переменные из соответствующих им элементов массива структур. В столбце **Класс** для всех тревог укажем класс **Error**. В столбце **Сообщение** укажем для каждой тревоги текст сообщения, которое будет отображаться в таблице тревог. В параметре **Архивация** укажем хранилище тревог **AlarmStorage**.

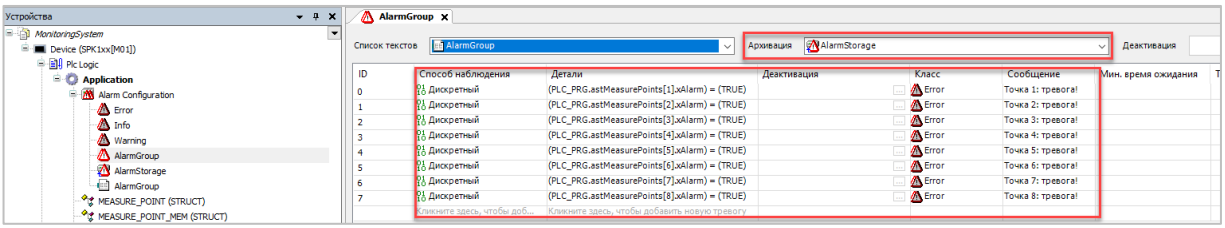


Рисунок 6.6.27 – Настройка группы тревог AlarmGroup

На рис. 6.6.27 виден столбец **Мин. время ожидания** – он позволяет настроить задержку активации тревоги, которую мы реализовали в коде программы с помощью таймера. В принципе, оба варианта легитимны; преимуществом организации задержки в коде является то, что при необходимости это позволяет изменять значение задержки (например, через визуализацию). В столбце **Мин. время ожидания** значения задержек указываются в виде констант, и изменить их в процессе работы приложения уже не получится.

В хранилище тревог **AlarmStorage** установите ограничение на размер файла тревог – по количеству записей или по размеру файла. В случае превышения ограничения самые старые записи файла начнут перезаписываться новыми (файл сохраняется в режиме циклического буфера). Если не установить ограничение (оставить режим «**No limit**»), то рано или поздно файл займет всю память контроллера, что приведет к невозможности его корректной работы.

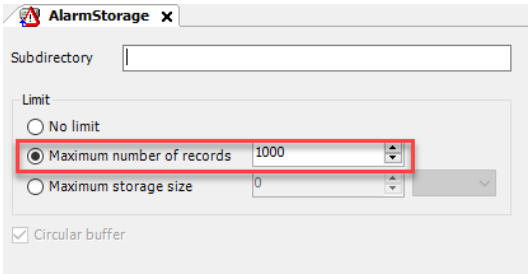


Рисунок 6.6.28 – Настройка хранилища тревог AlarmStorage

На этом настройка конфигурации тревог завершена – теперь можно переходить к созданию таблицы тревог.

6.6.5 Настройка экрана visuAlarmTable

Добавьте на экран **visuAlarmTable** элемент **Таблица тревог** из вкладки **Менеджер тревог**.

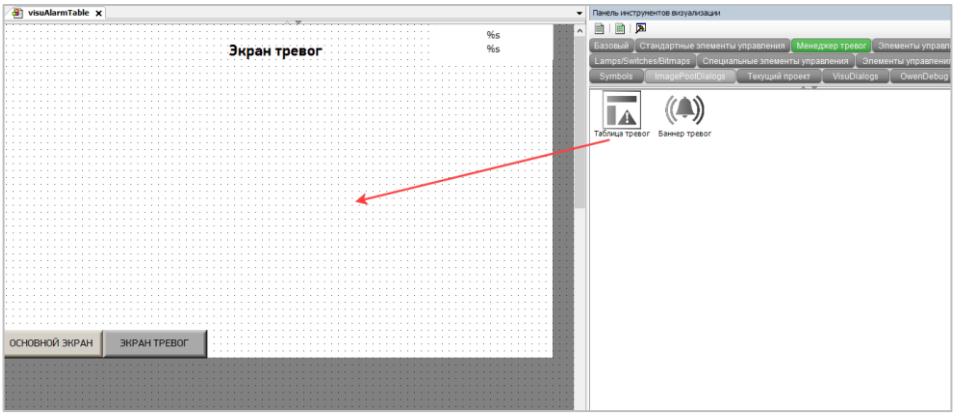


Рисунок 6.6.29 – Добавление элемента Таблица тревог

По умолчанию таблица содержит 2 столбца. В ее свойствах откройте группу свойств **Столбцы** и нажмите кнопку **Создать новый**.

- в первом столбце будет отображаться метка времени тревоги;
- во втором столбце – текст сообщения тревоги;
- в третьем столбце – метка времени подтверждения тревоги.

Задайте для столбцов заголовок, ширину и тип данных (тип данных определяет, какие именно данные будут отображаться в столбце). ***Обратите внимание***, что желательно сделать ширину таблицы (свойство **Позиция/Ширина**) равной сумме ширины столбцов и ширины столбца заголовка, в котором отображаются номера строк таблицы (свойство **Общая конфигурация таблицы/Ширина строки заголовков**). Если этого не сделать – то у таблицы будет отображаться полоса прокрутки по оси X.

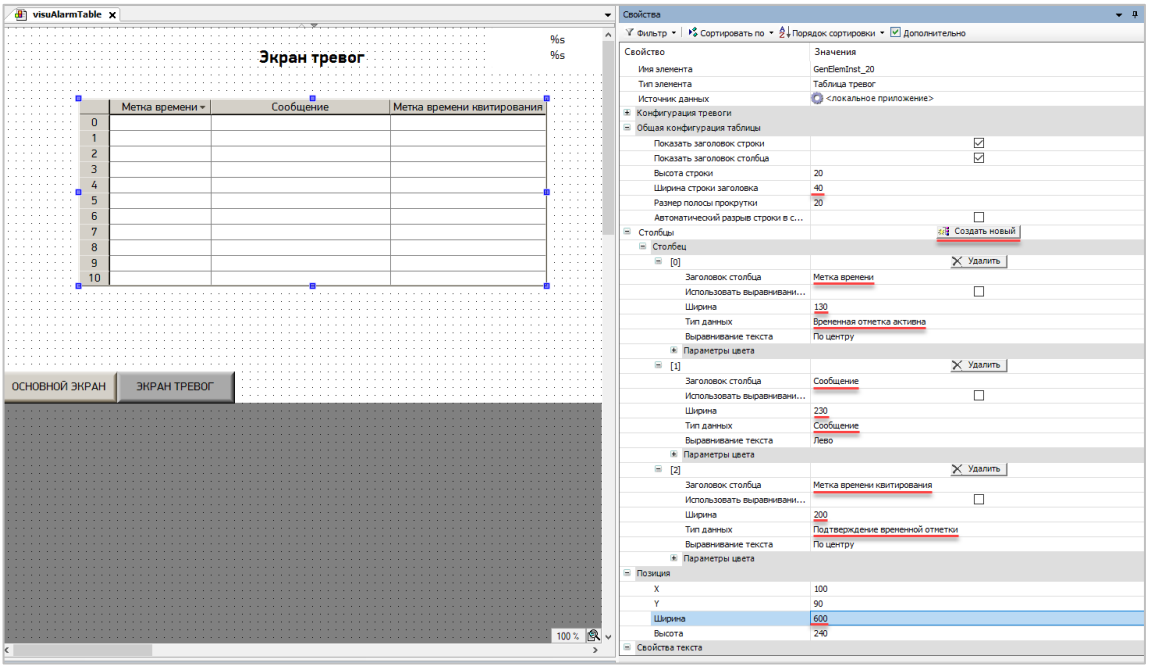


Рисунок 6.6.30 – Настройки таблицы тревог

Теперь добавим кнопки для подтверждения тревог и просмотра истории тревог. Нажмите правой кнопкой мыши на таблицу и используйте команду **Вставить элементы для подтверждения тревог**. Всего доступно 4 элемента:

- **Подтверждение выделенного** – кнопка подтверждения выделенной пользователем тревоги;
- **Подтверждение всех видимых** – кнопка подтверждения всех тревог таблицы;
- **История** – кнопка перехода в режим истории;
- **Фиксация положения** – кнопка «фиксации» таблицы в режиме просмотра истории тревог. По умолчанию при появлении новой тревоги в режиме истории происходит переход к первой строке таблицы, что может быть неудобно; в режиме фиксации при появлении новой тревоги перехода к первой строке не происходит.

Добавим первые 3 элемента – кнопку фиксации создавать не будем.

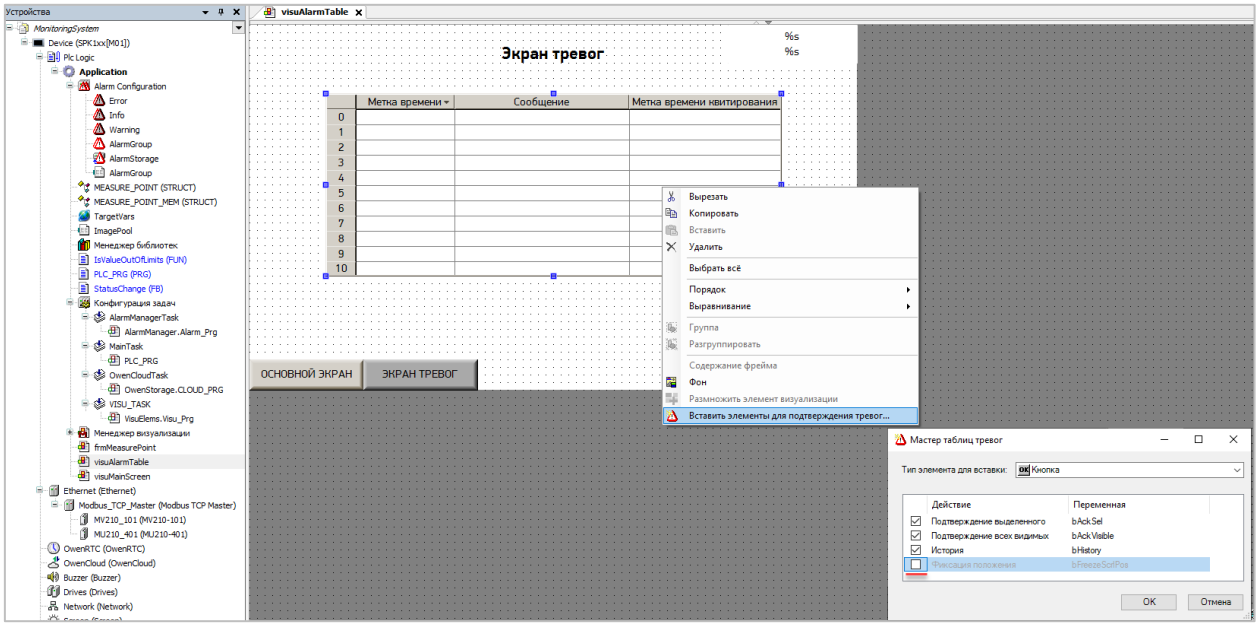


Рисунок 6.6.31 – Добавление кнопок подтверждения тревог

Настраивать кнопки не нужно – при их создании в интерфейсе экрана автоматически будут объявлены локальные переменные, которые также автоматически будут привязаны к кнопкам и к свойствам таблицы в группе свойств **Управляющие переменные**. Вам достаточно изменить только текст кнопок и, при желании, их внешний вид.

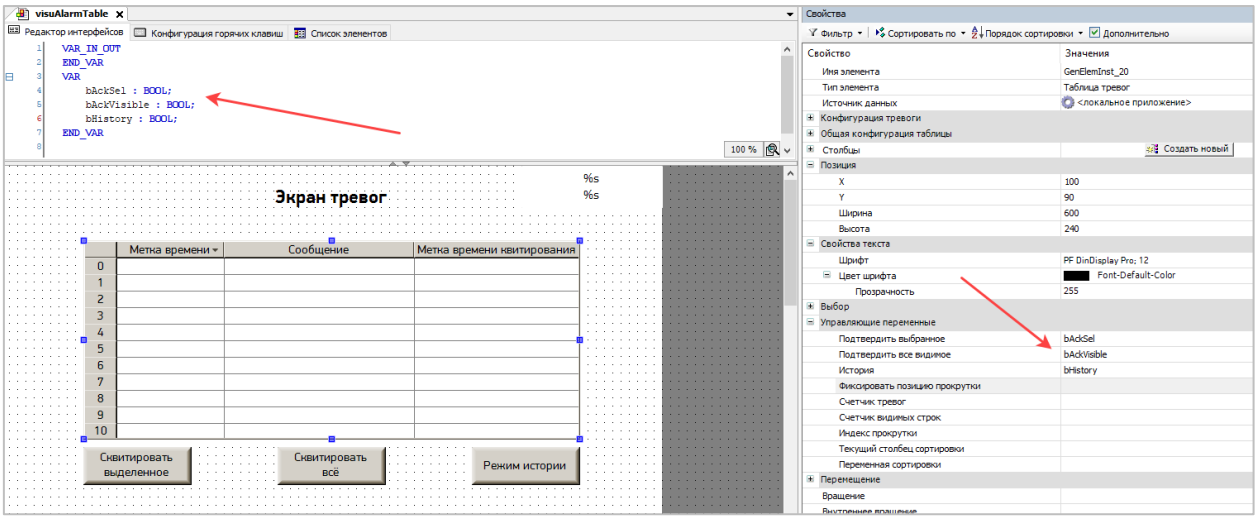


Рисунок 6.6.32 – Интерфейсные переменные для кнопок подтверждения

В п. 6.6.1 мы упоминали, что в момент объявления переменных экран становится фреймом и не может использоваться как обычный экран визуализации. Но уже тогда мы отметили, что речь идет о входных переменных (**VAR_INPUT**) и входах-выходах (**VAR_IN_OUT**). Объявление локальных переменных (**VAR**) не делает экран фреймом.

Локальные переменные экранов можно использовать в тех случаях, когда эти переменные не нужны вам за пределами визуализации (то есть вы не собираетесь использовать их в программе) – как, например, в рассмотренном нами случае с переменными кнопок подтверждения. В случае необходимости мы могли бы объявить эти переменные в программе и привязать их к свойствам таблицы – например, если бы нам требовалась возможность квитирования тревог по команде от другого контроллера или панели оператора.

Настройка экрана **visuAlarmTable** завершена – теперь давайте проверим, как работает наш пример, в котором программная и визуальная часть полностью готовы.

6.7 Проверка примера

Если вы эмулируете модули с помощью Modbus Universal MasterOPC Server – то не забудьте сначала его запустить (см. рис. 6.3.2). Задайте в нем значения для аналоговых входов – пусть все они будут в пределах допустимого диапазона **10…30** (мы задали этот диапазон при создании структуры **MEASURE_POINT_MEM** в п. 6.5.1).

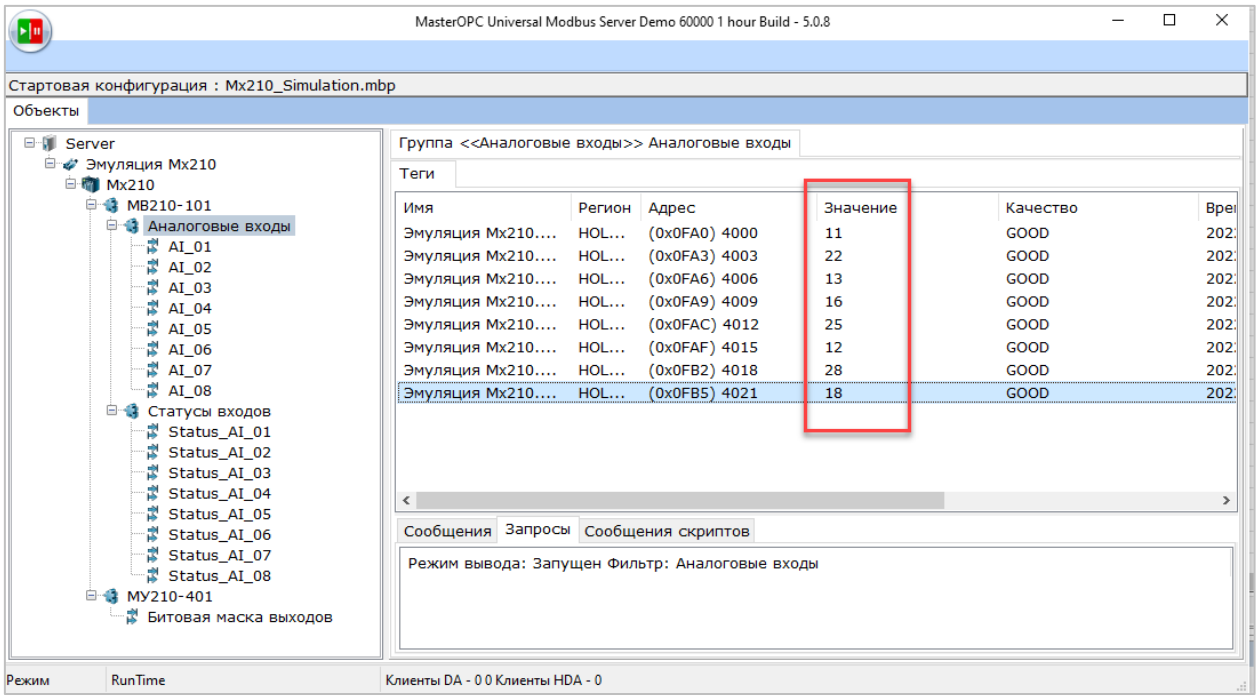


Рисунок 6.7.1 – Установка значений аналоговых входов модуля MB210-101 в OPC-сервере

```
MEASURE_POINT_MEM
1  // Структура энергонезависимых параметров одной точки измерения системы автоматизации
2  TYPE MEASURE_POINT_MEM :
3  STRUCT
4      // Нижний допустимый предел значения
5      rLowAlarmValue:  REAL := 10.0;
6      // Верхний допустимый предел значения
7      rHighAlarmValue: REAL := 30.0;
8      // Экземпляр счетчика ошибок
9      fbAlarmCounter:  CTU;
10 END_STRUCT
11 END_TYPE
```

Рисунок 6.7.2 – Начальные значения границ допустимого диапазона были заданы при создании структуры MEASURE_POINT_MEM

Загрузите ваше приложение в контроллер и запустите его (используйте для этого ярлыки команд **Логин** и **Старт** на панели инструментов).

Перейдите в web-браузер и введите следующий адрес:

<IP-адрес контроллера>:8080

где **8080** – это порт web-визуализации по умолчанию.

Если вы используете виртуальный контроллер, то введите **localhost:8080**

Откроется основной экран визуализации:

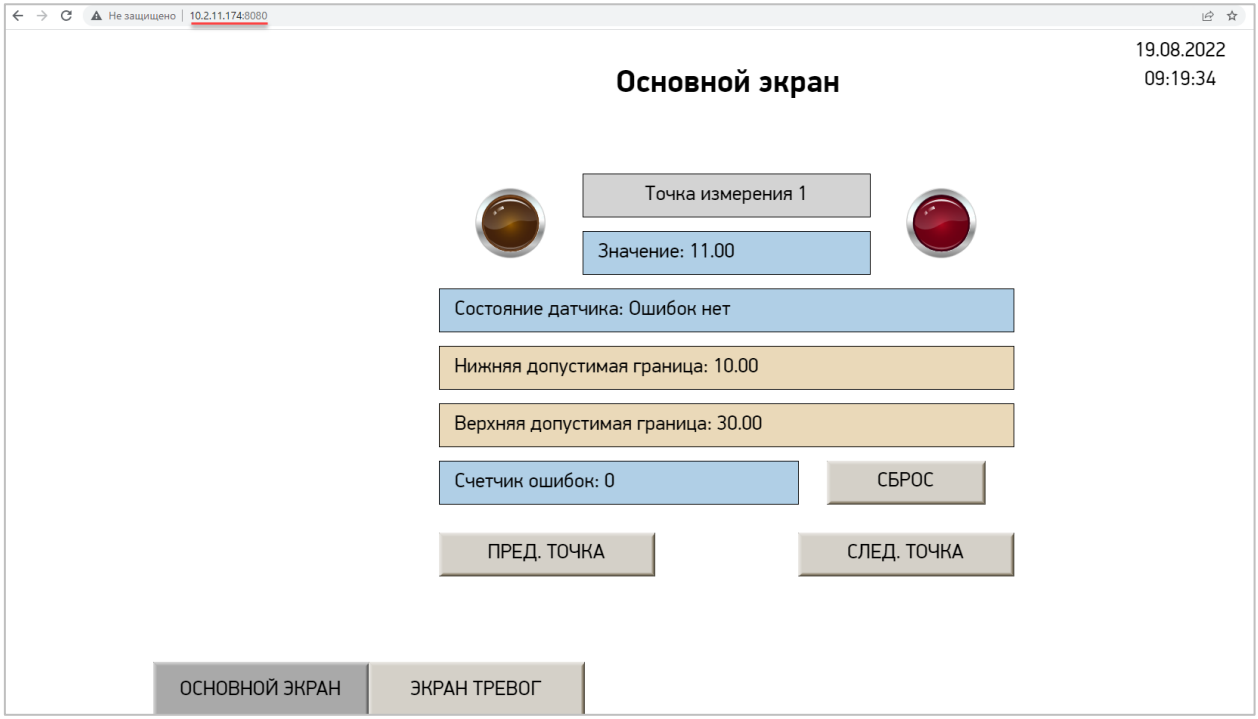


Рисунок 6.7.3 – Отображение основного экрана в web-визуализации контроллера

С помощью кнопок «СЛЕД. ТОЧКА» и «ПРЕД. ТОЧКА» просмотрите информацию по всем точкам измерения.

Для изменения допустимых границ нажмите левой кнопкой мыши на поле соответствующей границы, введите новое значение и нажмите **Enter**.

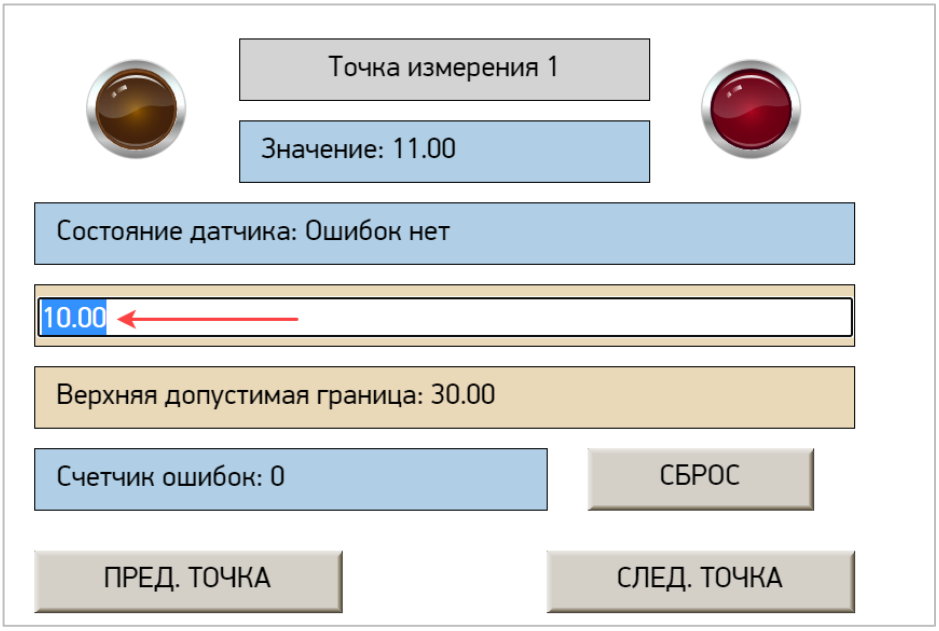


Рисунок 6.7.4 – Ввод значений допустимых границ

Давайте сэмулируем ошибку первой точки измерения. Введите в OPC-сервере значение, которое выходит за допустимую границу. Сразу загорится желтый индикатор (сигнал выхода за границу), а спустя 5 секунд – красный (сигнал тревоги). Счетчик ошибок увеличится на 1. Для обнуления счетчика нажмите на кнопку **Сброс**. Чтобы проверить изменение сообщения о состоянии датчика – нужно изменить в OPC-сервере его статус, задав для него особое значение (см. п. 6.4.4).



Рисунок 6.7.5 – Эмуляция ошибки точки измерения

Перейдите на экран тревог. В таблице тревог будет отображаться сообщение о сэмулированной нами ошибке.

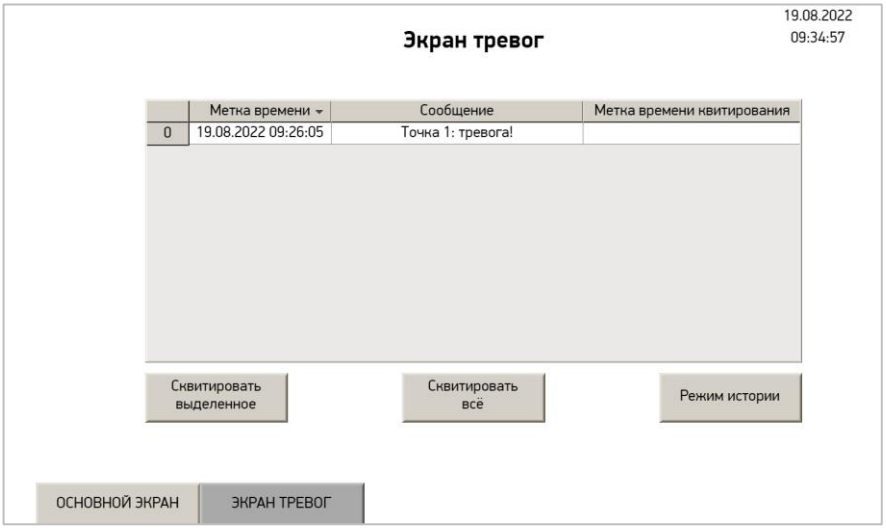


Рисунок 6.7.6 – Отображение сообщения активной тревоги в таблице тревог

Верните значение первой точки измерения в допустимый диапазон, чтобы деактивировать тревогу и сквитируйте ее (выделите строку тревоги в таблице и нажмите кнопку **Сквитировать выделенное** или просто нажмите кнопку **Сквитировать всё**). После этого тревога пропадет из таблицы. Переключите таблицу в режим просмотра истории – информация о тревоге (с меткой времени квитирования) теперь будет отображаться там.

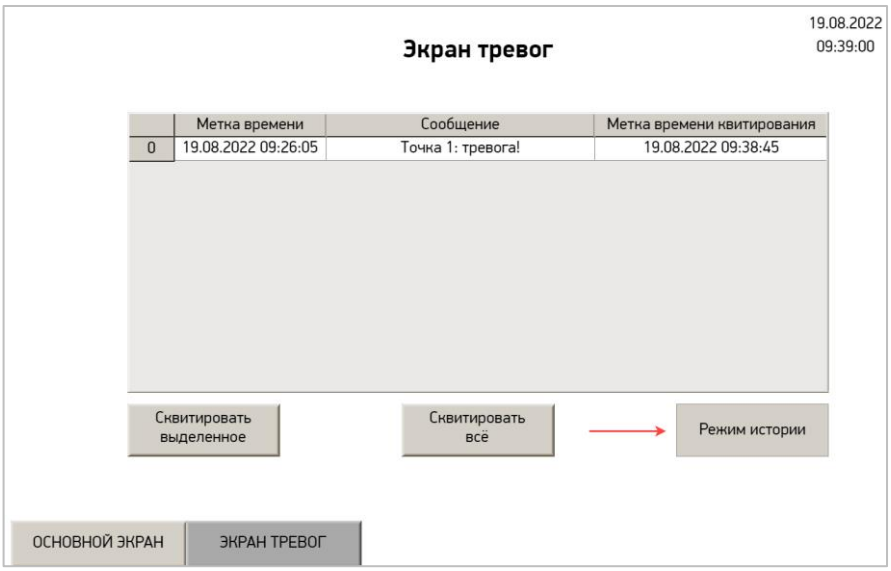


Рисунок 6.7.7 – Отображение истории тревог

Обратите внимание, что системное время, отображаемое в правом верхнем углу экранов визуализации, не будет отображаться при использовании виртуального контроллера – потому что в дереве его проекта нет системного узла **OwenRTC**, который мы использовали для получения системного времени.

Мы реализовали основную часть нашего примера. В следующих пунктах мы рассмотрим, как настроить архивацию данных и подключить контроллер к SCADA-системе и облачному сервису OwenCloud.

6.8 Настройка архивации

Одной из типовых задач программируемых контроллеров является архивирование данных о технологическом процессе для последующей обработки и анализа (например, для анализа причин аварийных ситуаций и оптимизации режима работы оборудования).

В данном пункте мы рассмотрим, как организовать архивацию данных в контроллерах ОВЕН с использованием компонента **OwenArchiver**.

6.8.1 Установка пакета архиватора

Пакет архиватора (**OwenArchiver**) можно загрузить на сайте ОВЕН в разделе CODESYS V3/Библиотеки и компоненты.

Установка пакета выполняется с помощью утилиты **CODESYS Installer** (**Инструменты – CODESYS Installer**) – мы уже рассматривали этот вопрос в п. 2.2.

6.8.2 Добавление и настройка архиватора

Для добавления архиватора в проект CODESYS нажмите правой кнопкой мыши на узел **Device**, используйте команду **Добавить устройство** и в папке **Разн.** выберите компонент **OwenArchiver.**

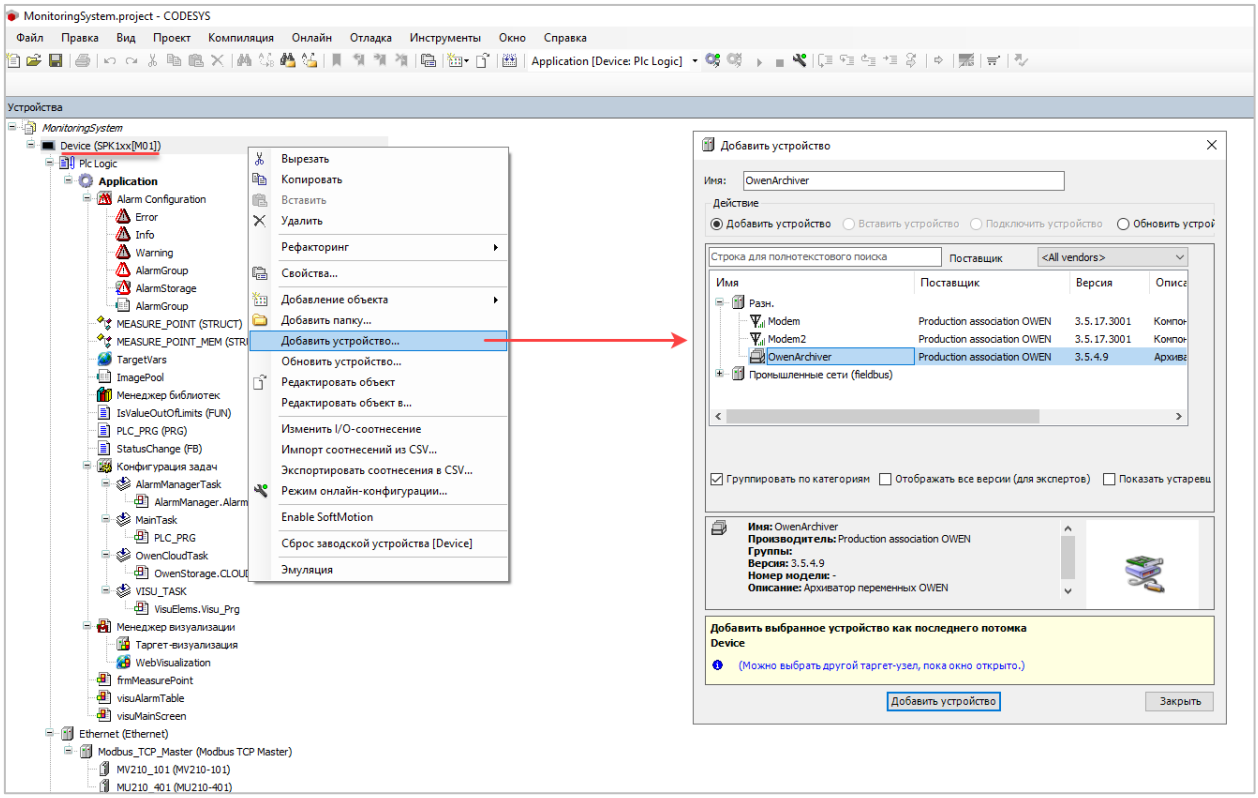


Рисунок 6.8.1 – Добавление архиватора в проект CODESYS

Обратите внимание, что при добавлении в проект архиватора в конфигурацию задач была автоматически добавлена задача **OwenArchiver**. Эту задачу не следует удалять (иначе возникнут ошибки компиляции) и не рекомендуется редактировать (то есть изменять ее приоритет и интервал вызова).

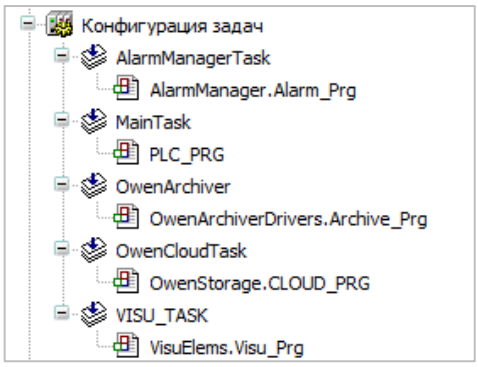


Рисунок 6.8.2 – Список задач проекта после добавления архиватора

Теперь добавим в компонент каналы архивируемых переменных. Для этого нажмите правой кнопкой мыши на узел **OwenArchiver**, используйте команду **Добавить устройство** и добавьте 8 каналов типа **REAL** (по числу точек измерения).

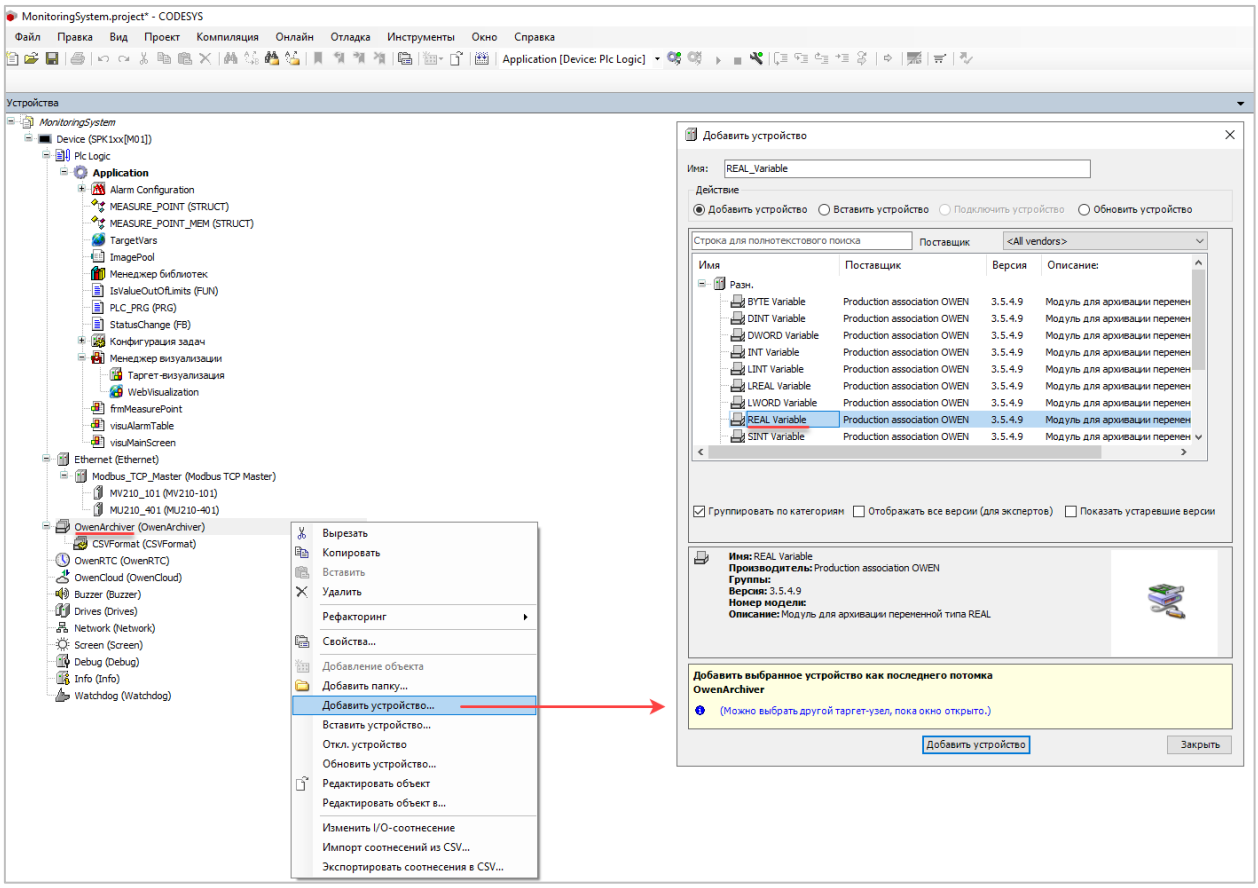


Рисунок 6.8.3 – Добавление каналов архивирования

Приступим к настройке архиватора.

Перейдите в компонент **OwenArchiver** и откройте вкладку **Конфигурация**. На этой вкладке задаются основные параметры архиватора – период архивации, максимальный размер файла архива, режим архивирования (циклически/по команде/по изменению переменной) и т. д. ***Обратите внимание*** на параметр **Устройство для ведения архива**. Архиватор позволяет сохранять файлы не только в памяти контроллера, но и на USB- или SD-накопителе. Но если вы запускаете проект на виртуальном контроллере – то учтите, что для него поддерживается только устройство **Директория CODESYS**.

В рамках примера оставим все параметры в значениях по умолчанию.

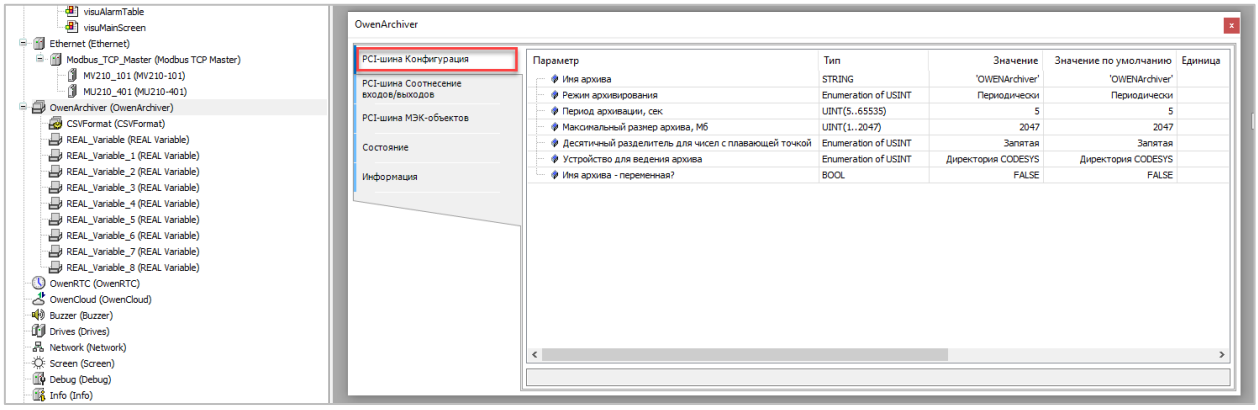


Рисунок 6.8.4 – Параметры конфигурации архиватора

На вкладке **Соотнесение входов-выходов** доступны каналы управления и диагностики архиватора, к которым можно привязать переменные проекта. Самый важный канал – канал запуска архиватора; чтобы архиватор работал – этот канал должен иметь значение **TRUE**. Объявим в программе **PLC_PRG** переменную **xEnableArchiver** типа **BOOL** и присвоим ей начальное значение **TRUE** – тогда архиватор будет автоматически запускаться после запуска приложения.

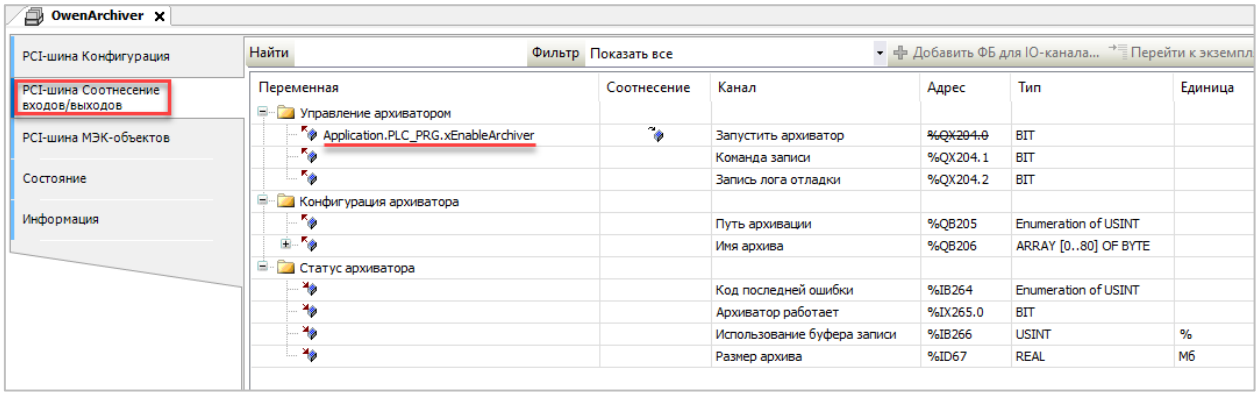


Рисунок 6.8.5 – Привязка переменной к каналу запуска архиватора

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:         ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                        INT;
   xInit:                    BOOL;
   iCurrentFrameIndex:       INT    := 1;
   xEnableArchiver:          BOOL   := TRUE;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:    INT   := 8;
   c_tAlarmDelay:            TIME  := T#5S;
END_VAR
```

В узле **CSVFormat** определяется структура архива. Доступно две основных структуры – непрерывный архив (весь архив представляет собой один файл) и суточный архив (каждую полночь создается новый файл архива, при этом для каждого года создается отдельная директория, в которой создаются директории с названиями месяцев и дней). Оставим структуру архива по умолчанию – **Год/Месяц/День**.

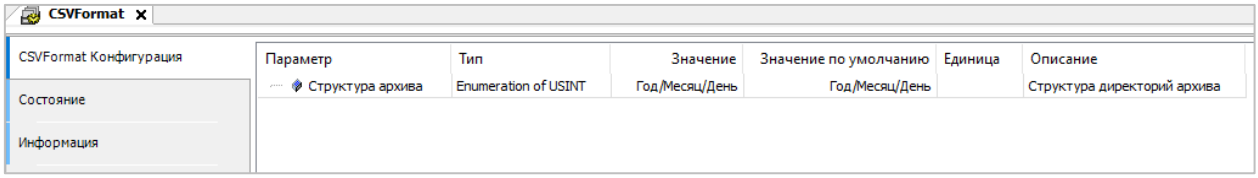


Рисунок 6.8.6 – Выбор структуры архива

Осталось настроить каналы архивации. Перейдите на вкладку **Конфигурация** первого канала и введите описание переменной (оно будет отображаться в строке заголовка файла архива) и число знаков после плавающей запятой, которое сохранится у записываемой в архив переменной.

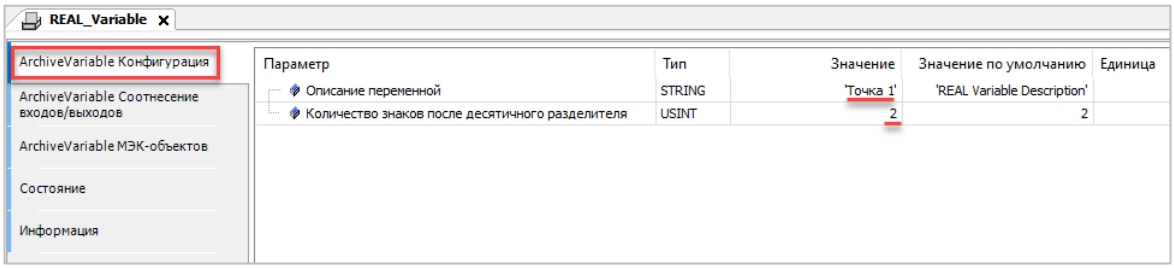


Рисунок 6.8.7 – Параметры конфигурации первого канала архивации

На вкладке **Соотнесение входов-выходов** привяжите к каналу архивации нужную переменную. В нашем случае это переменная **rValue** из массива структур **astMeasurePoints**. Для первого канала укажите индекс массива **1**, для второго – **2** и т.д. Настройте все 8 каналов архивации.

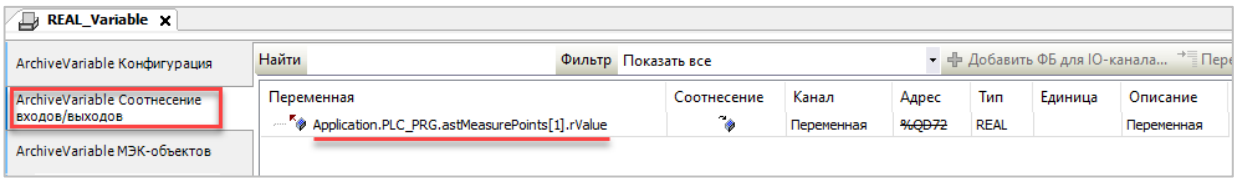


Рисунок 6.8.8 – Привязка переменной к первому каналу архивации

6.8.3 Проверка архивации

Для проверки архивации загрузите ваше приложение в контроллер и запустите его (используйте для этого ярлыки команд **Логин** и **Старт** на панели инструментов).

Если вы используете контроллер ОВЕН – то установите утилиту WinSCP (мы используем ее в рамках примера; в реальных проектах вы можете выгружать архивы по протоколу FTP, через web-визуализацию и другими способами). Запустите утилиту и на вкладке **Новое подключение** введите следующие параметры соединения:

- протокол – **SCP**;
- порт – **22**;
- имя хоста – IP-адрес вашего контроллера;
- имя пользователя – **root**;
- пароль – по умолчанию **owen** (пароль совпадает с паролем web-конфигуратора).

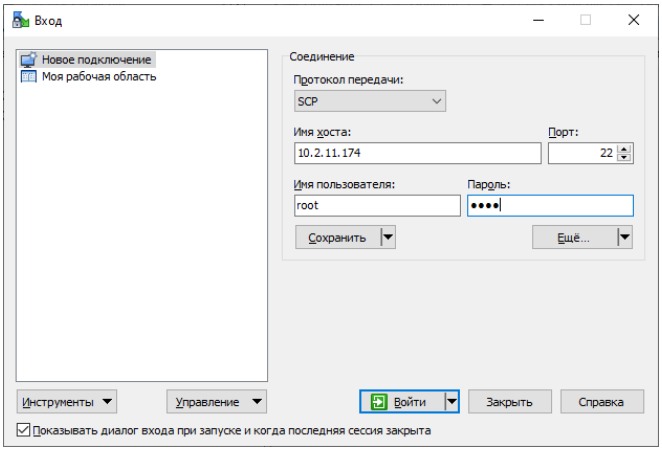


Рисунок 6.8.9 – Ввод параметров соединения в утилите WinSCP

Нажмите кнопку **Войти**. Откроется файловый менеджер, позволяющий работать с файлами в памяти контроллера. Если появится окно с необходимостью подтвердить связь с неизвестным сервером – нажмите в нем **Да**.

Напомним, что в настройках архиватора мы выбрали устройство **Директория CODESYS** (см. рис. 6.8.4). Поэтому перейдем в нее, используя путь **/home/root/CODESYS_WRK/PlcLogic** (для ПЛК2xx путь другой – **/root/CODESYS/PlcLogic**), и откроем папку **OwenArchiver** (название папки совпадает с именем архива, указанным в параметрах архиватора). Далее откроем папку с номером текущего года, а в ней – с номером текущего месяца. Здесь мы и увидим наш файл архива. Скопируйте его на свой ПК и откройте в **Microsoft Excel** или другом аналогичном ПО.

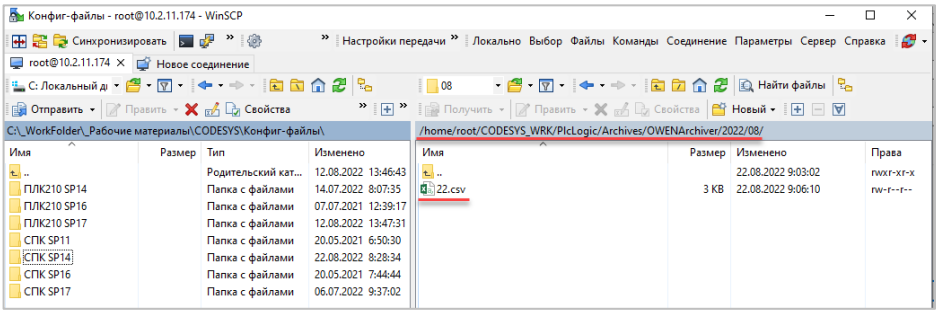
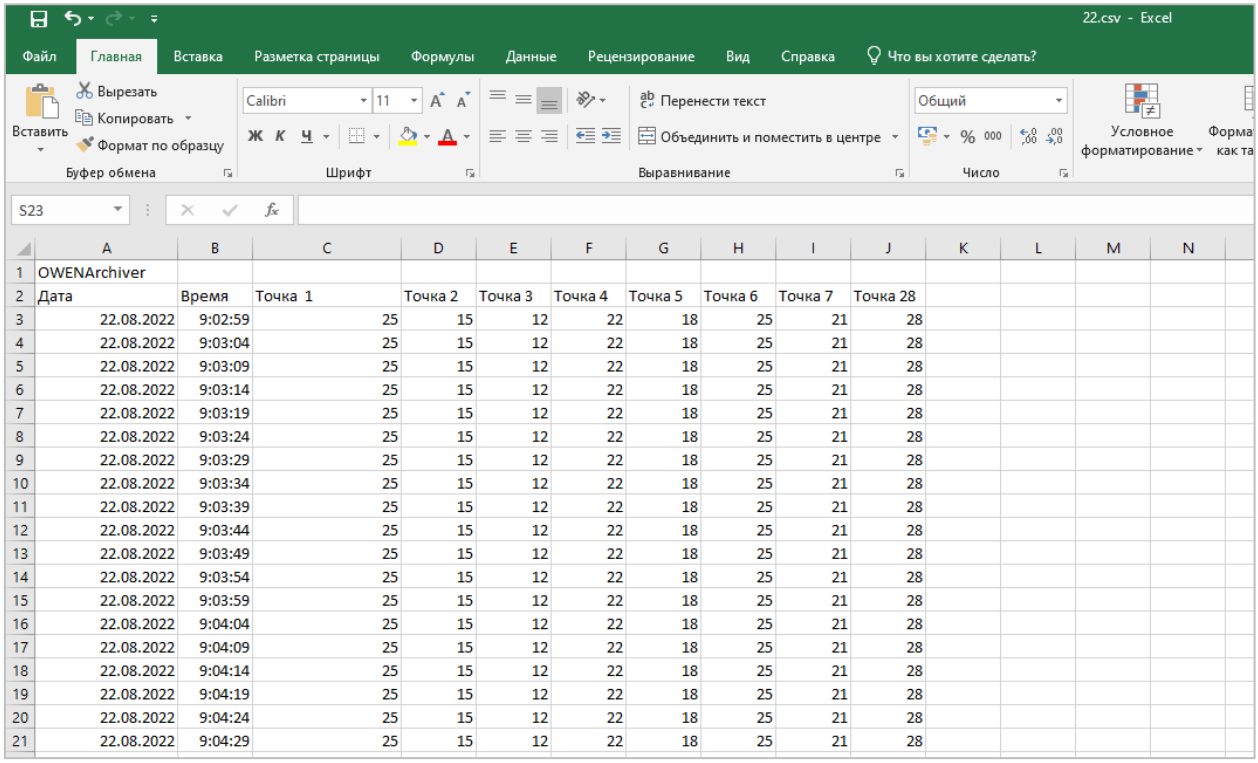


Рисунок 6.8.10 – Файл архива в контроллере



	A	B	C	D	E	F	G	H	I	J
1	OWENArchiver									
2	Дата	Время	Точка 1	Точка 2	Точка 3	Точка 4	Точка 5	Точка 6	Точка 7	Точка 28
3	22.08.2022	9:02:59	25	15	12	22	18	25	21	28
4	22.08.2022	9:03:04	25	15	12	22	18	25	21	28
5	22.08.2022	9:03:09	25	15	12	22	18	25	21	28
6	22.08.2022	9:03:14	25	15	12	22	18	25	21	28
7	22.08.2022	9:03:19	25	15	12	22	18	25	21	28
8	22.08.2022	9:03:24	25	15	12	22	18	25	21	28
9	22.08.2022	9:03:29	25	15	12	22	18	25	21	28
10	22.08.2022	9:03:34	25	15	12	22	18	25	21	28
11	22.08.2022	9:03:39	25	15	12	22	18	25	21	28
12	22.08.2022	9:03:44	25	15	12	22	18	25	21	28
13	22.08.2022	9:03:49	25	15	12	22	18	25	21	28
14	22.08.2022	9:03:54	25	15	12	22	18	25	21	28
15	22.08.2022	9:03:59	25	15	12	22	18	25	21	28
16	22.08.2022	9:04:04	25	15	12	22	18	25	21	28
17	22.08.2022	9:04:09	25	15	12	22	18	25	21	28
18	22.08.2022	9:04:14	25	15	12	22	18	25	21	28
19	22.08.2022	9:04:19	25	15	12	22	18	25	21	28
20	22.08.2022	9:04:24	25	15	12	22	18	25	21	28
21	22.08.2022	9:04:29	25	15	12	22	18	25	21	28

Рисунок 6.8.11 – Содержимое файла архива

Если вы используете виртуальный контроллер – то файл будет создан на ПК в директории **C:\ProgramData\CODESYS\CODESYSControlWinV3\<идентификатор виртуального контроллера>\PlcLogic\Archives\OWENArchiver\<год>\<месяц>**

Идентификатор виртуального контроллера – это произвольная последовательность символов и цифр. Если у вас установлено несколько разных версий CODESYS и виртуальных контроллеров, то ориентируйтесь на дату изменения – она должна соответствовать текущей дате.

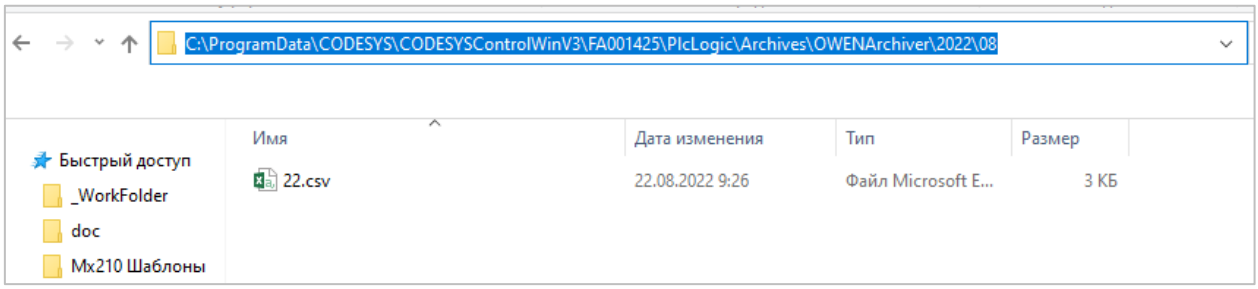


Рисунок 6.8.12 – Файл архива в директории виртуального контроллера

Мы рассмотрели только основы работы с архивами, не затронув вопросы сохранения архивов на USB- и SD-накопителе, выгрузки архивов по протоколу FTP и через web-визуализацию, и т. д. Подробная информация об архивировании приведена в документе **CODESYS V3.5. Архивация**, доступном для загрузки на сайте ОВЕН в разделе CODESYS V3/Документация.

6.9 Подключение контроллера к SCADA-системе

Достаточно часто требуется организовать подключение контроллера к верхнему уровню системы автоматизации – например, к SCADA-системе. Основные варианты решения этой задачи – использование протокола Modbus или технологий OPC DA / OPC UA. Наиболее простым и функциональным вариантом является использование OPC UA. Эта технология поддерживается большинством современных SCADA-систем. Наш контроллер будет выступать в роли OPC UA сервера, а SCADA-система – в роли OPC UA клиента.

В рамках нашего примера мы будем использовать SCADA-систему **MasterSCADA 3.11**, разработанную компанией МПС Софт (вы можете использовать и более современную версию этой SCADA – например, 3.12). Вам потребуется загрузить ее (подойдет любая демо-версия – с ограничением по числу тегов или с ограничением на время работы) и установить на свой ПК.

Для настройки контроллера в режиме OPC UA сервера добавьте в проект компонент **Символьная конфигурация**. В диалоге добавления компонента должна быть установлена галочка **Поддержка функций OPC UA**.

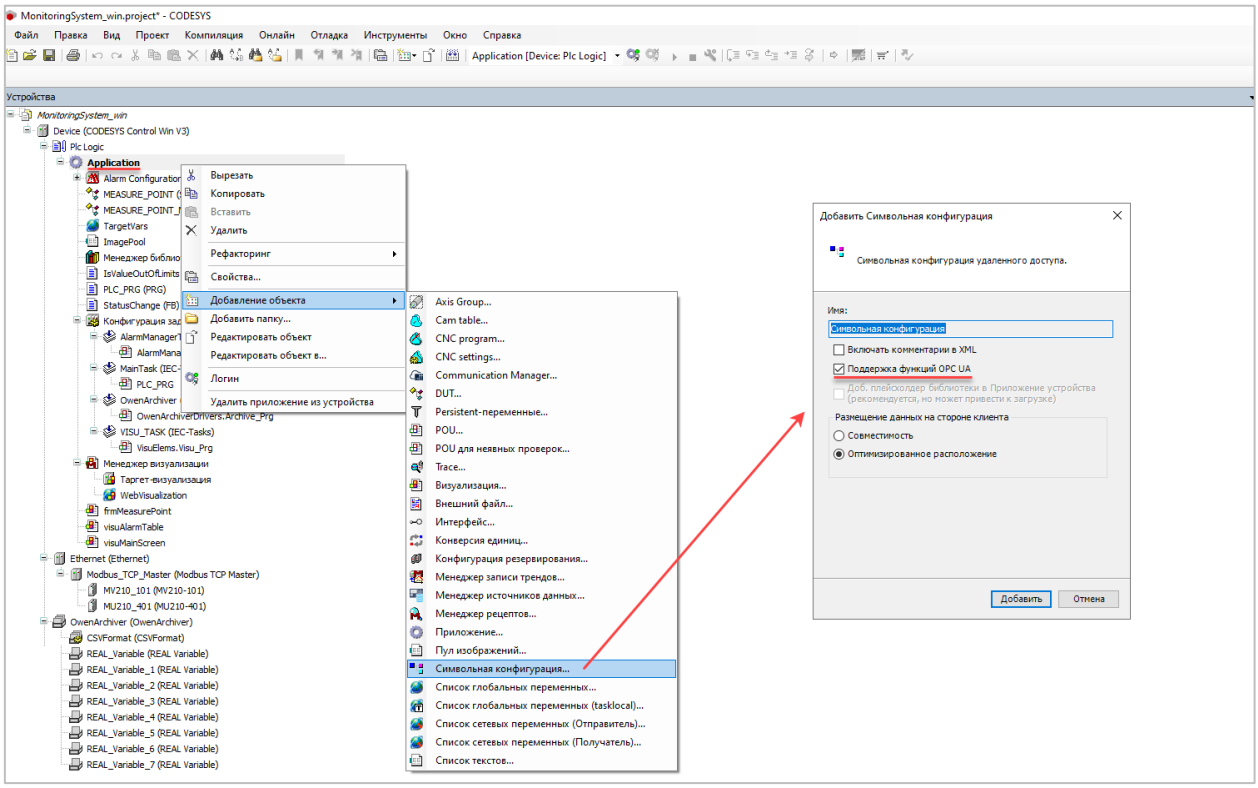


Рисунок 6.9.1 – Добавление в проект символьной конфигурации

Перейдите в редактор компонента и используйте команду **Компиляция** для обновления списка переменных контроллера.

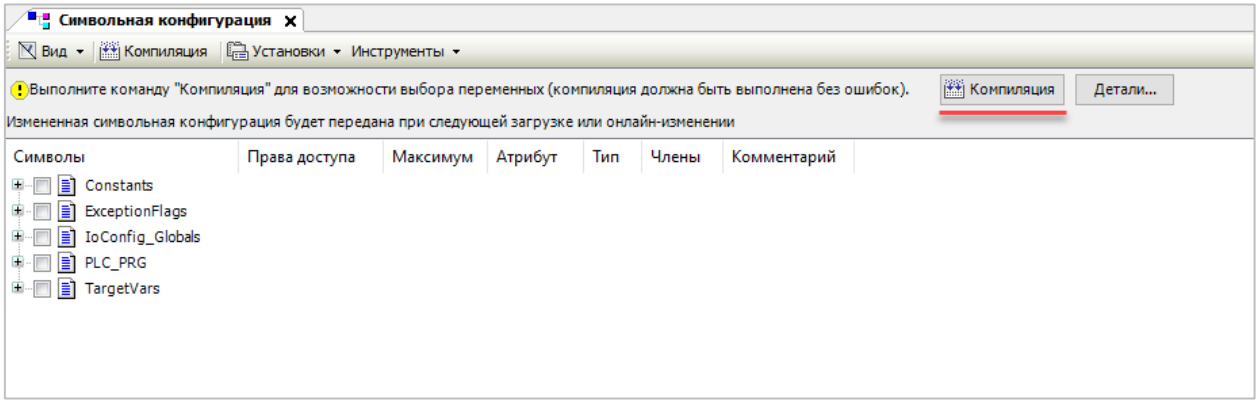


Рисунок 6.9.2 – Использование команды Компиляция для обновления списка переменных

После этого выделите переменные, которые должны быть доступны SCADA-системе, галочками. Мы будем передавать в SCADA оперативные данные по всем точкам измерения – поэтому достаточно установить галочку рядом с переменной **astMeasurePoints** программы **PLC_PRG**. Если требуется передавать не все переменные структуры, а лишь некоторые – то нажмите на кнопку «…» в столбце **Члены** и выделите нужные переменные структуры галочками. В этом же окне можно настроить права доступа для переменных – например, разрешить доступ только на чтение, чтобы SCADA не могла изменять значения переменных контроллера (по умолчанию для всех переменных контроллера установлен тип доступа **Чтение и запись**). Для «обычных» переменных (типа INT, REAL и т. д.) права доступа задаются непосредственно в списке переменных.

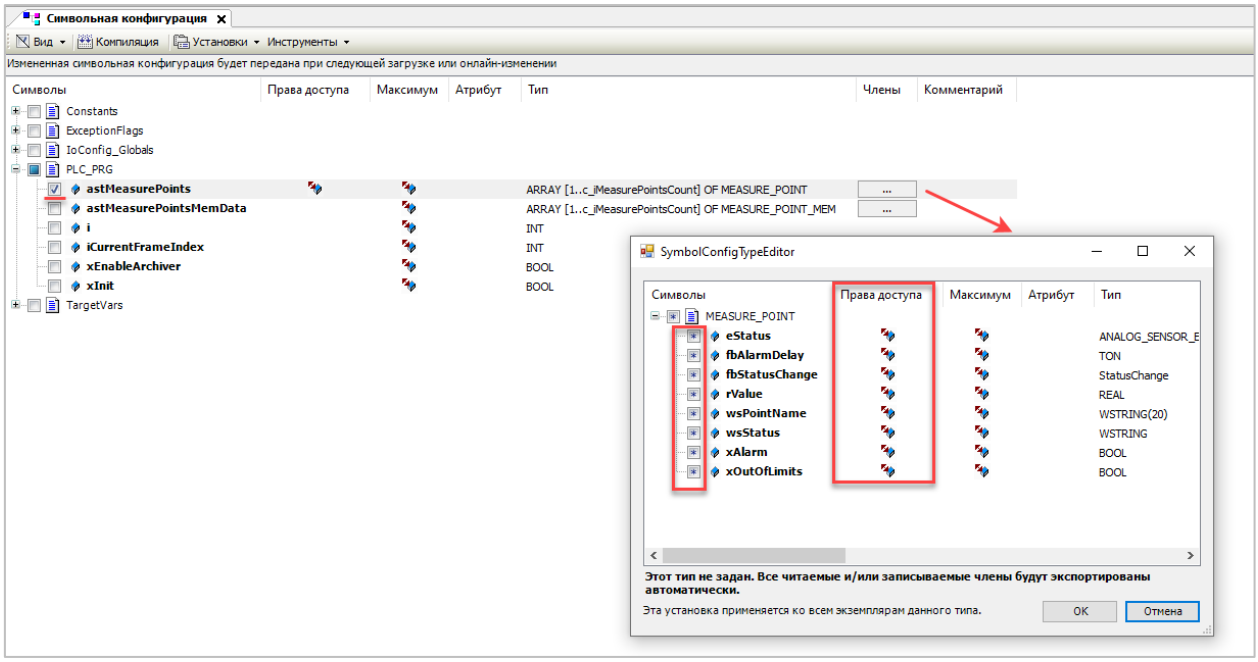


Рисунок 6.9.3 – Выбор переменных символьной конфигурации

Загрузите ваше приложение в контроллер и запустите его (используйте для этого ярлыки команд **Логин** и **Старт** на панели инструментов).

Теперь перейдем к настройке подключения к контроллеру в **MasterSCADA**. Запустите ее и создайте новый проект (**Проект – Создать**). Пароль доступа к проекту оставьте пустым.

Нажмите правой кнопкой мыши на узел **Система** и используйте команду **Вставить – Компьютер**.

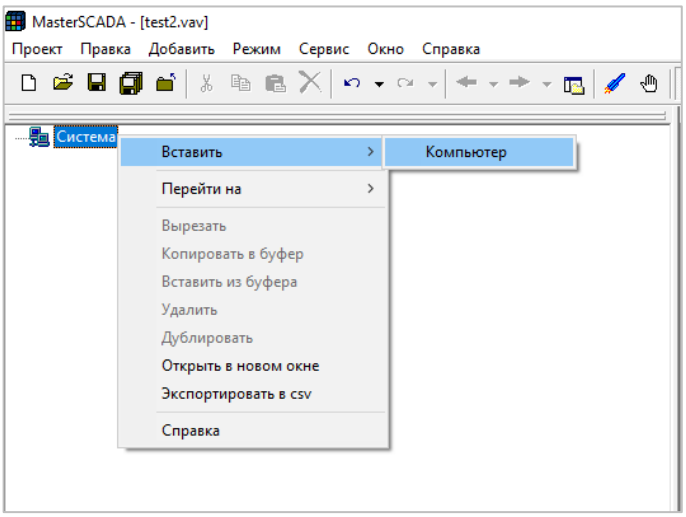


Рисунок 6.9.4 – Добавление узла Компьютер в проект MasterSCADA

Нажмите правой кнопкой на узел **Компьютер** и используйте команду **Вставить OPC UA сервер – OPC UA сервер**.

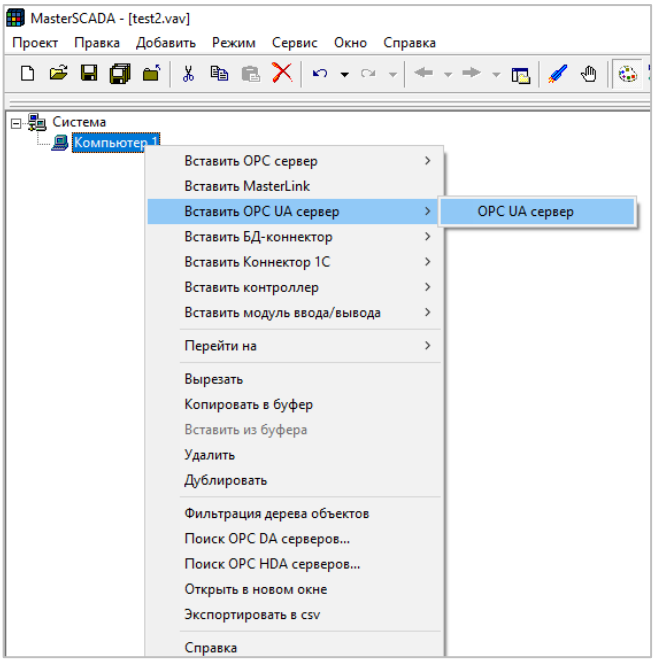


Рисунок 6.9.5 – Добавление узла OPC UA сервер

Перейдите на вкладку **Настройки OPC UA сервера** и нажмите кнопку **Настройки**.

В строке **Сервер** укажите IP-адрес вашего контроллера и порт **4840** (в формате **opc.tcp://<IP-адрес ПЛК>:4840**) – этот порт используется в CODESYS для подключения по OPC UA. Нажмите кнопку **Ок**.

В рамках примера мы используем незащищенное подключение к контроллеру; CODESYS позволяет защитить подключение по OPC UA с помощью логина/пароля или сертификата безопасности. Более подробная информация по этому вопросу приведена в данном видео.

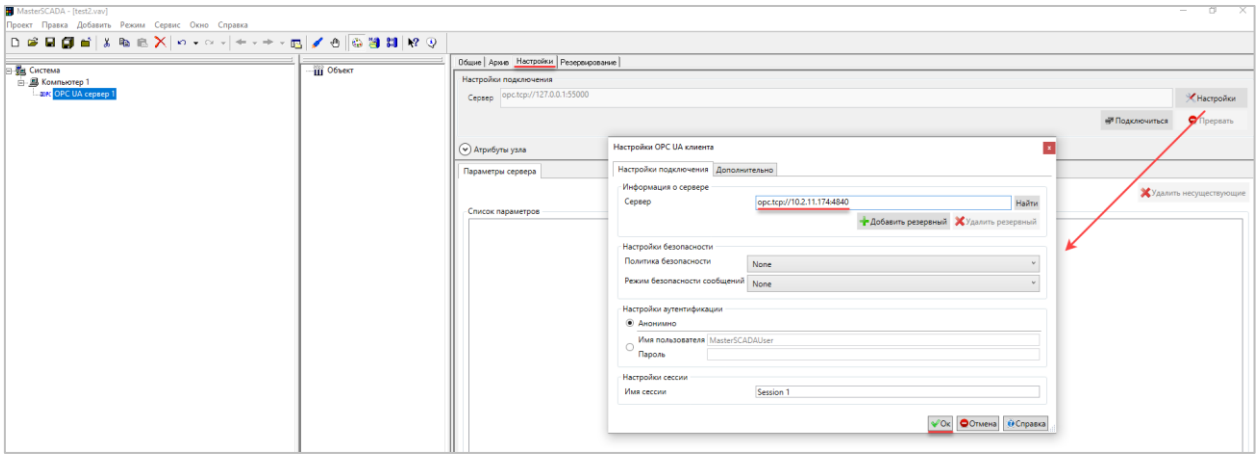


Рисунок 6.9.6 – Настройка подключения к OPC UA серверу

Нажмите кнопку **Подключиться**. Начнется выгрузка списка переменных из контроллера.

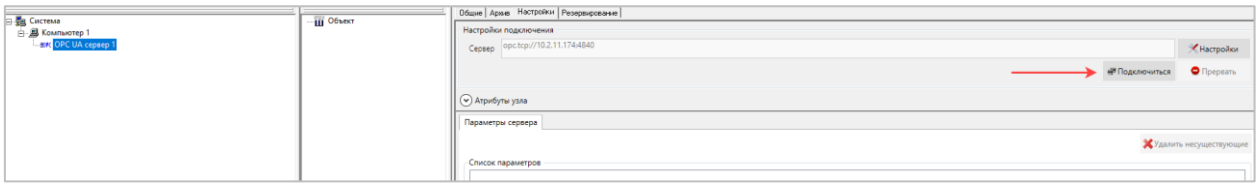


Рисунок 6.9.7 – Выгрузка списка переменных из контроллера

В дереве переменных выберите узел **Server**, в нем – узел, имя которого совпадает с моделью вашего контроллера, далее последовательно выделите узлы **Resources**, **Application** и **Programs**. В узле **Programs** вы увидите программу **PLC_PRG** из проекта **CODESYS**, а в ней – переменную **astMeasurePoints**. Раскройте каждый элемент массива структур и выделите галочками переменные структур, которые вы хотите увидеть в SCADA-системе. Необязательно выделять все переменные – например, можно оставить только **wsPointName**, **rValue**, **xOutOfLimits** и **xAlarm**. После этого нажмите кнопку **Применить**.

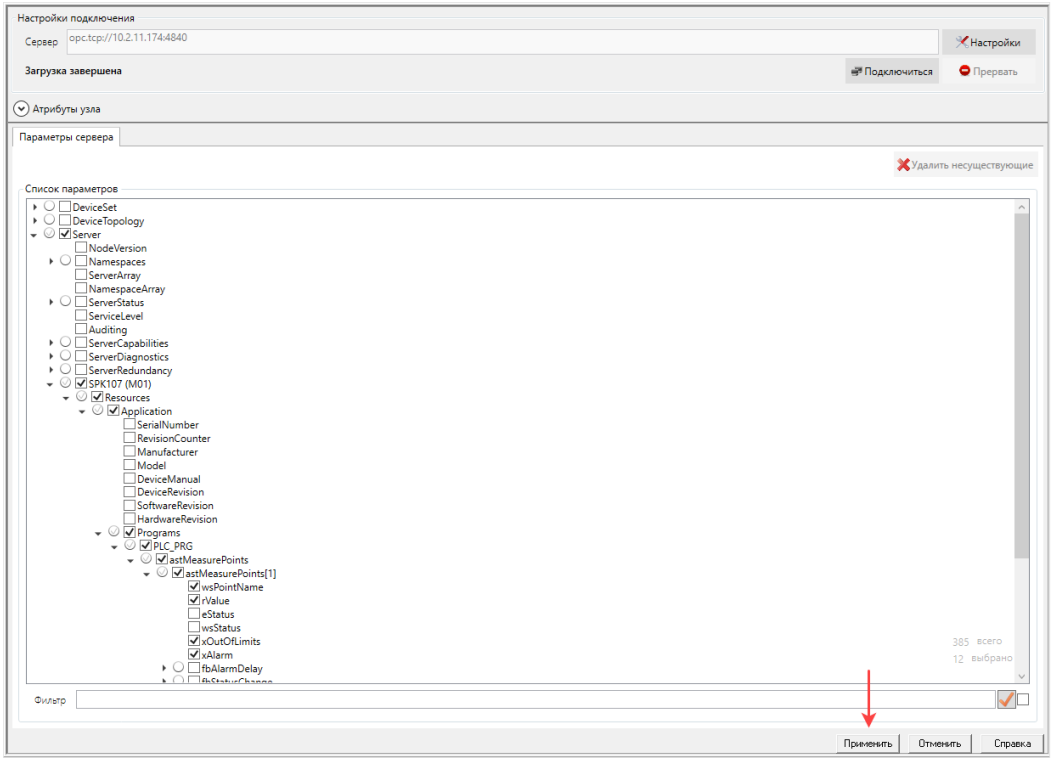


Рисунок 6.9.8 – Выбор переменных OPC UA сервера

В результате в дереве проекта будут добавлены выделенные вами переменные.

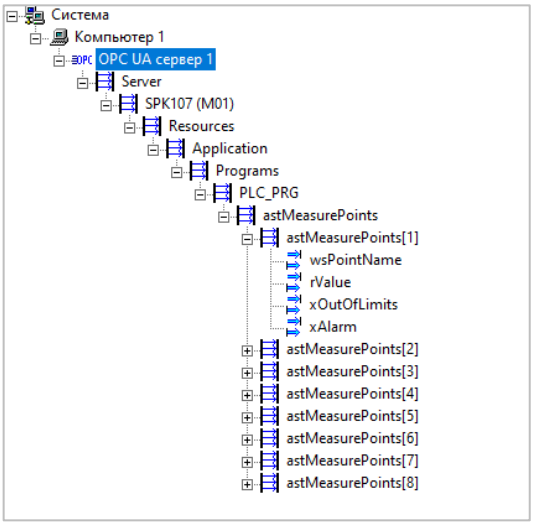


Рисунок 6.9.9 – Переменные контроллера в дереве проекта SCADA-системы

Запустите проект **MasterSCADA** в режиме исполнения с помощью кнопки «**Ракета**».

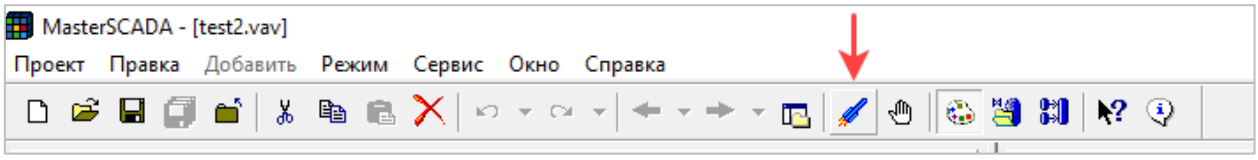


Рисунок 6.9.10 – Кнопка запуска проекта MasterSCADA в режиме исполнения

Теперь в дереве проекта рядом с переменными отобразятся их текущие значения. Вы можете использовать эти переменные на мнемосхемах проекта, сохранять их в архиве и т. д.

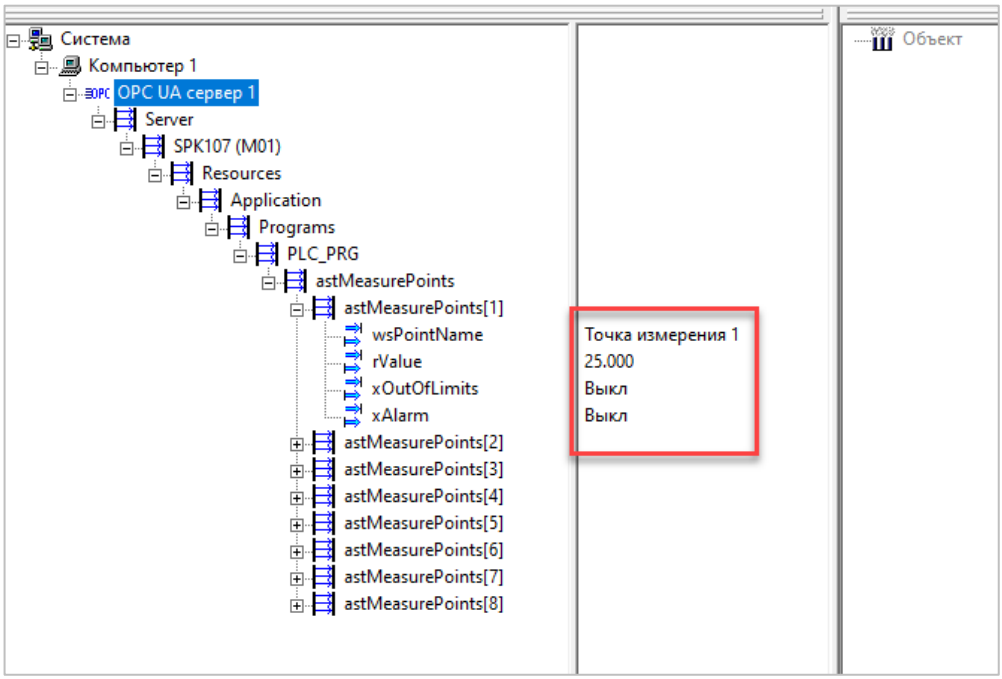


Рисунок 6.9.11 – Отображение текущих значений переменных

6.10 Подключение контроллера к облачному сервису OwenCloud

В прошлом пункте мы рассмотрели пример подключения контроллера к SCADA-системе. SCADA-система обычно устанавливается на ПК, расположенном в одной локальной сети с контроллером. Но в некоторых случаях на объекте ПК и персонал могут отсутствовать, и при этом требуется организовать сбор данных с контроллера. В этом случае удобно использовать облачный сервис OwenCloud. Подробнее о сервисе можно прочитать на его странице на сайте ОВЕН.

Обратите внимание, что подключение к облачному сервису получится организовать только в том случае, если вы используете контроллер ОВЕН; подключить к OwenCloud виртуальный контроллер не получится.

Как и в случае с OPC UA – добавление переменных происходит через символьную конфигурацию. Но облачный сервис не поддерживает импорт структур – поэтому потребуется в программе объявить дополнительные переменные и организовать копирование в них нужных значений.

```
PROGRAM PLC_PRG
VAR
   astMeasurePoints:         ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT;
   i:                        INT;
   xInit:                    BOOL;
   iCurrentFrameIndex:       INT   := 1;
   xEnableArchiver:          BOOL  := TRUE;

   // Точка измерения 1
   rValuePoint1_OwenCloud:   REAL;
   // Точка измерения 2
   rValuePoint2_OwenCloud:   REAL;
   // Точка измерения 3
   rValuePoint3_OwenCloud:   REAL;
   // Точка измерения 4
   rValuePoint4_OwenCloud:   REAL;
   // Точка измерения 5
   rValuePoint5_OwenCloud:   REAL;
   // Точка измерения 6
   rValuePoint6_OwenCloud:   REAL;
   // Точка измерения 7
   rValuePoint7_OwenCloud:   REAL;
   // Точка измерения 8
   rValuePoint8_OwenCloud:   REAL;
END_VAR
VAR RETAIN
   astMeasurePointsMemData: ARRAY [1..c_iMeasurePointsCount] OF MEASURE_POINT_MEM;
END_VAR
VAR CONSTANT
   c_iMeasurePointsCount:    INT  := 8;
   c_tAlarmDelay:            TIME := T#5S;
END_VAR
```

```
// Область кода

IF NOT(xInit) THEN

   astMeasurePoints[1].wsPointName := "Точка измерения 1";
   astMeasurePoints[2].wsPointName := "Точка измерения 2";
   astMeasurePoints[3].wsPointName := "Точка измерения 3";
   astMeasurePoints[4].wsPointName := "Точка измерения 4";
   astMeasurePoints[5].wsPointName := "Точка измерения 5";
   astMeasurePoints[6].wsPointName := "Точка измерения 6";
   astMeasurePoints[7].wsPointName := "Точка измерения 7";
   astMeasurePoints[8].wsPointName := "Точка измерения 8";

   xInit := TRUE;

END_IF


FOR i := 1 TO c_iMeasurePointsCount DO

   astMeasurePoints[i].xOutOfLimits := IsValueOutOfLimits(astMeasurePoints[i].rValue,
      astMeasurePointsMemData[i].rLowAlarmValue,
         astMeasurePointsMemData[i].rHighAlarmValue);
   astMeasurePoints[i].fbAlarmDelay(IN := astMeasurePoints[i].xOutOfLimits OR
      astMeasurePoints[i].eStatus <> Mx210Assistant.ANALOG_SENSOR_ERRORS.NO_ERROR,
         PT := c_tAlarmDelay, Q => astMeasurePoints[i].xAlarm);
   astMeasurePointsMemData[i].fbAlarmCounter(CU := astMeasurePoints[i].xAlarm);

   astMeasurePoints[i].fbStatusChange(eStatus := astMeasurePoints[i].eStatus);

   IF astMeasurePoints[i].fbStatusChange.Q THEN
      astMeasurePoints[i].wsStatus :=
         Mx210Assistant.ANALOG_SENSOR_ERROR_TO_WSTRING(astMeasurePoints[i].eStatus);
   END_IF

END_FOR


rValuePoint1_OwenCloud := astMeasurePoints[1].rValue;
rValuePoint2_OwenCloud := astMeasurePoints[2].rValue;
rValuePoint3_OwenCloud := astMeasurePoints[3].rValue;
rValuePoint4_OwenCloud := astMeasurePoints[4].rValue;
rValuePoint5_OwenCloud := astMeasurePoints[5].rValue;
rValuePoint6_OwenCloud := astMeasurePoints[6].rValue;
rValuePoint7_OwenCloud := astMeasurePoints[7].rValue;
rValuePoint8_OwenCloud := astMeasurePoints[8].rValue;
```

В символьной конфигурации потребуется внести несколько изменений. Во-первых, снимите галочку с переменной **astMeasurePoints** (так как облачный сервис не поддерживает массивы и структуры – то наличие переменных таких типов в символьной конфигурации не позволит установить с ним связь) и выделите галочками созданные нами переменные типа **REAL**. Чтобы добавленные в проект переменные отобразились в символьной конфигурации – нажмите кнопку **Компиляция** (см. рис. 6.9.2).

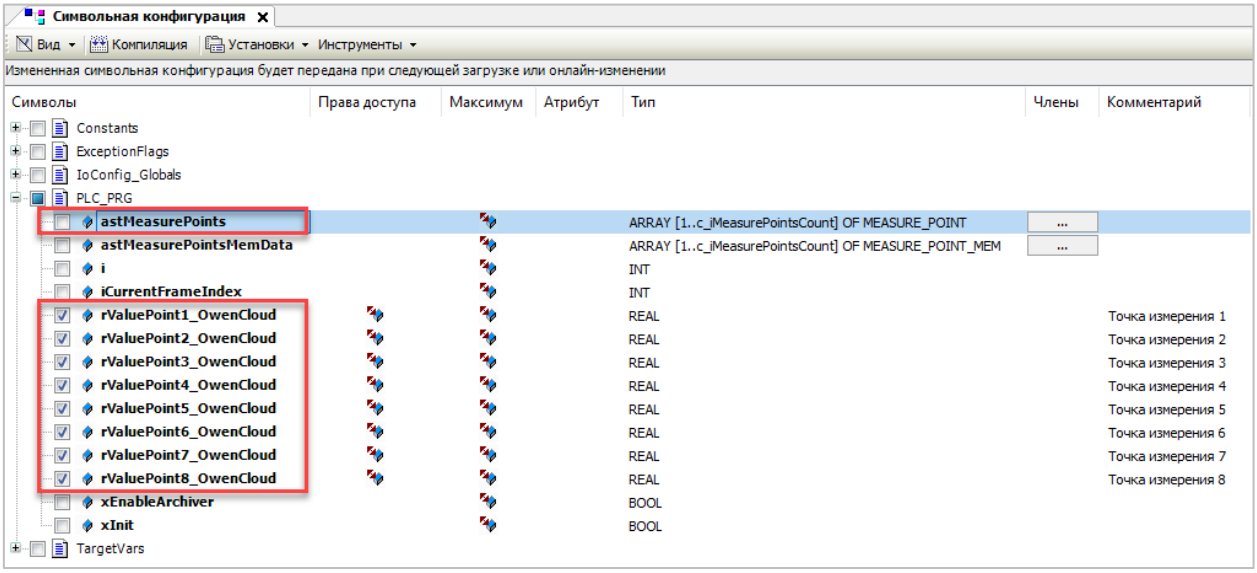


Рисунок 6.10.1 – Выбор переменных для передачи в облачный сервис

Во-вторых – нажмите кнопку **Установки**, используйте команду **Задать комментарии и атрибуты** и в появившемся окне поставьте все галочки, связанные с комментариями и атрибутами. Это необходимо для того, чтобы комментарии к переменным использовались в качестве названий параметров в облачном сервисе.

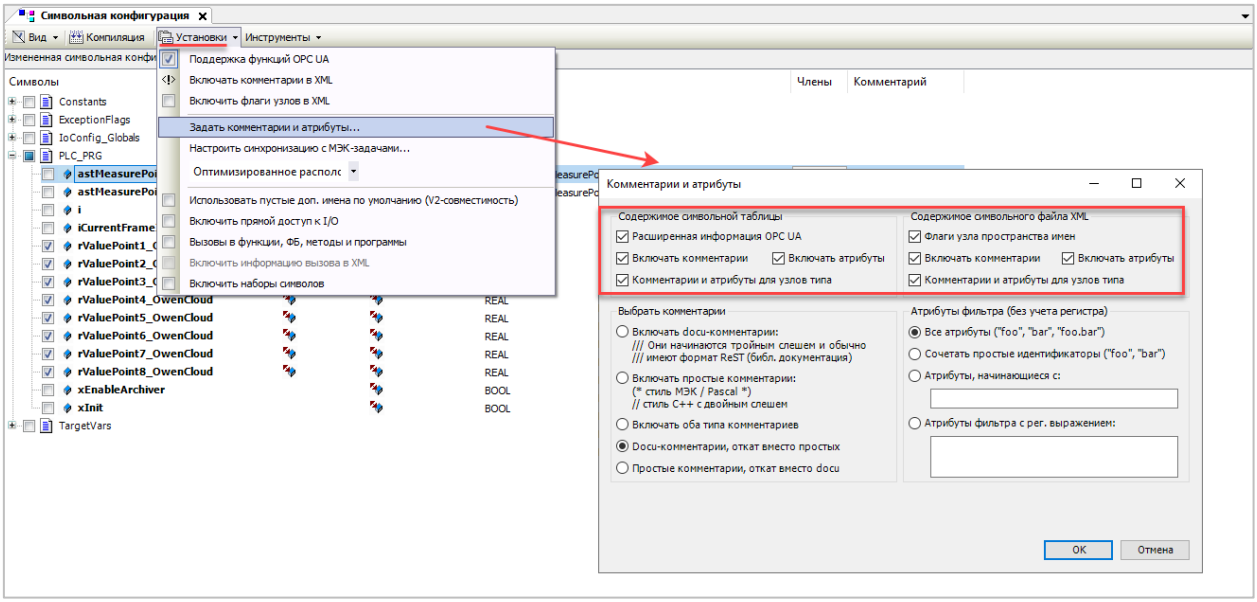


Рисунок 6.10.2 – Настройка комментариев и атрибутов

Настройка связи с облачным сервисом производится в системном узле таргет-файла **OwenCloud**. Значения параметров можно оставить по умолчанию, но ***обратите внимание*** на параметр **Password** – это пароль, который потребуется ввести при добавлении контроллера в облачный сервис. В рамках примера мы оставим пароль по умолчанию – **123456**.

На вкладке **Соотнесение входов-выходов** размещены каналы диагностики связи с облачным сервисом. Также на вкладке расположен канал **Enable OwenCloud** – если он имеет значение **TRUE**, то контроллер будет пытаться установить связь с облачным сервисом. По умолчанию канал имеет значение **TRUE** – так что можно ничего с ним не делать.

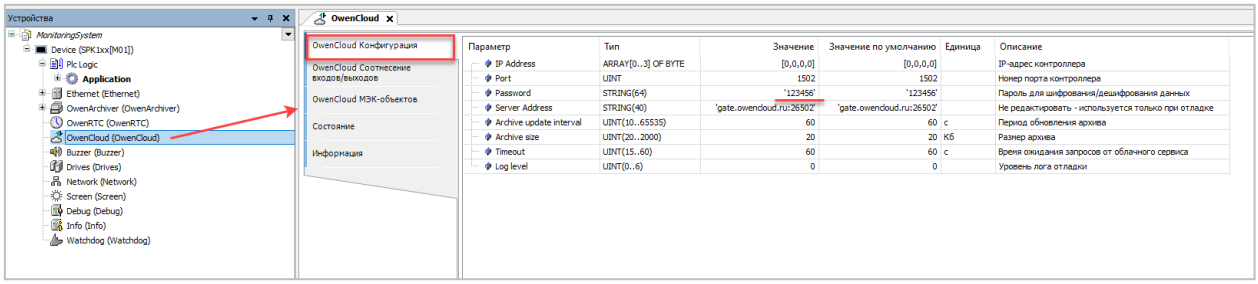


Рисунок 6.10.3 – Конфигурационные параметры компонента OwenCloud

Загрузите ваше приложение в контроллер и запустите его (используйте для этого ярлыки команд **Логин** и **Старт** на панели инструментов).

Убедитесь, что контроллер подключен к сети с доступом в интернет – это необходимое условие связи с облачным сервисом. Проще всего провести такую проверку: перейти в web-конфигуратор контроллера и на вкладке **Сеть/Диагностика** выполнить ping-запрос до сервера **gate.owencloud.ru**. В случае наличия связи – вы увидите ответы. Если же ответов нет, то следует проверить, что сеть, к которой подключен контроллер, имеет доступ в интернет, а сам контроллер имеет корректные для данной сети настройки (в частности – адрес шлюза).

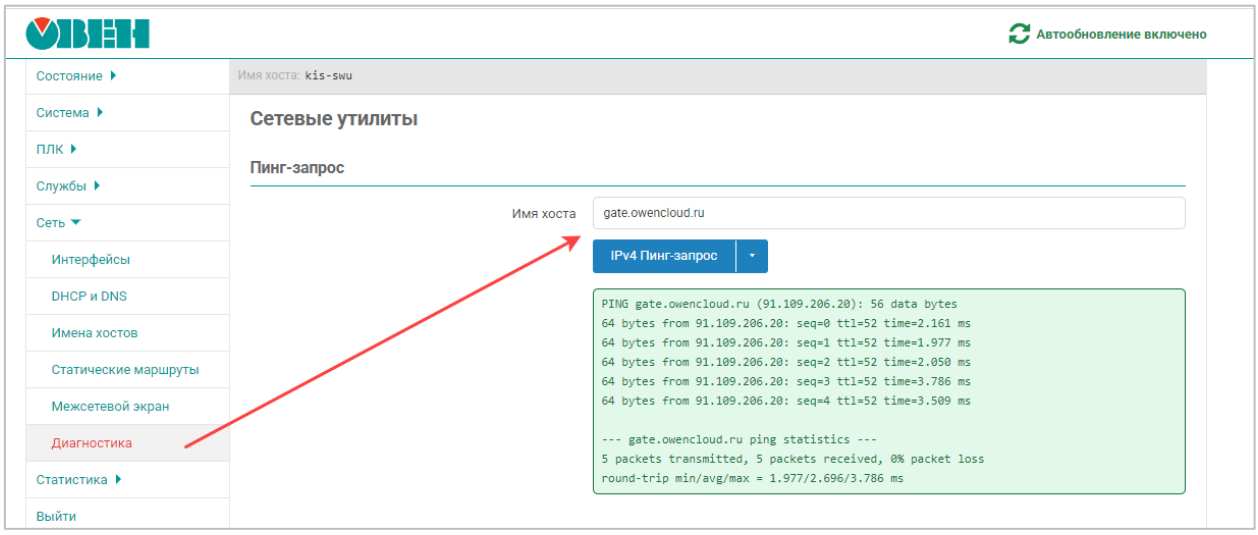


Рисунок 6.10.4 – Проверка наличия пинга с облачным сервисом через web-конфигуратор контроллера

Теперь перейдите на страницу облачного сервиса – https://web.owencloud.ru/

Если у вас еще нет аккаунта – то пройдите процедуру регистрации, после чего войдите в аккаунт.

Перейдите в раздел **Администрирование** и на вкладке **Приборы** нажмите кнопку **Добавить прибор**.

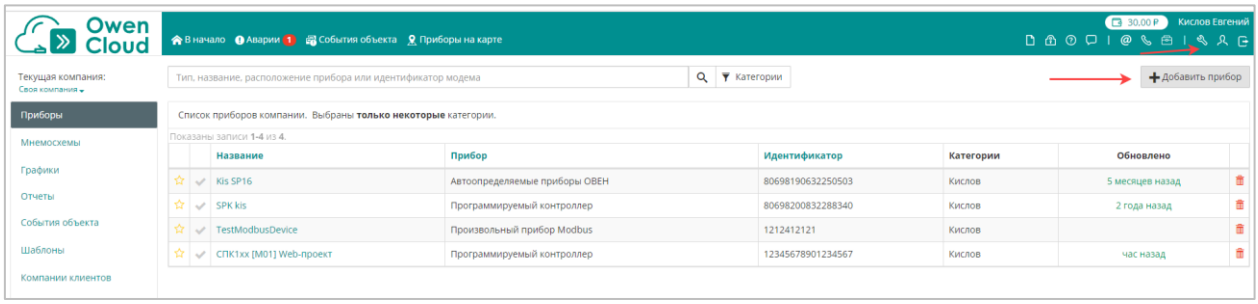


Рисунок 6.10.5 – Добавление прибора в облачный сервис

В окне добавления прибора укажите:

- Тип прибора – **Автоопределяемые устройства\Программируемый контроллер**;
- Идентификатор – серийный номер контроллера. Его можно посмотреть в web-конфигураторе (см. рис. 6.10.7) или на корпусе прибора;
- Название прибора;
- Часовой пояс, в котором находится прибор.

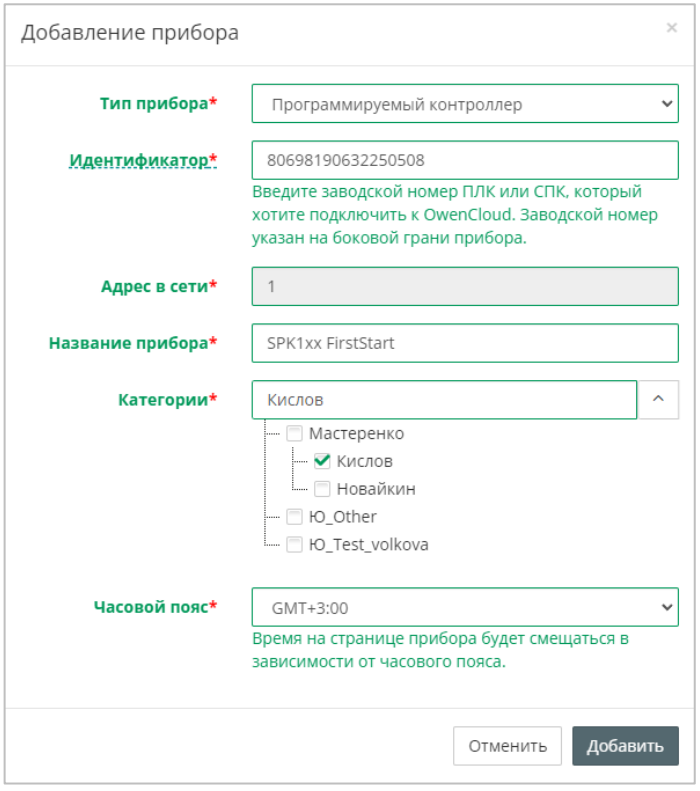


Рисунок 6.10.6 – Настройки, указываемые при добавлении прибора

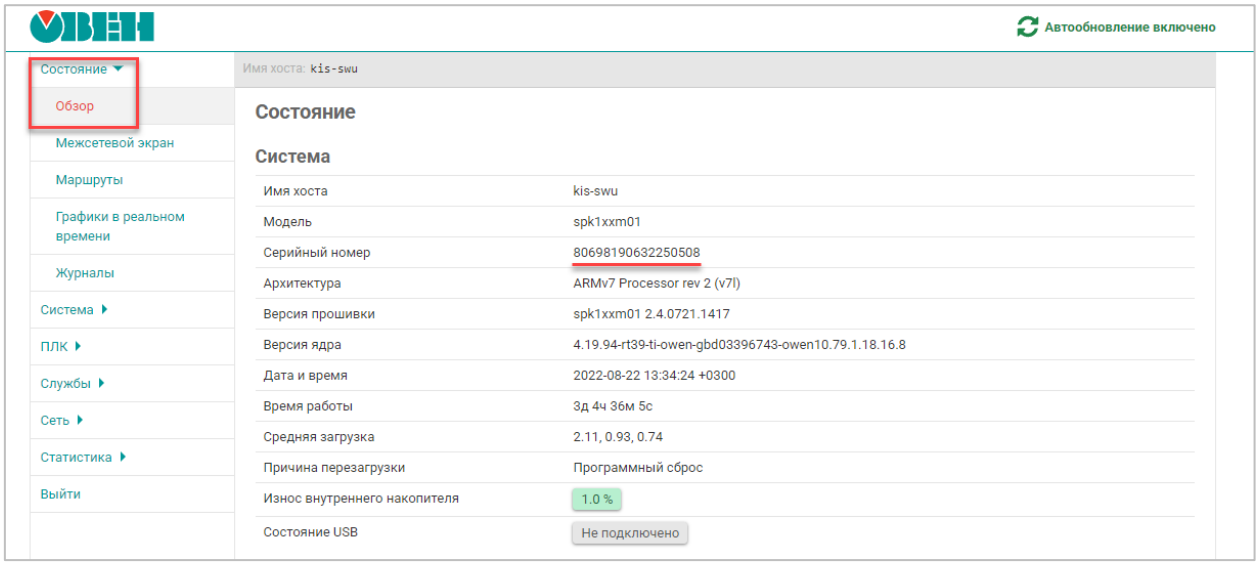


Рисунок 6.10.7 – Отображение серийного номера контроллера в web-конфигураторе

В открывшемся окно введите пароль, заданный в проекте контроллера в узле **OwenCloud** (см. рис. 6.10.3) и нажмите **Сохранить**.

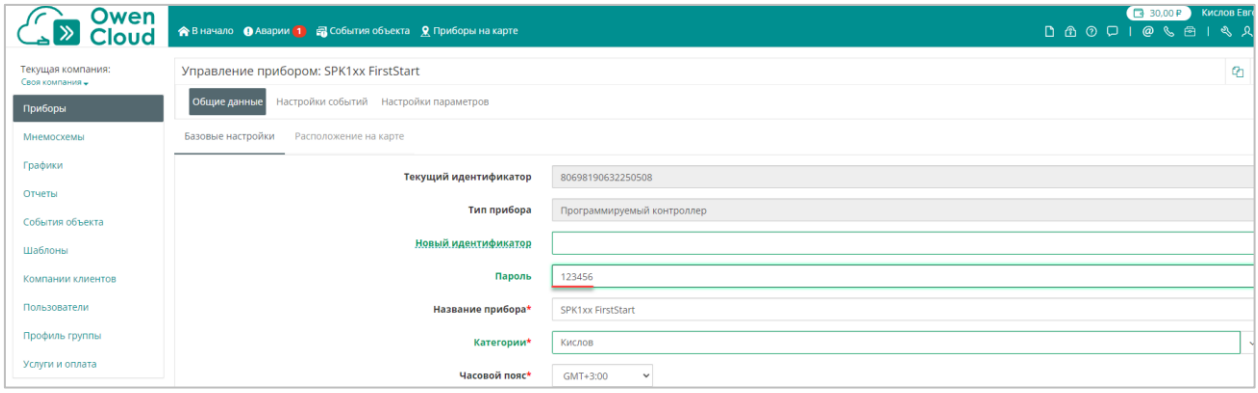


Рисунок 6.10.8 – Ввод пароля из узла OwenCloud

Нажмите кнопку **Просмотр прибора**.

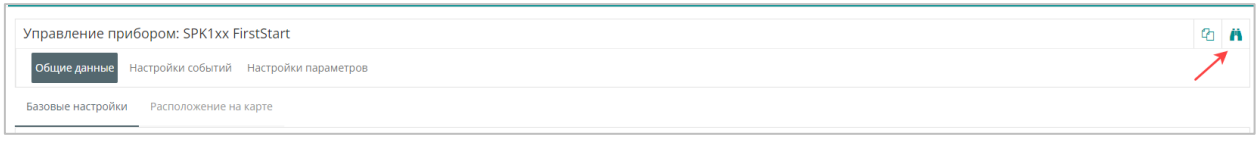


Рисунок 6.10.9 – Переход к просмотру прибора

Дождитесь соединения облачного сервиса с контроллером и выгрузки конфигурации контроллера. В зависимости от числа переменных символьной конфигурации и канала связи это может занять от 10 секунд до нескольких минут. Если после появления статуса **Обновлено** на странице не отобразятся переменные контроллера – то нажмите кнопку **F5** для обновления страницы.

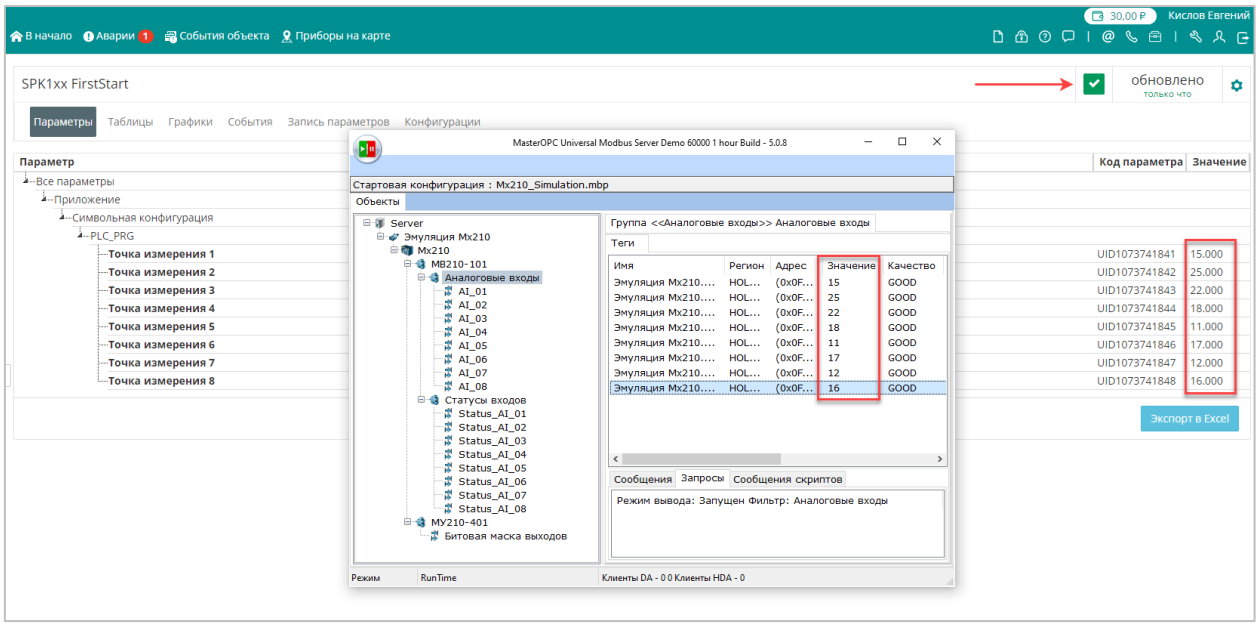


Рисунок 6.10.10 – Отображение значений переменных контроллера, считанных из OPC-сервера, в облачном сервисе OwenCloud

7 Отличия СПК и ПЛК

В рамках документа мы использовали панельный контроллер СПК1xx. Кроме них компания ОВЕН также выпускает линейку контроллеров ПЛК2xx, программируемых в среде **CODESYS V3.5**. Основное отличие ПЛК от СПК – у ПЛК нет дисплея, зато есть встроенные входы и выходы.

С точки зрения проекта CODESYS ПЛК2xx имеет следующие отличия от СПК:

1. В дереве проекта присутствует узел **PLC2xx_XX** (где **xx** – модель ПЛК, а **XX** – его модификация; например, PLC200-01 или PLC210-04), включающий подузлы **LeftSide** и **RightSide**.

Корневой узел включает каналы диагностики – например, температуру контроллера и состояние переключателя Старт/Стоп, расположенного на его корпусе (это важный переключатель – если перевести его в состояние СТОП и перезагрузить контроллер, то приложение CODESYS не будет запущено; такая возможность полезна на этапе отладки – например, если вы случайно организовали в программе циклическую перезагрузку ПЛК).

Подузел **LeftSide** отвечает за входы и выходы, размещенные на левой плате контроллера, а **RightSide** – на правой. Как и другие компоненты, эти узлы включают две вкладки: **Конфигурация** и **Соотнесение входов-выходов**. На вкладке **Конфигурация** производится настройка входов и выходов (время фильтрации входов, переключение входов в режим обработки сигналов энкодеров, режим работы выходов и т. д.), а на вкладке **Соотнесение входов-выходов** – привязка переменных к каналам битовых масок входов/выходов, счетчиков входов и т. д.

2. В дереве проекта отсутствуют узлы **Screen** и **Network**. Узел **Screen** отсутствует по очевидной причине – как мы упоминали выше, у ПЛК2xx нет дисплея. Отсутствие узла **Network** связано с тем, что ПЛК2xx имеют несколько сетевых интерфейсов, которые могут объединяться в различные схемы (например, в мост), поэтому сделать для них интерфейс конфигурирования в CODESYS было бы крайне трудоемкой задачей. Вместо этого настройка сетевых интерфейсов ПЛК производится в web-конфигураторе.

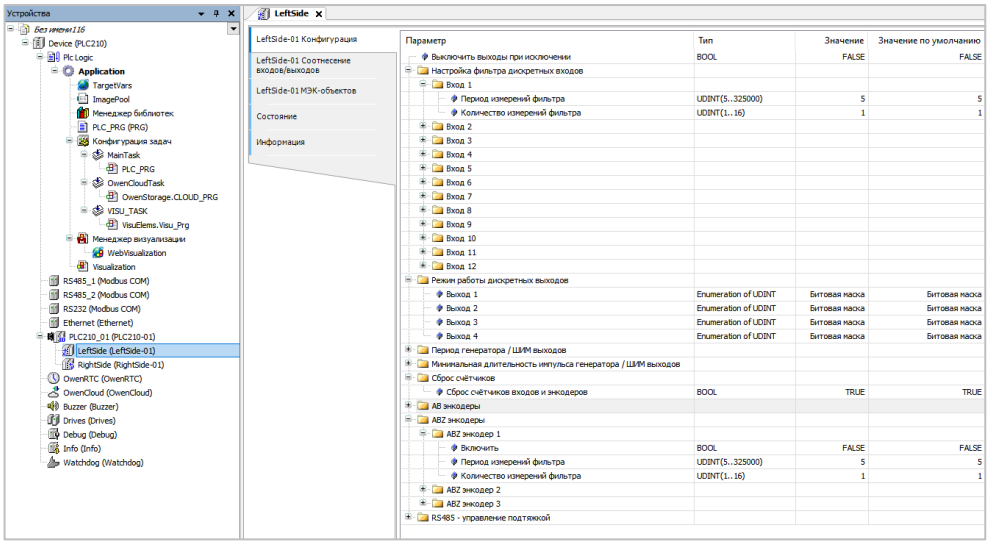


Рисунок 7.1 – Дерево проекта контроллера ПЛК210

3. В компонент **Менеджер визуализации** нельзя добавить подкомпонент **Таргет-визуализация** – опять же, потому что у ПЛК нет дисплея.

Других отличий при разработке проектов для СПК и ПЛК нет.

8 Импорт проектов из старых версий CODESYS

В рамках документа мы использовали среду **CODESYS V3.5 SP17 Patch 3** – именно в ней программируются контроллеры ОВЕН с текущими заводскими прошивками.

На сайте ОВЕН в разделе CODESYS V3/Примеры выложено множество примеров проектов для CODESYS V3.5, и часть из них создавалась в более старых версиях CODESYS – **V3.5 SP11 Patch 5**, **V3.5 SP14 Patch 3** и т. д.

В этом пункте мы расскажем, как открыть эти проекты в среде **CODESYS V3.5 SP17 Patch 3** и добиться их корректной работы. Рассмотрим это на примере проекта Сепарация нефти, созданного в версии **CODESYS V3.5 SP11 Patch 5**.

Загрузите архив проекта и откройте его в **CODESYS V3.5 SP17 Patch 3** (меню «Файл» – **Открыть**). Появится окно распаковки архива. Желательно поставить все галочки, чтобы извлечь из архива все доступные компоненты.

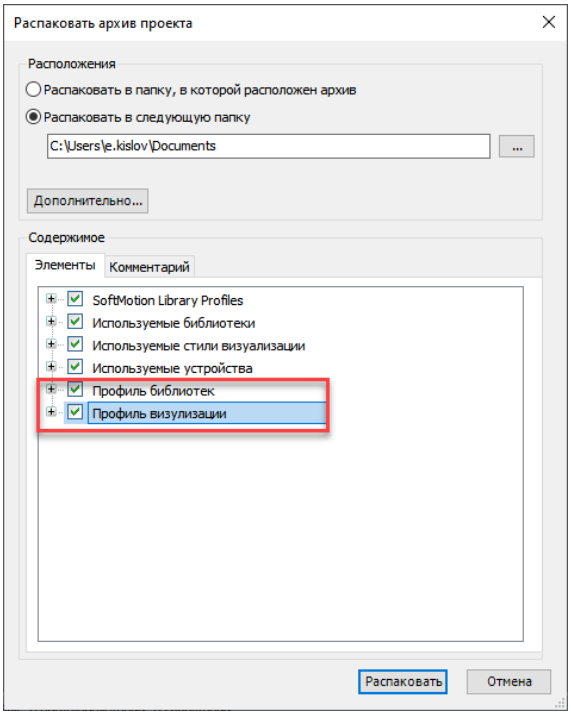


Рисунок 8.1 – Окно распаковки архива проекта

Если будут появляться сообщения с предложениями перезаписать что-либо – нажимайте **No**, чтобы не вносить изменений в ваше текущее рабочее окружение.

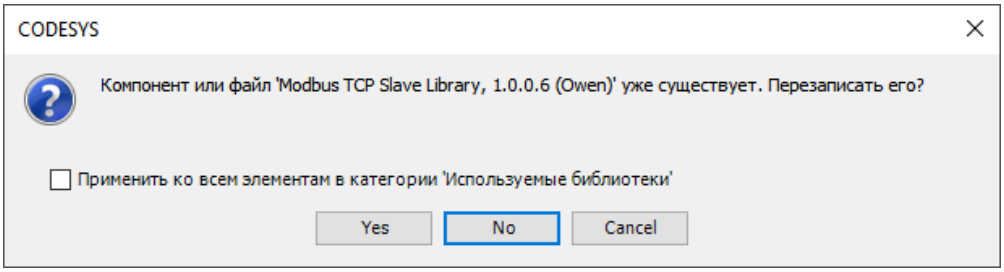


Рисунок 8.2 – Окно перезаписи компонентов

Если появится подобное окно – нажмите **ОК**.

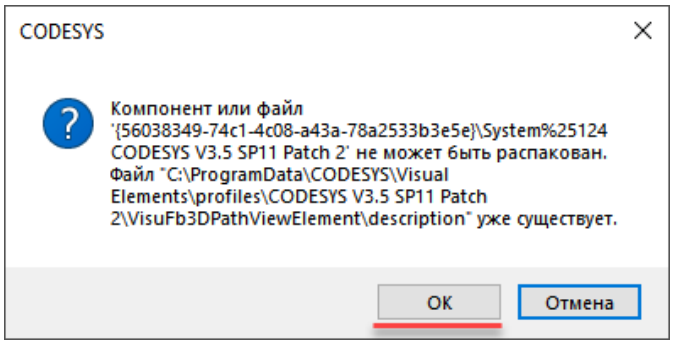


Рисунок 8.3 – Окно ошибки распаковки для уже существующего файла

Когда появится окно **Среда проекта** – нажмите кнопку **Отмена**. Это окно предлагает автоматическое обновление проекта до самых свежих версий установленных компонентов и библиотек; но вместо этого лучше произвести обновление вручную, так как если у вас установлены еще какие-то версии CODESYS, то самые свежие версии компонентов и библиотек подтянутся из них – а нужно, чтобы они соответствовали версии CODESYS, в которой вы открываете проект.

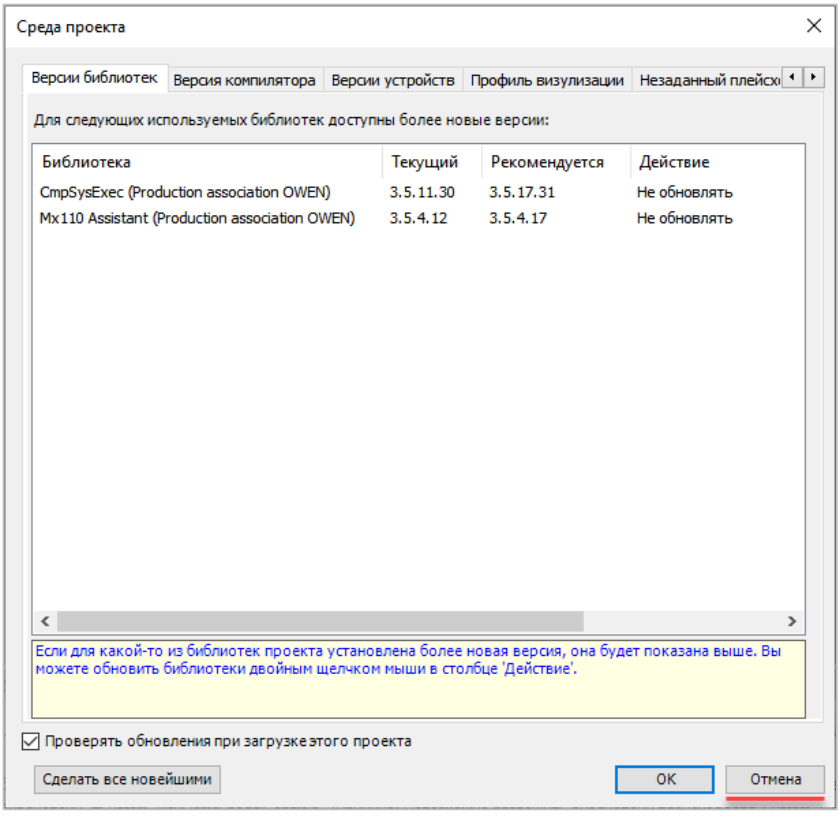


Рисунок 8.4 – Окно среды проекта

После этого появится возможность редактирования проекта.

Нажмите правой кнопкой мыши на узел **Device** и используйте команду **Обновить устройство**. Выберите нужную версию нужного таргет-файла (используйте галочку **Отображать все версии** для отображения всех установленных версий) и нажмите кнопку **Обновить устройство.** Мы импортируем проект в **CODESYS V3.5 SP17 Patch 3** – поэтому нужная нам версия таргет-файла должна иметь версию **3.5.17.3x** (где вместо **x** – это номер сборки).

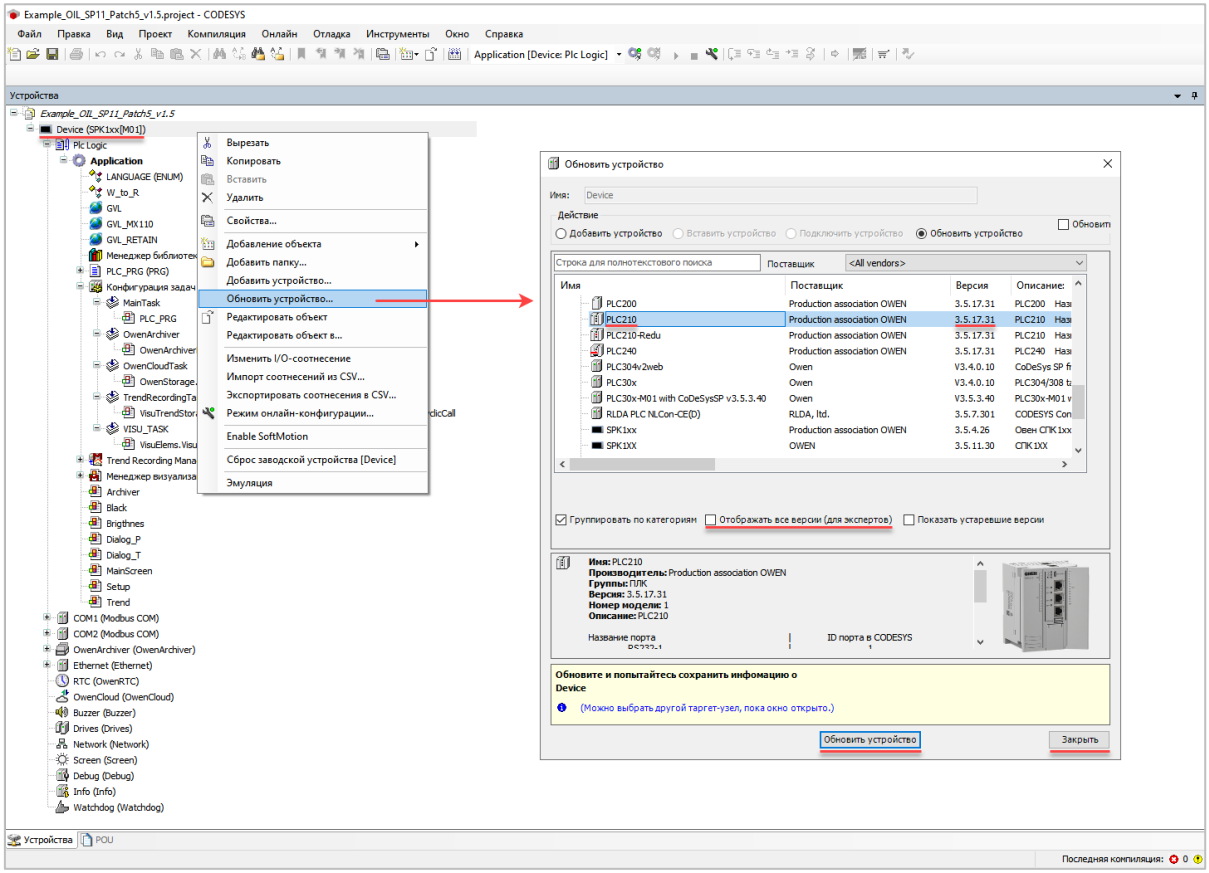


Рисунок 8.5 – Окно обновления таргет-файла

Если появится подобное окно – нажмите **Да**. Учтите, что исходный проект в этот момент будет пересохранен – так что в случае необходимости сохранить его исходный вариант вам нужно будет перед нажатием кнопки сделать его копию.

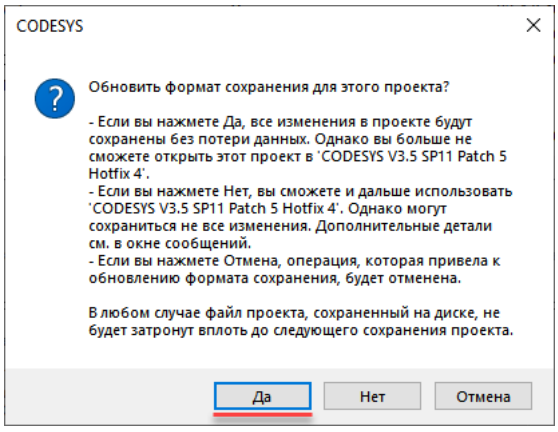


Рисунок 8.6 – Окно обновления формата сохранения проекта

Если вы переносите проект с таргет-визуализацией на **ПЛК2xx** – то удалите из дерева проекта узел **Таргет-визуализация**.

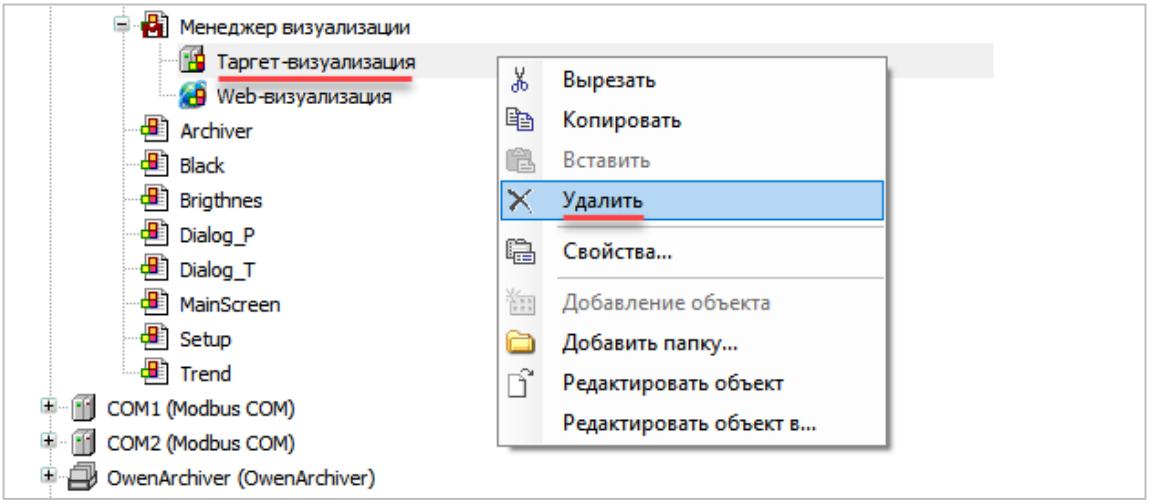


Рисунок 8.7 – Удаление таргет-визуализации из проекта ПЛК2xx

Обновите версии всех компонентов Modbus (**Modbus COM**, **Ethernet, Modbus Master** и т. д.) за исключением шаблонов опроса устройств ОВЕН. Версия компонента не должна превышать версию таргет-файла – например, в проекте с таргет-файлом **3.5.17.x** нельзя использовать компоненты Modbus версий **3.5.18.x**. Процедура обновления аналогична обновлению версии таргет-файла.

Начиная с версии **CODESYS V3.5 SP17** разработка компонентов CODESYS происходит независимо от разработки системы исполнения. Версионность таких компонентов изменена на **4.x.x.x**. Использовать эти версии компонентов – допустимо.

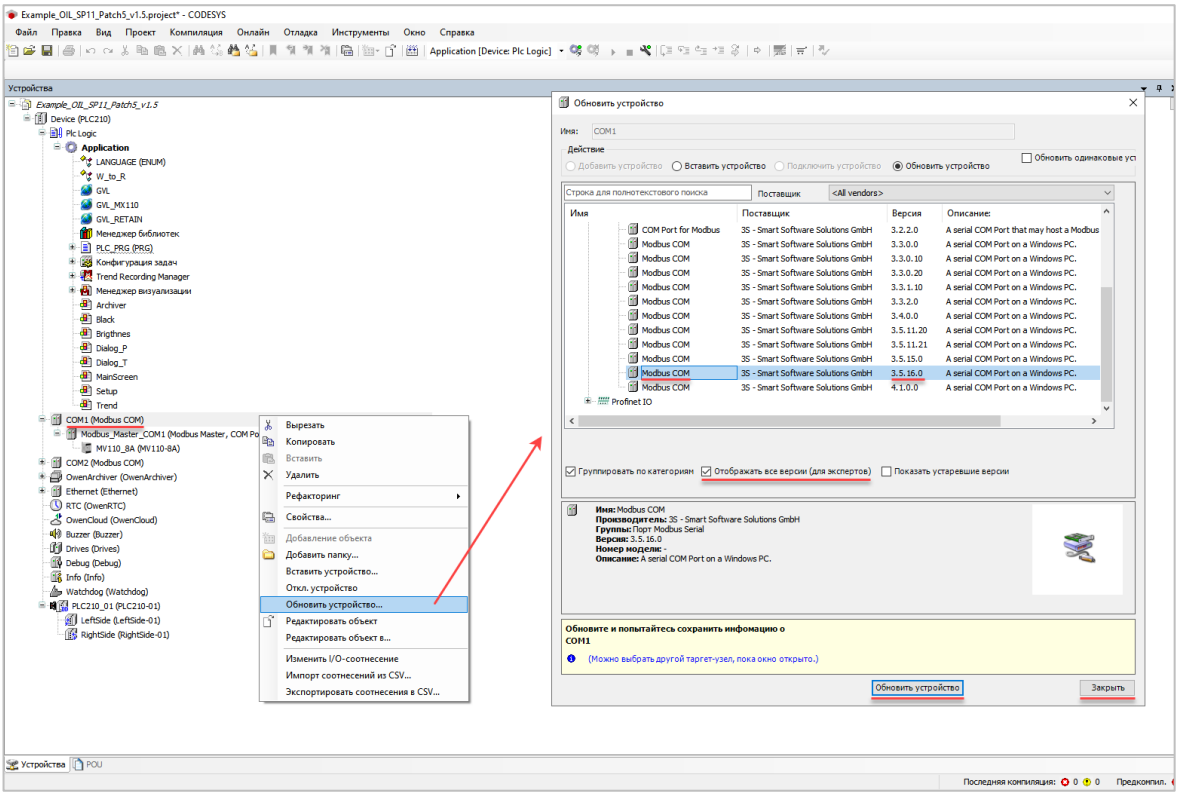


Рисунок 8.8 – Обновление версий компонентов Modbus

Шаблоны опроса устройств ОВЕН не поддерживают команду **Обновить устройство**. Для обновления их версии необходимо удалить текущие шаблоны из дерева проекта, добавить заново шаблоны новых версий (нажмите правой кнопкой мыши на компонент **Modbus Master** и используйте команду **Добавить устройство**), заново настроить конфигурационные параметры на вкладке **Конфигурация** (если у шаблона есть эта вкладка) и привязать переменные на вкладке **Соотнесение входов-выходов**. Предварительно следует установить пакет нужных вам версий шаблонов через CODESYS Installer.

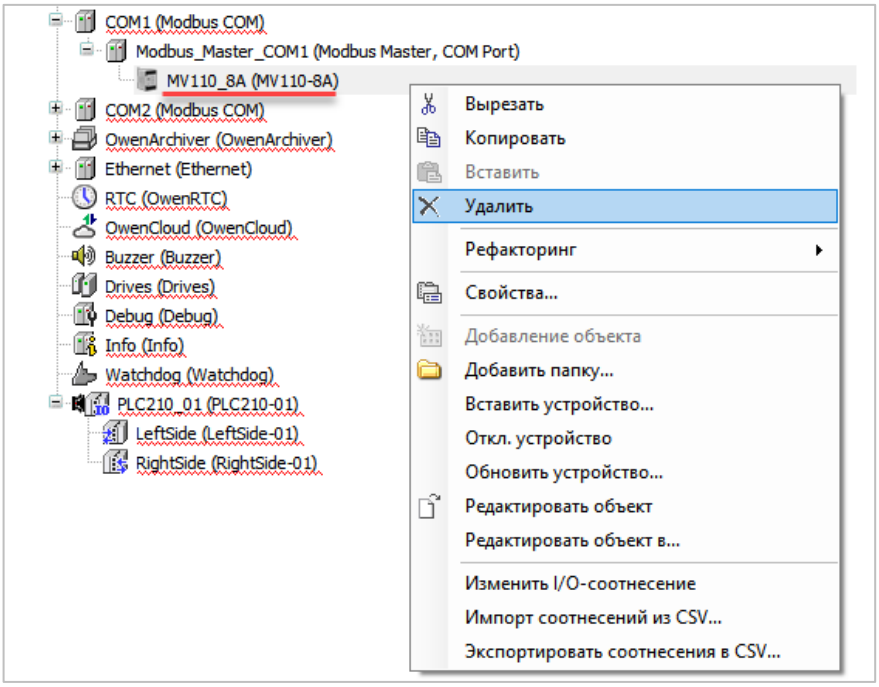


Рисунок 8.9 – Удаление шаблонов старых версий из дереве проекта

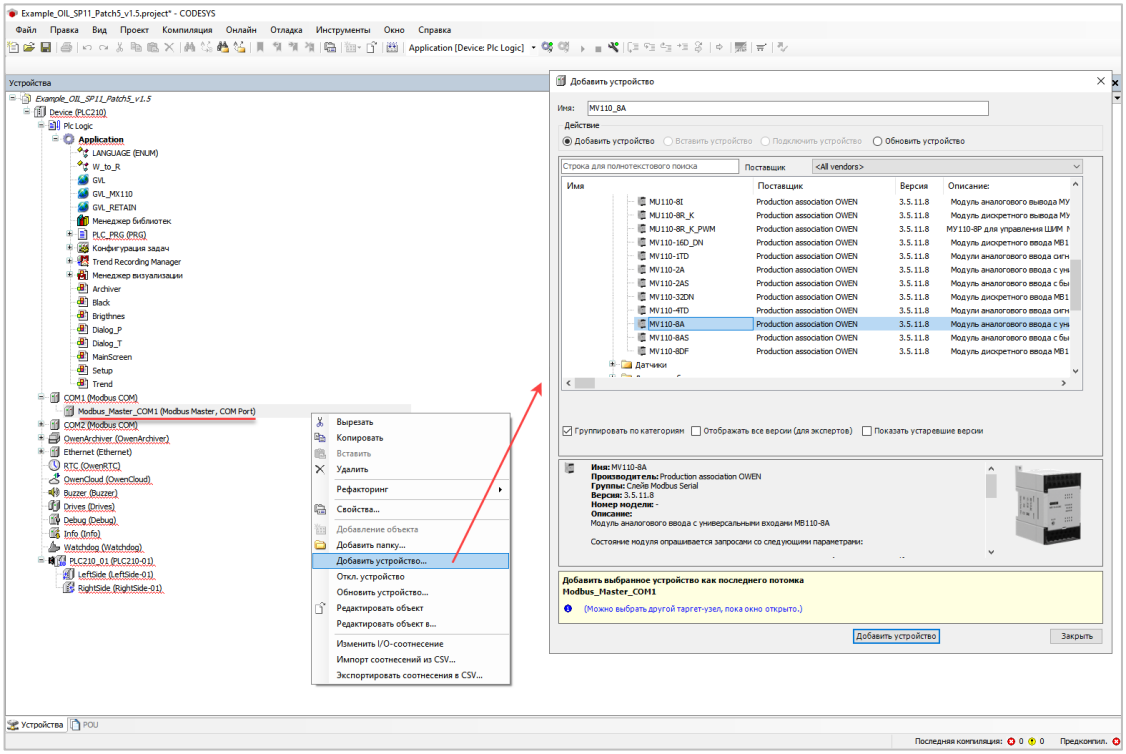


Рисунок 8.10 – Добавление новых версий шаблонов

В меню «Проект» выберите пункт **Установки проекта**. На вкладке **Опции компиляции** выберите версию компилятора, которая совпадает с версией среды программирования – например, для среды **V3.5 SP17 Patch 3** выберите версию **3.5.17.30**.

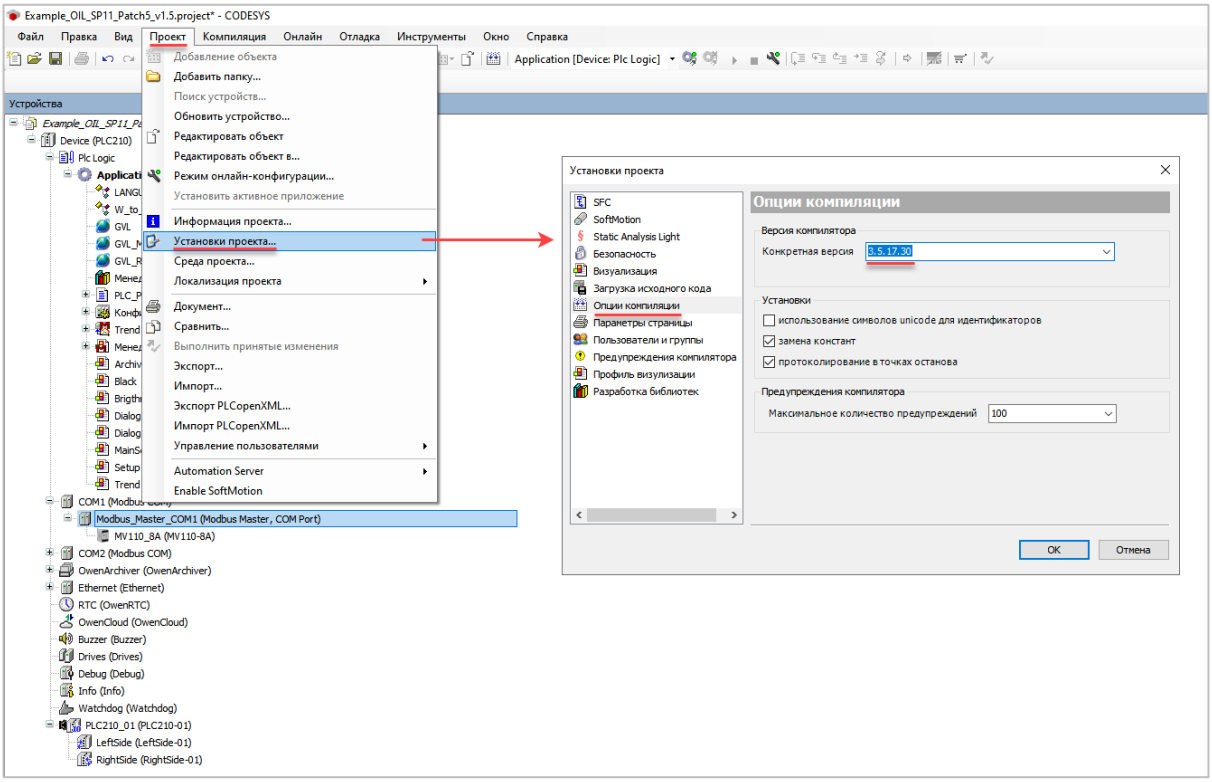


Рисунок 8.11 – Выбор версии компилятора в установках проекта

На вкладке **Профиль визуализации** выберите нужную версию профиля визуализации. В современных версиях CODESYS отображается только установленная в данном окружении версия профиля – так что ошибиться будет сложно. Для версии **V3.5 SP17 Patch 3** по умолчанию используется версия **4.1.1.0**. Нажмите кнопку **OK**.

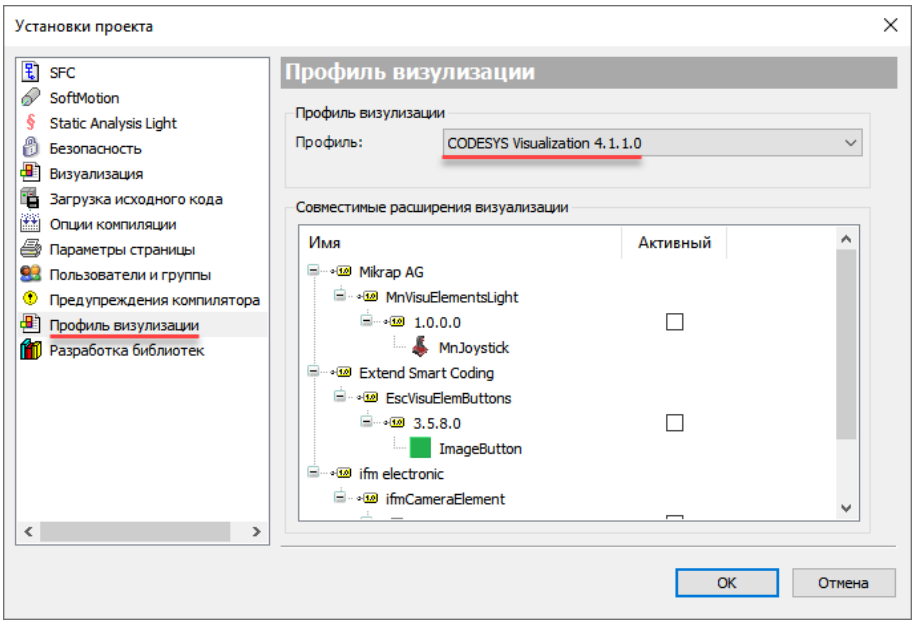


Рисунок 8.12 – Выбор версии профиля визуализации в установках проекта

Если появится подобное окно – нажмите **Да**.

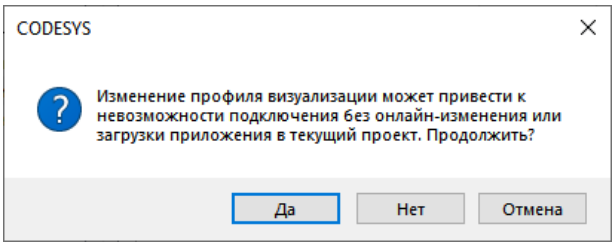


Рисунок 8.13 – Окно невозможности подключения с онлайн-изменением после изменения версии профиля визуализации

На этом основная часть процедуры импорта проекта завершена.

Но в ряде случаев может потребоваться обновить версии некоторых библиотек проекта. Есть две причины, с которыми это связано:

1. Версия библиотеки из старой версии CODESYS не поддерживается в новой версии CODESYS. Например, это касается библиотек **CmpSysExec**, **OwenYandexHome** и **OwenVisuDialogs** версий **3.5.16.30** – вы не сможете использовать их в **CODESYS V3.5 SP17 Patch 3**. Вам потребуется обновить их до версий **3.5.17.x**;

2. Вы получили проект, созданный на другом ПК, в виде файла формата **.project** (в этом случае в него не входят файлы используемых библиотек – в отличие от формата **.projectarchive**), и на вашем ПК нужные версии каких-то из этих библиотек отсутствуют.

В случае **1** откройте **Менеджер библиотек**, нажмите правой кнопкой мыши на нужную библиотеку, используйте команду **Свойства** и выберите конкретную версию библиотеки, требуемую для данной версии CODESYS.

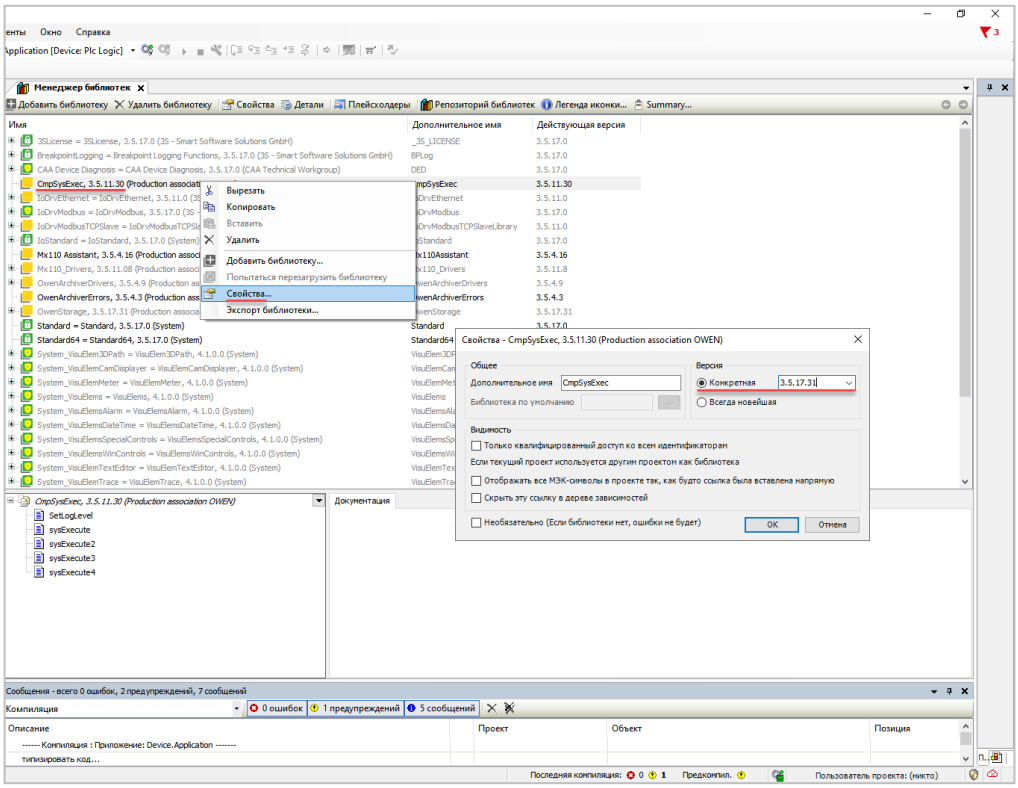


Рисунок 8.14 – Изменение версии библиотеки, используемой в проекте

В случае **2** отсутствующие на вашем ПК библиотеки будут подчеркнуты в менеджере библиотек синей волнистой линией. Если это системные библиотеки CODESYS – то нажмите кнопку **Загрузка отсутствующих библиотек**. Если это дополнительно устанавливаемая библиотека (например, одна из библиотек ОВЕН) – то нажмите на нее правой кнопкой мыши, используйте команду **Свойства** и выберите более свежую версию библиотеки, если она установлена на вашем ПК. Если такая версия не установлена (или на вашем ПК вообще не установлено ни одной версии этой библиотеки) – то следует загрузить ее, установить через Репозиторий библиотек и добавить в проект через Менеджер библиотек.

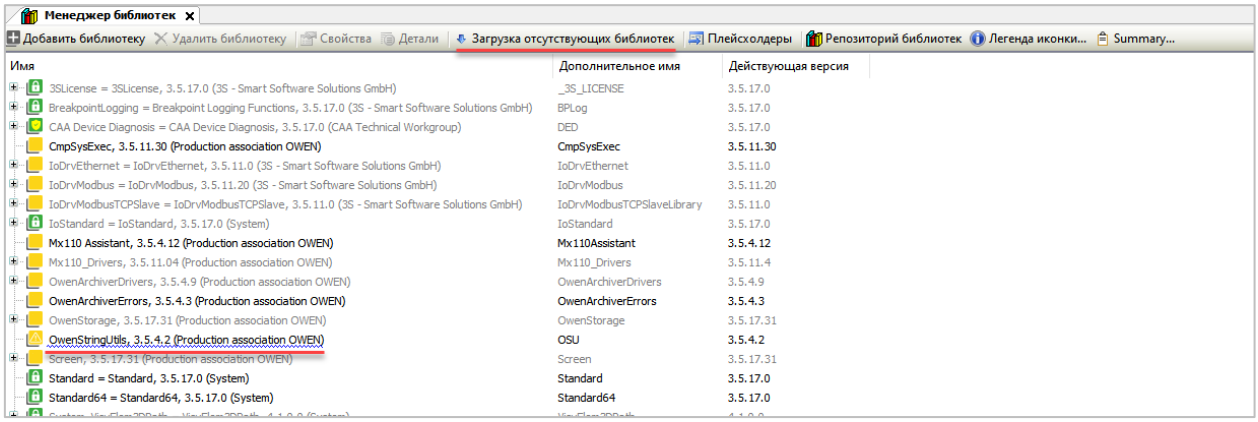


Рисунок 8.15 – Отображение отсутствующих на ПК библиотек

После завершения импорта не забудьте сохранить проект под новым названием.

Теперь можно загрузить его в контроллер.

9 Дополнительные материалы

В рамках документа мы рассмотрели лишь основные аспекты работы в **CODESYS V3.5**, наиболее важные для начинающего пользователя. Более подробная информация по конкретным вопросам приведена в других документах, доступных на сайте ОВЕН в разделе CODESYS V3/Документация. Обратите особое внимание на следующие документы:

- **CODESYS V3.5. Протокол Modbus** – полное и подробное описание использования протокола Modbus в CODESYS V3.5. В частности, в нем описывается, как настроить обмен с произвольным устройством (в рамках «первого старта» мы рассмотрели лишь добавление модулей Mx210 через готовые шаблоны, но для устройств других производителей таких шаблонов нет) и организовать работу контроллера в режиме **Modbus Slave**;
- **CODESYS V3.5. Визуализация** – инструкция по созданию экранов с перечислением всех свойств всех доступных элементов визуализации и примерами их использования;
- **CODESYS V3.5. Описание таргет-файлов** – описание системных узлов дерева проекта;
- **CODESYS V3.5. FAQ** – ответы на часто задаваемые вопросы по работе в среде CODESYS V3.5 и программированию контроллеров ОВЕН.

Полная и подробная официальная документация по CODESYS (описание редакторов программирования, пунктов меню и т. д.) доступна в онлайн-справке. Специфические вопросы рассмотрены в FAQ.

Ссылки на дистрибутивы CODESYS, таргет-файлы, дополнительные компоненты и библиотеки, прошивки контроллеров и т. д. доступны на сайте ОВЕН в разделе CODESYS V3.

На YouTube-канале ОВЕН доступны плейлисты с видеоуроками и вебинарами по программированию в CODESYS V3.5.

Пройти очное обучение по программированию в CODESYS V3.5 можно в учебном центре ОВЕН.

В случае возникновения каких-либо вопросов – вы можете задать их на форуме ОВЕН в разделе ПЛК (среда программирования CODESYS V3.5). Если ваш вопрос носит общий характер – то задайте его в одной из тем в корне раздела. Если вопрос связан с конкретным контроллером – то создайте новую тему в разделе соответствующего контроллера или задайте его в одной из закрепленных тем этого раздела. Отдельно обратите внимание на онлайн-FAQ – возможно, в нем уже есть ответ на ваш вопрос.

Желаем удачи в дальнейшем изучении среды CODESYS V3.5 и ее использовании в вашей профессиональной деятельности!