CODESYS V3.5

Первый старт

Руководство пользователя

01.12.2018
версия 2.0

Оглавление

Глоссарий

ПЛК – программируемый логический контроллер.

СПК – сенсорный панельный контроллер, сочетающий в себе свободно программируемое устройство с функциями панели оператора.

Прошивка – системное программное обеспечение, которое управляет работой контроллера на аппаратном уровне.

CODESYS – среда разработки, используемая для создания и отладки прикладного программного обеспечения и разработки интерфейса оператора.

Таргет-файл – файл, использующийся **CODESYS** для определения типа контроллера, для которого разрабатывается проект.

Проект – совокупность настроенных пользователем компонентов **CODESYS** (экраны визуализации, программные модули, модули связи и т. д.), которая сохраняется как файл формата **.project** и затем загружается в контроллер.

Библиотека – файл, содержащий готовые компоненты, которые могут быть использованы в процессе разработке пользовательского проекта. Стандартный набор библиотек входит в CODESYS, дополнительные библиотеки распространяются компанией ОВЕН, также пользователь может создавать собственные библиотеки.

POU (**Program Organizaton Unit, компонент**) – структурная единица проекта (например программа, экран визуализации, модуль связи с другим устройством и т. д.).

Задача – определяет частоту и правила вызова **POU**.

Экран визуализации – структурная единица интерфейса оператора, используемая для отображения информации о технологическом процессе и управления им.

ПК – персональный компьютер.

ЛКМ/**ПКМ** – левая кнопка мыши/правая кнопка мыши.

1 Введение

1.1 Цель руководства

Настоящее руководство дает вводную информацию для работы с контроллерами ОВЕН, программируемыми в среде CODESYS V3.5, и содержит следующие разделы:

1. Установка необходимого программного обеспечения.
2. Настройка связи между контроллером и компьютером.
3. Описание интерфейса **CODESYS**.
4. Подключение к контроллеру модулей ввода-вывода и их конфигурирование.
5. Создание и запуск демонстрационного проекта.

1.2 Список контроллеров, программируемых в CODESYS

В таблице ниже приведен список контроллеров ОВЕН, программируемых в среде CODESYS V3.5 (дата составления списка 12.2018):

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

Контроллер	Версия CODESYS	Статус
СПК1xx [M01]	3.5.11.5	В продаже
СПК1xx	3.5.5.5	Сняты с продажи
ПЛК304	3.5.5.5	В продаже
СПК207	3.5.5.5	Сняты с продажи
ПЛК323	3.5.5.5	Сняты с продажи

1.3 Общие сведения о CODESYS V3.5

CODESYS (Controller **De**velopment **Sy**stem) — программный комплекс промышленной автоматизации, основанный на стандарте **IEC (МЭК) 61131-3**. Производится и распространяется компанией **3S-Smart Software Solutions GmbH** (Германия).

CODESYS используется для создания и отладки прикладного программного обеспечения и разработки интерфейса оператора, которые в сочетании образуют пользовательский проект. Все взаимодействие с контроллером происходит с помощью **CODESYS**, другое программное обеспечение для этого не требуется.

CODESYS постоянно развивается, что приводит к периодическому выпуску новых версий. Начиная с **CODESYS V3.0**, версии устанавливаются независимо друг от друга (свежая версия не обновляет предыдущую, а устанавливается параллельно), но при этом требуют установки исключительно в порядке возрастания.

Среда программирования **CODESYS** полностью русифицирована.

1.4 Система версий CODESYS

Название версии CODESYS выглядит следующим образом:

CODESYS V3.**x** <SP**y**> <Patch **z**>,

где **V3.x** – номер **текущей версии** CODESYS.

Новая версия обычно включает в себя принципиальные нововведения, сопровождающиеся серьезными изменениями среды программирования и добавлением значительного количества новых функций.

y – номер пакета обновления (**service pack)**.

Пакет обновления может вносить изменения, касающиеся интерфейса среды программирования и добавления определенного функционала. Также пакет обновления включает в себя все патчи, выпущенные с момента релиза предыдущего пакета.

z – номер патча (**patch)**.

Патчи исправляют различные ошибки среды программирования.

Различные версии CODESYS устанавливаются **независимо друг от друга**. В системе может быть установлено множество версий CODESYS, и все они могут быть запущены одновременно, но ***необходимо*** соблюдать указания из п. 2.

1.5 Таргет-файл и прошивка контроллера

Таргет-файл (файл целевой платформы) содержит информацию о ресурсах контроллера и обеспечивает его связь с CODESYS. Каждая модель контроллера ОВЕН имеет соответствующий таргет-файл, который необходимо установить перед началом создания проекта в среду CODESYS. Таргет-файлы входят на диск с ПО из комплекта поставки и доступны в разделе **Сервисное ПО** соответствующей модели контроллера на сайте owen.ua.

ПРИМЕЧАНИЕ

Версия таргет-файла должна соответствовать версии прошивки контроллера. См. более подробную информацию в руководстве **CODESYS V3.5. FAQ**.

Прошивка – это системное программное обеспечение, которое управляет работой контроллера на аппаратном уровне. В связи с добавлением новых функций и исправлением ошибок, регулярно осуществляется выпуск новых версий прошивок. В случае необходимости пользователь может самостоятельно сменить версию прошивки (вся информация о перепрошивке находится в разделе **Сервисное ПО** на сайте owen.ua).

Версии прошивки и таргет-файла ***жестко связаны*** между собой, при этом версия CODESYS может превышать версию таргет-файла, но корректная работа гарантируется только в случае использования версий ПО с диска из комплекта поставки.

2 Установка ПО

Для работы с контроллером следует установить:

1. CODESYS V3.5 (версия **SP11 Patch 5 или выше**).
2. Репозиторий предыдущих версий системных библиотек для CODESYS (**CODESYS Repository Archive V3.5 SP4**).
3. Пакет таргет-файлов для CODESYS (**OwenTargets**).

Системные требования среды CODESYS:

Параметр	Минимальные	Рекомендуемые
ОС	7/8/10	Windows 7/8/10 (64 bit)
ОЗУ	1024 Мб	4 Гб или выше
Жесткий диск	не менее 2 Гб свободного места	не менее 5 Гб свободного места
Процессор	Pentium V, > 3.0 ГГц	Core i7, > 3.0 ГГц

ПРИМЕЧАНИЕ

Рекомендуется устанавливать различные версии CODESYS исключительно в порядке возрастания.

ПРИМЕЧАНИЕ

Рекомендуемая версии среды программирования зависит от версии прошивки контроллера. См. более подробную информацию в руководстве **CODESYS V3.5 FAQ**.

ПРИМЕЧАНИЕ

На диске с ПО из комплекта поставки и

е компании **ОВЕН** могут находиться более новые версии ПО, но сам процесс установки будет совершенно аналогичен.

Установка CODESYS:

1. Для начала установки **CODESYS** следует запустить соответствующее приложение (находится на диске с ПО из комплекта поставки, а также доступно на сайте ОВЕН). Появится окно загрузки и отобразятся необходимые дополнительные компоненты для установки CODESYS. Чтобы загрузить компоненты из Интернета или установить самостоятельно, следует нажать кнопку **Install**.

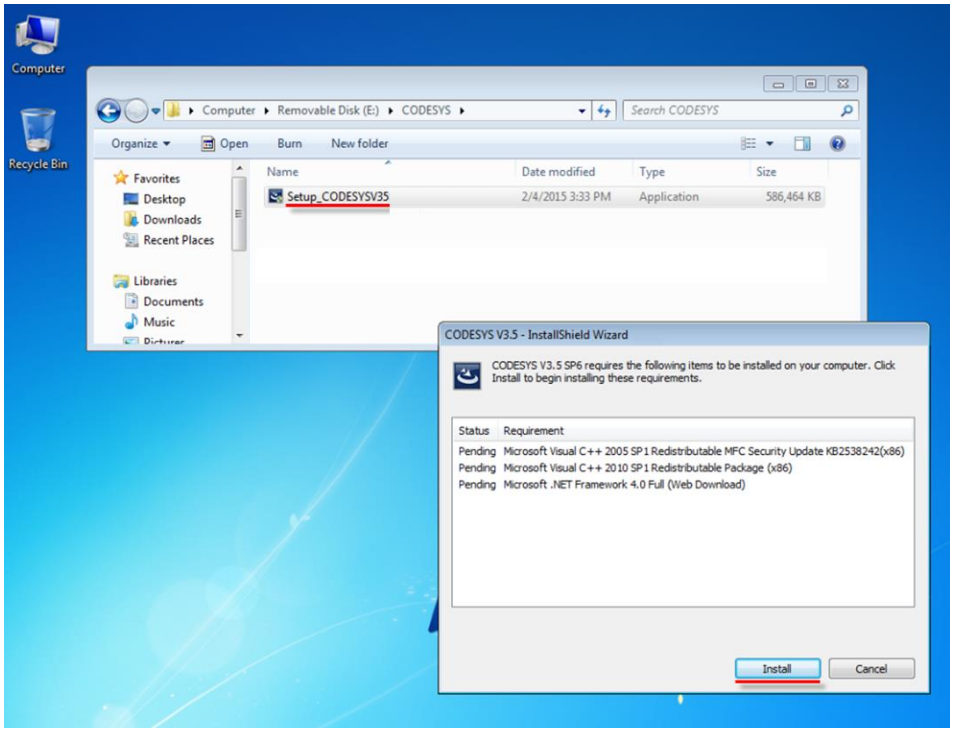


Рисунок 2.1 – Установка дополнительных компонентов

2. Для начала установки CODESYS следует нажать кнопку **Next**.

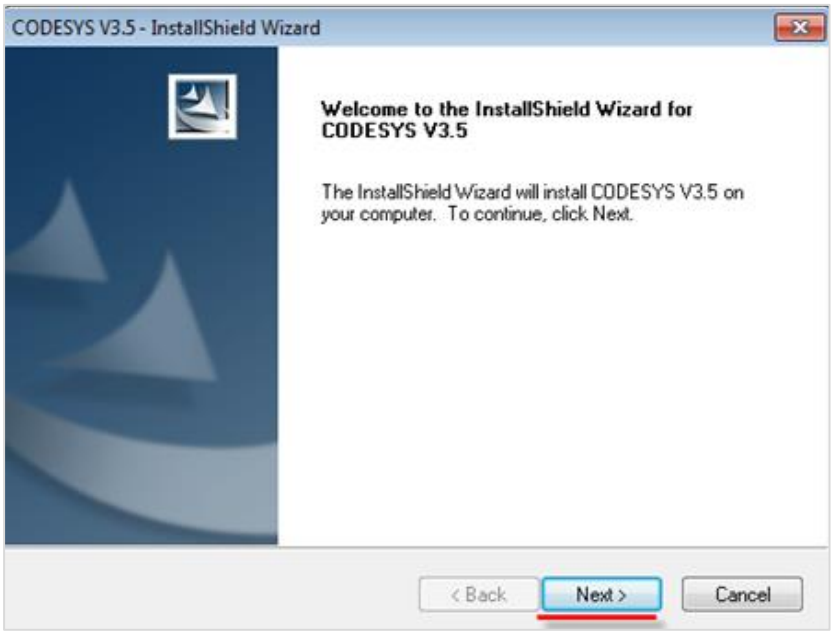


Рисунок 2.2 – Начало установки CODESYS

3. Чтобы продолжить установку после ознакомления с текстом лицензионного соглашения следует выбрать пункт **I accept…** и нажать кнопку **Next**.

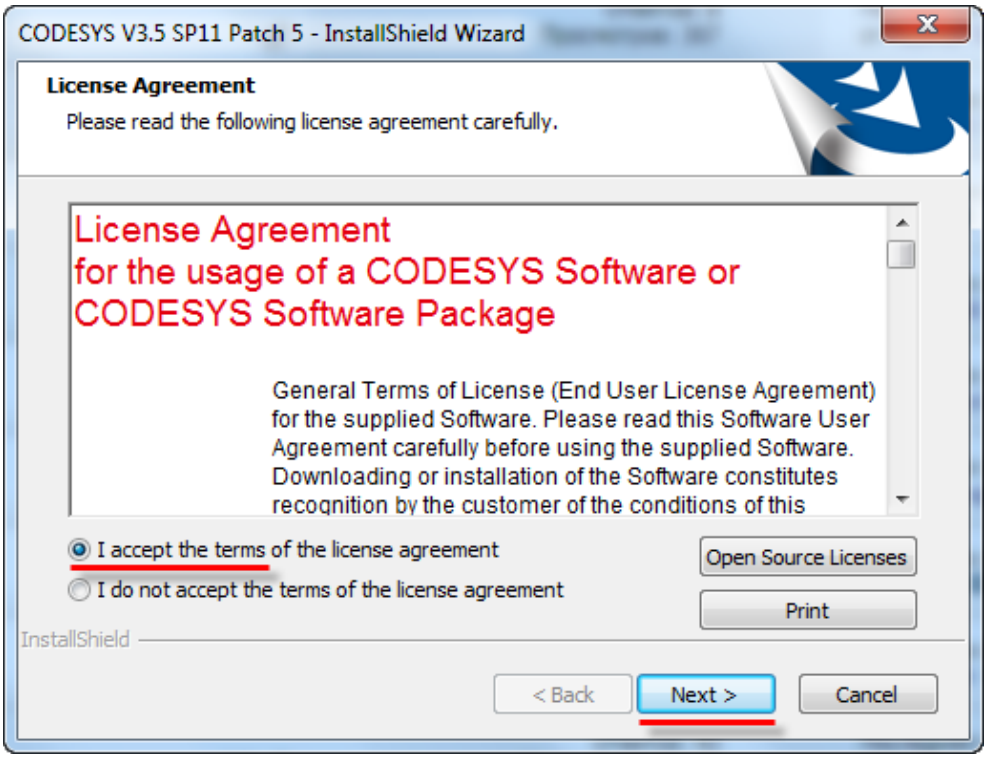


Рисунок 2.3 – Текст лицензионного соглашения

4. Затем следует указать папку, в которую будет установлен **CODESYS**, и нажать **Next**. Рекомендуется устанавливать разные версии среды программирования в разные папки.

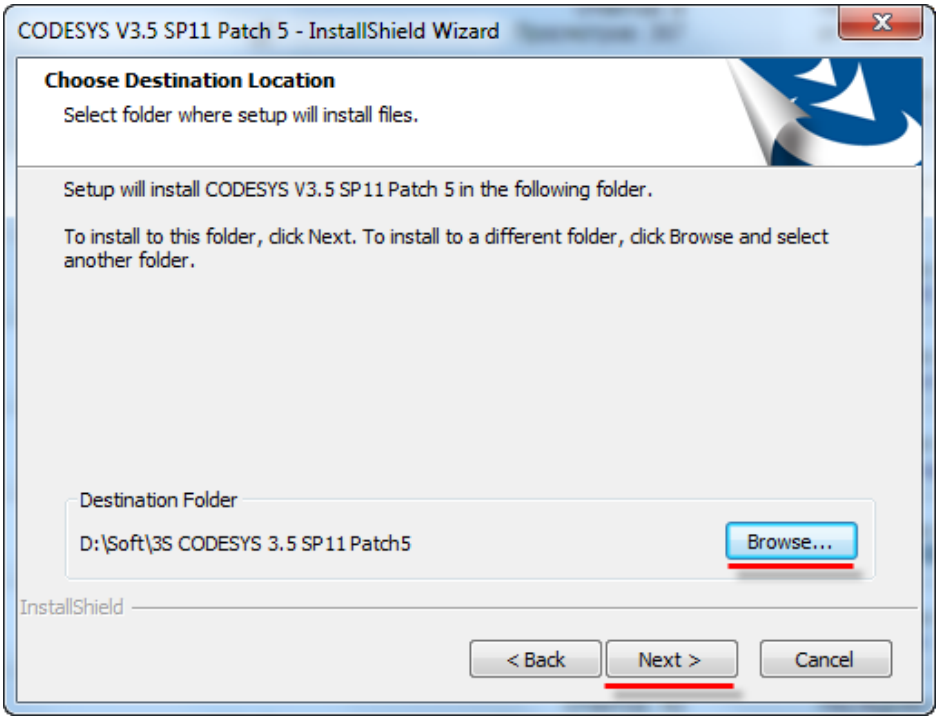


Рисунок 2.4 – Выбор папки установки CODESYS

5. В следующем окне выбираются устанавливаемые **компоненты** CODESYS. Рекомендуется устанавливать все компоненты.

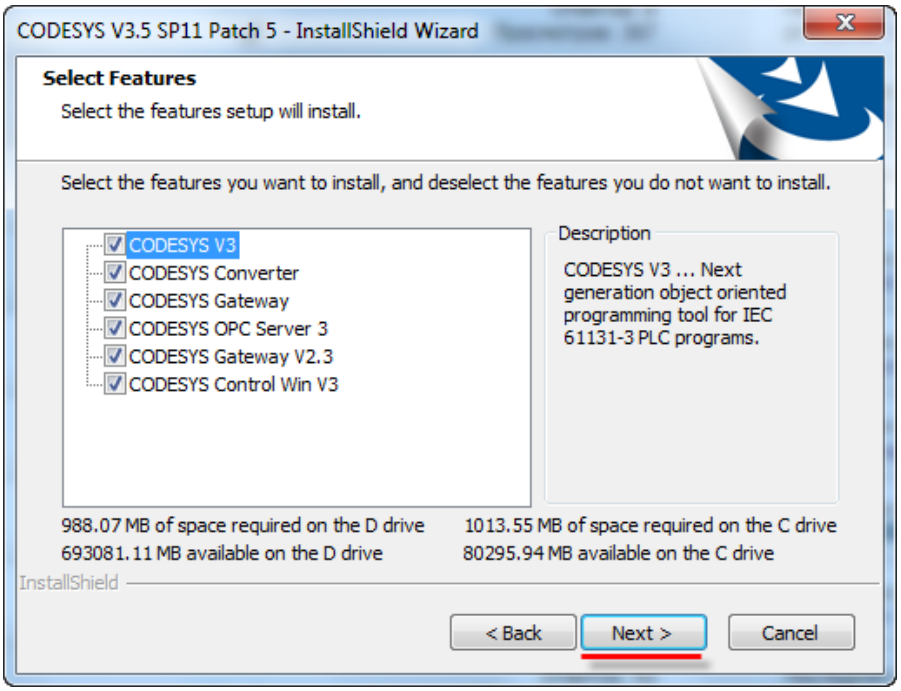


Рисунок 2.5 – Выбор устанавливаемых компонентов CODESYS

6. Затем следует выбрать папку меню **Пуск**, в которую будут добавлены ярлыки программы.

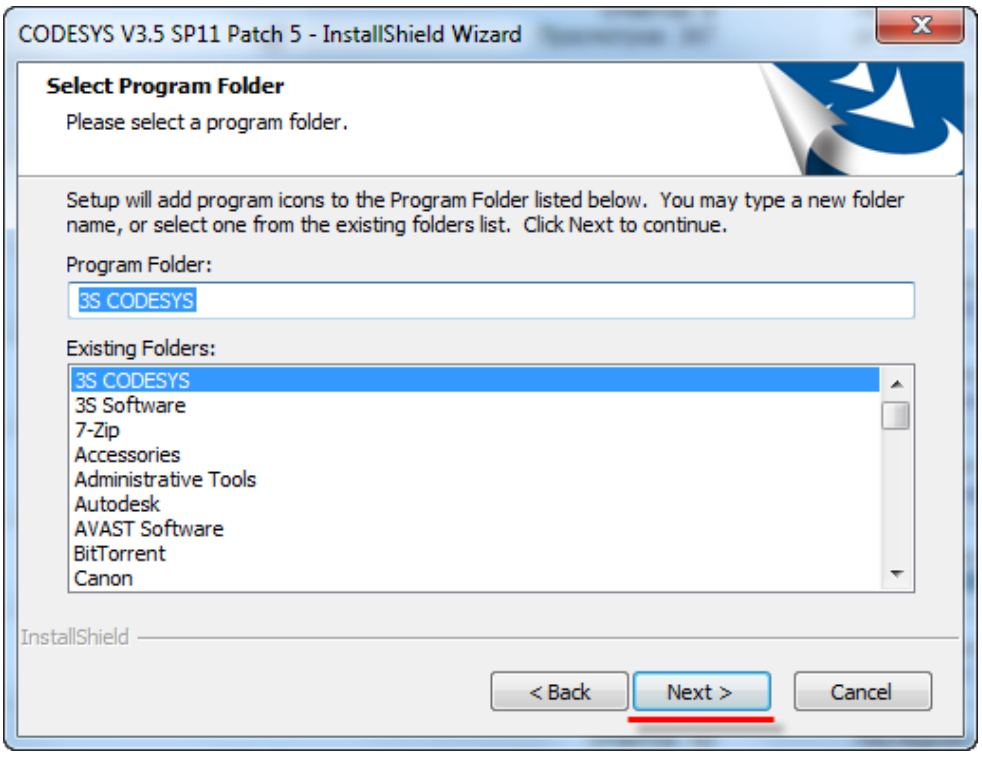


Рисунок 2.6 – Выбор папки ярлыков CODESYS

7. Для продолжения установки нажмите кнопку **Next**.

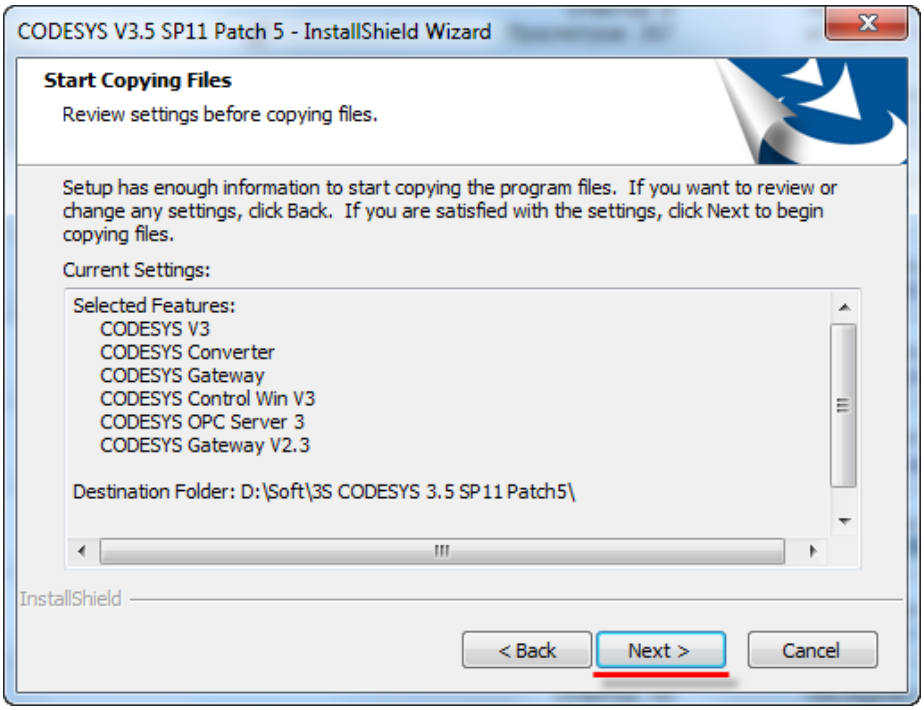


Рисунок 2.7 – Начало установки CODESYS

8. После завершения установки появится окно с информационным сообщением. Следует выбрать пункт **I have read…** и нажать кнопку **Next**.

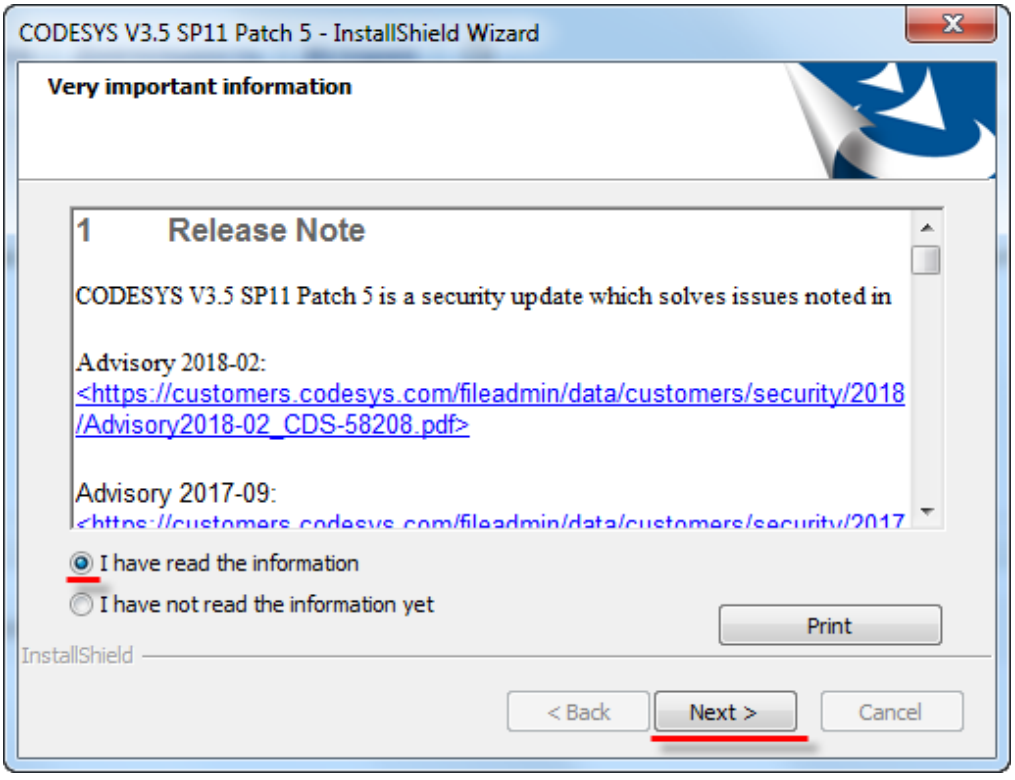


Рисунок 2.8 – Информационное сообщение CODESYS

9. Для завершения установки **CODESYS** следует нажать кнопку **Finish**.

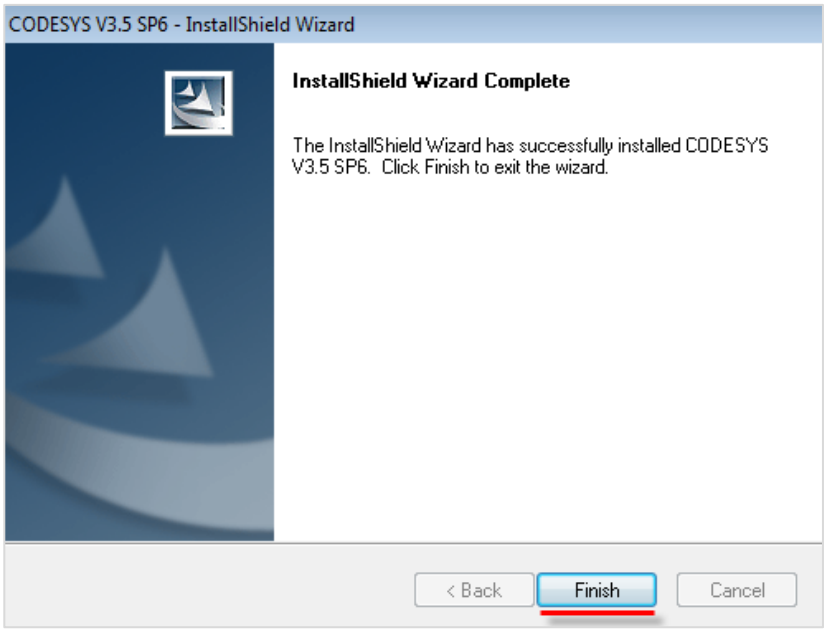


Рисунок 2.9 – Завершение установки CODESYS

Для установки дополнительных компонентов потребуется запустить **CODESYS**.

Для установки таргет-файлов следует:

1. В меню **Инструменты** выбрать пункт **Менеджер пакетов.**

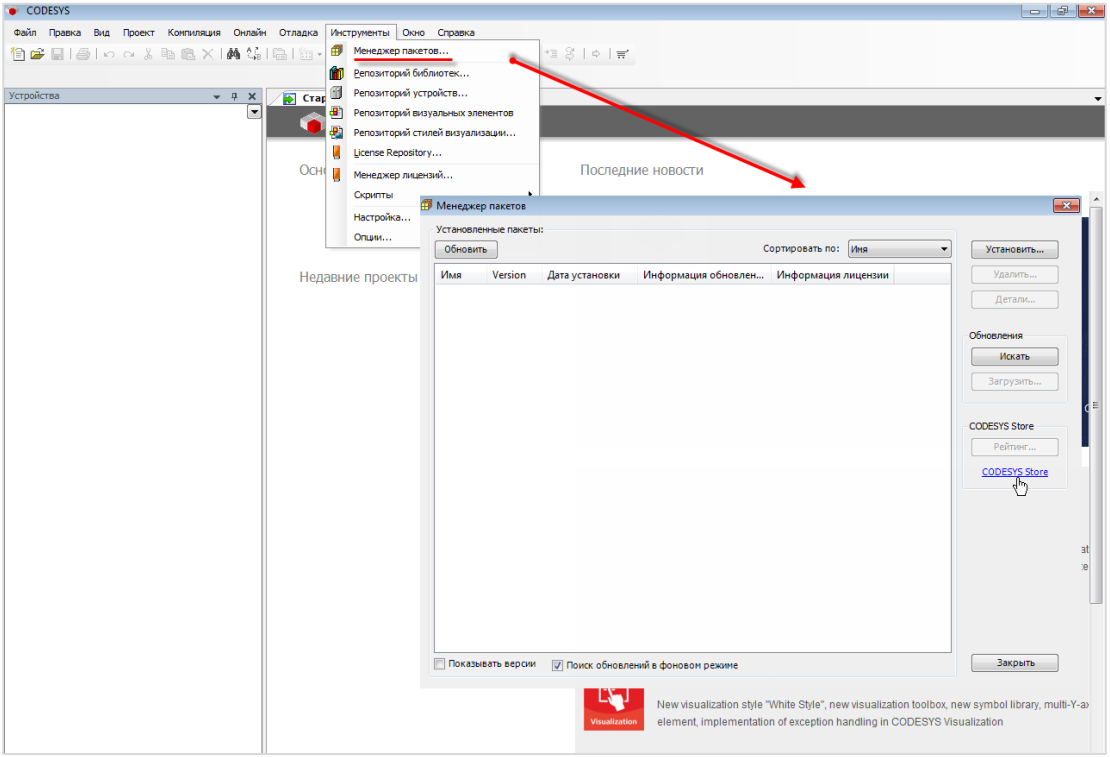


Рисунок 2.10 – Внешний вид Менеджера пакетов

2. Нажать кнопку **Установить** и выбрать пакет таргет-файлов:

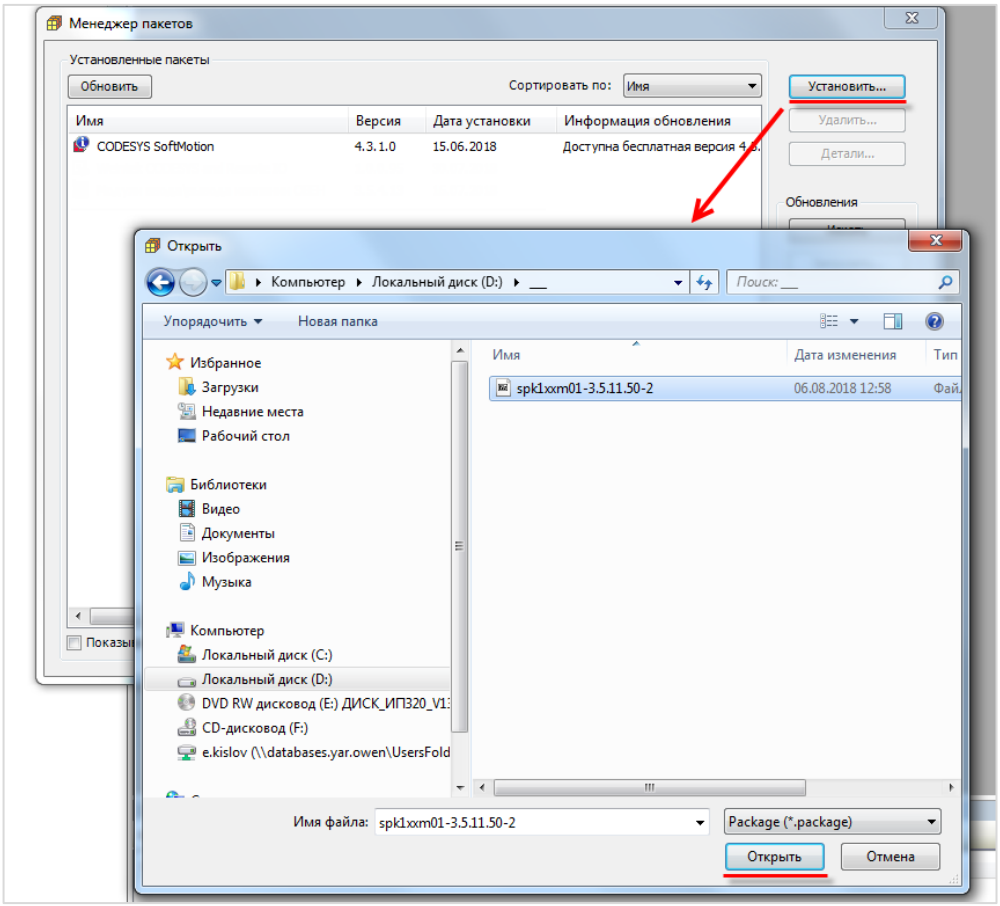


Рисунок 2.11 – Установка пакета в Менеджер пакетов

Рекомендуется выбрать режим полной установки и нажать кнопку **Next**.

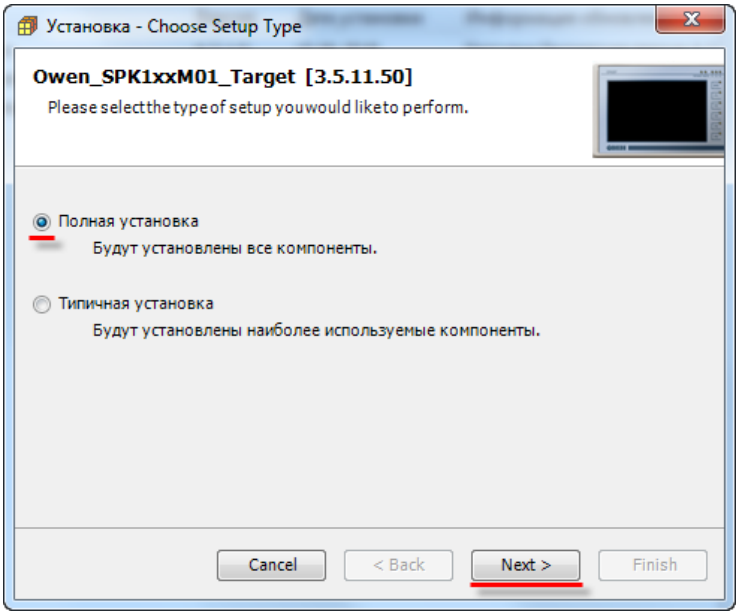


Рисунок 2.12 – Процесс установки пакета

В процессе установки пакета появится диалоговое окно установки шрифтов. Следует нажать кнопку **Next**.

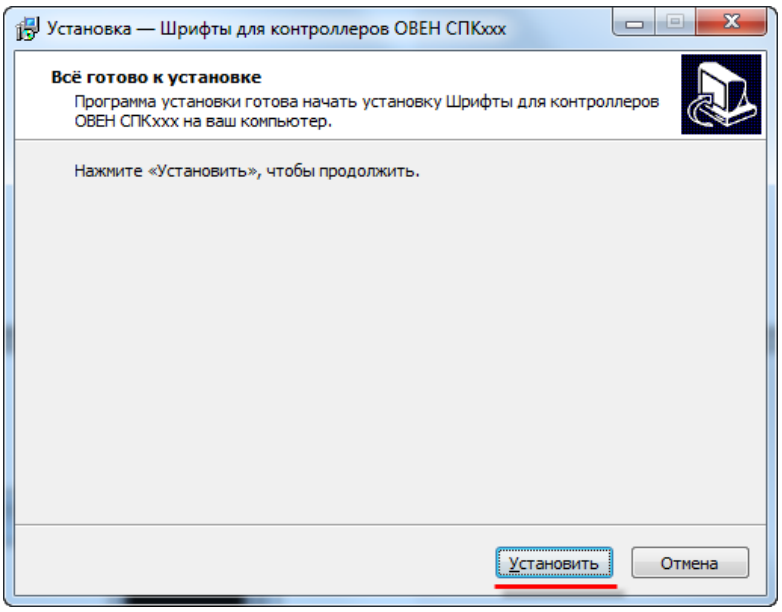


Рисунок 2.13 – Процесс установки шрифтов

Аналогичным образом (в соответствии с рисунками 2.11–2.13) устанавливаются пакеты с дополнительными компонентами.

Для установки архива репозитория следует:

1. Распаковать архив **CODESYS Repository Archive V3.5 SP4.zip**.
2. Запустить файл **CODESYS Repository Archive V3.5 SP4.msi**.
3. В появившемся окне нажать кнопку **Next:**

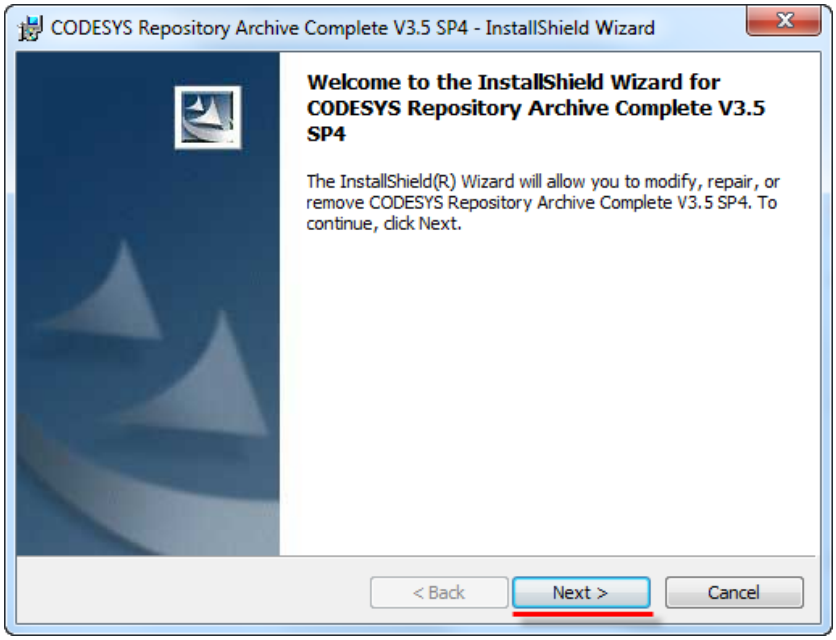


Рисунок 2.14 – Установка архива репозитория

3 Первый запуск CODESYS. Создание «пустого» проекта

CODESYS запускается двойным нажатием **ЛКМ** на соответствующий ярлык на рабочем столе:

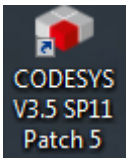


Рисунок 3.1 – Внешний вид ярлыка CODESYS

Если ярлык отсутствует, то можно воспользоваться файлом **Codesys.exe**, расположенным в папке ***...\3S CODESYS\CODESYS\Common*.**

В результате откроется **стартовое окно CODESYS**, которое имеет следующий вид:

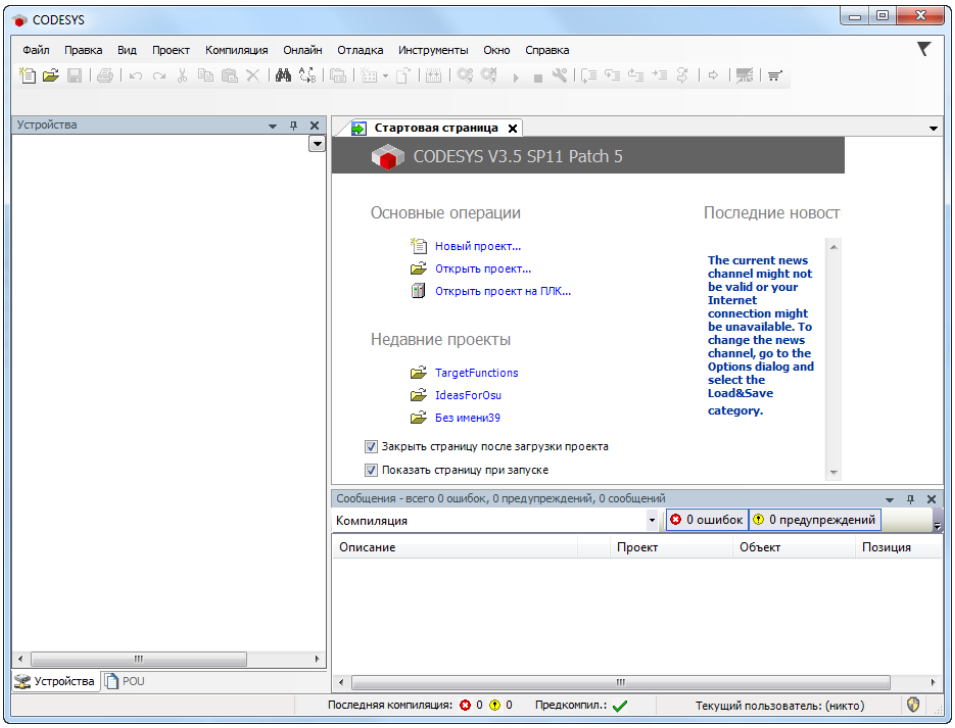


Рисунок 3.2 – Стартовое окно CODESYS

Чтобы создать новый проект следует нажать на кнопку **Новый проект** (расположена на стартовом экране и продублирована в меню **Файл**).

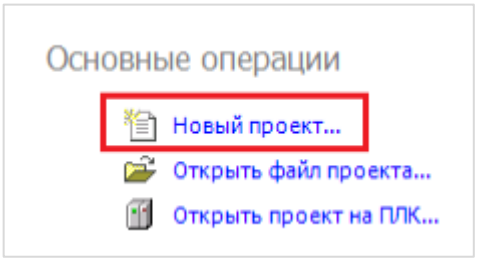


Рисунок 3.3 – Внешний вид кнопки Новый проект

Затем следует выбрать **тип** проекта (или библиотеки, если необходимо создать библиотеку), его **имя** и **папку**, в которой будут сохраняться файлы проекта. Для примера можно создать проект типа **Стандартный** с названием **FirstStart**, который будет храниться в папке ***C:\Users\Documents***:

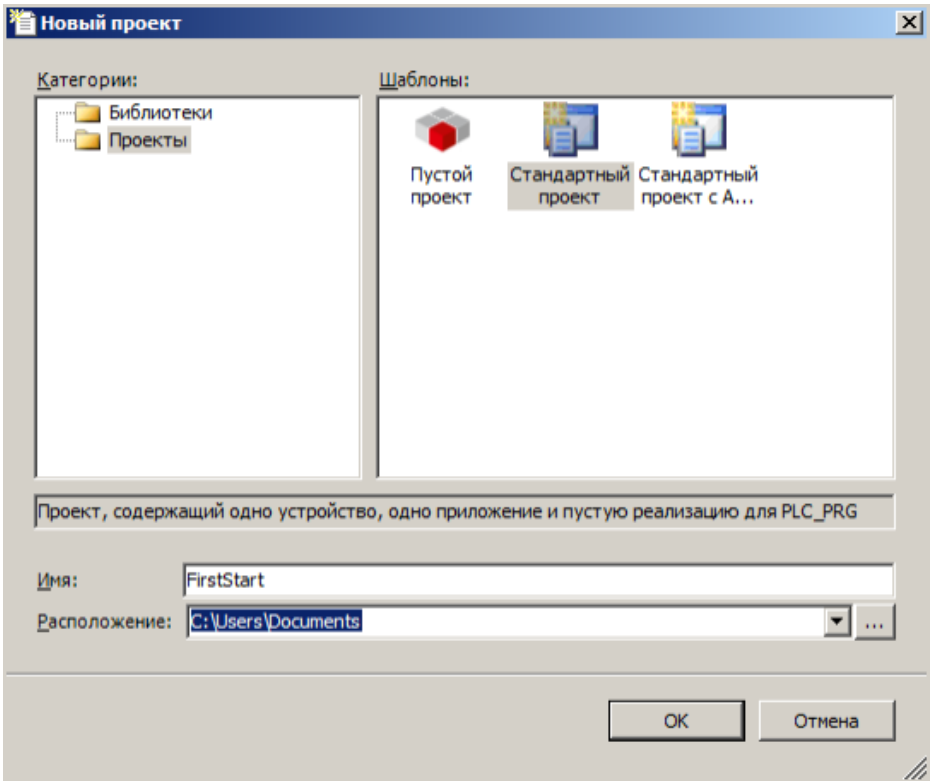


Рисунок 3.4 – Меню создания нового проекта

В появившемся меню следует выбрать **модель устройства** (контроллера) и **язык** создаваемой по умолчанию **программы**. В случае необходимости модель контроллера и язык программы ***можно поменять*** по ходу создания проекта. В рамках примера выбирается контроллер **СПК1xx [M01]** и язык **CFC**:

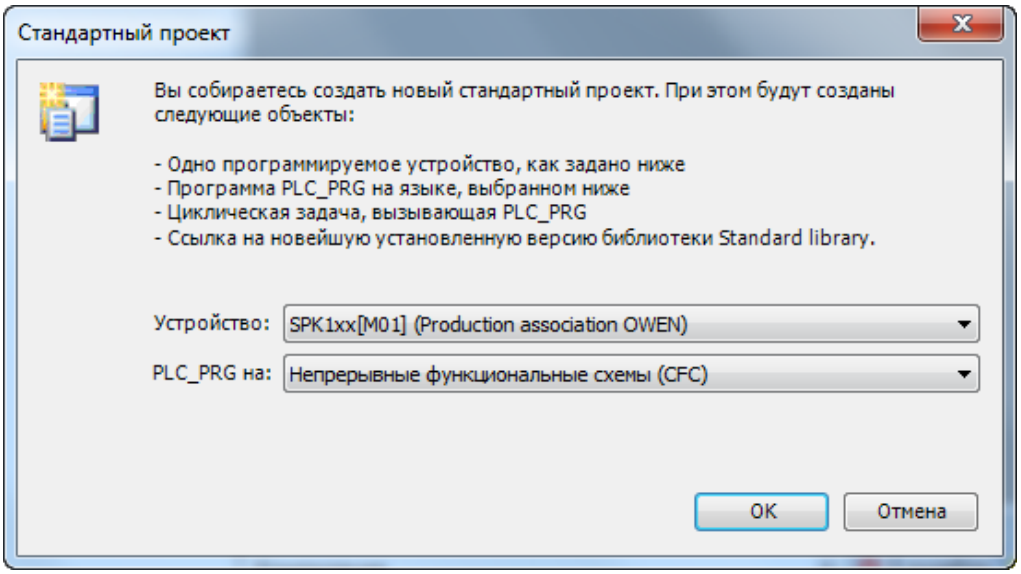


Рисунок 3.5 – Меню выбора контроллера и языка программирования

По умолчанию среда программирования запускается с **русскоязычным** интерфейсом. Язык можно поменять в меню **Инструменты**, вкладка **Опции**, пункт **Международные установки**:

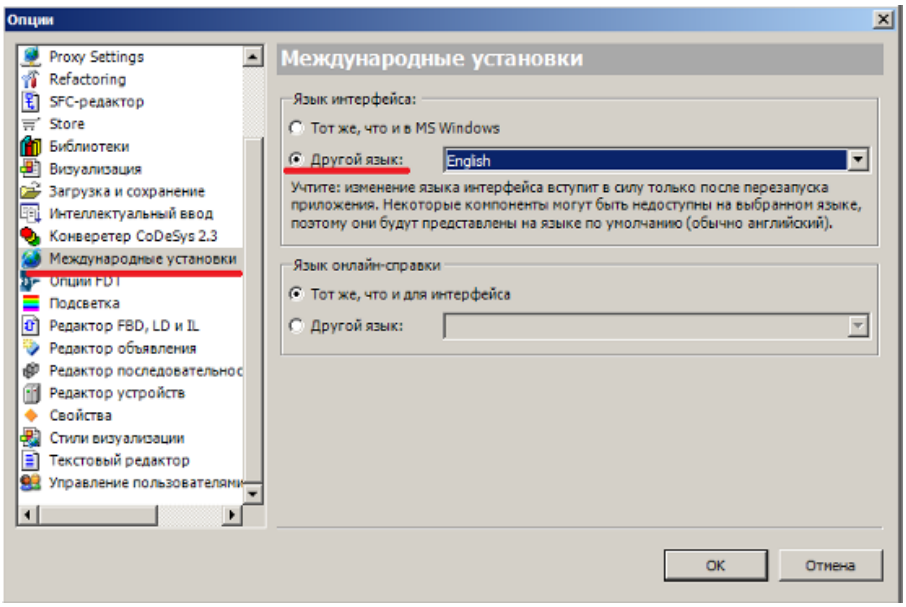


Рисунок 3.6 – Меню настроек языка CODESYS

Изменения вступят в силу ***после перезапуска*** CODESYS.

4 Интерфейс CODESYS

4.1 Компоненты интерфейса

После создания пустого проекта (см. п.3) окно **CODESYS** будет выглядеть следующим образом:

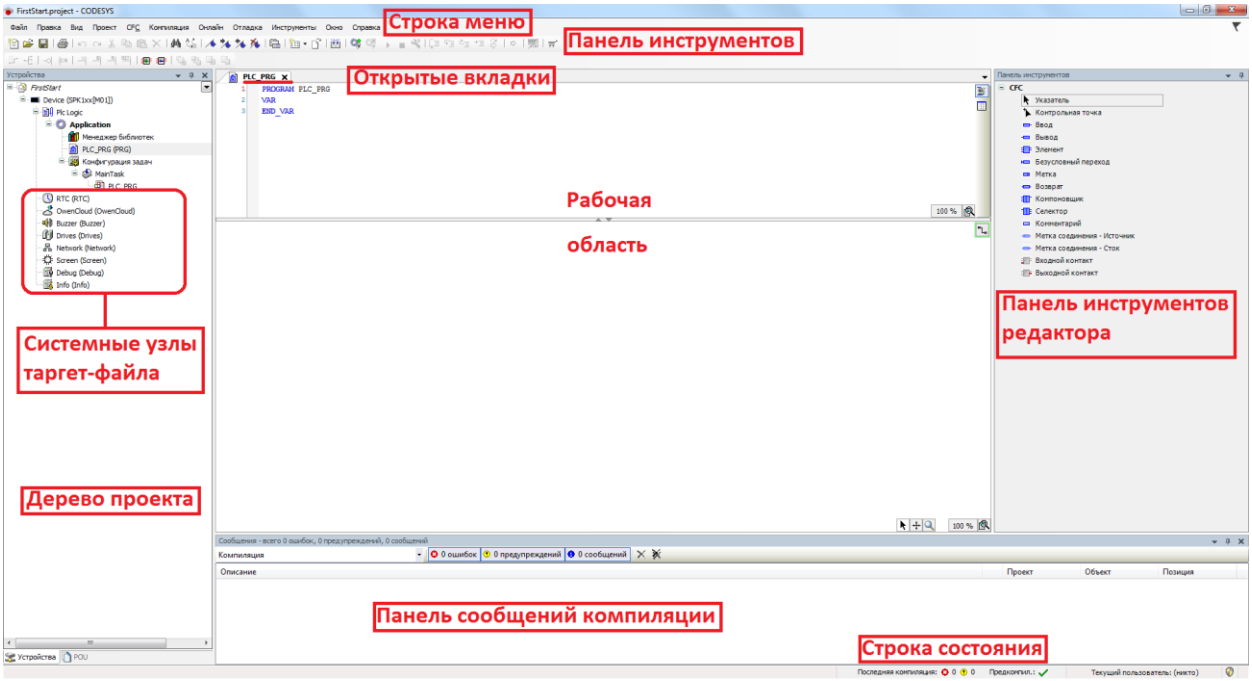


Рисунок 4.1 – Внешний вид интерфейса CODESYS

На экране расположены следующие компоненты:

1. **Строка меню** – содержит набор меню, используемых во время работы над проектом.
2. **Панель инструментов** – содержит набор ярлыков, дублирующих наиболее часто используемые пункты меню.
3. **Дерево проекта** – содержит древовидную структуру используемых в проекте компонентов.
4. **Системные узлы таргет-файлов** – позволяют настраивать системные параметры контроллера (сетевые настройки, время и т. д.). См. более подробную информацию в руководстве **CODESYS V3.5. Описание target-файлов**.
5. **Рабочая область** – содержит открытые в данный момент компоненты **дерева проекта**. В зависимости от типа компонентов может использоваться для разработки текстовых и графических программ, создания экранов визуализации, настройки проекта и т. д. Переключение между компонентами осуществляется с помощью **вкладок**, расположенных в верхней части рабочей области.
6. **Панель инструментов редактора (панель элементов)** – содержит функциональные блоки редакторов графических языков программирования и графические примитивы редактора визуализации.
7. **Панель сообщений компиляции** – содержит информацию о процессе компиляции, в частности – о возникших ошибках.
8. **Строка состояния** – содержит информацию о статусе компиляции, состоянии приложения в режиме **онлайн** (запущено/остановлено), текущем пользователе и т. д.

Подробнее о компонентах интерфейса см. в п. 7.

Интерфейс **CODESYS** характеризуется значительной гибкостью и может быть настроен как в части количества доступных пользователю меню и панелей, так и в плане количества/ расположения окон.

4.2 Панель меню

Панель меню **CODESYS** при стандартно настроенном интерфейсе имеет следующие пункты:

1. **Файл** – содержит команды для работы с файлом проекта (открытие, закрытие, сохранение и т. д.).
2. **Правка** – содержит команды для работы с текстом (копирование, вставка, удаление и т. д.).
3. **Вид** – содержит команды для работы с компонентами интерфейса, в том числе команды для открытия/закрытия рабочих панелей. Подробнее данный пункт меню рассматривается в п. 4.3.
4. **Проект** – содержит команды для работы с компонентами проекта.
5. **CFC** (название может отличаться в зависимости от используемого языка программирования) – содержит команды для работы с редактором соответствующего языка программирования. Данный пункт меню становится доступен только при выделении **POU** (программы) в дереве проекта.
6. **Компиляция** – содержит команды, используемые для компиляции проекта.
7. **Онлайн** – содержит команды, используемые для подключения к контроллеру и загрузки в него пользовательского проекта.
8. **Отладка** – содержит команды, используемые для отладки проекта.
9. **Инструменты** – содержит команды, используемые для добавления в проект вспомогательных компонентов (библиотек, target-файлов и т. д.) и настройки **CODESYS**.
10. **Окно** – содержит команды для управления открытыми окнами и панелями.
11. **Справка** – содержит команды для вызова справки по CODESYS, а также информацию о версии среды программирования.

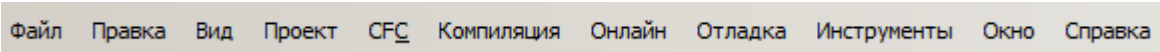


Рисунок 4.2 – Панель меню

Меню 1, 2, 10, 11 являются стандартными и не требуют отдельного рассмотрения. Меню 5, 6, 7, 8, 9 будут рассмотрены в п. 7, который посвящен аспектам разработки пользовательского проекта.

4.3 Меню «Вид»

Раскрытое меню **Вид** выглядит следующим образом:

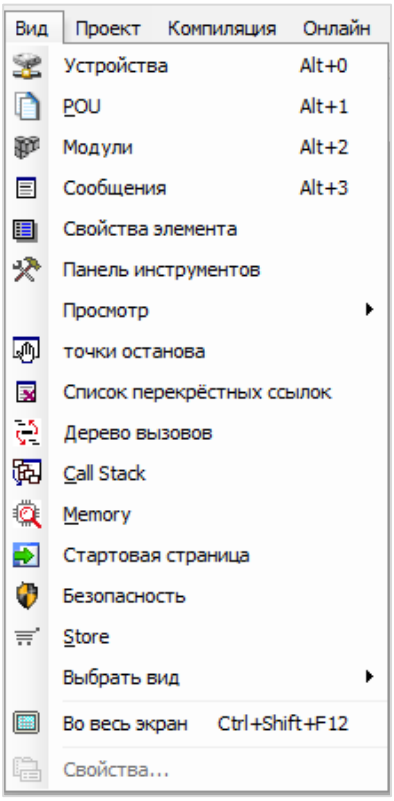


Рисунок 4.3 – Меню Вид

1. Вкладка **Устройства** – открывает **дерево проекта** (см. п. 4.1.).
2. Вкладка **POU** – открывает **Панель POU**, содержащую общие для всего проекта компоненты, которые могут использоваться на различных устройствах и в различных приложениях (в отличие от POU дереве проекта, которые строго привязаны к конкретному устройству и приложению).
3. Вкладка **Модули** – открывает **Панель модулей**. Для работы с модулями необходимо расширение **CODESYS Application Composer**, которое **не входит в комплект поставки** и может быть приобретено в **CODESYS Store** (см. ниже).
4. Вкладка **Сообщения** – открывает **Панель сообщений компиляции** (см. п. 4.1.).
5. Вкладка **Свойства элемента** – открывает **Панель свойств** для элементов визуализации.
6. Вкладка **Панель инструментов** – открывает **Панель инструментов** (см. п. 4.1.).

Описанные выше вкладки используются для настройки интерфейса.

Для переключения между панелями удобнее использовать соответствующие кнопки как на рисунке ниже.

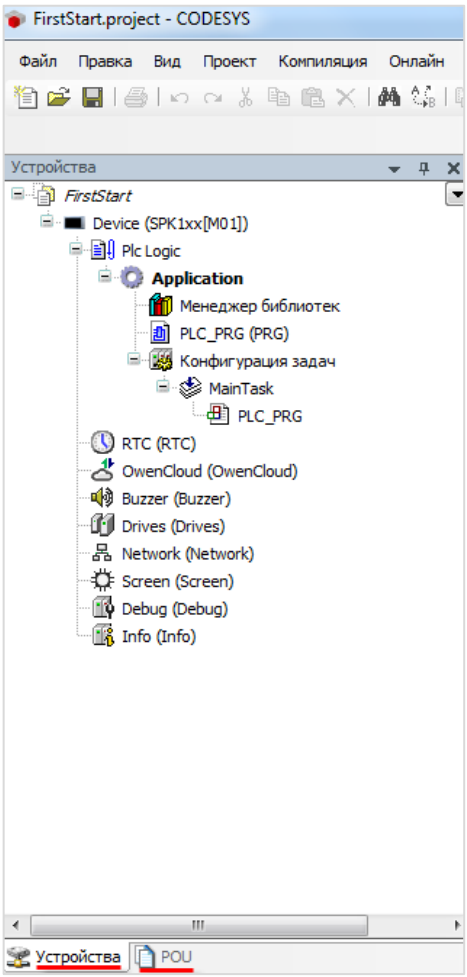


Рисунок 4.4 – Кнопки переключения между Панелью устройств и Панелью POU

7. Вкладка **Просмотр** – позволяет создавать списки переменных и просматривать в онлайн-режиме их значения (в процессе работы программы).
8. Вкладка **Точки останова** – позволяет создавать точки останова (используются при отладке проекта).
9. Вкладка **Список перекрестных ссылок** – с помощью этой команды можно открыть окно, содержащее список перекрестных ссылок переменной проекта, т. е. компонентов проекта, в которых используется данная переменная.
10. Вкладка **Стек вызовов** – позволяет контролировать процесс вызова программных модулей в онлайн-режиме.
11. Вкладка **Дерево вызовов** – отображает программную часть проекта в виде дерева вызовов программных модулей.
12. Вкладка **Memory** – позволяет в онлайн-режиме просматривать и сохранять область памяти приложения.
13. Вкладка **Безопасность** – позволяет настроить информационную безопасность контроллера и пользовательского проекта.
14. Вкладка **Стартовая страница** – открывает стартовую страницу **CODESYS**.
15. Вкладка **Store** – открывает **CODESYS Store** (интернет-магазин расширений среды **CODESYS**).
16. Вкладка **Во весь экран** – разворачивает окно **CODESYS** во весь экран.
17. Вкладка **Свойства** – открывает меню **Свойства** для выделенного компонента **дерева проекта**.

4.4 Меню «Проект»

Раскрытое меню **Проект** выглядит следующим образом:

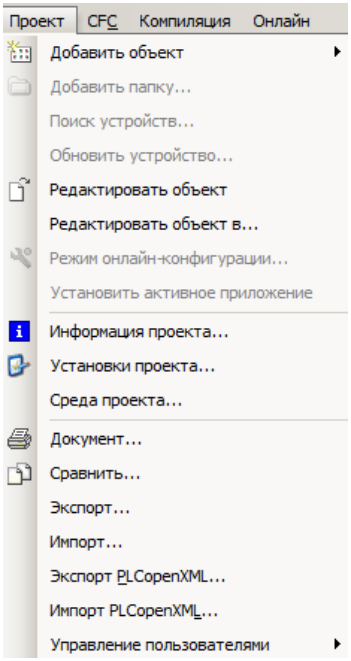


Рисунок 4.5 – Меню Проект

Вкладки **Добавить объект**, **Редактировать объект** и т. д. соответствуют вкладкам из контекстного меню компонентов **дерева проекта**:

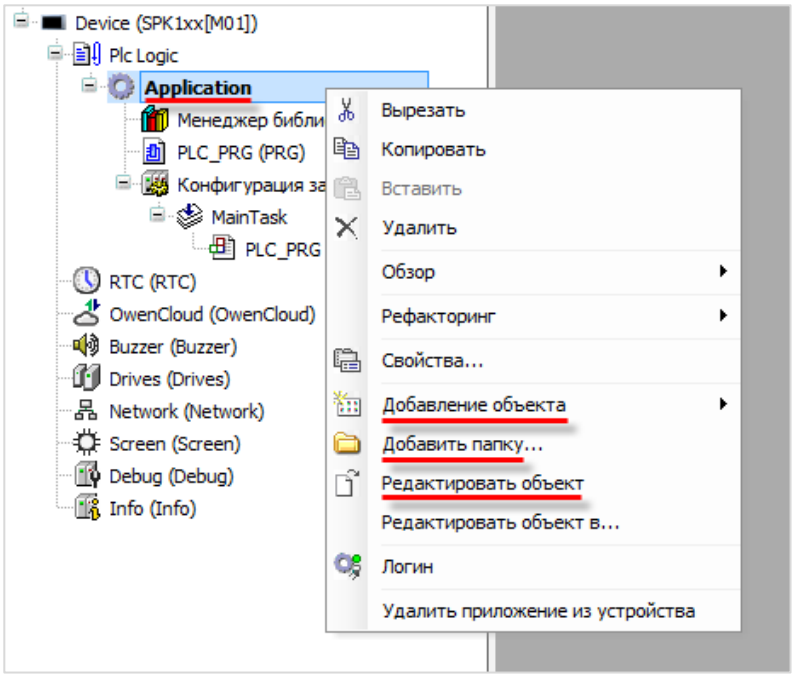


Рисунок 4.6 – Контекстное меню приложения Application

Наиболее часто используемые вкладки меню **Проект**:

1. Вкладка **Информация проекта** содержит данные о дате и времени создания проекта, пути его хранения, компании-разработчике, версии проекта и т. д.
2. Вкладка **Установки проекта** содержит общие настройки проекта, касающиеся особенностей компиляции, визуализации, ограничения прав пользователей и т. д.

5 Настройка связи между контроллером и ПК

Основными интерфейсами для подключения контроллера к ПК являются Ethernet и USB.

5.1 Настройка связи между контроллером и ПК по Ethernet

5.1.1 Сетевые настройки СПК1xx [M01]

Варианты взаимного сетевого расположения контроллера и компьютера:

1. Контроллер и компьютер находятся в **одной локальной сети**.
2. Контроллер и компьютер находятся в **разных локальных сетях**, связанных с помощью соответствующих сетевых устройств (маршрутизаторов).

Пример первого варианта подключения – порты Ethernet контроллера и компьютера соединяются напрямую с помощью кросс-кабеля (**не входит в комплект поставки**).

ПРИМЕЧАНИЕ

В локальных сетях не должно возникать конфликта IP-адресов, т. е. разные устройства не должны обладать совпадающими адресами.

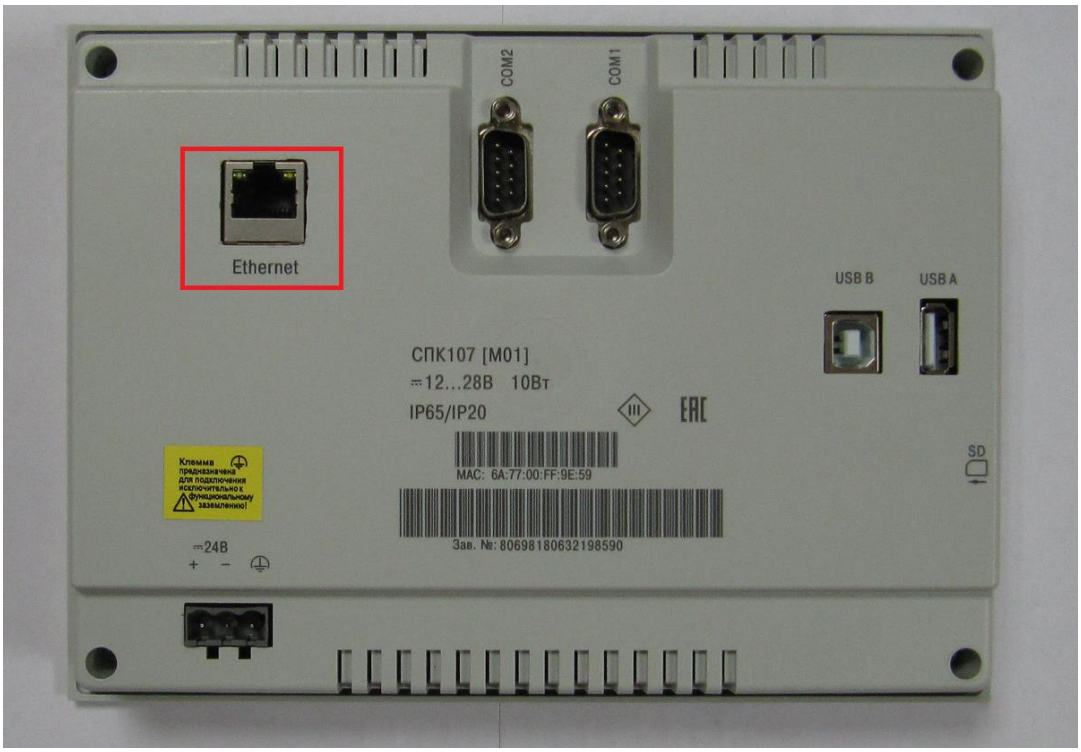


Рисунок 5.1 – Внешний вид задней панели СПК1xx [M01] (порт Ethernet выделен красным)

Рисунок 5.2 – Внешний вид кросс-кабеля

Сначала следует настроить **сетевые параметры** контроллера.

По умолчанию сетевые настройки контроллера следующие:

- режим DHCP – **отключен**;
- IP-адрес: 192.168.0.10;
- Маска подсети: **255.255.0.0**;
- IP-адрес шлюза: **192.168.0.1**.

В качестве примера будет рассмотрен процесс подключения к компьютеру контроллера с сетевыми настройками по умолчанию.

Чтобы изменить сетевые настройки контроллера следует после включения питания ***до начала загрузки проекта*** контроллера открыть **сервисное меню**, коснувшись экрана три раза.

Рисунок 5.3 – Процесс включения контроллера

Для запуска **программы-конфигуратора** следует выбрать соответствующий пункт меню:

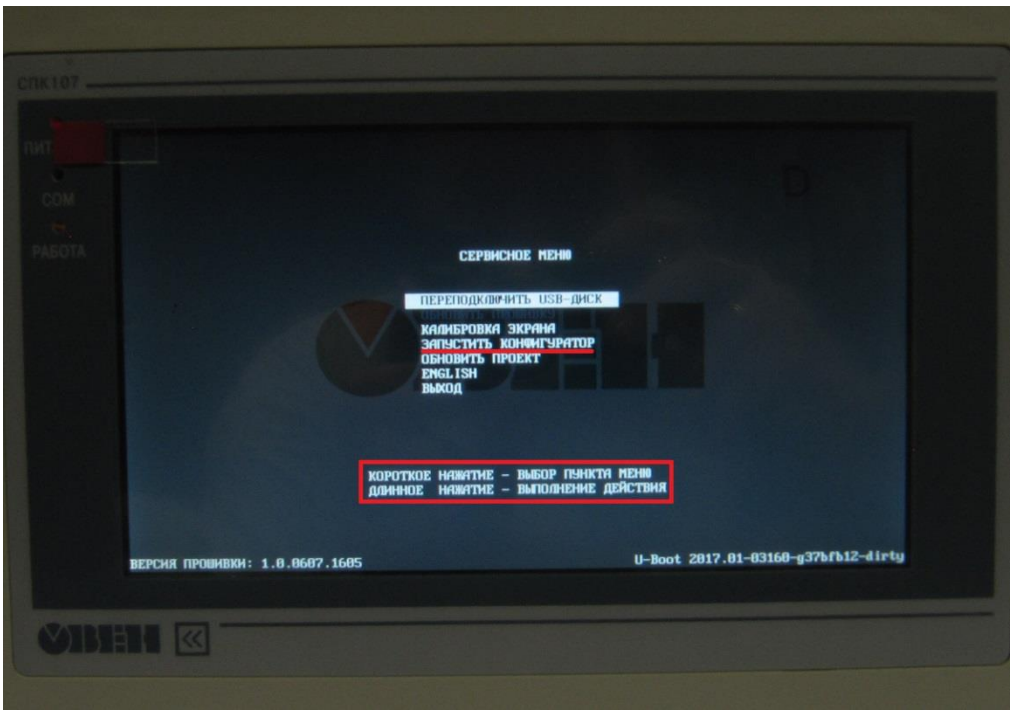


Рисунок 5.4 – Сервисное меню контроллера

ПРИМЕЧАНИЕ
Подробное описание сервисного меню приведено в документе **CODESYS V3.5. FAQ**.

Появится стартовое окно программы-конфигуратора. После нажатия на кнопку **Введите пароль** откроется соответствующее диалоговое окно с экранной клавиатурой.

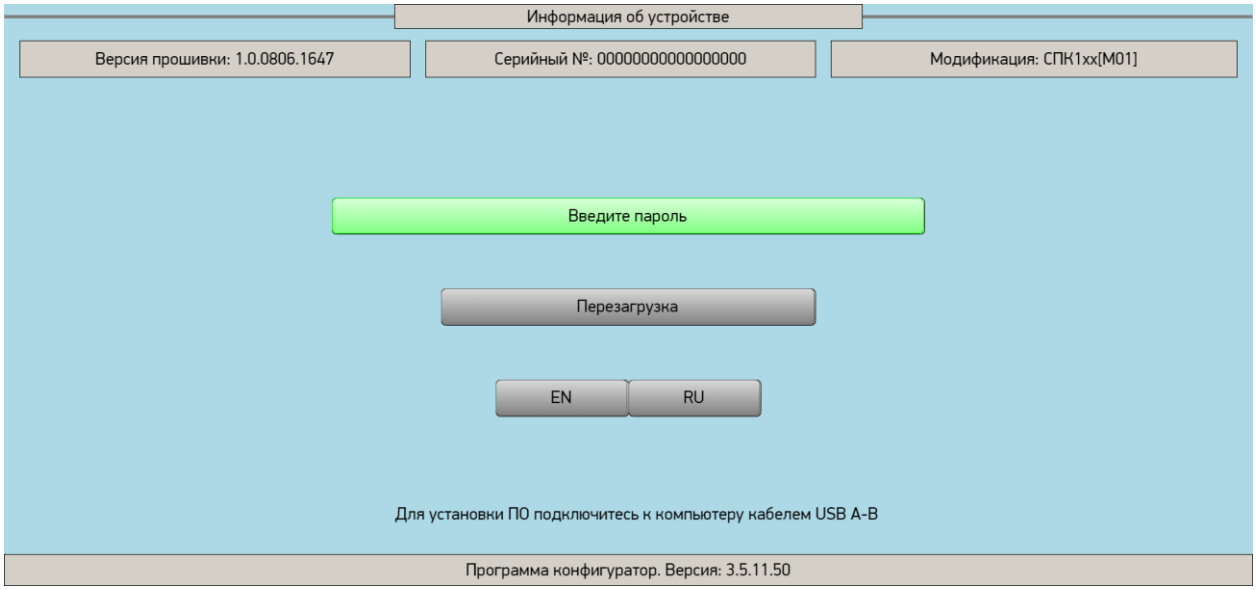


Рисунок 5.5 – Стартовое окно программы-конфигуратора

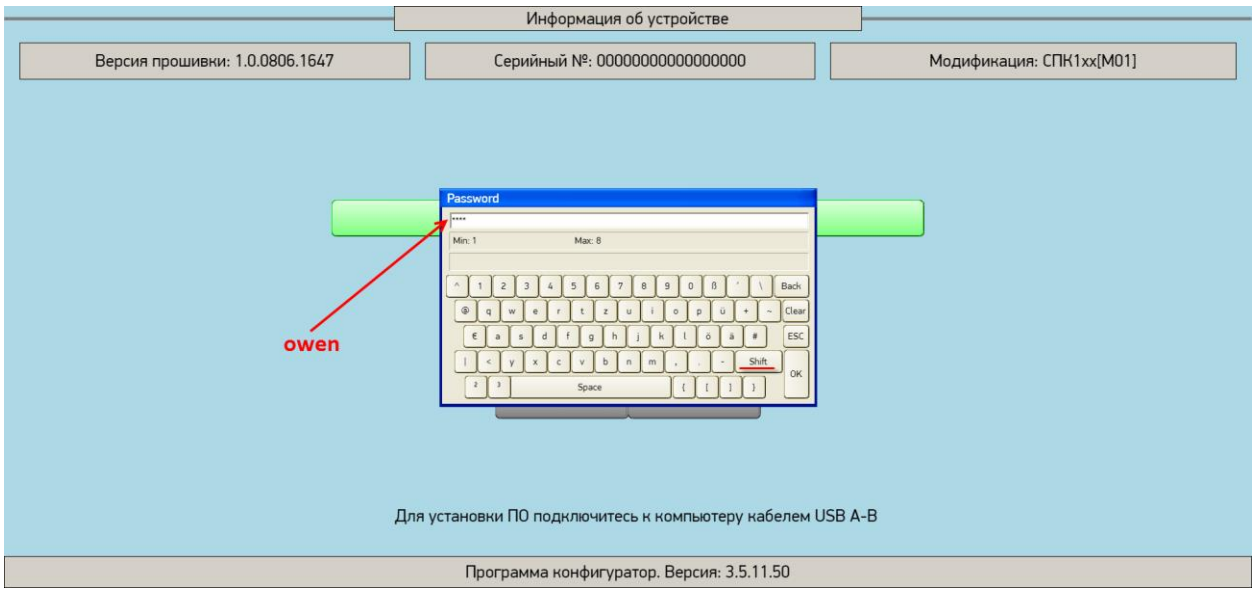


Рисунок 5.6 – Диалоговое окно ввода пароля

Пароль по умолчанию — **owen**. Экранная клавиатура переключается на нижний регистр клавишей **shift**. После ввода пароля следует нажать клавишу **OK**.

Откроется окно конфигуратора, которое имеет следующий вид:

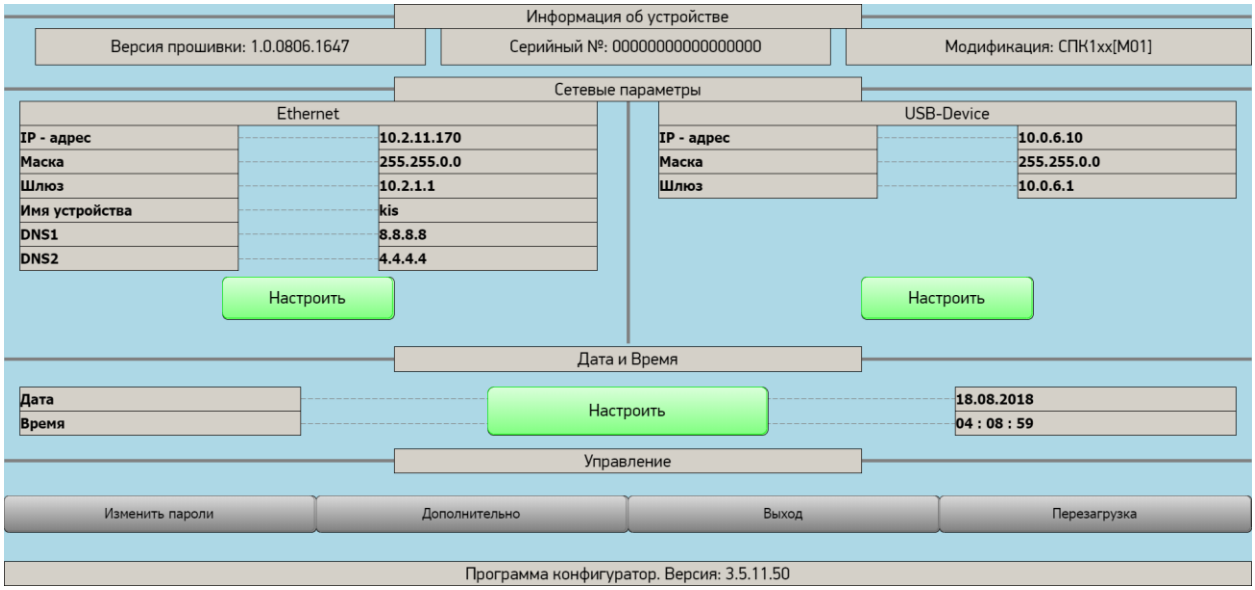


Рисунок 5.7 – Внешний вид конфигуратора

Назначение конфигуратора:

1. Настройка сетевых параметров контроллера.
2. Настройка даты и времени.
3. Отображение информации о контроллере.
4. Установка паролей.

ПРИМЕЧАНИЕ
Подробное описание конфигуратора приведено в руководстве **CODESYS V3.5. FAQ**.

Для настройки сетевых параметров следует нажать кнопку **Настроить** под соответствующим заголовком.

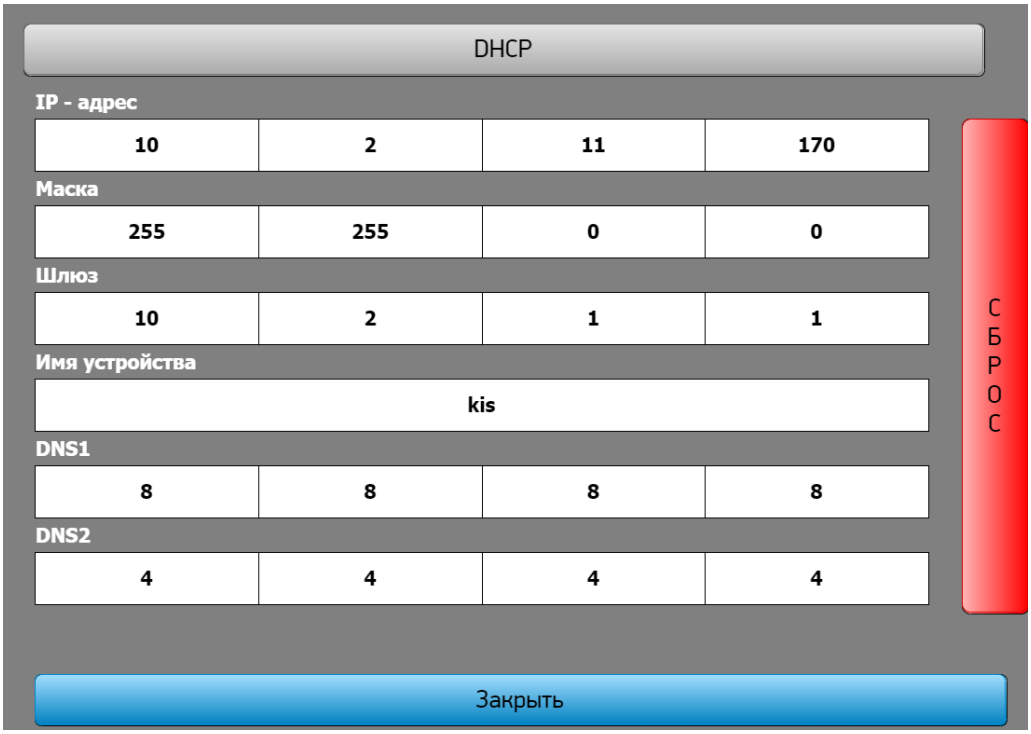


Рисунок 5.8 – Меню настройки сетевых параметров

Параметры можно изменить ***нажатием на каждую цифру***, что приводит к появлению экранной клавиатуры. Нажатие на кнопку **DHCP** включает и отключает **режим DHCP** (зеленый цвет кнопки соответствует включенному состоянию, серый — отключенному). Режим DHCP позволяет получить все сетевые настройки ***автоматически,*** если в пользовательской сети есть **DHCP-сервер**. В противном случае настройки нужно менять вручную.

Нажатие на кнопку **Сброс** возвращает сетевые настройки к их значениям **по умолчанию**.

После завершения работы с конфигуратором следует перезагрузить контроллер с помощью нажатия на кнопку **Перезагрузка** в конфигураторе, ***не заходя в сервисное меню***. В результате после загрузки контроллера запустится пользовательский проект или появится надпись **Отсутствует загрузочное приложение**.

5.1.2 Сетевые настройки компьютера

Для настройки сетевых параметров ПК с операционной системой Windows 7 следует открыть Центр управления сетями и общим доступом (**Пуск — Панель управления — Центр управления сетями и общим доступом**) и выбрать вкладку **Изменение параметров адаптера**.

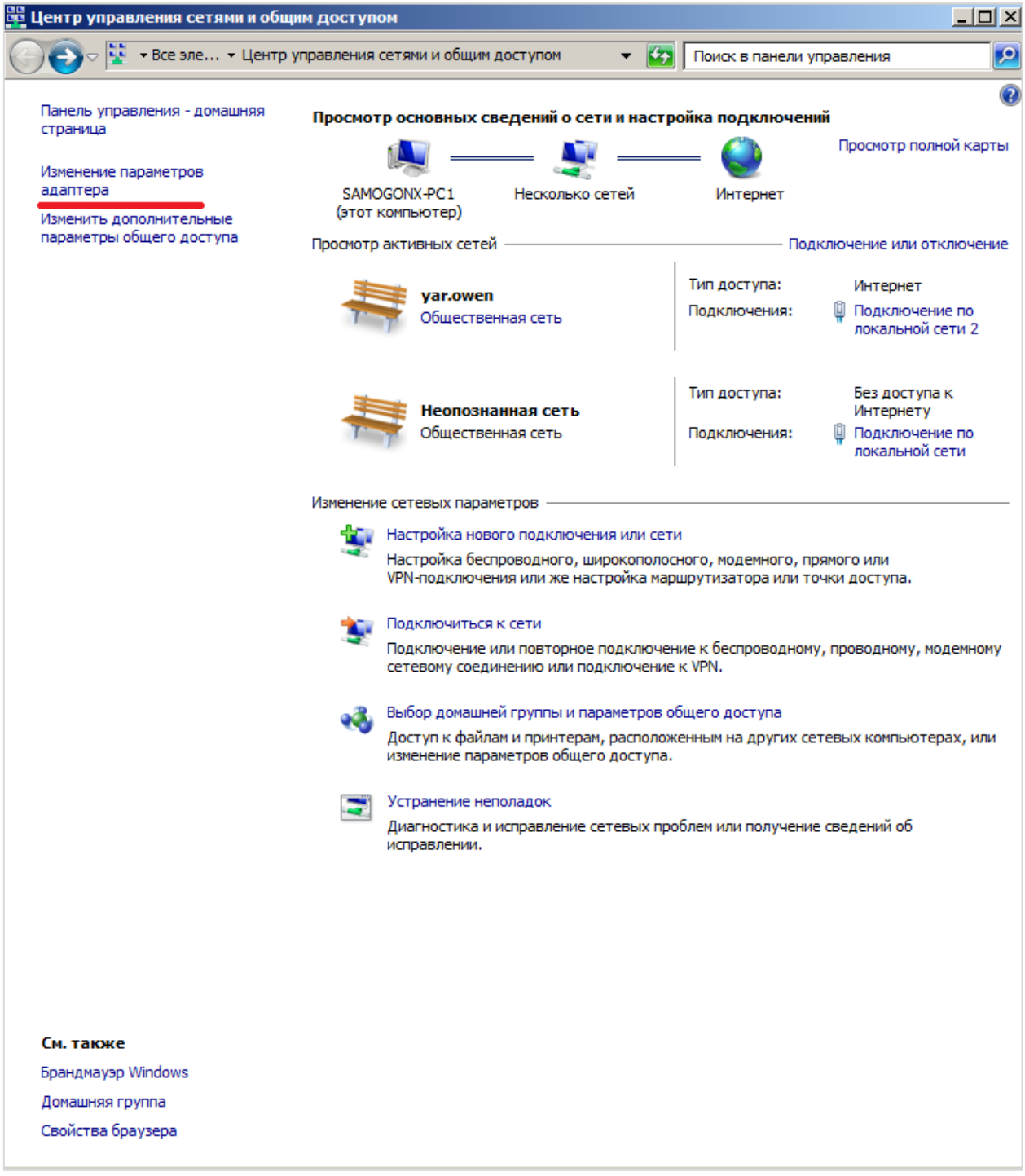


Рисунок 5.9 – Окно Центра управления сетями и общим доступом

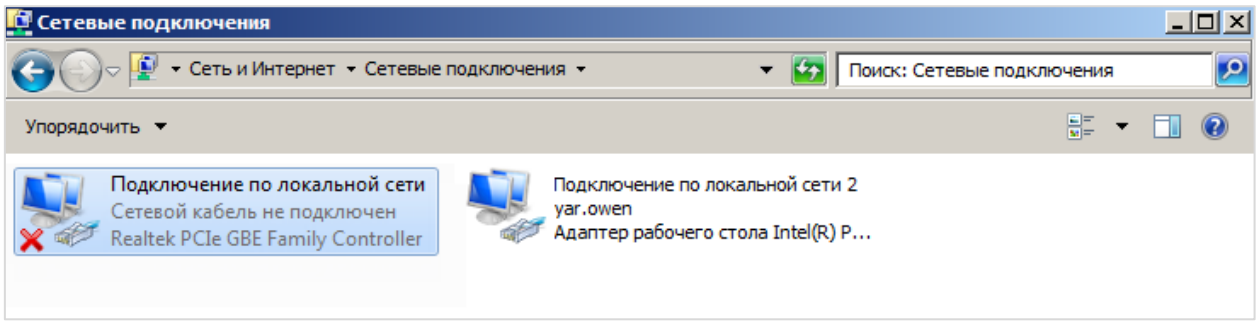


Рисунок 5.10 – Вкладка Изменение параметров адаптера (Сетевые подключения)

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

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

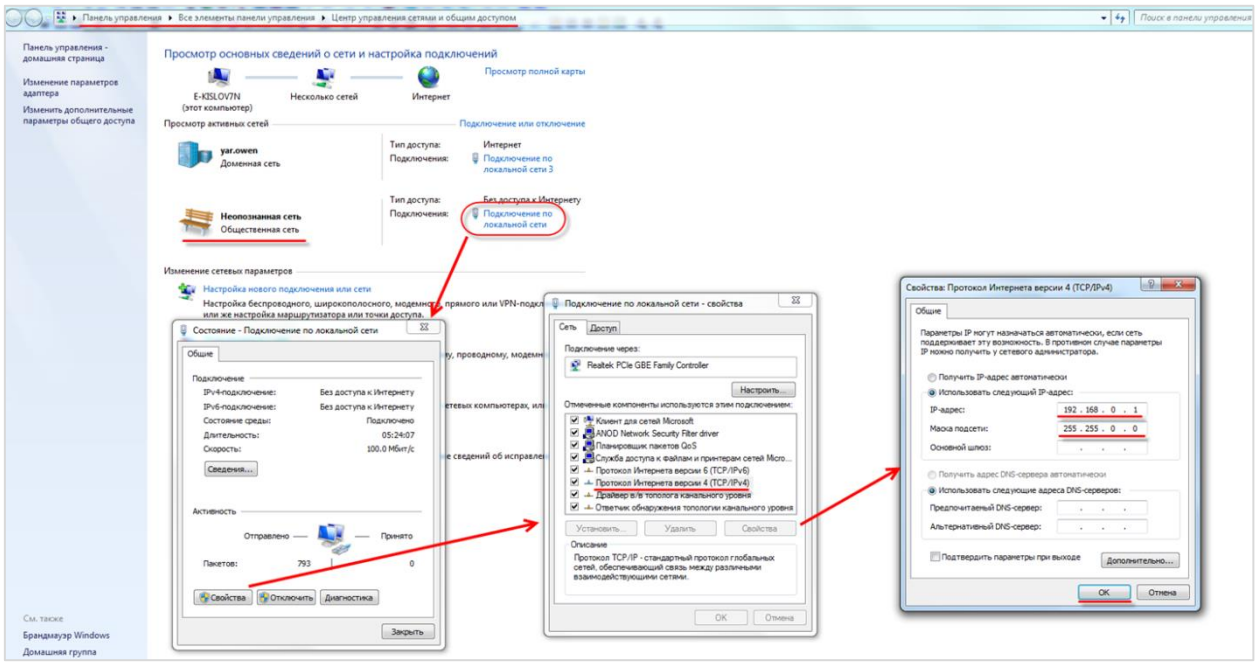


Рисунок 5.11 – Сетевые настройки ПК

Для ПК 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.

5. Настройка связи между контроллером и ПК

Чтобы проверить наличие связи между ПК и контроллером следует открыть **командную строку** (**Пуск** — **Все программы** — **Стандартные** — **Командная строка**) и ввести команду **ping 192.168.0.10 -t**.

Если связь есть, то результат будет следующим:

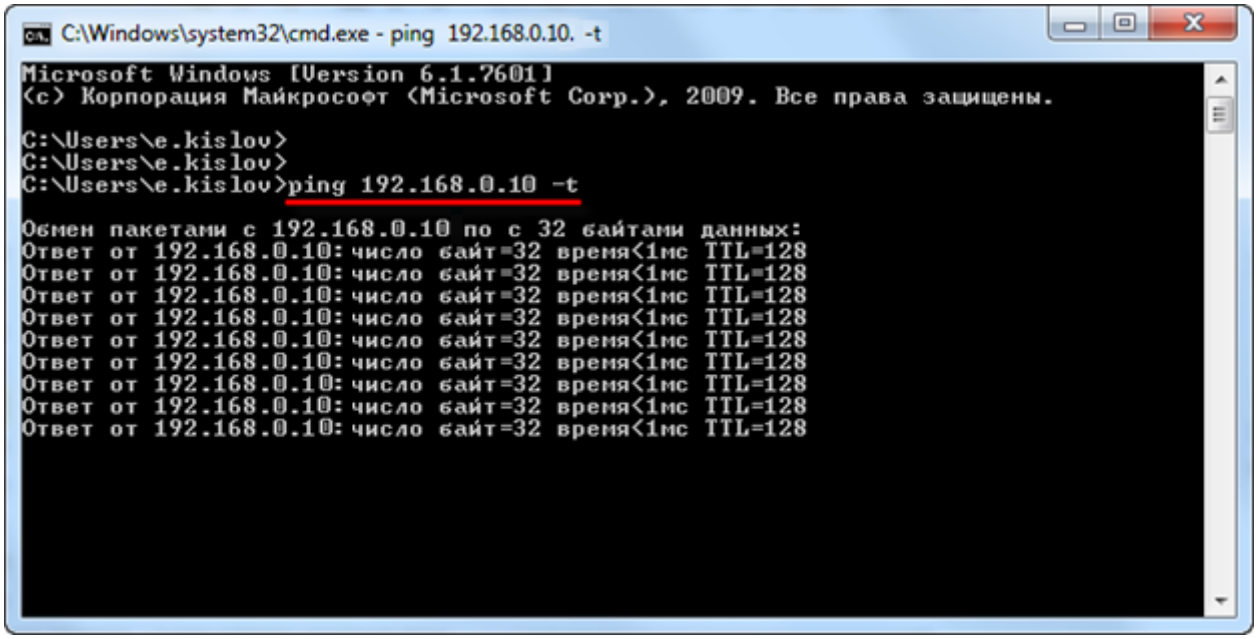


Рисунок 5.12 – Результат выполнения команды ping

5.2 Настройка связи между контроллером и ПК по USB

Связь между контроллером и пользовательским ПК осуществляется по интерфейсу **USB** с использованием кабеля типа **A – B** из комплекта поставки:

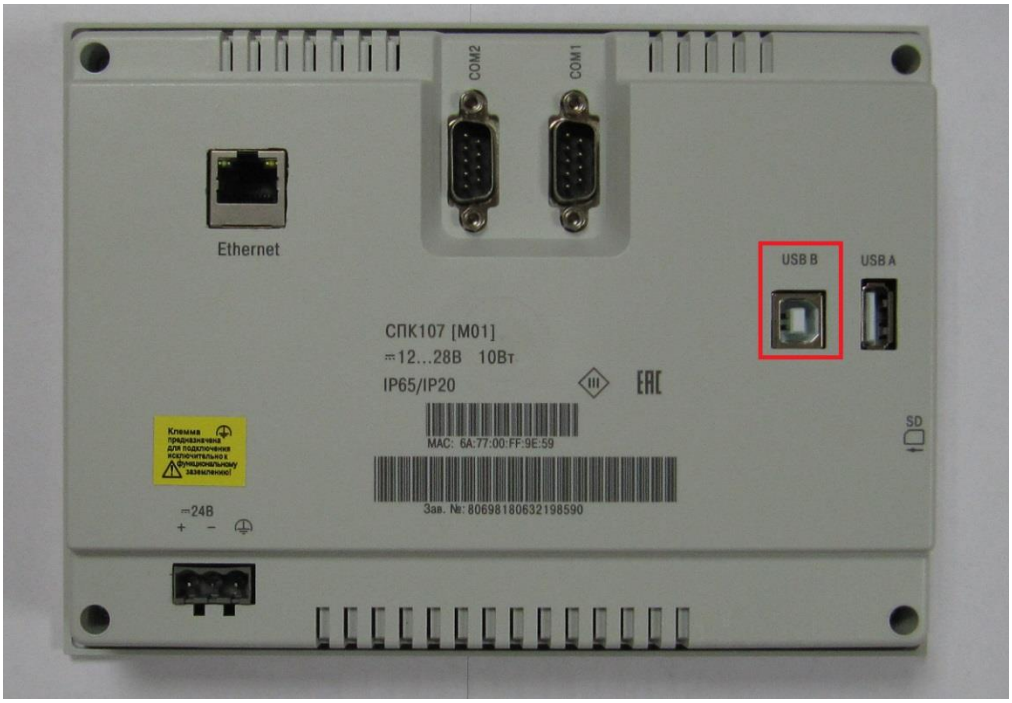


Рисунок 5.13 – Внешний вид задней панели СПК1xx [M01] (порт USB B обведен красным)

Рисунок 5.14 – Внешний вид кабеля USB A – B

В конфигураторе контроллера следует настроить сетевые параметры USB аналогично п. 5.1.1. Настройки по умолчанию:

- IP-адрес: **10.0.6.10**;
- Маска подсети: **255.255.0.0**;
- IP-адрес шлюза: **10.0.6.1.**

Во время первого подключения контроллера к ПК следует выполнить установку драйвера USB. Для этого следует запустить автоматический установщик, расположенный на диске с ПО из комплекта поставки – **SPK_USB_Driver**. Перед установкой может возникнуть следующее сообщение:

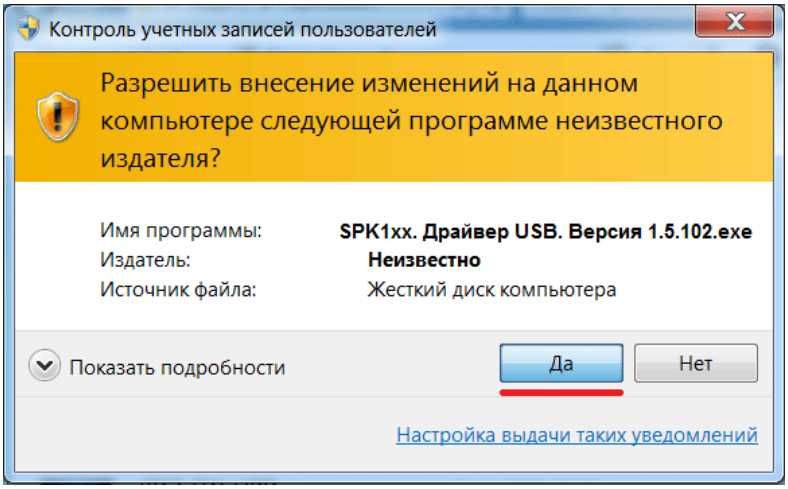


Рисунок 5.15 – Информационное сообщение перед установкой драйвера

Следует нажать **Да**.

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:

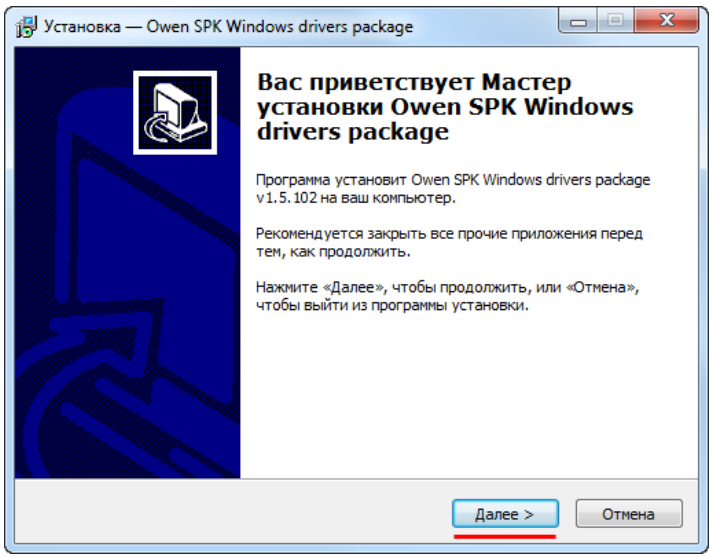


Рисунок 5.16 – Диалоговое окно Мастера установки драйверов

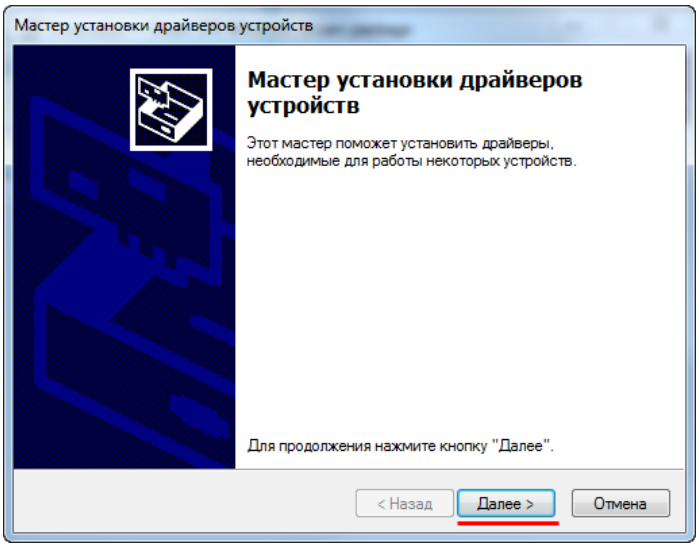


Рисунок 5.17 – Диалоговое окно Мастера установки драйверов устройств

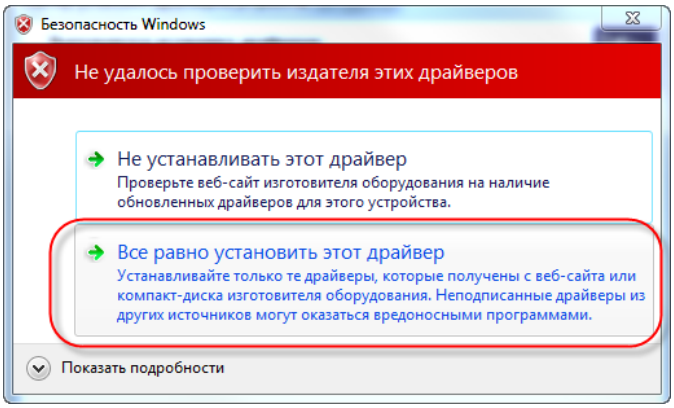


Рисунок 5.18 – Информационное сообщение брандмауэра Windows

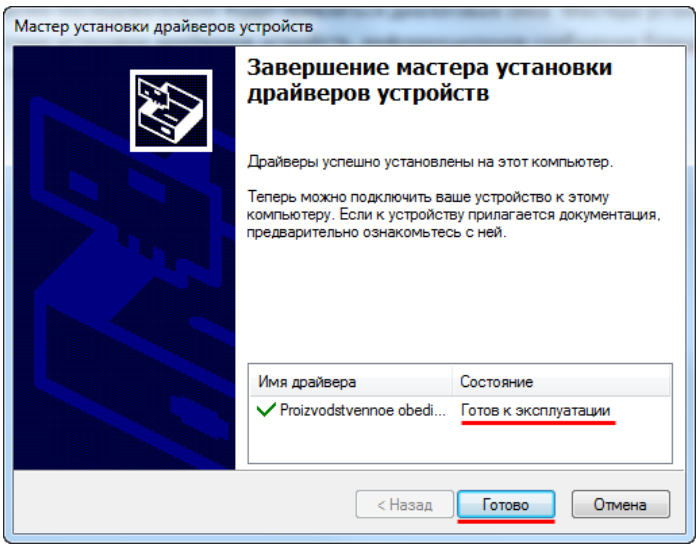


Рисунок 5.19 – Диалоговое окно завершения установки драйверов устройств

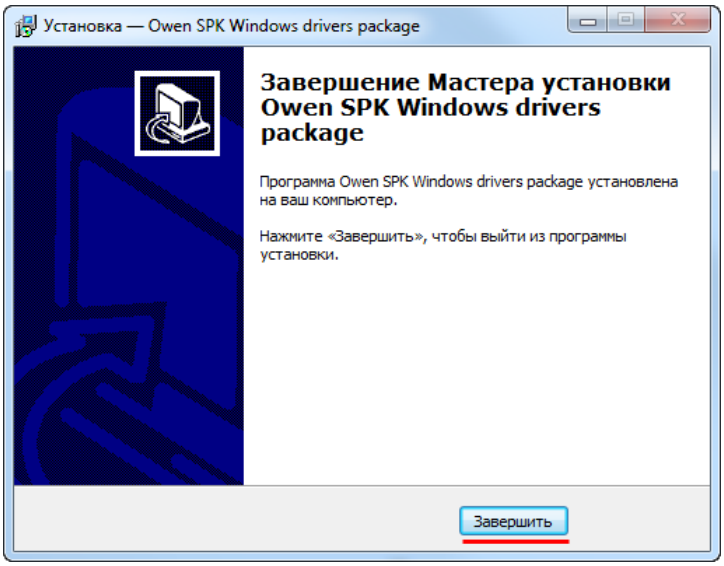


Рисунок 5.20 – Диалоговое окно завершения установки драйверов

Затем следует подсоединить конец B кабеля к USB B порту **включенного** контроллера, а конец A – к USB порту компьютера. В Диспетчере устройств (**Пуск – Панель управления – Диспетчер устройств**) появится новый сетевой адаптер – **Owen SPK**.

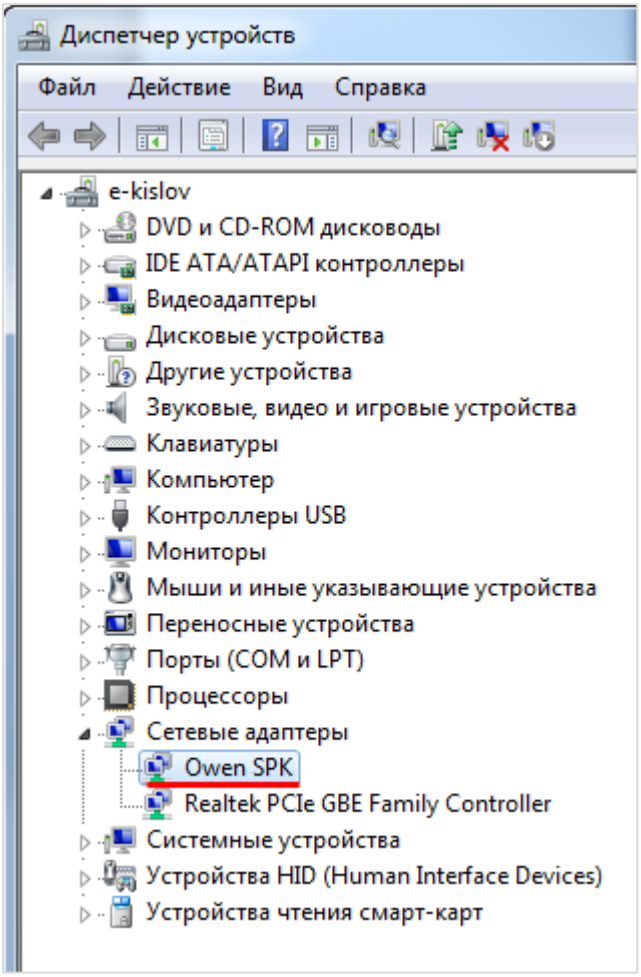


Рисунок 5.21 – Окно Диспетчера устройств после подключения СПК1xx [M01]

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

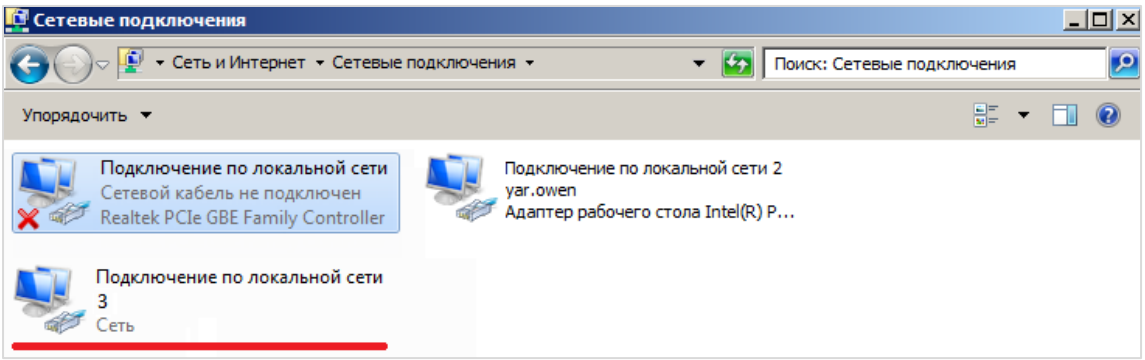


Рисунок 5.22 – Виртуальный сетевой адаптер для контроллеров ОВЕН

Настройки сетевого адаптера (по аналогии с рис. 5.11):

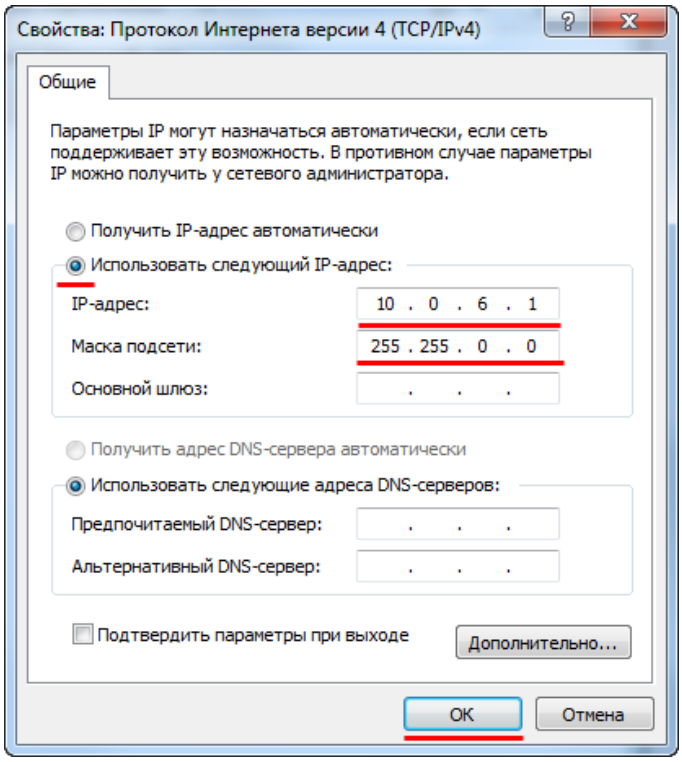


Рисунок 5.23 – Настройки виртуального сетевого адаптера для контроллеров ОВЕН

Для ПК задается IP-адрес **10.0.6.1** и маска **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** — **Стандартные** — **Командная строка**) и ввести команду:

ping 10.0.6.10 -t.

Если связь есть, то результат будет следующим:

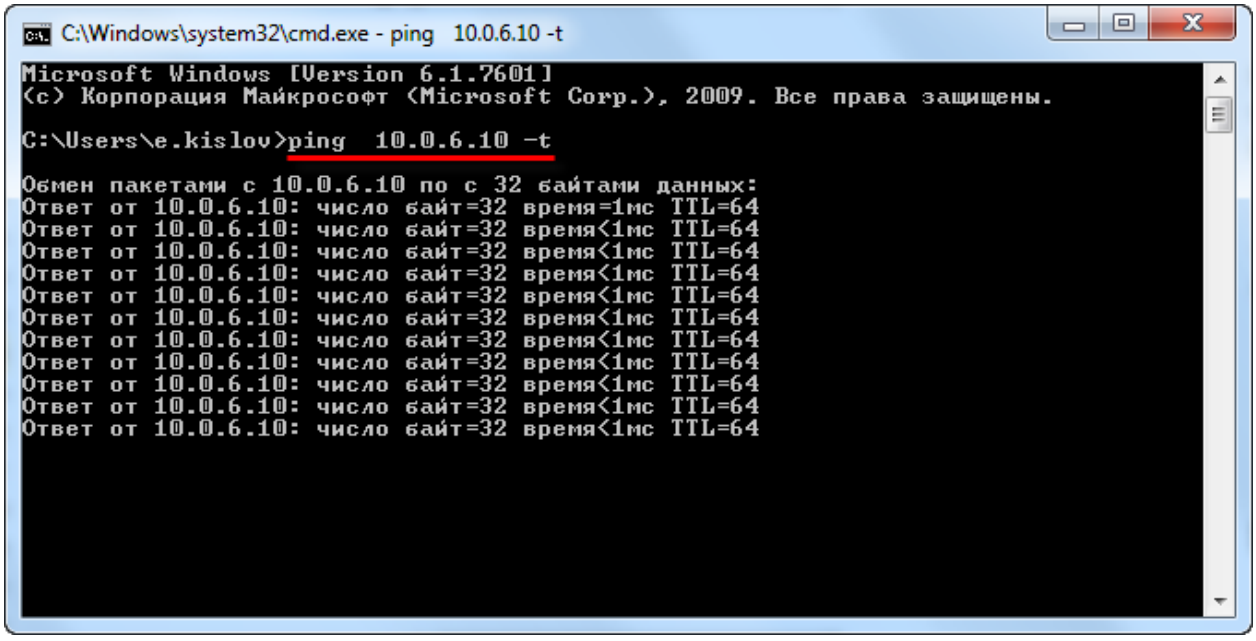


Рисунок 5.24 – Результат команды ping

5.3 Настройка связи контроллера и ПК в среде CODESYS

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

Компонент **Device** (который определяется соответствующим таргет-файлом – см. п. 1.5, п. 3) должен соответствовать модели контроллера. В случае необходимости тип устройства можно изменить, выбрав в **дереве проекта** компонент **Device** и, нажав на него **ПКМ**, открыть окно **Обновить устройство**:

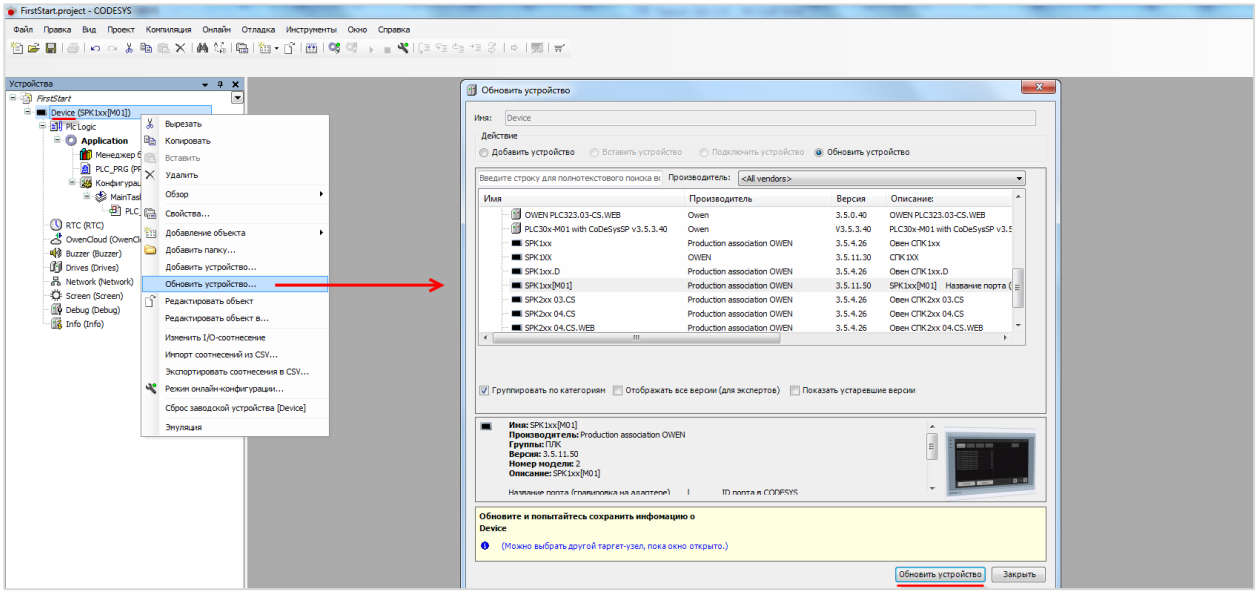


Рисунок 5.25 – Окно Обновить устройство

Затем следует выбрать устройство, соответствующее модели контроллера. **Название модели** указано в **конфигураторе** (рисунок 5.5, 5.6). Следует помнить про ***соответствие*** версии прошивки контроллера, среды программирования **CODESYS** и **таргет-файла** (см. п. 1.5).

После выбора устройства следует нажать кнопку **Обновить устройство** и закрыть окно. В **дереве проекта** у компонента **Device** отобразится название выбранного **устройства.**

Затем следует настроить **Gateway** (**шлюз**).

Двойным нажатием **ЛКМ** по компоненту **Device** (или одиночным нажатием по вкладке вверху рабочей области) можно перейти к настройкам устройства. Для создания нового шлюза следует открыть вкладку **Установки соединения**, нажать на кнопку **gateway** и выбрать пункт **Add new gateway**:

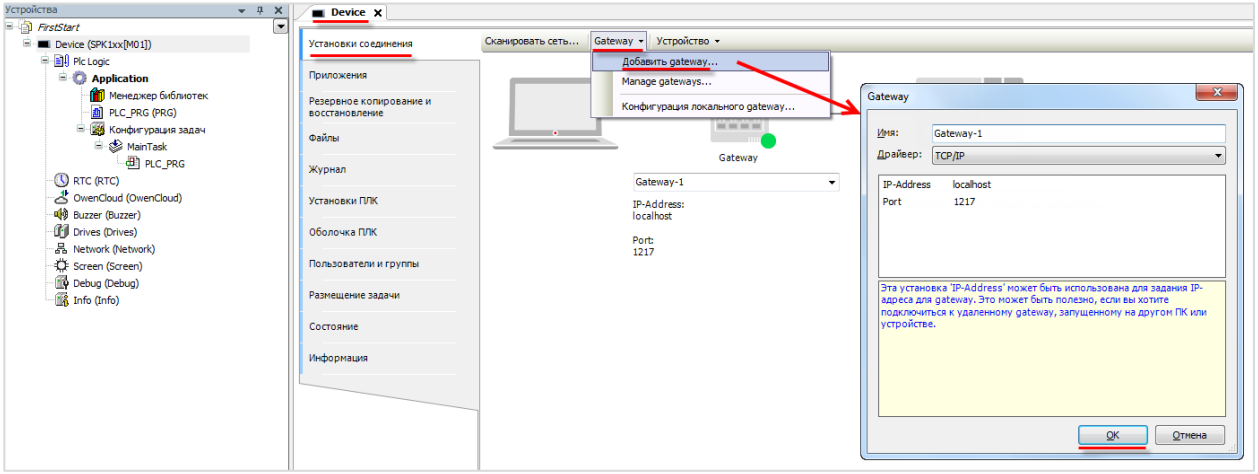


Рисунок. 5.26 – Создание нового gateway (шлюза)

Настройки рекомендуется оставить по умолчанию (имя – **Gateway-1**, IP-адрес – **localhost**). Затем закрыть окно настроек шлюза и нажать кнопку **Scan network**. В появившемся списке следует выбрать нужный контроллер и установить связь, нажав кнопку **OK**.

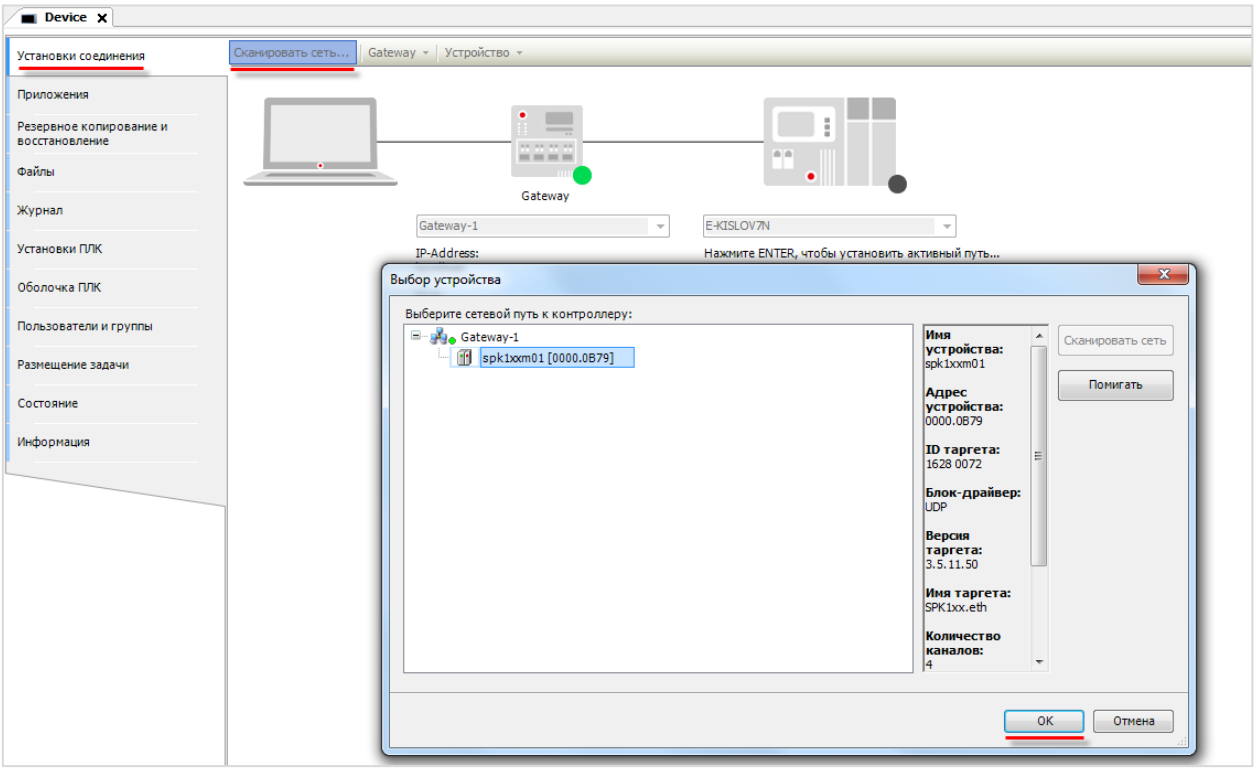


Рисунок 5.27 – Окно сканирования сети

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

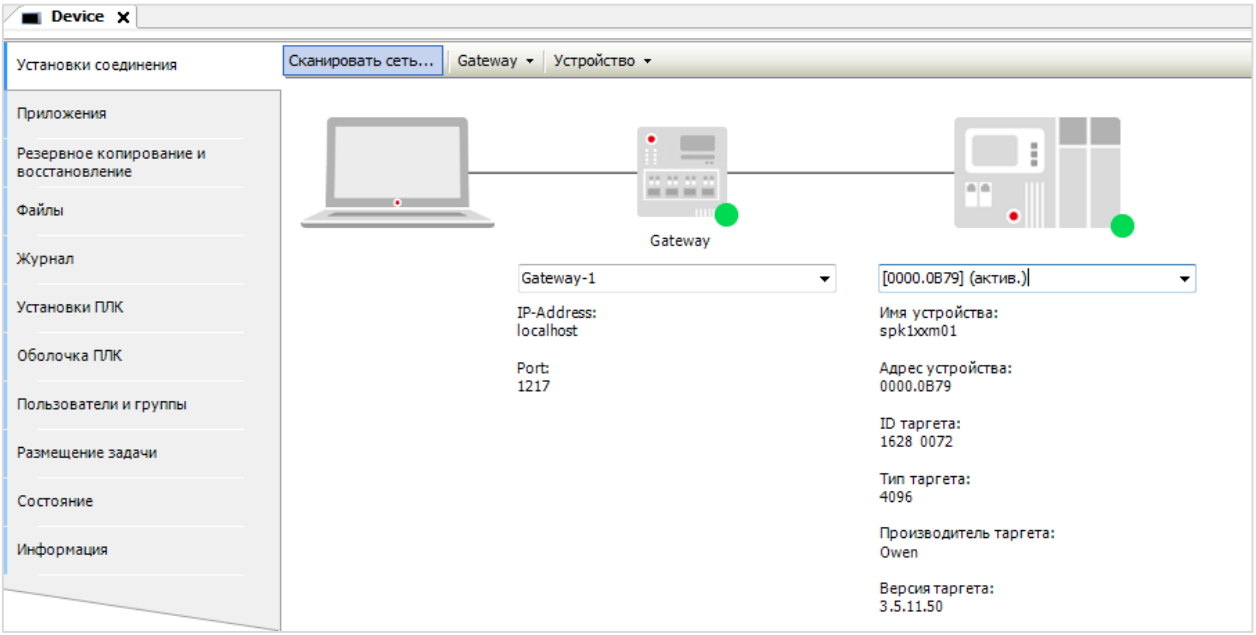


Рисунок 5.28 – Результат успешной установки связи

6 Загрузка и запуск «пустого» проекта

Загрузка пустого проекта не вызовет в контроллере видимых изменений. Для наглядности рекомендуется добавить в проект компонент **Визуализация** с названием по умолчанию:

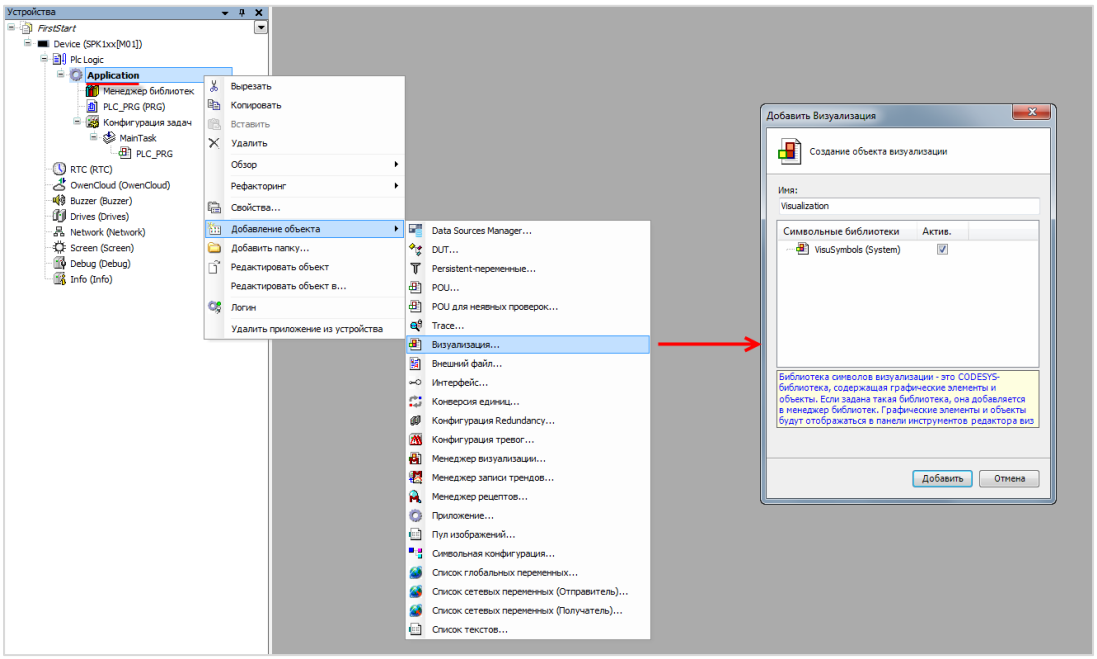


Рисунок 6.1 – Добавление компонента Визуализация

В результате дерево проекта примет следующий вид:

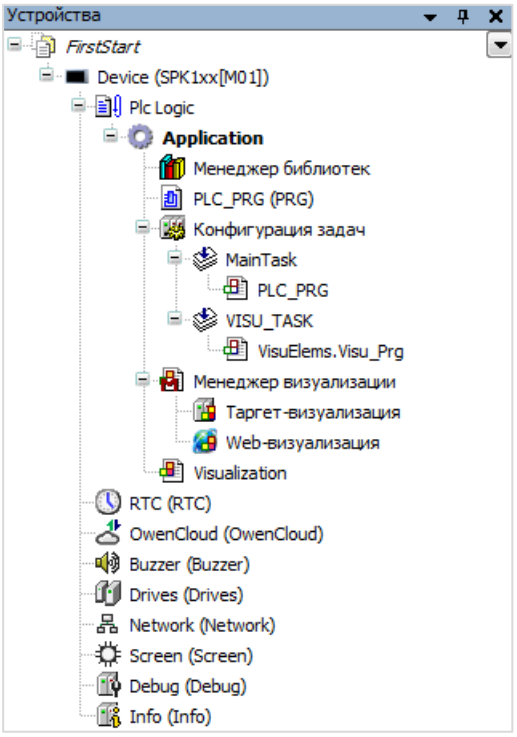


Рисунок 6.2 – Внешний вид дерева проекта после добавления компонента Визуализация

Вместе с компонентом **Визуализация** добавляется компонент **Менеджер визуализации** и новая **задача** с названием **VISU_TASK**. Подробнее все компоненты дерева проекта рассматриваются в п. 7.

Для загрузки проекта в **оперативную память** контроллера следует в меню **Онлайн** нажать кнопку **Логин** (она также продублирована на **Панели инструментов**):

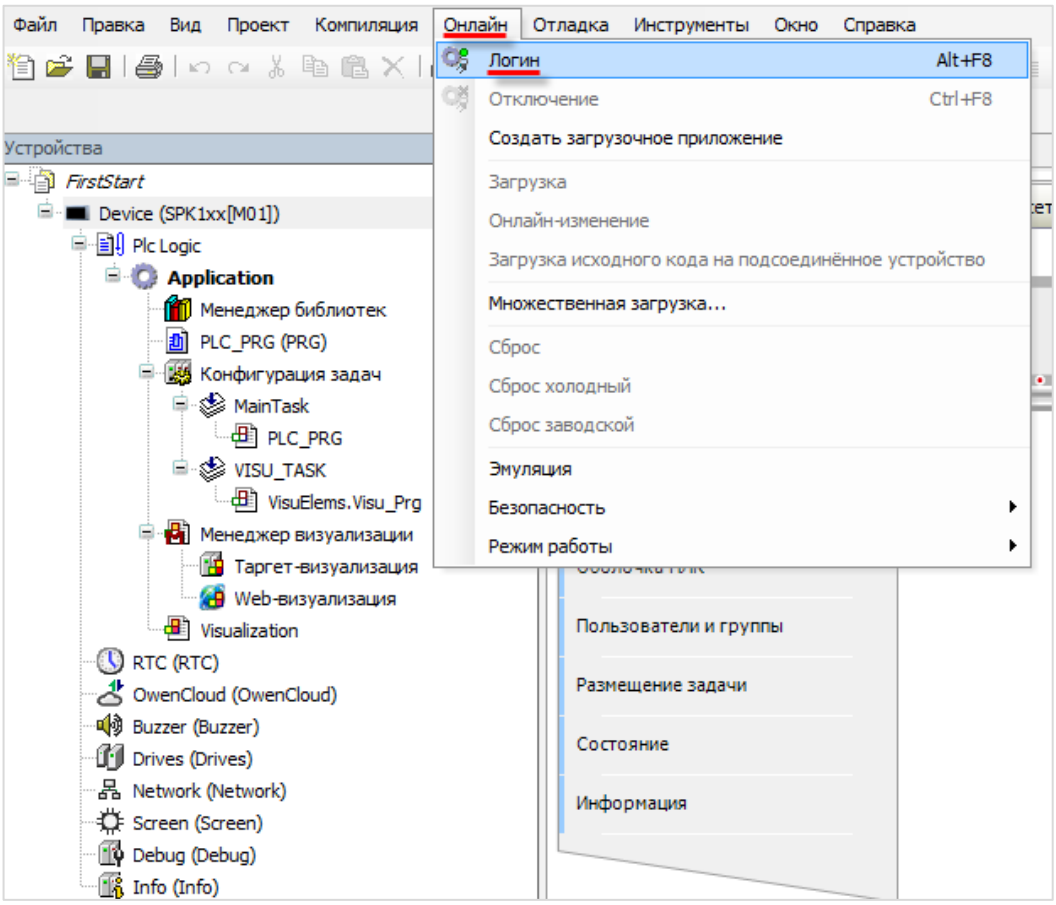


Рисунок 6.3 – Кнопка Логин

В случае если в контроллер уже загружен проект, то появится следующее диалоговое окно:

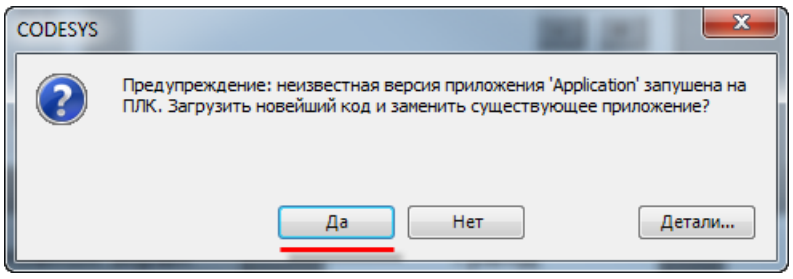


Рисунок 6.4 – Диалоговое окно загрузки проекта

Если нажать **Да**, то новый проект будет загружен в контроллер, а текущий загруженный проект будет удален.

Теперь проект загружен в **оперативную память** контроллера, информация из которой стирается в случае отключения питания. Чтобы проект оставался в памяти контроллера после перезагрузки следует загрузить его в **flash-память,** выполнив команду **Создать загрузочное приложение** из меню **Онлайн**.

ПРИМЕЧАНИЕ

Если проект не был сохранен в flash-память контроллера, то при перезагрузке он будет удален из оперативной памяти контроллера.

После загрузки **строка состояния** будет выглядеть следующим образом:

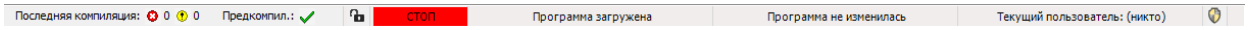


Рисунок 6.5. – Строка статуса загруженного проекта

По умолчанию приложение загруженного проекта ***не запущено*** (***остановлено*****).** Информация об этом продублирована в **строке состояния**. Компоненты **Device** и **Application** подсвечены зеленым, что характеризует установку связи и наличие пользовательского приложения в памяти контроллера.

Для запуска проекта следует нажать кнопку **Старт**, расположенную в меню **Отладка** (она также продублирована на **Панели инструментов**):

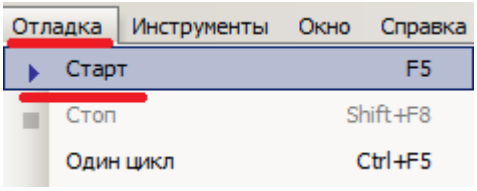


Рисунок 6.6 – Кнопка запуска проекта

Изображение на дисплее контроллера сменится на пустой белый экран, который был добавлен в проект в начале пункта, а дерево проекта и строка состояния будут выглядеть следующим образом:

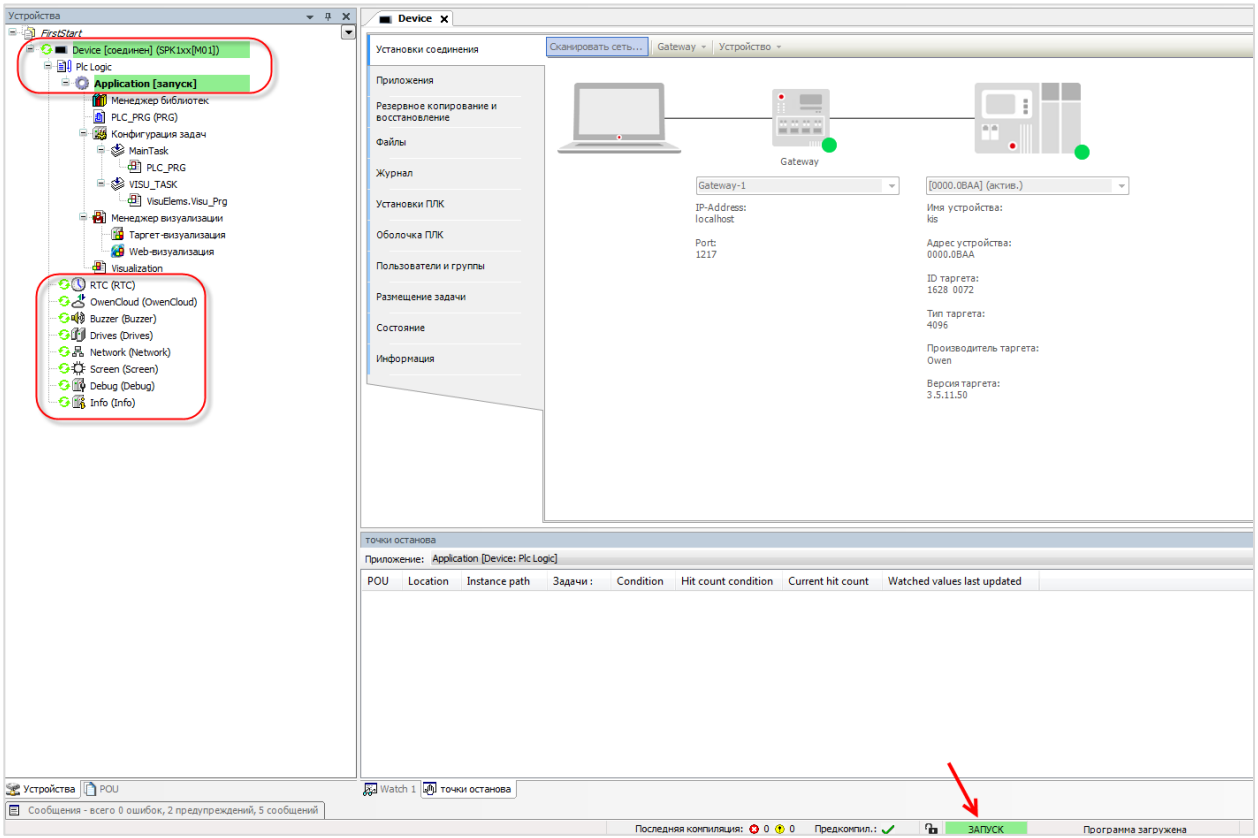


Рисунок 6.7 – Дерево проекта и строка состояния запущенного проекта

ПРИМЕЧАНИЕ

Пользователь может загрузить проект в контроллер с подключенного к нему **USB-** или **SD-накопителя**. Подробно этот процесс рассмотрен в руководстве **CODESYS V3.5. FAQ**.

7 Создание пользовательского проекта

7.1 Постановка задачи

В качестве примера будет рассмотрен процесс разработки пользовательского проекта в среде CODESYS для управления значением **температуры** с помощью **двухпозиционного регулятора**.

Двухпозиционный регулятор (компаратор) сравнивает значение **измеренной величины** с **эталонным (уставкой)**. Состояние выходного дискретного сигнала изменяется на противоположное, если входной сигнал (измеренная величина) пересекает пороговый уровень (уставку). Обычно двухпозиционный регулятор имеет задаваемую **зону гистерезиса** (Δ), которая используется для предотвращения «дребезга» (постоянного включения/выключения) управляющего выходного устройства (например, реле) вблизи значения уставки.

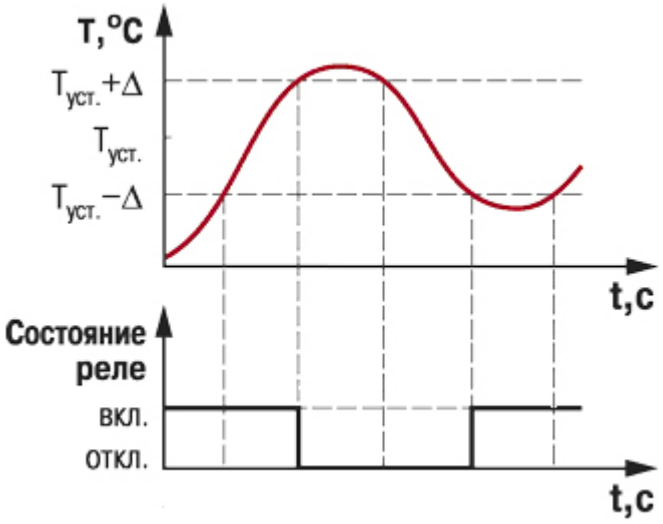


Рисунок 7.1 – Принцип действия двухпозиционного регулятора температуры

Задача:

Температура в помещении измеряется термодатчиком и контролируется кондиционером. Необходимо создать автоматическую систему управления температурой с двухпозиционным регулятором. Система должна обладать графическим интерфейсом со следующим функционалом:

- отображение текущего значения температуры;
- ручной ввод текущего значения температуры (используется в режиме эмуляции);
- настройка параметров двухпозиционного регулятора (ввод значения уставки по температуре и значения гистерезиса);
- возможность включения/выключения кондиционера;
- индикация режима работы кондиционера;
- ввод верхней и нижней аварийной уставки тревог по температуре;
- индикация выхода значения температуры за одну из аварийных уставок;
- построение тренда по температуре, уставкам гистерезиса, уставкам тревог;
- ведение Журнала событий с сигналами о выходе температуры за значения гистерезиса и аварийных уставок.

Система регулирования будет иметь **один входной аналоговый сигнал** (температура) и **два выходных дискретных** – состояние кондиционера (включен/выключен) и режим его работы (обогрев помещения/охлаждение помещения). Для ввода-вывода сигналов будут использованы **модули ОВЕН серии Mx110**: МВ110-224.8А и МУ110-224.8Р. Связь между контроллером и модулями осуществляется по протоколу **Modbus RTU** (см. п. 7.8).

Для наглядности в контроллере будет создана модель кондиционера для **эмуляции процесса регулирования** в случае отсутствия реальных сигналов.

ПРИМЕЧАНИЕ

Данная глава не демонстрирует эффективное решение конкретной задачи, но описывает общие принципы создания проекта в CODESYS с попыткой охватить как можно больший функционал среды программирования, что отражается на искусственности и нецелесообразности отдельных моментов.

7.2 Структура проекта. Общие аспекты создания проекта

Проект в CODESYS имеет иерархическую модульную структуру и состоит из отдельных узлов и их компонентов, расположенных в **дереве проекта**. В этой структуре можно выделить несколько уровней:

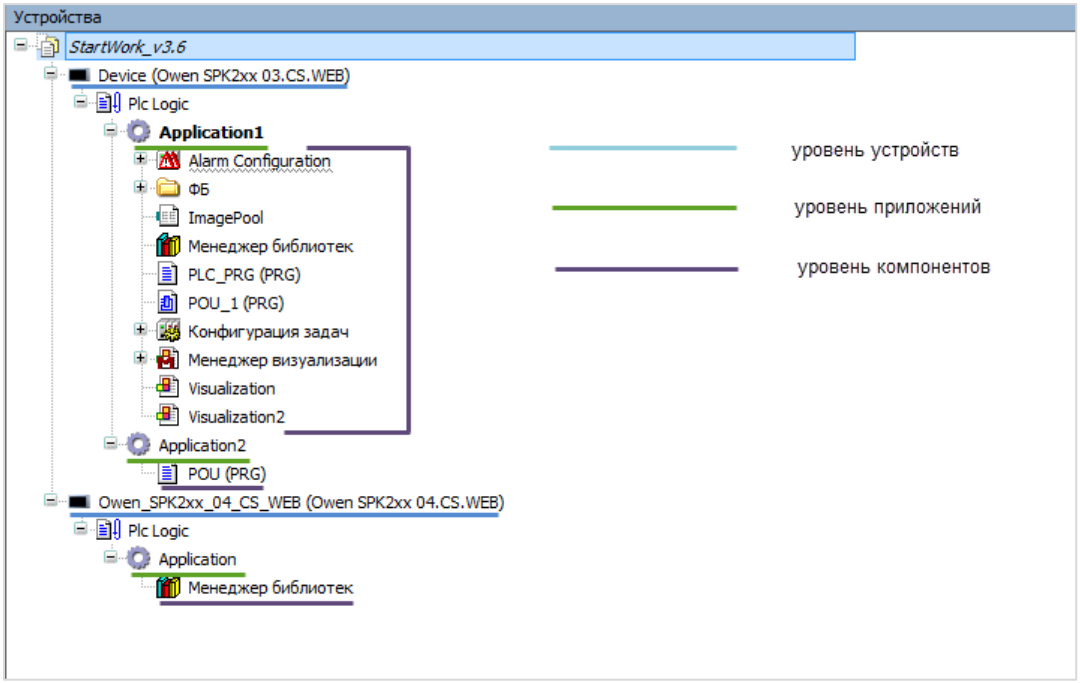


Рисунок 7.2 – Структура дерева проекта

1. **Device** – пользовательское устройство (например, контроллер). По умолчанию название устройства выглядит как **Device (Модель устройства),** где Device – имя устройства, которое может быть изменено пользователем, а название модели устройства, приведенное в скобках, определяется **target-файлом** и не может быть изменено. В проекте может быть несколько устройств. Каждое из устройств является либо **программируемым**, либо **параметрируемым** (на данный момент компании ОВЕН ***не производит*** параметрируемые устройства, поддерживающие CODESYS V3.x). Тип устройства также определяется target-файлом. Программируемые устройства содержат дополнительный узел **Plc_logic**.
2. **Application** – приложение, которое будет запускаться на устройстве. Имя приложения может быть изменено пользователем. Для каждого устройства может быть создано **несколько** приложений, но **активным** из них в каждый момент времени является только одно.

Соответственно, невозможно запустить несколько приложений **одновременно** и загрузить в контроллер более одного приложения.

3. **Компоненты** – отдельные модули, из которых состоит приложение. Часть из них автоматически создается одновременно с проектом, остальные по необходимости добавляются пользователем. Компоненты могут содержать **дочерние компоненты**. Имена некоторых (но не всех) компонентов могут быть изменены пользователем.

По умолчанию созданный проект содержит одно приложение, которое включает три компонента:

1. Пустую программу с название **PLC_PRG** (**язык** программы определяется на этапе создания проекта, см. п. 3).
2. Компонент **Конфигурация задач**, содержащий дочерний **компонент** задачу с названием **MainTask**, к которой привязана программа **PLC_PRG**. Задача определяет частоту и правила вызова привязанного к ней **POU**.
3. Компонент **Менеджер библиотек**, отвечающий за подключение к проекту внешних библиотек, которые добавляют дополнительный функционал среды программирования.

Процесс создания проекта состоит из следующих этапов:

1. Создание экранов визуализации.
2. Разработка пользовательских программ и алгоритмов.
3. Настройка требуемых компонентов (например, **Конфигурации задач**).
4. Настройка обмена данными между контроллером и другими устройствами (например, модулями ввода–вывода или датчиками), связь переменных из пользовательских программ с соответствующими реальными сигналами.

Последовательность этапов разработки определяется пользователем, в рамках примера следует придерживаться приведенной выше схемы.

7.3 Создание экранов визуализации

7.3.1 Предварительная настройка

Для решения сформулированной в п. 7.1 задачи требуется три экрана визуализации:

1. Основной экран (отображение информации и управление).
2. Экран тренда.
3. Экран журнала тревог.

На данном этапе в проекте имеется один экран визуализации с названием **Visualization**, созданный в п. 5. Его следует переименовать в **MainScreen** (использование кириллицы в названиях компонентов не поддерживается) с помощью двойного нажатия **ЛКМ** на название экрана. Затем следует создать еще два экрана с названиями **Trend** и **Alarm_log**:

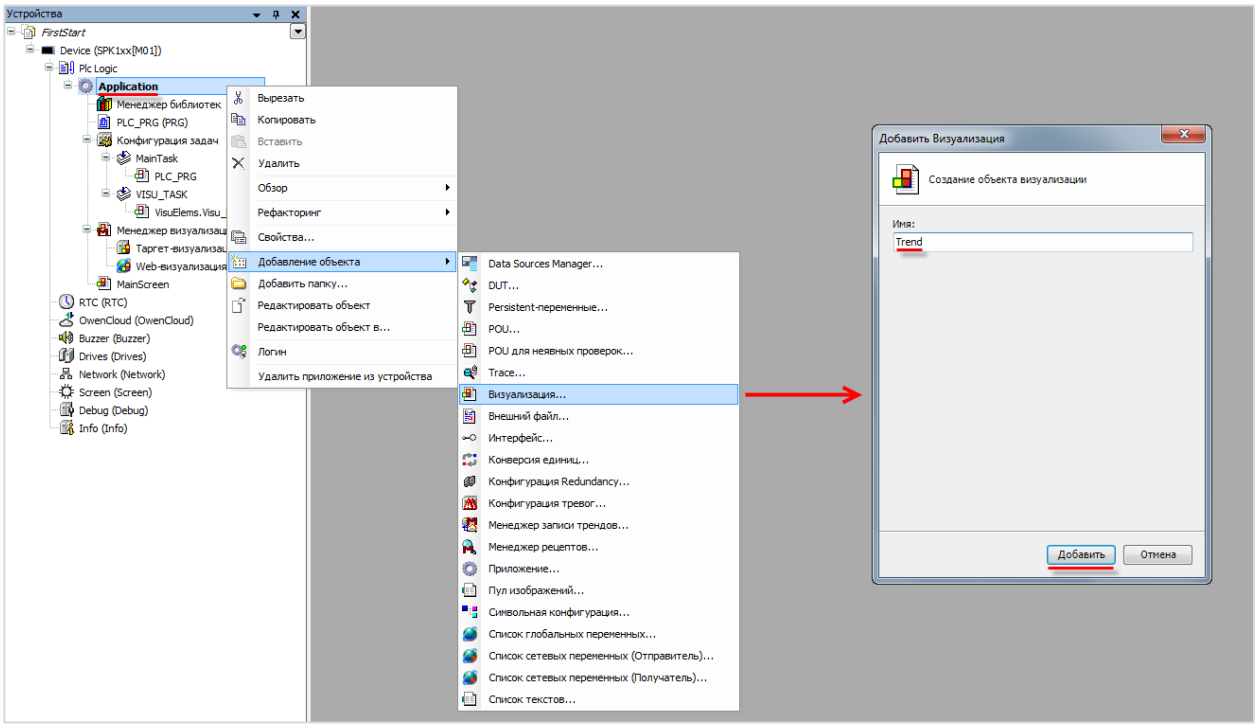


Рисунок 7.3 – Добавление экрана визуализации

Перед наполнением экранов графическими примитивами, следует настроить компонент **Менеджер визуализации** и его дочерние компоненты **Target-визуализация** и **Web-визуализация**.

Настройки вкладки **Установки** Менеджера визуализации:

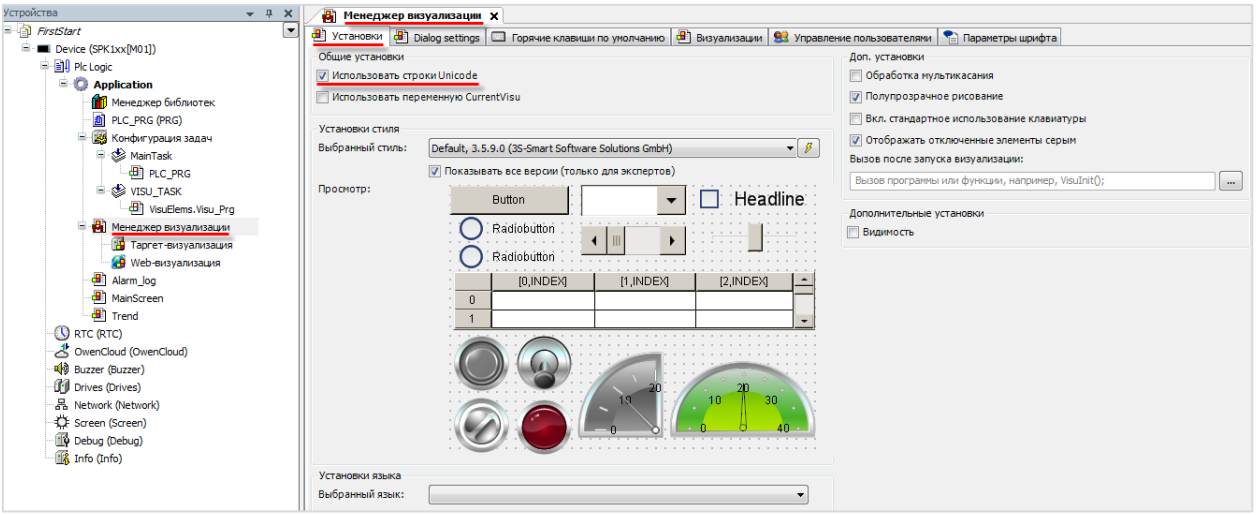


Рисунок 7.4 – Установки Менеджера визуализации

Для отображения в текстах визуализации **кириллицы** следует поставить галочку **Использовать строки Unicode**. Вторая опция – **Использовать переменную CurrentVisu** – добавляет в проект **одноименную строковую переменную**, которая определяет какой из экранов отображается в данный момент. Соответственно, записывая в нее названия экранов визуализации, можно переключаться между ними. В данном примере работа этой переменной ***рассмотрена не будет***.

Остальные настройки касаются стиля визуализации (который определяет внешний вид графических примитивов), выбора стартового языка визуализации (при реализации мультиязычной визуализации), используемых экранных клавиатур, объема памяти визуализации и т. д. Значения настроек следует оставить в значениях ***по умолчанию***.

Затем следует настроить компонент **Target-визуализация**, который является дочерним компонентом **Менеджера визуализации**:

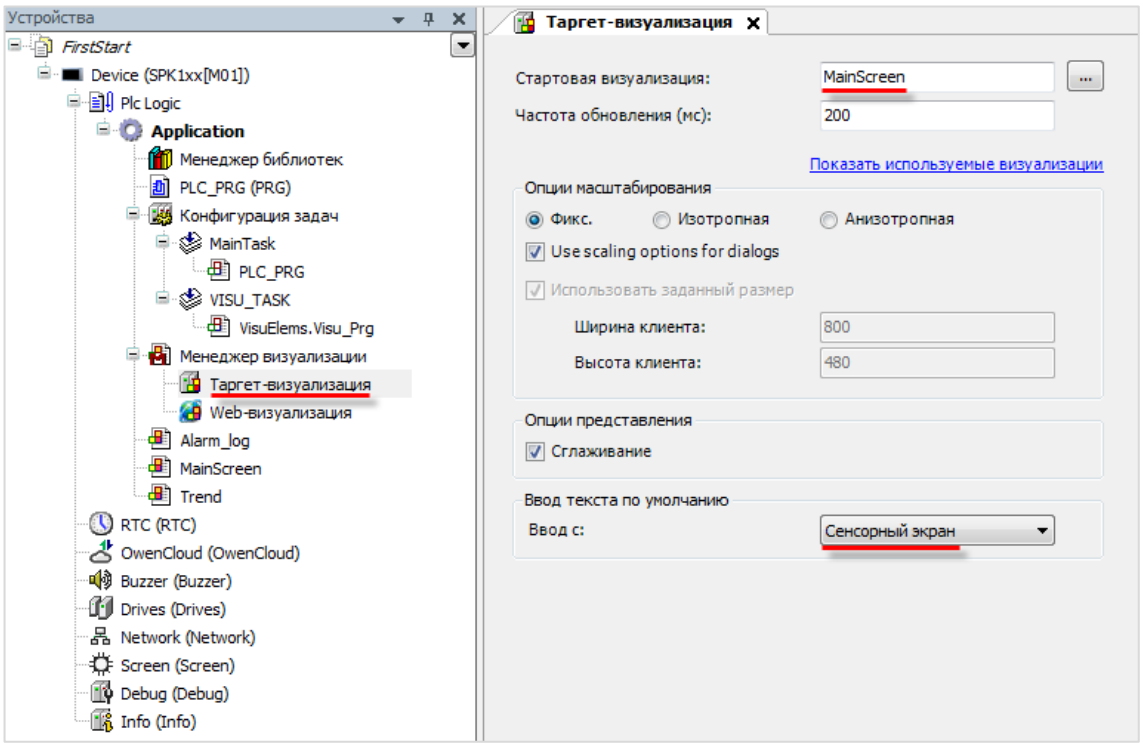


Рисунок 7.5 – Настройки таргет-визуализации

ПРИМЕЧАНИЕ
Данный компонент присутствует только у панельных контроллеров (например, СПК). У контроллеров без дисплея (ПЛК) данный компонент отсутствует.

Имя стартовой визуализации должно соответствовать названию экрана, который будет отображаться после запуска проекта. Для данной задачи это **MainScreen**.

Опция **Подгонка размера** определяет размер визуализации:

1. **Fixed** – отображается часть визуализации заданного размера, зависящего от разрешения дисплея контроллера.
2. **Isotropic** – экран визуализации будет растянут до разрешения дисплея **с сохранением соотношения сторон**.
3. **Anisotropic** – экран визуализации будет растянут до разрешения дисплея **без сохранения соотношения сторон**.

Опция **Ввод текста по умолчанию** определяет устройство ввода. Следует поставить значение **Сенсорный экран**.

Опции **web-визуализации** практически идентичны target-визуализации. Так как web-визуализация запускается на внешних устройствах, ее разрешение должен соответствовать разрешению этих устройств. Если в качестве устройства просмотра web-визуализации используется **Full HD** монитор, то рекомендуется указывать разрешение **1920 × 1080**. Для того чтобы указать разрешение следует выбрать режим **Fixed**. После указания разрешения следует выбрать режим **Isotropic**.

Имя **.htm-файла** определяет название web-страницы визуализации.

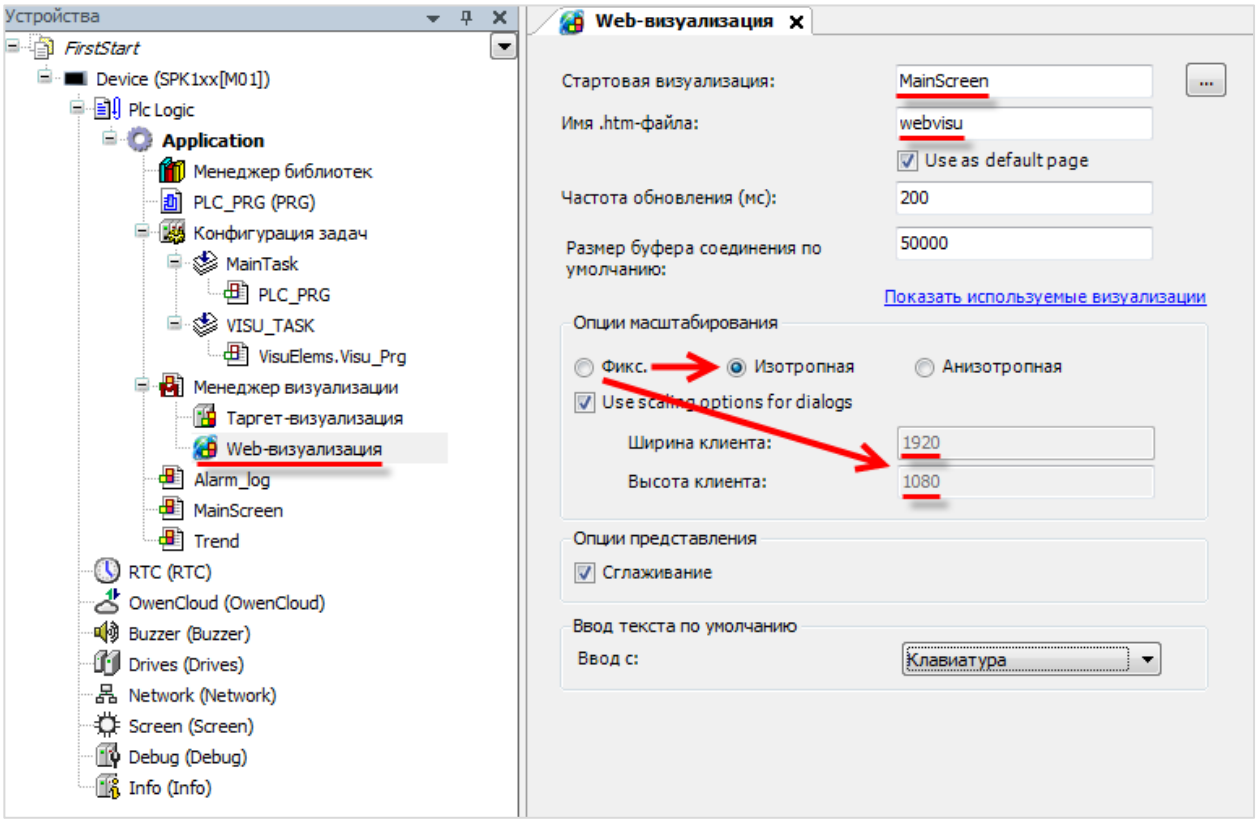


Рисунок 7.6 – Настройки web-визуализации

Помимо настройки разрешения таргет- и web-визуализаций следует установить **разрешение каждого экрана** визуализации. Для установки следует нажать **ПКМ** на название соответствующего экрана и выбрать пункт **Свойства**, вкладка **Визуализация**:

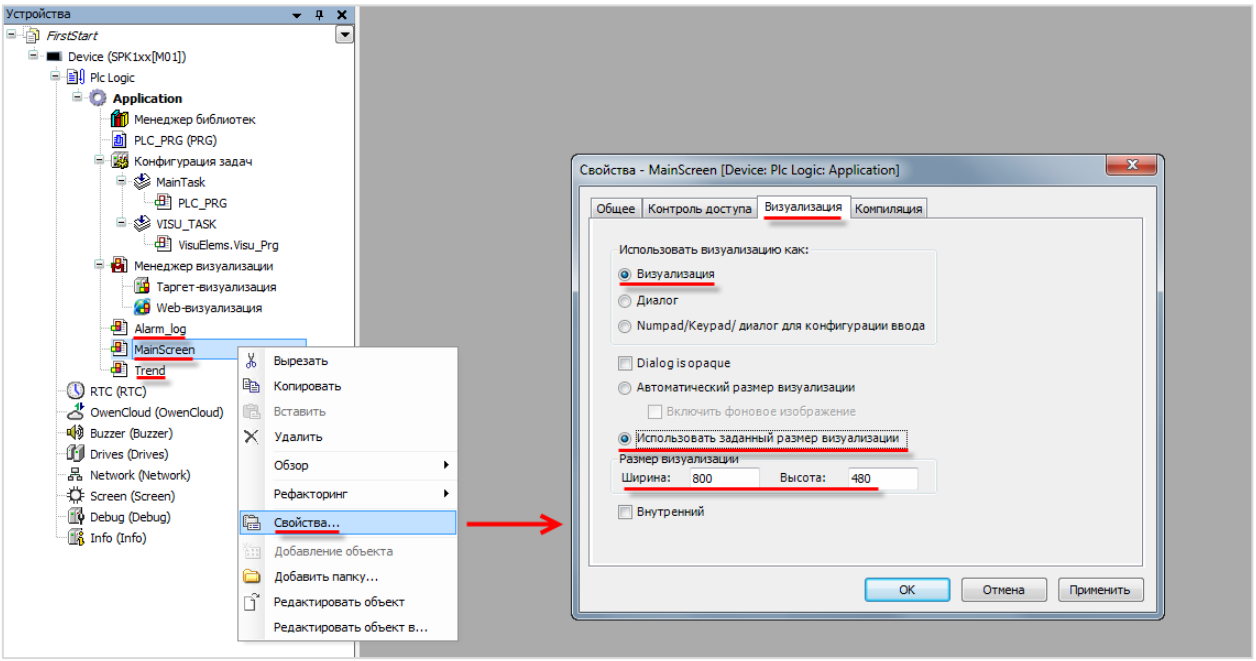


Рисунок 7.7 – Свойства экрана визуализации MainScreen

Перед началом разработки экранов визуализации рекомендуется в **Опциях** (**Инструменты** – **Опции**) включить **сетку**:

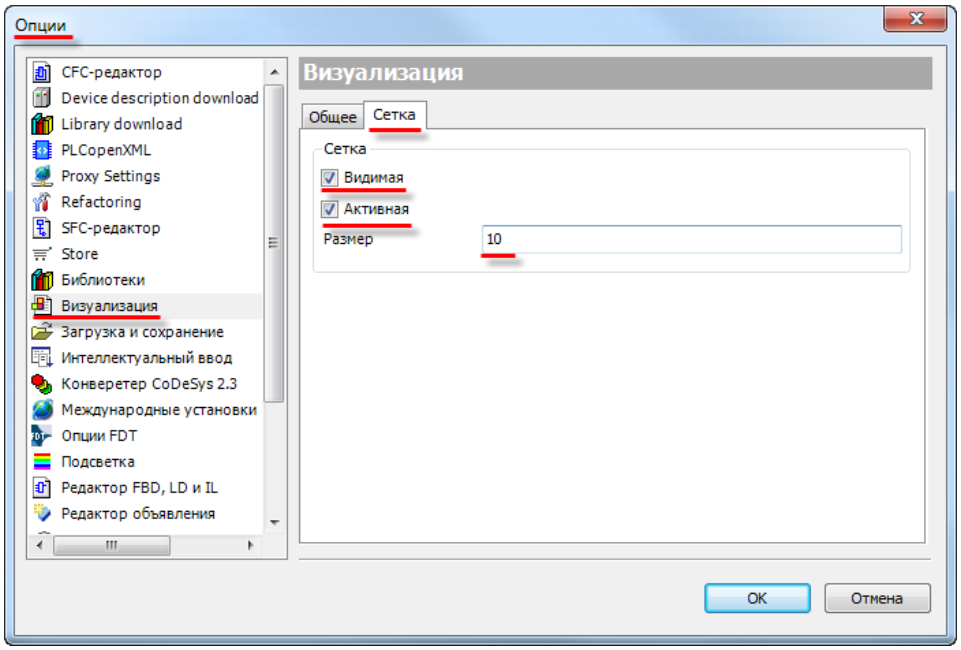


Рисунок 7.8 – Настройки сетки

7.3.2 Наполнение экрана MainScreen

Экран **MainScreen** будет использоваться для отображения текущего значения температуры, переключения режима измерения, установки значения температуры в режиме «Эмуляция (Ручной ввод)», задания уставки температуры и гистерезиса, управления кондиционером и индикации режима его работы.

Двойным нажатием **ЛКМ** на компонент **MainScreen** открывается **Редактор визуализации** соответствующего экрана:

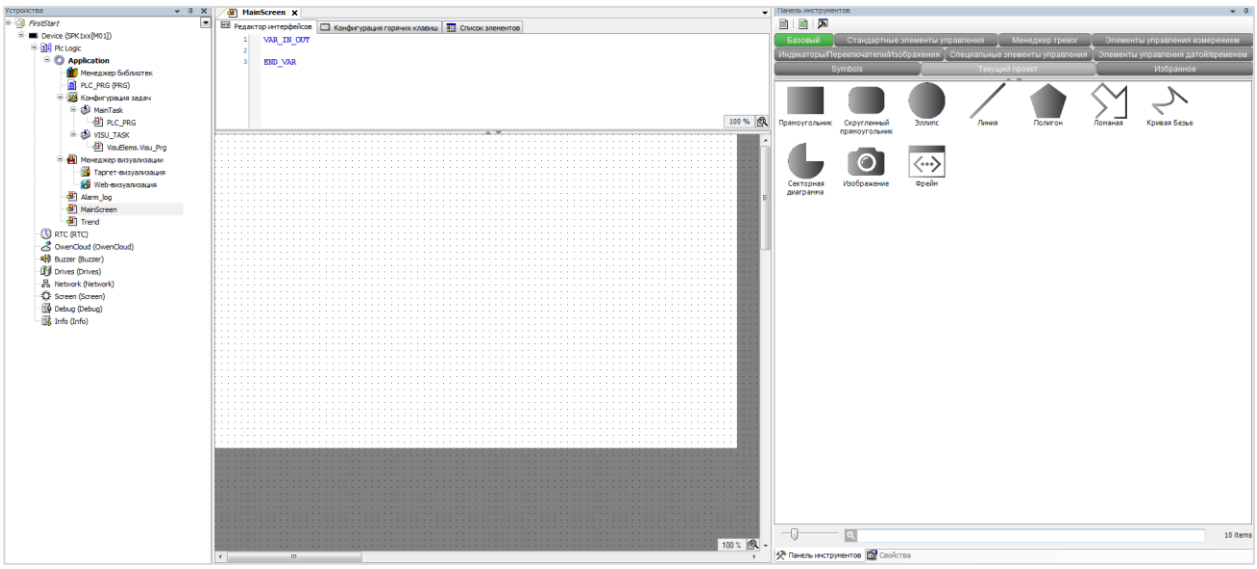


Рисунок 7.9 – Редактор визуализации

Справа от **рабочего поля** расположена **Панель инструментов редактора**, содержащая набор **графических примитивов**. Выделив **элемент**, можно с помощью одиночного нажатия **ЛКМ** разместить его на рабочем поле.

Функциональная настройка элементов и привязывание к ним переменных описываются в п. 7.5.

I. Элементы **Кнопка** и **Прямоугольник**

Для переключения между экранами используется элемент **Кнопка**. Во вкладке **Стандартные элементы управления** следует выбрать элемент **Кнопка** и разместить его на рабочем поле:

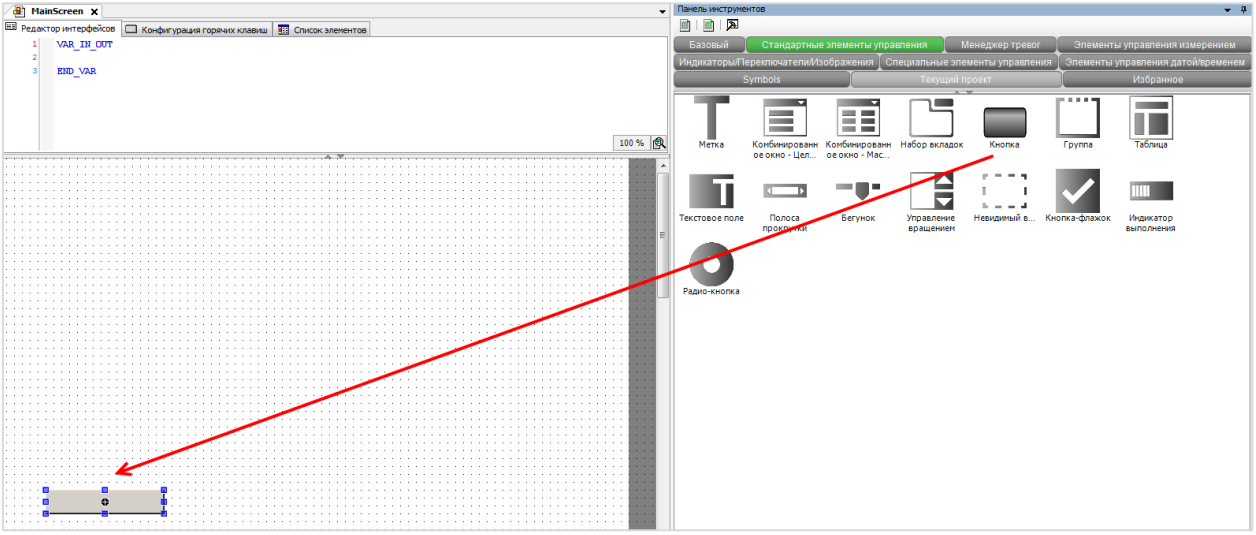


Рисунок 7.10 – Создание кнопки

Панель свойств кнопки открывается нажатием **ЛКМ** по кнопке на рабочем поле.

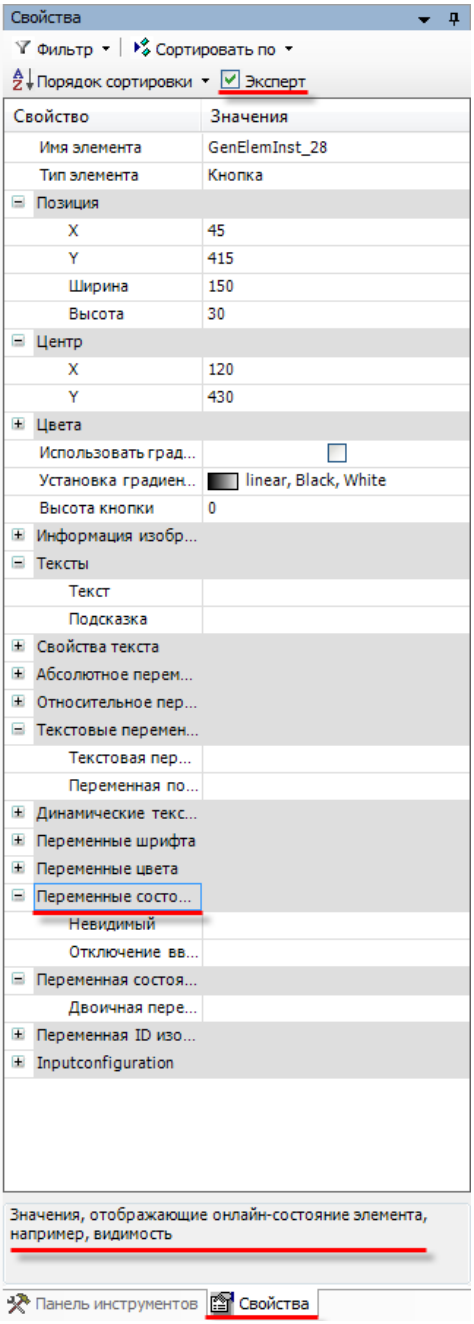


Рисунок 7.11 – Панель свойств кнопки

Панель свойств состоит из раскрывающихся вкладок с параметрами, часть которых являются идентичными для всех элементов, часть – уникальными. Для отображения всех вкладок следует использовать режим **Эксперт**, который включается вверху панели. Во время выделения любой из вкладок внизу отображается информационная подсказка. Для изменения параметра достаточно два раза нажать **ЛКМ** на его значение.

Далее рассматриваются только те настройки элементов, которые требуются для создания проекта. Подробное описание всех настроек приведено в руководстве **CODESYS V3.5. Визуализация**.

Настройки кнопки (изменения в стандартных настройках выделены красным):

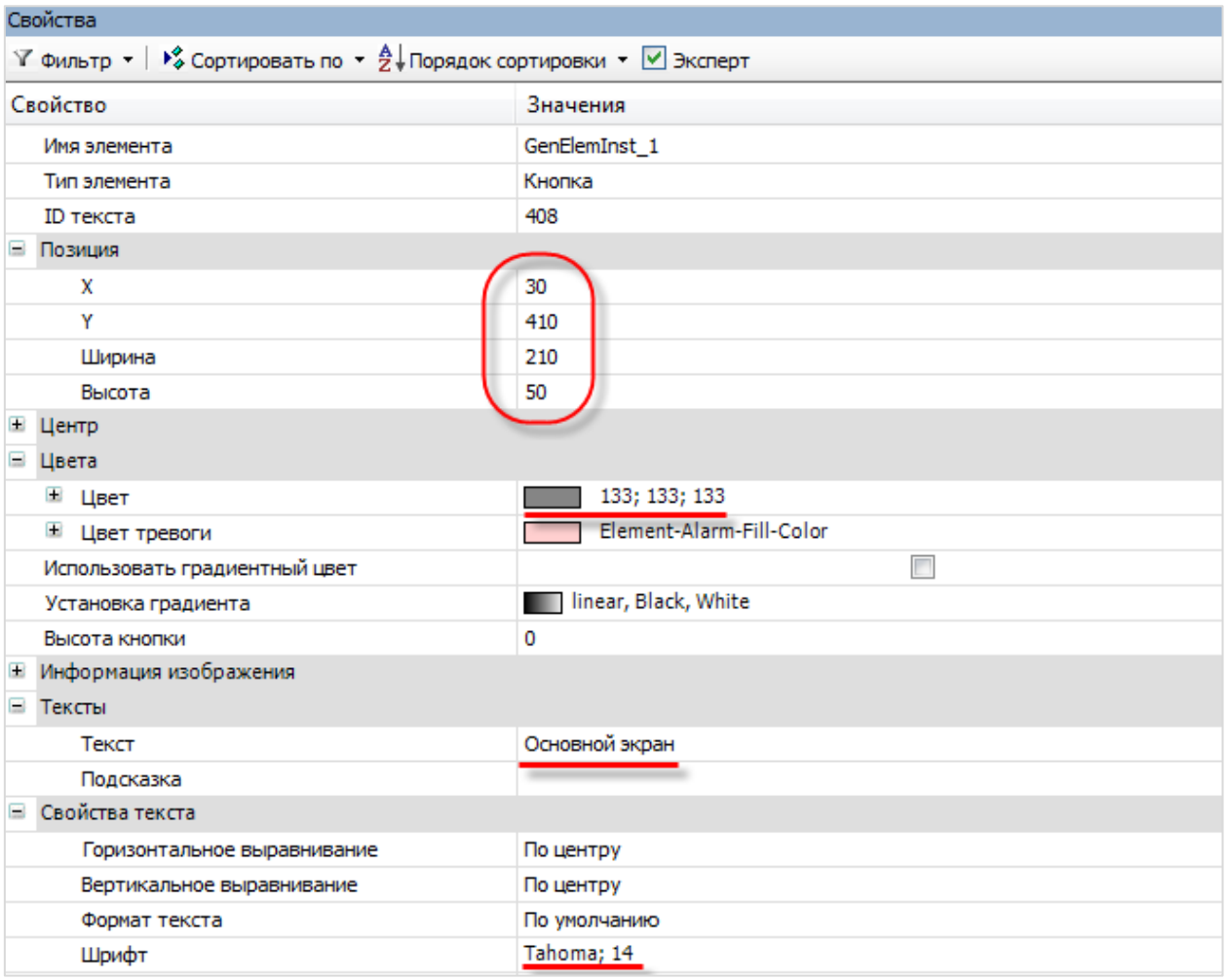


Рисунок 7.12 – Настройка кнопки перехода

Чтобы изменить размер элемента, его положение на экране и текст, необязательно менять настройки через **Панель свойств** – все эти операции можно проделать непосредственно нажимая **ЛКМ** на элемент:

1. Для **перетаскивания** следует выделить элемент, навести на него мышью для появления курсора перетаскивания ( ) и, зажав **ЛКМ**, перетащить элемент в нужное место экрана.

2. Для **изменения размера** следует выделить элемент, навести мышью на одну из его опорных точек ( ) для появления курсора деформации ( ) и, зажав **ЛКМ**, растянуть элемент до нужных размеров.

3. Для **изменения текста элемента** следует навести на текстовое поле элемента мышью для появления курсора ввода ( ), после чего однократным нажатием **ЛКМ** перейти к редактированию текста.

Созданный элемент можно копировать/вставлять с помощью **контекстного меню** (открывается по нажатию **ПКМ** на элемент), либо с помощью стандартных комбинаций клавиш **Ctrl+C/Ctrl+V**.

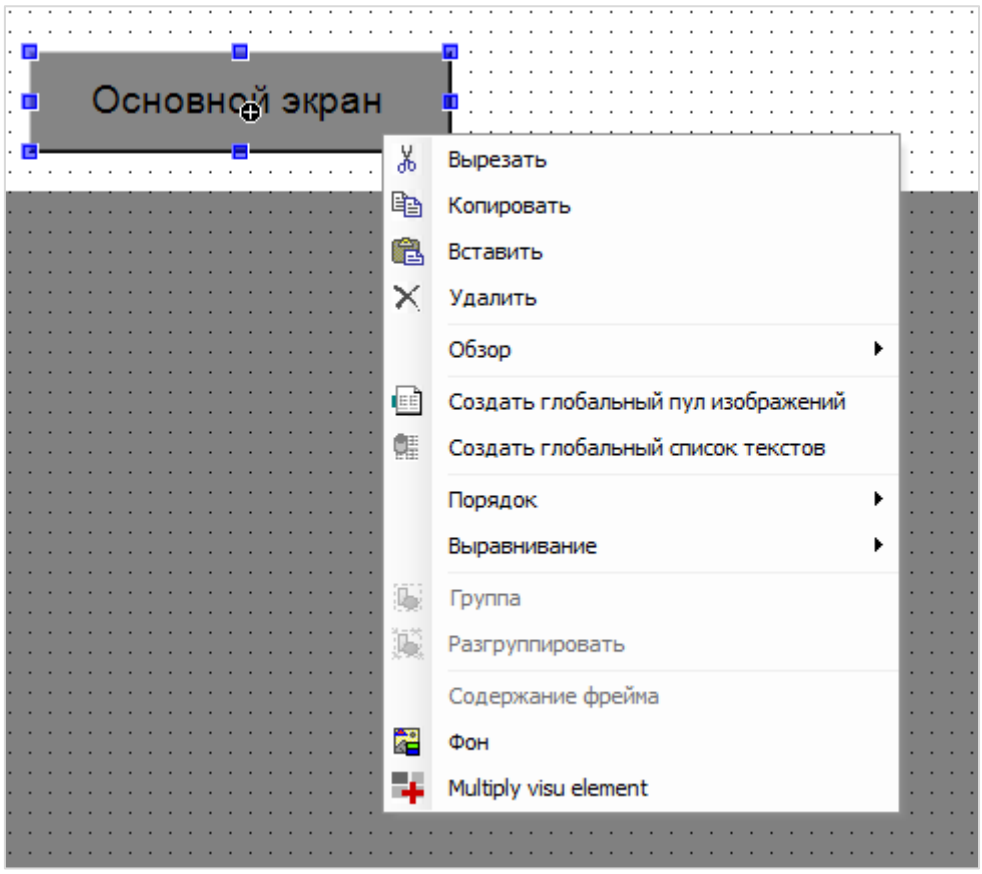


Рисунок 7.13 – Контекстное меню элемента

На основе созданной кнопки следует создать еще две (красным выделены настройки, значения которых отличаются от настроек кнопки **Основной экран**):

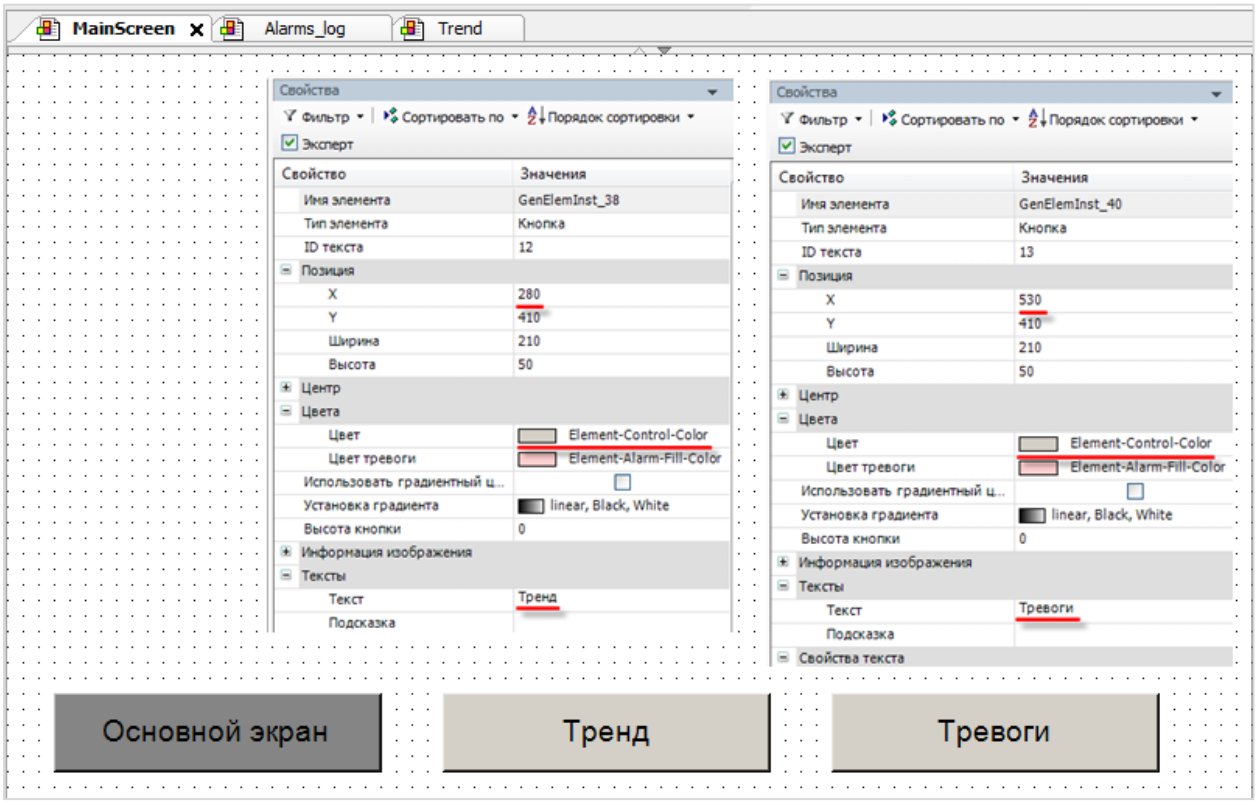


Рисунок 7.14 – Кнопки перехода

Темно-серый цвет кнопки **Основной экран** характеризует отображение этого экрана в данный момент времени. Соответственно, на **экране Trend** темно-серой будет **кнопка Тренд**, а цвета кнопок **Тревоги** и **Основной экран** будут совпадать.

Зажав **ЛКМ**, следует выделить сразу три кнопки и с помощью команд контекстного меню **Копировать**/**Вставить**, перенести их на экраны **Trend** и **Alarm_log**:

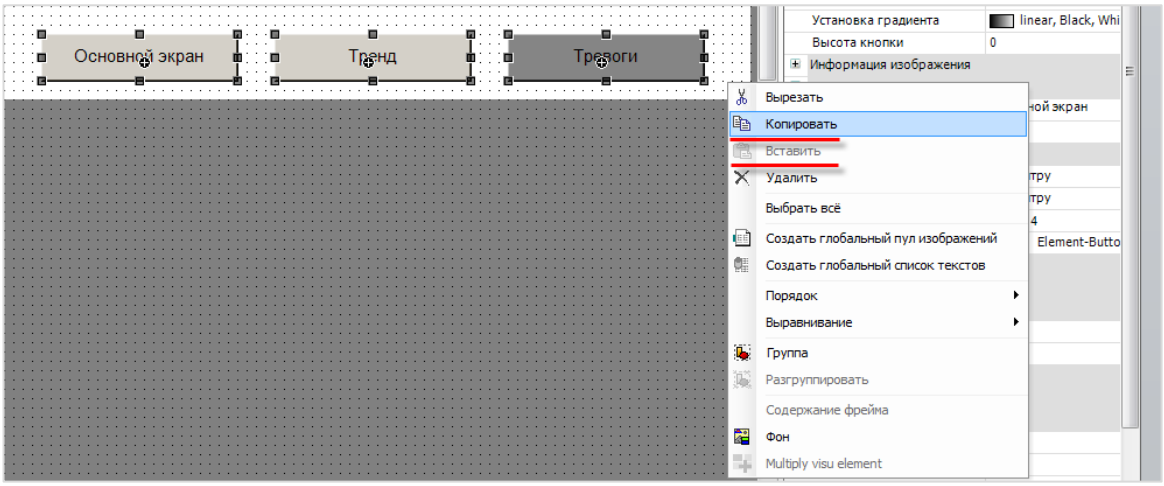


Рисунок 7.15 – Копирование группы элементов

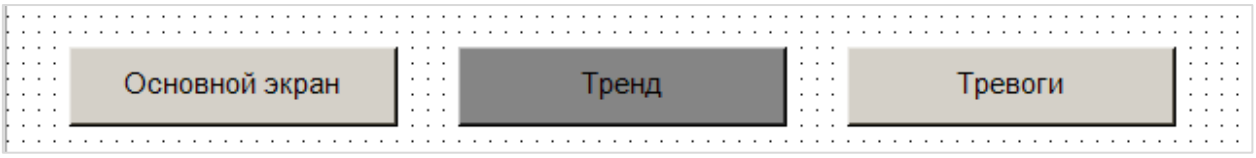


Рисунок 7.16 – Внешний вид кнопок на экране Trend

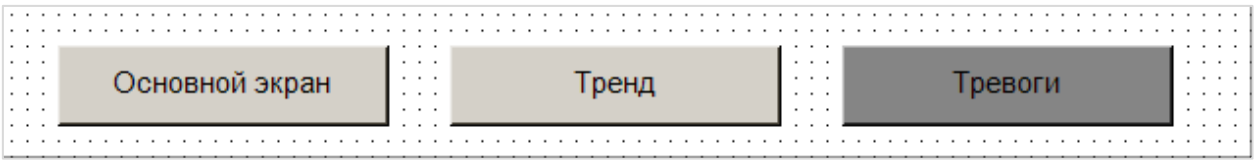


Рисунок 7.17 – Внешний вид кнопок на экране Alarm_log

Функциональная настройка кнопок переключения между экранами описывается в п. 7.5.

Затем следует создать дополнительные панели с помощью элементов **Прямоугольник** из вкладки **базовых** элементов:

- панель названия экрана;
- панель названия устройства управления (кондиционера);
- панель управления температурой;
- панель кондиционера.

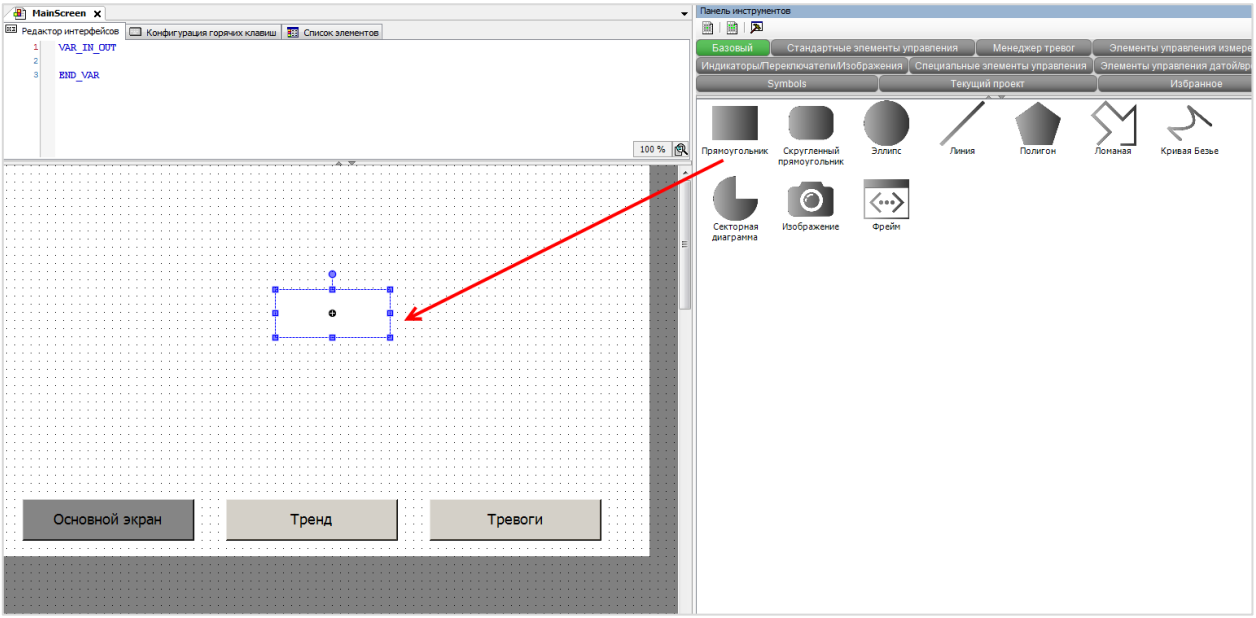


Рисунок 7.18 – Добавление элемента Прямоугольник

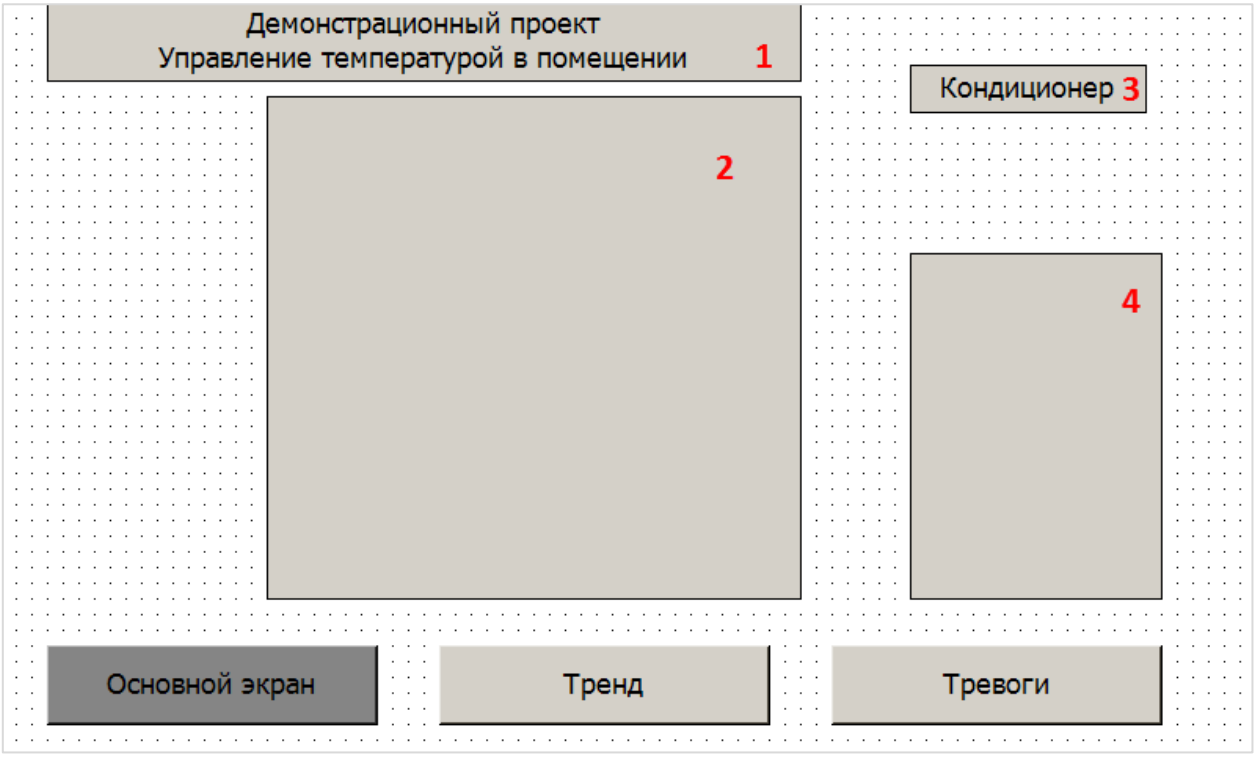


Рисунок 7.19 – Экран MainScreen после окончания размещения элементов

Настройки прямоугольников:

Номер панели на рисунке 7.19	Текст	Настройки			
		X	Y	Ширина	Высота
1	Демонстрационный проект Управление температурой в помещении	30	0	480	50
2	-	170	60	340	320
3	Кондиционер	580	40	150	30
4	-	580	160	160	220

Цвет элемента прямоугольник задается в параметре **Цвет/Цвет заливки**.

Для перехода на следующую строку во время набора текста (панель 1) используется комбинация клавиш **Ctrl+Enter**.

II. Элемент **Текстовое поле**

Для размещения на панелях нескольких надписей с разными настройками (размер, шрифт и т. д.) рекомендуется использовать элемент **Тестовое поле**, расположенный во вкладке **Стандартные элементы управления**.

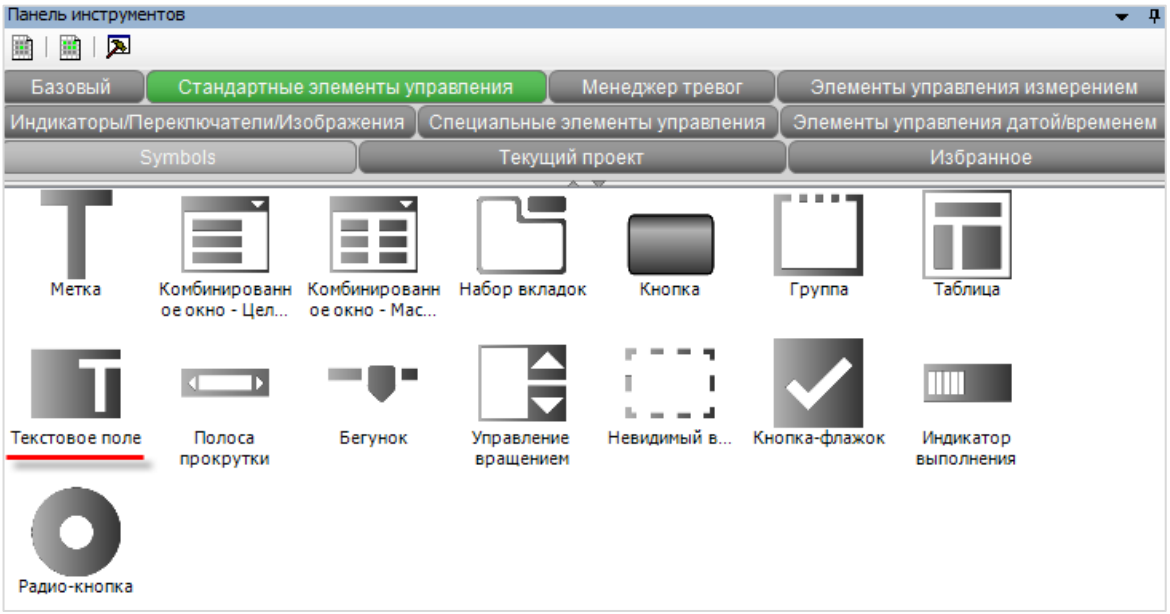


Рисунок 7.20 – Элемент Текстовое поле

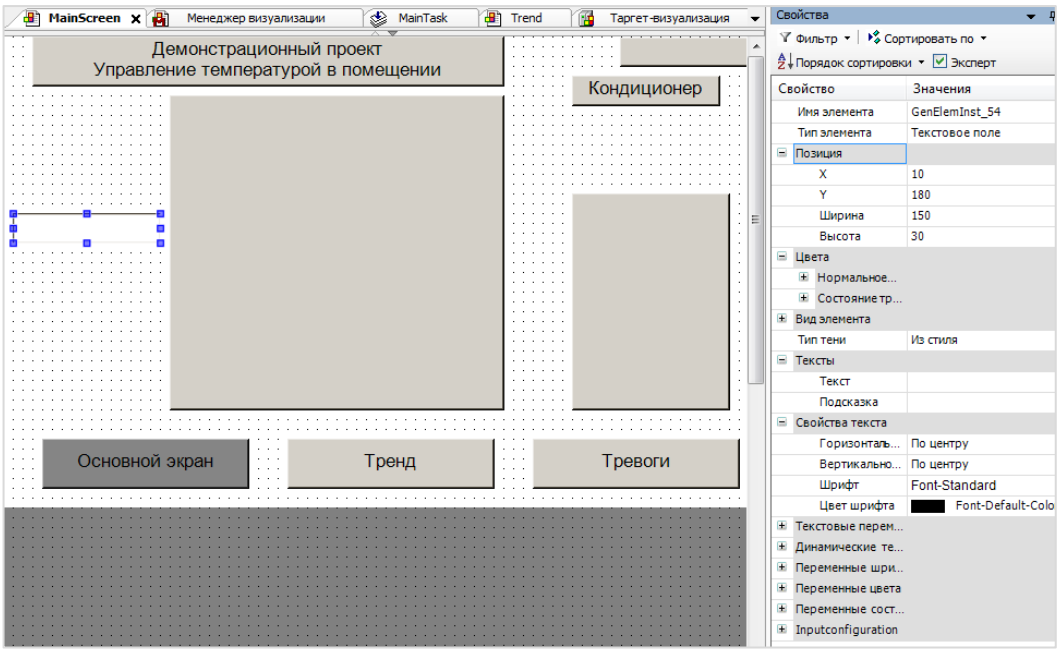


Рисунок 7.21 – Размещение элемента Текстовое поле в рабочей области и его настройки

Настройка элемента аналогична настройке элемента **Кнопка**.

Пример размещения элемента **Текстовое поле** для написания заголовка на панели управления температурой в проекте:

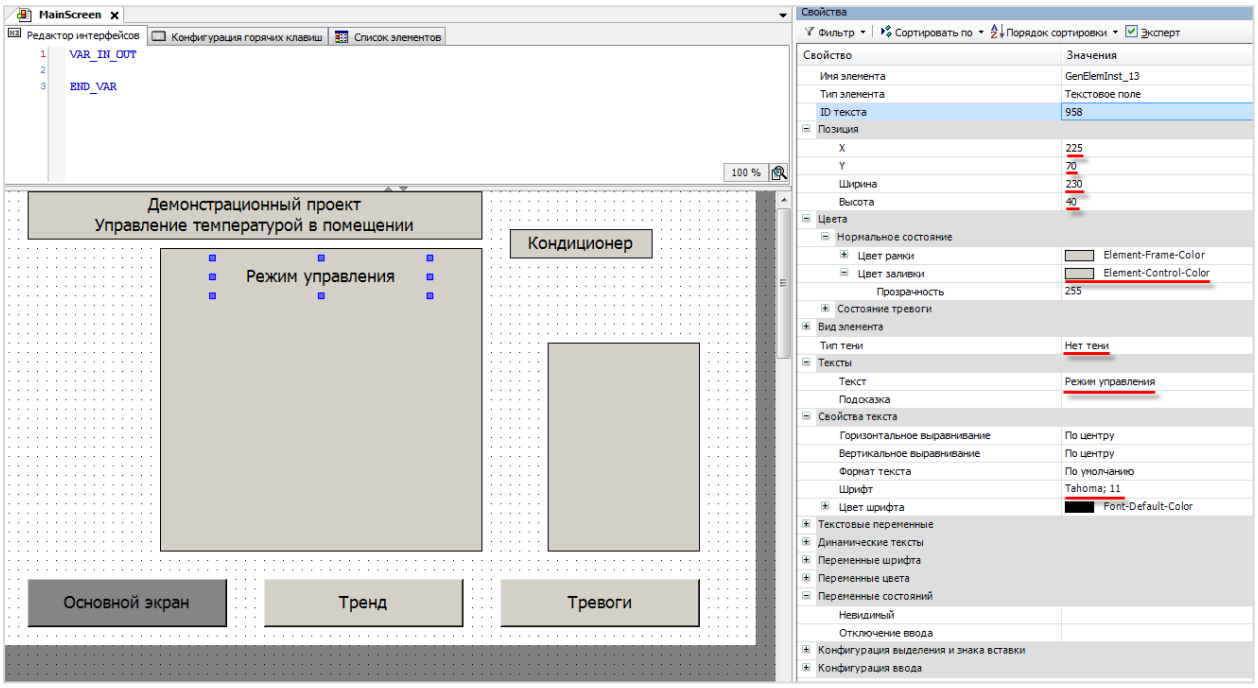


Рисунок 7.22 – Настройки заголовка панели управления температурой

Затем следует разместить четыре дополнительных текстовых поля.

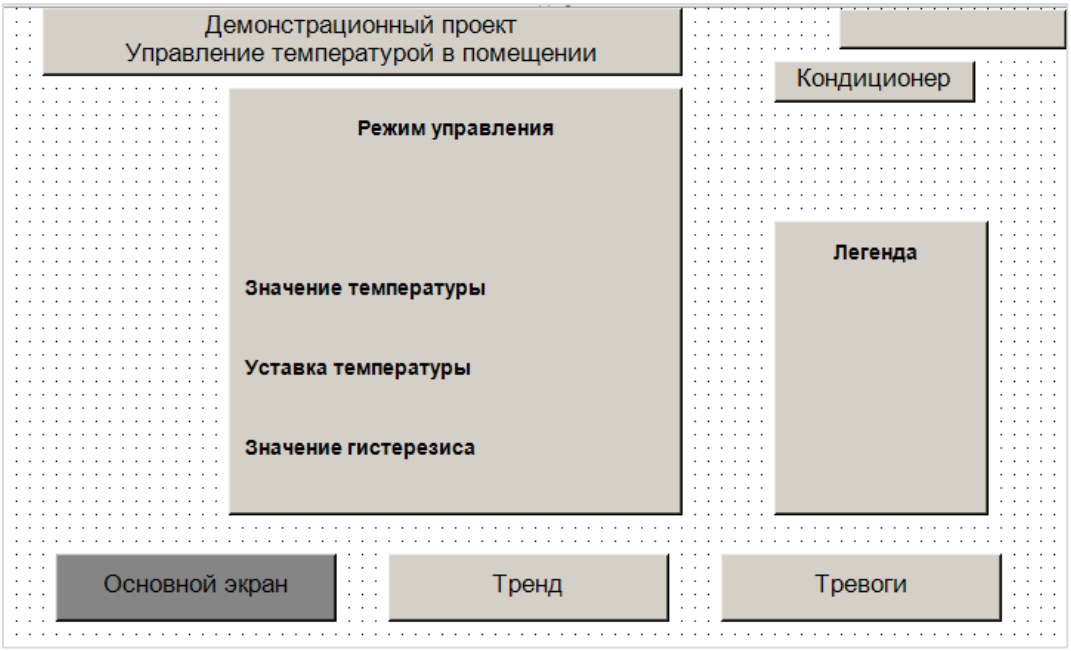


Рисунок 7.23 – Экран MainScreen после окончания размещения элементов Текстовое поле

Настройки текстовых полей:

Текст	Горизонтальное выравнивание (вкладка Свойства текста)	Настройки			
		X	Y	Ширина	Высота
Значение температуры	Лево	180	170	223	40
Уставка температуры	Лево	180	240	223	40
Значение гистерезиса	Лево	180	310	223	40
Легенда	По центру	610	169	90	30

Для операций с несколькими элементами следует, зажав **ЛКМ**, выделить область рабочего поля (или последовательно выделять элементы с зажатой клавишей **Shift**), после чего нажать **ПКМ** на любом месте рабочего поля для вызова контекстного меню:

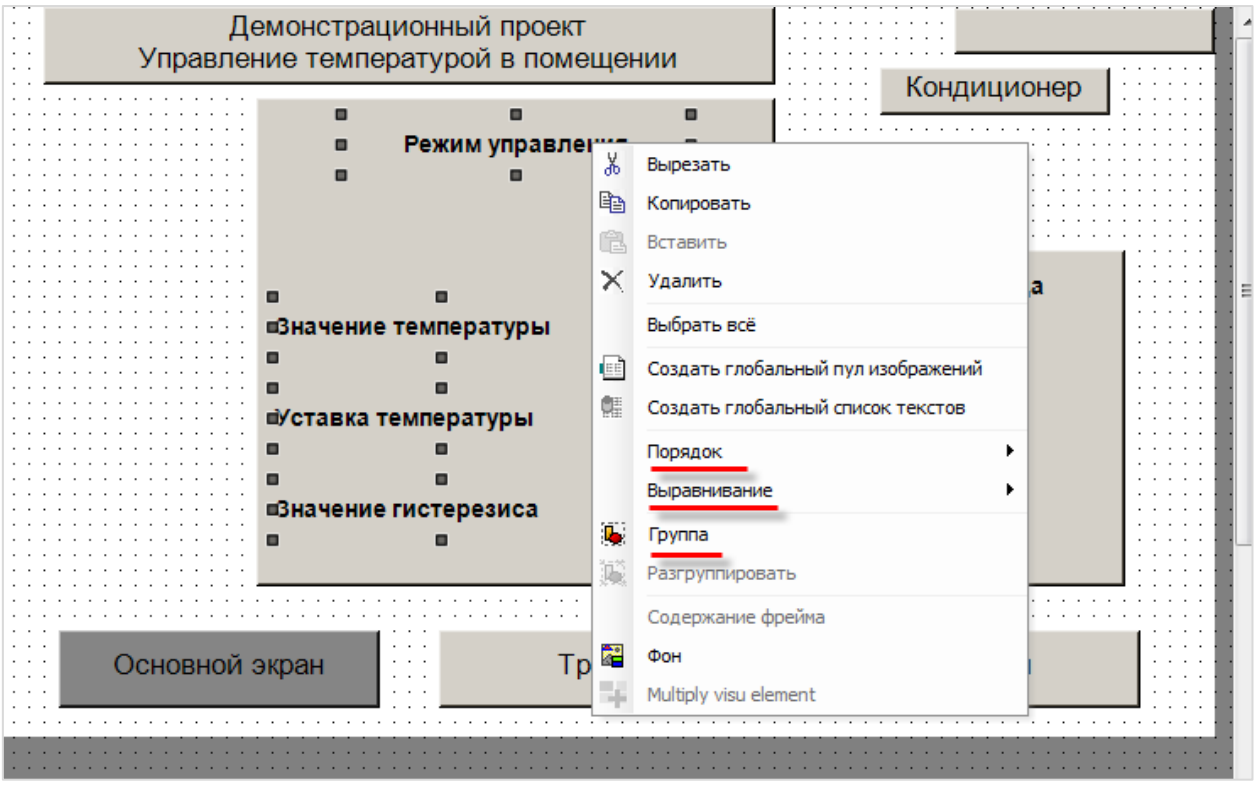


Рисунок 7.24 – Контекстное меню группы элементов

Вкладка **Выравнивание** позволяет выравнивать элементы относительно друг друга. Выравнивание происходит ***относительно элемента, выделенного последним***.

Вкладка **Группа** позволяет сгруппировать несколько элементов и взаимодействовать с ними как с одним элементом.

Вкладка **Порядок** (доступная и во время выделения одного элемента) позволяет размещать элементы в разных **слоях** относительно друг друга – это дает возможность накладывать элементы друг на друга без перекрывания.

III. Элемент **Кнопка-флажок (чекбокс)**

Кнопки-флажки (чекбоксы) позволяют управлять параметром с двумя состояниями – одно из них соответствует пустому чекбоксу, второе – заполненному (традиционно используются пиктограмма «галочка»).

В рамках примера будет использоваться чекбокс для переключения режимов управления температурой – **автоматического** (значение температуры в помещении измеряется датчиком, управляющий сигнал передается на реальное устройство) и **эмуляции** (показания датчика задаются пользователем, вместо выдачи сигнала управления осуществляется **эмуляция** процесса регулирования).

Под надписью **Режим управления** на экране **MainScreen** следует разместить элемент **Кнопка-флажок (Чекбокс)**:

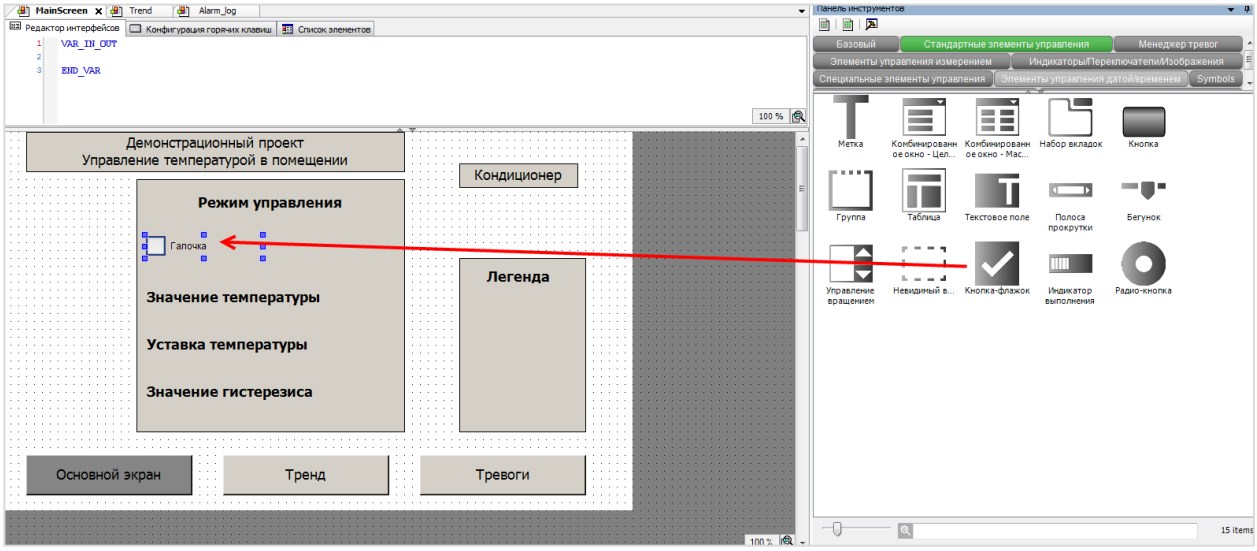


Рисунок 7.25 – Добавление элемента Чекбокс

Настройки элемента:

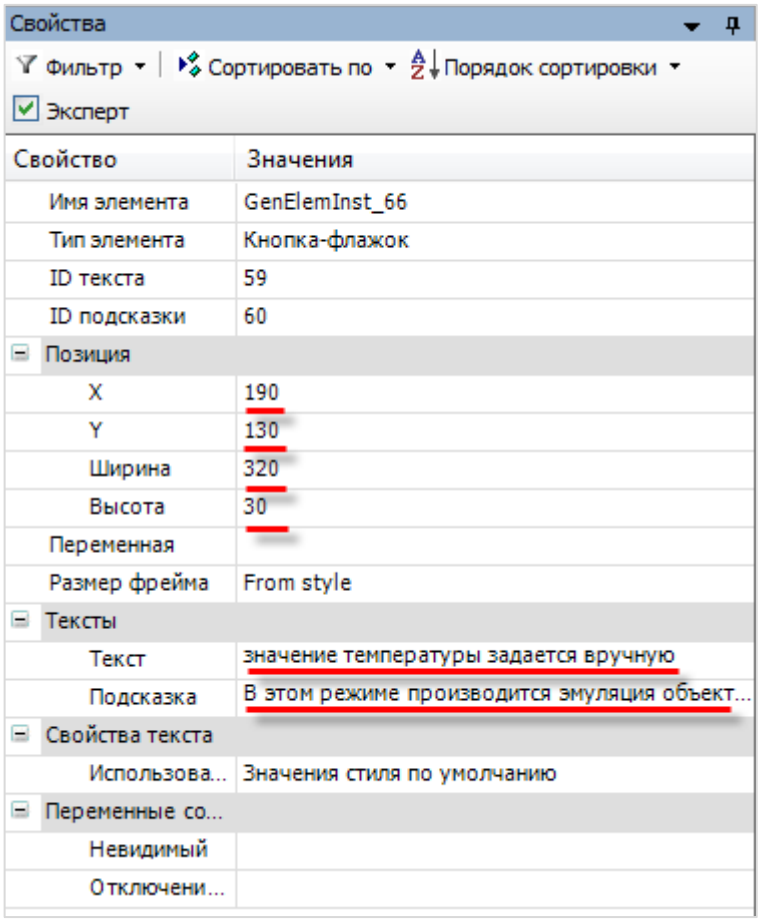


Рисунок 7.26 – Настройка элемента Чекбокс

Подсказка представляет собой текст, всплывающий при наведении на элемент курсора. Подсказка видна только в режиме запуска проекта.

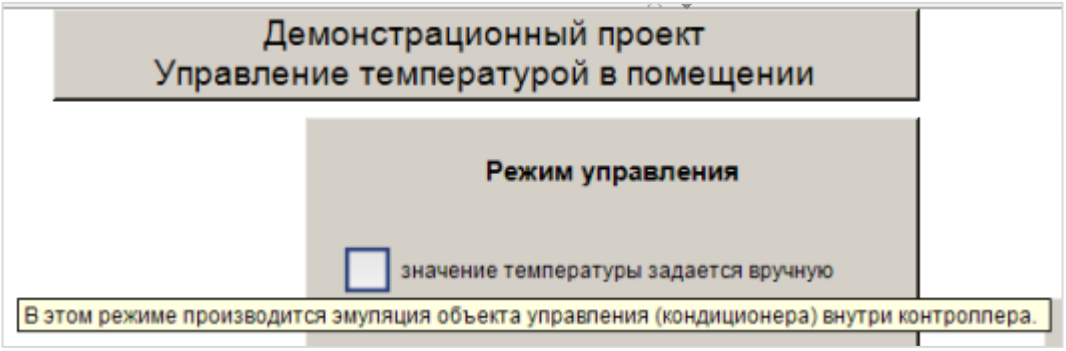


Рисунок 7.27 – Всплывающая подсказка

IV. Элемент **Кнопка** (продолжение пп. I)

Чтобы отображать значения параметров и управлять ими следует создать информационные окна и кнопки управления на панели управления температурой с помощью элемента **Кнопка**.

Процесс создания кнопок описан в пп. I. Так как шрифт, цвет и т. д. не имеют принципиального значения, то кнопки не требуют дополнительной настройки.

Чтобы разместить данные кнопки поверх панели управления температурой, их следует перенести в ***слой выше***:

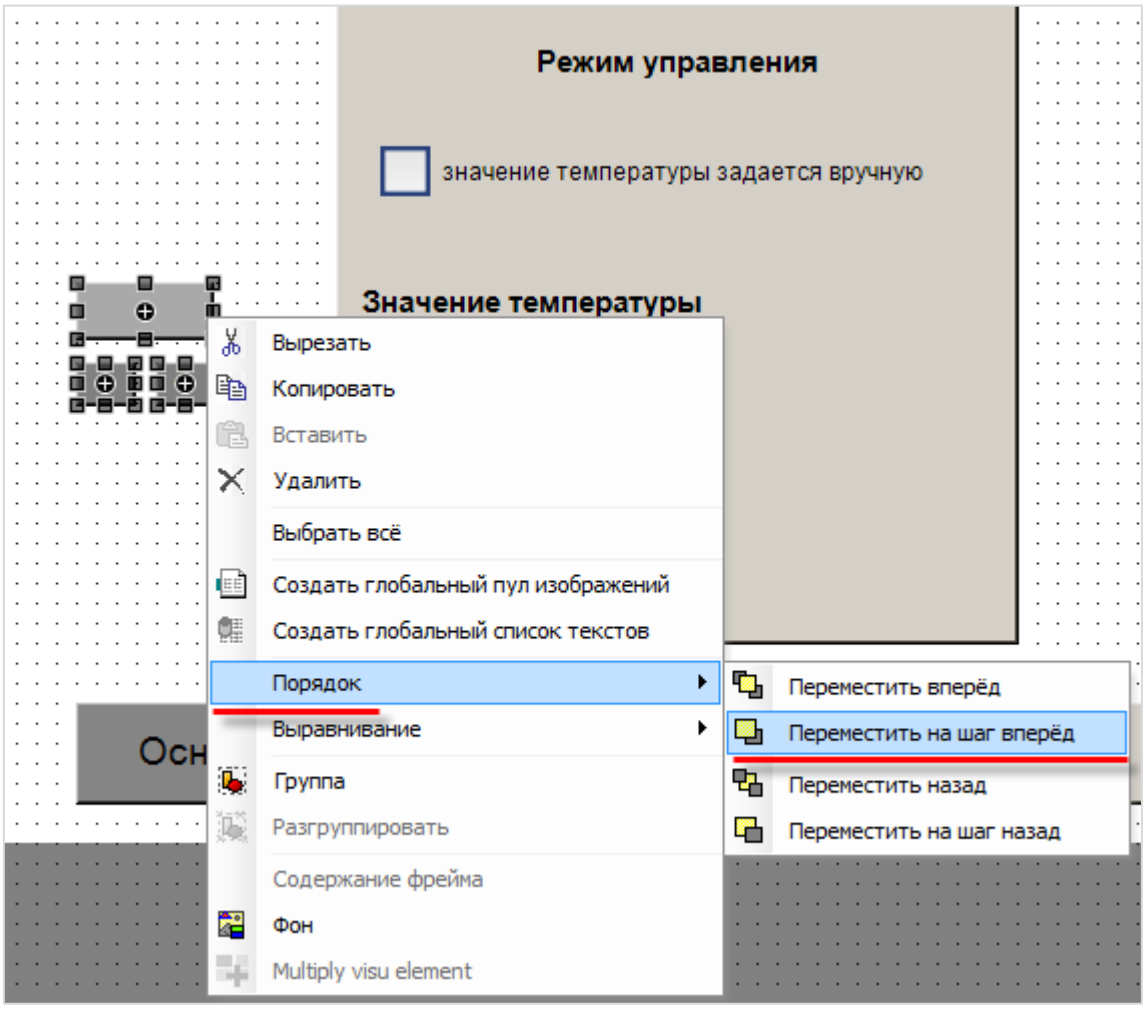


Рисунок 7.28 – Перенос элементов на слой выше

Затем созданные элементы следует скопировать и разместить на панели управления температурой, как на рисунке ниже:

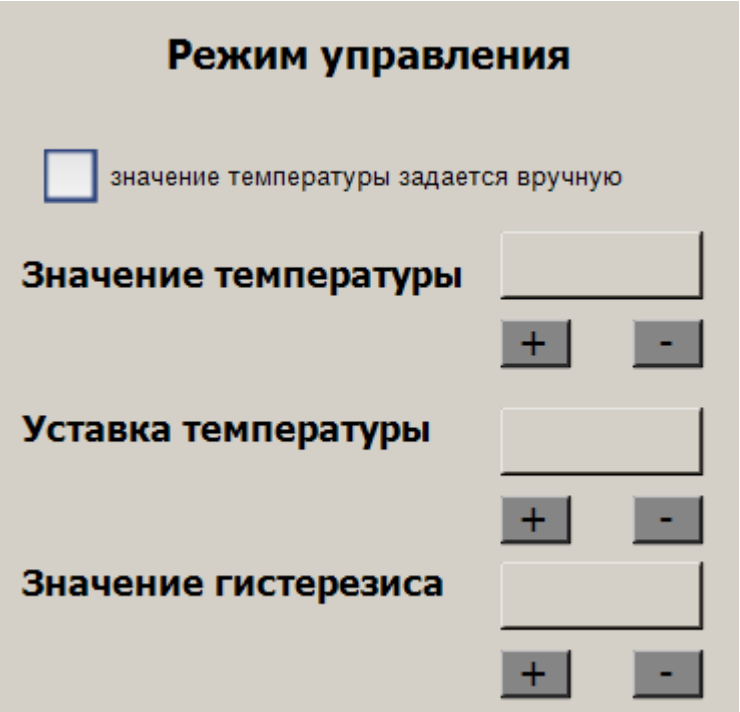


Рисунок 7.29 – Панель управления температурой

V. Элемент **Отображение линейки**

Для отображения текущего значения температуры будет использоваться элемент **Отображение линейки** (вкладка **Элементы управления измерением**).

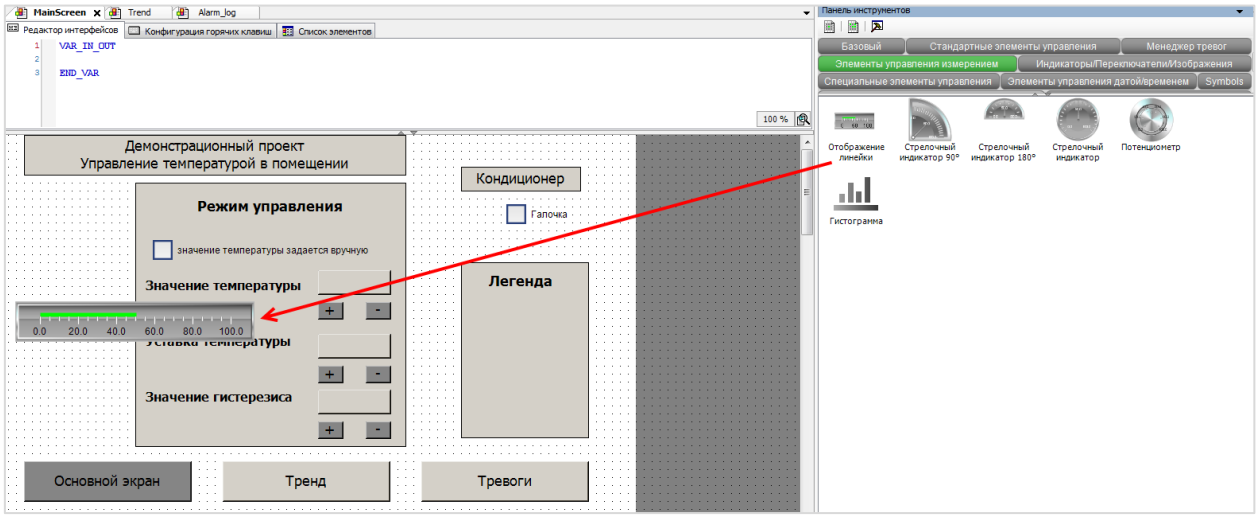


Рисунок 7.30 – Добавление элемента Отображение линейки

Настройки элемента:

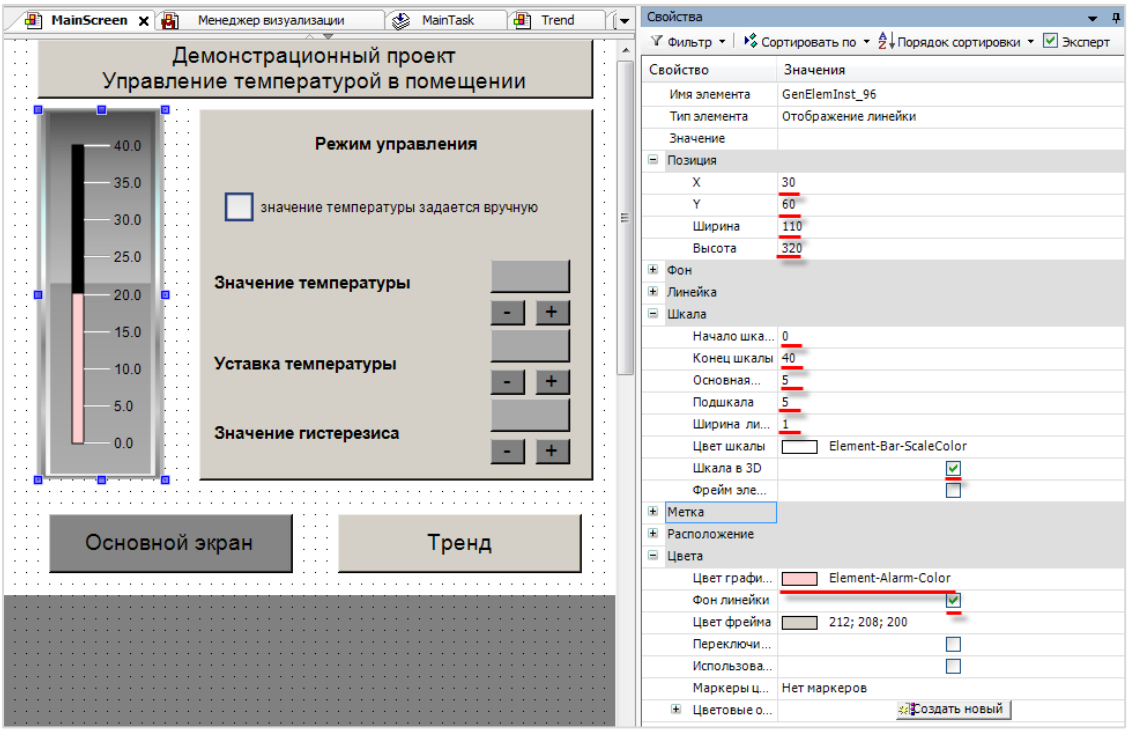


Рисунок 7.31 – Настройки элемента Отображение линейки

VI. Элемент **Клавишный выключатель**

Для включения/выключения кондиционера будет использоваться элемент Клавишный **выключатель** (вкладка **Индикаторы/Переключатели/Изображения**).

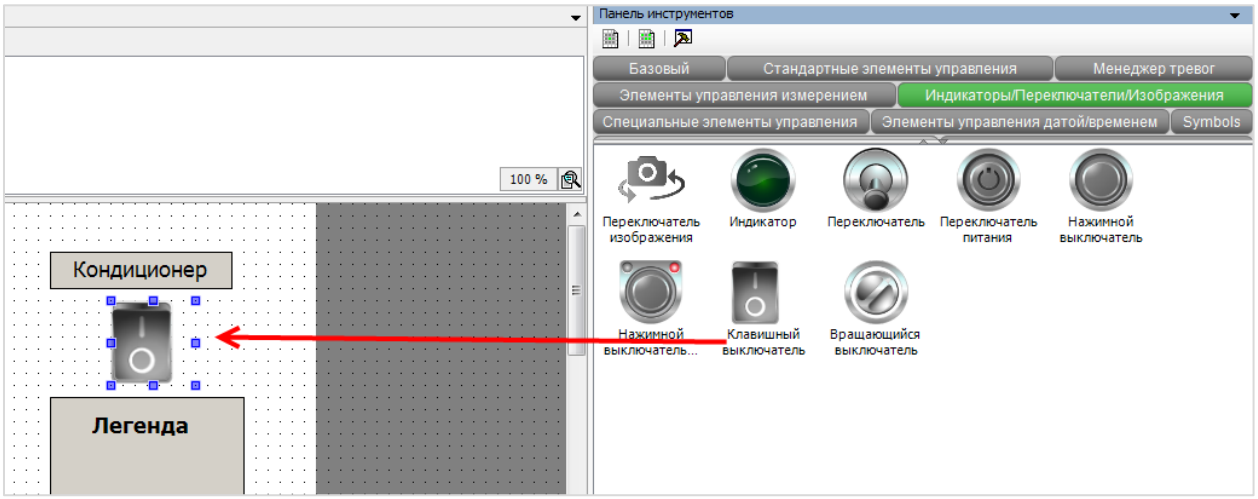


Рисунок 7.32 – Добавление элемента Клавишный выключатель

Настройки элемента:

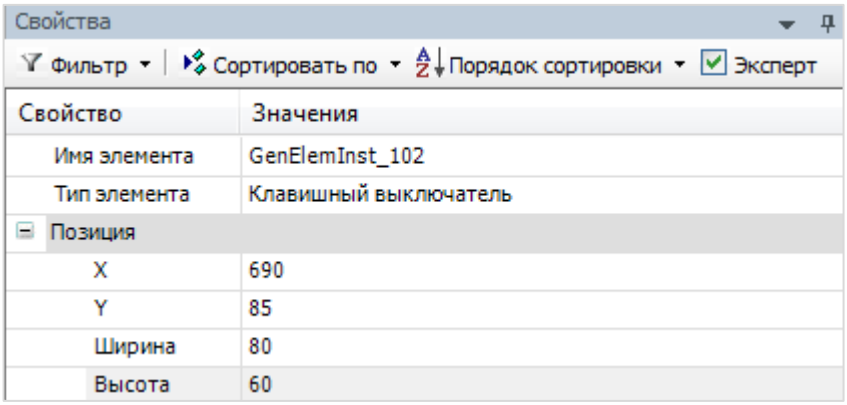


Рисунок 7.33 – Настройки элемента Клавишный выключатель

VII. Элемент **Индикатор**

Для отображения текущего режима работы кондиционера будет использоваться элемент **Индикатор** (вкладка **Индикаторы/Переключатели/Изображения**).

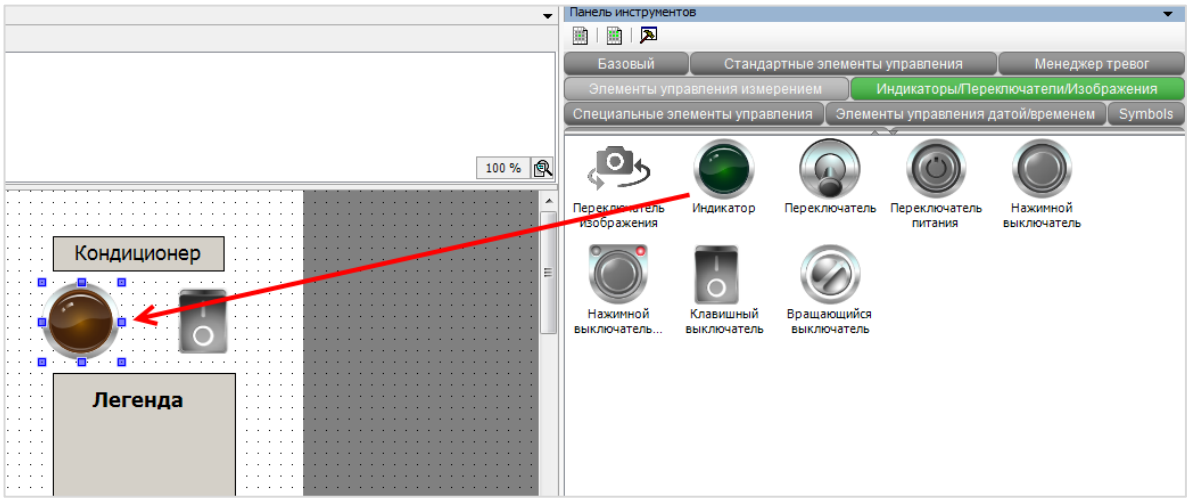


Рисунок 7.34 – Добавление элемента Индикатор

Режим работы кондиционера:

- нагрев;
- охлаждение;
- режим ожидания;
- выключен.

Для отображения четырех состояний воспользуемся четырьмя индикаторами разных цветов. После привязки к ним переменных (п. 7.5) следует наложить эти индикаторы друг на друга для создания эффекта **многопозиционного индикатора**.

Предварительно индикаторы следует разместить за пределами рабочей области – все они будут обладать одинаковыми размерами (ширина – 70, высота – 70) и разными цветами:

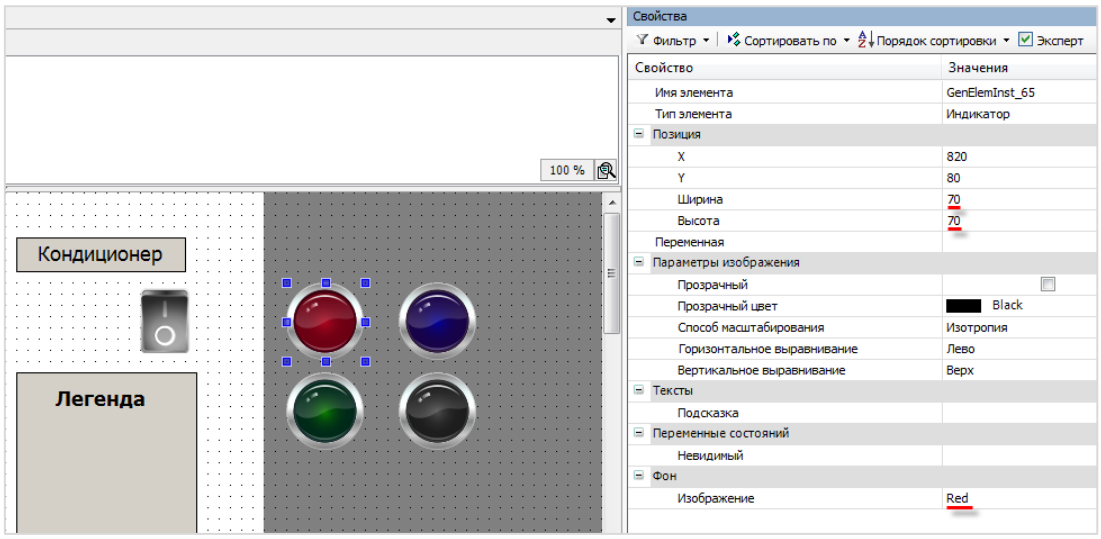


Рисунок 7.35 – Настройки элемента Индикатор

Такие же индикаторы следует добавить на панель легенды кондиционера и снабдить их поясняющими надписями (создание надписей описано в пп. II):

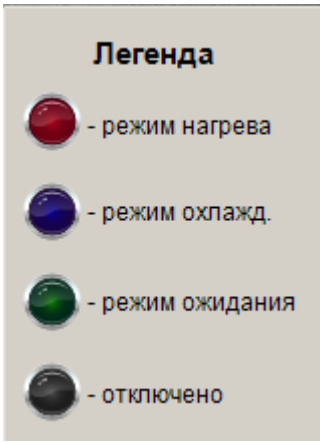


Рисунок 7.36 – Панель легенды кондиционера

После выполнения всех действий из п. 7.3.2 экран визуализации **MainScreen** должен выглядеть следующим образом:

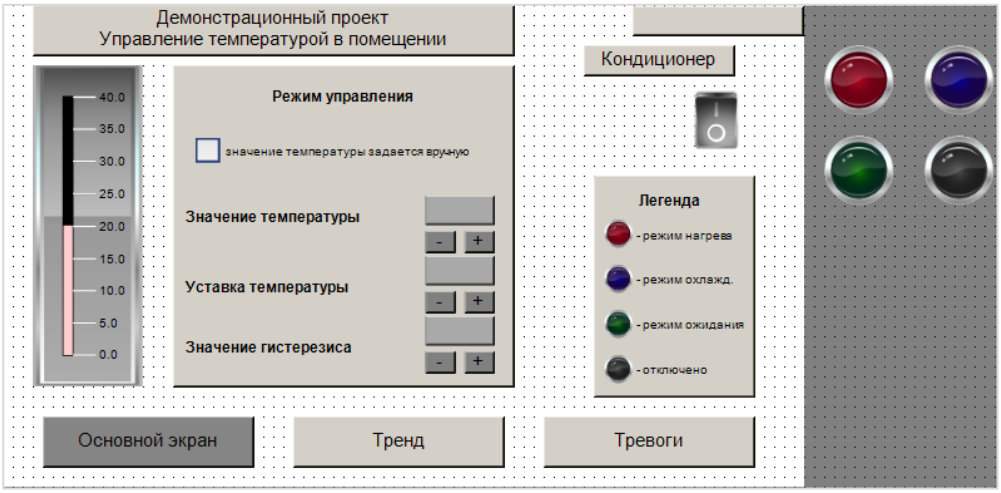


Рисунок 7.37 – Внешний вид экрана MainScreen (после п. 7.3.2)

Окончательная доработка экрана (привязка переменных к визуализации, настройка действий кнопок, наложение индикаторов друг на друга) описывается в п. 7.5.

7.3.3 Наполнение экрана Trend

Экран **Trend** будет использоваться для графического отображения текущего значения параметров (температуры, уставок температуры, уставок гистерезиса, аварийных уставок), установки значений аварийных уставок и сигнализации о выходе температуры за аварийные уставки с помощью мигания аварийной лампы.

После выполнения действий, описанных в п. 7.3.2 пп. l (добавление кнопок перехода между экранами и панели системного времени), экран **Trend** выглядит следующим образом:

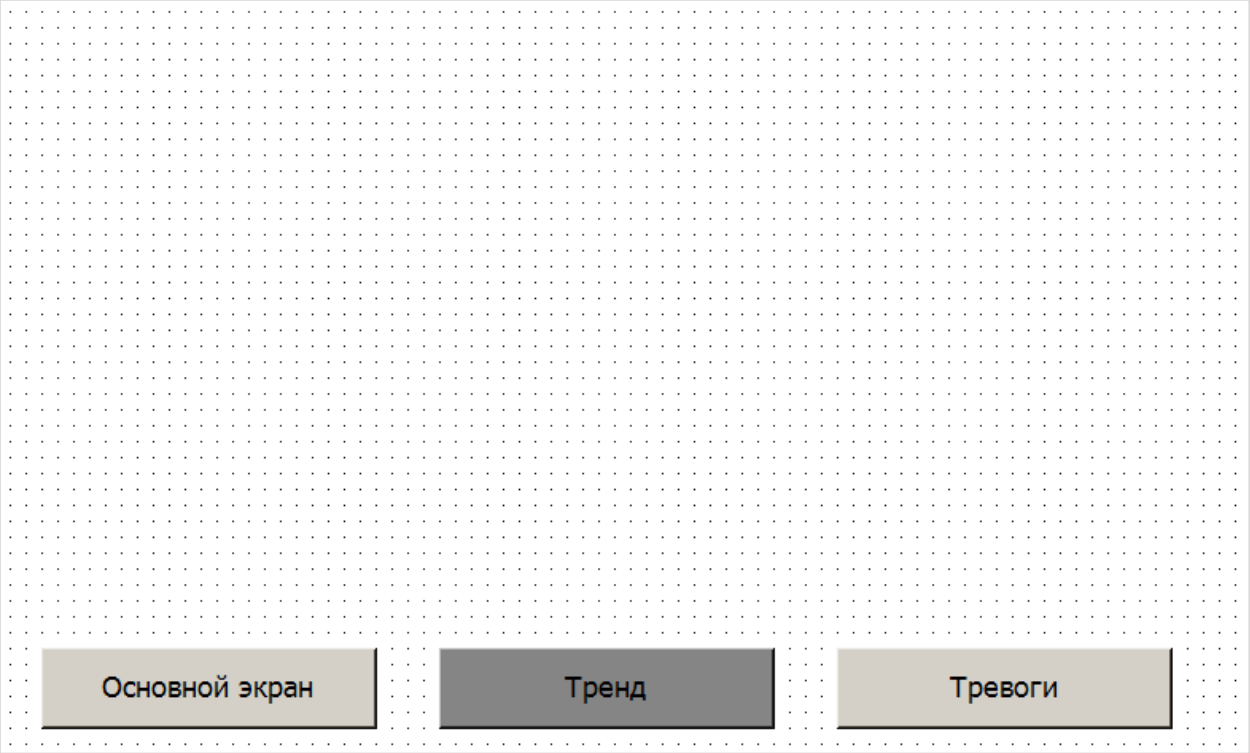


Рисунок 7.38 – Внешний вид экрана Trend (после п. 7.3.2)

Затем следует скопировать на него название экрана (элемент **Прямоугольник**), панель аварийных уставок (элемент **Прямоугольник** с наложенными поверх элементами **Текстовое поле**) и аварийную лампу (элемент **Индикатор**) с пустым названием (элемент **Кнопка**). Название лампы будет задаваться пользователем в процессе работы проекта.

Процесс добавления и настройки этих элементов описан в п. 7.3.2, пп. I, пп. IV (**Прямоугольник** и **Кнопка**), пп. II (**Текстовое поле**), пп. VII (**Индикатор**).

В результате экран должен выглядеть следующим образом:

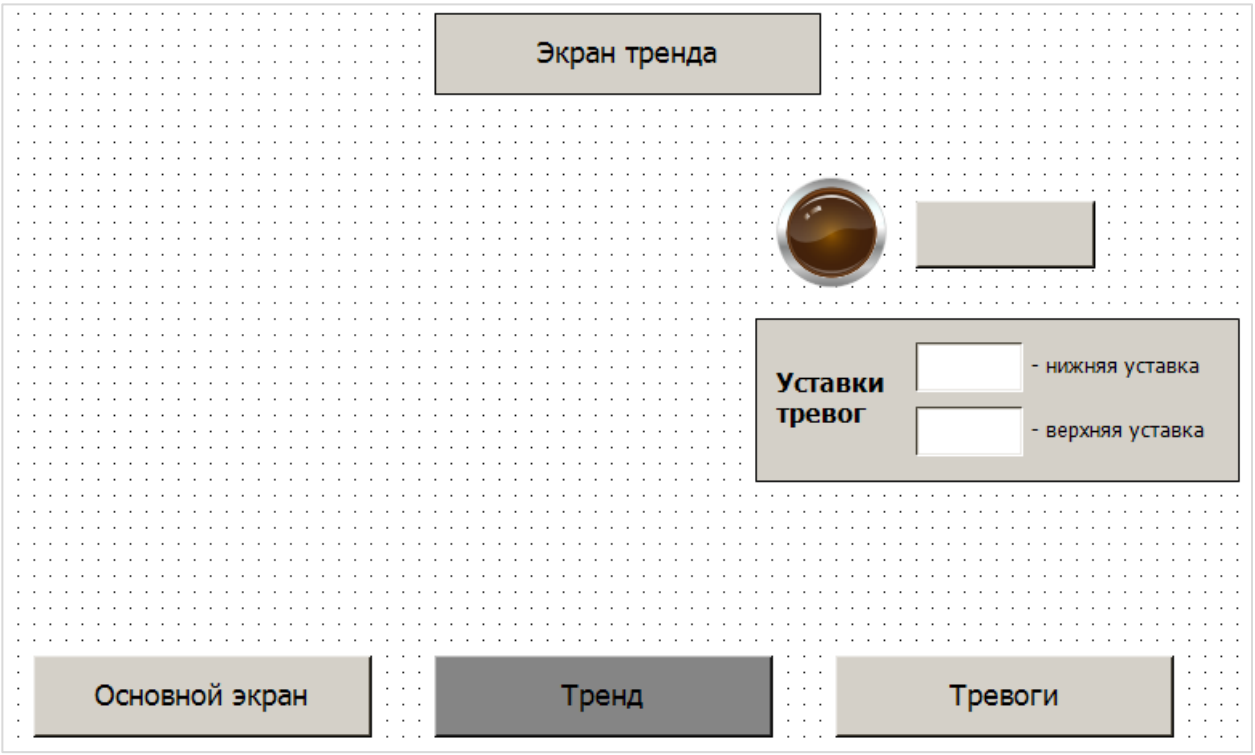


Рисунок 7.39 – Внешний вид экрана Trend

Для графического отображения значений переменных в **CODESYS** существует два элемента – **Трассировка** и **Тренд**. Трассировка отображает значения только за крайне ограниченный отрезок времени. Тренд позволяет просматривать историю изменений переменных. Настройка элементов практически аналогична. В данном примере будет использоваться только элемент **Тренд**.

Элемент **Тренд** следует добавить на экран **Trend** (вкладка **Специальные элементы управления**). Появится окно конфигурации тренда. Не производя никаких настроек (настройки будут рассмотрены в п. 7.5) следует нажать кнопку **ОК**:

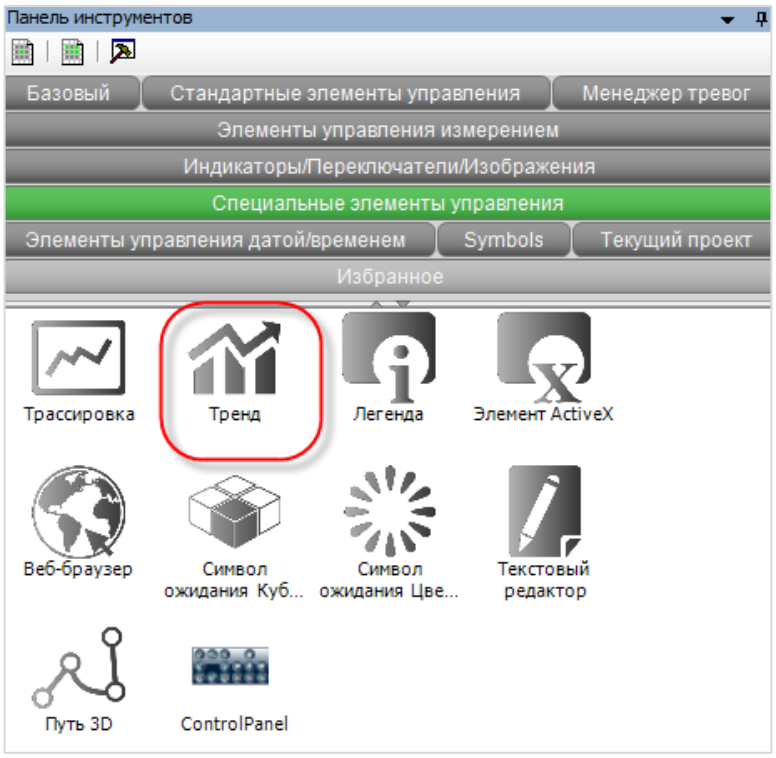


Рисунок 7.40 – Элемент Тренд на Панели инструментов

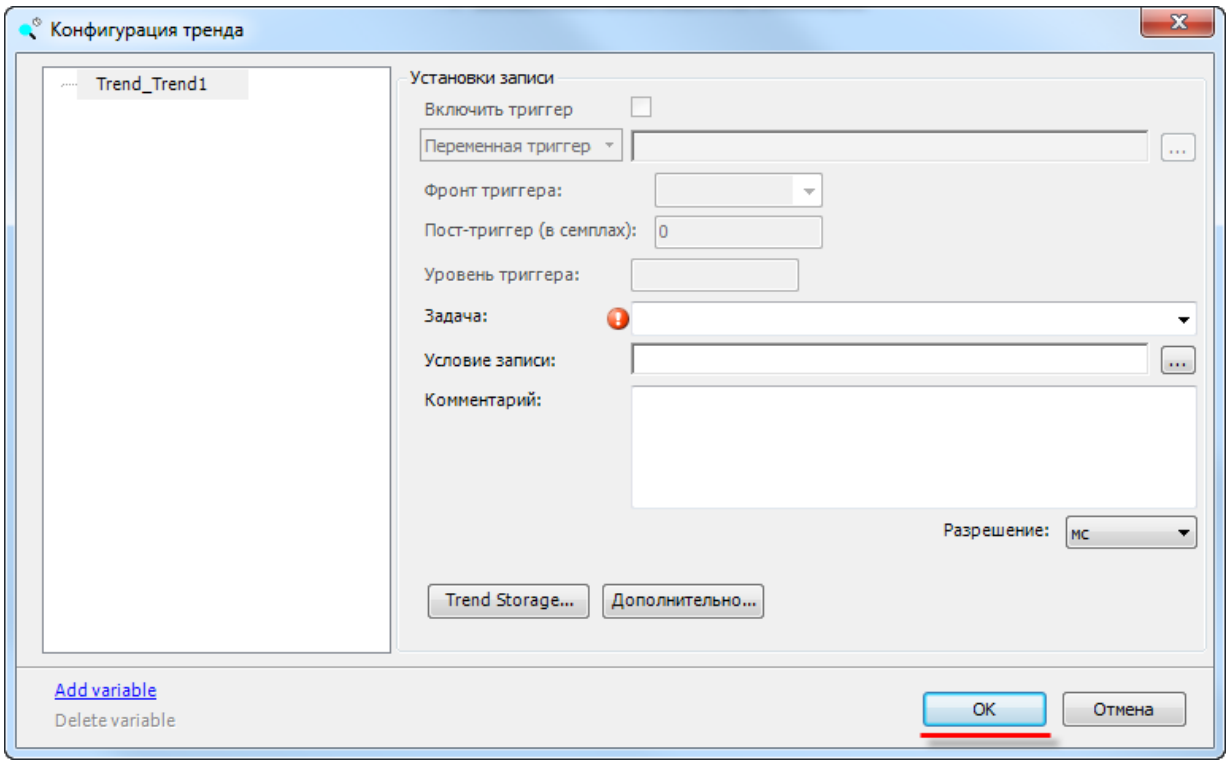


Рисунок 7.41 – Окно конфигурации тренда

После добавления элемента **Тренд** в дереве проекта появляются новые компоненты:

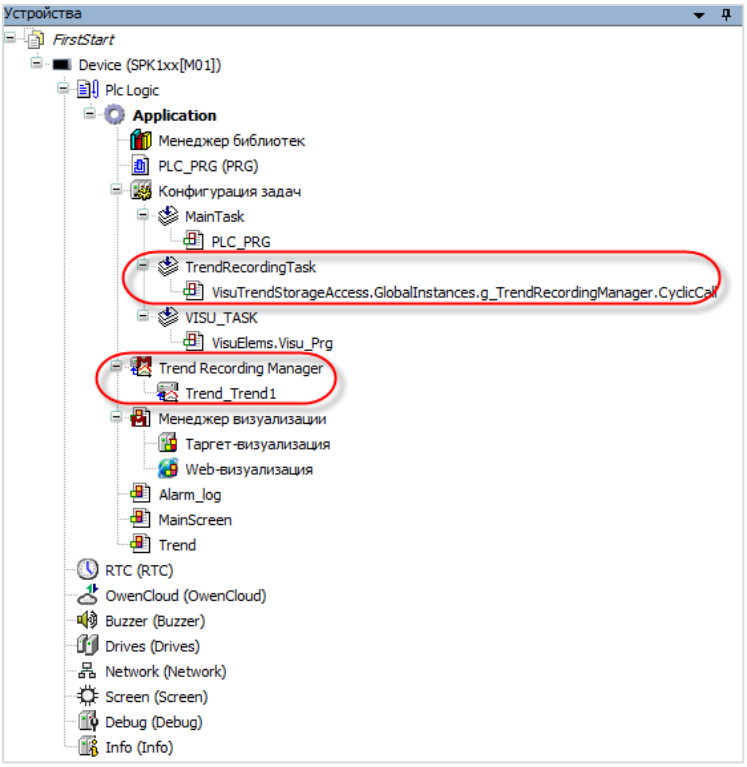


Рисунок 7.42 – Дерево проекта после добавления элемента Тренд

Настройки элемента **Тренд**:

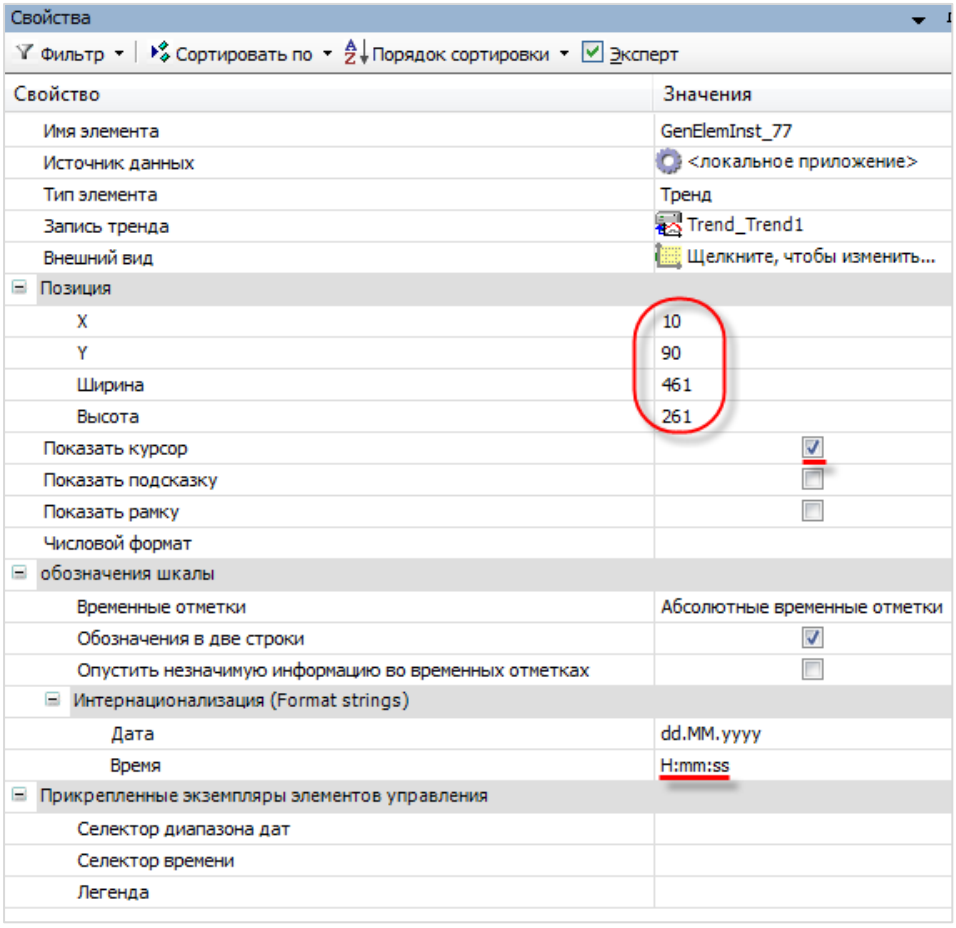


Рисунок 7.43 – Настройки элемента Тренд

После настройки экран **Trend** будет выглядеть следующим образом:

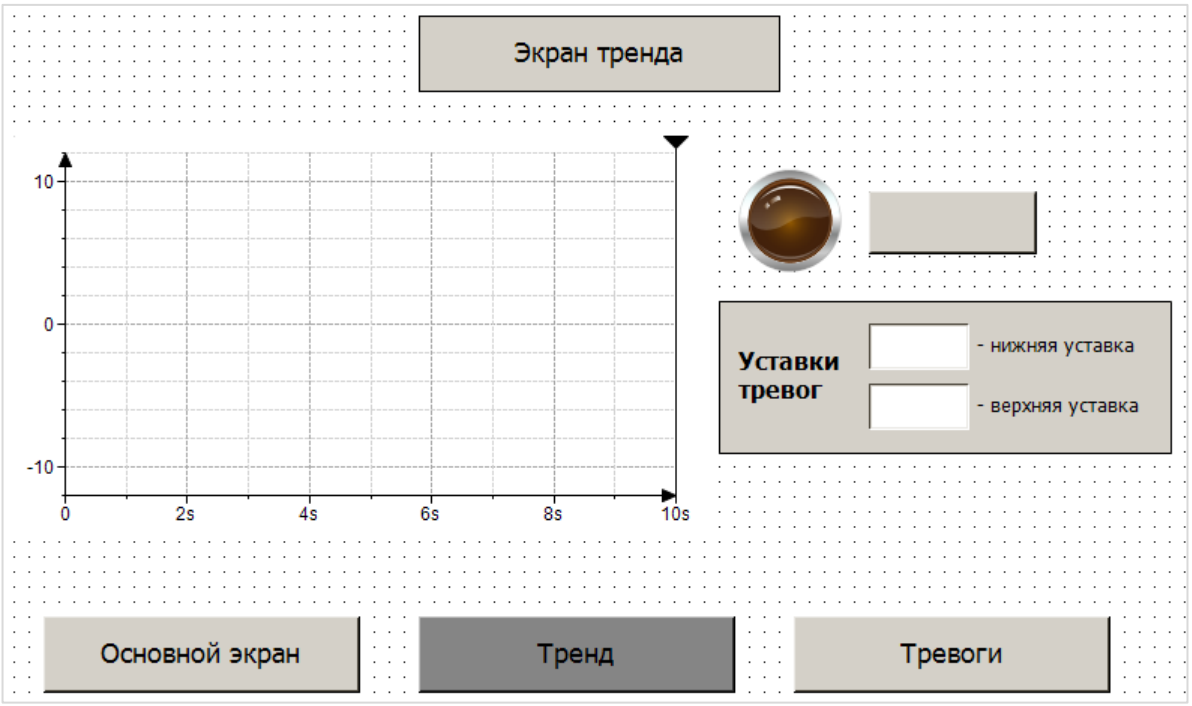


Рисунок 7.44 – Экран Trend после добавления тренда

Затем следует нажать на тренд **ПКМ** и выбрать пункт **Вставить компоненты для управления трендом**:

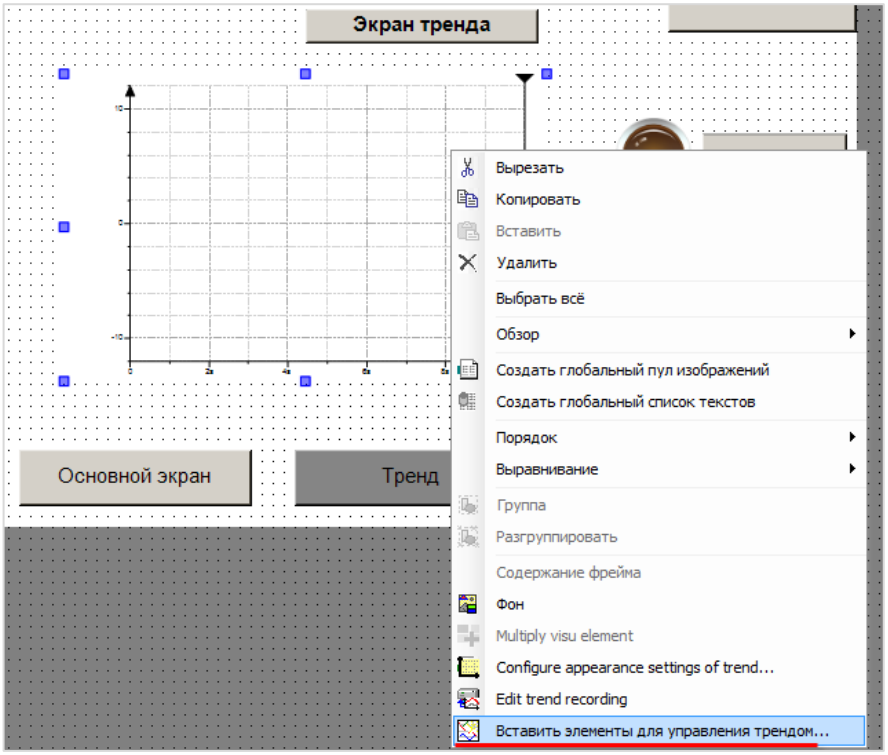


Рисунок 7.45 – Добавление элементов управления трендом

Настройки **Мастера трендов** следует оставить по умолчанию:

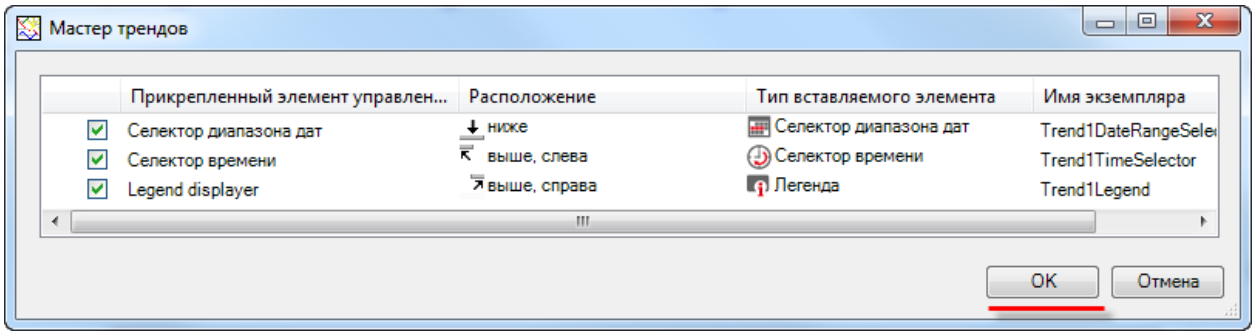


Рисунок 7.46 – Меню Мастера трендов

В результате экран **Trend** будет выглядеть так:

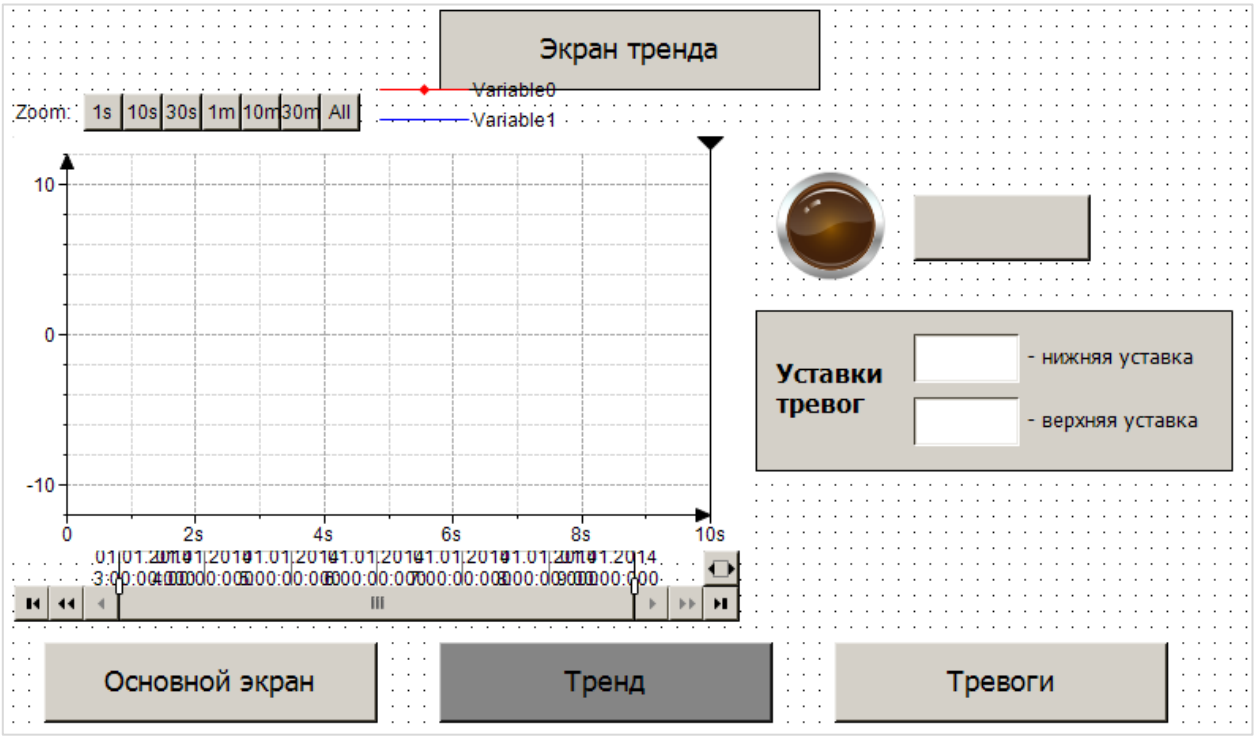


Рисунок 7.47 – Элементы управления трендом

Настройки элементов управления трендом:

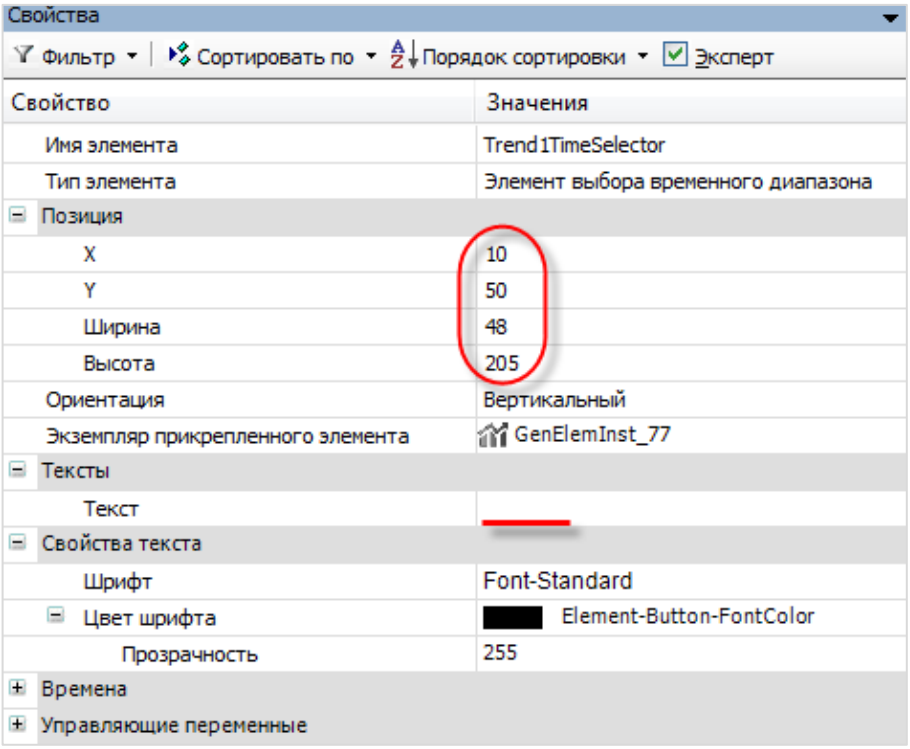


Рисунок 7.48 – Настройки элемента выбора временного диапазона

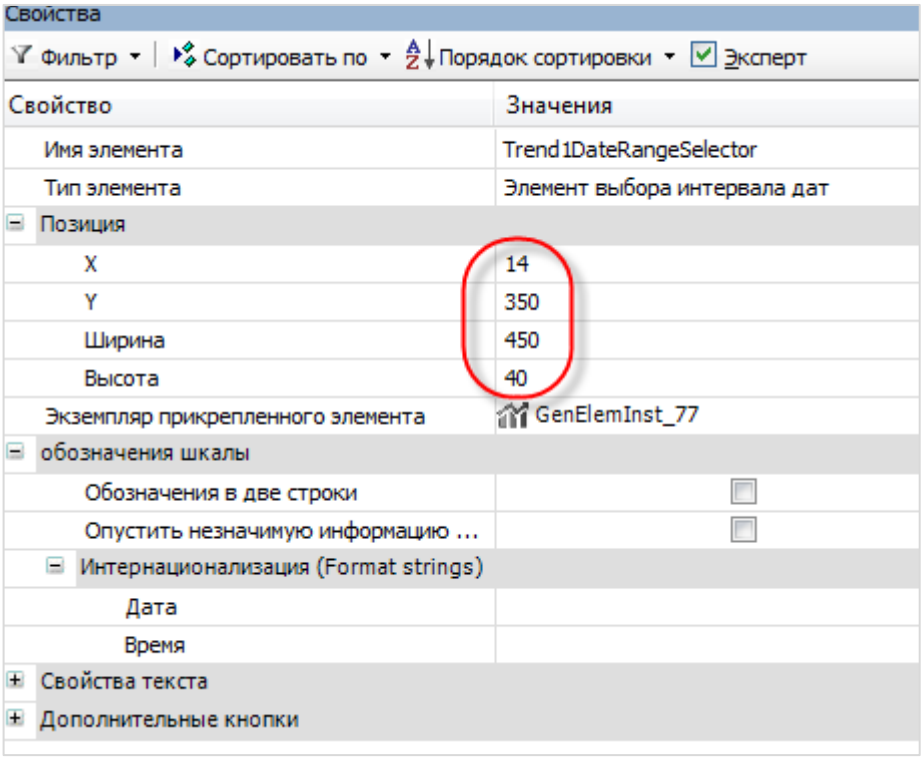


Рисунок 7.49 – Настройки элемента выбора интервала дат

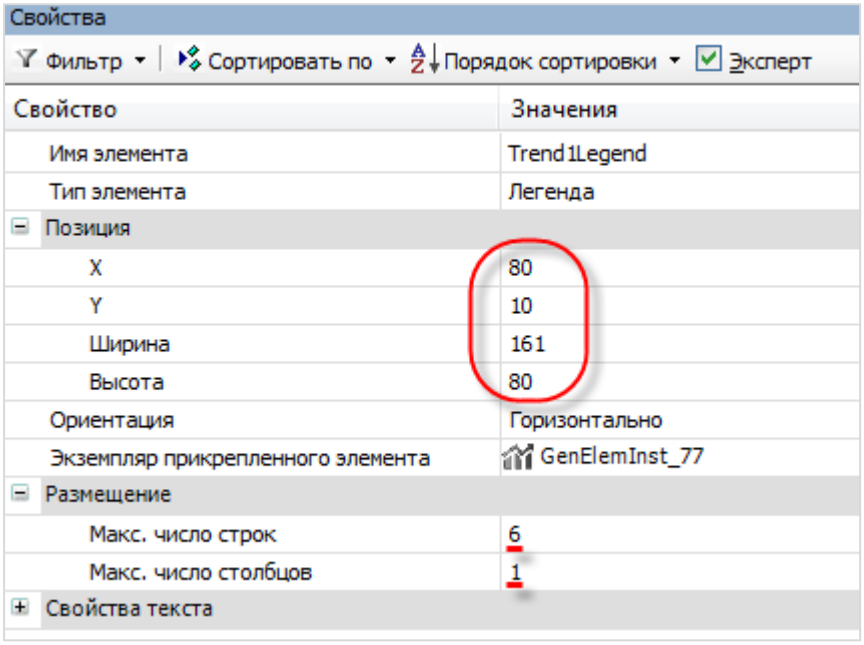


Рисунок 7.50 – Настройки легенды

Для визуальной настройки тренда (шаг сетки, цвет фона и т. д.) следует нажать на него **ПКМ** и в контекстном меню выбрать пункт **Параметры отображения тренда**.

После выполнения действий, описанных в п. 7.3.3, экран **Trend** должен выглядеть следующим образом:

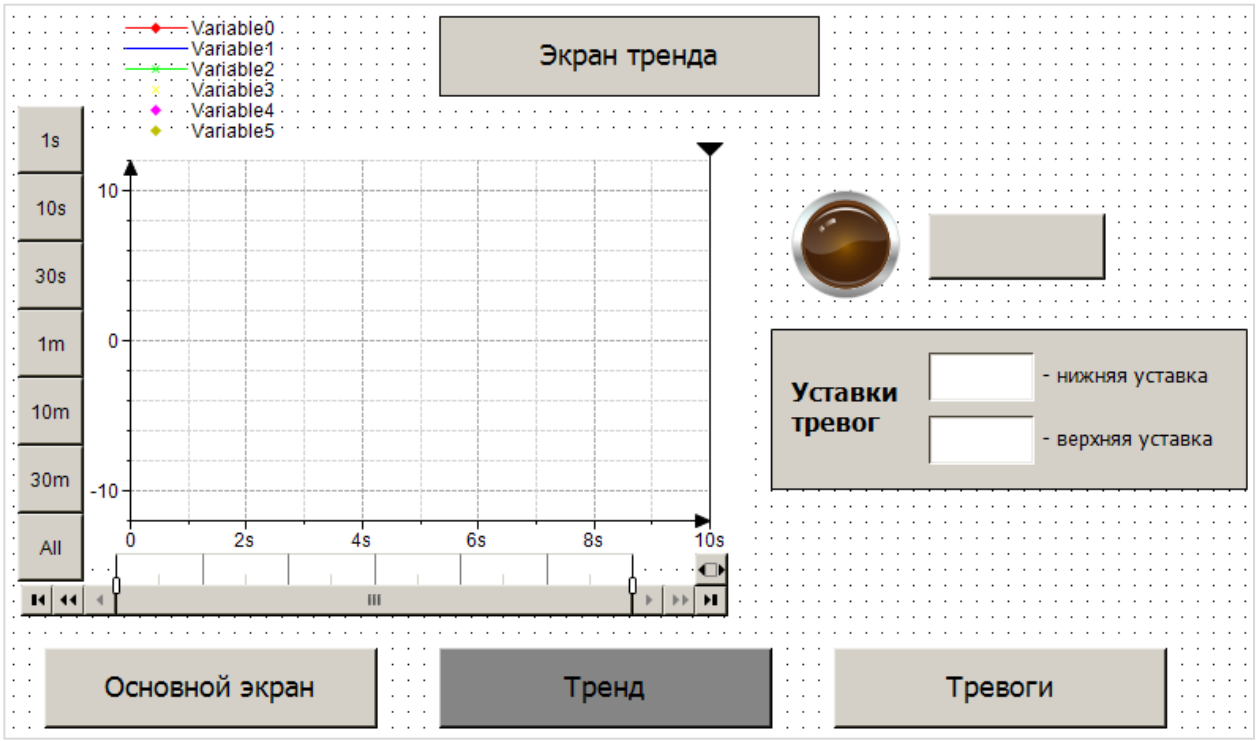


Рисунок 7.51 – Внешний вид экрана Trend (после п. 7.3.3)

7.3.4 Наполнение экрана Alarm_log

Экран **Alarm_log** будет использоваться для отображения **Журнала тревог**, в котором выводятся сообщения о выходе значения температуры за уставки гистерезиса и аварийные уставки.

После выполнения действий, описанных в п. 7.3.2 пп. l (добавление кнопок перехода между экранами и панели системного времени), экран **Alarm_log** должен выглядеть следующим образом:

Рисунок 7.52 – Экран Alarm_log (после п. 7.3.2)

Затем следует добавить название экрана с помощью элемента **Прямоугольник**:

Рисунок 7.53 – Экран Alarm_log – добавление названия экрана

Далее следует добавить элемент **Таблица тревог** (вкладка **Менеджер тревог**):

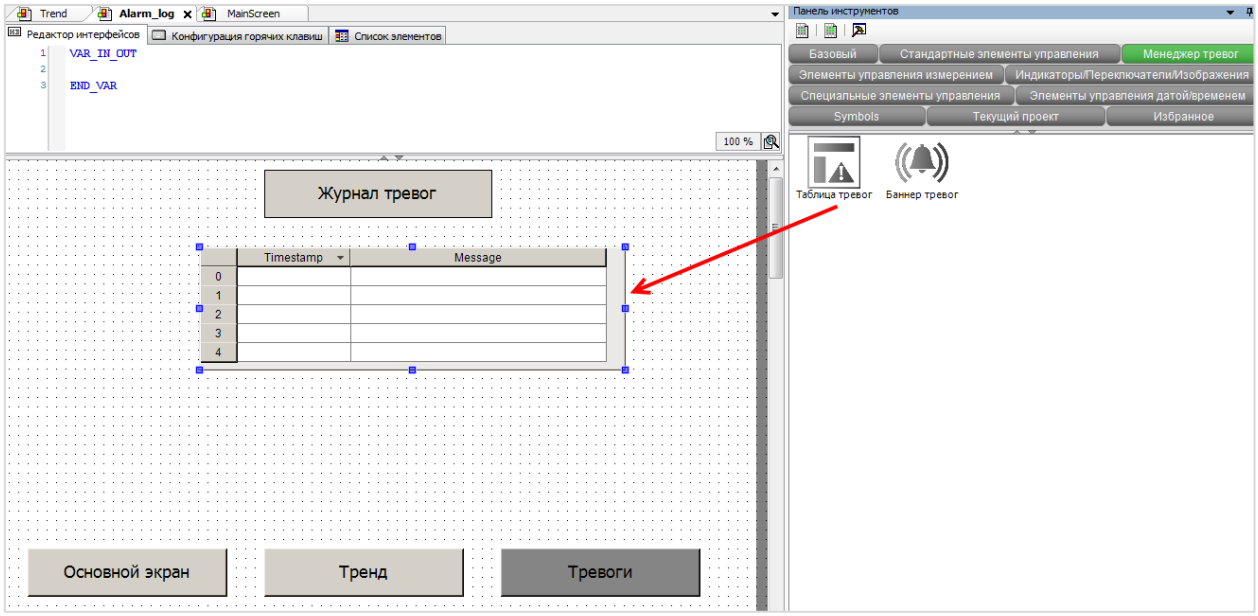


Рисунок 7.54 – Добавление элемента Таблица тревог

Настройки элемента:

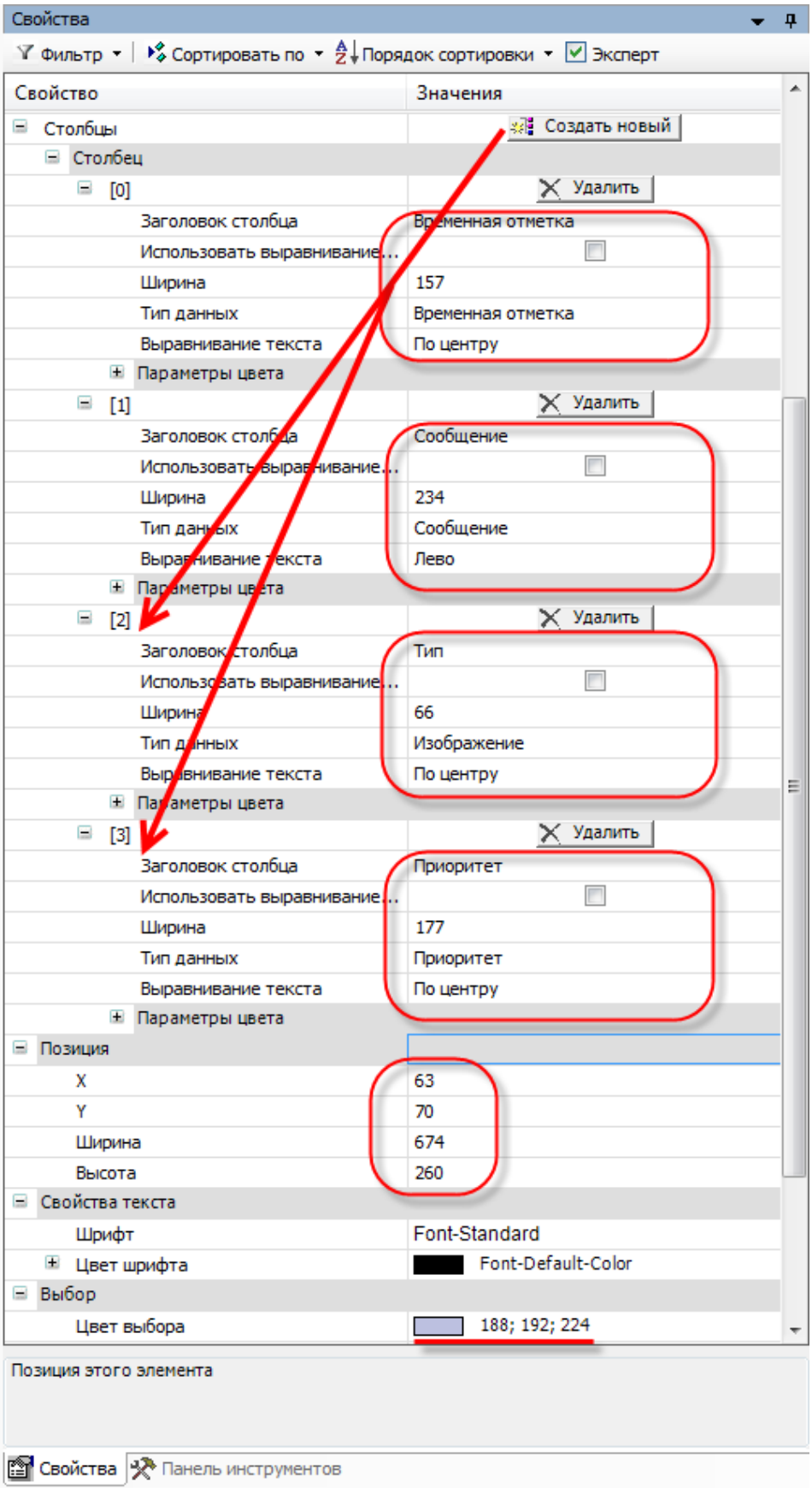


Рисунок 7.55 – Настройки элемента Таблица тревог

В результате **Таблица тревог** должна выглядеть следующим образом:

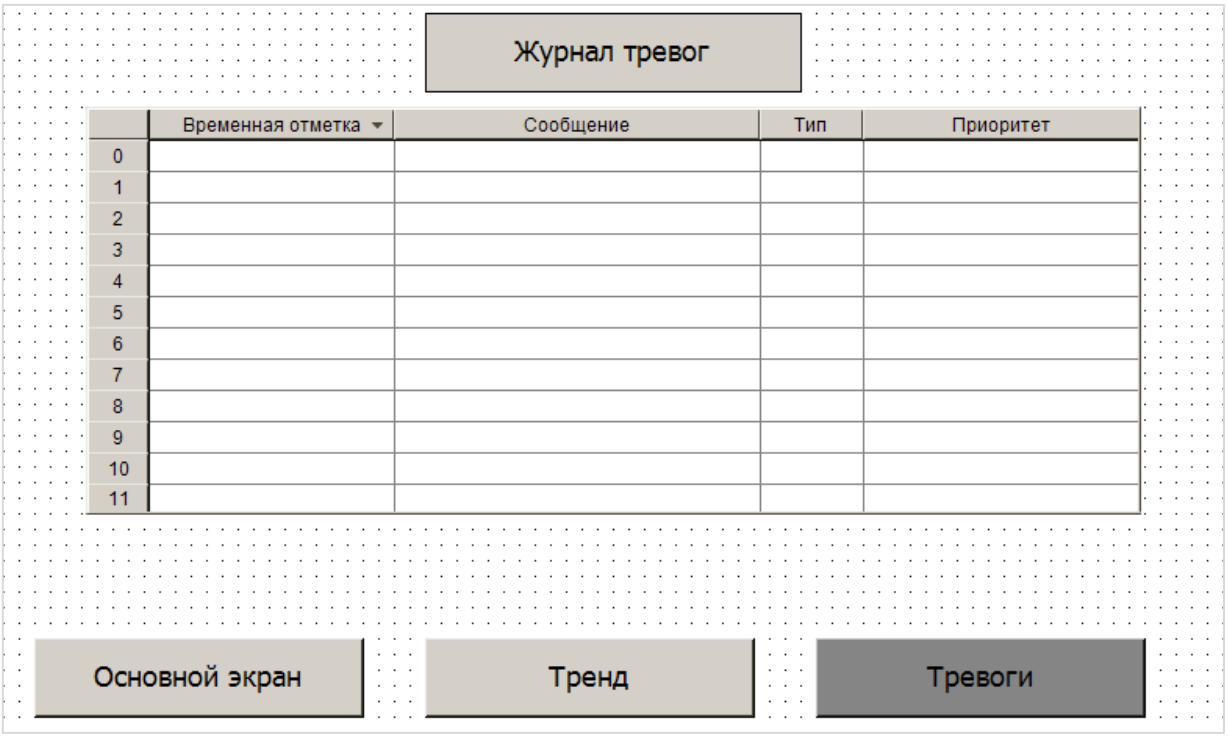


Рисунок 7.56 – Внешний вид Таблицы тревог после настройки

Далее следует нажать на Таблицу тревог **ПКМ** и выбрать пункт **Вставить элементы для подтверждения тревог**:

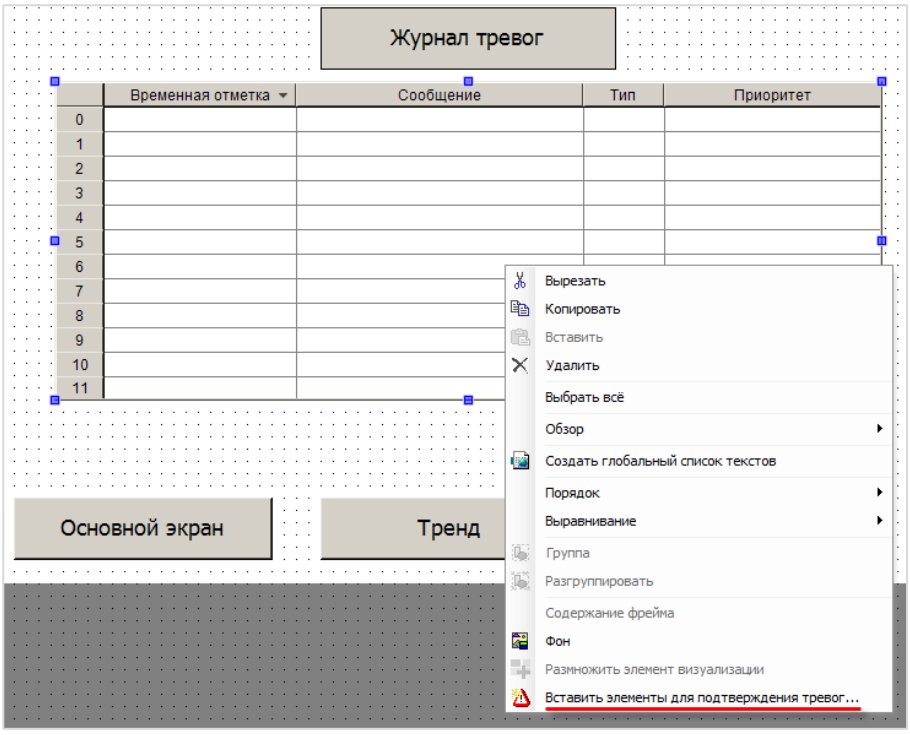


Рисунок 7.57 – Добавление элементов подтверждения тревог

Появится окно **Мастера таблицы тревог**. Его настройки следует оставить ***по умолчанию***.

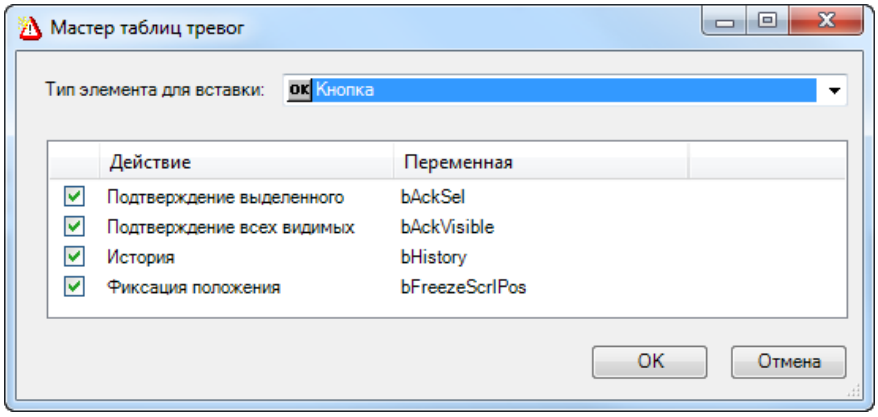


Рисунок 7.58 – Окно Мастера таблицы тревог

В результате появятся элементы подтверждения тревог:

Рисунок 7.59 – Элементы подтверждения тревог

В настройках элементов следует изменить размеры, положение и названия. После этого экран **Alarm_log** должен выглядеть следующим образом:

Рисунок 7.60 – Экран Alarm_log (после п. 7.3.4)

Кнопки подтверждения выполняют следующие функции:

1. Кнопка **Подтвердить выбранное** используется для подтверждения (квитирования) выделенной тревоги.
2. Кнопка **Подтвердить всё** используется для подтверждения всех тревог сразу.
3. Кнопка **История** переключает журнал в режим истории для отображения информации о прошедших тревогах.
4. Кнопка **Зафиксировать журнал** запрещает автоматическое перелистывание строк журнала в случае появления новых сообщений, что позволяет «заморозить» текущую страницу журнала.

7.3.5 Пул изображений

CODESYS позволяет загружать в проект пользовательские изображения, которые в дальнейшем могут использоваться в процессе разработки экранов визуализации (например, для создания фона экрана). Поддерживается большинство популярных форматов графических файлов, например .jpg, .png, .bmp, .svg и т. д.).

Изображения загружаются в проект через компонент **Пул изображений**, который следует добавить с помощью команды **Добавить объект**:

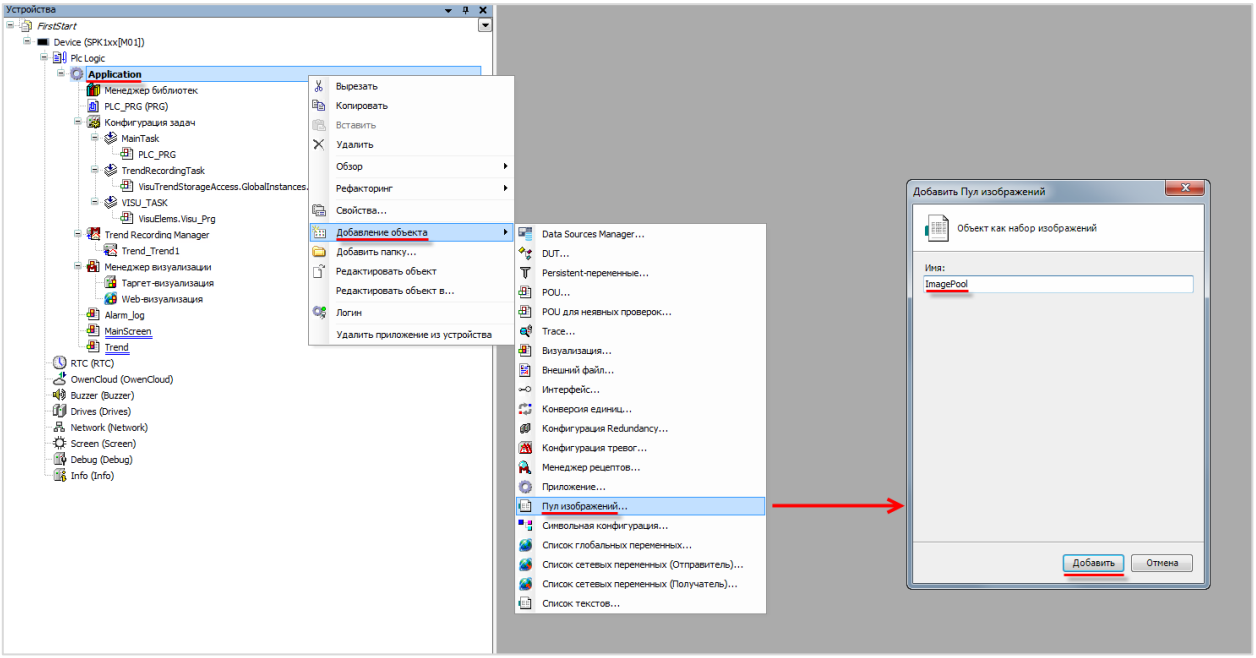


Рисунок 7.61 – Добавление Пула изображений

Компонент открывается двойным нажатием **ЛКМ** на его название:

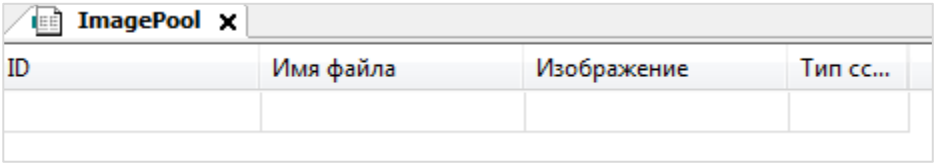


Рисунок 7.62 – Внешний вид Пула изображений

Для добавления изображения следует нажать **ЛКМ** на ячейку **Имя файла** и с помощью появившейся кнопки перейти к выбору файла (название файла не должно содержать кириллицы и спецсимволов):

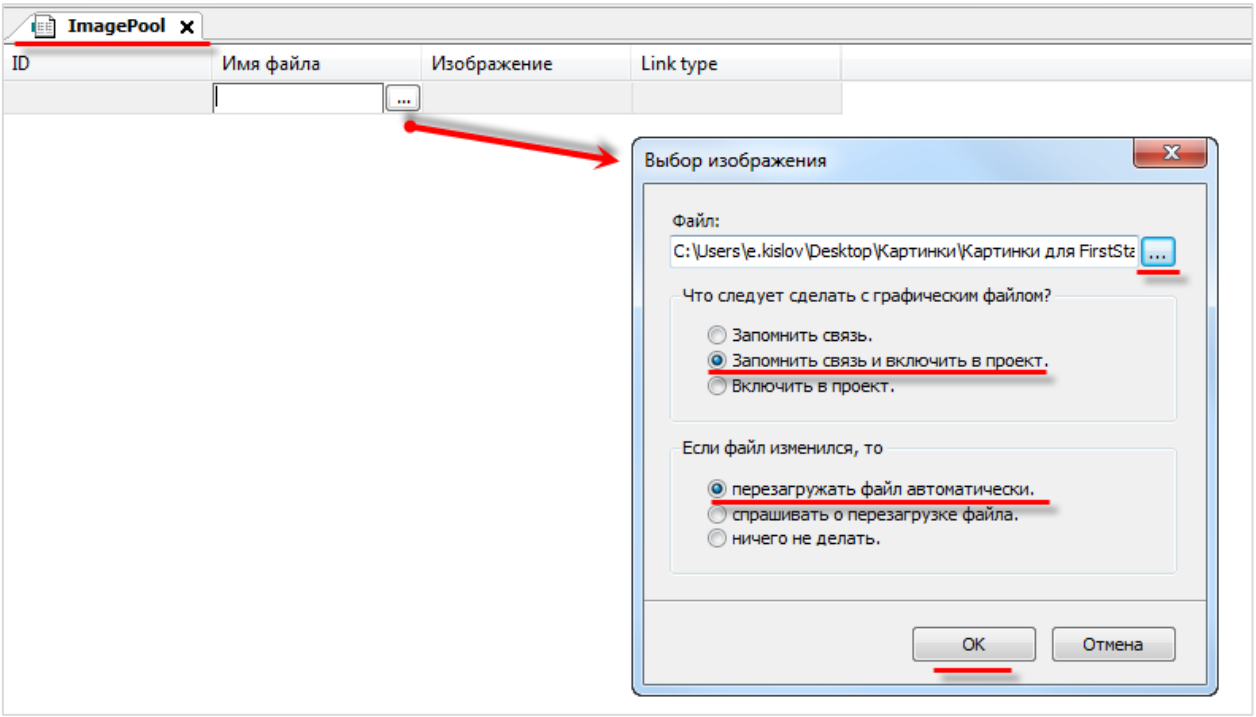


Рисунок 7.63 – Выбор изображения для загрузки

В появившемся окне следует указать путь к графическому файлу и в расположенных ниже меню выбрать пункты **Запомнить связь и включить в проект** и **Перезагружать файл автоматически**. Данные настройки позволяют не совершать дополнительных операций в случае изменения изображения – оно будет автоматически меняться в проекте.

После добавления изображения, его пиктограмма отобразится в **Пуле**, рядом с ним будет указан **идентификатор (ID)** и **тип связи**.

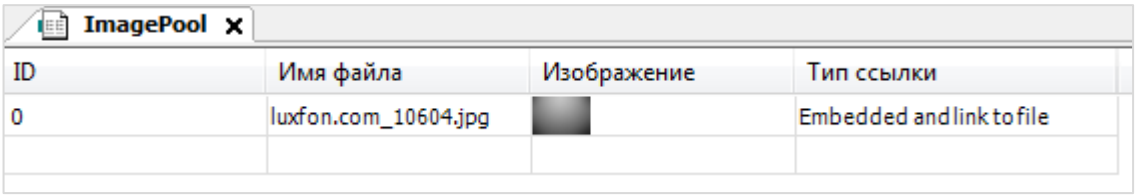


Рисунок 7.64 – Пул изображений после добавления файла

Чтобы использовать добавленную картинку в качестве фона экрана визуализации, следует нажать **ПКМ** на любое место экрана и в контекстном меню выбрать вкладку **Фон**. В появившемся диалоговом окне следует поставить галочку **Изображение** и с помощью кнопки выбора (перекрывается клавишей **Отмена**) открыть **Ассистент ввода изображений**:

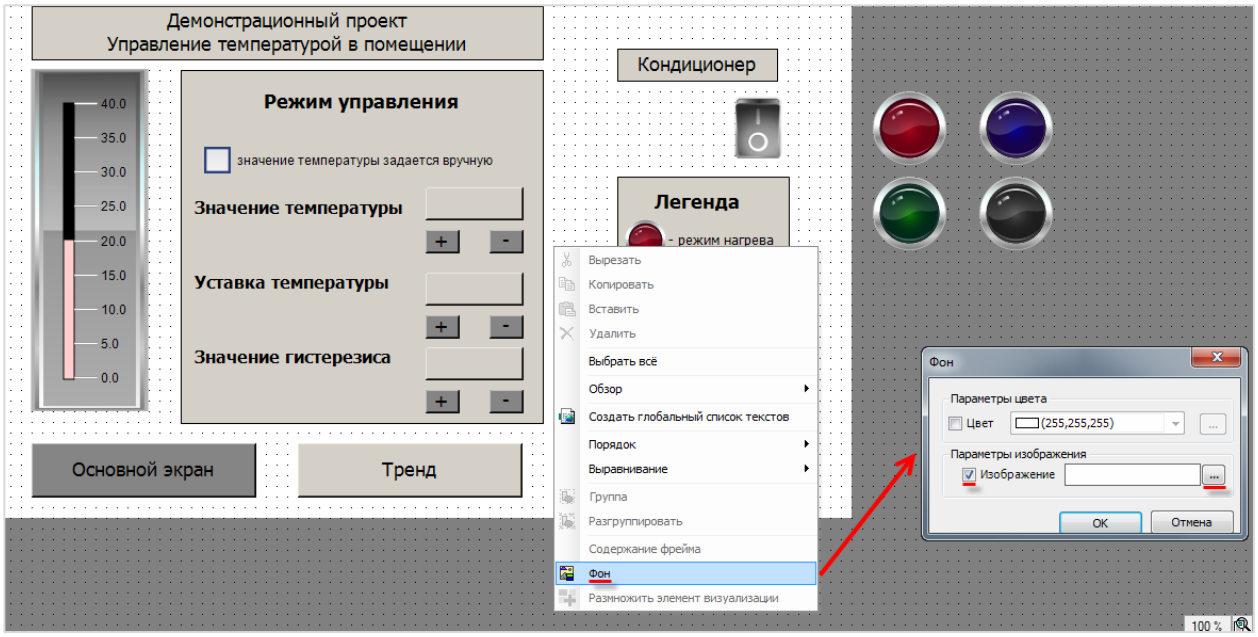


Рисунок 7.65 – Выбор фона экрана визуализации

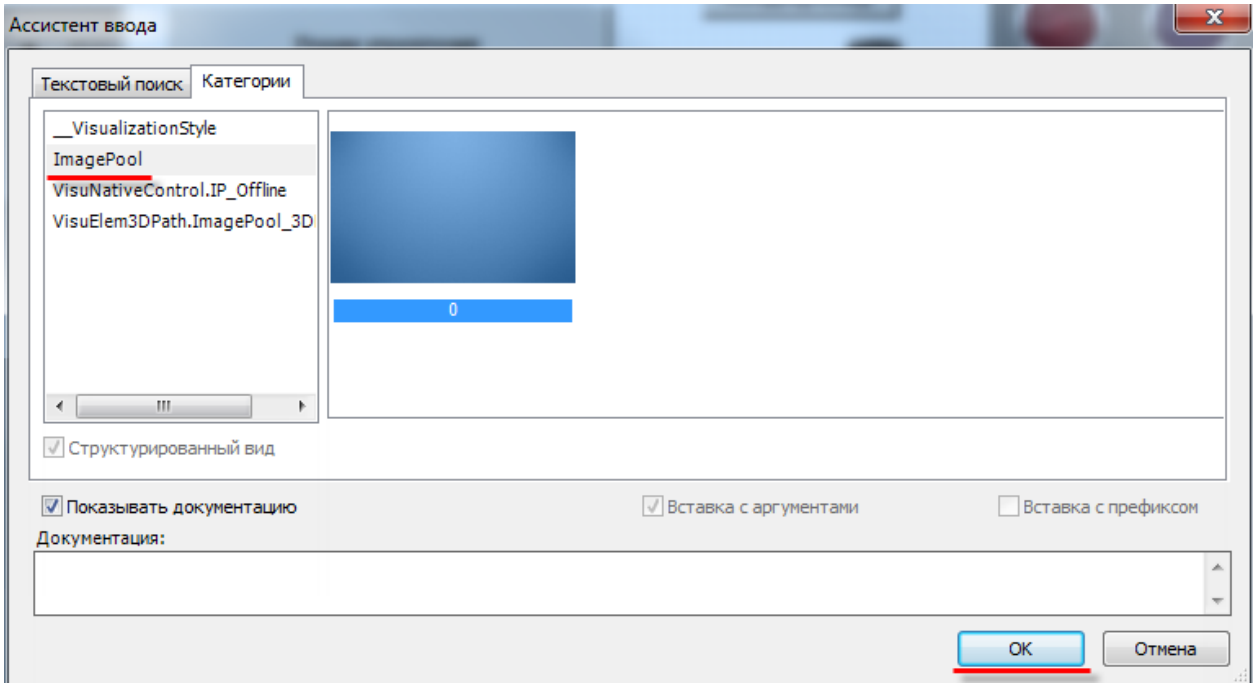


Рисунок 7.66 – Ассистент ввода изображений

На этом разработка экранов визуализации закончена.

Организацию компонентов в дереве проекта можно осуществлять с помощью создания **папок**. Нажатием **ПКМ** на компонент **Applicaton** в дереве проекта можно создать новую папку **Визуализация**, и, ***зажав* ЛКМ**, перенести туда экраны визуализации:

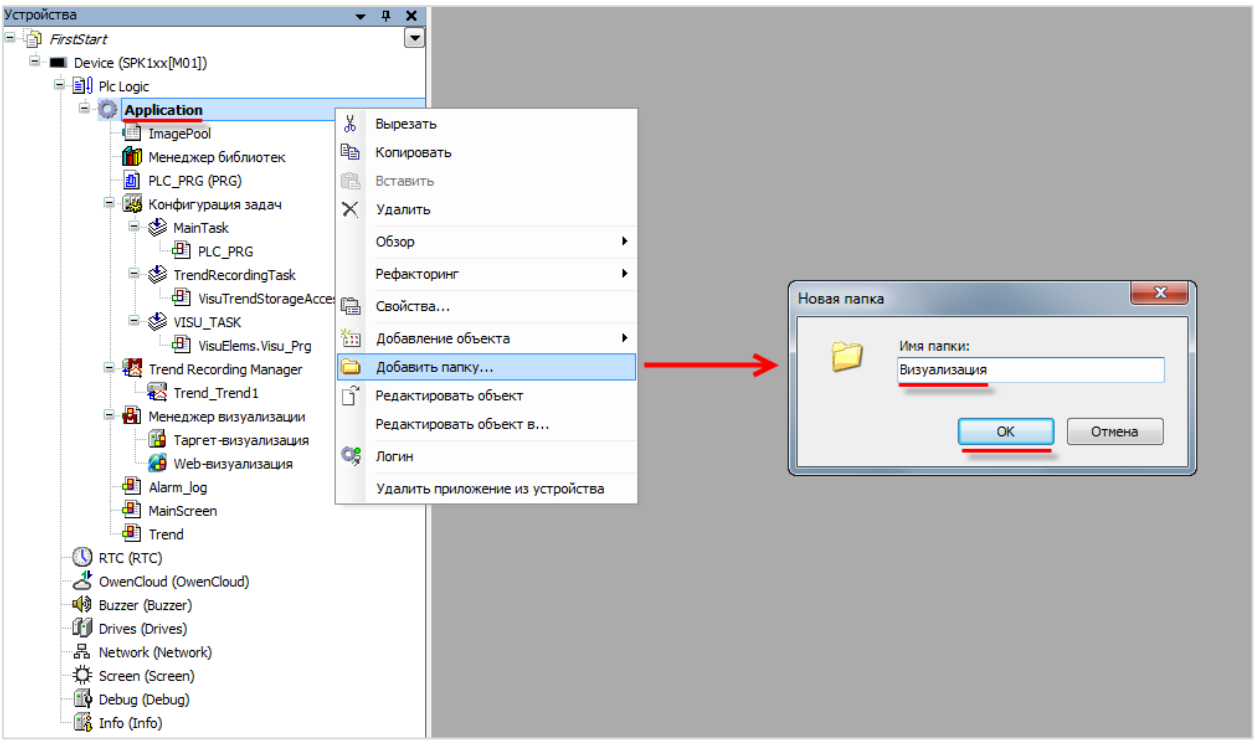


Рисунок 7.67 – Создание новой папки в приложении Application

Так должно выглядеть дерево проекта после создания папки:

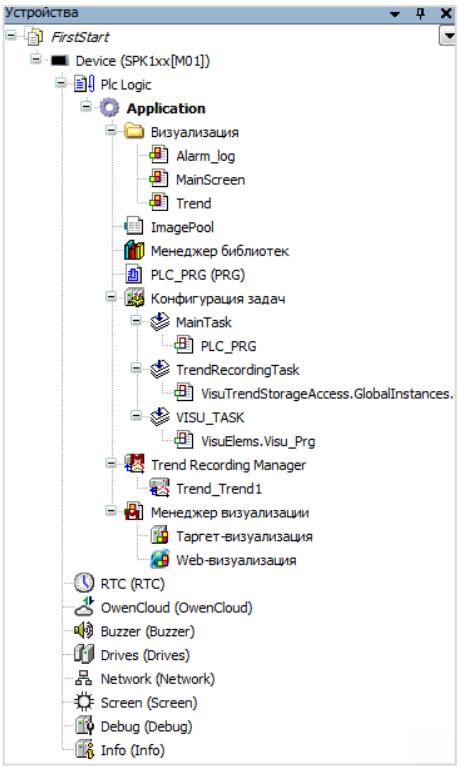


Рисунок 7.68 – Дерево проекта (после п. 7.3)

Экраны визуализации, созданные по указаниям пункта 7.3, должны выглядеть следующим образом:

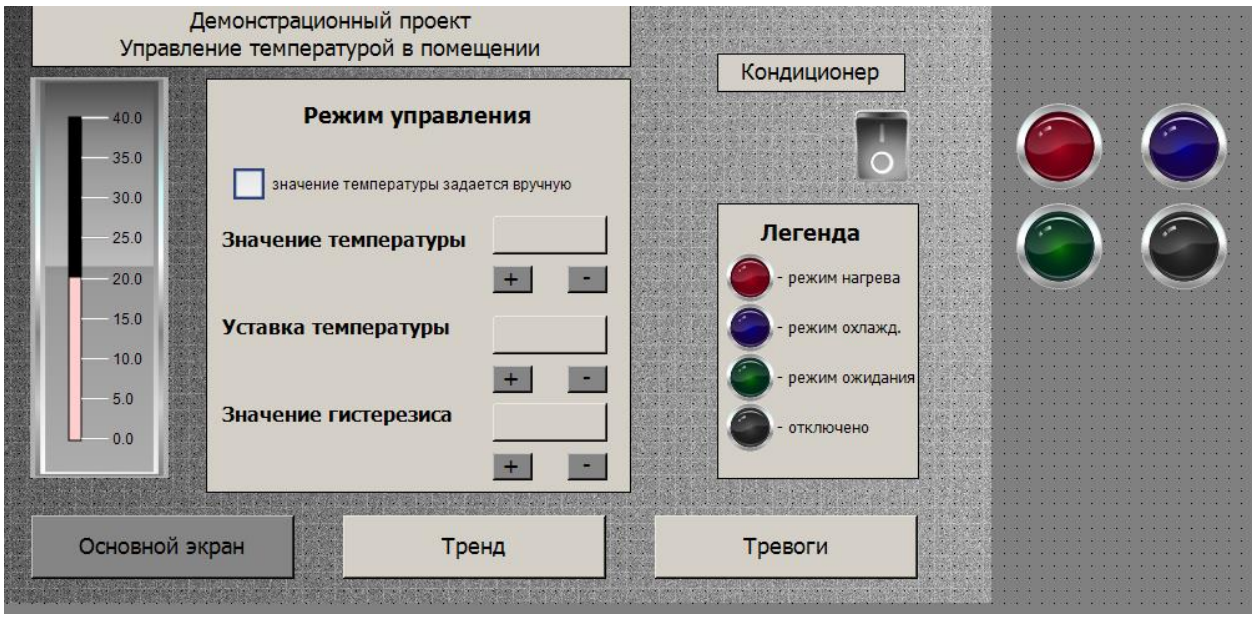


Рисунок 7.69 – Экран MainScreen (после п. 7.3)

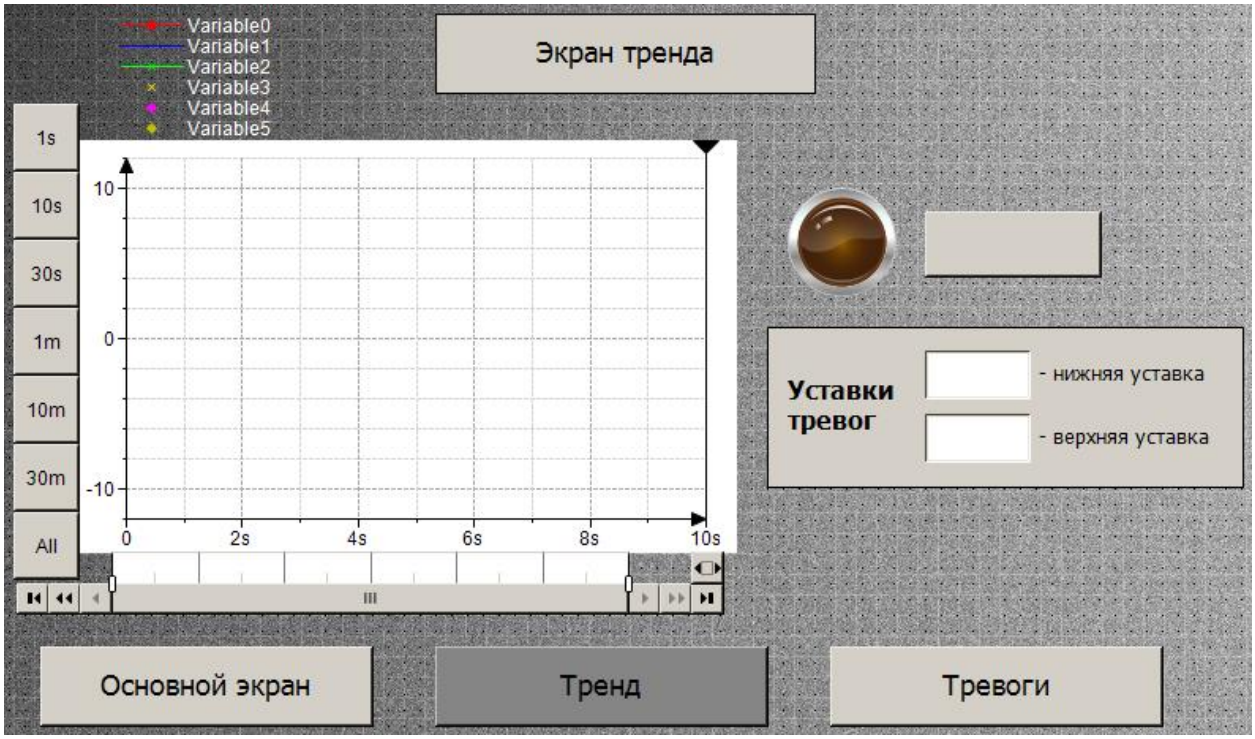


Рисунок 7.70 – Экран Trend (после п. 7.3)

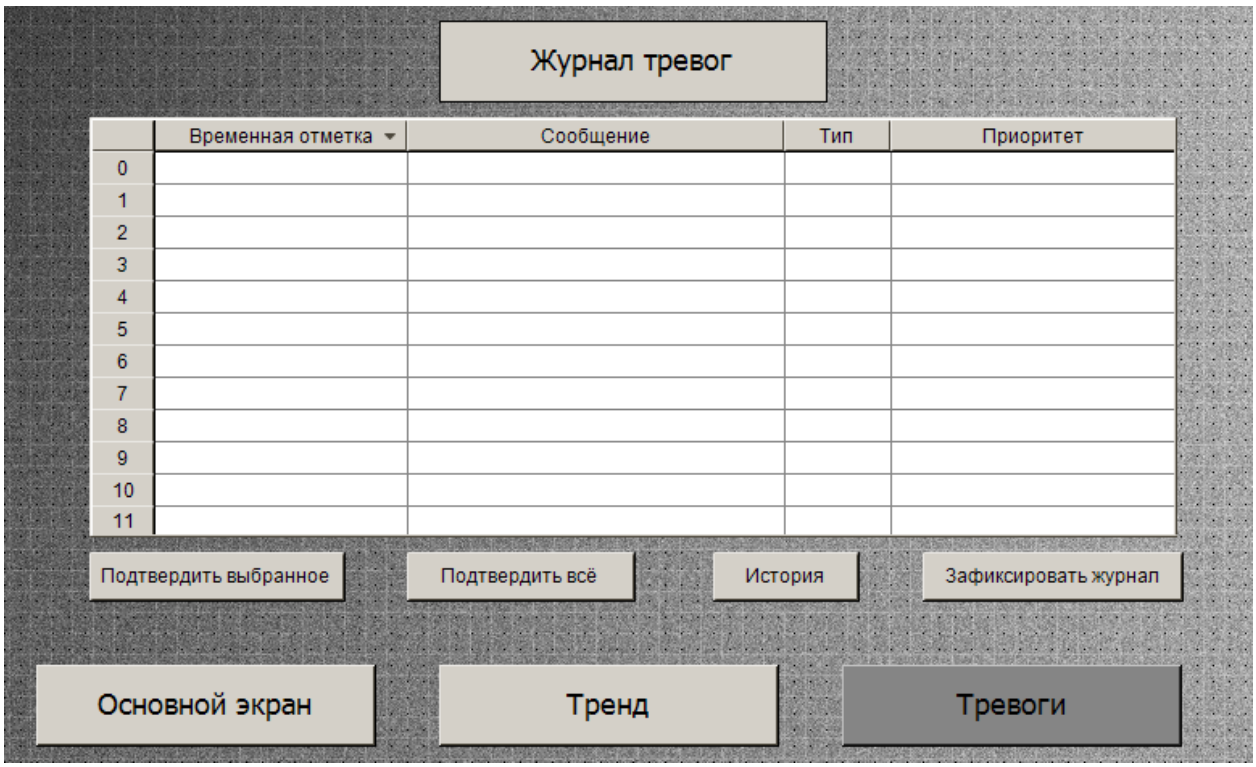


Рисунок – 7.71. Экран Alarm_log (после п. 7.3)

Функциональная настройка элементов и привязывание к ним переменных описываются в п. 7.5.

7.4 Разработка программ

7.4.1 POU и их типы. Языки программирования МЭК 61131-3. Структура приложения проекта FirstStart

Пользовательский проект может содержать одно или несколько приложений, которые состоят из компонентов, называемых **POU**. POU, которые используются для разработки **программной части проекта**, принадлежат к определенному **типу** и характеризуются **языком программирования**.

Существует **три типа** «программных» POU:

1. **Функция** – это POU, который не имеет внутренней памяти.
2. **Функциональный блок** – это POU, который имеет внутреннюю память. Вызов функционального блока осуществляется через его **экземпляр**.
3. **Программа** – это POU, который имеет внутреннюю память. Программы не имеют экземпляров и привязываются к **задачам**. Каждый проект должен содержать хотя бы одну программу, привязанную к задаче.

В данной главе рассматривается использование всех трех типов программных компонентов. Следует помнить, что рассматриваемая задача является учебным примером, поэтому некоторые используемые решения не являются оптимальными.

POU может быть написан на одном их шести **языков программирования** – пять из них входят в состав стандарта **МЭК 61131-3**, шестой является расширением языка FBD:

1. **IL (Instruction List**) – текстовый язык аппаратно-независимый низкоуровневый ассемблероподобный язык, в последнее время практически вышедший из употребления.
2. **LD (Ladder Diagram)** – графический язык релейно-контактных схем, подходящий для программирования релейной логики.
3. **SFC (Sequential Function Chart)** – графический высокоуровневый язык, созданный на базе математического аппарата сетей Петри. Описывает последовательность состояний и условий переходов.
4. **ST (Structured Text)** – текстовый паскалебодобный язык программирования.
5. **FBD (Functional Block Diagram)** – графический язык, программа на котором состоит из последовательно соединенных **функциональных блоков**.
6. **CFC (Continuous Function Chart)** – расширение языка FBD с возможностью определения порядка выполнения блоков и создания обратную связь между ними.

Язык программирования выбирается во время добавления в проект соответствующего программного компонента.

Написание программ на любом из языков происходит в соответствующих редакторах, каждый из которых имеет две области: область определения переменных (верхняя) и область программного кода (нижняя).

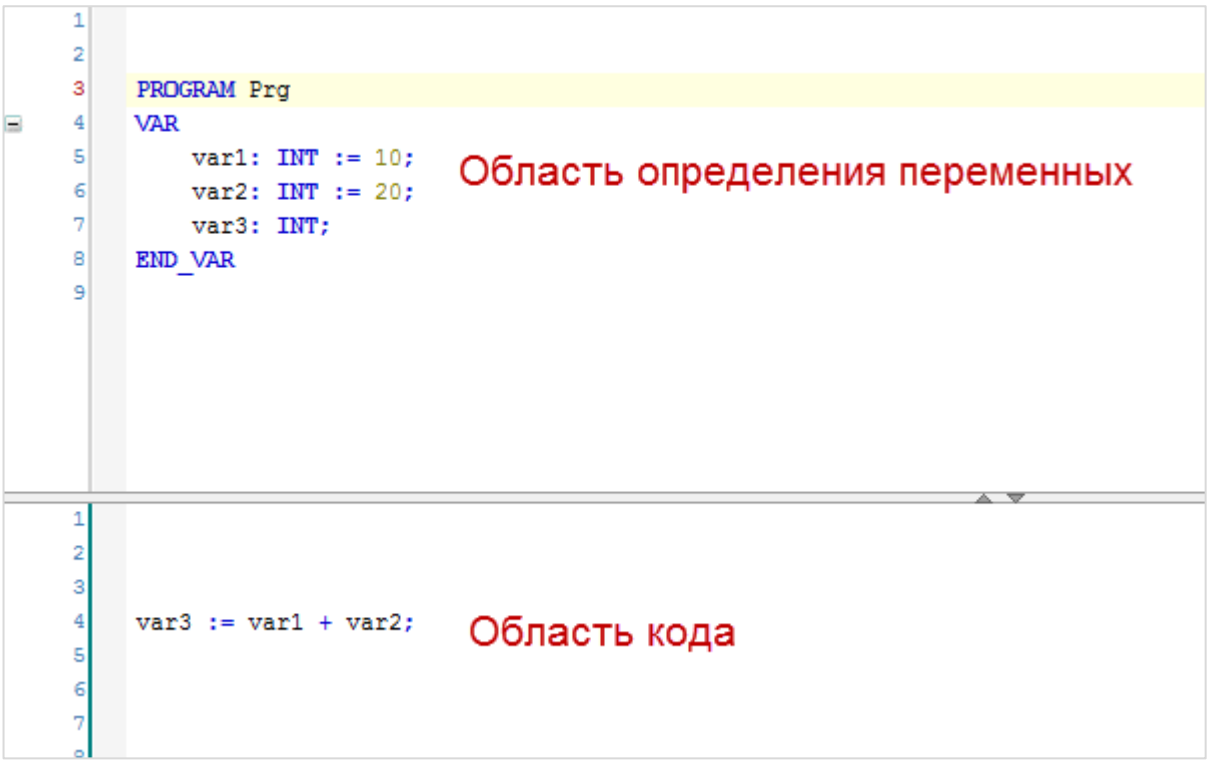


Рисунок 7.72 – Области редактора программирования

В рассматриваемом примере будут использоваться языки **ST** и **CFC** как наиболее распространенные и востребованные.

Структура программной части приложения:

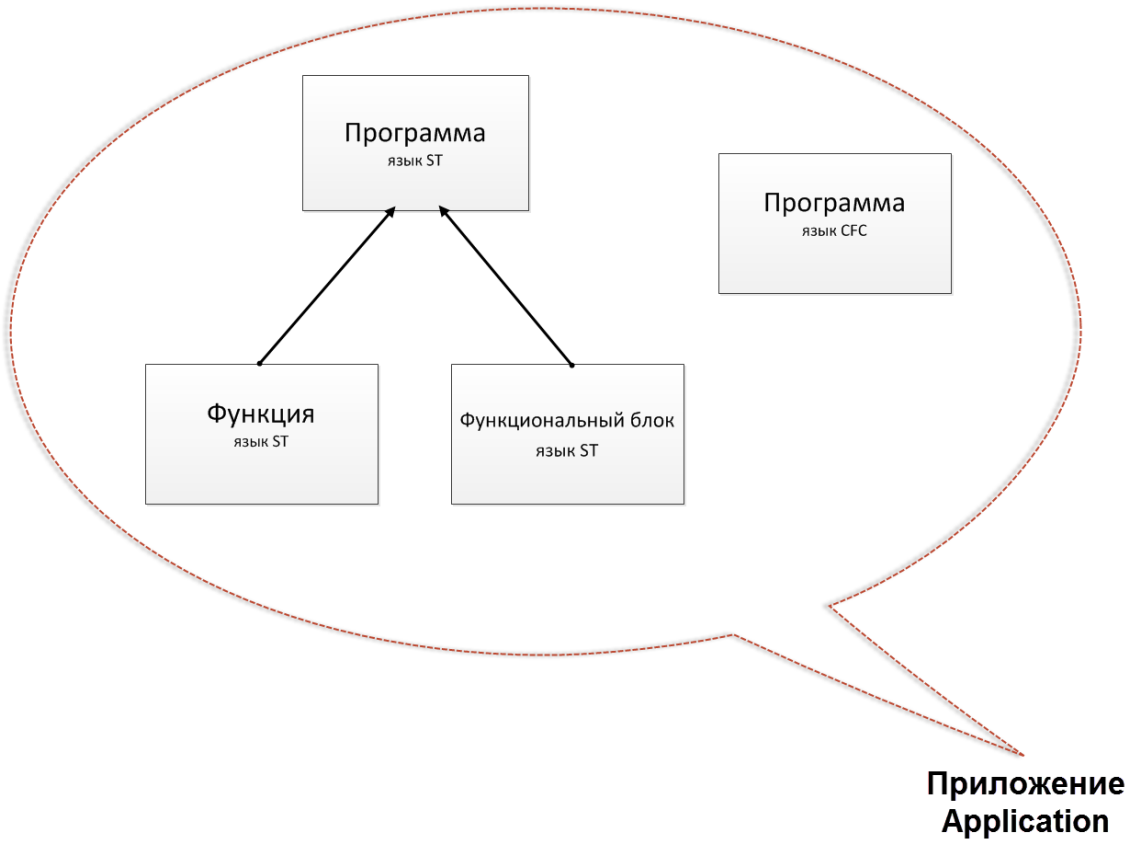


Рисунок 7.73 – Структура POU приложения Application

7.4.2 Виды переменных. Типы данных. Определение глобальных переменных

Переменные необходимы для ввода, хранения и вывода данных POU. Существует два типа переменных – **глобальные** (область видимости – весь проект) и **локальные** (область видимости – конкретный **POU**). Имена глобальных и локальных переменных могут совпадать, но локальная переменная имеет приоритет перед глобальной в **своем POU** (следует соблюдать осторожность в случае использования одинаковых имен переменных).

Переменная может обладать одним или несколькими атрибутами, из которых следует отметить **retain** – значения таких переменных после выключения контроллера сохраняются в **энергонезависимой памяти**.

Переменная обязательно принадлежит к какому-либо **типу данных**. Существует два вида типов данных: элементарные и составные.

Таблица 7.1 – Характеристики элементарных типов данных

Тип данных	Нижний предел	Верхний предел	Размер
Логические типы данных			
BOOL	FALSE	TRUE	8 бит
Целочисленные типы данных			
BYTE	0	255	8 бит
WORD	0	65535	16 бит
DWORD	0	4294967295	32 бита
LWORD	0	$2^{64}-1$	64 бита
INT	−32768	32767	16 бит
UINT	0	65535	16 бит
SINT	−128	127	8 бит
USINT	0	255	8 бит
DINT	−2147483648	2147483647	32 бит
UDINT	0	4294967295	32 бит
LINT	-2^{63}	$2^{63}-1$	64 бит
ULINT	0	$2^{64}-1$	64 бит
Типы данных с плавающей точкой			
REAL	$1.175494351e^{-38}$	$3.402823466e^{38}$	32 бит
LREAL	$2.225073858507201e^{-308}$	$1.7976931348623158e^{308}$	64 бит
Строковые типы данных			
STRING	Отсутствие символов	Нет (неограниченное количество символов), стандартные строковые функции могут работать со строками, длина которых не превышает 255 символов	Размер символа = 8 бит
WSTRING			Размер символа = 16 бит
Временные типы данных (подробнее о формате и ограничениях см. в справке CODESYS)			
TIME	0h0m0s0ms	49d17h2m37s295ms	32 бита
TIME_OF_DAY	0:0:0.0	23:59:59.999	32 бита
DATE	1970-1-1	2106-2-7	32 бита
DATE_AND_TIME	1970-1-1-0:0:0.0	2106-2-7-6:28:15	32 бита
LTIME	0h0m0s0ms0us0ns	213503d23h34m33s709ms551us615ns	64 бит

Для определения **глобальных переменных** создается соответствующий компонент – **Список глобальных переменных** с названием **GlobalVars**:

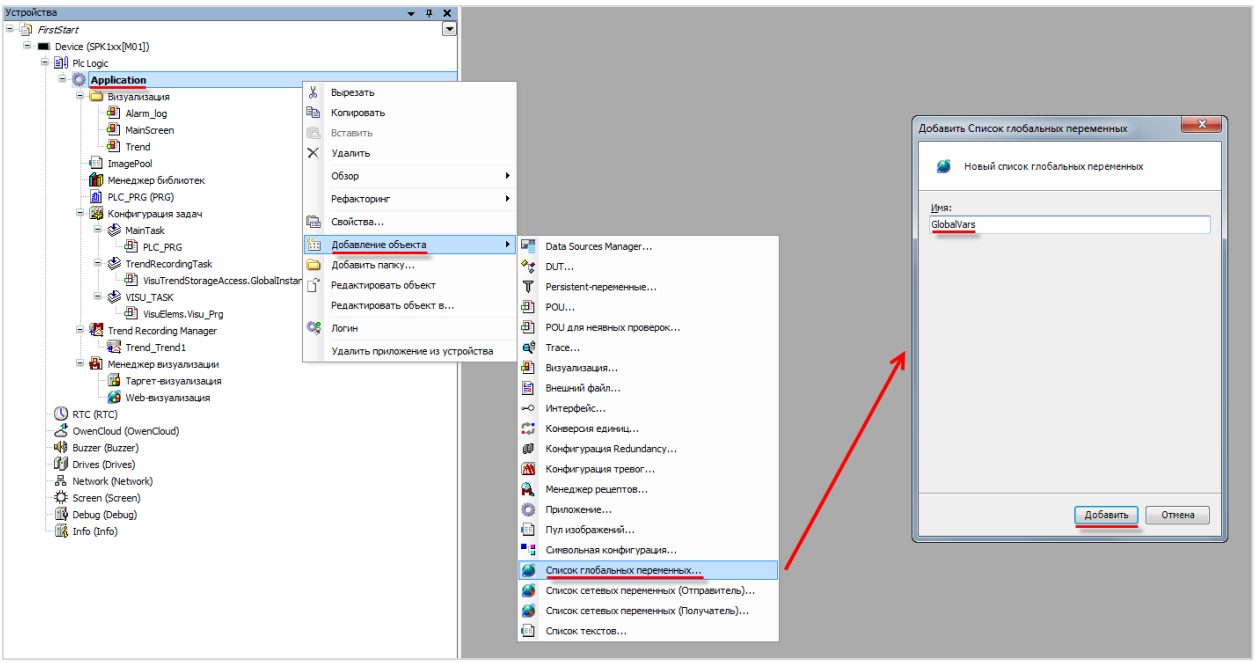


Рисунок 7.74 – Создание списка глобальных переменных

Определение глобальных переменных осуществляется между ключевыми словами **VAR_GLOBAL** и **END_VAR**, глобальных **retain** (энергонезависимых) переменных – между **VAR_GLOBAL RETAIN** и **END_VAR** соответственно (см. рисунок 7.75).

Синтаксис определения переменной следующий (угловые скобки использованы для наглядности, в среде программирования их не следует использовать):

<Имя переменной>: <Тип> {:=<начальное значение>};

где **<Имя переменной>** – индивидуальное название переменной, на которое накладываются определенные ограничения, приведенные в справке CODESYS. Регистр имени переменной не учитывается, имя не должно содержать пробелов, специальных и кириллических символов, содержать более одного подчеркивания подряд, совпадать с ключевыми словами (например, TIME). Локально не могут быть объявлены переменные с совпадающими именами;

<Тип> – тип переменной. Характеристики стандартных типов приведены в таблице 7.1;

<начальное значение> – начальное значение переменной (по умолчанию равно нулю);

[:=] – оператор присваивания;

[;] – символ, которым завершается любая процедура определения, присваивания, вызова функции и т. п.

Синтаксис определения **строковых переменных** несколько отличается от определения переменных других типов: после названия типа в скобках указывается **максимальная длина строки**, которая определяет количество резервируемой памяти (без указания – 80 символов). Содержимое строки для типа **STRING** (используемого для латиницы) указывается в **одиночных кавычках** (‘<содержимое строки>’), для типа **WSTRING** (кириллица) – в **двойных** (“<содержимое строки>”).

Список глобальных переменных, которые будут использоваться в рамках примера:

Device | GlobalVars | Менеджер библиотек | Конфигурация задач

```
VAR_GLOBAL
    temp_real:          REAL; (*температура помещения, полученная с датчика*)
    temp_user:          REAL; (*температура помещения, введенная пользователем*)

    temp:               REAL; (*текущая температура в помещении для использования внутри цикла*)

    temp_hyst_high:     REAL; (*температура верхней границы гистерезиса*)
    temp_hyst_low:      REAL; (*температура нижней границы гистерезиса*)

    measure_mode:       BOOL :=TRUE; (*режим измерения:          0 - с датчика,  1 - ввод пользователем*)
    control_mode:       BOOL;        (*режим управления:         0 - охлаждение, 1 - нагрев*)
    regulator_state:    BOOL;        (*состояние регулятора:     0 - выключен,   1 - включен*)
    conditioner_power:  BOOL;        (*состояние кондиционера:   0 - выключен,   1 - включен*)


    yellow_lamp_name:   WSTRING(20) :="Введите название"; (*название сигнальной лампы с экрана Тренда*)
END_VAR


VAR_GLOBAL RETAIN
    temp_ust:           REAL :=20; //уставка температуры

    hyst:               REAL :=5;  //значение гистерезиса в процентах

    alarm_high:         REAL;      //верхняя граница сигнала тревоги
    alarm_low:          REAL;      //нижняя граница сигнала тревоги
END_VAR
```

Рисунок 7.75 – Объявление глобальных переменных

Локальные переменные будут описаны в процессе разработки программ.

Для написания **комментариев** используются конструкции типа (*<комментарий>*) и //<комментарий>. Комментарий с конструкцией первого типа может занимать несколько строк, второго типа – только одну строку.

7.4.3 Программа на языке CFC (мигание лампы). Подключение дополнительных библиотек

Создание **мигающего индикатора** (в общем случае – логической переменной, состояния которой меняются с определенной частотой) – довольно распространенная задача при разработке автоматических систем. В данном примере такая программа будет использоваться для создания аварийной лампы, мигающей в случае выхода значения температуры за **аварийную уставку**.

В проекте уже имеется пустой **POU** типа **программа** на языке **CFC** с названием **PLC_PRG**. С помощью функции **Refactoring** следует изменить его название на **BlinkLamp**:

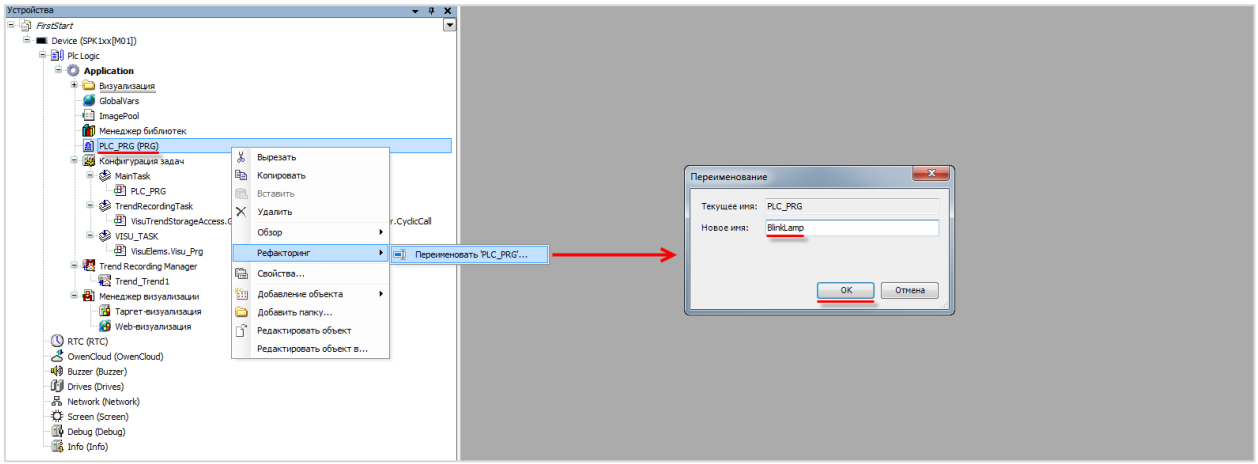


Рисунок 7.76 – Изменение название программы (refactoring)

Появится окно, в котором указаны компоненты, на которые повлияет смена названия:

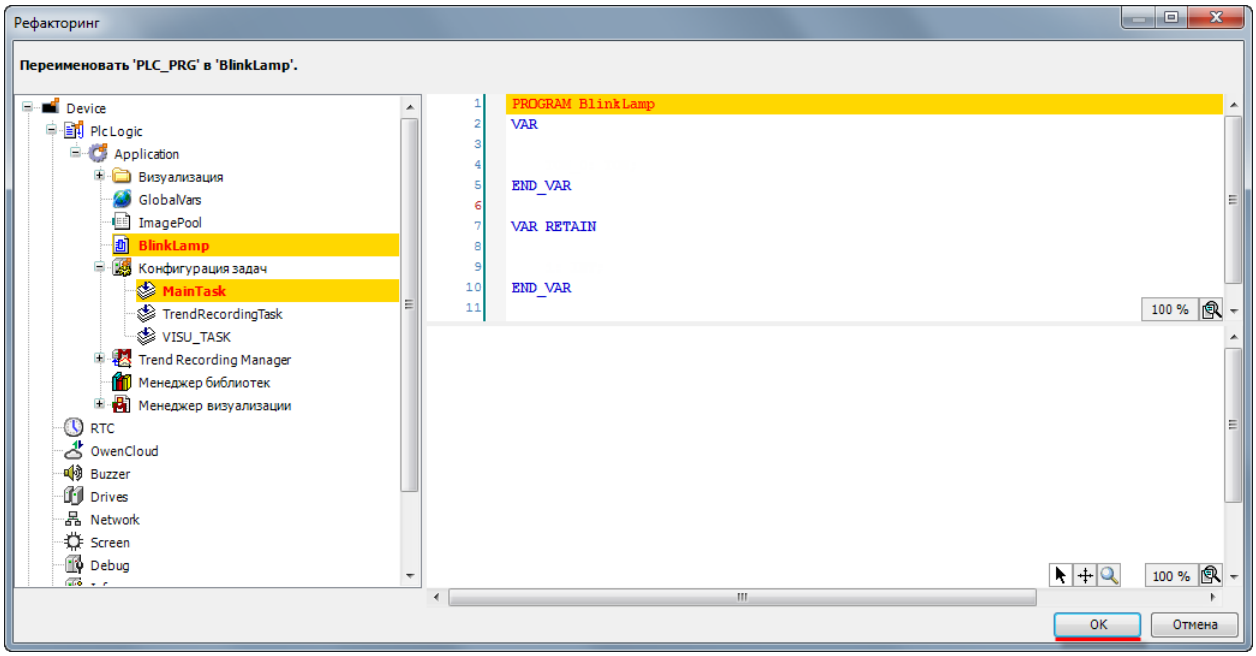


Рисунок 7.77 – Диалоговое окно изменения названия

Для разработки программы требуется функциональный блок **Blink** из библиотеки **Util**, которая по умолчанию **не включена** в проект. Чтобы добавить ее, следует выбрать компонент **Менеджер библиотек** и нажать кнопку **Добавить библиотеку** (предварительно библиотека должна быть установлена, см. п. 2):

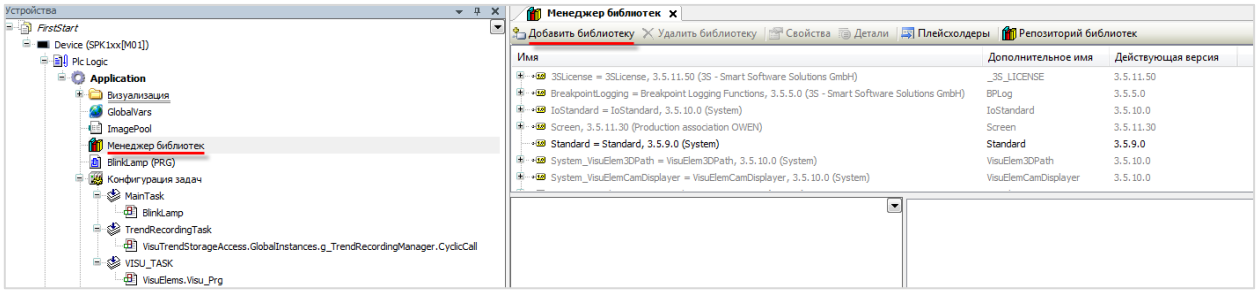


Рисунок 7.78 – Добавление библиотеки в проект

В открывшемся окне следует выбрать нужную библиотеку и нажать **ОК**.

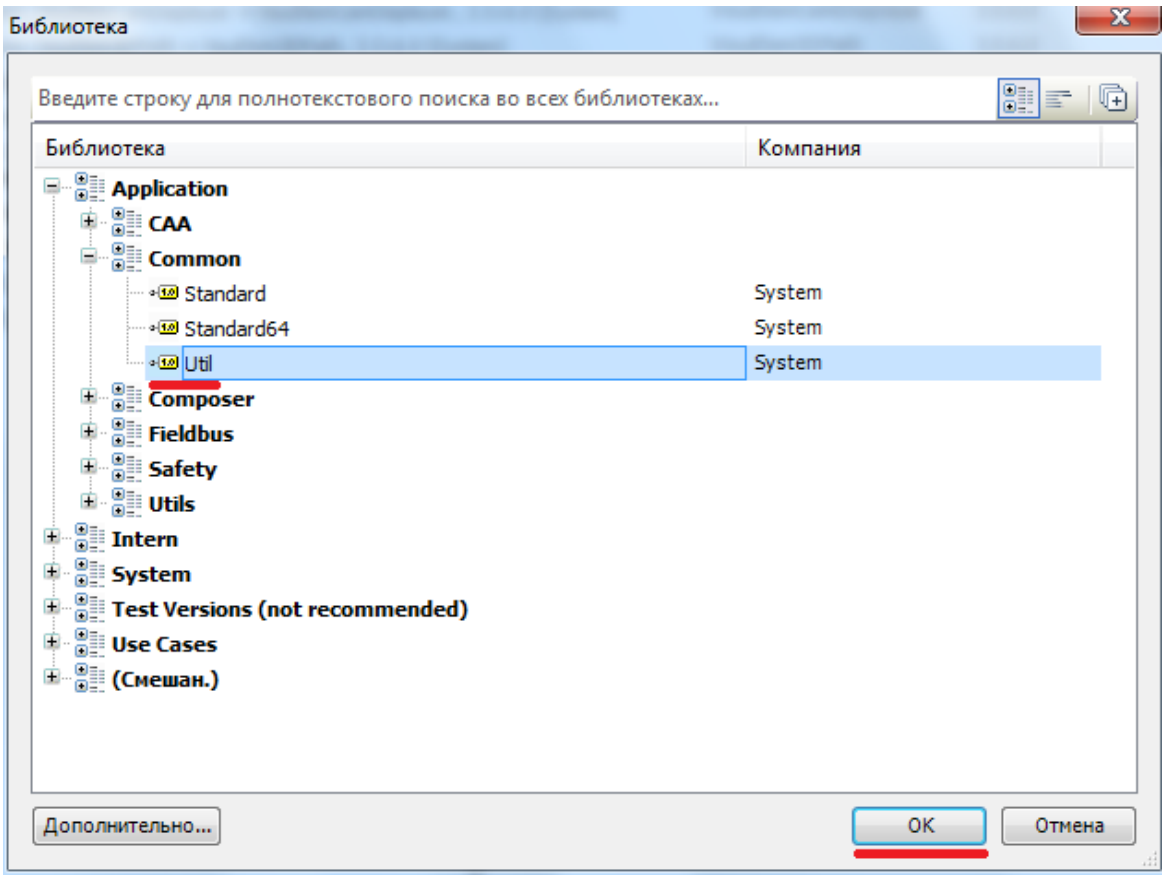


Рисунок 7.79 – Выбор библиотеки

Библиотека **Util** появится в списке используемых библиотек:

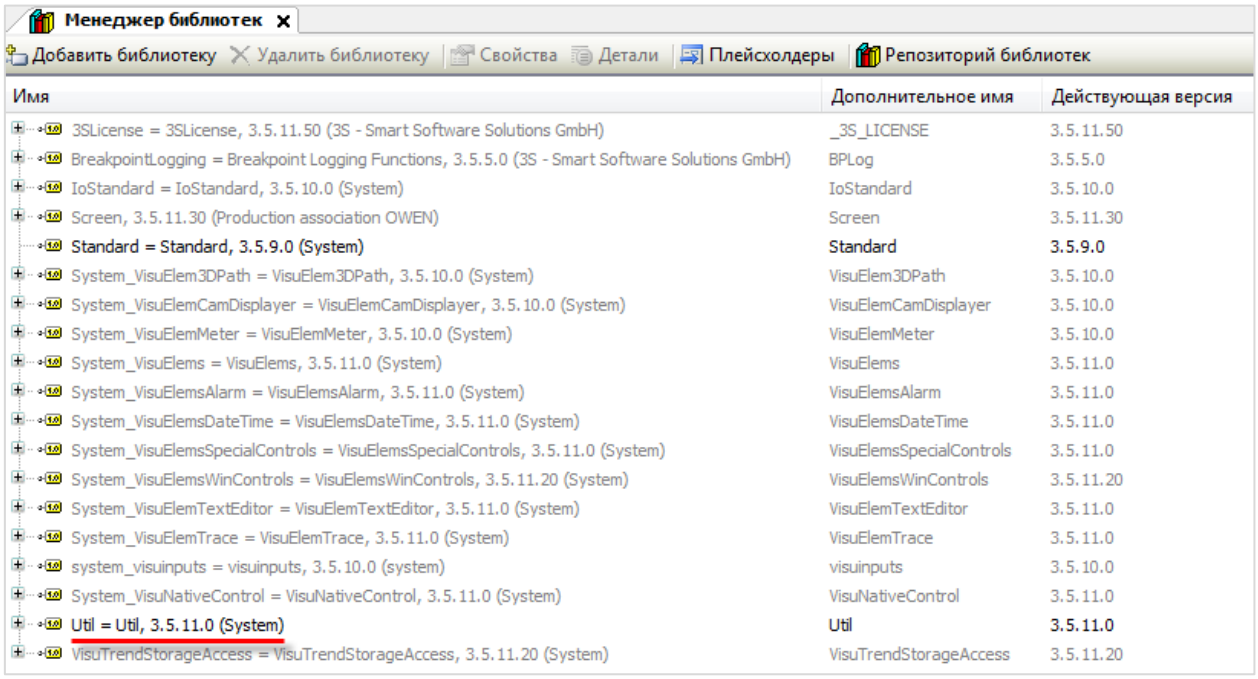


Рисунок 7.80 – Библиотека Util

После открытия **BlinkLamp** однократным нажатием **ЛКМ** следует выделить на **Панели инструментов** редактора элемент **«Элемент»** и однократным нажатием **ЛКМ** разместить его **в рабочей области**.

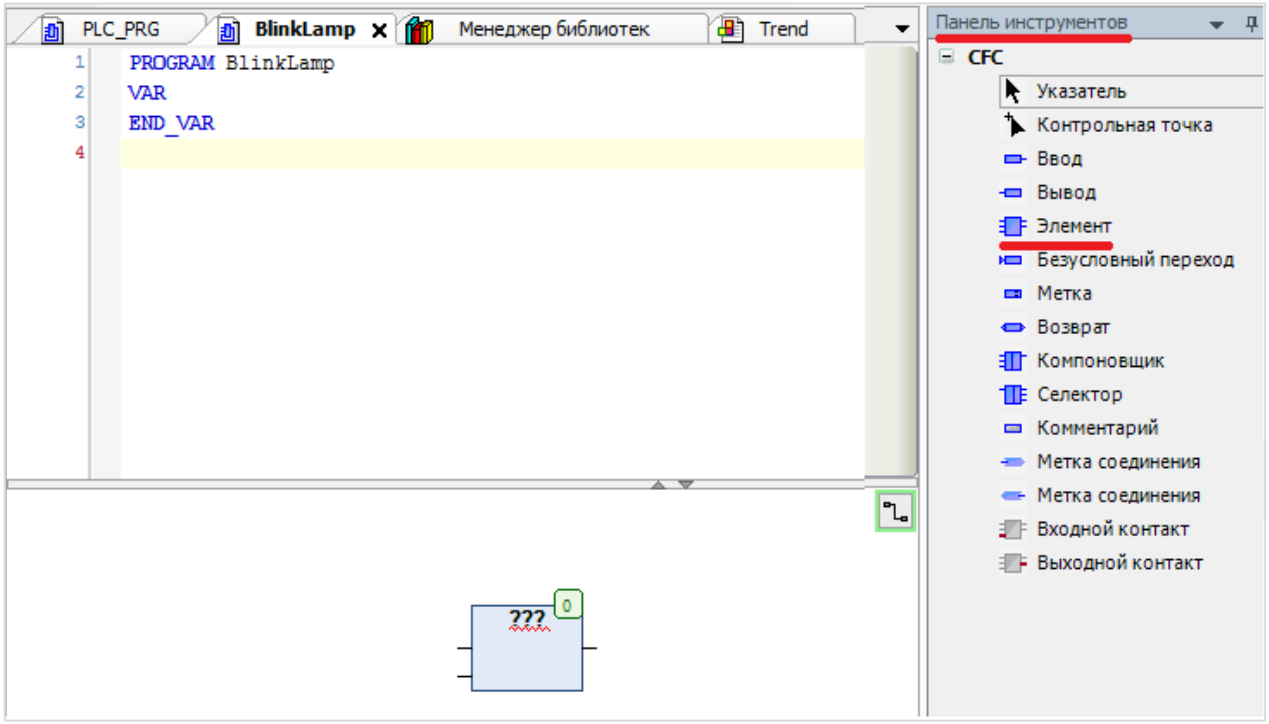


Рисунок 7.81 – Добавление элемента в редакторе CFC

Вместо «**???**» следует ввести тип функционального блока – **BLINK**. Также можно выбрать функциональный блок из списка доступных с помощью нажатия на контекстную кнопку «**…**», которая откроет **Ассистент ввода**.

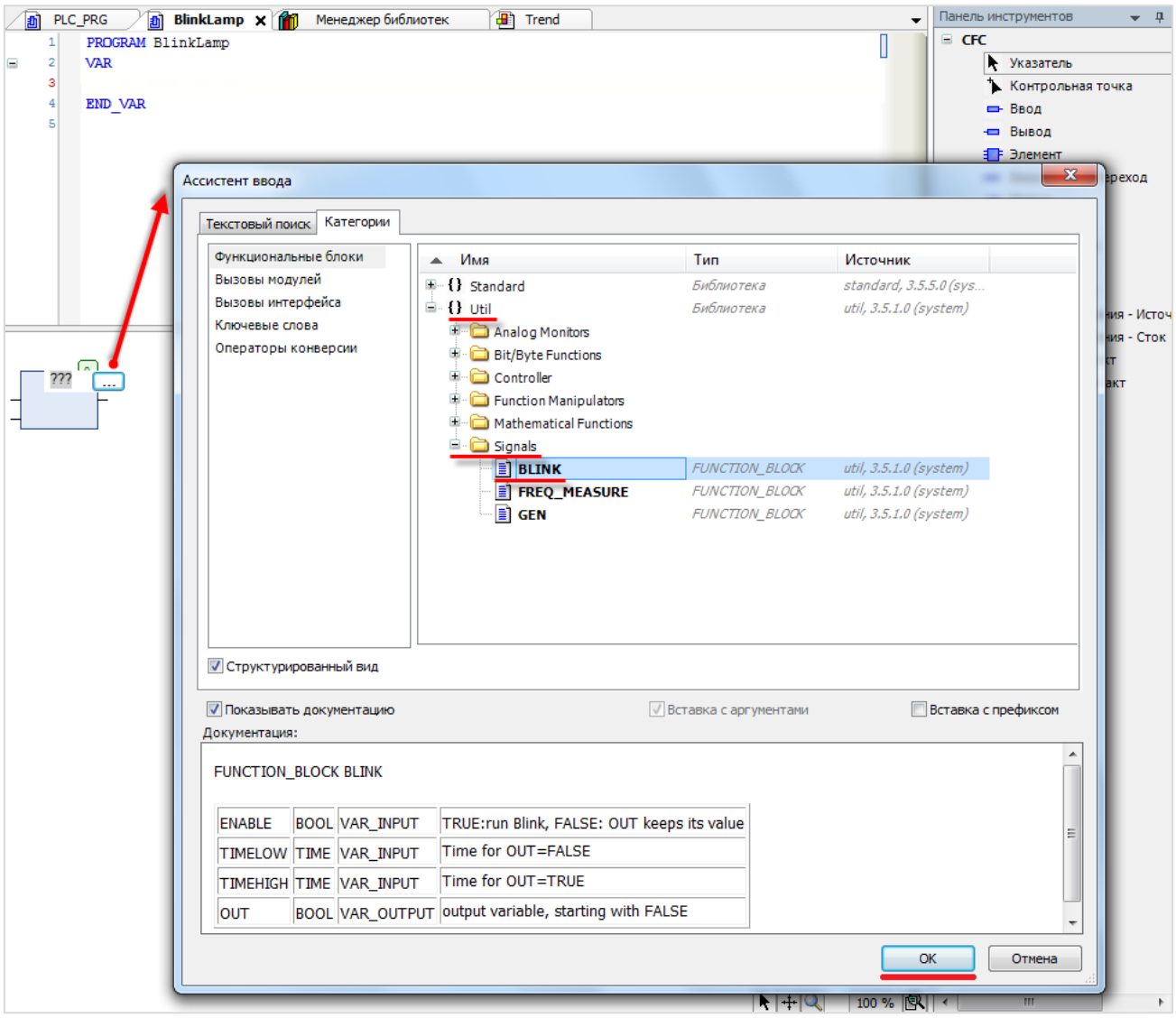


Рисунок 7.82 – Выбор функционального блока

Затем следует поменять название экземпляра функционального блока на **BLINKER** и нажать **ОК**, что приведет к появлению **Ассистента автообъявления** (т. к. экземпляр функционального блока еще не объявлен).

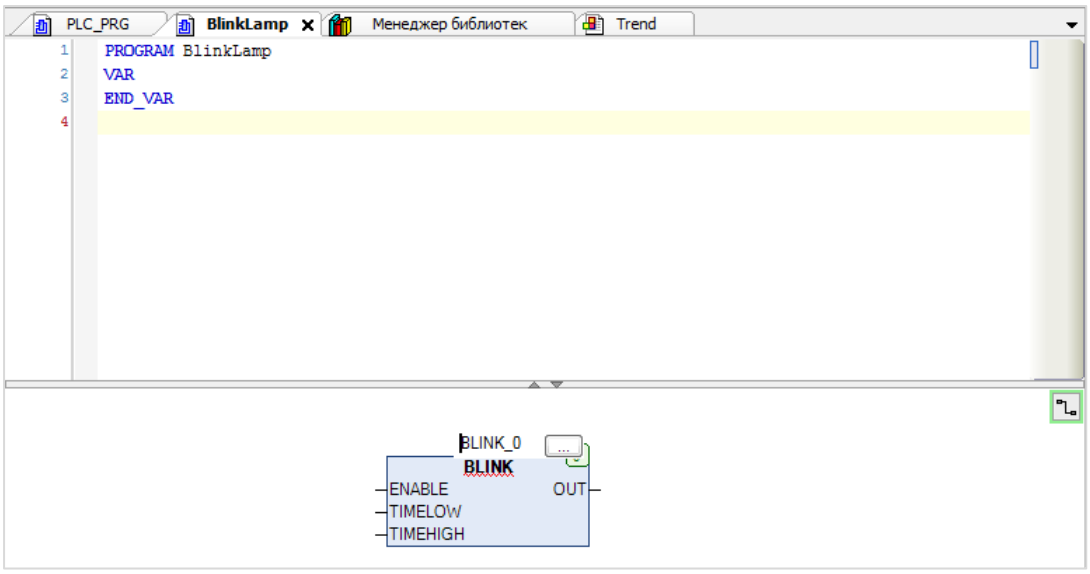


Рисунок 7.83 – Изменение названия экземпляра функционального блока

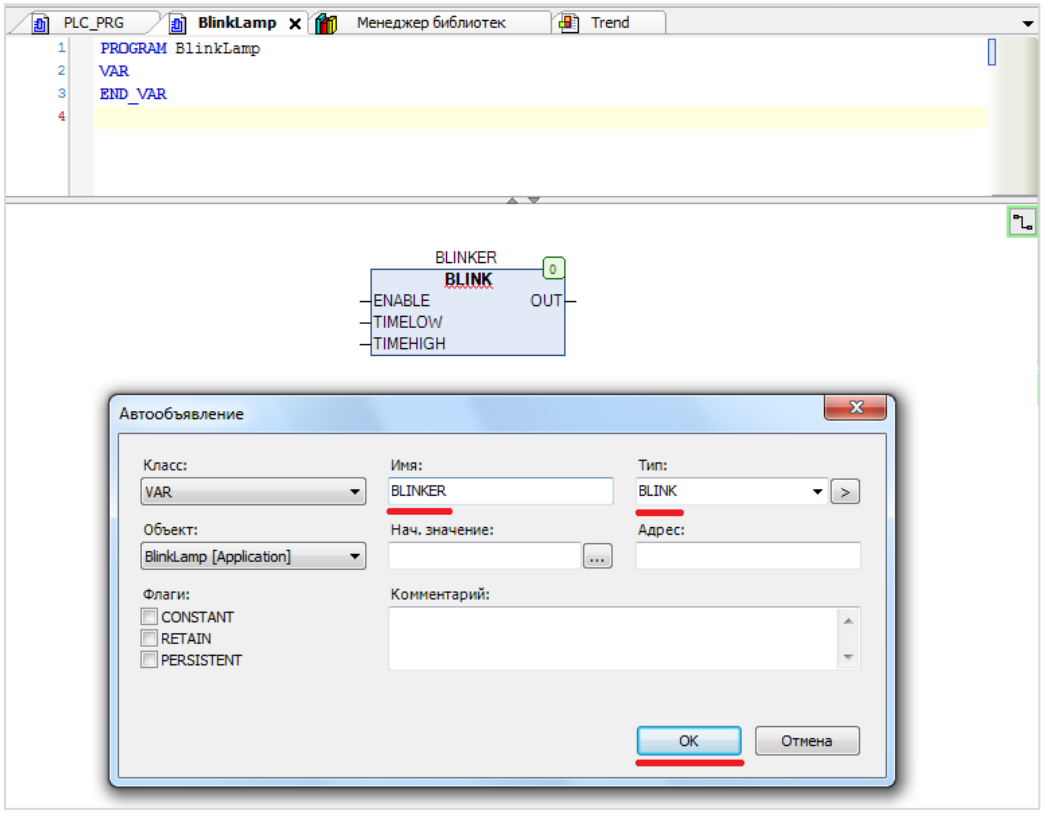


Рисунок 7.84 – Ассистент автообъявления

После нажатия на **ОК** в **области определения переменных** автоматически создастся экземпляр **BLINKER** функционального блока типа **BLINK**.

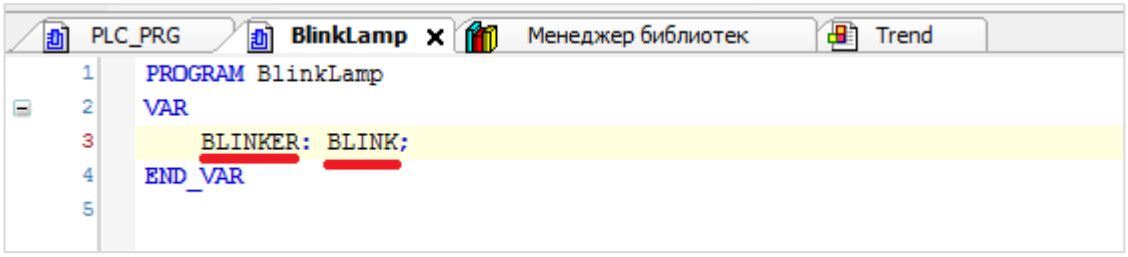


Рисунок 7.85 – Определение экземпляра функционального блока как переменной

Далее следует добавить локальную переменную **yellow_lamp** типа **BOOL**, которая будет характеризовать состояние желтой аварийной лампы на экране тренда: TRUE – лампа горит, FALSE – лампа не горит. Быстрая смена состояний лампы будет визуально соответствовать ее миганию. Остальные переменные, которые будут использоваться, уже определены в **Списке глобальных переменных** (см. п. 7.4.2).

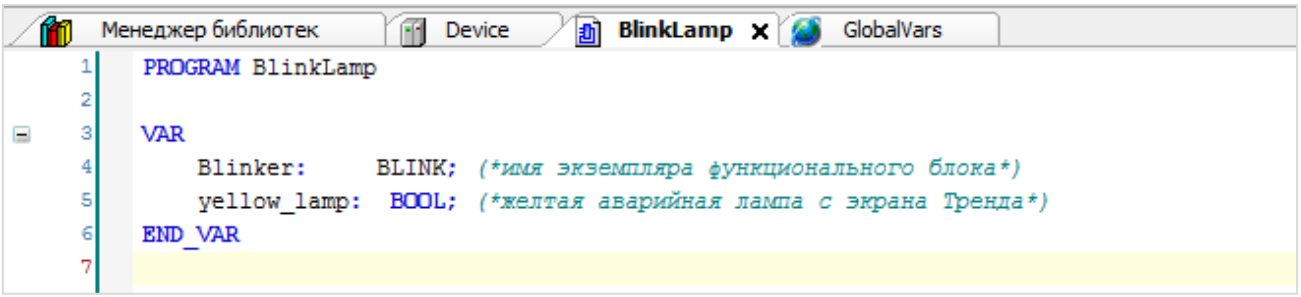


Рисунок 7.86 – Определение переменных программы BlinkLamp

Функциональный блок BLINK обладает тремя **входами** и одним **выходом**.

Таблица 7.2 – Характеристики переменных блока BLINK

Название	Тип	Назначение
Входы		
ENABLE	BOOL	**TRUE** – запустить блок, **FALSE** – остановить блок (на выходе остается текущее значение)
TIMELOW	TIME	Время, которое на выходе блока сохраняется **FALSE**
TIMEHIGH	TIME	Время, которое на выходе блока остается **TRUE**
Выходы		
OUT	BOOL	Выход блока (по умолчанию имеет значение **FALSE**)

Функциональный блок **BLINK** работает следующим образом:

1. При значении ENABLE = TRUE:
   1.1. OUT = FALSE на время TIMELOW;
   1.2. OUT = TRUE на время TIMEHIGH;
   далее последовательно выполняются пункты 1.1. и 1.2.
2. При значении ENABLE = FALSE:
   OUT сохраняет последнее принятое им значение.

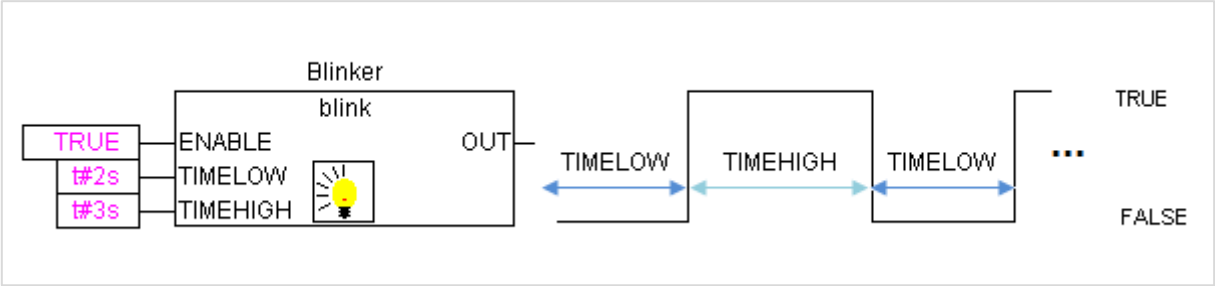


Рисунок 7.87 – Графическая интерпретация работы BLINK

Готовая программа для мигания лампы в случае выхода температуры за значения аварийных уставок будет выглядеть следующим образом:

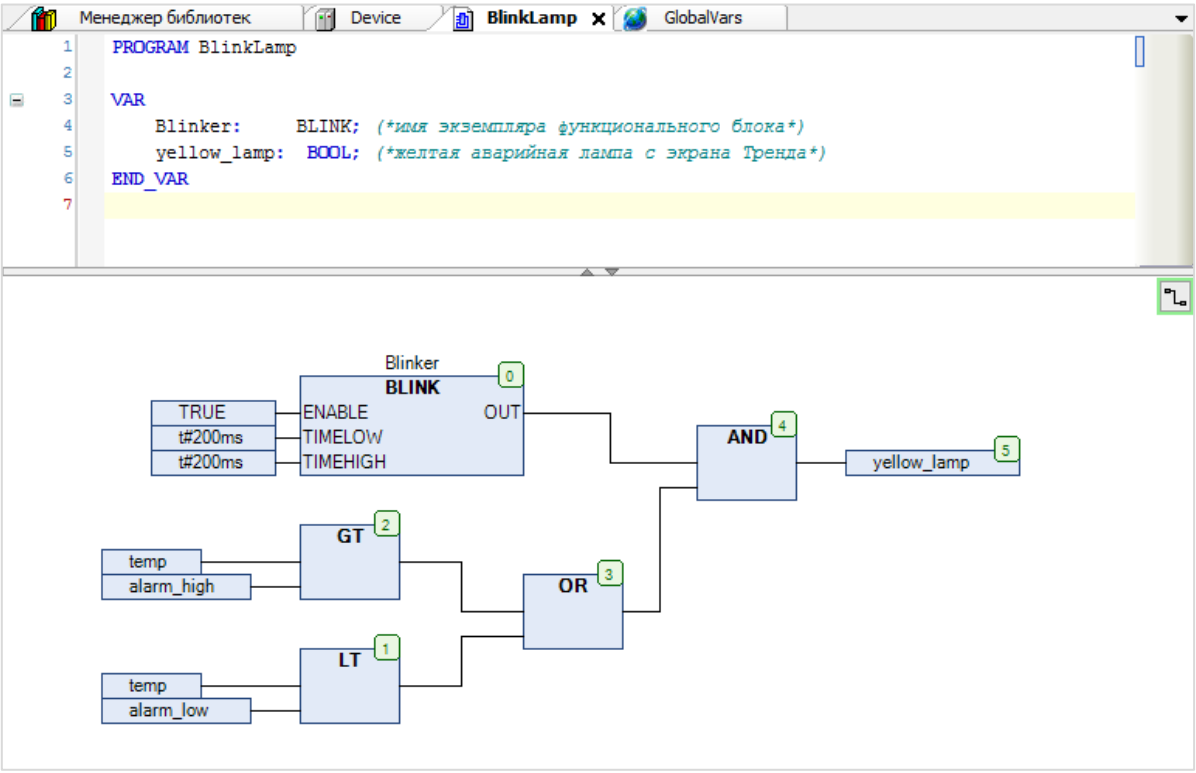


Рисунок 7.88 – Программа BlinkLamp

Если температура будет находиться в пределах уставок, лампа останется потухшей.

Блоки **GT**, **LT**, **AND** и **OR** создаются по аналогии с блоком **BLINK**: сначала на **Панели инструментов редактора** нажатием **ЛКМ** выделяется элемент «**Элемент**», затем одиночным нажатием **ЛКМ** размещается в рабочей области, после чего «заполняется» соответствующим функциональным блоком с помощью ввода его названия вручную или же выбора через **Ассистент** (вкладка **Ключевые слова**):

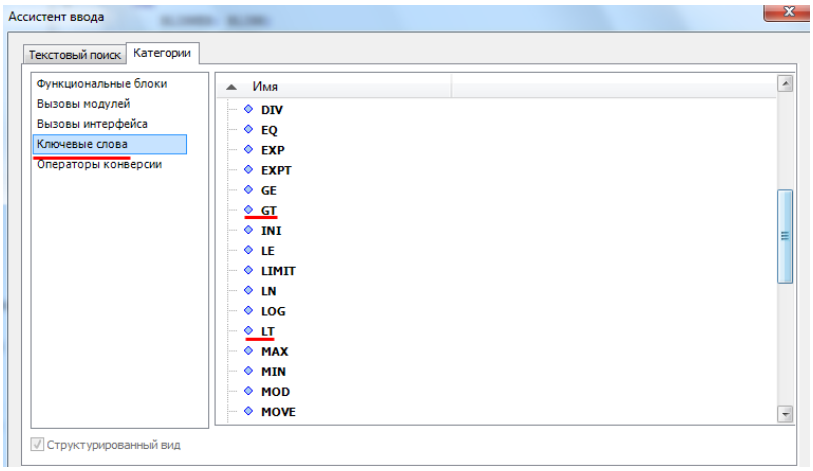


Рисунок 7.89 – Выбор ключевых слов

Ключевые слова не имеют экземпляров.

GT и **LT** – это операторы сравнения, выходы которых принимают значение TRUE, если значение первого (верхнего) входа больше (**GT**) или меньше (**LT**) значения второго входа. **AND** и **OR** – операторы логического И и ИЛИ соответственно.

Чтобы присвоить входу блока значение, следует выделить его однократным нажатием ЛКМ и начать вводить имя переменной (или константы), что приведет к появлению соответствующей строки с кнопкой **Ассистента ввода**:

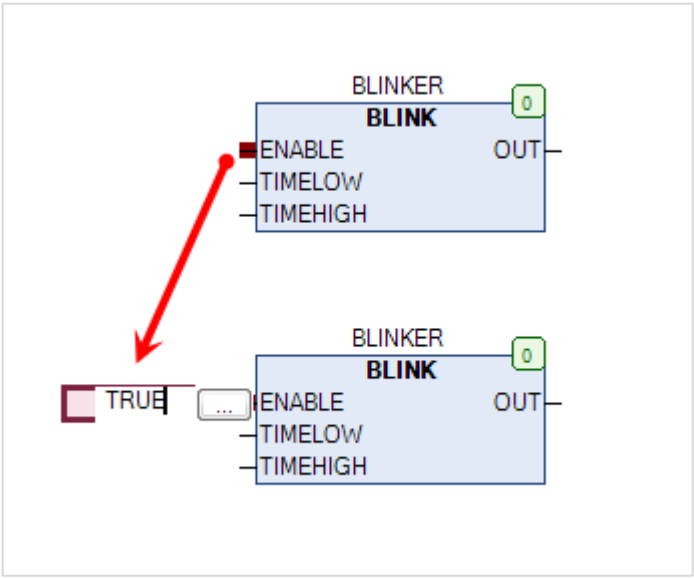


Рисунок 7.90 – Ввод значений входных переменных функционального блока

Значения входов блока BLINK **TIMELOW** и **TIMEHIGH** для корректной работы должны быть согласованы с **временем цикла** программы (либо равны, либо кратны ему, и не должны быть меньше его значения), которое определяется соответствующей **задачей** (см. п. 7.7).

Используемые в программе переменные **Temp, alarm_low** и **alarm_high** определены в **Списке глобальных переменных** (см. п. 7.4.2).

Чтобы **связать** блоки между собой следует выделить нажатием **ЛКМ** вход/выход первого блока и ***не отпуская*** кнопки мыши перетащить курсор на выход/вход второго.

Значение выхода блока AND присваивается переменной **yellow_lamp** с помощью элемента «**Вывод**».

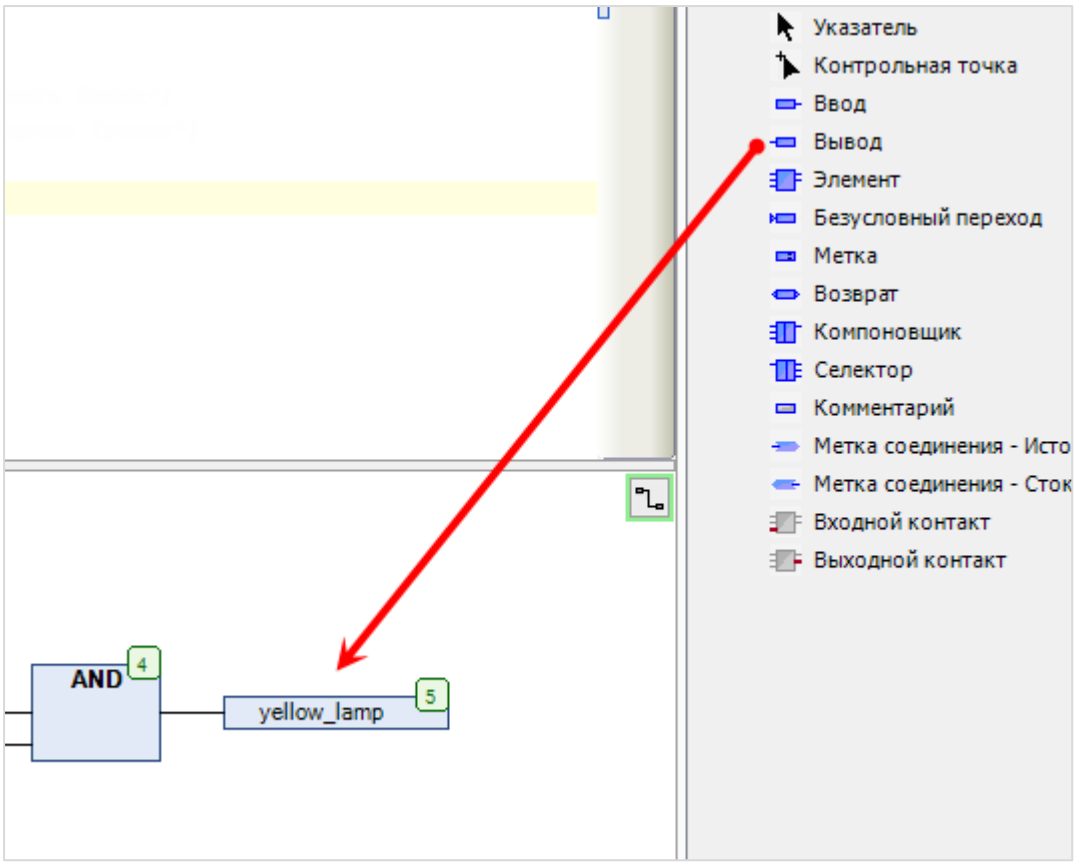


Рисунок 7.91 – Добавление элемента Вывод в редакторе CFC

Программа с рисунка 7.88 работает следующим образом: значение на выходе блока **BLINK** (TRUE или FALSE) раз в 200 мс меняется на противоположное. **Выход блока OR** принимает значение TRUE в том случае, если значение температуры выходит за верхнюю или нижнюю аварийную уставку. Если температура находится в допустимых пределах, **выход блока OR** принимает значение FALSE. Соответственно, значение **выхода блока AND**, которое присваивается переменной **yellow_lamp**, однозначно определяется **выходом блока OR**:

1. Если выход OR = TRUE, тогда **yellow_lamp** раз в 200 мс меняет свое значение на противоположное (с FALSE на TRUE или с TRUE на FALSE). Визуально это соответствует **миганию** лампы.
2. Если выход OR = FALSE, тогда **yellow_lamp** = FALSE. Визуально это соответствует **неактивной** (потухшей) лампе.

7.4.4 Функция на языке ST (расчет границ гистерезиса)

Функция – это **POU**, который не имеет внутренней памяти. Назначение функций:

1. Избавление от необходимости многократно повторять в тексте программы аналогичные фрагменты.
2. Улучшение структуры программы и облегчение понимания ее работы;
3. Повышения устойчивости к ошибкам программирования и непредвиденным последствиям в случае модификации программы.

В рамках примера функция будет использоваться для пересчета значений границ **гистерезиса** из процентов (относительно уставки температуры) в градусы Цельсия.

Сначала следует создать **POU** типа **функция** на языке **ST** с названием **temp_hyst**, который возвращает значение типа **Real**:

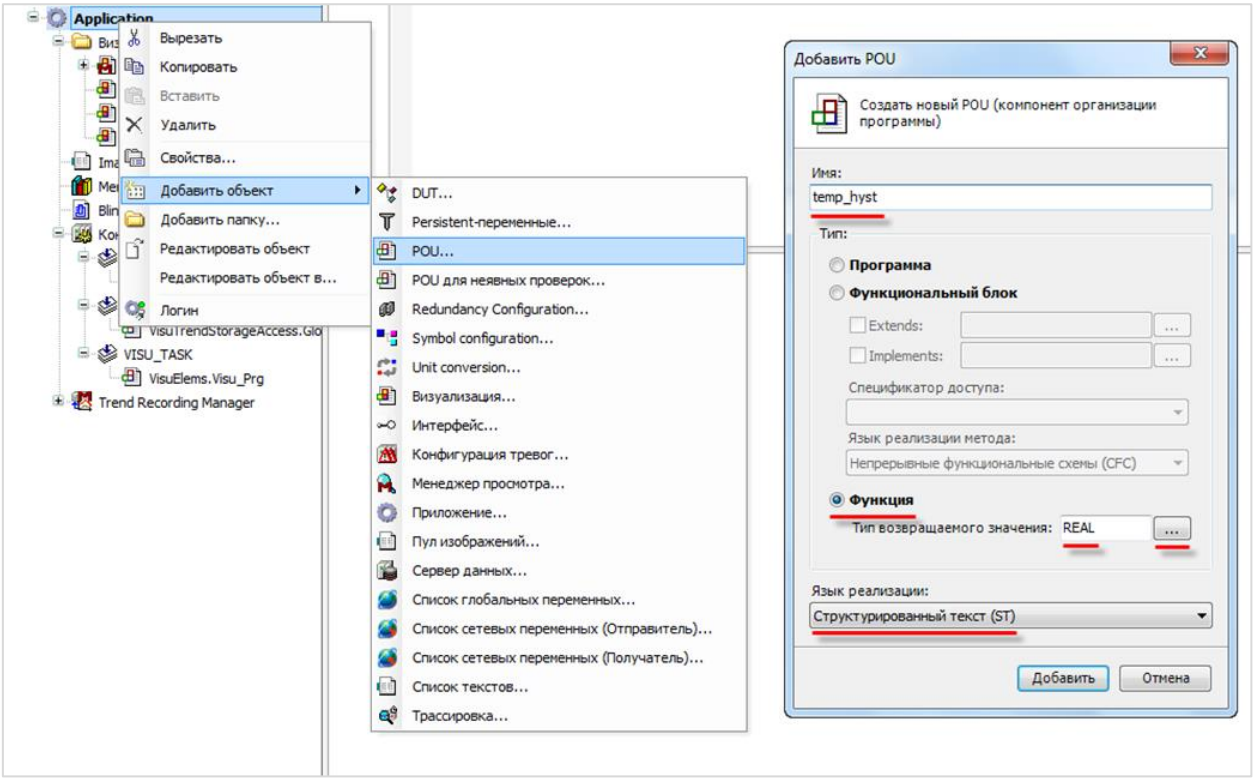


Рисунок 7.92 – Создание функции

Пользователь задает уставку температуры **temp_ust** в градусах Цельсия и значение гистерезиса в процентах от этой уставки. Данное значение будет пересчитано в верхнюю и нижнюю температурные границы гистерезиса, которые соответственно определяются формулой:

$$t_{\text{гист}} = t_{\text{уст}} \pm \frac{hyst}{100} \cdot t_{\text{уст}}$$

Определение переменных и код функции будет выглядеть следующим образом:

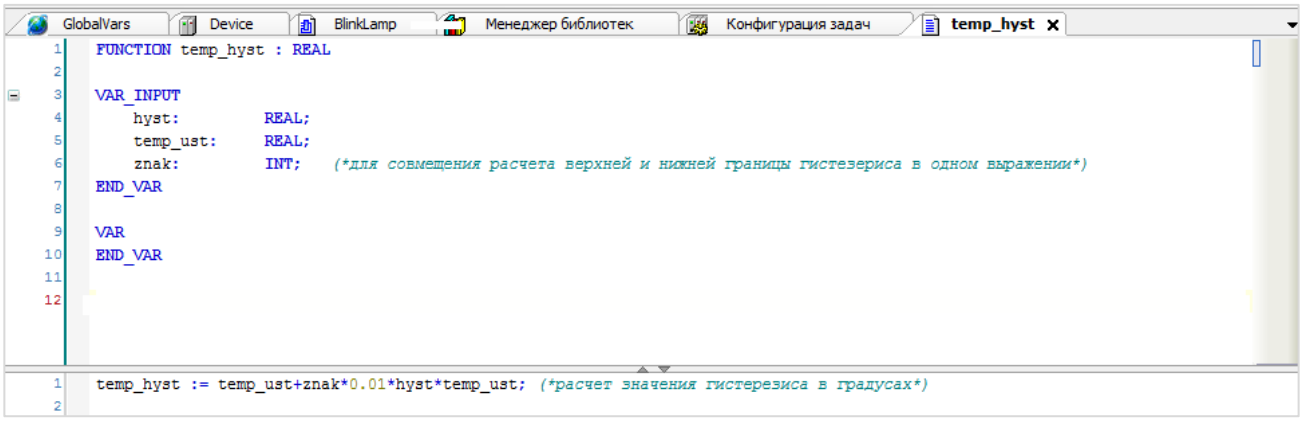


Рисунок 7.93 – Определение переменных и код функции

Локальные входные переменные функции объявляются между ключевыми словами **VAR_INPUT** и **END_VAR**. ***В данном примере*** в случае вызова функции (см. п. 7.4.6.) им присваиваются значения **одноименных глобальных переменных**.

ПРИМЕЧАНИЕ

Несмотря на одинаковые названия, это **разные** переменные, относящиеся к **разным видам**: определенные в функции – **локальные**, определенные в Списке глобальных переменных – **глобальные**. В общем случае название локальной переменной ***не обязано соответствовать*** названию переменной, чье значение она принимает. Иными словами, в определении переменных функции можно использовать **любые** (отвечающие требованиям CODESYS) названия.

Результат вычисления присваивается **непосредственно самой функции**, поэтому определение выходной переменной не требуется.

Целочисленная переменная **znak** принимает значение «−1» при расчете нижней границы гистерезиса и «+1» – при расчете верхней.

Процедура **вызова функции** описывается в п. 7.4.6.

7.4.5 Функциональный блок на языке ST (четырехцветная лампа)

Наряду с мигающей лампой, реализация которой рассмотрена в п. 7.4.3, другой типичной задачей является создание **многопозиционного индикатора**, характеризующего состояние какого-либо объекта. В рамках примера таким индикатором является **аварийная лампа** с двумя состояниями – **Авария** (красный цвет лампы) и **Норма** (зеленый цвет лампы).

Среди **графических примитивов** CODESYS присутствуют только **двухпозиционные** лампы, причем оба состояния каждой из них ***жестко фиксированы***: TRUE – свечение соответствующего оттенка, FALSE – неактивное состояние (лампа не горит). Чтобы создать многопозиционную лампу следует в **Редакторе визуализации** наложить несколько ламп друг на друга и подготовить программу, переключающую их состояния.

Цвет многопозиционной лампы будет отражать **состояние кондиционера** – включен или отключен. Если кондиционер включен, то цвет лампы будет характеризовать **режим его работы**. Принцип действия лампы будет описан в **функциональном блоке** на языке **ST**. В п. 7.4.3 описывается использование готового функционального блока. Далее описывается самостоятельное создание подобного элемента.

Сначала создается **функциональный блок** на языке **ST** с названием **Lamp**. Параметры Extends, Implements и Спецификатор доступа используются при **объектно-ориентированном подходе** к написанию программ, который в данном примере ***не рассматривается.***

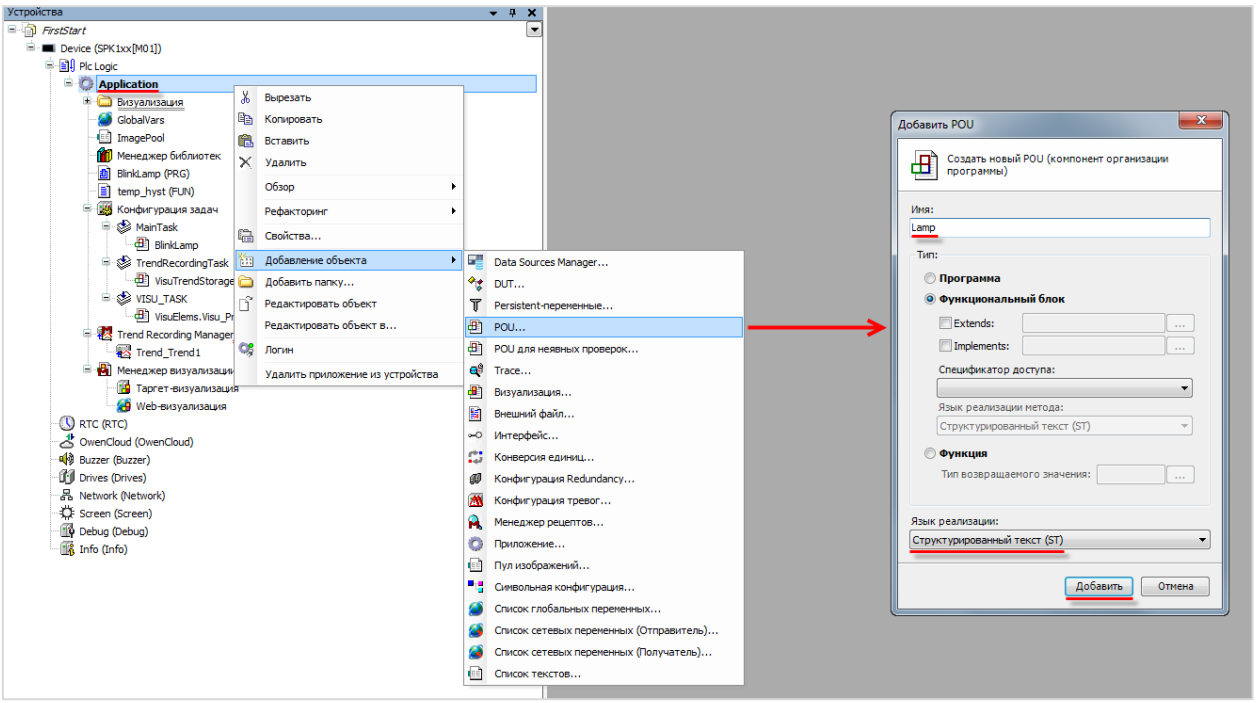


Рисунок 7.94 – Создание функционального блока

Цвет лампы будет зависеть от **состояния кондиционера** (вкл./откл.), **режима работы регулятора** (вкл./откл.) и **режима работы кондиционера** (нагрев/охлаждение), которые определяются глобальными переменными **conditioner_power, regulator_state** и **control_mode** соответственно. Таблица возможных сочетаний и соответствующих им цветов лампы приведена ниже:

Таблица 7.3 – Таблица состояний лампы

Состояние переменной conditioner_power	Состояние переменной regulator_state	Состояние переменной control_mode	Цвет лампы
TRUE	TRUE	FALSE	
TRUE	TRUE	TRUE	
TRUE	FALSE	Не важно	
FALSE	Не важно	Не важно	

Локальными входными переменными функционального блока будут являться **conditioner_power_fb, regulator_state_fb** и **control_mode_fb**, которым при вызове блока будут присваиваться значения соответствующих глобальных переменных **conditioner_power**, **regulator_state** и **control_mode**. Выходами блока являются семь локальных переменных, описывающих состояние четырех ламп (одна переменная отвечает за свечение лампы, вторая – за невидимость, неактивной серой лампе не нужна переменная свечения). Выходные переменные определяются между ключевыми словами **VAR_OUTPUT** и **END_VAR**.

```
1  FUNCTION_BLOCK Lamp
2  VAR_INPUT                    (*входные параметры функционального блока*)
3      conditioner_power_fb: BOOL;
4      regulator_state_fb: BOOL;
5      control_mode_fb: BOOL;
6  END_VAR
7
8  VAR_OUTPUT                   (*выходные параметры функционального блока*)
9          blue_light_fb  :BOOL;
10         red_light_fb   :BOOL;
11         green_light_fb :BOOL;
12
13         blue_inv_fb    :BOOL :=TRUE;
14         red_inv_fb     :BOOL :=TRUE;
15         green_inv_fb   :BOOL :=TRUE;
16         gray_inv_fb    :BOOL :=TRUE;
17 END_VAR
18
19 VAR
20 END_VAR
```

Рисунок 7.95 – Определение переменных функционального блока

Код функционального блока будет выглядеть следующим образом:

```
(*цвет горящей лампы в зависимости от состояния кондиционера, состояния регулятора и режима работы кондиционера*)

IF conditioner_power_fb = TRUE AND regulator_state_fb = TRUE AND control_mode_fb = FALSE THEN
        blue_light_fb  := TRUE;
        blue_inv_fb    := FALSE;

        red_inv_fb     := TRUE;
        green_inv_fb   := TRUE;
        gray_inv_fb    := TRUE;
    ELSIF conditioner_power_fb = TRUE AND regulator_state_fb = TRUE AND control_mode_fb = TRUE THEN
        red_light_fb   := TRUE;
        red_inv_fb     := FALSE;

        blue_inv_fb    := TRUE;
        green_inv_fb   := TRUE;
        gray_inv_fb    := TRUE;
    ELSIF conditioner_power_fb = TRUE AND regulator_state_fb = FALSE THEN
        green_light_fb := TRUE;
        green_inv_fb   := FALSE;

        blue_inv_fb    := TRUE;
        red_inv_fb     := TRUE;
        gray_inv_fb    := TRUE;
    ELSIF conditioner_power_fb = FALSE THEN
        blue_inv_fb    := TRUE;
        red_inv_fb     := TRUE;
        green_inv_fb   := TRUE;
        gray_inv_fb    := FALSE;
END IF
```

Рисунок 7.96 – Код функционального блока

Синтаксис **конструкции с оператором IF** (форматирование использовано исключительно для наглядности):

IF <Логическое условие 1> THEN <Действие 1>
{ELSIF <Логическое условие 2> THEN <Действие 2>
. . .
ELSIF <Логическое условие n> THEN <Действие n>
ELSE <Действие n+1>}
END_IF;

Использование операторов **ELSIF** и **ELSE** является опциональным.

Конструкция работает следующим образом:

1. Если Логическое условие 1 возвращает значение TRUE, то выполняется Действие 1.
2. Если Логическое условие 1 возвращает значение FALSE, то осуществляется переход к оператору ELSIF с Логическим условием 2, который работает аналогично оператору IF.
3. Если все n Логических условий вернули FALSE, то осуществляется переход к оператору ELSE и выполняется соответствующее Действие n+1.

Для реализации четырехпозиционной лампы четыре лампы различных цветов накладываются друг на друга и к ним привязываются соответствующие переменные функционального блока. В результате, в каждый момент времени будет видна только одна лампа (определенная значениями входных переменных), а остальные лампы не будут видны.

Процедура **вызова функционального блока** описывается в п. 7.4.6.

7.4.6 Программа на языке ST (регулятор и модель объекта)

Теперь следует создать **программу MainPrg** на языке **ST**, в которой будут вызываться POU из п. 7.4.4 и п. 7.5.5 (см. рисунок 7.73):

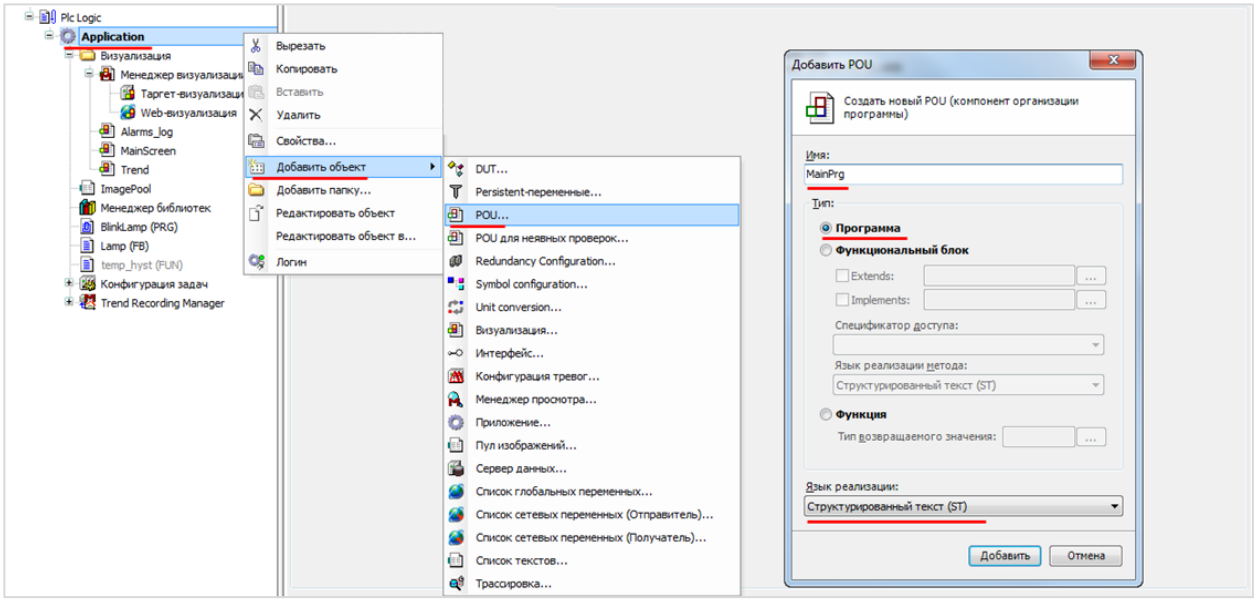


Рисунок 7.97 – Создание программы MainPrg

Программа будет выполнять **следующие действия**:

1. Определять текущий режим измерения (измерение с датчика или режим эмуляции измерения).
2. С помощью вызова функции **temp_hyst** рассчитывать значения границ гистерезиса.
3. На основании положения значения температуры относительно границ гистерезиса переключать кондиционер в соответствующий режим.
4. В случае задания температуры пользователем эмулировать процесс изменения температуры при работе кондиционера.
5. С помощью вызова функционального блока **Lamp** подсвечивать индикатор кондиционера цветом, соответствующим режиму его работы.

Локальные переменные программы:

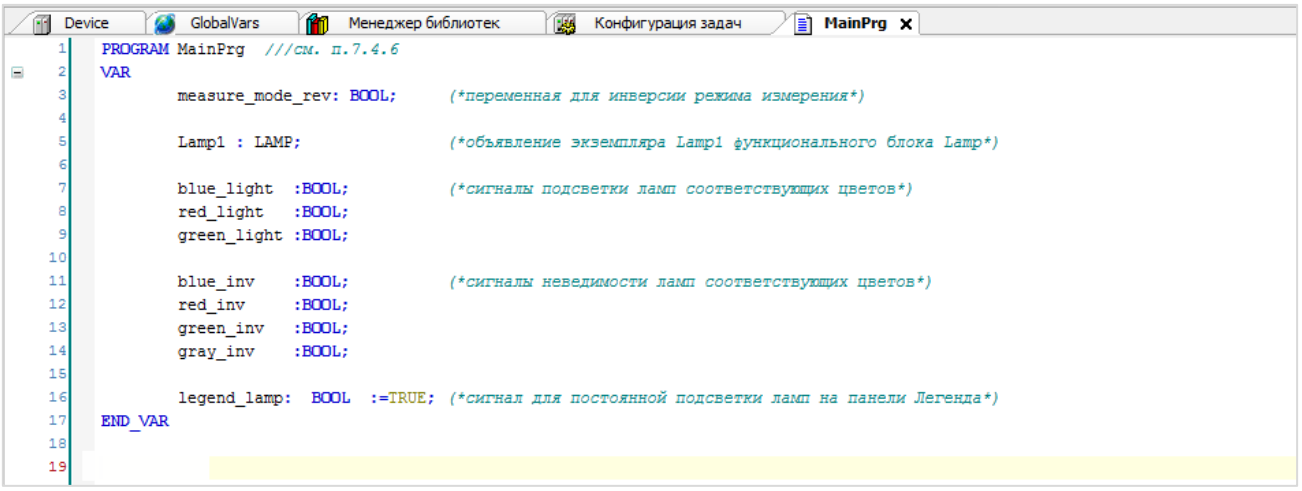


Рисунок 7.98 – Объявление переменных программы

Все переменные, за исключением **measure_mode_rev**, используются для реализации **четырехпозиционной лампы**.

Логическая переменная **measure_mode_rev** используется для **инверсии** значения глобальной переменной **measure_mode**:

```
measure_mode_rev := NOT measure_mode;
```

Соответственно, она принимает значение TRUE, когда measure_mode = FALSE и наоборот.

Данная переменная используется для ***скрытия кнопок управления*** температурой при получении ее значения с датчика. Чтобы кнопки стали **невидимыми** переменная состояния, привязанная к ним, должна принять значение TRUE. Но значение TRUE переменной **measure_mode** соответствует **режиму ручного задания температуры**, и поэтому данная переменная ***не может быть использована*** как переменная состояния. Поэтому следует создать **дополнительную** инвертированную переменную.

Далее описывается пошаговый процесс создания программы в соответствии с приведенной выше последовательностью выполняемых ею действий.

I. Определение режима измерения

```
5
6
7   ////////////////////  п.7.4.6, действие 1  ////////////////////
8
9   (*определение режима измерения*)
10  IF          measure_mode = FALSE THEN temp := temp_real;
11      ELSIF   measure_mode = TRUE  THEN temp := temp_user;
12  END_IF
13
14  ////////////////////////////////////////////////////////////////
15
```

Рисунок 7.99 – Код действия I

Режим измерения (показания с датчика или эмуляция измерения) определяется пользователем с помощью **чекбокса** на экране **MainScreen**. Если установлена галочка, то пользователь задает значение температуры ***вручную***, а контроллер эмулирует работу кондиционера, в противном случае используется значение температуры с датчика. Каждому из режимов измерения соответствует своя переменная температуры (**temp_real** и **temp_user**). Значение переменной текущего режима присваивается переменной **temp** для дальнейшего использования в коде программы с помощью конструкции **IF** из п. 7.4.5.

II. Расчет границ гистерезиса. Вызов функции

```
17
18  ////////////////////  п.7.4.6, действие 2  ////////////////////
19
20  (*вызов функции temp_hyst для пересчета гистерезиса из процентов в температуру*)
21  temp_hyst_low  := temp_hyst(hyst,temp_ust,-1);
22  temp_hyst_high := temp_hyst(hyst,temp_ust,1);
23
24  ////////////////////////////////////////////////////////////////
25
```

Рисунок 7.100 – Код действия II

Пользователь задает уставку температуры **temp_ust** в градусах Цельсия и значение гистерезиса в процентах от этой уставки. Чтобы перевести значения границ **гистерезиса** из процентов в градусы Цельсия, используется **функция temp_hyst** из п. 7.4.4.

Синтаксис вызова функции:

<Имя переменной> := <Имя функции> (<аргумент функции 1>, …, <аргумент функции n>);

где <**Имя переменной**> – имя переменной, которой присваивается результат вызова функции. Тип переменной должен ***совпадать*** с типом функции, иначе возможна потеря данных или некорректная работа программы;

<**Имя функции**> – название функции;

<**аргумент функции**> – переменная, значение которой присваивается входной переменной функции. Число аргументов должно ***строго соответствовать*** числу входных переменных, а порядок аргументов – последовательности объявления переменных функции.

III. Определение режима работы кондиционера

```
28  ////////////////////  п.7.4.6, действие 3  ////////////////////
29
30  (*выбор режима работы кондиционера*)
31  IF        temp < temp_hyst_low  AND conditioner_power  = TRUE THEN
32                                      control_mode      := TRUE;
33                                      regulator_state   := TRUE;
34  (*температура ниже гистерезиса--->включаем подогрев*)
35      ELSIF temp > temp_hyst_high AND conditioner_power  = TRUE THEN
36                                      control_mode      := FALSE;
37                                      regulator_state   := TRUE;
38  (*температура выше гистерезиса--->включаем охлаждение*)
39      ELSIF temp >= temp_hyst_low AND  temp <= temp_hyst_high   THEN regulator_state := FALSE;
40  (*температура в пределах гистерезиса--->переводим кондиционер в режим ожидания*)
41  END_IF
42
43  ////////////////////////////////////////////////////////////
```

Рисунок 7.101 – Код действия III

Режим работы кондиционера определяется положением температуры относительно границ гистерезиса, а также состояниями кондиционера и регулятора (вкл./откл.). Если кондиционер включен, то он может работать в режиме нагрева/охлаждения (если включен регулятор и температура выходит за нижнюю/верхнюю уставку гистерезиса) или режиме ожидания (регулятор включен, но температура находится в допустимых пределах). Это реализуется с помощью **конструкции IF**. От приведенных ранее примеров данная вариация конструкции отличается лишь тем, что в ее условии используется **оператор AND** (логическое И). Оператор AND ***нельзя*** использовать в **действиях**, т. е. следующая запись некорректна:

```
IF <Логическое условие 1>   THEN   <Действие 1>  AND <Действие 2> ;
END_IF;
```

Если **конструкция IF** подразумевает исполнение **нескольких** действий в результате выполнения логического условия, то они перечисляются через пробел. Каждое действие должно заканчиваться точкой с запятой (**;**).

IV. Эмуляция изменения температуры

```
47  ////////////////// п.7.4.6, действие 4 //////////////////
48
49  IF        conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = FALSE AND measure_mode = TRUE THEN
50      temp := temp-3;
51  (*работа кондиционера в режиме охлаждения*)
52      ELSIF conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = TRUE  AND measure_mode = TRUE THEN
53      temp := temp+3;
54  (*работа кондиционера в режиме подогрева*)
55  END_IF
56
57
58  IF          measure_mode = FALSE THEN temp_real := temp;  (*запись температуры после _работы кондиционера в данном цикле_...*)
59      ELSIF   measure_mode = TRUE  THEN temp_user := temp;  (*...для возможности переключения режима измерения в ходе цикла *)
60  END_IF
61
62  ////////////////////////////////////////////////////////
63
64
```

Рисунок 7.102 – Код действия IV

В случае отсутствия реальных входных и выходных сигналов программа должна работать в **режиме эмуляции**. Эмуляция входного сигнала осуществляется вводом пользователем текущего значения температуры при условии, что выбран соответствующий режим измерения (**measure_mode** = TRUE). Для эмуляции выходных сигналов следует реализовать изменение значения температуры во время работы кондиционера в режиме нагрева или охлаждения. Это отражено в приведенном выше коде. **Число «3»** является «***мощностью***» кондиционера – если кондиционер работает в режиме нагрева или охлаждения, при каждом выполнении программы температура в помещении увеличивается или уменьшается на **3 градуса Цельсия** соответственно. Затем новое значение присваивается температурной переменной режима измерения, что необходимо для переключения режима измерения.

V. Подсветка лампы. Вызов функционального блока

```
41  ////////////////////////// п. 7.4.6, действие 5 ///////////////////////
42
43  Lamp1 (*вызов экземпляра Lamp1 функционального блока Lamp*)
44      (
45      conditioner_power_fb := conditioner_power,
46      regulator_state_fb := regulator_state,
47      control_mode_fb      := control_mode,
48
49      blue_light_fb  => blue_light,
50      red_light_fb   => red_light,
51      green_light_fb => green_light,
52
53      blue_inv_fb    => blue_inv,
54      red_inv_fb     => red_inv,
55      green_inv_fb   => green_inv,
56      gray_inv_fb    => gray_inv,
57      );
58
59  ////////////////////////////////////////////////////////////////////////
```

Рисунок 7.103 – Код действия V

Для реализации **четырехпозиционной лампы**, цвет которой зависит от режима работы кондиционера (см. табл. 7.3), мы используем **функциональный блок Lamp**, созданный в п. 7.4.5 (там же описан и принцип работы лампы).

Вызов функционального блока осуществляется через обращение к его **экземпляру**.

Пример синтаксиса вызова функционального блока:

```
Block_ex (in1: = a, in2 := b, out => c);
```

где <**Block_ex**> – имя экземпляра функционального блока;

<**in1**>, <**in2**> – имена входных переменных функционального блока;

<**a**>, <**b**> – имена переменных, значения которых присваиваются входным переменным функционального блока;

<**out**> – имя выходной переменной функционального блока;

<**c**> – имя переменной, которой присваивается выходное значение функционального блока (требуется специальный оператор присваивания **=>**

В приведенном выше коде каждой входной переменной функционального блока при его вызове присваивается значение локальной переменной программы, а значение каждой выходной переменной функционального блока в свою очередь присваивается соответствующей локальной переменной программы. Требования не является обязательным, функциональный блок можно вызывать и **без обращения** ко всем его входам/выходам.

Допускается вызов функции без обращения к входам/выходам:

```
Block_ex();
```

Альтернативный вариант синтаксиса вызова функционального блока (с поочередным обращением к его входам/выходам):

```
Block_ex.in1 := a;
Block_ex.in2 := b;
Block_ex();              (*вызов блока *);
c := Block_ex.out ;
```

Для удобства размещения POU в дереве проекта можно с помощью нажатия **ПКМ** на компонент **Applicaton** в дереве проекта создать новую папку **Программы**, и, ***зажав* ЛКМ**, перенести туда созданные POU:

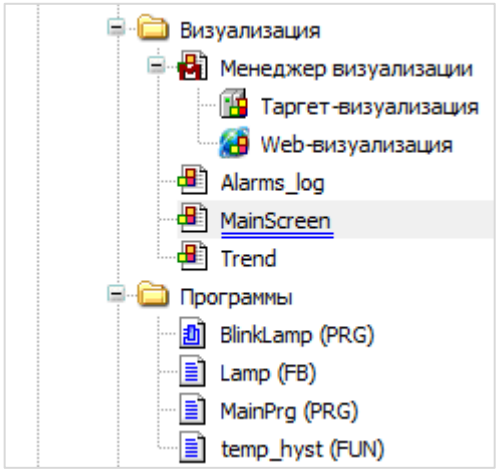


Рисунок 7.104 – Папка Программы в дереве проекта

Полный код программы **MainPrg** выглядит следующим образом:

```
(*сигнал для скрытия кнопок изменения текущей температуры при получении значения температуры с датчика*)
measure_mode_rev := NOT measure_mode;



////////////////////  п.7.4.6, действие 1  ////////////////////

(*определение режима измерения*)
IF          measure_mode = FALSE THEN temp := temp_real;
    ELSIF   measure_mode = TRUE  THEN temp := temp_user;
END_IF

////////////////////////////////////////////////////////////



////////////////////  п.7.4.6, действие 2  ////////////////////

(*вызов функции temp_hyst для пересчета гистерезиса из процентов в температуру*)
temp_hyst_low  := temp_hyst(hyst,temp_ust,-1);
temp_hyst_high := temp_hyst(hyst,temp_ust,1);

////////////////////////////////////////////////////////////



////////////////////  п.7.4.6, действие 3  ////////////////////

(*выбор режима работы кондиционера*)
IF        temp < temp_hyst_low  AND conditioner_power  = TRUE THEN
                                    control_mode      := TRUE;
                                    regulator_state   := TRUE;
(*температура ниже гистерезиса--->включаем подогрев*)
    ELSIF temp > temp_hyst_high AND conditioner_power  = TRUE THEN
                                    control_mode      := FALSE;
                                    regulator_state   := TRUE;
(*температура выше гистерезиса--->включаем охлаждение*)
    ELSIF temp >= temp_hyst_low AND  temp <= temp_hyst_high   THEN regulator_state := FALSE;
(*температура в пределах гистерезиса--->переводим кондиционер в режим ожидания*)
END_IF

////////////////////////////////////////////////////////////



////////////////////  п.7.4.6, действие 4  ////////////////////

IF        conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = FALSE AND measure_mode = TRUE THEN
    temp := temp-3;
(*работа кондиционера в режиме охлаждения*)
    ELSIF conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = TRUE  AND measure_mode = TRUE THEN
    temp := temp+3;
(*работа кондиционера в режиме подогрева*)
END_IF


IF          measure_mode = FALSE THEN temp_real := temp;  (*запись температуры после _работы кондиционера в данном цикле_...*)
    ELSIF   measure_mode = TRUE  THEN temp_user := temp;  (*...для возможности переключения режима измерения в ходе цикла *)
END_IF

////////////////////////////////////////////////////////////



////////////////////  п.7.4.6, действие 5  ////////////////////

Lamp1 (*вызов экземпляра Lamp1 функционального блока Lamp*)
    (
    conditioner_power_fb := conditioner_power,
      regulator_state_fb := regulator_state,
    control_mode_fb      := control_mode,

    blue_light_fb  => blue_light,
    red_light_fb   => red_light,
    green_light_fb => green_light,

    blue_inv_fb    => blue_inv,
    red_inv_fb     => red_inv,
    green_inv_fb   => green_inv,
    gray_inv_fb    => gray_inv,
    );

////////////////////////////////////////////////////////////
```

Рисунок 7.105 – Полный код программы MainPrg

7.5 Связь визуализации и программных переменных. Настройка кнопок

После подготовки визуальной и программной части нашего приложения следует привязать переменные программ к элементам визуализации, а также настроить **действия** кнопок.

7.5.1 Экран MainScreen

I. Привязка переменных

Для начала следует привязать к элементу **Отображение линейки** переменную **temp**, которая характеризует текущее значение температуры. Для привязки следует выделить элемент и дважды нажать **ЛКМ** на параметр **Значение** в его свойствах:

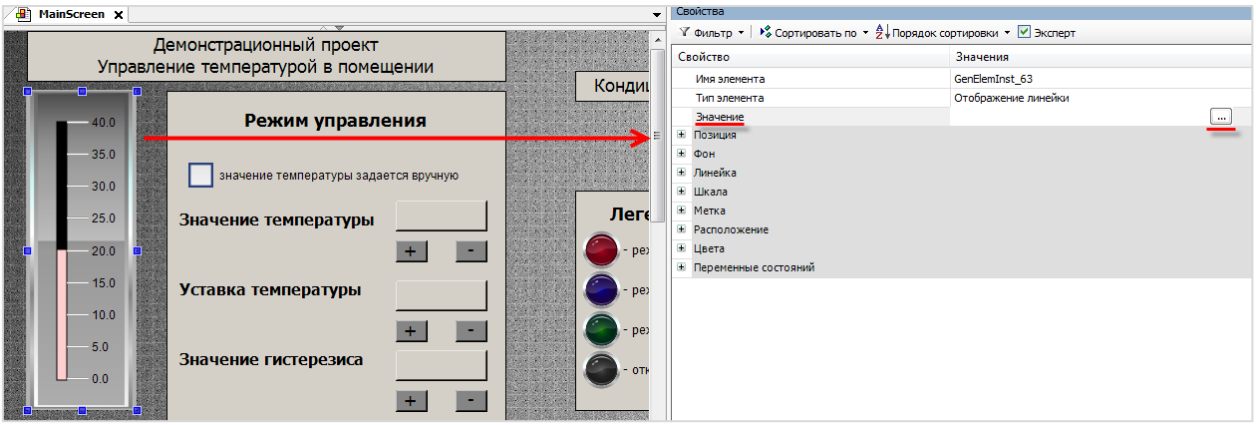


Рисунок 7.106 – Параметр Значение в свойствах элемента

После этого можно ввести название переменной вручную или с помощью нажатия соответствующей кнопки выбрать ее через **Ассистент ввода**:

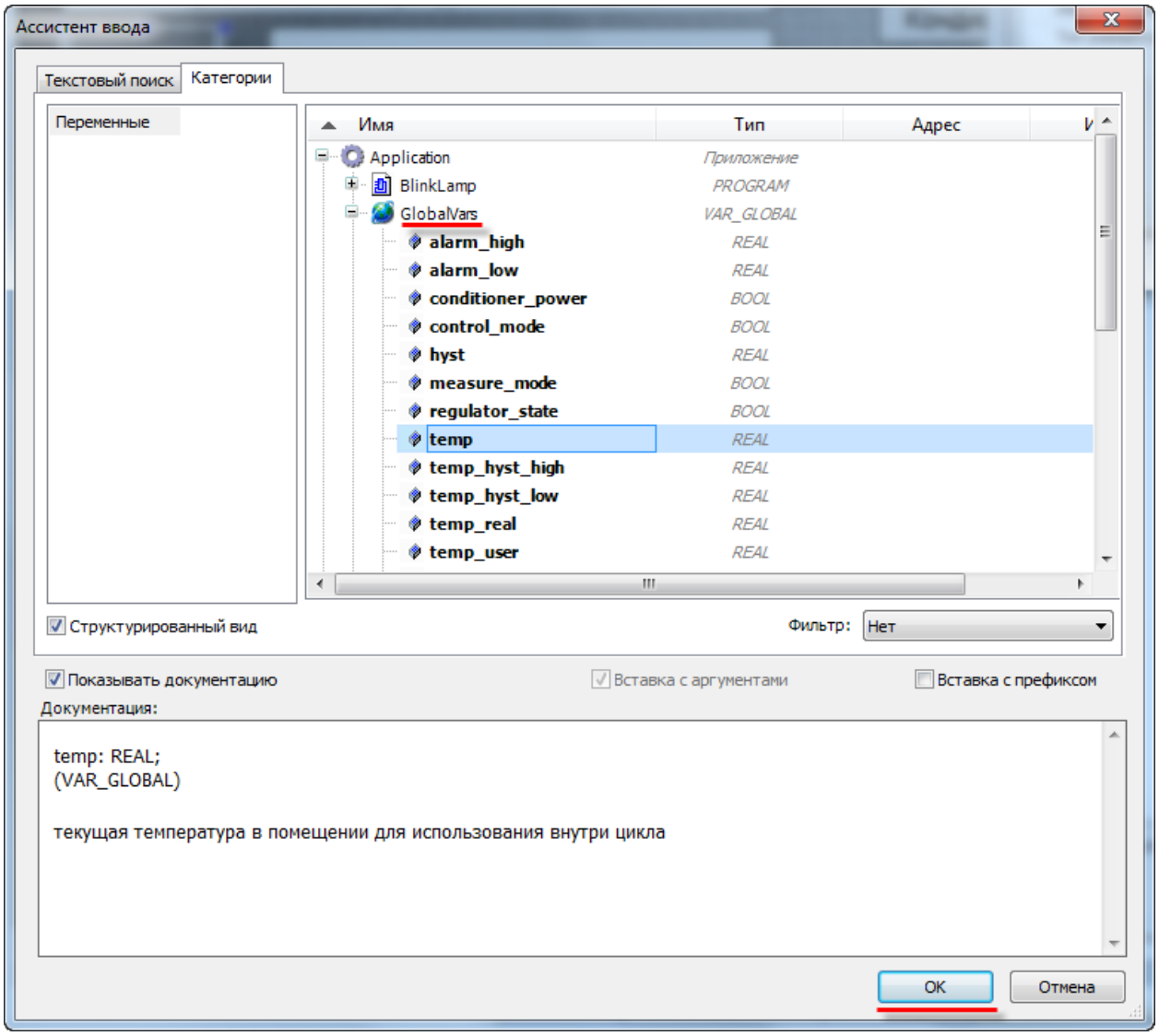


Рисунок 7.107 – Ассистент привязки переменной

Привязка переменной к данному графическому элементу завершена.

К другим элементам переменные привязываются аналогично, за исключением того, что параметр **Значение** у некоторых элементов может называться **Переменная**.

К элементу **Чекбокс** привязывается переменная **measure_mode** (выбор режима измерения), к **Клавишному выключателю** – **conditioner_power** (управление питанием выключателя).

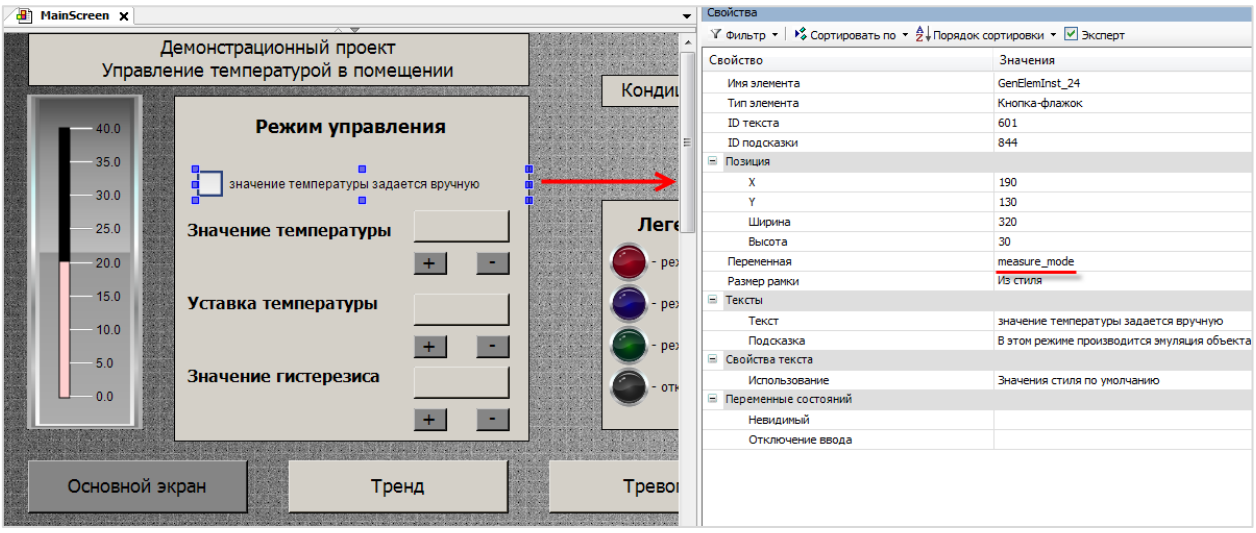


Рисунок 7.108 – Привязка переменной measure_mode к чекбоксу

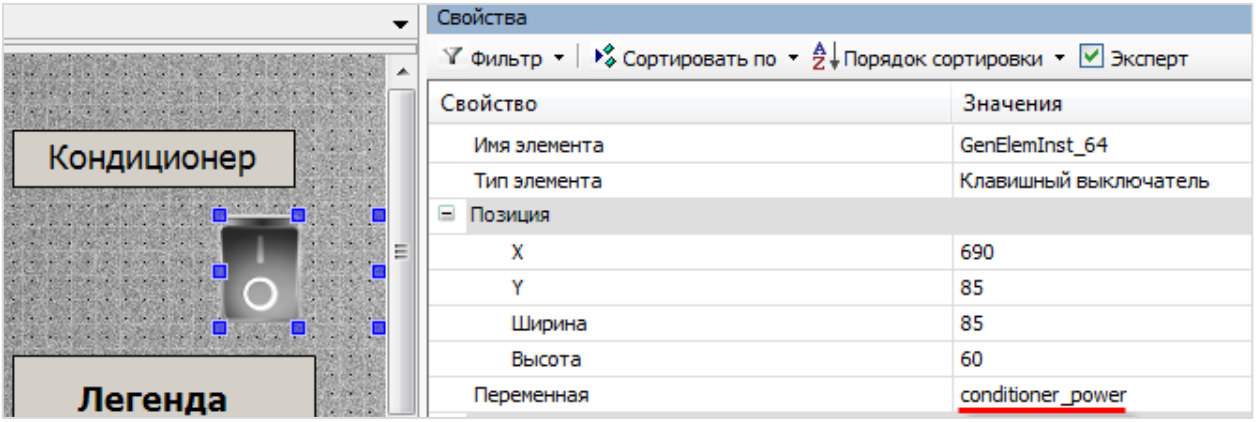


Рисунок 7.109 – Привязка переменной conditioner_power к выключателю

К цветным индикаторам за пределами рабочего поля будет привязано по две переменных – одна определяет состояние индикатора (святящийся/потухший), вторая – видимость (невидимый/видимый). Данная привязка необходима для реализации многопозиционного индикатора (см. п. 7.4.5).

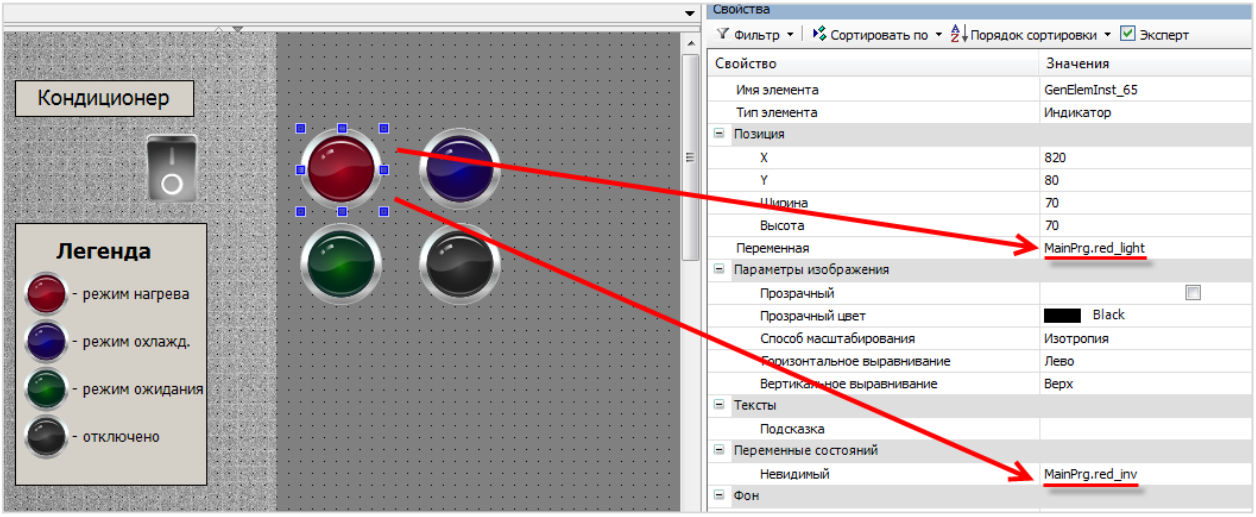


Рисунок 7.110 – Привязка переменных к индикатору red_lamp

ПРИМЕЧАНИЕ

Переменные **red_light** и **red_inv** являются ***локальными*** переменными программы **MainPrg**, поэтому во время набора их названий вручную следует указывать также название программы.

В случае добавления переменных через **Ассистент ввода** правильное название формируется автоматически.

Аналогичным образом привязываются переменные к остальным индикаторам (см. таблицу 7.4). К серому индикатору будет привязана только переменная **невидимости**, т. к. лампа никогда не будет подсвечиваться серым цветом. Затем следует наложить эти лампы друг на друга и поместить получившийся **многопозиционный индикатор** рядом с выключателем кондиционера:

Рисунок 7.111 – Многопозиционный индикатор, полученный наложением четырех индикаторов друг на друга

К цветным индикаторам на панели **Легенда** следует привязать переменную **legend_lamp**, значение которой всегда равно TRUE (сама панель носит поясняющий характер, эти лампы будут гореть всегда). К серой лампе переменные не привязываются.

Переменную можно привязать сразу к группе выделенных графических элементов:

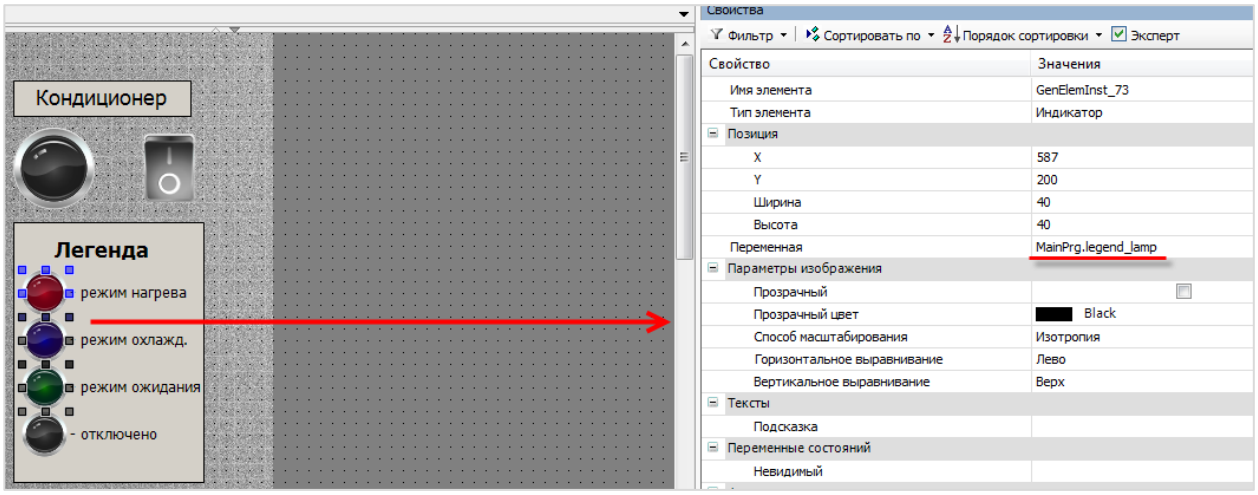


Рисунок 7.112 – Привязка переменной к группе элементов

Список переменных, привязанных к экрану **MainScreen** после действий из п. 7.5.1., пп. I, представлен в таблице ниже.

Табл. 7.4 – Переменные, привязанные к графическим элементам экрана MainScreen

Элемент	Параметр	Привязанная переменная
Отображение линейки	Значение	temp
Чекбокс (кнопка-флажок)	Переменная	measure_mode
Клавишный выключатель	Переменная	conditioner_power
Большие индикаторы		
Красный индикатор (red)	Переменная	red_light
	Переменная состояния (невидимость)	red_inv
Синий индикатор (blue)	Переменная	blue_light
	Переменная состояния (невидимость)	blue_inv
Зеленый индикатор (green)	Переменная	green_light
	Переменная состояния (невидимость)	green_inv
Серый индикатор (gray)	Переменная состояния (невидимость)	gray_inv
Малые индикаторы (панель Легенда)		
Красный индикатор (red)	Переменная	legend_lamp
Синий индикатор (blue)	Переменная	legend_lamp
Зеленый индикатор (green)	Переменная	legend_lamp
Серый индикатор (gray)	-	-

II. Вывод значений переменных

На панели **Режим управления** размещены три **поля вывода** значений параметров: температуры, ее уставки и уставки гистерезиса. Помимо привязки к ним переменных следует также настроить **формат вывода** значений, который указывается во вкладке **Тексты**.

Настройки отображения значения температуры:

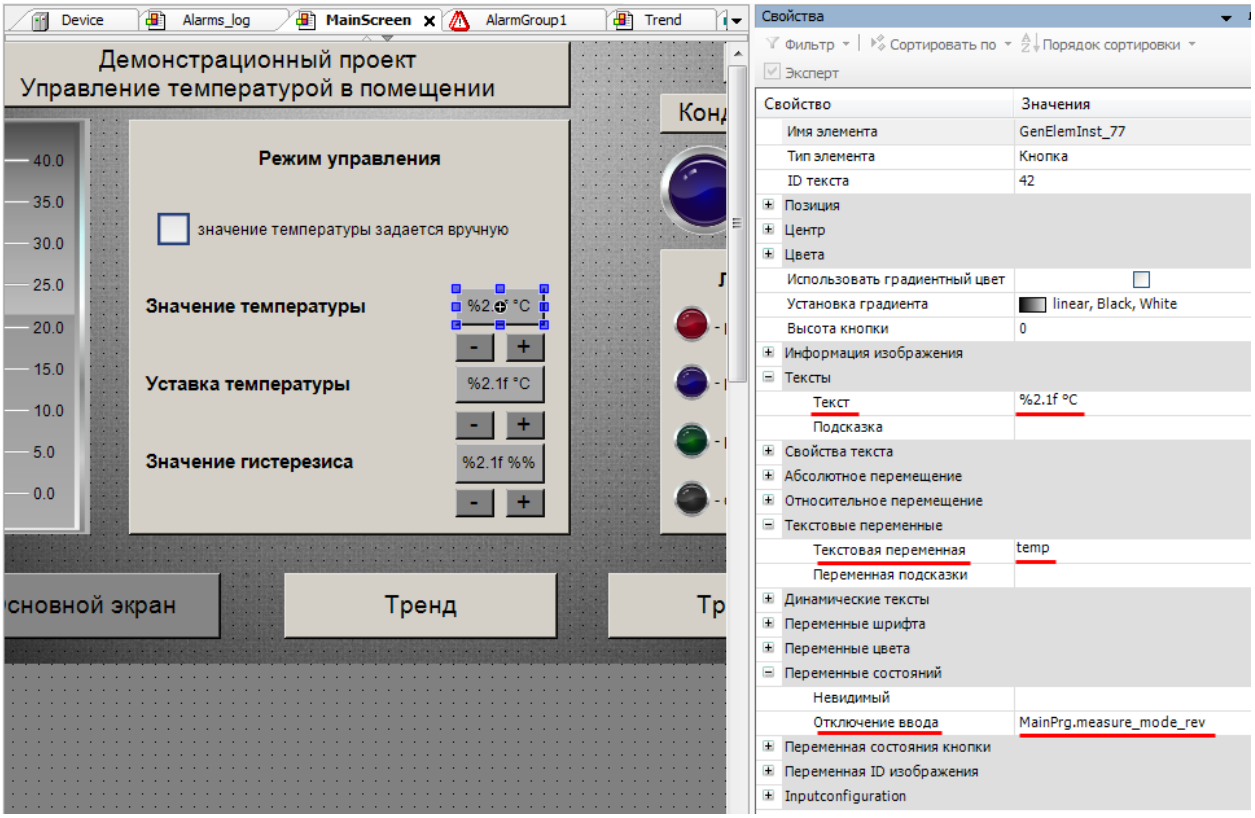


Рисунок 7.113 – Настройки вывода значения переменной temp

В поле **Отключение ввода** вкладки **Переменные состояний** указывается переменная, которая, принимая значение TRUE, делает данный элемент неактивным для управления. В режиме эмуляции значение температуры будет задаваться пользователем с помощью нажатия на данное поле вывода (или на расположенные под ним кнопки). Чтобы исключить эту возможность в случае получения значения температуры с датчика следует привязать к данному параметру переменную **Measure_mode_rev**.

В поле **Текстовая переменная** указывается переменная, значение которой будет выводиться элементом. В данном случае это переменная **temp**, характеризующая текущее значение температуры.

В поле **Текст** указывается **форматирование** вывода. Общий синтаксис форматирования следующий:

<Текст> %<мин. ширина>.<точность><спецификатор переменной> <Текст>

где **<Текст>** – дополнительный текст (например, название и размерность переменной);

% – спецсимвол, после которого указывается тип переменной (перед типом могут быть указаны настройки вывода). Если необходимо вывести символ процента в качестве текста, его следует записать как %%;

<мин. ширина> – ширина поля вывода;

<точность> – количество выводимых символов после запятой (для целочисленных переменных), количество выводимых символов (для строковых переменных);

<спецификатор переменной> – определяет тип выводимой переменной.

Для значения температуры используется форматирование **%2.1f °C**, т. е. тип данных – действительное число с отображением одного знака после запятой.

Для корректного отображения значений, **спецификатор** переменной должен соответствовать **типу данных**, которому принадлежит переменная (см. таблицу 7.1). Список спецификаторов приведен в таблице ниже:

Таблица 7.5 – Спецификаторы типов переменных

Символ	Тип отображаемых данных	Пример использования		
		Фактическое значение	Форматирование	Отображаемое значение
%d	Десятичное число	10	%d	10
%b	Двоичное число	10	%b	00000000 00001010
%o	Восьмеричное число без знака (без ведущего нуля)	10	%o	12
%x	Шестнадцатеричное число без знака (без ведущего нуля)	10	%x	a
%u	Десятичное число без знака	−10	%u	10
%c	Символ таблицы ASCII (переменная типа BYTE)	33	%c	!
%s	Строка	abcdef	%s %.3s	abcdef abc
%f	Действительное число	10.1111	%f %2.2f	10.111100 10.11
%t	Время	12:48:52	%t[HH:mm:ss]	12:48:52

Поля вывода значений уставки температуры и гистерезиса настраиваются по аналогии с выводом температуры – к ним привязываются текстовые переменные **temp_ust** и **hyst**. Поле **Отключение ввода** остается незаполненным, т. к. управление данными переменными не зависит от режима измерения.

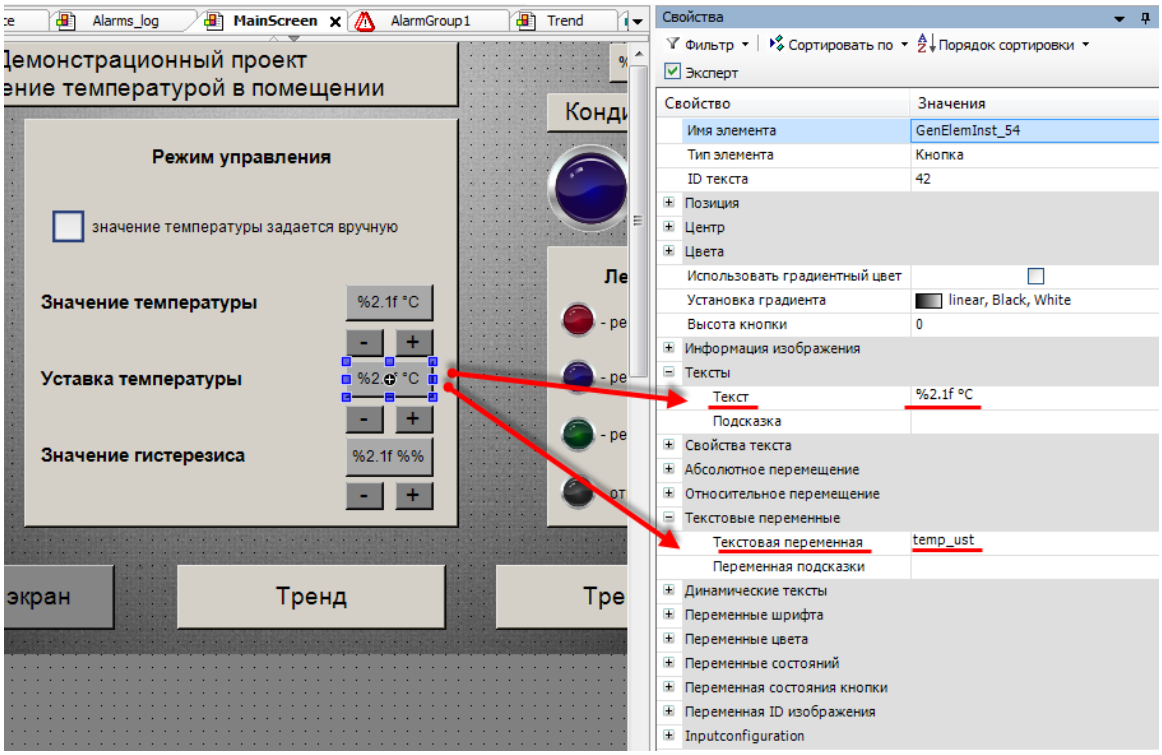


Рисунок 7.114 – Настройки вывода значения переменной temp_ust

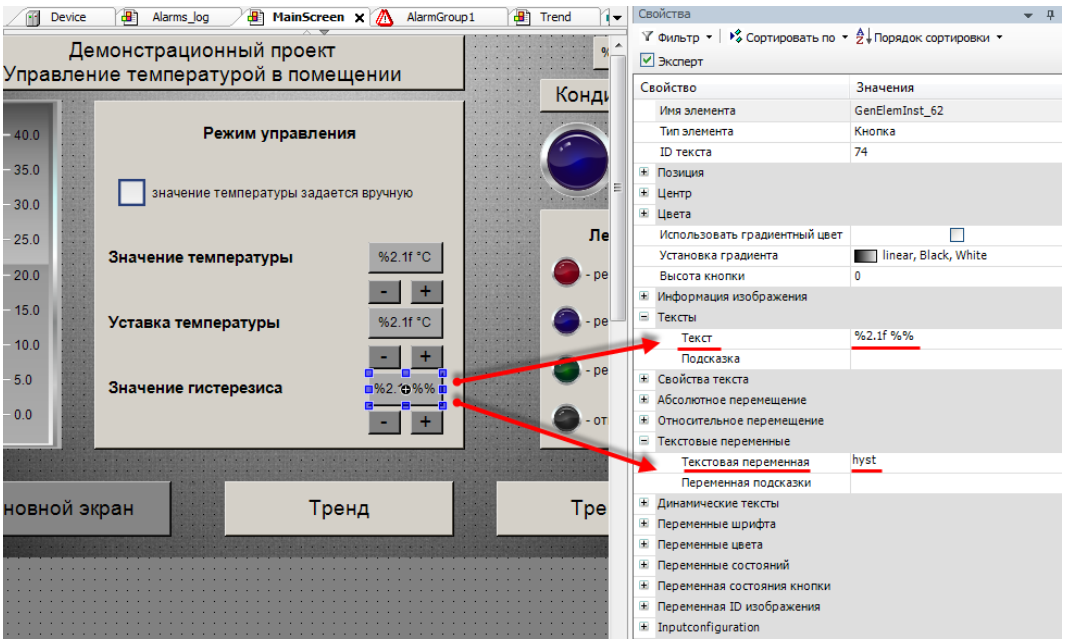


Рисунок 7.115 – Настройки вывода значения переменной hyst

Помимо **отображения** значений параметров эти поля будут использоваться для их **ввода**. Необходимые настройки описываются в п. III.

III. Настройка действий

На экране **MainScreen** расположено 11 кнопок, для которых будут настроены действия: 2 кнопки перехода, 3 кнопки ввода значений, 6 кнопок «увеличить»/«уменьшить»:

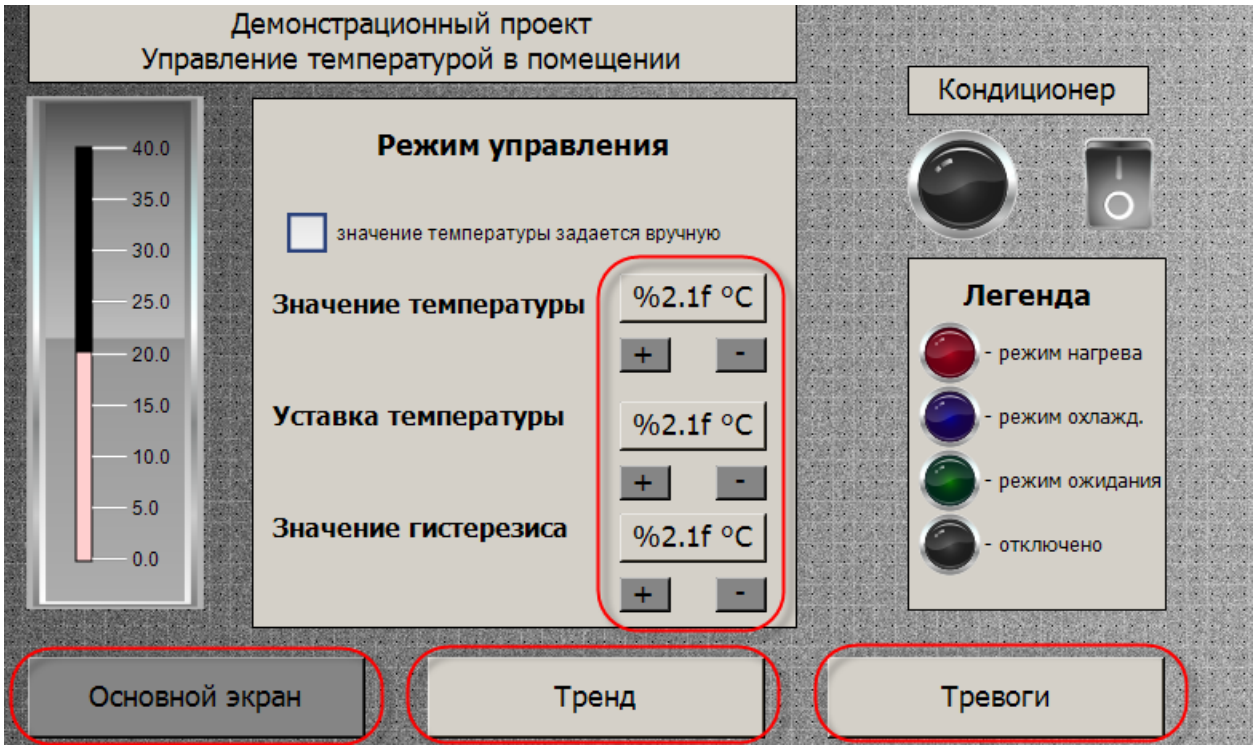


Рисунок 7.116 – Кнопки экрана MainScreen, для которых будут настроены действия

Сначала настраиваются кнопки перехода с экрана **MainScreen** на экраны **Trend** и **Alarm_log**. Переход будет осуществляться нажатием **ЛКМ** на соответствующую кнопку. Настраивать переход с экрана **MainScreen** на экран **MainScreen** не требуется.

Затем следует выделить кнопку **Тренд,** выбрать в ее настройках вкладку **Inputconfiguration** и нажать **ЛКМ** на поле **OnMouseClick**. Откроется диалоговое окно **Конфигурации ввода**.

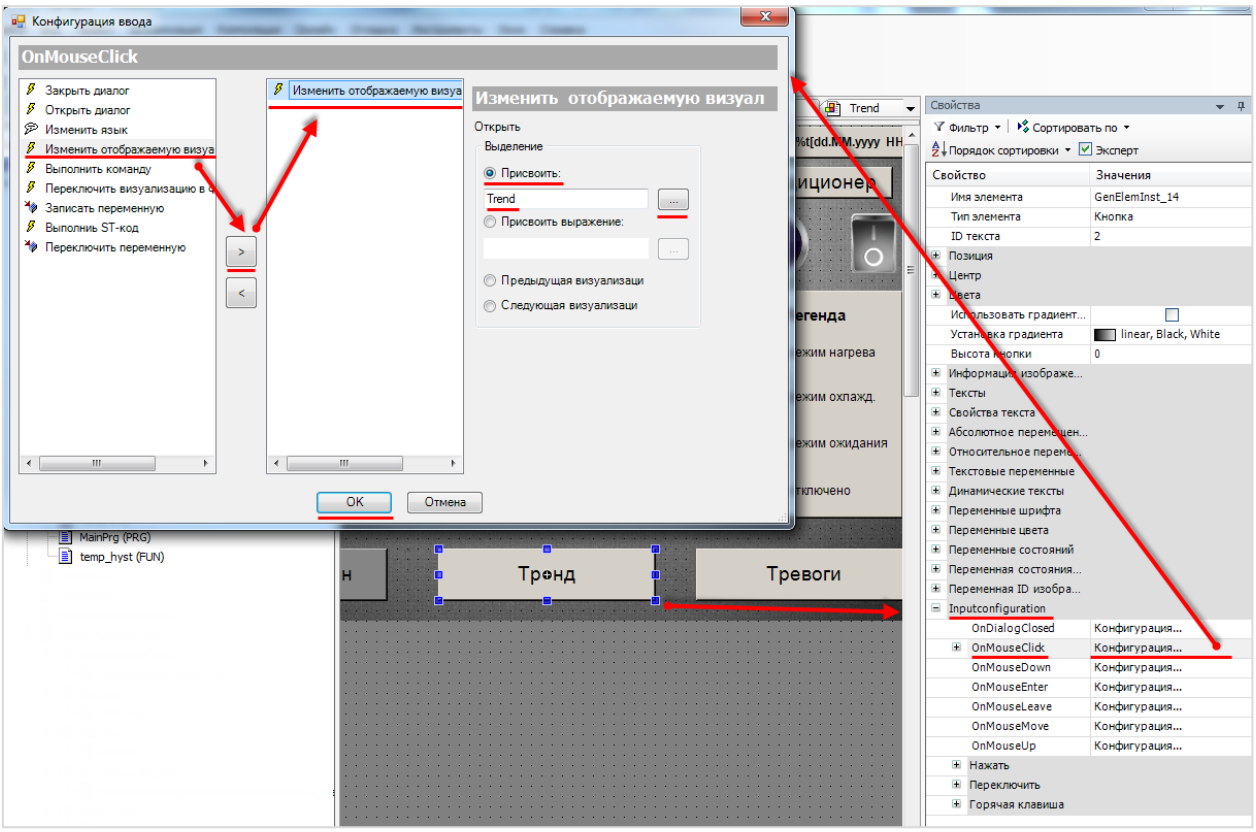


Рисунок 7.117 – Диалоговое окно Конфигурации ввода (изменение отображаемой визуализации)

На левой панели следует выбрать действие **Изменить отображаемую визуализацию**, с помощью кнопки «**>**» присвоить его кнопке **Тренд**. В настройках действия следует выбрать экран, на который должен осуществлять переход – экран **Trend**, и нажать кнопку **ОК**.

Аналогичным образом настраивается **Тревоги** (переход на экран **Alarm_log**). Далее следует настроить кнопки перехода на остальных экранах проекта.

Далее настраиваются поля вывода параметров таким образом, чтобы по нажатию **ЛКМ** на них открывалось диалоговое окно **задания значения переменной**.

Для настройки следует выделить вывода **Значение температуры**, выбрать в его настройках вкладку **Inputconfiguration** и нажать **ЛКМ** на поле **OnMouseClick**. Откроется диалоговое окно **Конфигурации ввода**.

ПРИМЕЧАНИЕ

Данное диалоговое окно отображается некорректно в русскоязычной версии CODESYS V3.5 SP11 Patch 5. Переключить язык можно в меню **Опции – Международные установки**.

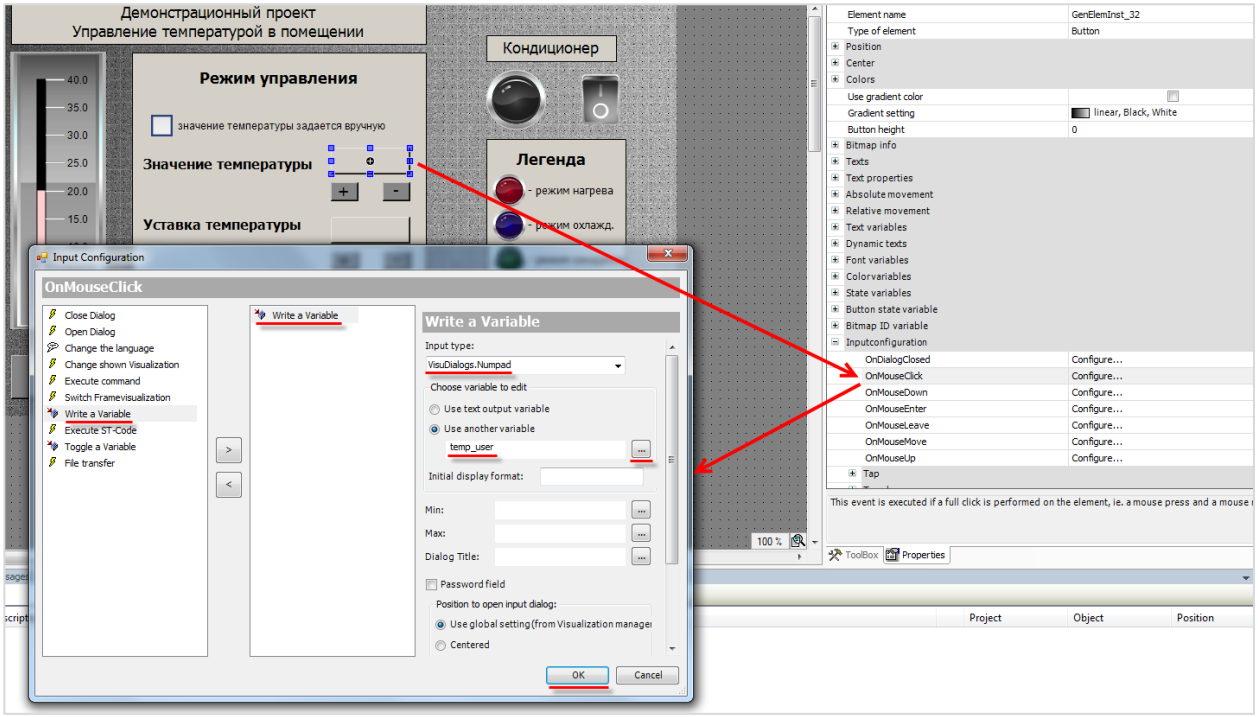


Рисунок 7.118 – Диалоговое окно Конфигурации ввода (запись переменной)

На левой панели следует выбрать действие **Записать переменную**, с помощью кнопки «**>**» присвоить его выделенному полю ввода, в настройках действия выбрать **тип клавиатуры** (в данном случае – цифровая) и записываемую переменную (поскольку задание значения температуры возможно только в режиме эмуляции, используется переменная **temp_user** вместо **temp**). После завершения настройки следует нажать кнопку **ОК**.

Аналогично настраиваются поля ввода **Уставка температуры** (запись переменной **temp_ust**) и **Значение гистерезиса** (запись переменной **hyst**). Так как в данном случае запись происходит в текстовые переменные элементов, то вместо их указания достаточно выбрать пункт **Использовать текстовую выходную переменную**.

Для гистерезиса также задается нижний и верхний предел записи переменной – 0 и 100 соответственно (т. к. гистерезис задается в процентах относительно температуры уставки).

Далее следует выделить кнопку **«увеличить»,** расположенную под полем вывода значения температуры, выбрать в ее настройках вкладку **Inputconfiguration** и нажать **ЛКМ** на поле **OnMouseClick**. Откроется диалоговое окно **Конфигурации ввода**.

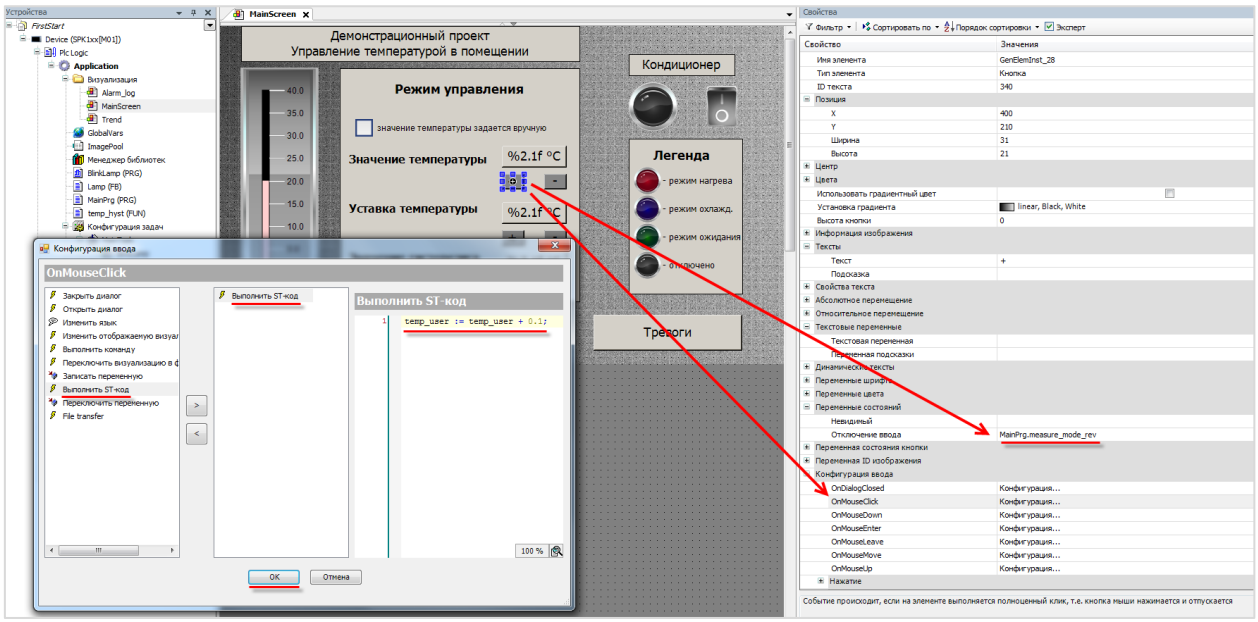


Рисунок 7.119 – Диалоговое окно Конфигурации ввода (выполнение ST-кода)

Затем на левой панели следует выбрать действие **Выполнить ST-код**, с помощью кнопки «**>**» присвоить его выделенному полю ввода и написать **код на языке ST,** который будет выполняться при нажатии кнопки:

```
temp_user := temp_user + 0.1;
```

Каждое нажатие на кнопку «уменьшить» будет приводить к увеличению температуры на 0,1 градус Цельсия.

Так как задавать значение температуры можно только в режиме эмуляции, то в настройках кнопки во вкладке **Переменные состояний** привязываются к параметру **Невидимый** переменную **measure_mode_rev**. Тогда в режиме «измерение с датчика» кнопки «увеличить»/«уменьшить» отображаться не будут. Невидимость носит **функциональный**, но не визуальный характер – т. е. нажатие на область расположения невидимой кнопки ***не приведет*** к выполнению какого-либо действия.

Аналогичным образом настраивается кнопка **«уменьшить».** Единственным отличием в программном коде будет знак вычитания:

```
temp_user := temp_user - 0.1;
```

Далее настраиваются кнопки «увеличить»/«уменьшить» для уставки температуры и гистерезиса. Для этих кнопок настраивать **невидимость** не нужно. Исполняемый ими код будет выглядеть следующим образом:

- для температуры уставки:

  temp_ust := temp_ust - 0.1;

  temp_ust := temp_ust + 0.1;

- для значения гистерезиса:

  hyst := hyst - 0.1;

  hyst := hyst + 0.1;

Вкладка **Inputconfiguration** предоставляет возможность настройки различных вариантов по взаимодействию с кнопками – можно привязывать действия не только к их нажатию, но и зажатию, отпусканию, наведению курсора мыши и т. д. Список возможных действий также не ограничивается упомянутыми выше действиями. В рамках примера все возможные действия рассматриваться не будут. Подробную информацию и о них можно найти в руководстве **CODESYS V3.5. Визулизация**.

7.5.2 Экран Trend

I. Привязка переменных, настройка кнопок

В первую очередь следует повторить действия и из предыдущих пунктов:

1. Настроить кнопки перехода на другие экраны (см. п. 7.5.1 пп. III) и отображение системного времени (см. п. 7.5.1 пп. II).
2. Привязать к желтой аварийной лампе переменную **yellow_lamp** (см. п. 7.5.1 пп. I).
3. Настроить поле для **ввода и отображения** названия лампы следующим образом (см. п. 7.5.1 пп. III):

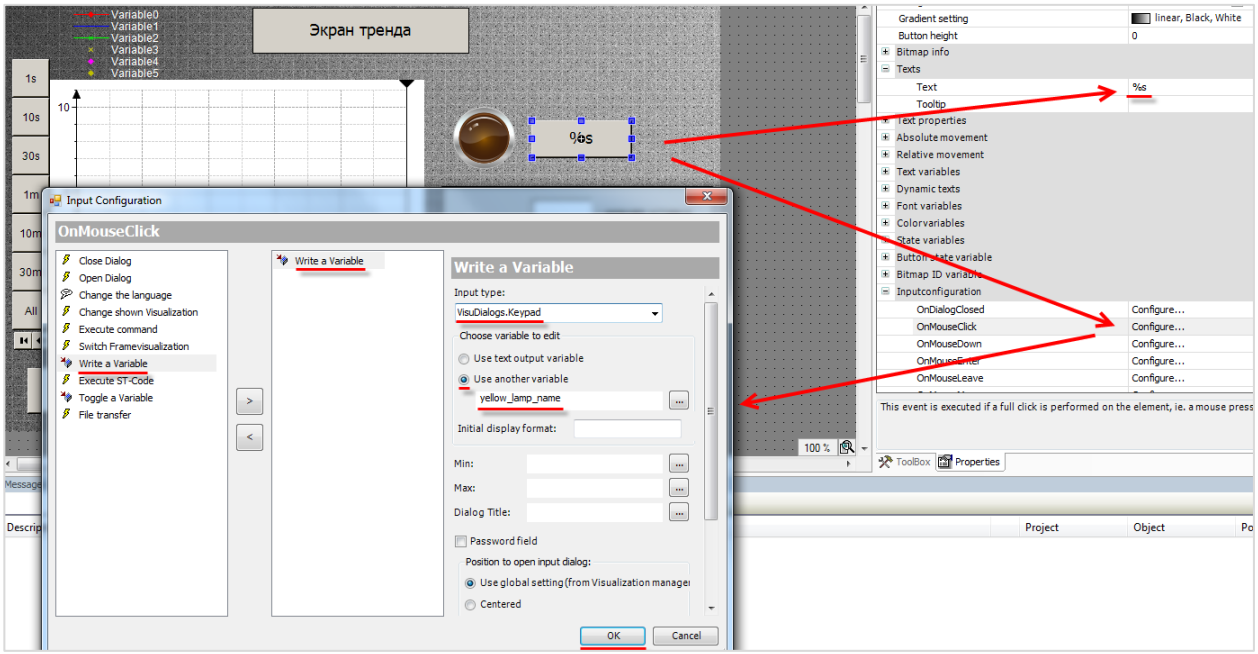


Рисунок 7.120 – Настройка поля ввода/отображения названия сигнальной лампы

ПРИМЕЧАНИЕ

Данное диалоговое окно отображается некорректно в русскоязычной версии CODESYS V3.5 SP11 Patch 5. Переключить язык можно в меню **Опции – Международные установки**.

4. Настроить поля ввода и отображения значений уставок тревог.
Форматирование вывода – **%2.1f**, конфигурация действия следующая:

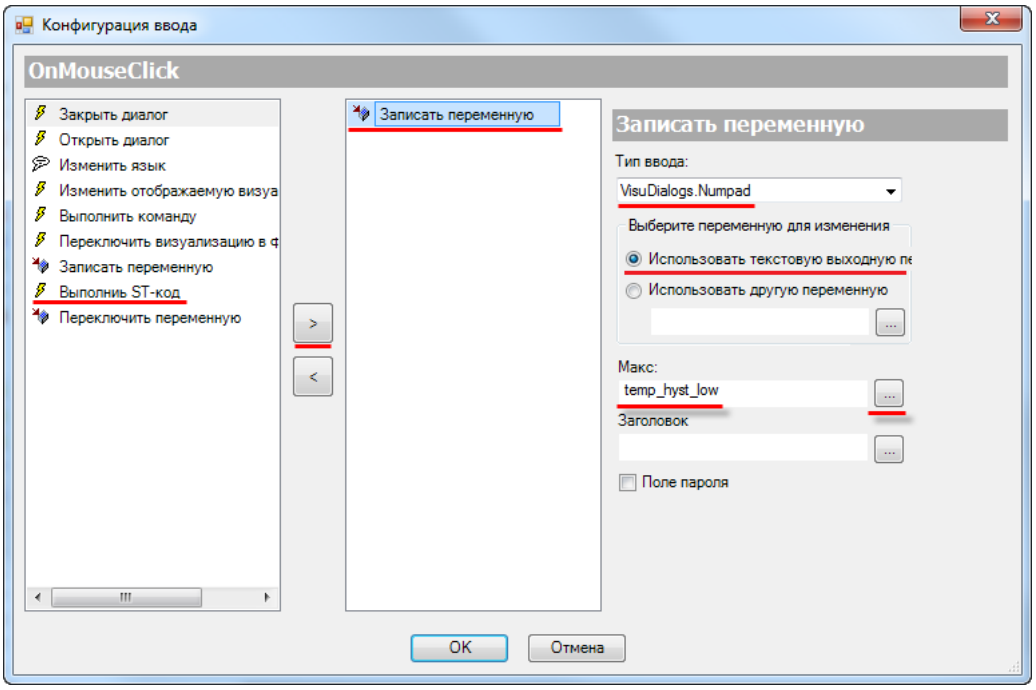


Рисунок 7.121 – Настройка действия поля нижней уставки тревоги

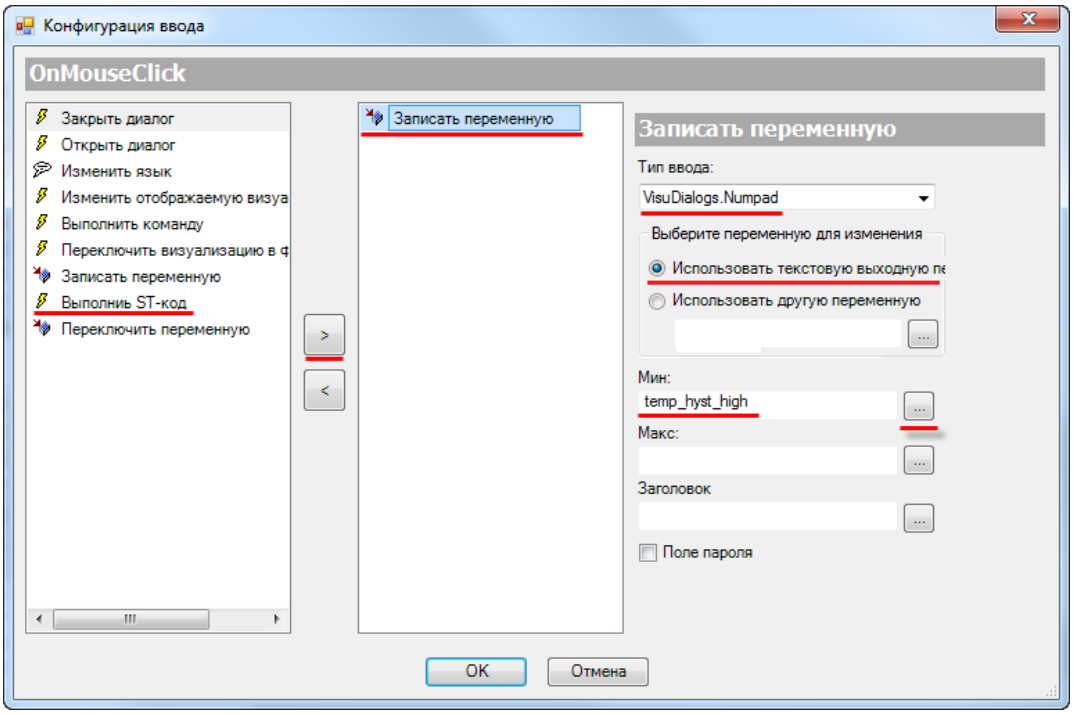


Рисунок 7.122 – Настройка действия поля верхней уставки тревоги

II. Настройка элемента Тренд

Для настройки тренда следует нажать на него **ПКМ** и в контекстном меню выбрать пункт **Изменить запись** (или нажать на компонент **Trend_Trend1** в дереве проекта):

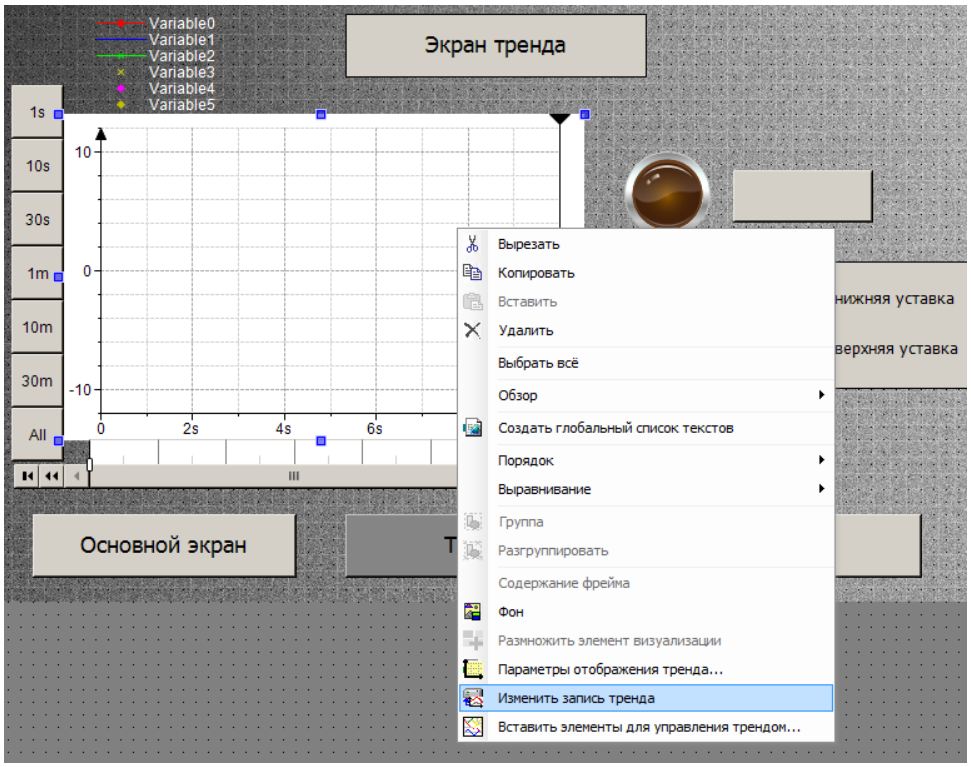


Рисунок 7.123 – Открытие окна конфигурации тренда

Появится **окно конфигурации тренда**, которое выглядит следующим образом:

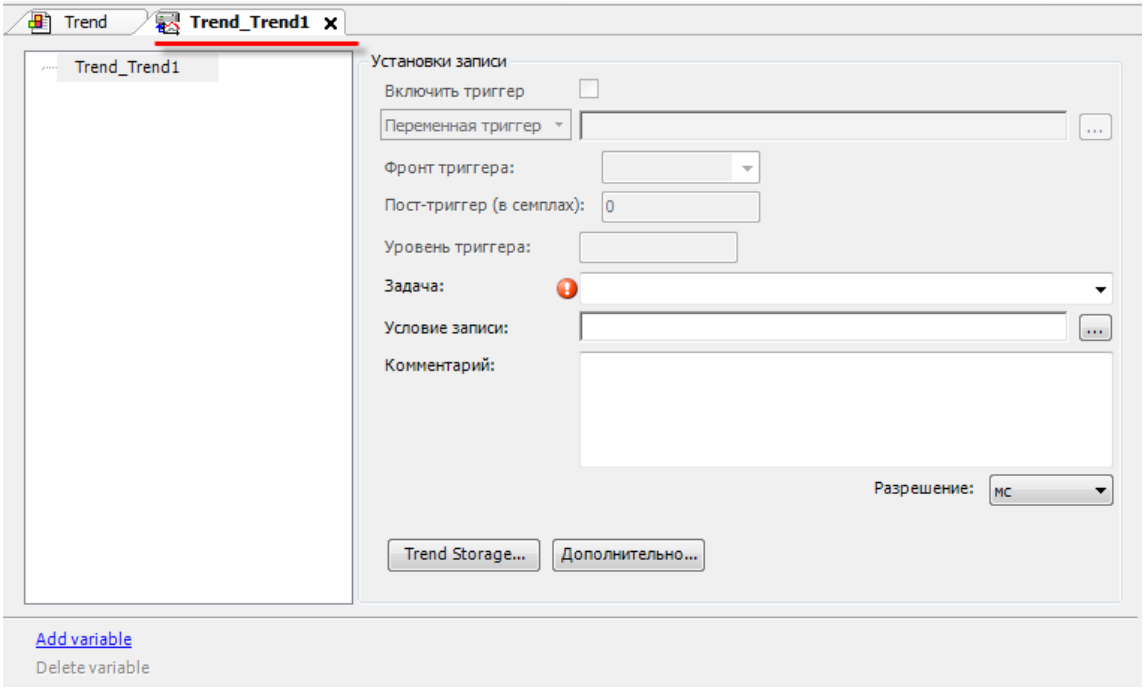


Рисунок 7.124 – Окно конфигурации тренда

В поле **Задача** следует указать задачу, к которой будет привязан тренд. Подробнее настройка задач описывается в п. 7.7. В выпадающем меню следует выбрать задачу, которая автоматически добавится в проект после создания тренда – **TrendRecordingTask**.

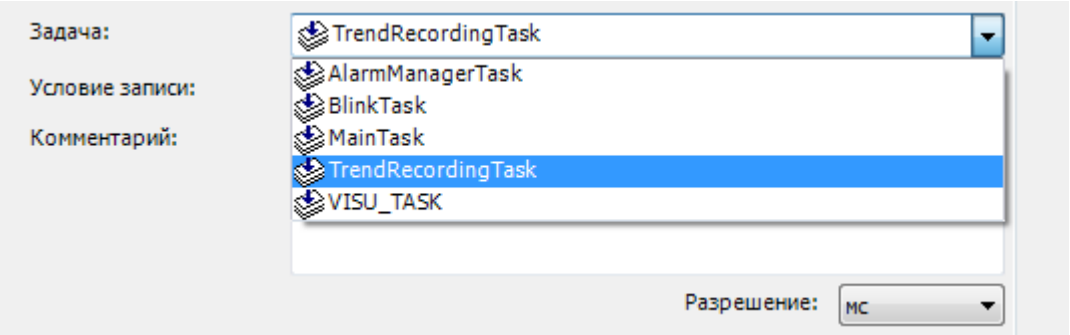


Рисунок 7.125 – Выбор задачи тренда

Вкладка **Trend Storage** позволяет настроить **параметры архивации** тренда – число записываемых переменных, размер файла записи и т. д.

Затем следует добавить переменные, которые будут отображаться на тренде, нажатием кнопки **Добавить переменную**:

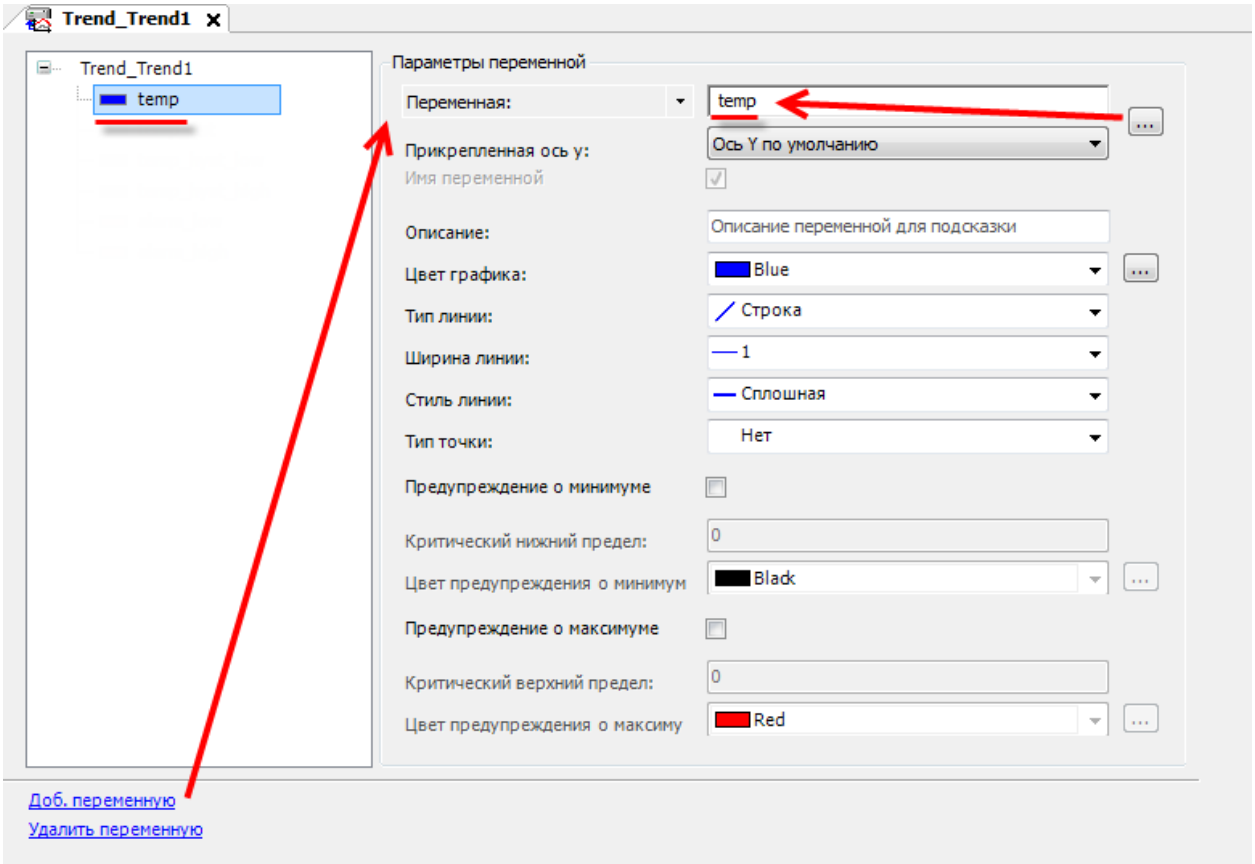


Рисунок 7.126 – Добавление переменной на тренд

Аналогичным образом следует добавляются переменные **temp_ust**, **temp_hyst_low**, **temp_hyst_high**, **alarm_low, alarm_high**. Переменным следует задать различные цвета, границам гистерезиса указать **Тип точки** cross.

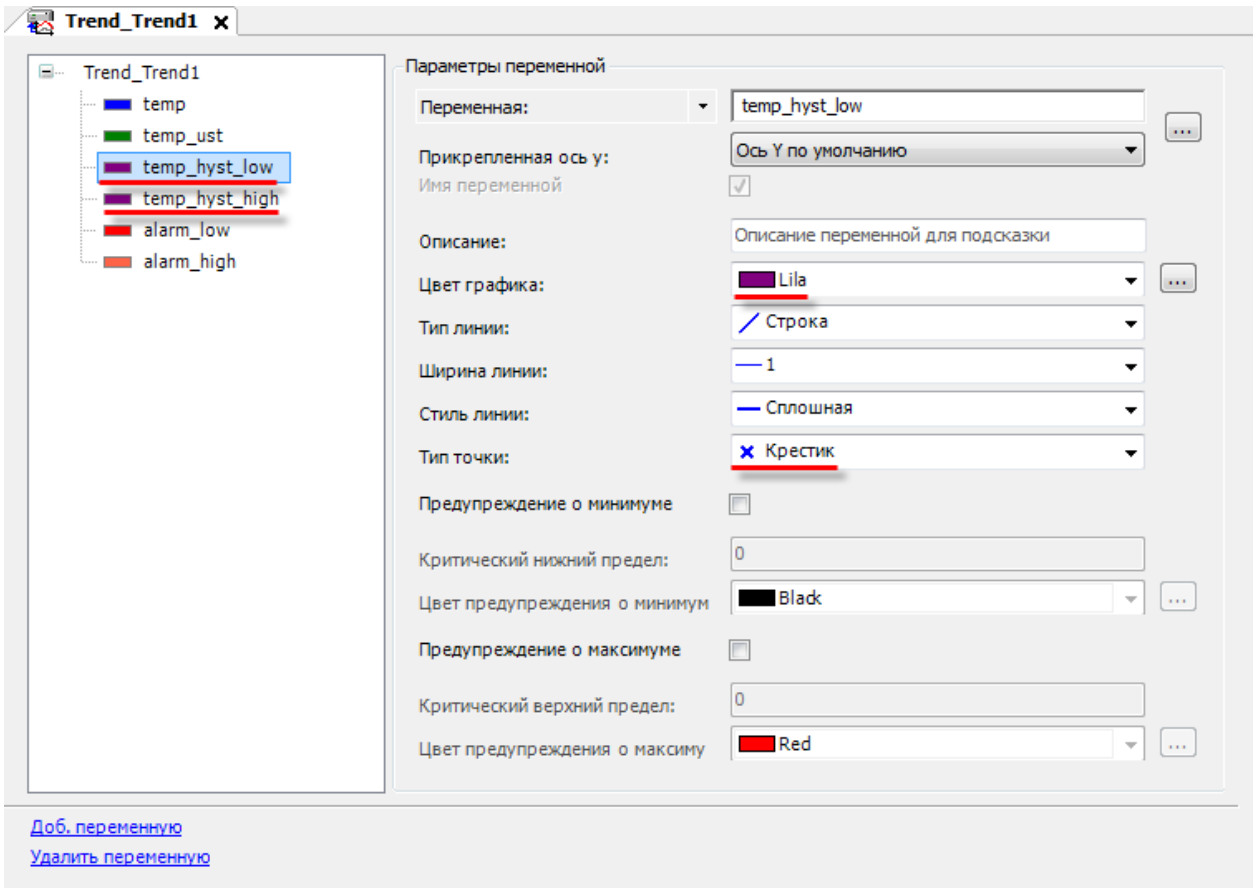


Рисунок 7.127 – Сконфигурированный тренд

Так как автоматически созданные дополнительные элементы (легенда, селектор времени, селектор даты – см. п. 7.3.3) настройки не требуют, то на этом конфигурирование тренда завершено.

7.5.3 Экран Alarm_log

Таблица тревог не требует привязки переменных – для ее работы следует добавить и настроить компонент **Конфигуратор тревог** (см. п. 7.6). Автоматически созданные кнопки (**Подтвердить**, **История** и т. д.) настройки не требуют.

7.6 Настройка конфигуратора тревог

7.6.1 Добавление Конфигуратора тревог в проект

В случае необходимости ведения в проекте **Журнала событий** (частным случаем которого является **Журнал тревог**) используется компонент **Конфигуратор тревог**, который обеспечивает работу графических примитивов **Таблица тревог** и **Баннер тревог**.

В рамках примера будут использоваться тревоги для отображения информационных сообщений в **Таблице тревог** в случае выхода значения температуры за **границы гистерезиса** и **аварийные уставки**.

Следует добавить в проект компонент **Конфигуратор тревог** (его добавление приведет также к автоматическому созданию соответствующей **задачи**):

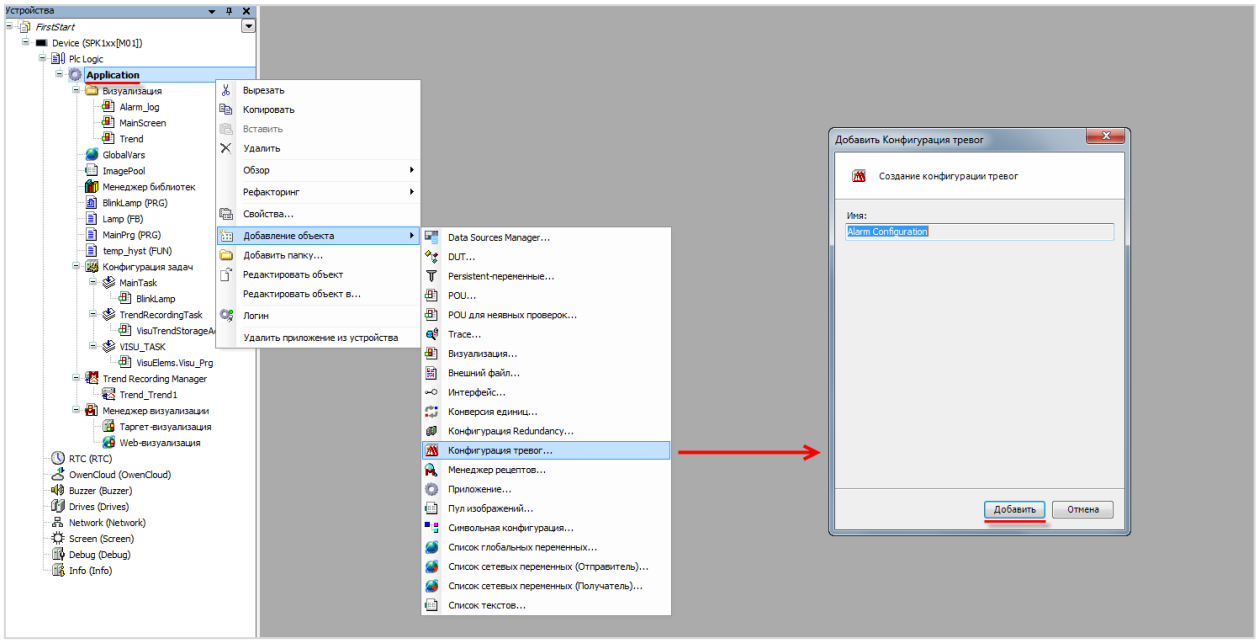


Рисунок 7.128 – Добавление в проект Конфигуратора тревог

По умолчанию **Конфигуратор тревог** содержит **четыре дочерних компонента**: **три класса тревог** (Error, Info, Warning) и **хранилище тревог** Alarm Storage. Следует еще один дочерний компонент – **группу тревог** с названием **AlarmGroup1**. При ее добавлении ***автоматически*** создастся **список текстов** с идентичным названием.

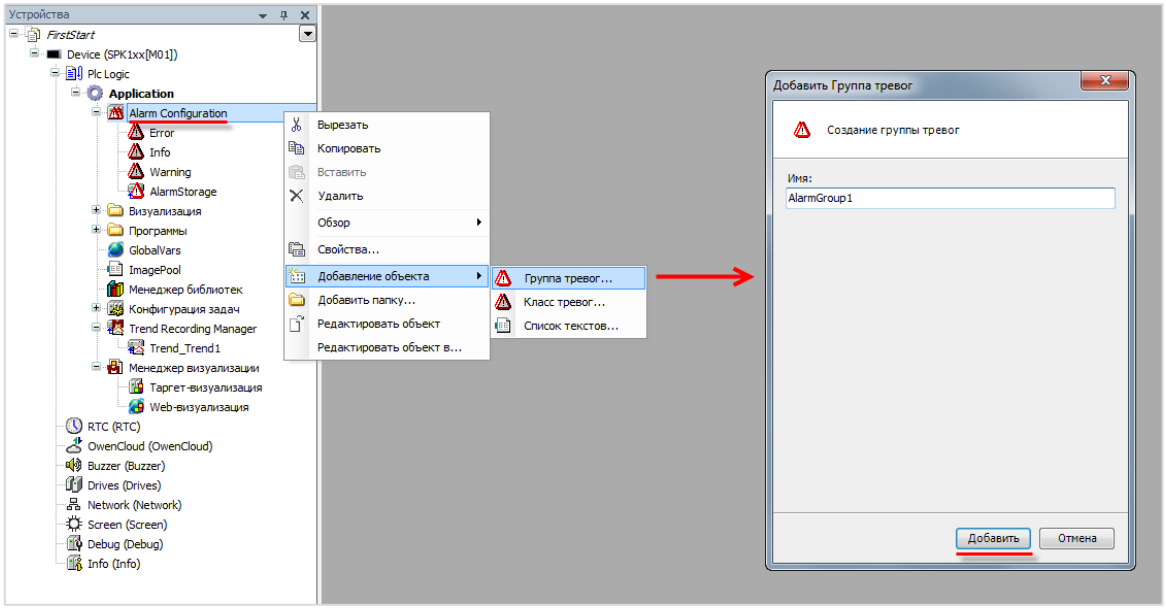


Рисунок 7.129 – Добавление группы тревог AlarmGroup1

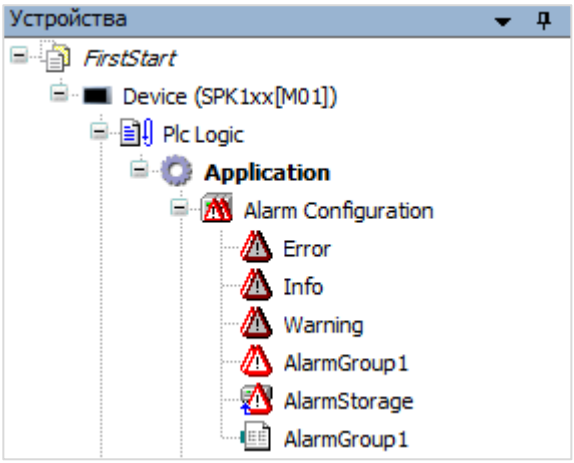


Рисунок 7.130 – Внешний вид Конфигуратора тревог в дереве проекта

Дочерние компоненты **Конфигуратора тревог**:

1. **Классы тревог** предназначены для разделения тревог на отдельные типы. Они определяют набор базовых параметров (таких, как приоритет, способ подтверждения и т. д.), который может соответствовать как одной, так и нескольким тревогам. Пользователь может создавать собственные классы тревог, помимо имеющихся по умолчанию.
2. **Группы тревог** представляют собой наборы тревог, которые могут относиться как к одному, так и к разным классам. На уровне группы настраиваются индивидуальные параметры тревоги – условия появления и пропадания, текст сообщения и т. д.
3. **Список текстов** содержит тексты сообщений тревог и их идентификаторы.
4. **Хранилище тревог** обеспечивает их запись в базу данных.

7.6.2 Настройка классов тревог

Настройки **класса тревог** на примере созданного по умолчанию класса **Error:**

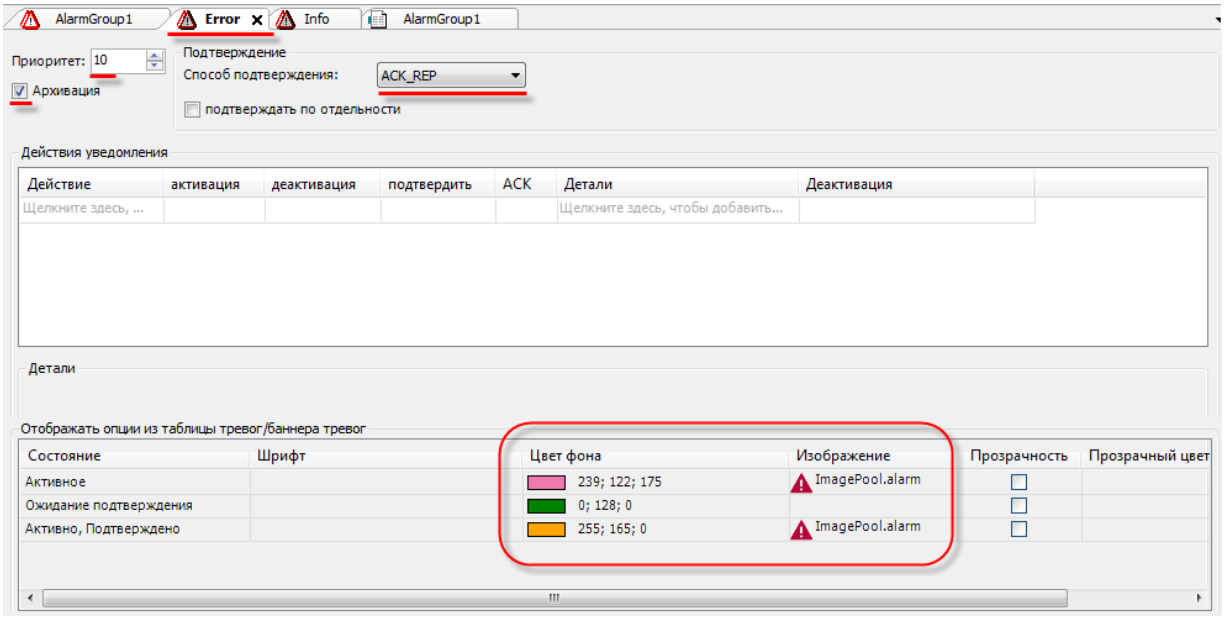


Рисунок 7.131 – Настройки класса тревог Error

1. **Приоритет** характеризует степень важности тревоги. Это информационный параметр, который может быть отображен в **Таблице тревог**. Самый высокий приоритет – 0, самый низкий – 255.
2. **Способ подтверждения** влияет на условие пропадания тревоги из **Таблицы тревог**. Подтверждение подразумевает нажатие пользователем на кнопку **Подтвердить** или **Подтвердить всё**.

Таблица 7.6 – Способы подтверждения тревог

Способ подтверждения тревоги	Возможные состояния тревоги	Условие исчезновения тревожного сообщения
REP	Активное	Условие появления тревоги перестало выполняться
ACK	Активное	Тревога подтверждена пользователем
REP_ACK	Активное, Ожидание подтверждения	Условие появления тревоги перестало выполняться, ***после чего*** она была подтверждена пользователем
ACK_REP	Активное, Ожидание Подтверждения, Подтвержденное	Условие появления тревоги перестало выполняться, и она была подтверждена пользователем (порядок этих событий не имеет значения)
ACK_REP_ACK	Активное, Ожидание Подтверждения, Подтвержденное	Условие появления тревоги перестало выполняться, после чего она была подтверждена пользователем **или** активная тревога была подтверждена пользователем, после чего условие ее появления перестало выполняться, что было также подтверждено пользователем

Если установлена галочка **Подтверждать по отдельности**, то пользователь не может подтвердить сразу все тревоги.

3. Чекбокс **Архивация** отвечает за сохранение тревог в памяти контроллера.
4. Меню **Действие уведомления** позволяет задать список действий, которые выполняются в случае смены состояния тревоги. Количество состояний тревоги зависит от способа ее подтверждения.

5. Меню **Отображать опции** позволяет настроить отображение тревог в **Таблице/Баннере тревог**, в частности, задать ее состояниям индивидуальный шрифт, цвет фона, пиктограмму и т. д. Для задания пиктограммы следует кликнуть на соответствующее поле, ввести имя пиктограммы (любое), после чего нажать на Enter и указать путь к соответствующему изображению (поддерживаются все распространенные графические формы типа .jpg, .png, .bmp и т. д.). В случае необходимости использовать эту же пиктограмму во второй раз (для другого состояния или класса), достаточно просто указать ее имя. По умолчанию пиктограммы тревог добавляются в **Глобальный пул изображений**, расположенный на **Панели POU**.

В рамках примера будет использоваться класс **Error** для тревог о выходе температуры за **аварийные уставки** и **Info** – за **границы гистерезиса**. Их настройки приведены на рисунке 7.131 и рисунке ниже.

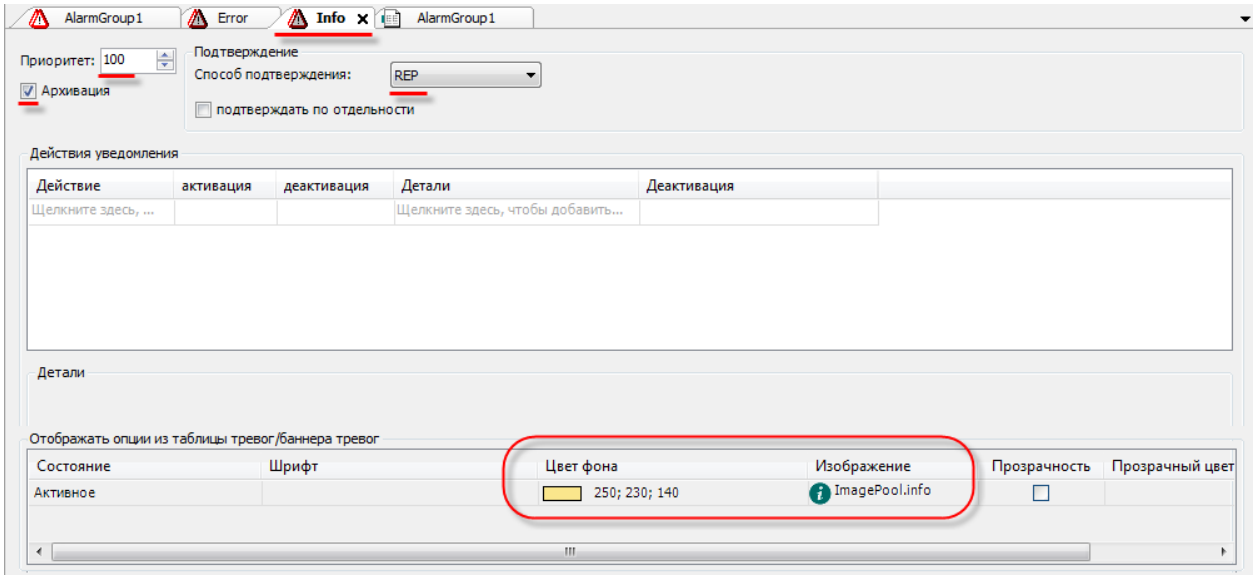


Рисунок 7.132 – Настройки класса тревог Info

7.6.3 Создание группы тревог

Настройки для группы тревог **AlarmGroup1** из п. 7.6.1:

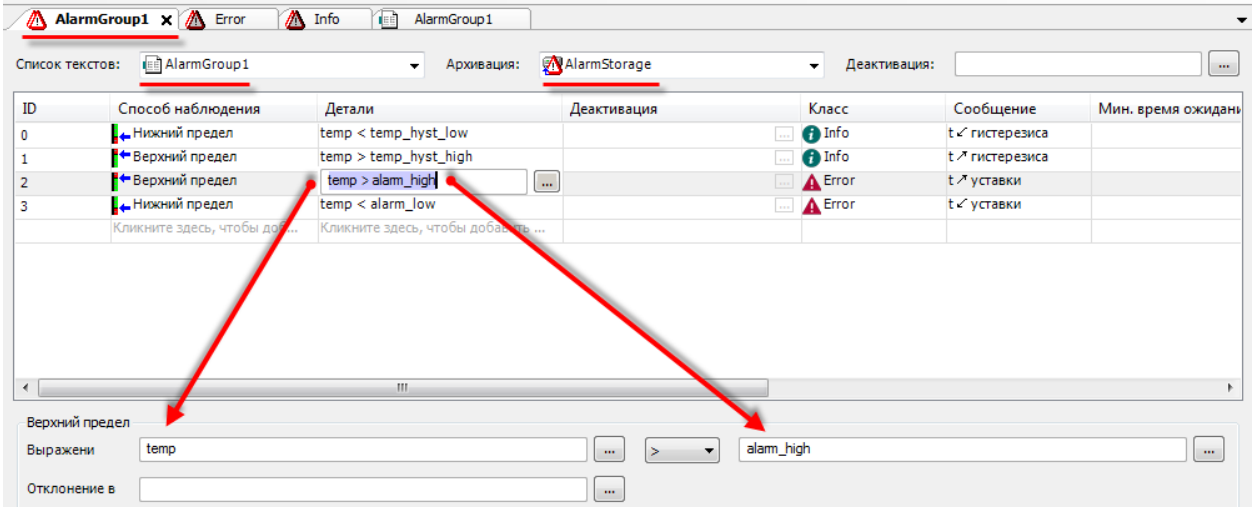


Рисунок 7.133 – Настройка группы тревог AlarmGroup1

Для группы можно указать используемый ею **Список текстов**, **Хранилище** и логическую переменную в поле **Деактивация**, которая включает (при значении TRUE) или отключает (при значении FALSE) возможность активации тревог этой группы.

Добавить новую тревогу можно двойным нажатием **ЛКМ** по ячейке столбца **Способ наблюдения**. После того, как будет выбран способ наблюдения для новой тревоги, автоматически сформируется ее **ID** и появятся подсказки по заполнению оставшихся полей. Поля, еще не заполненные надлежащим образом, помечаются иконкой .

Тревога обладает следующими настройками:

1. **ID** – идентификатор, который автоматически присваивается тревоге при ее создании. ID соответствует номеру, используемому в списке текстов тревог. Его можно изменить, но он должен оставаться уникальным в группе тревог. В случае изменения ID в таблице, он автоматически обновляется и в списке текстов, и наоборот.
2. **Способ наблюдения** определяет условие появления тревог. Описание доступных способов приведено в таблице 7.7:

Таблица 7.7 – Способы наблюдения тревоги

Способ наблюдения тревоги	Возможные настройки
Дискретный	**Выражение**: в левой части вводится наблюдаемое выражение, в правой части – выражение, с которым следует его сравнить (предел); посередине выбирается нужный оператор сравнения (= или <>)
Верхний предел	**Выражение**: то же, что и для типа «Дискретный» (см. выше), но с операторами сравнения > или >= и возможным параметром **Отклонение в %**[1]
Нижний предел	**Выражение**: то же, что и для типа «Дискретный» (см. выше), но с операторами сравнения < или <= и возможным параметром **Отклонение в %**[1]
Внутренний диапазон	**Выражение**: вводится наблюдаемое выражение. **Область**: тревожная ситуация возникнет, как только наблюдаемое выражение примет значение из заданного диапазона. Слева вводится выражение, определяющее нижнюю границу области, справа – верхнюю границу. Наблюдаемое выражение будет отображено в поле посередине. Следует выбрать нужные операторы сравнения и по желанию задать **Отклонение в %**[1]
Внешний диапазон	**Выражение**: вводится наблюдаемое выражение. **Область**: тревожная ситуация возникнет, как только наблюдаемое выражение примет значение вне заданного диапазона. Слева вводится выражение, определяющее нижнюю границу области, справа – верхнюю границу. Наблюдаемое выражение будет отображено в поле посередине. Следует выбрать нужные операторы сравнения и по желанию задать **Отклонение в %**[1]
Изменение	**Выражение**: вводится наблюдаемое выражение. Тревожная ситуация возникнет, как только его значение изменится.
Событие	Состояние тревоги переключается через приложение с помощью функций библиотеки **AlarmManager**

[1] **Отклонение в %**: если задан этот параметр, то тревожная ситуация будет иметь место до тех пор, пока не будет достигнуто определенное отклонение от указанного предельного значения. Размер отклонения задается в процентах (%) от предельного значения.

Пример: Верхний предел: «i_temp >= 30», Отклонение: «10%». Как только значение переменной i_temp достигнет или превысит 30, возникнет тревожная ситуация. Пока значение не упадет до 27, сообщение о тревоге будет сохраняться

3. **Детали** отображают текущую конфигурацию тревоги.
4. **Деактивация** позволяет указать переменную, значение которой будет влиять на исчезновение **данной** тревоги.
5. В столбце **Класс** указывается **класс тревог**, к которому принадлежит данная тревога.

6. В столбце **Сообщение** можно указать текст сообщения, которое будет отображаться в **Таблице тревог** в случае активации тревоги. Данный текст автоматически добавляется в **Список текстов тревог**, заданный для группы. Помимо фиксированного текста, можно использовать следующие заполнители:

Таблица 7.8 – Заполнители сообщений Таблицы тревог

Заместитель	Значение
DATE	Дата перехода тревоги в текущее состояние
TIME	Время последнего изменения состояния тревоги
EXPRESSION	Выражение (задается в настройках тревоги), переключившее тревогу из одного состояния в другое
PRIORITY	Приоритет тревоги (задается в настройках класса тревог)
TRIGGERVALUE[1]	Значение, вызвавшее тревогу
ALARMID	ID тревоги (задается в настройках тревоги)
CLASS	Имя класса тревог (задается в настройках тревоги)
CURRENTVALUE[1]	Текущее значение наблюдаемой переменной
LATCH1, LATCH2[1]	Значение первой и второй триггерной переменной (см. ниже)
ALARM	TRUE, если состояние тревоги «активное», FALSE в любом другом случае
STATE	Состояние тревоги: 0 – «отсутствие тревоги», 1 – «активное», 2 – «ожидание подтверждения», 3 – «активное, подтверждено»
[1] Для TRIGGERVALUE, CURRENTVALUE, LATCH1 и LATCH2 можно также использовать форматирование, например, CURRENTVALUE %2.2f	

7. **Мин. время ожидания** определяет время задержки активации тревоги.
8. **Первая и вторая триггерная** переменная используются в случае необходимости записи значений во время активации тревоги. Триггерная переменная не должна быть массивом (в т. ч. строкой).
9. В последнем столбце можно указать существующую **тревогу с более высоким приоритетом** по отношению к создаваемой. В этом случае при одновременном наступлении двух тревог будет автоматически подтверждаться текущая. Это позволяет выполнять двухэтапную обработку тревоги, так как тревога с меньшим приоритетом перекрывается тревогой с большим приоритетом.

Пример для системы контроля температуры: задается тревога с меньшим приоритетом (например, 10) для ***предупреждения*** о температуре, превышающей 30 °C. Предварительно была задана другая тревога с более высоким приоритетом (например, 1), которая срабатывает при достижении температуры 50 °C (***критическое состояние***). Данную критическую тревогу можно ввести в конфигурации ***предупреждающей*** тревоги как «Тревогу с большим приоритетом». Существующая ***предупреждающая*** тревога будет автоматически подтверждаться в случае срабатывании ***критической*** тревоги.

Следует настроить группу тревог **AlarmGroup1** в соответствии с рисунком 7.133.

7.6.4 Хранилище тревог и Список текстов

Компонент **Хранилище тревог** содержит настройки хранения файла тревог.

Файл, в который записывается информация о тревогах, хранится во **внутренней** памяти контроллера (возможность записи на USB накопитель ***отсутствует***). Пользователь не может изменять его имя, поскольку оно автоматически формируется из имени приложения следующим образом: **<имя приложения>.<имя хранилища тревог>.sqlite**. Использование файла записи определяется для классов (галочка **Архивация**) и групп тревог (указание **Хранилища**).

Хранилище тревог имеет следующие настройки:

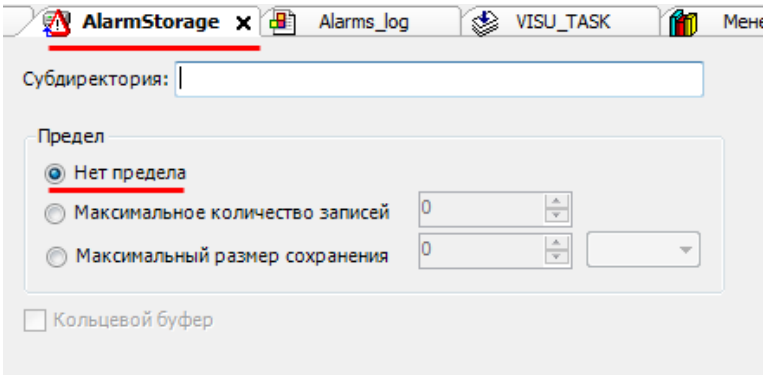


Рисунок 7.134 – Внешний вид Хранилища тревог

1. **Субдиректория** – это папка в рабочей директории ПЛК, в которой будет сохраняться история тревог.
2. **Предел** позволяет ограничить количество записей в базе данных – по максимальному количеству или по размеру файла записи. В случае превышения ограничения старые записи начнут удаляться (в режиме кольцевого буфера).

В случае выполнения **заводского сброса** файл записи удаляется.

Список текстов тревог представляет собой набор текстовых сообщений, используемых для отображения в **Таблице тревог**.

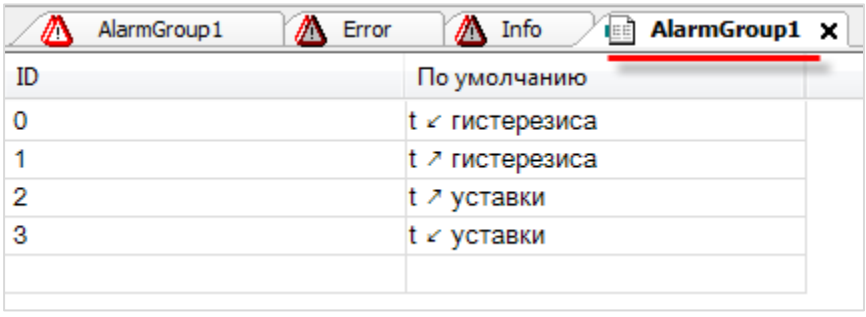


Рисунок 7.135 – Внешний вид Списка текстов тревог

7.7 Настройка задач

Задача определяет **приоритет** компонента, **тип вызова** и его настройки. Компонент **Конфигурация задач** автоматически добавляется в проект при его создании.

Пример должен содержать четыре задачи:

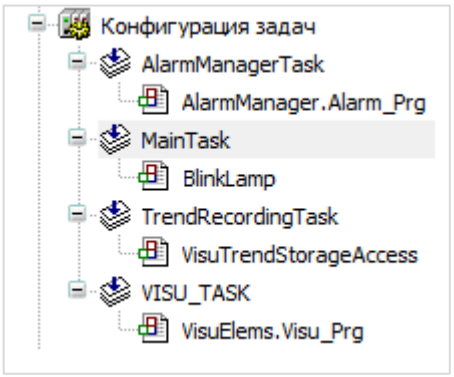


Рисунок 7.136 – Компонент Конфигурация задач

1. **AlarmManagerTask** была автоматически создана во время добавления в проект **Конфигуратора тревог** (см. п. 7.6).
2. **MainTask** была автоматически создана вместе с проектом и содержала программу PLC_PRG, которая была переименована в **BlinkLamp** (см. п. 7.4.3).
3. **TrendRecordingTask** была автоматически создана во время добавления в визуализацию элемента **Тренд** (см. п. 7.3.3).
4. **Visu_Task** была автоматически создана во время добавления в проект первой визуализации (см. п. 6).

Задачу **MainTask** следует переименовать (с помощью двойного нажатия **ЛКМ** на название задачи или одиночного нажатия **ПКМ** и выбора пункта Refactoring) в **BlinkTask**. Затем следует создать новую задачу с названием **MainTask**:

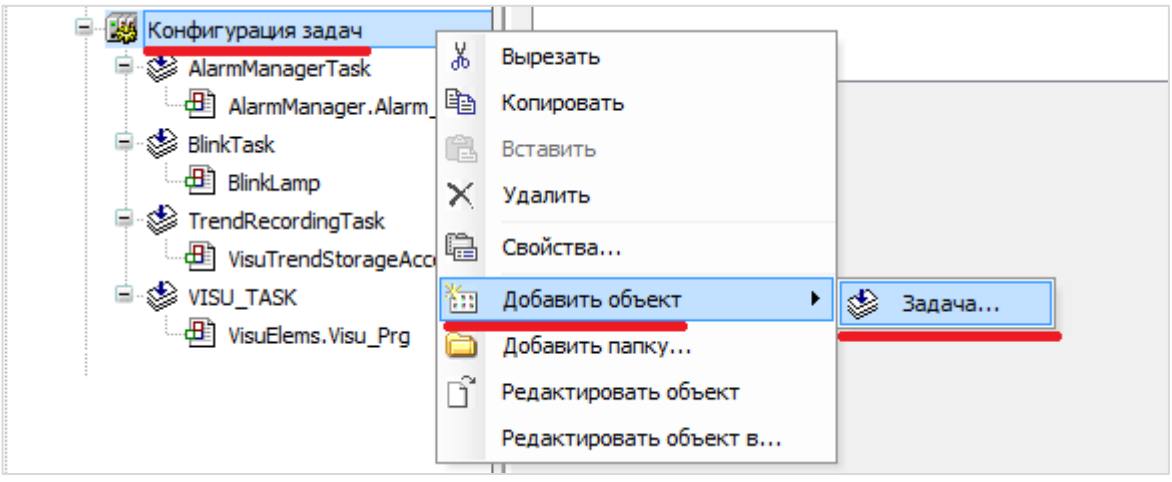


Рисунок 7.137 – Создание новой задачи

Далее следует выбрать программу **MainPrg**, зажать **ЛКМ** и, ***не отпуская ее***, перетащить программу под соответствующую задачу.

В итоге компонент **Конфигурация задач** будет выглядеть следующим образом:

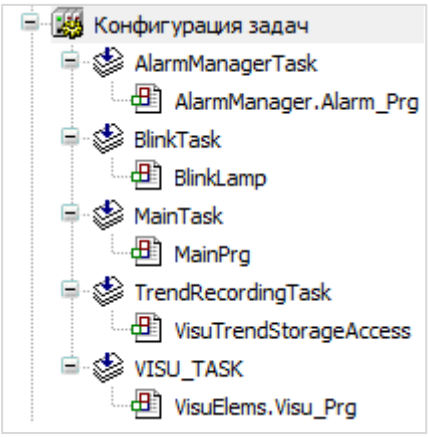


Рисунок 7.138 – Компонент Конфигурация задач после окончательной настройки

Настройки задачи на примере **BlinkTask**:

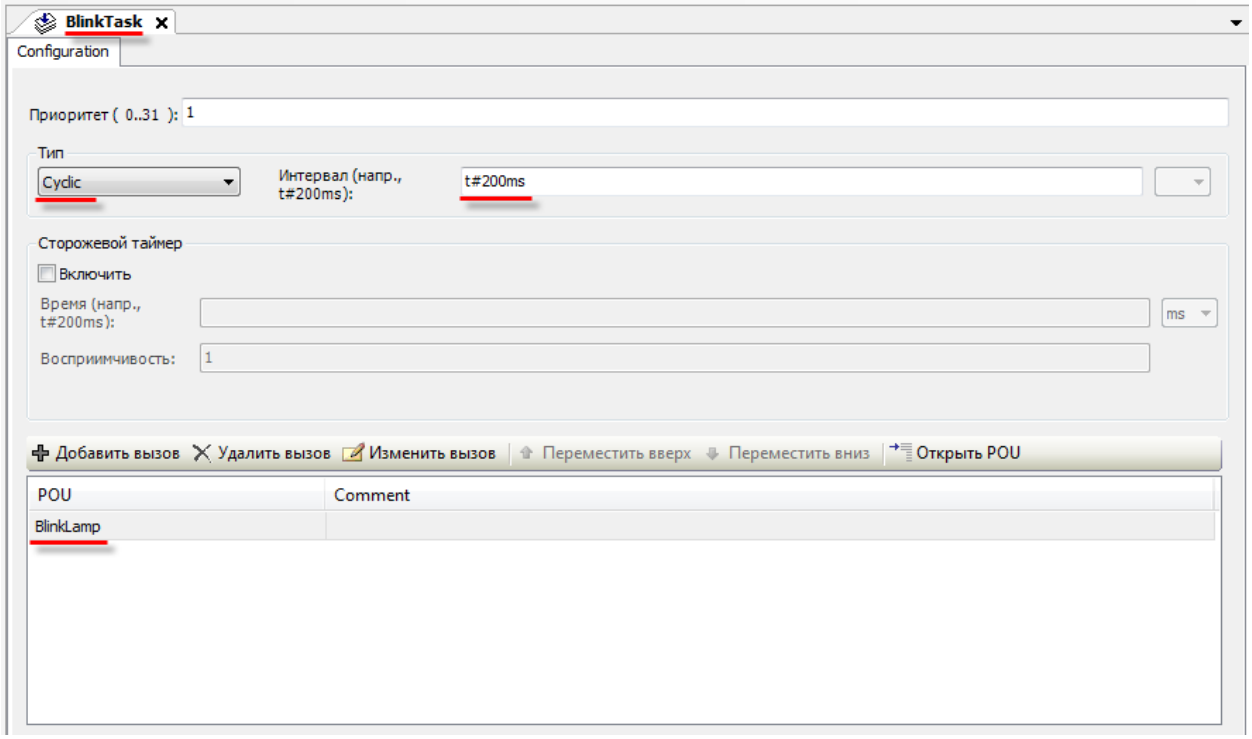


Рисунок 7.139 – Настройки задачи BlinkTask

1. **Приоритет** характеризует порядок выполнения задачи. При прочих равных условиях первой будет выполняться задача с меньшим приоритетом. Самый низкий приоритет – **0**, самый высокий – **31**.

2. **Тип** определяет условие вызова задачи:

Таблица 7.9 – Типы выполнения задач

Тип	Условие вызова задачи
Циклическая	Задача вызывается циклически через заданный интервал времени (интервал определяет время между вызовами задачи)
Свободное выполнение	Задача вызывается во время, свободное от выполнения других задач
Статус	Задача вызывается в режиме свободного выполнения, если логическая переменная, заданная в поле **Событие**, имеет значение TRUE
Событие	Задача однократно вызывается, если логическая переменная, заданная в поле **Событие**, меняет свое значение с FALSE на TRUE (выполнение происходит по переднему фронту)

3. **Сторожевой таймер** (watchdog) позволяет сгенерировать исключение, если время выполнения задачи превысило заданный допустимый предел.

4. **Меню вызова POU** определяет список компонентов, исполняемых в рамках данной задачи.

Настройки задач **BlinkTask** и **MainTask** приведены на рисунках 7.139 и 7.140.
В рамках примера для задачи **BlinkTask** используется интервал цикла, равный **200 мс**.

Для задачи **MainTask** используется интервал, равный **одной секунде**, т. к. в программе **MainPrg**, связанной с этой задачей, реализована эмуляция изменения температуры – т. е. во время работы регулятора значение температуры будет изменяться на 3 градуса Цельсия каждый цикл. Чтобы сделать процесс изменения температуры более наглядным, следует задать соответствующее время цикла.

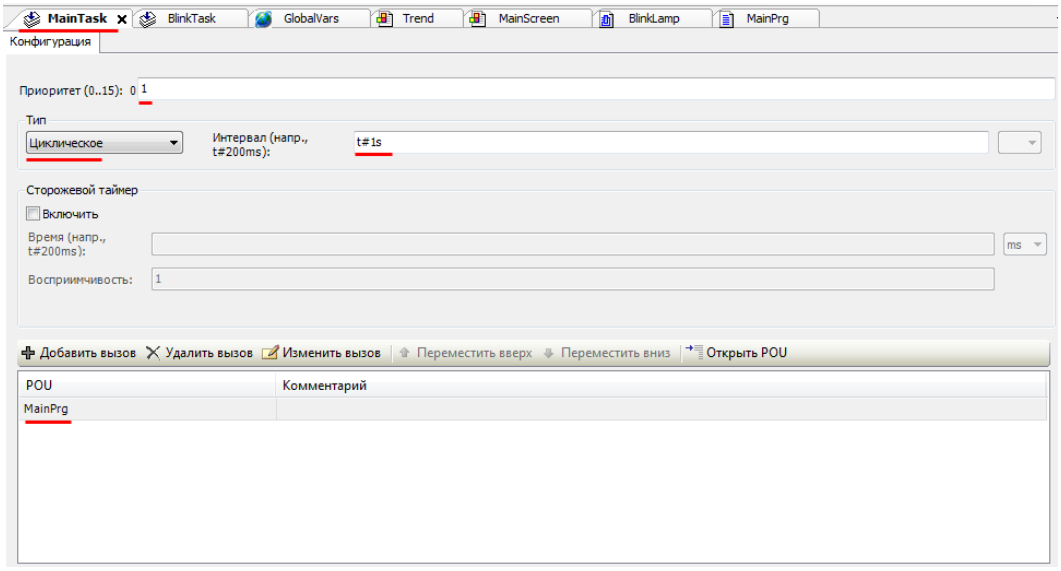
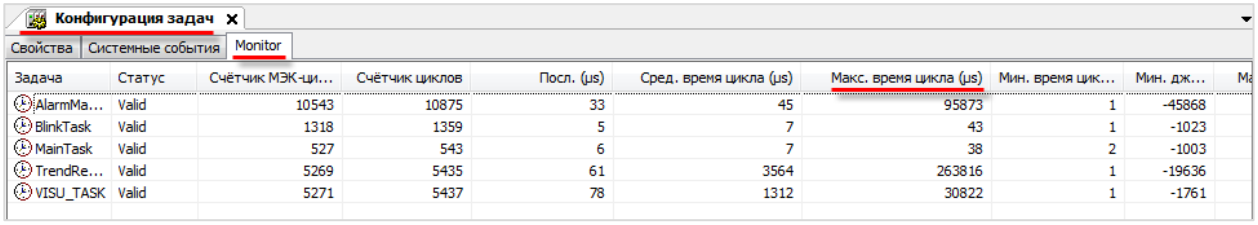


Рисунок 7.140 – Настройки задачи MainTask

Во время исполнения проекта в ПЛК становится активной (заполняется данными) вкладка **Монитор** компонента **Конфигурация задач**.



Задача	Статус	Счётчик МЭК-ци…	Счётчик циклов	Посл. (µs)	Сред. время цикла (µs)	Макс. время цикла (µs)	Мин. время цик…	Мин. дж…	Ма
AlarmMa…	Valid	10543	10875	33	45	95873	1	-45868	
BlinkTask	Valid	1318	1359	5	7	43	1	-1023	
MainTask	Valid	527	543	6	7	38	2	-1003	
TrendRe…	Valid	5269	5435	61	3564	263816	1	-19636	
VISU_TASK	Valid	5271	5437	78	1312	30822	1	-1761	

Рисунок 7.141 – Вкладка Монитор Конфигурации задач

Вкладка содержит статистическую информацию о работе проекта. В частности, рекомендуется обратить внимание на параметр **Макс. время цикла (**отображается в **микро**секундах**)** – его значение ***не должно превышать*** время цикла, заданное для конкретной задачи (если задача исполняется циклически). В противном случае рекомендуется увеличить время цикла задачи или «облегчить» содержимое компонентов, привязанных к ней.

7.8 Настройка обмена данными по протоколу Modbus RTU

Контроллеры ОВЕН позволяют подключать к себе различные устройства по протоколам **ModBus** (RTU, ASCII, TCP), ОВЕН и нестандартным протоколам, реализованным пользователем.

В данном примере рассматривается подключение к контроллеру модулей ввода-вывода сигналов **ОВЕН Mx110**: MB110-8A (модуль аналогового ввода, используемый для получения значения температуры с датчика) и МУ110-8Р (модуль дискретного вывода, используемый для формирования сигналов управления кондиционером) по протоколу **Modbus RTU**.

7.8.1 Конфигурирование и подключение модулей

В первую очередь следует подключить модули к ПК и настроить с помощью программы Конфигуратор Mx110. Для настройки потребуются следующие документы:

1. Руководство по эксплуатации MB110-8A.
2. Руководство по эксплуатации МУ110-8Р.
3. Руководство пользователя программы Конфигуратор Mx110.

С помощью конфигуратора модулям задаются следующие настройки:

Таблица 7.10 – Сетевые настройки модулей Mx110

Параметр	MB110-8A	МУ110-8Р
Скорость обмена	115200	
Длина слова данных	8	
Четность	Отсутствует	
Количество стоп-бит	1	
Базовый адрес прибора	1	2

Будем считать, что сигнал с датчика температуры заведен на первый вход модуля **MB110-8A**, а сигналы управления кондиционером – на первый и второй дискретные выходы модуля **МУ110-8Р**.

В рамках примера оба модуля подключаются к порту **COM1 (RS-485-1)** контроллера СПК1xx [M01].

7.8.2 Установка шаблонов модулей в среду CODESYS

Модули имеют шаблоны для **CODESYS**, облегчающие процесс настройки опроса. Для работы с шаблонами следует установить в среду программирования пакет **Mx110_drivers_3.5.11.1** (или его более свежую версию).

Пакет доступен на диске с ПО из комплекта комплект поставки и на сайте компании ОВЕН в разделе **CODESYS V3/Библиотеки**.

Для установки пакета в **CODESYS** в меню **Инструменты** следует выбрать пункт **Менеджер пакетов**, после чего указать путь к файлу пакета и нажать кнопку **Установить**:

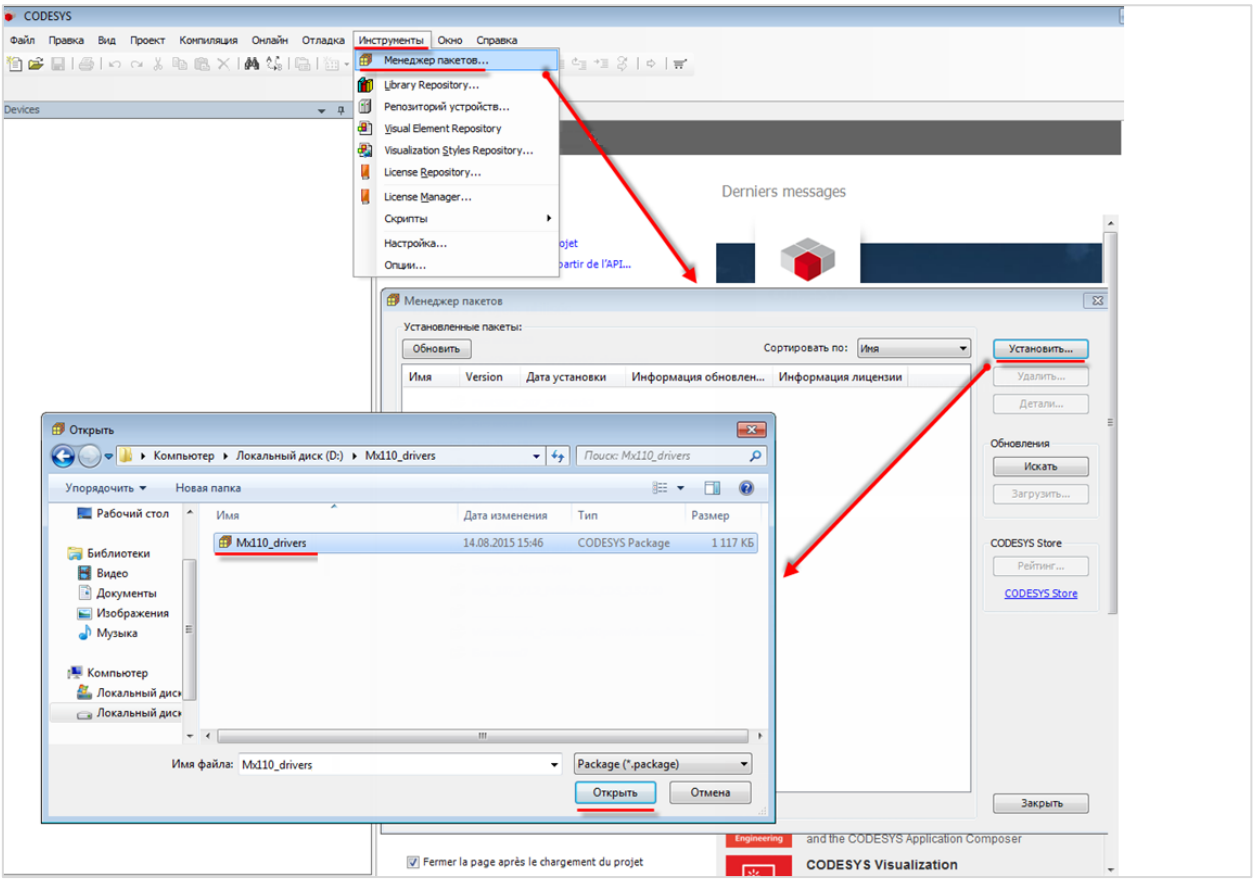


Рисунок 7.142 – Установка пакета Mx110_drivers в среду CODESYS

7.8.3 Добавление и настройка шаблонов

Чтобы добавить в проект устройство **Modbus COM** следует нажать **ПКМ** на компонент **Device** и перейти во вкладку **Modbus/Порт Modbus Serial**.

ПРИМЕЧАНИЕ
Версия компонента не должна превышать версию таргет-файла.

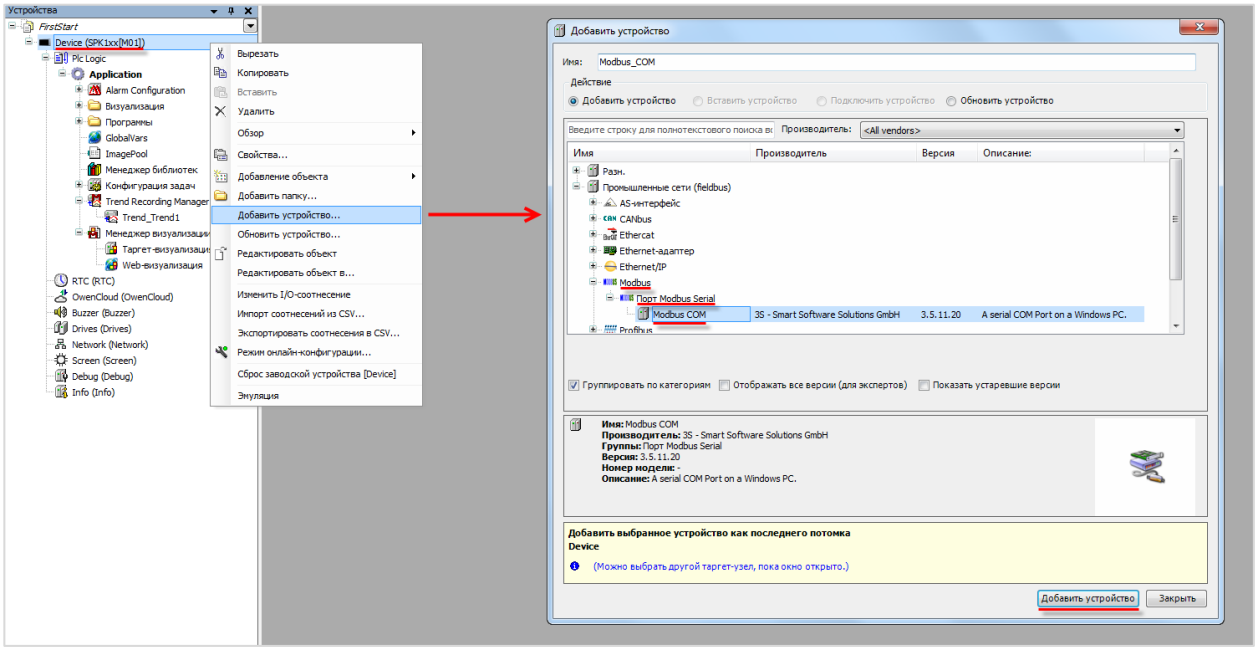


Рисунок 7.143 – Добавление компонента Modbus COM

Компонент должен быть настроен в соответствии с [таблицей 7.10](). Информация о нумерации COM-портов контроллера в CODESYS приведена на вкладке **Информация** узла **Device.**

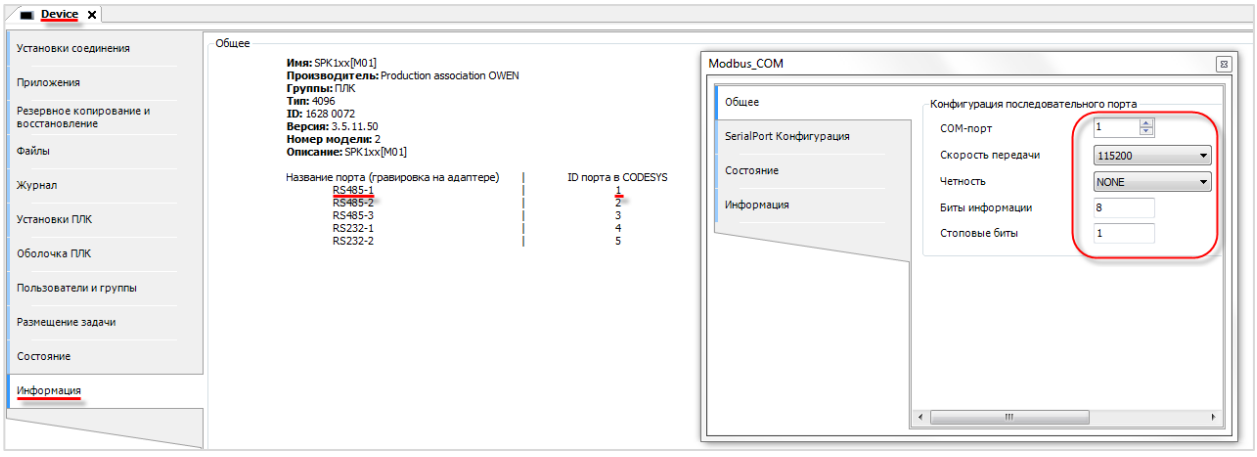


Рисунок 7.144 – Настройки компонента Modbus COM

Чтобы добавить в проект устройство **Modbus Master COM port,** следует нажать **ПКМ** на компонент **Modbus COM**.

ПРИМЕЧАНИЕ

Версия компонента не должна превышать версию таргет-файла.

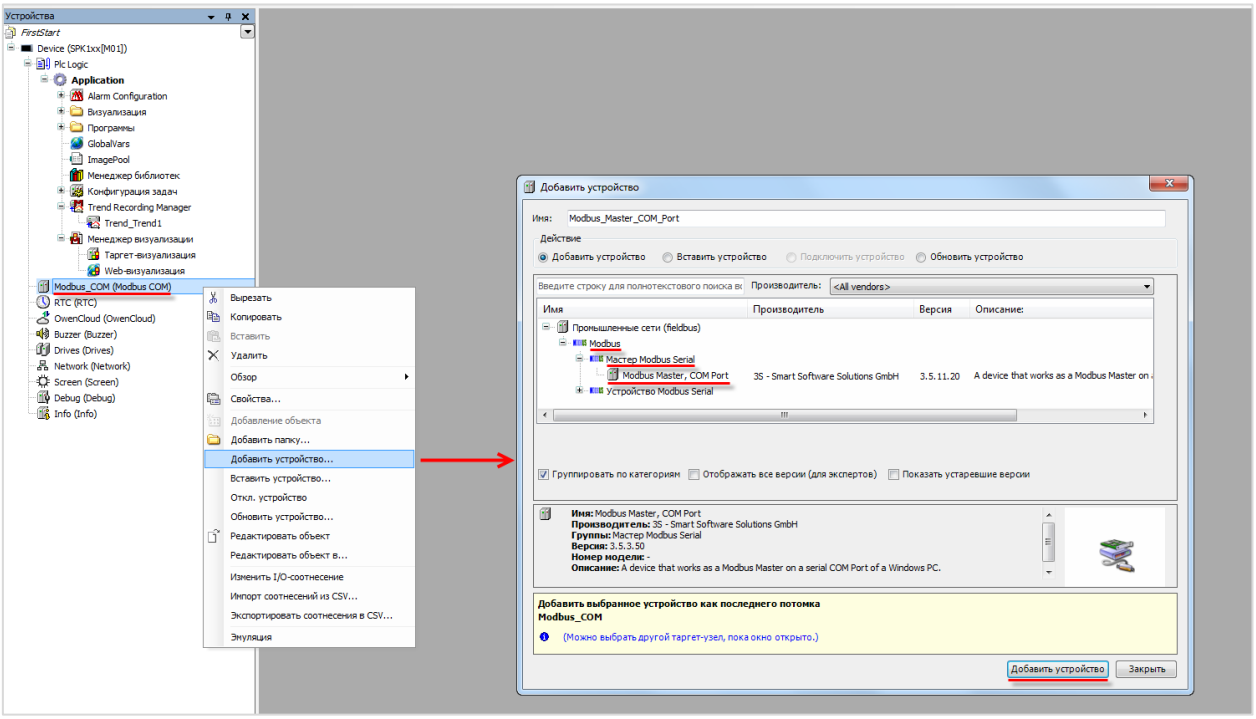


Рисунок 7.145 – Добавление компонента Modbus Master COM port

В настройках компонента следует установить галочку **Автоперезапуск соединения**:

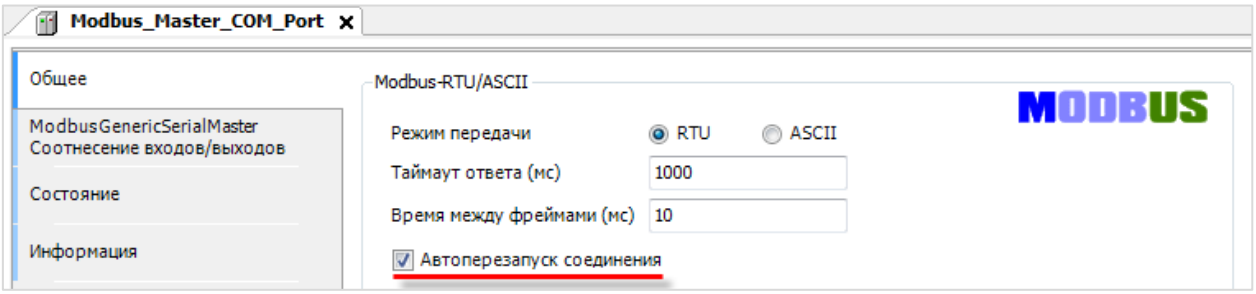


Рисунок 7.146 – Настройки компонента Modbus Master COM port

Чтобы добавить в проект устройства **MV110_8A** и **MU110_8R_K** следует нажать **ПКМ** на компонент **Modbus Master COM port**.

ПРИМЕЧАНИЕ
Версия компонента должна соответствовать версии таргет-файла.

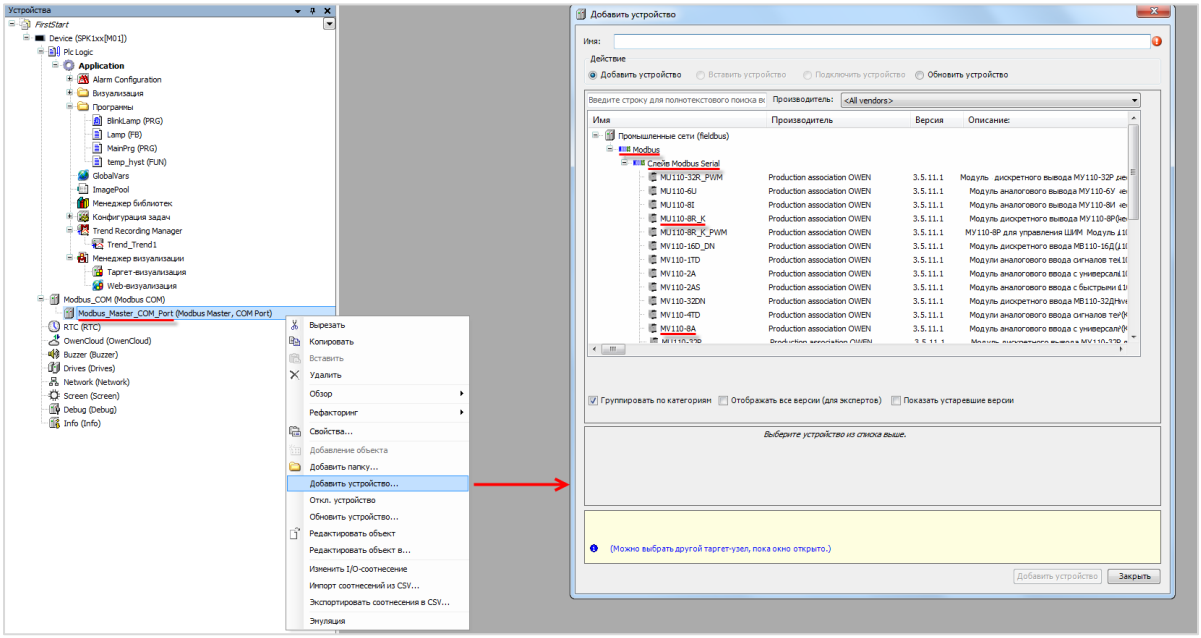


Рисунок 7.147 – Добавление в проект модулей MV110_8A и MU110_8R_K

Модулю **MV110_8A** согласно таблице 7.10 следует задать slave-адрес **1**, модулю **MU110_8R_K** – **2**.

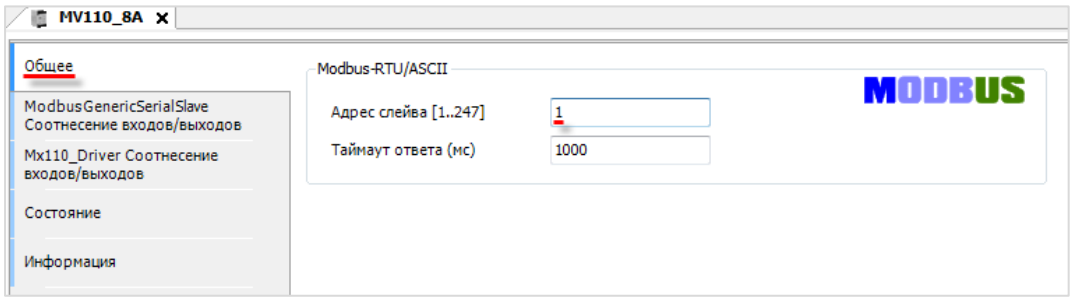


Рисунок 7.148 – Настройки модуля MV110_8A

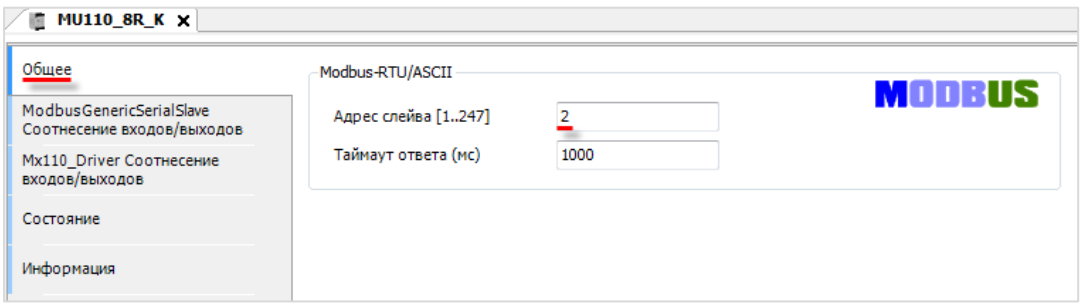


Рисунок 7.149 – Настройки модуля MU110_8R_K

7.8.4 Использование переменных модулей в программе

После добавления и настройки шаблонов модулей, их можно использовать в проекте. В рамках примера сигнал с датчика температуры заведен на первый вход модуля **MB110-8A**, а сигналы управления кондиционером – на первый и второй дискретные выходы модуля **МУ110-8P**.

Согласно созданному проекту значение температуры с датчика должно записываться в переменную **temp_real** программы **MainPRG,** а для управление кондиционером используются переменные **condition_power** (состояние кондиционера: вкл/выкл) и **control_mode** (режим работы кондиционера: охлаждение/нагрев).

В настройках шаблона **MV110_8A** на вкладке **Соотнесение входов/выходов** следует привязать к параметру **Измеренное значение** папки **Вход 1** переменную **temp_real**.

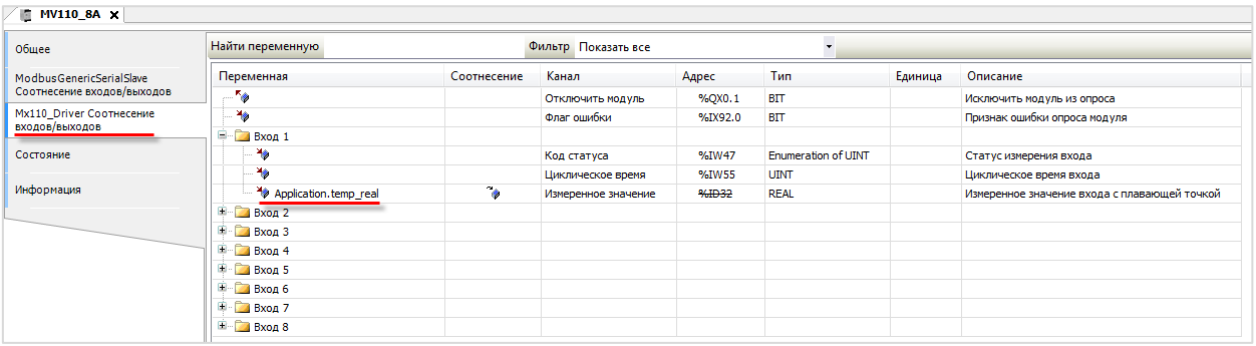


Рисунок 7.150 – Привязка переменной к шаблону MV110_8A

В настройках шаблона **MU110_8R _K** на вкладке **Соотнесение входов/выходов** следует привязать к параметру **Выход 1** переменную **conditioner_power**, а к параметру **Выход 2** – переменную **control_mode**.

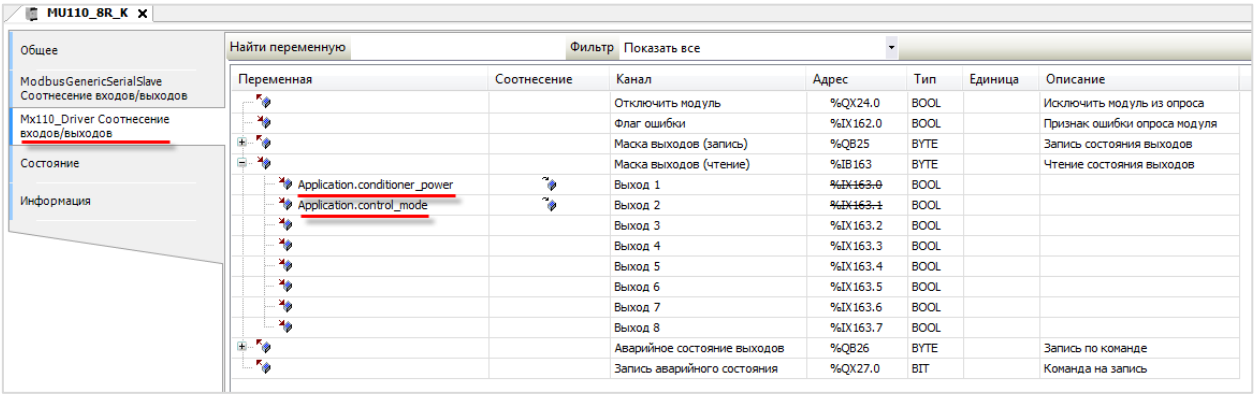


Рисунок 7.151 – Привязка переменных к шаблону MU110_8R_K

ПРИМЕЧАНИЕ

Более подробная информация о настройке обмена приведена в руководстве **CODESYS V3.5. Modbus**.

8 Компиляция и загрузка проекта

8.1 Компиляция проекта

Перед тем, как загрузить готовый проект в контроллер, следует произвести **компиляцию** с помощью соответствующей вкладки на **Панели инструментов**:

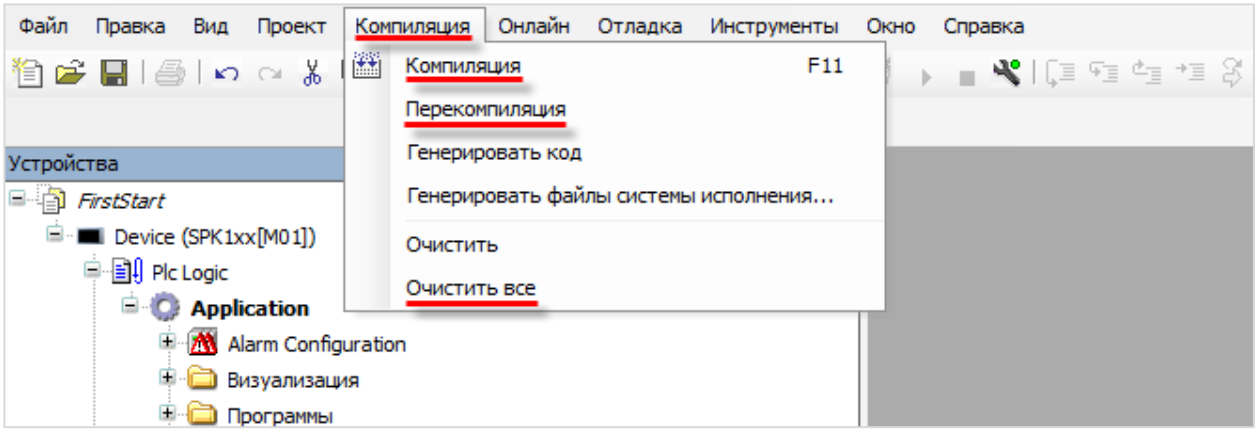


Рисунок 8.1 – Команды вкладки Компиляция

Компиляция представляет собой проверку синтаксиса пользовательского приложения. Команда **Перекомпиляция** позволяет выполнить компиляцию ранее скомпилированного проекта.

После выполнения компиляции в папке, где хранится проект, создаются файлы, содержащие информацию о результатах компиляции. Удалить информацию о результатах предыдущих компиляций можно с помощью команды **Очистить все**.

Перед загрузкой проекта в контроллер (и после любых значительных изменений в проекте) **рекомендуется** выполнять последовательно команды **Очистить все** и **Перекомпиляция**.

Информация об ошибках и предупреждениях, возникших в ходе компиляции, отображается на соответствующей панели:

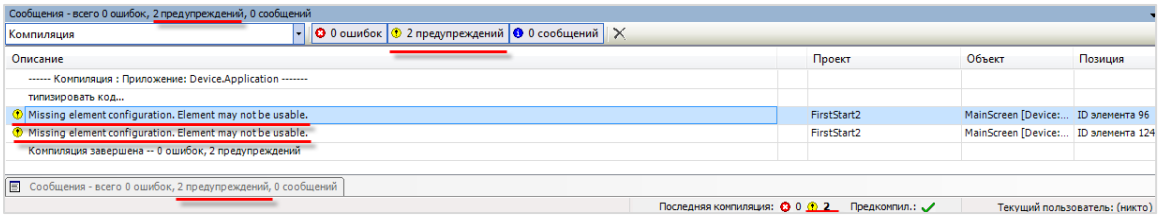


Рисунок 8.2 – Панель сообщений компиляции

На рисунке 8.2 видны два предупреждения – они связаны с тем, что к серым индикаторам на экране **MainScreen** не привязаны переменные (т. к. эти индикаторы не должны светиться).

Проект, скомпилированный с **предупреждениями**, ***можно*** загрузить в контроллер. Проект, скомпилированный с **ошибками,** загрузить в контроллер ***нельзя***.

8.2 Загрузка проекта в контроллер

Загрузить проект можно с ПК, который подключен к контроллеру, или с USB/SD накопителя. Процесс загрузки проекта с накопителя описан в руководстве **CODESYS V3.5. FAQ**.

Для загрузки проект в контроллер с пользовательского ПК должна быть настроена связь между контроллером и компьютером (см. п. 5). Процесс загрузки проекта в **оперативную память** контроллера описывается в п. 6.

ПРИМЕЧАНИЕ
Информация из **оперативной** памяти ***удаляется*** после выключения контроллера. Содержимое **flash-памяти** сохраняется после выключения контроллера.

Для загрузки проекта во flash-память следует:

1. Выбрать в меню **Онлайн** команду **Логин** для подключения к контроллеру:

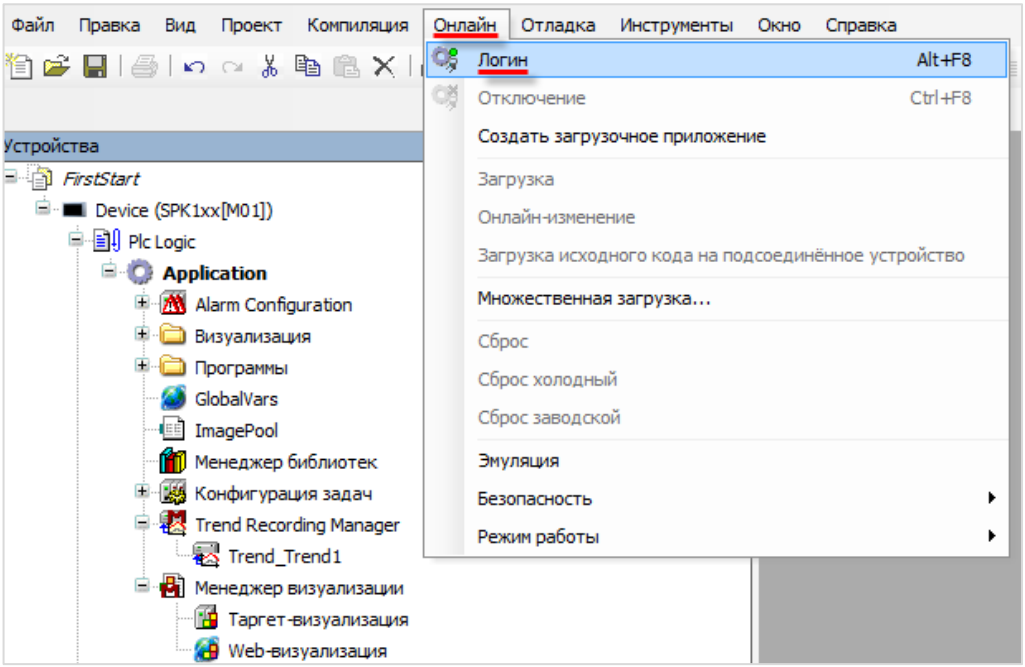


Рисунок 8.3 – Команда Логин (подключение к контроллеру)

2. Так как в контроллер уже загружен пустой проект (см. п. 3), то появится следующее информационное сообщение:

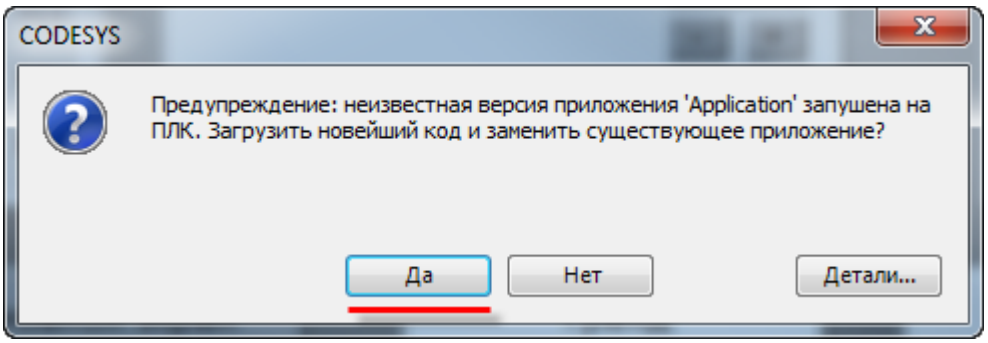


Рисунок 8.4 – Информационное окно загрузки проекта (при наличии в контроллере другого проекта)

Следует нажать кнопку **Да**. Пустой проект будет ***удален***, а новый проект (созданный в п. 7) будет записан в **оперативную память** контроллера.

3. Для загрузки проекта во **flash-память** следует нажать кнопку **Создать загрузочное приложение** в меню **Онлайн**:

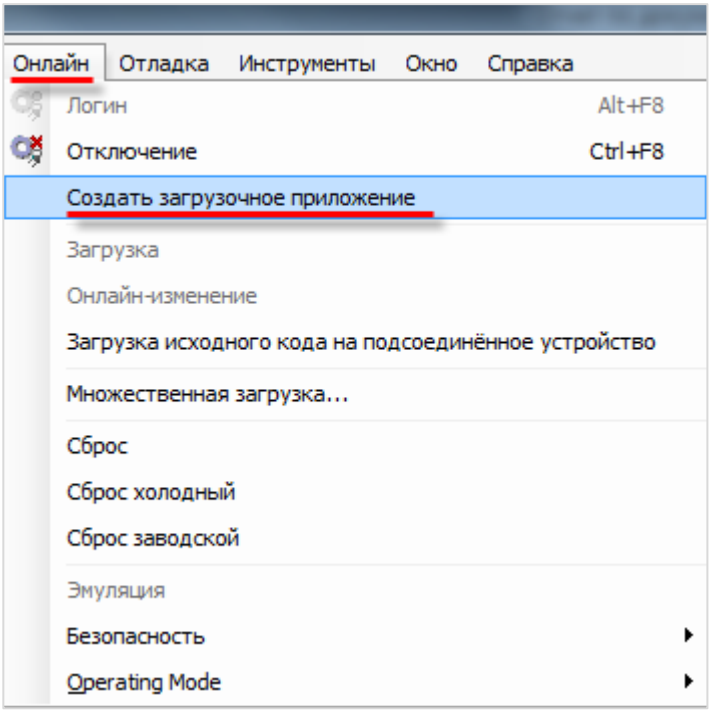


Рисунок 8.5 – Создание загрузочного приложения (загрузка проекта во flash-память контроллера)

4. Также рекомендуется выполнить **Загрузку исходного кода на подсоединенное устройство,** что позволит в случае необходимости ***выгрузить*** проект CODESYS из памяти контроллера:

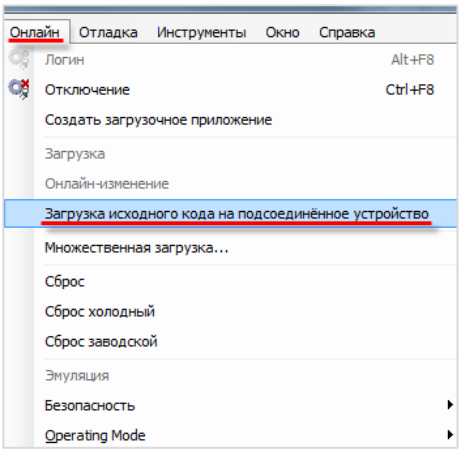


Рисунок 8.6 – Загрузка исходного кода на подсоединенное устройство

5. Затем проект следует запустить (меню **Отладка** – **Старт**):

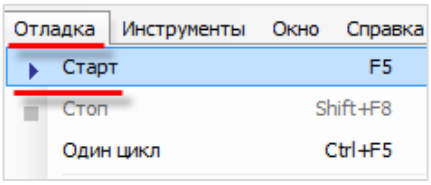


Рисунок 8.7 – Запуск проекта

Проект загружен во **flash-память** контроллера и ***запущен***. В случае перезагрузки контроллера проект ***сохранится*** в памяти контроллера и будет ***автоматически перезапущен***.

Чтобы ***удалить проект*** из контроллера, следует воспользоваться командой **Сброс заводской** из меню **Отладка**:

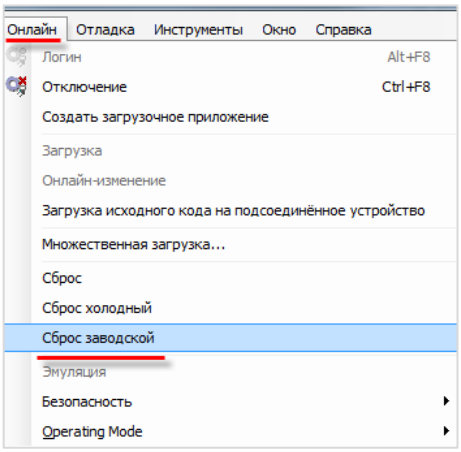


Рисунок 8.8 – Удаление проекта из контроллера

Команды типа **Сброс** выполняют следующие действия:

1. **Сброс** – заново инициализирует все переменные, ***кроме*** энергонезависимых (retain).
2. **Сброс холодный** – заново инициализирует все переменные, ***включая*** энергонезависимые (retain).
3. ***Сброс заводской*** – заново инициализирует все переменные, включая энергонезависимые (retain), ***после чего удаляет проект*** из контроллера.

В процессе разработки проекта зачастую приходится вносить в него изменения и ***заново загружать*** в контроллер – в данном случае при загрузке можно увидеть следующее информационное окно:

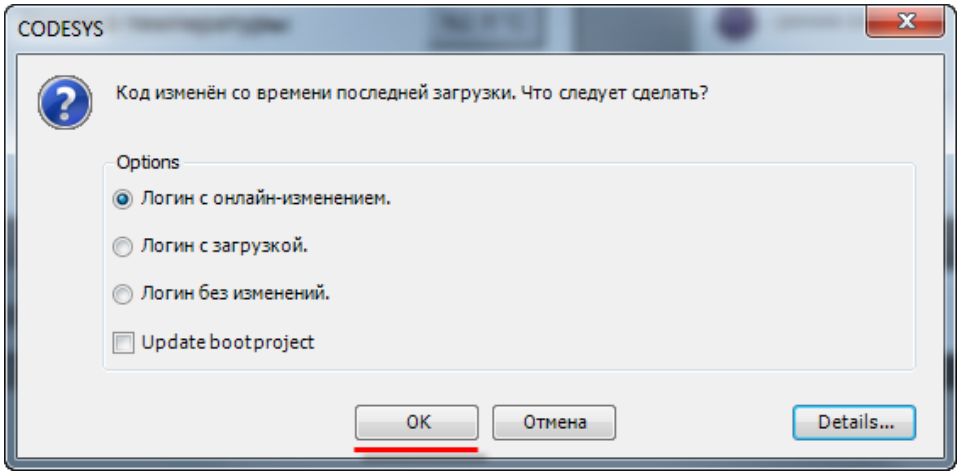


Рисунок 8.9 – Диалоговое окно обновления проекта

1. Команда **Логин с онлайн-изменением** загружает в **оперативную память** контроллера ***только те части проекта***, которые подверглись изменениям с момента последней загрузки. Процедура онлайн-изменения опирается на использовании **информации компиляции**, поэтому невозможна после выполнения команды **Очистить** из меню **Компиляция**.
2. Команда **Логин с загрузкой** загружает проект в оперативную память контроллера (стандартная процедура загрузки).
3. Команда **Логин без изменений** осуществляет подключение к контроллеру ***без загрузки проекта***.
4. Если установлена галочка **Update bootproject,** то при выполнении любой из вышеописанных команд происходит обновление проекта во flash-памяти контроллера.

ПРИМЕЧАНИЕ
Рекомендуется использовать команду **Логин с загрузкой**.

9 Графический дизайн проекта

Во время разработки экранов визуализации (см. п. 7.3) использовались только графические примитивы, входящие в состав **CODESYS**. Но для разработки сложного и эргономичного интерфейса оператора может возникнуть необходимость в использовании дополнительных изображений. Для этого можно воспользоваться элементом **Переключатель изображений**, расположенном на вкладке **Индикаторы/Переключатели/Изображения**.

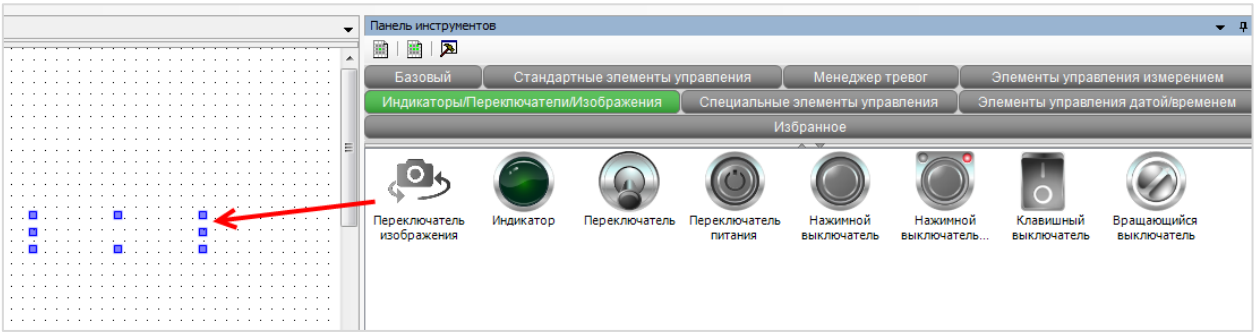


Рисунок 9.1 – Добавление элемента Переключатель изображения

Настройки элемента:

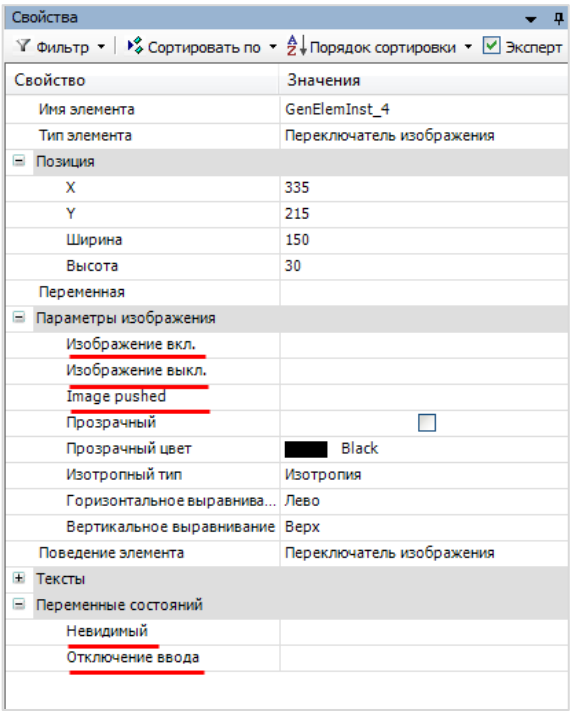


Рисунок 9.2 – Настройки элемента Переключатель изображения

Принцип работы элемента: если привязанная к нему логическая переменная принимает значение **TRUE**, то отображается изображение, указанное в поле **Изображение вкл.** Если переменная принимает значение **FALSE** – отображается изображение, указанное в поле **Изображение выкл**. Предварительно оба изображения должны быть добавлены в проект через **Пул изображений** (см. п. 7.3.5).

В случае нажатия на элемент переменная меняет свое состояние (с TRUE на FALSE или наоборот). Во время нажатия элемент на короткое время отображается изображение, заданное в поле **Image pushed**.

Как и для других элементов, для **Переключателя изображений** можно настроить **Невидимость** и **Отключение ввода**.

С помощью этого элемента для проекта, разработанного в п. 7, был создан новый дизайн. Также на этом этапе был добавлен стартовый экран проекта. Поскольку использованные на нем элементы (текстовые поля и кнопка перехода) уже рассмотрены в п. 7, то процесс создания этого экрана не описывается. Методика создания мультиязычного проекта описана в документе **CODESYS V3.5. Визуализация**.

Экраны визуализации в новой версии проекта выглядят следующим образом:

Рисунок 9.3 – Экран Start (дизайнерская версия проекта)

Рисунок 9.4 – Экран MainScreen (дизайнерская версия проекта)

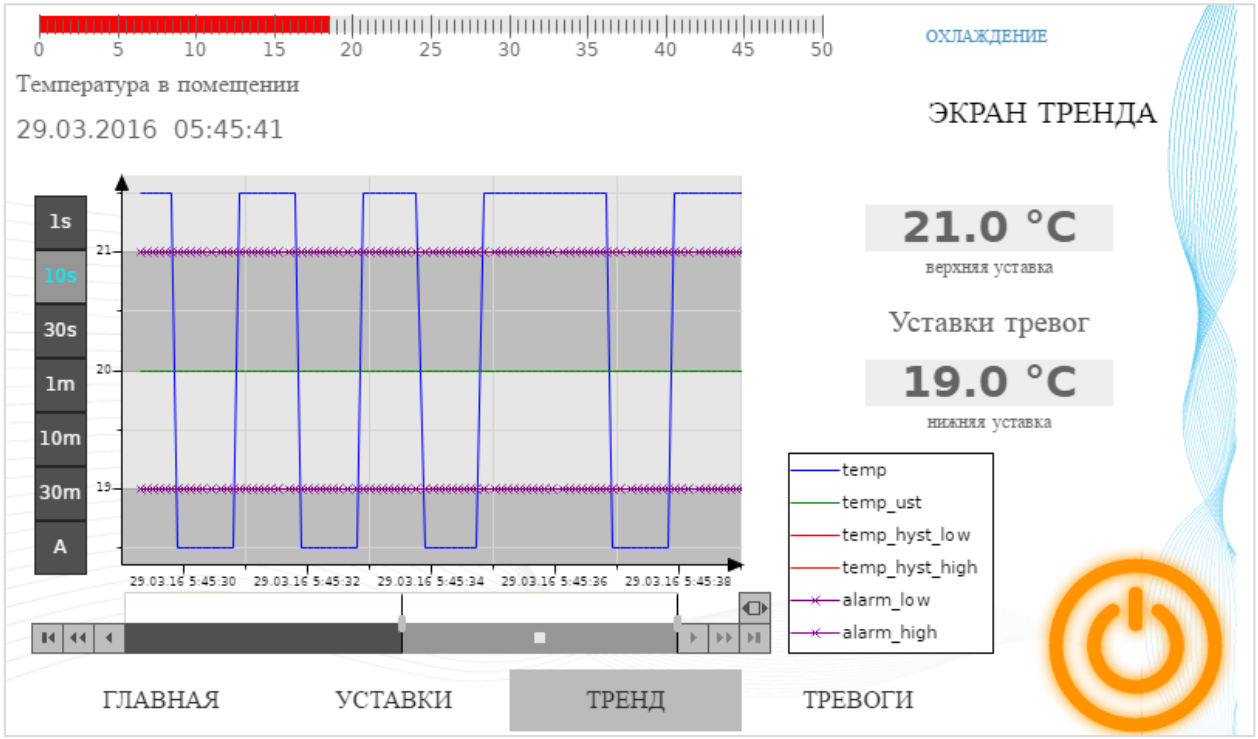


Рисунок 9.5 – Экран Trend (дизайнерская версия проекта)

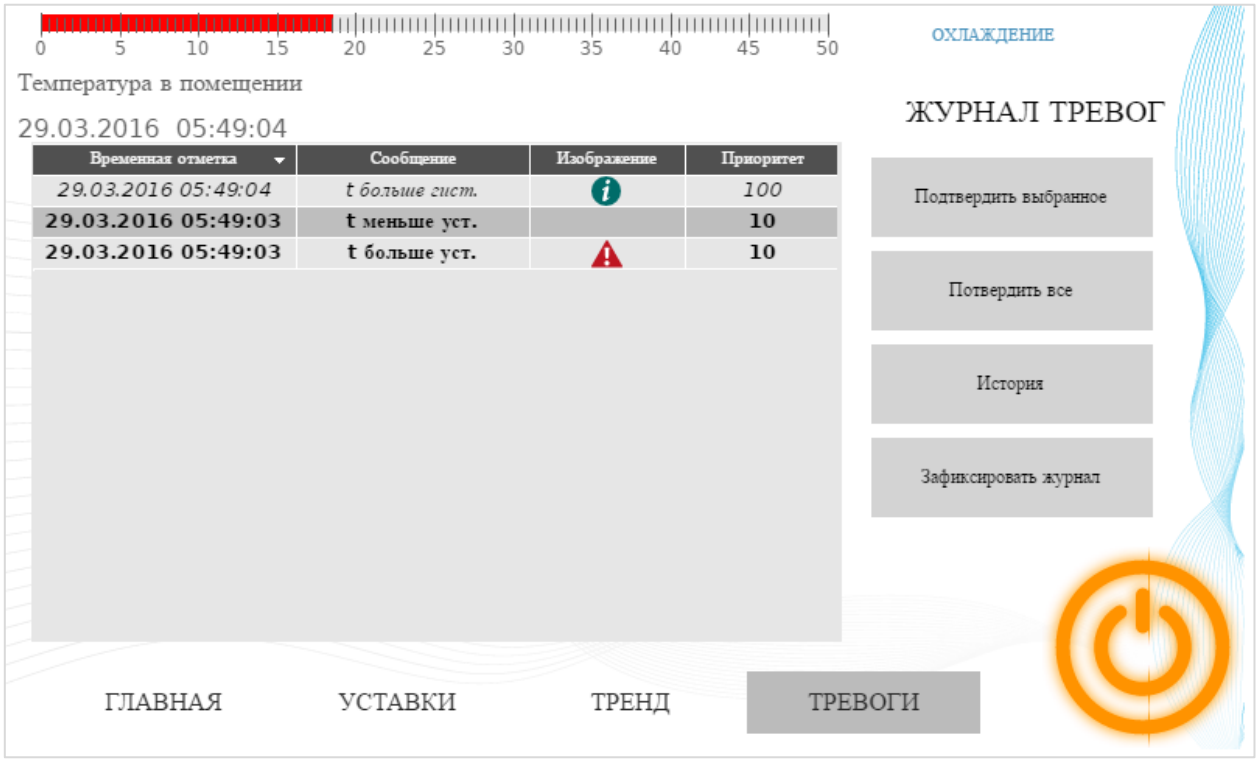


Рисунок 9.6 – Экран Alarm_log (дизайнерская версия проекта)

Проект создан в среде **CODESYS V3.5 SP11 Patch 5** и подразумевает запуск на **СПК1xx [M01]** с таргет-файлом **3.5.11.x.** В случае необходимости запуска проекта на другом устройстве следует изменить таргет-файл в проекте (**ПКМ** на узел **Device** – **Обновить устройство).**

Проект доступен для скачивания: Example_FirstStart.projectarchive

10 Работа с демонстрационным проектом

После загрузки проекта будет автоматически отображен стартовый экран визуализации.

Если контроллер ***поддерживает*** web-визуализацию, то для ее запуска следует в браузере (подходит любой современный браузер с поддержкой **HTML5**), открыть страницу:

http://<IP-адрес контроллера>:8080/<имя страницы>.htm

Адрес страница для контроллера с настройками по умолчанию:

http://192.168.0.10:8080/webvisu.htm

Рисунок 10.1 – Экран Start (дизайнерская версия проекта)

Для перехода на экран **Mainscreen** следует нажать кнопку **Старт проекта**.

По умолчанию кондиционер **включен**, уставка температуры – **20 °C**, гистерезис – **5 %.**

Пользователь может включать/отключать кондиционер, изменять текущее значение температуры, ее уставку и значение гистерезиса.

Рисунок 10.2 – Экран MainScreen, ввод значения температуры

Для перехода на экран **Тренд** следует нажать кнопку **Тренд**.

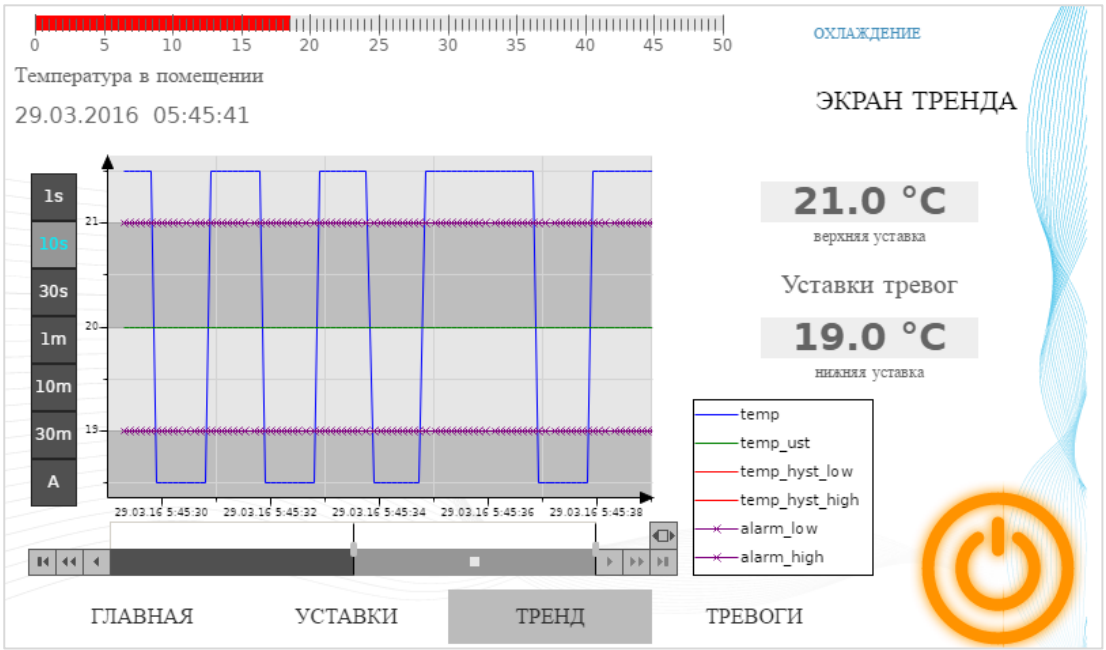


Рисунок 10.3 – Экран Trend (дизайнерская версия проекта)

Пользователь может просматривать историю тренда и изменять значения уставок тревог.

Для изменения периода времени, отображаемого на тренде, следует воспользоваться **левым рядом кнопок**.

Для прокрутки истории следует воспользоваться **ползунком**, расположенным под трендом:

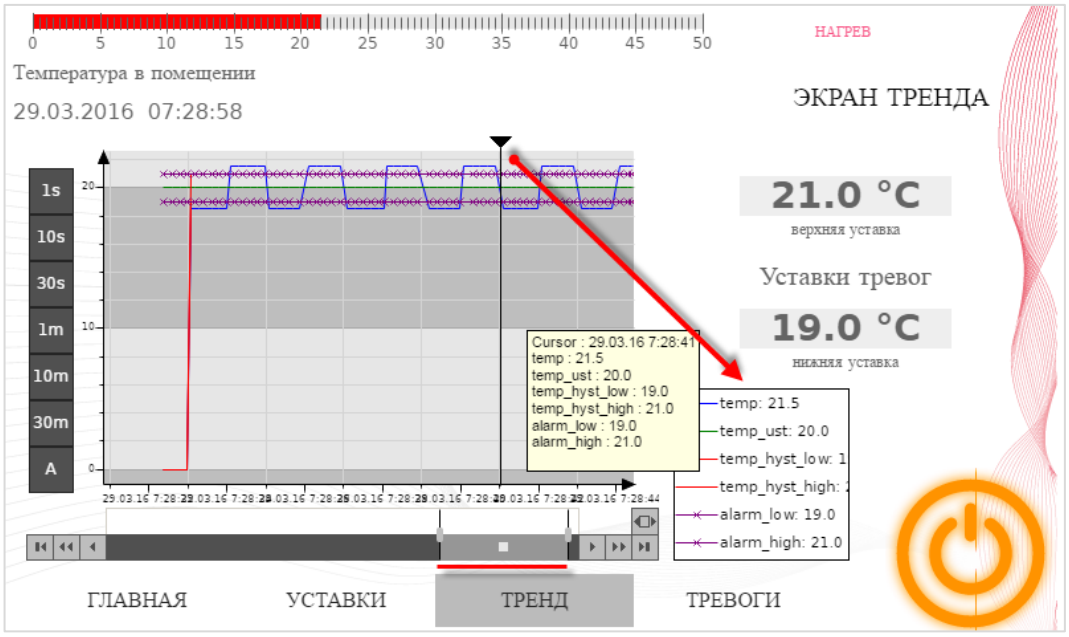


Рисунок 10.4 – Экран Trend, просмотр истории

Во время просмотра истории обновление тренда не происходит. Также в этом режиме появляется **маркер,** который позволяет посмотреть значения переменных тренда в тот или иной момент времени (значения отображаются рядом с **легендой**).

Чтобы вернуться к отображению информации в реальном времени следует нажать на самую **правую нижнюю кнопку** тренда.

ПРИМЕЧАНИЕ

Так как аварийные уставки являются **энергонезависимыми (retain)** переменными, их значения будут ***сохраняться после перезагрузки*** контроллера.

Чтобы перейти на экран **Тревоги** следует нажать на кнопку **Тревоги**.

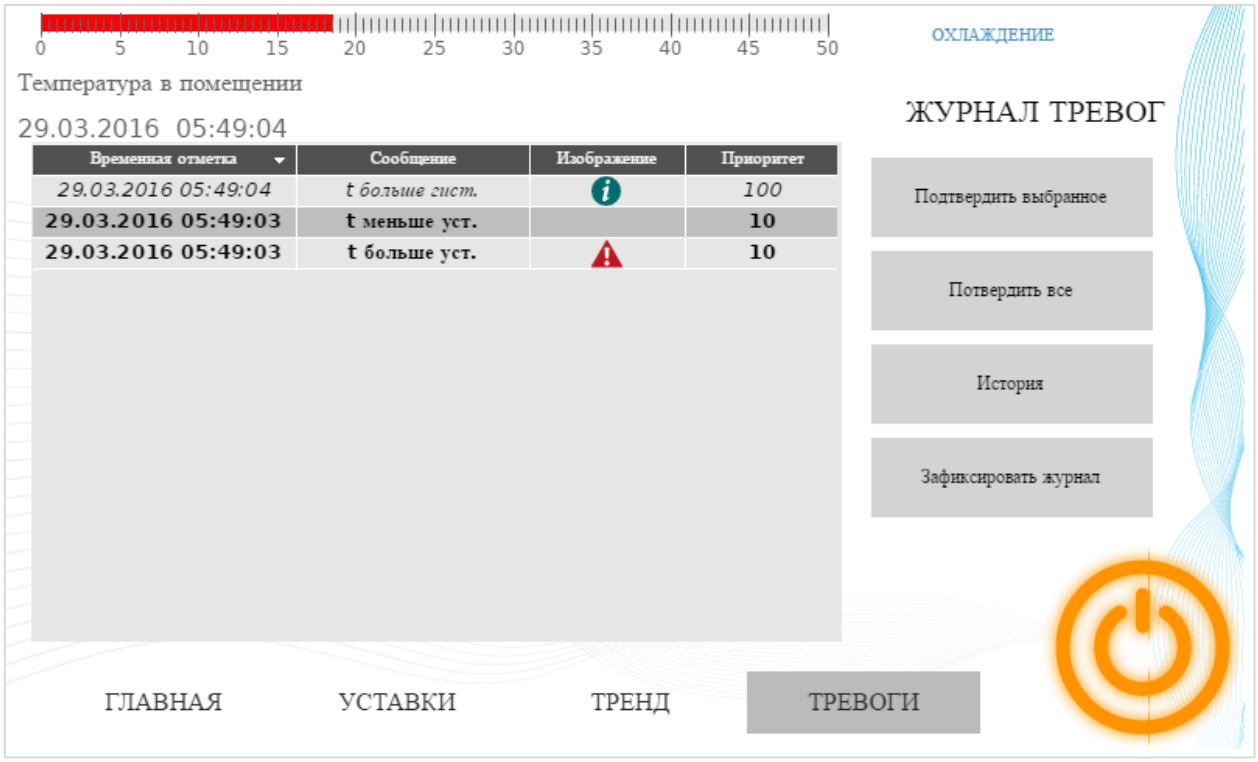


Рисунок 10.5 – Экран Alarm_log (дизайнерская версия проекта)

Нажатие на кнопку **История** позволяет посмотреть историю журнала тревог:

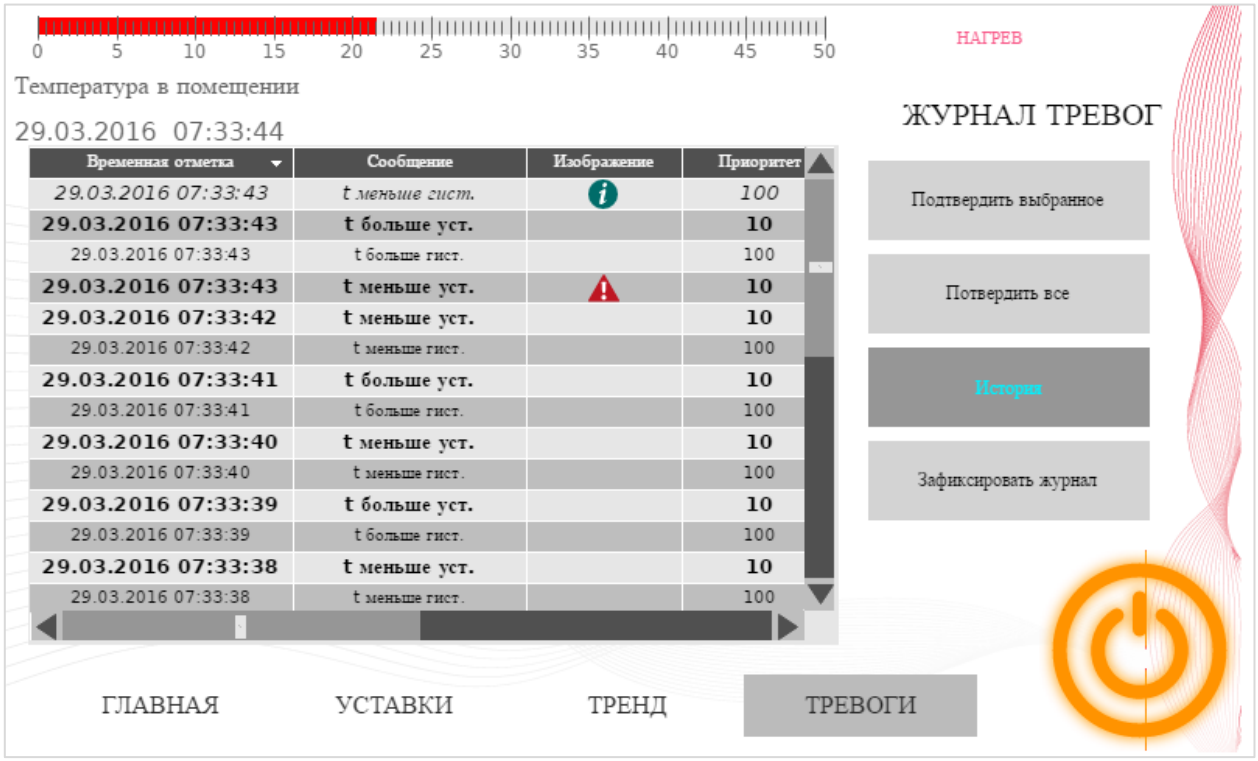


Рисунок 10.6 – Экран Alarm_log (дизайнерская версия проекта)

Для возвращения в режим реального времени следует повторно нажать кнопку **История**.