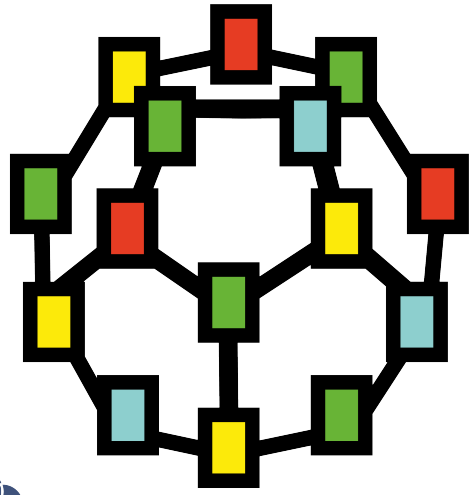
Полигон. Библиотека paCore



ПромАвтоматика

Руководство пользователя

06.2024
версия dev8.0

Содержание

Используемые термины и сокращения

ЛКМ – левая кнопка мыши.

Место работы – набор программ, вызывающийся заданным способом.

Модуль – основной узел проекта в Полигон, в котором задаются настройки для подключения к контроллеру и др. параметры для создания пользовательского приложения.

ОС – операционная система.

ПК – персональный компьютер.

ПКМ – правая кнопка мыши.

ПЛК – программируемый логический контроллер.

ПО – программное обеспечение.

Представления – тип окон программы Полигон, предназначенных для редактирования проекта (например, дерево, страница, редактор и т.д.).

Программа – относительно независимая алгоритмическая задача или группа однотипных задач.

Сборка – компиляция исходного кода из одного или нескольких файлов и последующее связывание этих файлов в исполняемый файл.

Составной функциональный блок – пользовательский алгоритм, используемый несколько раз в проекте – функциональный блок, состоящий из других функциональных блоков или написанный на **C++**.

Таймер (место работы) – поток, выполняющийся в режиме реального времени с заданной периодичностью.

Трансляция – процесс перевода программы с одного языка на другой, состоит из компиляции и интерпретации.

ФБ – функциональный блок – элементарный алгоритм, реализованный как класс **C++**.

Фон (место работы) – поток, выполняющийся циклически.

C++ – компилируемый язык программирования со строгой типизацией, поддерживающий парадигмы процедурного и объектно-ориентированного программирования.

FTP (File Transfer Protocol) — протокол прикладного уровня для передачи файлов по сети.

NTP (Network Time Protocol) — сетевой протокол для синхронизации внутренних часов устройств в сети.

SQL (**Structured Query Language**) – язык программирования для хранения и обработки информации в реляционной базе данных.

SSH (Secure SHell) – протокол прикладного уровня для удаленного управления операционной системой с шифрованием трафика.

Unix-время – количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года (четверг).

UTC (Temps Universel Coordonné) – всемирное координированное время.

Введение

Полигон – это среда графического программирования на языке функциональных блоков. Не только алгоритмы, но и драйверы и протоколы обмена, реализованы в виде функциональных блоков. Прикладному программисту нет необходимости описывать переменные – система создает их, обеспечивая уникальность, следит за корректным преобразованием типов.

Проект имеет единую иерархическую структуру для описания аппаратной конфигурации, программы и обмена данными с другими системами.

Окончательным этапом создания программы на языке функциональных блоков является трансляция, после которой создается исполняемый файл **.o**. Для тестирования полученной программы можно воспользоваться отладчиком.

Открыть или создать проект (файл с расширением **.pl2**) можно в окне Проекты. Рекомендуется размещать каждый новый проект в отдельном каталоге.

Открытый проект можно отобразить в рабочих окнах на экране, для этого предназначены представления Дерево  и Страница  . Для того чтобы добавить в проект функциональные блоки, необходимо также открыть соответствующие библиотеки (файлы с расширением **.ll2**).

Расположение окон, а также ссылки на открытые проекты и библиотеки сохраняются в файле на диске и восстанавливаются при следующем запуске Полигона. Можно создавать разные конфигурации рабочего экрана и сохранять их при помощи меню Экран.

Одно из рабочих окон обычно является текущим (обозначается символом ), в это окно отображаются страницы при нажатии на ссылки на полях и двойном нажатии в ***Дереве***. Окно можно зафиксировать (флаг ***Фиксировать*** в верхней панели окна), чтобы оно при нажатии не становилось текущим (это полезно, например, для дерева, в котором всегда отображается структура проекта).

Документ соответствует версии среды Полигон 2 – **1921** и выше, версии библиотеки ***paCore*** – **979** и выше.

1 Знакомство со средой разработки Полигон

1.1 Установка и первый запуск Полигон

1.1.1 Установка Полигон под ОС Windows

Скачать инсталлятор среды Полигон под ОС Windows можно по ссылке.

Рекомендуемые системные требования:

- ОС: Windows 7/8/10/11 (32/64 Bit);
- Оперативная память: 4 Гб и выше;
- Память на диске: 2 Гб и выше.

Для установки среды следует:

1. Распаковать архив с установщиком и запустить ***Polygon2Installer_LinuxOwen3.exe***.

Рисунок 1.1 – Инсталлятор Полигон 2

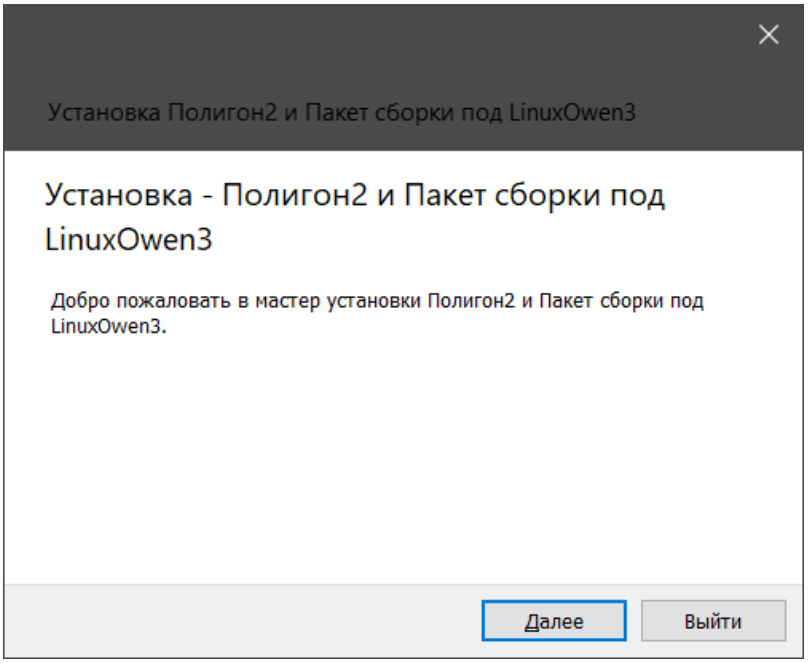


Рисунок 1.2 – Установка Полигон 2. Приветственное окно

2. Выбрать путь для установки среды Полигон. По умолчанию среда устанавливается на диск **C**.

ВНИМАНИЕ
Путь установки среды не должен содержать кириллицу и пробелы.

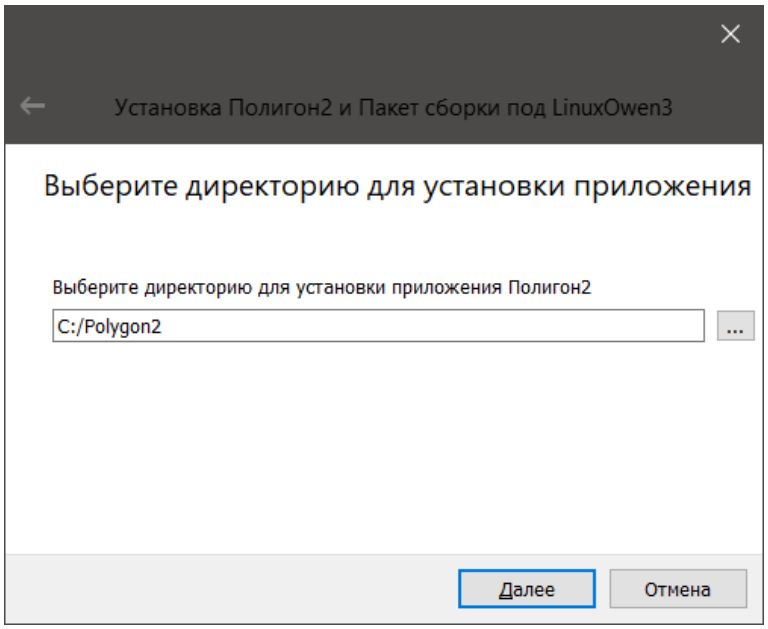


Рисунок 1.3 – Выбор директории установки

3. Выбрать компоненты для установки.

По умолчанию инсталлятор устанавливает саму среду Полигон и пакет сборки под текущую заводскую прошивку ПЛК210. При необходимости можно установить компоненты отдельно путем установки соответствующих флагов.

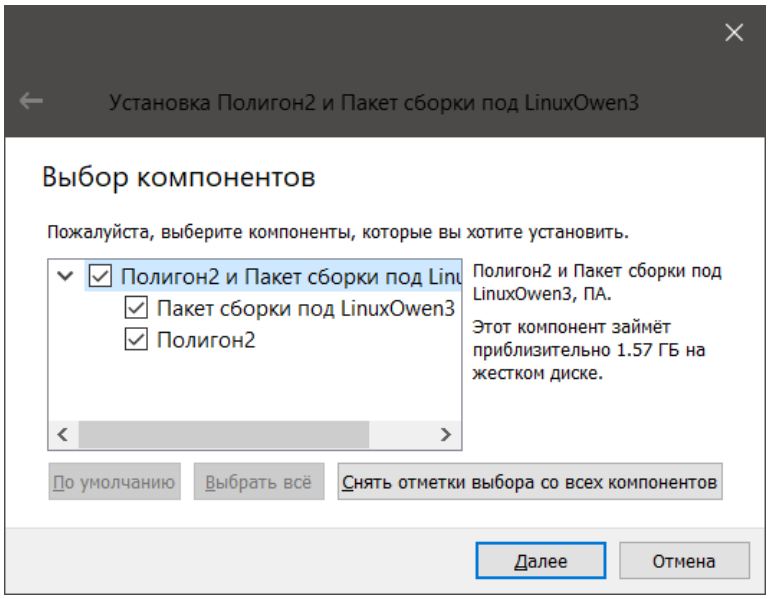


Рисунок 1.4 – Установка Полигон 2. Выбор компонентов установки

4. Ознакомиться с лицензионным соглашением и поставить флаг согласия.

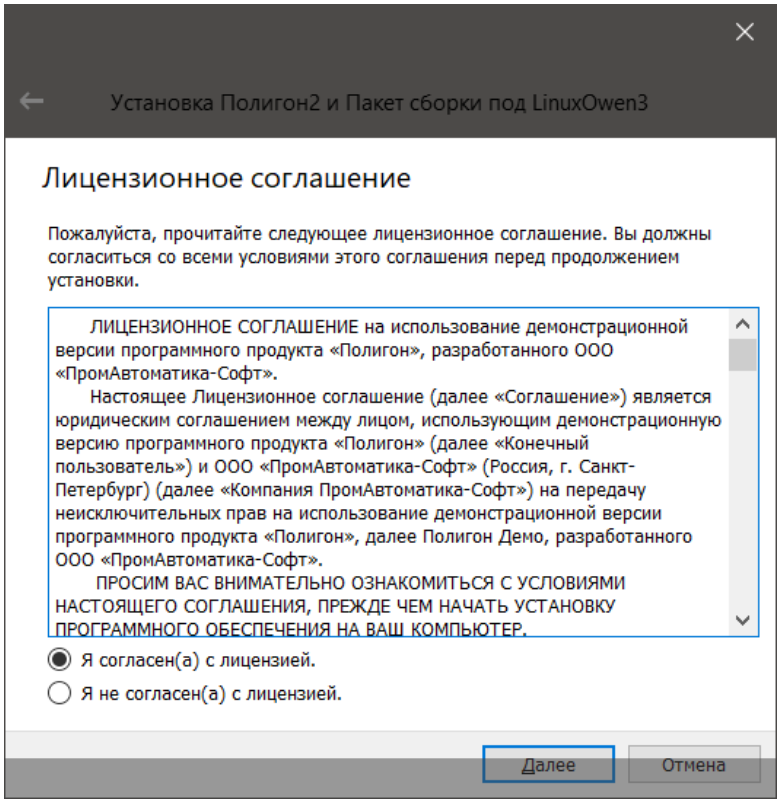


Рисунок 1.5 – Установка Полигон 2. Лицензионное соглашение

5. Если необходимо создать папку для ярлыков в меню «Пуск».

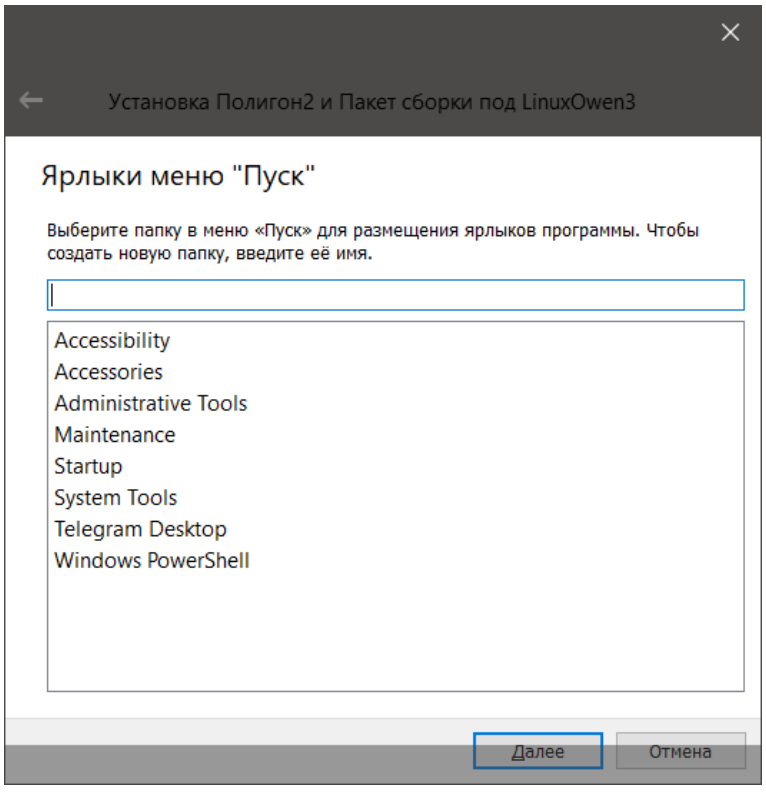


Рисунок 1.6 – Установка Полигон 2. Ярлыки меню «Пуск»

6. Нажать ***Установить***.

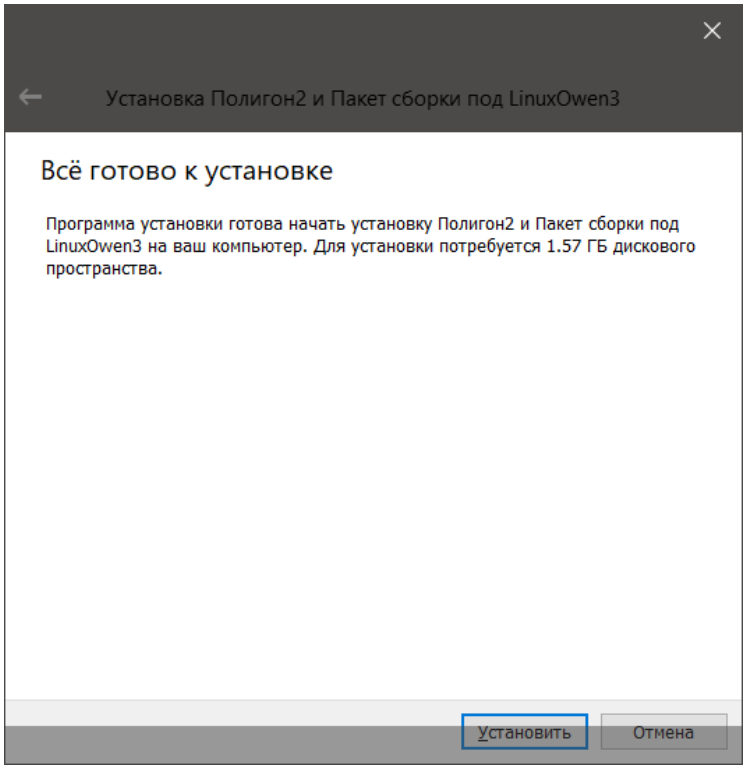


Рисунок 1.7 – Установка Полигон 2. Завершение установки

По завершению установки на рабочем столе появится ярлык программы Полигон. Инсталлятор предложит открыть краткое описание демонстрационного проекта.

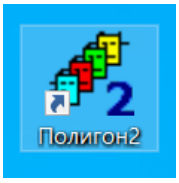


Рисунок 1.8 – Ярлык Полигон 2

Инсталлятор также предложит установить или обновить драйверы Guardant.

После установки среды рекомендуется обновить версии среды и библиотек до актуальных. Добавление библиотек в среде Полигон описано в разделе 1.4.2. Процедура обновления среды и библиотек описана в разделе 1.5.

1.1.2 Установка Полигон под ОС Linux

Скачать скрипт установки среды Полигон под ОС Linux можно по ссылке.

Рекомендуемые системные требования:

- ОС: Astra Linux Special Edition 1.7, Ubuntu (версия ядра Linux 5.10 и выше);
- Оперативная память: 4 Гб и выше;

– Память на диске: 2 Гб и выше.

Для установки среды следует:

1. Распаковать архив со скриптом установки.
2. Дать права на исполнение скрипта установки ***Polygon2Installer_LinuxOwen3.sh***.

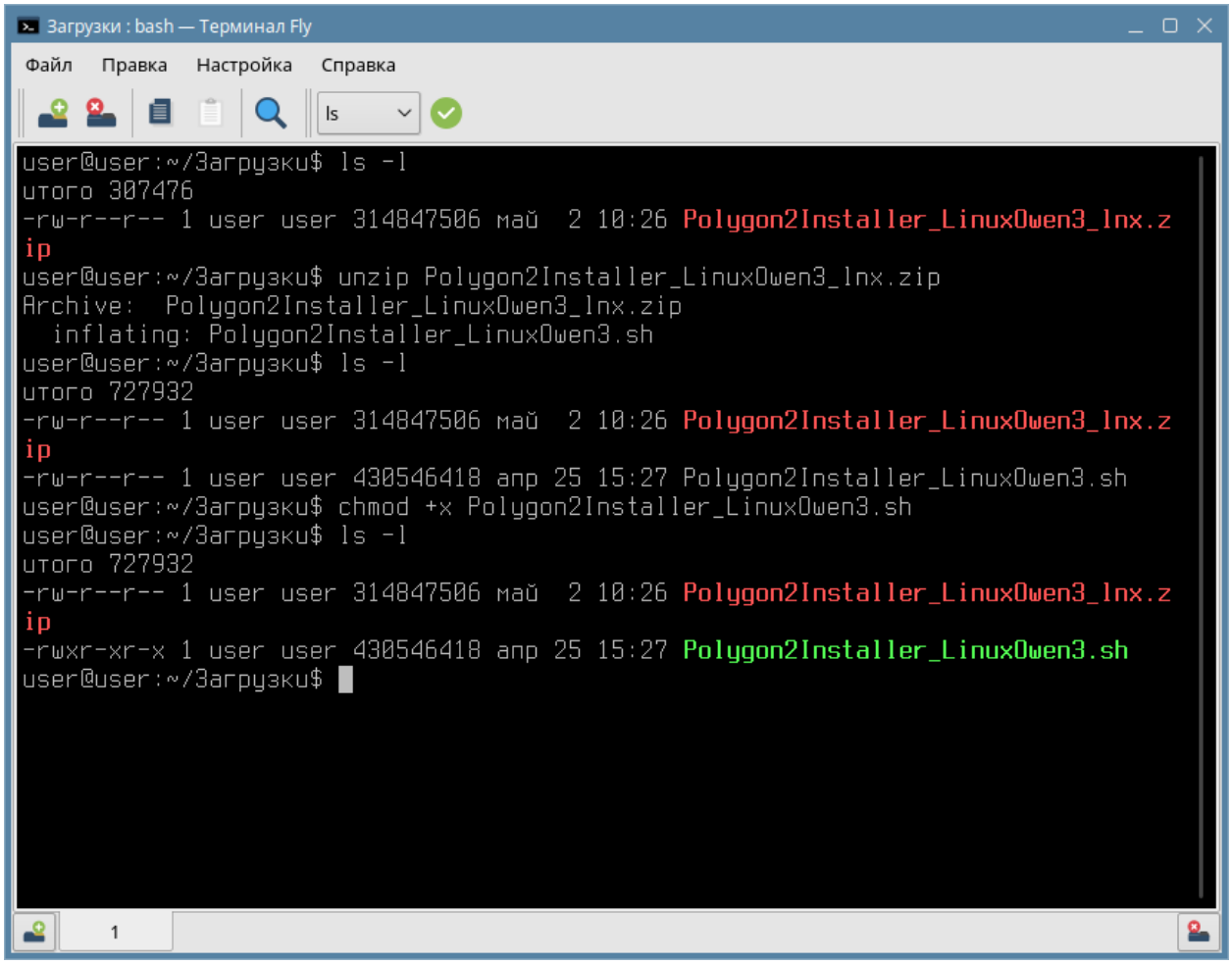


Рисунок 1.9 – Распаковка архива и разрешение исполнения скрипта установки

3. Запустить скрипт установки.
4. Следовать инструкциям установщика.

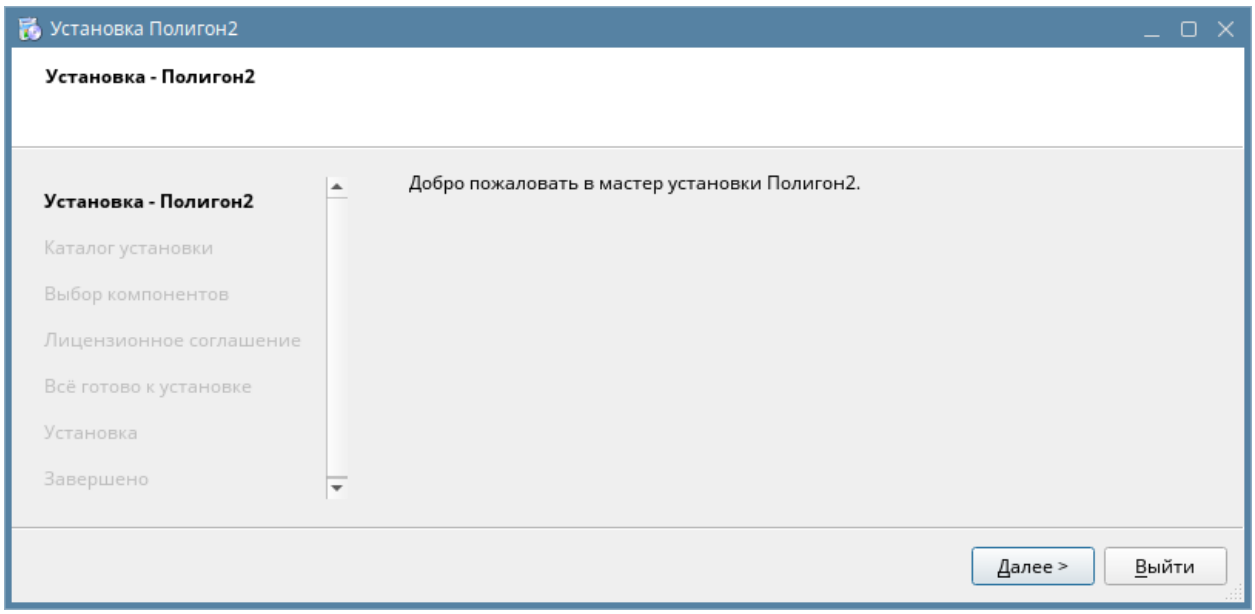


Рисунок 1.10 – Установка Полигон 2. Приветственное окно

5. Указать директорию для установки среды Полигон.

ВНИМАНИЕ
Путь установки среды не должен содержать кириллицу и пробелы.

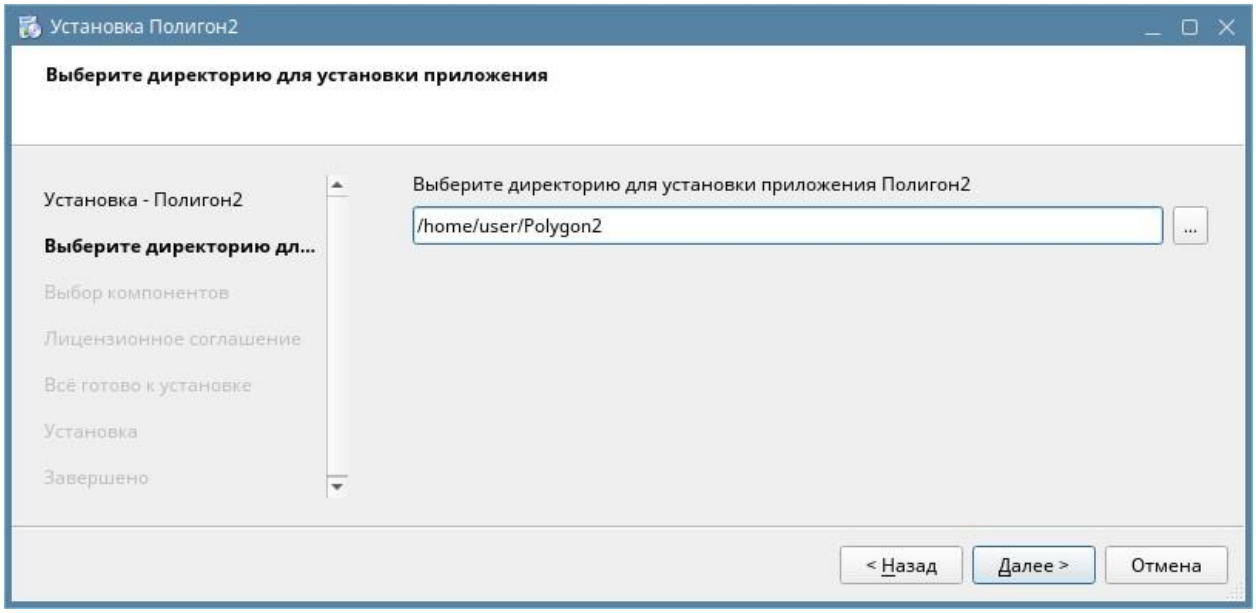


Рисунок 1.11 – Установка Полигон 2. Выбор директории для установки

6. Выбрать компоненты среды для установки.

По умолчанию инсталлятор устанавливает саму среду Полигон и пакет сборки под текущую заводскую прошивку ПЛК210.

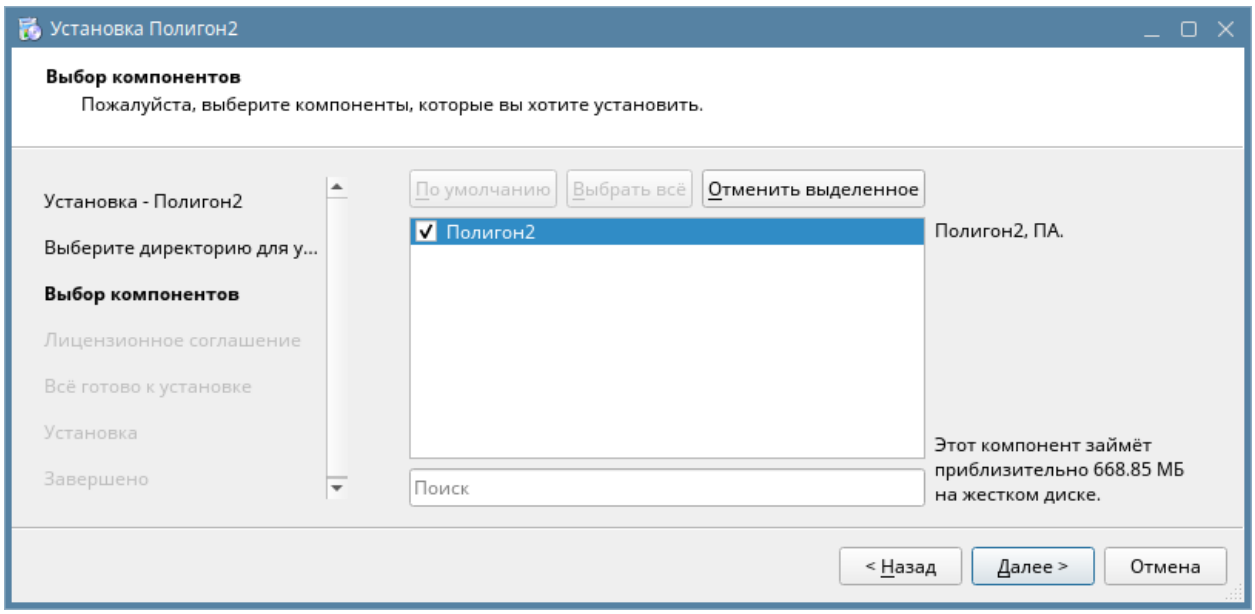


Рисунок 1.12 – Установка Полигон 2. Выбор компонентов для установки

7. Ознакомиться с лицензионным соглашением и поставить флаг согласия.

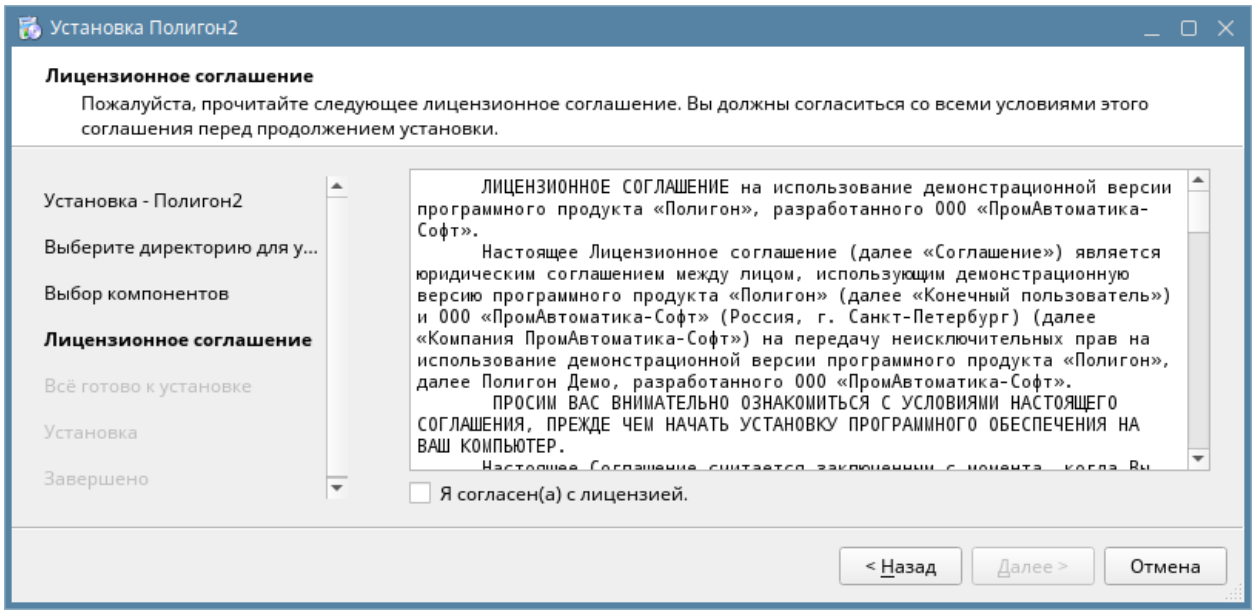


Рисунок 1.13 – Установка Полигон 2. Лицензионное соглашение

8. Нажать ***Установить***.

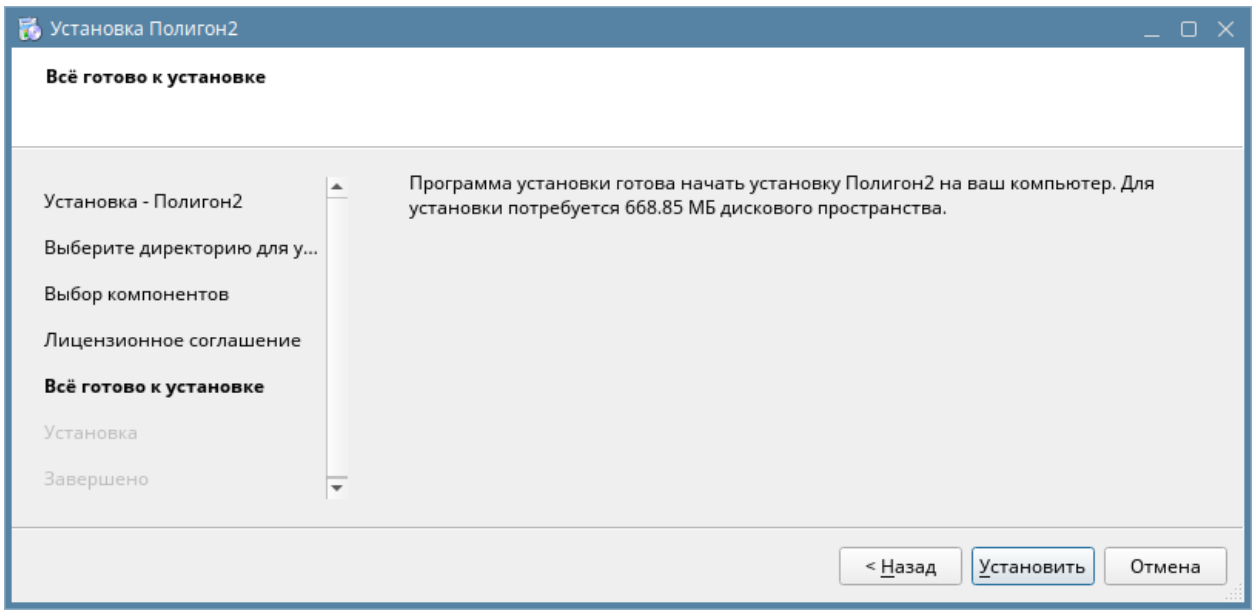


Рисунок 1.14 – Установка Полигон 2. Завершение установки

По завершению установки на рабочем столе появится ярлык программы Полигон (только для ПК с ОС Astra Linux). Инсталлятор предложит открыть краткое описание демонстрационного проекта.

Рисунок 1.15 – Ярлык Полигон 2

После установки среды рекомендуется обновить версии среды и библиотек до актуальных. Добавление библиотек в среде Полигон описано в разделе 1.4.2. Процедура обновления среды и библиотек описана в разделе 1.5.

1.1.3 Первый запуск Полигон

Полигон рекомендуется запускать с ярлыка на рабочем столе или из папки ***Polygon2***: для ОС Windows исполняемый файл ***polygon2.exe***, для ОС Linux – ***startPolygon2.sh***.

При первом запуске среды откроется демонстрационный проект ***mnu_demo.pl2***. Краткое описание демонстрационного проекта приведено в документе ***Описание.pdf*** в папке ***Polygon2***.

Если необходимо, демонстрационный проект можно запустить на виртуальном контроллере через ***Панель отладки***.

Для этого следует:

1. Добавить ***Панель отладки*** через меню ***Окна/Панели инструментов/Панель отладки***.
2. Нажать на ***Панели отладки*** кнопку ***Запустить*** на виртуальном контроллере.
3. В отдельном окне запуститься приложение виртуального контроллера.
4. Среда попросит ввести пароль для доступа отладчиком среды. Пароль демонстрационного проекта – **123**.
5. Среда подключится отладчиком среды к запущенному контроллеру.

Рисунок 1.16 – Запуск демо-проекта на виртуальном контроллере

1.2 Контроллеры

В настоящее время компания ОВЕН выпускает следующие модификации контроллеров с исполнительной средой Полигон:

Таблица 1.1 – ПЛК ОВЕН с исполнительной средой Полигон

Модификация контроллера	Версия прошивки и инструкция по обновлению
ПЛК210-11-PL-X[1]	3.6.0610.1024
ПЛК210-12-PL-X	
ПЛК210-14-PL-X	

1 X – версия лицензии runtime. Описание версий лицензий приведено в разделе 1.3

1.3 Лицензирование Полигон

1.3.1 Описание лицензионных пакетов

Для контроллеров ОВЕН с исполнительной средой Полигон доступны следующие версии лицензий runtime:

Таблица 1.2 – Описание лицензий для контроллеров ОВЕН с исполнительной средой Полигон

Лицензия	Описание	Состав доступных библиотек[1]
BASE	Базовая лицензия	***paCore*** ***paOwenIO*** ***paOpcUA*** ***paControls*** ***paModbus*** ***profiLogger*** ***profiLoggerLight***
BASE104	Лицензия с поддержкой протоколов **МЭК 60870-5-101** и **МЭК 60870-5-104**	Библиотеки базовой лицензии + ***paIEC104***

Продолжение таблицы 1.2

BASE850	Лицензия с поддержкой протокола **МЭК 61850**	Библиотеки базовой лицензии + ***paIEC850***
BASE-R	Лицензия с поддержкой программного резервирования ПЛК	Библиотеки базовой лицензии + ***paSync***
BASE104-R	Лицензия с поддержкой протоколов **МЭК 60870-5-101** и **МЭК 60870-5-104** и программного резервирования ПЛК	Библиотеки базовой лицензии + ***paIEC104***, ***paSync***
BASE850-R	Лицензия с поддержкой протокола **МЭК 61850** и программного резервирования ПЛК	Библиотеки базовой лицензии + ***paIEC850***, ***paSync***

1 Описание библиотек приведено в разделе 1.4

Состав библиотек, доступных пользователю для полноценной работы можно посмотреть в web-конфигураторе контроллера в разделе ***Состояние/Обзор***.

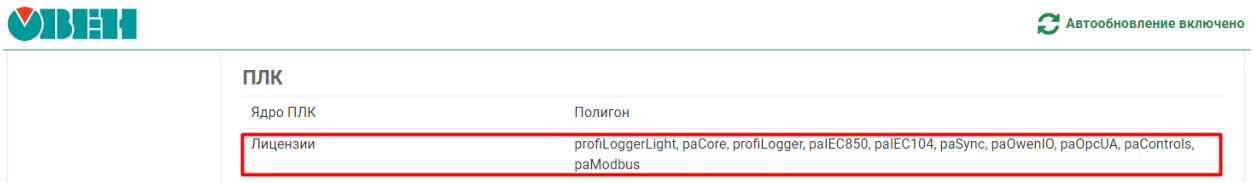


Рисунок 1.17 – Состав доступных библиотек в web-конфигураторе

ВНИМАНИЕ
Если в пользовательском проекте используются блоки из библиотек без соответствующих лицензий, проект прекратит свою работу по истечении **1** часа. При подключении отладчиком на модуле будет указан ***Демо-режим*** и время до окончания выполнения программы.

1.3.2 Опции лицензионных пакетов

При необходимости можно приобрести опции для работы с блоками библиотек ***paSync***, ***paIEC104***, ***paIEC850*** отдельно:

Таблица 1.3 – Опции лицензий runtime

Опция	Описание	Состав доступных библиотек[1]
Redu	Опция поддержки программного резервирования ПЛК	***paSync***
IEC104	Опция поддержки протоколов **МЭК 60870-5-101** и **МЭК 60870-5-104**	***paIEC104***
IEC850	Опция поддержки протокола **МЭК 61850**	***paIEC850***

1 Описание библиотек приведено в разделе 1.4

Загрузка файлов лицензии в контроллер описана в разделах 1.3.2.1, 1.3.2.2, 1.3.2.3.

1.3.2.1 Загрузка файлов лицензий в ПЛК в среде Полигон

Пакет лицензий можно загрузить из среды Полигон.

Для этого следует настроить свойства модуля для подключения к контроллеру. Подключаться к контроллеру можно через **USB Device** (IP адрес ПЛК при подключении через **USB Device** – **172.16.0.1**) или через **Ethernet**.

Подробно подключение к контроллеру описано в разделе 6.

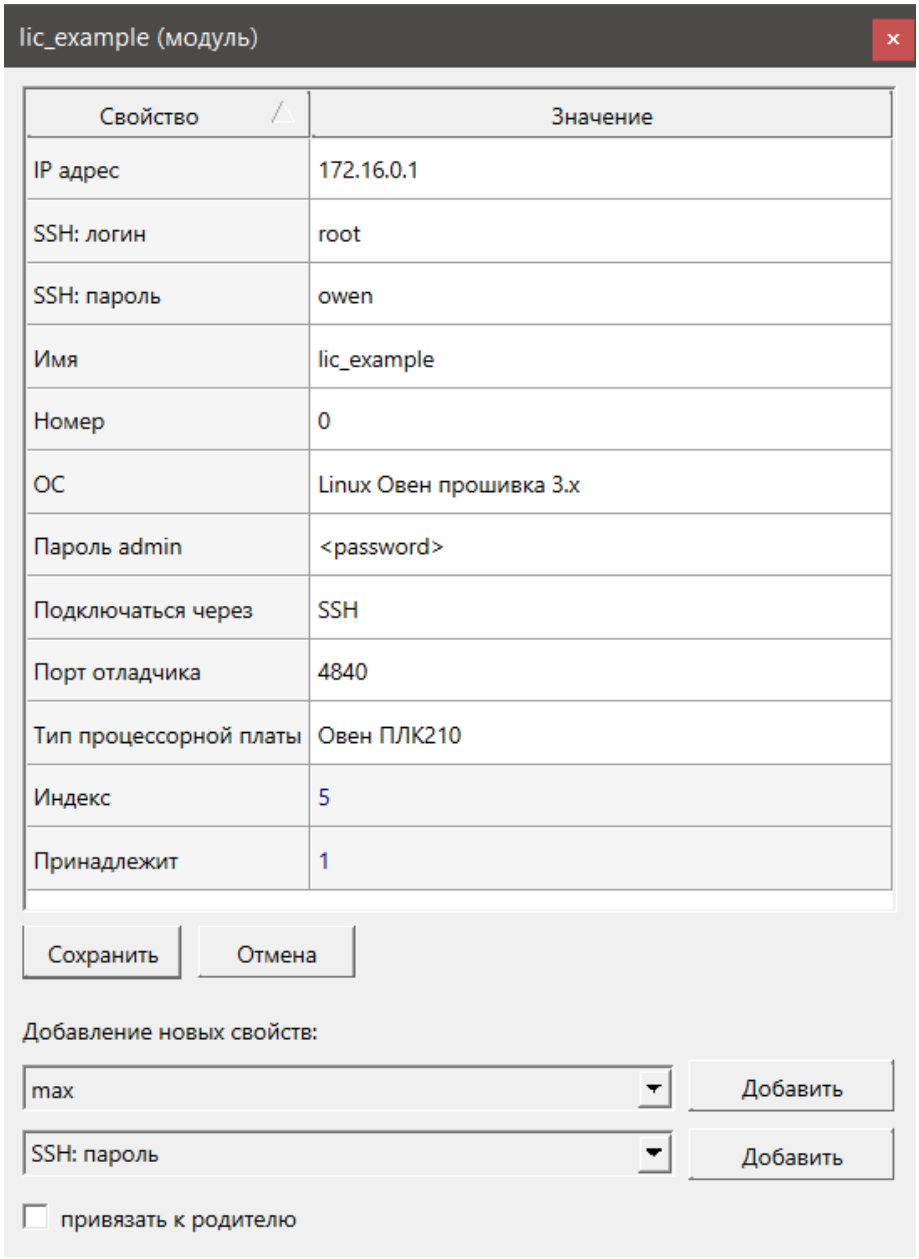


Рисунок 1.18 – Свойства модуля для подключения к ПЛК через USB Device

Для загрузки лицензий следует нажать ПКМ на модуле и выбрать команду ***Записать лицензии***.

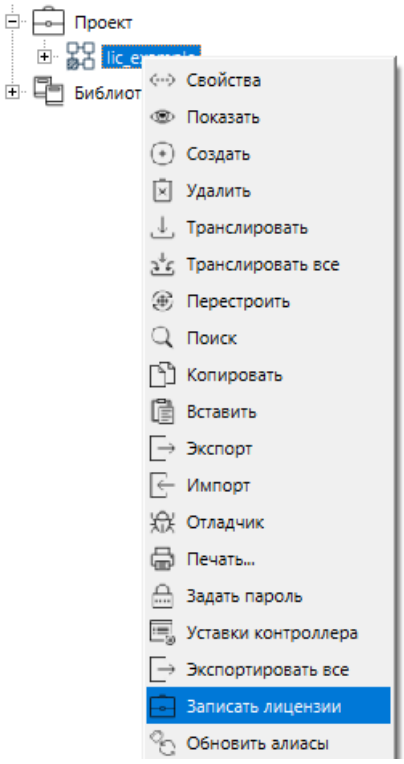


Рисунок 1.19 – Записать лицензии

В появившемся окне следует ввести заводской номер контроллера. Заводской номер выгравирован на корпусе прибора и приведен в паспорте на прибор (входит в комплект поставки прибора).

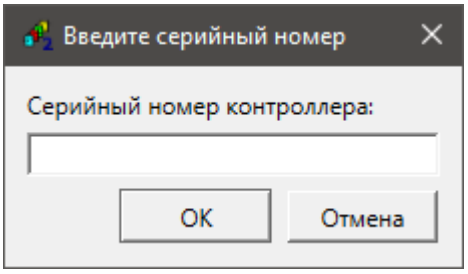


Рисунок 1.20 – Ввод серийного номера

После нажатия ***ОК*** начнется процесс скачивания лицензий с сервера **pa.ru** и загрузка их в контроллер. По окончанию загрузки лицензий появится окно:

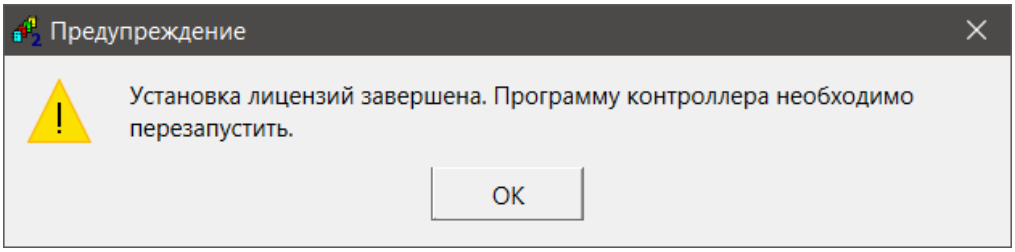


Рисунок 1.21 – Успешная загрузка лицензий в контроллер

Также заводской номер прибора можно ввести в свойстве модуля ***Серийный номер***, его можно добавить из выпадающего списка свойств снизу в окне свойств модуля.

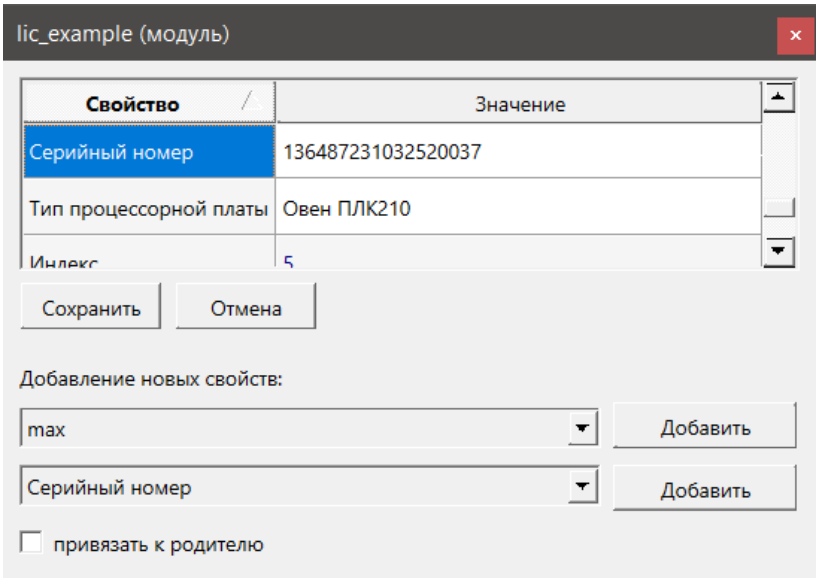


Рисунок 1.22 – Серийный номер в свойствах модуля

Если данное свойство прописано в модуле, то среда при записи лицензий не будет требовать ввести заводской номер контроллера.

Свойство ***Серийный номер*** также добавляется средой автоматически при вводе заводского номера в окне ***Введите серийный номер*** при вызове команды ***Записать лицензии***.

ВНИМАНИЕ
Для загрузки лицензий в контроллер через среду Полигон требуется доступ к сети Интернет.

Загрузка файлов лицензии через утилиту WinSCP (для ОС Windows) описана в разделе 1.3.2.2. Загрузка файлов лицензии через консоль Linux описана в разделе 1.3.2.3.

ВНИМАНИЕ
После загрузки дополнительных файлов лицензий для обновления списка доступных библиотек на странице web-конфигуратора ***Состояние/Обзор*** следует загрузить в контроллер проект с блоком ***OwenHWInfo*** из библиотеки ***paOwenIO***.

1.3.2.2 Загрузка файлов лицензий в ПЛК через WinSCP

При отсутствии возможности подключения к серверу **pa.ru** пакет лицензий можно загрузить через утилиту **WinSCP** (или через любой менеджер файлов). Утилита распространяется бесплатно и может быть загружена с сайта https://winscp.net/eng/download.php

Подключаться к контроллеру можно как через **USB Device** (IP адрес ПЛК при подключении через **USB Device** – **172.16.0.1**), так и через **Ethernet**.

После запуска утилиты следует настроить соединение по протоколу **SFTP**, указав IP адрес контроллера, имя пользователя – **root** и пароль (по умолчанию – **owen**, может быть изменен в web-конфигураторе). Чтобы подключиться к контроллеру, следует нажать ***Войти***.

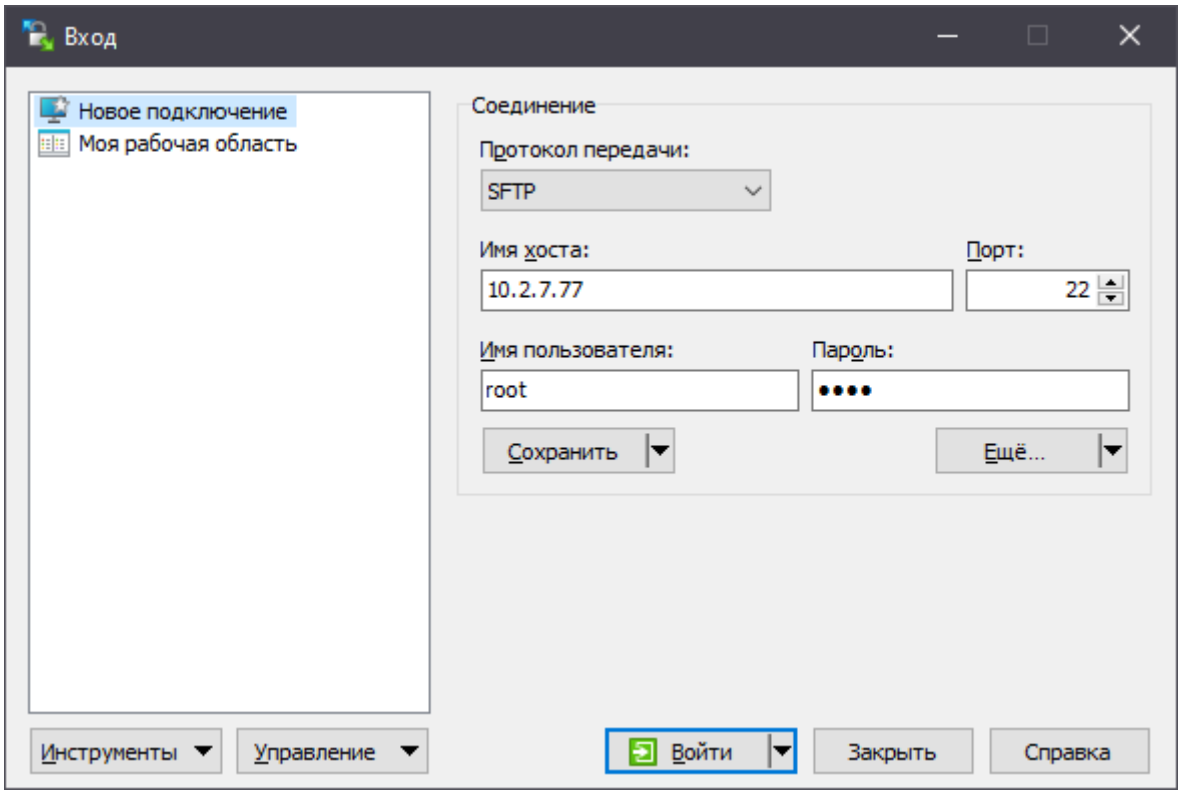


Рисунок 1.23 – Настройки подключения в WinSCP

Файлы лицензии необходимо загрузить в рабочую директорию контроллера ***/home/root***.

ВНИМАНИЕ
После загрузки дополнительных файлов лицензий для обновления списка доступных библиотек на странице web-конфигуратора ***Состояние/Обзор*** следует загрузить в контроллер проект с блоком ***OwenHWInfo*** из библиотеки ***paOwenIO***.

1.3.2.3 Загрузка файлов лицензий в ПЛК через терминал Linux

При отсутствии возможности подключения к серверу **pa.ru** пакет лицензий можно загрузить через терминал Linux по протоколу **SSH** с помощью команды ***scp***.

Подключаться к контроллеру можно как через **USB Device** (IP адрес ПЛК при подключении через **USB Device** – **172.16.0.1**), так и через **Ethernet**.

Синтаксис команды ***scp***:

```
$ scp опции пользователь1@хост1:файл пользователь2@хост2:файл
```

Пример команды для загрузки файла лицензии ***paSync*** при подключении ПЛК через **USB Device**:

```
$ scp /home/user/Documents/paSync root@172.16.0.1:/home/root
```

Для подключения к **SSH**-серверу контроллера необходимо указать пароль (по умолчанию – **owen**, может быть изменен в web-конфигураторе).

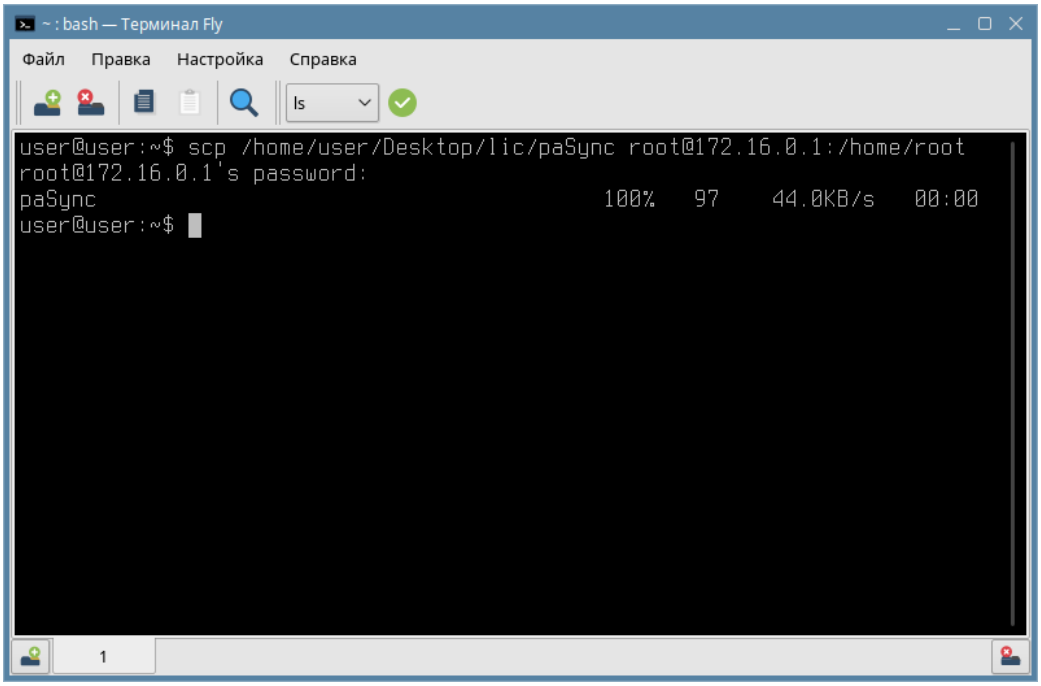


Рисунок 1.24 – Загрузка файла лицензии через терминал Linux

Проверить, что файлы лицензии загружены на контроллер можно подключившись к **SSH**-серверу контроллера и введя команду ***ls***.

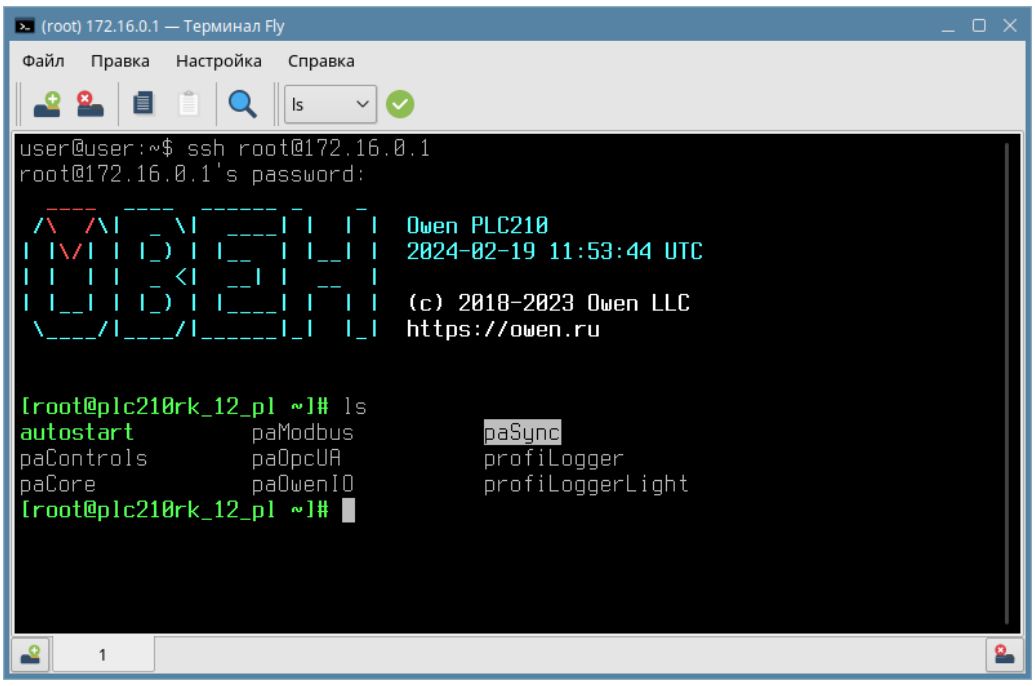


Рисунок 1.25 – Проверка загрузки файла лицензии

ВНИМАНИЕ
После загрузки дополнительных файлов лицензий для обновления списка доступных библиотек на странице web-конфигуратора ***Состояние/Обзор*** следует загрузить в контроллер проект с блоком ***OwenHWInfo*** из библиотеки ***paOwenIO***.

1.4 Библиотеки Полигон

1.4.1 Стандартные библиотеки

В среде Полигон реализованы следующие стандартные библиотеки:

Таблица 1.4 – Библиотеки Полигон

Библиотека	Описание	Документация
paCore	Основная библиотека. Содержит базовые блоки простых операций (арифметических, логических, триггеры и т.п.), а также некоторые блоки для конфигурации ПЛК, блоки работы с данными. В данной библиотеке описаны типы данных, используемые в других библиотеках	Описание библиотеки приведено в разделе 10
paOwenIO	Данная библиотека предназначена для конфигурирования ввода/вывода ПЛК ОВЕН, а также настройки системных функций ПЛК (внешних накопителей, зуммера, светодиодов и т.п.)	Работа с ОВЕН ПЛК. Библиотека paOwenIO
paOpcUA	Данная библиотека предназначена для реализации обмена ПЛК с другими устройствами/ПО по протоколу **OPC UA**	Обмен с верхним уровнем. Библиотека paOpcUA
paModbus	Данная библиотека предназначена для реализации обмена ПЛК с другими устройствами/ПО по протоколам **Modbus RTU**, **Modbus TCP**	Обмен по протоколу Modbus. Библиотека paModbus
paControls	Данная библиотека содержит основные блоки для реализации управления процессами и обработки сигналов с датчиков	Алгоритмы управления. Библиотека paControls
profiLogger, profiLoggerLight	Данные библиотеки содержат основные блоки для реализации архивирования	Архивирование и сохранение уставок
paSync	Данная библиотека предназначена для синхронизации проектов контроллеров, реализации горячего резервирования	Синхронизация проектов и реализация резервирования. Библиотека paSync
paIEC104	Данная библиотека предназначена для реализации обмена ПЛК с другими устройствами/ПО по протоколам стандартов **МЭК 60870-5-101** и **МЭК 60870-5-104**	Обмен по протоколам МЭК 60870-5. Библиотека paIEC104
paIEC850	Данная библиотека предназначена для реализации обмена ПЛК с другими устройствами/ПО по протоколу стандартна **МЭК 61850**	Обмен по протоколу МЭК 61850. Библиотека paIEC850

1.4.2 Установка библиотек

Рекомендуется хранить файлы библиотек в одном каталоге отдельно от папки ***Polygon2***, чтобы в случае переустановки среды не загружать и обновлять их повторно.

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

ВНИМАНИЕ
Путь к файлам библиотек не должен содержать кириллицу и пробелы.

Для добавления библиотеки в проект следует:

1. Перейти в меню ***Окна/Проекты***. В появившемся окне отобразится текущий проект и добавленные библиотеки.

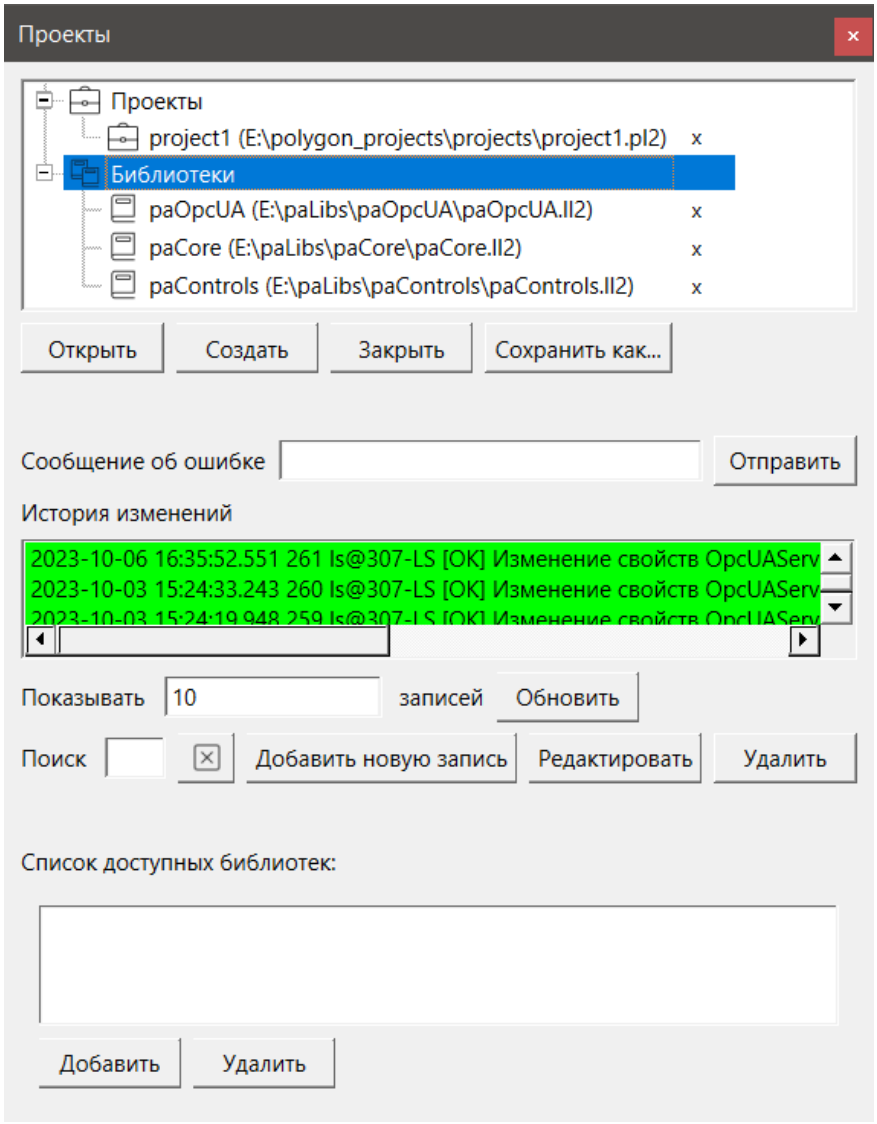


Рисунок 1.26 – Добавление библиотеки в проект

2. Нажать кнопку ***Открыть*** и перейти в папку с файлами библиотеки, которую необходимо добавить. Затем в выпадающем списке выбрать тип файла ***Библиотека Полигон 2 (*.ll2)***.

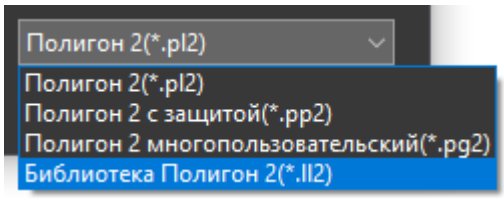


Рисунок 1.27 – Добавление библиотеки в проект

3. В окне появится файл библиотеки с расширением **.ll2**. Следует выбрать его и нажать открыть.

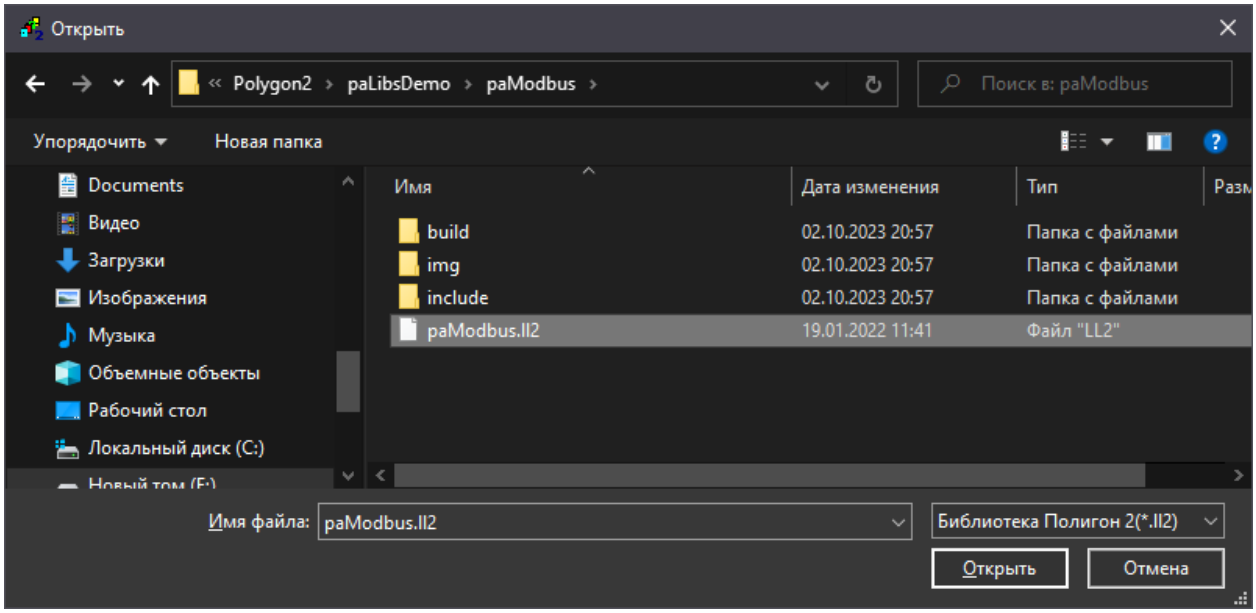


Рисунок 1.28 – Добавление библиотеки в проект

Если необходимо можно сразу открыть библиотеку в отдельном окне типа ***Дерево*** с помощью всплывающего окна.

Добавленная библиотека отобразится в окне ***Проекты***.

1.5 Обновление

В системном окне ***Окна/О программе*** отображаются установленные версии среды Полигон и библиотек.

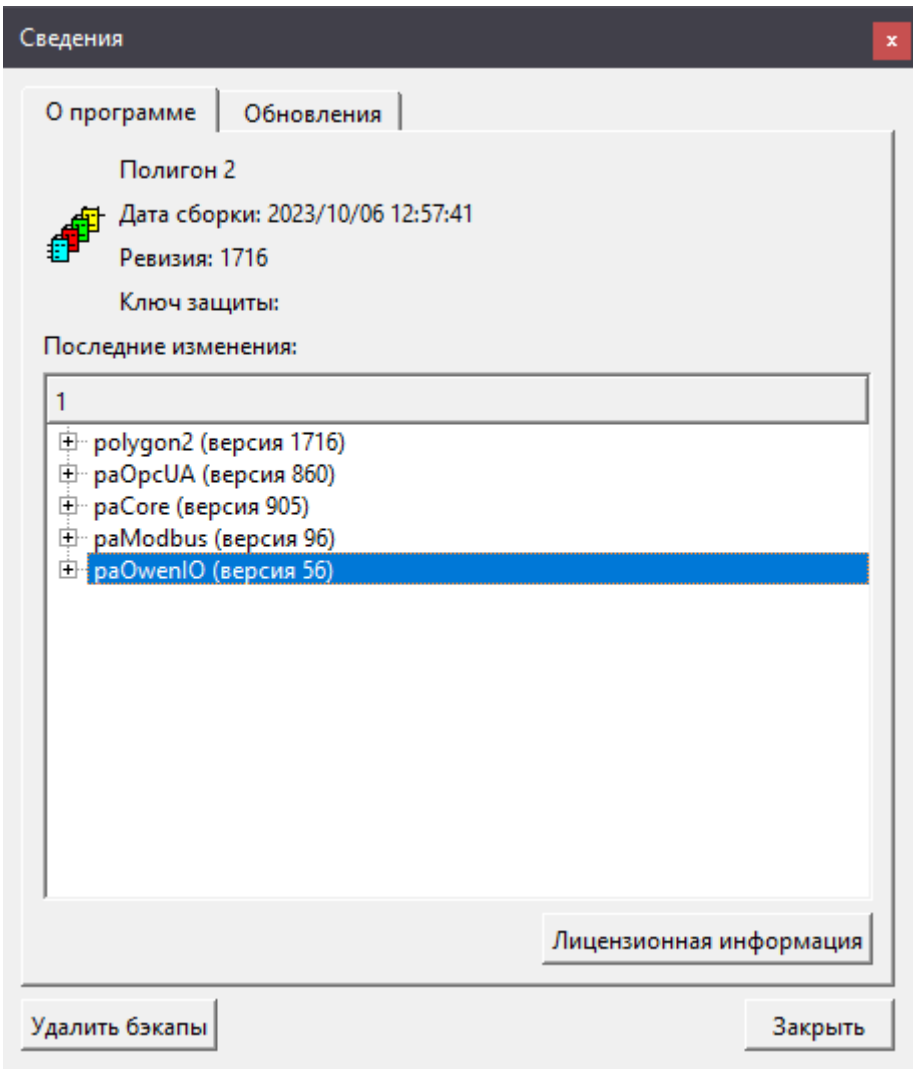


Рисунок 1.29 – Окно О программе

Во вкладке ***Обновления*** есть возможность проверить и скачать обновления.

Также при запуске среды, если на сервере есть обновления для Полигон или библиотек, будет выведено сообщение об этом.

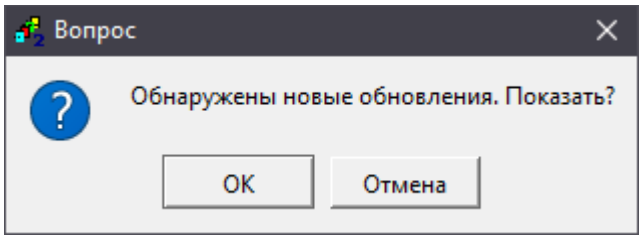


Рисунок 1.30 – Сообщение о наличии обновлений

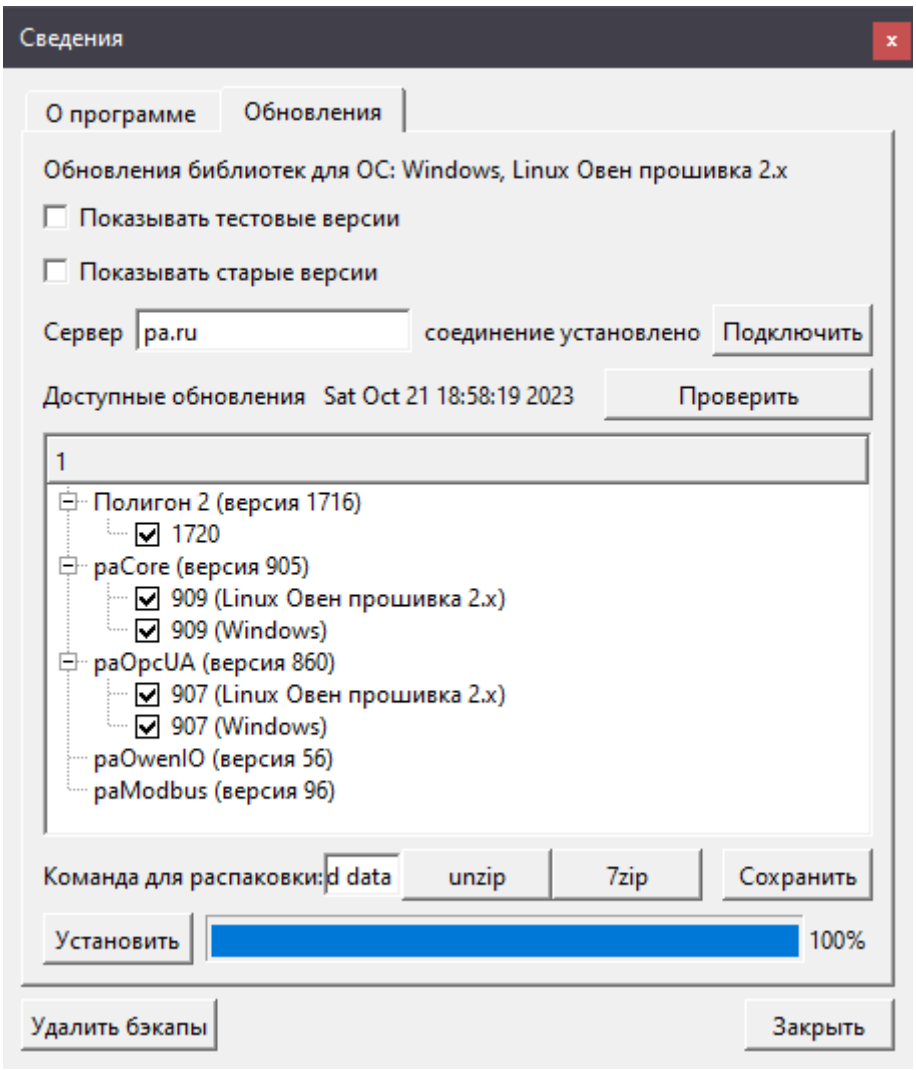


Рисунок 1.31 – Окно Обновления

Для того, чтобы установить последние версии среды и библиотек следует:

1. Убедиться, что соединение с сервером **pa.ru** установлено, нажав кнопку ***Подключить***.

ВНИМАНИЕ
Для установки обновлений для среды и библиотек необходимо иметь доступ к сети Интернет.

2. Нажать на кнопку ***Проверить***. В окне появятся версии среды и библиотек, доступные для установки.
3. Кнопками снизу выбрать распаковщик ***unzip*** или ***7zip*** и нажать ***Сохранить***.
4. Нажать ***Установить***.

Для установки обновлений среда будет перезапущена. Появится окно предупреждения о закрытии программы.

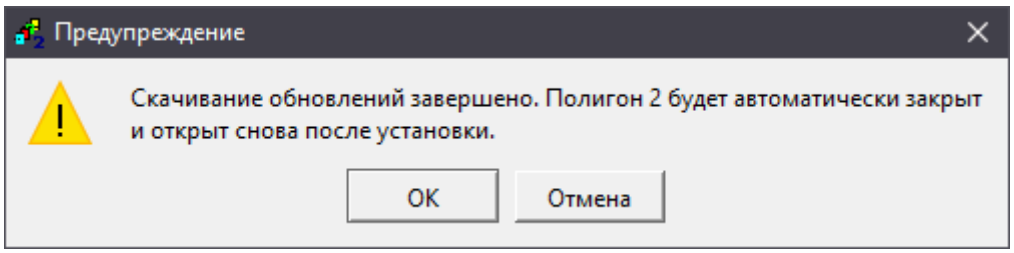


Рисунок 1.32 – Предупреждение о закрытии программы

После завершения установки обновлений среда снова откроется и в окне ***О программе*** отобразятся последние установленные версии среды и библиотек.

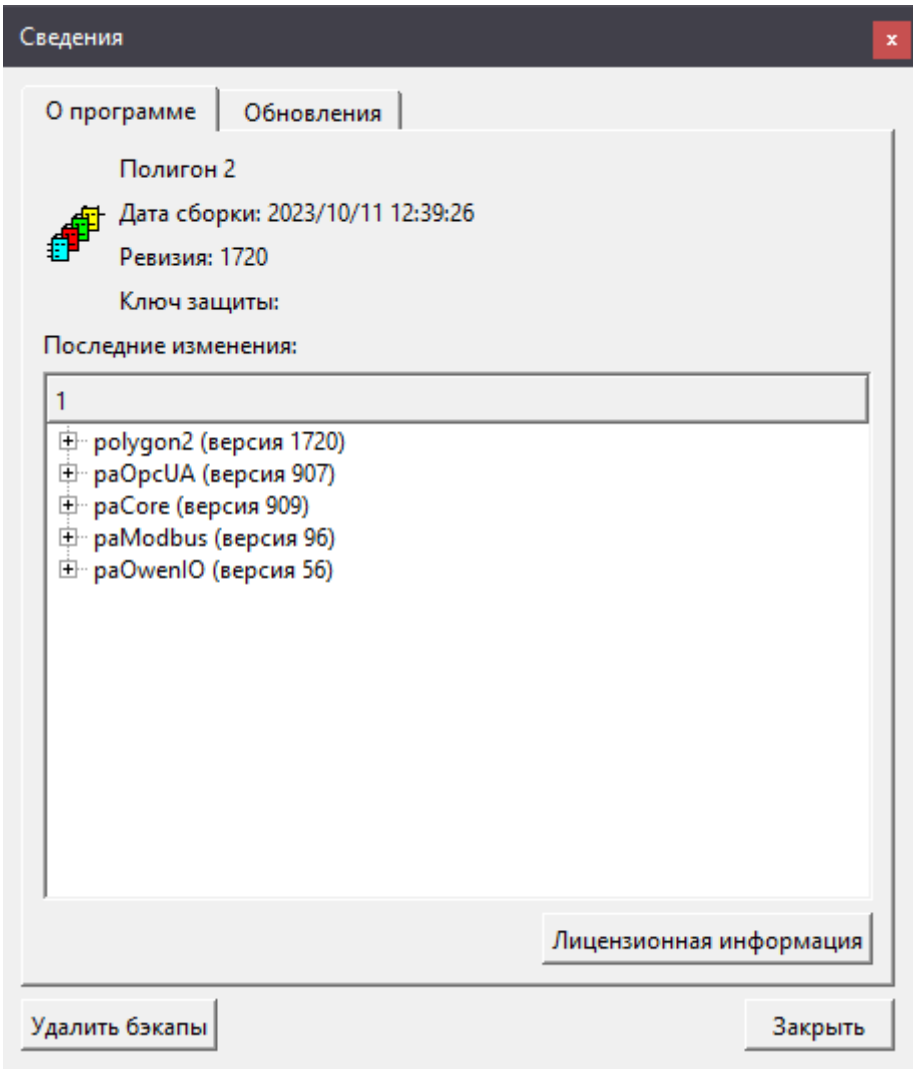


Рисунок 1.33 – Успешная установка обновлений

ВНИМАНИЕ
Если установка обновлений версий среды и библиотек не произошла, следует прописать полный путь к файлу распаковщика в окне ***Команда для распаковки*** (например, **"C:\Program Files (x86)\7-Zip\7z.exe"**).

ПРИМЕЧАНИЕ
При возникновении других проблем с установкой обновлений среды и библиотек следует обратиться на почту технической поддержки support@owen.ru

1.6 Контроль версий

Текущую версию запущенного проекта можно посмотреть во всплывающем тултипе при наведении мышью на модуль.

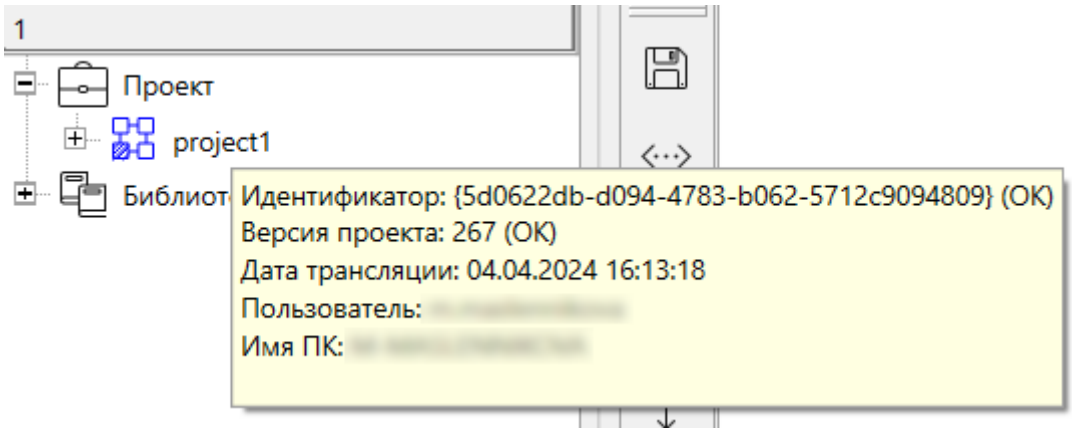


Рисунок 1.34 – Тултип с информацией о модуле

Также текущую версию запущенного проекта можно посмотреть в web-конфигураторе контроллера в разделе ***ПЛК/Информация***.

ВНИМАНИЕ
Для обновления информации о запущенном проекте в web-конфигураторе контроллера в разделе ***ПЛК/Информация*** в проекте должен быть добавлен блок ***OwenHWInfo*** из библиотеки ***paOwenIO***.

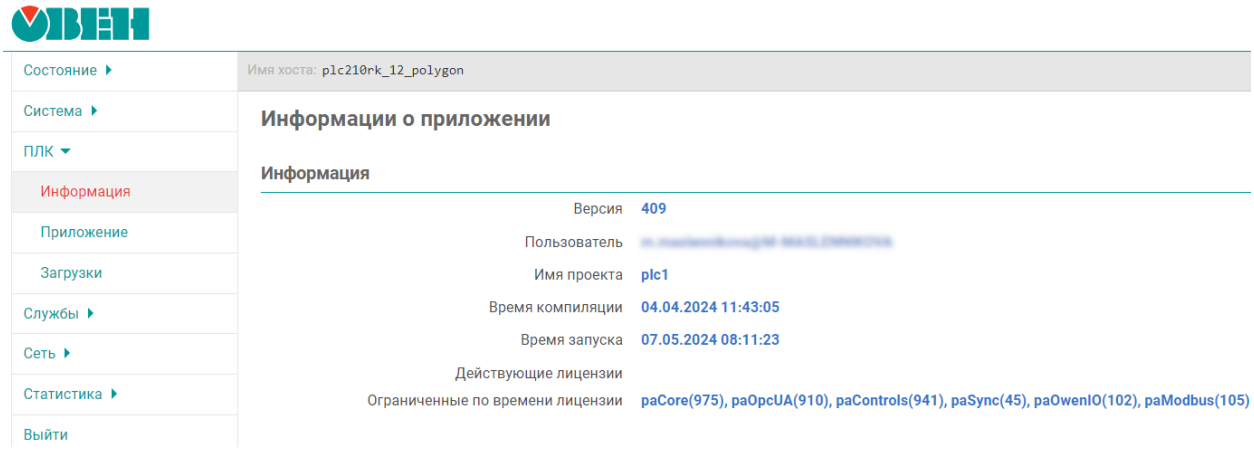


Рисунок 1.35 – Информация о запущенной программе в web-конфигураторе

В файле ***versions.txt*** в папке, куда транслируется проект, можно получить информацию о версии проекта и используемых в нем библиотек.

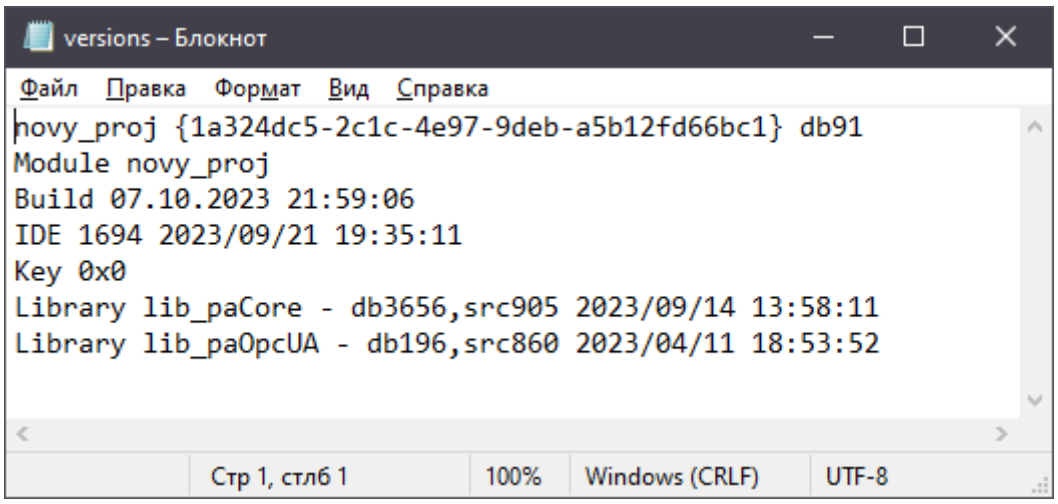


Рисунок 1.36 – Файл versions.txt

Если версия проекта в среде и на контроллере не совпадают, то при запуске отладчика иконка модуля будет отображаться с предупреждающим знаком. При наведении на него мышью появится тултип с информацией о предупреждении.

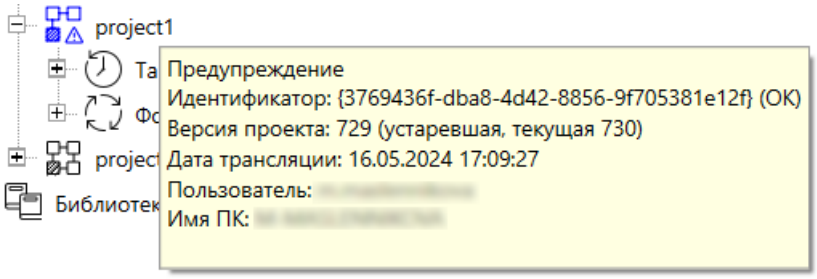


Рисунок 1.37 – Предупреждение о несовпадении версий проекта

При клике правой кнопкой мышки на запущенном модуле в дереве и выборе в открывшемся меню ***Показать все*** в окне трансляции (системное окно Прогресс) можно увидеть данные по версиям библиотек, используемых в проекте.

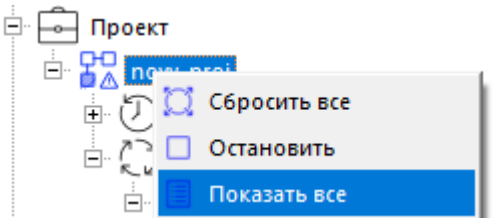


Рисунок 1.38 – Показать все

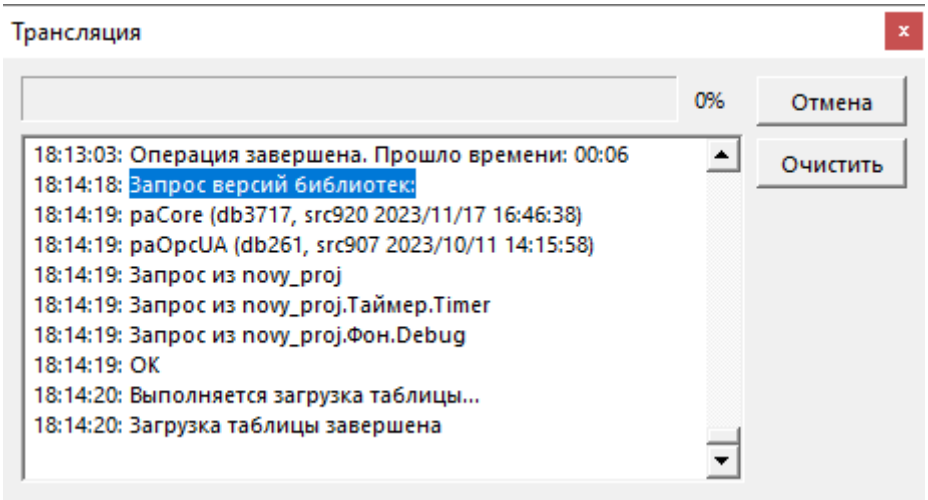


Рисунок 1.39 – Информация о версиях библиотек

1.7 Обслуживание

При работе со средой разработки, трансляции программы возможны появления ошибок. В таком случае программа выведет предупреждающее сообщение и предложит отправить сообщение об ошибке.

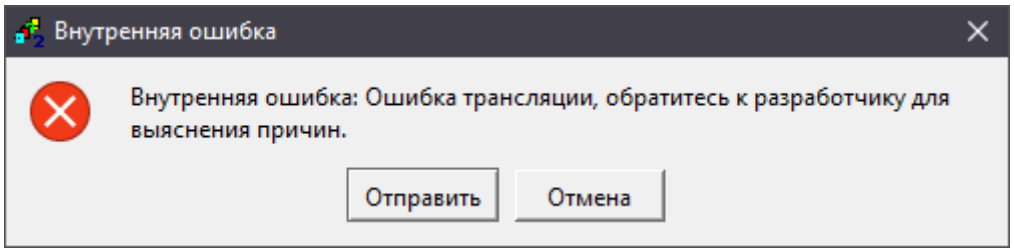


Рисунок 1.40 – Ошибка трансляции

При выборе ***Отправить*** лог ошибки и сам проект будут подготовлены на диске для отправки разработчикам. Данные файлы следует направить на почту support@owen.ru с подробным описанием проблемы.

При выборе ***Отмена*** окно будет закрыто. Информацию по ошибкам можно самостоятельно посмотреть в окне Прогресс. Ошибки в тексте выделены красным шрифтом.

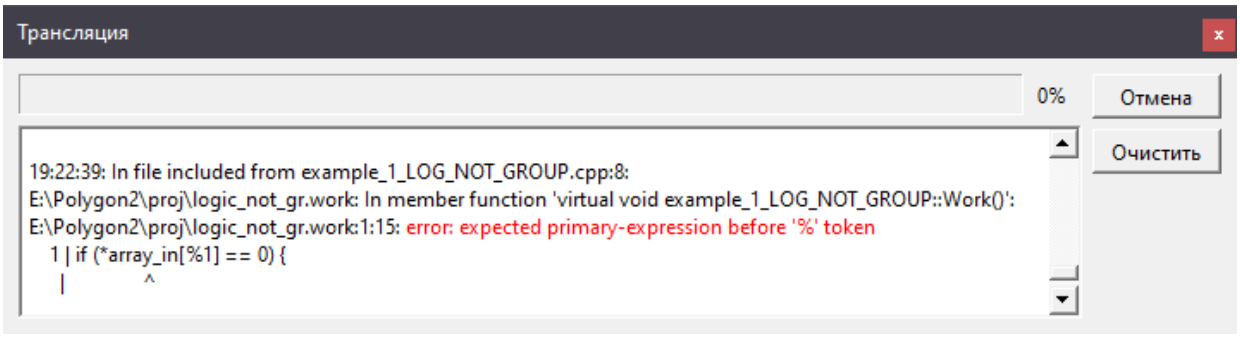


Рисунок 1.41 – Окно Прогресс при ошибке трансляции

2 Общие сведения о среде Полигон

2.1 Проект

Однопользовательский проект в Полигон представляет собой файл с расширением **.pl2**. В нем хранится все содержимое проекта за исключением дополнительных файлов, которые могут использоваться, например, при создании пользовательского составного блока на C++. Файлы проектов можно копировать и таким образом передавать другим разработчикам. Каждый проект рекомендуется размещать в отдельной папке, поскольку внутри нее среда разработки создает временные файлы при трансляции и копии проекта (бэкапы).

При редактировании проекта все изменения происходят в памяти, а в файл записываются при сохранении командой ***Сохранить***  в панели инструментов или комбинацией клавиш **Ctrl + s**. Последние **10** действий можно отменить командой ***Отменить***  или комбинацией клавиш **Ctrl + z** независимо от того был ли сохранен проект на диск или нет.

Автоматическое сохранение проекта по умолчанию производится каждые **5 минут**. Период сохранения проекта можно изменить в меню Экран/Настройки (при установке **0** автосохранения производится не будут). При каждом сохранении на диск создается копия проекта (бэкап) с таким же именем плюс суффикс с датой и временем сохранения.

О создании многопользовательского проекта см. в разделе 8.5.

2.2 Структура проекта

Проект в среде разработки Полигон имеет иерархическую структуру, которую можно увидеть в представлении ***Дерево***. Узлы дерева могут быть разных типов. Основной узел с точки зрения создания программы для контроллера – ***Модуль***.

В свойствах модуля задаются параметры подключения к контроллеру. Внутри модуля располагаются графическая структура алгоритмов и настройки для контроллера, из модуля транслируется исполняемый файл.

Проект может содержать несколько модулей.

Внутренняя структура модуля жестко определена: ***Модуль > Место работы > Программа > Страница > Функциональный блок***.

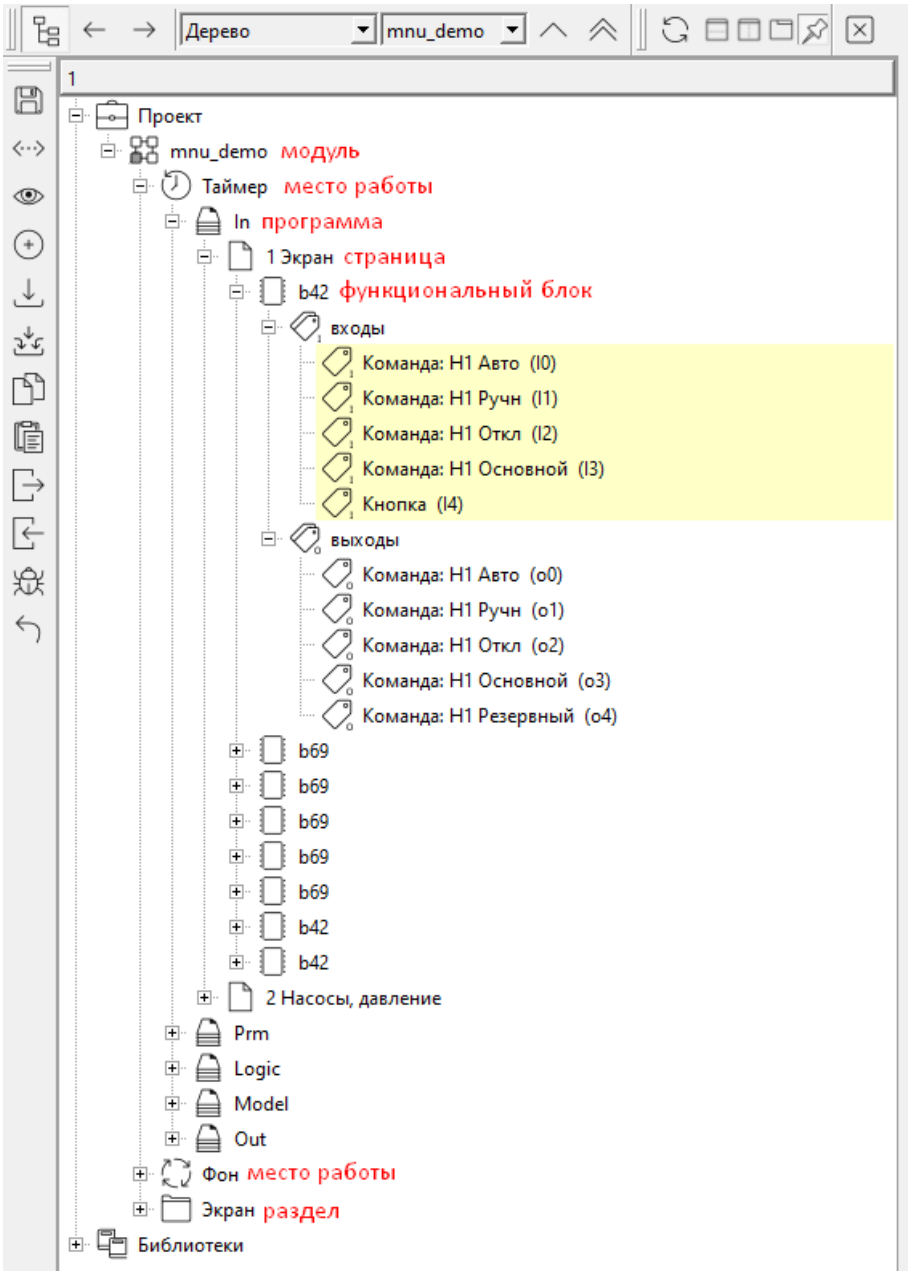


Рисунок 2.1 – Структура проекта

Место работы характеризует способ выполнения находящихся в нем программ, т.е. поток, в котором программы будут выполняться.

ПО контроллера формируется таким образом, что выполнение каждого функционального блока возможно в одном из двух потоков: таймерном прерывании – место работы ***Таймер*** или в фоне – место работы ***Фон***.

Часть алгоритма, помещенная в таймерном прерывании, всегда выполняется в режиме реального времени с заданной периодичностью (периодичность задается свойством места работы ***Таймерный промежуток***) – это поток с самым высоким приоритетом.

Необходимо соизмерять размер программ (количество функциональных блоков), находящихся в таймере, со временем таймерного цикла. При недостаточном интервале

таймерного цикла, возможно возникновение ошибки времени исполнения – **Time Out**. Данная ошибка повлечет за собой остановку программы (отслеживать реальное время выполнения можно с помощью функционального блока SysInfo).

Существуют функциональные блоки, такие как логические таймеры, которые целесообразно помещать только в ***Таймер***.

ПРИМЕЧАНИЕ
В проекте можно добавить до двух таймерных потоков. При этом второй таймерный поток – это ***Ввод-вывод***. Подробнее о создании потоков в разделе 4.2.

Фоновый поток представляет собой бесконечный цикл, который выполняется с более низким приоритетом в оставшееся от таймерного потока время. В фон необходимо помещать блоки некритичные ко времени исполнения и не привязанные к внешним событиям.

Большинство блоков, обеспечивающих обмен по интерфейсам, осуществляющих запись в файлы и т.п., целесообразно помещать только в ***Фон***.

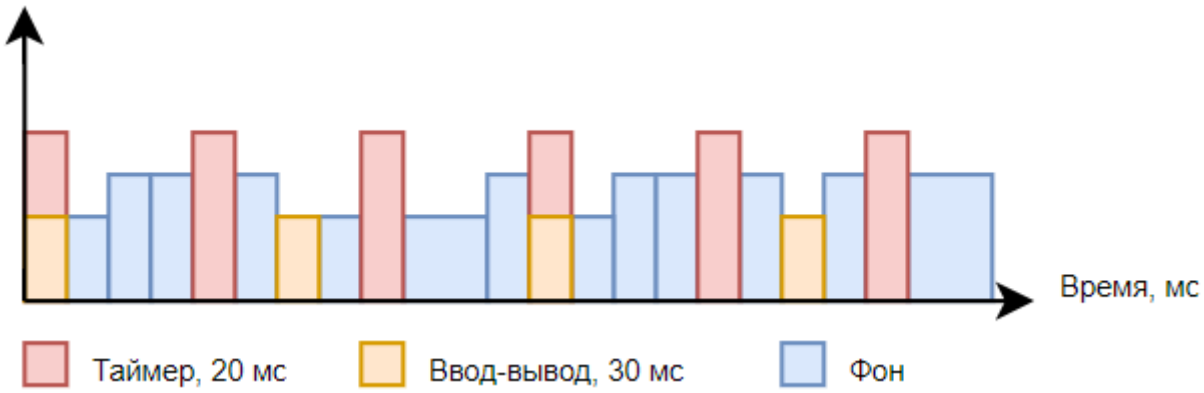


Рисунок 2.2 – Диаграмма выполнения потоков

Программа – это относительно независимая алгоритмическая задача или группа однотипных задач.

Ради простоты дальнейшей поддержки проекта следует избегать двух крайностей: малого числа слишком длинных программ (т.е. с большим числом страниц) и слишком большого числа коротких программ.

Имя программы должно состоять только из символов латинского алфавита и нижнего подчеркивания **«_»**, поскольку это имя при трансляции определяет имя файла исходных текстов и объектного файла. Важно давать программам осмысленные имена, чтобы впоследствии было проще понять их назначение. Желательно всегда присваивать разные порядковые номера выполнения программам одного потока.

Для удобства просмотра программы разбиты на ***Страницы***. Количество страниц определяется пользователем.

Страница, в свою очередь, содержит ***Функциональные блоки***, соединенные между собой связями.

2.3 Поток данных

Программа, получаемая после трансляции, имеет двухпоточную структуру и сторожевой таймер.

При создании алгоритма функциональные блоки могут быть размещены либо в фоновом, либо в таймерном потоке. Для этого блок создается в определенном ***Месте работы***.

Порядок выполнения алгоритма внутри потока определяется следующим образом:

1) порядок выполнения программы (свойство ***Номер***);
2) номер страницы (свойство ***Номер***);
3) порядок выполнения блока на странице (свойство ***Порядок***).

Если блоки на одной странице имеют одинаковый порядок выполнения, то их очередность определяется положением блока на странице – слева направо, сверху вниз. Во избежание логических ошибок, вызванных перемещением блоков, рекомендуется всегда задавать разные порядки выполнения для блоков, расположенных на одной странице.

Поток данных осуществляется исключительно за счет проведения связей между входами и выходами функциональных блоков.

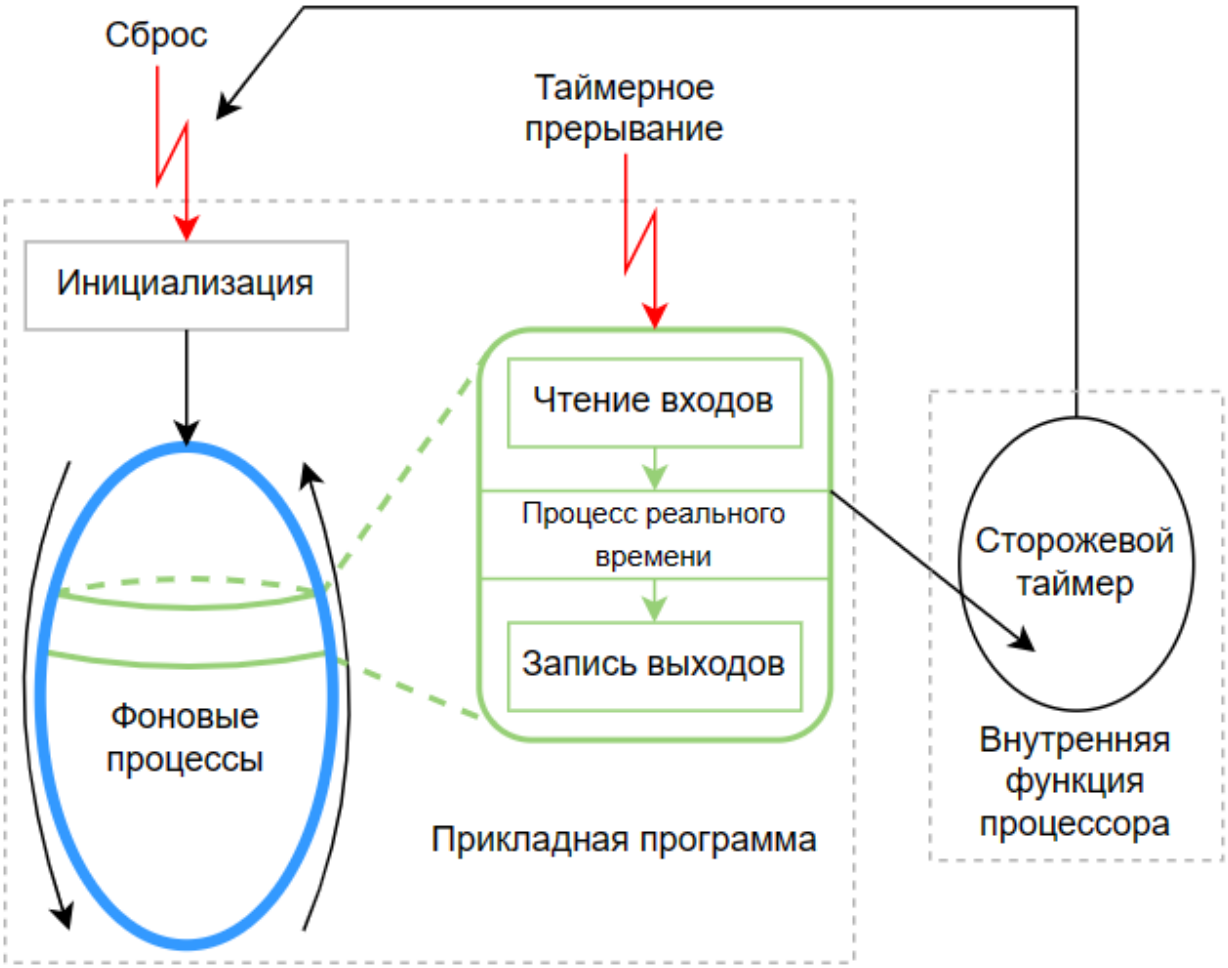


Рисунок 2.3 – Поток данных

2.4 Страница

Для удобства представления ***Программа*** делится на ***Страницы***.

Страница имеет порядковый номер (свойство ***Номер***), который влияет на порядок выполнения блоков (поток данных) и может иметь другие свойства (***Комментарии***, ***Размер листа***).

Страницы можно копировать, переносить в другую программу и удалять (соответствующие команды выбираются в контекстном меню при перетаскивании страницы в дереве). При удалении страницы все блоки, находящиеся на ней, также удаляются.

В дереве проекта страница отображается со своим номером и названием, которое задается в свойстве ***Комментарии***.

Графически страница представляется в виде двух областей:

1. ***Рабочая область*** содержит функциональные блоки. Все манипуляции с функциональными блоками (перемещение, создание, удаление, копирование, проведение связей) проводятся здесь. Команды редактирования доступны из контекстного меню блоков и пустого пространства страницы (по клику правой кнопки мыши).

2. ***Поля*** – область, содержащая информацию о входах и выходах блоков, находящихся на других страницах, связанных с входами и выходами блоков данной страницы.

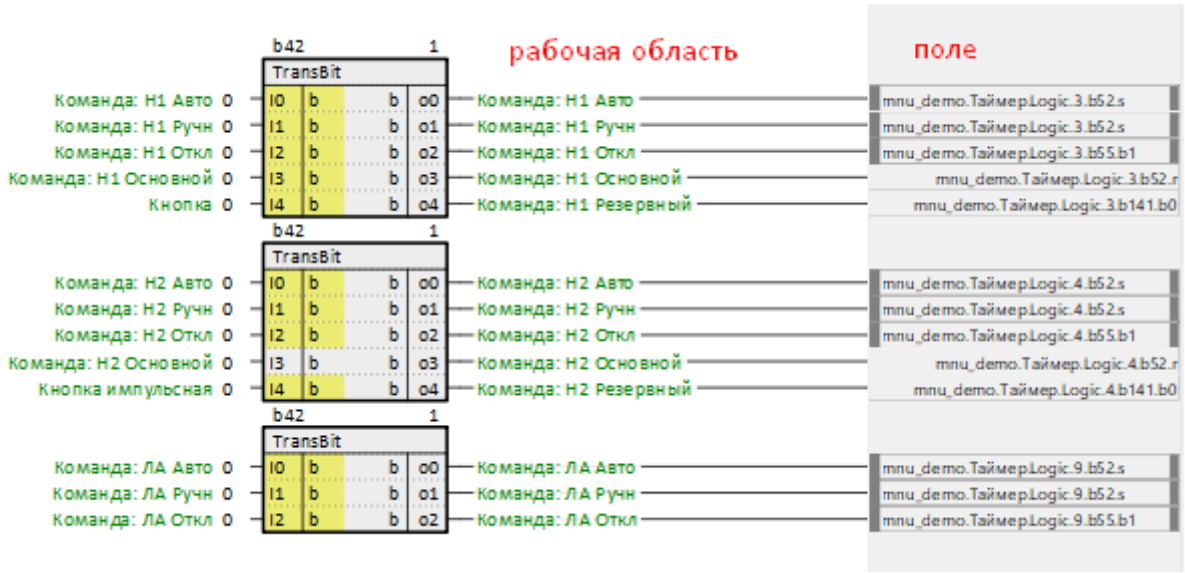


Рисунок 2.4 – Страница

Вход или выход на поле обозначается следующим образом:

Рисунок 2.5 – Обозначение входа/выхода на поле страницы

Для того чтобы быстро найти вход или выход, указанный на поле, необходимо щелкнуть по нему левой кнопкой мыши (для правого поля может понадобиться выбрать один из входов из списка). После этого в текущем рабочем окне появится страница, которой принадлежит данный вход или выход.

Размер страницы можно поменять при помощи свойства ***Размер листа***, однако следует учитывать, что слишком большая страница, на которой размещено много блоков, может долго открываться.

Масштаб каждой открытой страницы можно менять при помощи колеса мыши при нажатом **Ctrl**. Масштаб открытых страниц меняется относительно общего масштаба всего приложения, который задается комбинациями **Ctrl +** и **Ctrl -**.

Для перемещения по странице можно использовать ползунки справа – перемещение вверх-вниз и снизу – влево-вправо. Также можно использовать колесо мыши – перемещение вверх-вниз и **Shift** + колесо мыши – влево-вправо.

2.5 Функциональный блок

Функциональный блок представляет из себя элементарный алгоритм. Программа в среде Полигон состоит только из функциональных блоков, входы и выходы которых связаны между собой связями.

Типы функциональных блоков описаны в библиотеках (файлы с расширением **.ll2**) и реализованы как классы **C++**. Информацию о работе блока можно получить из справки библиотеки, выделив блок и нажав **F1**.

Графическое отображение функционального блока на странице несет информацию об его свойствах и свойствах его входов и выходов.

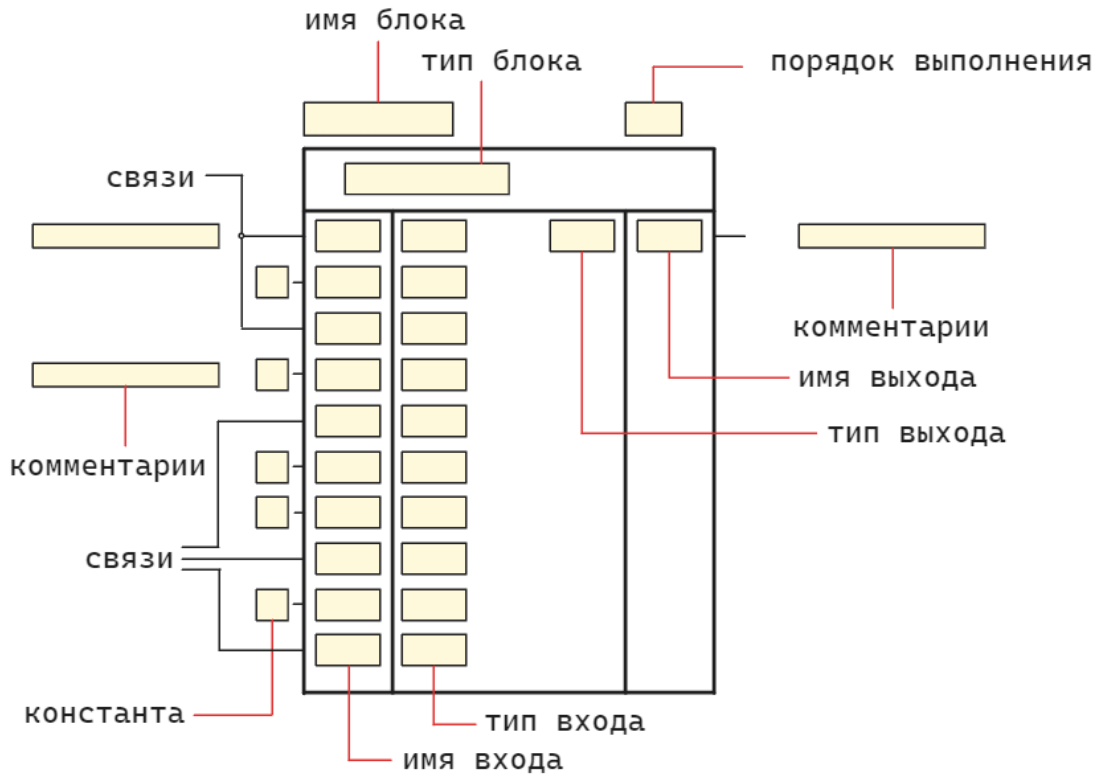


Рисунок 2.6 – Представление функционального блока на странице

Таблица 2.1 – Основные свойства функционального блока и его входов/выходов

Свойство	Описание
Имя блока	Любая строка, удобная для навигации по проекту
Порядок выполнения	Положительное целое число, характеризующее порядок выполнения блока на странице (поток данных), для корректного выполнения программы рекомендуется не использовать одинаковые порядки выполнения для блоков на одной странице
Тип блока	Имя типа функционального блока, тип блока выбирается при создании и не может быть изменен в процессе работы
Имя входа	Строка, уникальная для данного типа блока (может содержать число – индекс в массиве, в случае циклических входов)
Тип входа	Сокращенное имя типа данных
Константа	Выражение, формат которого зависит от типа входа, константа задается на входе, не связанном ни с каким выходом и определяет, в этом случае, значение входа при выполнении программы. Полигон не ограничивает возможные форматы констант. Если константа будет задана неправильно (например, на вход типа **flt** (float) будет задана строковая константа **"s"**), при трансляции возникнет ошибка
Имя выхода	Строка, уникальная для данного типа блока (может содержать число – индекс в массиве, в случае циклических выходов)
Тип выхода	Сокращенное имя типа данных
Связи	Связи между входами и выходами блоков на одной странице или между блоками, расположенных на разных страницах
Комментарии	Комментарии, относящиеся к входам/выходам, комментарии отображаются в дереве проекта

Свойства блока и его входов/выходов можно менять в окне Свойства. Кроме этого, для изменения имени, порядка выполнения, констант на входах и комментариев достаточно дважды кликнуть мышкой на соответствующем параметре блока в представлении страница и ввести новое значение.

2.5.1 Входы

Входы функционального блока могут быть следующих типов:

1. ***Циклические*** – входы, количество которых может быть переменным. Входы добавляются командой ***Создать*** в контекстном меню блока.

2. ***Циклические группы*** – входы, объединенные в группы, количество которых может быть переменным (аналогично циклическим входам). Группа соответствует структуре в **C++**.

3. ***Нециклические*** – входы, количество которых не может быть изменено и определяется исключительно типом функционального блока.

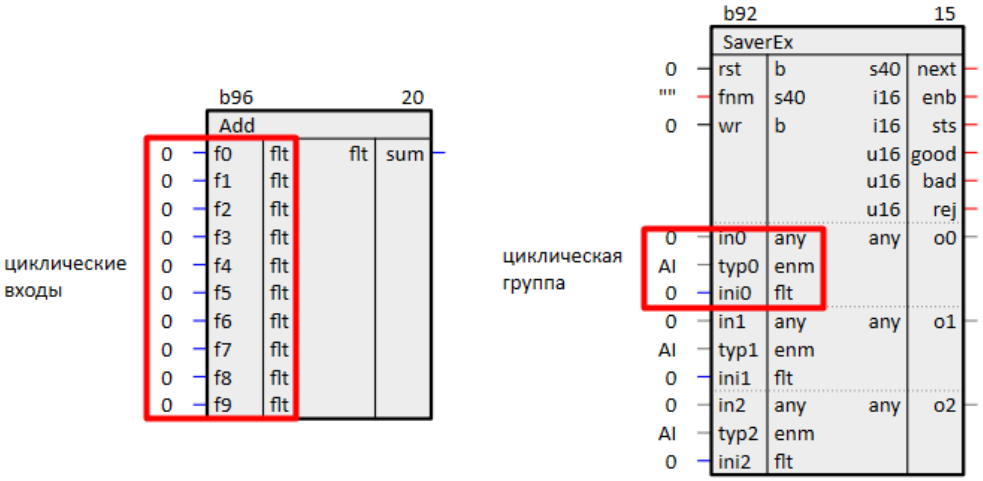


Рисунок 2.7 – Типы входов ФБ

По отношению к возможности проведения связи входы делятся на следующие типы (задается свойством входа ***Тип связей***):

1. ***Обычный вход*** – вход, на котором может быть задана константа или проведена связь (свойство ***Тип связей*** **= 0**).
2. ***Константный вход*** – вход, на котором должна быть задана константа и не может быть проведена связь (***Тип связей*** **= 1**).
3. ***Необязательный вход*** – вход, на который может (необязательно) быть проведена связь и не может быть задана константа. При отсутствии связи вход не используется в программе. Данный вход обычно обозначается символом ***#*** в поле констант и имеет свойство ***Тип связей*** **= 2**.
4. ***Обязательный вход*** – вход, на который должна быть проведена связь и не может быть задана константа. Если на вход данного типа не будет проведена связь, то при трансляции будет выдана ошибка. Данный вход обозначается обычно символами ***???*** в поле констант и имеет свойство ***Тип связей*** **= 3**.

2.5.2 Выходы

Выходы функционального блока могут быть двух типов:

1. ***Циклические*** – выходы, количество которых может быть переменным. Выходы добавляются командой ***Создать*** в контекстном меню блока. Количество циклических выходов может зависеть от количества входов: при изменении количества входов меняется и количество выходов и наоборот.

2. ***Нециклические*** – обычные выходы, количество которых не может быть изменено.

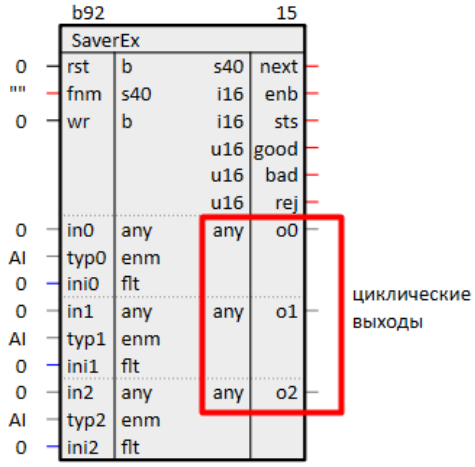


Рисунок 2.8 – Типы выходов ФБ

2.5.3 Связи

Для того чтобы провести ***Связь***, необходимо щелкнуть левой кнопкой мыши на входе или выходе ФБ и, не отпуская левую кнопку мыши, подвести указатель к другому входу или выходу ФБ. При отпускании левой кнопки мыши связь будет проведена.

При создании связи между блоками, находящимися на разных страницах, необходимо открыть эти страницы в рабочих окнах и провести связь как указано выше, при этом на каждой странице будут проведены связи соответственно в левое и правое поле.

При проведении связи учитываются типы данных входа и выхода, и, если типы оказываются несовместимы (нет соответствующего блока преобразования типов), выдается сообщение об ошибке и связь не проводится. Невозможно проведение связи на константный вход.

Цвет связей отображает тип данных входа:

- ***черный*** – булевое;
- ***красный*** – целое;
- ***синий*** – вещественное;
- ***серый*** – строковое.

Подробнее о проведении связей см. в разделе 4.4.

2.6 Составной блок

Часть алгоритма, повторяющаяся в проекте несколько раз, может быть выделена в новый ***Составной функциональный блок***. Обычно это упрощает отладку и модификацию этой части алгоритма, а также уменьшает время трансляции проекта. Составной блок при трансляции превращается в класс.

Подробнее о создании составных блоков см. в разделе 8.1.

3 Интерфейс Полигон

3.1 Окна

В Полигоне существует два вида окон – ***Окна представления*** и ***Системные окна***.

Окна представления предназначены для редактирования и отладки проекта и их нельзя вынести поверх других окон в среде.

Системные окна предназначены для работы со средой разработки: подключения к контроллеру, управления проектами и библиотеками, поиска по проекту и т.д. Системные окна можно как встраивать рядом с окнами представления, так и выносить поверх них.

3.1.1 Окна представления

Окна представления могут быть следующих типов:

Таблица 3.1 – Типы окон представления

Тип окна представления	Пиктограмма	Функция
Дерево		Навигация по проектам и библиотекам
Страница		Отображение страницы с функциональными блоками
Составной блок		Отображение содержания составного блока
Функциональный блок		Графическое отображение функционального блока
Редактор		Отображение файлов-ссылок для редактирования
Таблица		Отображение информации о функциональных блоках и их входах/выходах в табличном виде
График		Отображение значений входов/выходов в виде графиков
Экран отладчика		Отображение значений входов/выходов в виде графических элементов

Новое окно представления можно добавить двумя способами:

1. Открыть меню ***Окна/Новое окно***, выбрать расположение нового окна относительно открытых окон – в окне ***Выберите место для нового окна*** отображается прямоугольниками серого цвета, выбрать тип нового окна из списка.

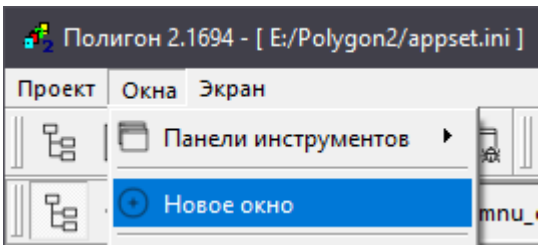


Рисунок 3.1 – Создание нового окна представления

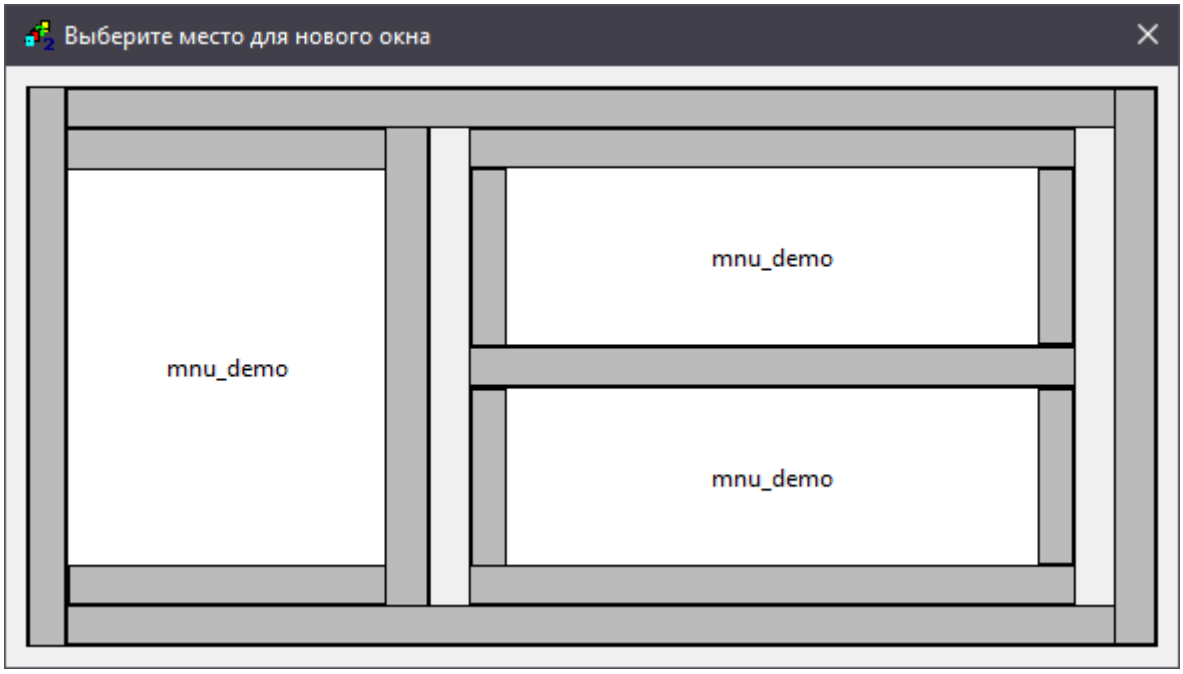


Рисунок 3.2 – Выбор места для окна представления

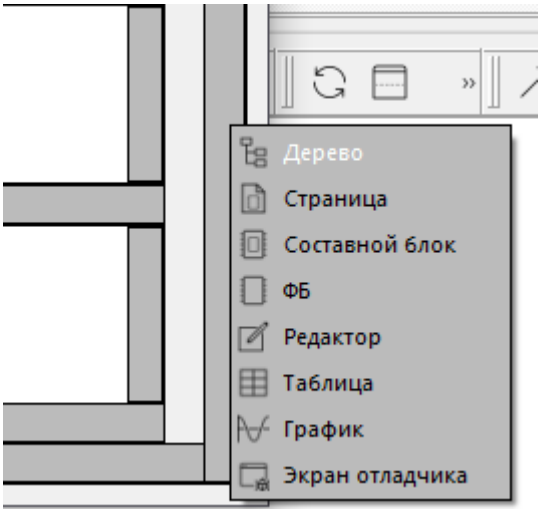


Рисунок 3.3 – Выбор типа окна представления

2. Выбрать тип нового окна на панели ***Представления***, выбрать положение нового окна относительно открытых окон. Панель ***Представления*** добавляется через меню ***Окна/Панели инструментов/Представления***.

Рисунок 3.4 – Панель Представления

3.1.1.1 Панели инструментов для работы с окнами представления

Для работы с окнами представления используются следующие панели, которые можно добавить или убрать с помощью установки или снятия флагов в меню ***Окна/Панели инструментов***:

1. ***Инструменты*** – панель основных инструментов для работы с окнами представления.

Рисунок 3.5 – Панель Инструменты

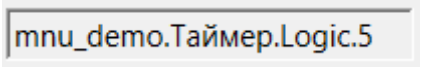
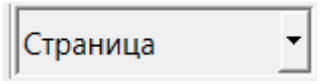
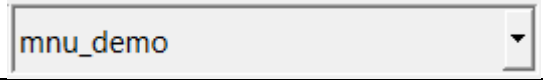
Таблица 3.2 – Элементы панели Инструменты

Элемент	Пиктограмма	Функция
Сохранить или **Ctrl + s**		Сохранить изменения на странице
Свойства		Открыть панель Свойства страницы или элемента
Показать		Открыть окно с выбранным элементом (страницей, ФБ, входом, выходом и т.д.)
Создать		Открыть окно создания нового элемента (страницы в дереве, ФБ на странице и т.д.)
Транслировать все		Запустить трансляцию проекта
Копировать		Копировать выбранные элементы
Вставить		Вставить скопированные элементы
Экспорт		Экспортировать элементы для редактирования в MS Excel (см. раздел 8.3)
Импорт		Импортировать элементы из MS Excel (см. раздел 8.3)
Отладчик		Запустить отладчик
Отменить или **Ctrl + z**		Отменить последнее действие (программа сохраняет в памяти **10** последних действий)

2. ***Навигация*** – панель, обеспечивающая навигацию по окнам.

Рисунок 3.6 – Панель Навигация

Таблица 3.3 – Элементы панели Навигация

Элемент	Пиктограмма	Функция
Текущее окно		Сделать окно текущим. При установке данное окно становится рабочим – в него будут отображаться страницы при двойном нажатии в ***Дереве*** и при нажатии на полях страниц
Списки выбора	В соответствии с типом окна	Отобразить список типов окон представления
Назад и ***Вперед***		Открыть предыдущую или последующую открытую страницу
Путь		Путь до страницы в дереве проекта (для окон типа ***Страница***)
Представления		Список типов окон представления. При смене окна его тип выбирается в этом выпадающем списке
Проекты		Список добавленных проектов и библиотек
На уровень вверх или ***Сброс***		Переместиться на уровень вверх или переместиться на верхний уровень (для окон типа ***Дерево***)
Предыдущая или ***Следующая страница***		Открыть предыдущий или следующий элемент в дереве в окнах типа ***Страница***, ***Составной блок***, ***Функциональный блок***, ***Редактор***, ***Таблица***

3. ***Вид*** – панель, определяющая взаимное расположение окон.

Рисунок 3.7 – Панель Вид

Таблица 3.4 – Элементы панели Вид

Элемент	Пиктограмма	Функция
Обновление		Обновить вид окна
Вертикальное размещение		Разместить окна вертикально
Горизонтальное размещение		Разместить окна горизонтально

Продолжение таблицы 3.4

Вкладки		Разместить окна по вкладкам. При открытии нового окна появляется вариант поместить его как вкладку
Фиксировать		Закрепить текущее окно. В закрепленное окно нельзя отобразить другие окна (пиктограмма ***Текущее окно*** пропадает). Это полезно, например, для окон типа ***Дерево***, в которых всегда отображается структура проекта

4. ***Редактирование*** – панель для окон типов ***Страница*** и ***Экран отладчика*** для добавления графических элементов: ***стрелка***, ***фон***, ***текст*** (подробнее см. в разделе 4.6).

Рисунок 3.8 – Панель Редактирование

Таблица 3.5 – Элементы панели Редактирование

Элемент	Пиктограмма	Функция
Стрелка		Разместить стрелку
Фон		Разместить фон
Текст		Разместить текст

3.1.1.2 Дерево

Окно представления ***Дерево*** позволяет осуществлять навигацию по проектам и библиотекам. Переключение между проектами и библиотеками происходит в выпадающем списке в верхнем меню окна.

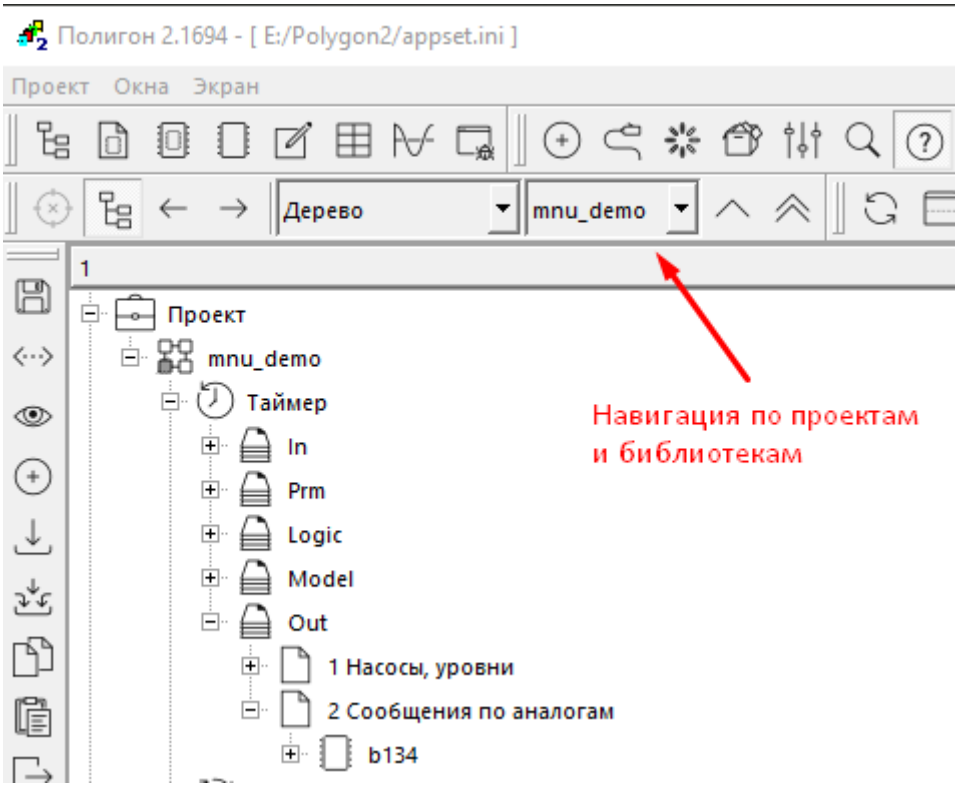


Рисунок 3.9 – Окно представления Дерево

В окне ***Дерево*** отображается структура выбранного проекта и реализованы раскрывающиеся списки на модулях, местах работы, программах, страницах, функциональных блоках и библиотеках.

В окне ***Дерево*** также можно редактировать проект – создавать, удалять, копировать части проекта и библиотек (см. раздел 4).

3.1.1.3 Страница

Окно представления ***Страница*** предназначено для отображения страницы с функциональными блоками.

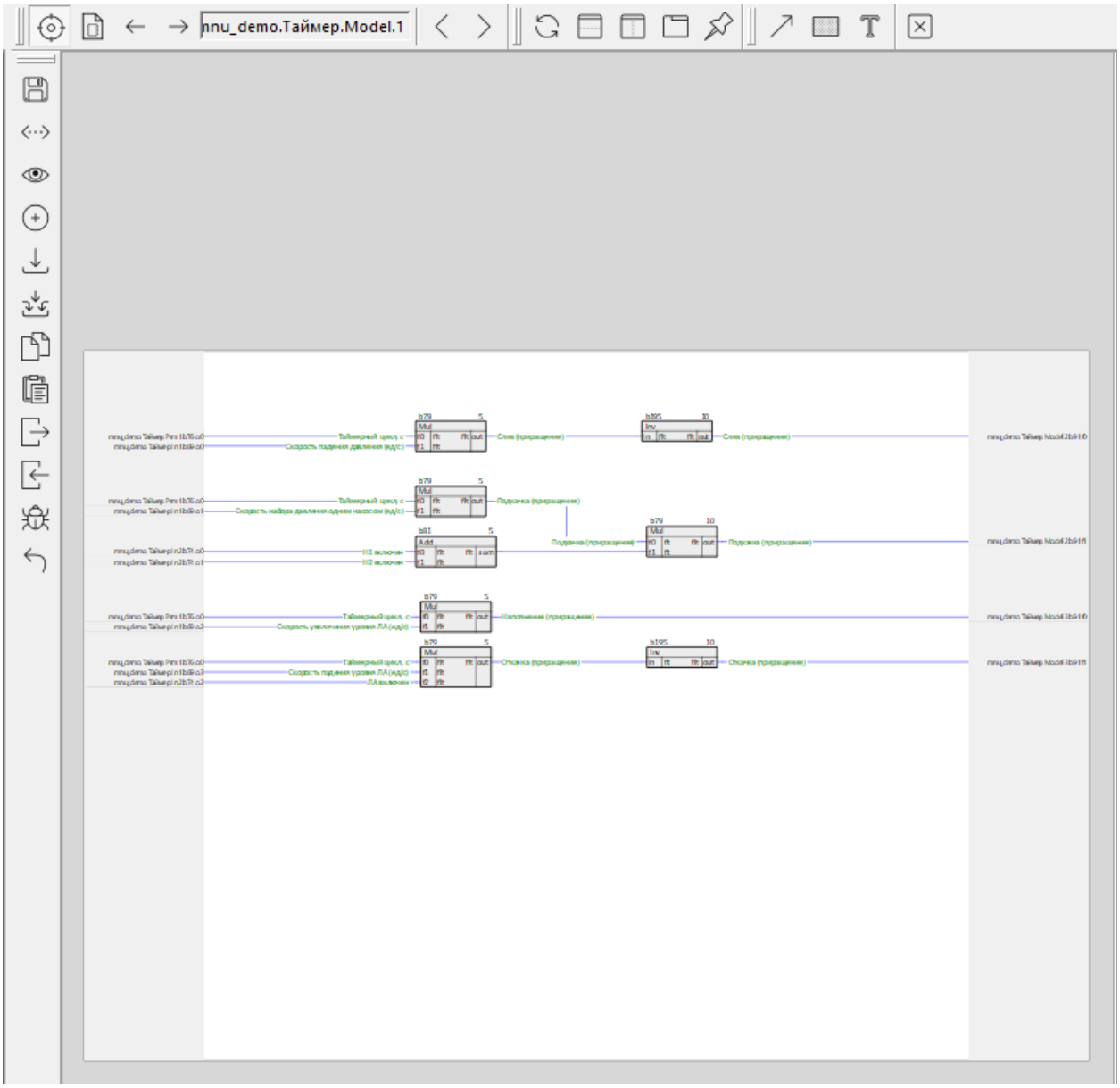


Рисунок 3.10 – Окно представления Страница

В окне ***Страница*** производится редактирование алгоритма: создание функциональных блоков, проведение/удаление связей между блоками, задание констант у входов, а также написание комментариев.

Во время отладки на странице отображаются текущие значения входов и выходов (если необходимо, их можно подменить).

Таблица 3.6 – Важные свойства страницы

Свойство	Описание
Комментарии	Название страницы в дереве
Номер	Номер в дереве, данное свойство определяет порядок выполнения программы (поток данных)
Размер листа	Размер страницы: от **A4** до **A0** (по умолчанию установлен размер **A4**)

3.1.1.4 Составной блок

Окно представления ***Составной блок*** предназначено для отображения содержания составного блока с алгоритмом, написанным на функциональных блоках.

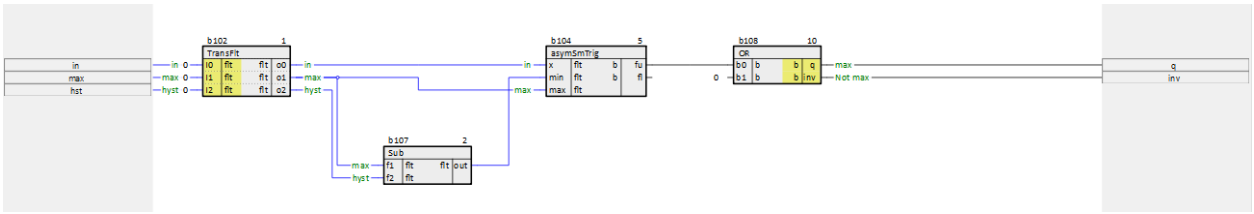


Рисунок 3.11 – Окно представления Составной блок

Входы и выходы внутренних блоков, назначенные внешними, подсвечиваются желтым цветом. Эти входы и выходы, являющиеся входами и выходами составного блока, отображаются на полях страницы.

Подробно о создании составного блока см. в разделе 8.1.

3.1.1.5 Функциональный блок

Окно представления ***Функциональный блок*** предназначено для отображения названия, имен входов и выходов и типов входов и выходов функционального блока.

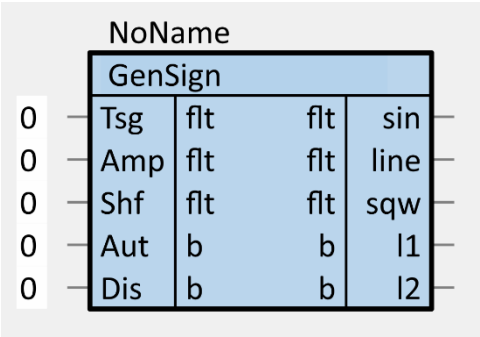


Рисунок 3.12 – Окно представления Функциональный блок

Подробнее о функциональном блоке см. в разделе 2.5.

3.1.1.6 Редактор

Для отображения и редактирования файлов-ссылок в проекте предназначено окно представления ***Редактор***.

Окно ***Редактор*** открывается автоматически при двойном щелчке на файле в дереве проекта.

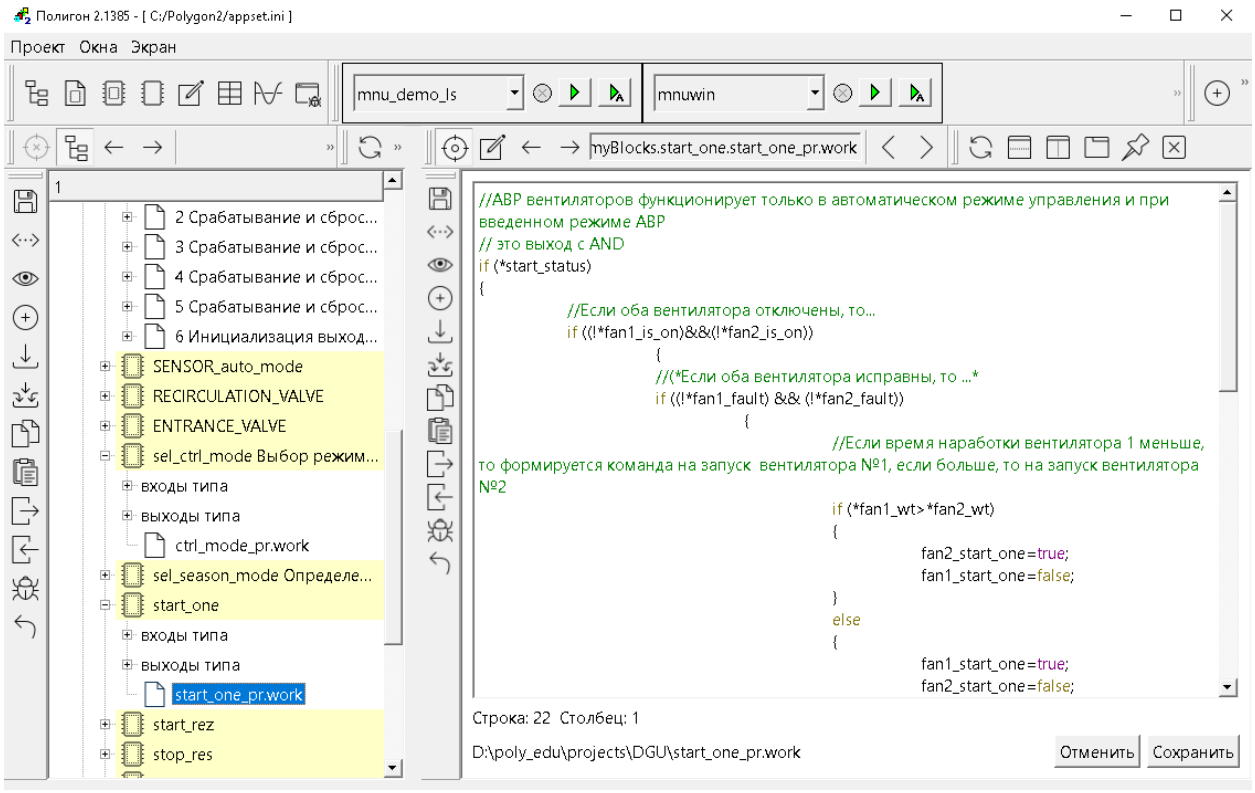


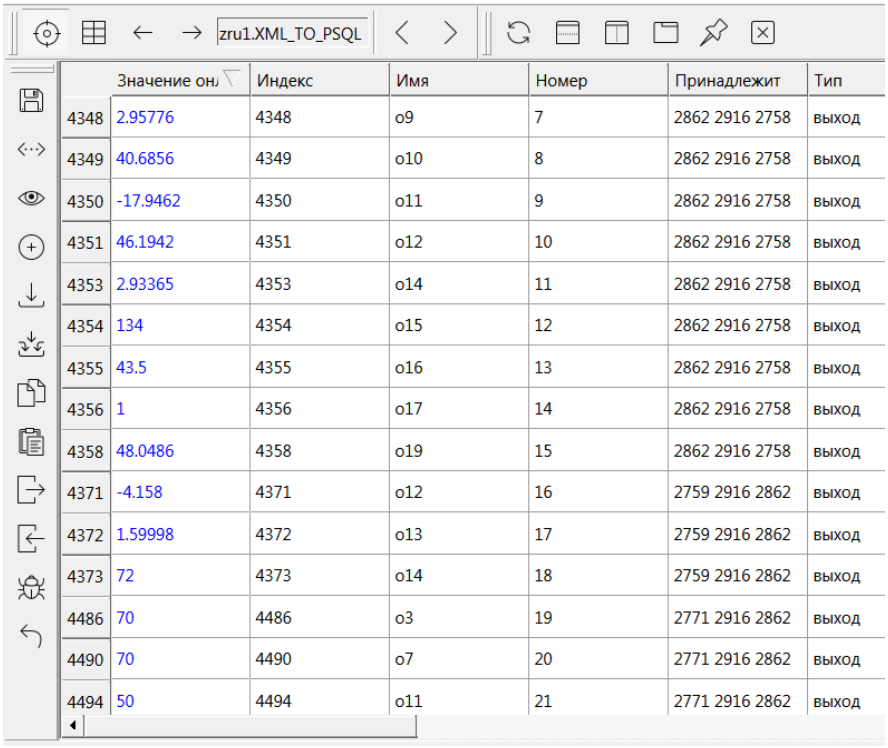
Рисунок 3.13 – Окно представления Редактор

В окне ***Редактор*** пишется программный код при создании составного функционального блока. Подробнее о создании составного ФБ см. в разделе 8.1.

Местоположение файла на диске указано в левом нижнем углу окна. Так как файлы хранятся отдельно, то для их редактирования можно пользоваться сторонним редактором. По умолчанию файлы хранятся в той же папке, где расположен проект/библиотека.

3.1.1.7 Таблица

Окно представления ***Таблица*** отображает информацию о функциональных блоках, их входах/выходах и свойствах в табличном виде. При представлении данных в виде таблицы удобно редактировать свойства, смотреть комментарии и уставки, подмененные значения во время отладки.



	Значение он	Индекс	Имя	Номер	Принадлежит	Тип
4348	2.95776	4348	o9	7	2862 2916 2758	выход
4349	40.6856	4349	o10	8	2862 2916 2758	выход
4350	-17.9462	4350	o11	9	2862 2916 2758	выход
4351	46.1942	4351	o12	10	2862 2916 2758	выход
4353	2.93365	4353	o14	11	2862 2916 2758	выход
4354	134	4354	o15	12	2862 2916 2758	выход
4355	43.5	4355	o16	13	2862 2916 2758	выход
4356	1	4356	o17	14	2862 2916 2758	выход
4358	48.0486	4358	o19	15	2862 2916 2758	выход
4371	-4.158	4371	o12	16	2759 2916 2862	выход
4372	1.59998	4372	o13	17	2759 2916 2862	выход
4373	72	4373	o14	18	2759 2916 2862	выход
4486	70	4486	o3	19	2771 2916 2862	выход
4490	70	4490	o7	20	2771 2916 2862	выход
4494	50	4494	o11	21	2771 2916 2862	выход

Рисунок 3.14 – Окно представления Таблица

Нажатием правой кнопки мыши по верхнему меню таблицы можно настроить отображение нужных столбцов. Также по каждому столбцу можно установить фильтр, нажав на стрелку вниз на интересующем столбце.

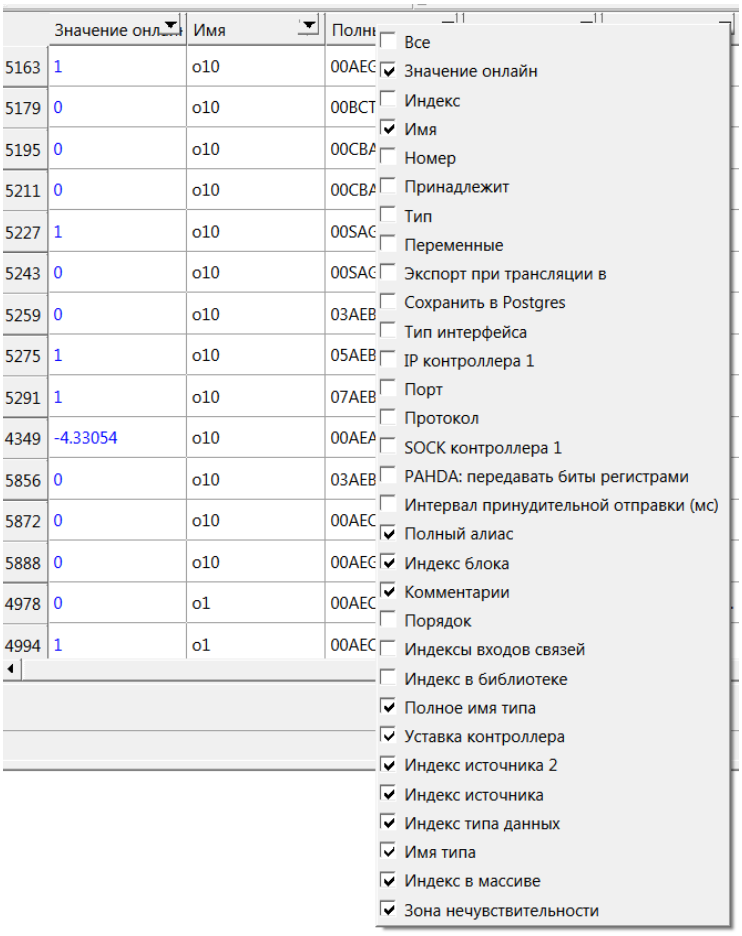


Рисунок 3.15 – Выбор отображаемых свойств в таблице

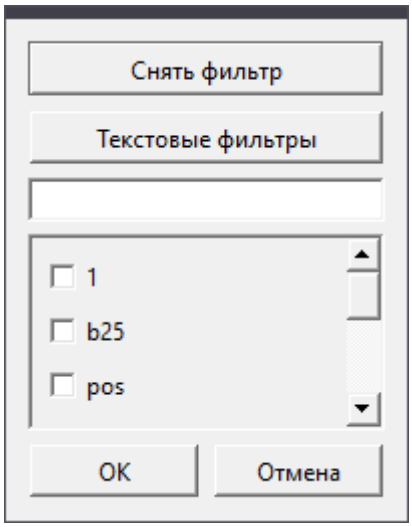


Рисунок 3.16 – Установка фильтра по столбцам в таблице

3.1.1.8 График

Окно представления ***График*** позволяет просматривать значения на входах и выходах функциональных блоков в графическом представлении во время отладки (подробнее об отладке см. в разделе 7).

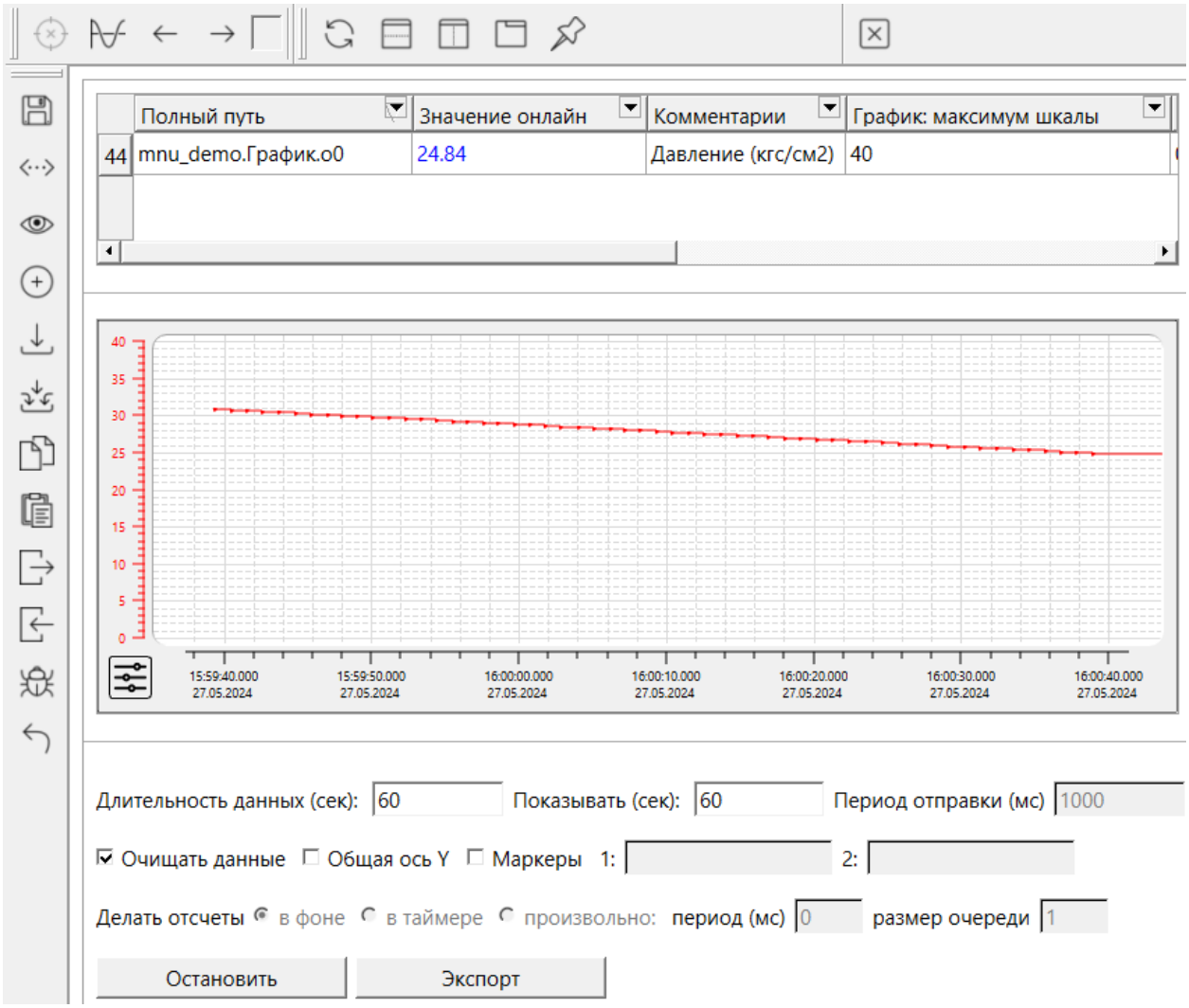


Рисунок 3.17 – Окно представления График

Для добавления входа или выхода для отображения на графике следует перетащить его со страницы с зажатым **Ctrl** в верхнюю часть окна ***График***. При этом ***Модуль*** добавленного входа или выхода автоматически станет текущим для графика. Для текущего ***Модуля*** можно добавлять входы/выходы на график из дерева с помощью перетаскивания по одному.

Для добавления на график группы входов или выходов следует добавить их в ***Раздел***.

Создать ***Раздел*** можно кликнув по модулю правой кнопкой мыши и нажав ***Создать***. В появившемся окне необходимо выбрать пункт ***раздел*** и ввести ***имя*** нового раздела. Входы и выходы проекта добавляются в ***Раздел*** путем перетаскивания из дерева или со страницы (с

зажатым **Ctrl**). В выпадающем меню при перетаскивании входа или выхода в раздел следует выбрать ***Добавить***.

Для отображения входов и выходов раздела на графике следует перетащить его из дерева проекта в верхнюю часть окна ***График*** – данный ***Раздел*** автоматически станет текущим для графика. Для текущего ***Раздела*** в таблицу графика автоматически добавляются входы/выходы из этого раздела. Если добавить новый вход или выход проекта в таблицу графика, он автоматически добавится в раздел.

Выбор свойств входов/выходов, отображаемых в столбцах таблицы графика, открывается нажатием ПКМ по заголовку таблицы.

Таблица 3.7 – Свойства для отображения в таблице графика

Свойство	Описание
Значение онлайн	Текущее значение входа/выхода
Маркер	Значение входа/выхода в момент, отмеченный маркером
График: минимум шкалы	Минимум индивидуальной шкалы входа/выхода
График: максимум шкалы	Максимум индивидуальной шкалы входа/выхода
Цвет	Цвет графика сигнала и его индивидуальной шкалы
Получено	Количество полученных значений
Ошибок	Количество ошибок
Зона нечувствительности	Если модуль разницы между значениями сигналов меньше данной величины, то будет считаться, что значение не изменилось
Период опроса (мс)	Количество миллисекунд между опросами
Размер очереди	Количество значений, которые могут быть накоплены перед отправкой (при недостаточном размере очереди можно наблюдать появление ошибок)

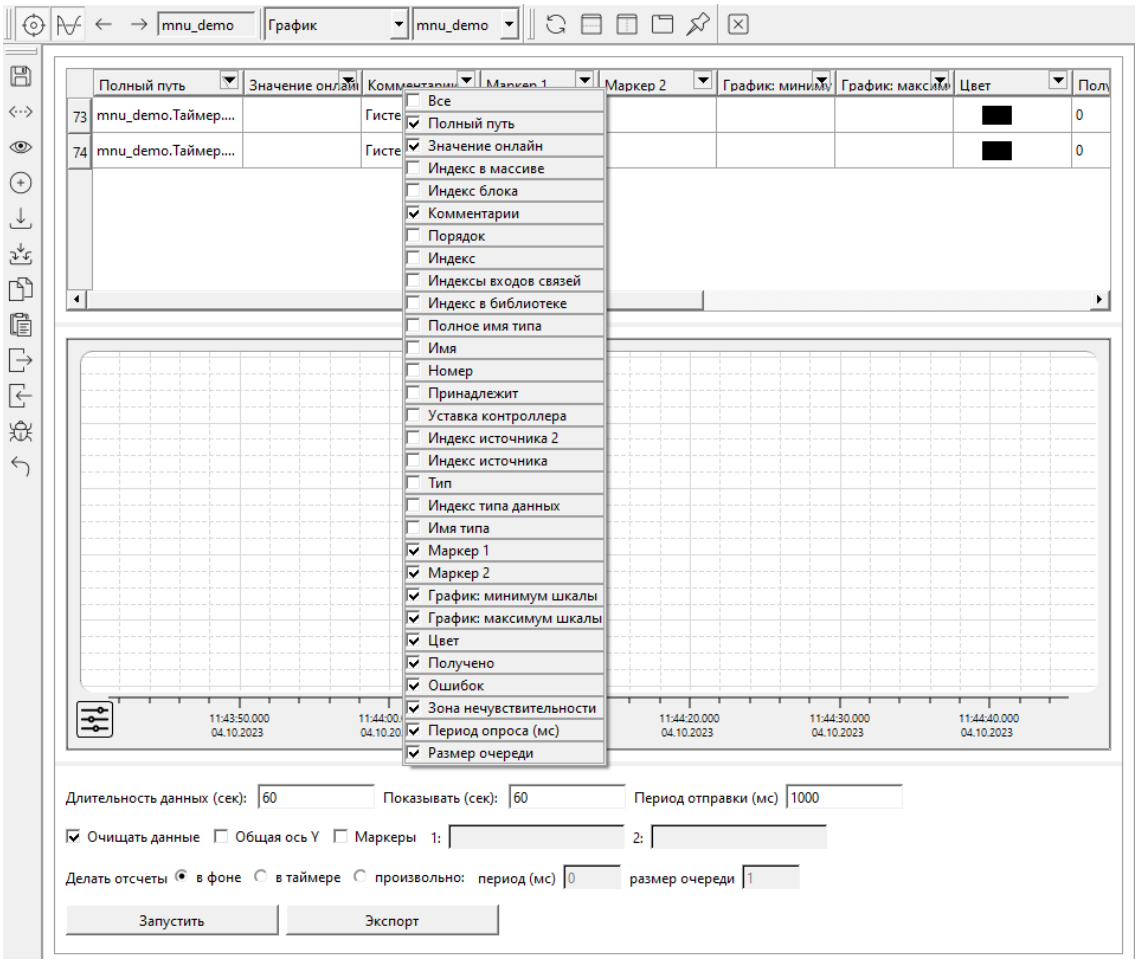


Рисунок 3.18 – Свойства входов/выходов в таблице графика

Свойства ***Получено*** и ***Ошибок*** отображают количество полученных значений данного сигнала и количество ошибок, сигнализирующих о переполнении очереди.

Значения остальных свойств могут быть заданы вручную (двойным щелчком на соответствующей ячейке таблицы или в окне свойств входа или выхода), иначе они будут браться из окна общих настроек, расположенного под графиком.

Правой кнопкой мыши на строке таблицы, соответствующей входу/выходу, вызывается контекстное меню, которое позволяет:

- ***Скрыть*** или ***Показать*** вход или выход на графике;
- ***Скрыть ось*** или ***Показать ось*** для входа или выхода;
- ***Удалить*** вход или выход из таблицы.

В окне под графиком настраиваются следующие параметры:

Таблица 3.8 – Настройки графика

Свойство	Описание
Длительность данных (сек)	Количество данных, которое нужно хранить
Показывать (сек)	Количество данных, которое нужно показывать на окне

Продолжение таблицы 3.8

Период отправки шкалы	Частота отправки значений сигналов контроллером на график (отправка происходит не чаще этого времени)
Очищать данные	Удалять старые данные с графика при запуске
Общая ось (Y)	Используется для отображения данных на общей оси
Маркеры 1, 2	Вертикальные прямые, с помощью которых можно просмотреть значение на графике в нужный момент времени
Делать отсчеты	• ***в фоне*** – по окончании периода отсчета будет посылаться одно текущее значение с каждого входа/выхода, • ***в таймере*** – по окончании периода отсчета для каждого входа/выхода будут посылаться значения, накопленные с заданным таймерным промежутком, • ***произвольно*** – значения накапливаются с заданным периодом, ***Период*** и ***Размер*** очереди устанавливается справа. Эти значения применяются для всех входов/выходов, у которых не установлены свои значения.

Для просмотра текущих значений входов/выходов на графике следует нажать кнопку ***Запустить*** (запуск проекта на контроллере описан в разделе 6). Для остановки отрисовки графика следует нажать ***Остановить***.

ПРИМЕЧАНИЕ
Для корректной работы графика необходимо синхронизировать системное время контроллера с браузером через web-конфигуратор (***Система/Время/Системное время*** – ***Синхронизировать с браузером***).

График подключается к запущенному проекту как клиент по протоколу **OPC UA**. По умолчанию график оформляет подписку на данные проекта с дискретностью фонового цикла.

Для получения данных из таймера в проекте должен быть создан функциональный блок ***OpcUAServerTimer*** из библиотеки ***paOpcUA***. Блок должен быть создан в ***Таймере*** для работы подписок с точностью таймерного цикла (период отображения данных задается свойством модуля ***Таймерный промежуток*** в мс).

Подробнее см. в документе Обмен с верхним уровнем. Библиотека paOpcUA.

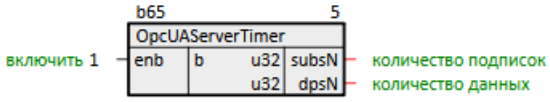


Рисунок 3.19 – Блок OpcUAServerTimer

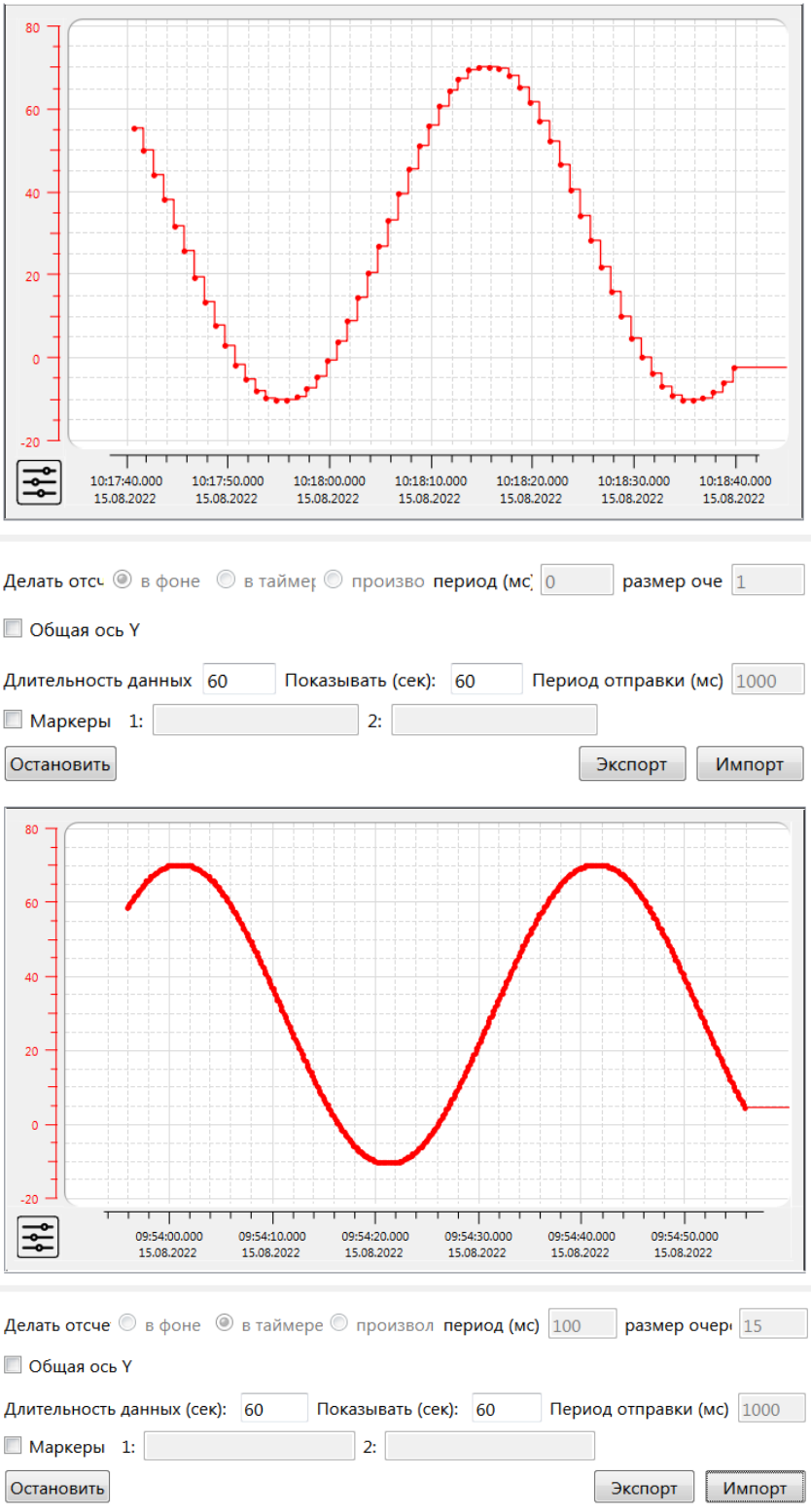


Рисунок 3.20 – Пример работы графика в фоне (сверху) и в таймере (снизу)

Масштабировать график можно с помощью колеса мыши с зажатым **Ctrl** на области оси. При выделении мышкой области слева направо и сверху вниз график увеличивается.

Перемещаться по графику влево/вправо и вверх/вниз можно с помощью мышки с зажатой левой кнопкой на соответствующей оси графика.

Нажав правой кнопкой мыши в окне тренда, можно сбросить масштаб, либо выбрать следующий, предыдущий или исходный масштаб.

Также можно выбрать режим ***Отображение на разных полотнах***.

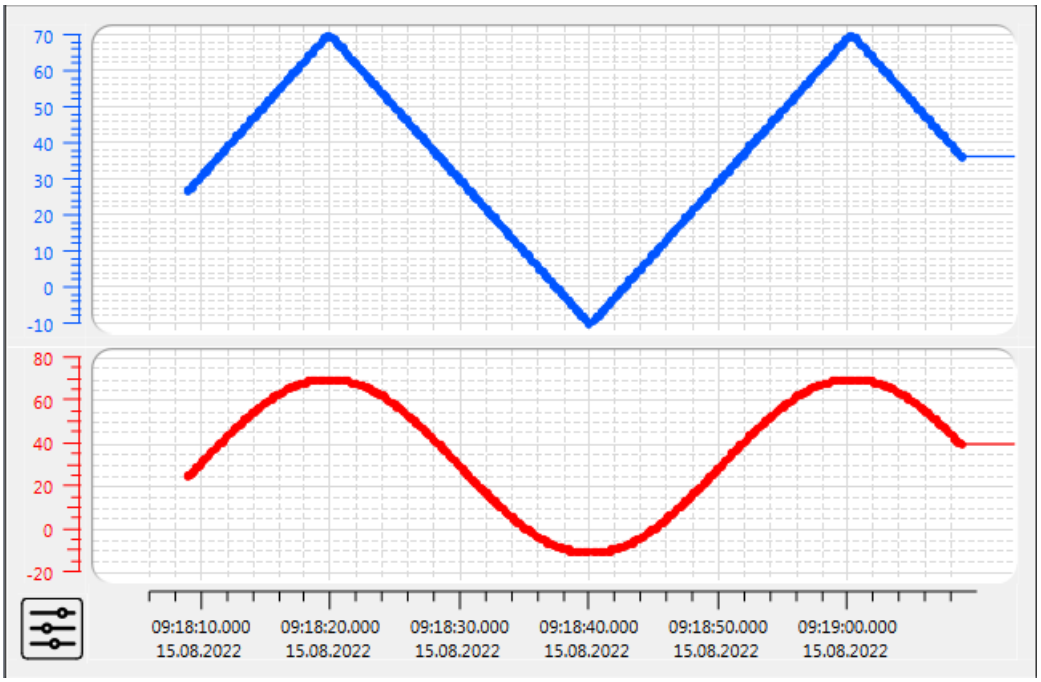


Рисунок 3.21 – Отображение графиков на разных полотнах

Клик правой кнопкой мыши по оси X или У открывает их свойства. Также кнопки свойств осей можно отобразить, нажав на кнопку с ползунками в левой нижней части графика.

В свойствах оси X можно выбрать тип тренда, временное ограничение буфера данных, параметры оси.

В свойствах оси Y можно выбрать границы отображаемого интервала данных.

С помощью кнопки ***Экспорт*** можно выгрузить график на диск в форматах **.txt**, **.pdf** или **.png**.

ВНИМАНИЕ
Путь сохранения не должен содержать кириллицу и пробелы.

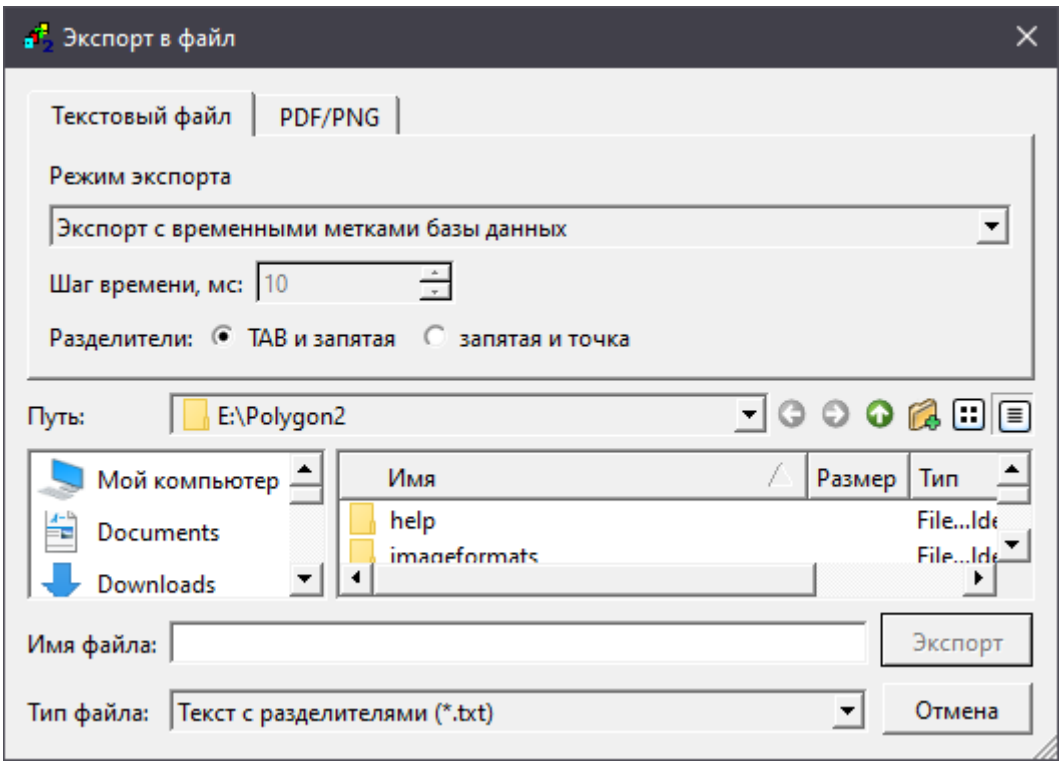


Рисунок 3.22 – Экспорт графика

3.1.1.9 Экран отладчика

Окно представления ***Экран отладчика*** позволяет просматривать и изменять значения на входах и выходах функциональных блоков во время отладки (подробнее об отладке см. в разделе 7).

Экран отладчика соответствует определенному ***Разделу***. Входы и выходы, которые нужно отобразить на экране, следует добавить в этот раздел.

Создать ***Раздел*** можно кликнув по модулю правой кнопкой мыши и нажав ***Создать***. В появившемся окне необходимо выбрать пункт ***раздел*** и ввести ***имя*** нового раздела.

Для того чтобы добавить необходимые входы/выходы в раздел экрана нужно выделить интересующий вход/выход на странице, зажав **Ctrl**, перетащить его в раздел, выбрать ***Добавить*** – данный вход/выход появится в разделе. Также входы/выходы можно перетащить на экран из дерева или страницы, тогда они автоматически добавятся в текущий раздел.

Для открытия экрана отладчика необходимо в окнах представления в верхнем меню выбрать ***Экран отладчика***, сделать окно активным и дважды кликнуть на необходимый раздел в дереве, либо перетащить раздел на активную страницу.

Входы и выходы раздела можно отобразить следующими типами графических элементов:

Таблица 3.7 – Типы графических элементов

Свойство	Класс отображения	Описание
Кнопка импульсная	pushbutton	При нажатии, если на входе не **0**, сначала посылает **0**, затем **1**, затем **0**
Кнопка	togglebutton	При нажатии посылает **1**, при отжатии – **0**
Слайдер	slider	Отображает значения в диапазоне от минимума до максимума, в случае входа позволяет редактировать значение
Шкала	dial	Аналогично ***слайдеру*** отображает значения в диапазоне от минимума до максимума, в случае входа позволяет редактировать значение
Значение	value	Отображает значение целым или вещественным числом
Лампа	lamp	При нулевом или отрицательном значении загорается красным цветом, при положительном – загорается зеленым (цвета ***лампы*** можно поменять в окне свойств)

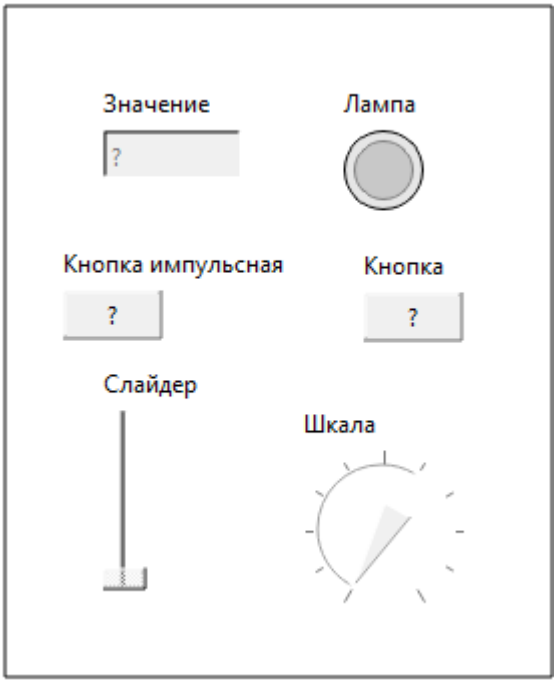


Рисунок 3.23 – Вид графических элементов на экране отладки

Один вход или выход в проекте может соответствовать только одному графическому элементу на экране.

При добавлении входа или выхода в раздел экрана ему по умолчанию присваивается класс отображения в соответствии с правилом:

- если это логический вход – ***кнопка***, иначе – ***слайдер***;
- если это логический выход – ***лампа***, иначе – ***значение***.

Класс отображения можно поменять в окне свойств. Для этого следует:

1. Открыть свойства интересующего графического элемента.
2. Добавить свойство ***Экран: класс отображения*** из нижнего выпадающего списка.

3. Установить в свойстве нужное значение класса отображения.

Двойным нажатием на графическом элементе на экране отладчика отображается соответствующий вход или выход на странице в текущее окно.

Удаляется графический элемент выбором команды ***Удалить*** в контекстном меню при нажатии ПКМ. Эта команда также удаляет вход или выход из раздела, открытого в качестве экрана.

На экран отладчика также можно добавить элементы панели ***Редактирование***: стрелку, фон, текст.

Масштабирование экрана отладчика выполняется с помощью колеса мыши с зажатым **Ctrl**.

В режиме работы отладчика для изменения значения на входе, нужно дважды щелкнуть на соответствующем элементе и отредактировать значение. Это возможно для графических элементов: ***значение***, ***слайдер*** и ***шкала***.

Важные свойства графических элементов:

Таблица 3.7 – Свойства графических элементов экрана отладчика

Свойство	Описание
Комментарии	Отображается в качестве комментария около входа/выхода, в дереве, над графическим элементом
Номер	Определяет порядок отображения в дереве, начиная с **0**
Экран: класс отображения	Выбор типа графического элемента
Экран: разрешить изменения	Разрешает редактировать значение во время отладки для графических элементов ***значение***, ***слайдер***, ***шкала***
Экран: ***x*** и ***Экран: y***	Координаты элемента
Экран: min и ***Экран: max***	Задают диапазон изменения (минимальное и максимальное значения) для графических элементов ***слайдер***, ***шкала***
Экран: шаг	Задает шаг шкалы для ***слайдера*** и ***шкалы***
Экран: радиус скругления	Задает радиус скругления для лампы
Экран: цвет_1 и ***Экран: цвет_2***	Задают цвет графического элемента лампа во «включенном» и «выключенном» состояниях
Экран: ширина элемента и ***Экран: высота элемента***	Задают ширину и высоту графического элемента
Экран: отключить надписи	Убирает отображение свойства ***Комментарии*** над графическим элементом

3.1.2 Системные окна

Системные окна могут быть следующих типов:

Таблица 3.1 – Типы системных окон

Тип окна представления	Пиктограмма	Функция
Контроллер		Подключение к контроллеру и загрузка проекта

Продолжение таблицы 3.1

Прогресс		Отслеживание процессов исполнения
Проекты		Управление проектами и библиотеками
Настройки сборки		Пути к установленным сборкам
Поиск		Поиск по проекту
Справка или ***F1***		Информация по среде разработки и добавленным библиотекам
О программе		Сведения о программе, библиотеках, загрузка обновлений
Печать		Печать страниц проекта

Системные окна добавляются из меню ***Окна*** или с помощью панели ***Окна***, которую можно открыть из меню ***Окна/Панели инструментов/Окна***.

Рисунок 3.24 – Панель Окна

Системные окна можно размещать на экране также как окна представления либо выносить поверх них. Также можно разместить системные окна в виде вкладок.

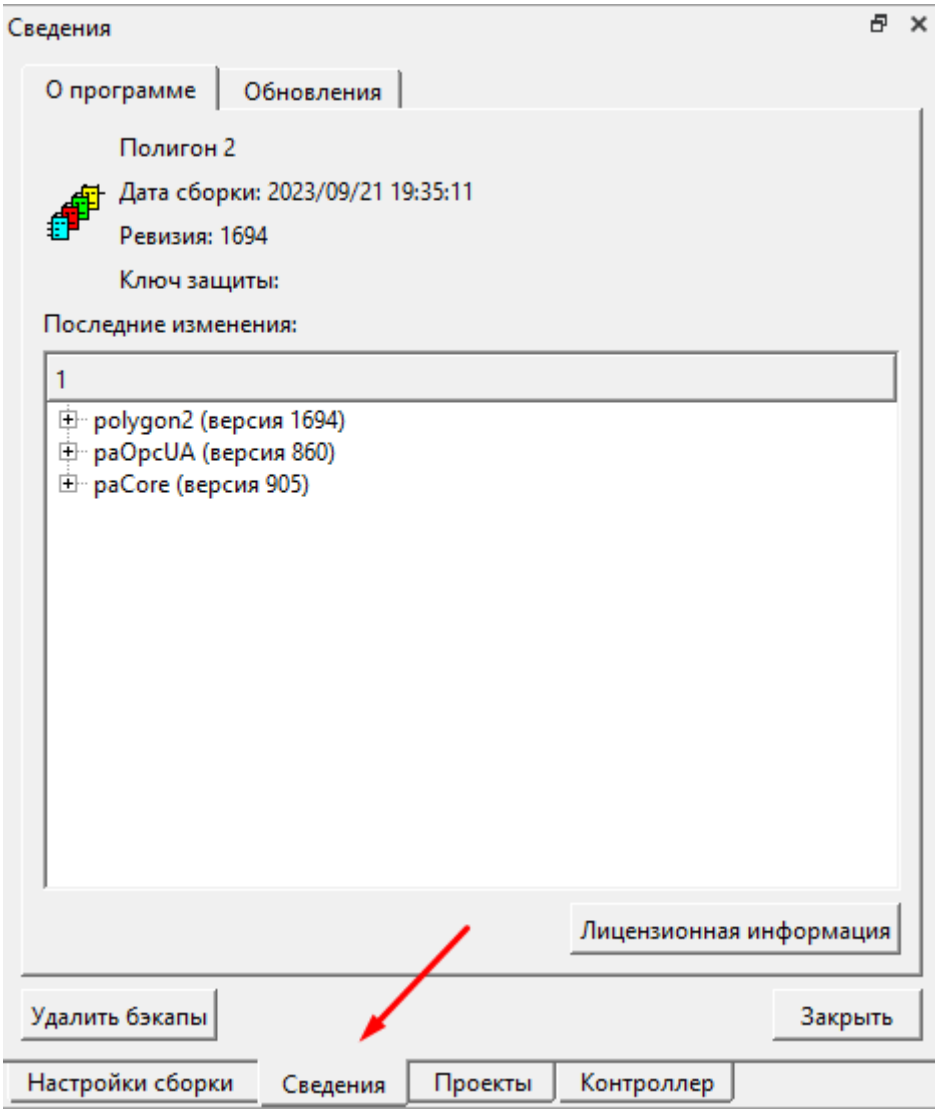


Рисунок 3.25 – Размещение системных окон по вкладкам

3.1.2.1 Контроллер

Инструменты для загрузки и запуска программ на контроллере находятся в окне ***Контроллер***.

Подробно о способах загрузки проекта на контроллер см. в разделе 6.

Подключение к контроллеру для загрузки и запуска проекта осуществляется по протоколу **SSH**.

При установке флага ***Параметры модуля*** – настройки подключения и путь к исполняемому файлу проекта подгружаются из соответствующих свойств модуля, выбираемого из выпадающего списка. При убранном флаге ***Параметры модуля*** настройки подключения и путь к исполняемому файлу можно прописать вручную.

При успешном подключении к контроллеру в консольном окне будут отображаться результаты выполнения команд.

Кнопкой ***Загрузить*** выполняется загрузка проекта в контроллер.

Кнопками ***Запустить модуль*** и ***Остановить модуль*** выполняется, соответственно, запуск и останов работы загруженного приложения на контроллере.

Кнопкой ***Закрыть сессию*** выполняется отключение от ПЛК.

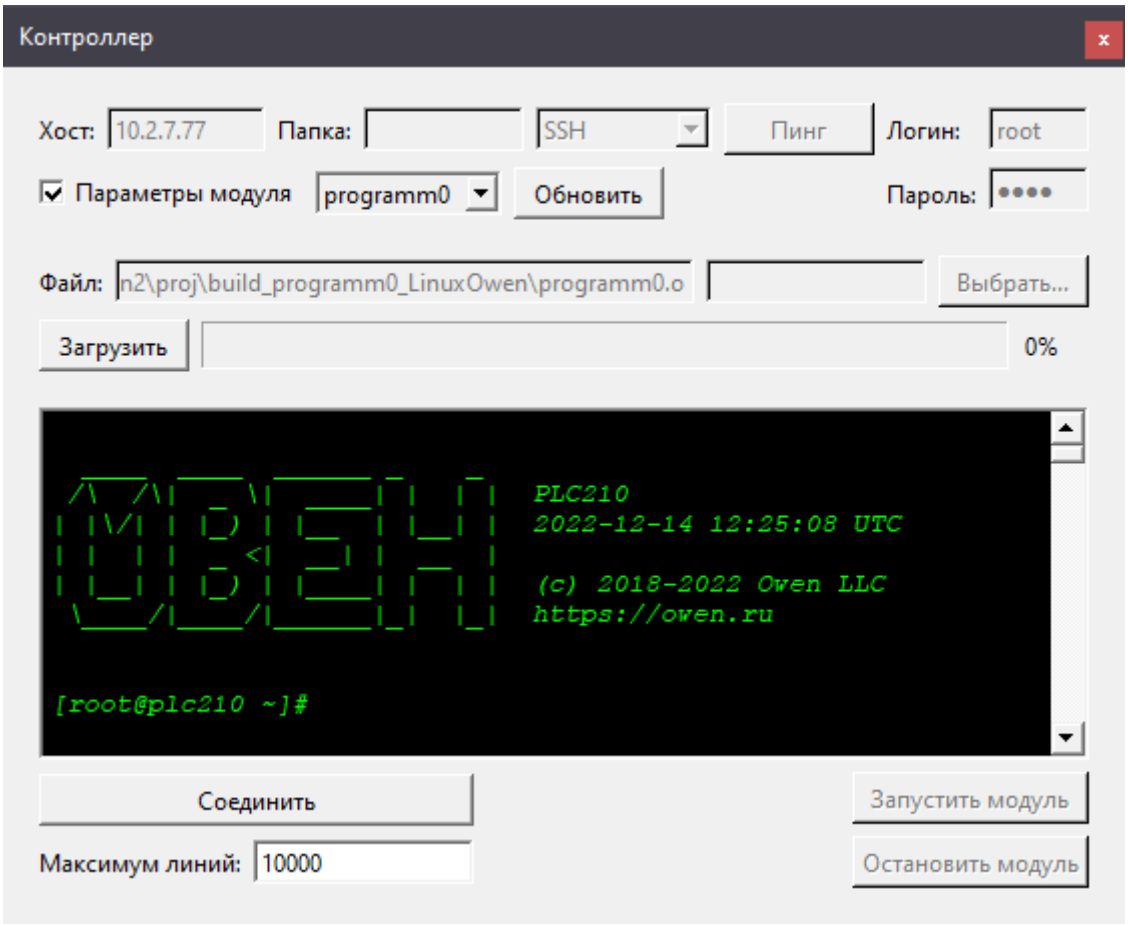


Рисунок 3.26 – Системное окно Контроллер

3.1.2.2 Прогресс

Для отслеживания процесса выполнения используется окно ***Прогресс***.

При трансляции в окне ***Прогресс*** отображается ход выполнения, при наличии ошибок они указываются с описанием.

По итогу трансляции можно видеть результат – сообщение об успешности выполнения и время выполнения.

Подробно о трансляции проекта см. в разделе 5.

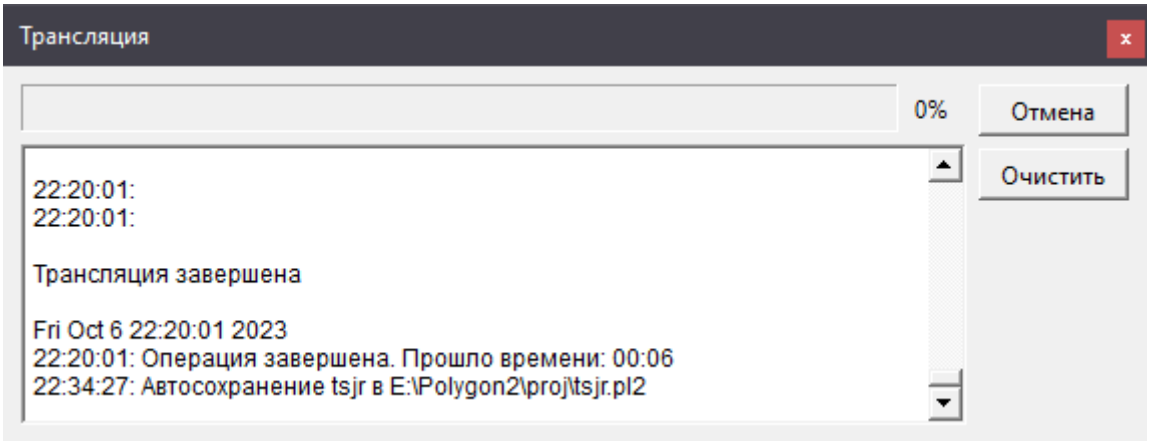


Рисунок 3.27 – Системное окно Прогресс

3.1.2.3 Проекты

Для добавления в среду ссылок на проекты и библиотеки используется окно ***Проекты***.

В окне ***Проекты*** с помощью соответствующих кнопок можно открыть существующие проект и/или библиотеку, создать новые, закрыть (для этой функции также может быть использован значок «**x**» закрытия справа от проекта/библиотеки в списке).

В списке указывается путь до добавленных проекта и/или библиотеки на диске.

В окне ***История изменений*** можно получить информацию по логам выбранного проекта и/или библиотеки.

Через окно проекты можно сохранить копию проекта под другим названием с помощью кнопки ***Сохранить как…***, а также создать зашифрованную версию проекта (подробнее о защите проекта см. в разделе 9).

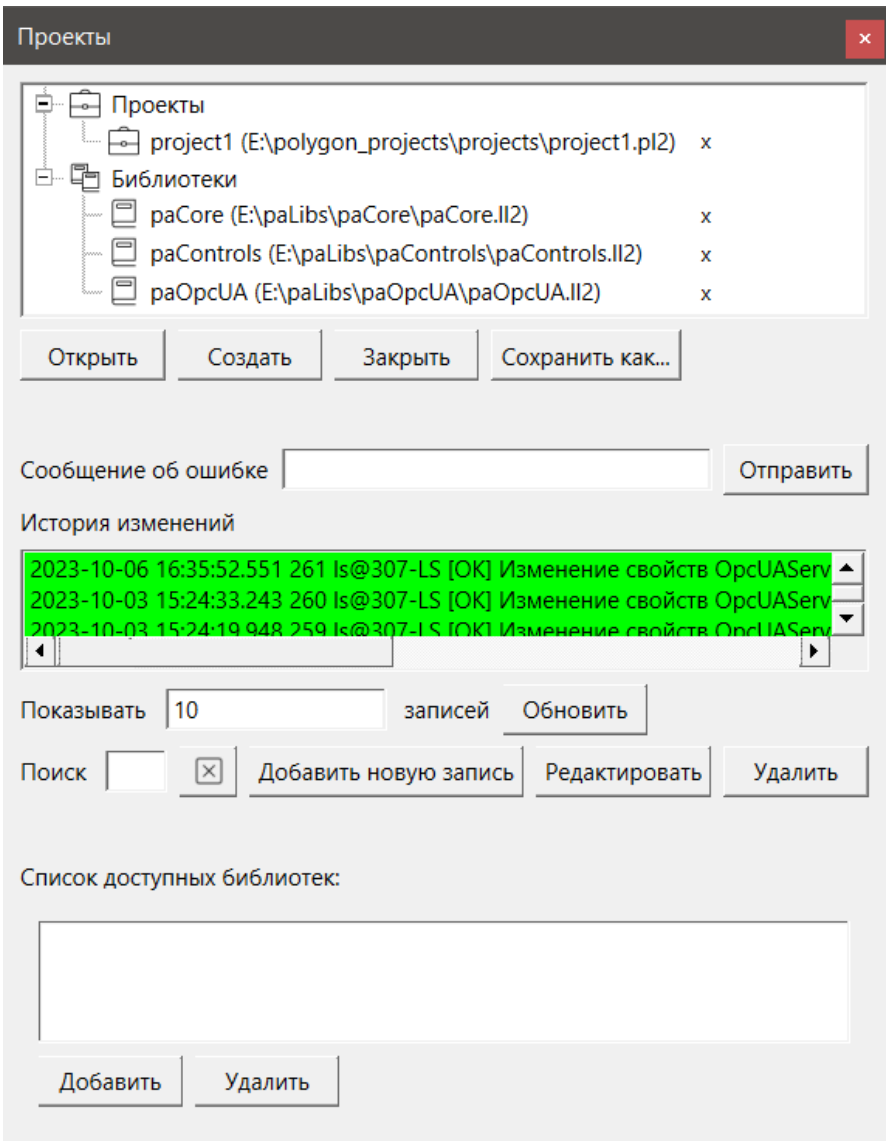


Рисунок 3.28 – Системное окно Проекты

3.1.2.4 Настройки сборки

Для настроек сборок используется окно ***Настройки сборки***.

В окне ***Настройки сборки*** указаны пути к установленным сборкам. Настройки сборки устанавливаются инсталлятором, пользователю остается убедиться в их установке.

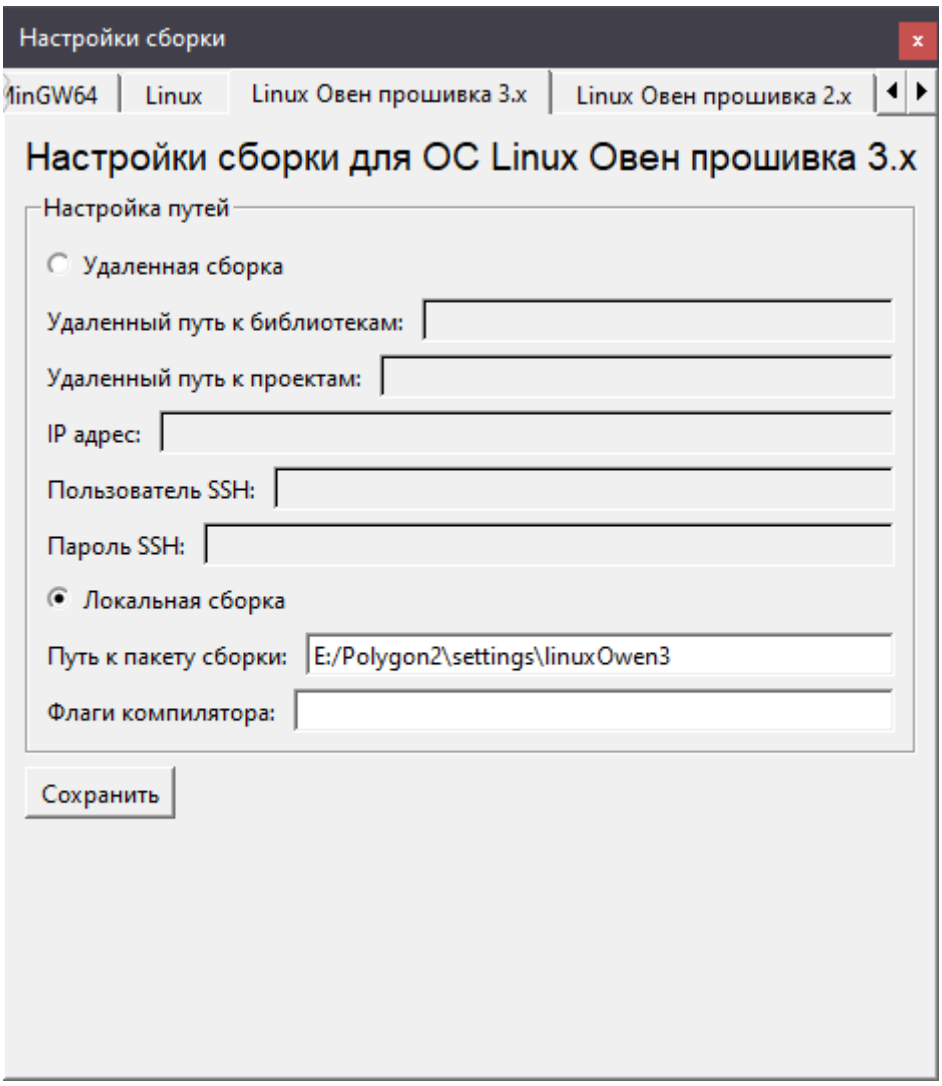


Рисунок 3.29 – Системное окно Настройки сборки

3.1.2.5 Поиск

Для поиска по проекту используется окно ***Поиск***.

В окне ***Поиск*** выбирается проект и задаются критерии поиска. Для быстрого поиска возможно добавление условий.

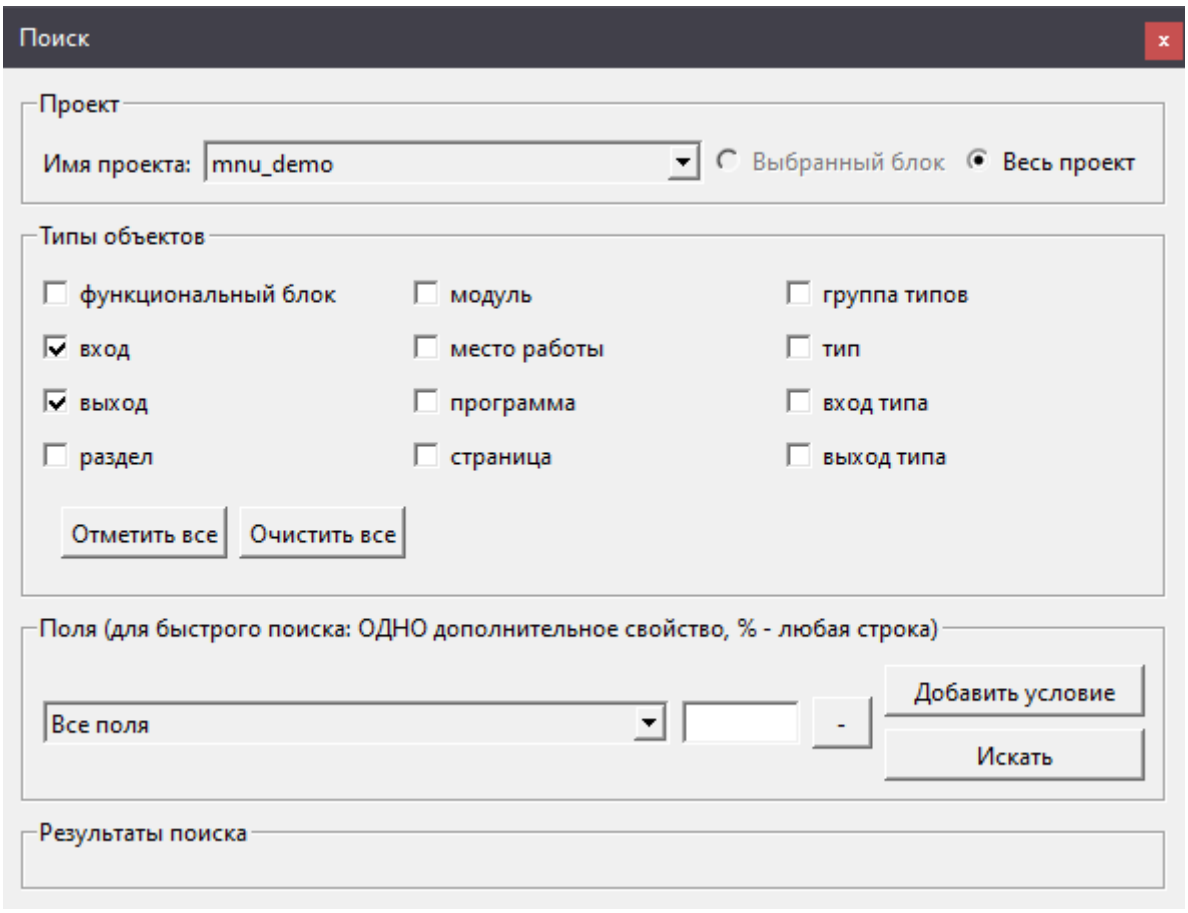


Рисунок 3.30 – Системное окно Поиск

3.1.2.6 Справка

Для получения информации по среде разработки или по работе и назначению входов/выходов функционального блока из библиотеки, используется окно ***Справка*** (горячая клавиша **F1**).

В верхнем выпадающем списке окна ***Справка*** можно выбрать интересующий раздел справки – среду разработки или конкретную библиотеку.

В нижнем выпадающем списке можно выбрать отображение: содержания раздела, поиска по разделу или указателя на ключевые слова раздела.

Открыть справку на конкретный функциональный блок в проекте можно, выбрав его в дереве (в проекте или в библиотеке) или непосредственно на странице проекта и нажав **F1**. Или выбрав ***Справка*** в выпадающем контекстном меню при нажатии ПКМ на блоке.

При создании составных функциональных блоков и/или пользовательских библиотек можно создавать для них справку. Подробнее см. в разделе 8.

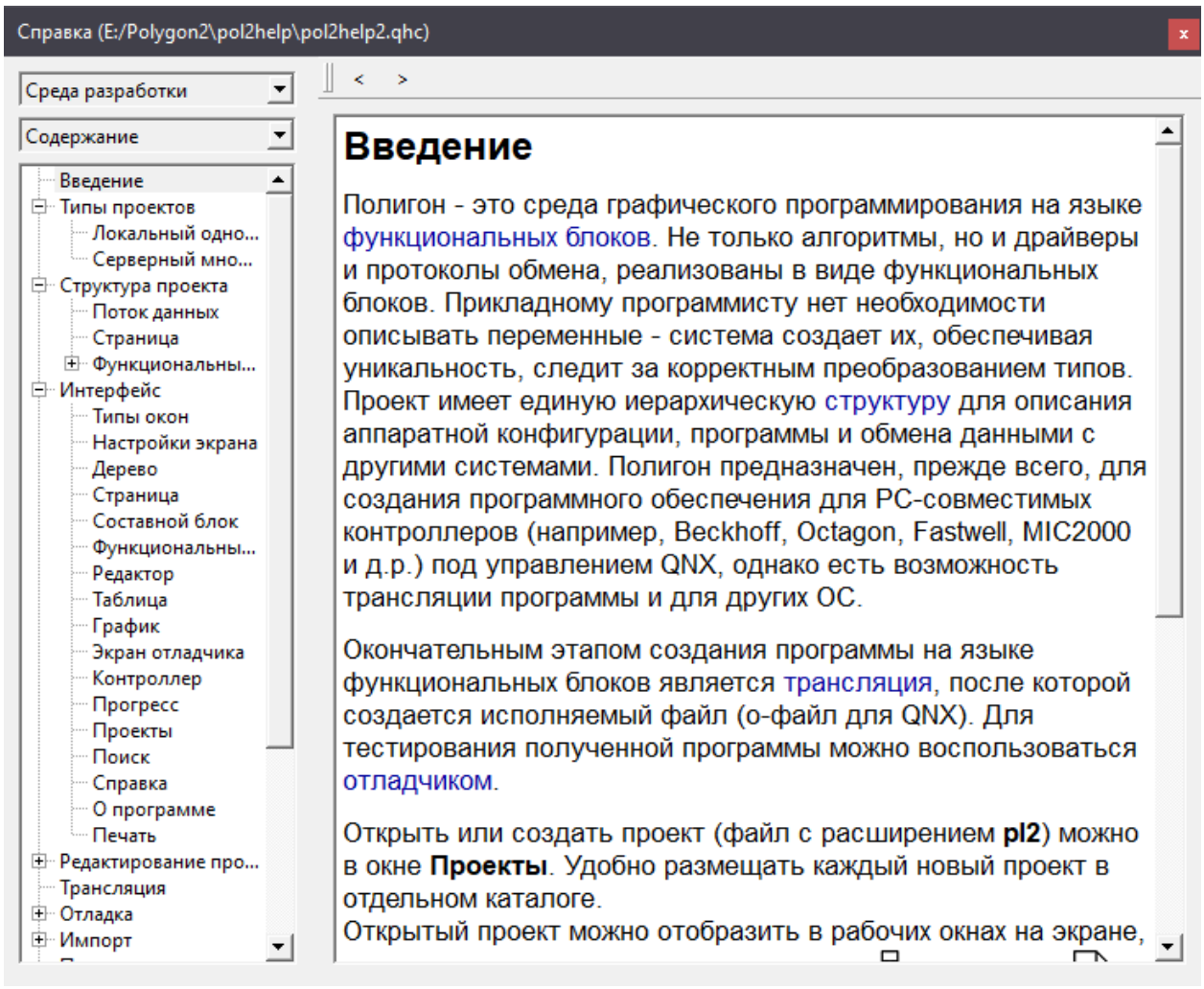


Рисунок 3.31 – Системное окно Справка

3.1.2.7 О программе

Для получения сведений о программе, версиях библиотек и наличии обновлений предназначено системное окно ***О программе***.

Во вкладке окна ***О программе*** можно получить информацию о текущих установленных версиях среды Полигон и библиотек.

При нажатии кнопки ***Лицензионная информация*** открывается окно со сведениями о системных лицензиях, используемых в среде разработки библиотек.

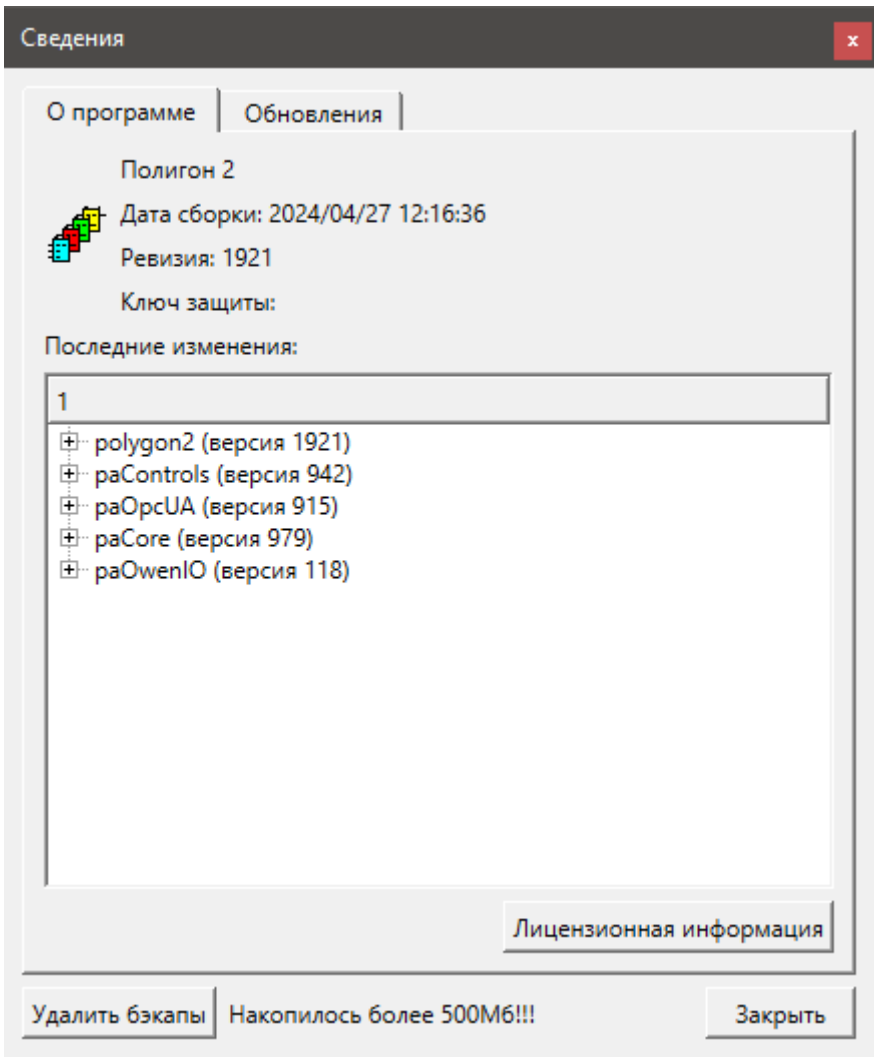


Рисунок 3.32 – Системное окно О программе

Во вкладке окна ***Обновления*** при успешном подключении к серверу **pa.ru** при нажатии ***Проверить*** проверяется наличие обновлений.

После нажатия кнопки ***Установить*** программа произведет установку найденных обновлений и выполнит перезагрузку среды.

При установке флага ***Показывать тестовые версии*** в списке обновлений также будут доступны тестовые релизы среды и библиотек.

При установке флага ***Показывать старые версии*** будут показаны доступные для установки прошлые версии релизов среды и библиотек.

В нижней части окна выбирается распаковщик – ***unzip*** или ***7zip***.

Подробно обновление среды и/или библиотек описано в разделе 1.5.

В левом нижнем углу окна расположена кнопка ***Удалить бэкапы***. В новом окне можно выбрать бэкапы (резервные копии проекта) для удаление. При превышении общего объема бэкапов в **500 Мб** в окне выводится предупреждающее сообщение.

При запуске среды также выводится окно с предложением удалить бэкапы.

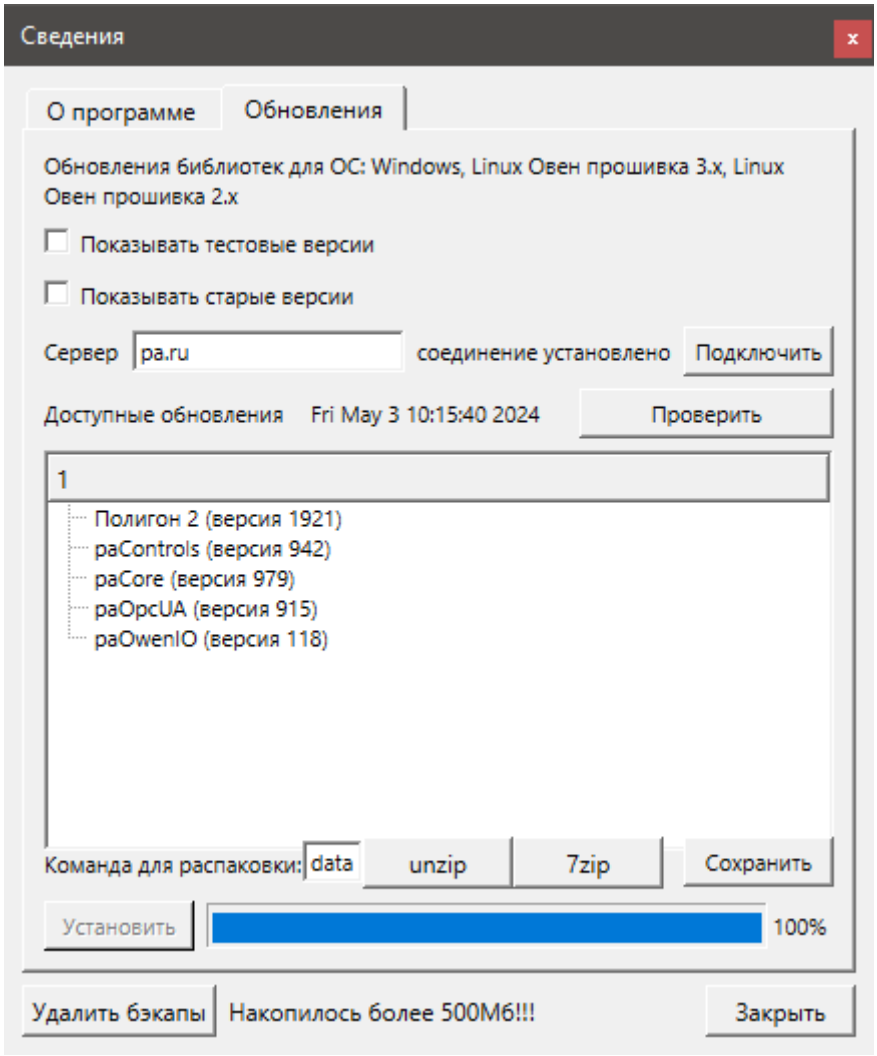


Рисунок 3.33 – Вкладка Обновления

3.1.2.8 Печать

Для печати страниц проекта используется окно ***Печать***.

В окне выбирается проект для печати.

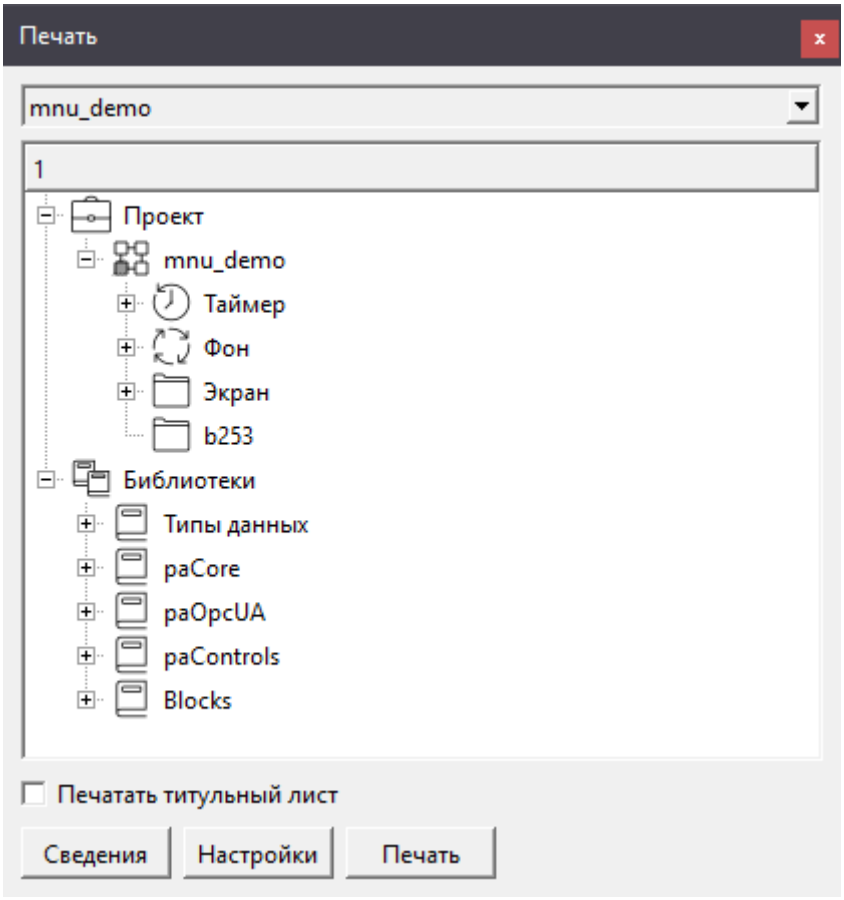


Рисунок 3.34 – Системное окно Печать

Во вкладке ***Сведения*** добавляются данные для титульного листа и штампов.

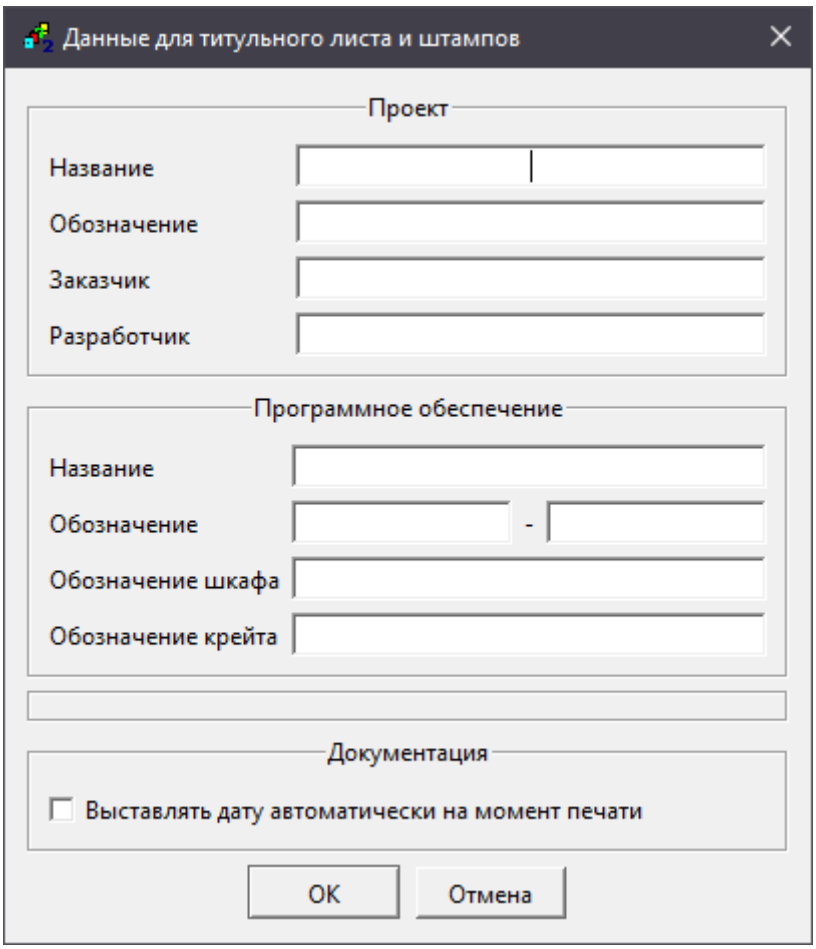


Рисунок 3.35 – Данные для титульного листа и штампов

Во вкладке ***Настройки*** выбирается размер печати и вариант печати – на бумажном носителе или в файл.

При выборе печати в файл – результат выполнения сохраняется в рабочую папку проекта.

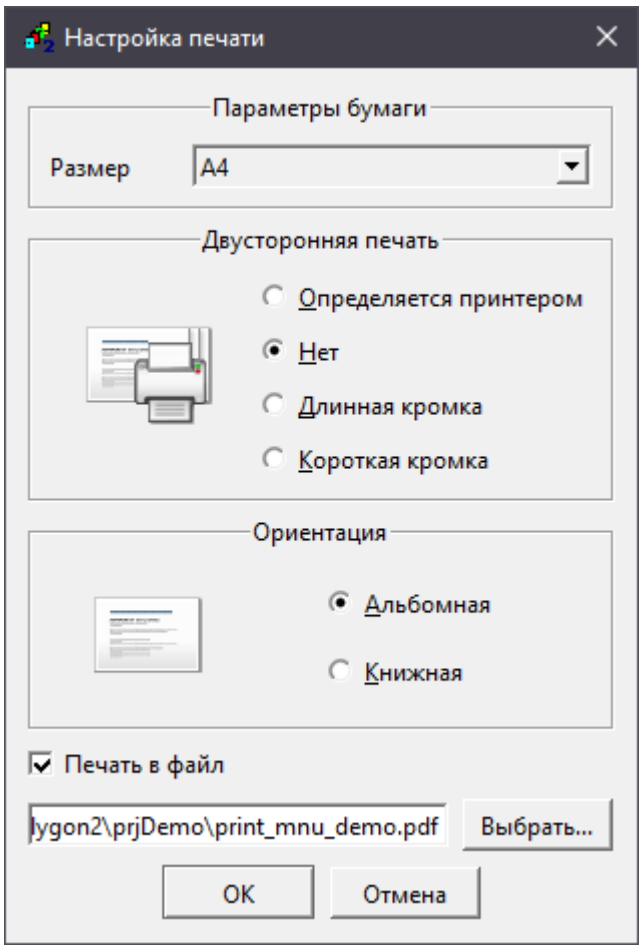


Рисунок 3.36 – Настройки печати

3.1.3 Окно Свойства элементов проекта

Окно ***Свойства*** можно открыть, вызвав контекстное меню ПКМ на нужном элементе проекта и выбрав ***Свойства***. Также свойства можно открыть, нажав соответствующую пиктограмму на панели Инструменты.

Окно ***с*** можно как разместить среди окон представления, так и вынести над ними.

В списке в окне указываются установленные свойства элемента. Серым цветом отображаются свойства, недоступные для редактирования пользователем.

Снизу в двух выпадающих списках выбираются новые свойства для добавления.

В первом списке расположены основные свойства, такие как ***Комментарии***, ***Номер***, ***Порядок*** и т.д.

Во втором списке приведены дополнительные и специфические свойства для элементов. Для ускорения поиска в выпадающем списке можно набрать первые буквы искомого свойства.

При нажатии кнопки ***Добавить*** новое свойство появляется в списке свойств элемента. После добавления следует задать ему значение и нажать ***Сохранить***, чтобы изменения вступили в силу.

При нажатии ***Отмена*** отменяются последние изменения свойств – добавление, удаление или установка нового значения.

Удалить свойство можно, нажав на него ПКМ и выбрав – ***Удалить свойство***.

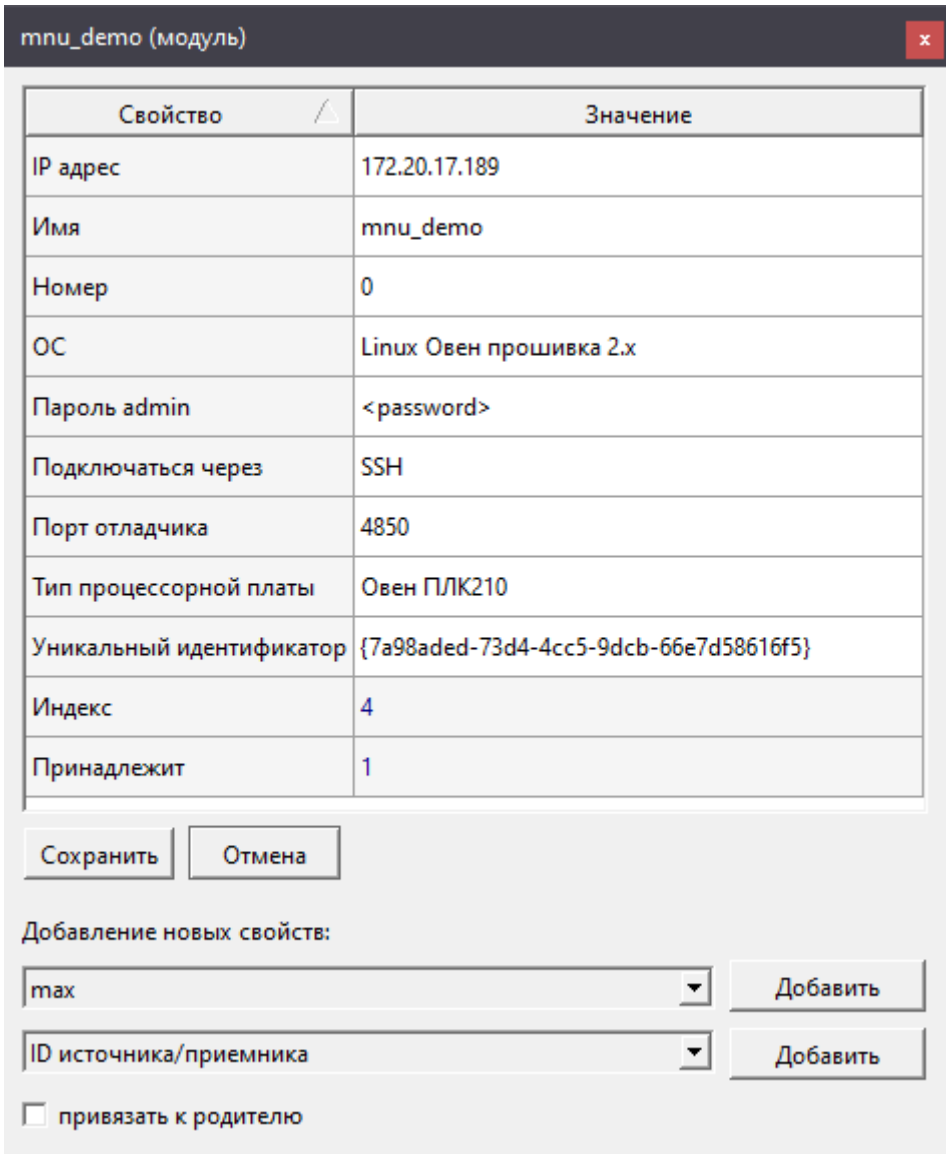


Рисунок 3.37 – Окно Свойства

Включенный флаг ***Привязать к родителю*** позволяет задать свойство текущего узла только внутри текущего родителя.

Если, например, вход добавлен в несколько разделов, то можно раскрыть в дереве конкретный раздел, выбрать этот вход и добавить ему свойство с флагом ***Привязать к родителю***, тогда в других разделах у данного входа этого свойства не появится. Такому же свойству у входа в других разделах можно задать другое значение также с флагом ***Привязать к родителю***.

Таким образом, можно, например, добавить вход в разделы нескольких OPC UA-клиентов и для каждого задать свое значение свойства ***Зона нечувствительности***.

В окне ***Свойства*** также можно добавить свойства: ***Пользовательское свойство 00***, ***Пользовательское свойство 01***, …, ***Пользовательское свойство 20***.

Их использование полезно, например, в свойствах модуля для установки дополнительных IP адресов и обращения к ним по SQL-запросам на входах блоков в проекте. Таким образом, IP адреса можно изменять в одном месте проекта.

Пример SQL-запроса пользовательского свойства ***Пользовательское свойство 00***:

```
<sql> SELECT value FROM blocks_prop WHERE indx=:module AND type="prop_0"</sql>
```

Для указания свойства модуля используется идентификатор, в примере: **prop_0**. Идентификатор свойства отображается при наведении на него ЛКМ в окне свойств.

3.2 Конфигурация экрана

Расположение окон, а также ссылки на открытые проекты и библиотеки сохраняются в файле на диске и восстанавливаются при следующем запуске среды Полигон в качестве файла конфигурации с расширением **.ini**. Можно создавать разные конфигурации рабочего экрана и сохранять их через меню ***Экран***.

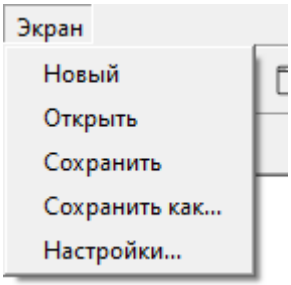


Рисунок 3.38 – Действия с файлом конфигурации в меню Экран

3.3 Настройки приложения

В окне ***Экран/Настройки*** предоставляется возможность выбора некоторых свойств навигации, трансляции, а также пользовательских установок. В том числе в данном окне можно настраивать время сохранения бэкапов или отключить сохранение бэкапов.

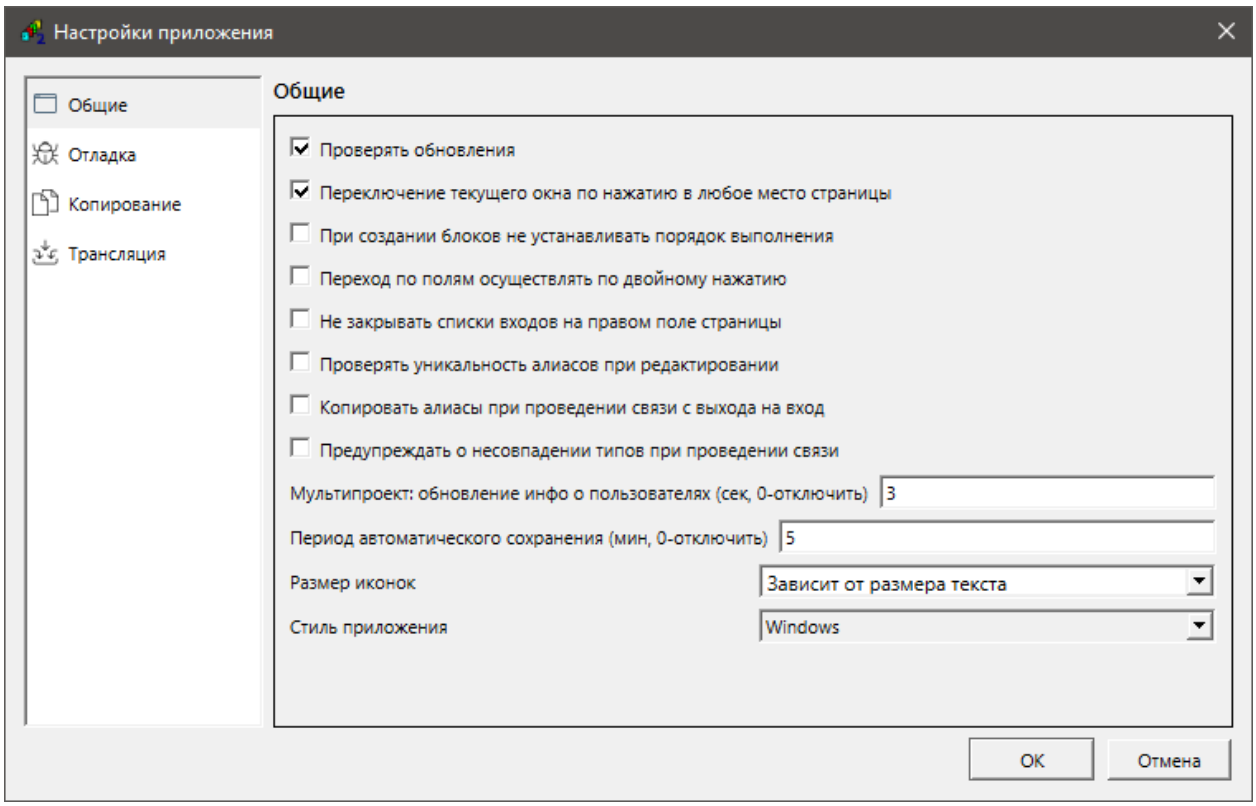


Рисунок 3.39 – Окно Настройки приложения

4 Редактирование проекта

4.1 Создание нового проекта

Создать новый проект можно двумя способами.

1. Выполнить меню ***Проект/Создать***.

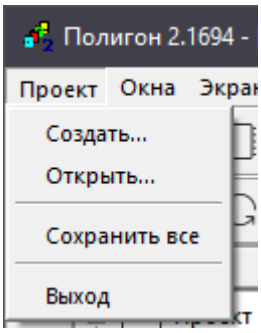


Рисунок 4.1 – Создание нового проекта через меню Проект/Создать

2. Открыть ***Окно/Проекты*** и нажать ***Создать***.

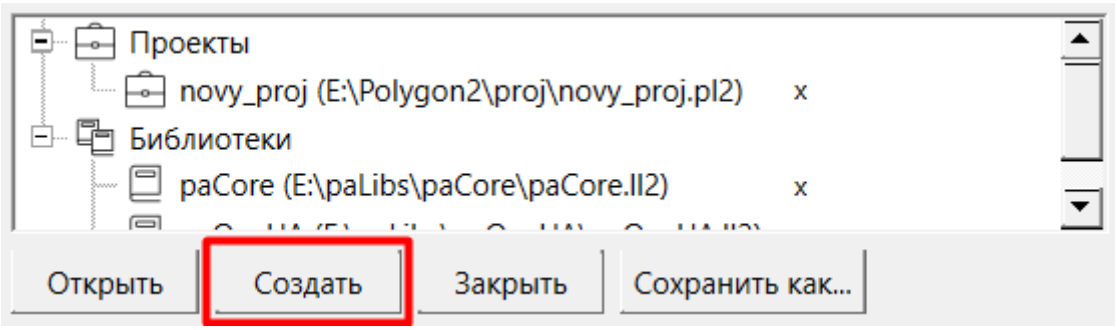


Рисунок 4.2 – Создание нового проекта через окно Проекты

В открывшемся окне следует выбрать место для сохранения и ввести имя нового проекта. Допускается использование латинских букв, символа **«_»** и цифр. Каждый новый проект рекомендуется размещать в отдельной папке, поскольку внутри нее среда разработки создает временные файлы при трансляции и копии проекта (бэкапы).

ВНИМАНИЕ
Путь к файлам проекта не должен содержать кириллицу и пробелы.

Далее следует выбрать шаблон создаваемого проекта – ***Пустой*** проект или ***Модуль с отладчиком для контроллера***. Рекомендуется всегда создавать проект с отладчиком для контроллера, чтобы не настраивать его подключение вручную.

ВНИМАНИЕ
Для создания модуля с отладчиком для контроллера обязательно наличие в среде ссылок на библиотеки ***paCore*** и ***paOpcUA***. Подробнее о добавлении библиотек в среде см. в разделе 1.4.2.

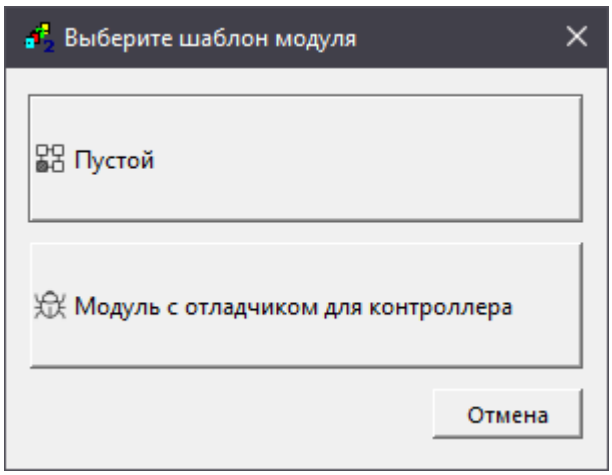


Рисунок 4.3 – Окно выбора шаблона модуля

Отладчик среды Полигон подключается к запущенному проекту как клиент OPC UA. При выборе шаблона модуля с отладчиком в новом проекте в месте работы ***Фон*** автоматически создается программа ***Debug***, на странице которой добавлен блок OPC UA-сервера (***OpcUAServer***) из библиотеки ***paOpcUA***. На входы блока **ip** – IP адрес и **prt** – локальный порт контроллера прописываются соответствующие свойства модуля в виде SQL-запросов.

Запрос IP адреса:

```
"<sql>SELECT value FROM blocks_prop WHERE indx=:module AND type="prop_ip"</sql>"
```

Запрос номера порта:

```
<sql>SELECT value FROM blocks_prop WHERE indx=:module AND type="prop_debug_port"</sql>
```

Подробнее об отладчике среды Полигон см. в разделе 7.

Подробнее реализация протокола **OPC UA** в среде Полигон описана в документе Обмен с верхним уровнем. Библиотека paOpcUA.

После выбора шаблона для модуля следует задать пароль для доступа отладчика к запущенному проекту. Если необходимо его можно будет поменять в свойстве модуля ***Пароль admin***. Изменение вступит в силу после трансляции проекта.

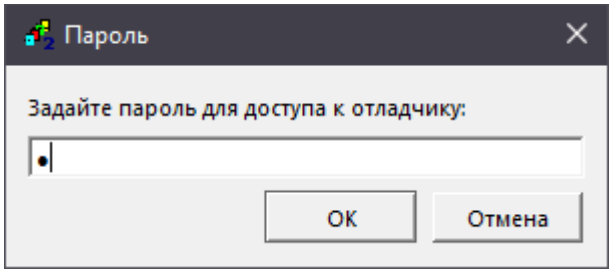


Рисунок 4.4 – Окно задания пароля для доступа к отладчику

После задания пароля среда выведет сообщение об успешном создании проекта и предложением открыть его в окне представления типа ***Дерево***.

Ссылку на созданный проект можно увидеть в окне ***Проекты***. Для начала работы с созданным проектом необходимо открыть его в окнах представления типов ***Дерево*** и ***Страница***.

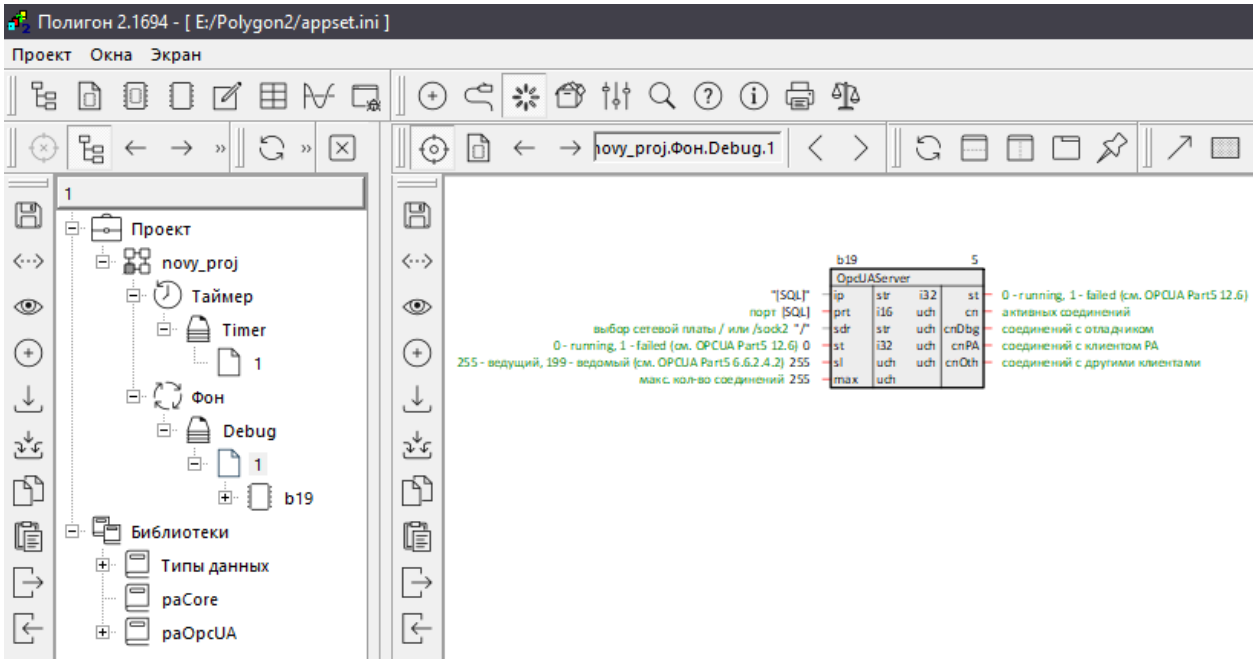


Рисунок 4.5 – Новый проект в представлении Дерево и Страница

4.2 Создание компонентов проекта

Создать компоненты проекта внутри существующих можно в представлении ***Дерево***. Структура компонентов проекта описана в разделе 2.2.

Таким образом, внутри проекта создаются ***Модули***. Внутри ***Модуля*** – ***Места работы*** и ***Разделы***. Внутри ***Мест работы*** создаются ***Программы*** и т.д.

Чтобы создать новый компонент в проекте следует:

1. Кликнуть ПКМ на модуле, месте работы (таймере или фоне), программе и т.д.
2. В контекстном меню выбрать команду ***Создать***.
3. Выбрать тип нового компонента и, если необходимо, ввести имя, номер, количество.

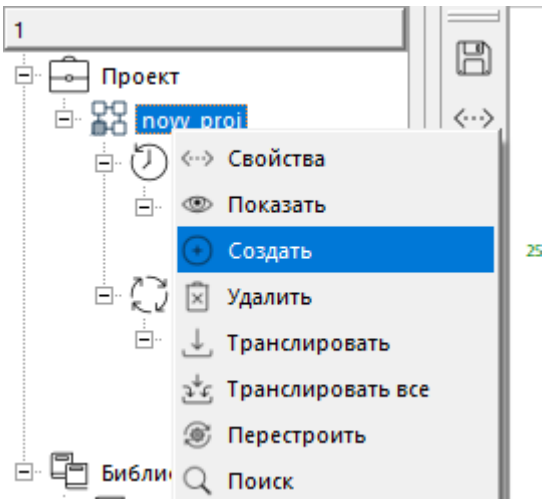


Рисунок 4.6 – Создание компонентов внутри модуля

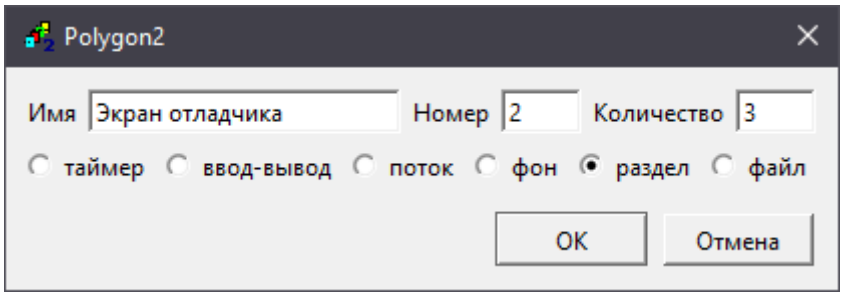


Рисунок 4.7 – Пример: создание разделов внутри модуля

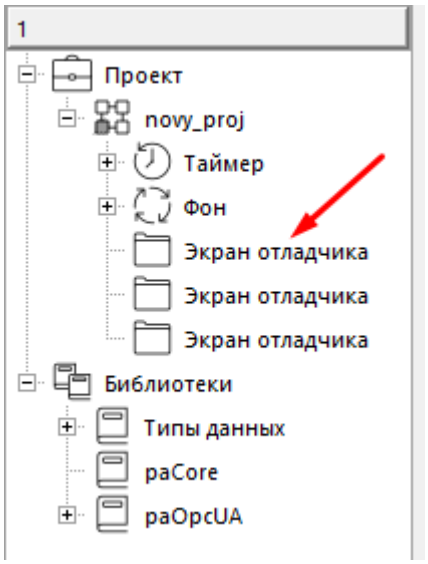


Рисунок 4.8 – Результат создания разделов внутри модуля

4.2.1 Создание таймерного потока Ввод-вывод

По умолчанию в проекте добавлено два места работы – ***Таймер*** и ***Фон***. Если необходимо в проект можно добавить второй таймерный поток – ***Ввод-вывод***.

Для его добавления следует в контекстном меню модуля нажать ***Создать*** и выбрать ***ввод-вывод***.

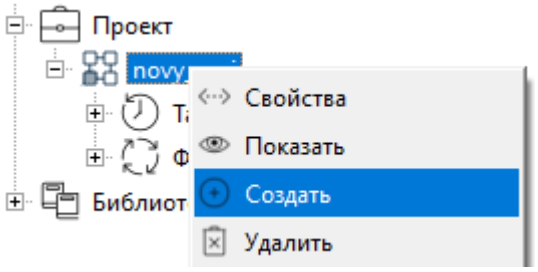


Рисунок 4.9 – Создание второго таймерного потока в модуле

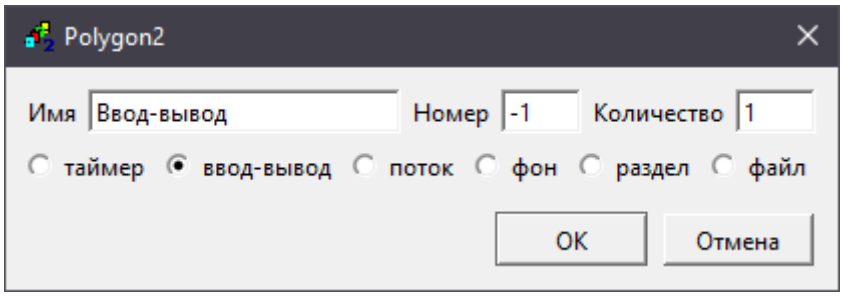


Рисунок 4.10 – Создание потока Ввод-вывод

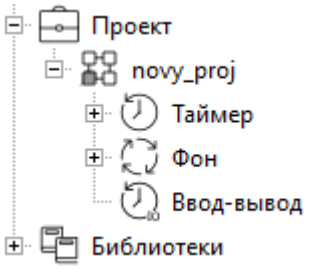


Рисунок 4.11 – Создание второго таймерного потока

Основное свойство ***Таймера*** и ***Ввода-вывода*** – ***Таймерный промежуток*** – время, через которое повторно вызывается программа.

Необходимо соизмерять размер программ (количество функциональных блоков), находящихся в месте работы ввод-вывод, со временем таймерного цикла. При недостаточном интервале таймерного цикла, возможно возникновение ошибки времени исполнения – **Time Out**. Данная ошибка повлечет за собой остановку программы (отслеживать реальное время выполнения можно с помощью функционального блока SysInfo).

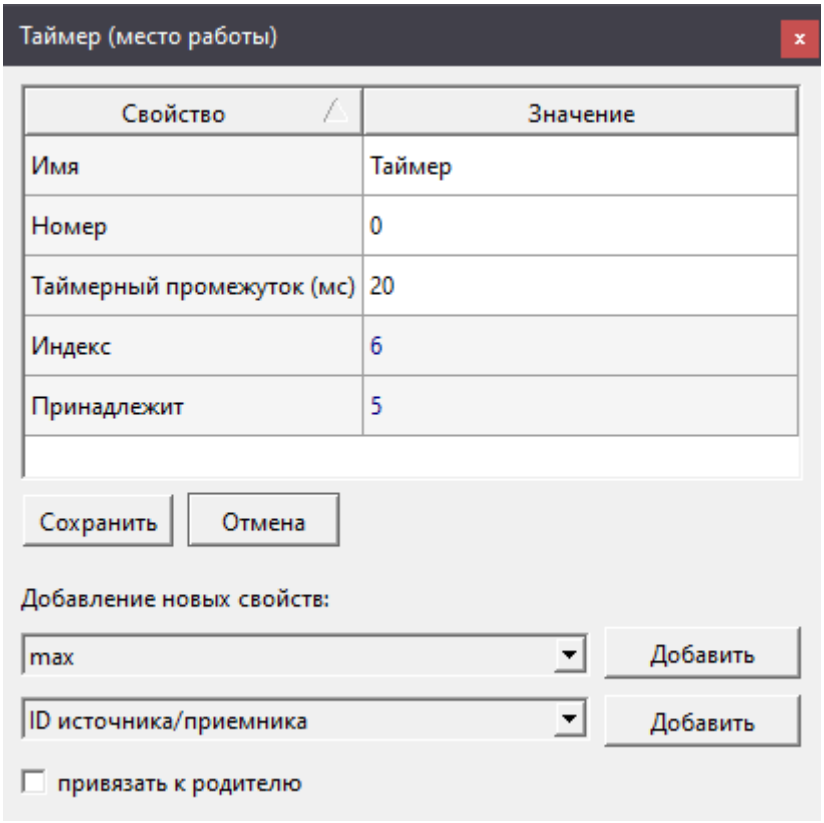


Рисунок 4.12 – Свойства таймерных потоков

4.2.2 Создание места работы Поток

По умолчанию в проекте добавлено два места работы – ***Таймер*** и ***Фон***. В проект также можно добавить несколько мест работы, выполняющихся в фоновых потоках – ***Потоки***.

Для добавления ***Потока*** следует в контекстном меню модуля нажать ***Создать*** и выбрать ***поток***.

Основное свойство ***Фона*** и ***Потока*** – ***Приоритет фона***. Приоритет фона определяет поведение данного потока по отношению к другим потокам.

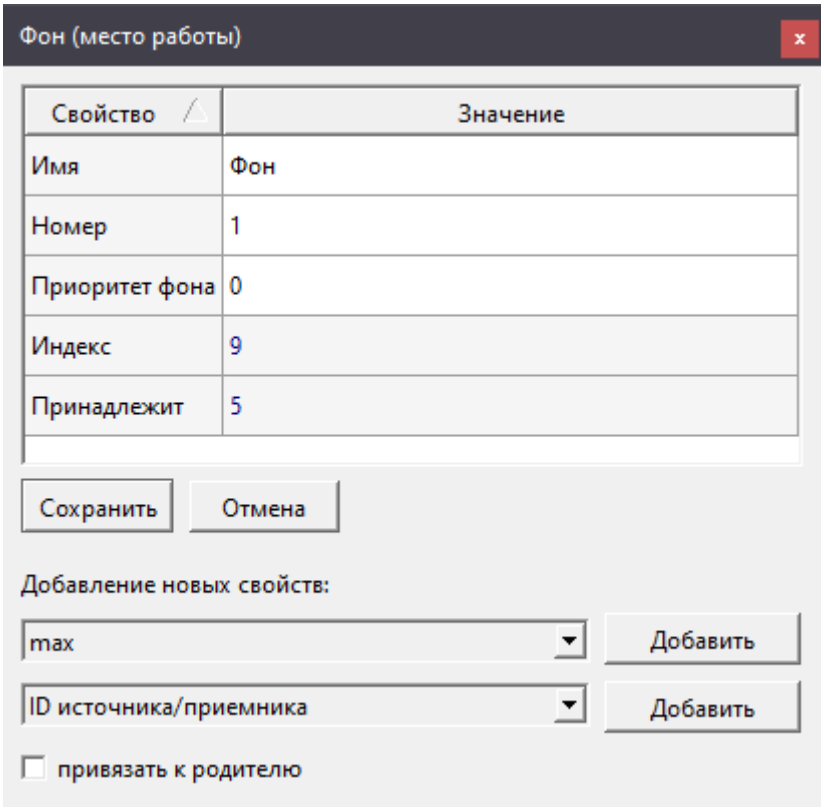


Рисунок 4.13 – Свойства фоновых потоков

4.3 Создание функционального блока

Функциональные блоки создаются внутри страниц. Для создания функционального блока есть несколько способов:

1. Открыть нужную страницу в активном окне, нажать ПКМ на свободном месте и выбрать ***Создать***.

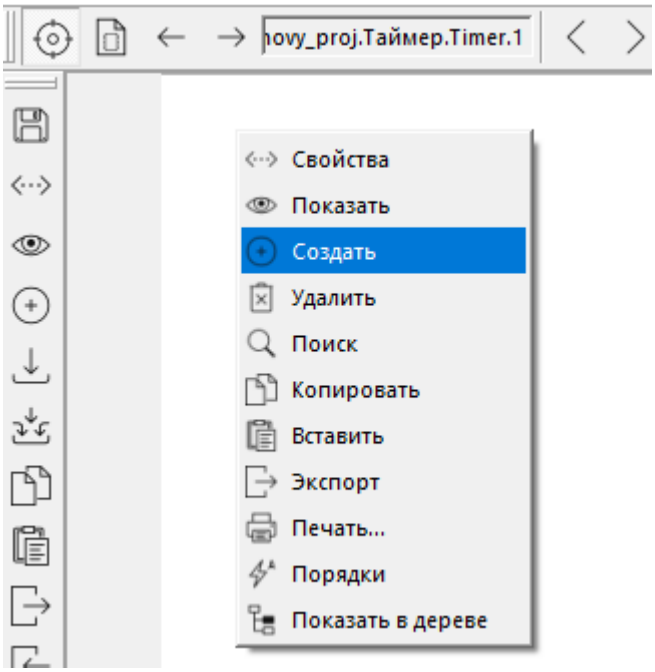


Рисунок 4.14 – Создание функционального блока на открытой странице

2. Нажать на пиктограмму ***Создать*** на панели Инструменты открытой страницы.

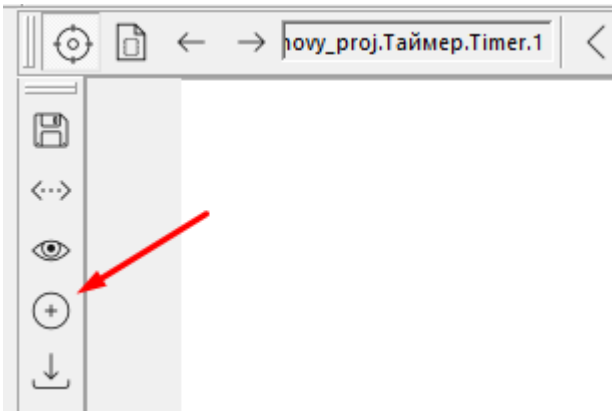


Рисунок 4.15 – Создание функционального блока через панель Инструменты

3. Открыть контекстное меню страницы в представлении ***Дерево*** и нажать ***Создать***.

После выбора команды ***Создать*** откроется окно создания функционального блока.

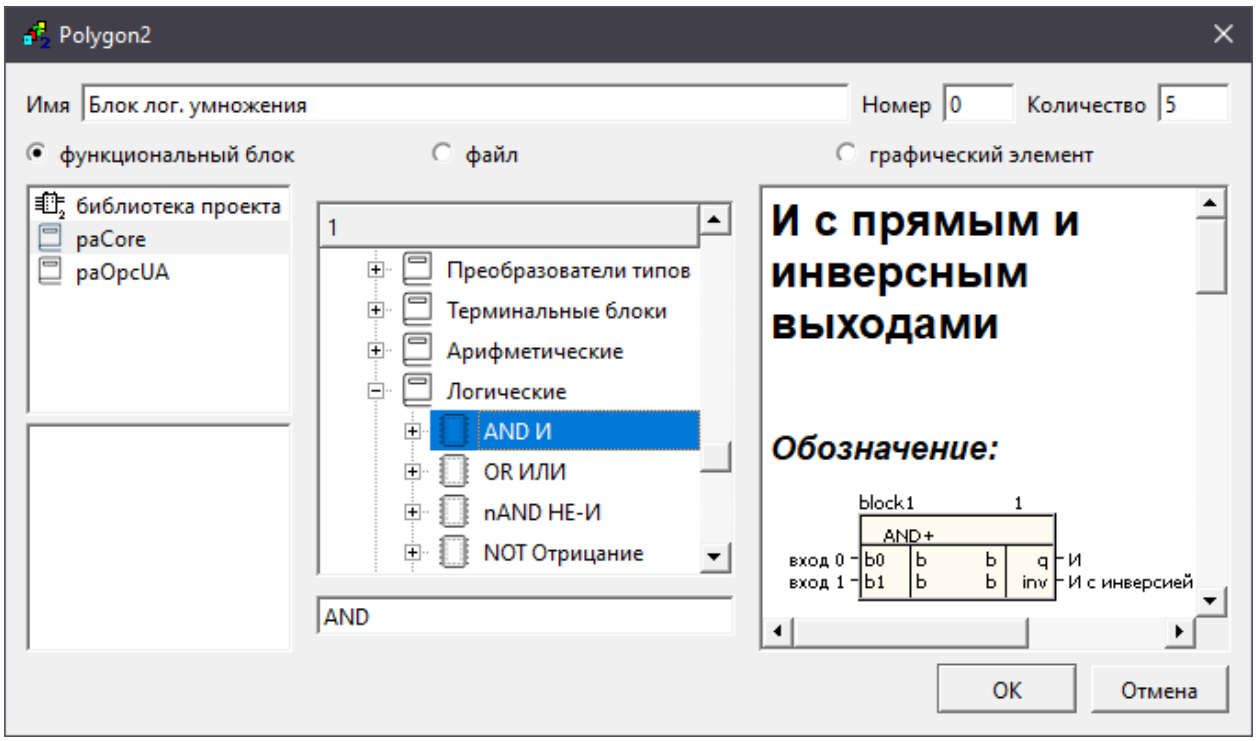


Рисунок 4.16 – Окно создания функционального блока

Сверху окна в поле ***Имя*** можно ввести имя нового функционального блока, которое будет отображаться сверху над блоком и в дереве проекта как комментарий. Если оставить данное поле пустым, то имя блока будет присвоено автоматически по шаблону ***b<индекс блока>***.

В поле ***Количество*** можно указать необходимое количество блоков выбранного типа для создания. По умолчанию создается один блок выбранного типа.

Для создания блока следует:

1. В левом поле окна выбрать библиотеку, функциональный блок из которой планируется создать.
2. В поле посередине отобразиться список разделов выбранной библиотеки, внутри разделов – списки функциональных блоков. Следует выбрать нужный блок.
3. При выделении блока в правой части окна отобразится его справка. Справку на блок, созданный на странице, также можно открыть, нажав на него ПКМ и выбрав ***Справка***.
4. После выделения нужного блока нажать ***ОК*** – новый блок появится на странице.

В поле снизу списка функциональных блоков можно ввести имя нужного функционального блока для поиска по разделам библиотеки.

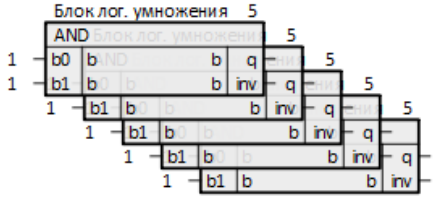


Рисунок 4.17 – Результат создания функциональных блоков

Также можно создать ФБ на странице путем перетаскивания его из дерева библиотеки. Для этого следует:

1. Открыть в представлении Дерево нужную библиотеку.
2. Выбрать функциональный блок.
3. Перетащить блок на открытую страницу.
4. При отпускании мыши появится контекстное меню, в котором следует выбрать ***Создать***.

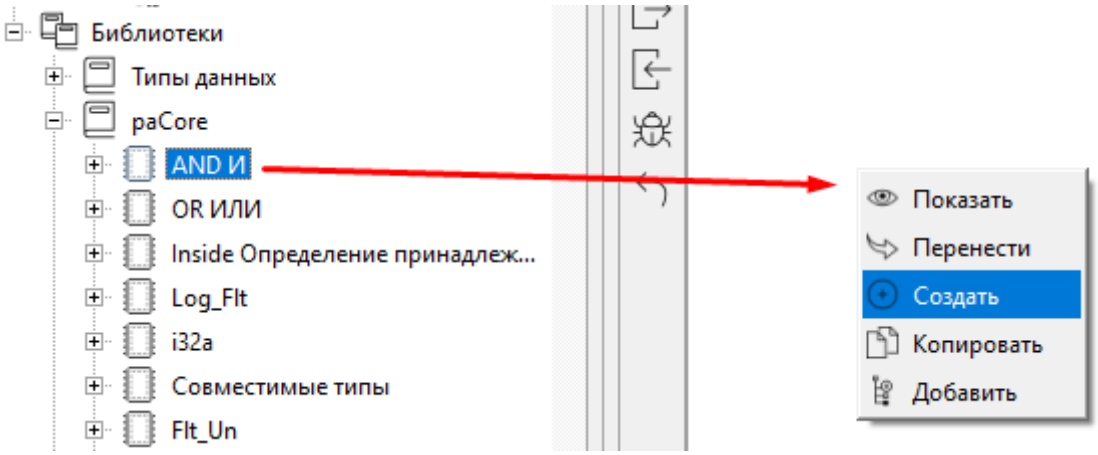


Рисунок 4.18 – Создание функционального блока из дерева библиотеки

Некоторые ФБ могут быть размещены только в таймерном потоке или в фоне (определяется свойством ***Может работать только в…***), при попытке разместить такой блок не в указанном месте работы появится окно соответствующей ошибки.

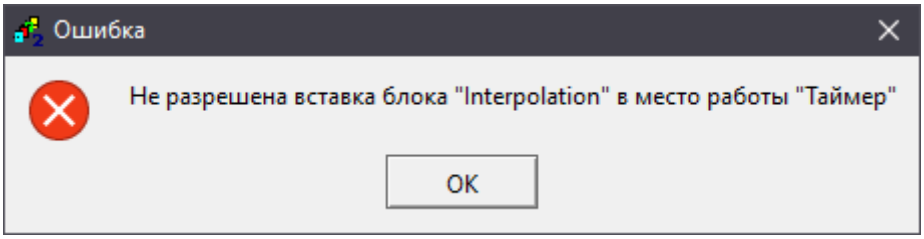


Рисунок 4.19 – Ошибка вставки блока

4.4 Создание циклических входов/выходов и групп входов/выходов

Многие блоки в проекте могут иметь разное количество входов и выходов. Такие входы и выходы блока, а также группы входов и выходов, называются циклическими (см. раздел 2.5.1).

Для того чтобы добавить входы/выходы у блока, необходимо выделить его ПКМ и выбрать команду ***Создать***. В появившемся окне следует ввести необходимое ***Количество*** добавляемых входов/выходов и нажать ***ОК***.

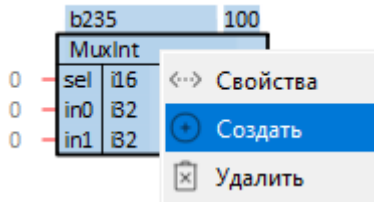


Рисунок 4.20 – Создание входов/выходов блока

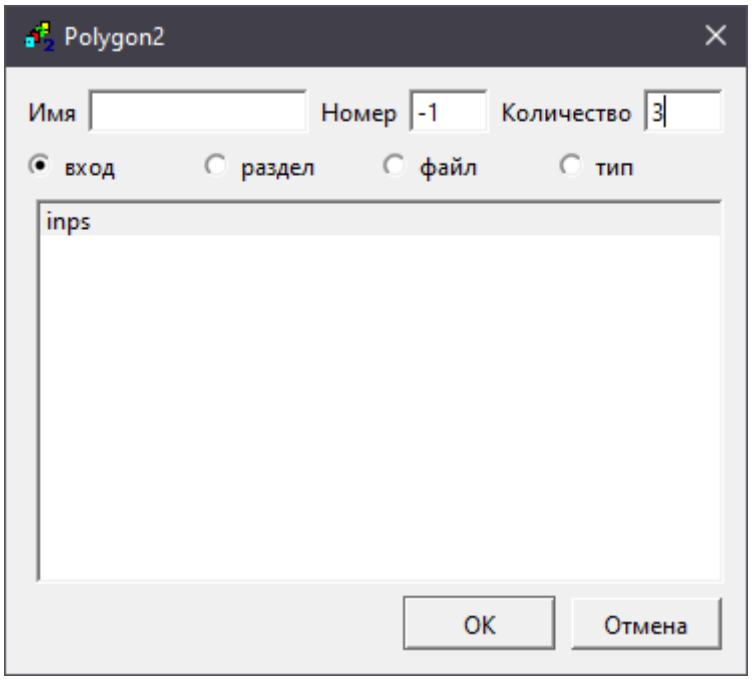


Рисунок 4.21 – Создание входов/выходов блока

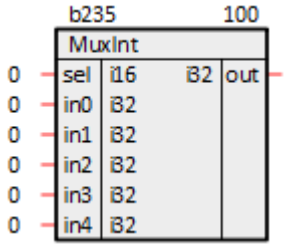


Рисунок 4.22 – Результат создания входов/выходов блока

Для удаления лишних входов/выходов можно выделить их с зажатым **Shift** и выполнить: ПКМ – ***Удалить***.

4.5 Комментарии у входов/выходов блоков

Около входов/выходов ФБ можно оставлять комментарии. Добавить комментарий входу/выходу можно двумя способами:

1. Выделить необходимый вход/выход, зайти через контекстное меню в его ***Свойства***, добавить свойство ***Комментарии***, ввести необходимый комментарий, нажать ***Сохранить***. Комментарий появится около входа.

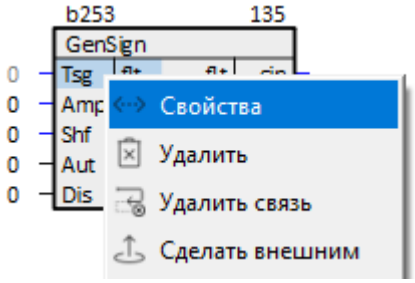


Рисунок 4.23 – Добавление комментария входу/выходу

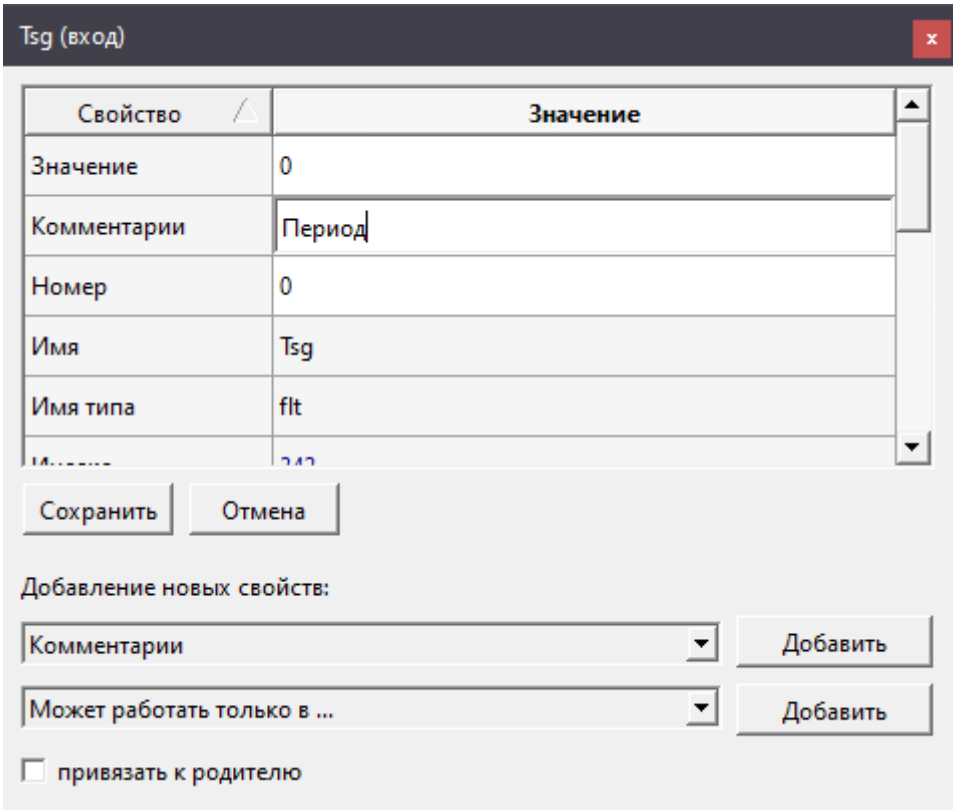


Рисунок 4.24 – Свойство Комментарии

2. Дважды кликнуть около интересующего входа/выхода – появится строка ввода комментария, следует ввести комментарий, он отобразится зеленым цветом.

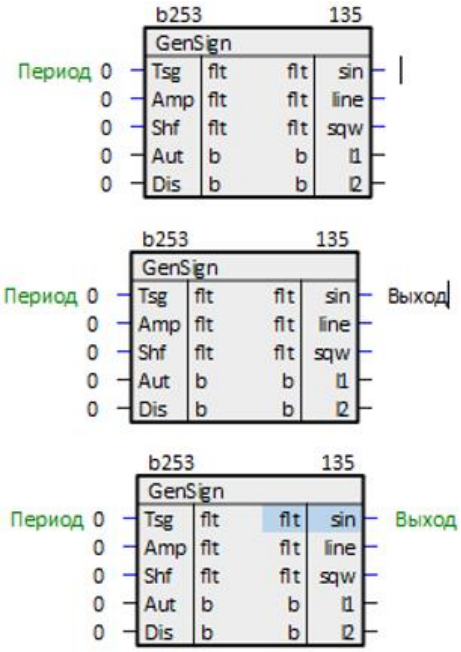


Рисунок 4.25 – Добавление комментария входу/выходу у блока

4.6 Проведение связей

Для того чтобы создать связь между входом и выходом функциональных блоков на странице следует щелкнуть ЛКМ на входе или выходе и, не отпуская левую кнопку мыши, подвести указатель к другому выходу или входу. При отпускании левой кнопки мыши связь будет проведена.

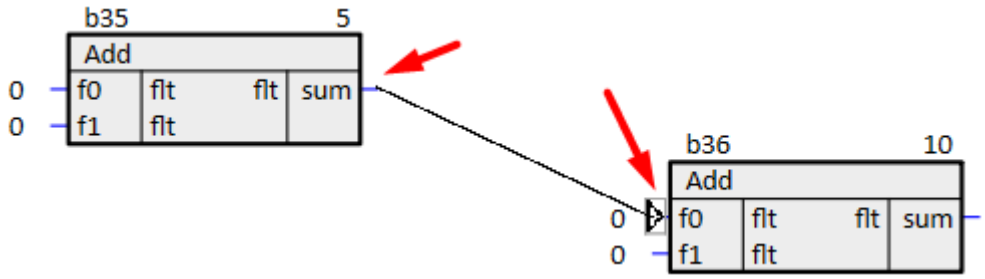


Рисунок 4.26 – Создание связи между ФБ

При проведении связи между блоками, находящимися на разных страницах, необходимо открыть эти страницы в рабочих окнах и провести связь как указано выше. При этом на каждой странице будут проведены связи, соответственно, в левое и правое поле. На полях страницы будет указан адрес, куда идет связь.

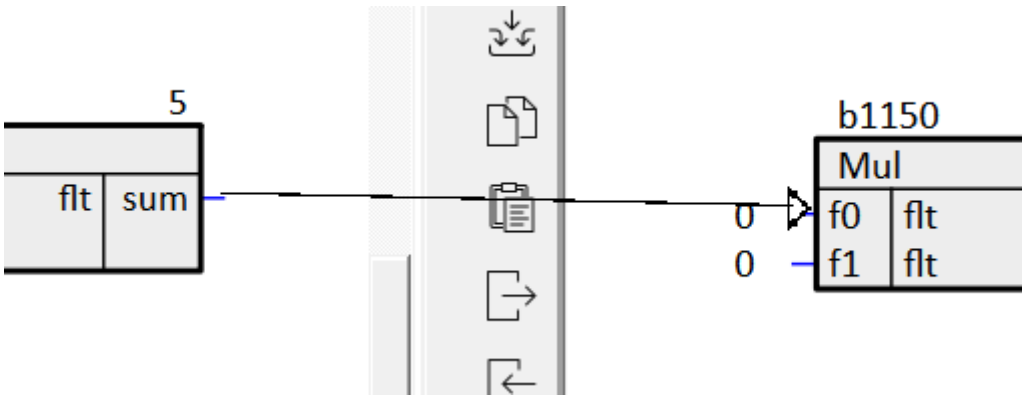


Рисунок 4.27 – Создание связи между ФБ, находящимися на разных страницах

Рисунок 4.28 – Ссылки на полях

При создании связи учитываются типы данных входа и выхода, и, если типы оказываются несовместимы, выдается сообщение об ошибке и связь не проводится. Невозможно создание связи на Константный вход.

Цвет связей отображает тип данных входа:

- ***черный*** – булевое;
- ***красный*** – целое;
- ***синий*** – вещественное;
- ***серый*** – строковое.

Для удаления связи нужно выделить правой кнопкой мыши вход, куда приходит связь. При этом откроется меню, в котором нужно выбрать ***Удалить связь***.

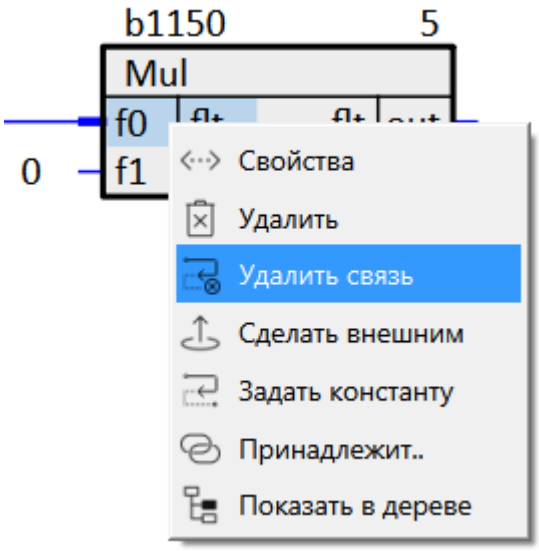


Рисунок 4.29 – Удаление связи

Связь можно переносить с одного входа на другой. Для этого нужно «схватить» связь на входе и, не отпуская, перенести ее на новый вход.

Связь можно создавать с входа/выхода на странице проекта к выходу/входу в дереве. Для этого следует захватить интересующий вход/выход и, зажав **Ctrl**, перетащить его на выход/вход в дереве, к которому необходимо провести связь. В появившемся меню необходимо выбрать ***Провести связь***.

Аналогично можно провести связь с входа/выхода в дереве проекта к выходу/входу на странице. Для этого вход/выход из дерева следует захватить мышью и перетащить на выход/вход на странице. В появившемся меню выбрать ***Провести связь***.

Из «дерева в дерево» также можно провести связь: захватить нужный вход/выход, перетащить его на выход/вход, к которому необходимо провести связь – выбрать в меню ***Провести связь***.

ПРИМЕЧАНИЕ
При проведении связей следует помнить: если провести связь от выхода к входу – комментарий выхода дублируется на вход. Если проводить связь от входа к выходу – комментарий не дублируется. Подробно о комментариях см. в разделе 4.5.

Для установления скрытой связи можно использовать ***Глобальные константы***. Для этого у выхода, с которого будет идти связь, нужно добавить свойства ***Полный алиас*** и ***Глобальная константа***.

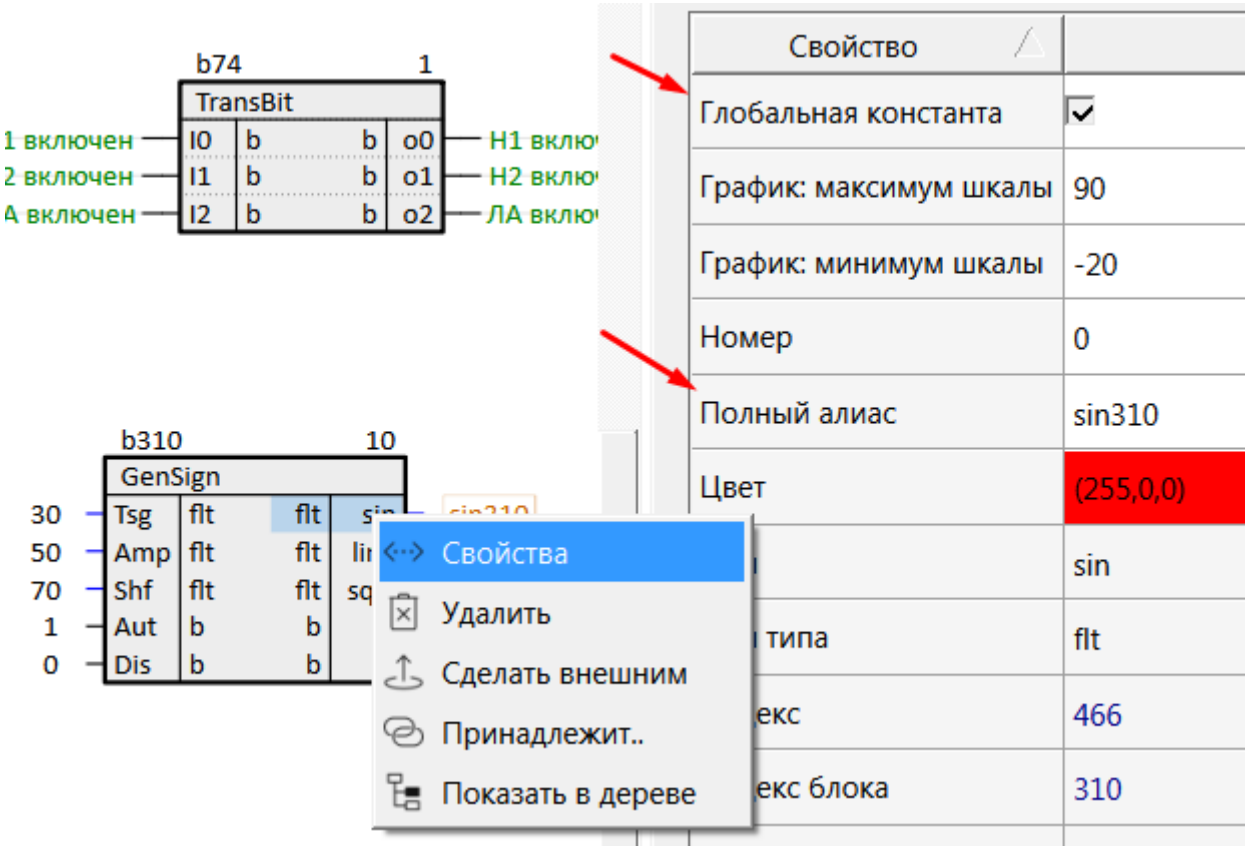


Рисунок 4.30 – Создание глобальной константы

На входе, на который должна идти связь, следует щелкнуть ПКМ и выбрать ***Задать константу***. В открывшемся списке выбрать нужную константу и нажать ***Установить***.

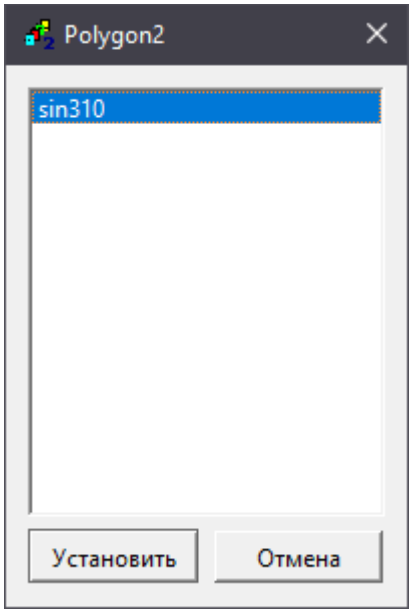


Рисунок 4.31 – Установка глобальной константы

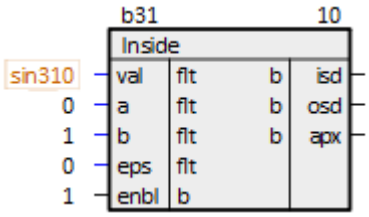


Рисунок 4.32 – Результат проведения скрытой связи

Вид связи можно менять с помощью контекстного меню, которое вызывается щелчком ПКМ на связи. При установке ***Ручное*** на связи появляются точки, потянув за которые можно придать связи любой вид. При установке ***Авто*** связь приобретает автоматически определяемый вид.

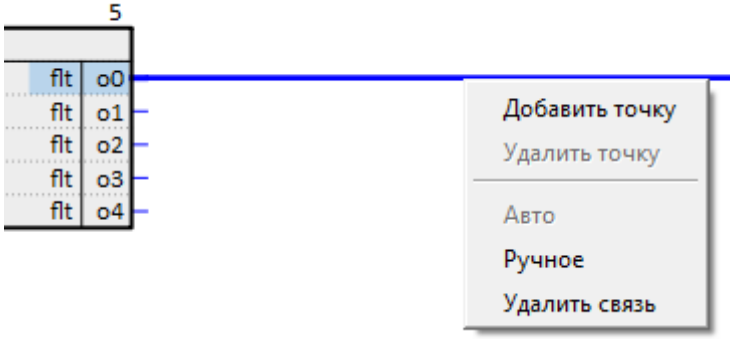


Рисунок 4.33 – Контекстное меню связи

4.6.1 Проведение связей между модулями

При редактировании в проекте двух и более модулей есть возможность проводить связи между ними.

Обмен данными при проведении связей между модулями обеспечивается по протоколу OPC UA.

Подробно реализация протокола OPC UA в среде Полигон описана в документе Обмен с верхним уровнем. Библиотека paOpcUA.

Для проведения связи между модулями необходимо открыть страницы модулей в рабочих окнах и провести связь.

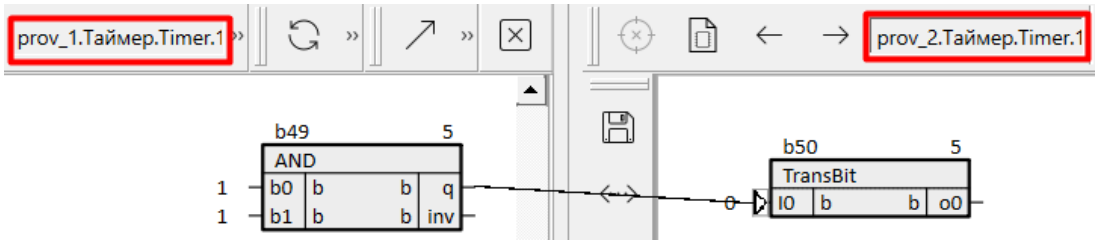


Рисунок 4.34 – Проведение связи между модулями

В появившемся окне необходимо настроить связь между модулями. В одном из модулей выбирается блок OPC UA-сервера, во втором – блок OPC UA-клиента. После настройки связи необходимо нажать ***Провести связь между модулями***.

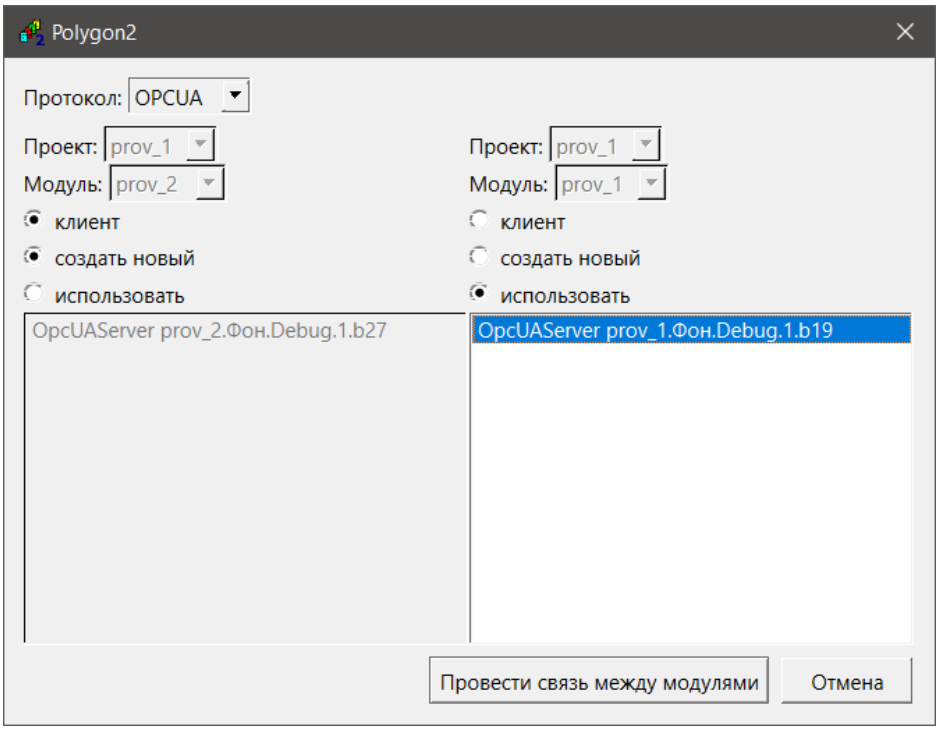


Рисунок 4.35 – Проведение связи между модулями

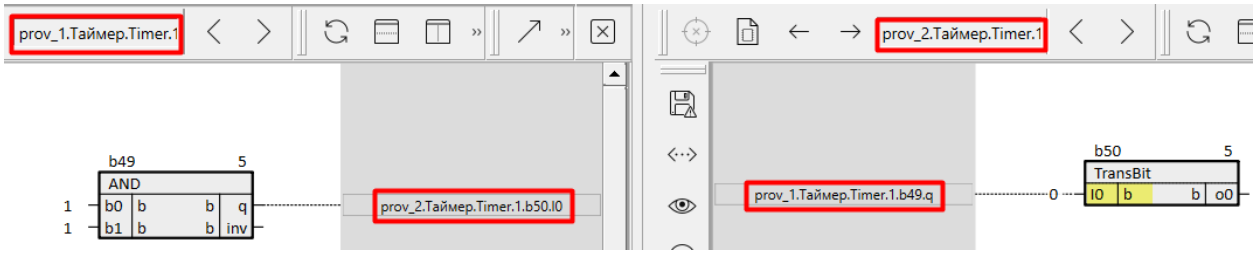


Рисунок 4.36 – Результат проведения связи между модулями

В раздел ***Данные*** выбранного блока OPC UA-клиента добавится вход с прописанным свойством ***ID источника/приемника***, равным индексу входа или выхода в модуле, который был выбран как OPC UA-сервер.

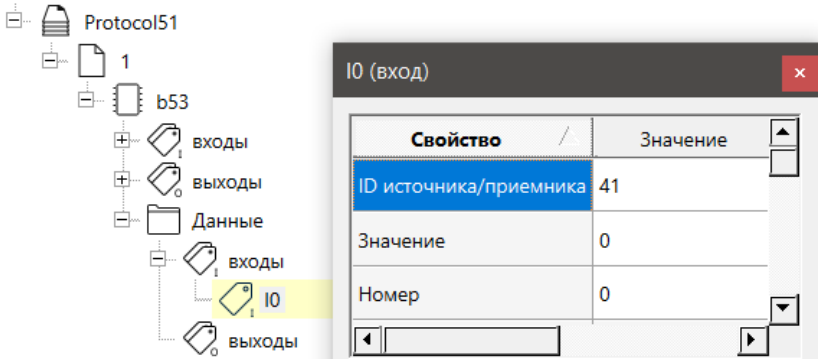


Рисунок 4.37 – Свойства входа, добавленного в раздел Данные

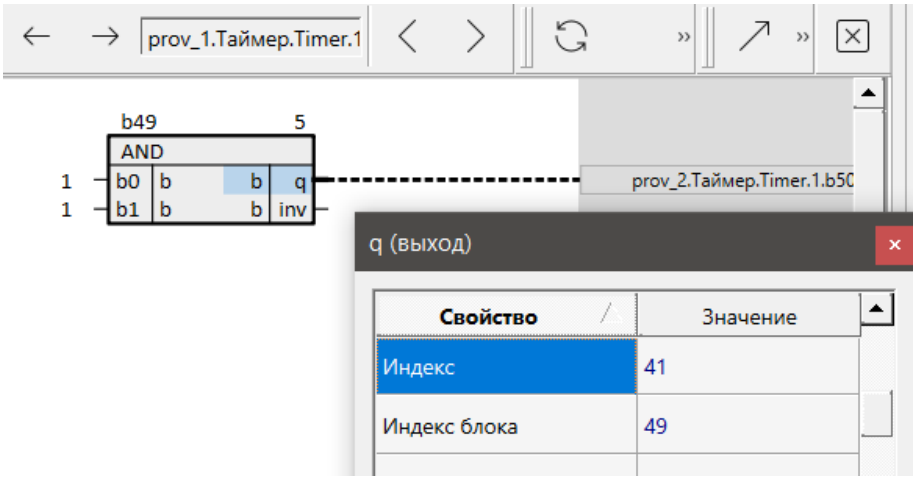


Рисунок 4.38 – Свойства выхода-источника

4.7 Задание порядка выполнения функциональных блоков на странице

Для задания очередности выполнения блоков на странице следует нажать на свободном месте страницы ПКМ и выбрать ***Порядки***.

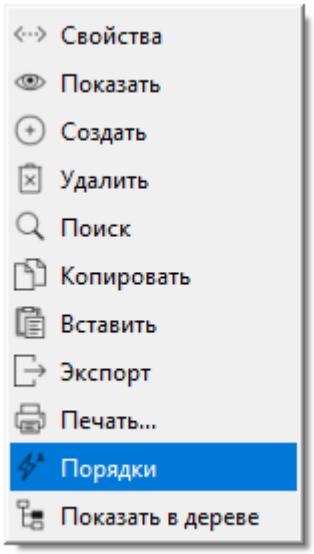


Рисунок 4.39 – Порядки

В открывшемся окне можно выбрать следующие порядки:

- ***Одинаковый порядок***;
- ***Порядок по возрастанию***;
- ***Порядок по потоку данных***.

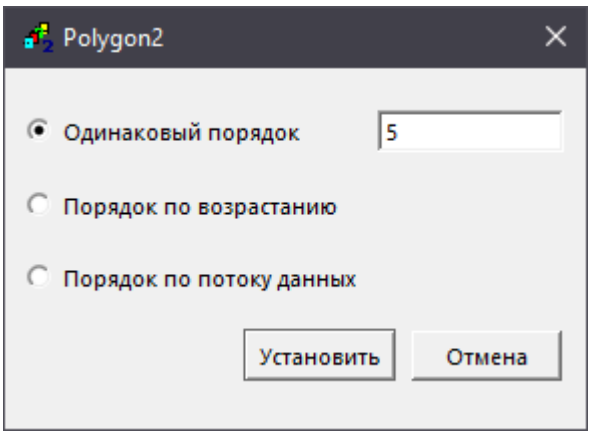


Рисунок 4.40 – Установка порядков блоков на странице

При выборе пункта ***Одинаковый порядок*** всем блокам на странице будет присвоен порядок выполнения, указанный в поле справа.

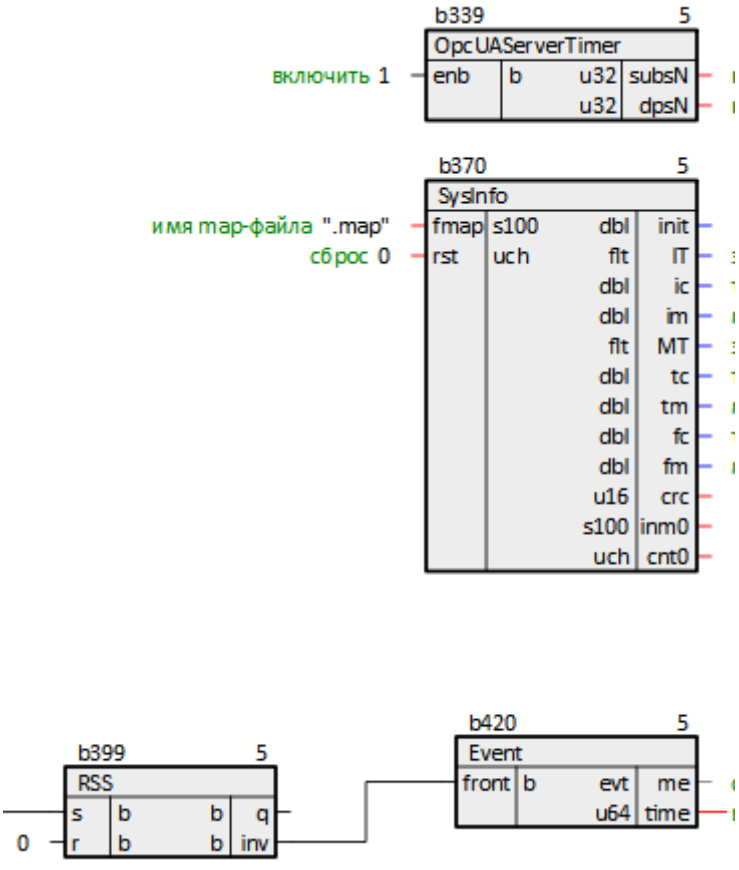


Рисунок 4.41 – Одинаковый порядок выполнения

При выборе пункта ***Порядок по возрастанию*** блокам будут присвоены порядки от **5**, начиная с верхнего левого угла, кратно **5**. Чем левее и выше находится блок, тем меньший порядок выполнения у него будет. Чем правее и ниже – тем больший порядок выполнения.

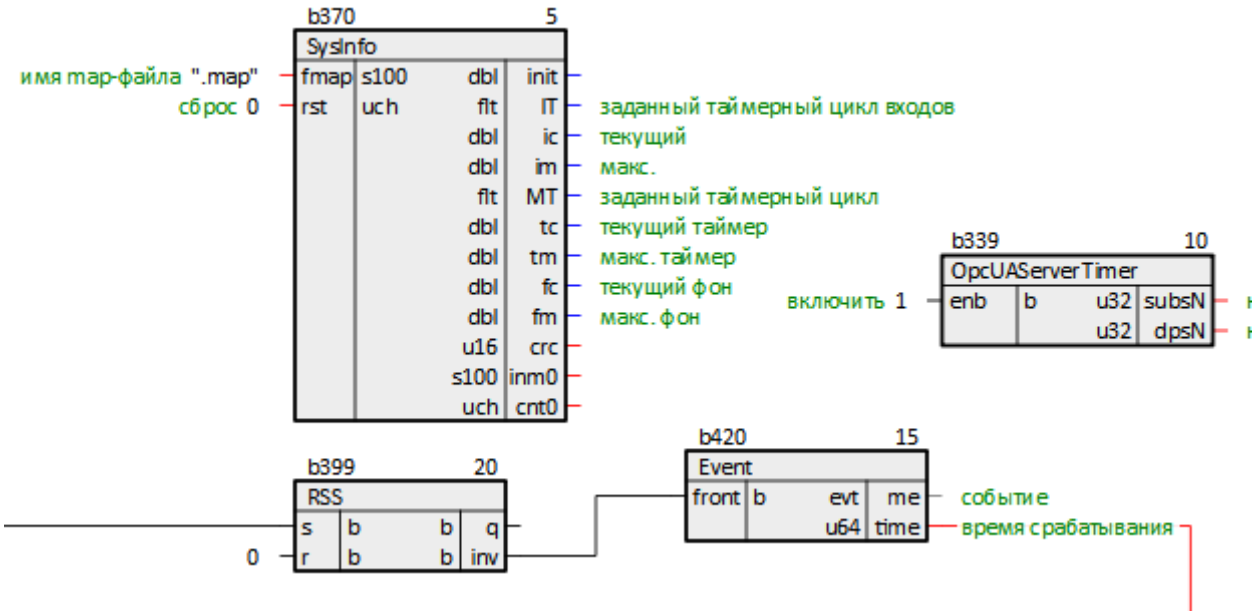


Рисунок 4.42 – Порядок выполнения по возрастанию

При выборе пункта ***По потоку данных*** очередность блоков будет определена их связями друг с другом на текущей странице – если с выхода блока идет связь на вход другого блока, то первый получит меньший порядок исполнения.

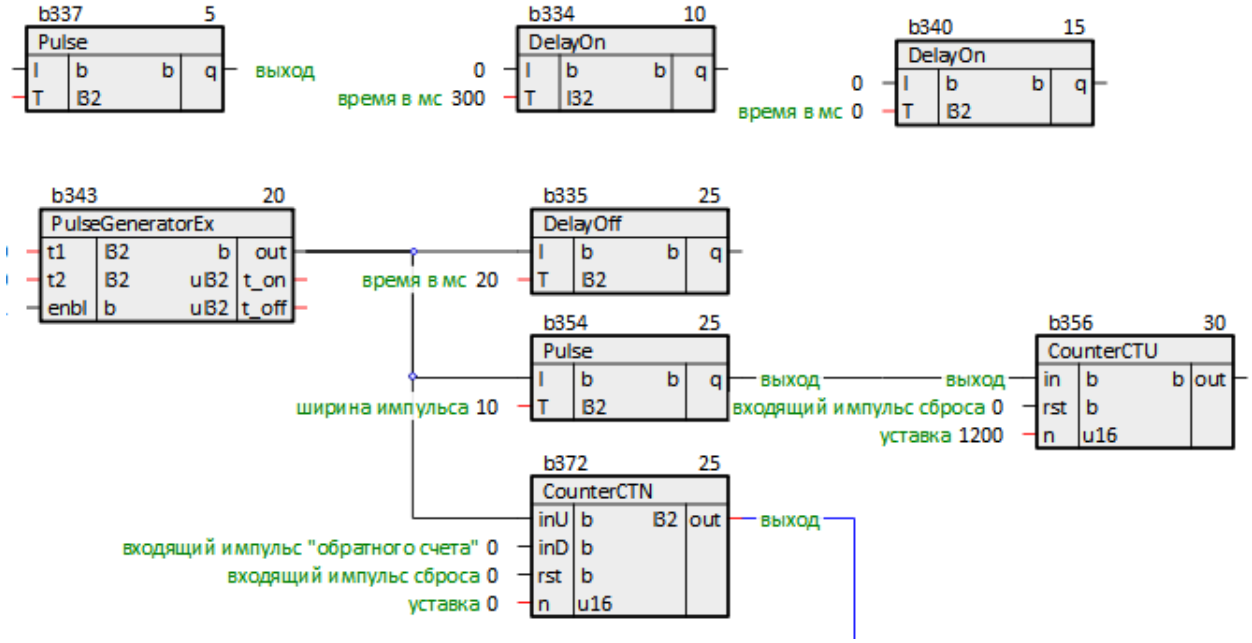


Рисунок 4.43 – Порядок выполнения по потоку данных

4.8 Добавление стрелок, фона и текста на страницы

Для добавления комментариев и заметок на страницы проекта и экраны отладчика предусмотрены элементы ***стрелка***, ***фон*** и ***текст***. Добавить их на страницу можно с панели инструментов ***Редактирование***.

Для добавления панели следует в меню ***Окнв/Панели инструментов*** поставить флаг ***Редактирование***.

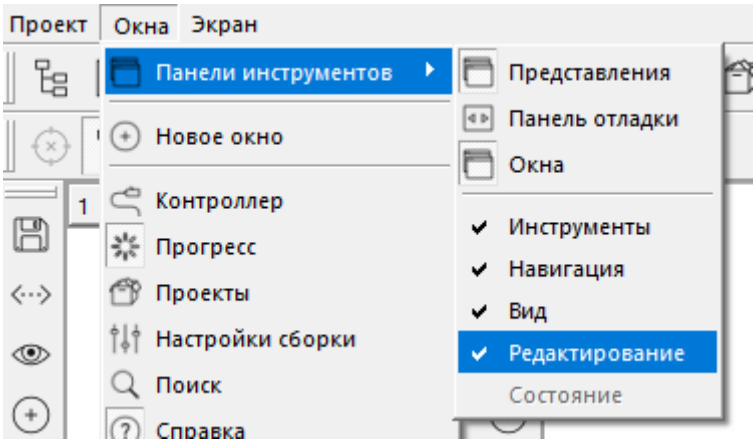


Рисунок 4.44 – Добавление панели Редактирование

Вверху активной страницы появится панель, на которой можно выбрать элементы для редактирования: ***стрелка***, ***фон***, ***текст***. В свойствах созданных элементов есть возможность менять их координаты, задавать размеры и цвет.

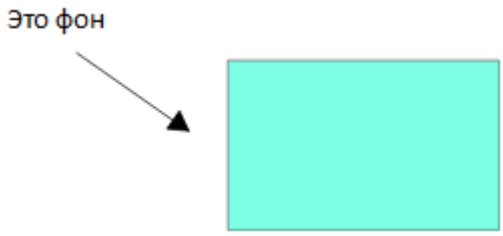


Рисунок 4.45 – Отображение элементов панели редактирования на странице

4.9 Добавление входов и выходов в разделы

В компонент проекта ***Раздел*** можно добавлять входы/выходы со страниц программ для отображения на экране отладчика, для вывода на график, для передачи сигналов OPC UA-клиенту и др.

Для добавления входа или выхода в раздел следует выделить нужный вход или выход и, зажав клавишу **Ctrl** и левую кнопку мыши, перетащить его на нужный раздел в дереве проекта и выбрать ***Добавить*** в контекстном меню.

Также возможно добавить сразу несколько входов/выходов одного блока в раздел, для этого следует выделять их с зажатой клавишей **Shift**.

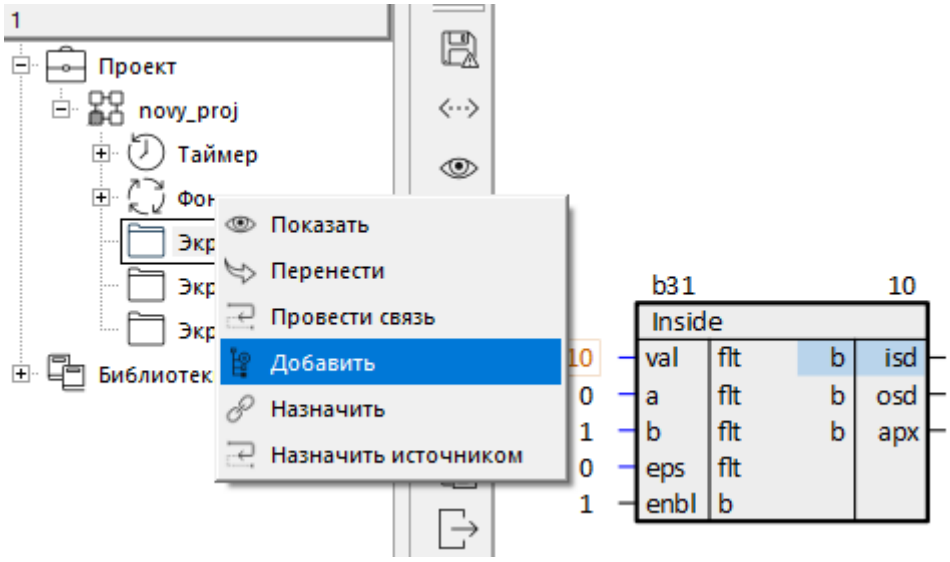


Рисунок 4.46 – Добавление входа/выхода в раздел

После добавления в раздел вход или выход блока окрасится в желтый цвет. При нажатии на нем правой кнопкой мыши и выборе в меню ***Принадлежит*** в списке будет отображен функциональный блок и разделы, к которым принадлежит этот вход или выход.

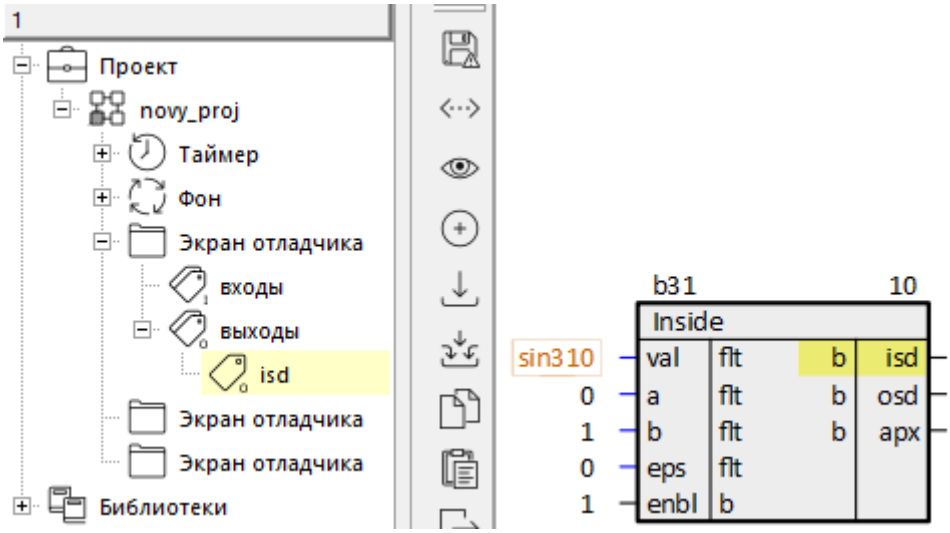


Рисунок 4.47 – Добавление входа/выхода в раздел

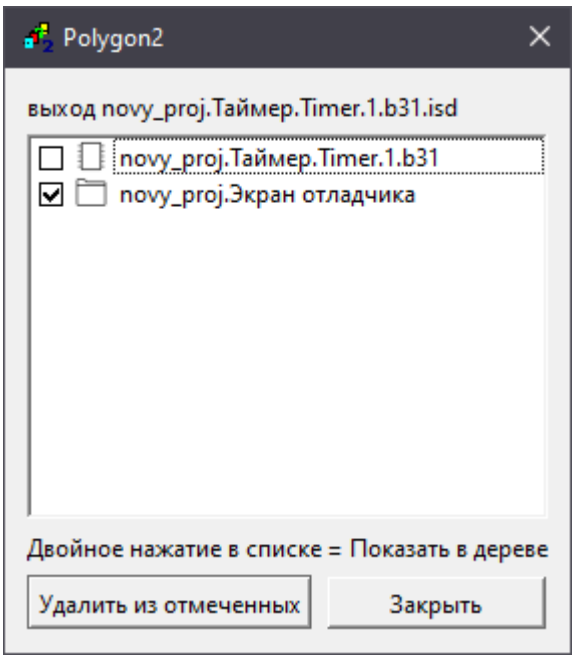


Рисунок 4.48 – Окно Принадлежит

При двойном щелчке по строке списка откроется расположение входа или выхода в дереве.

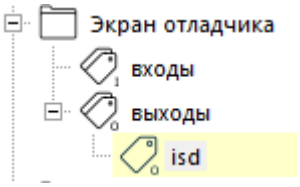


Рисунок 4.49 – Отображение входа/выхода в дереве

4.10 Навигация по проекту

Для навигации по проекту можно воспользоваться панелью ***Навигация***, которая добавляется установкой флага ***Навигация*** в меню ***Окна/Панели инструментов***.

Подробно панели инструментов окон представления описаны в разделе 3.1.1.1.

При нажатии на кнопку ***Списки выбора*** на панели отобразиться поле, в выпадающем меню которого можно выбрать соответствующее окно представления для активной страницы.

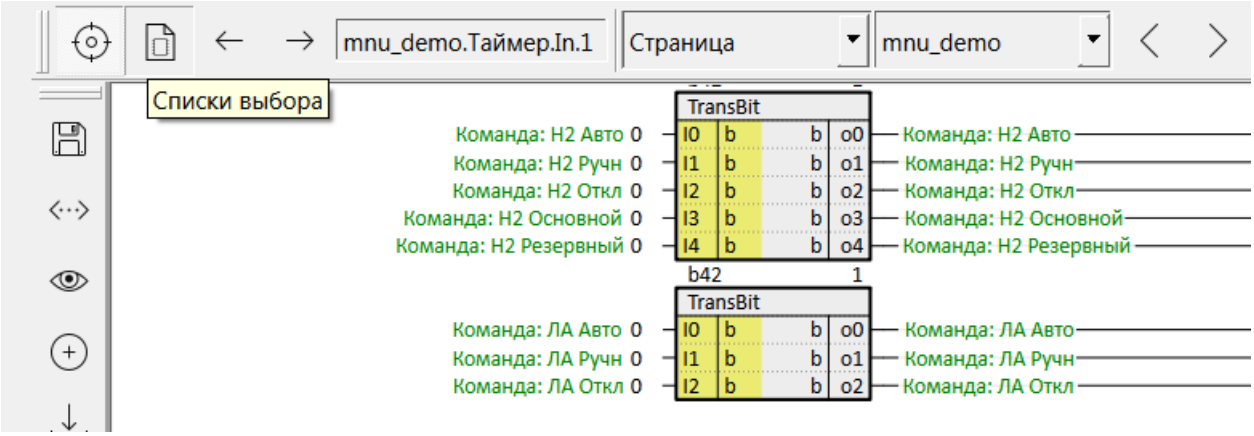


Рисунок 4.50 – Навигация по проекту

Используя кнопки ***Назад*** и ***Вперед*** осуществляется перемещение между ранее открытыми страницами. Кнопки ***Предыдущая страница*** и ***Следующая страница*** производят переключение в последовательности, как страницы расположены в проекте (сверху-вниз).

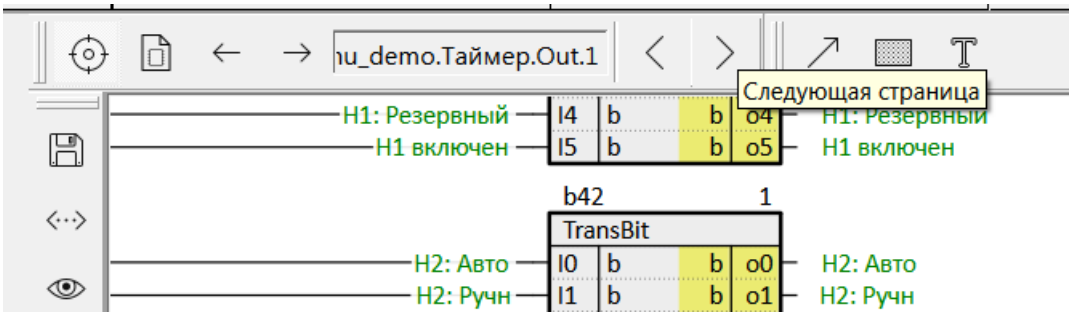


Рисунок 4.51 – Кнопки Назад и Вперед

Также возможно перемещение по полям страниц – следует щелкнуть мышью на адрес связи и в активное окно отобразится соответствующая страница.

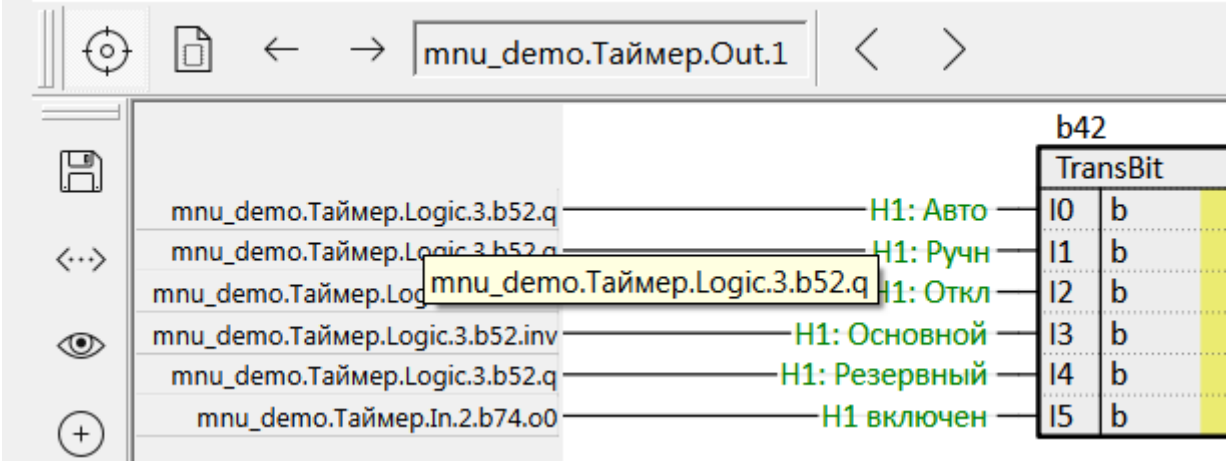


Рисунок 4.52 – Перемещение по адресам связи

Открыть нужную страницу в активном окне также можно два раза щелкнув на ней мышью в дереве, либо перетащив ее из дерева в это окно и нажав ***Показать***.

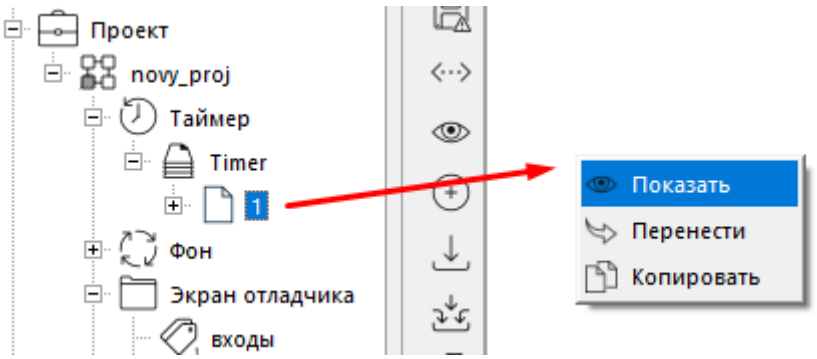


Рисунок 4.53 – Открытие страницы из дерева

4.11 Копирование частей проекта

Для копирования части проекта следует выделить нужную часть при помощи мыши (для выделения нескольких частей – зажав **Shift**) и нажать ***Копировать*** на панели инструментов. Также можно воспользоваться комбинацией **Ctrl + c**.

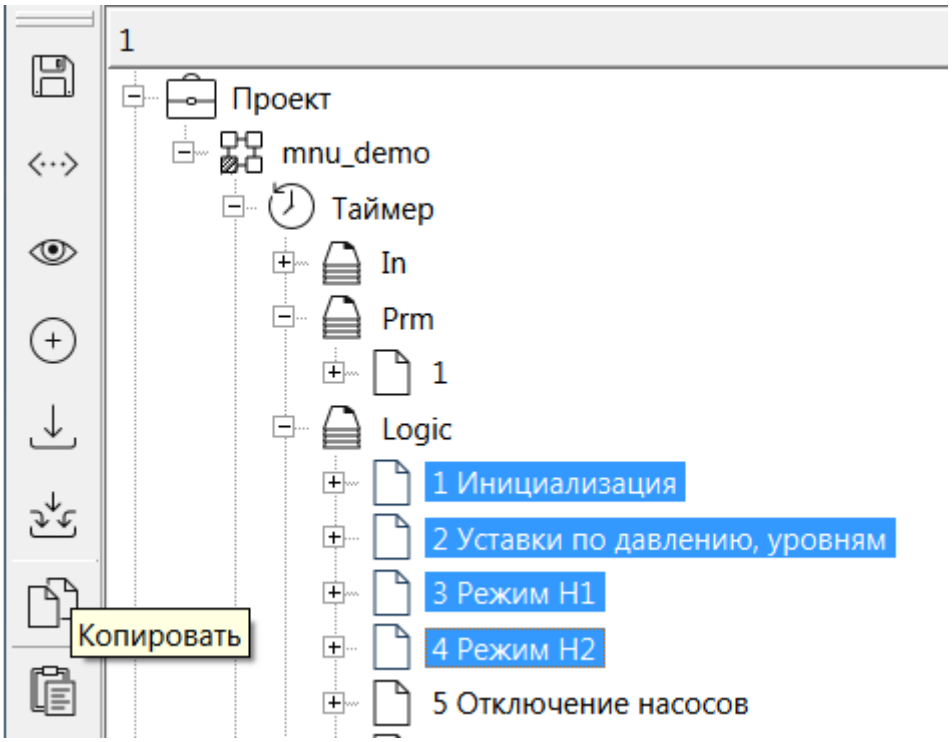


Рисунок 4.54 – Копирование частей проекта

Далее перейти в то место проекта, куда следует произвести вставку и нажать на панели инструментов ***Вставить***. Также произвести вставку можно через контекстное меню, открывающееся при нажатии ПКМ на компоненте дерева, или сочетанием клавишей **Ctrl + v**.

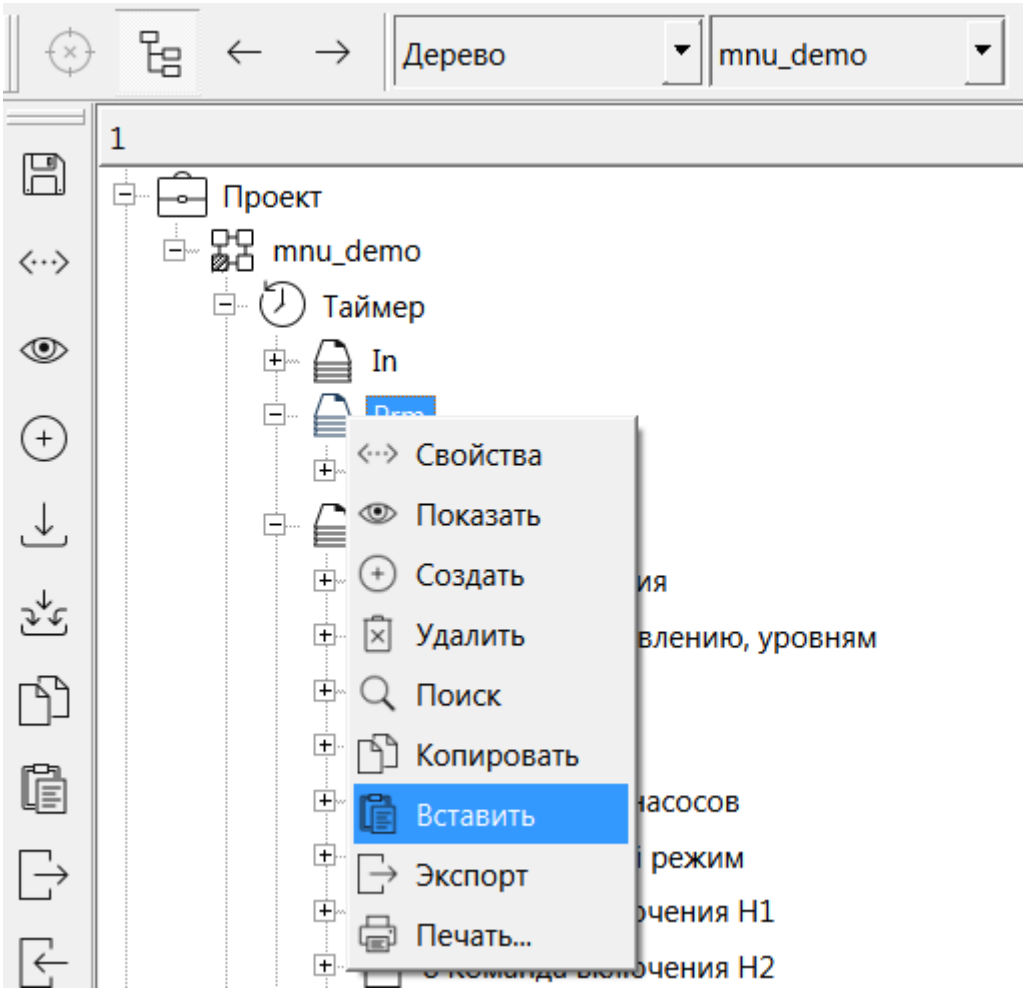


Рисунок 4.55 – Вставка скопированных разделов

Расположение компонентов и связи, которые были внутри вставляемой части, сохранятся.

Также копировать части проекта можно перетаскиванием. Для этого нужно «схватить» мышью требуемую часть проекта и перетащить в новое место. В открывшемся меню выбрать команду ***Копировать*** (если выбрать ***Перенести***, то часть проекта удалится из начального места).

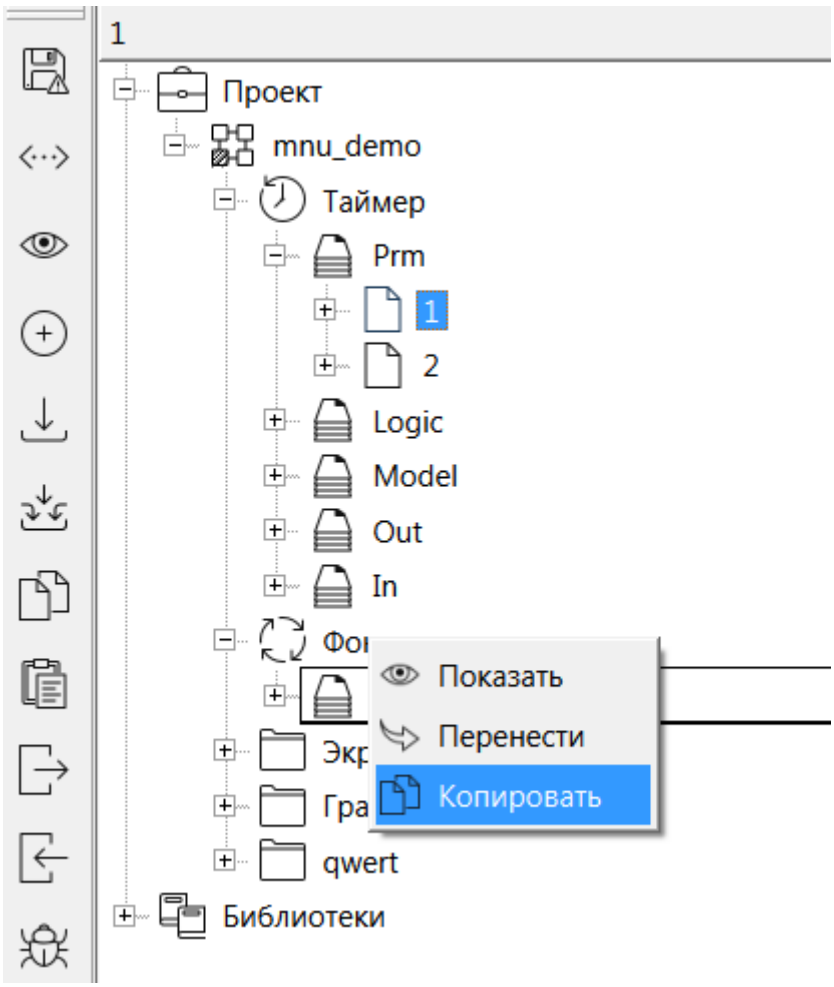


Рисунок 4.56 – Копирование части проекта перетаскиванием

Копировать компоненты возможно не только в рамках одного проекта, но и из разных.

При копировании модулей, мест работы, программ, функциональных блоков, графических элементов имена объектов изменяются автоматически в соответствии со следующими правилами:

1. Имя вида ***b<индекс блока>*** присваивается при создании блоков, для которых не задано пользовательское имя. При копировании в имени нового блока отобразится его индекс.

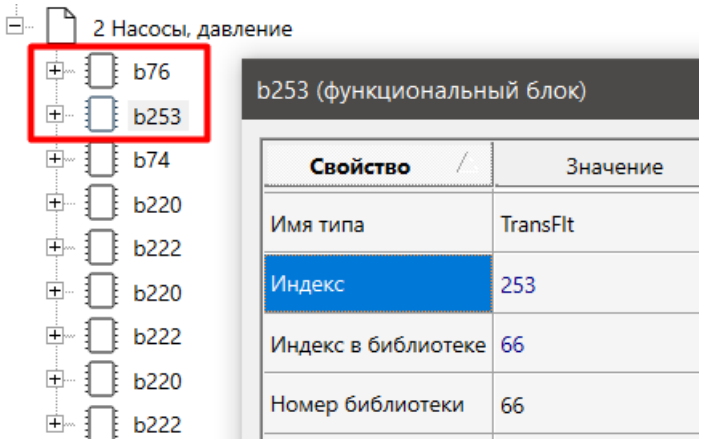


Рисунок 4.57 – Копирование блока b76. Имя нового блока b253 определяется по индексу

2. Кастомные имена могут быть заданы пользователем или сформированы при генерации блоков из MS Excel. Переименование таких блоков зависит от наличия порядкового номера в имени.
   - Если имя без цифр в конце, то добавится суффикс **«_»** и номер вида ***01*** (дополнение до двух цифр).

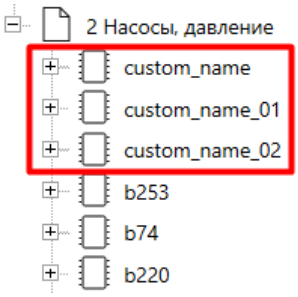


Рисунок 4.58 – Копирование блока custom_name

   - Если в имени есть номер в конце, то при копировании блока номер меняется, определяя формат по количеству «лишних» нулей вначале числа. Для генерации порядкового номера производится поиск всех нумерованных имен, которые начинаются также как исходное, и добавляется номер на **1** больше максимального из них. Если при копировании необходимо сохранять номер в имени, то рекомендуется обеспечить окончание имени нечисловым символом, например, знаком **«_»**.

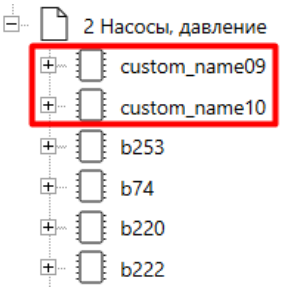


Рисунок 4.59 – Копирование блока custom_name09

Автоматическое изменение имен блоков при копировании можно отключить в окне ***Настройки приложения*** во вкладке ***Копирование***.

4.12 Перенос блоков, страниц, программ

Для переноса блоков, страниц или программ, нужно «схватить» их мышью и перетащить в требуемое место. При отпускании выбрать ***Перенести***.

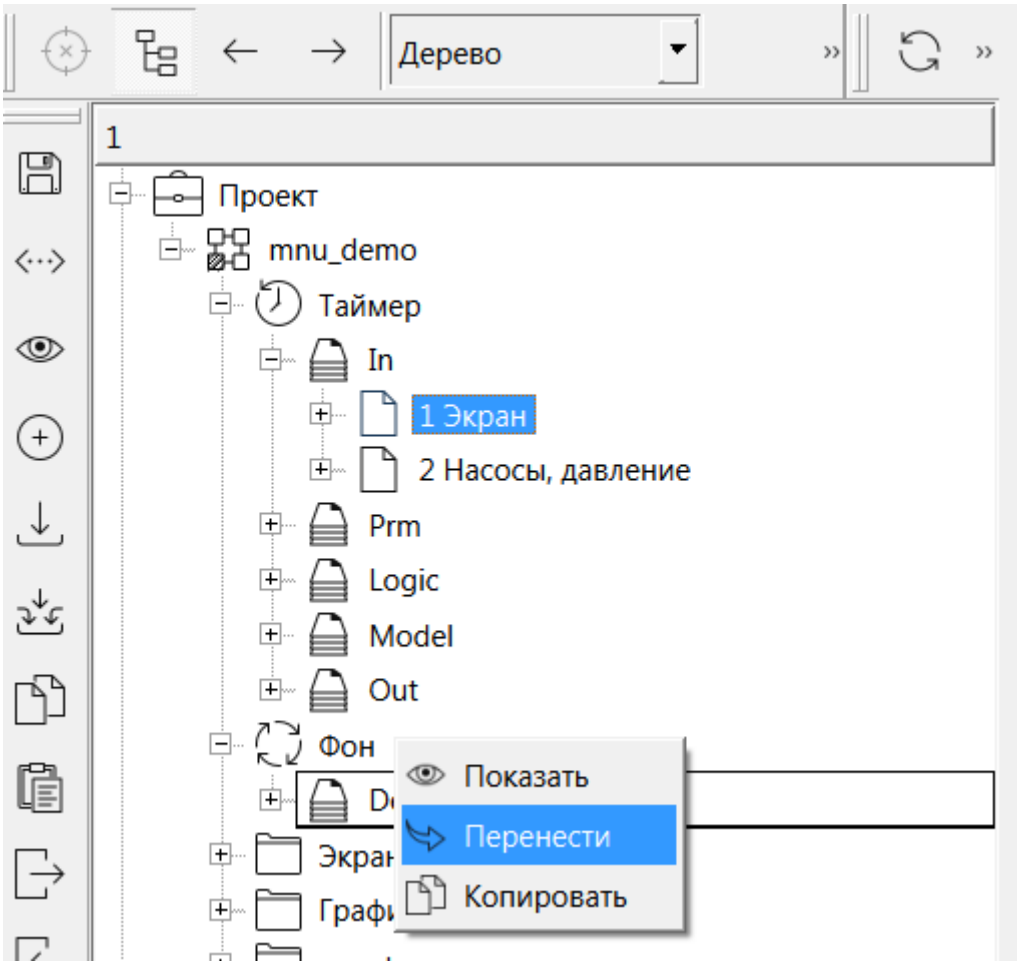


Рисунок 4.60 – Перенос частей проекта

Перенесенный функциональный блок, страница или программа, будут удалены со старого места и вставлены в новое. Расположения компонентов и связи будут сохранены.

Перенос компонентов возможен только в рамках одного проекта. Для переноса блоков, страниц и программ из одного проекта в другой можно использовать команды ***Копировать*** и ***Удалить***.

5 Трансляция

Завершающим этапом создания проекта является ***Трансляция***, созданного алгоритма в исполняемый файл.

Трансляция проводится в два этапа. Графическое представление проекта преобразуется в исходные тексты программы на **C++**. Данные файлы с расширениями **.cpp** и **.hpp**, хранятся в каталоге проекта в папке с именем ***build_имя модуля_ОС***. Они создаются заново каждый раз при трансляции и их редактирование не имеет смысла.

На каждую программу проекта создается несколько файлов:

- ***[имя программы].cpp*** – содержит вызовы алгоритмов функциональных блоков в соответствии с порядком выполнения;
- ***i_[индекс программы].cpp*** – содержит вызовы инициализации функциональных блоков в соответствии с порядком выполнения;
- ***n_[индекс программы].cpp*** и ***e_[индекс программы].cpp*** – содержат описания функциональных блоков.

Индекс функционального блока используется при трансляции как имя объекта, отвечающего за работу этого блока. При этом класс объекта соответствует типу функционального блока и описан в библиотеке функциональных блоков.

Созданные исходные тексты компилируются соответствующим компилятором **C++** в зависимости от выбранной операционной системы. В результате получается исполняемый файл, который необходимо загрузить на контроллер и запустить.

Имя исполняемого файла для операционной системы Linux – ***имя модуля.o***, для Windows – ***имя модуля.exe***.

При трансляции используются следующие свойства модуля:

Таблица 5.1 – Свойства модуля, используемые при трансляции

Свойство	Описание
IP адрес	IP адрес контроллера
ОС	Тип операционной системы для трансляции
Тип процессорной платы	Тип процессорной платы контроллера
Watchdog	Флаг включить/выключить сторожевой таймер (отладку удобнее производить при выключенном сторожевом таймере, при эксплуатации его необходимо включить)
Автозапуск	Флаг включить/выключить автоматический запуск проекта на исполнение при включении контроллера, на контроллер загружается файл ***autostart***
Порт отладчика	Порт OPC UA-сервера в программе, если отличается от стандартного **4840** (вход **prt** блока ***OpcUAServer***)

Для того чтобы выполнить трансляцию в Полигоне можно использовать одну из команд из контекстного меню, открываемого нажатием правой кнопкой мыши на модуле:

- ***Транслировать все*** – трансляция всего модуля;
- ***Перестроить*** – компиляция проекта без изменения исходных текстов. Данную команду следует применять при трансляции неизмененного проекта с новыми библиотеками.

При возникновении сомнений рекомендуется использовать команду ***Транслировать все***. Для трансляции проекта также можно использовать соответствующую кнопку на панели Инструменты.

Если ранее проект транслировался в данную папку, возникнет окно подтверждения трансляции, следует выбрать – ***Да***.

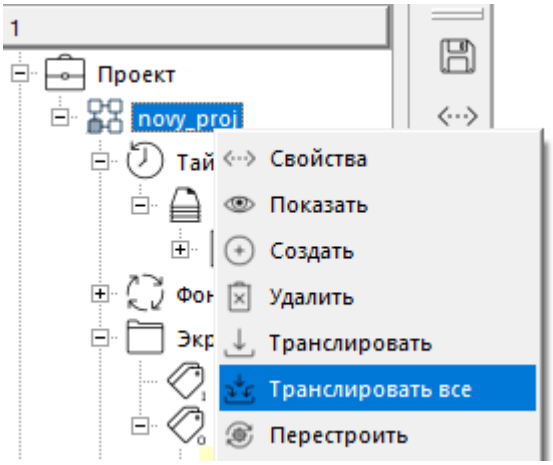


Рисунок 5.1 – Трансляция

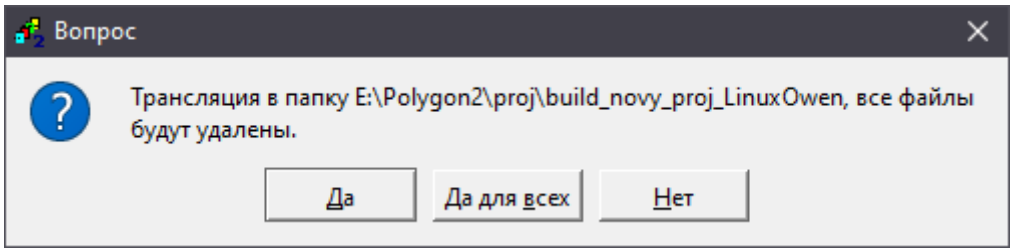


Рисунок 5.2 – Подтверждение трансляции

При трансляции в окне ***Прогресс*** (меню ***Окна/Прогресс***) отображается ход выполнения, при наличии ошибок они указываются с описанием.

По итогу трансляции можно видеть результат – сообщение об успешности трансляции, время выполнения.

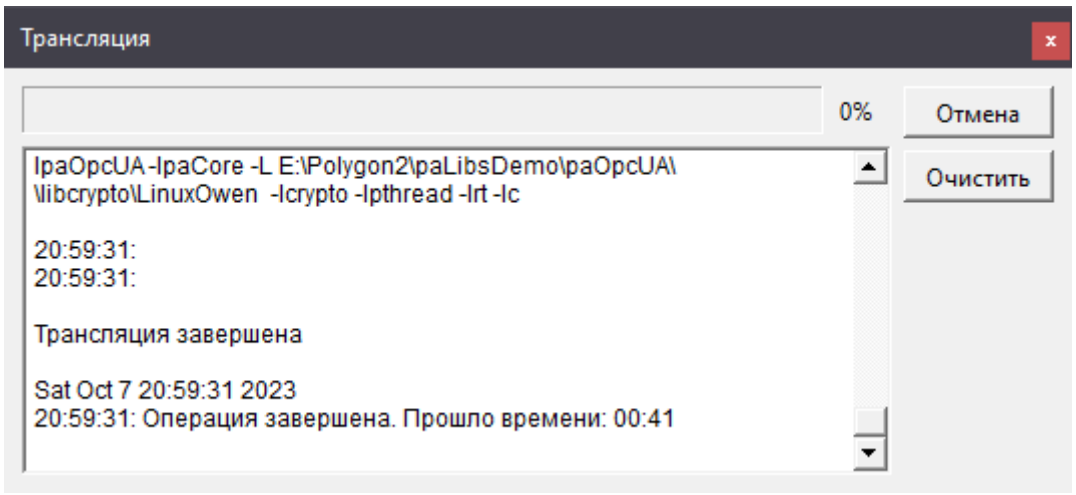


Рисунок 5.3 – Окно прогресс: сообщение об успешной трансляции

Примеры задания свойств модуля для трансляции под ПЛК210 и виртуальный контроллер приведены в разделе 6.

6 Загрузка и запуск проекта

После трансляции проекта его следует загрузить на контроллер.

Для контроллеров ОВЕН загрузку и запуск проекта можно произвести тремя различными способами:

- через ***Панель отладки***;
- через web-интерфейс конфигурации во вкладке ***ПЛК/Приложение***;
- через системное окно ***Контроллер***.

6.1 Свойства модуля для загрузки проекта в ПЛК210

Проект в ПЛК210 загружается через протокол **SSH**.

Для трансляции и загрузки проекта на контроллер следует задать следующие свойства модуля:

Таблица 6.1 – Свойства модуля, используемые при трансляции под ПЛК210

Свойство	Описание	Значение
IP адрес	IP адрес контроллера	К контроллеру можно подключаться из среды через интерфейсы: • **USB Device** – IP адрес **172.16.0.1**; • **Ethernet** – IP адрес по умолчанию **192.168.0.10** (порты Eth 1…3)
Подключаться через	Протокол для подключения к контроллеру	**SSH**
SSH: логин	Логин для подключения к контроллеру	**root**
SSH: пароль	Пароль для подключения к контроллеру	По умолчанию **owen**
ОС	Тип операционной системы для трансляции	**Linux Овен прошивка 3.x**
Тип процессорной платы	Тип процессорной платы контроллера	**Овен ПЛК210**
Watchdog	Флаг включить/выключить сторожевой таймер	На время отладки рекомендуется отключать
Автозапуск	Флаг включить/выключить автоматический запуск проекта на исполнение при включении контроллера, на контроллер загружается файл ***autostart***	-
Порт отладчика	Порт OPC UA-сервера для доступа отладчика Полигон	По умолчанию **4840**

Недостающие в окне свойства можно добавить из выпадающего списка снизу. Работа с окном ***Свойства*** подробно описана в разделе 3.1.3.

При трансляции проекта под ПЛК210 файлы трансляции записываются в папку ***build_имя модуля_LinuxOwen3***. Исполняемый файл программы для контроллера – ***имя модуля.o***.

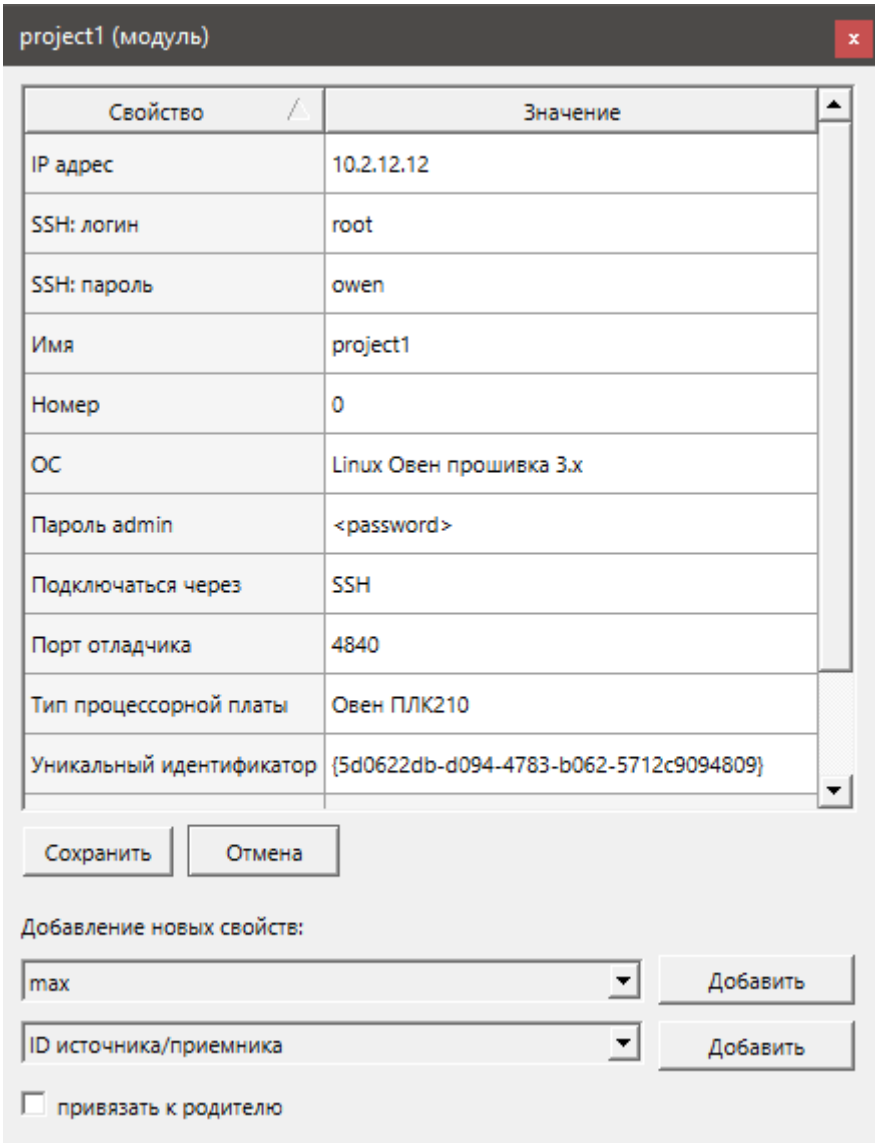


Рисунок 6.1 – Свойства модуля для трансляции и загрузки проекта в ПЛК210

6.2 Панель отладки

Для трансляции, загрузки и запуска проекта на контроллере в среде Полигон предназначена ***Панель отладки***. Панель отладки добавляется через меню ***Окна/Панели инструментов*** – ***Панель отладки***.

Рисунок 6.2 – Панель отладки

Таблица 6.2 – Элементы панели отладки

Элемент	Пиктограмма	Функция
Модуль		Выбор модуля для запуска
Индикатор статуса		Отображает статус запущенного проекта
Перезапустить во временной сессии		Трансляция модуля, загрузка исполняемого файла на контроллер и запуск на «переднем плане» **ВНИМАНИЕ** При закрытии среды разработки выполнение программы на контроллере будет остановлено
Перезапустить для постоянной работы		Трансляция модуля, загрузка исполняемого файла на контроллер и запуск со свойством ***Автозапуск*** **ВНИМАНИЕ** Для запуска программы со свойством ***Автозапуск*** контроллер будет перезагружен
Запустить на виртуальном контроллере		Трансляция модуля и запуск локально на ПК. Отладчик подключается к запущенному модулю, независимо от того, какие выбраны ***IP адрес*** и ***ОС*** для трансляции. Подробнее см. в разделе 6.5

Для запуска проекта на контроллере через панель отладки следует:

1. Установить свойства модуля в соответствии с таблицей 6.1.
2. В выпадающем списке на панели отладки выбрать ***Модуль*** (среди модулей проектов, открытых в данный момент в среде разработки).
3. Выполнить запуск модуля одним из способов:
   - ***Перезапустить во временной сессии*** – будет произведена трансляция модуля, загрузка исполняемого файла на контроллер, запуск его и подключение к нему отладчиком.

ВНИМАНИЕ
При запуске программы во временной сессии ее выполнение на контроллере будет прекращено при закрытии среды разработки.

   - ***Перезапустить для постоянной работы*** – будет произведена трансляция модуля со свойством ***Автозапуск***, загрузка исполняемого файла на контроллер, перезагрузка контроллера и последующее подключение отладчика к запущенной программе.

ВНИМАНИЕ

При загрузке нового проекта на контроллер перезапишется проект только с таким же названием. Ненужные файлы проектов можно удалить через web-конфигуратор контроллера (вкладка ***ПЛК/Приложение*** – ***Удаление…***), через любой файловый менеджер или через консоль.

После запуска программы станет активной кнопка остановки программы.

Если на контроллере уже была запущена программа, то при попытке подключения к контроллеру появится предупреждающее окно с вопросом об ее остановке.

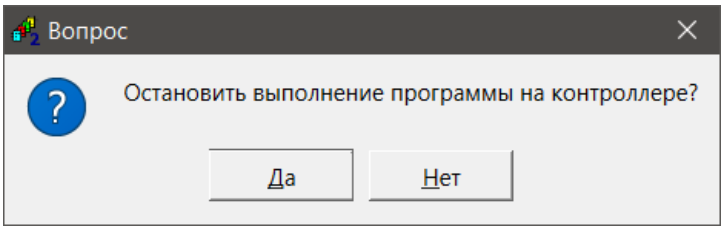


Рисунок 6.3 – Предупреждение о запущенной программе

Информацию о запущенной программе на контроллере можно посмотреть в web-конфигураторе в разделе ***ПЛК/Информация***.

ВНИМАНИЕ

Для обновления информации в разделе ***ПЛК/Информация*** в запущенном проекте должен быть добавлен блок ***OwenHWInfo*** из библиотеки ***paOwenIO***.

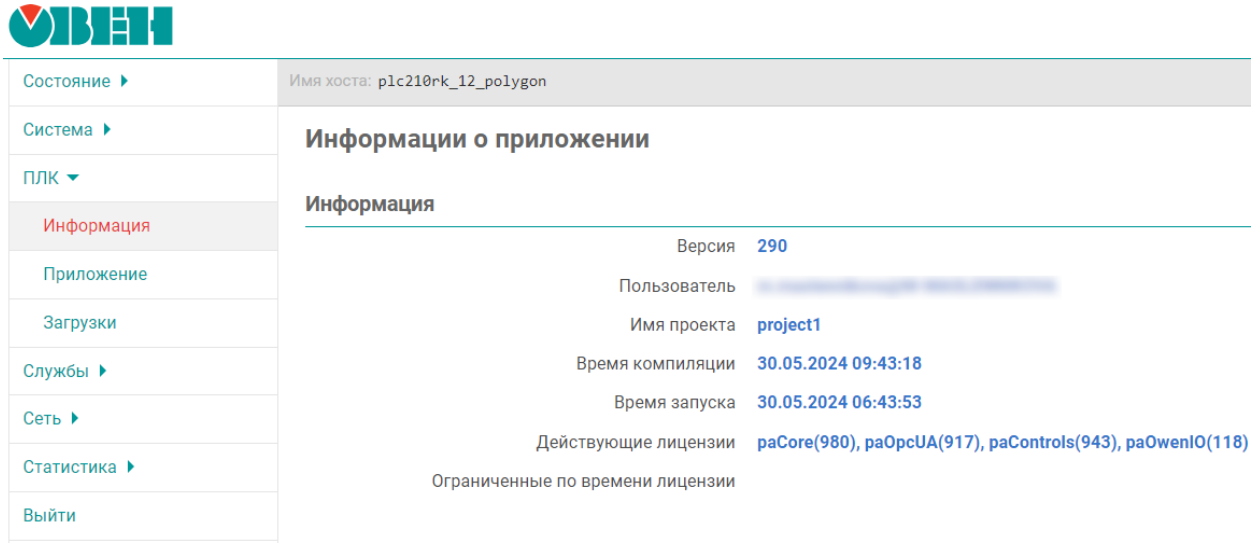


Рисунок 6.4 – Информация о запущенном проекте в web-конфигураторе

6.3 Запуск проекта через web-интерфейс конфигурации

Инструменты для загрузки и запуска программы на контроллере через web-интерфейс конфигурации находятся во вкладке ***ПЛК/Приложение***. Для запуска программы следует:

1. Установить свойства модуля в соответствии с таблицей 6.1 и транслировать модуль – в папке ***build_имя модуля_LinuxOwen3*** на диске появится исполняемый файл программы для контроллера – ***имя модуля.o***.
2. Зайти в web-конфигуратор контроллера и перейти во вкладку ***ПЛК/Приложение***.
3. Нажать ***Загрузка приложения…*** и выбрать файл ***имя модуля.o***.

ВНИМАНИЕ
При загрузке нового проекта на контроллер перезапишется проект только с таким же названием. Ненужные файлы проектов можно удалить через web-конфигуратор контроллера (вкладка ***ПЛК/Приложение*** – ***Удаление…***), через любой файловый менеджер или через консоль.

4. Нажать кнопку ***Запуск…*** напротив ***имя модуля.o***.

ВНИМАНИЕ
Для автоматического запуска приложения при перезагрузке контроллера следует установить для него ***Автозапуск*** (кнопка ***Установить Автозапуск…***). Для корректного останова приложения из web-конфигуратора следует снять для него ***Автозапуск***.

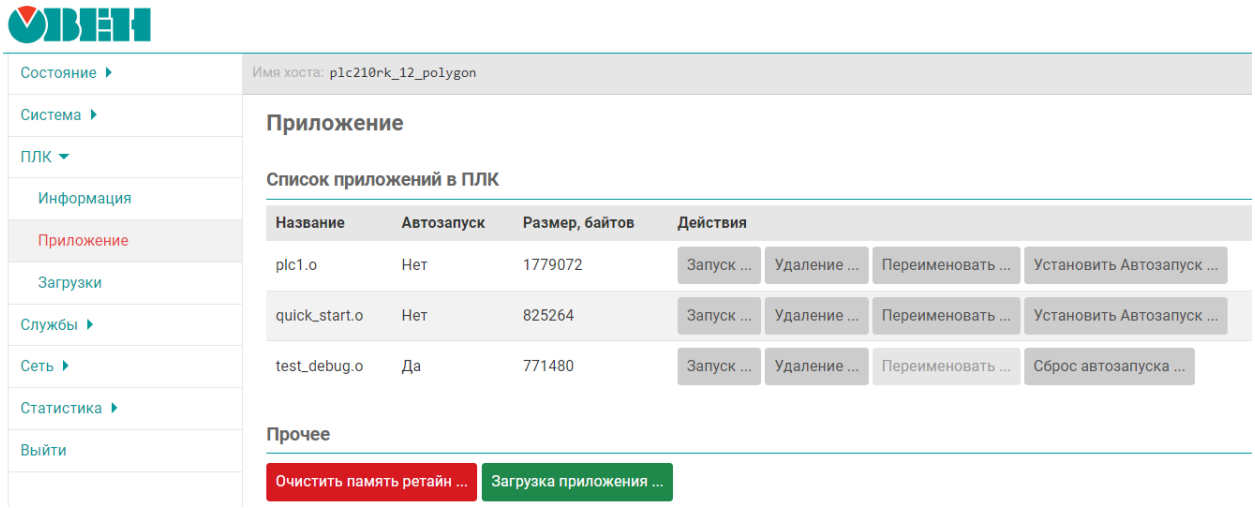


Рисунок 6.5 – Вкладка ПЛК/Приложение

После запуска программы станет активной кнопка остановки программы.

ВНИМАНИЕ
Для корректной остановки программы через web-конфигуратор следует снять ***Автозапуск***.

Информацию о запущенной программе на контроллере можно посмотреть в web-конфигураторе в разделе ***ПЛК/Информация***.

ВНИМАНИЕ
Для обновления информации в разделе ***ПЛК/Информация*** в запущенном проекте должен быть добавлен блок ***OwenHWInfo*** из библиотеки ***paOwenIO***.

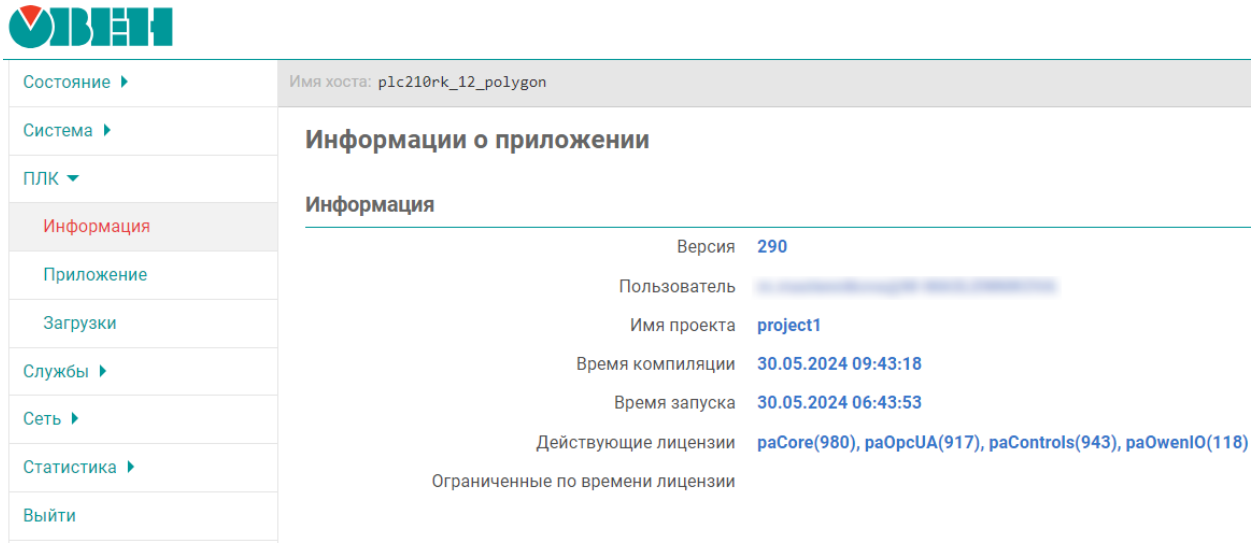


Рисунок 6.6 – Информация о запущенном проекте в web-конфигураторе

6.4 Запуск проекта через окно Контроллер

Для загрузки и запуска проекта на контроллере можно воспользоваться системным окном ***Контроллер***. Для этого следует:

1. Установить свойства модуля в соответствии с таблицей 6.1 и транслировать модуль.
2. Открыть окно ***Контроллер*** через меню ***Окна/Контроллер***.
3. В выпадающем списке выбрать модуль, который будет загружен в контроллер.
4. Кнопкой ***Соединить*** выполнить подключение к контроллеру по **SSH**.

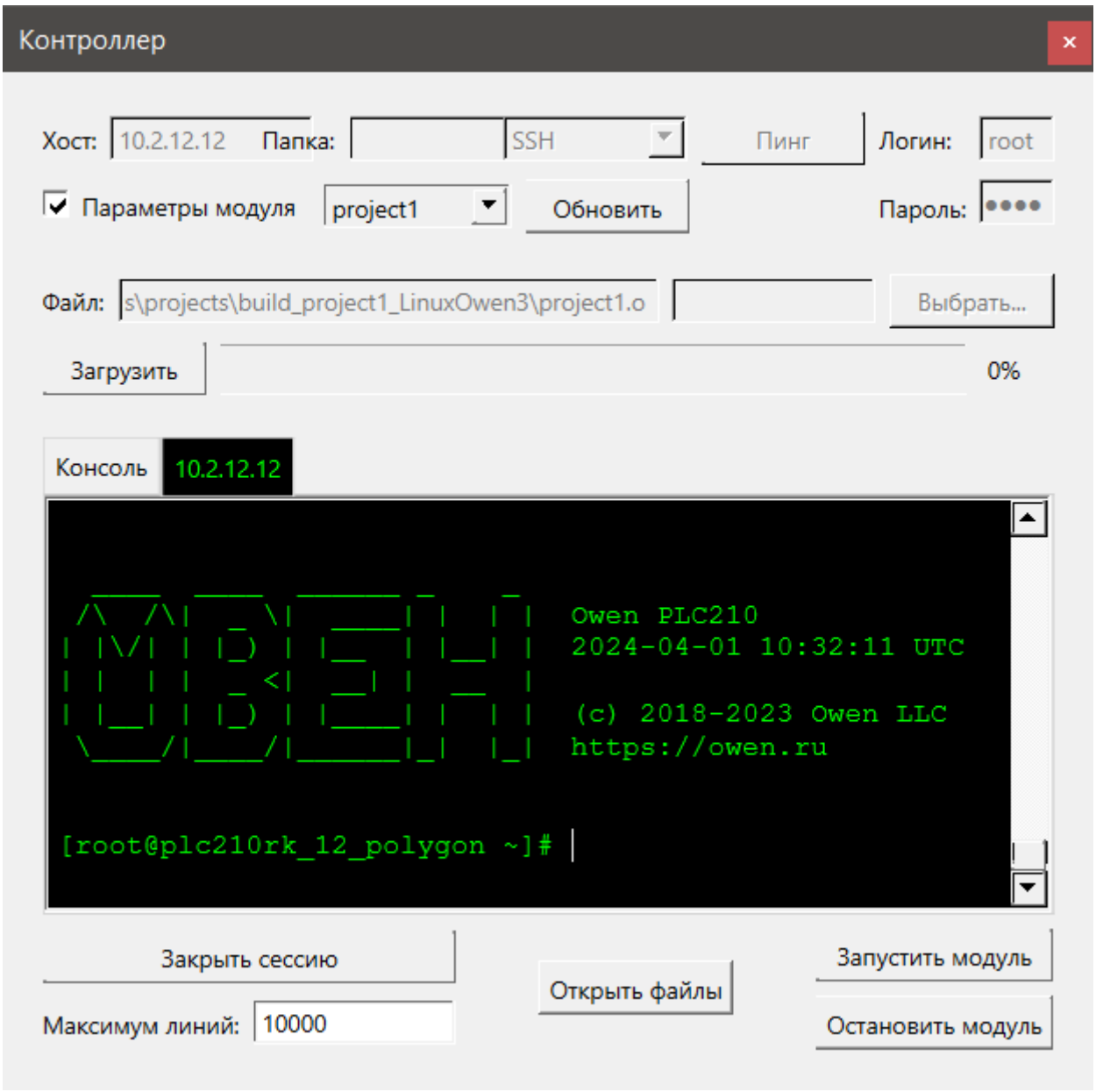


Рисунок 6.7 – Успешное подключение к контроллеру

5. Кнопкой ***Загрузить*** выполнить загрузку проекта в контроллер.

ВНИМАНИЕ
При загрузке нового проекта на контроллер перезапишется проект только с таким же названием. Ненужные файлы проектов можно удалить через web-конфигуратор контроллера (вкладка ***ПЛК/Приложение*** – ***Удаление…***), через любой файловый менеджер или через консоль.

6. Кнопкой ***Запустить модуль*** выполнить запуск загруженного проекта.
7. Подключится к запущенному проекту отладчиком среды через панель Инструменты.

После запуска программы станет активной кнопка остановки программы – ***Остановить модуль***.

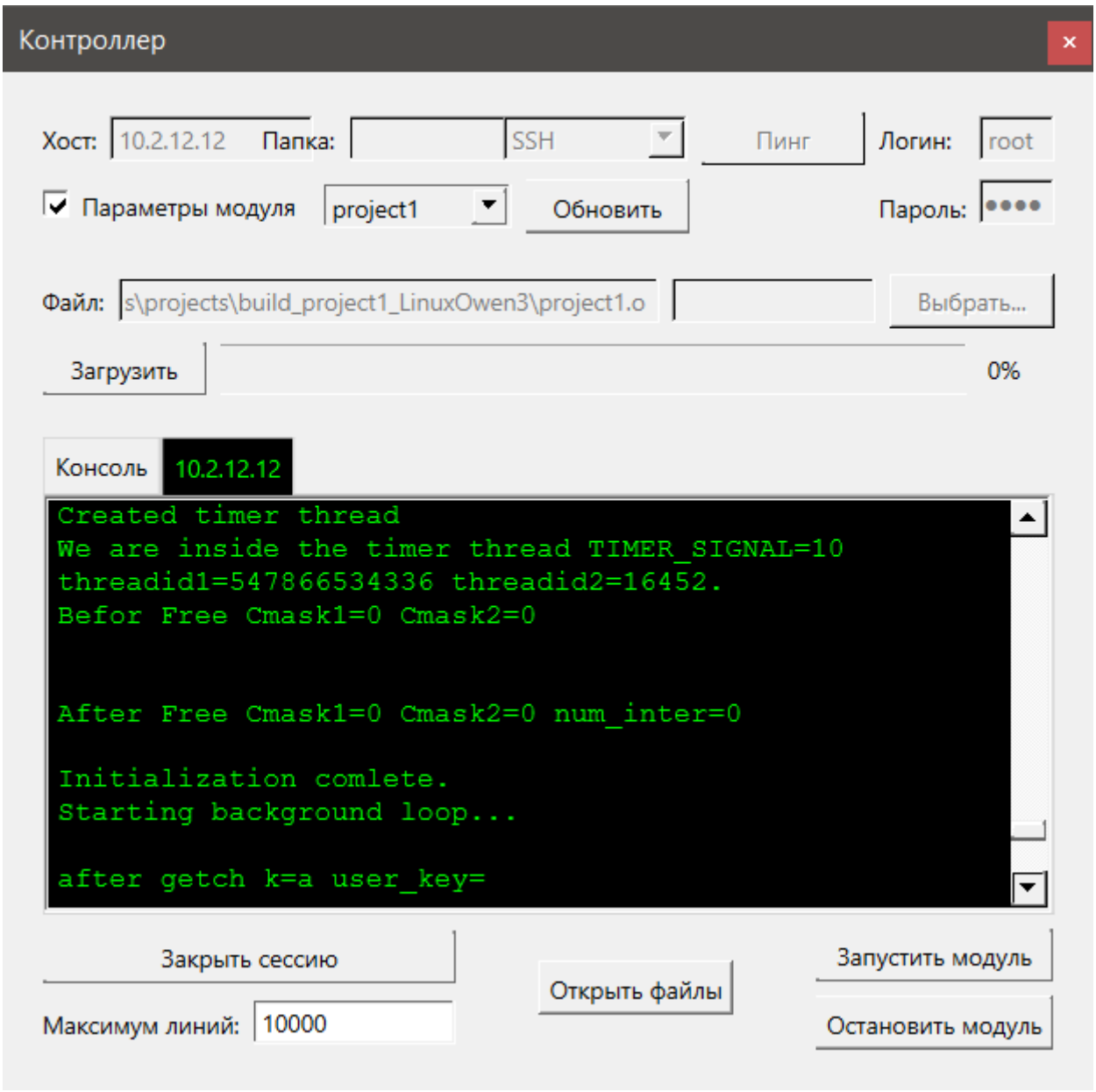


Рисунок 6.8 – Успешный запуск проекта на контроллере

ВНИМАНИЕ
Через окно ***Контроллер*** проект запускается в **foreground**, поэтому при закрытии сессии проект на контроллере перестанет выполняться. Для старта проекта в **background** следует установить ***Автозапуск***, повторно транслировать модуль, загрузить в ПЛК, выполнить перезагрузку ПЛК по питанию или команду ***reboot*** из терминала. Проект запустится автоматически после загрузки ПЛК. Аналогично можно запустить проект для постоянной работы через Панель отладки или web-конфигуратор ПЛК.

Информацию о запущенной программе на контроллере можно посмотреть в web-конфигураторе в разделе ***ПЛК/Информация***.

ВНИМАНИЕ
Для обновления информации в разделе ***ПЛК/Информация*** в запущенном проекте должен быть добавлен блок ***OwenHWInfo*** из библиотеки ***paOwenIO***.

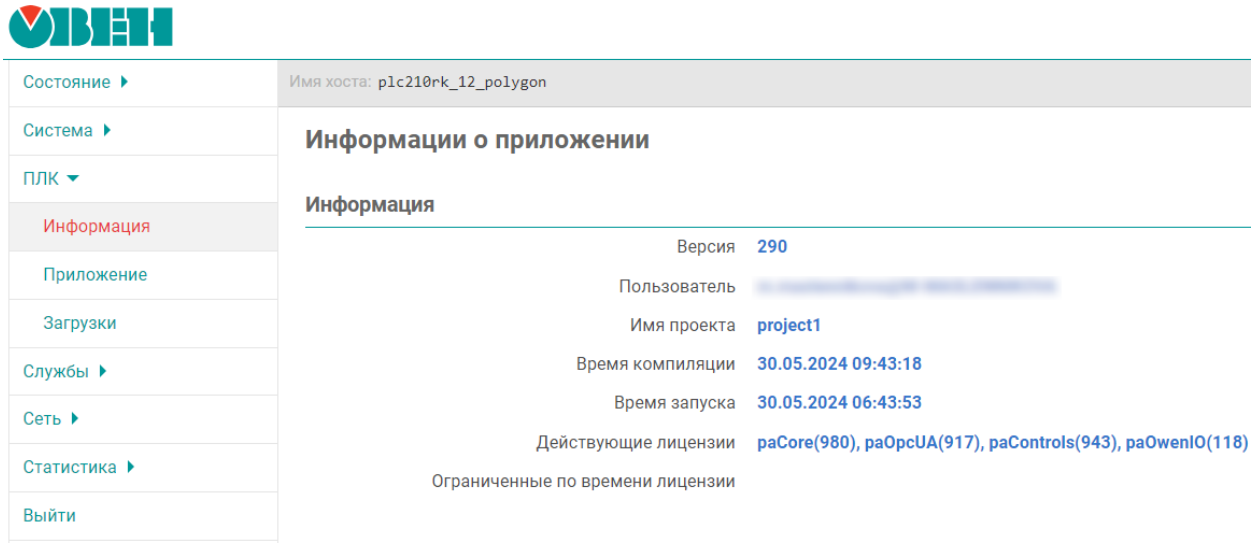


Рисунок 6.9 – Информация о запущенном проекте в web-конфигураторе

6.5 Запуск проекта на виртуальном контроллере

Запустить проект на виртуальном контроллере можно с помощью ***Панели отладки*** (см. раздел 6.2). Для этого следует:

1. Добавить панель отладки через меню ***Окна/Панели инструментов – Панель отладки***.
2. В выпадающем списке на панели отладки выбрать ***Модуль*** (среди модулей проектов, открытых в данный момент в среде разработки).
3. Выполнить запуск модуля на виртуальном контроллере кнопкой ***Запустить на виртуальном контроллере*** – в новом окне запустится приложение виртуального контроллера, к нему подключится отладчик среды.

После запуска программы станет активной кнопка остановки программы.

Запустить виртуальный контроллер можно вручную. Для этого следует:

1. Установить свойства модуля в соответствии с таблицей:

Таблица 6.3 – Свойства модуля, используемые при трансляции на ПК

Свойство	Описание	Значение для ОС Windows	Значение для ОС Linux
IP адрес	Локальный IP адрес	**127.0.0.1**	
Подключаться через	Протокол для подключения к контроллеру	Не используется	**SSH**
ОС	Тип операционной системы для трансляции	**Windows**	**Astra Linux SE1.7**
Тип процессорной платы	Тип процессорной платы	**ПК**	
Порт отладчика	Порт OPC UA-сервера для доступа отладчика Полигон	По умолчанию **4840**	По умолчанию **4850**
Запускать с ключами	Используется для игнорирования блоков из библиотеки ***paOwenIO***	**-I**	

Недостающие свойства можно добавить из выпадающего списка добавления новых свойств внизу окна. Работа с окном ***Свойства*** подробно описана в разделе 3.1.3.

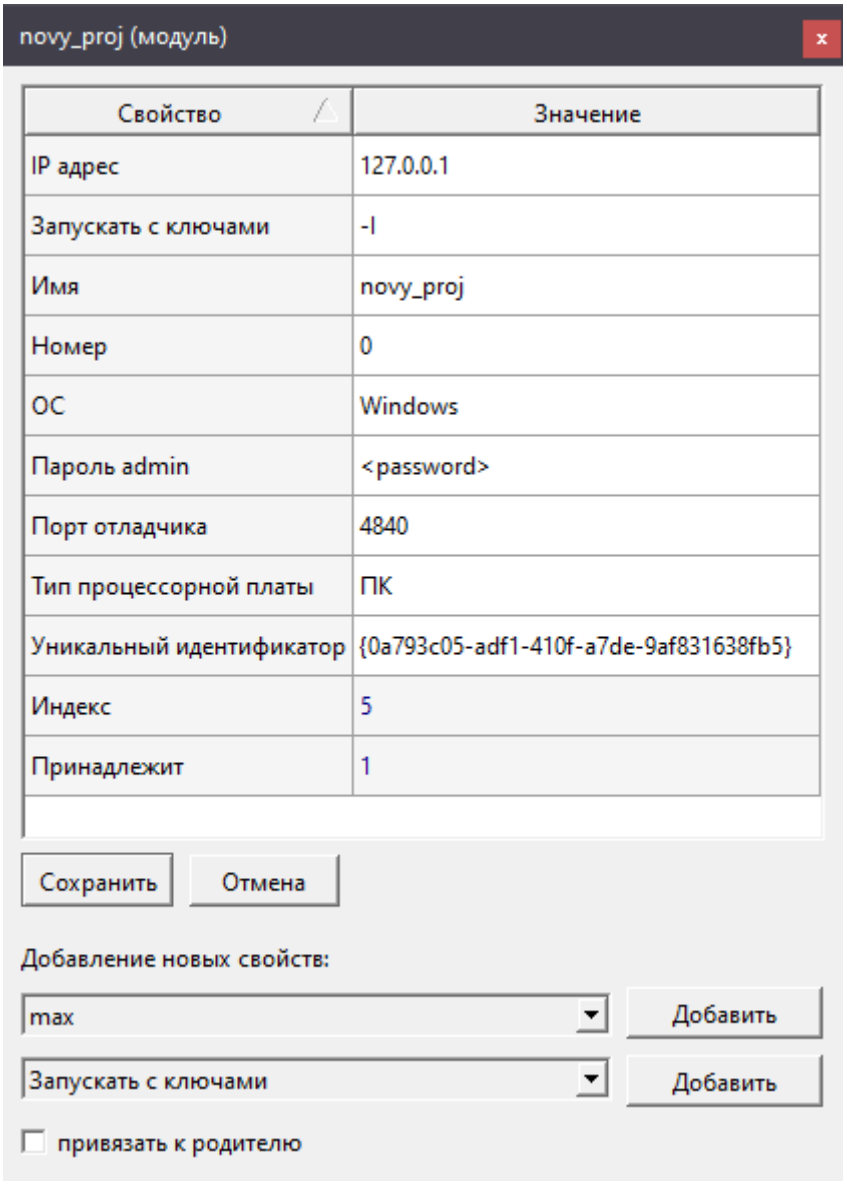


Рисунок 6.10 – Свойства модуля для трансляции под ОС Windows

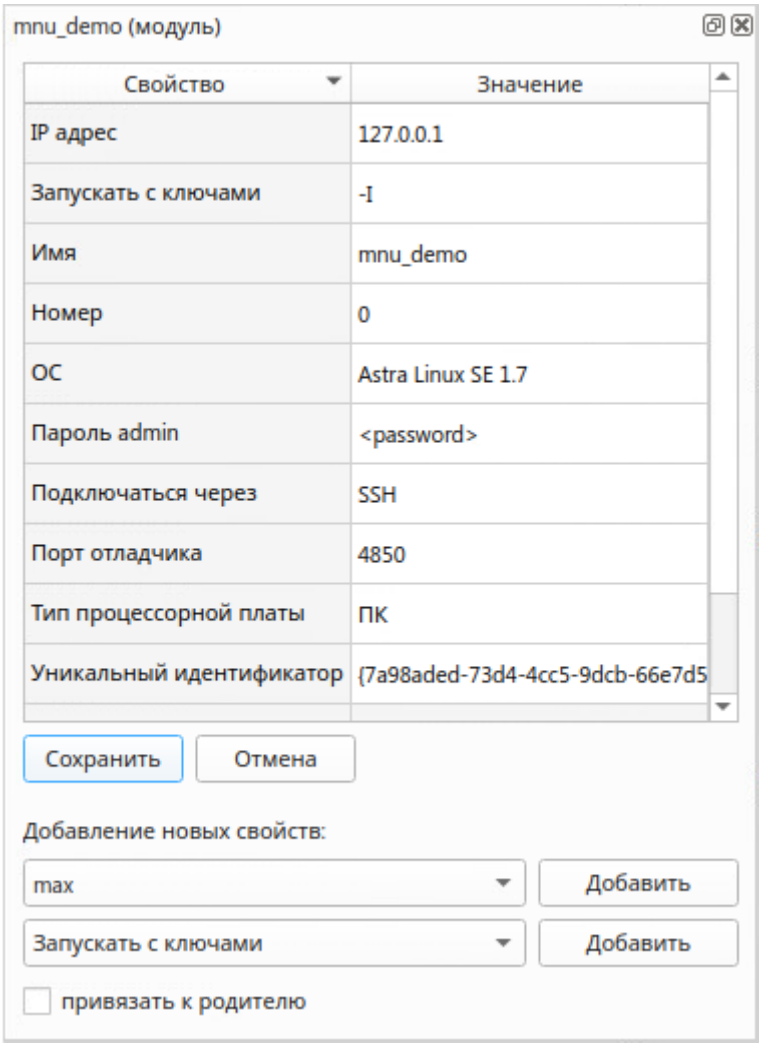


Рисунок 6.11 – Свойства модуля для трансляции под ОС Linux

2. Транслировать модуль.
3. Запустить модуль:
   - для ПК с ОС Windows – запустить исполняемый файл ***имя модуля.exe*** из папки ***build_имя модуля_WINmingw***;
   - для ПК с ОС Linux – запустить исполняемый файл ***имя модуля.o из папки build_имя модуля_LinuxAstra17***.
4. Подключится к запущенному приложению отладчиком среды через панель Инструменты.

ВНИМАНИЕ
Ограничение на работу виртуального контроллера составляет **1 час**. Виртуальный контроллер работает неограниченное время, если у пользователя есть USB-ключ.

7 Отладка проекта

Для отладки программы в Полигоне существует возможность просмотра текущих значений на входах/выходах функциональных блоков. Для этого необходимо запустить программу на контроллере и выбрать команду ***Отладчик*** на панели Инструменты.

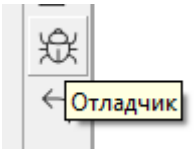


Рисунок 7.1 – Отладчик

При первом подключении отладчиком к запущенному проекту запрашивается пароль, указанный при создании проекта.

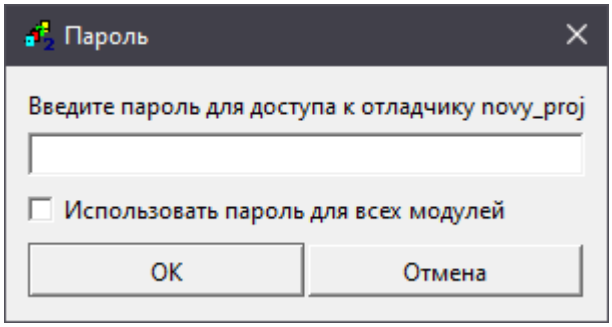


Рисунок 7.2 – Окно ввода пароля для доступа к отладчику

В режиме отладки Полигон запрашивает у контроллера значения входов и выходов блоков, расположенных на страницах, открытых в рабочих окнах, и отображает полученные значения на экране.

Обмен данными осуществляется по протоколу **OPC UA**, поэтому в проекте обязательно должен быть добавлен блок ***OpcUAServer*** из библиотеки ***paOpcUA***. На входе блока **ip** – IP адрес следует прописать IP адрес интерфейса контроллера, по которому следует выполнять подключение отладчика.

При выборе шаблона модуля с отладчиком при создании нового проекта в месте работы ***Фон*** автоматически создается программа ***Debug***, на странице которой добавлен блок OPC UA-сервера. На входы блока **ip** – IP адрес и **prt** – локальный порт контроллера прописываются соответствующие свойства модуля в виде SQL-запросов.

Запрос IP адреса:

```
"<sql>SELECT value FROM blocks_prop WHERE indx=:module AND type="prop_ip"</sql>"
```

Запрос номера порта:

```
<sql>SELECT value FROM blocks_prop WHERE indx=:module AND type="prop_debug_port"</sql>
```

Подробнее реализация протокола **OPC UA** в среде Полигон описана в документе Обмен с верхним уровнем. paOpcUA.

Остановить выполнение проекта на контроллере можно через ***Панель отладки*** или, если проект запускался через окно ***Контроллер***, кнопкой ***Остановить модуль***. Также выполнение проекта можно остановить через контекстное меню модуля в дереве проекта, нажав ***Остановить***.

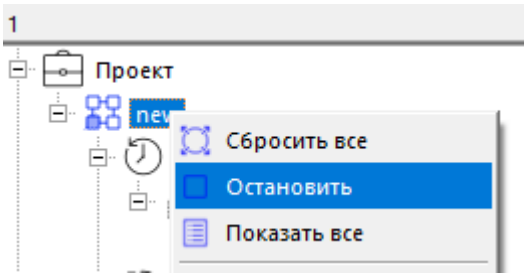


Рисунок 7.3 – Остановка выполнения проекта

Для отключения отладчика без остановки программы необходимо повторно нажать на кнопку ***Отладчик*** на панели Инструменты.

7.1 Подмена значений на входах/выходах

В режиме отладки текущие значения входов/выходов блока отображаются синим цветом. Справа от названия типа блока зеленым цветом отображается время выполнения блока в **микросекундах**.

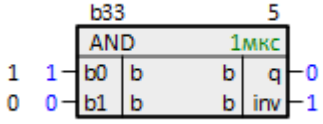


Рисунок 7.4 – ФБ в режиме отладки

Текущие значения на входах блоков можно подменить. Для этого необходимо дважды щелкнуть мышкой на текущем значении входа и задать новое значение. Подмененные значения отображаются красным цветом.

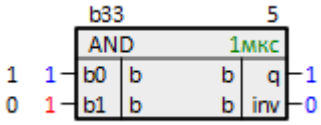


Рисунок 7.5 – Подмена значения в режиме отладки

Заданная константа действует только в течение данного запуска программы. Чтобы вернуть значение на входе к заданному в программе надо выбрать команду ***Сбросить*** в контекстном меню входа.

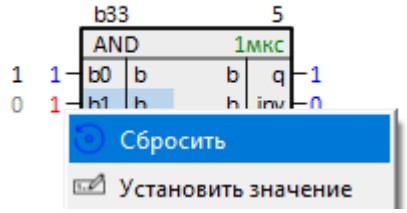


Рисунок 7.6 – Сброс подмененного значения

Подмена значения на входе игнорирует константы и связи.

При подмене значения на выходе возможны следующие варианты:

1. Подмена выключает блок. По умолчанию, при двойном клике на текущем значении, срабатывает этот вариант.
2. Подмена не выключает блок, а подменяет значения на входах, с которыми связан данный выход.

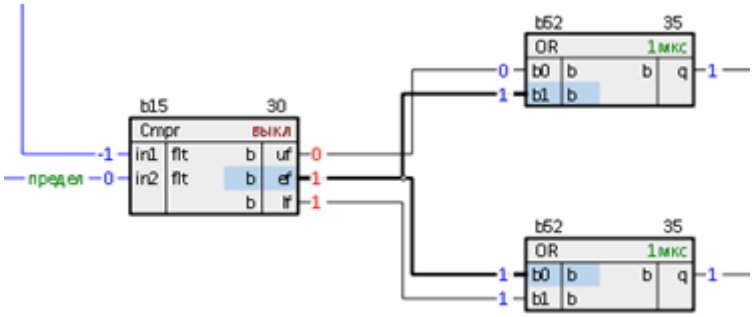


Рисунок 7.7 – Подмена выхода с выключением блока

Для подмены значения выхода без выключения блока следует:

1. Нажать на выходе ПКМ.
2. Выбрать ***Установить значение***.
3. Выбрать ***не выключать блок*** и установить значение на выходе – введенное значение установится на всех входах, связанных с данным выходом. Выход при этом продолжит меняться в соответствии с алгоритмом блока и подсветится оранжевым цветом.

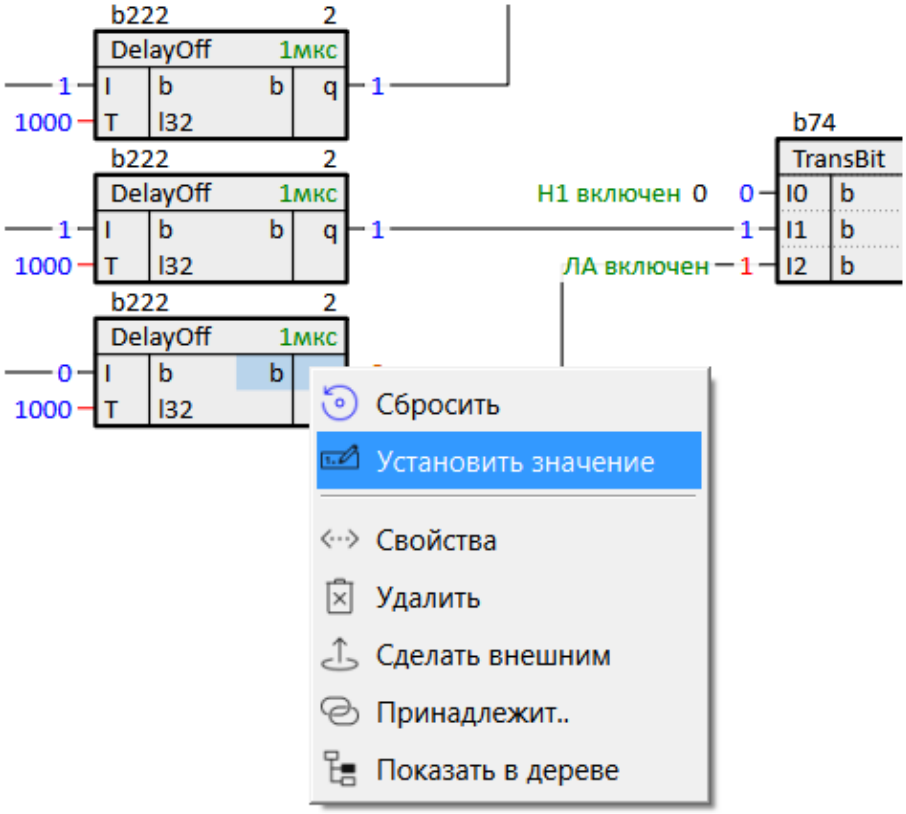


Рисунок 7.8 – Установка значения выхода

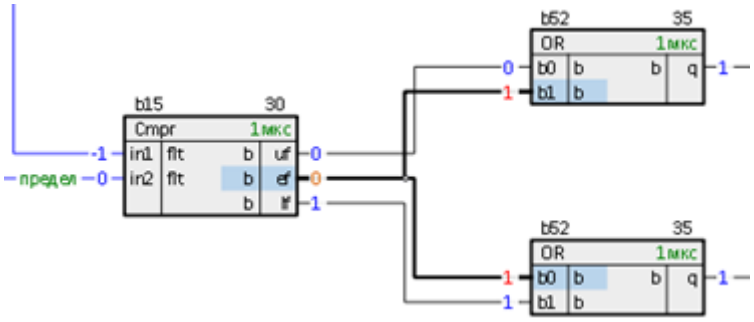


Рисунок 7.9 – Подмена выхода без выключения блока

Сбросить значение можно нажав на выходе ПКМ и выбрав ***Сбросить***.

Во время отладки можно получить список всех подмененных значений командой ***Показать все***, а также отменить все подмены командой ***Сбросить все***. Эти команды появляются при щелчке правой кнопкой мыши в дереве проекта на узле модуля.

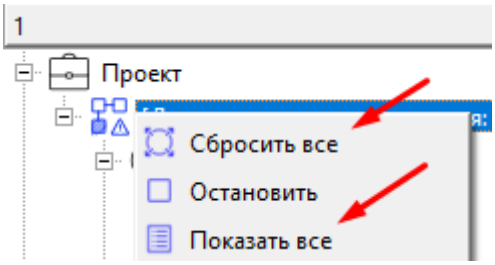


Рисунок 7.10 – Получение списка и сброс подмененных значений

При нажатии ***Показать все*** отображается таблица подмененных значений, о работе с представлением таблица подробнее см. в разделе 3.1.1.7.

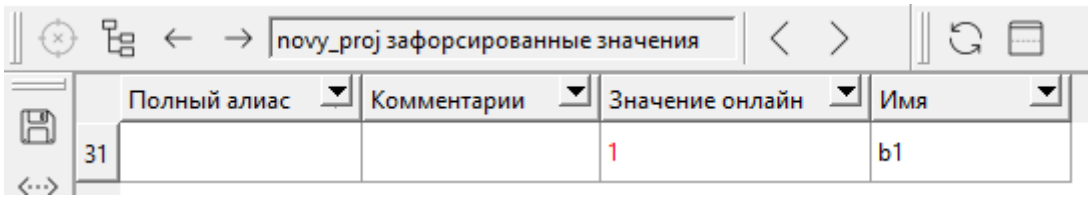


Рисунок 7.11 – Таблица подмененных (зафорсированных) значений

7.2 График

График позволяет просматривать изменения значений на входах и выходах функциональных блоков в графическом представлении в режиме отладки.

Для добавления входа или выхода в таблицу над графиком нужно перетащить его со страницы (при этом его ***Модуль*** автоматически станет текущим для графика). Если текущим будет ***Модуль***, то в таблицу с помощью перетаскивания по очереди можно добавлять входы/выходы этого модуля из дерева.

Если текущим будет ***Раздел***, то в таблицу автоматически добавятся входы/выходы из раздела. При этом если добавить новый вход или выход в таблицу, он автоматически добавится в ***Раздел***.

Подробнее об элементах графика см. в разделе 3.1.1.8.

Для просмотра значений входов/выходов нужно загрузить исполняемый файл на контроллер и нажать кнопку ***Запустить***. Для остановки – ***Остановить***.

График подключается к OPC UA-серверу контроллера в качестве клиента (аналогично отладчику) с добавленными в него данными.

ПРИМЕЧАНИЕ
Для корректной работы графика необходимо синхронизировать системное время контроллера с браузером через web-конфигуратор (***Система/Время/Системное время*** – ***Синхронизировать с браузером***).

7.3 Экран отладчика

В режиме отладки текущие значения можно просматривать и изменять на ***Экране отладчика***.

Входы/выходы, которые нужно отобразить на экране, необходимо добавить в соответствующий ***Раздел***. Их также можно перетащить на экран из дерева или страницы, тогда они автоматически добавятся в текущий раздел. При этом входу или выходу автоматически присваивается класс отображения следующим образом:

– если это логический вход – ***кнопка***, иначе – ***слайдер***;
– если это логический выход – ***лампа***, иначе – ***значение***.

Класс отображения можно поменять в окне свойств.

Подробнее о работе с экраном отладчика и свойствах элементов см. в разделе 3.1.1.9.

В режиме работы отладчика для изменения значения на входе, нужно дважды щелкнуть на соответствующем элементе и отредактировать значение. Это возможно для графических элементов ***значение***, ***слайдер*** и ***шкала***.

7.4 Информация о запущенном проекте

В режиме отладки можно получить различную информацию о запущенной программе.

При наведении курсора на запущенный модуль в дереве всплывает тултип с информацией о проекте: идентификатор модуля, версия проекта, дата трансляции, пользователь и имя компьютера, с которого была произведена трансляция проекта.

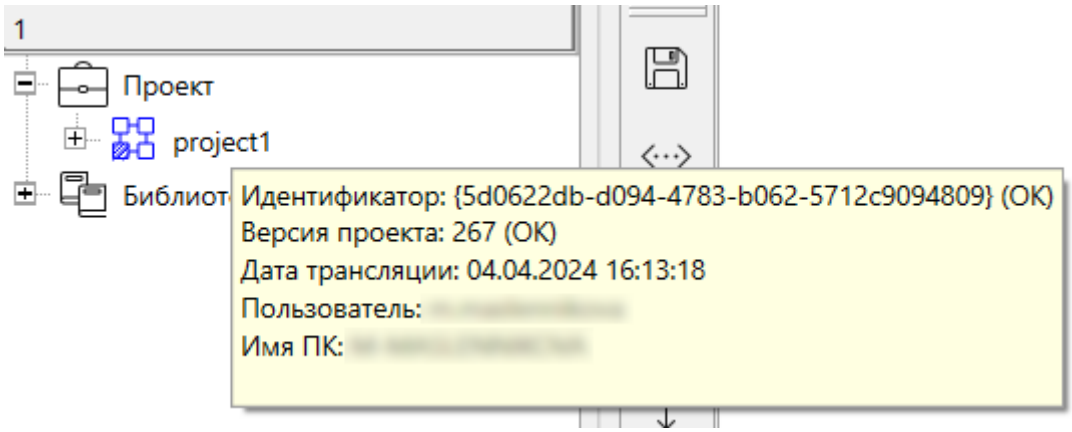


Рисунок 7.12 – Тултип с информацией о модуле

Информацию о запущенном приложении на контроллере также можно посмотреть в web-конфигураторе в разделе ***ПЛК/Информация***.

ВНИМАНИЕ
Для обновления информации в разделе ***ПЛК/Информация*** в запущенном проекте должен быть добавлен блок ***OwenHWInfo*** из библиотеки ***paOwenIO***.

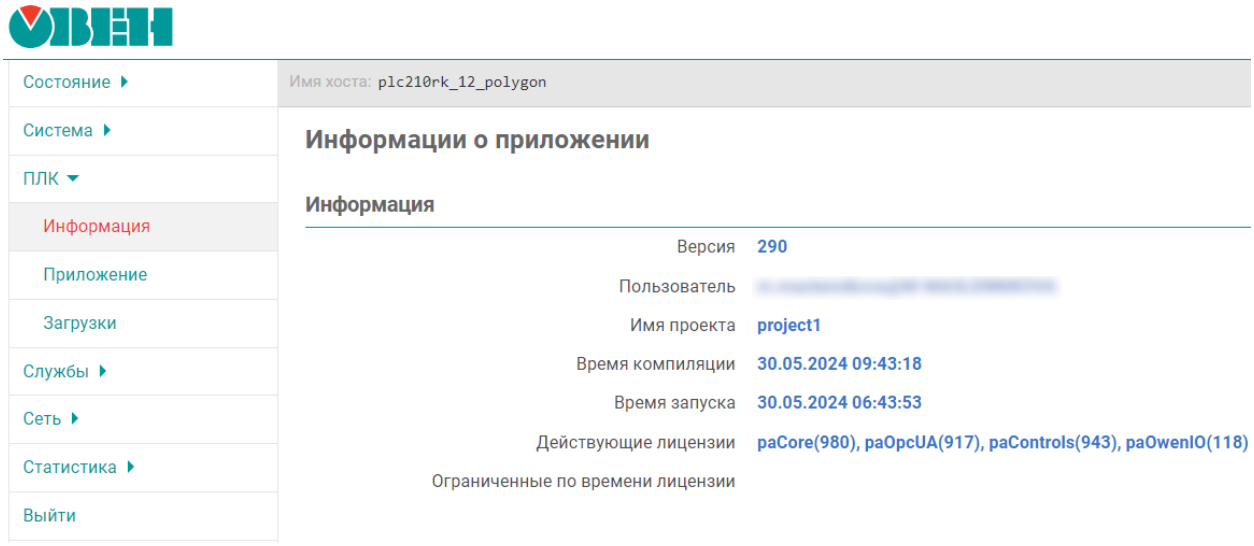


Рисунок 7.13 – Информация о запущенном проекте в web-конфигураторе

При клике правой кнопкой мышки на запущенном модуле в дереве и выборе в открывшемся меню ***Показать все*** в окне трансляции (системное окно ***Прогресс***) можно увидеть данные по версиям библиотек.

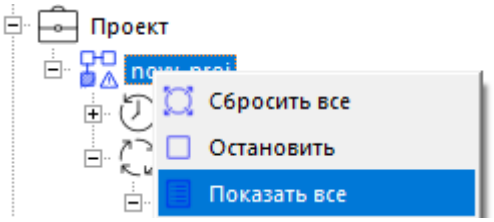


Рисунок 7.14 – Показать все

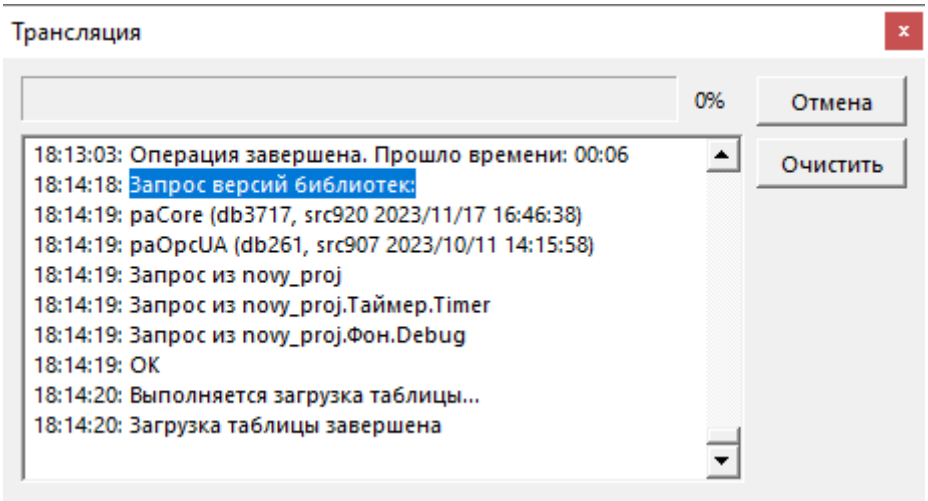


Рисунок 7.15 – Информация о версиях библиотек

Также при трансляции в папке ***build_имя модуля_ОС*** создается файл ***versions.txt***, в котором показаны идентификатор проекта, название модуля, дата и время трансляции, версии библиотек.

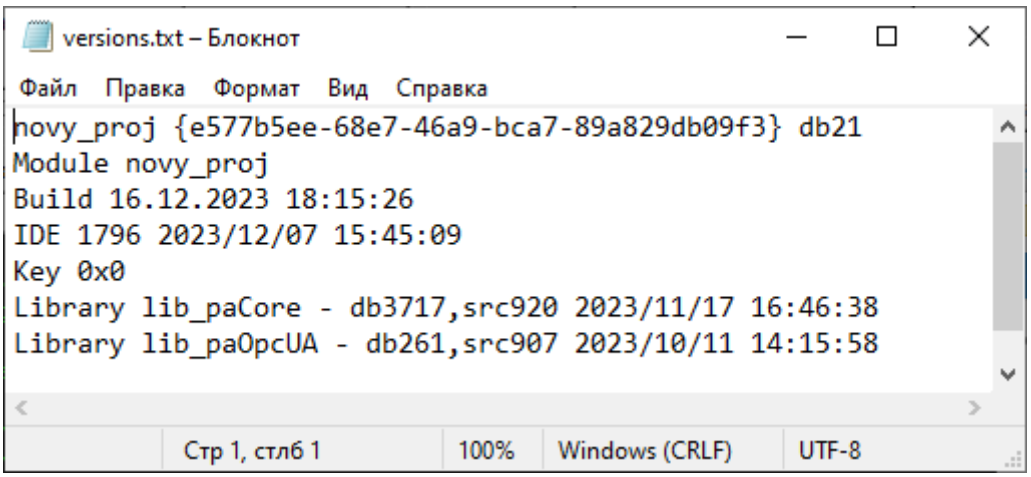


Рисунок 7.16 – Файл versions.txt

Отслеживать время выполнения мест работы и программ модуля можно при добавлении в свойства модуля свойства ***Показывать время выполнения***. После трансляции и запуска проекта в дереве будут отображаться времена выполнения программ модуля.

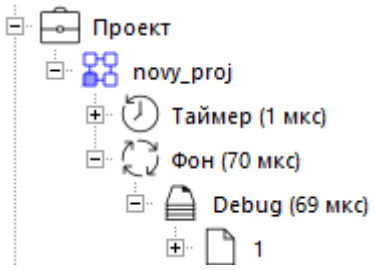


Рисунок 7.17 – Отображение времен выполнения программ модуля

Получить время выполнения мест работы в программе также можно с помощью ФБ SysInfo.

8 Возможности пользовательской конфигурации

8.1 Создание составного блока

Часть алгоритма, повторяющаяся в проекте несколько раз, может быть выделена в новый ***Составной функциональный блок***. Обычно это упрощает отладку и модификацию этой части алгоритма, а также уменьшает время трансляции проекта. Составной блок при трансляции превращается в класс. Составные блоки можно создавать как в пользовательских проектах, так и в пользовательских библиотеках.

О защите пользовательского составного блока см. в разделе 9.2.

Для того чтобы создать составной функциональный блок в проекте следует создать ***Группу типов*** в узле ***Библиотеки*** в дереве проекта.

ВНИМАНИЕ
Имя ***Группы типов*** должно содержать только латинские буквы, цифры и **«_»**. Если оставить поле ввода имени пустым – оно будет назначено автоматически по шаблону ***b<индекс>***.

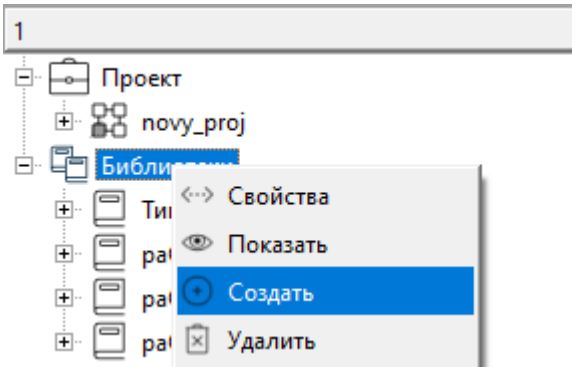


Рисунок 8.1 – Узел Библиотеки в дереве проекта

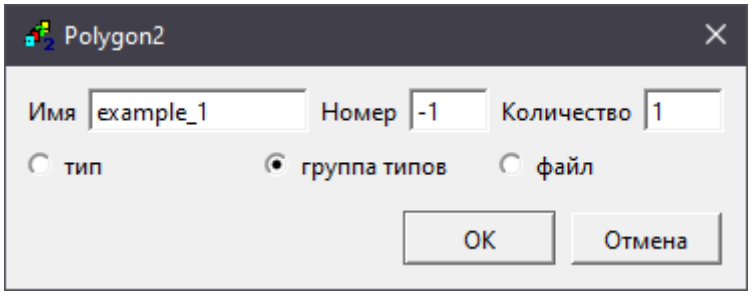


Рисунок 8.2 – Создание Группы типов в проекте

Для того чтобы создать функциональный блок в пользовательской библиотеке следует открыть необходимую библиотеку в представлении ***Дерево*** и создать в ней ***Группу типов*** с любым именем.

О создании пользовательской библиотеки см. в разделе 8.2.

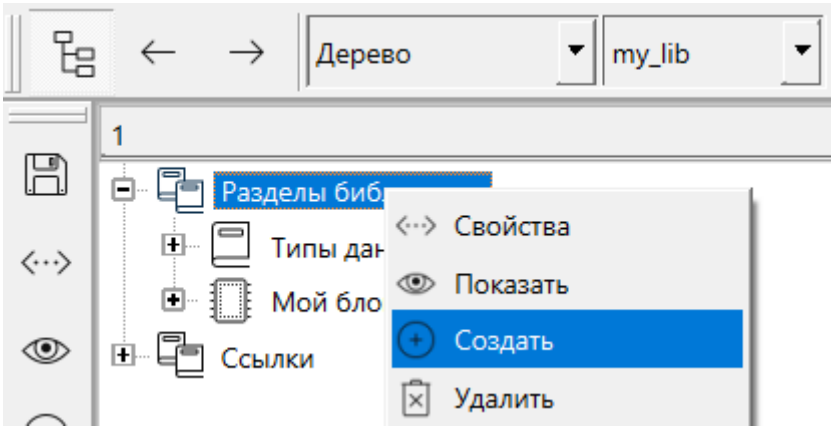


Рисунок 8.3 – Библиотека в представлении Дерево

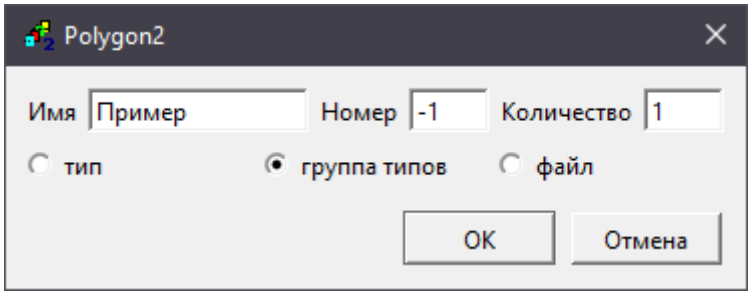


Рисунок 8.4 – Создание Группы типов в библиотеке

Теперь в новой группе типов следует создать ***Тип***. При создании типа указывается ***Имя типа*** – имя класса, который будет создан при трансляции.

ВНИМАНИЕ

Имя типа должно быть уникально в проекте и содержать только латинские буквы, цифры и **«_»**. Если оставить поле ввода имени пустым – оно будет назначено автоматически по шаблону ***b<индекс>***.

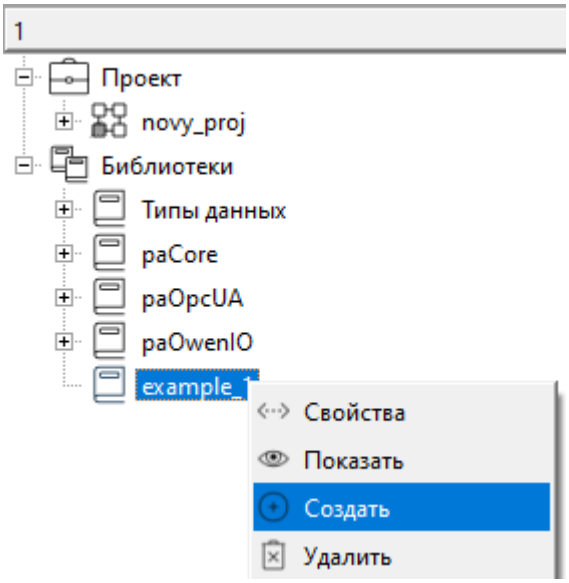


Рисунок 8.5 – Группа типов в дереве проекта

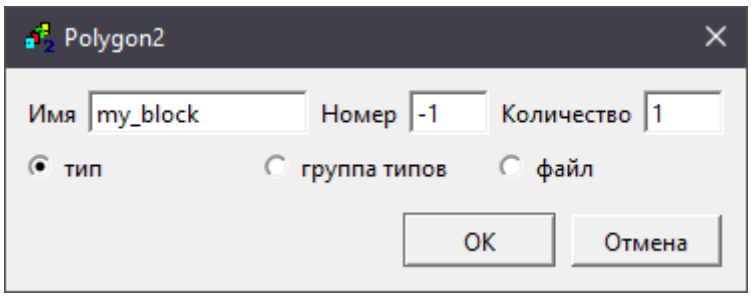


Рисунок 8.6 – Создание Типа

На данном этапе был создан класс (***Тип***) нового функционального блока. Приступим к реализации алгоритма, который будет выполняться экземплярами класса функционального блока в проекте.

8.1.1 Создание составного блока из функциональных блоков

Рассмотрим создание составного функционального блока с алгоритмом, составленным из функциональных блоков.

В качестве примера реализуем блок, возвращающий значение линейной функции **y = kx + b**. Для этого следует:

1. Создать Тип.
2. Создать ***Страницу*** в ***Типе***.

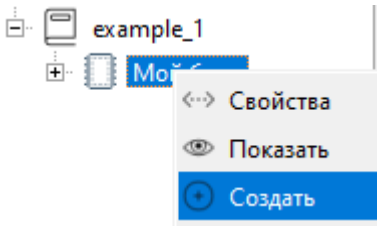


Рисунок 8.7 – Тип в дереве проекта

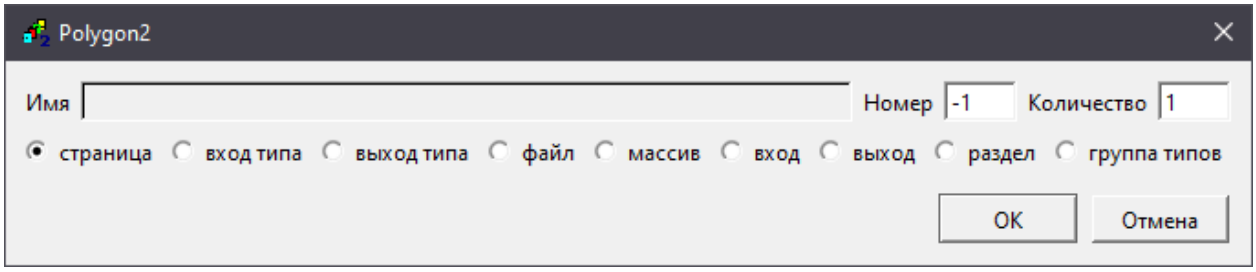


Рисунок 8.8 – Создание Страницы в Типе

На страницах ***Типа*** реализуется алгоритм блока.

Страницы в ***Тип*** можно скопировать из проекта. При этом, если необходимо чтобы связи между блоками на разных страницах сохранились, копировать страницы следует вместе, выделив их в дереве с зажатым **Shift**.

3. Создать на новой странице блоки ***Mul*** и ***Add*** из библиотеки ***paCore*** и соединить их как показано на рисунке ниже.

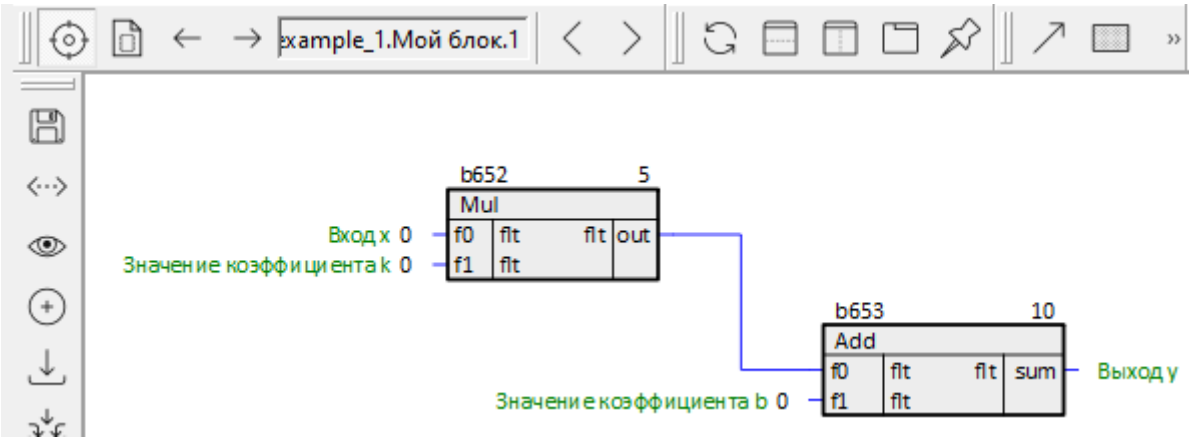


Рисунок 8.9 – Алгоритм работы составного блока

4. Назначить внутренние входы/выходы блоков как внешние входы/выходы составного блока.

Для этого следует в контекстном меню нужного входа/выхода выбрать команду ***Сделать внешним*** – вход/выход станет одновременно принадлежать и внутреннему блоку, и ***Типу***. Входы/выходы внутренних блоков, назначенные внешними, подкрасятся желтым и на ***Странице,*** и в ***Дереве***.

ПРИМЕЧАНИЕ
Рекомендуется входы/выходы составного блока, которые необходимо назначить внешними, выводить на терминальные блоки (типа Trans). Терминальные блоки повторяют значения своих входов на выходах. Таким образом, если возникнет необходимость изменить внутренность составного блока, когда он уже будет добавлен в проект, не потребуется повторно выполнять проведение связей.

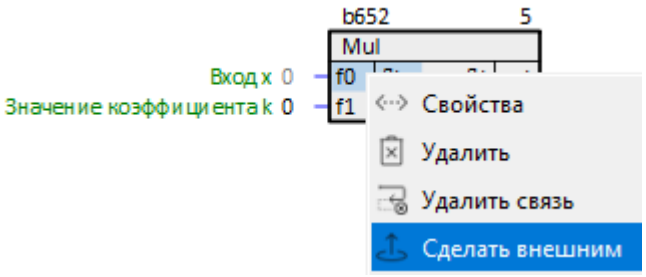


Рисунок 8.10 – Назначение внешними входов/выходов составного блока

Рисунок 8.11 – Внешние входы/выходы составного блока

5. Транслировать проект для проверки корректности созданного типа. После сообщения об успешной трансляции – ***Трансляция завершена*** экземпляры нового типа можно создавать на страницах проекта.

Создать экземпляр составного блока в проекте можно двумя способами:

1. Перетащить блок из ***Дерева*** проекта или библиотеки на ***Страницу*** проекта и выбрать ***Создать*** в контекстном меню.

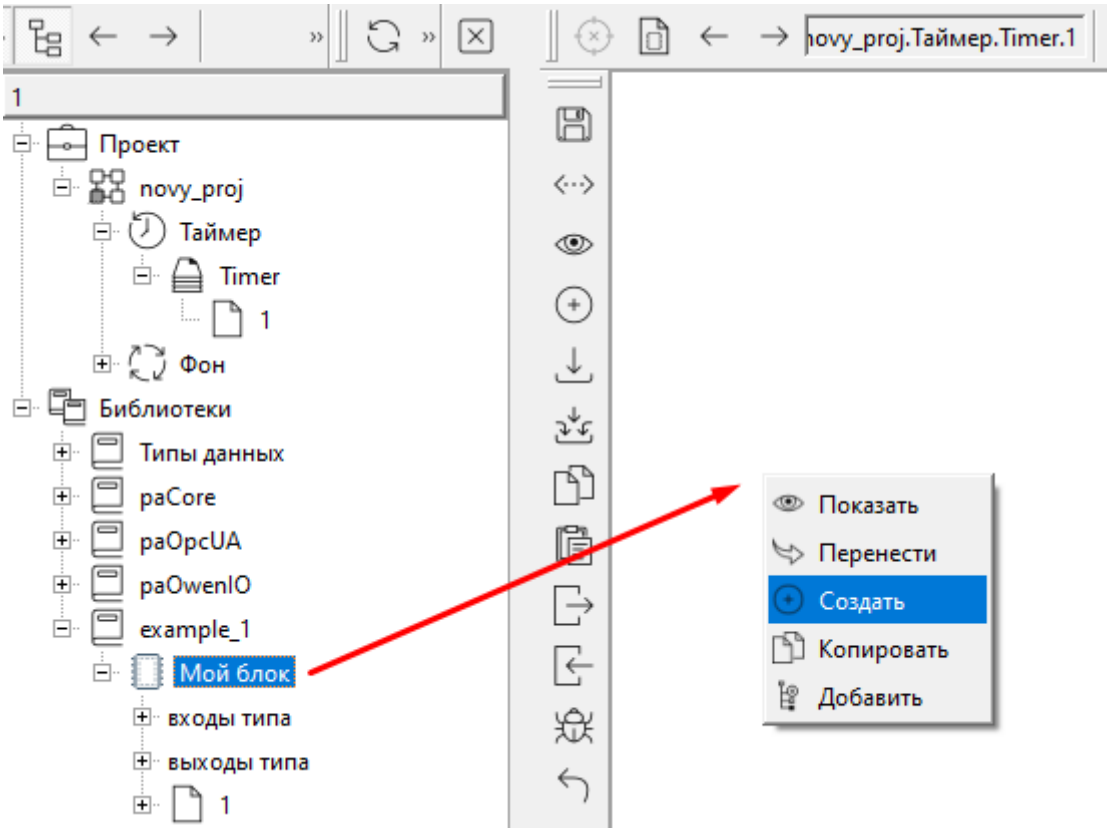


Рисунок 8.12 – Добавление пользовательского составного блока в проект

2. На пустом месте ***Страницы*** щелкнуть ПКМ – ***Создать*** и выбрать необходимый блок в разделе ***Библиотека проекта***, нажать **ОК**.

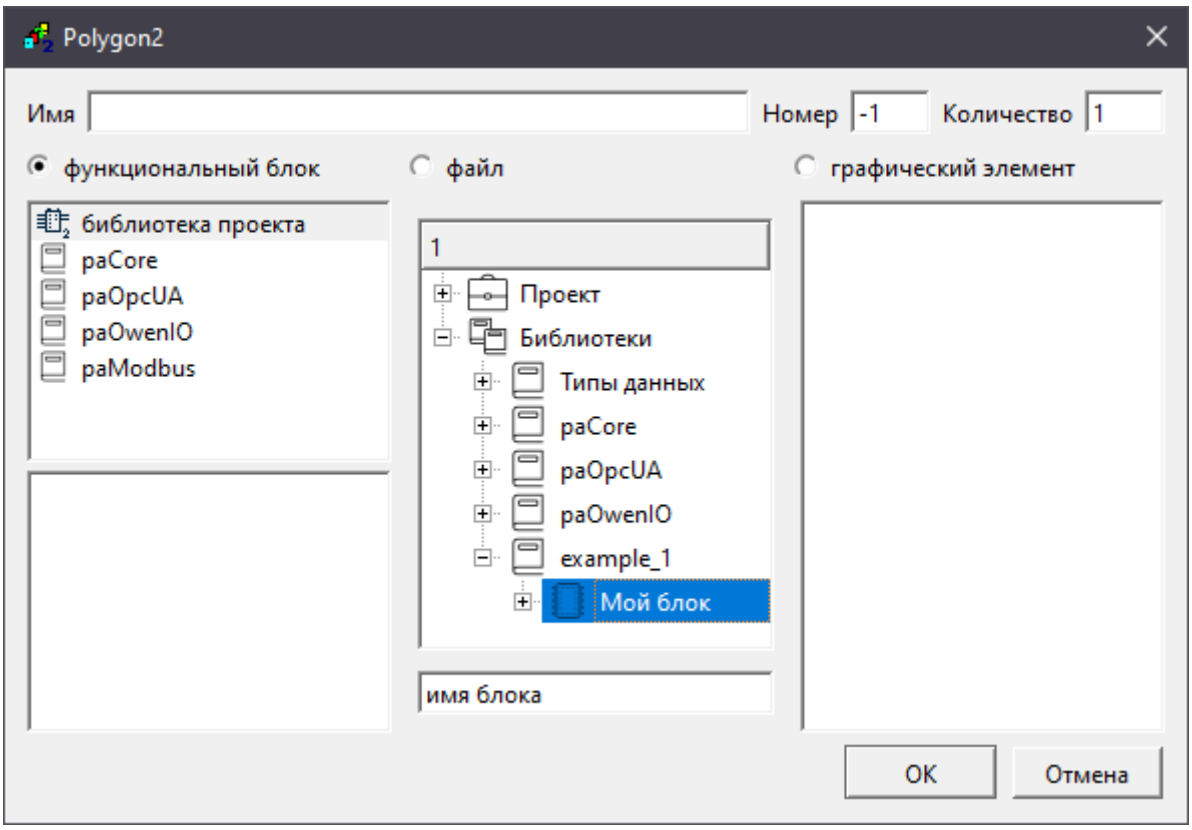


Рисунок 8.13 – Добавление пользовательского составного блока в проект

На графическом отображении составных блоков на страницах проекта изображается знак **#**, что означает, что данный блок – составной.

В дереве проекта или библиотеки ***Тип***, экземпляр которого был создан на странице проекта, подсветится желтым.

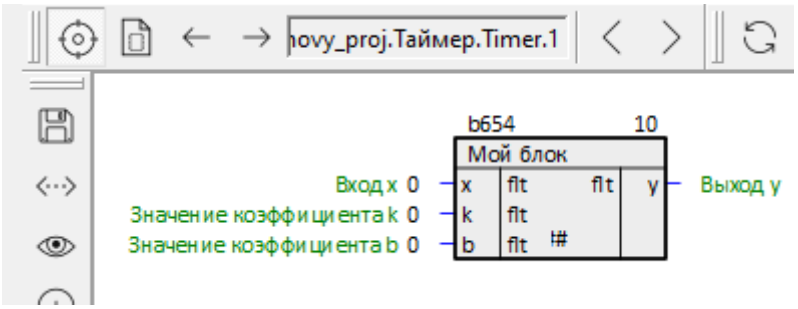


Рисунок 8.14 – Составной блок в проекте

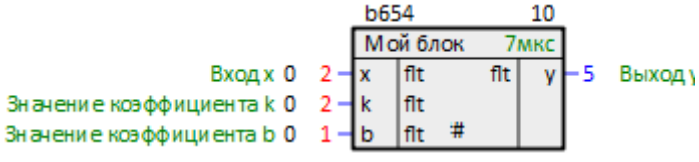


Рисунок 8.15 – Работа составного блока

8.1.2 Создание составного блока на C++

Рассмотрим создание составного функционального блока с алгоритмом, написанным на **C++**.

В качестве примера создадим блок RS-триггера с приоритетом по сбросу. Для этого следует:

1. Создать Тип.
2. Создать внешние входы/выходы типа через контекстное меню ***Типа*** командой ***Создать***. В появившемся окне следует выбрать ***вход типа*** или ***выход типа*** и ввести ***Имя*** входа/выхода.

ВНИМАНИЕ
Имя входа или выхода должно содержать только латинские буквы, цифры и **«_»**. Если оставить поле ввода имени пустым – оно будет назначено автоматически по шаблону ***b<индекс>***.

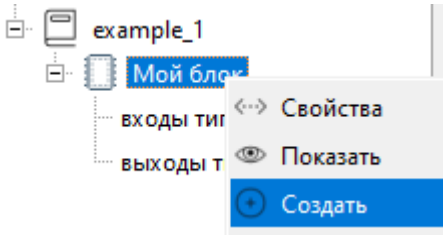


Рисунок 8.16 – Создание внешних входов/выходов типа

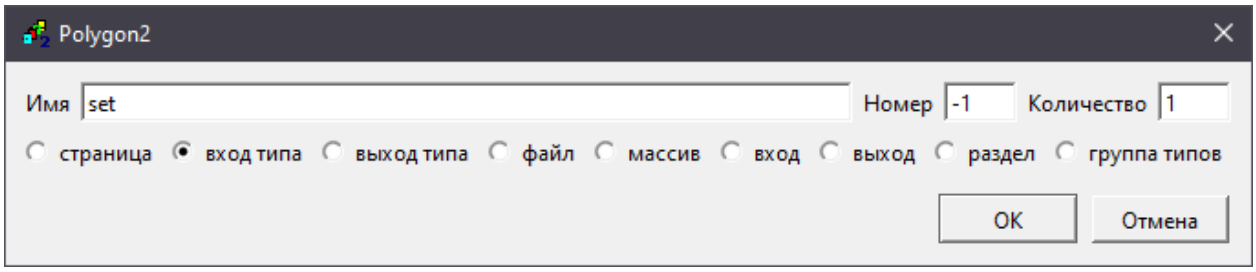


Рисунок 8.17 – Создание внешнего входа типа

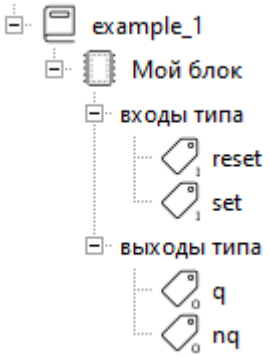


Рисунок 8.18 – Результат создания внешних входов/выходов типа

3. Задать свойства добавленным входам/выходам.

Таблица 8.1 – Свойства входов/выходов типа

Свойство	Описание
Имя	отображается как имя входа/выхода блока
Имя переменной	Переменная класса, соответствующая этому входу/выходу, должно быть уникальным в пределах класса
Номер	Определяет положение входа/выхода в дереве и в составном блоке
Комментарии	Отображается у входа/выхода в дереве и у составного блока
Индекс типа данных	Индекс типа данных входа/выхода из раздела ***Библиотеки/Типы данных***
Тип связей	Для **входов** определяет правила, по которым разрешается проводить связи на данный вход: **0** – можно провести связь или задать константу (переменная – указатель); **1** – связь проводить нельзя, вход константный (переменная – не указатель); **2** – нельзя задать константу, можно провести связь, но она не обязательна (при отсутствии связи на вход будет передан **0**, переменная – указатель); **3** – нельзя задать константу, можно провести связь, и она обязательна (при отсутствии связи при трансляции будет выдана ошибка, переменная – указатель)
Указатель	Для **выходов**: **Установлен флаг** – признак того, что переменная является указателем; **Не установлен флаг** – обычный выход **ПРИМЕЧАНИЕ** По умолчанию добавляемые выходы не являются указателями, если необходимо соответствующее свойство можно добавить в разделе ***Свойства***.

Для того чтобы задать тип данных входа/выхода следует:

4. Найти нужный тип данных в разделе ***Библиотеки/Типы данных***. В данном примере – тип данных ***b*** (bool).
5. Открыть свойства ***b***.
6. Скопировать значение свойства ***Индекс***.

ПРИМЕЧАНИЕ
Свойство ***Индекс*** типов данных в разделе ***Библиотеки/Типы данных*** может различаться в разных проектах.

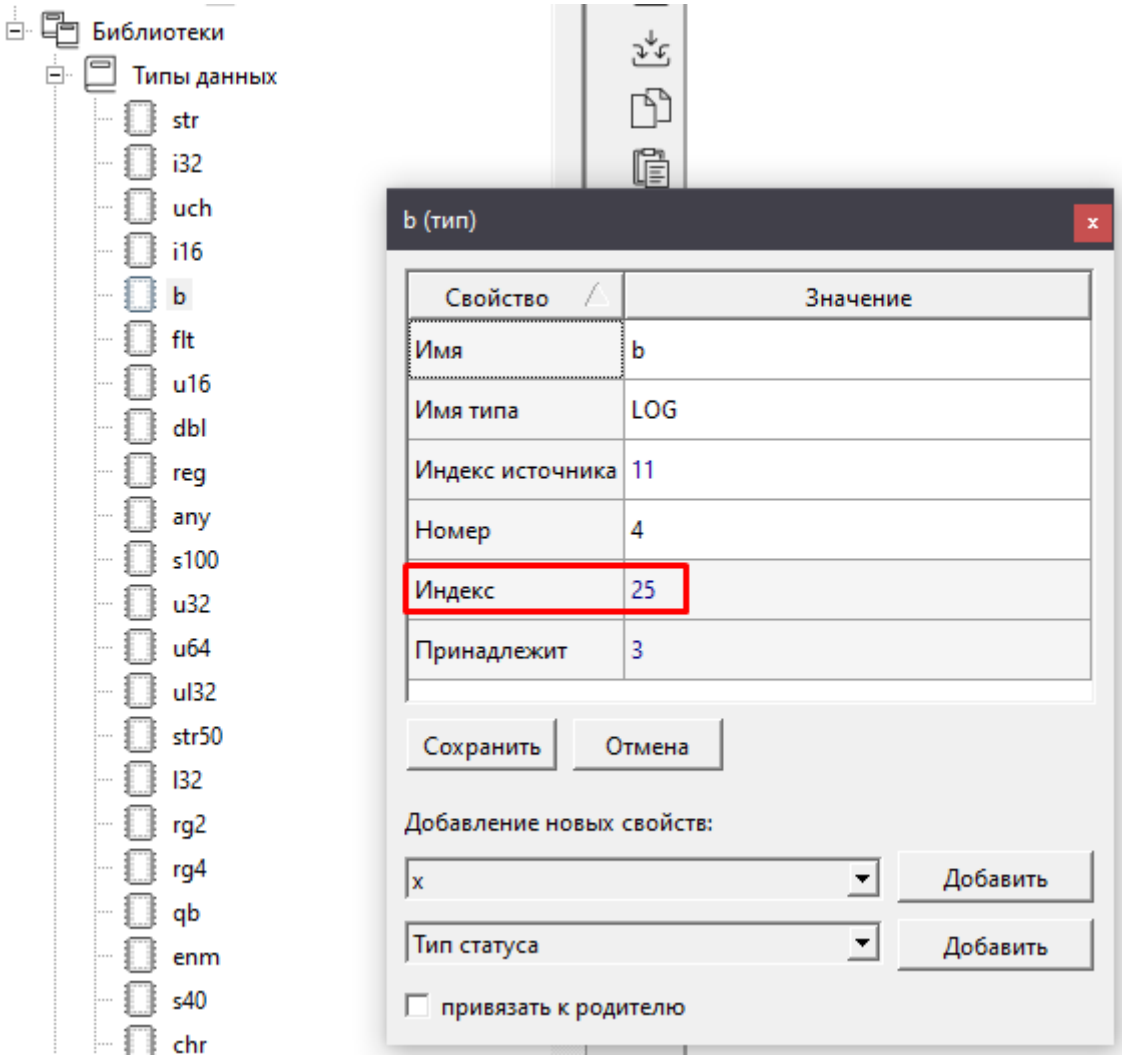


Рисунок 8.19 – Индекс для типа данных b

7. Подставить скопированное значение в свойство ***Индекс типа данных*** нужного входа или выхода.

При правильной подстановке индекса типа данных в свойствах входа или выхода появится ***Имя типа***, соответствующее указанному типу. Если в разделе ***Библиотеки/Типы данных*** нет подходящего типа данных, его можно скопировать из основной библиотеки.

8. Задать таким же образом ***Индекс типа данных*** для всех созданных входов/выходов типа.
9. Установить свойство входов ***Тип связей = 0***.

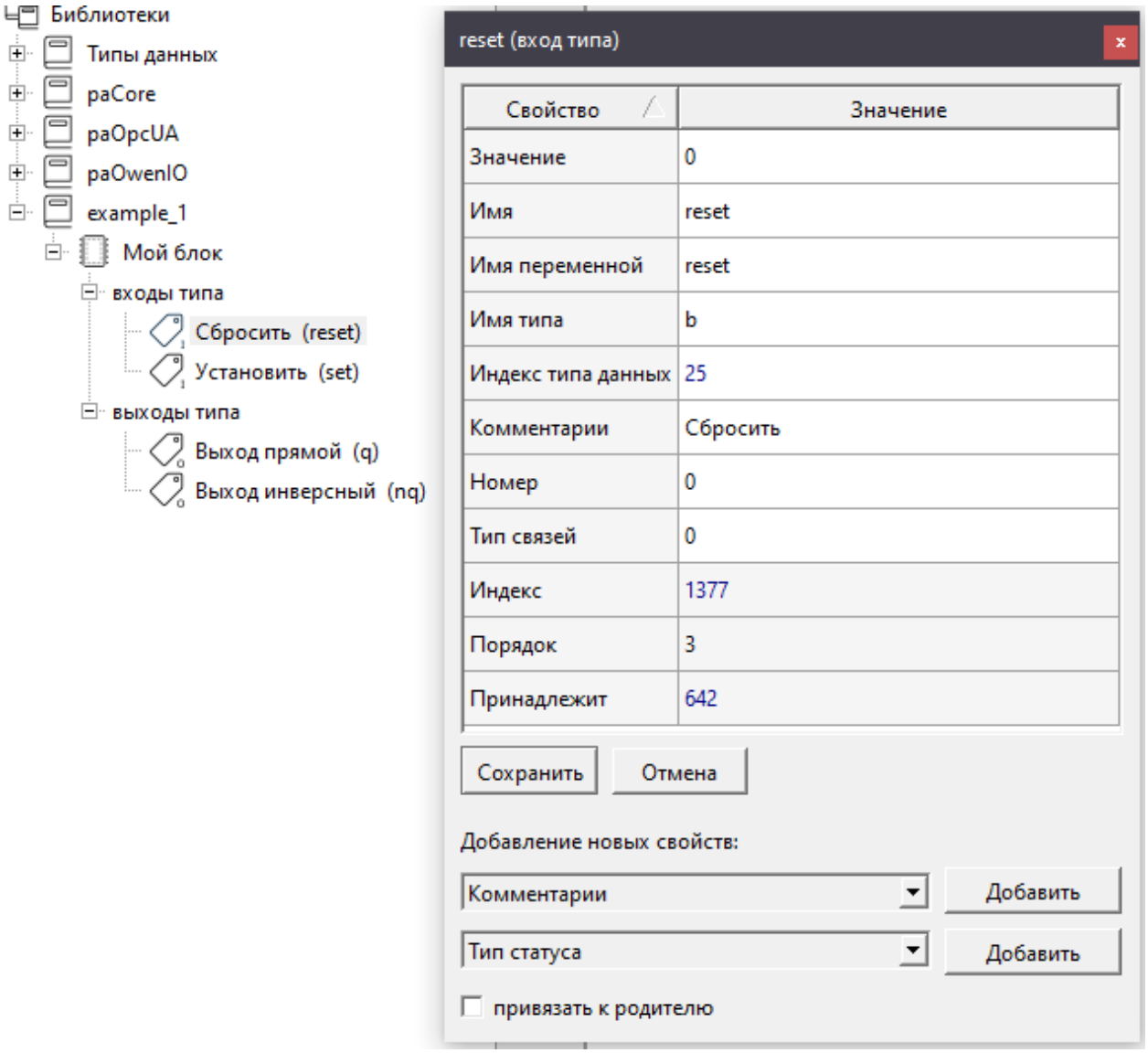


Рисунок 8.20 – Свойства входа reset

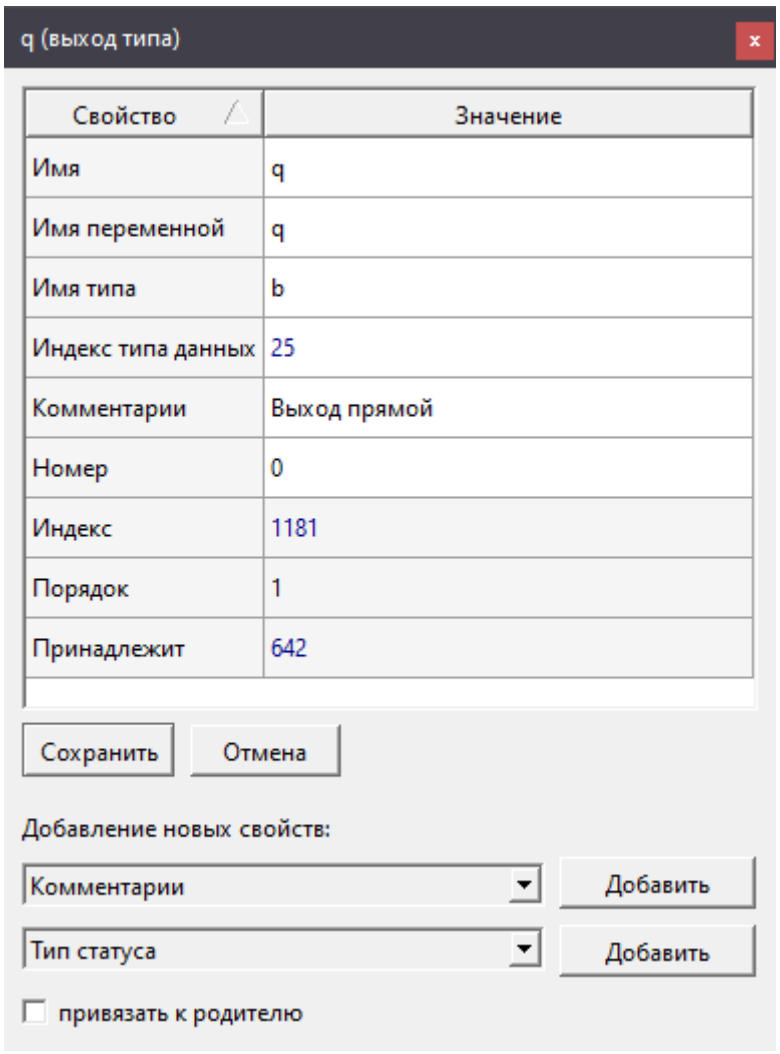


Рисунок 8.21 – Свойства выхода q

После создания внешних входов и выходов типа можно приступить к реализации алгоритма работы блока.

Для того чтобы написать блок на **C++** следует добавить в ***Тип*** текстовые файлы. Для создания алгоритмов на **C++** используются следующие расширения файлов:

- **.work** – вызывается из функции ***Work*** каждый цикл выполнения программы;
- **.init** – вызывается из функции ***InitInp*** один раз при инициализации;
- **.hpp** – включается внутрь описания класса, это могут быть объявления дополнительных переменных и функций;
- **.cpp** – включается в **.cpp**-файл класса – реализация дополнительных функций, добавление библиотек.

Для создания блока необходимо добавить как минимум один файл с расширением **.work**.

Для реализации RS-триггера понадобится создать файлы с расширениями **.work** для написания логики работы триггера и **.hpp** для того, чтобы хранить значение выхода во внутренней памяти блока.

Чтобы добавить текстовый файл в ***Тип*** следует:

10. Выполнить команду ***Создать*** в контекстном меню типа.
11. Выбрать ***файл*** и задать ему имя и расширение.

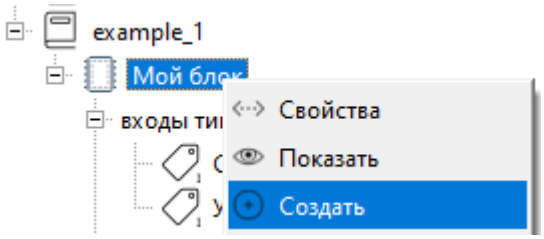


Рисунок 8.22 – Создание файла в типе

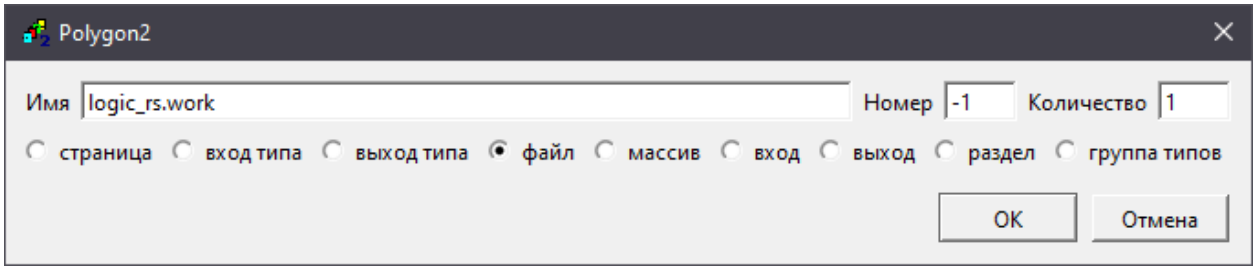


Рисунок 8.23 – Создание файла с расширением .work

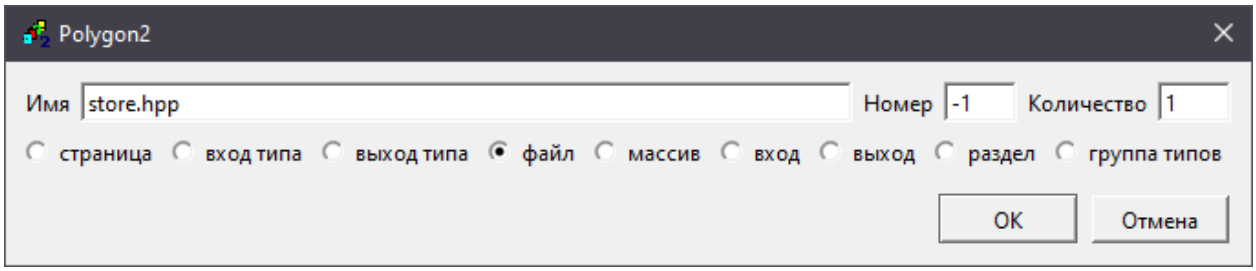


Рисунок 8.24 – Создание файла с расширением .hpp

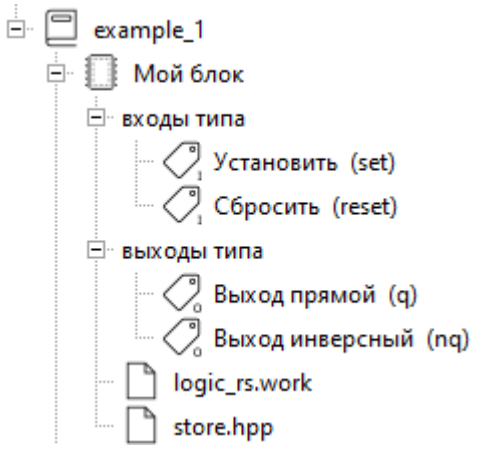


Рисунок 8.25 – Созданные файлы в дереве проекта

Для редактирования текстовых файлов в Полигоне предназначено окно типа Редактор, которое открывается автоматически при двойном щелчке на текстовом файле в дереве проекта

Если файла на диске не существует, то он будет создан при нажатии кнопки ***Сохранить***. По умолчанию файлы хранятся в той же папке, где расположен проект или библиотека.

12. Прописать в файле *logic_rs.work* логику работы RS-триггера.
13. Нажать ***Сохранить***.

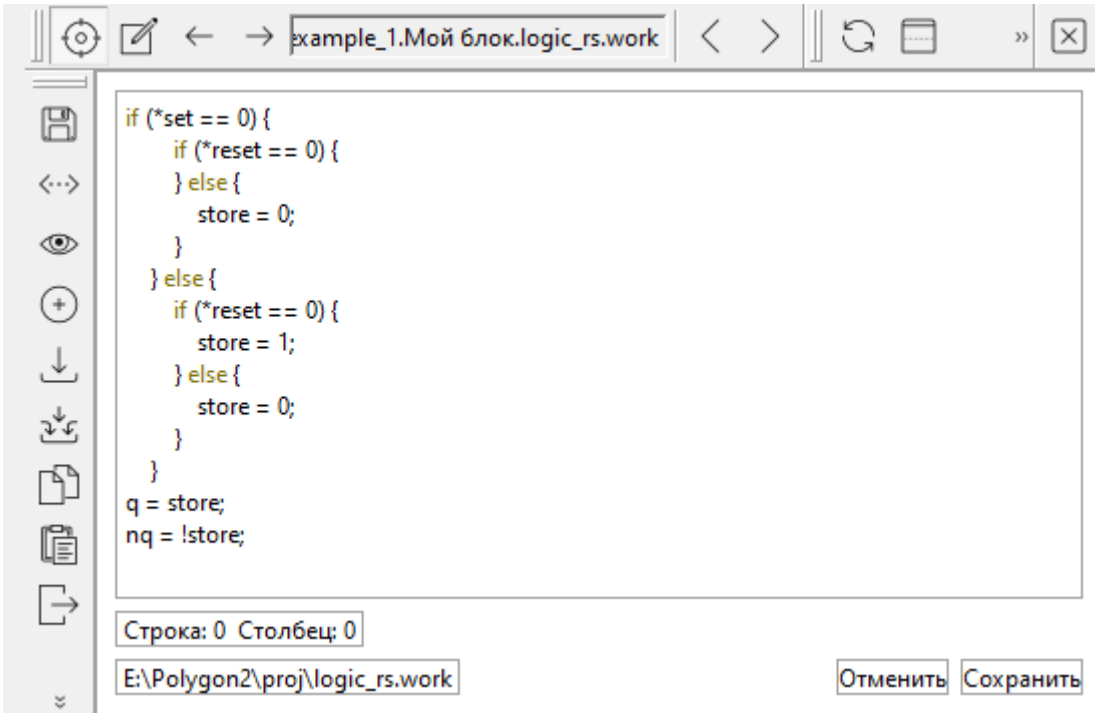


Рисунок 8.26 – Пример содержания файла с расширением .work

14. В файле *store.hpp* объявить переменную для хранения значения триггера.
15. Нажать ***Сохранить***.

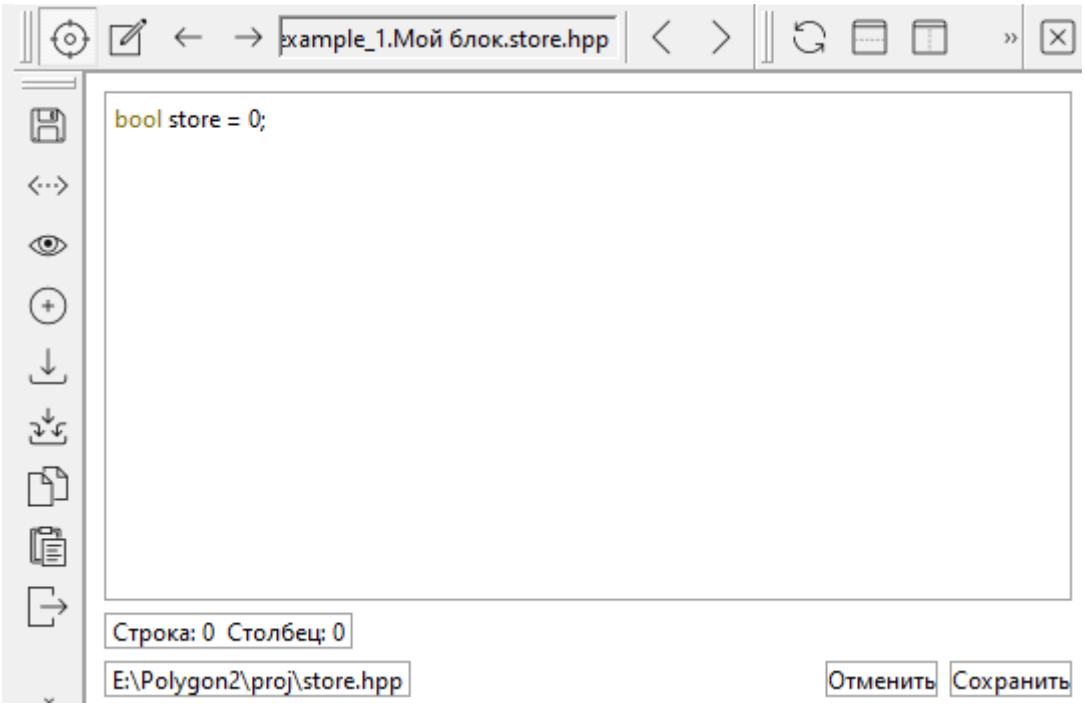


Рисунок 8.27 – Пример содержания файла с расширением .hpp

16. Транслировать проект для проверки корректности созданного типа. После сообщения об успешной трансляции – ***Трансляция завершена*** экземпляры нового типа можно создавать на страницах проекта.

Создать экземпляр составного блока в проекте можно двумя способами:

1. Перетащить блок из ***Дерева*** проекта или библиотеки на ***Страницу*** проекта и выбрать ***Создать*** в контекстном меню.

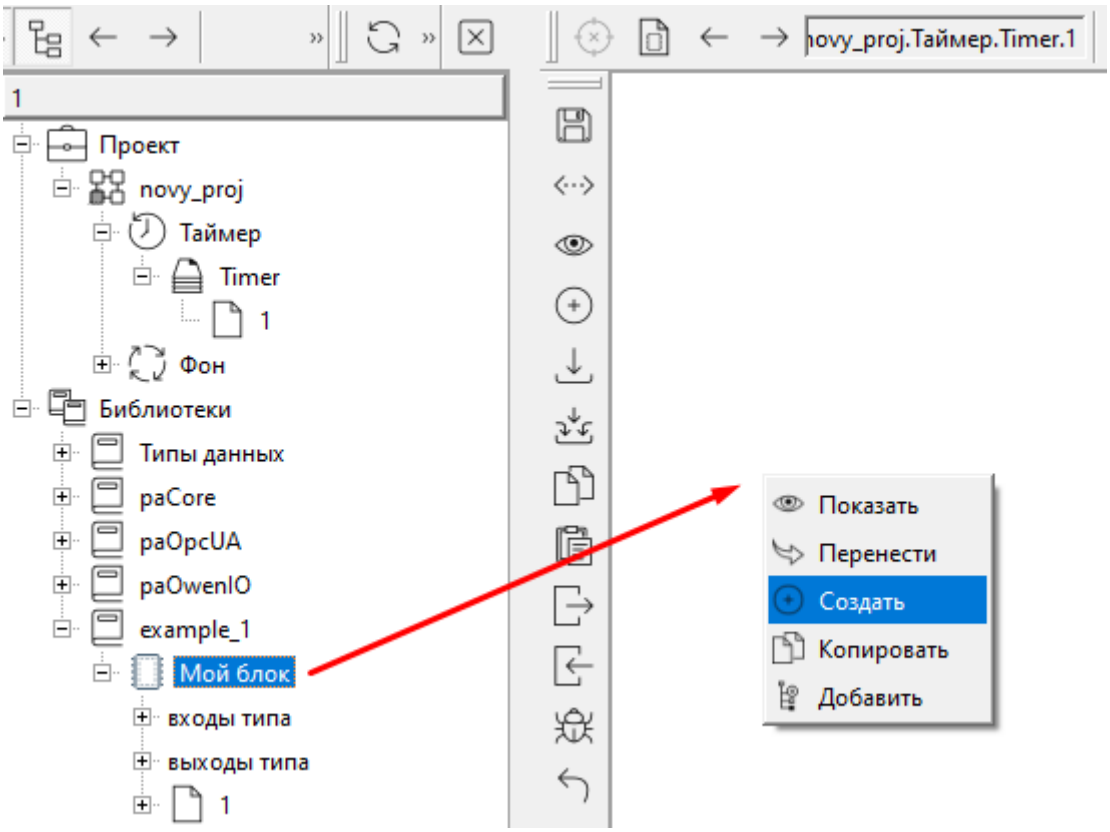


Рисунок 8.28 – Добавление пользовательского составного блока в проект

2. На пустом месте ***Страницы*** щелкнуть ПКМ – ***Создать*** и выбрать необходимый блок в разделе ***Библиотека проекта***, нажать **ОК**.

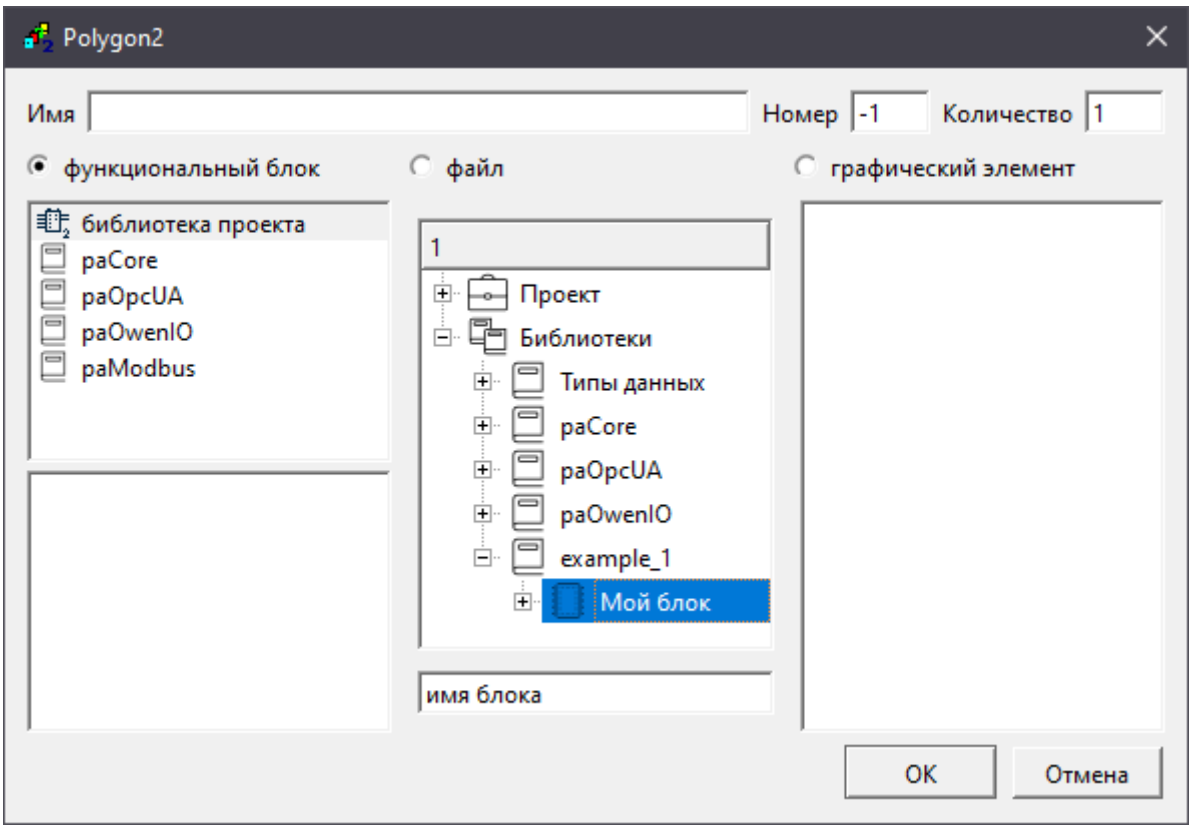


Рисунок 8.29 – Добавление пользовательского составного блока в проект

На графическом отображении составных блоков на страницах проекта изображается знак **#**, что означает, что данный блок – составной.

В дереве проекта или библиотеки ***Тип***, экземпляр которого был создан на странице проекта, подсветится желтым.

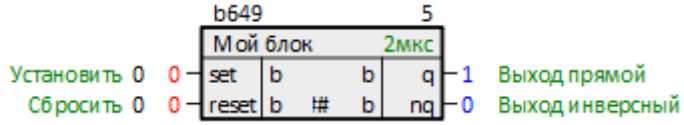


Рисунок 8.30 – Работа составного блока

8.1.2.1 Создание составных блоков с переменным числом входов/выходов на C++

Одно из важных преимуществ создания пользовательских составных блоков на **C++** – возможность создавать блоки с переменным количеством входов/выходов.

Для того чтобы создать входы/выходы переменного количества в блоке следует создать ***Массив*** в ***Типе***.

В ***Массиве*** следует создать либо входы, либо выходы, **но не вместе**. Возможно два варианта:

1. ***Простой массив*** состоит из одного входа или выхода. ***Имя переменной*** входа/выхода определяется именем массива. Например, если массив имеет ***Имя*** – **arr**, то ***Имя переменной*** его входа/выхода – ***arr[%1]***. Простой массив соответствует **циклическим** входам или выходам.

2. ***Массив структур*** состоит из нескольких входов или выходов. Имя структуры задается свойством ***Имя типа***. ***Имя переменной*** входа/выхода определяется именем массива и именем переменной внутри структуры, например, ***arr[%1].in***. Массив структур соответствует **циклическим группам** входов или выходов.

Имена переменных внутри массива заполняются автоматически, если перед добавлением входов/выходов правильно задать свойство ***Имя типа***.

Массивы входов и выходов могут быть объединены так, чтобы при создании входа (или группы входов) у блока автоматически создавался связанный с ним выход (или группа выходов).

Для этого следует обоим массивам входов и выходов задать одинаковое значение свойства ***Принадлежит к группе***, например, **1**.

Рассмотрим пример создания блока логического НЕ с переменным числом входов/выходов, следует:

1. Создать Тип.

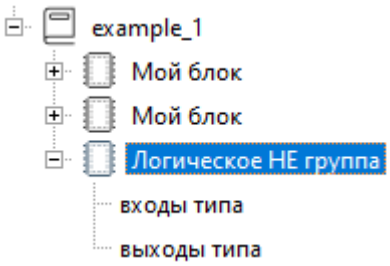


Рисунок 8.31 – Создание Типа

2. Создать в ***Типе*** два массива, первый – массив входов, второй – массив выходов.

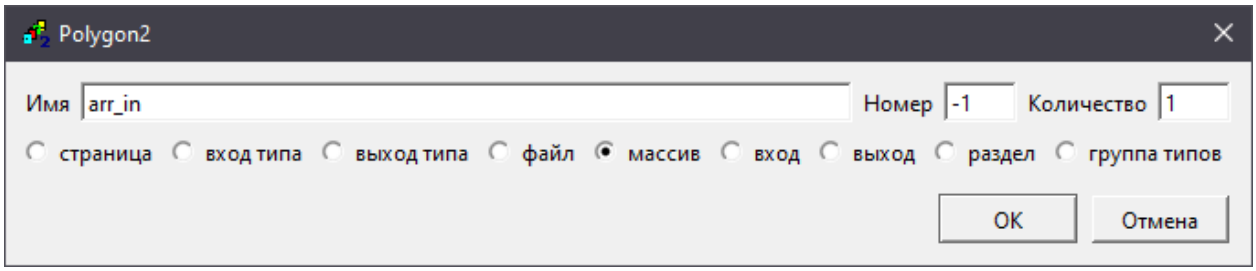


Рисунок 8.32 – Создание массива входов

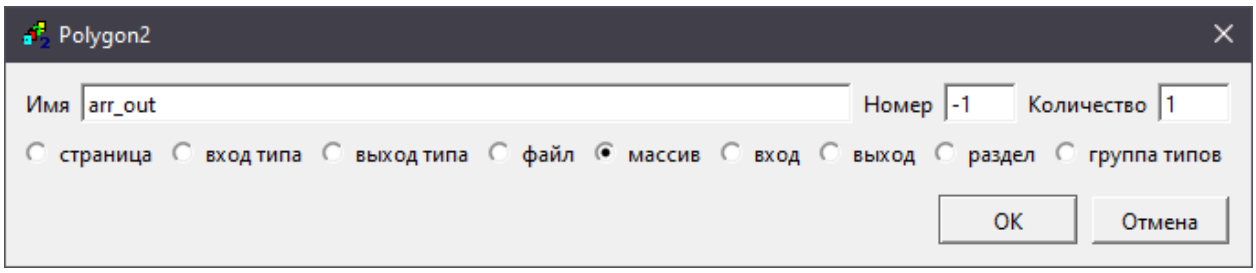


Рисунок 8.33 – Создание массива выходов

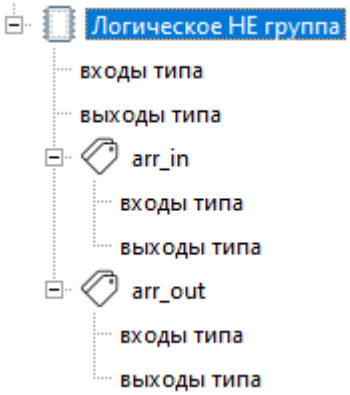


Рисунок 8.34 – Результат создания массивов

3. В свойствах обоих массивов добавить свойство ***Принадлежит к группе = 1***.

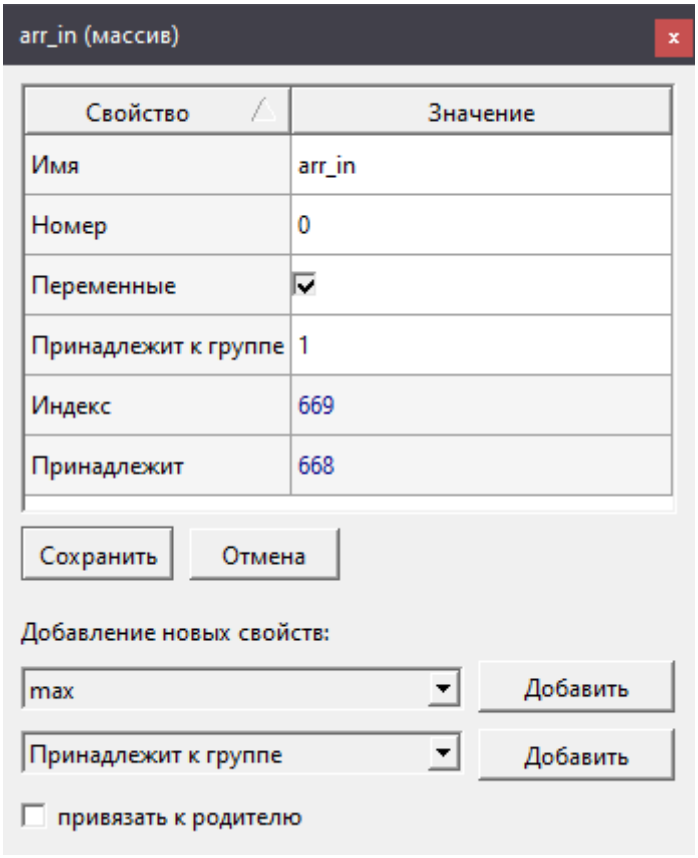


Рисунок 8.35 – Свойства массива входов

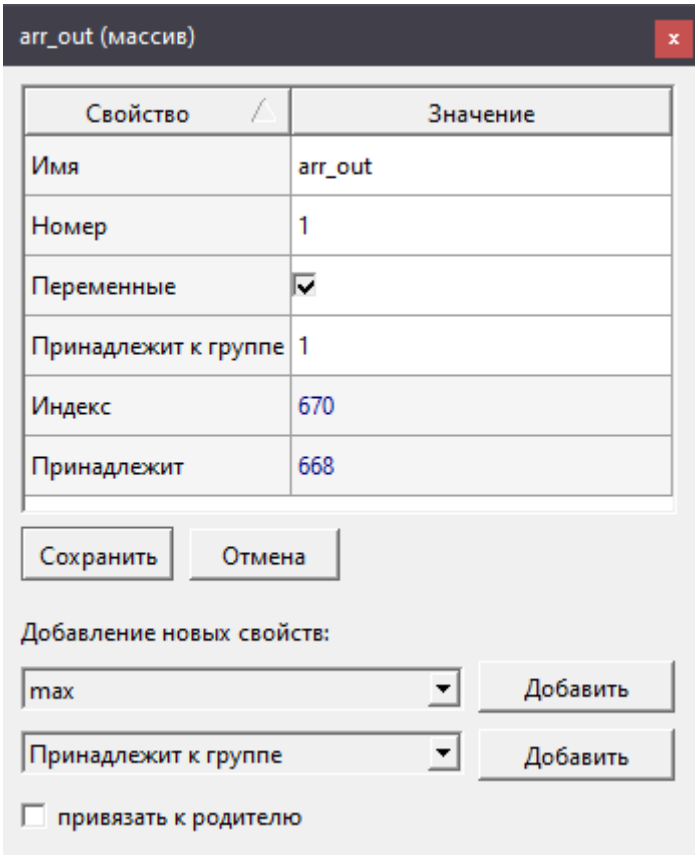


Рисунок 8.36 – Свойства массива выходов

4. Создать ***вход типа*** в массиве входов, создать ***выход типа*** в массиве выходов.

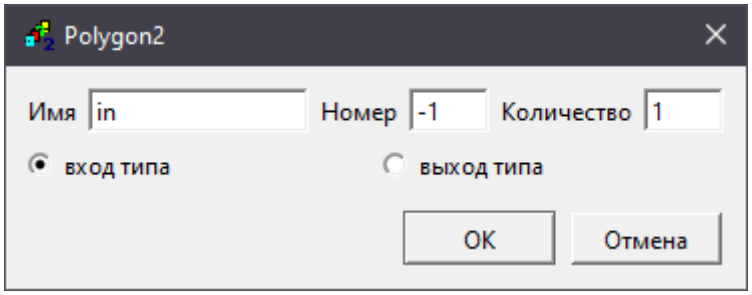


Рисунок 8.37 – Создание входа

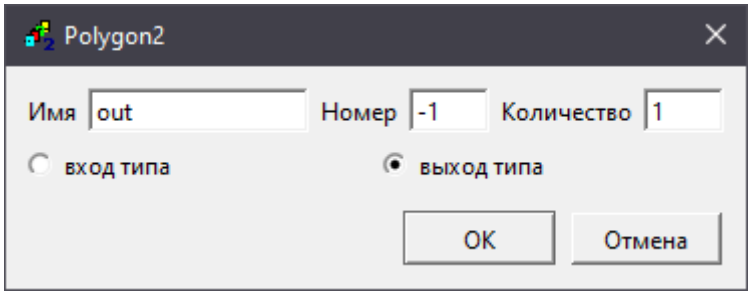


Рисунок 8.38 – Создание выхода

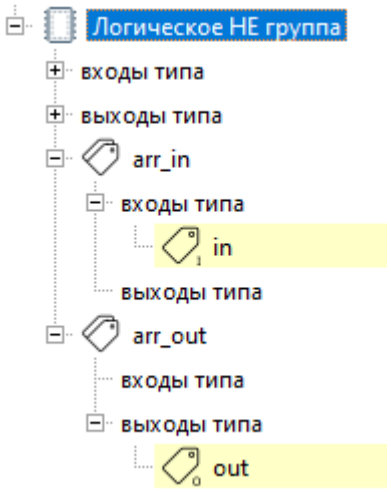


Рисунок 8.39 – Результат создания входа и выхода

5. В свойствах входа и выхода добавить свойство ***Индекс типа данных*** и скопировать в него значение ***Индекса*** типа данных ***b*** из раздела ***Библиотеки/Типы данных*** аналогично предыдущему разделу.

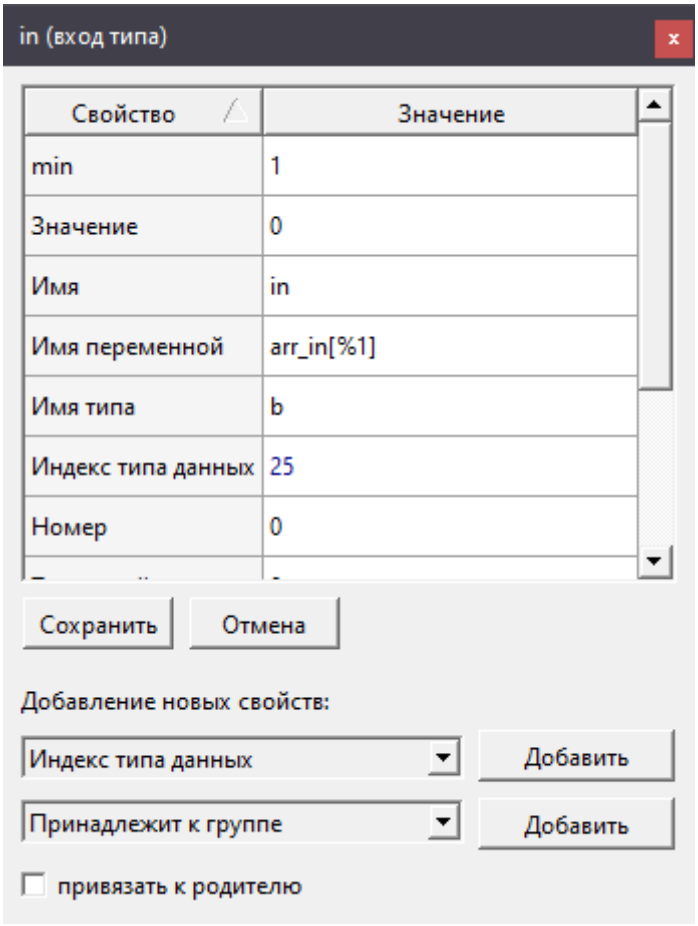


Рисунок 8.40 – Свойства входа

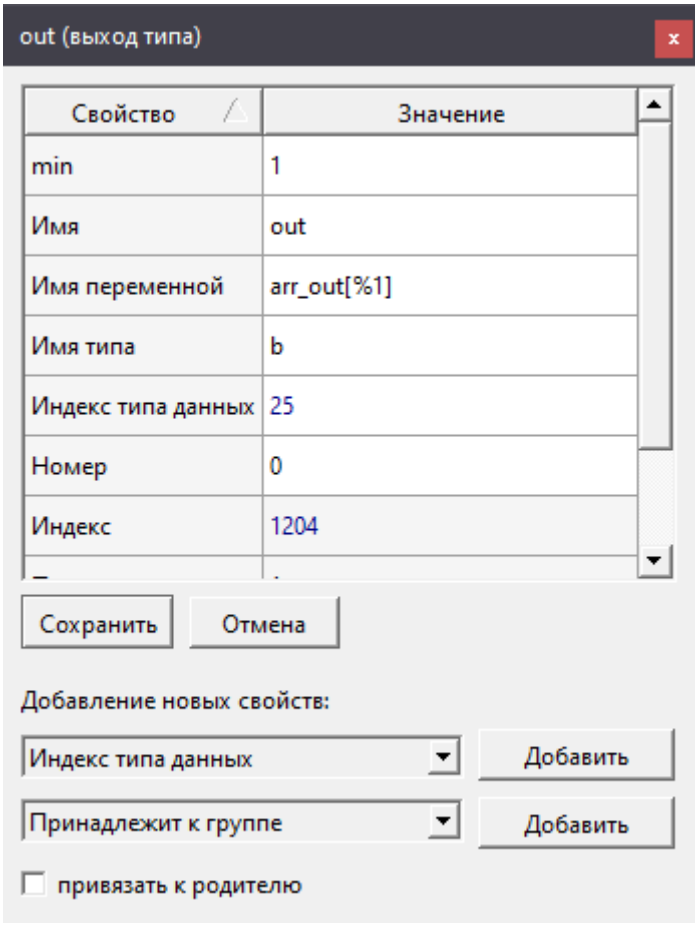


Рисунок 8.41 – Свойства выхода

6. Добавить в ***Тип*** файл **.work**, в котором расписать логику работы блока.

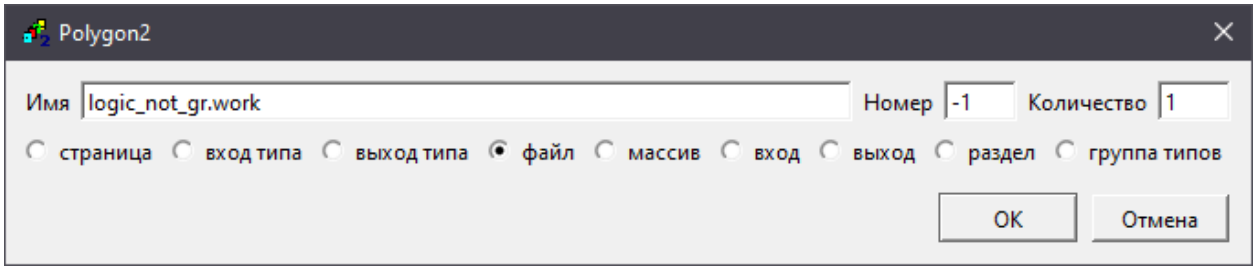


Рисунок 8.42 – Добавление файла с расширением .work

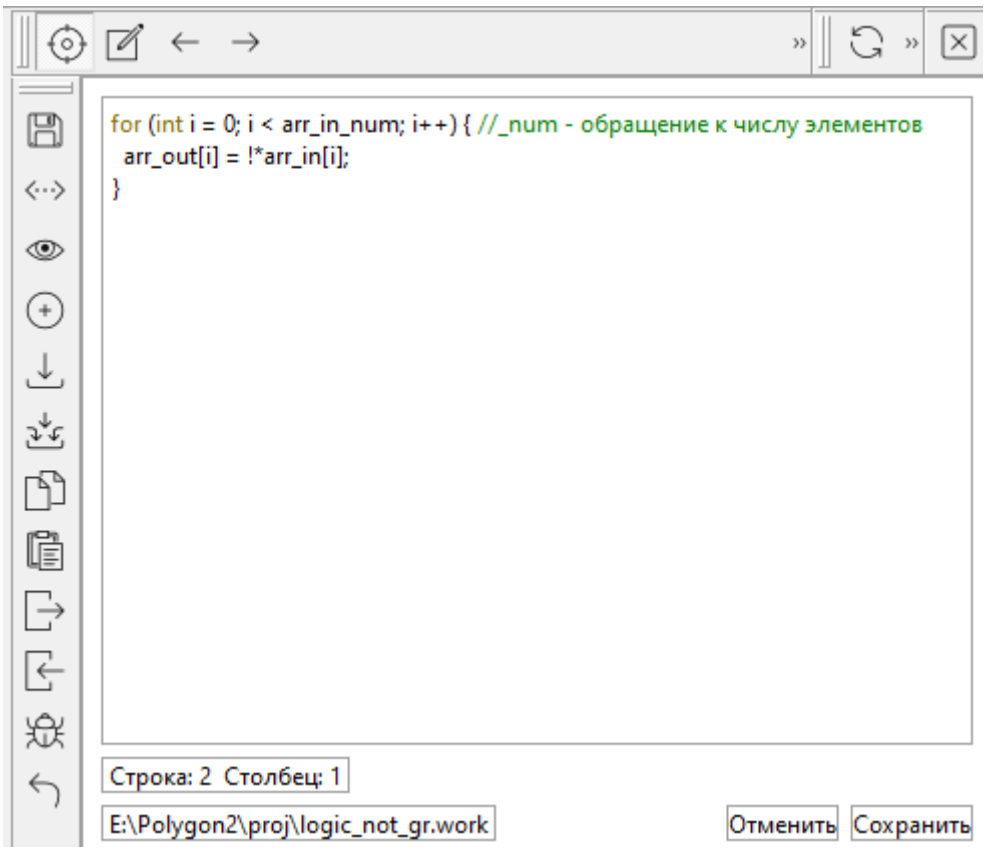


Рисунок 8.43 – Логика работы блока

7. Транслировать проект для проверки корректности созданного типа. После сообщения об успешной трансляции – ***Трансляция завершена*** экземпляры нового типа можно создавать на страницах проекта.
8. Создать экземпляр блока в проекте.

Создание экземпляра блока на странице проекта описано в предыдущем разделе.

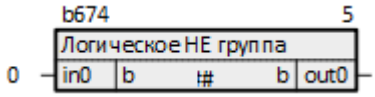


Рисунок 8.44 – Блок на странице проекта

9. Создать у блока дополнительные входы – одновременно с входами добавилось соответствующее количество выходов.

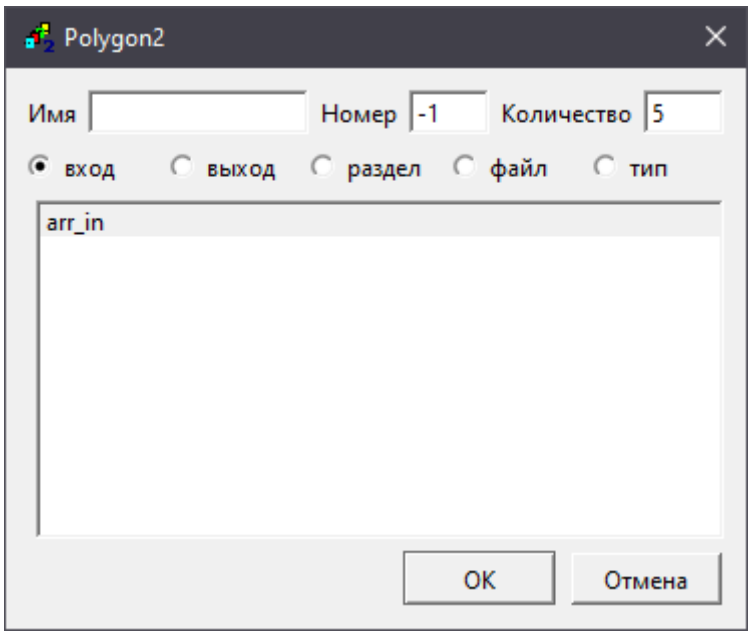


Рисунок 8.45 – Добавление входов блока

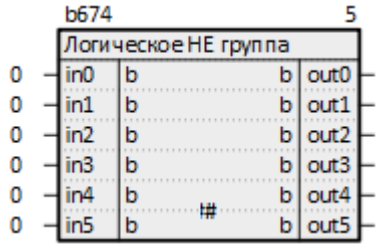


Рисунок 8.46 – Блок с добавленными входами

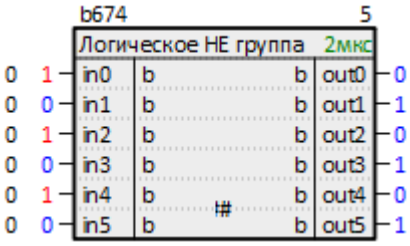


Рисунок 8.47 – Работа блока

8.1.2.2 Доступ к системной информации из кода составного блока на C++

Доступ к системной информации можно получить из статических функций класса **runtimeInfo**:

1. **runtimeInfo::nsAbsTime()** возвращает значение **uint64_t** равное количеству **наносекунд** прошедших от **01.01.1970**, можно использовать как абсолютное время в **UTC**, например, для меток времени. Может подстраиваться **NTP**-клиентом, поэтому не стоит использовать это значение при измерениях интервалов. Обновляется в потоке ***Таймер***, если он один, или в потоке ***Ввод-вывод***, один раз за цикл.

2. **runtimeInfo::nsMonotonic()** возвращает значение **uint64_t** равномерно меняющегося счетчика в **наносекундах**. Не подстраивается **NTP** и при изменении системного времени, поэтому можно использовать для измерения интервалов, больших таймерного цикла. Обновляется в потоке ***Таймер***, если он один, или в потоке ***Ввод-вывод***, один раз за цикл.
3. **runtimeInfo::timerCycle()** возвращает значение типа **float** равное циклу таймерного потока, в котором выполняется блок, в **миллисекундах**. Если блок выполняется в фоновом потоке, то значение не определено.
4. **runtimeInfo::cpuCoresNum()** возвращает количество ядер процессора (тип **uint16_t**).

8.1.3 Свойства типов составных блоков и их входов/выходов

Для удобства отображения в дереве проекта или библиотеки ***Типу*** можно задать свойство ***Имя*** – это имя, которое отображается в дереве проекта или библиотеки.

Еще одно полезное свойство ***Типа*** – ***Может работать только в…***. Его можно добавить из нижнего выпадающего списка в окне ***Свойства***. Данное свойство определяет, в каком ***Месте работы*** можно создать экземпляр блока данного типа. При попытке создать блок в другом ***Месте работы*** появится окно ошибки.

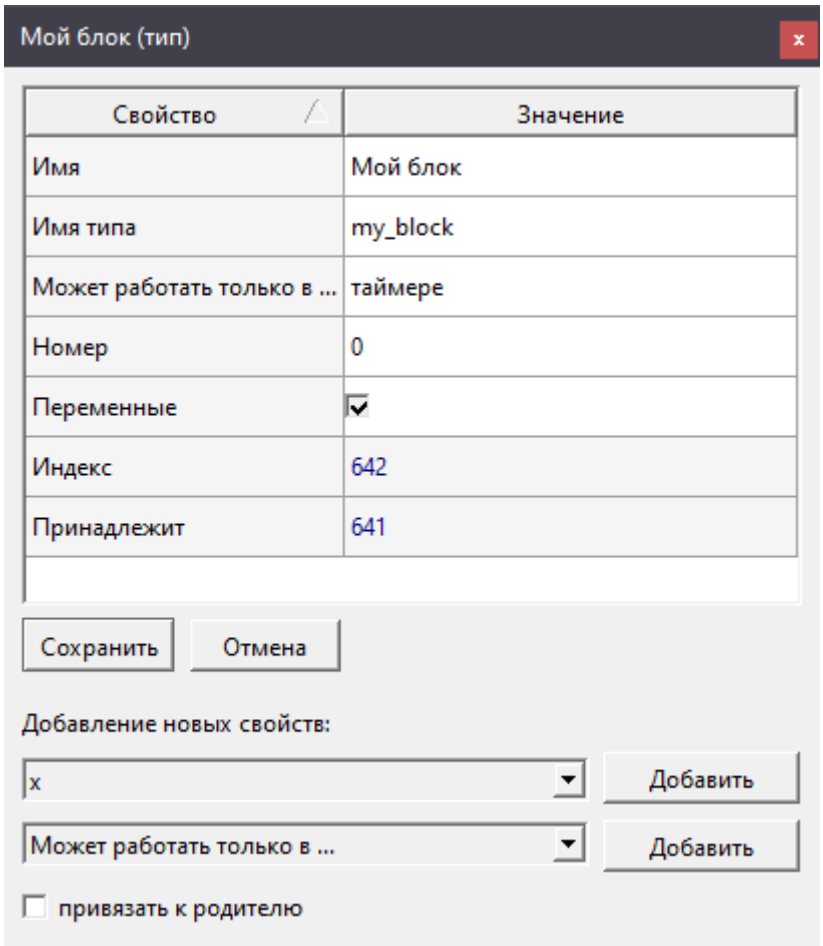


Рисунок 8.48 – Свойства Типа

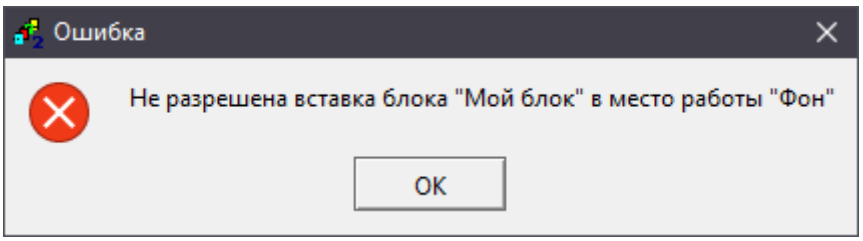


Рисунок 8.49 – Ошибка вставки блока

Основные свойства внешних входов/выходов составного блока:

- ***Имя*** – отображается как имя входа/выхода блока;
- ***Номер*** – определяет положение входа/выхода в дереве и в составном блоке;
- ***Комментарии*** – отображается у входа/выхода в дереве и у составного блока.

При изменении свойства ***Номер*** внешнего входа или выхода ***Типа*** появится сообщение ***Обновить блоки типа…*** – применить изменение ко всем экземплярам блока данного типа в проекте.

Если необходимо изменить свойства ***Имя*** и/или ***Комментарии*** внешних входов/выходов, то для применения изменений следует пересоздать блок на странице проекта.

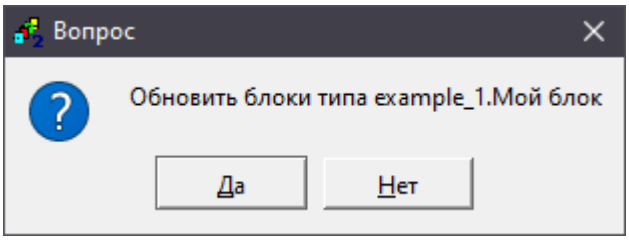


Рисунок 8.50 – Изменение свойств входов/выходов составного блока

8.1.4 Создание справки для составных блоков

Для составных блоков пользователь может добавлять собственное описание.

Для создания справки следует:

1. В контекстном меню ***Типа*** выбрать команду ***Описание***.

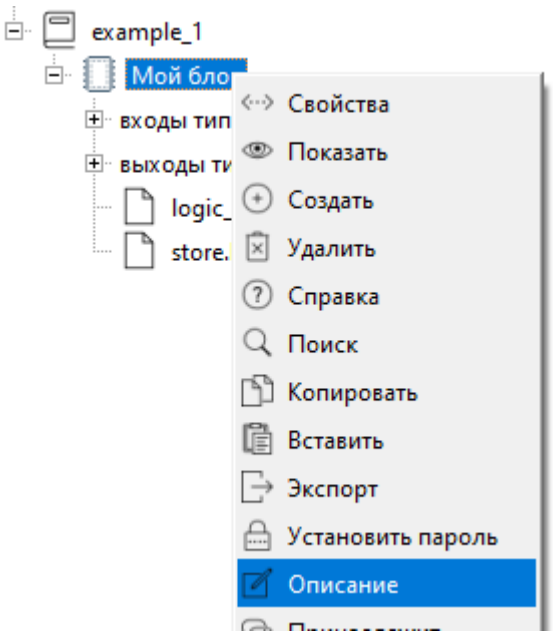


Рисунок 8.51 – Создание описания типа

2. В текущем окне отобразится окно ***Редактора***. В папке проекта (если тип создан в проекте, иначе в папке библиотеки) автоматически создастся папка ***userHelp*** и в ней файл с расширением **.html** и именем типа.
3. Добавить в файл описание блока.

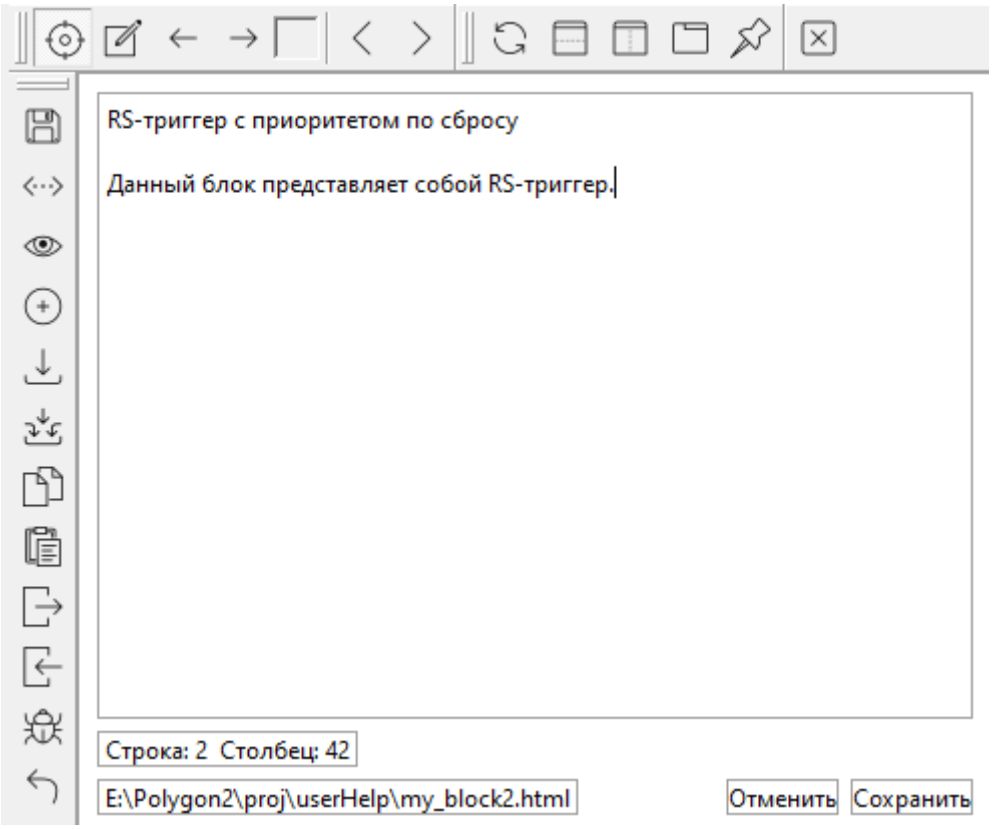


Рисунок 8.52 – Редактирование файла с расширением .html

В папке ***userHelp*** следует размещать все файлы, которые используются для создания описаний для ***Типов*** данной ***Группы типов***.

4. Если необходимо добавить картинки в описание, следует подготовить изображения с типом **.png**, имена которых начинаются с имени соответствующего файла справки, и поместить их в папку ***userHelp***.

Например, для описания *my_block2.html* следует добавить файлы изображений с названиями *my_block2_1.png*, *my_block2_2.png* и т.д.

Вызвать описание блока можно, выделив ***Тип*** в дереве и выполнив команду ***Справка*** в контекстном меню или нажав горячую клавишу **F1**: в нижней части окна ***Справка*** отобразится созданное описание блока, картинки автоматически отобразятся в начало описания.

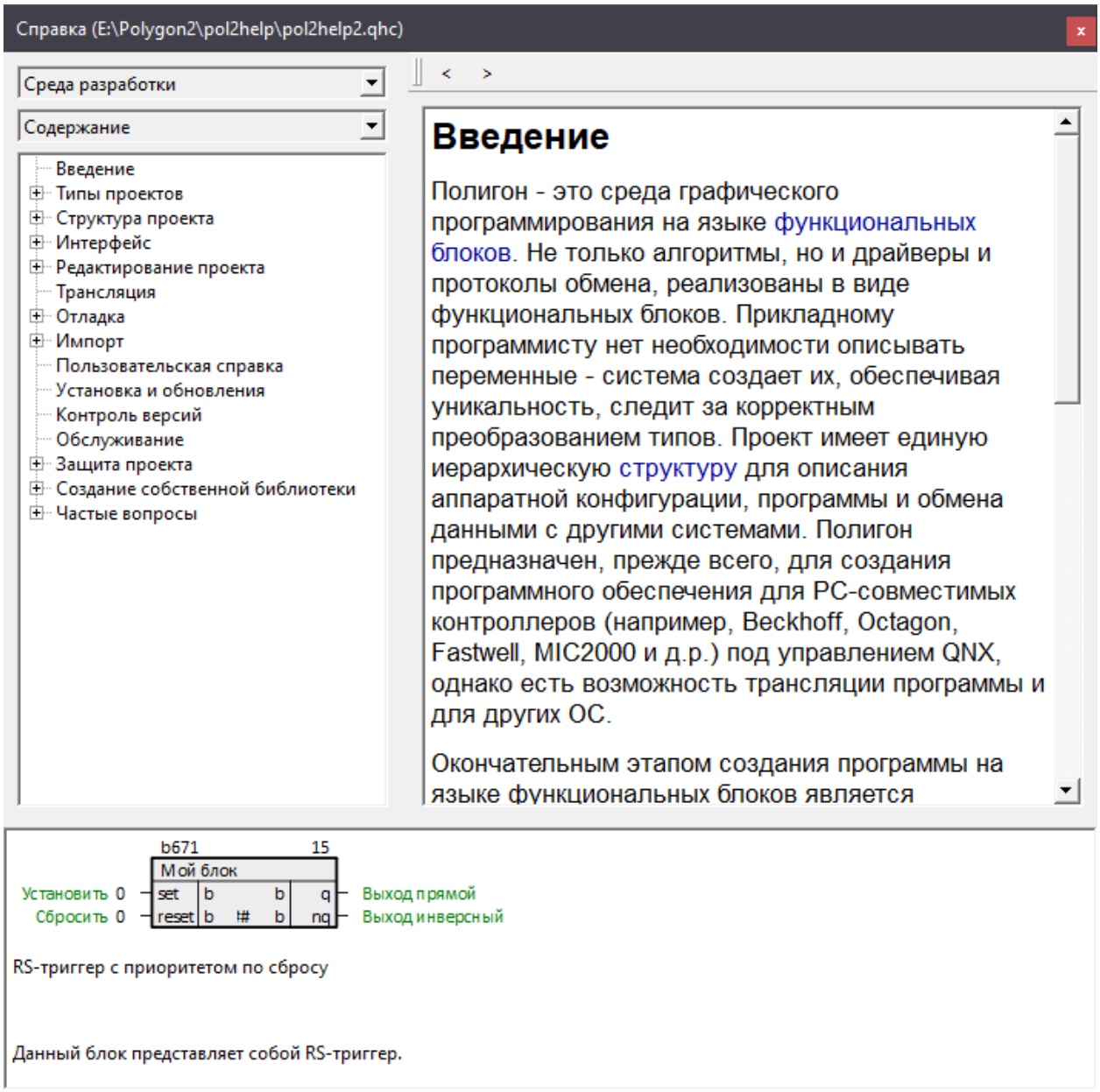


Рисунок 8.53 – Описание типа в окне Справка

8.2 Создание библиотеки

Рассмотрим создание пользовательской библиотеки.

Для каждой новой библиотеки рекомендуется заводить отдельную папку.

ВНИМАНИЕ
Путь к файлам библиотек не должен содержать кириллицу и пробелы.

Для создания библиотеки следует:

1. Перейти в окно ***Проекты*** и нажать ***Создать***.

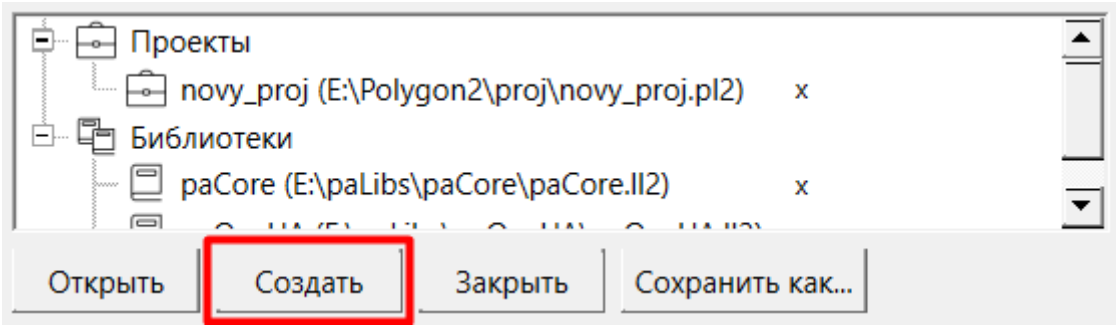


Рисунок 8.54 – Создание пользовательской библиотеки

2. В открывшемся окне перейти папку, где будет создана библиотека, ввести имя новой библиотеки, в выпадающем списке выбрать тип ***Библиотека Полигон 2 (*.ll2)*** и нажать ***Создать***.

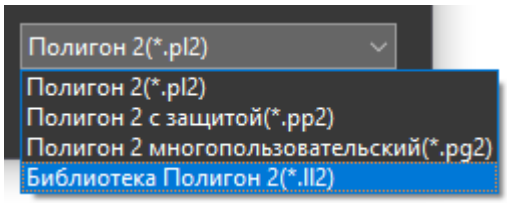


Рисунок 8.55 – Создание пользовательской библиотеки

3. Для работы с библиотекой следует открыть ее в представлении ***Дерево***. Среда при создании библиотеки сразу предложит открыть ее в дереве – следует нажать ***Да***.

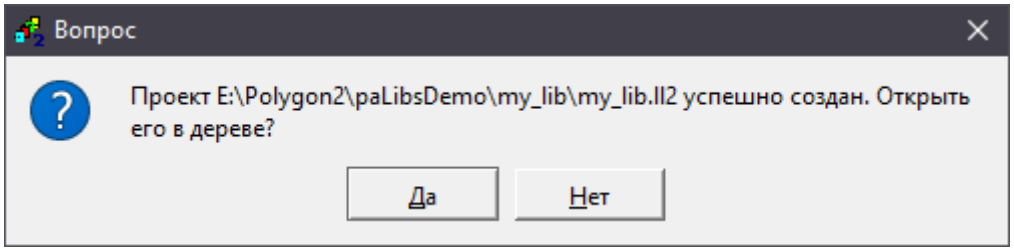


Рисунок 8.56 – Создание пользовательской библиотеки

4. Созданная библиотека отобразится в окне ***Проекты***.

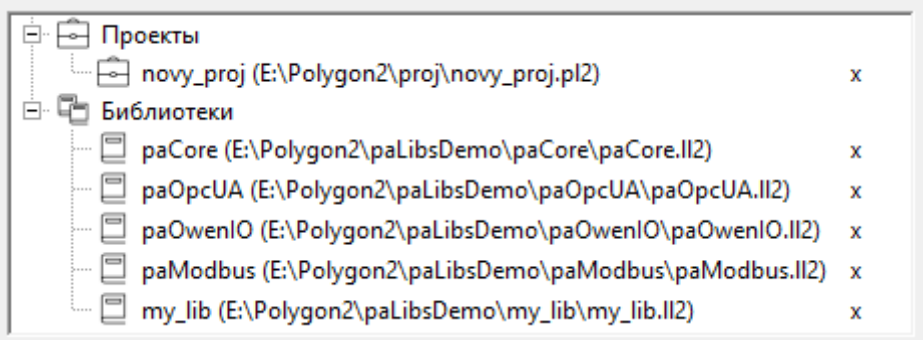


Рисунок 8.57 – Результат Создания пользовательской библиотеки

Для создания блоков в библиотеки следует добавить в нее типы данных. По умолчанию раздел ***Типы данных*** в библиотеке пустой. Его следует скопировать из основной библиотеки ***paCore***. Для этого следует:

5. Открыть библиотеку ***paCore*** в соседнем окне типа ***Дерево***.

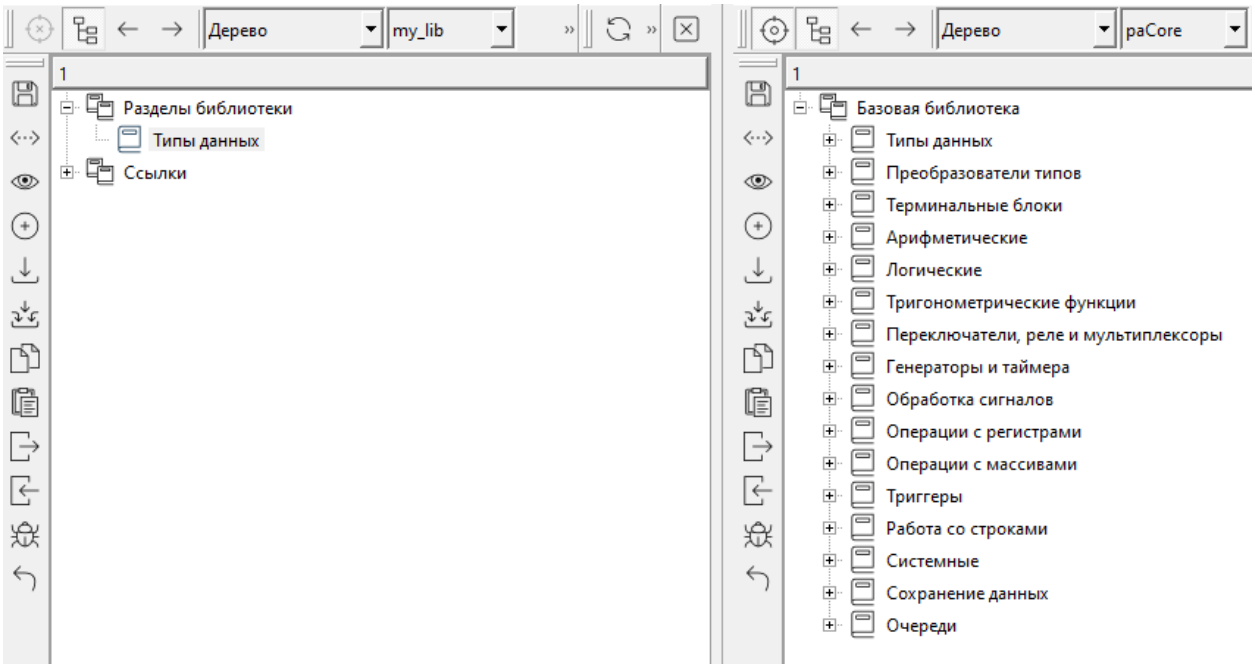


Рисунок 8.58 – Добавление типов данных в пользовательскую библиотеку

6. Открыть раздел ***Типы данных*** в ***paCore***, выделить первый тип данных раздела и с зажатым **Shift** выделить последний тип данных раздела.
7. С помощью команд **Ctrl + c** и **Ctrl + v** выполнить копирование типов данных в раздел ***Типы данных*** новой библиотеки.

Теперь в библиотеке можно создавать пользовательские составные блоки, копируя индексы типов данных для входов и выходов из раздела ***Типы данных*** библиотеки.

После создания необходимого числа блоков библиотеку следует транслировать:

8. В окне ***Экран/Настройки*** выбрать ***ОС для трансляции библиотек***.

ВНИМАНИЕ
Библиотеку необходимо транслировать для всех ОС, в которых ее планируется использовать.

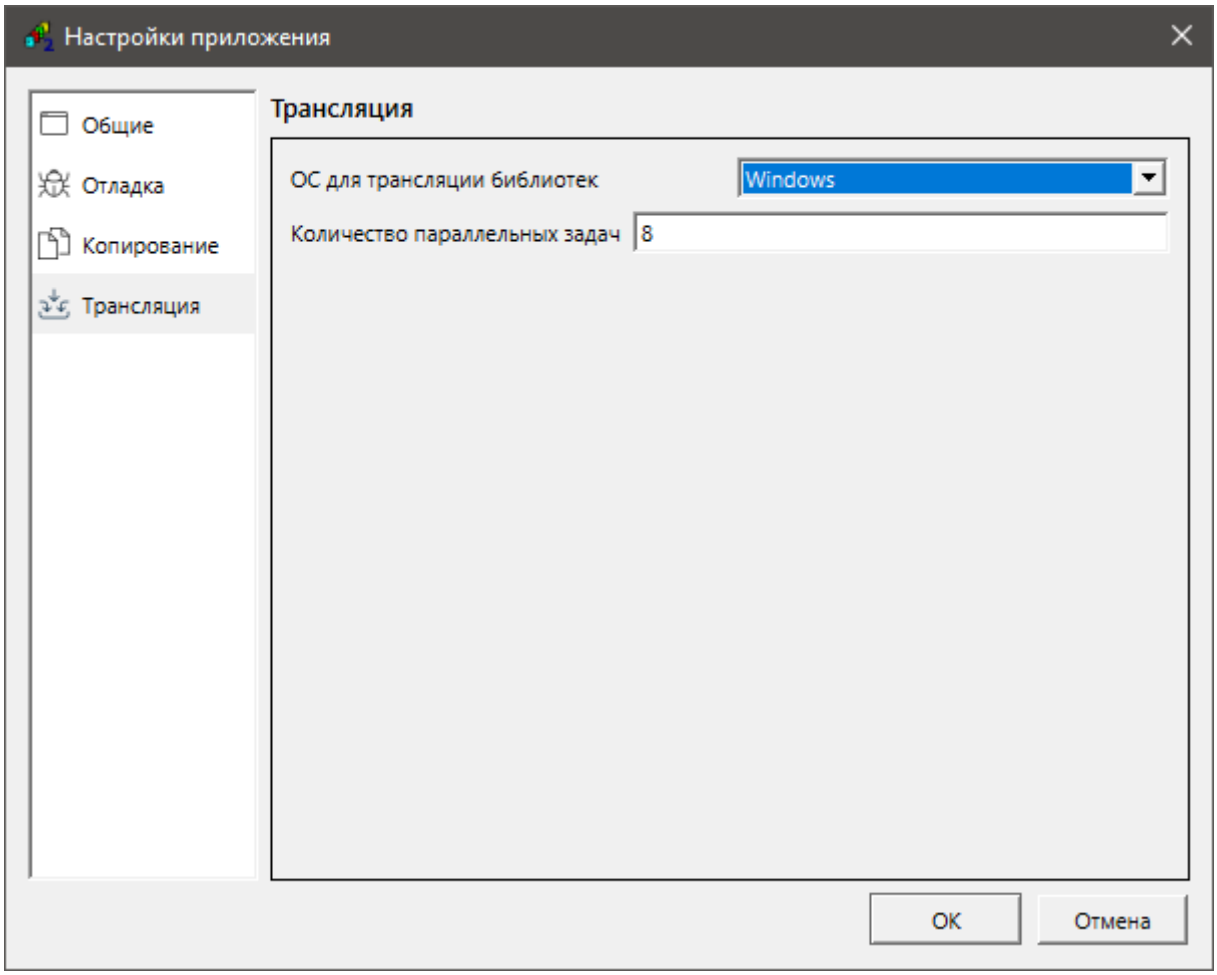


Рисунок 8.59 – Трансляция библиотеки

9. Выполнить ***Транслировать все*** на панели Инструменты окна библиотеки.
10. В папке библиотеки создастся папка ***build*** и в ней папка с файлами трансляции под выбранную ОС: ***Windows*** для Windows, ***Astra Linux SE 1.7*** для Astra Linux SE 1.7, ***Linux Овен прошивка 3.x*** для ОС ПЛК210.

Теперь новую библиотеку можно использовать в проектах.

8.2.1 Создание справки для библиотек

ПРИМЕЧАНИЕ
Возможность создания полноценной справки для библиотеки есть только при работе на ПК с ОС Windows. При работе на ПК с ОС Linux см. создание справки для отдельных блоков в разделе 8.1.4.

Чтобы создать справку для библиотеки следует:

1. Скопировать папку ***help*** с шаблонами (из папки ***Polygon2***) в папку библиотеки, для которой будет создаваться справка.
2. В папке ***help*** переименовать файлы в соответствии с именами библиотеки и функциональных блоков, например, ***my_lib*** и ***my_block***:
   - ***paControls.qhp*** в ***my_lib.qhp*** – список индивидуальных описаний блоков в справке библиотеки;
   - ***paControls.qhcp*** в ***my_lib.qhcp*** – указания для генератора справки;
   - ***paControls.bat*** в ***my_lib.bat*** – исполняемый **.bat**-файл создания бинарных файлов библиотеки для Полигон;
   - ***mode.html*** в ***my_block.html*** – описание блока.
3. Отредактировать переименованные файлы. Рекомендуется пользоваться редактором Notepad++ для удобства.
4. Редактирование файла ***my_lib.qhp*** на примере части содержимого из ***paControls.qhp***:
   - ***paControls*** заменяется на имя библиотеки, например, ***my_lib***;
   - Редактируется/удаляется и создается нужное количество разделов. Например, ***Функциональные блоки*** вместо ***Регуляторы***;
   - Редактируются описания блоков и ссылки на их файлы описания. Например, вместо ***Выдача данных с задержкой*** вставляется ***Вычисление линейной функции k*x + b***, вместо ***delayv.html*** вставляется ***my_block.html***;
   - В ключевых словах указывается ***my_block*** вместо ***delayv***, другие строки удаляются.

ПРИМЕЧАНИЕ
При редактировании следует сохранить строку, где в ключевом слове указано точное имя типа (в примере указаны ***my_block*** и ***C_ my_block***). Это требуется для вызова справки на конкретный блок по клавише **F1**.

В результате должно получиться следующее содержание ***my_lib.qhp***:

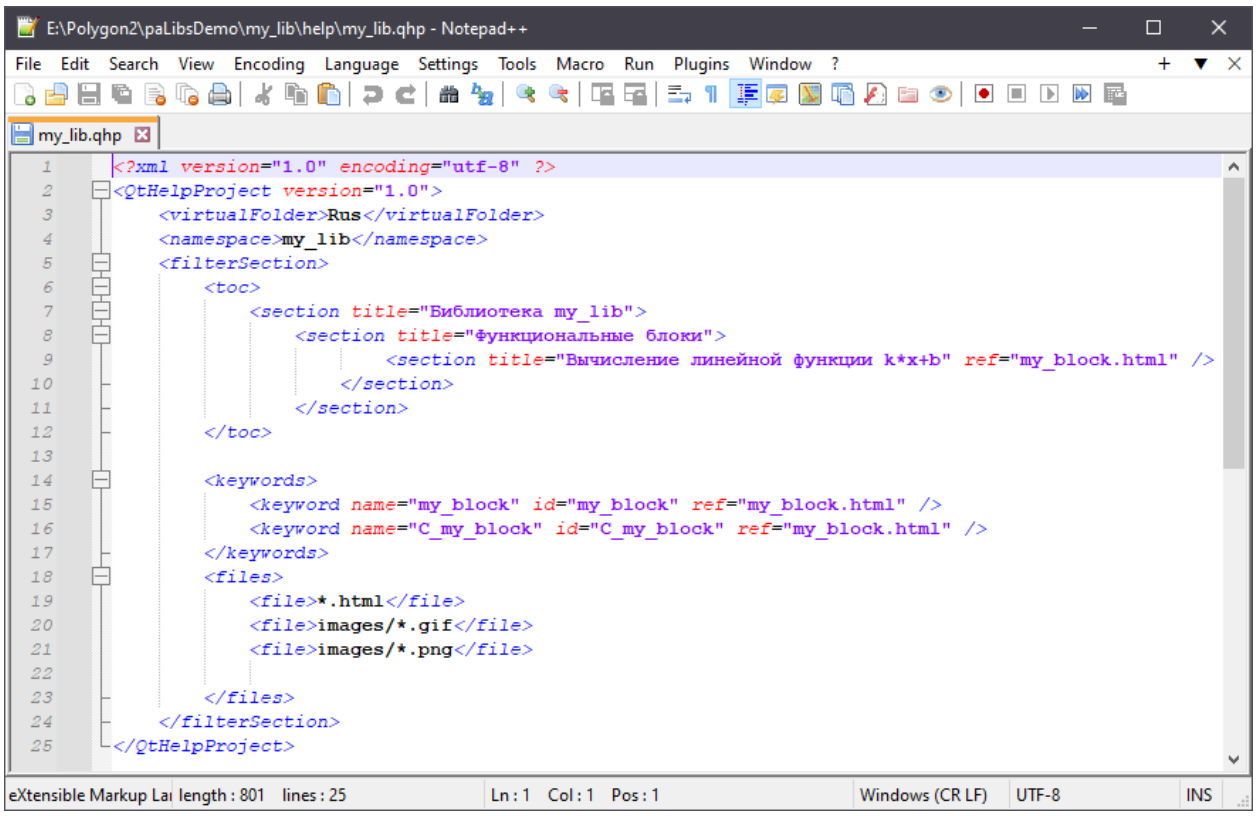


Рисунок 8.60 – Редактирование файла с расширением .qhp

5. Редактирование файла ***my_lib.qhcp***: заменить ***paControls*** на ***my_lib***.

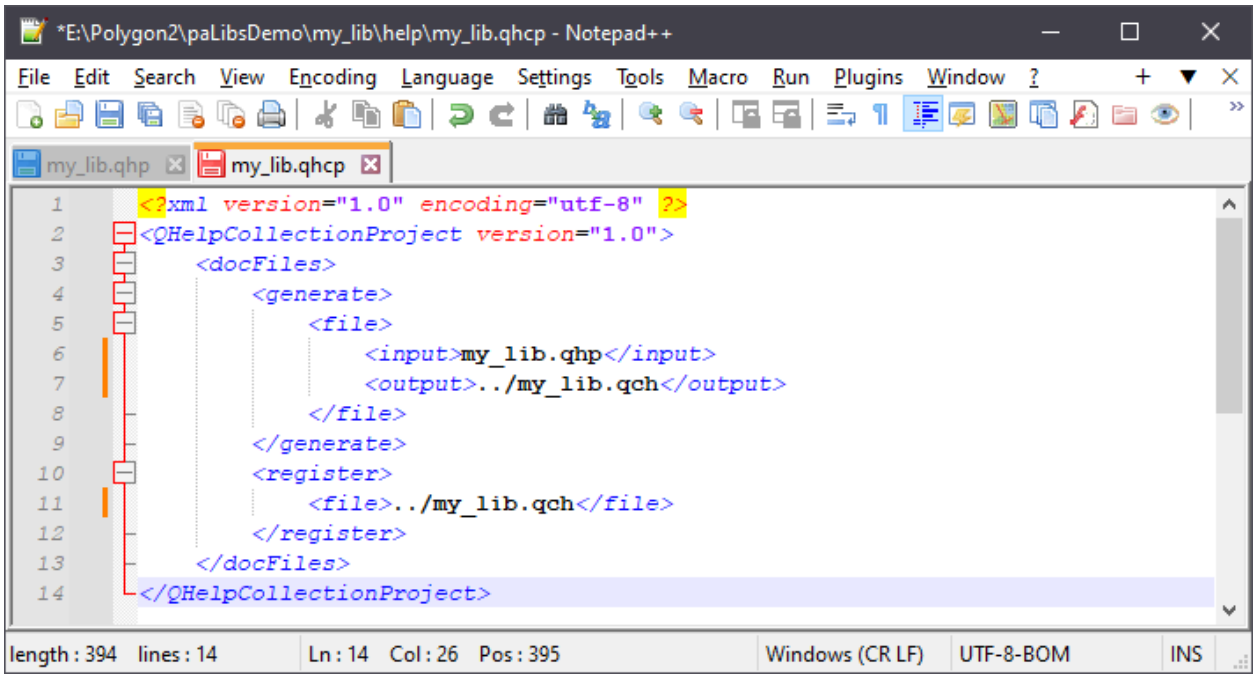


Рисунок 8.61 – Редактирование файла с расширением .qhcp

6. Редактирование файла ***my_lib.bat***:
   - Заменить ***paControls*** на ***my_lib***;

– Убедиться, что указан правильный путь к ***qhelpgenerator.exe***. Например, ***E:\Polygon2\qhelpgenerator.exe***.

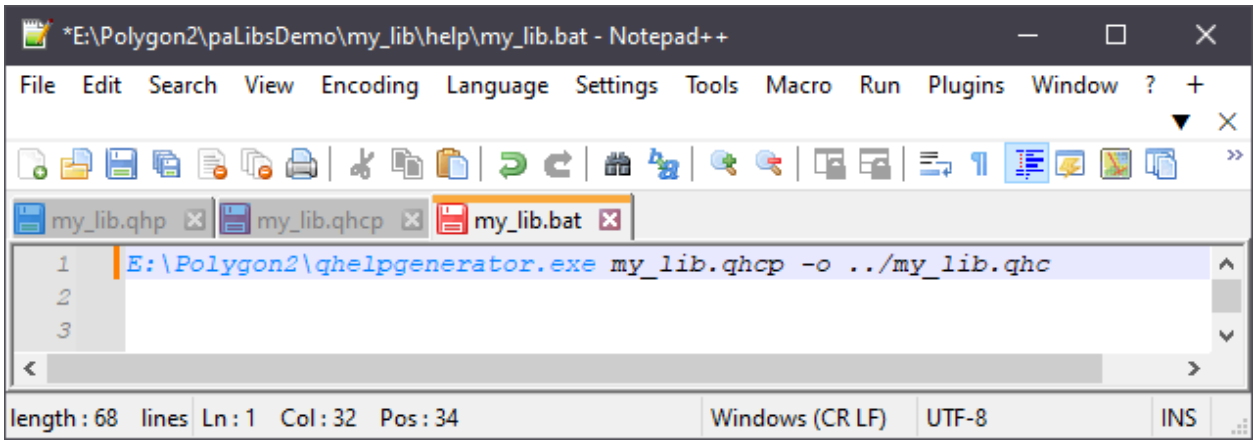


Рисунок 8.62 – Редактирование файла с расширением .bat

7. Редактирование файла ***my_lib.html***:
   – В файле описывается работа блока с форматированием текста с помощью HTML-кода;
   – Возможно задание используемых стилей (шрифт, тип, размер) в контейнерах, в которых в дальнейшем заключается текст;
   – Возможна блочная верстка страницы, подключение метаданных, форматирование, вставка изображений и т.д.
8. В папку ***images*** следует поместить картинки **.png**, **.gif** к описанию блока. Стандартно имена задаются именем блока и цифрой. В примере используется ***my_block.png***.
9. Запустить файл ***my_block.bat***, в результате чего создадутся файлы **.qch** и **.qhc**.
10. В папке библиотеки создадутся файлы справки: ***my_lib.qch*** и ***my_lib.qhc***.

Последние используются справкой Полигон для отображения описания блоков библиотеки. Они должны передаваться вместе с ней, когда исходные файлы в папке ***help*** остаются у разработчиков библиотеки.

Пример получившейся справки:

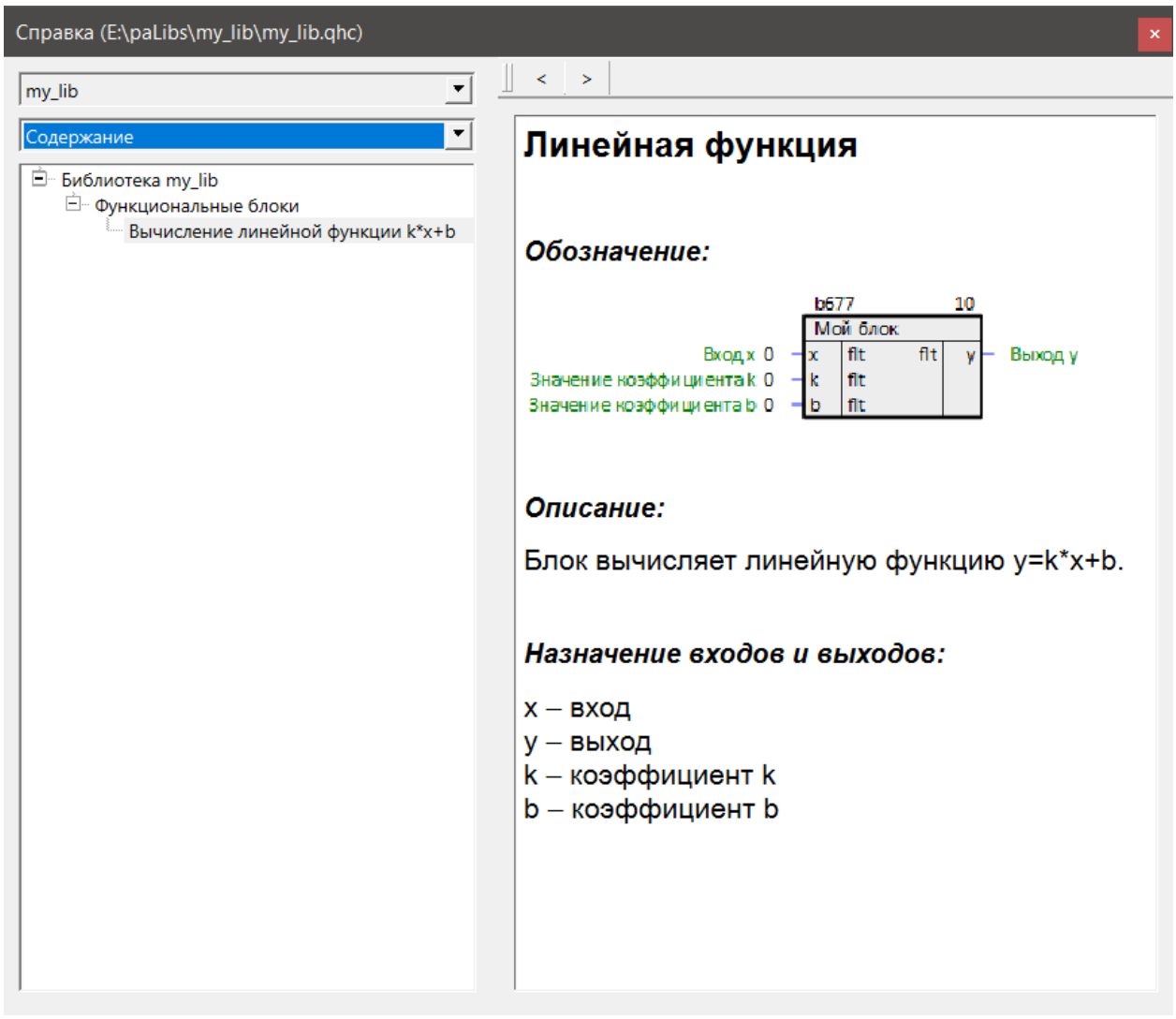


Рисунок 8.63 – Результат создания пользовательской справки

ПРИМЕЧАНИЕ

Возможные ошибки при запуске **.bat**-файла:

- Указан неверный путь к файлу генератора – в файле **.bat** следует указать прямой путь к ***qhelpgenerator.exe*** или создать переменную среды ***%DirQtInstall%***;
- Библиотека ***qminimal.dll*** неверной версии – следует обратиться к разработчику за верным файлом;
- Название проекта в ***<namespace>*** задано кириллицей: задать название латинскими символами;
- Файл **.qhc** не получилось обновить – сейчас файл используется приложением Полигон, следует закрыть его и повторить операцию;
- Справка создалась, можно самостоятельно в дереве справки выбрать описание блока, но оно не вызывается на блоке командой **F1** – в разделе ключевых слов файла **.qhp** не указано точное имя типа блока. Следует добавить эту строчку со ссылкой на описание блока.

8.3 Экспорт/импорт свойств из MS Excel

Полигон поддерживает выгрузку и загрузку свойств компонентов проекта в виде таблицы MS Excel с помощью команд ***Экспорт/Импорт*** на панели Инструменты. Это может быть полезно для быстрого массового редактирования свойств функциональных блоков и их входов/выходов.

С помощью команды ***Экспорт*** можно выгрузить свойства компонентов ***Модуля***, ***Места работы***, ***Программы***, ***Страницы***, ***Раздела***.

Команда ***Экспорт*** выбирается на панели ***Инструменты*** в дереве проекта, в этом случае предварительно следует выделить экспортируемый компонент проекта.

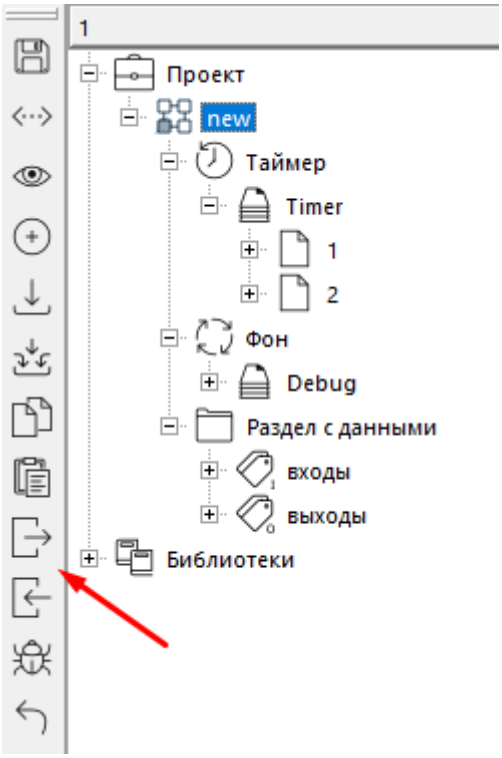


Рисунок 8.64 – Команды Экспорт/Импорт на панели Инструменты

Также можно выбрать команду ***Экспорт*** в контекстном меню нужного компонента, которое открывается при нажатии на него ПКМ.

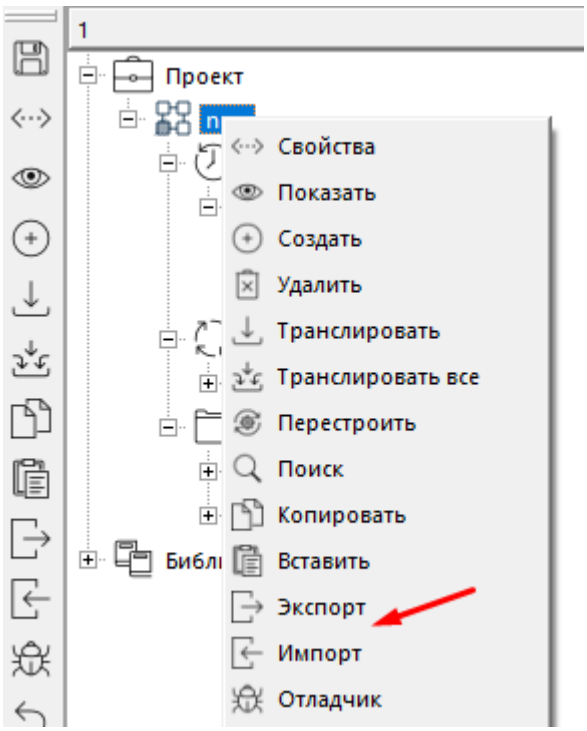


Рисунок 8.65 – Команды Экспорт/Импорт в контекстном меню

Аналогично можно экспортировать и импортировать свойства функциональных блоков и входов/выходов на открытой странице. Для этого следует нажать кнопку ***Эскпорт*** на панели ***Инструменты***.

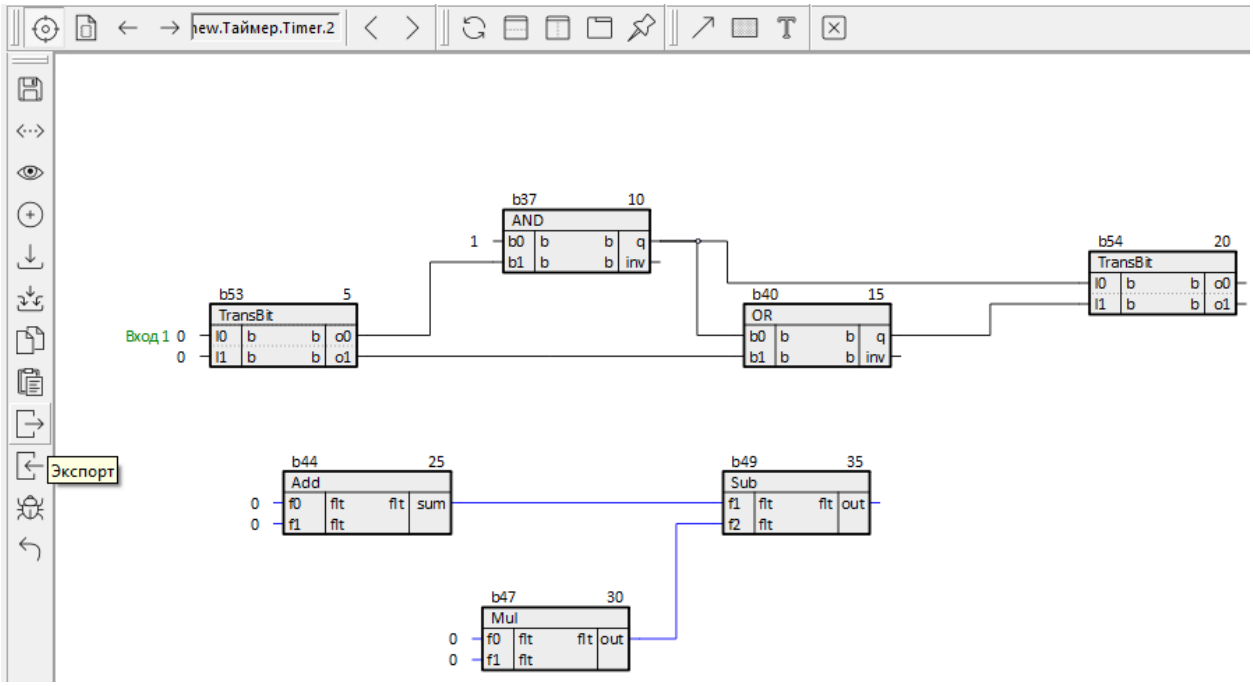


Рисунок 8.66 – Исходный вид страницы

После выполнения команды ***Экспорт*** откроется окно, в котором следует:

1. Установить в графе тип файла ***MS Excel (*.xlsx)*** – справа в окне отобразится предпросмотр таблицы свойств страницы, блоков и входов/выходов на этой странице.
2. Нажатием ПКМ на заголовке таблицы открыть контекстное меню, в котором следует установить флаги на необходимых для экспорта свойствах блоков и входов/выходов.

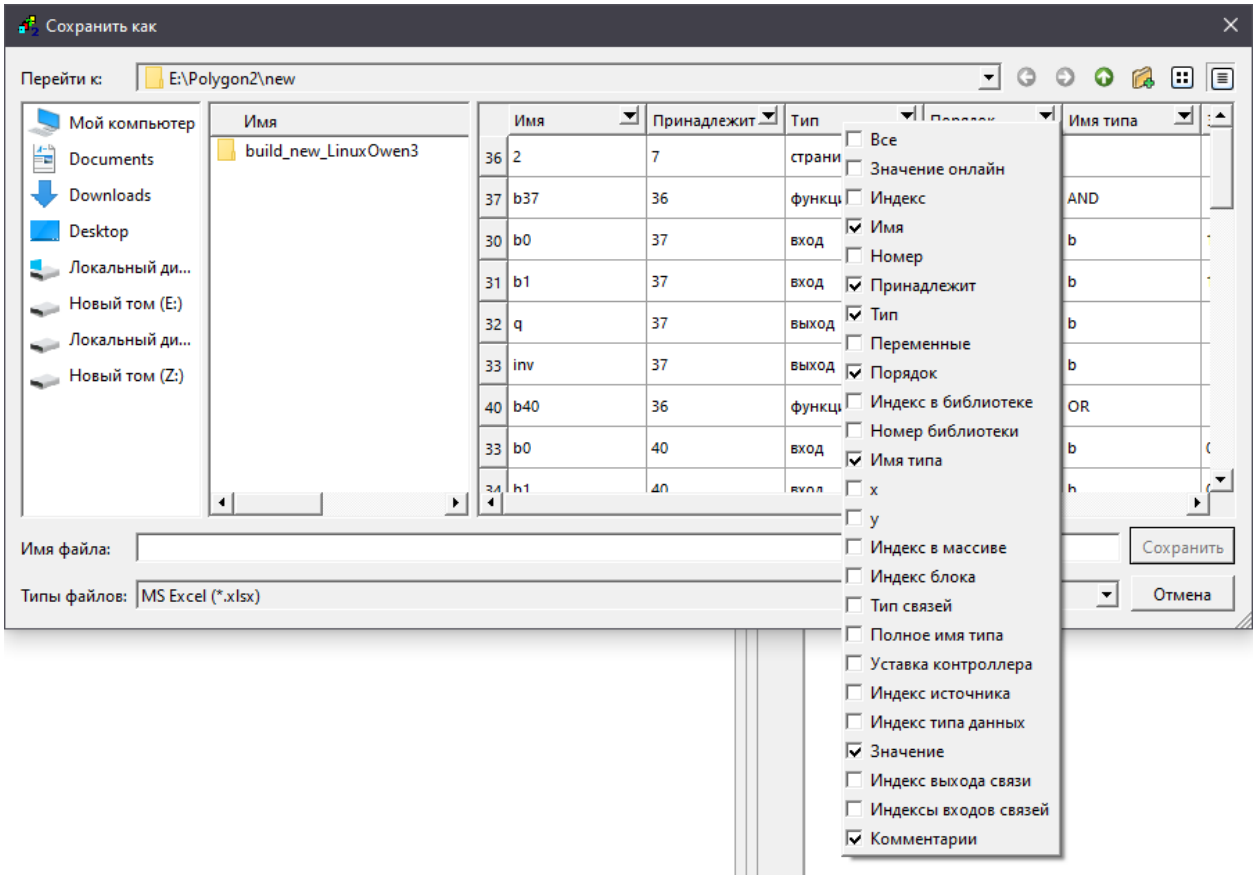


Рисунок 8.67 – Выбор экспортируемых свойств

3. Ввести имя файла.
4. Выбрать его расположение на диске.
5. Нажать ***Сохранить***.

Сохраненный файл следует открыть в программе MS Excel.

ПРИМЕЧАНИЕ
При работе на ПК с ОС Linux для редактирования таблицы можно использовать редактор ONLYOFFICE.

В таблице можно редактировать или назначать необходимые свойства (например, имена блоков, комментарии, значения у входов, порядки и т.д.).

	A	B	C	D	E	F	G	H	J
1	new	{3458b54c-f9b2-45c2-8c28-99d9dedd0e95}							
2	rows	columns							
3	33	6	Имя	Принадлежит	Порядок	Имя типа	Значение	Комментарии	
4	index	blocktype	name	parents	id	typename	value	comments	
5	36	7	2	7				**Страница**	
6	37	0	b37	36	10	AND			
7	30	1	b0	37	1	b	1		
8	31	1	b1	37	2	b	1		
9	32	2	q	37	1	b			
10	33	2	inv	37	2	b			
11	40	0	b40	36	15	OR			
12	33	1	b0	40	1	b	0		
13	34	1	b1	40	2	b	0		
14	36	2	q	40	1	b			
15	37	2	inv	40	2	b			
16	44	0	**Сумматор**	36	25	Add			
17	36	1	f0	44	1	flt	0	**Слагаемое 1**	
18	37	1	f1	44	2	flt	0	**Слагаемое 2**	
19	39	2	sum	44	1	flt			
20	47	0	b47	36	30	Mul			
21	39	1	f0	47	1	flt	**5**		
22	40	1	f1	47	2	flt	**25,5**		
23	41	2	out	47	1	flt			
24	49	0	b49	36	35	Sub			
25	43	1	f1	49	1	flt	0		
26	44	1	f2	49	2	flt	0		
27	43	2	out	49	1	flt		**Результат**	
28	53	0	b53	36	**1**	TransBit			
29	46	1	I0	53	1	b	0	Вход 1	
30	47	1	I1	53	2	b	**1**	**Вход 2**	
31	45	2	o0	53	1	b			
32	46	2	o1	53	2	b			
33	54	0	b54	36	20	TransBit			
34	48	1	I0	54	1	b	0		
35	49	1	I1	54	2	b	0		
36	47	2	o0	54	1	b			
37	48	2	o1	54	2	b			

Рисунок 8.68 – Свойства после редактирования в MS Excel

Отредактированный файл следует сохранить на диске в формате ***MS Excel (*.xlsx)***.

Чтобы импортировать новые свойства следует нажать кнопку ***Импорт*** на панели ***Инструменты*** и выбрать в открывшемся окне сохраненный файл.

Новые свойства применятся к странице, блокам и входам/выходам на этой странице.

ВНИМАНИЕ

Место свойств в проекте определяется однозначно уникальными свойствами ***Индекс*** (index) входов/выходов, блоков и т.д., поэтому импорт можно произвести в любом месте проекта, изменения применятся только к свойствам тех элементов, которые были экспортированы.

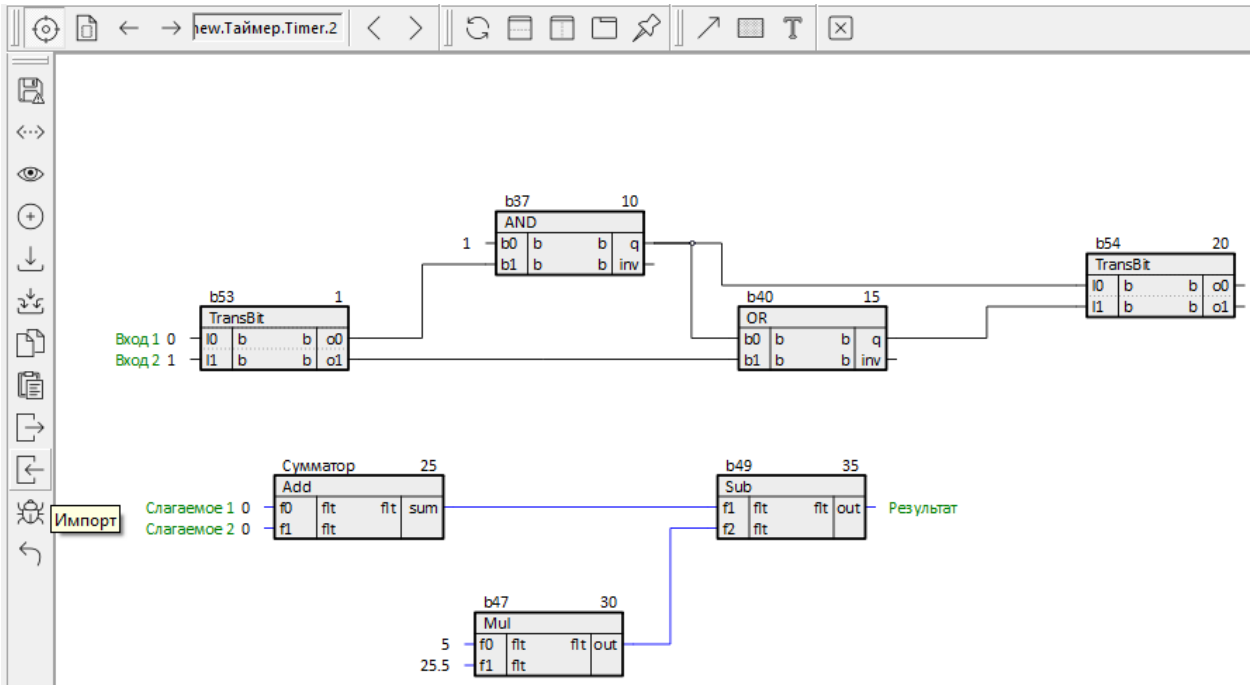


Рисунок 8.69 – Вид страницы после импорта свойств

8.4 Генерация из MS Excel

Полигон поддерживает генерацию программ из MS Excel с помощью команды ***Импорт***.

Генерация из MS Excel в Полигон позволяет:

- Добавлять программы, составные блоки и блоки из библиотек;
- Изменять значения и свойства входов и выходов блоков;
- Устанавливать связи между входами и выходами блоков.

ПРИМЕЧАНИЕ
При работе на ПК с ОС Linux для редактирования таблиц можно использовать редактор ONLYOFFICE.

Для генерации требуется **.xlsx**-файл определенного формата.

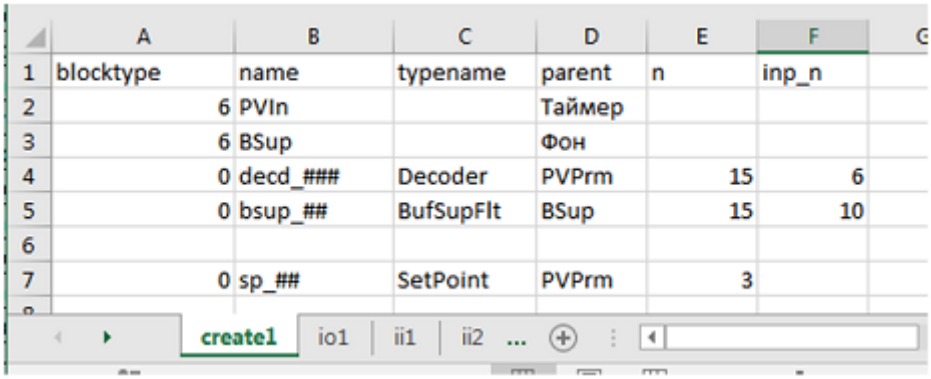
На модуле следует нажать ***Импорт***, указать файл и в качестве типа файла выбрать ***Генерация из MS Excel (*.xlsx)***. Полигон проверяет начало названия листа и выполняет соответствующий алгоритм импорта.

Листы файла **.xlsx**:

- ***create*** – листы данного типа предназначены для добавления программ и блоков;
- ***io*** – предназначен для изменения значений, комментариев и прочих свойств входов и выходов;
- ***ii*** – предназначен для изменения значений на группах входов;
- ***links*** – предназначен для проведения связей.

Если при импорте будет обнаружена ошибка, то изменения не применяются и происходит откат версии до начала импорта. Генерация оперирует с именами блоков, поэтому они должны быть уникальны. Листы считываются слева направо, поэтому вначале необходимо разместить листы типа ***create***. Процесс импорта сопровождается сообщениями в окне ***Прогресс*** и завершается выдачей информации о количестве добавляемых блоков, программ и т.п. Алгоритм импорта не выполняет удаление блоков, программ, свойств и связей.

В итоговом файле для импорта не должны содержаться формулы, все значения должны быть посчитаны.



	A	B	C	D	E	F	G
1	blocktype	name	typename	parent	n	inp_n	
2	6	PVIn		Таймер			
3	6	BSup		Фон			
4	0	decd_###	Decoder	PVPrm	15	6	
5	0	bsup_##	BufSupFlt	BSup	15	10	
6							
7	0	sp_##	SetPoint	PVPrm	3		

Рисунок 8.70 – Лист create. Добавление программ и блоков

Заголовок таблицы предопределен, при несоответствии импорт прекратится с выдачей предупреждения на указанный лист.

Таблица 8.2 – Заголовки таблицы create

Заголовок	Описание	Ограничения
blocktype	**6** – для программ, **0** – для блоков	Значения отличные от **0** и **6** остановят импорт
name	Если необходимо создать множество блоков используйте спецсимвол **«#»** в названии. Количество **#** определяет маску номера блока. **bsup_##** – **bsup_00**, **bsup_01**, … **decd_###** – **decd _000**, **decd _001**, …	Имя может состоять из букв латинского алфавита, цифр, символа **«_»** и спецсимвола **«#»**, другие символы не допускаются. **#** – разрешен только для блоков. Нельзя создать несколько программ одной строчкой. При использовании **#** ввод количества блоков в столбец ***n*** обязателен. **#** могут располагаться только в конце названия
typename	Имя блока из библиотеки или имя составного блока из корня проекта	Обязательное поле для блоков
parent	Имя родителя в дереве. Для программ – место работы, для блоков – имя программы	Обязательное поле для программ и блоков
n	Количество блоков	Поле считывается только при наличии **#** в названии

Продолжение таблицы 8.2

inp_n	Количество циклических входов/выходов блока	Только для блоков. Если блок не имеет циклических входов/выходов поле должно быть пустым

Алгоритм импорта выполняется построчно сверху вниз до обнаружения пустой ячейки. Столбцы, начиная с **6**, не анализируются, их можно использовать для комментирования.

Для программы: при наличии указанной программы в проекте (независимо от места работы), программа не добавляется. В противном случае проверяется наличие указанного места работы и добавляется программа.

Для блока: при наличии указанного количества блоков (независимо от типа блоков и программы), блоки не добавляются. В противном случае проверяется наличие указанной программы, и добавляются блоки, начиная с **0** номера. Если в программе присутствует часть блоков, новые блоки добавляются, начиная с последнего номера **+1**. Блоки вставляются друг за другом, если у блока много входов/выходов используется формат **A3**.

	A	B	C	D	E	F	G	H	I
1	type_io	block	name	value	comments	alias	prop_deadzone		
2	0	bsup_00	adr2	154	150				
3	1	decd_012	o4		Включить ПТ3	KKS_0043	0.03		
4	1	bsup_00	dan0		DW bsup_00.dan0				
5	1	bsup_00	dan1		DW bsup_00.dan1				
6	1	bsup_00	dan2		DW bsup_00.dan2				
7	1	bsup_00	dan3		DW bsup_00.dan3				

create1 | io1 | ii1 | ii2 | links1 | Лист ...

Рисунок 8.71 – Лист io. Изменение/добавление свойств входа/выхода

Заголовок таблицы ***io*** предопределен частично (первые три столбца), при несоответствии импорт прекратится с выдачей предупреждения на указанный лист.

Таблица 8.3 – Заголовки таблицы io

Заголовок	Описание	Ограничения
type_io	**0** – для входов, **1** – для выходов	Значения отличные от **0** и **1** остановят импорт
block	Имя блока	Если блок не найден, импорт прекращается
name	Имя входа/выхода	Если вход/выход у блока не найден, импорт прекращается

Остальные заголовки должны иметь название, совпадающее со свойствами Полигона. Если оставить ячейку пустой соответствующее свойство не будет добавлено/изменено.

Алгоритм импорта выполняется построчно сверху вниз до обнаружения пустой строки. Столбцы, начиная с пустого, не анализируются.

Также можно организовывать циклы и повторения в таком же формате, как и ***links***.

204	1	prm4_##	o#		comment1	alias1
205	1	prm3_00	o0		prm33_00.o0	
206			o1		prm33_00.o1	
207			o2		prm33_00.o2	
208			o3		prm33_00.o3	
209			o4		prm33_00.o4	
210			o5		prm33_00.o5	
211			o6		prm33_00.o6	
212			o7		prm33_00.o7	
213			o8		prm33_00.o8	
214			o9		prm33_00.o9	

Рисунок 8.72 – Пример организация циклов

В данном примере строка **204** приведет к тому что у всех блоков **prm4_00**, **prm4_01**, … на всех выходах типа **«o»** выставятся одинаковые комментарии **comment1** и алиас **alias1**.

Строки **206…214** будут использовать имя блока с предыдущей строки.

	A	B	C	D	E	F	G	H
1	block	array	adr	ini	min	max	typ	
2	bsup_00	1	150	-25	-50	50	II	
3	bsup_01	3	151	-26	-51	49	AI	
4								
5								
6								
7								

create1 | io1 | ii1 | ii2 | links1 | Лист ...

Рисунок 8.73 – Лист ii. Изменение значений на группах входов

Заголовок таблицы ***ii*** предопределен частично (первые два столбца), при несоответствии импорт прекратится с выдачей предупреждения на указанный лист.

Таблица 8.4 – Заголовки таблицы ii

Заголовок	Описание	Ограничения
block	Имя блока	Если блок не найден, импорт прекращается
array	Номер входа	-

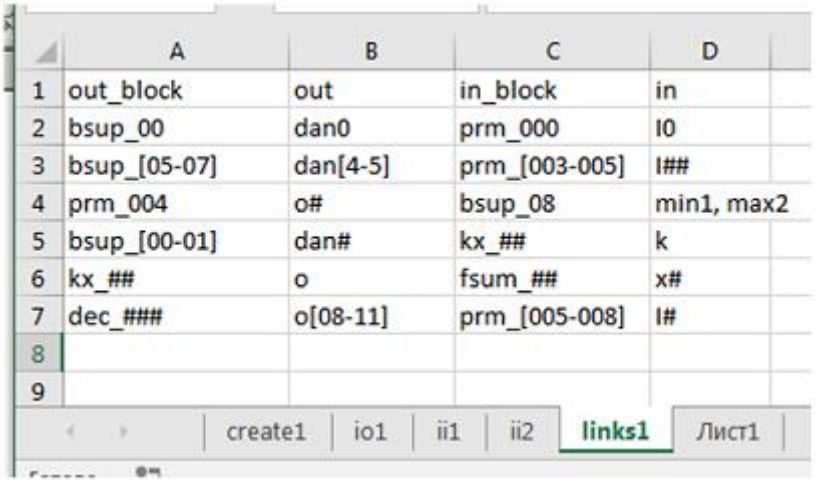
Остальные заголовки должны совпадать с началом имени входа блока. В примере, указанном на рисунке выше, будут выполнены следующие действия:

- **bsup_00.adr1 = 150**, **bsup_00.ini1 = -25**, **bsup_00.min1 = -50**, **bsup_00.max1 = 50**, **bsup_00.typ1 = II**;
- **bsup_01.adr3 = 151**, **bsup_01.ini3 = -26**, **bsup_01.min3 = -51**, **bsup_01.max3 = 49**, **bsup_01.typ3 = AI**.

Алгоритм работает не только на циклических входах, главное, чтобы ***<имя заголовка + номер входа>*** присутствовал у блока.

Если оставить ячейку пустой соответствующее значение не будет изменено.

Алгоритм импорта выполняется построчно сверху вниз до обнаружения пустой ячейки. Столбцы, начиная с пустого, не анализируются.



	A	B	C	D
1	out_block	out	in_block	in
2	bsup_00	dan0	prm_000	I0
3	bsup_[05-07]	dan[4-5]	prm_[003-005]	I##
4	prm_004	o#	bsup_08	min1, max2
5	bsup_[00-01]	dan#	kx_##	k
6	kx_##	o	fsum_##	x#
7	dec_###	o[08-11]	prm_[005-008]	I#
8				
9				

create1 | io1 | ii1 | ii2 | links1 | Лист1

Рисунок 8.74 – Лист links. Проведение связей

Заголовок таблицы ***links*** предопределен, при несоответствии импорт прекратится с выдачей предупреждения на указанный лист.

Таблица 8.5 – Заголовки таблицы links

Заголовок	Описание	Ограничения
out_block	Имя блока с выходом	Для указания циклических связей используются символы **#**, **[,]**
out	Имя выхода	-
in_block	Имя блока с входом	-
in	Имя входа	-

Для проведения простой единичной связи укажите блоки и входы/выходы.

Алгоритм импорта выполняется построчно сверху вниз до обнаружения пустой строки. Столбцы, начиная с **5**, не анализируются.

ВНИМАНИЕ
Обязательным условием использования циклических связей является непрерывность имен блоков. Например, если существуют **bsup_00** и **bsup_03**, обязаны существовать **bsup_01**, **bsup_02**, иначе импорт завершится ошибкой. А также обязательна нумерация с **0**, т.е. если имеется три блока **bsup_01**, **bsup_02**, **bsup_03**, к ним не получится обратится **bsup_##**.

Для задания циклических связей используются символы **«#»**, **«,»**, **«[-]»**. Возможны следующие типы циклов.

Таблица 8.6 – Типы циклов

Цикл	Формат	Описание	Ограничения
Определенный цикл	**bsup_[05-07] dan[4-8] min[00-14]**	В квадратных скобках задается начальный и конечный номера блоков через **«-»**	Для блока: Номер необходимо задавать с учетом лидирующих нулей. Пример – для блоков **bsup_1**, **bsup_2** указывается **bsup_[1-2]**, для блоков **bsup_09**, **bsup_11** указывается **bsup_[09-11]** Для входа/выхода: У входов/выходов лидирующих нулей нет, но необходимо чтобы начальный и конечный номер имели одинаковое количество цифр. Пример: для выходов **dan4**, **dan5** указывается **dan[4-5]**, для выходов **dan9**, **dan10** указывается **dan[09-10]**
Перечисление	**dan1**, **Ту**, **Кр**	Имена входов/выходов задаются через запятую. Пробелы игнорируются	Нельзя использовать для блоков, только для входов/выходов
Неопределенный цикл	**bsup_## dan#**	В качестве начального берется нулевой элемент. Конечный элемент определяется по ходу	Для блока: Количество **#** определяет маску номера блока **bsup_##** – **bsup_00**, **bsup_01**,… **decd_###** – **decd _000**, **decd _001**,… Для входа/выхода: Количество **#** не имеет значения
Повторение	Пустая ячейка	Если оставить поле **out_block** или **in_block** пустым, то используется информация с прошлой строки с сохранением позиции в цикле	-

Примеры задания циклических связей можно посмотреть в разделе ***Генерация из MS Excel*** в справке среды.

8.5 Создание серверного многопользовательского проекта

Многопользовательский проект может редактироваться несколькими разработчиками одновременно. В этом случае проект представляет собой базу данных PostgreSQL. Клиентом к серверной базе данных выступает среда разработки Полигон.

Для создания нового многопользовательского проекта в среде разработки следует:

1. Выполнить команду ***Создать*** в окне ***Проекты***.

2. В списке типа проекта выбрать ***Полигон 2 многопользовательский (*.pg2)***.

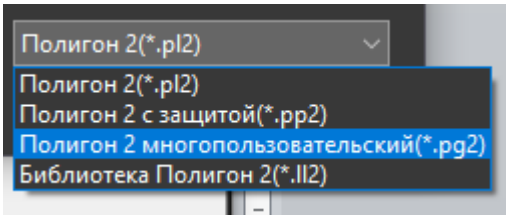


Рисунок 8.75 – Создание многопользовательского проекта

3. Определить местоположение файла проекта на диске. Рекомендуется каждый проект размещать в отдельной папке.

ВНИМАНИЕ
Полный путь к файлу проекта не должен содержать символов кириллицы.

4. Задать имя проекта. Имя проекта также определяет имя базы данных на сервере PostgreSQL.

ВНИМАНИЕ
Имя проекта должно содержать только символы латинского алфавита, цифры и **«_»**, начинаться с буквы.

5. В открывшемся окне ввести IP адрес или имя компьютера и порт, на котором работает сервер PostgreSQL.

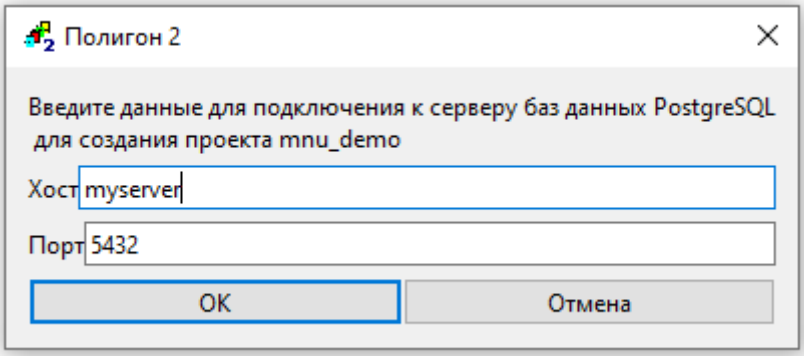


Рисунок 8.76 – Настройка подключения к серверу баз данных PostgreSQL

6. Ввести имя пользователя и пароль.

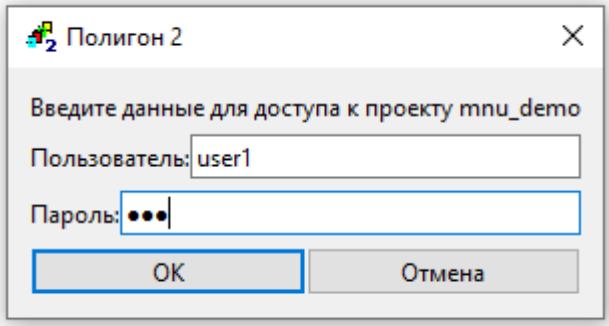


Рисунок 8.77 – Настройка подключения к серверу баз данных PostgreSQL

Если база данных с таким именем уже существует, то будет установлено соединение, если нет, то будет предложено ее создать.

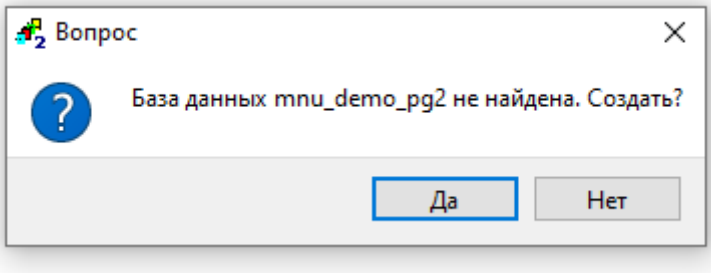


Рисунок 8.78 – Создание базы данных

Файл проекта с расширением **.pg2** можно скопировать к другим пользователям и открыть в среде разработки. Для этого следует:

1. Выполнить команду ***Открыть*** в меню ***Проекты***.
2. В списке типа проекта выберать ***Полигон2 многопользовательский (*.pg2)***.
3. Найти на диске файл типа **.pg2** и нажать ***Открыть***.
4. Ввести имя пользователя и пароль.

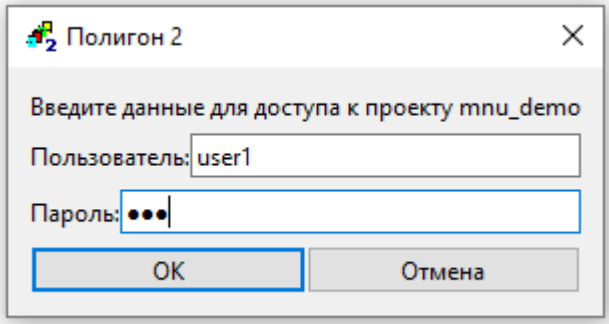


Рисунок 8.79 – Подключение к серверу баз данных PostgreSQL

Для корректной работы диагностики и логирования необходимо на каждой клиентской машине использовать разных пользователей для подключения к проекту.

Рекомендуется организовать работу над проектом таким образом, чтобы пользователи редактировали разные ***Программы*** проекта. Для этого следует создать структуру проекта, состоящую из ***Мест работы*** и ***Программ*** одним пользователем и затем привлечь к работе остальных.

Когда один пользователь вносит изменения в проект, остальные этих изменений не видят, пока он не сохранит проект на сервер. При этом остальным пользователям видны узлы проекта, в которых изменения в данный момент происходят, они подсвечиваются красным фоном в представлении ***Дерево***. Для пользователя, который изменения вносит, они обозначаются в ***Дереве*** курсивом. При наведении курсора мыши на подсвеченный узел ***Дерева*** можно увидеть, какой именно пользователь его редактирует.

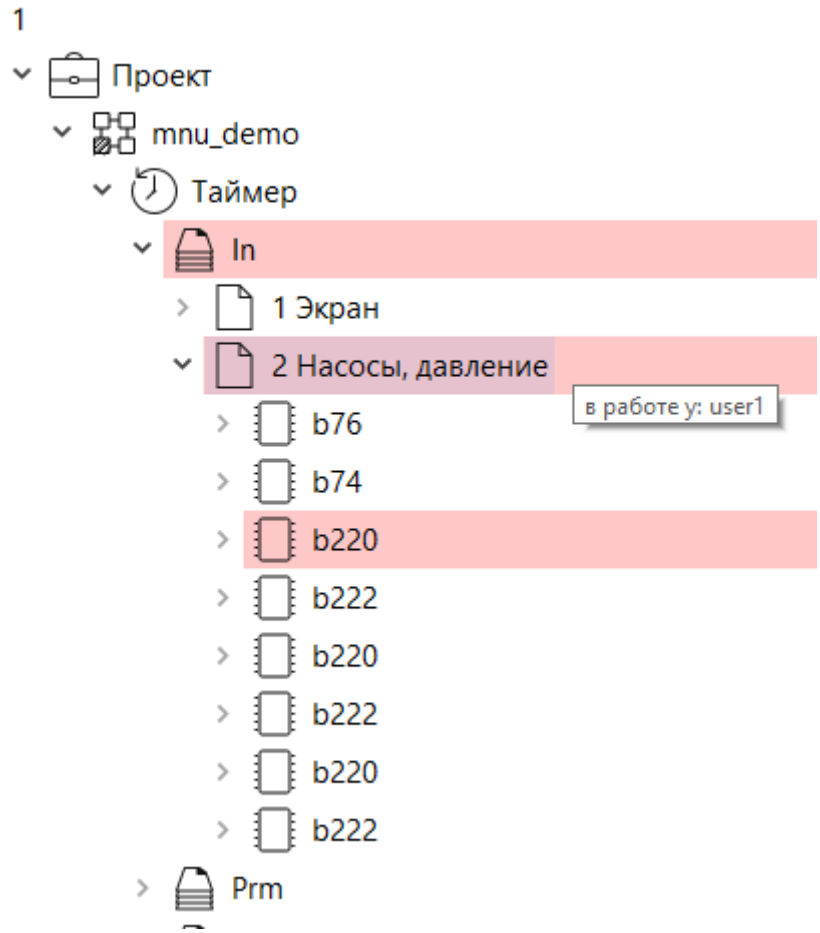


Рисунок 8.80 – Редактирование многопользовательского проекта

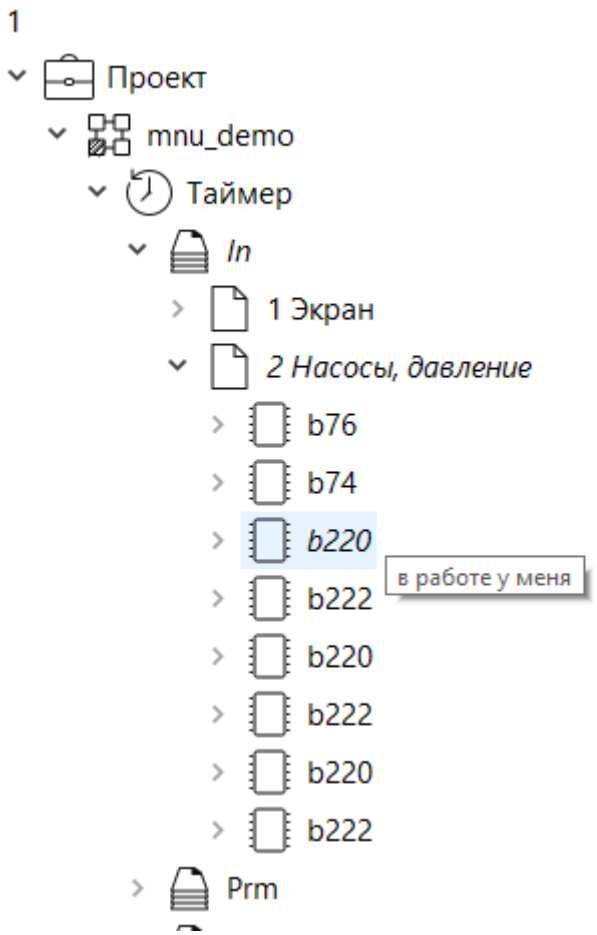


Рисунок 8.81 – Редактирование многопользовательского проекта

Среда разработки не сохраняет изменения проекта на сервер автоматически, однако если несохраненные изменения есть, то каждые **5 минут** выдается предложение сохранить проект. Время сохранения можно изменить/отключить в меню Экран/Настройки.

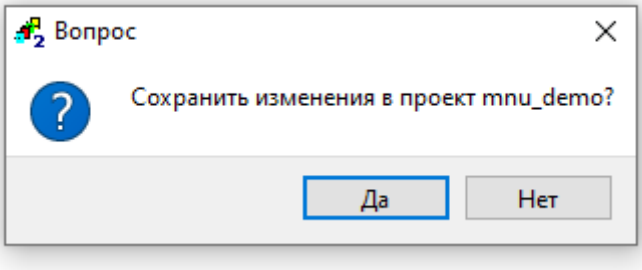


Рисунок 8.82 – Предложение о сохранении проекта

При этом кнопка ***Сохранить*** в панели инструментов показывает, что есть необходимость сохранить изменения. Последние **60** действий можно отменить командой ***Отмена*** или комбинацией клавиш **Ctrl + z** пока проект не сохранен на сервер. При переполнении очереди изменений будет предложено ее очистить или не выполнять последнюю операцию. После сохранения отмена изменений невозможна, и они становятся видны остальным пользователям.

После каждого сохранения проекта на локальной машине также создается копия проекта в виде ***sql dump*** для того, чтобы иметь возможность при необходимости восстановить определенную версию проекта. Файл с копией имеет такое же имя как основной проект плюс суффикс, обозначающий версию проекта на момент создания (например, файл ***mnu_demo_27.pg2*** содержит копию проекта ***mnu_demo*** версии **27**). Для восстановления копии необходимо открыть соответствующий файл проекта в среде разработки, тогда проект будет развернут на сервер в новую базу данных (например, ***mnu_demo_27_pg2***).

Рекомендуется не редактировать узлы проекта, подсвеченные в дереве красным фоном, то есть те, в которых в данный момент вносят изменения другие пользователи. Однако строгого запрета нет и некоторые изменения возможны. Например, можно создавать новые программы, страницы и функциональные блоки. Конфликты редактирования могут возникнуть, если попытаться изменить свойство компонента, которое было изменено другим пользователем и изменения еще не сохранены. В случае конфликта последняя попытка будет отменена с сообщением о блокировке другим пользователем.

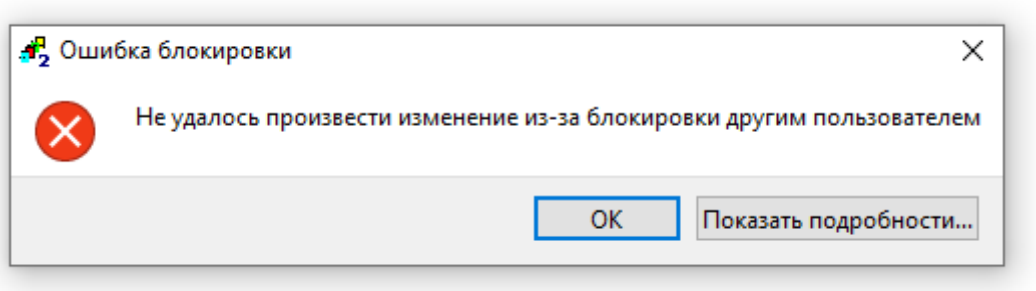


Рисунок 8.83 – Ошибка блокировки

Массовые операции, такие как импорт, генерация из MS Excel и копирование больших частей проекта рекомендуется проводить в момент минимальной активности остальных пользователей.

9 Защита проекта

9.1 Шифрование однопользовательского проекта

Проект в среде разработки Полигон – это база данных Sqlite. Защитить проект можно с помощью создания зашифрованной версии. Для этого следует:

1. В системном окне ***Проекты*** выделить интересующий проект и нажать кнопку ***Сохранить как…***.
2. В появившемся окне выбрать расширение ***Полигон 2 с защитой(*.pp2)***.
3. Нажать ***Сохранить***.

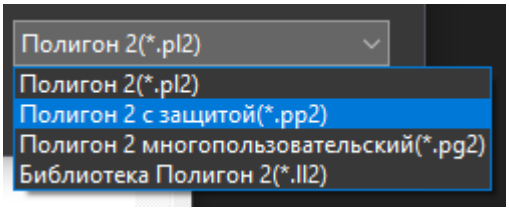


Рисунок 9.1 – Сохранение защищенного проекта

В окне ***Проекты*** отобразится защищенная копия проекта.

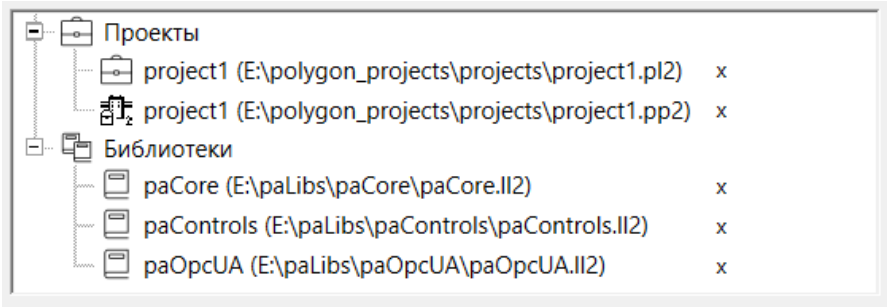


Рисунок 9.2 – Защищенный проект в окне Проекты

9.2 Пароли на составных блоках

В среде имеется возможность установить пароль на составном блоке для защиты его содержимого.

Для установки пароля следует:

1. Выбрать составной блок в дереве проекта.
2. Кликнуть ПКМ и выбрать в контекстном меню ***Установить пароль***.

ПРИМЕЧАНИЕ
Защита содержимого составного блока при помощи пароля эффективна только тогда, когда файл проекта хранится в защищенном формате.

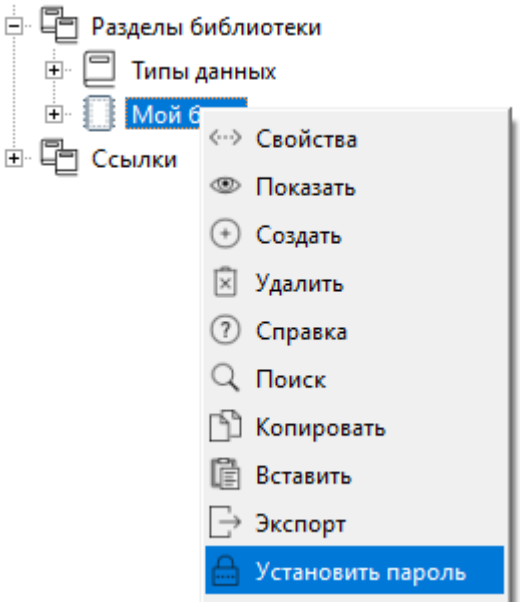


Рисунок 9.3 – Задание пароля составного блока

3. Задать пароль в открывшемся окне – на иконке блока в дереве отобразится знак «замка».

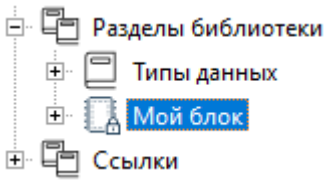


Рисунок 9.4 – Составной блок с паролем

Если на составном блоке установлен пароль, то у данного блока не удастся открыть его внутреннее содержание.

Для снятия пароля необходимо оставить пустым поле ввода.

9.3 Пароль отладчика. Управление пользователями

Для ограничения возможности доступа к проекту в свойствах модуля следует задать свойство ***Пароль admin***.

Чтобы изменение ***Пароль admin*** вступило в силу, следует перетранслировать проект.

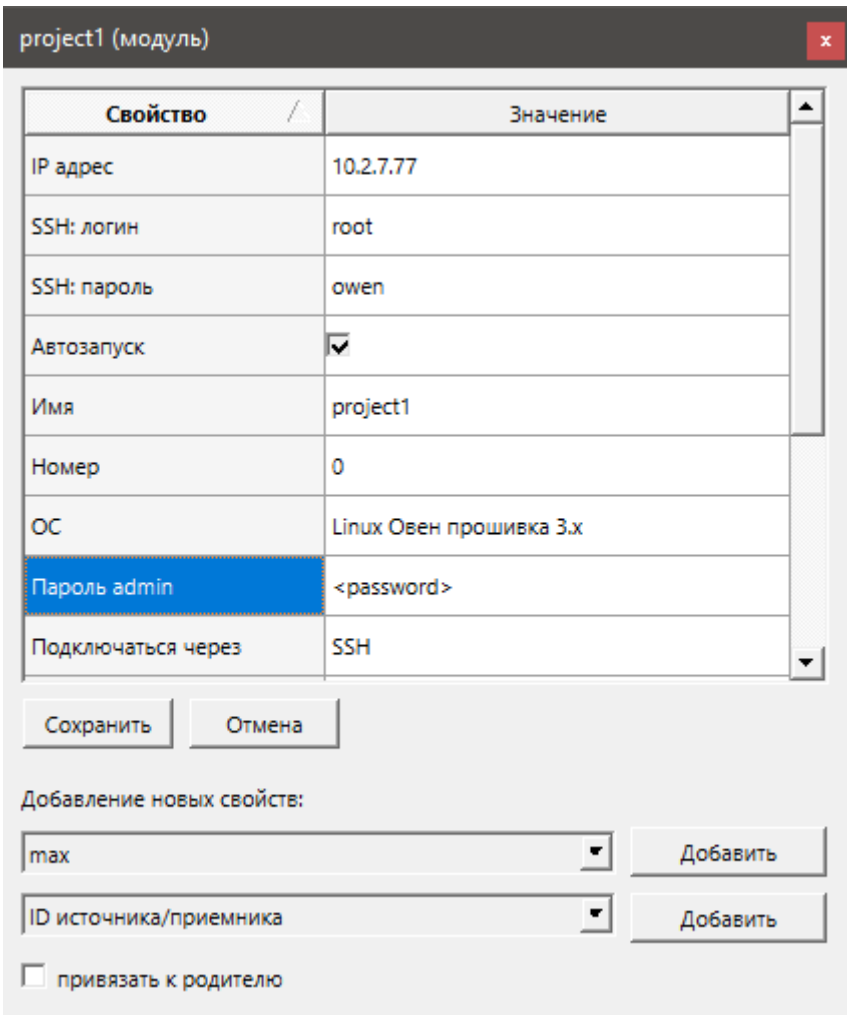


Рисунок 9.5 – Свойство Пароль admin

Если установлен ***Пароль admin***, то при первом запуске модуля будет выводиться окно с запросом пароля.

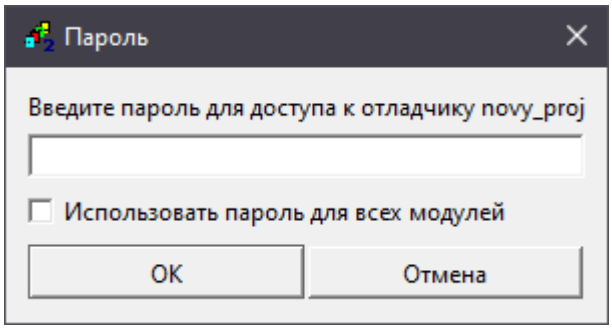


Рисунок 9.6 – Окно ввода пароля для доступа к отладчику

Окно с запросом пароля отладчика появляется только **1 раз**. В случае ввода неверного пароля запуск отладчика будет выдавать ошибку, а окно ввода пароля уже не появится.

Чтобы ввести корректный пароль, следует щелкнуть по модулю в дереве ПКМ и выбрать команду ***Задать пароль***, ввести пароль в появившемся окне.

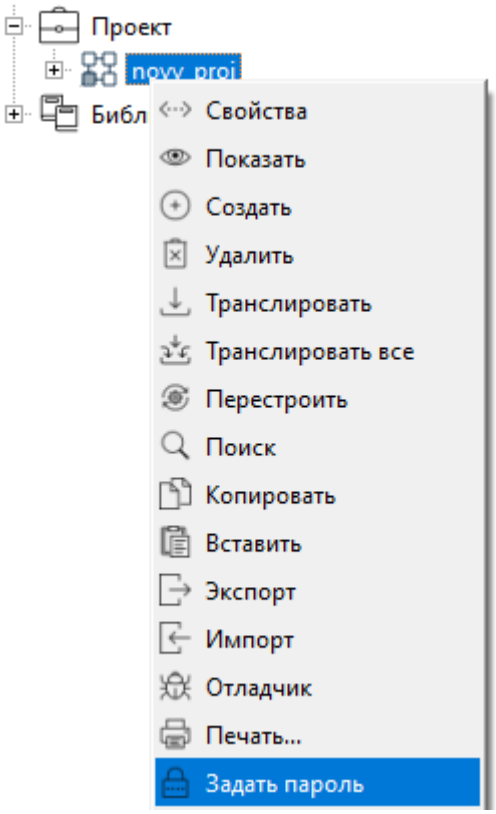


Рисунок 9.7 – Задать пароль

Также в среде есть возможность установить пароли для пользователей ***user1***…***user7***. Итого 8 прав доступа – 1 ***admin*** и 7 ***user***. Их можно использовать для ограничения доступа к данным проекта клиентам по протоколу OPC UA. См. подробнее в документе Обмен с верхним уровнем. Библиотека paOpcUA.

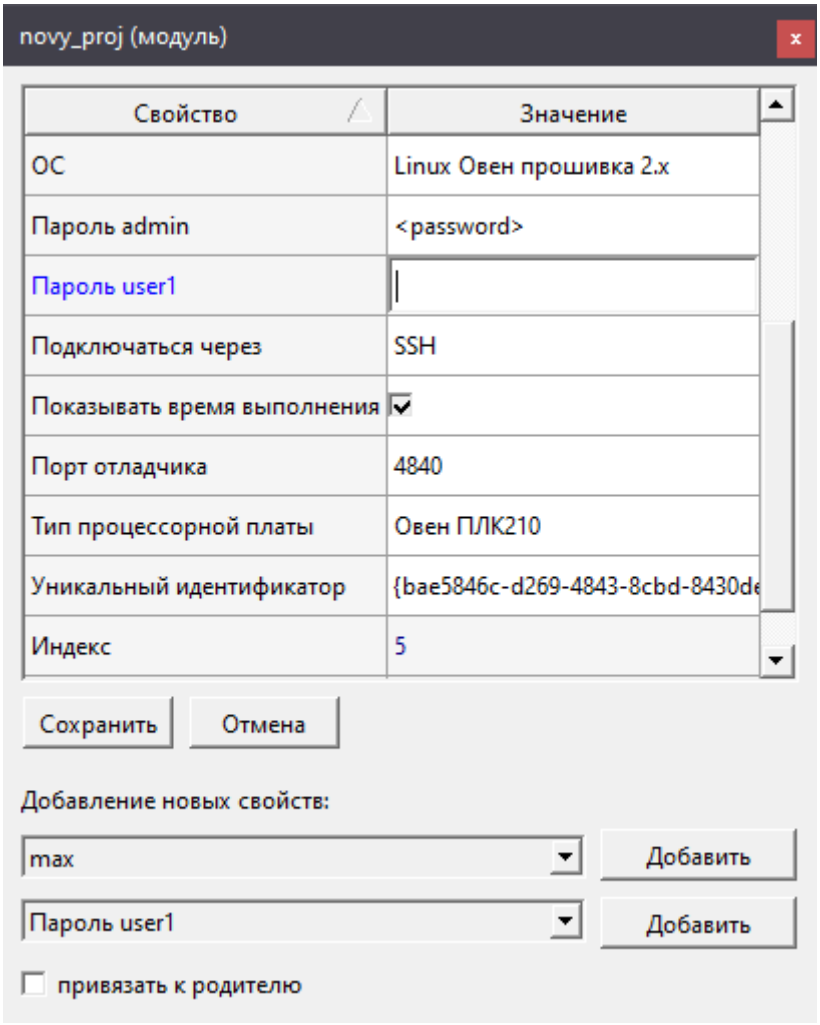


Рисунок 9.8 – Задание паролей пользователей

10 Основная библиотека проекта

В данном разделе приведено описание основных блоков обязательной в проекте библиотеки ***paCore***. Описание соответствует версии библиотеки – **979** и выше.

Об установке библиотек см. в разделе 1.4.2.

10.1 paCore

10.1.1 Работа с типами данных

10.1.1.1 Типы данных

В библиотеке ***paCore*** используются следующие типы данных:

Таблица 10.1 – Типы данных paCore

Обозначение в среде	Название типа	Описание	Размер в памяти	Диапазон значений	Аналоги названий
chr, **i8**	Байт	Целое значение со знаком. Signed 8-bit integer	1 байт	-128…127	sint, int8, TINYINT, INT1
uch, **ui8**, **reg**	Байт без знака	Целое значение без знака. Unsigned 8-bit integer	1 байт	0…255	Byte
int, **i16**	Целое	Целое значение со знаком. Signed 16-bit integer	2 байта	-32768…32767	INT, SMALLINT, INT2
uns, **u16**, **rg2**	Целое без знака	Целое значение без знака. Unsigned 16-bit integer	2 байта	0…65535	WORD
lng, **i32**, **l32**	Длинное целое	Целое значение со знаком. Signed 32-bit integer	4 байта	-2147483648… 2147483647	DINT, INT4
ulg, **u32**, **rg4**, **ul32**	Длинное целое без знака	Целое значение без знака. Unsigned 32-bit integer	4 байта	0… 4294967295	UDINT
i64	Двойное длинное целое	Двойное целое значение со знаком. Signed 64-bit integer	8 байт	-9223372036854775808…9223372036854775807	-
u64	Двойное длинное целое без знака	Двойное целое значение без знака. Unsigned 64-bit integer	8 байт	0…18446744073709551615	-

Продолжение таблицы 10.1

flt	Вещественное	Вещественное значение со знаком. Данные в формате с плавающей запятой	4 байта	-3.4E+38…1.18E-38 и 1.18E-38…3.4E+38	Float, Real
double	Вещественное двойное	Вещественное значение со знаком. Данные в формате с плавающей запятой	8 байта	-1.7E+308…1.18E-308 и 1.18E-308…1.7E+308	Double
b, **bool**, **boolean**, **bit**	Логический	Принимает значение 0 или 1	1 бит	0,1	Bit
s40, **s50**, **s100**, **str**	Строковый	Строки символов	-*	-	String

***** Максимальный размер строки определяет количество резервируемой памяти, размер задан в квадратных скобках, тип **str** ограничений на размер не имеет

Кроме представленных в таблице типов данных используются массивы и перечисления.

Массив – массив данных определенного типа:

- ***i32a*** - массив типа int32;
- ***fa*** - массив данных типа float.

Перечисление – список констант, определяемый конкретным входом. Входу, имеющему данный тип, невозможно задать значение отличное от заданных в выпадающем списке. Обозначение: ***enm***.

Очередь битов – позволяет накапливать очередь битовых значений в таймерном потоке. Обозначение: ***qb***. Подробнее в разделе 10.1.14.

Другие типы, кроме вышеперечисленных, находящиеся в разделе ***Типы данных***, являются **системными**.

При проведении связей между входами и выходами блоков среда разработки автоматически проверяет соответствие типов данных.

Цвет связей отображает тип данных входа:

- ***черный*** – булевое;
- ***красный*** – целое;
- ***синий*** – вещественное;
- ***серый*** – строковое.

Преобразование типов также можно осуществить вручную, используя блоки раздела Преобразователи типов.

Преобразовать можно только совместимые типы, иначе проведение связи запрещается, что отображается в соответствующем всплывающем окне предупреждения.

Если вход/выход блока имеет системный тип данных, то создать связь можно **только** с выходом/входом **того же** типа.

10.1.1.2 Преобразователи типов

Преобразование типов можно осуществить вручную, используя блоки ***Flt_Un***, ***Flt_Ul***, ***Uc_flt***, ***Us_Flt*** и др. из раздела **Преобразователи типов**.

Блок преобразует значение одного типа данных в другой.

В большинстве случаев преобразование типов осуществляется на этапе трансляции, подходящий блок добавляется средой разработки автоматически.

Алгоритм:

```
out = in
```

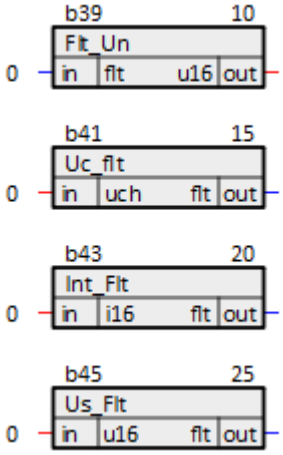


Рисунок 10.1 – Преобразователи типов

10.1.1.3 Константы

Блоки ***Константа*** повторяют на выходе значение входа – ***Real***, ***Integer***, ***Reg2*** и др. из раздела **Преобразователи типов**. Блок инициализируется константой на входе, не нулем.

Инициализация:

```
y = fi
```

Алгоритм:

```
y = fi
```

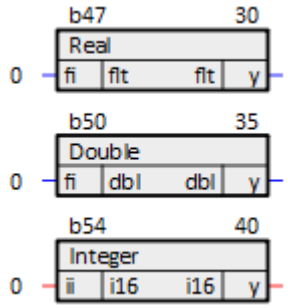


Рисунок 10.2 – Константы

10.1.1.4 Терминальные блоки

Терминальные блоки повторяют на выходе значение входа – ***TransFlt***, ***TransInt***, ***TransBit*** и др. Данные блоки удобно использовать в качестве сбора входных и выходных параметров на одной странице программы. Блок инициализируется константой на входе, не нулем.

Назначение входов и выходов:

I0, **I1**, **…**, **In** – входные значения;
o0, **o1**, **…**, **on** – выходные значения.

Инициализация:

```
oi = Ii, где I = 0…n
```

Алгоритм:

```
oi = Ii, где I = 0…n
```

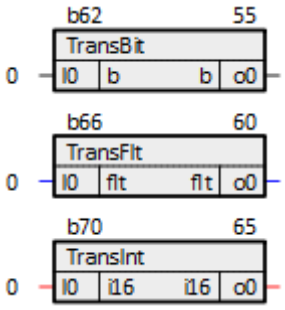


Рисунок 10.3 – Терминальные блоки

Для передачи массивов используются терминальные блоки ***Transi32A***, ***TransFA***, ***TransStor*** и др. Блок повторяет на выходе массив соответствующего входа. Данные блоки удобно использовать в качестве сбора входных и выходных параметров на одной странице программы. Блок инициализируется массивом на входе, не нулем.

При использовании блока, необходимо следить, чтобы его порядок выполнения находился между порядками выполнения блоков, с которыми он соединен, т.е. выполнялся после блоков, генерирующих входные массивы и перед блоками, которые получают выходные массивы.

Назначение входов и выходов:

in0, **in1**, **…**, **inn** – входные массивы;
o0, **o1**, **…**, **on** – выходные массивы.

Инициализация:

```
oi = ini, где I = 0…n
```

Алгоритм:

```
oi = ini, где I = 0…n
```

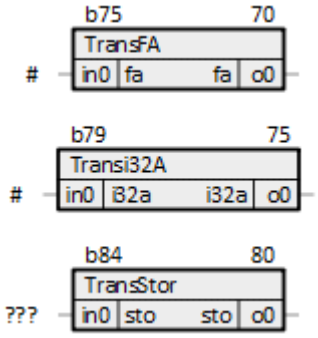


Рисунок 10.4 – Терминальные блоки передачи массивов

10.1.2Арифметические блоки

10.1.2.1 Сложение (Add)

Функция ***Add*** вычисляет сумму своих входов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

f0, **f1**, **…**, **fn** – слагаемые;
sum – сумма.

Инициализация:

```
sum = 0
```

Алгоритм:

```
sum = f0 + f1 + … + fn
```

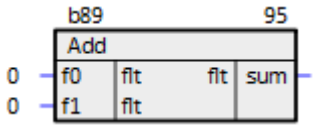


Рисунок 10.5 – Сложение (Add)

10.1.2.2 Целочисленное сложение (Addl)

Функциональный блок ***Addl*** производит целочисленное сложение. Аргументы являются целыми числами. Если на вход подается связь с выхода другого типа, то при выполнении программы происходит преобразование типов. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

in0, **in1**, **…**, **inn** – слагаемые;
sum – сумма.

Инициализация:

```
sum = 0
```

Алгоритм:

```
sum = in0 + in1 + … + inn
```

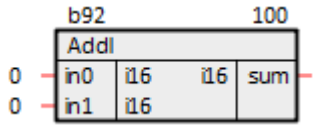


Рисунок 10.6 – Целочисленное сложение (Addl)

10.1.2.3 Вычитание (Sub)

Функция ***Sub*** вычисляет разность двух аргументов. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

f1 – уменьшаемое;
f2 – вычитаемое;
out – разность.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = f1 - f2
```

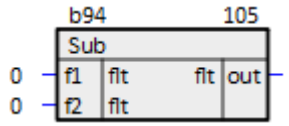


Рисунок 10.7 – Вычитание (Sub)

10.1.2.4 Умножение (Mul)

Функция ***Mul*** вычисляет произведение своих входов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

f0, **f1**, **…**, **fn** – множитель;
out – произведение множителей.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = f1 · f2 ·…· fn
```

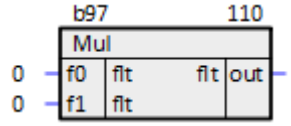


Рисунок 10.8 – Умножение (Mul)

10.1.2.5 Деление (Div)

Функция ***Div*** вычисляет частное своих аргументов. Если делитель равен нулю, то выставляется флаг ошибки, на выходе устанавливается **1**. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **1**.

Назначение входов и выходов:

f1 – делимое;
f2 – делитель;
out – частное от деления;
zf – флаг ошибки.

Инициализация:

```
out = 1;

zf = 0
```

Алгоритм:

```
Если f2 = 0 → zf = 1, out = 1,

иначе out = f1/f2, zf = 0
```

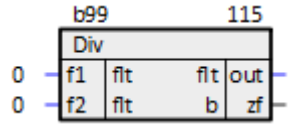


Рисунок 10.9 – Деление (Div)

10.1.2.6 Целочисленное деление (Divl)

Функция ***Divl*** производит целочисленное деление. Вычисляет неполное частное и остаток. Если делитель равен нулю, то выставляется флаг ошибки, на выходе остается прежнее значение. Аргументы являются целыми числами. Если на вход подается связь с выхода другого типа, то при выполнении программы происходит преобразование типов. Раздел библиотеки: ***Арифметические***. Выход инициализируется **1**.

Назначение входов и выходов:

i1 – делимое;
i2 – делитель;
out – частное от деления;
rem – остаток от деления;
zf – флаг ошибки.

Инициализация:

```
out = 1;

rem = 0;

zf = 0
```

Алгоритм:

```
Если i2 = 0 → zf = 1, out = out_old,

иначе out = i1/i2, rem = i1 - (out · i2), zf = 0,

где out_old – значение выхода в предыдущем цикле выполнения программы
```

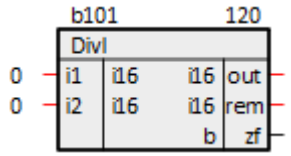


Рисунок 10.10 – Целочисленное деление (Divl)

10.1.2.7 Корень квадратный (Sqrt)

Функция ***Sqrt*** вычисляет корень квадратный своего аргумента. Если аргумент отрицательный, то выставляется флаг ошибки, на выходе корень квадратный из модуля аргумента. Аргумент является вещественным числом. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in – аргумент;
out – значение корня квадратного от аргумента;
nf – флаг ошибки.

Инициализация:

```
out = 0;
zf = 0
```

Алгоритм:

```
Если in ≤ 0 → nf = 1, out = √(|in|),
иначе out = √in, nf = 0
```

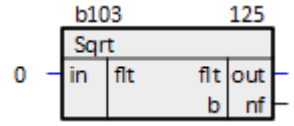


Рисунок 10.11 – Корень квадратный (Sqrt)

10.1.2.8 Инверсия значения (Inv)

Функция ***Inv*** инвертирует знак входного значения. Аргумент является вещественным числом. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

in – аргумент;
out – инвертированный вход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = -in
```

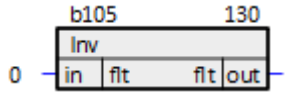


Рисунок 10.12 – Инверсия значения (Inv)

10.1.2.9 Абсолютное значение (Abs)

Функция ***Abs*** определяет модуль и знак числа. Если на вход подается отрицательное значение, то выставляется флаг знака. Аргумент является вещественным числом. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in – аргумент;
out – модуль входа;
sf – флаг знака.

Инициализация:

```
out = 0,
sf = 0
```

Алгоритм:

```
out = |in|, sf – знак in,
если in > 0 → sf = 0,
иначе sf = 1
```

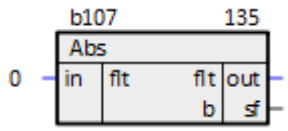


Рисунок 10.13 – Абсолютное значение (Abs)

10.1.2.10 Минимальное/максимальное значение (MinMax)

Функция ***MinMax*** ищет минимальное и максимальное значения среди набора входов. Число входов задается при создании функционального блока, и может быть изменено в процессе

редактирования программы. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in0, **in1**, **…**, **inn** – входы;
max – максимальное значение;
min – минимальное значение.

Инициализация:

```
max = 0,
min = 0
```

Алгоритм:

```
min = min (in0, in1, …, inn),
max = max (in0, in1, …, inn)
```

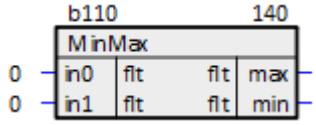


Рисунок 10.14 – Минимальное/максимальное значение (MinMax)

10.1.2.11 Селектор минимального/максимального значения (MinMax2)

Функция ***MinMax2*** ищет минимальное и максимальное значения среди набора входов, и определяет индекс минимального и максимального входа. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in0, **in1**, **…**, **inn** – входы;
max – максимальное значение;
min – минимальное значение;
mxn – индекс максимума;
mnn – индекс минимума.

Инициализация:

```
max = 0,
min = 0,
mxn = 0,
```

```
mnn = 0
```

Алгоритм:

```
min = min (in0, in1, …, inn),

max = max (in0, in1, …, inn),

mxn – индекс максимального значения,

mnn – индекс минимального значения
```

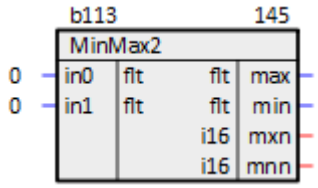


Рисунок 10.15 – Селектор минимального/максимального значения (MinMax2)

10.1.2.12 Сравнение (Cmpr)

Функция ***Cmpr*** сравнивает два входа. В зависимости от их отношения выставляет флаги больше, меньше или равно. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in1 – вход 1;
in2 – вход 2;
uf – флаг «больше»;
ef – флаг «равно»;
if – флаг «меньше».

Инициализация:

```
uf = 0,

ef = 0,

if = 0
```

Алгоритм:

	uf	ef	if
in1 > in2	1	0	0
in1 = in2	0	1	0
in1 < in2	0	0	1

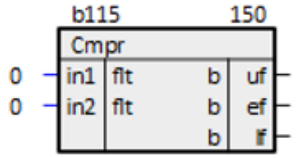


Рисунок 10.16 – Сравнение (Cmpr)

10.1.2.13 Маскирование (Mask)

Функция ***Mask*** производит побитное умножение аргумента на маску. Аргументы являются целыми числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

i1 – вход;
i2 – маска;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = i1 & i2
```

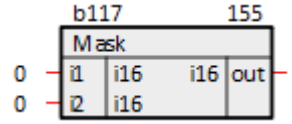


Рисунок 10.17 – Маскирование (Mask)

10.1.2.14 Сдвиг влево (Shrl)

Функция ***Shrl*** производит побитовый сдвиг влево числа **i1** на **i2** бит. Аргументы являются целыми числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

i1 – вход;
i2 – величина сдвига;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = i1≪ i2
```

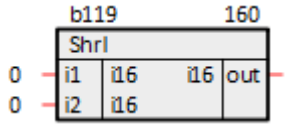


Рисунок 10.18 – Сдвиг влево (Shrl)

10.1.2.15 Сдвиг вправо (Shrr)

Функция ***Shrr*** производит побитовый сдвиг вправо числа **i1** на **i2** бит. Аргументы являются целыми числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

i1 – вход;
i2 – величина сдвига;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = i1 ≫ i2
```

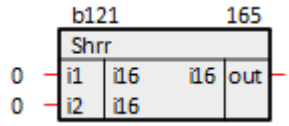


Рисунок 10.19 – Сдвиг вправо (Shrr)

10.1.2.16 Табличная функция (TabF)

Функциональный блок производит кусочно-линейную аппроксимацию зависимости **y = f(x)**, заданной табличной функцией и вычисление в заданной точке. Табличная функция задается набором пар **(x, y)**. Для корректной работы блока наборы пар **(x, y)** должны быть упорядочены по возрастанию **x**. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

val – значение **x**;

x0, **x1**, **...**, **xn** – набор абсцисс;

y0, **y2**, **...**, **yn** – набор соответствующих ординат;

out – значение **y**.

Инициализация:

```
out = 0
```

Алгоритм: при нахождении значения функции используется алгоритм кусочно-линейной аппроксимации

```
Если val < x0 → out = y0,
```

```
если val > xn → out = yn
```

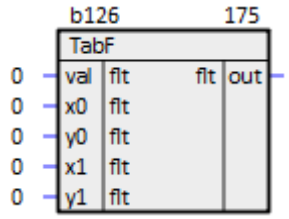


Рисунок 10.20 – Табличная функция (TabF)

10.1.2.17 Экспонента (Exp)

Функция ***Exp*** производит вычисление экспоненциальной функции от своего аргумента. Аргумент является вещественным числом. Раздел библиотеки: ***Арифметические***. Выход инициализируется **0**.

Назначение входов и выходов:

in – аргумент;

out – выход, значение ограничено интервалом **[exp(-37), exp(10)]**.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = exp(in)
```

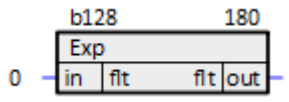


Рисунок 10.21 – Экспонента (Exp)

10.1.2.18 Логарифм (Ln)

Функция ***Ln*** вычисляет натуральный логарифм своего аргумента. Аргумент является вещественным числом. Раздел библиотеки: ***Арифметические***. Выход инициализируется **1**.

Назначение входов и выходов:

in – аргумент;
out – выход, снизу ограничен **ln(1.0e - 15)**.

Инициализация:

```
out = 1
```

Алгоритм:

```
out = ln(in)
```

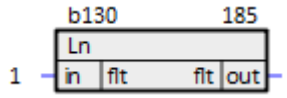


Рисунок 10.22 – Логарифм (Ln)

10.1.2.19 Возведение в степень (Pow)

Функция ***Pow*** производит возведение в степень аргумента функции. Если производится возведение в степень отрицательного числа в нецелочисленную степень, то производится округление степени до ближайшей целой. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выход инициализируется **1**.

Назначение входов и выходов:

in – аргумент;
pw – степень;
out – выход, значение ограничено интервалом **[1.0e - 18; 1.0e + 18]**.

Инициализация:

```
out = 1
```

Алгоритм:

```
out = in^pw
```

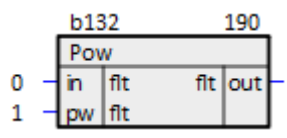


Рисунок 10.23 – Возведение в степень (Pow)

10.1.2.20 Сложение некоторых слагаемых (SumGR)

Блок ***SumGR*** вычисляет сумму входов, у которых **en ≠ 0**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in0, **in1**, **...**, **inn** – слагаемые;
en0, **en1**, **...**, **enn** – разрешающие флаги;
sum – сумма;
cnt – количество слагаемых.

Инициализация:

```
sum = 0,
cnt = 0
```

Алгоритм: при нахождении значения функции используется алгоритм кусочно-линейной аппроксимации

```
Если en0, en1, …, enn ≠ 0 → sum = in0 + in1 + … + inn, cnt = n
```

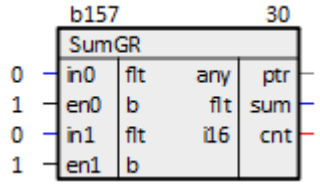


Рисунок 10.24 – Сложение некоторых слагаемых (SumGR)

10.1.2.21 Пакетное умножение и округление (mb_ai)

Блок ***mb_ai*** производит умножение входа на множитель. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

x0, **x1**, **...**, **xn** – аргументы;
m0, **m1**, **...**, **mn** – множители;
y0, **y1**, **...**, **yn** – выходы.

Инициализация:

```
y0, y1, ..., yn = 0
```

Алгоритм:

```
yi = xi · mi, I = 1, 2, …, n
```

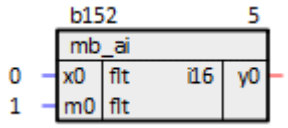


Рисунок 10.25 – Пакетное умножение и округление (mb_ai)

10.1.2.22 Пакетное умножение (mb_ao)

Блок ***mb_ao*** производит умножение входа на множитель. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Аргументы являются целыми числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

x0, **x1**, **...**, **xn** – аргументы;
m0, **m1**, **...**, **mn** – множители;
y0, **y1**, **...**, **yn** – выходы.

Инициализация:

```
y0, y1, ..., yn = 0
```

Алгоритм:

```
yi = xi · mi, I = 1, 2, …, n
```

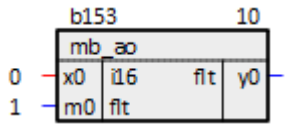


Рисунок 10.26 – Пакетное умножение (mb_ao)

10.1.2.23 Округление (flt_round)

Блок ***flt_round*** вычисляет несколько вариантов округления для входного значения. Аргумент является вещественным числом. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in – значение;

rnd – до ближайшего целого;

up – в большую сторону;

dn – в меньшую сторону.

Инициализация:

```
rnd = 0,
up = 0,
dn = 0
```

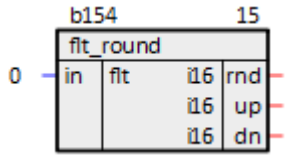


Рисунок 10.27 – Округление (flt_round)

10.1.2.24 Определение знака (Sign)

Функция ***Sign*** производит определение знака входного значения относительно базового. Аргументы являются вещественными числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in – вход;

x0 – базовое значение;

out – знак;

inv – инверсия знака;

zer – признак равенства.

Инициализация:

```
out = 0,
inv = 0,
zer = 0
```

Алгоритм:

	out	inv	zer
in > x0	1	-1	0
in = x0	0	0	1
in < x0	-1	1	0

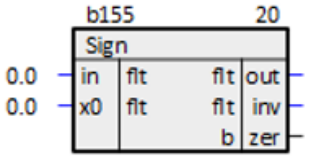


Рисунок 10.28 – Определение знака (Sign)

10.1.2.25 Переворачивание байт в слове (SwapBytes)

Функция ***SwapBytes*** меняет местами байты в двухбайтном числе. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Аргументы являются целыми числами. Раздел библиотеки: ***Арифметические***. Выходы инициализируются **0**.

Назначение входов и выходов:

in0, **in1**, **...**, **inn** – входы;
o0, **o1**, **...**, **on** – выходы.

Инициализация:

```
o0, o1, ..., on = 0
```

Алгоритм:

```
ini [byte1, byte2] → oi = ini [byte2, byte1], I = 1, 2, …, n
```

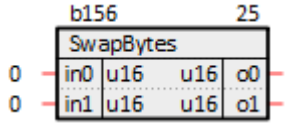


Рисунок 10.29 – Переворачивание байт в слове (SwapBytes)

10.1.2.26 Арифметические блоки для аргументов double, long, Uint64

Арифметические блоки для аргументов ***double***, ***long***, ***Uint64*** представлены в соответствующих группах в разделе библиотеки ***Арифметические***. Описание представленных блоков аналогично описанию соответствующих блоков, приведенному выше.

10.1.3Логические блоки

10.1.3.1 Логическое И (AND)

Функциональный блок ***AND*** производит логическое умножение своих входов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b0, **b1**, **...**, **bn** – входы;
q – логическое произведение;
inv – инвертированное логическое произведение.

Инициализация:

```
q = 0,
inv = 1
```

Алгоритм:

```
q = b0 & b1 &…& bn, inv = !(q)
```

Пример для блока с двумя входами:

b0	b1	q	inv
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

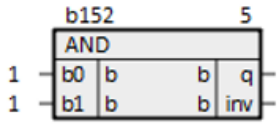


Рисунок 10.30 – Логическое И (AND)

10.1.3.2 Логическое ИЛИ (OR)

Функциональный блок ***OR*** производит логическое сложение своих входов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b0, **b1**, **...**, **bn** – входы;
q – логическое сложение;

inv – инвертированное логическое сложение.

Инициализация:

```
q = 0,
inv = 1
```

Алгоритм:

```
q = b0 ∪ b1 ∪ … ∪ bn, inv = !(q)
```

Пример для блока с двумя входами:

b0	b1	q	inv
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

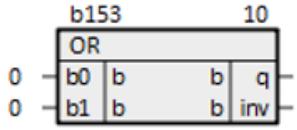


Рисунок 10.31 – Логическое ИЛИ (OR)

10.1.3.3 Логическое НЕ-И (nAND)

Функциональный блок ***nAND*** производит логическое умножение с инверсией своих входов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b0, **b1**, **...**, **bn** – входы;
q – логическое произведение инверсных входов;
inv – инвертированное **q**.

Инициализация:

```
q = 0,
inv = 1
```

Алгоритм:

```
q = !(b0) & !(b1) & … & !(bn), inv = !(q)
```

Пример для блока с двумя входами:

b0	b1	q	inv
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

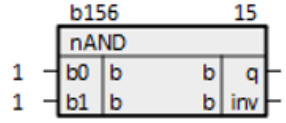


Рисунок 10.32 – Логическое НЕ-И (nAND)

10.1.3.4 Логическое НЕ (NOT)

Функциональный блок ***NOT*** производит логическую инверсию своего входа. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b1 – вход;
q – логическое отрицание.

Инициализация:

```
q = 0
```

Алгоритм:

```
q = !(b1)
```

b1	q
0	1
1	0

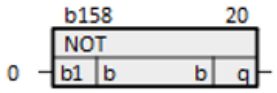


Рисунок 10.33 – Логическое НЕ (NOT)

10.1.3.5 Логическое НЕ-ИЛИ (nOR)

Функциональный блок ***nOR*** производит логическое сложение с инверсией своих входов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b0, **b1**, **...**, **bn** – входы;
q – логическое сложение инверсных входов;

inv – инвертированное **q**.

Инициализация:

```
q = 0,
inv = 1
```

Алгоритм:

```
q = !(b0) ∪ !(b1) ∪ … ∪ !(bn), inv = !(q)
```

b0	b1	q	inv
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

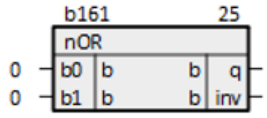


Рисунок 10.34 – Логическое НЕ-ИЛИ (nOR)

10.1.3.6 Логическое сложение по модулю два (XOR)

Функциональный блок ***XOR*** производит логическое суммирование по модулю два своих входов. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b1, **b2** – входы;
q – сумма по модулю два.

Инициализация:

```
q = 0
```

Алгоритм:

```
q = b1 ⊕ b2
```

b0	b1	q
0	0	0
0	1	1
1	0	1
1	1	0

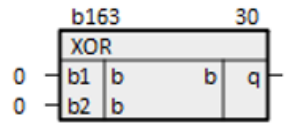


Рисунок 10.35 – Логическое сложение по модулю два (XOR)

10.1.3.7 Логическое ИЛИ с блокировкой (Bl_Or)

Функциональный блок ***Bl_Or*** осуществляет алгоритм логического ИЛИ среди входов **x** и **y**. Наличие единицы на входе блокировки запрещает все выходы, кроме собственного. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Логические***. Выходы инициализируются **0**.

Назначение входов и выходов:

x0, **x1**, **...**, **xn** – входы;
y0, **y1**, **...**, **yn** – соответствующие блокировки;
out0, **out1**, **...**, **outn** – выходы.

Инициализация:

```
out0, out1, ..., outn = 0
```

Алгоритм:

```
outi = xi | yi & !(y0 | y1 | … | yi – 1 | yi + 1 | … | yn)
```

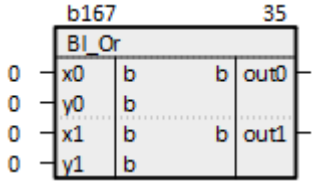


Рисунок 10.36 – Логическое ИЛИ с блокировкой (Bl_Or)

10.1.3.8 Детектор фронтов (Fronts)

Функциональный блок ***Fronts*** сравнивает предыдущее значение входа с настоящим и анализирует на наличие фронтов и спадов. Раздел библиотеки: ***Логические***. Выходы инициализируются **0**.

Назначение входов и выходов:

i – вход;
f01 – бит, сигнализирующий о приходе фронта на входе;
f10 – бит, сигнализирующий о приходе спада на входе.

Инициализация:

```
f01 = 0,
f10 = 0
```

Алгоритм:

```
Если i_old = 0 & i = 1 → f01 = 1,
```

```
если i_old = 1 & i = 0 → f10 = 1,

где i_old – значение входа на предыдущем цикле выполнения программы
```

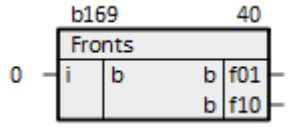


Рисунок 10.37 – Детектор фронтов (Fronts)

10.1.3.9 Детектор логического канала (bDetect)

Функциональный блок ***bDetect*** определяет номер входа, на котором произошел перепад значения из **0** в **1**. Если перепад произошел на нескольких входах, обнаруживается только самый поздний. Если перепада нет ни на одном входе, то **num = -1**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b0, **b1**, **...**, **bn** – входы;
num – номер входа, на котором произошел перепад.

Инициализация:

```
num = 0
```

Алгоритм: **num** – номер входа, на котором произошел перепад из **0** в **1**, начинается с **0**

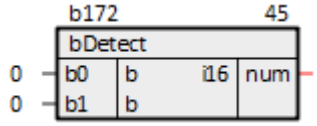


Рисунок 10.38 – Детектор логического канала (bDetect)

10.1.3.10 Контроль ключа (LogContr)

Функциональный блок ***LogContr*** определяет запрещенные состояния ключа, заданные двумя логическими сигналами. Раздел библиотеки: ***Логические***. Выход инициализируется **0**.

Назначение входов и выходов:

b0, **b1** – входы;
n_z – запрещенное состояние;
m_1 – больше одной единицы.

Инициализация:

```
n_z = 0,
m_1 = 0
```

Алгоритм:

b0	b1	n_z	m_1
0	0	1	0
0	1	0	0
1	0	0	0
1	1	1	1

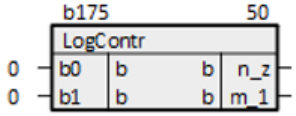


Рисунок 10.39 – Контроль ключа (LogContr)

10.1.4Тригонометрические функции

10.1.4.1 Синус (Sin)

Функция ***Sin*** вычисляет синус аргумента. Раздел библиотеки: ***Тригонометрические функции***. Выход инициализируется **0**.

Назначение входов и выходов:

in – вход;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = sin(in)
```

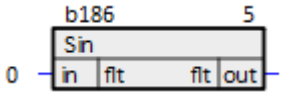


Рисунок 10.40 – Синус (Sin)

10.1.4.2 Косинус (Cos)

Функция ***Cos*** вычисляет косинус аргумента. Раздел библиотеки: ***Тригонометрические функции***. Выход инициализируется **0**.

Назначение входов и выходов:

in – вход;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = cos(in)
```

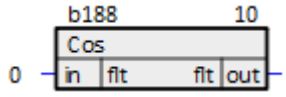


Рисунок 10.41 – Косинус (Cos)

10.1.4.3 Тангенс (Tan)

Функция ***Tan*** вычисляет тангенс аргумента. Раздел библиотеки: ***Тригонометрические функции***. Выход инициализируется **0**.

Назначение входов и выходов:

in – вход;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = tan(in)
```

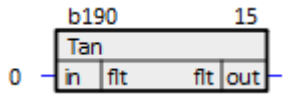


Рисунок 10.42 – Тангенс (Tan)

10.1.4.4 Арктангенс (ATan)

Функция ***ATan*** вычисляет арктангенс аргумента. Раздел библиотеки: ***Тригонометрические функции***. Выход инициализируется **0**.

Назначение входов и выходов:

in – вход;

out – выход, диапазон изменения **[-π/2 ≤ out ≤ π/2]**.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = atan(in)
```

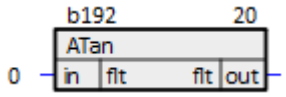


Рисунок 10.43 – Арктангенс (ATan)

10.1.4.5 Арксинус (ASin)

Функция ***ASin*** вычисляет арксинус аргумента. Раздел библиотеки: ***Тригонометрические функции***. Выход инициализируется **0**.

Назначение входов и выходов:

in – вход, диапазон изменения **[-1.0 ≤ in ≤ 1.0]**;
out – выход, диапазон изменения **[-π/2 ≤ out ≤ π/2]**;
q – флаг ошибки.

Инициализация:

```
out = 0,
q = 0
```

Алгоритм:

```
Если -1.0 ≤ in ≤ 1.0 → out = asin(in), q = 0,
если in > 1 → out = π/2, q = 1,
если in < -1 → out = -π/2, q = 1
```

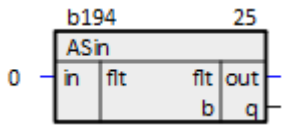


Рисунок 10.44 – Арксинус (ASin)

10.1.4.6 Арккосинус (ACos)

Функция ***ACos*** вычисляет арксинус аргумента. Раздел библиотеки: ***Тригонометрические функции***. Выход инициализируется **0**.

Назначение входов и выходов:

in – вход, диапазон изменения **[-1.0 ≤ in ≤ 1.0]**;
out – выход, диапазон изменения **[0 ≤ out ≤ π]**;
q – флаг ошибки.

Инициализация:

```
out = 0,
q = 0
```

Алгоритм:

```
Если -1.0 ≤ in ≤ 1.0 → out = acos(in), q = 0,
если in > 1 → out = 0, q = 1,
если in <-1 → out = π, q = 1
```

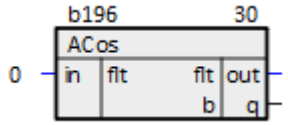


Рисунок 10.45 – Арккосинус (ACos)

10.1.5Переключатели, реле и мультиплексоры

10.1.5.1 Дешифратор (Decoder)

Функциональный блок ***Decoder*** выставляет **1** на одном из своих выходов. Индекс выхода задает входная переменная. Если значение входа выходит за рамки интервала **[0…количество выходов - 1]**, то все выходы равны **0**. Число выходов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

sel – номер выхода;
o0, **o1**, **...**, **on** – выходы.

Алгоритм:

```
o[sel] = 1
```

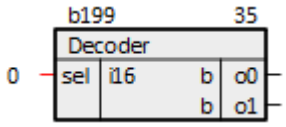


Рисунок 10.46 – Дешифратор (Decoder)

10.1.5.2 Ключ (Key)

Функциональный блок ***Key*** передает значение своего входа на выход при **1** на управляющем входе и выдает **0** на выходе при **0** на управляющем входе. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

x – вход;
i – управляющий вход;
out – выход.

Алгоритм:

```
Если i = 1 → out = x,
иначе out = 0
```

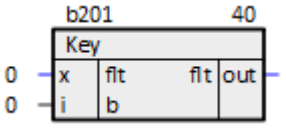


Рисунок 10.47 – Ключ (Key)

10.1.5.3 Коммутатор 8-битных регистров (R8Switch)

Функциональный блок ***R8Switch*** передает на свой выход значение одного из своих входов в зависимости от управляющего сигнала **i**. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

in1, **in2** – входы типа reg;
i – управляющий бит;
out – выход типа reg.

Алгоритм:

```
Если i = 1 → out = in1,
иначе out = in2
```

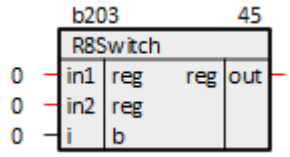


Рисунок 10.48 – Коммутатор 8-битных регистров (R8Switch)

10.1.5.4 Коммутатор вещественных переменных (fSwitch)

Функциональный блок ***fSwitch*** передает на свой выход значение одного из своих входов в зависимости от управляющего сигнала **i**. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

x1, **x2** – входы типа flt;
i – управляющий бит;
out – выход типа flt.

Алгоритм:

```
Если i = 1 → out = x1,
иначе out = x2
```

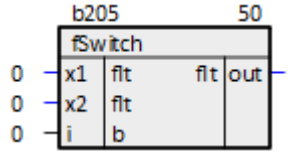


Рисунок 10.49 – Коммутатор вещественных переменных (fSwitch)

10.1.5.5 Коммутатор логических переменных (lSwitch)

Функциональный блок ***lSwitch*** передает на свой выход значение одного из своих входов в зависимости от управляющего сигнала **i**. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

x1, **x2** – входы логического типа;
i – управляющий бит;
out – выход логического типа.

Алгоритм:

```
Если i = 1 → out = x1,
```

```
иначе out = x2
```

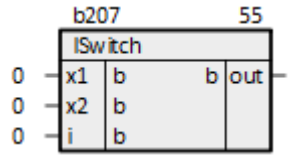


Рисунок 10.50 – Коммутатор логических переменных (lSwitch)

10.1.5.6 Коммутатор строковых переменных (txtSwitch)

Функциональный блок ***txtSwitch*** передает на свой выход значение одного из своих входов в зависимости от управляющего сигнала **i**. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

str1, **str2** – входы типа str;
cross – управляющий бит;
str1o, **str2o** – выходы типа str.

Алгоритм:

```
Если cross = 1 → str1o = str2, str2o = str1,
иначе str1o = str1, str2o = str2
```

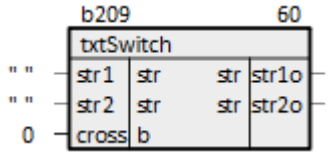


Рисунок 10.51 – Коммутатор строковых переменных (txtSwitch)

10.1.5.7 Коммутатор целых переменных (iSwitch)

Функциональный блок ***iSwitch*** передает на свой выход значение одного из своих входов в зависимости от управляющего сигнала **i**. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

x1, x2 – входы целого типа;
i – управляющий бит;
out – выход целого типа.

Алгоритм:

```
Если i = 1 → out = x1,

иначе out = x2
```

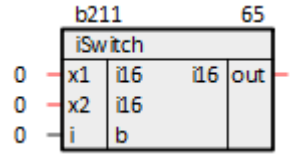


Рисунок 10.52 – Коммутатор целых переменных (iSwitch)

10.1.5.8 Релейный каскад вещественных переменных (fltRelay)

Функциональный блок ***fltRelay*** в зависимости от управляющего входа передает на каждый свой выход значение одного из входов группы с тем же индексом. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

x0, **x1**, **…**, **xn** – входы типа flt;
y0, **y1**, **…**, **yn** – входы типа flt;
upr – управляющий бит;
o0, **o1**, **…**, **on** – выходы типа flt.

Алгоритм:

```
Если upr = 1 → oi = xi,

иначе oi = yi, где i = 1, 2, …, n
```

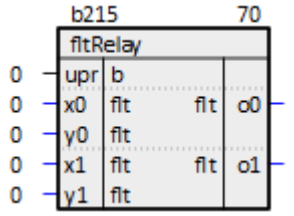


Рисунок 10.53 – Релейный каскад вещественных переменных (fltRelay)

10.1.5.9 Релейный каскад логических переменных (logRelay)

Функциональный блок ***logRelay*** в зависимости от управляющего входа передает на каждый свой выход значение одного из входов группы с тем же индексом. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

x0, **x1**, **…**, **xn** – входы логического типа;
y0, **y1**, **…**, **yn** – входы логического типа;
upr – управляющий бит;
o0, **o1**, **…**, **on** – выходы логического типа.

Алгоритм:

```
Если upr = 1 → oi = xi,

иначе oi = yi, где i = 1, 2, …, n
```

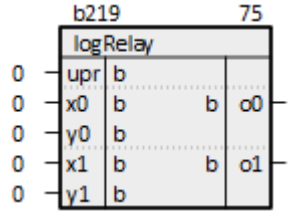


Рисунок 10.54 – Релейный каскад логических переменных (logRelay)

10.1.5.10 Управление реле (SetSW)

Блок ***SetSW*** осуществляет потенциальное и импульсное управление реле с обратной связью. Выход потенциального управления **out** выставляется в **1**, если управление **inp** и разрешение **enb** равны **1**.

На выходах импульсного управления **on** и **off** выдаются импульсы длинной в один таймерный интервал (если блок находится в ***Таймере***) соответственно при включении и выключении выхода **out**.

Ошибка **err** выдается, если обратный сигнал **ret** не появился в течение таймаута **cnt** (задается в миллисекундах). Для правильной работы таймера ошибки данный блок необходимо размещать в ***Таймере***, а значения **cnt** задавать кратными времени таймерного цикла. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

inp – управление;
ret – обратная связь;
cnt – таймер ошибки, мс;
enb – разрешение;
out – потенциальное управление;
on – включение (импульсное);
off – выключение (импульсное);
err – ошибка.

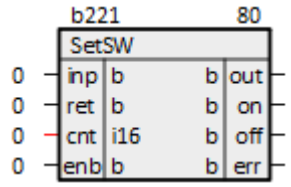


Рисунок 10.55 – Управление реле (SetSW)

10.1.5.11 Выбор одного из целых значений (SelectUI)

Блок ***SelectUI*** выводит на выход первое из входных значений, для которого **en = 1**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

val0, **val1**, **…**, **valn** – входы;
en0, **en1**, **…**, **enn** – выбор входа;
out – выбранное значение.

Алгоритм:

```
Если eni = 1 → out = vali, где i = 1, 2, …, n
```

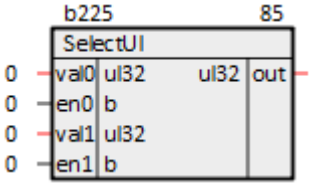


Рисунок 10.56 – Выбор одного из целых значений (SelectUI)

10.1.5.12 Вывод значения на заданный выход (ShowSW)

Блок ***ShowSW*** выводит значение входа **spc** на соответствующий выход **o** при появлении его номера на входе **num** (входы нумеруются от **1**). Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

num – номер входа;
def – значение по умолчанию;
spc0, **spc1**, **…**, **spcn** – входы,
o0, **o1**, **…**, **on** – выходы.

Инициализация:

```
oi = def
```

Алгоритм:

```
Для каждого входа, если i = num - 1, oi = spci, иначе oi = def
```

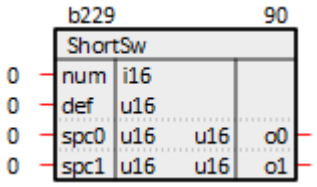


Рисунок 10.57 – Вывод значения на заданный выход (ShowSW)

10.1.5.13 Мультиплексор вещественных значений (MuxFlt)

Функциональный блок ***MuxFlt*** передает значение своего входа с индексом **sel** на выход. Если индекс выходит за рамки интервала **[0…количество входов - 1]** то на выходе остается предыдущее значение. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***. Изначально выход инициализируется **0**.

Назначение входов и выходов:

sel – номер входа;
in0, **in1**, **…**, **inn** – входы,
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = in[sel]
```

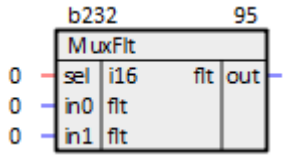


Рисунок 10.58 – Мультиплексор вещественных значений (MuxFlt)

10.1.5.14 Мультиплексор целых значений (MuxInt)

Функциональный блок ***MuxFlt*** передает значение своего входа с индексом **sel** на выход. Если индекс выходит за рамки интервала **[0…количество входов - 1]** то на выходе остается предыдущее значение. Число входов задается при создании функционального блока, и может быть

изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***. Изначально выход инициализируется **0**.

Назначение входов и выходов:

sel – номер входа;
in0, **in1**, **…**, **inn** – входы,
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = in[sel]
```

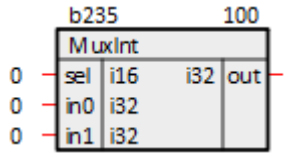


Рисунок 10.59 – Мультиплексор целых значений (MuxInt)

10.1.5.15 Выбор одного из вещественных значений (FCascSw)

Блок ***FCascSw*** выводит на выход первое из входных значений, для которого **I = 1**. Если все **I = 0**, то на выход выводится значение по умолчанию **DEF**, и признак **Q** выставляется в **1**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***. Изначально выход инициализируется **0**.

Назначение входов и выходов:

DEF – значение выхода по умолчанию;
x0, **x1**, **…**, **xn** – входы,
I0, **I1**, **…**, **In** – выбрать значение,
Y – выход,
Q – признак значения по умолчанию.

Алгоритм:

```
Если li = 1 → Y = xi, где i = 1, 2, …n,

если I0, I1, …, In = 0 → Y = DEF, Q = 1
```

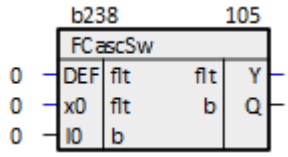


Рисунок 10.60 – Выбор одного из вещественных значений (FCascSw)

10.1.5.16 Выбор строки (Switch_str)

Блок ***Switch_str*** выводит на выход строку, для которой **cond = 1**. Если несколько входов имеют значение **cond = 1**, то на выход выводится значение строки из входа, у которого **cond = 1** и имеющего наибольший порядковый номер. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Назначение входов и выходов:

str0, **str1**, **…**, **strn** – входы,
cond0, **cond1**, **…**, **condn** – выбрать строку,
str_out – выход.

Алгоритм:

```
Если condi = 1 → str_out = stri, где i = 1, 2, …n
```

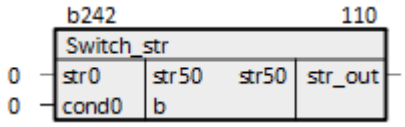


Рисунок 10.61 – Выбор строки (Switch_str)

10.1.5.17 Дешифратор уставок (Settings)

Блок ***Settings*** представляет собой дешифратор уставок. Раздел библиотеки: ***Переключатели, реле и мультиплексоры***.

Таблица 10.2 – Назначение входов и выходов Settings

Входы	
nust	Текущий номер уставки
val	Значение текущей уставки
nout	Номер уставки для вывода на выход **vout**
n0	Начальный номер уставок **in**
wen	Разрешение записи из **val** в **o**
ren	Разрешение чтения из **o** в **vout**
in	Значение инициализации уставки (циклический)
Выходы	
vout	Последнее прочитанное/записанное значение уставки с номером **nout**
wrtng	Фронт записи
rdng	Фронт чтения
o	Текущее значение уставки (циклический)

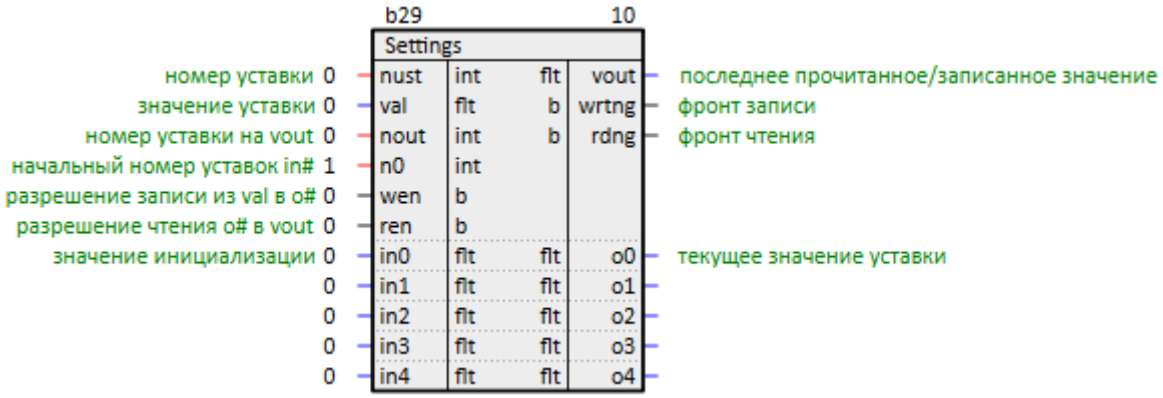


Рисунок 10.62 – Дешифратор уставок (Settings)

10.1.6Генераторы и таймера

10.1.6.1 Генератор импульсов (Pulse)

Функциональный блок ***Pulse*** генерирует импульсы заданной ширины в ответ на появление фронта на его управляющем входе. Ширина импульса **T** – задается в миллисекундах. Длительность паузы соответствует величине цикла, в котором размещен блок. Раздел библиотеки: ***Генераторы и таймера***.

Для адекватной работы блока, необходимо размещать его только в ***Таймере***. Не рекомендуется задавать ширину импульса меньше времени таймерного цикла. Оптимальными значениями ширины импульса являются числа кратные величине таймерного цикла (не кратные значения округляются до времени таймерного цикла в большую сторону).

Назначение входов и выходов:

I – управление;
T – ширина импульса, мс;
q – выход.

Инициализация:

```
q = 0
```

Алгоритм:

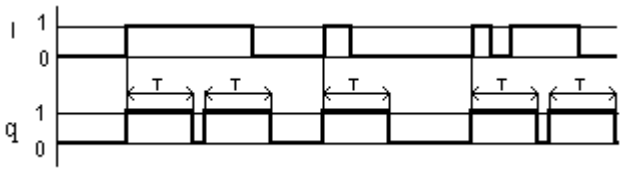


Рисунок 10.63 – Алгоритм блока Pulse

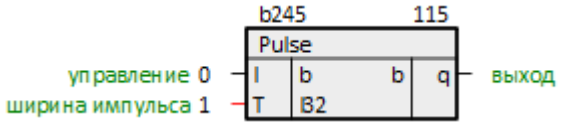


Рисунок 10.64 – Генератор импульсов (Pulse)

10.1.6.2 Включение с задержкой (DelayOn)

Функциональный блок ***DelayOn*** реагирует на фронт на управляющем входе **I** через заданное время задержки **T**. Если управляющий бит сбросится, логический таймер также сбросится и будет ожидать следующего фронта на управляющем входе. Раздел библиотеки: ***Генераторы и таймера***.

Для адекватной работы блока, необходимо размещать его только в ***Таймере***. Не рекомендуется задавать время задержки меньше времени таймерного цикла. Оптимальными значениями времени задержки являются числа кратные величине таймерного цикла (не кратные значения округляются до времени таймерного цикла в большую сторону).

Назначение входов и выходов:

I – управление;
T – время задержки, мс;
q – выход.

Инициализация:

```
q = 0
```

Алгоритм:

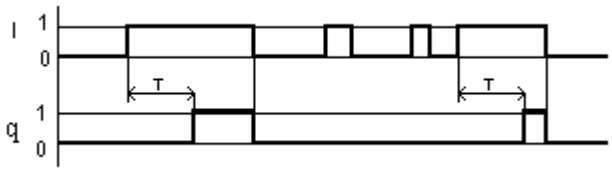


Рисунок 10.65 – Алгоритм блока DelayOn

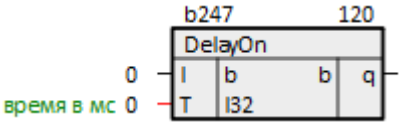


Рисунок 10.66 – Включение с задержкой (DelayOn)

10.1.6.3 Выключение с задержкой (DelayOff)

Функциональный блок ***DelayOff*** при установлении управляющего бита **I** устанавливает выходной бит **q**, инициализирует логический таймер – блок переходит в режим ожидания спада

управляющего бита. После спада управляющего бита выходной бит сбрасывается через время задержки **T**. Если до истечения времени задержки выставится управляющий бит, логический таймер сбросится и будет ожидать следующего спада управляющего бита. Раздел библиотеки: ***Генераторы и таймера***.

Для адекватной работы блока, необходимо размещать его только в ***Таймере***. Не рекомендуется задавать время задержки меньше времени таймерного цикла. Оптимальными значениями времени задержки являются числа кратные величине таймерного цикла (не кратные значения округляются до времени таймерного цикла в большую сторону).

Назначение входов и выходов:

I – управление;
T – время задержки, мс;
q – выход.

Инициализация:

```
q = 0
```

Алгоритм:

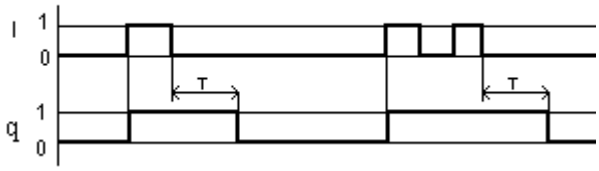


Рисунок 10.67 – Алгоритм блока DelayOff

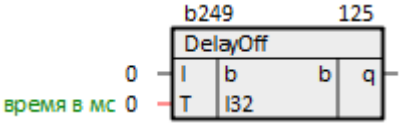


Рисунок 10.68 – Выключение с задержкой (DelayOff)

10.1.6.4 Половинный импульс после половины паузы (HalfPulse)

Блок ***HalfPulse*** выдает импульс длительности **T/2** после паузы **T/2** по перепаду **0** в **1** на входе **I**. Раздел библиотеки: ***Генераторы и таймера***.

Для адекватной работы блока, необходимо размещать его только в ***Таймере***. Не рекомендуется задавать время длительности меньше времени таймерного цикла. Оптимальными значениями времени длительности являются числа кратные величине таймерного цикла (не кратные значения округляются до времени таймерного цикла в большую сторону).

Назначение входов и выходов:

I – управление;

T – длительность, мс;

q – выход.

Инициализация:

```
q = 0
```

Алгоритм:

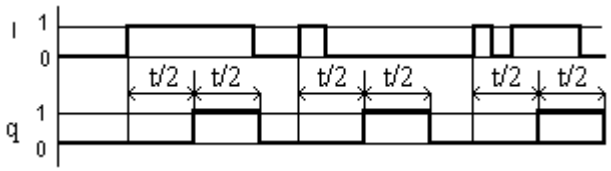


Рисунок 10.69 – Алгоритм блока HalfPulse

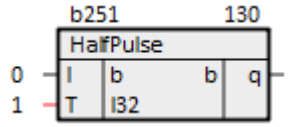


Рисунок 10.70 – Половинный импульс после половины паузы (HalfPulse)

10.1.6.5 Генератор сигналов (GenSign)

Блок ***GenSign*** генерирует сигналы различной формы. Раздел библиотеки: ***Генераторы и таймера***.

Для адекватной работы блока, необходимо размещать его только в ***Таймере***. Не рекомендуется задавать время длительности меньше времени таймерного цикла. Оптимальными значениями времени длительности являются числа кратные величине таймерного цикла (не кратные значения округляются до времени таймерного цикла в большую сторону).

Назначение входов и выходов:

Tsg – период сигнала в секундах;

Amp – амплитуда сигнала;

Shf – смещение сигнала по оси ординат;

Aut – включение периодичности сигнала;

Dis – сброс сигнала;

sin – синусоида;

line – пила;

sqw – меандр;

l1,**l2** – логические биты.

Инициализация:

```
sin = line = sqw = l1 = l2 = 0
```

Алгоритм:

1. Если **Aut = 0**, тогда сигналы на выходе имеют следующий вид:

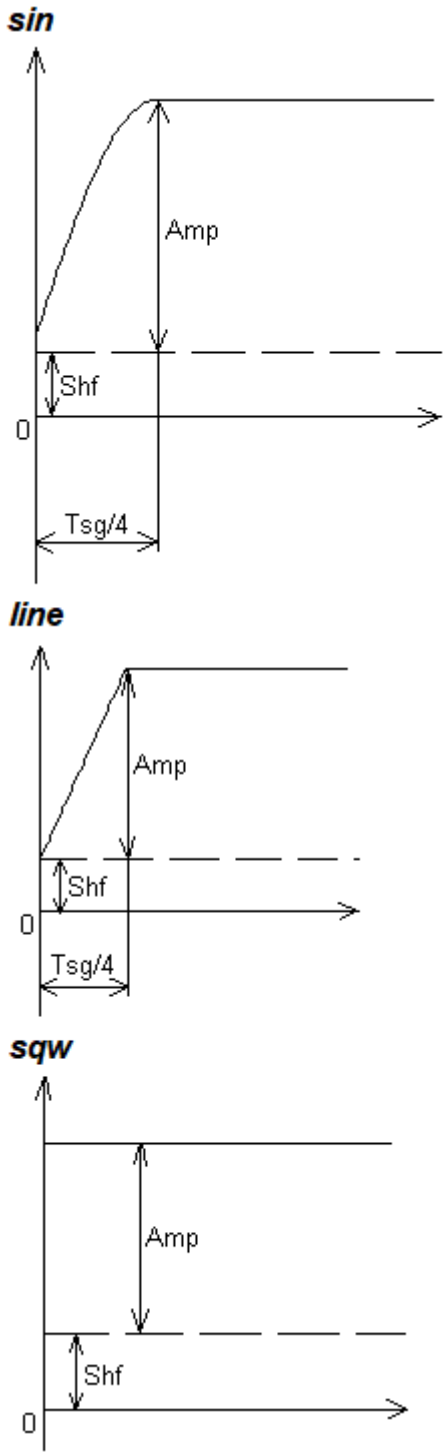


Рисунок 10.71 – Алгоритм блока GenSign при Aut = 0

2. Если **Aut = 1**, тогда сигналы на выходе имеют следующий вид:

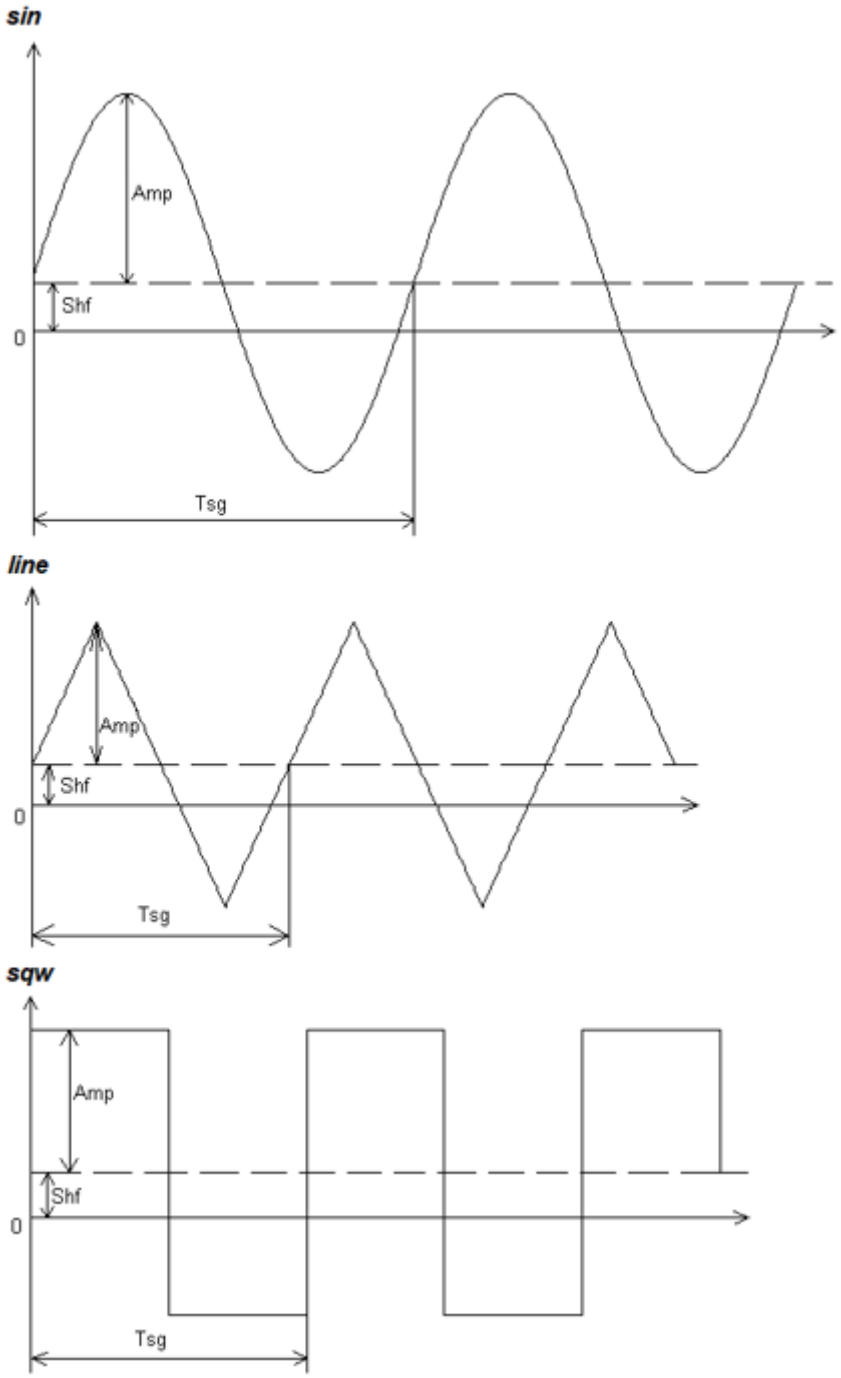


Рисунок 10.72 – Алгоритм блока GenSign при Aut = 1

Если установить и сбросить бит **Dis**, сигнал начнет генерироваться со значения **Shf**.

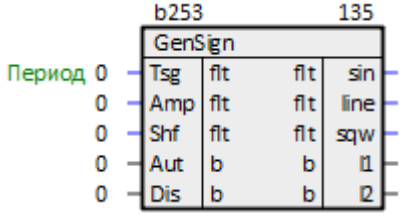


Рисунок 10.73 – Генератор сигналов (GenSign)

10.1.6.6 Логический таймер со взводом (AdvTimer)

Блок ***AdvTimer*** передает на выход **q** с управляющего входа после задержки по времени **T**. Взвод осуществляется по входу **set** при **I = 1**. Раздел библиотеки: ***Генераторы и таймера***.

Для адекватной работы блока, необходимо размещать его только в ***Таймере***. Не рекомендуется задавать время задержки меньше времени таймерного цикла. Оптимальными значениями времени задержки являются числа кратные величине таймерного цикла (не кратные значения округляются до времени таймерного цикла в большую сторону).

Назначение входов и выходов:

I – управление;
T – задержка, мс;
set – задержка;
q – выход,
t – значение таймера.

Инициализация:

```
q = 0,
t = T
```

Алгоритм:

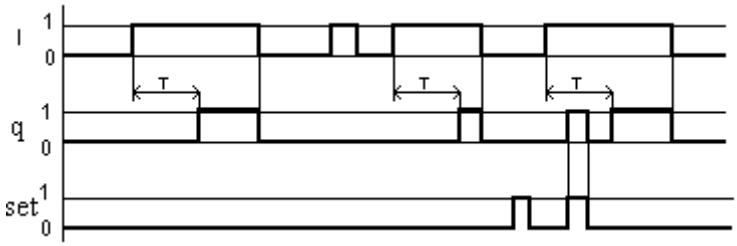


Рисунок 10.74 – Алгоритм блока AdvTimer

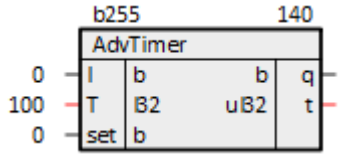


Рисунок 10.75 – Логический таймер со взводом (AdvTimer)

10.1.6.7 Генератор импульсов с заданной скважностью (PulseGeneratorEx)

Блок ***PulseGeneratorEx*** при установленном в **1** входе **enbl** на выходе формирует логические импульсы заданной скважности: длительность импульса задается входом **t1**, длительность паузы – входом **t2**. Раздел библиотеки: ***Генераторы и таймера***.

Для адекватной работы блока, необходимо размещать его только в ***Таймере***. Не рекомендуется задавать время длительности меньше времени таймерного цикла. Оптимальными значениями времени длительности являются числа кратные величине таймерного цикла (не кратные значения округляются до времени таймерного цикла в большую сторону).

Назначение входов и выходов:

t1, **t_on** – импульс, мс;
t2, **t_off** – пауза, мс;
enbl – разрешение;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

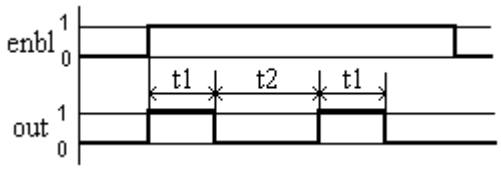


Рисунок 10.76 – Алгоритм блока PulseGeneratorEx

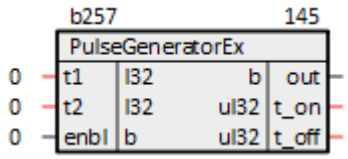


Рисунок 10.77 – Генератор импульсов с заданной скважностью (PulseGeneratorEx)

10.1.6.8 Генератор случайных чисел (rndgen)

Блок ***rndgen*** выдает на выходе случайные числа в диапазоне от **-amp** до **+amp**. Раздел библиотеки: ***Генераторы и таймера***.

Назначение входов и выходов:

amp – амплитуда;

out – случайное значение.

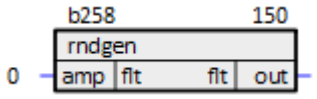


Рисунок 10.78 – Генератор случайных чисел (rndgen)

10.1.6.9 Универсальный счетчик (CounterCTN)

Универсальный счетчик ***CounterCTN*** используется для прямого и обратного счета. Операция «прямой счет» выполняется по переднему фронту импульса на входе прямого счета **inU**, что увеличивает значение выходного сигнала **out**. Импульсы, приходящие на вход **inD** («обратный счет»), уменьшают значение выхода **out**. В случае поступления на вход **rst** сигнала логической **1**, выход счетчика **out** устанавливается в значение входа **n**. Раздел библиотеки: ***Генераторы и таймера***.

В случае одновременного поступления сигналов на входы **inU** и **inD** приоритетным является сигнал входа **inU**. Допустимый диапазон значений числа импульсов **n**: от **0** до **65535**.

Назначение входов и выходов:

inU – вход импульсов прямого счета;
inD – вход импульсов обратного счета;
rst – логический вход;
n – установленное значение импульсов;
out – выход.

Алгоритм:

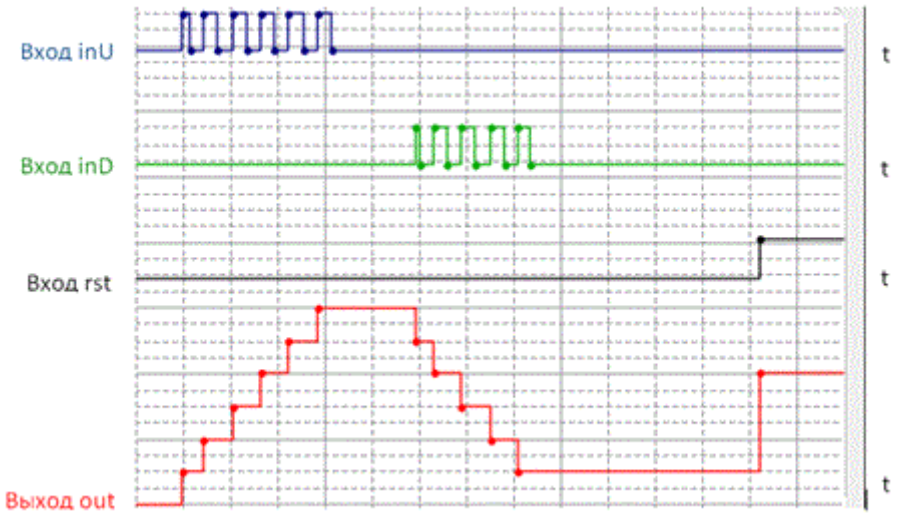


Рисунок 10.79 – Алгоритм блока CounterCTN

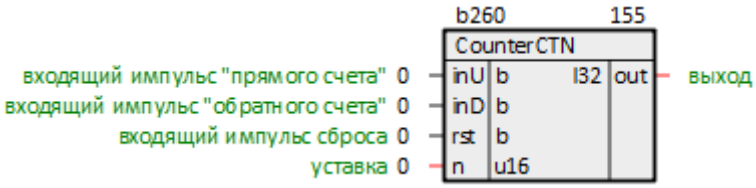


Рисунок 10.80 – Универсальный счетчик (CounterCTN)

10.1.6.10 Инкрементный счетчик (CounterCTU)

Инкрементный счетчик ***CounterCTU*** используется для подсчета числа импульсов, приходящих на вход **in**. На выходе **out** счетчика появится импульс сигнала логической **1**, если число приходящих на вход импульсов достигнет установленного значения на входе **n**. Счетчик сбрасывается в **0** по переднему фронту импульса на входе **rst**. Раздел библиотеки: ***Генераторы и таймера***.

В случае одновременного поступления сигналов на входы приоритетным является сигнал входа **rst**. Допустимый диапазон значений числа импульсов **n**: от **0** до **65535**.

Назначение входов и выходов:

in – вход импульсов прямого счета;
rst – импульсный вход для сброса выхода счетчика в **0**;
n – установленное значение импульсов;
out – выход.

Алгоритм:

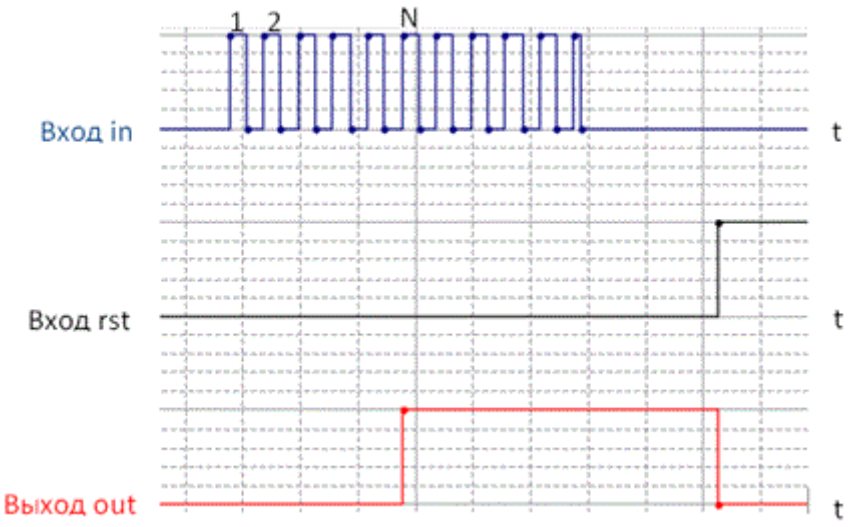


Рисунок 10.81 – Алгоритм блока CounterCTU

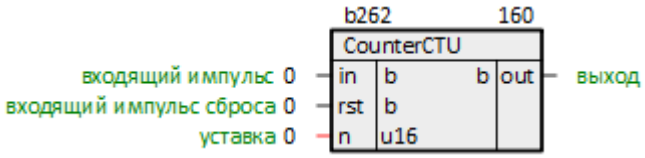


Рисунок 10.82 – Инкрементный счетчик (CounterCTU)

10.1.6.11 Инкрементный счетчик с автосбросом (CounterCT)

Инкрементный счетчик с автосбросом ***CounterCT*** используется для подсчета заданного числа импульсов **n**. На выходе **out** счетчика появится импульс сигнала логической **1** длительностью в один цикл, если число приходящих на вход **in** импульсов достигнет установленного значения **n**. Раздел библиотеки: ***Генераторы и таймера***.

Допустимый диапазон значений числа импульсов **n**: от **0** до **65535**.

Назначение входов и выходов:

in – вход импульсов;
n – устанавливаемое значение импульсов;
out – выход.

Алгоритм:

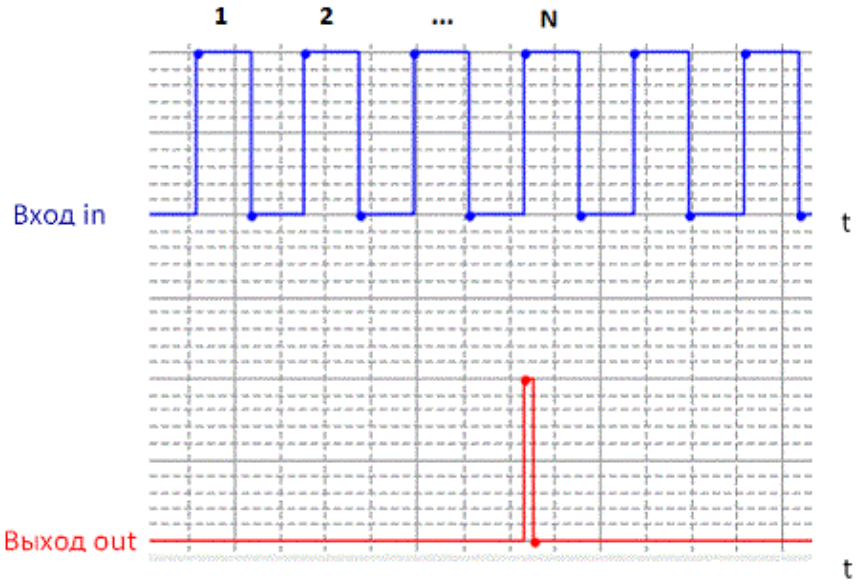


Рисунок 10.83 – Алгоритм блока CounterCT

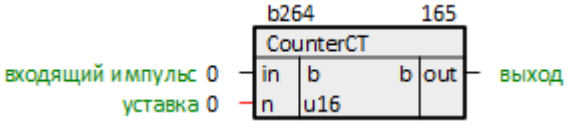


Рисунок 10.84 – Инкрементный счетчик с автосбросом (CounterCT)

10.1.7Обработка сигналов

10.1.7.1 Симметричный ограничитель сигнала (LimSim)

Функциональный блок ***LimSim*** является симметричным (на величину **val**) ограничителем входного сигнала **inp**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

inp – входной сигнал;
val – величина ограничения (абсолютное значение);
dis – команда обнуления выхода;
out – выходной сигнал ограничителя.

Инициализация:

```
out = 0
```

Алгоритм:

```
dis = 0 → out = inp, если inp > val → out = val,
если inp < -val → out = -val,
dis = 1 → out = 0
```

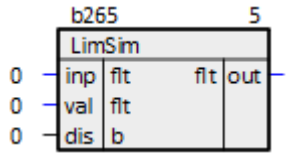


Рисунок 10.85 – Симметричный ограничитель сигнала (LimSim)

10.1.7.2 Аппроксимация обратной функции (Approx)

Блок ***Approx*** для заданной табличной функции **y = f(x)** производит аппроксимацию обратной функции **x = g(y)**. Табличная функция **y = f(x)** задается массивом **tab_y** значений **y**, соответствующих значениям **x** аргумента, выбранным из диапазона **[minx...maxx]** с шагом **hx**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

y – табличная функция;
hx – шаг **x**;
minx – минимум **x**;
maxx – максимум **x**;
ty0, **ty1**, **…**, **tyn** – значения **y** для **x0**, **x1**, **…**, **xn**;
x – обратная функция.

Инициализация:

```
out = 0
```

Алгоритм: используется кусочно-линейная аппроксимация по двум соседним точкам.

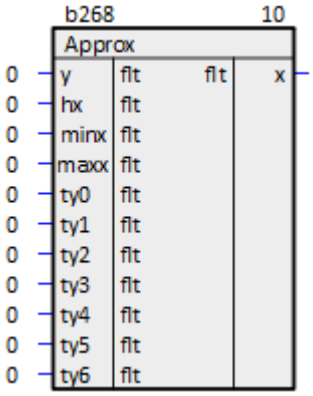


Рисунок 10.86 – Аппроксимация обратной функции (Approx)

10.1.7.3 Кусочно-линейная интерполяция (KL_Interp)

Блок вычисляет значение функции **y**, заданную таблично **(x0**, **y0**, **...)**, при значении аргумента, равного **in**. При этом используется кусочно-линейная интерполяция. Значение **dlt** задает смещение значения **y**, относительно значений в точках **x0**, **x1**, **…**, которое используется если **edl = 1**. Если значение **x** не входит в диапазон, заданный точками **x0**, **x1**, **…**, то выставляется флаг ошибки **err**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

in – значение **x**;
dlt –смещение **y**;
edl – учитывать смещение;
t0, **t1**, **…**, **tn** – значение **y0**, **y1**, **…**, **yn**;
c0, **c1**, **…**, **cn** – значение **x0**, **x1**, **…**, **xn**;
out – значение **y**;
err – флаг ошибки.

Инициализация:

```
out = 0, err = 0
```

Алгоритм: вычисление значения функции основано на кусочно-линейной интерполяции.

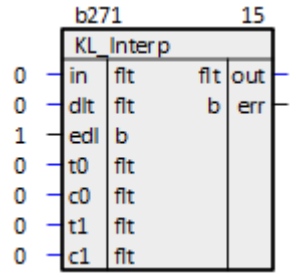


Рисунок 10.87 – Кусочно-линейная интерполяция (KL_Interp)

10.1.7.4 Несимметричный ограничитель сигнала (Lim)

Функциональный блок ***Lim*** является Функциональный блок является ограничителем входного сигнала **inp**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

inp – входной сигнал;
max – верхняя граница;
min – нижняя граница;
out – выход;
uf – **inp > max**;
if – **inp < min**.

Инициализация:

```
out = 0
```

Алгоритм:

```
out = inp, если inp > max → out = max, uf = 1,
если inp < min → out = min, if = 1
```

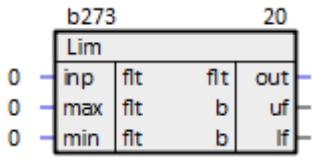


Рисунок 10.88 – Несимметричный ограничитель сигнала (Lim)

10.1.7.5 Группа ограничителей сигнала (Lims)

Функциональный блок ***Lims*** представляет собой группу ограничителей сигнала **val**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

vali– входы;
maxi – верхняя граница -го входа;
mini – нижняя граница -го входа;
oi – выходы.

Инициализация:

```
oi = 0
```

Алгоритм:

```
o = val, если val > max → o = max,
если val < min → o = min
```

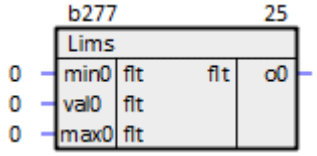


Рисунок 10.89 – Группа ограничителей сигнала (Lims)

10.1.7.6 Градиентный фильтр (GradLim)

Функциональный блок ***GradLim*** ограничивает рост входного сигнала **Q**, сравнивая приращение этой величины с заданной величиной допустимого приращения **dQ**. Если приращение не превышает заданного, блок работает как усредняющий по двум значениям фильтр. В противном случае на выходе блока устанавливается величина **out[i - 1] ± dQ**, а индикатор **If** устанавливается в единицу. Дискретизация по времени задается стробирующим входом **enbl**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

Q – вход;
dQ – допустимое приращение;
enbl – стробирующий вход;
out – выход;
if – приращение больше заданного.

Инициализация:

```
out = 0, if = 0
```

Алгоритм:

```
out = out[i - 1] ± dQ, при -dQ < Q - out[i - 1] < dQ,
out = out[i - 1] + Q, иначе
```

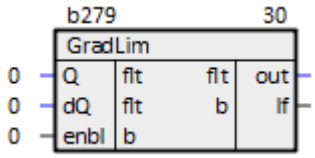


Рисунок 10.90 – Градиентный фильтр (GradLim)

10.1.7.7 Определение принадлежности к диапазону (Inside)

Функциональный блок ***Inside*** определяет принадлежность значения входной переменной **val** к заданному диапазону **(a…b)** с допуском нечувствительности **eps** (от **-eps/2** до **+eps/2**). Если значение **val** находится внутри диапазона, в единицу устанавливается выход **isd** (внутри), если же значение **val** находится вне диапазона, в единицу устанавливается выход **osd** (снаружи). Единица на выходе **apx** указывает на то, что значение **val** находится вблизи границы диапазона (в зоне нечувствительности). Это свойство может оказаться полезным для задания гистерезисных зависимостей. Если это не нужно, необходимо задать **eps = 0**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

val – вход;
a – нижняя граница;
b – верхняя граница;
eps – гистерезис;
enbl – разрешение на работу блока;
isd – внутри;
osd – снаружи;
apx – в зоне нечувствительности.

Инициализация:

```
isd = 0, osd = 0, apx = 0
```

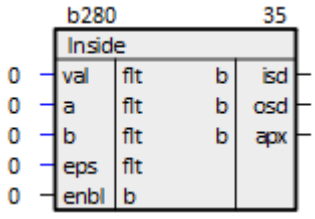


Рисунок 10.91 – Определение принадлежности к диапазону (Inside)

10.1.7.8 Параболический ограничитель (SqLim)

Блок ***SqLim*** обнаруживает выход входного значения за ограничивающие параболы. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

dp – вход;
q – x параболы;
k1 – коэффициент 1;
k2 – коэффициент 2;
gt – больше верхней;
lt– ниже нижней;
alm – выход за пределы.

Инициализация:

```
gt = 0, lt = 0, alm = 0
```

Алгоритм:

```
Если q > 0.01 → gt = 1, если dq > k2·q·q, itlt1, если dq < k1·q·q,
иначе gt = 1, если dq > 0, lt = 1, если dq < 0,
alm = gtUlt.
```

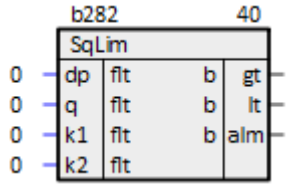


Рисунок 10.92 – Параболический ограничитель (SqLim)

10.1.7.9 Фильтр Баттерворта (BatterFilter)

Блок ***BatterFilter*** представляет из себя фильтр Баттерворта с постоянной времени **Tx** и коэффициентом демпфирования **g**. Для адекватной работы фильтра необходимо размещать данный блок в ***Таймере***. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

in – вход;
Tx – постоянная времени;
g – коэффициент демпфирования;
out – выход.

Инициализация:

```
out = 0
```

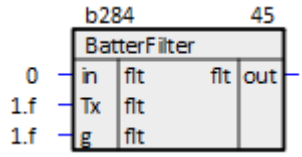


Рисунок 10.93 – Фильтр Баттерворта (BatterFilter)

10.1.7.10 Нормализация шкалы (Scale)

Блок ***Scale*** осуществляет линейное преобразование диапазонов без ограничителя. Аргументы являются вещественными числами. Раздел библиотеки: ***Обработка сигналов***. Выход инициализируется **0**.

Назначение входов и выходов:

x – вход;
minx – минимум входа;
maxx – максимум входа;
miny – минимум выхода;
maxy – максимум выхода;
y – выход.

Инициализация:

```
y = 0
```

Алгоритм:

```
Если (max(x) - min(x)) > 0.001 → y =  (max(y) - min(y))/(max(x) - min(x))·(x - min(x)) + min(y),

иначе y = min(y)
```

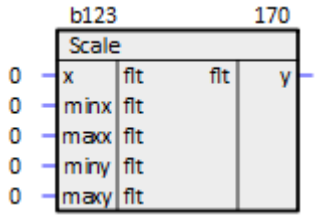


Рисунок 10.94 – Нормализация шкалы (Scale)

10.1.7.11 Преобразователь для модуля термосопротивления (5B34Conv)

Блок ***5B34Conv*** предназначен для согласования 50-Омных термосопротивлений типа 50П и 50М с входными модулями типа 50П. Для согласования точки 0 °С последовательно с датчиком подключается постоянное сопротивление 50 Ом. Для пересчета показаний результат измерения с функционального блока аналогового входа заводится на вход **t** блока ***5B34Conv***. При использовании

датчика типа 50М вход **type** устанавливается в **1**, а для 50П – в **2**. Раздел библиотеки: ***Обработка сигналов***.

Диапазон работы функционального блока: для 50П -150 °C … 200 °C, для 50М -50 °C … 200 °C.

Назначение входов и выходов:

t – вход;
type – тип (**1**, **2**);
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
type = 1: t > -50,5 → out = 2,02·t,
иначе out = 1.33·t-32.8.
type = 2: t > -10,8 → out = 1,83·t,
иначе out = 4.62·t + 29.9
```

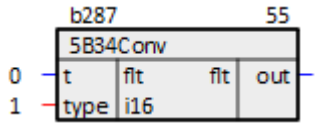


Рисунок 10.95 – Преобразователь для модуля термосопротивления (5B34Conv)

10.1.7.12 Интерполяция (Interpolation)

Блок ***Interpolation*** производит интерполяцию функции, заданной на входах **x** и **y**. Если функция задана неправильно, либо количество **num** превышает заданное, выставляется признак ошибки **sts**. Если **en = 0**, то **out = in**. Раздел библиотеки: ***Обработка сигналов***.

Блок разрешается устанавливать только в ***Фон***.

Назначение входов и выходов:

en – включение блока;
typ – тип (**0** – линейная, 1 – сплайн Акима);
num – количество точек;
ctrl – не используется;
cfg – не используется;
in – вход;
x0, **x1**, **…**, **xn** – значения **x**;
y0, **y1**, **…**, **yn** – значения **y**;
out – выход;

sts – ошибка.

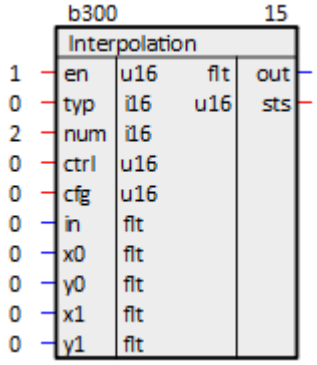


Рисунок 10.96 – Интерполяция (Interpolation)

10.1.7.13 Симметричная мертвая зона с гистерезисом (FdeadS)

Функциональный блок ***FdeadS*** является симметричной (на величину **val**) мертвой зоной по отношению к сигналу **inp**. Отличие данного ФБ от блока Симметричная мертвая зона состоит в том, что при выходе сигнала **inp** за пределы мертвой зоны, в блоке формируется внутренний флаг, который скачком отключает ее. Сброс флага происходит в момент смены полярности входного сигнала **inp**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

inp – вход;
val – мертвая зона;
dis – отключение мертвой зоны;
out – выход.

Инициализация:

```
out = 0
```

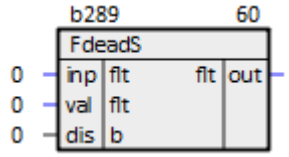


Рисунок 10.97 – Симметричная мертвая зона с гистерезисом (FdeadS)

10.1.7.14 Симметричная мертвая зона (DbandS)

Функциональный блок ***FdeadS*** является симметричной (на величину **val**) мертвой зоной по отношению к сигналу **inp**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

inp – вход;
val – мертвая зона;
dis – отключение мертвой зоны;
out – выход.

Инициализация:

```
out = 0
```

Алгоритм:

```
dis = 0: out = 0, если inp > val → out = inp - val,
если inp < -val → out = inp + val.
type = 2:t > -10,8 → out = 1,83·t,
dis = 1: out = inp
```

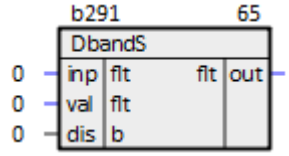


Рисунок 10.98 – Симметричная мертвая зона (DbandS)

10.1.7.15 Диагностика сигналов датчиков (SensSpd)

Блок ***SensSpd*** анализирует скорость изменения входного сигнала **inp** и ограничивает его, если скорость превышает заданную **spd**. При этом сравниваются текущее значение сигнала **inp** и его значение на предыдущем цикле выполнения программы, а скорость **spd** задает максимально допустимое изменение сигнала в секунду. При отключении диагностики **dis = 0** сигнал **inp** передается на выход **out** без изменений, **hsp = 0** и **nul = 0**. Первые **10** циклов программы диагностика отключена независимо от значения **dis**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

inp – вход;
spd – скорость;
dis – включить;
out – выход;
hsp – высокая скорость;
nul – не изменился.

Инициализация:

```
out = 0, hsp = 0, nul = 0
```

Алгоритм:

```
Если inp = inp[i - 1] → nul = 1, out = inp,

если inp - out[i - 1] > spd·таймерный интервал,

out = inp + spd·таймерный интервал,

hsp = 1,

если inp - out[i - 1] < -spd·таймерный интервал,

out = inp - spd·таймерный интервал,

hsp = 1,

иначе out = inp, hsp = 0, nul = 0
```

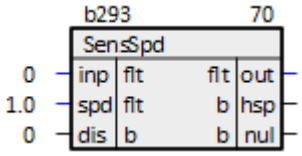


Рисунок 10.99 – Диагностика сигналов датчиков (SensSpd)

10.1.7.16 Изменение значения за заданное время (ValDif)

Функциональный блок ***ValDif*** передает на выход изменение значения **val** за установленное время **time**. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

val – вход;
time – время, мс;
diff – разница за время **time**.

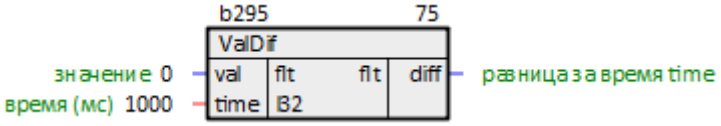


Рисунок 10.100 – Изменение значения за заданное время (ValDif)

10.1.7.17 Арифметическое скользящее среднее (MovAverage)

Блок ***MovAverage*** реализует алгоритм определения среднего значения исходного входного сигнала за предыдущие периоды (циклы). При инициализации блока происходит выделение памяти под массив сохраняемых значений входного сигнала с заданными пользователем максимальными размерами. В дальнейшем пользователь может динамически изменять фактическую величину выборки, участвующей в расчете для определения оптимальных параметров фильтрации. При подаче активного логического сигнала на вход балансировки

значение входного сигнала пробрасывается на выход блока без фильтрации. Раздел библиотеки: ***Обработка сигналов***.

Назначение входов и выходов:

in – вход;

W – количество значений входного сигнала для расчета (должно быть меньше **mW**);

mW – максимально количество значений входного сигнала для расчета, неизменяемый вход, значение которого принимается один раз при компиляции, допустимые значения в диапазоне **2…255**;

bal – максимально количество значений входного сигнала для расчета, неизменяемый вход, значение которого принимается один раз при компиляции, допустимые значения в диапазоне **2…255**;

out – выход.

Инициализация:

```
out = in
```

Алгоритм:

```
out = 1/W·∑in_i
```

```
где in_i – исходные сохраненные за предыдущие периоды значения входного сигнала,
W = k - l – сглаживающий интервал – количество значений входного сигнала,
используемых для расчета.
```

```
Если bal = 1 → out = in
```

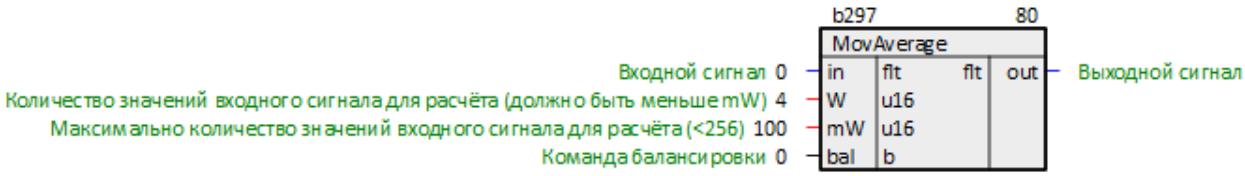


Рисунок 10.101 – Арифметическое скользящее среднее (MovAverage)

10.1.8Операции с регистрами

10.1.8.1 Объединение 20 битов в регистр (ToReg20)

Блок ***ToReg20*** объединяет **20 битов** в **регистр**. **Бит 0** – младший бит регистра, **бит 19** – старший. Раздел библиотеки: ***Операции с регистрами***.

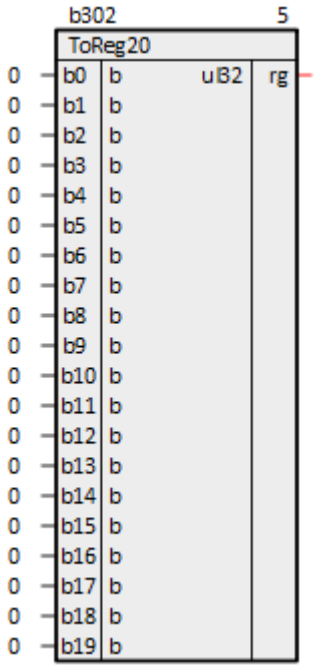


Рисунок 10.102 – Объединение 20 битов в регистр (ToReg20)

10.1.8.2 Объединение 8-ми битов в регистр (ToReg8)

Блок ***ToReg8*** объединяет **8 битов** в **регистр**. **Бит 0** – младший бит регистра, **бит 7** – старший. Раздел библиотеки: ***Операции с регистрами***.

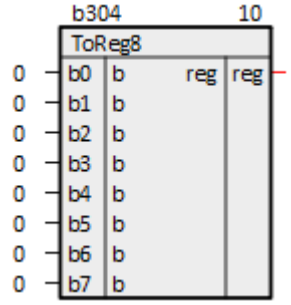


Рисунок 10.103 – Объединение 8-ми битов в регистр (ToReg8)

10.1.8.3 Выбор 8 битов из регистра (FromReg8)

Блок ***FromReg8*** раскладывает **байт** на **8 бит**. Первый выходной бит будет младшим битом байта, последний – старшим. Раздел библиотеки: ***Операции с регистрами***.

Рисунок 10.104 – Выбор 8 битов из регистра (FromReg8)

10.1.8.4 Объединение 16-ти битов в регистр (ToReg16)

Блок ***ToReg16*** объединяет **в слово 16 бит**. Первый входной бит будет младшим битом слова, последний – старшим. Раздел библиотеки: ***Операции с регистрами***.

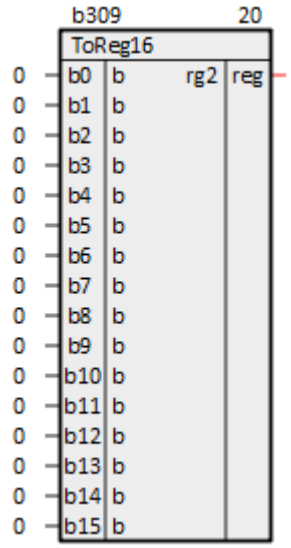


Рисунок 10.105 – Объединение 16-ти битов в регистр (ToReg16)

10.1.8.5 Выбор 20 битов из регистра (FromReg20)

Блок ***FromReg20*** раскладывает **4 байта** в **20 бит**. Первый выходной бит будет младшим битом слова, последний – старшим. Раздел библиотеки: ***Операции с регистрами***.

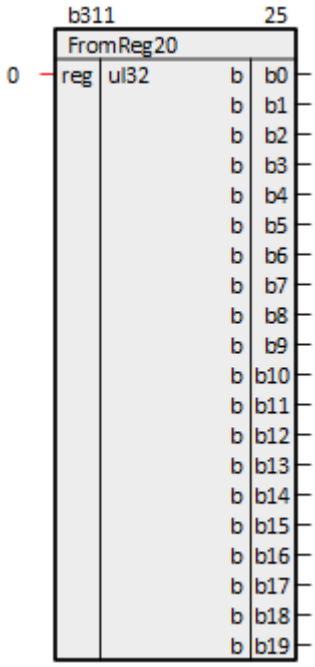


Рисунок 10.106 – Выбор 20 битов из регистра (FromReg20)

10.1.8.6 Выбор 16 битов из регистра (FromReg16)

Блок ***FromReg16*** раскладывает **слово в 16 бит**. Первый выходной бит будет младшим битом слова, последний – старшим. Раздел библиотеки: ***Операции с регистрами***.

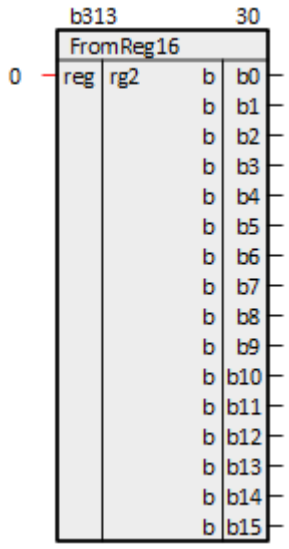


Рисунок 10.107 – Выбор 16 битов из регистра (FromReg16)

10.1.8.7 Логическое И для 16-битных регистров (rAND)

Функциональный блок ***rAND*** работает аналогично блоку И, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Операции с регистрами***.

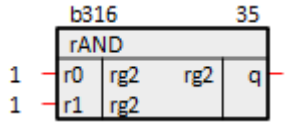


Рисунок 10.108 – Логическое И для 16-битных регистров (rAND)

10.1.8.8 Логическое ИЛИ для 16-битных регистров (rOR)

Функциональный блок ***rOR*** работает аналогично блоку ИЛИ, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Операции с регистрами***.

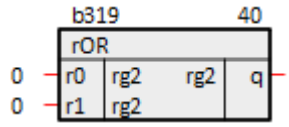


Рисунок 10.109 – Логическое ИЛИ для 16-битных регистров (rOR)

10.1.8.9 Логическое НЕ для 16-битных регистров (rNOT)

Функциональный блок ***rNOT*** работает аналогично блоку НЕ, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Раздел библиотеки: ***Операции с регистрами***.

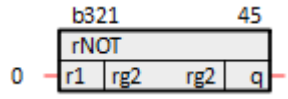


Рисунок 10.110 – Логическое НЕ для 16-битных регистров (rNOT)

10.1.8.10 Логическое И с отрицанием для 16-битных регистров (rnAND)

Функциональный блок ***rAND*** работает аналогично блоку И с отрицанием, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Число

входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Операции с регистрами***.

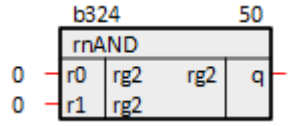


Рисунок 10.111 – Логическое И с отрицанием для 16-битных регистров (rNAND)

10.1.8.11 Логическое ИЛИ с отрицанием для 16-битных регистров (rnOR)

Функциональный блок ***rNOR*** работает аналогично блоку ИЛИ с отрицанием, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Операции с регистрами***.

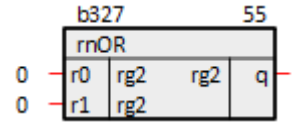


Рисунок 10.112 – Логическое ИЛИ с отрицанием для 16-битных регистров (rNOR)

10.1.8.12 Сложение по модулю для 16-битных регистров (rXOR)

Функциональный блок ***rXOR*** работает аналогично блоку XOR, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Раздел библиотеки: ***Операции с регистрами***.

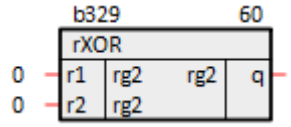


Рисунок 10.113 – Сложение по модулю для 16-битных регистров (rXOR)

10.1.8.13 Склеивание байта из двух тетрад (R4to8)

Блок ***R4to8*** собирает **8-ми битный регистр** из младшей тетрады регистра **low** и старшей тетрады регистра **high**. Раздел библиотеки: ***Операции с регистрами***.

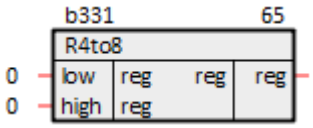


Рисунок 10.114 – Склеивание байта из двух тетрад (R4to8)

10.1.8.14 Детектор фронтов для 16-битных регистров (Fronts2)

Функциональный блок ***Fronts2*** работает аналогично блоку детектора фронтов, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Раздел библиотеки: ***Операции с регистрами***.

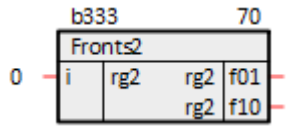


Рисунок 10.115 – Детектор фронтов для 16-битных регистров (Fronts2)

10.1.8.15 Склеивание слова из двух байт (R8to16)

Блок ***R8to16*** собирает **16-ти битный регистр** из младшего регистра **low** и старшего регистра **high**. Раздел библиотеки: ***Операции с регистрами***.

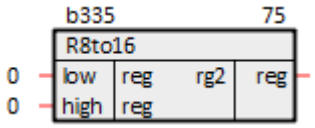


Рисунок 10.116 – Склеивание слова из двух байт (R8to16)

10.1.8.16 Преобразование 16-битного регистра в два 8-битных (R16to8)

Блок ***R16to8*** разбивает **16-битный регистр** на **два 8-битных**. Раздел библиотеки: ***Операции с регистрами***.

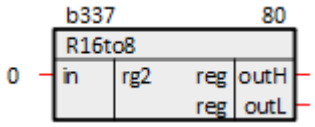


Рисунок 10.117 – Преобразование 16-битного регистра в два 8-битных (R16to8)

10.1.8.17 Преобразование двух 16-битных регистров в 32-битный (R16to32)

Блок ***R16to32*** склеивает **32-битный регистр** из **двух 16-битных**. Раздел библиотеки: ***Операции с регистрами***.

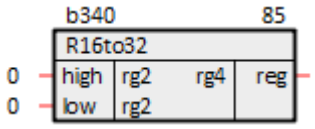


Рисунок 10.118 – Преобразование двух 16-битных регистров в 32-битный (R16to32)

10.1.8.18 Преобразование 32-битного регистра в два 16-битных (R32to16)

Блок ***R32to16*** разбивает **32-битный регистр** на **два 16-битных**. Раздел библиотеки: ***Операции с регистрами***.

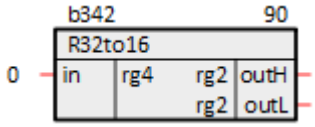


Рисунок 10.119 – Преобразование 32-битного регистра в два 16-битных (R32to16)

10.1.8.19 Объединение 32 битов в регистр (ToReg32)

Блок ***ToReg32*** объединяет **32 бита** в **регистр**. **Бит 0** – младший бит регистра, **бит 31** – старший. Раздел библиотеки: ***Операции с регистрами***.

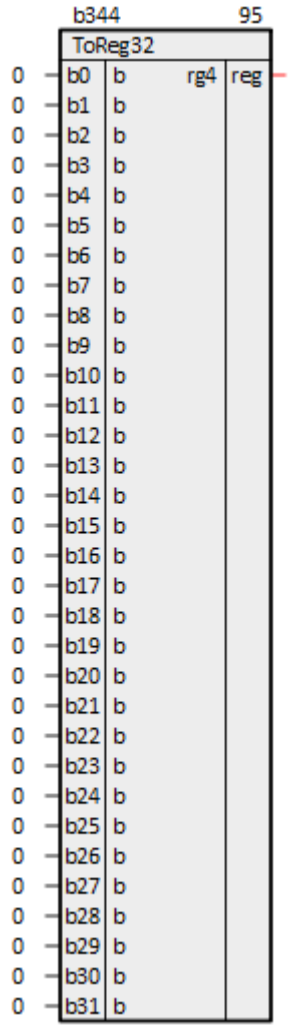


Рисунок 10.120 – Объединение 32 битов в регистр (ToReg32)

10.1.8.20 Выбор 32 битов из регистра (FromReg32)

Блок ***FromReg32*** раскладывает **4 байта** в **32 бита**. Первый выходной бит будет младшим битом слова, последний – старшим. Раздел библиотеки: ***Операции с регистрами***.

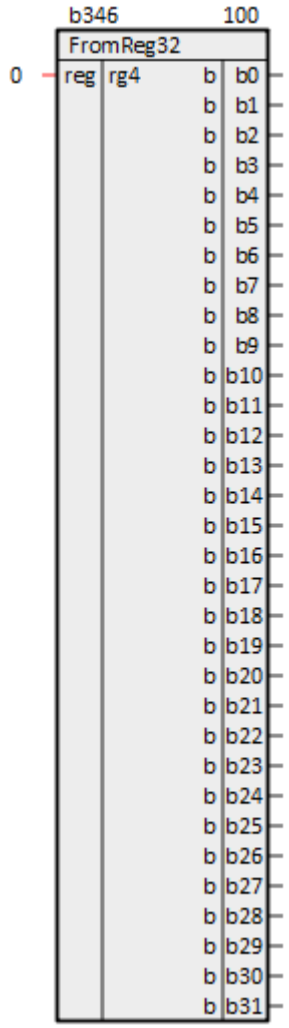


Рисунок 10.121 – Выбор 32 битов из регистра (FromReg32)

10.1.8.21 Включение с задержкой для 16-битных регистров (rDelayOn)

Функциональный блок ***rDelayOn*** работает аналогично блоку Включение с задержкой, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Раздел библиотеки: ***Операции с регистрами***.

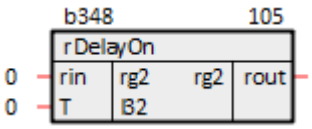


Рисунок 10.122 – Включение с задержкой для 16-битных регистров (rDelayOn)

10.1.8.22 Выключение с задержкой для 16-битных регистров (rDelayOff)

Функциональный блок ***rDelayOff*** работает аналогично блоку Выключение с задержкой, но обрабатывает параллельно **16** логических сигналов, предварительно объединенных в **16-битный регистр**. Раздел библиотеки: ***Операции с регистрами***.

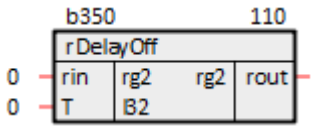


Рисунок 10.123 – Выключение с задержкой для 16-битных регистров (rDelayOff)

10.1.8.23 Выбор бита из регистра (BitReg16)

Функциональный блок ***BitReg16*** выбирает бит из регистра **ireg** по указанному номеру **ibit**. Раздел библиотеки: ***Операции с регистрами***.

Назначение входов и выходов:

ireg – регистр;
ibit – номер бита;
obit – выходной бит.

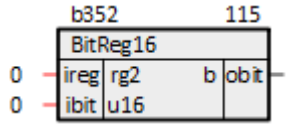


Рисунок 10.124 – Выбор бита из регистра (BitReg16)

10.1.9Операции с массивами

10.1.9.1 Массив вещественных чисел (FloatArray)

Блок ***FloatArray*** создает массив из элементов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Операции с массивами***.

Назначение входов и выходов:

in0, **in1**, **…**, **inn** – элементы типа flt;
p_a – массив.

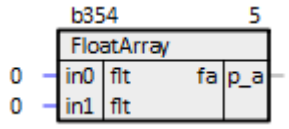


Рисунок 10.125 – Массив вещественных чисел (FloatArray)

10.1.9.2 Массив 4-байтных целых (i32Array)

Блок ***i32Array*** создает массив из элементов. Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Операции с массивами***.

Назначение входов и выходов:

in0, **in1**, **…**, **inn** – элементы типа i32;
p_a – массив.

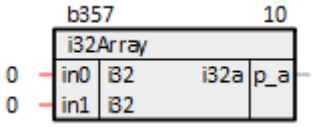


Рисунок 10.126 – Массив 4-байтных целых (i32Array)

10.1.9.3 Линейная аппроксимация массивов (LineAprox)

Блок ***LineAprox*** рассчитывает значение таблично заданных функций при значении аргумента **inp**. Текущие значения функций выдаются на соответствующие выходы **o**. Функция задается массивом аргументов **arg** (общим для всех функций) и массивом значений **fun**. Размерности всех массивов должны совпадать, иначе на выход будет выдана ошибка **err - (-1 - [номер функции]).**

Массив аргументов **arg** должен быть упорядочен либо по возрастанию, либо по убыванию. Если порядок нарушен, то на выход **err** выдается номер этого элемента (считая с **1**). Число входов задается при создании функционального блока, и может быть изменено в процессе редактирования программы. Раздел библиотеки: ***Операции с массивами***.

ВНИМАНИЕ
На входы **arg** и **fun** должны быть проведены связи только с выходов блоков Массив вещественных чисел, в противном случае работа программы будет некорректной.

Назначение входов и выходов:

inp – аргумент;
arg – аргументы;
fun0, **fun1**, **…**, **funn** – функции;
err – ошибка;
o0, **o1**, **…**, **on** – значения функций.

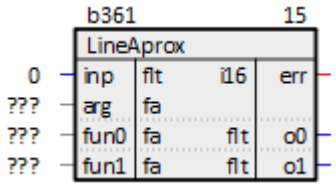


Рисунок 10.127 – Линейная аппроксимация массивов (LineAprox)

10.1.9.4 Значения массива вещественных чисел (From_fArr)

Блок ***From_fArr*** позволяет получить значения элементов вещественного массива, поданного на вход. Раздел библиотеки: ***Операции с массивами***.

Коды ошибок статуса **err**: **0** – ошибок нет; **1** – выходов блока меньше, чем элементов у входного массива; **2** – выходов больше, чем элементов у входного массива; **3** – на входе отсутствует массив (может быть связано с порядком выполнения блоков).

Назначение входов и выходов:

in – массив элементов типа flt;
err – статус;
out0, **out1**, **…**, **outn** – значения элементов массива.

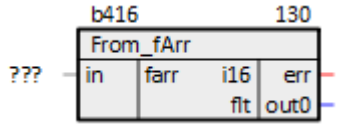


Рисунок 10.128 – Значения массива вещественных чисел (From_fArr)

10.1.9.5 Элемент из массива (one_From_fArr)

Блок ***one_From_fArr*** позволяет получить значение **i**-го элемента из массива, поданного на вход. Раздел библиотеки: ***Операции с массивами***.

Коды ошибок статуса **err**: **0** – ошибок нет; **1** – выбранный номер элемента находится за пределами массива; **2** – на входе отсутствует массив (может быть связано с порядком выполнения блоков).

Назначение входов и выходов:

in – массив элементов типа flt;
i – номер элемента;
err – статус;
out – значения элемента.

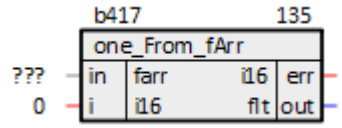


Рисунок 10.129 – Элемент из массива (one_From_fArr)

10.1.9.6 Нахождение минимума/максимума в массиве (minmax_fArr)

Блок ***minmax_fArr*** определяет минимальное и максимальное значение из элементов массива поданного на вход. Раздел библиотеки: ***Операции с массивами***.

Коды ошибок статуса **err**: **0** – ошибок нет; **1** – на входе отсутствует массив (может быть связано с порядком выполнения блоков).

Назначение входов и выходов:

in – массив элементов типа flt;

err – статус;

min – минимум;

max – максимум.

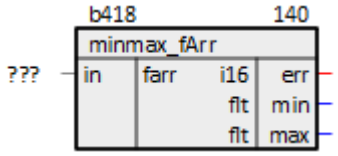


Рисунок 10.130 – Нахождение минимума-максимума в массиве (minmax_fArr)

10.1.9.7 Нахождение разницы между элементами массива (minmax_sub_fArr)

Блок ***minmax_sub_fArr*** минимальное и максимальное значение разницы *соседних* элементов массива, поданного на вход. Раздел библиотеки: ***Операции с массивами***.

Коды ошибок статуса **err**: **0** – ошибок нет; **1** – на входе отсутствует массив (может быть связано с порядком выполнения блоков).

Назначение входов и выходов:

in – массив элементов типа flt;

err – статус;

min – минимальная разница соседей;

max – максимальная разница соседей.

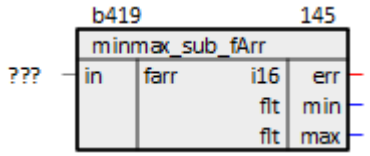


Рисунок 10.131 – Нахождение минимума/максимума разницы между элементами массива (minmax_sub_fArr)

10.1.9.8 Переключатель массивов (switch_fArr)

Блок ***switch_fArr*** принимает на входы два массива вещественных чисел и выдает на выход один из них, в зависимости от состояния управляющего входа **i**: при **i = 0** — массив **in2**, в остальных случаях — массив **in1**. Раздел библиотеки: ***Операции с массивами***.

Коды ошибок статуса **err**: **0** – ошибок нет; **1** – на входе отсутствует массив (может быть связано с порядком выполнения блоков).

Назначение входов и выходов:

in1, **in2** – массивы элементов типа flt;

err – статус;

p_a – выходной массив.

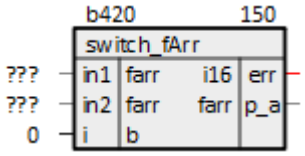


Рисунок 10.132 – Переключатель массивов (switch_fArr)

10.1.9.9 Значения массива целых (From_i32Arr)

Блок ***From_i32Arr*** позволяет получить значения элементов целочисленного массива, поданного на вход. Раздел библиотеки: ***Операции с массивами***.

Коды ошибок статуса **err**: **0** – ошибок нет; **1** – выходов блока меньше, чем элементов у входного массива; **2** – выходов больше, чем элементов у входного массива; **3** – на входе отсутствует массив (может быть связано с порядком выполнения блоков).

Назначение входов и выходов:

in – массивы элементов типа i32;

err – статус;

out0, **out1**, **…**, **outn** – значения элементов массива.

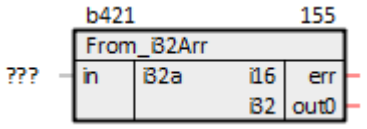


Рисунок 10.133 – Значения массива целых (From_i32Arr)

10.1.10 Триггеры

10.1.10.1 Триггер Шмидта (SmTrig)

Блок ***SmTrig*** осуществляет алгоритм симметричного триггера Шмидта. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

x – аргумент;
m – уставка;
dm – гистерезис;
fu – больше;
fl – меньше.

Алгоритм:

`fl = 1 при x ≤ m + dm,`

`fu = 1 при x > m + dm`

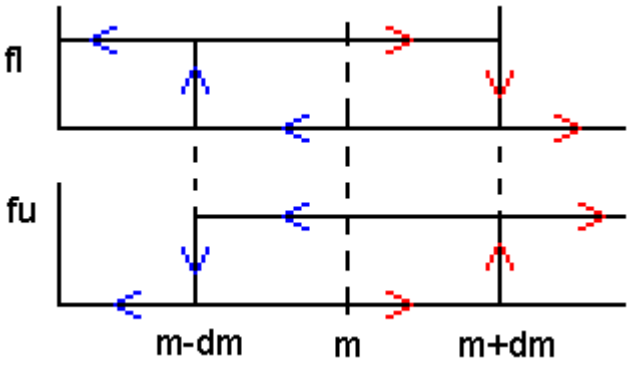


Рисунок 10.134 – Алгоритм SmTrig

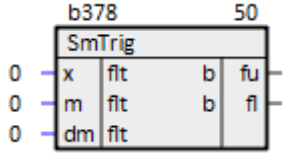


Рисунок 10.135 – Триггер Шмидта (SmTrig)

10.1.10.2 Двойной триггер Шмидта (SmTrig2)

Блок ***SmTrig*** осуществляет алгоритм двойного симметричного триггера Шмидта. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

x – аргумент;
m – центр уставки;
dm – гистерезис;
l – отклонение уставки;
fu – больше;
fm – между;
fl – меньше.

Алгоритм:

`fl = 1 при x < m - l + dm,`

`fm = 1 при m - l + dm ≤ x < m + l - dm,`

`fu = 1 при x ≥ m + l - dm`

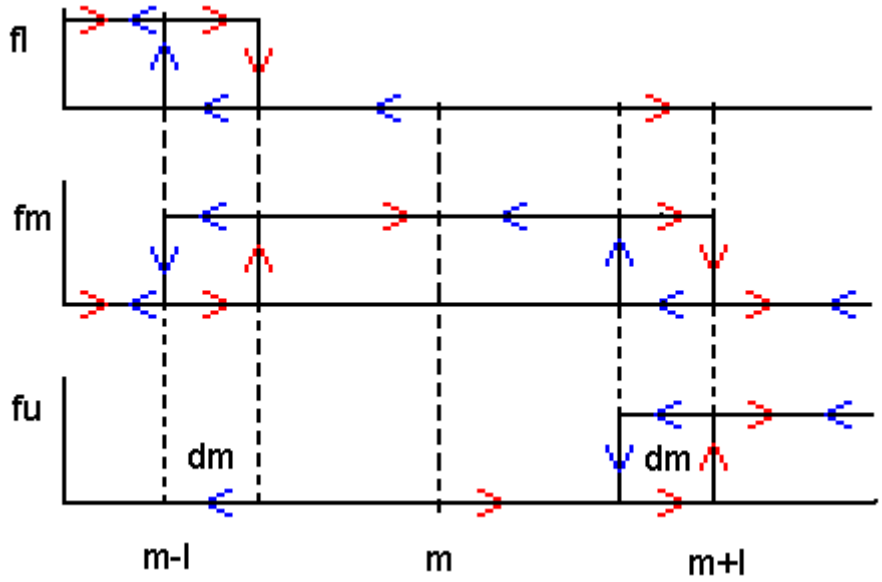


Рисунок 10.136 – Алгоритм SmTrig2

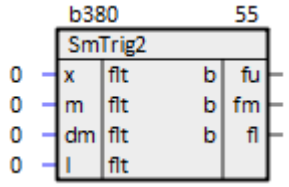


Рисунок 10.137 – Двойной триггер Шмидта (SmTrig2)

10.1.10.3 RSS-триггер (RSS)

RSS – RS-триггер с S-доминантой. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

s – бит установить (set);
r – бит сбросить (reset);
q – выход;
inv – инверсный выход.

Инициализация:

```
q = 0
```

Алгоритм:

s	r	q
0	0	пред. знач.
0	1	0
1	0	1
1	1	1

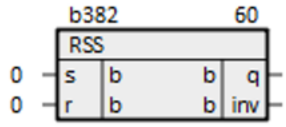


Рисунок 10.138 – RSS-триггер (RSS)

10.1.10.4 RSR-триггер (RSR)

RSR – RS-триггер с R-доминантой. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

s – бит установить (set);
r – бит сбросить (reset);
q – выход;
inv – инверсный выход.

Инициализация:

```
q = 0
```

Алгоритм:

s	r	q
0	0	пред. знач.
0	1	0
1	0	1
1	1	0

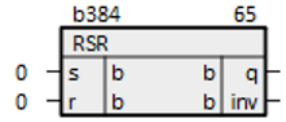


Рисунок 10.139 – RSR-триггер (RSR)

10.1.10.5 D-триггер с инициализацией выхода (DFR_i)

DFR_i – D-триггер с R- доминантой. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

i – перепад;
d – потенциал;
s – бит установить (set);
r – бит сбросить (reset);
ini – инициализация;
q – выход;
qn – инверсный выход.

Инициализация:

```
q = ini
```

Алгоритм:

d	i	s	r	q	qn
0	0→1	0	0	0	1
1	0→1	0	0	1	0
любой	1→0	0	0	пред. знач.	пред. знач.
любой	любой	0	1	0	1
любой	любой	1	0	1	0
любой	любой	1	1	0	1

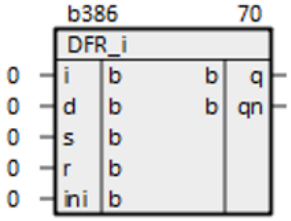


Рисунок 10.140 – D-триггер с инициализацией выхода (DFR_i)

10.1.10.6 JK-триггер с инициализацией (JK _i)

JK _i – JK-триггер с инициализацией. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

j – бит job;
k – бит keep;
s – бит установить (set);
r – бит сбросить (reset);
ini – начальное значение **q**;
q – выход;
qn – инверсный выход.

Инициализация:

```
q = ini
```

Алгоритм:

j	k	s	r	q	qn
0→1	0	0	0	0	1
0→1	1	0	0	1	0
0	0→1	0	0	0	1
1	0→1	0	0	1	0
any	any	0	1	0	1
any	any	1	0	1	0
any	any	1	1	0	1

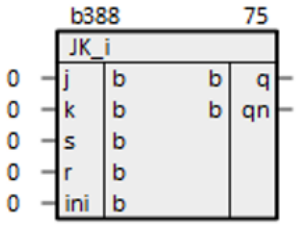


Рисунок 10.141 – JK-триггер с инициализацией (JK _i)

10.1.10.7 RSS-триггер с инициализацией выхода (RSS_i)

RSS – RS-триггер с S-доминантой с инициализацией. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

s – бит установить (set);
r – бит сбросить (reset);
i – начальное значение **q**;
q – выход.

Инициализация:

```
q = i
```

Алгоритм:

s	r	q
0	0	пред. знач.
0	1	0
1	0	1
1	1	1

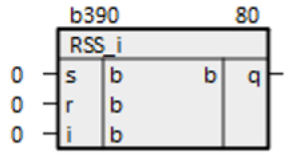


Рисунок 10.142 – RSS-триггер с инициализацией (RSS_i)

10.1.10.8 RSR-триггер с инициализацией выхода (RSR)

RSR – RS-триггер с R-доминантой с инициализацией. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

s – бит установить (set);
r – бит сбросить (reset);
i – начальное значение **q**;
q – выход.

Инициализация:

```
q = i
```

Алгоритм:

s	r	q
0	0	пред. знач.
0	1	0
1	0	1
1	1	0

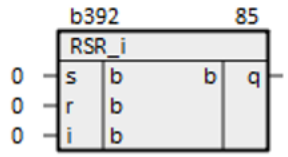


Рисунок 10.143 – RSR-триггер (RSR)

10.1.10.9 D-триггер (DFR)

DFR – D-триггер с R- доминантой. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

i – перепад;
d – потенциал;
s – бит установить (set);
r – бит сбросить (reset);
q – выход;
qn – инверсный выход.

Инициализация:

```
q = 0
```

Алгоритм:

d	i	s	r	q	qn
0	0→1	0	0	0	1
1	0→1	0	0	1	0
любой	1→0	0	0	пред. знач.	пред. знач.
любой	любой	0	1	0	1
любой	любой	1	0	1	0
любой	любой	1	1	0	1

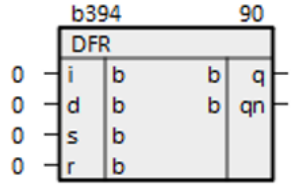


Рисунок 10.144 – D-триггер (DFR)

10.1.10.10 DV-триггер (DFRV)

DFRV – DV-триггер с R- доминантой. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

is – потенциал;
i – перепад;
s – бит установить (set);
r – бит сбросить (reset);
q – выход;
qn – инверсный выход.

Инициализация:

```
q = 0
```

Алгоритм:

i	s	r	q	qn
0→1	0	0	is	!is
любой	0	1	0	1
любой	1	0	1	0
любой	1	1	0	1

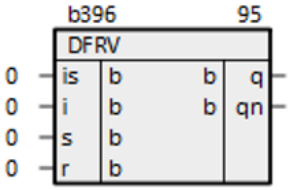


Рисунок 10.145 – DV-триггер (DFRV)

10.1.10.11 Несимметричный триггер Шмидта (asymSmTrig)

Блок ***SmTrig*** осуществляет алгоритм несимметричного триггера Шмидта. Раздел библиотеки: ***Триггеры***.

Назначение входов и выходов:

x – аргумент;
min – нижний гистерезис;
max – верхний гистерезис;
l – отклонение уставки;
fu – больше;
fl – меньше.

Инициализация:

```
fu = 0, fl = 0
```

Алгоритм: первые три цикла **fu = 1** при **x > max**, **fl = 1** при **x ≤ max**, затем см. рисунок ниже.

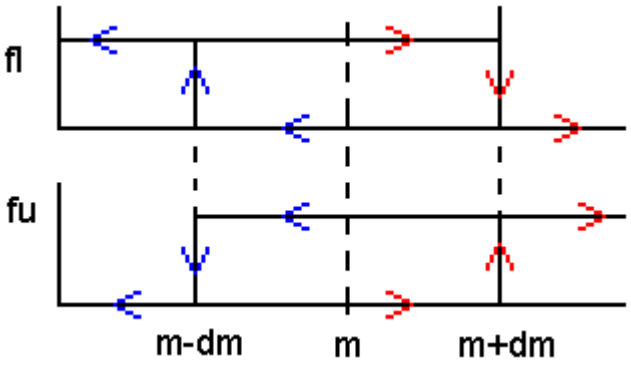


Рисунок 10.146 – Рисунок 11.1.147 – Алгоритм asymSmTrig

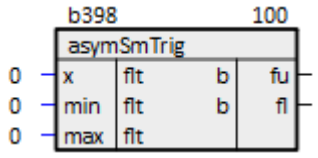


Рисунок 10.147 – Несимметричный триггер Шмидта (asymSmTrig)

10.1.10.12 Шаговая последовательность (Steper)

Блок ***Steper*** осуществляет последовательное переключение заданного количества шагов. Раздел библиотеки: ***Триггеры***.

Таблица 10.3 – Назначение входов и выходов Steper

Входы	
enb	Разрешение работы
str	Пуск
rst	Сброс
t_f	Не используется
i	Разрешение (циклический)
t	Таймаут (циклический)
a	Сигнал (циклический)
Выходы	
ind	Номер шага
s_w	Идет работа
s_t	Таймаут
s_r	Сброс
s_f	Конец работы
w	Признак работы (циклический)
t	Признак таймаута (циклический)
r	Авария (циклический)

Работа последовательности возможна только при наличии **1** на входе разрешения **enb**:

- если **enb = 0** до запуска, то запуск не произойдет;
- если **enb = 0** после запуска, то последовательность сбрасывается и на выходы выдается авария: **s_r = 1** и **ri = 1** (для всех **i**).

Запуск последовательности происходит при появлении перепада **0 → 1** на входе **str**.

Каждый шаг представлен группой входов **i**, **t** и **a** и выходов **w**, **t** и **r**. Включение шага происходит после завершения предыдущего шага и при **i = 1**, при этом на выходе признака работы **w** выставляется **1**. Если к моменту включения шага его разрешение **i** не равно **1**, то последовательность сбрасывается и на выходы выдается авария: **s_r = 1** и **ri = 1** (для всех ***i***). Завершение работы шага происходит при получении **1** на входе ответного сигнала **a**. Если сигнал **a** не получен в течение времени **t** (задается в мс) с момента включения шага, то последовательность сбрасывается и на выход выдается признак таймаута: **s_t = 1** и **t = 1**.

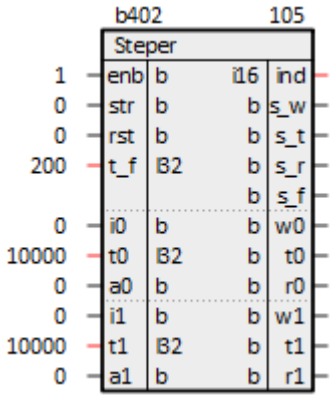


Рисунок 10.148 – Шаговая последовательность (Steper)

10.1.11 Системные

В данном разделе приведено описание основных блоков библиотеки ***paCore*** из раздела ***Системные***.

10.1.11.1 Информация о временах выполнения (SysInfo)

Блок ***SysInfo*** выдает информацию о временах выполнения. Раздел библиотеки: ***Системные***.

Таблица 10.4 – Назначение входов и выходов SaverEx

Входы	
fmap	Имя map-файла
rst	Сброс
Выходы	
init	Не используется
IT	Заданный таймерный цикл ввода-вывода
ic	Текущий таймерный цикл ввода-вывода
im	Максимальный таймерный цикл ввода-вывода
MT	Заданный таймерный цикл
tc	Текущий таймерный цикл
tm	Максимальный таймерный цикл
fc	Текущий фон
fm	Максимальный фон
crc	Не используется
inm0	Не используется (циклический)
cnt0	Не используется (циклический)

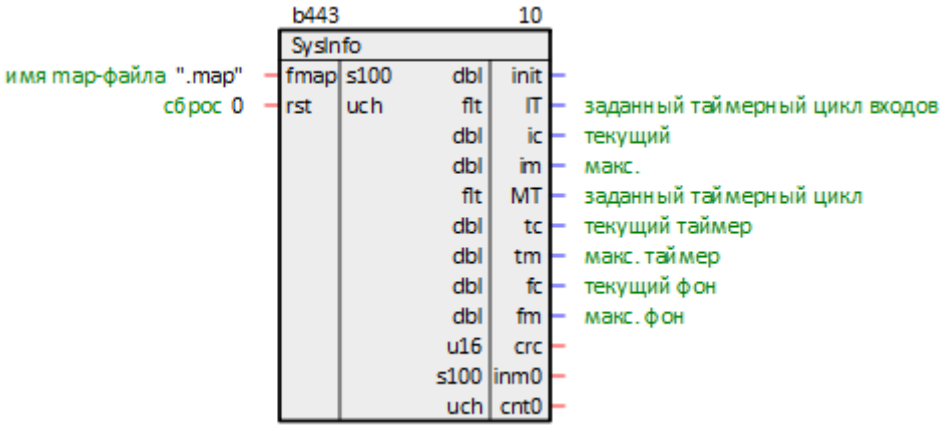


Рисунок 10.149 – Информация о временах выполнения (SysInfo)

10.1.11.2 Получить время и дату, день недели (getTDN)

Блок ***getTDN*** выдает на выходы текущее время и дату, день недели, установленные в ОС контроллера. Блок можно размещать как в ***Таймере***, так и в ***Фоне***. Раздел библиотеки: ***Системные***.

Назначение входов и выходов:

f – не используется;

h – часы;

m – минуты;

s – секунды;

dw – день недели;

d – число;

mn – месяц;

y – год.

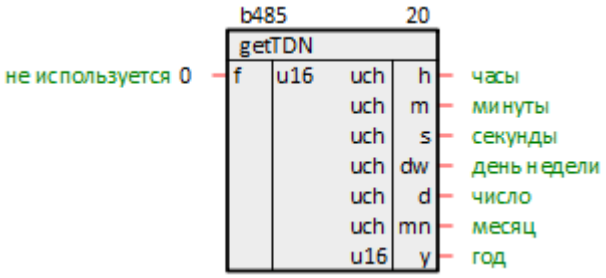


Рисунок 10.150 – Получить время и дату, день недели (getTDN)

10.1.11.3 Получить время и дату (getTD)

Блок ***getTD*** выдает на выходы текущее время и дату, установленные в ОС контроллера. Блок можно размещать как в ***Таймере***, так и в ***Фоне***. Раздел библиотеки: ***Системные***.

Назначение входов и выходов:

f – не используется;

h – часы;

m – минуты;

s – секунды;

d – число;

mn – месяц;

y – год.

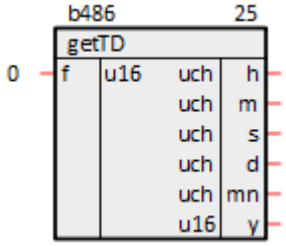


Рисунок 10.151 – Получить время и дату (getTD)

10.1.11.4 Завершение программы (Exit)

Блок ***Exit*** осуществляет корректное завершение программы, когда на входе **b1** появляется **1**. Раздел библиотеки: ***Системные***.

Назначение входов и выходов:

b1 – выйти;

q – значение входа.

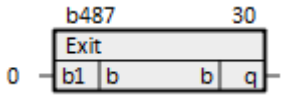


Рисунок 10.152 – Завершение программы (Exit)

10.1.11.5 Разность тиков процессора (GetTicksDiff)

Блок ***GetTicksDiff*** предназначен для проведения высокоэффективных измерений малых временных промежутков. Раздел библиотеки: ***Системные***.

Блок определяет разность между двумя входными значениями **start** и **end** (полученными при помощи блоков Счетчик тиков процессора), переводя ее в единицы измерения времени (мс) на выходе **ms**.

Выход **min** отображает длительность одного тика в миллисекундах.

ВНИМАНИЕ

Для получения корректного результата последовательность выполнения блоков должна удовлетворять следующим условиям: ***GetTicks*(start) < *GetTicks*(end) < *GetTicksDiff***.

Назначение входов и выходов:

start – тик начала;

end – тик окончания;

ms – разница между входами, мс;

pday – не используется;

min – длительность тика, мс.

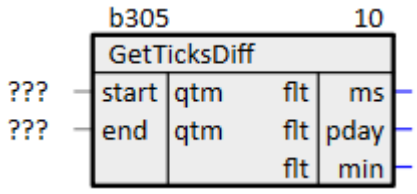


Рисунок 10.153 – Разность тиков процессора (GetTicksDiff)

10.1.11.6 Счетчик тиков процессора (GetTicks)

Блок ***GetTicks*** в связке с блоком Разность тиков процессора позволяет выполнять высокоэффективные измерения малых временных промежутков. Раздел библиотеки: ***Системные***.

Если на вход блока **enb** подан ненулевой сигнал, то на выходе **tcks** текущее значение внутреннего независимого цикличного счетчика.

При нулевом сигнале на входе, на выходе фиксируется последнее полученное значение счетчика (если измерений не было – **0**).

Назначение входов и выходов:

end – включение;

tcks – тик.

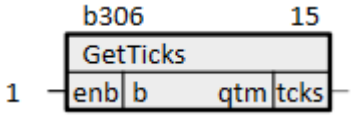


Рисунок 10.154 – Счетчик тиков процессора (GetTicks)

10.1.11.7 Загруженность таймерного потока (Effect)

Блок ***Effect*** выдает информацию о загруженности таймерного потока. Раздел библиотеки: ***Системные***.

Таблица 10.5 – Назначение входов и выходов Effect

Входы	
rst	Сброс
Выходы	
mt	Таймерный цикл основной

Продолжение таблицы 10.5

it	Таймерный цикл ввода-вывода
ctm	Средний процент заполнения таймерного цикла основного программой таймера
cti	Средний процент заполнения ввода-вывода цикла программой ввода-вывода
mtm	Средний процент заполнения ввода-вывода цикла программой ввода-вывода
mti	Максимальный процент заполнения ввода-вывода цикла программой ввода-вывода
cf	Средний процент от величины таймера работы фоновой программы
mf	Максимальный процент от величины таймера работы фоновой программы

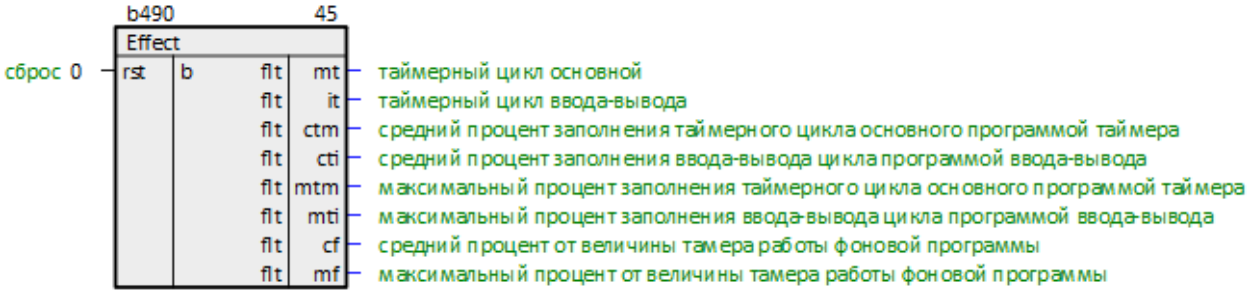


Рисунок 10.155 – Загруженность таймерного потока (Effect)

10.1.11.8 Событие по фронту (Event)

Блок ***Event*** предназначен для определения времени между событиями. На вход блока **front** заводится бит события. По входному фронту на выходе **time** блока будет значение времени в наносекундах. Раздел библиотеки: ***Системные***.

Назначение входов и выходов:

front – входной фронт;

me – событие;

time – время срабатывания, нс.

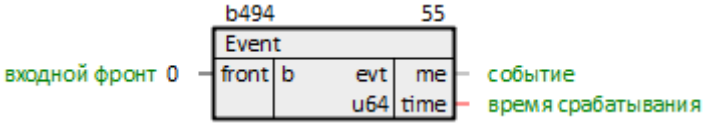


Рисунок 10.156 – Событие по фронту (Event)

10.1.11.9 TCP/IP клиент (TcpIpClA)

Блок ***TcpIpClA*** представляет собой TCP/IP-клиент для обеспечения работы протоколов (например, **Modbus TCP Master**). Раздел библиотеки: ***Системные***.

Так как работа блока занимает значительное время, может быть размещен только в ***Фоне***.

Назначение входов и выходов:

lprt – локальный порт;

lip – локальный IP адрес;

sdr – сетевой стек, для ПЛК ОВЕН **"/"**;

rprt – удаленный порт;

ip – IP адрес удаленного сервера;

cnc – связь с блоком протокола;

stat – статус работы (**0** – есть связь с TCP/IP сервером, **>0** – нет связи).

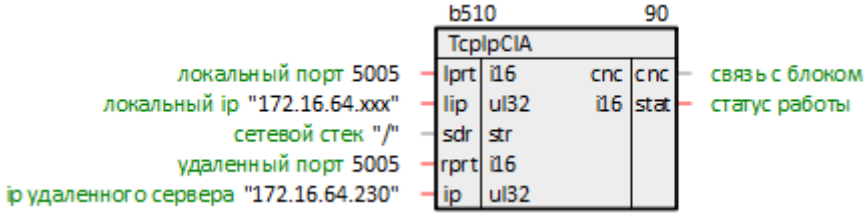


Рисунок 10.157 – TCP/IP клиент (TcpIpClA)

10.1.11.10 TCP/IP сервер (TcpIpSrA)

Блок ***TcpIpSrA*** представляет собой TCP/IP-сервер для обеспечения работы протоколов (например, **Modbus TCP Slave**). Сервер поддерживает одновременно не более **20** подключений. Раздел библиотеки: ***Системные***.

Назначение входов и выходов:

prt – порт;

lip – локальный IP адрес;

sdr – сетевой стек, для ПЛК ОВЕН **"/"**;

wait – время в миллисекундах до закрытия пустого канала (**0** – никогда);

cnc – связь с блоком протокола;

stat – статус работы (**0** – есть подключения, **>0** – нет подключений).

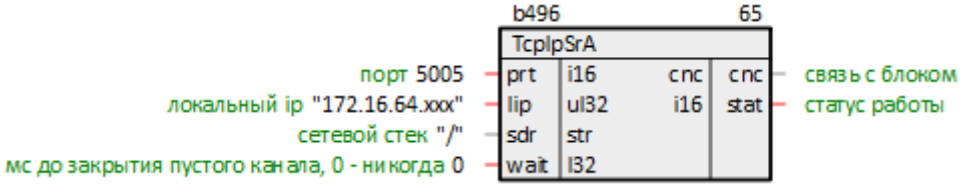


Рисунок 10.158 – TCP/IP сервер (TcpIpSrA)

10.1.11.11 Информация о сборке (BuildVersionInfo)

Блок ***BuildVersionInfo*** передает на свои выходы информацию о сборке. Раздел библиотеки: ***Системные***.

Назначение выходов:

ver – версия проекта;

date – дата и время трансляции;
guid – уникальный идентификатор (**GUID**);
usr – имя пользователя;
pc – имя компьютера.

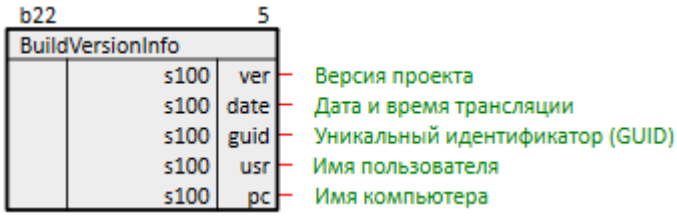


Рисунок 10.159 – Информация о сборке (BuildVersionInfo)

10.1.11.12 Информация о накопителе (DriveInfo)

Функциональный блок ***DriveInfo*** используется для получения информации о накопителе по имени (адресу устройства). Раздел библиотеки: ***Системные***.

По факту наличия восходящего фронта на вход **start** ФБ выполняет разовую выдачу информации о накопителях.

Для цикличного обновления информации о накопителе необходимо обеспечить цикличные подачу восходящего фронта на вход **start**.

Назначение входов и выходов:

start – запуск работы блока по восходящему фронту (**0 → 1**);
driveName – адрес накопителя в системе (до **49** символов);
error – накопитель отсутствует или недоступен;
capacity – общий объем накопителя, Кбайт;
used – занятый объем накопителя, Кбайт;
available – доступный объем накопителя, Кбайт.

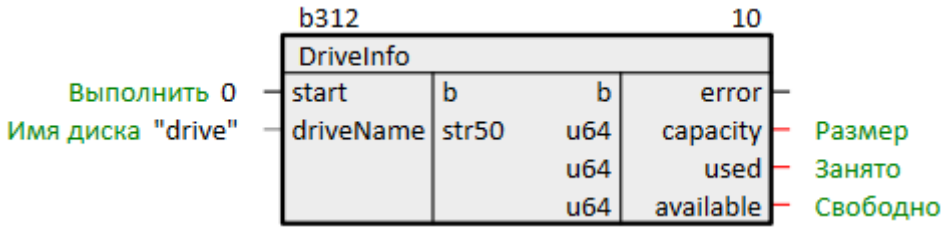


Рисунок 10.160 – Информация о накопителе (DriveInfo)

10.1.11.13 Асинхронное выполнение команд Linux (SysExecute)

Функциональный блок ***SysExecute*** используется для отправки команды в терминал Linux и получения ответа. Раздел библиотеки: ***Системные***.

Интерфейс данного блока соответствует **CAA Behavior Model** (PLCopen Behavior Model).

Бизнес-логика ФБ выполняется в отдельном потоке и не оказывает влияние на время работы циклов таймера и фона.

ПРИМЕЧАНИЕ
Вызываемые команды могут выполняется в неблокирующем режиме – т.е. поток может освобождаться после подачи команды и через определенное время забирать ее ответ. Для этого в конце команды необходимо добавить знак **'&'** (амперсанд).

В случае если результат выполнения команды содержит более **1023** символов, выполнение команды будет прервано, будут подняты флаги **done**, **error**, но результат работы будет содержаться в **output**.

Запуск выполнения команды не произойдет, если поднят флаг прерывания **abort**.

Назначение входов и выходов:

execute – запуск выполнения команды по восходящему фронту (**0 → 1**);
command – текст команды;
abort – запуск прерывания исполнения команды по восходящему фронту (**0 → 1**);
priority – приоритет потока ожидания выполнения команды, по умолчанию **5**;
output – результат выполнения команды (менее **1024** символов);
done – выполнение завершено;
aborted – выполнение было прервано;
busy – идет работа;
error – работа была прервана из-за переполнения результирующей строки.

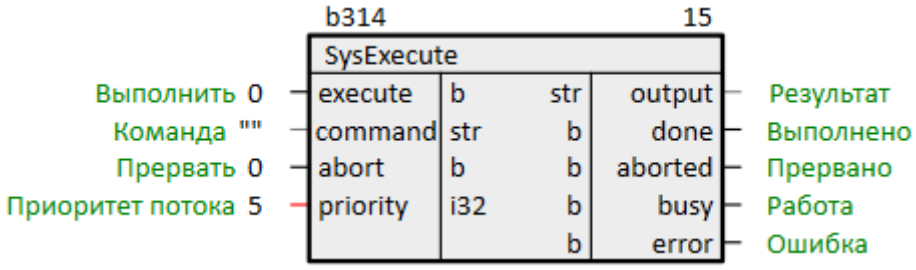


Рисунок 10.161 – Асинхронное выполнение команд Linux (SysExecute)

10.1.11.14 Менеджер потоков (ThreadMan)

Блок ***ThreadMan*** предоставляет возможности управления и диагностики фоновых потоков исполнительной системы. Раздел библиотеки: ***Системные***.

Назначение входов и выходов:

slp – время сна (простоя) фоновых потоков в мкс, в этот промежуток времени управление передается другим потокам; если задано значение **0** или блок ***ThreadMan*** не добавлен в проект, то время простоя задается таймерным промежутком места работы ***Таймер***;

btd – включение диагностики времени выполнения блоков: **0** – выключена, **1** – включена; если диагностика включена, то в отладчике у каждого блока отображается время его выполнения, если выключена, то время выполнения всегда отображается как 1 мкс, что означает, что блок работает; выключение диагностики может на некоторых контроллерах существенно оптимизировать время выполнения всей программы;

num – количество запущенных дополнительных фоновых потоков (не включает ***Таймер***, ***Ввод-вывод*** и ***Фон***);

prio_max – максимально разрешенный приоритет для дополнительных фоновых потоков, зависит от ОС и не может быть изменен; при использовании в проекте места работы ***Поток*** следует выбирать приоритет ниже **prio_max**;

pmin – минимальный приоритет из запущенных дополнительных фоновых потоков;

pmax – максимальный приоритет из запущенных дополнительных фоновых потоков.

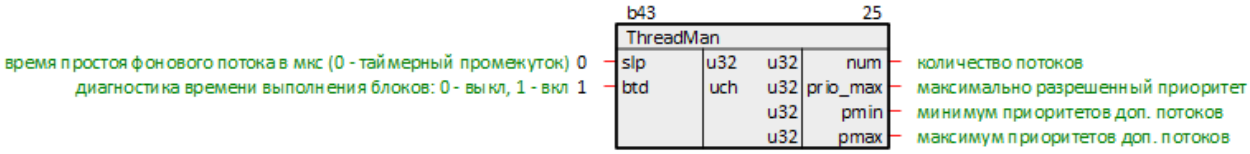


Рисунок 10.162 – Менеджер потоков (ThreadMan)

10.1.12 Работа со строками

10.1.12.1 Длина строки (StrLen)

Блок ***StrLen*** осуществляет подсчет количества символов **len** в строке **s**. Раздел библиотеки: ***Работа со строками***.

Назначение входов и выходов:

s – строка;

len – длина.

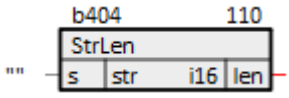


Рисунок 10.163 – Длина строки (StrLen)

10.1.12.2 Часть строки (StrNmp)

Блок ***StrNmp*** выделяет часть входной строки ***s*** начиная с позиции **n** и длинной **m** (1-й символ строки имеет позицию **0**). Если **n** и **m** заданы так, что длины исходной строки недостаточно, то выходная строка **res** ограничивается концом входной строки. Раздел библиотеки: ***Работа со строками***.

Назначение входов и выходов:

s – строка;

n – начало;

m – длина;

res – выходная строка.

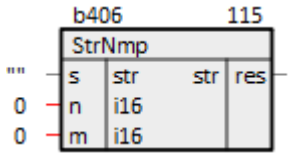


Рисунок 10.164 – Часть строки (StrNmp)

10.1.12.3 Слияние строк (StrApp)

Блок ***StrApp*** сливает входные строки **s** в одну результирующую строку **str**. В выходной строке строки расположены в том же порядке, что и входы **s**. Длина каждой строки **s** не должна превышать **60** символов. Раздел библиотеки: ***Работа со строками***.

Назначение входов и выходов:

s0, **s1**, **…**, **sn** – строки;

res – результирующая строка.

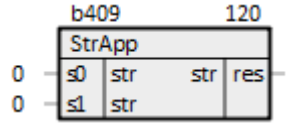


Рисунок 10.165 – Слияние строк (StrApp)

10.1.13 Сохранение данных

10.1.13.1 Хранение параметров на диске (SaverEx)

Блок ***SaverEx*** обеспечивает хранение данных в архиве на диске контроллера в виде бинарного файла. Раздел библиотеки: ***Сохранение данных***.

Поскольку операции файлового ввода/вывода занимают значительное время, данный блок следует размещать только в ***Фоне***.

Таблица 10.6 – Назначение входов и выходов SaverEx

Входы	
rst	Сброс ошибок записи
fnm	Абсолютный путь и имя файла на диске (может быть пустым – задается автоматически), расширение игнорируется. При сохранении данных на внешнем накопителе следует использовать путь, указанный на выходе блока ***210-SD-USB*** из библиотеки ***paOwenIO*** (константный)
wr	Запись на диск
in	Значение параметра (циклический)
typ	Тип параметра (циклический, константный): **DI** – 8-ми битный регистр; **AI** – вещественное значение; **II** – 16-ти битный регистр
ini	Значение для инициализации (циклический, константный)
Выходы	
next	Имя следующего файла
enb	Запись разрешена
sts	Статус: **0** – после сброса; **1** – записан; **2** – прочитан; **<0** – ошибка
good	Количество удачных записей
bad	Количество ошибок записи
rej	Количество отклоненных записей
o	Текущее значение параметров (циклический)

Имя файла и путь к нему задается на входе **fnm**, может быть пустым, тогда имя файла будет выбрано автоматически по индексу блока. В этом случае файл сохраняется в рабочую директорию контроллера.

Данные организуются в виде переменных **in** с жестко заданным типом **typ**.

Поскольку входы **in** имеют тип **any**, следует строго соблюдать правила преобразования типов при проведении связей.

Если файла не существует на диске – выходы инициализируются значениями инициализации **ini**, происходит запись в файл.

Если файл существует на диске, выходы инициализируются сохраненными значениями. Запись в файл осуществляется только при изменении значений на входах **in**.

Запись на диск можно осуществить принудительно, подав команду **wr**.

Для надежной сохранности данных одновременно на диске находятся два файла, соответствующие одному архиву. Поэтому, если контроллер будет перезагружен в момент записи на диск, данные не пропадут, а будут доступны предыдущие значения переменных, записанные в другом файле.

При чтении содержимое файла контролируется с помощью контрольной суммы, и только при ее корректности выдается на выходы (поэтому в случае, например, добавления новой переменной, значения, записанные в файл, сбросятся на инициализирующие **ini**).

В случае однократной ошибки при записи файла на диск, блок пытается переименовать текущий файл и снова произвести запись. Если повторная запись оказывается удачной, то продолжается работа в обычном режиме, а выход **bad** инкрементируется. В этом случае следует принять меры по диагностике или замене носителя, поскольку сбои при записи могут быть следствием скорого выхода его из строя. Файл, на котором произошел сбой, остается на диске под тем же именем с добавленным к нему суффиксом равным метке времени сбоя (в мс от 1 января 1970 г). Удалять его не желательно, чтобы повторно не использовать потенциально сбойный сектор.

Если происходит повторный сбой записи, то блок блокируется (выход **enb = 0**) и больше не производит попыток переименований файлов и записи до тех пор, пока ошибки не будут сброшены фронтом на входе **rst**.

ВНИМАНИЕ
При изменении числа входов блока ***SaverEx*** файлы на диске перезаписываются.

Подробнее о возможностях и работе блока ***SaverEx*** см. в документе Архивирование и сохранение уставок.

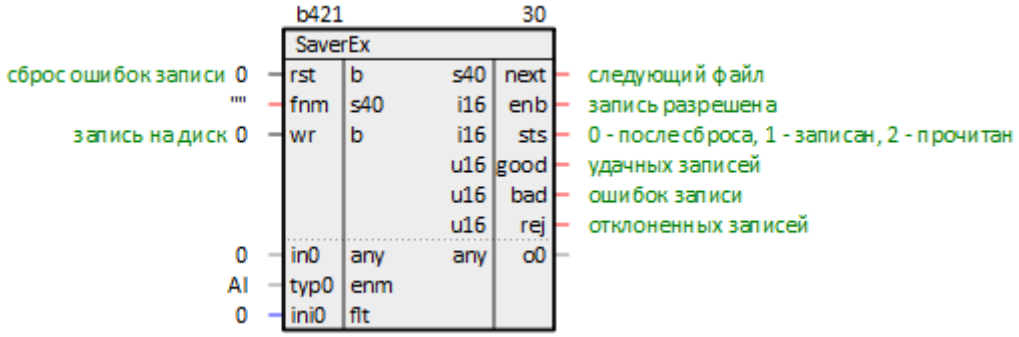


Рисунок 10.166 – Хранение параметров на диске (SaverEx)

10.1.13.2 Счетчик времени наработки (CounterMEx)

Блок ***CounterMEx*** предназначен для сохранения в файл наработки устройств, например, насосов. Раздел библиотеки: ***Сохранение данных***.

Поскольку операции файлового ввода/вывода занимают значительное время, данный блок следует размещать только в ***Фоне***.

Таблица 10.7 – Назначение входов и выходов CounterMEx

Входы	
rst	Сброс ошибок записи
fn	Абсолютный путь и имя файла (может быть пустым – задается автоматически), расширение игнорируется. При сохранении данных на внешнем накопителе следует использовать путь, указанный на выходе блока ***210-SD-USB*** из библиотеки ***paOwenIO*** (константный)
ask	Запись на диск
slv	Используется для изменения значений счетчиков, если **slv = 1**, то **cnt = mas**, **cfrn = mcfrn**
enbl	Работа устройства (циклический): пока **enbl = 1**, увеличивается время наработки **cfrn**, при изменении **enbl** с **0** на **1** число включений **cnt** увеличивается на **1**
rst	Сброс времени наработки **cnt** (циклический)
frm	Формат отображения для времени наработки. Не используется
rfr	Сброс числа включений **cfrn** (циклический)
mas	Балансировка времени наработки **cnt** (циклический), если **slv = 1**, то **cnt = mas**
mcfrn	Балансировка числа включений **cfrn** (циклический), если **slv = 1**, то **cfrn = mcfrn**
Выходы	
next	Имя следующего файла
enb	Запись разрешена
good	Количество удачных записей
bad	Количество ошибок записи
cnt	Время наработки в секундах (циклический)
hour	Время наработки в формате часы (циклический)
min	Время наработки в формате минуты (циклический)
sec	Время наработки в формате секунды (циклический)
day	Текущий день (циклический)
mnth	Текущий месяц (циклический)
year	Текущий год (циклический)
cfrn	Число включений (циклический)

Блок анализирует входы **enbl**.

На выходе **cfrn** отображается число включений устройства (количество изменений **enbl** с **0** на **1**), на выходе **cnt** отображается время наработки устройства (сколько секунд **enbl** был равен **1**).

Блок может сохранять число включений и время наработки в файл по фронту на входе $\boldsymbol{ask}$. Сохраненные значения считываются из файла при инициализации.

Имя файла и путь к нему задается на входе **fn**, может быть пустым, тогда имя файла будет выбрано автоматически по индексу блока. В этом случае файл сохраняется в рабочую директорию контроллера.

Для изменения числа включений и времени наработки следует подать **1** на вход **slv**, тогда **cnt = mas**, **cfrn = mcfrn**. Это может быть полезно для синхронизации в дублированных системах.

Для надежной сохранности данных одновременно на диске находятся два файла, соответствующие одному архиву. Поэтому, если контроллер будет перезагружен в момент записи

на диск, данные не пропадут, а будут доступны предыдущие значения наработки, записанные в другом файле.

При чтении содержимое файла контролируется с помощью контрольной суммы, и только при ее корректности выдается на выходы (поэтому в случае, например, добавления нового устройства, значения, записанные в файл, сбросятся).

В случае однократной ошибки при записи файла на диск, блок пытается переименовать текущий файл и снова произвести запись. Если повторная запись оказывается удачной, то продолжается работа в обычном режиме, а выход **bad** инкрементируется. В этом случае следует принять меры по диагностике или замене носителя, поскольку сбои при записи могут быть следствием скорого выхода его из строя. Файл, на котором произошел сбой, остается на диске под тем же именем с добавленным к нему суффиксом равным метке времени сбоя (в мс от 1 января 1970 г). Удалять его не желательно, чтобы повторно не использовать потенциально сбойный сектор.

Если происходит повторный сбой записи, то блок блокируется до тех пор, пока ошибки не будут сброшены фронтом на входе **rst**.

ВНИМАНИЕ
При изменении числа входов блока ***CounterMEx*** файлы на диске перезаписываются.

Подробнее о возможностях и работе блока ***CounterMEx*** см. в документе Архивирование и сохранение уставок.

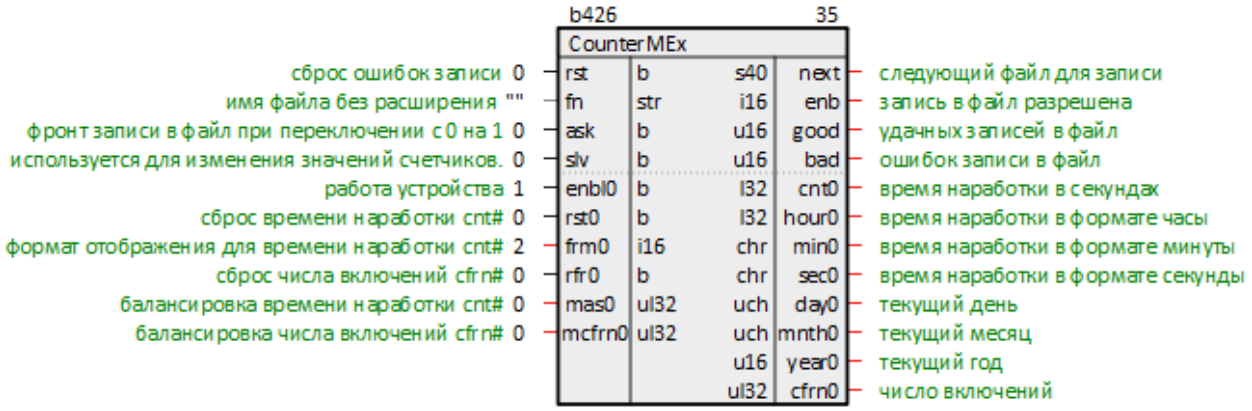


Рисунок 10.167 – Счетчик времени наработки (CounterMEx)

10.1.13.3 Буфер чтения/записи уставок (BufSupEx)

Блок ***BufSupEx*** представляет собой двунаправленный буфер данных интерфейса, при этом данные сохраняются в бинарном файле на диске контроллера. Блок сохраняет все значения на диске контроллера аналогично блоку SaverEx. Раздел библиотеки: ***Сохранение данных***.

Поскольку операции файлового ввода/вывода занимают значительное время, данный блок следует размещать только в ***Фоне***.

Таблица 10.8 – Назначение входов и выходов BufSupEx

Входы	
inter	Связь от интерфейса, к которому принадлежит данный буфер
group	Номер группы (константный)
fnm	Абсолютный путь и имя файла (может быть пустым – задается автоматически), расширение игнорируется. При сохранении данных на внешнем накопителе следует использовать путь, указанный на выходе блока ***210-SD-USB*** из библиотеки ***paOwenIO*** (константный)
mask	Не используется
rst	Сброс ошибок записи
wr	Запись на диск
dan	Значение, которое записывается в буфер при **czap = 1** (циклический)
czap	Запись значения **dan** (циклический)
typ	Тип параметра (циклический, константный): **DI**, **DO** – 8-ми битный регистр; **AI**, **AO** – вещественное значение; **II**, **IO** – 16-ти битный регистр
adr	Адрес параметра (циклический, константный)
ini	Значение для инициализации (циклический, константный)
min	Минимум. Если принятое значение меньше **min**, то оно игнорируется (циклический)
max	Максимум. Если принятое значение больше **max**, то оно игнорируется (циклический)
Выходы	
pkt	Подключение к блокам ***OpcUAClient***, ***UABufSups*** из библиотеки ***paOpcUA***
next	Имя следующего файла
enb	Запись разрешена
sts	Статус: **0** – после сброса; **1** – записан; **2** – прочитан; **<0** – ошибка
good	Количество удачных записей
bad	Количество ошибок записи
rej	Количество отклоненных записей
dan	Значение параметра, полученное по интерфейсу или на вход **dan** (после проверки на **min** и **max**)
chn	Признак изменения, выставляется в **1** на один цикл выполнения программы, если значение **dan** изменилось
zap	Признак записи, выставляется в **1** на один цикл выполнения программы, если значение с входа **dan** было записано

Номер группы **group** используется в качестве Slave ID при подключении к блоку интерфейса Modbus Slave.

Имя файла и путь к нему задается на входе **fnm**, может быть пустым, тогда имя файла будет выбрано автоматически по индексу блока. В этом случае файл сохраняется в рабочую директорию контроллера.

Адрес переменной **adr** зависит от интерфейса, к которому подключен буфер, например, адрес регистра Modbus.

Поскольку входы **dan** имеют тип ***any*** следует строго соблюдать правила преобразования типов при проведении связей.

Если файл существует на диске, выходы инициализируются сохраненными значениями. Если файла не существует – выходы инициализируются значениями инициализации **ini**.

Запись в файл осуществляется при изменении значений на входах **dan** или по интерфейсу. Если файла на диске не существует и выходы **dan** приняли значения **ini**, то можно записать их на диск принудительно, подав команду **wr**.

Для надежной сохранности данных одновременно на диске находятся два файла, соответствующие одному архиву. Поэтому, если контроллер будет перезагружен в момент записи на диск, данные не пропадут, а будут доступны предыдущие значения переменных, записанные в другом файле.

При чтении содержимое файла контролируется с помощью контрольной суммы, и только при ее корректности выдается на выходы (поэтому в случае, например, добавления новой переменной, значения, записанные в файл, сбросятся на инициализирующие **ini**).

В случае однократной ошибки при записи файла на диск, блок пытается переименовать текущий файл и снова произвести запись. Если повторная запись оказывается удачной, то продолжается работа в обычном режиме, а выход **bad** инкрементируется. В этом случае следует принять меры по диагностике или замене носителя, поскольку сбои при записи могут быть следствием скорого выхода его из строя. Файл, на котором произошел сбой, остается на диске под тем же именем с добавленным к нему суффиксом равным метке времени сбоя (в мс от 1 января 1970 г). Удалять его не желательно, чтобы повторно не использовать потенциально сбойный сектор.

Если происходит повторный сбой записи, то блок блокируется (выход **enb = 0**) и больше не производит попыток переименований файлов и записи до тех пор, пока ошибки не будут сброшены фронтом на входе **rst**.

ВНИМАНИЕ
При изменении числа входов блока ***BufSupEx*** файлы на диске перезаписываются.

Подробнее о возможностях и работе блока ***BufSupEx*** см. в документе Архивирование и сохранение уставок.

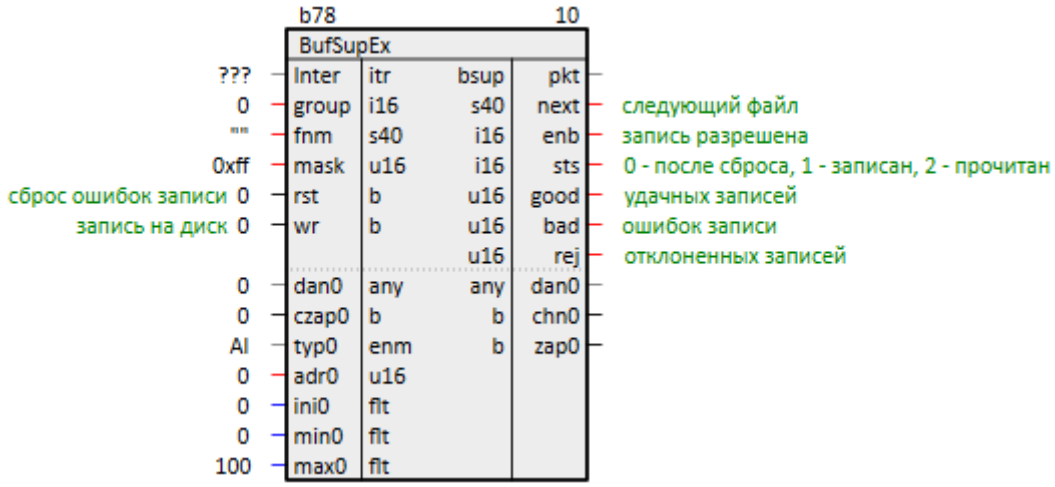


Рисунок 10.168 – Буфер чтения/записи уставок (BufSupEx)

10.1.13.4 Запись логов в оперативную память (RamLog)

Блок ***RamLog*** обеспечивает возможность сбора логов по заданным параметрам, для сохранения в оперативную память. Запись происходит в несколько буферов, которые при заполнении или по таймауту попадают в очередь для передачи на выход (сама передача происходит при условии, что предыдущий прочтен). Раздел библиотеки: ***Сохранение данных***.

Так как работа блока занимает значительное время, может быть размещен только в ***Фоне***.

Таблица 10.9 – Назначение входов и выходов RamLog

Входы	
enb	Включение блока
bsize	Размер буфера – позволяет задать максимальное количество символов, хранимых в отдельном буфере: **128…4096**
bnum	Количество буферов: **2…64**
init	Создать ли все буферы при инициализации. Полезно, если требуемые диагностические сообщения имеют большой объем и генерируются с самого начала работы программы. Изначально, при инициализации создается минимальное количество буферов и, если требуется, увеличивается в процессе работы. Если данный вход выставлен в **1**, то сразу будет создано **bnum** буферов
tmout	Таймаут неполного буфера, мс - определяет временной промежуток, через который буфер, не заполненный до конца, будет считаться готовым к выдаче. Отсчет ведется с момента последнего дописанного сообщения. Полезно, если диагностические сообщения генерируются редко или имеют малый объем
glvl	Уровень ошибок, если у сообщения уровень меньше или равен заданному, то оно будет напечатано независимо от кодов и индексов (**ERR = 1**, **WRN = 2**, **INF = 3, DBG = 10**, **DBG + N**)
dbglvl	Уровень отладки, если у сообщения уровень превышает заданный, оно не будет напечатано
dscrd	Сброс, очищает списки отслеживаемых блоков и кодов. Срабатывает при изменении значения с **0** на **1**

Продолжение таблицы 10.9

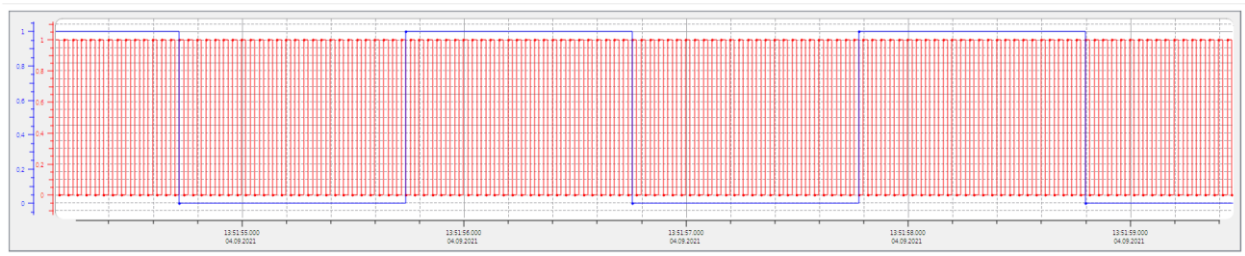
code	Код отладки – значение, которое будет добавлено в список отслеживаемых при изменении входа **addcd** или удалено при изменении входа **delcd**
addcd	Добавить код
delcd	Удалить код
indx	Индекс блока – значение, которое будет добавлено в список отслеживаемых при изменении входа **addix** или удалено при изменении входа **delix**
addix	Добавить индекс
delix	Удалить индекс
print	Выводить записываемые логи в консоль
Выходы	
sts	Статус: **0** – нормальная работа; **+2** – сообщение, записываемое в буфер, превышает его максимальный размер, часть сообщения обрезана; **+4** – все буферы заполнены, происходит периодическая потеря сообщений. Для устранения ошибок следует изменить соответствующие входные параметры: размер **bsize**, или количество **bnum**
log	Текущий буфер, доступный для чтения
cds	Список отслеживаемых диагностических кодов
indxs	Список индексов отслеживаемых блоков

Рисунок 10.169 – Запись логов в оперативную память (RamLog)

10.1.14 Очереди

10.1.14.1 Очередь битов qb

Тип данных ***qb*** позволяет накапливать очередь битовых значений в таймерном потоке. Если выход типа ***qb*** использовать в обмене по протоколу OPC UA, то переданы будут все накопленные данные независимо от настроек подписки.

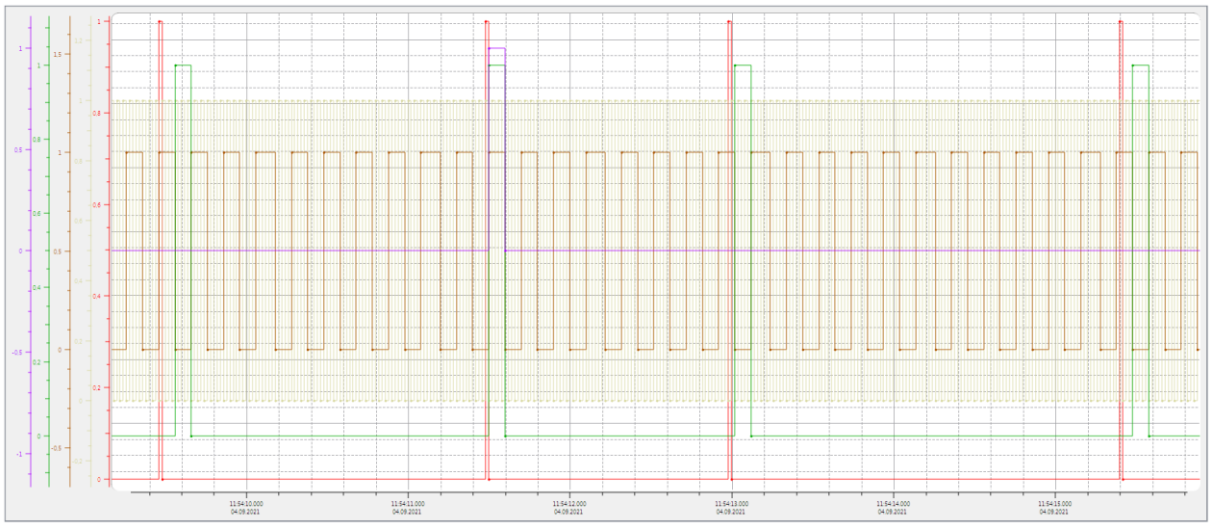


Комментарии	Цвет
очередь битовых значений	
меандр таймера	

Рисунок 10.170 – Очередь битов qb

В случае слишком медленной передачи или слишком частых изменений входного сигнала, очередь может переполниться, что будет видно на выходе блока – источника очереди (например, R2toQuBit). Для изменения размера очереди необходимо перетранслировать программу.

При соединении выхода типа ***qb*** с битовым входом, расположенным в фоновом потоке, будет автоматически выполнено преобразование (блок QuBit_Bit), которое обеспечит гарантированную передачу бита. Пример ниже показывает разницу между обычным битом и битов с накоплением очереди. Обычный импульс, сформированный в таймере, не может быть гарантированно получен в фоновом потоке, если длительность импульса достаточно мала.



Комментарии	Цвет
меандр таймера	
меандр фона	
исходный бит (длиной 1 таймерный цикл)	
бит из таймера с гарантированной доставкой	
бит из таймера	

Рисунок 10.171 – Разница между обычным битом и битов с накоплением очереди

10.1.14.2 Преобразование регистра в очереди битов (R2toQuBit)

Блок ***R2toQuBit*** разбирает входной регистр на биты, каждый из которых может накапливать очередь значений в таймерном потоке. Раздел библиотеки: ***Очереди***.

Назначение входов и выходов:

qus – размер очереди (на каждый выходной бит);
reg – входной регистр;
ov – количество переполнений очереди;
n – текущее количество очередей;
b0, **…**, **b15** – биты.

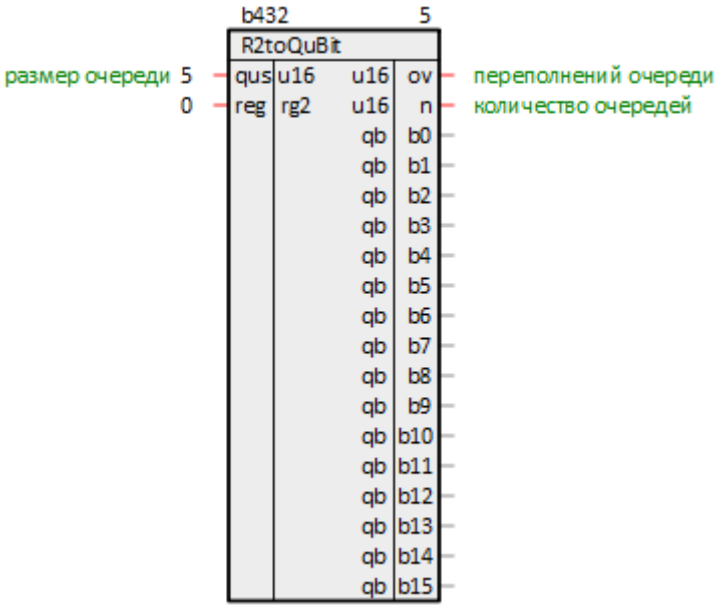


Рисунок 10.172 – Преобразование регистра в очереди битов (R2toQuBit)

10.1.14.3 Преобразование очереди битов в бит (QuBit_Bit)

Блок ***QuBit_Bit*** преобразует очередь битов в бит, который можно дальше использовать в фоновом потоке. Среда разработки автоматически добавляет этот блок, если проведена связь между выходом типа **qb** и входом типа **b**. Раздел библиотеки: ***Очереди***.

Назначение входов и выходов:

in – входная очередь;
val – какое значение ловить;
out – выход.

Алгоритм работы: значения из входной очереди сравниваются с заданным (вход **val**), и если встречается хотя бы одно, то выход **out** устанавливается равным **val**.

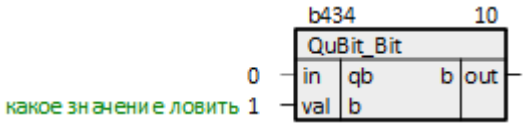


Рисунок 10.173 – Преобразование очереди битов в бит (QuBit_Bit)

Россия, 111024, Москва, 2-я ул. Энтузиастов, д. 5, корп. 5
тел.: +7 (495) 641-11-56, факс: (495) 728-41-45
тех. поддержка 24/7: 8-800-775-63-83, support@owen.ru
отдел продаж: sales@owen.ru
Веб-сайт ООО "ПромАвтоматика-Софт": www.pa.ru
рег.:1-RU-dev-8.0