НЕФТЕАВТОМАТИКА

УТВЕРЖДЕН
КДСА.421457.190А-01 34 УЛ

Контроллер программируемый логический *MKLogic200®*

Программирование в среде Beremiz MKLogic200

Руководство пользователя

КДСА.421457.190А-01 34

Листов 111

2018 г.

Аннотация

Настоящее руководство пользователя (далее – Руководство) описывает порядок разработки прикладных программ пользователя для **Контроллера программируемого логического MKLogic200** в среде разработки Beremiz MKLogic200.

Руководство содержит информацию о назначении среды разработки, условиях, необходимых для её выполнения, об элементах пользовательского интерфейса, о работе с динамически подключаемыми модулями – плагинами, о редакторе языков стандарта IEC 61131-3.

Содержание

1 Общие сведения о среде разработки

1.1 Назначение

Среда разработки Beremiz MKLogic200 (далее среда Beremiz) предназначена для создания и отладки прикладных программ пользователя (далее программ пользователя) для Контроллера программируемого логического (далее ПЛК) MKLogic200 на языках стандарта IEC 61131-3:

- Structured Text (далее ST);
- Instruction List (далее IL);
- Function Block Diagram (далее FBD);
- Ladder Diagram (далее LD);
- Sequential Function Chart (далее SFC).

Целевым устройством при работе со средой Beremiz, т.е. устройством к которому подключается среда, и в которое производится загрузка готовой программы пользователя, является Контроллер промышленный МК201 КДСА 421457.190А (далее Контроллер), входящий в состав ПЛК MKLogic200.

1.2 Основные компоненты

Основными компонентами среды Beremiz являются:

- редактор PLCOpen для текстовых (IL и ST) и графических языков (FBD, LD, SFC) стандарта IEC 61131-3;
- компилятор MatIEC, преобразующий логику и алгоритмы программных модулей (из которых состоит прикладная программа), описанных на языках стандарта IEC-61131-3 в эквивалентный код на языке Си;
- механизм плагинов, позволяющий связывать внешние источники данных;
- средства отладки прикладной программы в режиме исполнения.

Программы, написанные на языках стандарта IEC 61131-3, с помощью компилятора преобразуются в исполняемый файл, который загружается непосредственно в контроллер.

Программные модули, написанные пользователем на текстовых (ST, IL) и/или графических (FBD, SFC, LD) языках в соответствии со стандартом IEC 61131-3 объединены в проект. Каждый такой проект представлен в формате XML и хранится в отдельной папке.

Исполняемый файл, благодаря средствам среды Beremiz, может быть размещён в Контроллере, где он запускается и в процессе работы выполняет следующие действия:

- обменивается данными с внешними модулями;
- исполняет алгоритмы и логику, определённую пользователем в программных модулях проекта;
- принимает и передаёт отладочную информацию.

Созданная прикладная программа может быть запущена в режиме отладки. В данном режиме появляется возможность вводить и изменять значения переменных (аналогично процессу отладки в IDE для широко используемых языков программирования) программных модулей, из которых состоит прикладная программа, отображать их в виде графика.

1.3 Системные требования

Требования к техническим средствам (компонентам), необходимые для установки и исполнения среды Beremiz, приведены в Табл. 1.

Табл. 1 – Технические требования для работы среды Beremiz

Характеристики технических средств	Требования
Тактовая частота процессора ПК	1000 МГц (32 бита/ 64 бит)
Оперативная память ПК	1 Гб
Операционная система	Windows XP/Vista/7/8/10 Linux
Монитор	не менее 17’’
Минимальное разрешение экрана	1600x900
Свободное место на жестком диске	1 Гб

2 Подготовка к работе

2.1 Установка и настройка среды Beremiz

Для установки среды Beremiz необходимо запустить установочный файл beremiz-setup_VX.X.X.exe, где X.X.X номер текущей версии.

Примечание: Установочный файл находится на CD из комплекта поставки ПЛК MKLogic200. В случае отсутствия CD-диска в комплекте поставки, необходимо связаться с представителями АО «Нефтеавтоматика» для его получения (контакты можно найти на сайте компании — www.nefteavtomatika.ru).

После запуска установочного файла пользователю будет предложено выбрать язык установки (Рис. 1).

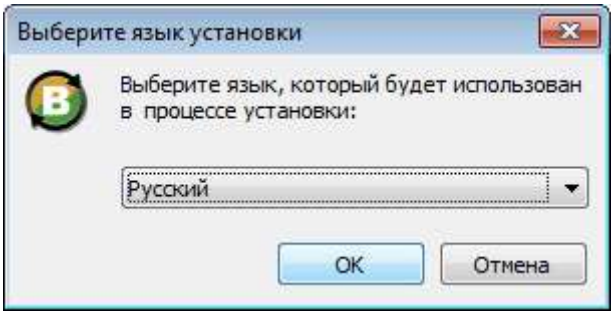


Рисунок 1 – Выбор языка в процессе установки среды Beremiz MKLogic200

Далее будет отображена серия появляющихся друг за другом окон, в которых необходимо выбрать опции установки (Рис. 2). Можно выбрать следующие опции: создание значка на рабочем столе (Рис. 2б), выбор папки установки (Рис. 2в), наименование папки в меню «Пуск» (Рис. 2г).

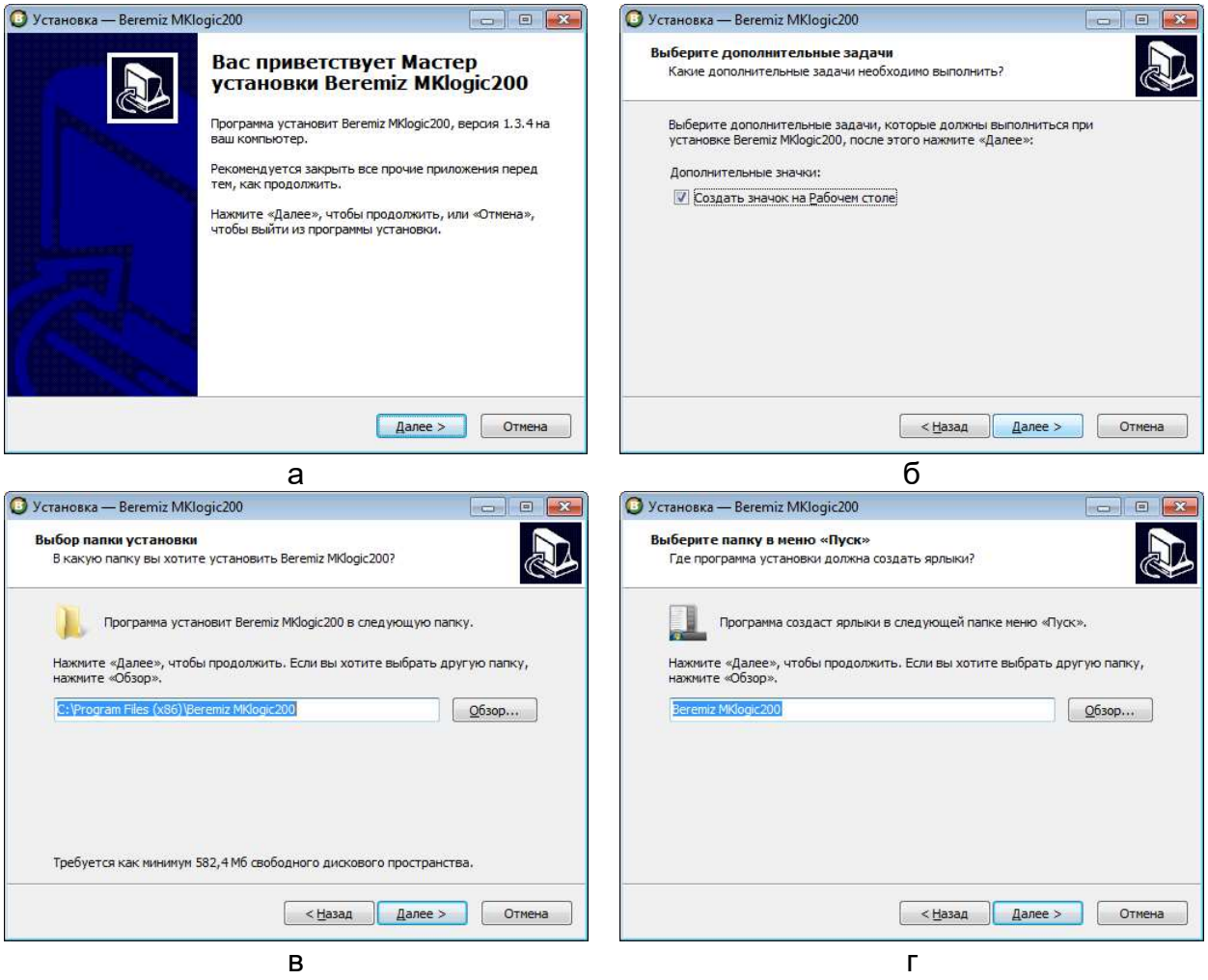


Рисунок 2 – Настройка опций установки среды Beremiz

После выбора опций установки необходимо нажать кнопку «Установить» (Рис. 3а). По завершению процесса установки (Рис. 3б) появится окно, сигнализирующее об его окончании (Рис. 3в).

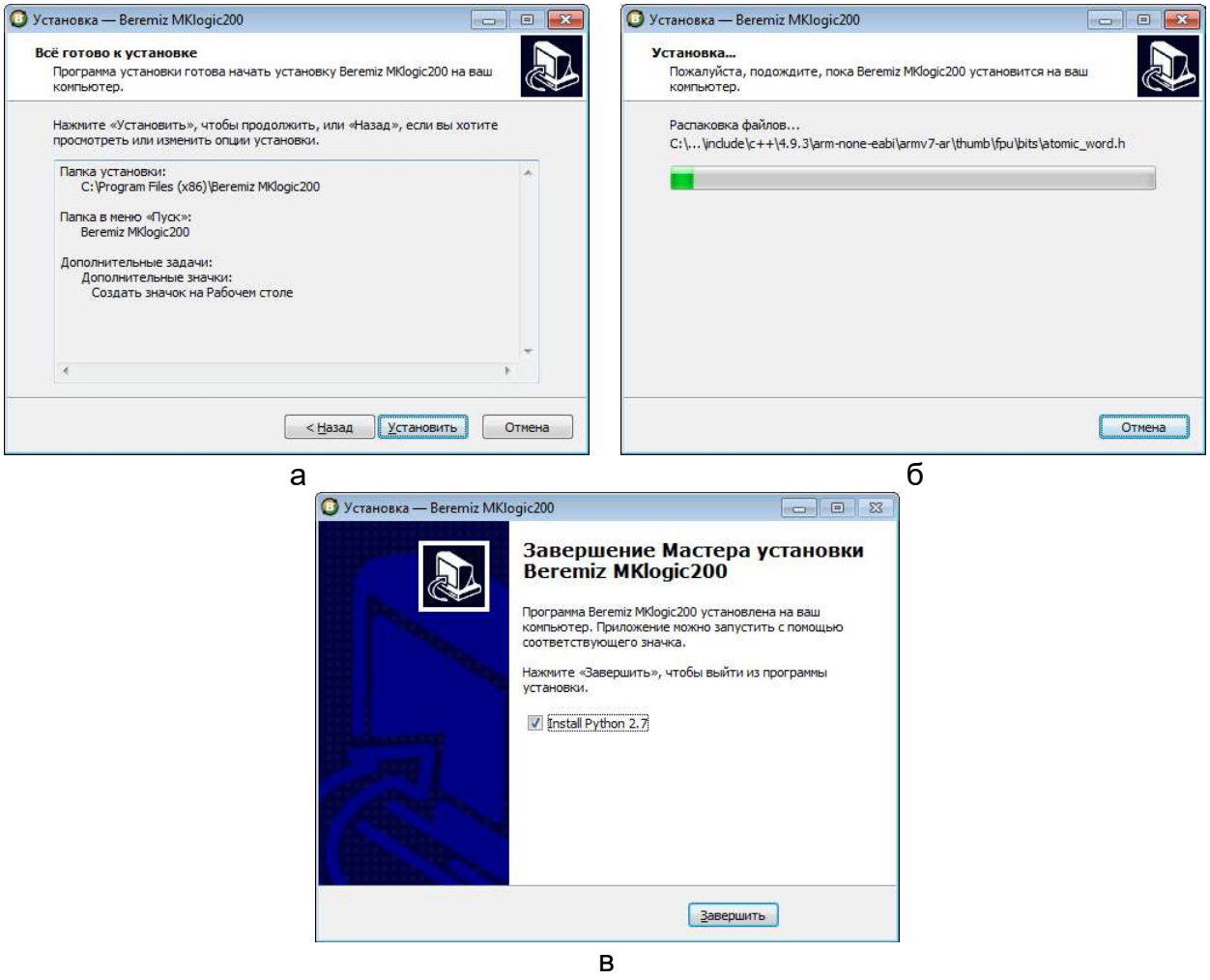


Рисунок 3 – Процесс установки среды Beremiz MKLogic200

В конце установки будет предложено установить Python версии 2.7. Данный пакет является обязательным для среды Beremiz, поэтому рекомендуется установить галочку «Install Python 2.7» и нажать кнопку «Завершить». Будет запущен стандартный установщик Python 2.7, руководство по его установке можно найти на сайте www.python.org.

Допускается пропустить этот шаг, в случае если Python 2.7 уже установлен в системе, или при условии, что Python 2.7 будет установлен позже независимо от среды Beremiz.

После выполнения всех описанных шагов, среда Beremiz установлена на ПК и готова к работе.

3 Работа в среде разработки Beremiz MKLogic200

3.1 Описание интерфейса

Запуск среды Beremiz осуществляется тремя способами: при помощи ярлыка на рабочем столе (если он был создан); при помощи ярлыка в меню Пуск; запуском файла run.bat в директории установке среды. После запуска на экране монитора появится начальное окно среды Beremiz (рис. 4)

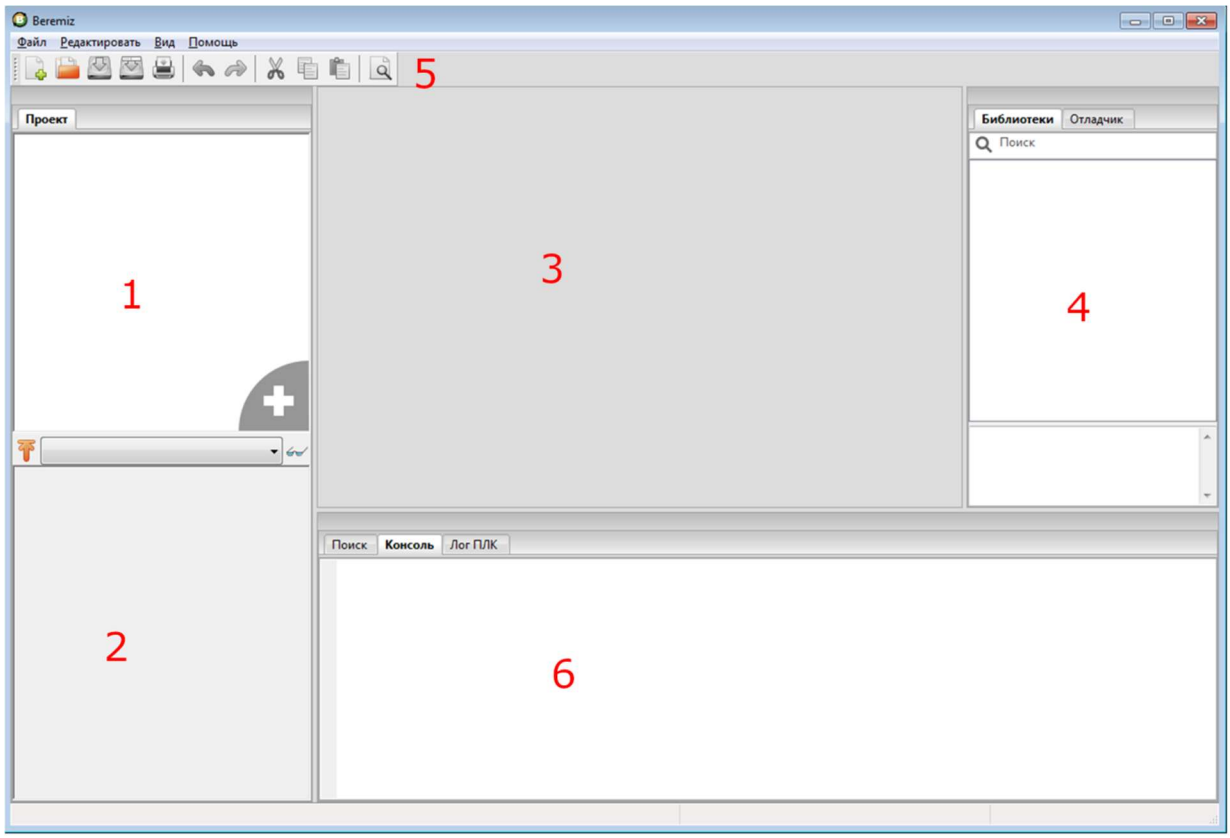


Рисунок 4 – Начальное окно среды Beremiz

Начальное окно содержит следующие элементы:

- главное меню программы;
- дерево проекта (1);
- панель списка переменных и констант (2);
- основная рабочая область (3);
- панель библиотеки функций и функциональных блоков (4);
- панель отладки (4);
- панель инструментов (5);
- консоль (6), содержащая также отладочную консоли и панель поиска элементов в проекте.

3.1.1 Главное меню программы

Главное меню программы (см. рис. 4) содержит следующие пункты:

- «Файл»;
- «Редактировать»;
- «Вид»;
- «Помощь».

3.1.1.1 Меню «Файл»

Меню «Файл» предназначено для работы с проектом и предоставляет следующие пункты:

Таблица 2 – Назначение пунктов меню «Файл»

Пункт меню	**Назначение**	**«Горячие» клавиши**
«Новый»	Создание нового проекта	CTRL + N
«Открыть»	Открытие существующего проекта	CTRL + O
«Недавние проекты»	Быстрый доступ к одному из десяти последних проектов	
«Сохранить»	Сохранение изменений в текущем проекте	CTRL + S
«Сохранить как»	Сохранение текущего проекта в другой директории	CTRL + SHIFT + S
«Закрыть вкладку»	Закрытие активной вкладки (например, вкладки переменных плагина, конфигурации и т.д.) для открытого проекта	CTRL + W
«Закрыть проект»	Закрытие текущего, открытого, проекта	CTRL + SHIFT + W
«Параметры страницы»	Настройка параметров вывода на печать активной программы, представленной в виде диаграммы	CTRL + ALT + P
«Предпросмотр»	Предварительный просмотр перед печатью на принтере активной программы	CTRL + SHIFT + P
«Печать»	Печать на принтере активной программы	CTRL + P
«Выход»	Закрытие текущего проекта и выход из среды Beremiz	CTRL+ Q

3.1.1.2 Меню «Редактировать»

Меню «Редактировать» предназначено для работы с редакторами языков стандарта IEC 61131-3 и предоставляет следующие возможности:

Таблица 3 – Назначение пунктов меню «Редактировать»

Пункт меню	**Назначение**	**«Горячие» клавиши**
«Отменить»	Отмена последнего действия в редакторе	CTRL + Z
«Повторить»	Повтор отменённого действия в редакторе	CTRL + Y
«Вырезать»	Удаление выделенного в редакторе элемента в буфер обмена	CTRL+ X
«Копировать»	Копирование выделенного в редакторе элемента в буфер обмена	CTRL + C
«Вставить»	Вставка в редактор элементов из буфера обмена	CTRL + V
«Поиск в проекте»	Вызов диалогового окна поиска данных в проекте	CTRL + SHIFT + F
«Добавить элемент»	Добавление нового элемента в текущий проект (см. раздел 3.5)	
«Выделить всё»	Выделение всех элементов в активной вкладке редактора	CTRL + A
«Удалить»	Удаление программного модуля из дерева проекта (см. раздел 3.5).	

3.1.1.3 Меню «Вид»

Меню «Вид» предназначено для работы с видами элементов среды. Основные возможности по работе с видами представлены в таблице 4.

Таблица 4 – Назначение пунктов меню «Вид»

Пункт меню	**Назначение**	**«Горячие» клавиши**
«Обновить»	Обновление данных в редакторе (при обновлении данных снимаются все выделения объектов)	CTRL + R
«Очистить ошибки»	Очистка указателей ошибок в редакторе	CTRL + K
«Масштаб»	Выбор масштаба отображения текущего редактора в процентах	нет
«Сброс расположения панелей»	Восстановление расположения панелей Beremiz в исходное состояние	нет

3.1.1.4 Меню «Помощь»

Меню «Помощь» позволяет посмотреть информацию о текущей версии и авторах среды Beremiz. Для этого необходимо выбрать пункт «О программе».

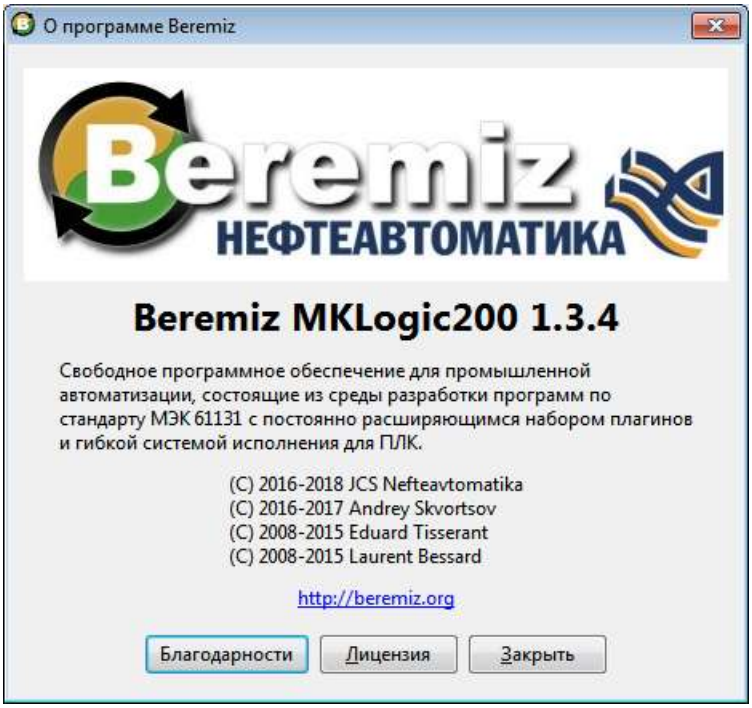


Рисунок 5 – Окно «О программе»

3.1.2 Панели инструментов

На панелях инструментов расположены кнопки быстрого доступа к часто используемым функциям среды разработки Beremiz. Кнопки сгруппированы по назначению: панель инструментов главного меню, панель инструментов статуса и панель инструментов редактирования программных модулей.

Начальное окно содержит только панель главного меню. Панель статуса появляется при создании проекта. Панель редактирования программных модулей появляется при открытии панели редактирования программного модуля (см. раздел 3.6) и будет описана в соответствующих разделах.

Панель инструментов главного меню приведена на рис. 6.

Рисунок 6 – Панель инструментов главного меню

Функциональное назначение кнопок панели инструментов главного меню приведено в таблице 5.

Таблица 5 – Кнопки панели инструментов главного меню

Наименование кнопки	**Функциональное назначение кнопки**
Новый проект	Создание нового проекта
Открыть проект	Открытие существующего проекта
Сохранить проект	Сохранение текущего проекта в директорию, заданную при создании проекта
Сохранить проект как	Сохранение текущего проекта в произвольную директорию, отличную от заданной при создании проекта
Печать	Вывод на печатающее устройство листинга текущей программы
Отменить	Отмена последнего произведенного действия в среде разработки
Повторить	Возврат последнего отмененного действия
Вырезать	Удаление в буфер обмена выделенного элемента элемент(ы) в редакторе
Копировать	Копирование в буфер обмена выделенного элемента элемент(ы) в редакторе
Вставить	Вставка элемента из буфера обмена в редактор
Поиск в проекте	Вызов диалога поиска данных в проекте
Выделение объекта	Включение/отключение возможности выделения объектов в редакторе, используя указатель мыши

Панель инструментов управления проектом (рис. 7) предназначена для компоновки и компиляции текущего проекта, просмотра сгенерированного кода, конфигурирования Контроллера, а также передачи и запуска полученного исполняемого файла в Контроллере.

Рисунок 7 – Панель инструментов статуса

Панель инструментов статуса по умолчанию располагается в верхней части окна среды разработки.

Кнопки панели инструментов управления проектом и их описание представлены в таблице 6.

Таблица 6 – Кнопки панели инструментов управления проектом

Иконка	Наименование	Описание
	Сборка проекта в директории сборки	Компиляция и компоновка текущего проекта во вложенную директорию «build». Директория «build» расположена в директории проекта.
	Очистить директорию сборки проекта	Удаление содержимого директории «build».
	Настройки подключения	Выбор варианта подключения целевого устройства и настройка подключения.
	Подключиться к целевому ПЛК	Соединение с целевым устройством в соответствии с заданными настройками.
	Отключиться от ПЛК	Разрыв соединения с целевым устройством.
	Показать код, сгенерированный PLCGenerator	Отображение в отдельном окне текстового редактора (см. раздел 3.6.1) кода на языках ST и LD, который был транслирован в C код.
	Передать ПЛК	Передача исполняемого файла, полученного в ходе сборки проекта, на целевое устройство.
	Запустить ПЛК	Запуск на исполнение на целевом устройстве собранной прикладной программы.
	Конфигуратор MK201 Setup	Конфигурация целевого устройства.

3.2 Создание проекта

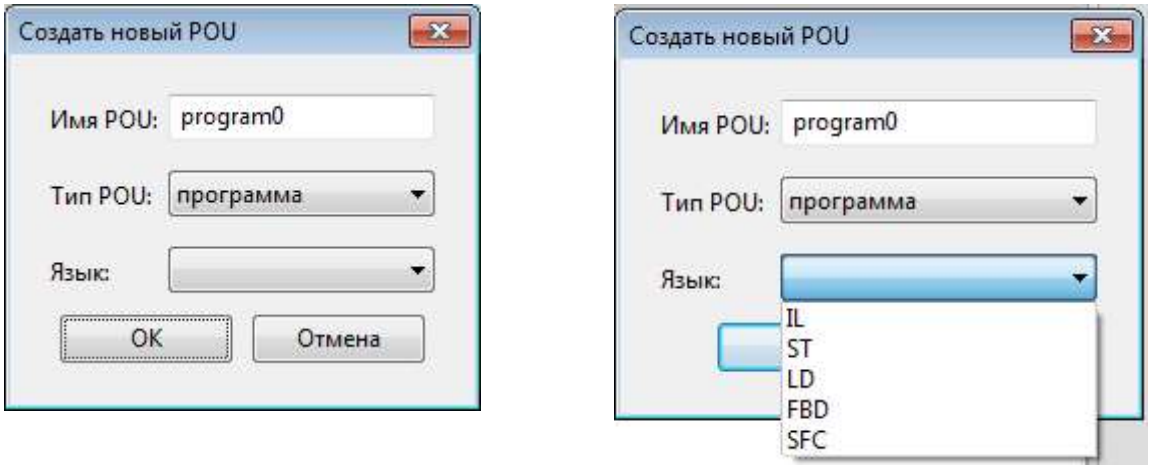
Для создания проекта необходимо выбрать пункт меню Файл -> Новый. После чего будет предложено выбрать директорию для нового проекта.

Примечание: В среде Beremiz именем проекта считается имя директории, в которой проект расположен.

Примечание: Для создания нового проекта может использоваться только пустая директория.

После указания директории появится окно создания нового программного модуля (POU – Program organization unit). В данном окне (рис. 8а) нужно выбрать «имя POU», «тип POU» и «Язык».

Примечание: Каждый проект в среде Beremiz должен содержать хотя бы одну программу. Поэтому при создании проекта автоматически появляется окно создания нового программного продукта, в котором доступен только одни тип POU – «программа». От создания программы на данном этапе можно отказаться, но нужно помнить, что программу нужно обязательно создать позже.



а б

Рисунок 8 – Окно создания программного модуля

Язык выбирается из списка доступных IEC 61131-3 языков (см. рис. 8б).

После создания проекта появится обновленное окно среды. В основной рабочей области появится редактор программы. Для программ на языках ST и IL — это текстовый редактор, для программ на языках FBD, SFC, LD — это графический редактор диаграмм.

На панели «Библиотеки» появится список всех доступных функций и функциональных блоков.

На рис. 9 показана панель редактирования ресурсов после создания нового проекта. Данная панель в свою очередь состоит из панели переменных (рис. 9, поз. 1), панели задач (рис. 9, поз. 2) и панели экземпляров объектов (рис. 9, поз. 3).

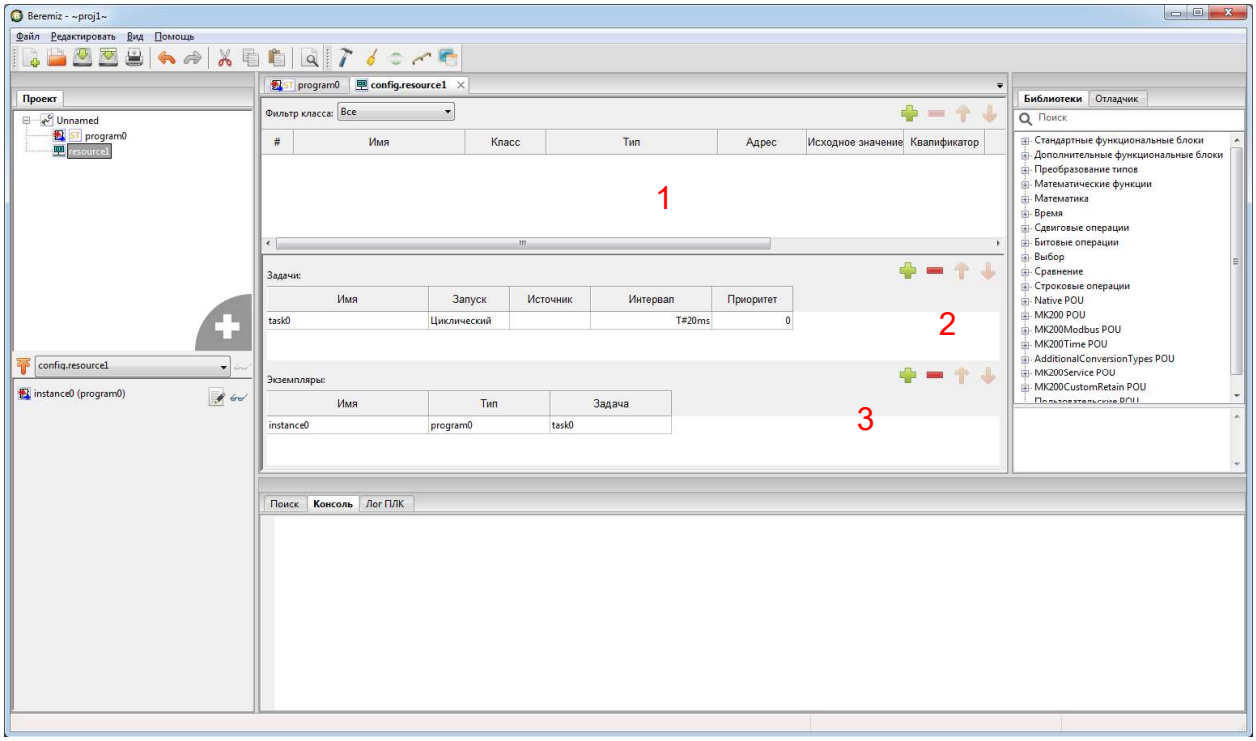


Рисунок 9 – Панель редактирования «Ресурсов»

Как видно (см. рис. 9), при создании проекта и создании программы, была автоматически создана задача с именем «task0» и экземпляр объекта с именем «instance0».

3.3 Структура проекта

Структура проекта представлена в дереве проекта. Структура элементов проекта может включать в себя:

- ресурсы;
- программные модули (функции, функциональные блоки и программы) и их составные части;
- типы данных;
- плагины коммуникационных протоколов и каналов ввода-вывода.

Минимальный проект в среде Beremiz состоит из двух обязательных элементов: программа, написанная на одном из языков стандарта и элемента «Ресурсы». Элемент «Ресурсы» предназначен для создания

- глобальных переменных,
- задач (см. ниже),
- экземпляров объектов (см. ниже).

ВНИМАНИЕ! Проект может содержать только один элемент «Ресурсы»! В данной версии среды возможно добавление двух и более ресурсов, но проект при этом перестанет собираться.

Управление строками во всех таблицах панелей среды Beremiz осуществляется при помощи специальных кнопок (таблица 7).

Таблица 7 – Кнопки управления строками в таблицах среды Beremiz

Иконка	Функциональное назначение кнопки
	Добавить новую строку (например, переменную) со значениями по умолчанию
	Удалить выделенную строку (например, переменную)
	Переместить строку (например, переменную) в таблице на одну строку выше
	Переместить строку (например, переменную) в таблице на одну строку ниже

В верхней части панели переменных расположен, фильтр по классу переменной и кнопки управления переменными (таблица 7).

Каждая переменная задается отдельной строкой в таблице переменных. Назначение столбцов таблицы переменных приводится в таблице 8.

Таблица 8 – Таблица переменных панели переменных

Наименование столбца	**Описание столбца**
Имя	Уникальный идентификатор переменной в пределах её области видимости и области действия.
Класс	Класс памяти переменной, определяет область видимости переменной, а также как долго переменная находится в памяти: «Глобальная», «Входная», «Выходная», «Входная/Выходная», «Локальная», «Внешняя», «Временная».
Тип	Тип переменной: – базовый тип (в соответствии со стандартом IEC 61131-3); – пользовательский тип (псевдоним и поддиапазон существующего типа, перечисление, массив, структура); – тип «Native» (не используется в данной версии); – функциональный блок (стандартный или пользовательский); – массив.
Адрес	Идентификатор, необходимый для связывания данной переменной с переменной модулей связи и ввода вывода (см. раздел 3.10.6).
Исходное значение	Инициализация переменной некоторым начальным значением.
Квалификатор	Принимает одно из трех значений: – Константа (неизменяемая переменная); – Retain (сохранение значения переменной в энергонезависимой памяти); – Не-Retain (сохранение значения переменной в ОЗУ).
Описание	Комментарий к переменной или константе.

Задачи задаются на панели задач. Задача в среде Beremiz — это программная единица, выполняемая по прерыванию или периодически. В проекте можно создать множество различных задач. Каждая задача создается в виде отдельной строки таблицы панели «Задачи» (рис. 10).

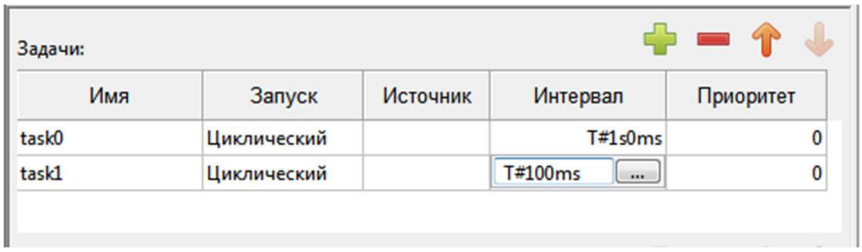


Рисунок 10 – Таблица «Задачи» панели ресурсов

Описание столбцов таблицы «Задачи» приведено в таблице 9.

Таблица 9 – Панель «Задачи»

Наименование столбца	**Описание**
Имя	Имя задачи. Задач с одинаковыми именами быть не может.
Запуск	Тип запуска задачи: «Циклический» – задача запускается периодически; «Прерывание» – задача запускается по прерыванию. В данной версии реализован только тип запуска «Циклический».
Источник	Источник запуска, доступно при выборе типа запуска «Прерывание». В данной версии реализован только тип запуска «Циклический», поэтому данное поле не используется.
Интервал	Интервал запуска, доступно при выборе типа запуска «Циклический»
Приоритет	Приоритет задачи

Интервал запуска вводится по нажатию кнопки ... . При этом открывается окно ввода времени (рис. 11).

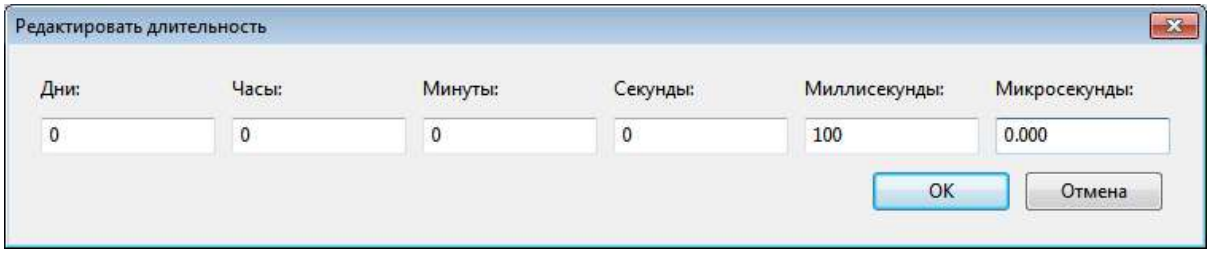
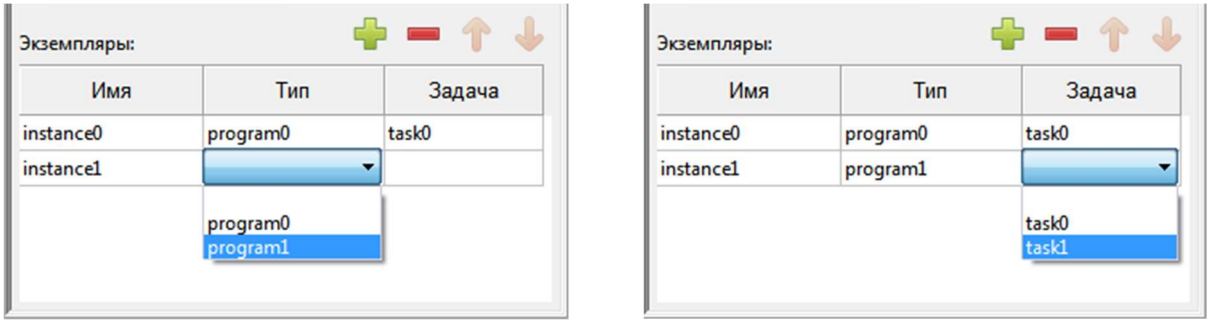


Рисунок 11 – Окно ввода временных интервалов

Примечание: В проекте должна быть хотя бы одна задача. В противном случае сгенерированный из проекта код не будет выполнять никаких действий при его загрузке в Контроллер. С задачей должна быть связана хотя бы одна программа.

Связывание задач и имеющихся в проекте программ осуществляется при помощи экземпляров объектов. Каждый экземпляр объекта записывается в отдельной строке таблицы панели «Экземпляры» (рис. 12). Каждому экземпляру необходимо задать уникальное имя, а также выбрать из соответствующих выпадающих списков задачу и программу.



а б

Рисунок 12 – Связывание программ и задачи при помощи экземпляров в панели «Ресурсы»

Взаимосвязи программ и задач могут быть множественными, т.е. каждая задача может быть связана с множеством программ, а каждая программа может быть связана с несколькими задачами. Порядок выполнения программ в рамках одной задачи задается порядком следования экземпляров объектов в таблице панели «Экземпляры».

3.4 Настройка параметров проекта

Настройка параметров проекта осуществляется в панели настройки проектов. Данная панель вызывается двойным нажатием на корневой элемент в дереве проектов и состоит из трех вкладок: «Конфигурационные переменные», «Свойства проекта», «Конфигурация».

Вкладка «Конфигурационные переменные» (рис. 13) позволяет добавлять дополнительные глобальные переменные. Назначение всех элементов на данной вкладке совпадает с назначением элементов панели ввода глобальных переменных на панели редактирования ресурсов.

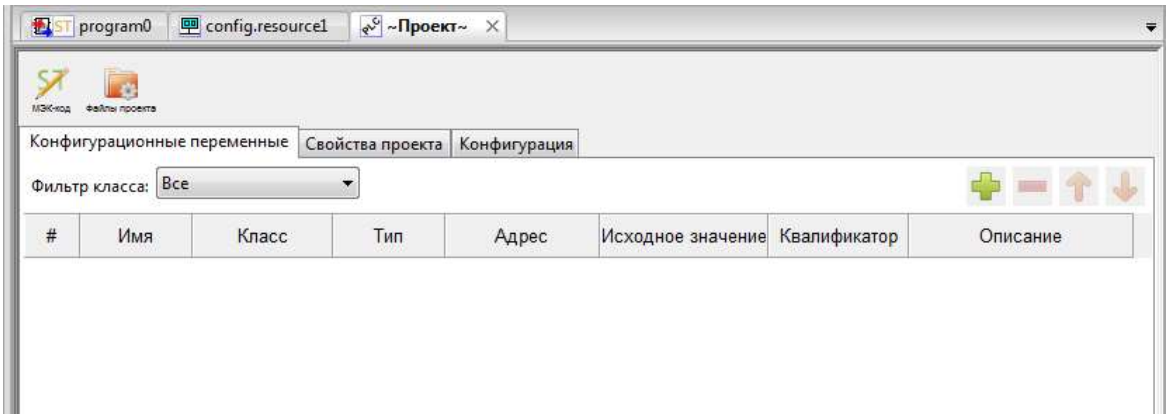


Рисунок 13 – Вкладка «Конфигурационные переменные» панели настройки проектов

Вкладка «Свойства проекта» позволяет задавать свойства проекта (рис. 14). На данной вкладке присутствуют следующие дополнительные вкладки:

- «Проект» (рис. 14а) — вводится имя и версия проекта (эта информация после передачи проекта в Контроллер будет доступна по запросу через протокол Modbus), а также имя, версия и релиз продукта;
- «Автор» (рис. 14б) — информация об авторах и компании, реализующей проект;
- «Графика» (рис. 14в) — дополнительные настройки представления графических языков программирования;

– «Прочее» (рис. 14г) — язык и текстовая информация о проекте.

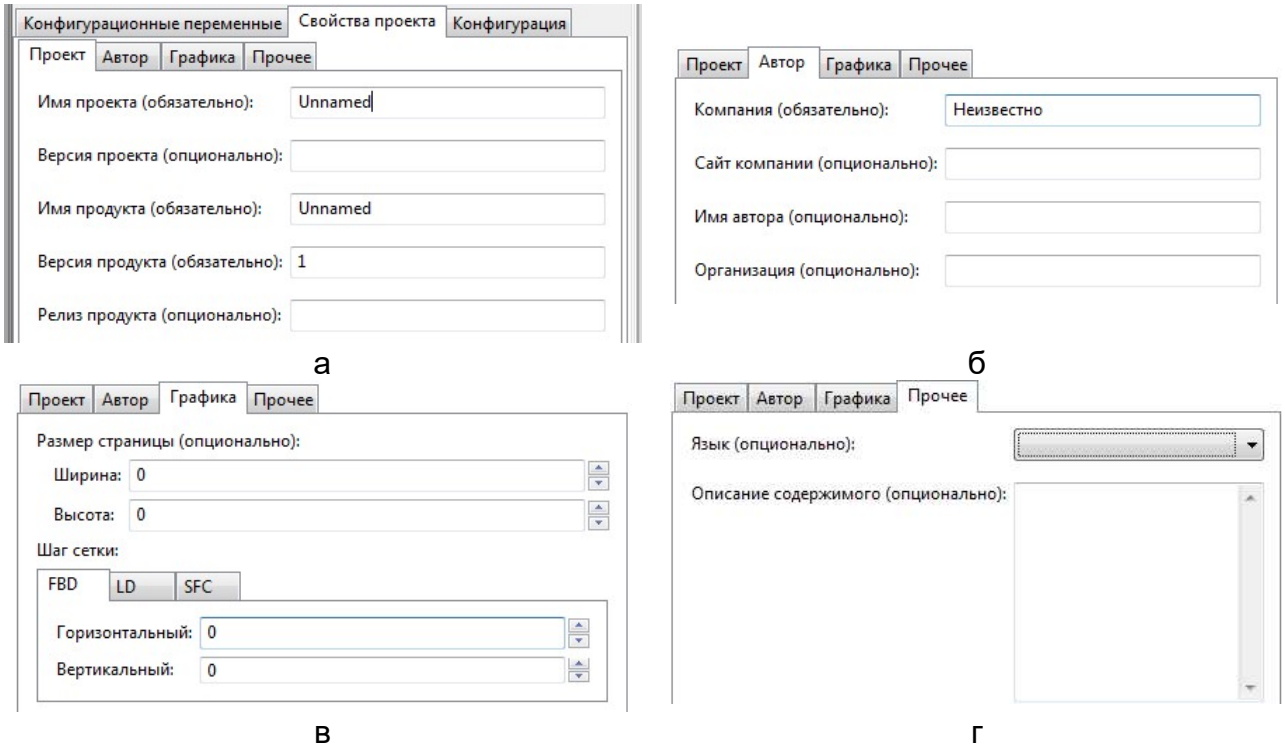


Рисунок 14 – Вкладка «Свойства проекта» панели настройки проектов

На вкладке «Конфигурация» содержатся конфигурационные параметры проекта, такие как: включение/выключение специфичных для Контроллера библиотек, целевая платформа, в которой можно настроить ключи сборки проекта. Пользователю доступен выбор версии целевой платформы из двух вариантов: «MKLogic201_HV_3_1» и «MKLogic201_HV_2_0». Платформа «MKLogic201_HV_2_0» устарела и снята с производства, поэтому в большинстве случаев необходимо выбрать платформу «MKLogic201_HV_3_1». Узнать версию платформы можно у производителя. Все остальные конфигурационные параметры на вкладке «Конфигурация» следует оставить без изменений.

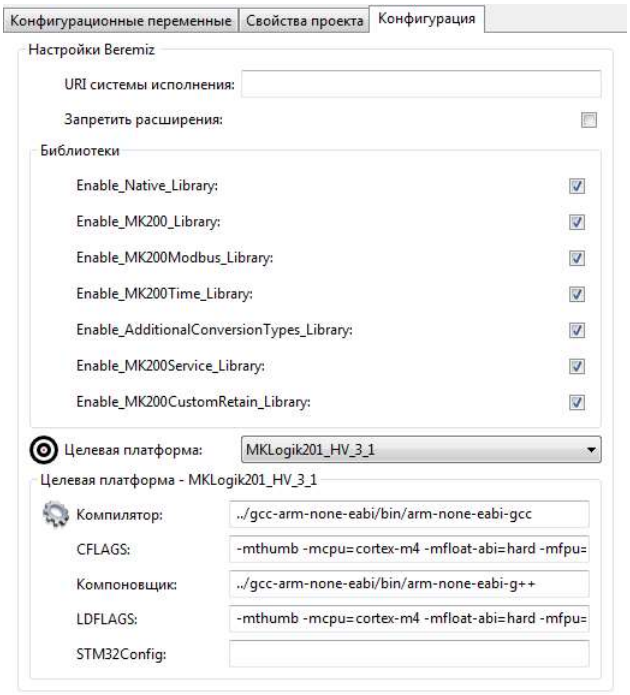


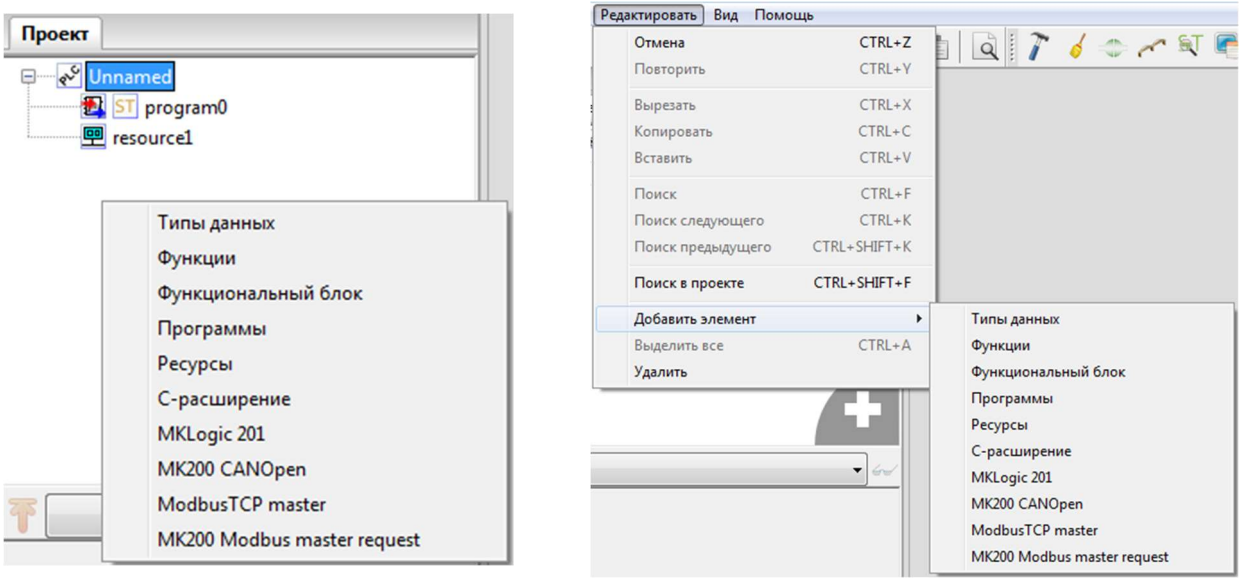
Рисунок 15 – Вкладка «Конфигурация» панели настройки проектов

3.5 Редактирование структуры проекта

Элементы внутри «Дерева проекта» можно добавлять и удалять. Для программных модулей дополнительно доступны операции копирования и вставки.

Добавление нового элемента осуществляется из меню добавления нового объекта, в котором содержатся список всех доступных для добавления элементов. Доступ к данному меню можно получить двумя способами:

– нажать левую кнопку мыши на пустом месте дерева проектов (рис. 16а);
– перейти в меню «Редактировать» → «Добавить элемент» (рис. 16б).

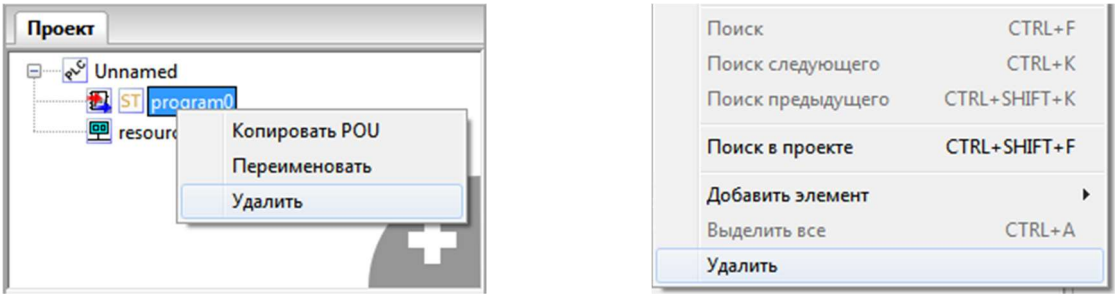


а б

Рисунок 16 – Способы добавления элемента в проект

Удалить элемент из проекта, можно также двумя способами:

– нажать правую кнопку мыши на элементе в дереве проекта и, в появившемся контекстном меню, выбрать пункт «Удалить» (рис. 17а);
– перейти в меню «Редактировать» → «Удалить» (рис. 17б).



а б

Рисунок 17 – Удаление элемента из проекта

3.6 Работа с программами проекта

3.6.1 Редактор текстовых языков программирования (ST и IL)

Для редактирования программ, написанных на языках ST и IL в среде Beremiz предусмотрен специализированный текстовый редактор, а для программ на языках FBD, SFC, LD — графический редактор.

Редактор языка ST представляет собой текстовый редактор с нумерацией строк (рис. 18 поз. 2). В верхней части редактора расположена панель переменных программы (рис. 18 поз.1).

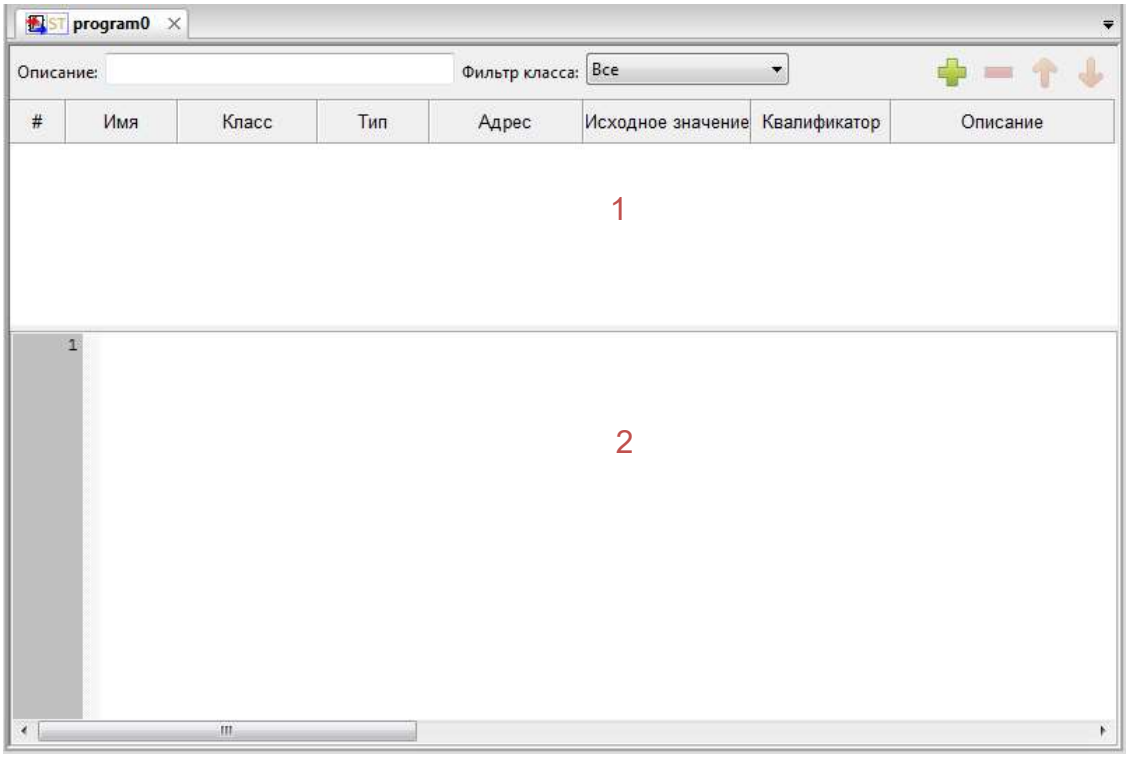


Рисунок 18 – Панель редактирования программы на языке ST

Панель переменных программы практически полностью совпадает с панелью переменных окна «Ресурсы» (рис. 9, поз. 1) и отличается наличием дополнительного фильтра по описанию переменной.

Редактор языков ST и IL предоставляет следующие возможности:

- подсветка синтаксиса кода (выделение синтаксических конструкций текста с использованием различных цветов, шрифтов и начертаний), что значительно облегчает чтение исходного кода и улучшает его визуальное восприятие;
- нумерация строк, которая существенно облегчает поиск и идентификацию ошибок в программе, так как транслятор кода ST в C выдаёт номер строки, в которой найдена ошибка;
- сворачивание кода структурных элементов языка: возможность редактора, позволяющая скрывать определённый фрагмент редактируемого кода или текста, оставляя лишь одну строку. Для того, чтобы свернуть фрагмент, нужно нажать на символ «–» слева от него. Чтобы увидеть весь фрагмент, то есть развернуть его, нужно нажать на символ «+», появляющийся у свернутых фрагментов;
- увеличение или уменьшение размера шрифта, которые производятся либо сочетанием клавиш Ctrl + «+» (увеличение) и Ctrl + «–» (уменьшение), либо колесом мыши, удерживая клавишу Ctrl.

Описание синтаксиса, основных конструкций и примеров использования языков ST и IL приведены в приложениях Б и В соответственно.

3.6.2 Редактор графических языков программирования (FBD, SFC, LD)

3.6.2.1 Редактор языка FBD

Основными элементами языка FBD являются:

- переменные;
- функциональные блоки;
- подключения.

При открытии редактора языка FBD, появится панель инструментов редактора FBD (рис. 1), посредством которой можно управлять диаграммой и добавлять на нее новые элементы.

Рисунок 19 – Панель инструментов редактора FBD

Назначение кнопок панели инструментов редактора FBD приведено в таблице 10.

Таблица 10 – Назначение кнопок панели инструментов редактора FBD

Внешний вид кнопки	**Наименование кнопки**	**Функциональное назначение кнопки**
	Выбор объекта	Переключение графического редактора в режим выбора объектов при помощи указателя мыши
	Переместить отображение	Активирует режим быстрого перемещения по области просмотра
	Создать новый комментарий	Открывает диалог ввода комментария
	Создать новую переменную	Добавление новой переменной
	Создать новый блок	Добавление нового функционального блока
	Создать новое подключение	Добавление нового подключения

Добавление нового элемента на диаграмму FBD возможно двумя способами:

- нажать кнопку на панели инструментов редактора FBD, соответствующую добавляемому элементу, затем щелкнуть указателем мыши в свободном месте области редактирования FBD диаграммы;
- нажать правую клавишу мыши в свободном месте области редактирования FBD-диаграммы и в появившемся контекстном меню выбрать пункт «Добавить», в котором в свою очередь выбрать нужный элемент.

В появившемся действий окне необходимо задать требуемые параметры.

Свойства любого элемента на диаграмме доступны по двойному щелчку кнопкой мыши по данному элементу.

При добавлении нового блока будет отображено окно свойства блока (рис. 20). Для блока необходимо задать количество входов и определить порядок исполнения (подробнее о порядке исполнения см. ниже). Кроме того, можно добавить дополнительный вход управления исполнением.

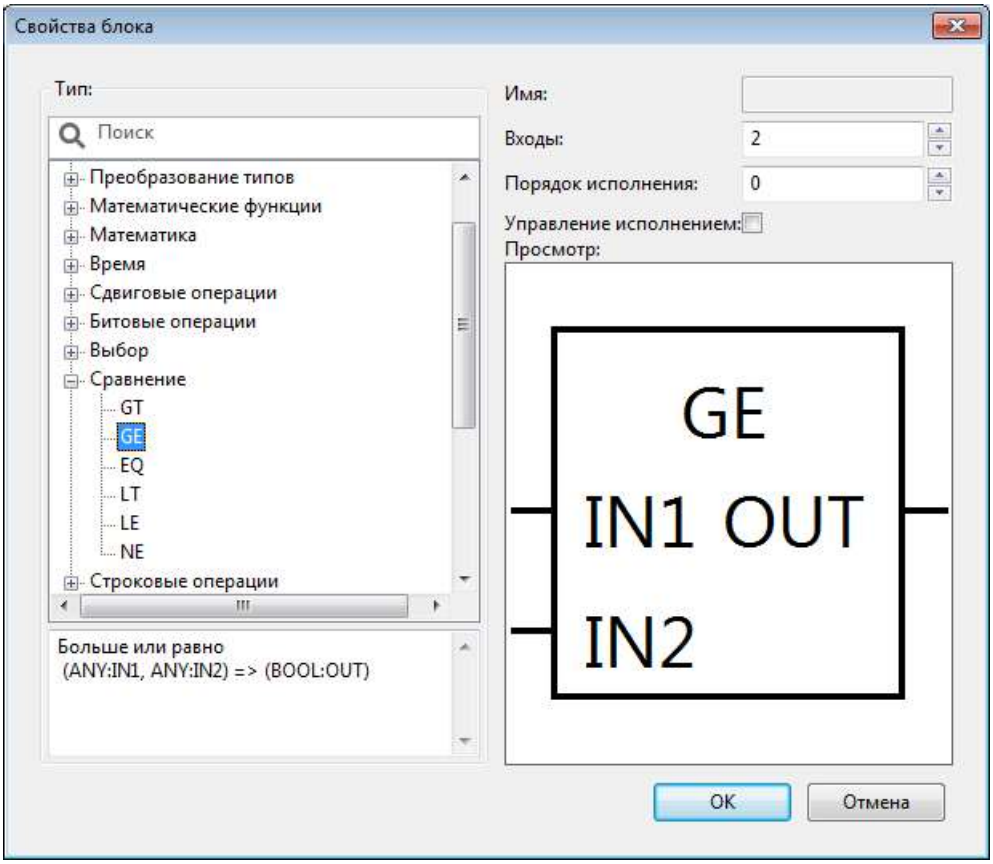


Рисунок 20 – Окно добавления нового блока в редакторе FBD

Примечание: Все добавленные блоки будут отображены также в таблице на панели переменных. Однако добавление новых блоков, использую данную таблицу, невозможно — с ее помощью можно добавить только переменные.

Добавление новой переменной происходит в два этапа. Сначала необходимо добавить переменную в таблицу панели переменных и определить ее свойства. Далее нужно поместить созданную переменную на диаграмму. Это можно сделать описанным выше способом выбрав пункт «Переменная» контекстного меню или нажав соответствующую кнопку (см. таблицу 10). При этом появится окно «Свойства переменных» (рис. 21), в котором нужно выбрать переменную из списка, указать Класс и порядок исполнения. Класс определяет направление переменной на диаграмме («Вход» — переменная передает значение на вход другого элемента диаграммы; «Выход» — переменная получает значение с выхода другого элемента диаграммы; «вход/выход» — получает и передает значение) и жестко не связан с Классом в таблице переменных.

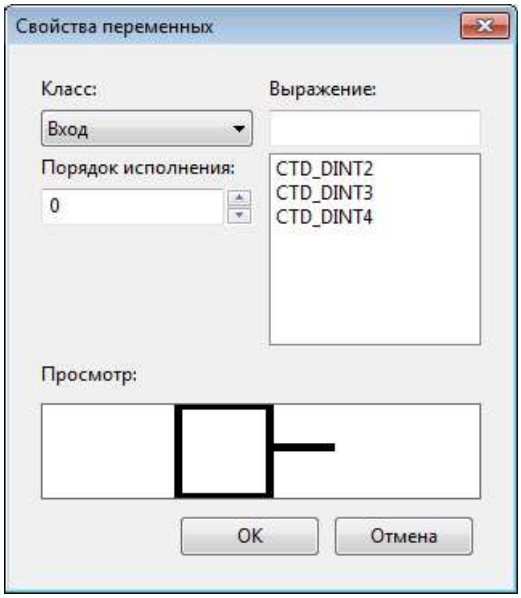


Рисунок 21 – Окно добавления новой переменной в редакторе FBD

Кроме того, переменную можно перетащить мышкой в область редактирования FBD диаграмм. Для этого нужно щелкнуть левой кнопкой мыши по номеру переменной и, не отпуская кнопку, перевести указатель в область редактирования диаграмм.

Добавление подключения осуществляется также как и добавление блока. При этом отображается окно «Свойства подключения» (рис. 22), в котором необходимо ввести имя подключения и выбрать его тип.

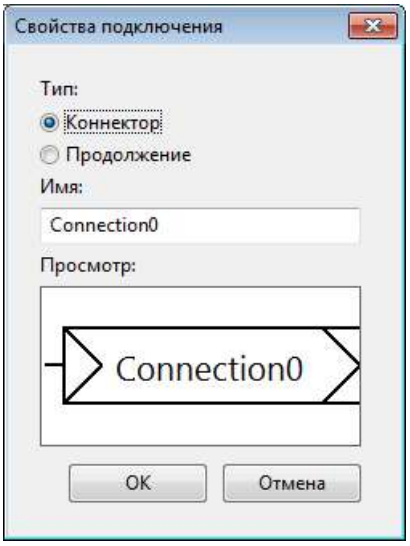


Рисунок 22 – Окно добавления нового подключения в редакторе FBD

Возможны два типа подключения:

- «Коннектор» — принимает значение с выхода другого элемента;
- «Продолжение» — передает принятое значение на вход другого элемента диаграммы.

Добавление комментариев осуществляется аналогично добавлению других элементов диаграммы. При добавлении комментариев на диаграмму отображается окно, в котором требуется ввести новый комментарий (рис. 23).

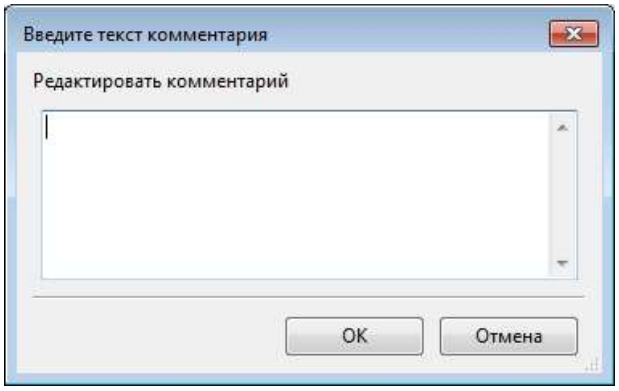


Рисунок 23 – Окно добавления нового комментария в редакторе FBD

Последовательность выполнения элементов диаграммы определяется либо вручную параметром «Порядок выполнения», либо автоматически для элементов, у которых этот параметр равен 0.

Автоматическая последовательность выполнения регламентируется следующим образом: чем выше и левее расположен верхний левый угол графического отображения элемента диаграммы, тем раньше данная элемент будет выполнен, то есть элементы диаграммы выполняются слева направо и сверху вниз.

При ручном определении порядка выполнения (параметр «Порядок выполнения» не равен 0) в правом нижнем углу элемента появляется порядковый номер его выполнения.

Для блоков языка FBD доступно быстрое добавление связанных блоков. Для этого необходимо щелкнуть левой кнопкой мыши по выходу уже установленного на диаграмму блока. Появится контекстное меню, в котором следует выбрать тип нового элемента, который требуется подключить (рис. 24). После выбора типа элемента в данном меню появится стандартный диалог добавления элемента. Отличием данного метода является то, что добавленный быстрым способом элемент оказывается уже подключенным к исходному блоку (на выходе которого вызывалось контекстное меню). Таким образом, в данном способе отсутствует необходимость дополнительно соединять элементы между собой.

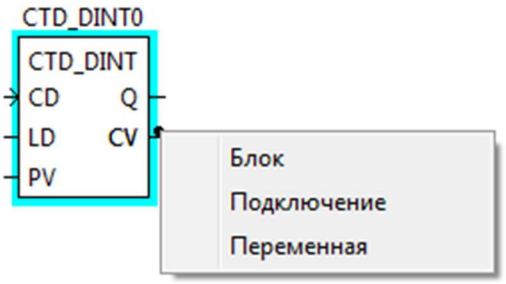


Рисунок 24 – Быстрое добавление блоков в программе на языке FBD

Описание языка FBD, основных его конструкций, а также пример его использования приведены в приложении Г.

3.6.2.2 Редактор языка LD

Язык LD (язык релейно-контактной логики) представляет собой графическую форму записи логических выражений в виде контактов и катушек реле. Основными элементами языка LD являются: шина питания, катушка, контакт.

При открытии редактора языка LD, появится соответствующая панель инструментов (рис. 25), посредством которой можно управлять диаграммой и добавлять на нее новые элементы. На одной диаграмме могут быть использованы как элементы языка LD, так и FBD.

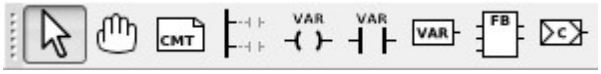


Рисунок 25 – Панель инструментов редактора диаграмм (для языка LD)

В таблице 11 приведено функциональное назначение кнопок панели инструментов редактора для языка LD. Назначение остальных кнопок совпадает с назначением кнопок для языка FBD.

Таблица 11 – Назначение кнопок панели инструментов редактора диаграмм (для языка LD)

Внешний вид кнопки	**Наименование кнопки**	**Функциональное назначение кнопки**
	Создать новую линию питания	Добавление новой шины питания
	Создать новую катушку	Добавление новой катушки
	Создать новый контакт	Добавление нового контакта

Добавление нового элемента на диаграмму осуществляется так же как для языка FBD.

При добавлении новой шины питания откроется окно «Свойства шины питания» (рис. 26), в котором можно выбрать направление ответвлений шины в поле «Тип» («Левая шина питания» или «Правая шина питания»), а также количества пинов (поле «Номер пина»).

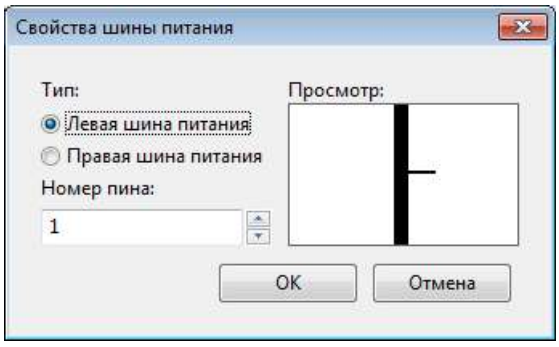


Рисунок 26 – Окно добавления новой шины питания

Элементы катушка и контакт требуют для своей работы наличия в программе булевой переменной. Поэтому перед созданием этих элементов необходимо в панели переменных данного программного модуля создать переменную типа BOOL.

При добавлении новой катушки откроется окно редактирования значения катушки (рис. 26), в котором нужно выбрать «Модификатор» катушки и переменную (ранее созданную).

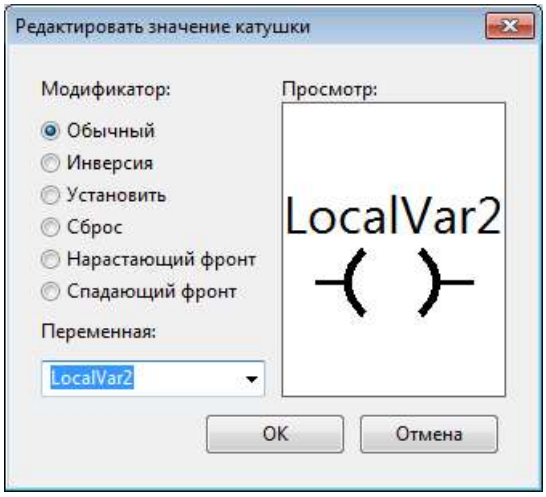


Рисунок 27 – Окно добавления новой шины питания

Для катушки доступны следующие модификаторы: «Обычный», «Инверсия», «Установить», «Сброс», «Нарастающий фронт», «Спадающий фронт». Описание этих модификаторов можно найти в приложении Д.

Катушку можно добавить в область LD-диаграммы из таблицы переменных и констант простым перетаскиванием указателем мыши (нажать левую кнопку на номере переменной и перетащить в область диаграммы). При этом переменная должна относиться к классу «Выход» (рис. 28).

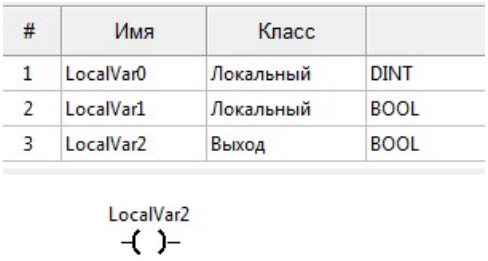


Рисунок 28 – Добавление катушки на диаграмму из панели переменных

При добавлении на диаграмму нового контакта откроется окно редактирования значения контакта (рис. 29), в котором нужно выбрать «Модификатор» катушки и переменную (ранее созданную).

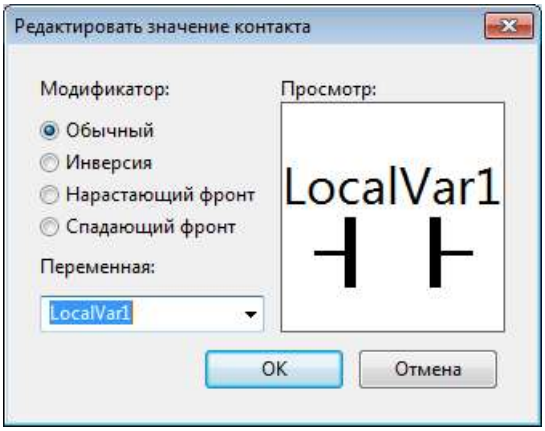


Рисунок 29 – Добавление контакта на диаграмму

Для контакта можно задать следующие модификаторы: «Нормальный», «Инверсный», «Нарастание фронта», «Спад фронта». Описание этих модификаторов можно найти в приложении Д.

Контакт, также как и катушку, можно добавить в область LD-диаграммы из таблицы переменных и констант простым перетаскиванием указателем мыши (нажать левую кнопку на номере переменной и перетащить в область диаграммы). При этом переменная должна относиться к любому из доступных классов, кроме «Выход».

Для элементов языка LD также доступно быстрое добавление связанных элементов. Для этого левой кнопкой мыши на выходе уже установленного на диаграмму элемента вызывается контекстное меню, в котором можно выбрать тип нового элемента (рис. 30). Добавленный таким способом элемент оказывается уже подключенным к исходному блоку (на выходе которого вызывалось контекстное меню).

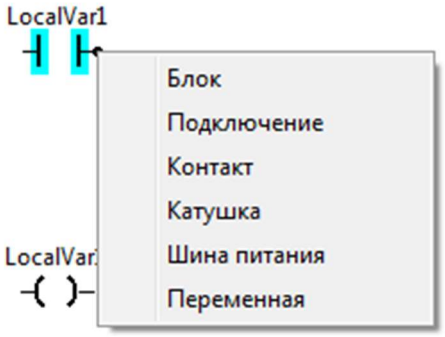


Рисунок 30 – Быстрое добавление блоков в программе на языке LD

Описание языка LD, основных его конструкций, а также пример его использования приведены в приложении Д.

3.6.2.3 Редактор языка SFC

Основными элементами языка SFC являются: начальный шаг, шаг, переход, блок действий, ветвление и объединение, безусловный переход. Программа на языке SFC состоит из набора шагов, связанных переходами.

Панель редактирования SFC-диаграммы показана на рис. 31.

Рисунок 31 – Панель инструментов редактора (для языка SFC)

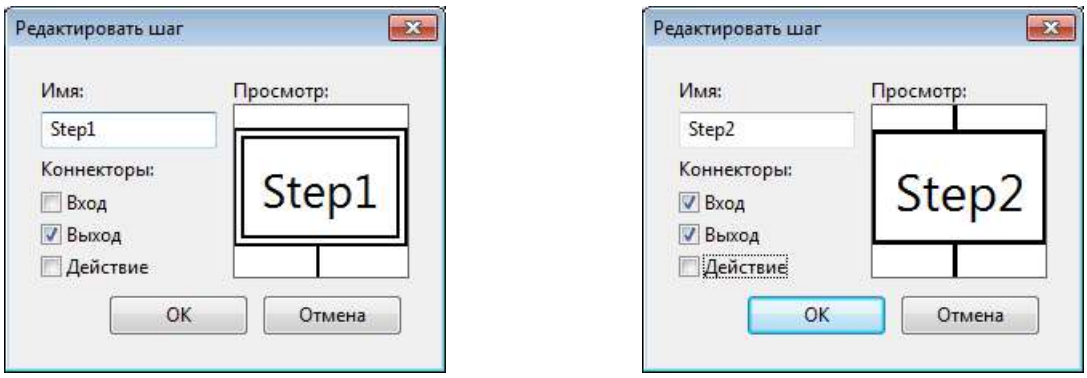
Функциональное назначение кнопок панели инструментов редактора для языка SFC представлено в таблице 12. Назначение остальных кнопок совпадает с назначением кнопок для языков LD и FBD.

Таблица 12 – Назначение кнопок панели инструментов редактора (для языка SFC)

Внешний вид кнопки	**Наименование кнопки**	**Функциональное назначение кнопки**
	Создать новый исходный шаг	Вызов диалога редактирования исходного шага
	Создать новый шаг	Вызов диалога редактирования шага
	Создать новый переход	Вызов диалога редактирования перехода
	Создать новый блок действий	Вызов диалога редактирования блока действий
	Создать новое ветвление	Вызов диалога создания нового ветвления или объединения
	Создать новый безусловный переход	Вызов диалога создания нового безусловного перехода

Согласно стандарту IEC 61131-3, на SFC-диаграмме должен быть один шаг инициализации (исходный шаг), который характеризует начальное состояние SFC-диаграммы. Шаг инициализации отображается в виде прямоугольника с двойной рамкой.

При добавлении шага (исходного и обычного) появляется окно редактирования шага (рис. 32). Окно редактирования исходного шага отличается от обычного только отображением шага на диаграмме.



а б

Рисунок 32 – Окно редактирования шага (для языка SFC):
а) исходного шага, б) обычного шага.

Для шага нужно указать имя, выбрать коннекторы: «Вход», «Выход» и «Действие». Вход шага отображается сверху, выход — снизу, а действие — справа.

Шаги в SFC диаграмме соединяются между собой посредством переходов. При добавлении нового перехода появится окно редактирования перехода (рис. 33), в котором нужно выбрать тип перехода и его приоритет.

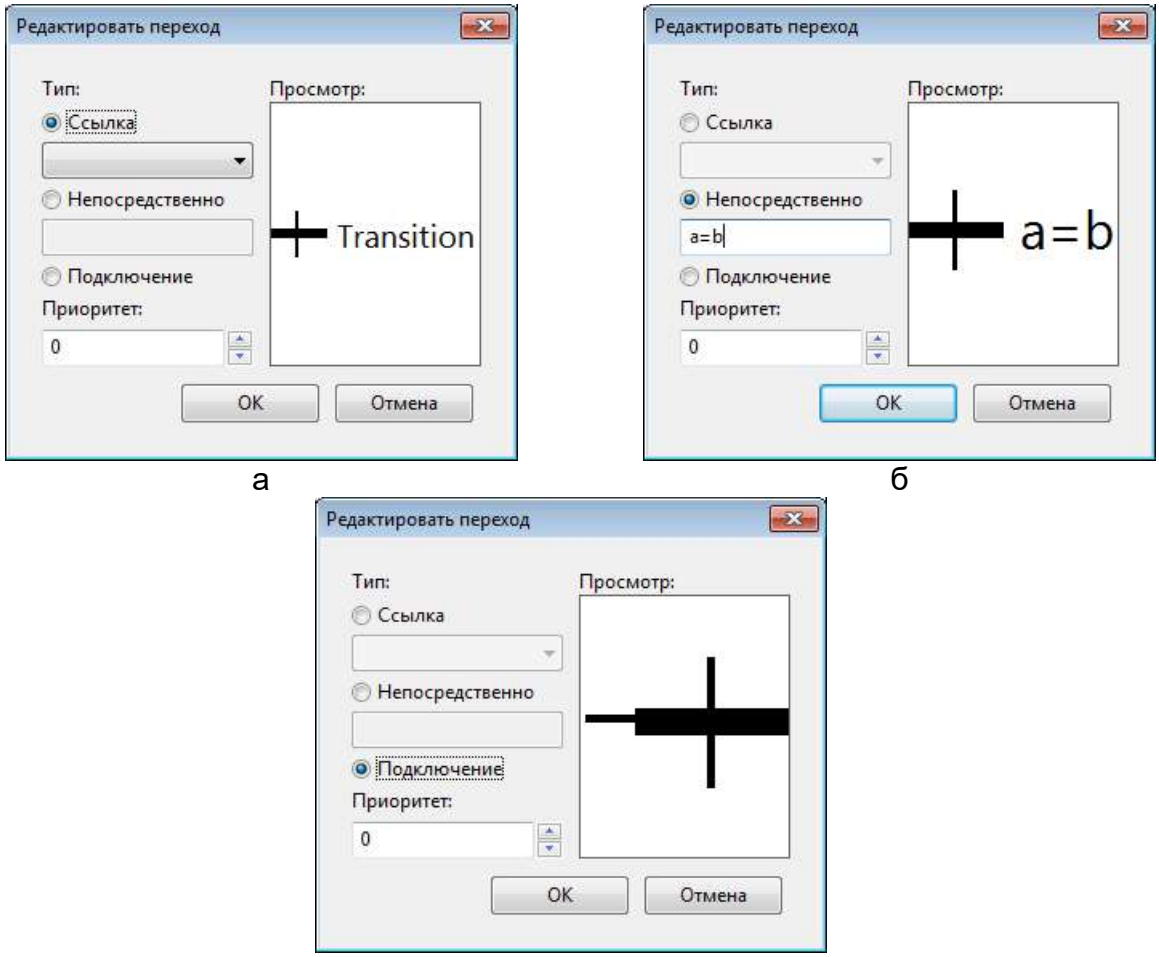


Рисунок 33 – Окно редактирования перехода

Возможны следующие типы переходов: «Ссылка», «Непосредственно» и «Подключение».

Для перехода типа «Ссылка» в выпадающем списке следует выбрать переходы, предопределённые в дереве проекта для SFC-модуля. Добавление предопределённого перехода описывается ниже после описания всех добавляемых элементов языка SFC.

Для перехода типа «Непосредственно», условие перехода можно задать логическим выражением на языке ST в соответствующей строке окна. Данное выражение может быть, например, сравнением каких-либо переменных. Можно в качестве условия использовать имя переменной типа BOOL, либо другого целочисленного типа; в этом случае условие не выполняется при значении равном 0 и выполняется при всех остальных значениях.

Для перехода типа «Подключение» в качестве условия перехода можно использовать выходные значения элементов на языках FBD или LD.

Действие к шагам добавляется при помощи блока действие. При его добавлении появляется окно редактирования свойства блока действия.

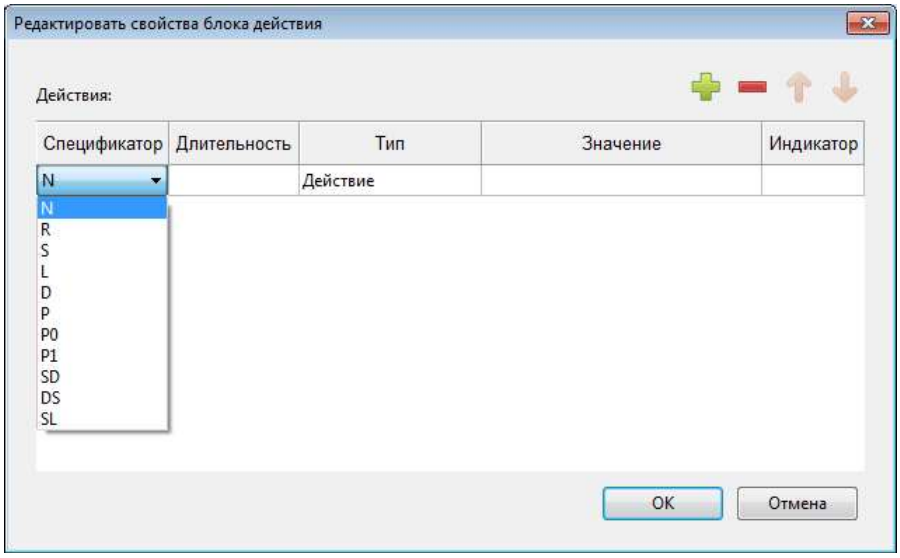


Рисунок 34 – Окно редактирования свойств действия.

Блок действия может содержать множество действий. Каждое действие описывается отдельной строкой в таблице окна редактирования свойств действия. Добавление, удаление и перемещение действий осуществляется стандартным образом (см. табл. 7). После добавления для каждого действия требуется установить необходимые параметры:

- «Квалификатор»;
- «Продолжительность»;
- «Тип» («Действие», «Переменная» и «Непосредственно»);
- «Значение»;
- «Индикатор».

Поле «Квалификатор» определяет время начала действия, его продолжительность и момент окончания. Квалификатор выбирается из выпадающего списка в ячейке «Квалификатор» (см. рис. 34). Описание квалификаторов приведено в таблице 13.

Таблица 13 – Квалификаторы действий SFC диаграммы

Наименование квалификатора	Поведение блока действия
D	Действие начинает выполняться через некоторое заданное время (если шаг еще активен) и выполняется до тех пор, пока данный шаг активен
L	Действие выполняется в течение некоторого заданного интервала времени, после чего выполнение действия останавливается
N	Действие выполняется, пока данный шаг активен
P	Действие выполняется один раз, как только шаг стал активен
S	Действие активируется и остается активным пока SFC диаграмма выполняется
R	Действие выполняется, когда диаграмма деактивизируется
DS	Действие начинается выполняться через некоторое заданное время, только в том случае если шаг еще активен
SL	Действие активно в течение заданного интервала
SD	Действие начинается выполняться через некоторое время, даже в том случае если шаг уже не активен

В поле «Продолжительность» устанавливается интервал времени, необходимый для выполнения некоторых квалификаторов, описанных в таблице 13.

Содержимое поля «Тип» определяет код или конкретную манипуляцию, которая будет выполняться во время активации действия.

Если тип действия задан как «Действие», то будет выполнено одно из предопределённых действий заданных для данного программного модуля (рис. 35а). Подробнее о предопределенных действиях можно ознакомиться в конце этого раздела.

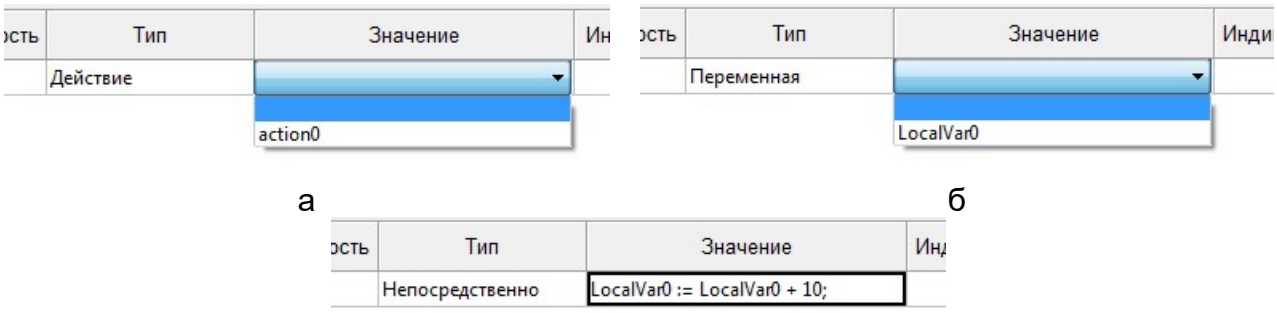


Рисунок 35 – Выбор типа действия в окне редактирования блока действия.

Если значение ячейки «Тип» выбрано «Переменная», то в ячейке «Значение» необходимо выбрать переменную из выпадающего списка переменных, заданных в панели переменных данного программного модуля.

Значение переменной в этом случае определяется состоянием действия и шага:

- при активации шага — принимает нулевое значение (0, 0.0, FALSE и др. в зависимости от типа);
- в начале выполнения блока действия — принимает единичное значение (1, 1.0, TRUE и др. в зависимости от типа);
- по завершению выполнения блока — снова принимает нулевое значение.

Если тип действия определен как «Непосредственно», в поле «Значение» вводится программный код на языке ST, который будет выполняться, когда действие становится активным.

Примечание: В отличие от типа «Непосредственно» для перехода, где вводится логическое выражение на языке ST, в действии вводится код на языке ST, который производит некоторые действия. Поэтому данное выражение должно заканчиваться символом «;», как требует синтаксис языка ST.

После того, как блок действия будет добавлен на диаграмму SFC, необходимо его связать с конкретным шагом. Операция связывания выполняется добавлением провода между коннектором «Действие» у шага (с правой стороны графического обозначения) и коннектором с левой стороны блока действия (рис. 36).

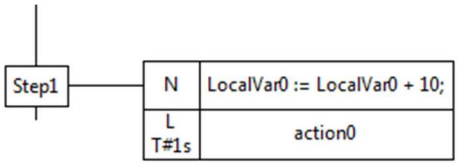


Рисунок 36 – Связывание шага и блока действия.

Для управления множественными переходами в языке SFC также предназначены ветвления и объединения. При добавлении нового ветвления появляется окно создания нового ветвления или объединения (рис. 37), в котором нужно выбрать тип ветвления и указать количество ветвей.

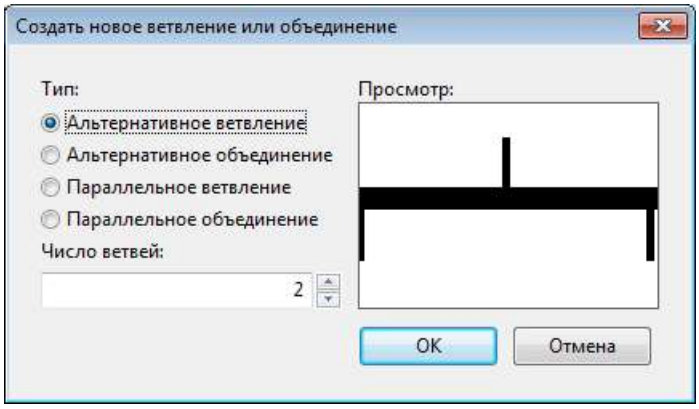


Рисунок 37 – Связывание шага и блока действия.

Разница между альтернативным и параллельным ветвлением/объединением заключается в том, что ветви параллельного ветвления/объединением можно подключать напрямую к входам/входам шагов, а ветви альтернативного ветвления/объединения только через дополнительные переходы.

В SFC диаграммах SFC существует аналог оператора GOTO других языков программирования — безусловный переход, т.е. переход к определённой точке программы, обозначенной номером строки либо меткой. Для добавления безусловного перехода на SFC-диаграмму, в окне добавления нового безусловного перехода (рис. 18), необходимо выбрать из списка шаг, к которому будет происходить безусловный переход.

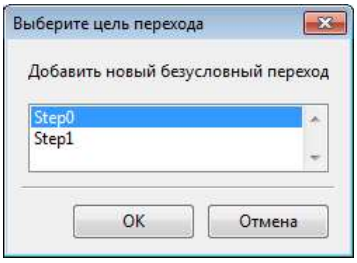


Рисунок 38 – Окно добавления безусловного перехода

Согласно стандарту IEC 61131-3, между шагом и безусловным переходом должен обязательно быть определён переход. На SFC-диаграмме безусловный переход отображается стрелкой, которую необходимо соединить с переходом. Справа от стрелки располагается наименование шага, к которому осуществляется безусловный переход в случае выполнения условия перехода, находящегося выше и соединённого с ней.

В редакторе SFC существует дополнительная возможность быстрого добавления элементов диаграммы (рис. 39). Для этого по выходу (вывод направленный вниз) элемента нужно щелкнуть левой кнопкой мыши. При этом появится контекстное меню со списком элементов, доступных для подключения к выходу данного элемента. Выбором соответствующего пункта контекстного меню, открывается окно ввода свойств данного элемента. После чего, на диаграмме появится элемент, уже связанный с элементом, выбранным в начале процедуры быстрого добавления.

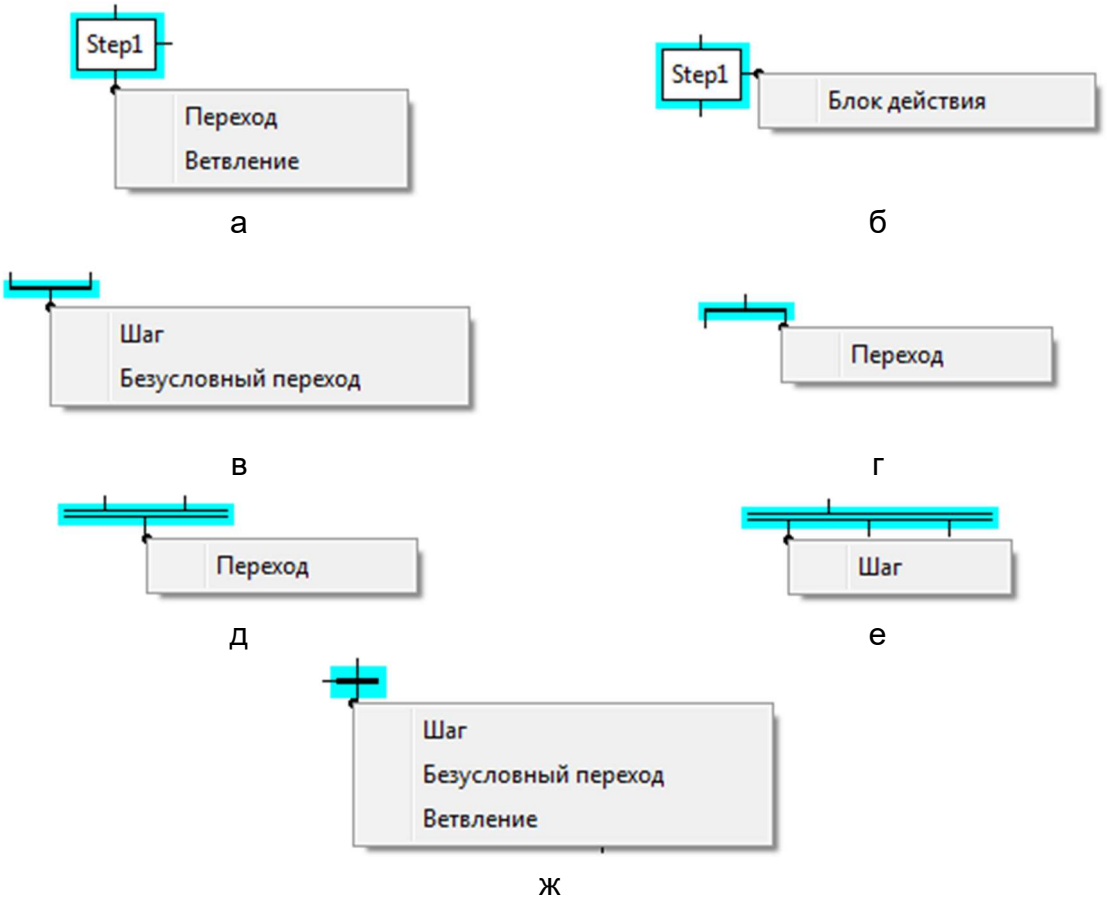


Рисунок 39 – Быстрое добавление блоков в программе на языке SFC

Кроме того, в программе на языке SFC имеется возможность добавлять предопределенные переходы и действия.

Предопределенные переходы используются в случаях, когда необходимо использовать одно и то же условие перехода между множеством шагов. Чтобы определить такое условие перехода, необходимо в контекстном меню SFC-диаграммы в дереве проекта выбрать «Добавить переход» (рис. 40).

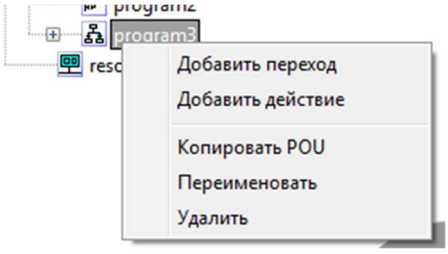
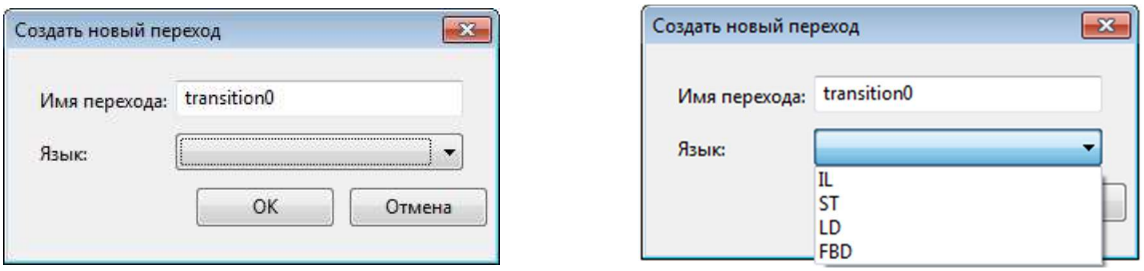


Рисунок 40 – Контекстное меню в дереве проектов программы на языке SFC

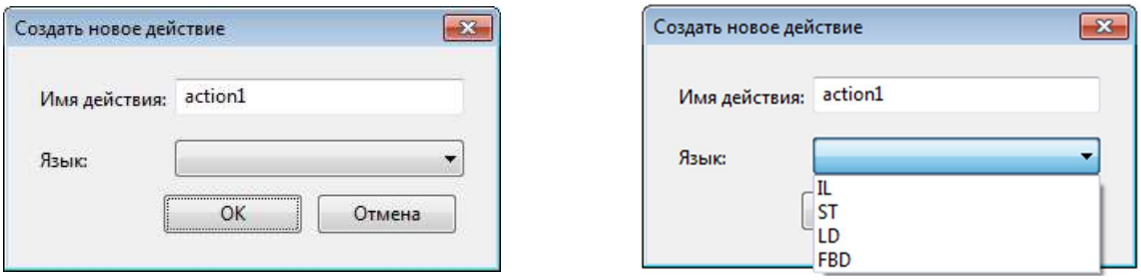
Далее в окне «Создать новый переход» (рис. 41а) необходимо задать уникальное имя для перехода и язык описания условия перехода. Условие перехода можно задать на четырех языках IL, ST, LD и FBD (рис. 41б).



а б

Рисунок 41 – Окно создания нового предопределенного перехода

Для того, чтобы добавить предопределенное действие нужно в контекстном меню (рис. 40) выбрать «Добавить действие». После чего в окне «Создать новое действие» (рис. 42) задать уникальное имя действия и язык описания действия. Действие, как и условие перехода, можно задать на четырех языках IL, ST, LD и FBD (рис. 42б).



а б

Рисунок 42 – Окно создания нового предопределенного действия

После того как действие создано, необходимо реализовать его код на выбранном текстовом или графическом языке. После добавления переходов и действий в дерево проекта, они становятся доступными для множественного использования.

Более подробное описание языка SFC, основных его конструкций и примеры его использования приведены в приложении Е.

3.7 Работа с типами данных

В среде Beremiz существует возможность создания нового типа данных путем добавлением элемента «Типы данных» в проект.

После добавления нового типа откроется панель редактирования нового типа данных (рис. 43). Главным параметром типа данных является механизм создания нового типа. Существует возможность выбора механизма создания нового типа из выпадающего списка, который содержит: «Синоним», «Поддиапазон», «Перечисление», «Массив», «Структура».

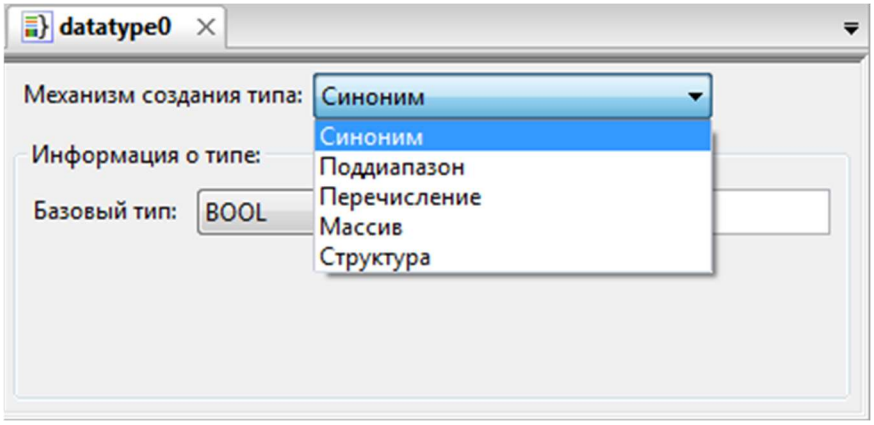


Рисунок 43 – Панель редактирования нового типа данных

Пользовательские типы данных могут применяться наряду со стандартными типами при реализации алгоритмов и логики выполнения программных модулей.

Механизм создания нового типа «Синоним» определяет новое имя стандартного типа. Для нового типа можно задать исходное значение по умолчанию.

При выборе механизма создания нового типа «Поддиапазон», помимо базового типа и начального значения, необходимо задать параметры «Минимум» и «Максимум», которые соответствуют минимальному и максимальному значениям создаваемого типа данных. Базовым типом может выступать только целочисленные типы.

Перечисляемый тип создается выбором механизма создания нового типа «Перечисление» (рис. 44). При этом в таблицу «Значения» необходимо занести набор значений, которые могут быть использованы в программном коде для переменных данного типа.

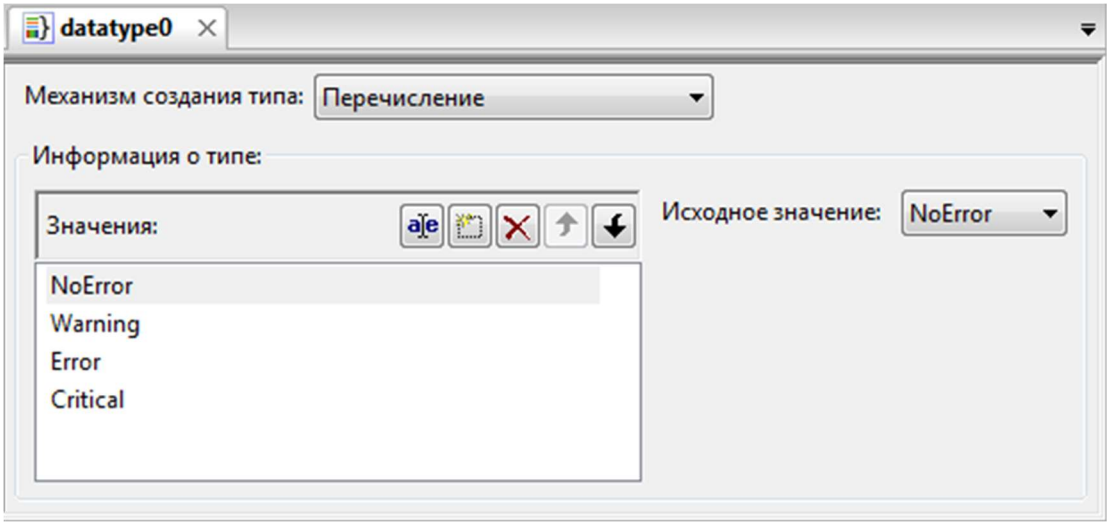


Рисунок 44 – Пример создания перечисляемого типа данных

При выборе механизма создания нового типа «Массив» (рис. 45) необходимо указать базовый тип, начальное значение и размерность массива. Размерность массива задаётся в формате: *<начальное значение>..<конечное значение>.*

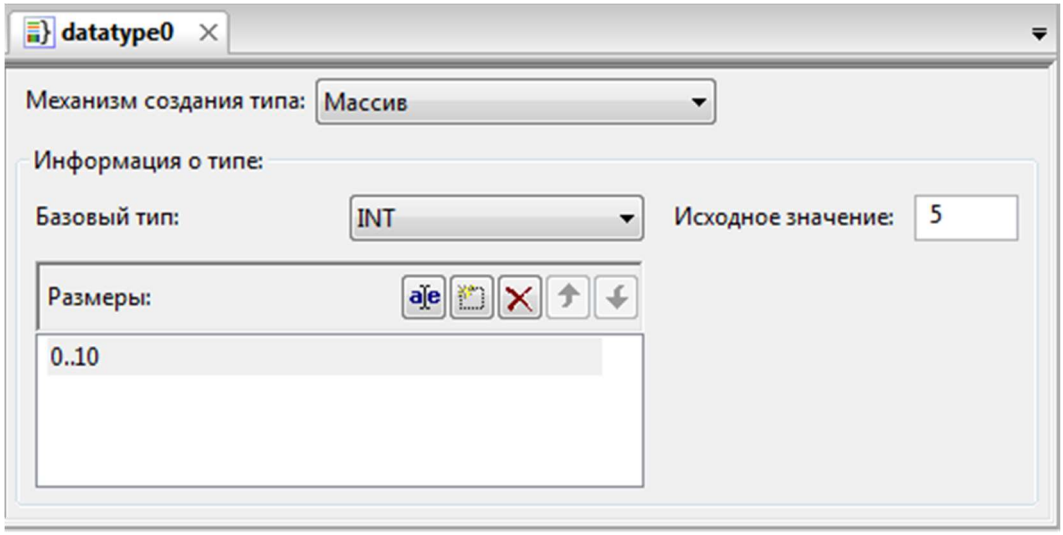


Рисунок 45 – Пример создания типа данных массив

Механизм создания нового типа «Структура» (рис. 46) позволяет определить тип, основанный на объединении несколько типов. При выборе этого типа появится панель похожая на панель переменных в программе, в которой необходимо добавить все переменные члены структуры.

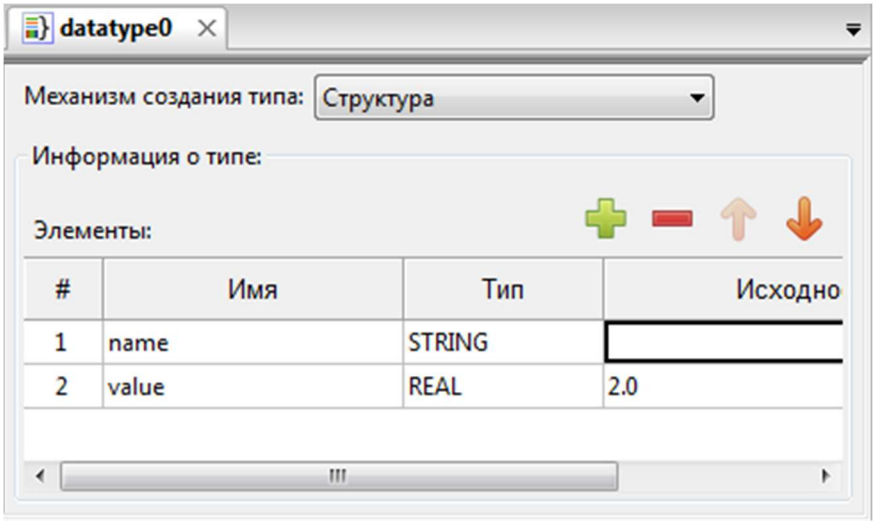


Рисунок 46 – Пример создания типа данных структуры

3.8 Работа с функциями

Функция является программным модулем в среде Beremiz и может быть элементом проекта. При добавлении новой функции будет отображено окно создания нового POU (рис. 47).

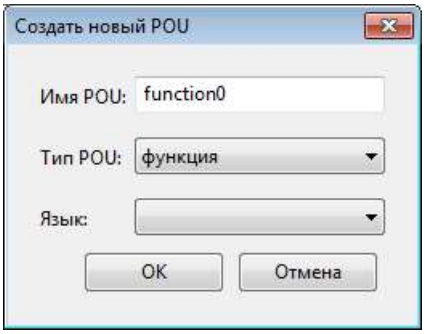


Рисунок 47 – Окно добавления новой функции

Функция может быть написана только на четырех языках IL, ST, FBD, LD. Язык SFC не доступен для функций. После выбора языка и создания функции появится панель редактирования функции (рис. 48), где нужно выбрать возвращаемый тип функции, добавить переменные и константы, написать код функции.

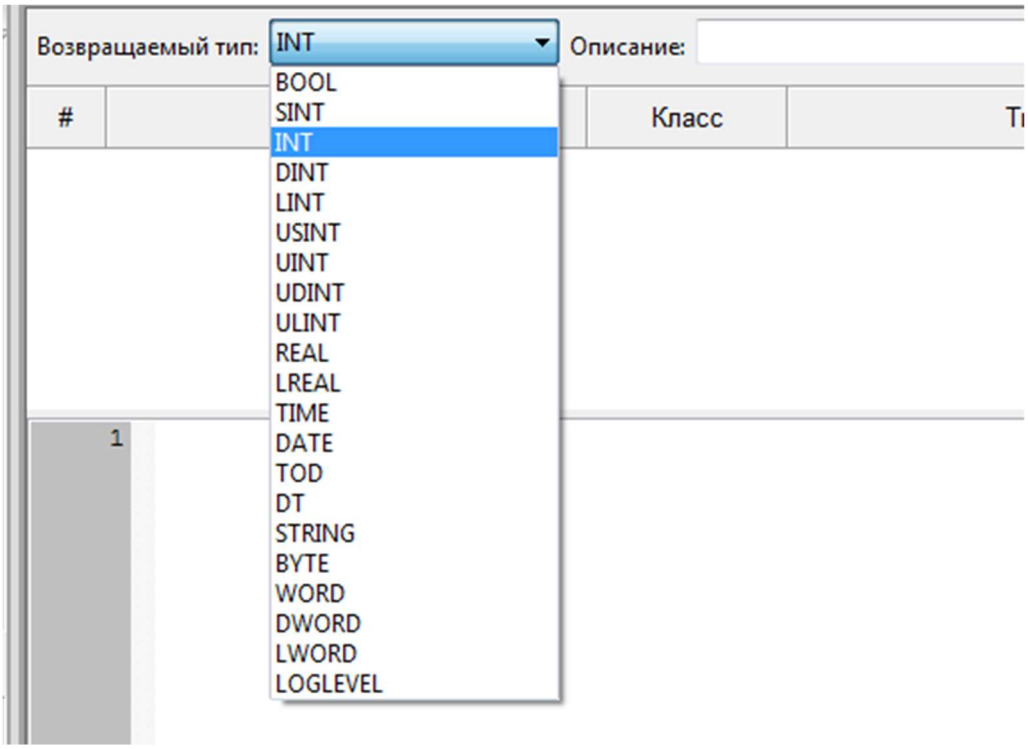


Рисунок 48 – Панель редактирования новой функции

В языках стандарта IEC 61131-3 функция всегда возвращает какое-либо значение — невозможно создать функцию без возвращаемого значения, как, например, в языке Си.

Внутри функции имя самой функции рассматривается как переменная и значение, переданное этой переменной, будет возвращено в результате вызова функции (рис. 49).

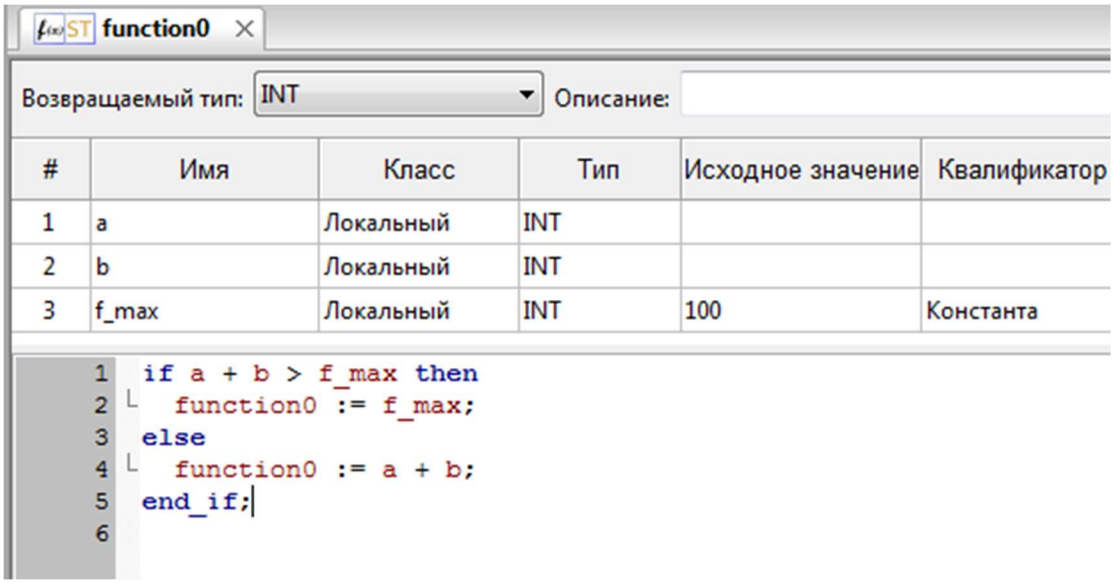


Рисунок 49 – Пример простой функции

В среде Beremiz предусмотрено преобразование функции в другие программные модули. Для этого в контекстном меню на элементе функционального блока в дереве проекта нужно выбрать пункт «Сменить тип POU на» (рис. 53), в котором выбрать новый тип программного модуля. Функцию можно преобразовать либо в программу, либо в функциональный блок.

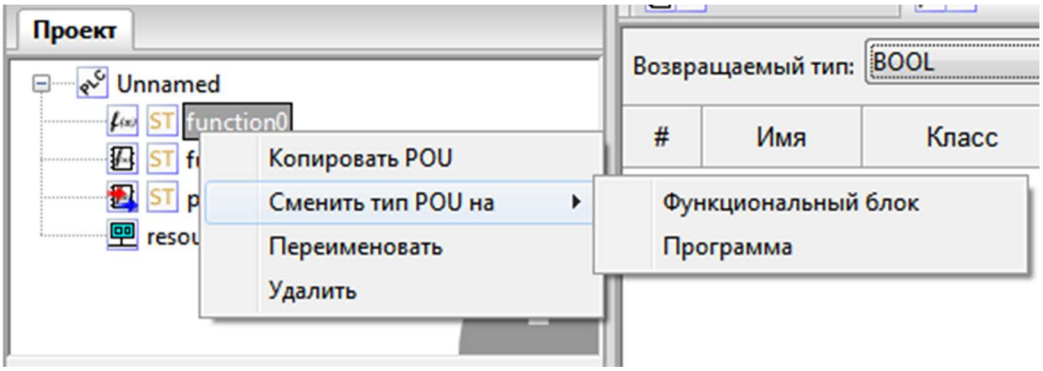


Рисунок 50 – Смена типа программного модуля функционального блока

Обратное преобразование программы в функцию или функционального блока в функцию невозможно. Но его можно отменить, выбрав в главном меню пункт «Редактировать» -> «Отмена».

Все добавленные в проект функции появятся на панели «Библиотеки» вместе со всеми остальными функциями и функциональными блоками в разделе «Пользовательские POU».

3.9 Работа с функциональными блоками

Функциональный блок – это программный модуль, который принимает и возвращает произвольное число значений. Функциональный блок имеет сходство одновременно и с функцией и типом данных. Так же, как и в случае с типом данных, для работы с функциональным блоком требуется создание одного или нескольких его экземпляров. Каждый экземпляр имеет собственные независимые данные, содержащие входные, выходные и внутренние переменные. Но в тоже время, аналогично функции,

функциональный блок имеет исполняемую часть, которая написана на одном из языков стандарта IEC 61131-3. Вызов исполняемой части происходит по имени экземпляра.

Входные и выходные переменные функционального блока, при использовании его на диаграмме, отображаются как коннекторы слева и справа соответственно. Функциональный блок может использовать все поддерживаемые средой языки, в том числе SFC.

Добавление пользовательского функционального блока происходит аналогично добавлению функции. В окне создания нового функционального (рис. 51) блока нужно выбрать его имя и язык программирования.

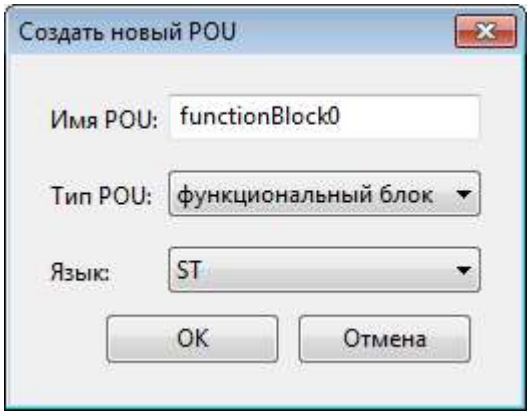
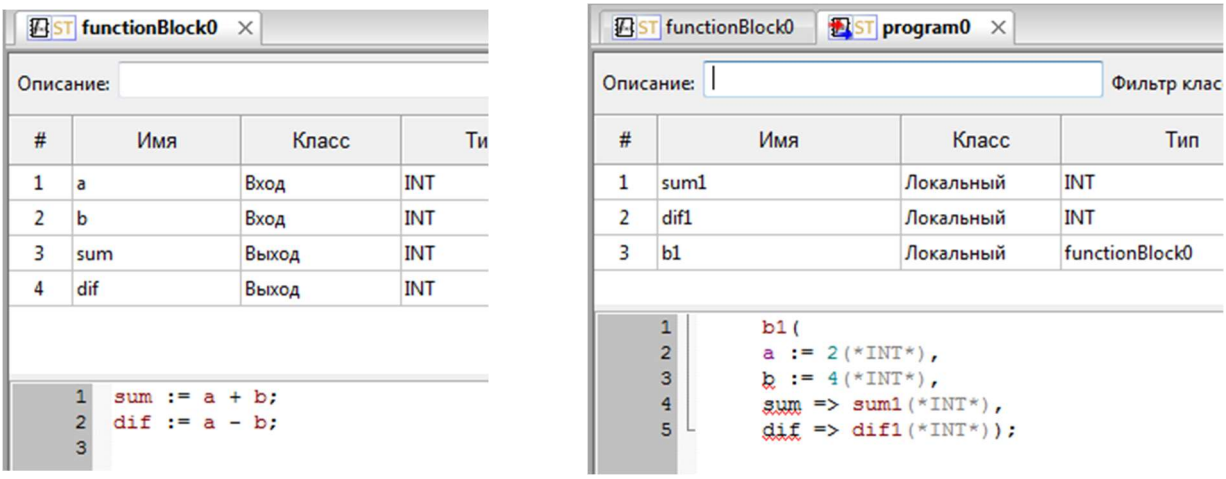


Рисунок 51 – Окно создания нового функционального блока

Панель редактирования функционального блока не отличается от панели редактирования программы. На рисунке 52 представлен пример простого функционального блока, подсчитывающего сумму и разность двух чисел, и пример его использования в программе. В функциональном блоке и программе из примера используется язык ST.



а б

Рисунок 52 – Пример создания и использования функционального блока

В среде Beremiz предусмотрено преобразование функционального блока в программу. Для этого в контекстном меню на элементе функционального блока в дереве проекта нужно выбрать пункт «Сменить тип POU на» -> «Программа» (рис. 53).

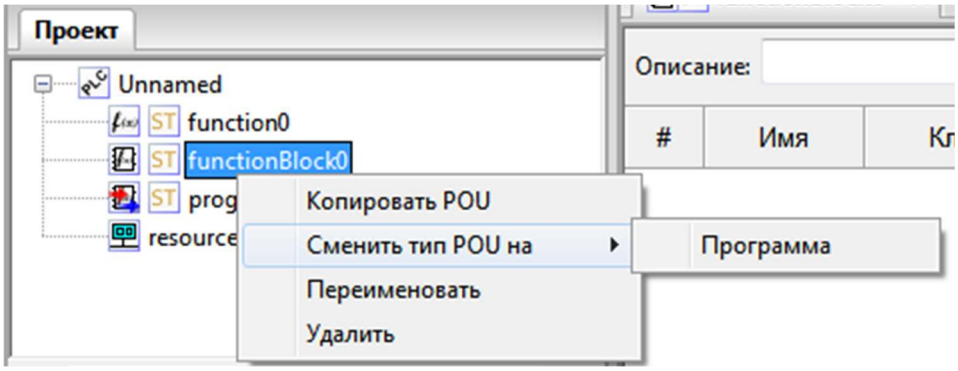


Рисунок 53 – Смена типа программного модуля функционального блока

Обратное преобразование программы в функциональный блок невозможно. Но его можно отменить, выбрав в главном меню пункт «Редактировать» -> «Отмена».

Все добавленные в проект функциональные блоки появляются на панели «Библиотеки» вместе со всеми остальными функциями и функциональными блоками в разделе «Пользовательские POU».

3.10 Использование возможностей Контроллера в проекте

Среда Beremiz обеспечивает механизм связывания внешних источников данных, например, модулей ввода-вывода (их параметров, состояний), с программными модулями (в частности с их переменными) на языках IEC 61131-3, из которых состоит прикладная программа. Данный механизм реализован в виде модулей расширения — плагинов, описывающих каждый доступный источник внешних данных.

В проект в среде Beremiz можно добавить следующие плагины:

- «MK Logic201» — плагин каналов ввода вывода Контроллера;
- «MK200 CANOpen» — плагин, реализующий работу Контроллера по протоколу CANOpen;
- «ModbusTCP Master» — плагин, реализующий работу Контроллера по протоколу Modbus TCP в режиме ведущего;
- «MK200 Modbus master request» — плагин, реализующий работу Контроллера по протоколу Modbus RTU в режиме ведущего;
- «MK211», «MK234», «MK241», «MK242», «MK243», «MK245» — плагины модулей ввода-вывода ПЛК MKLogic200 (используются в рамках плагина «MK200 CANOpen»).

Добавление и удаление плагинов осуществляется стандартным образом: выбором соответствующего плагина в контекстном меню или через меню «Редактировать» -> «Добавить». После чего плагин появляется в дереве проекта (рис. 54).

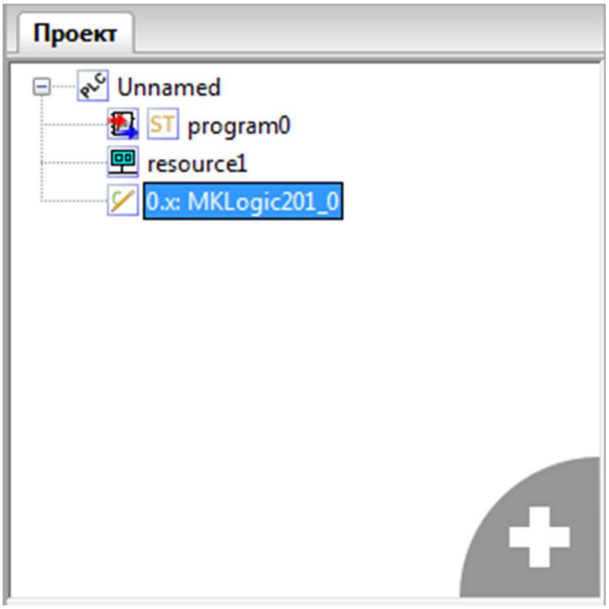


Рисунок 54 – Вид плагина каналов ввода-вывода Контроллера в дереве проекта

В начале имени плагина указывается префикс адресов его переменных — МЭК адреса переменных ввода-вывода будут начинаться с этого номера. Эти адреса используются для привязки внутренних данных плагина к переменным в программных модулях.

Панель редактирования параметров плагина открывается двойным щелчком мыши по имени плагина в дереве проектов. В верхней части этой панели (см. рис. 56) находится поле имени плагина, в котором это имя можно изменить. Слева от поля имени находится поле изменения префикса адресов данного плагина. Префикс можно изменить стрелками вверх и вниз.

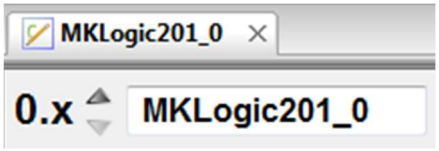


Рисунок 55 – Общая часть панели редактирования всех плагинов

В дереве проекта все плагины располагаются в порядке увеличения их адресных префиксов. При изменении префикса адреса плагина, он автоматически смещается в дереве проекта, занимая новое место в соответствие с новым номером.

3.10.1 Плагин «MKLogic 201»

Для получения доступа из программных модулей проекта к каналам ввода-вывода Контроллера в проект необходимо добавить плагин «MKLogic 201». Плагин содержит все каналы ввода-вывода Контроллера, привязываемые к переменным программы пользователя посредством МЭК адресов, и позволяет их настраивать.

Настройка каналов ввода-вывода Контроллера производится в панели редактирования «MKLogic 201» (рис. 56), вызов которой осуществляется двойным щелчком мыши по имени плагина в дереве проекта.

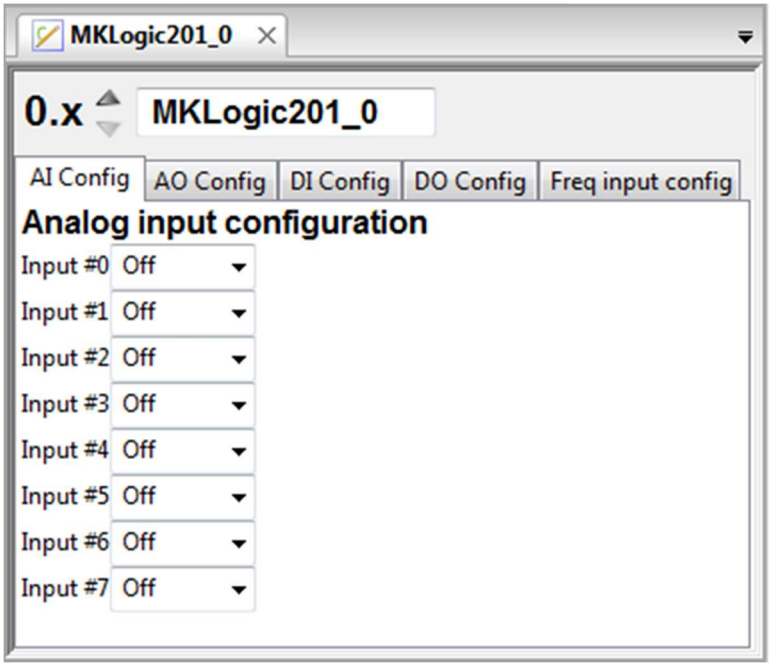


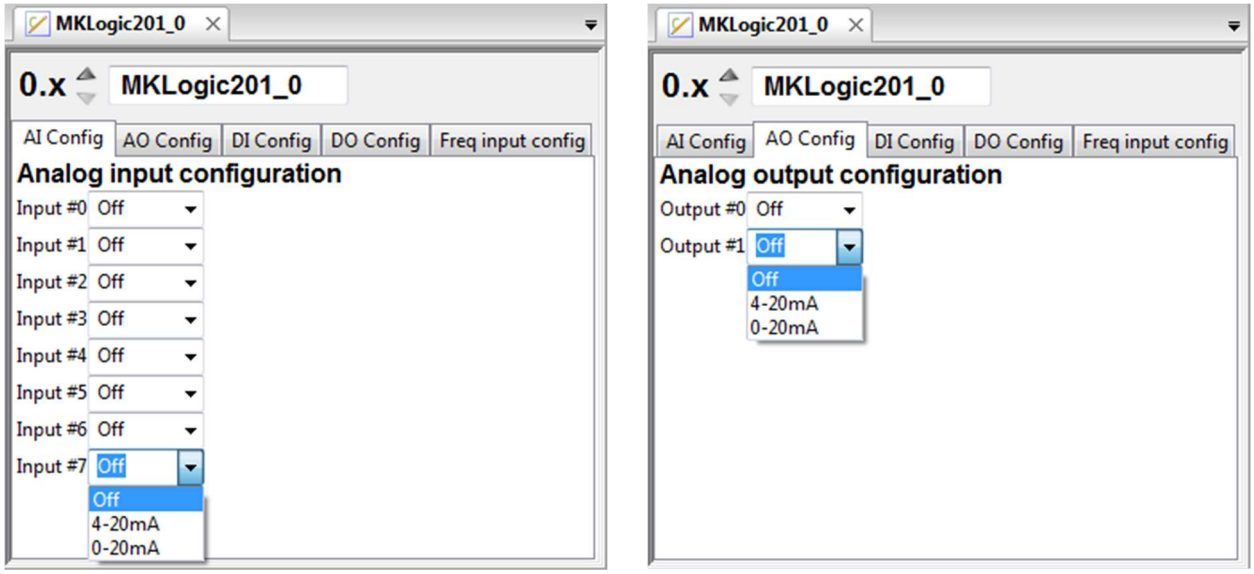
Рисунок 56 – Панель настройки каналов ввода-вывода Контроллера

Панель настроек каналов ввода-вывода Контроллера содержит вкладки для:
- аналоговых входов/выходов («AI Config», «AO Config»);
- дискретных входов/выходов («DI Config», «DO Config»);
- счетно-частотных входов («Freq input config»).

3.10.1.1 Настройка аналоговых входов-выходов Контроллера

Настройка аналоговых входов-выходов Контроллера производится во вкладках «AI Config» и «AO Config» панели настроек каналов ввода-вывода Контроллера (рис. 57). Всего в Контроллере имеется 8 аналоговых входов и 2 аналоговых выхода.

Вкладка «AI Config» (рис. 57а) содержит перечень аналоговых входов Контроллера, каждый из которых можно настроить следующим образом:
- «Off» — канал выключен;
- «4-20mA» — канал работает в диапазоне значений 4-20 мА;
- «0-20mA» — канал работает в диапазоне значений 0-20 мА.



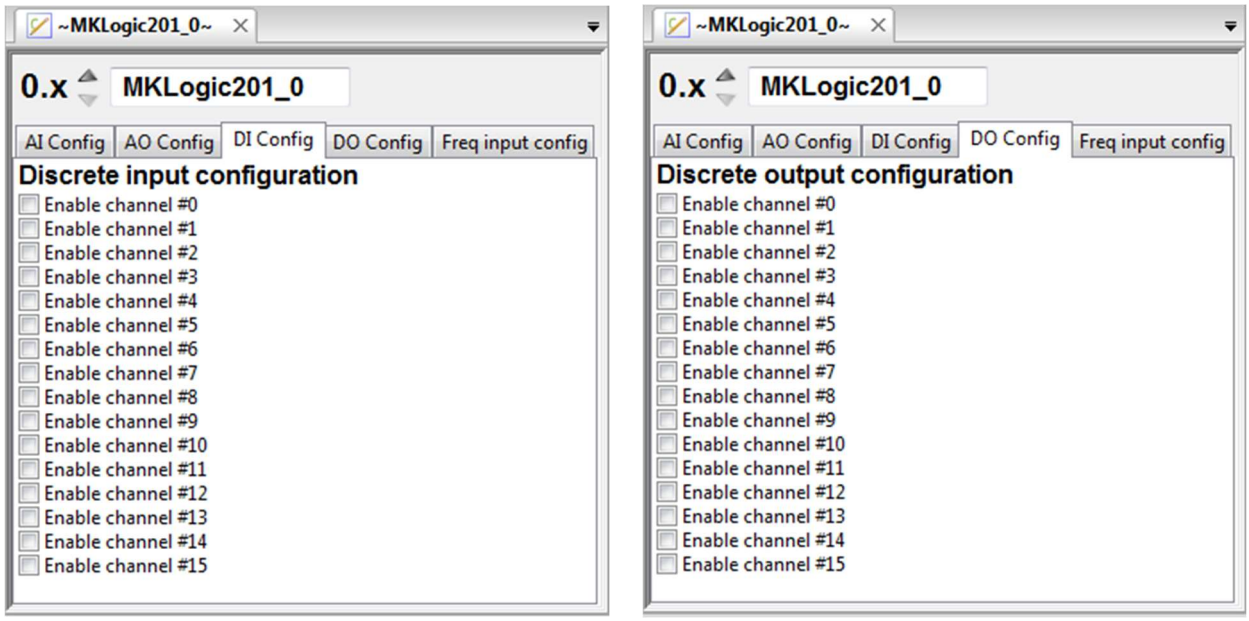
а б

Рисунок 57 – Настройка аналоговых входов-выходов Контроллера

Аналогичным образом производится настройка аналоговых выходов во вкладке «AO Config» (рис. 57б).

3.10.1.2 Настройка дискретных входов-выходов Контроллера

Настройка дискретных входов-выходов Контроллера производится во вкладках «DI Config» (рис. 58а) и «DO Config» (рис. 58б) панели настроек каналов ввода-вывода Контроллера.



а б

Рисунок 58 – Настройка дискретных входов Контроллера

Данные вкладки содержат элементы управления для включения/выключения дискретных входов-выходов. В Контроллере имеется 16 каналов дискретных входов и 16 каналов дискретных выходов. Если на соответствующем канале установлена галочка, то канал включен, иначе — выключен.

3.10.1.3 Настройка счетно-частотных входных портов целевого устройства

Настройка счетно-частотных входных портов целевого устройства производится во вкладке «Freq input config» (рис. 59).

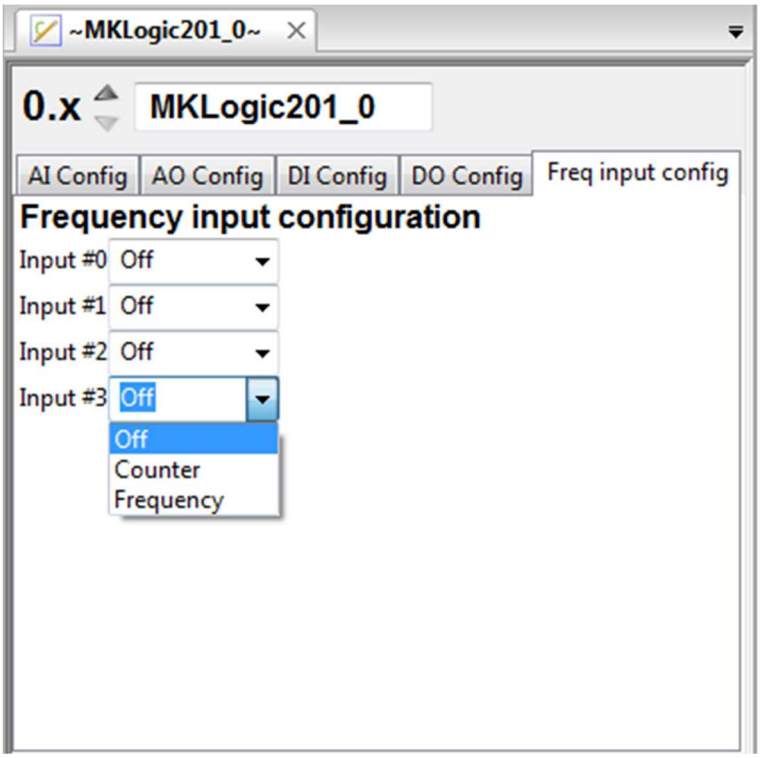


Рисунок 59 – Настройка дискретных входов Контроллера

Для каждого канала счетно-частотных входов из выпадающего списка можно выбрать режим работы. Счетно-частотные входы могут работать в следующих режимах:

- «Off» — отключен;
- «Counter» — счетчик (подсчет количества импульсов на входе);
- «Frequency» — частотомер (измерение частоты сигнала на входе).

3.10.1.4 МЭК-адреса плагина

МЭК-адреса используются для привязки внутренних данных плагина к переменным в программных модулях. Описание МЭК-адресов плагина «MKLogic201» и их типов приведено в таблице 14.

Таблица 14 – МЭК-адреса плагина «MKLogic201»

Наименование в дереве	Адрес	Описание	Тип
Analog Inputs ├ Channel #0 ├ … └ Channel #7	%QD[x].0.0 … %QD[x].0.7	Каналы аналоговых входов (с 0 по 7)	REAL
Analog Outputs ├ Channel #0 └ Channel #1	%QD[x].1.0 %QD[x].1.1	Каналы аналоговых выходов (с 0 по 7)	REAL
Frequency/Counter inputs └ Mode ├ Channel #0 ├ … └ Channel #3	%QB[x].2.0.0 … %QB[x].2.0.3	Режим работы каналов счетно-частотных входов (с 0 по 3)	BYTE
Frequency/Counter inputs └ Frequency ├ Channel #0 ├ … └ Channel #3	%QD[x].2.1.0 … %QD[x].2.1.3	Значение измеренных частот каналов счетно-частотных входов (с 0 по 3)	REAL
Frequency/Counter inputs └ Counter start ├ Channel #0 ├ … └ Channel #3	%QX[x].2.2.0 … %QX[x].2.2.3	Переменные запуска счетчика соответствующего каналы счетно-частотных входов (с 0 по 3)	BOOL
Frequency/Counter inputs └ Counter value ├ Channel #0 ├ … └ Channel #3	%QD[x].2.0.0 … %QD[x].2.0.3	Значение счетчиков каналов счетно-частотных входов (с 0 по 3)	DINT
Digital Inputs ├ Channel #0 ├ … └ Channel #15	%QX[x].3.0 … %QX[x].3.15	Каналы дискретных входов (с 0 по 15)	BOOL
Digital Outputs ├ Channel #0 ├ … └ Channel #15	%QD[x].4.0 … %QD[x].4.15	Каналы аналоговых выходов (с 0 по 15)	BOOL
RTC value ├ Milliseconds ├ Second ├ Minutes ├ Hour ├ Date ├ Month └ Year	 %QW[x].5.0 %QW[x].5.1 %QW[x].5.2 %QW[x].5.3 %QW[x].5.4 %QW[x].5.5 %QW[x].5.6	Переменные для чтения значения часов реального времени	 UINT UINT UINT UINT UINT UINT UINT

RTC set		Переменные для задания значения часов реального времени	
├ Milliseconds	%QW[x].5.0		UINT
├ Second	%QW[x].5.1		UINT
├ Minutes	%QW[x].5.2		UINT
├ Hour	%QW[x].5.3		UINT
├ Date	%QW[x].5.4		UINT
├ Month	%QW[x].5.5		UINT
└ Year	%QW[x].5.6		UINT
Diagnostics		Переменные диагностики	
└Can			
├ Last Error	%QB[x].7.0		BYTE
├ Receive Error Counter	%QB[x].7.1		BYTE
├ Transmit Error Counter	%QB[x].7.2		BYTE
├ Request MailBox0 Flag	%QX[x].7.3		BOOL
├ Request MailBox1 Flag	%QX[x].7.4		BOOL
├ Request MailBox2 Flag	%QX[x].7.5		BOOL
├ FIFO0 Message Pending Flag	%QX[x].7.6		BOOL
├ FIFO0 Full Flag	%QX[x].7.7		BOOL
├ FIFO0 Overrun Flag	%QX[x].7.8		BOOL
├ FIFO1 Message Pending Flag	%QX[x].7.9		BOOL
├ FIFO1 Full Flag	%QX[x].7.10		BOOL
├ FIFO1 Overrun Flag	%QX[x].7.11		BOOL
├ Wakeup Flag	%QX[x].7.12		BOOL
├ Sleep Acknowledge Flag	%QX[x].7.13		BOOL
├ Error Warning Flag	%QX[x].7.14		BOOL
├ Error Passive Flag	%QX[x].7.15		BOOL
├ Bus-Off Flag	%QX[x].7.16		BOOL
└ Last Error Code Flag	%QX[x].7.17		BOOL

Подробнее о привязке внутренних данных плагина к переменным программ см. в разделе 3.10.6.

3.10.2 Плагин «MK200 CANOpen»

Плагин «MK200 CANOpen» предназначен для работы с подключенными к Контроллеру внешними модулями ввода-вывода ПЛК MKLogic200 по протоколу CANOpen. В текущей версии можно подключить следующие модули:

- MK211 — 8 аналоговых входов, 24 дискретных входа;
- MK234 — 8 аналоговых входов, 2 аналогового выхода;
- MK241 — 32 дискретных входа;
- MK242 — 32 дискретных выхода;
- MK243 — 16 дискретных входов, 8 дискретных выходов;
- MK245 — 8 счетно-частотных входов.

На панели редактирования плагина «MK200 CANOpen» (рис. 60) можно ввести следующие параметры:

- «Node ID» — адрес CANOpen ведущего устройства (ведущим устройством является Контроллер), значение по умолчанию 127;
- «Heartbeat Time (ms)» — период испускания heartbeat пакетов ведущим устройством в миллисекундах, значение по умолчанию 500;
- «Baud (b/s)» — скорость шины CAN в килобитах в секунду, значение по умолчанию 1000;
- «Data update time (ms)» — период испускания PDO пакетов ведомыми устройствами, значение по умолчанию 500.

Примечание: Каждый модуль должен быть заранее настроен для работы в режиме CAN-интерфейса со скоростью указанной в поле «Baud (b/s)».

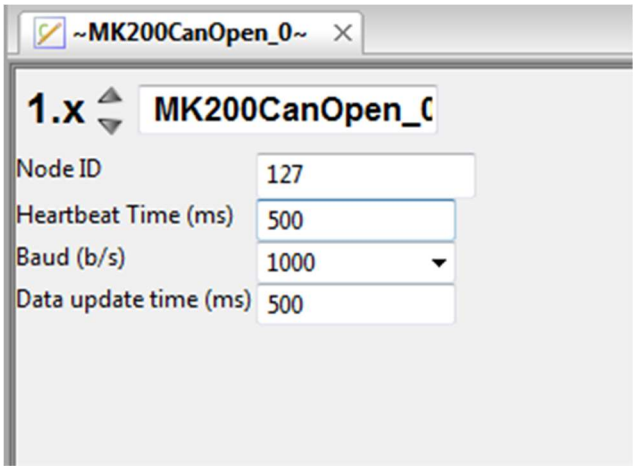
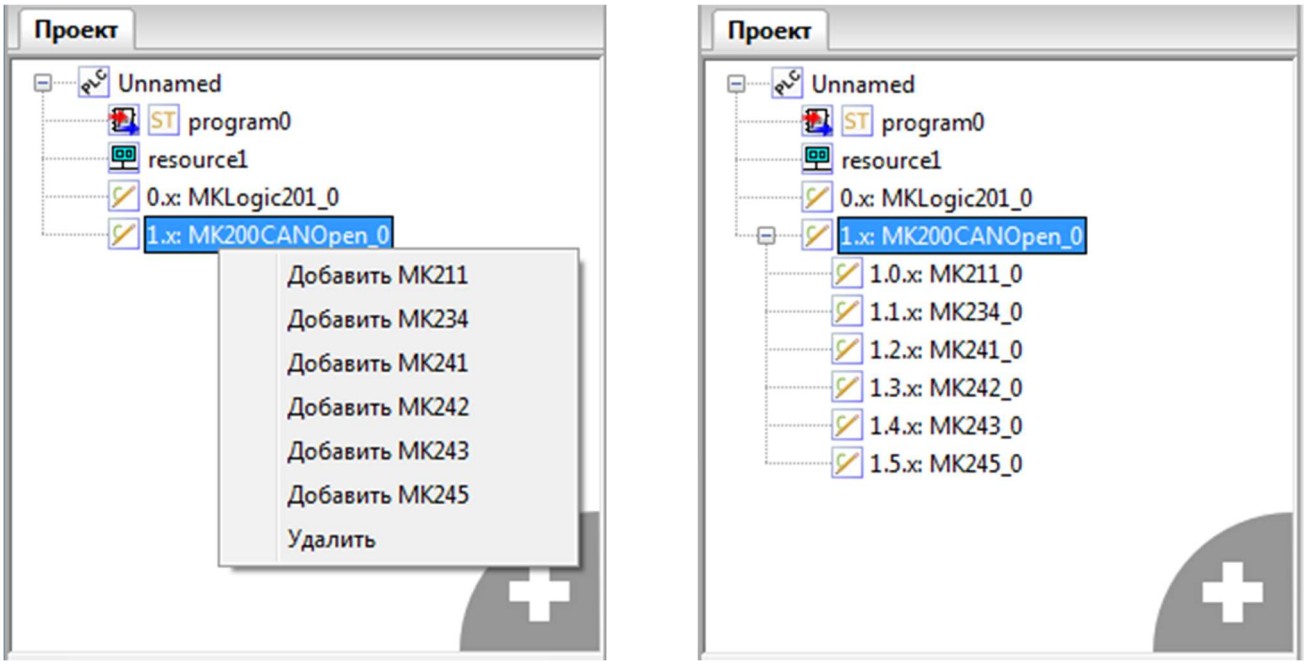


Рисунок 60 – Панель настройки параметров плагина «MK200 CANOpen»

Добавление модулей ввода-вывода в проект осуществляется из контекстного меню, которое появляется при щелчке правой кнопкой мыши по элементу плагина «MK200 CANOpen» в дереве проекта (рис. 61а).



а б

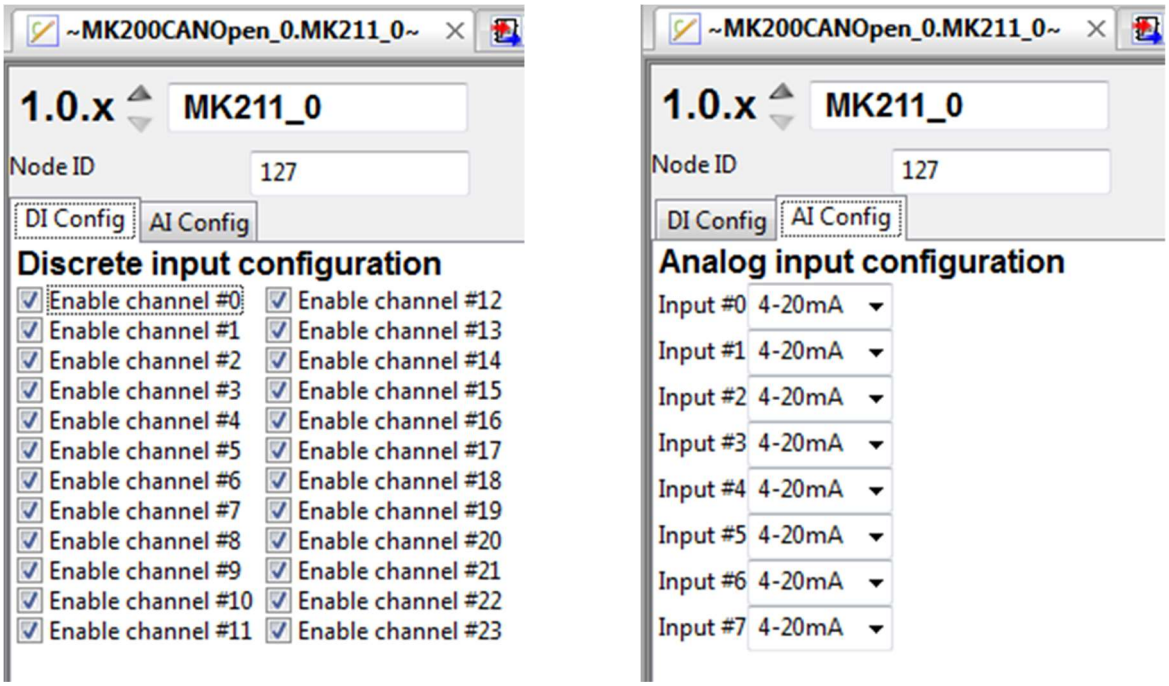
Рисунок 61 – Добавление модулей ввода вывода в проект

После добавления необходимых модулей, в дереве проекта они будут сгруппированы внутри плагина «MK200 CANOpen» (см. рис. 61б). По двойному щелчку по соответствующему модулю открывается панель его редактирования. Данная панель для всех модулей содержит следующие поля: имя плагина, префикс его МЭК адресов и «Node ID» — адрес устройства по протоколу CANOpen. Кроме того, на панели редактирования модулей расположены вкладки настройки их каналов ввода-вывода.

Примечание: Каждый модуль должен иметь уникальный адрес. В панели редактирования для каждого внесенного в проект модуля, нужно указать заранее настроенный адрес.

Примечание: Адрес модуля и скорость работы его CAN-интерфейса задается в отдельной программе «Конфигуратор MKLogic200» при непосредственном подключении модуля к ПК. Подробное описание процедуры конфигурирования можно найти в документе 643.00137093.0516.29-01 34 «Контроллер программируемый логический MKLogic200. Программное обеспечение для настройки модулей «Конфигуратор MKLogic200». Руководство пользователя»

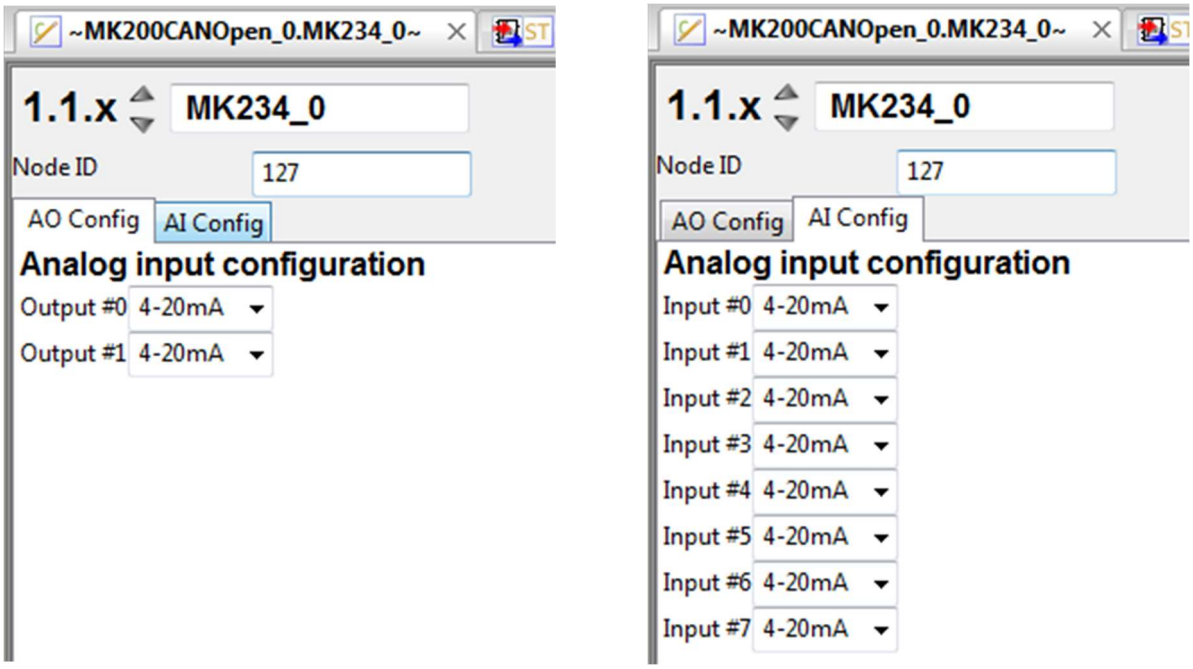
Панель редактирования модуля МК211 содержит вкладки конфигурирования каналов дискретных (рис. 62а) и аналоговых (рис. 62б) входов. Их настройка производится аналогично каналам ввода-вывода плагина «MKLogic 201».



а б

Рисунок 62 – Панель редактирования модуля МК211

Панель редактирования модуля МК234 содержит вкладки настройки аналоговых выходов (рис. 63а) и аналоговых входов (рис. 63б).



а б

Рисунок 63 – Панель редактирования модуля MK234

Панель редактирования модуля MK241 содержит вкладки конфигурирования дискретных входов (рис. 64).

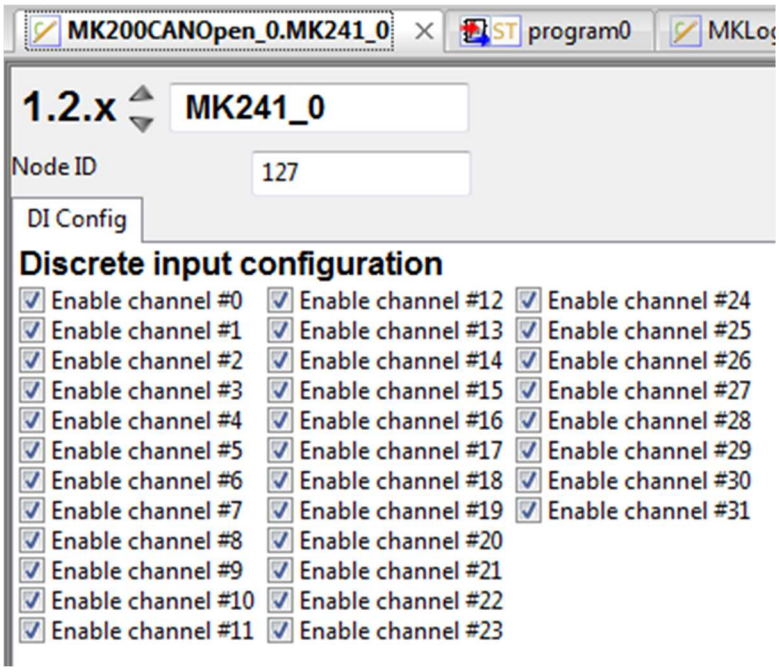


Рисунок 64 – Панель редактирования модуля MK241

Панель редактирования модуля MK242 содержит вкладки настройки дискретных выходов (рис. 65).

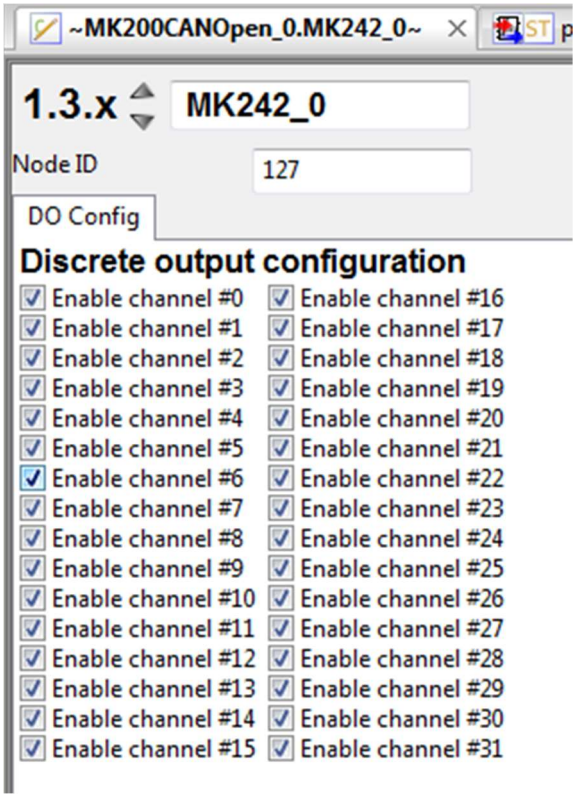
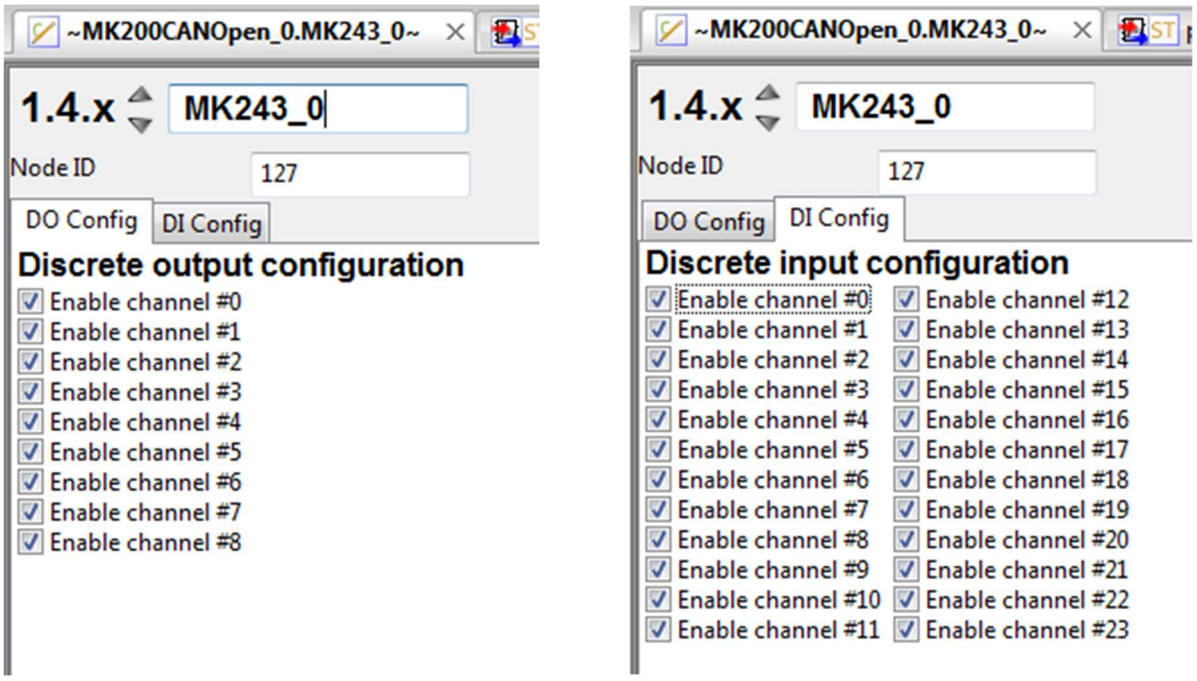


Рисунок 65 – Панель редактирования модуля МК242

Панель редактирования модуля МК243 содержит вкладки настройки дискретных выходов (рис. 66а) и дискретных входов (рис. 66б).



а б

Рисунок 66 – Панель редактирования модуля МК243

Панель редактирования модуля МК245 содержит вкладки настройки счетно-частотных входов (рис. 65).

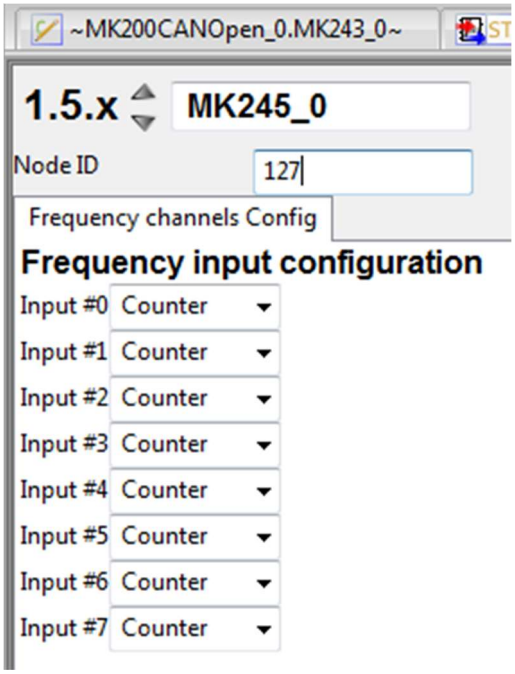


Рисунок 67 – Панель редактирования модуля МК245

3.10.2.1 МЭК-адреса плагинов модулей ввода-вывода

МЭК-адреса используются для привязки внутренних данных плагина к переменным в программных модулях. В таблице 15 приведены МЭК-адреса модуля МК211 и их формат.

Таблица 15 – МЭК-адреса модуля МК211 плагина «МК200 CANOpen»

Наименование в дереве	Адрес	Описание	Тип
Analog inputs (Float) ├ Channel #0 ├ … └ Channel #7	%QD[x].[y].0.0 … %QD[x].[y].0.7	Каналы аналоговых входов (с 0 по 7); значение тока в мА в формате с плавающей точкой	REAL
Analog inputs (U16) ├ Channel #0 ├ … └ Channel #7	%QD[x].[y].1.0 … %QD[x].[y].1.7	Каналы аналоговых входов (с 0 по 7); коды АЦП	DINT
Digital Inputs ├ Channel #0 ├ … └ Channel #23	%QX[x].[y].2.0 … %QX[x].[y].2.23	Каналы дискретных входов (с 0 по 23)	BOOL
Diagnostic ├ Connected └ Error	%QX[x].[y].3.0 %QD[x].[y].3.1	Диагностические переменные Connected — статус соединения с модулем; Error — код ошибки связи с модулем.	BOOL DINT

В таблице 16 приведены МЭК-адреса модуля МК234 и их формат.

Таблица 16 – МЭК-адреса модуля МК234 плагина «МК200 CANOpen»

Наименование в дереве	Адрес	Описание	Тип
Analog inputs (Float) ├ Channel #0 ├ … └ Channel #7	%QD[x].[y].0.0 … %QD[x].[y].0.7	Каналы аналоговых входов (с 0 по 7); значение тока в мА в формате с плавающей точкой	REAL
Analog inputs (U16) ├ Channel #0 ├ … └ Channel #7	%QD[x].[y].1.0 … %QD[x].[y].1.7	Каналы аналоговых входов (с 0 по 7); коды АЦП	DINT
Analog Outputs ├ Channel #0 └ Channel #1	%QD[x].[y].2.0 %QD[x].[y].2.1	Каналы аналоговых выходов (с 0 по 1)	REAL
Diagnostic ├ Connected └ Error	 %QX[x].[y].3.0 %QD[x].[y].3.1	Диагностические переменные Connected — статус соединения с модулем; Error — код ошибки связи с модулем.	 BOOL DINT

В таблице 17 приведены МЭК-адреса модуля МК241 и их формат.

Таблица 17 – МЭК-адреса модуля МК241 плагина «МК200 CANOpen»

Наименование в дереве	Адрес	Описание	Тип
Digital Inputs ├ Channel #0 ├ … └ Channel #31	%QX[x].[y].0.0 … %QX[x].[y].0.31	Каналы дискретных входов (с 0 по 31)	BOOL
Diagnostics ├ Connected └ Error	 %QX[x].[y].2.0 %QD[x].[y].2.1	Диагностические переменные Connected — статус соединения с модулем; Error — код ошибки связи с модулем.	 BOOL DINT

В таблице 18 приведены МЭК-адреса модуля МК242 и их формат.

Таблица 18 – МЭК-адреса модуля МК242 плагина «МК200 CANOpen»

Наименование в дереве	Адрес	Описание	Тип
Digital Outputs ├ Channel #0 ├ … └ Channel #31	%QX[x].[y].0.0 … %QX[x].[y].0.31	Каналы дискретных выходов (с 0 по 31)	BOOL

Diagnostics ├ Connected └ Error	 %QX[x].[y].2.0 %QD[x].[y].2.1	Диагностические переменные Connected — статус соединения с модулем; Error — код ошибки связи с модулем.	 BOOL DINT

В таблице 19 приведены МЭК-адреса модуля МК243 и их формат.

Таблица 19 – МЭК-адреса модуля МК243 плагина «МК200 CANOpen»

Наименование в дереве	Адрес	Описание	Тип
Digital Inputs ├ Channel #0 ├ … └ Channel #15	%QX[x].[y].0.0 … %QX[x].[y].0.15	Каналы дискретных входов (с 0 по 15)	BOOL
Digital Outputs ├ Channel #0 ├ … └ Channel #7	%QX[x].[y].1.0 … %QX[x].[y].1.7	Каналы дискретных выходов (с 0 по 7)	BOOL
Diagnostics ├ Connected └ Error	 %QX[x].[y].2.0 %QD[x].[y].2.1	Диагностические переменные Connected — статус соединения с модулем; Error — код ошибки связи с модулем.	 BOOL DINT

В таблице 20 приведены МЭК-адреса модуля МК245 и их формат.

Таблица 20 –МЭК-адреса модуля МК245 плагина «МК200 CANOpen»

Наименование в дереве	Адрес	Описание	Тип
Frequency/Counter inputs └ Mode ├ Channel #0 ├ … └ Channel #7	%QD[x].[y].0.0 … %QD[x].[y].0.7	Режим работы каналов счетно-частотных входов (с 0 по 7)	DINT
Frequency/Counter inputs └ Counter value ├ Channel #0 ├ … └ Channel #7	%QD[x].[y].1.0 … %QD[x].[y].1.7	Значение измеренных счетчика каналов счетно-частотных входов (с 0 по 7)	DINT
Frequency/Counter inputs └ Frequency value ├ Channel #0 ├ … └ Channel #7	%QD[x].[y].2.0 … %QD[x].[y].2.23	Значение измеренных частот каналов счетно-частотных входов (с 0 по 7)	REAL

Diagnostics ├ Connected └ Error	%QX[x].[y].3.0 %QD[x].[y].3.1	Диагностические переменные Connected — статус соединения с модулем; Error — код ошибки связи с модулем.	BOOL DINT

Подробнее о привязке внутренних данных плагина к переменным программ см. в разделе 3.10.6.

3.10.3 Плагин «ModbusTCP master»

Для создания запросов к внешним устройствам по протоколу ModbusTCP используется плагин «ModbusTCP master». Данный плагин добавляется в проект аналогично описанным выше плагинам.

Панель редактирования плагина «ModbusTCP master» содержит стандартные элементы: имя плагина и префикс его МЭК адресов (рис. 68).

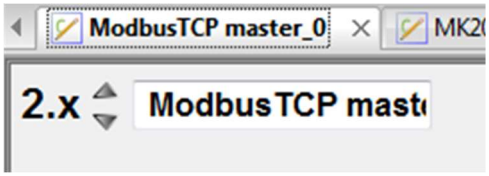
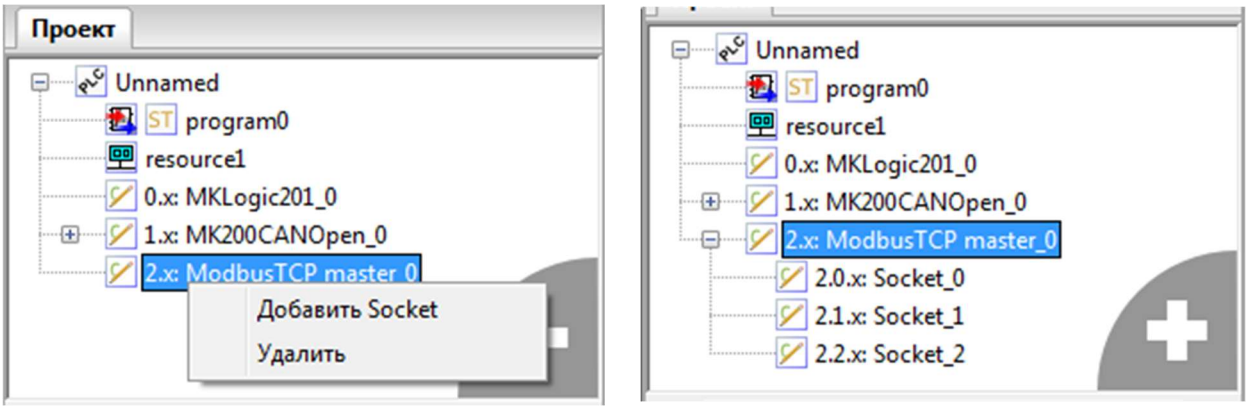


Рисунок 68 – Панель редактирования плагина «ModbusTCP master»

Запросы в данном плагине добавляются в рамках одного сокета. Для того чтобы добавить сокет в плагин нужно выбрать «Добавить Socket» в контекстном меню доступном по щелчку правой кнопкой мышки по элементу плагина «ModbusTCP master» в дереве проекта (рис. 69а).



а б

Рисунок 69 – Контекстное меню добавления сокета в плагин «ModbusTCP master»

После добавления необходимых сокетов, в дереве проекта они будут сгруппированы внутри плагина «ModbusTCP master» (см. рис. 69б). По двойному щелчку на соответствующем сокете открывается панель его редактирования. Данная панель для всех модулей содержит обязательные поля — имя плагина сокета с префиксом его МЭК адресов, а также специальные элементы:

– «Ip address» — IP-адрес ведомого устройства (сервера);

- «Port» — номер TCP-порта соединения (стандартный порт для протокола Modbus TCP — 502, но его можно изменить на усмотрения проектировщика);
- таблица запросов.

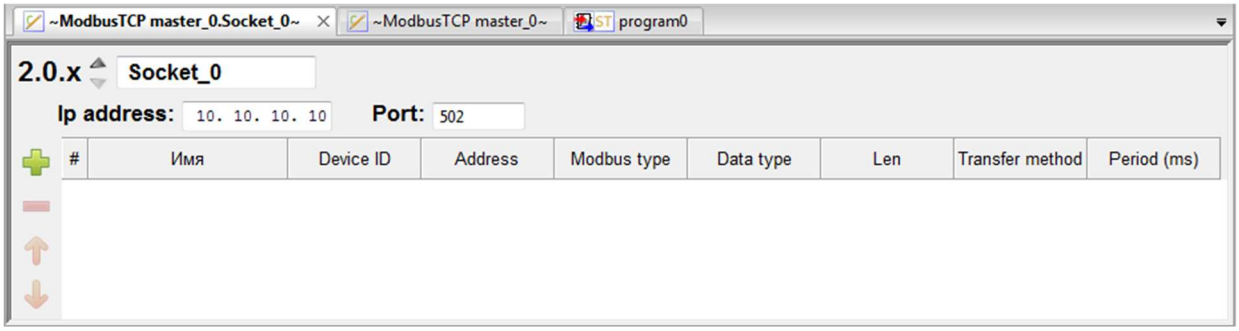


Рисунок 70 – Панель редактирования плагина «ModbusTCP master»

В таблице запросов задаются Modbus запросы ведомому устройству. Описание столбцов таблицы приведено в таблице 21.

Таблица 21 – Назначение столбцов таблицы запросов плагина «ModbusTCP master»

Столбец	**Назначение**
«#»	Номер запроса
«Имя»	Уникальное имя запроса
«Device ID»	Modbus адрес ведомого устройства
«Address»	Адрес начальных данных Modbus (в зависимости от типа запроса: регистра или битового поля)
«Modbus type»	Тип запроса протокола Modbus: «Holding read» — Чтение регистров «Holding registers»; «Holding write» — Запись регистров «Holding registers»; «Input» — Чтение регистров «Input registers»; «Coil read» — Чтение битовых полей типа «Coils»; «Coil write» — Запись битовых полей типа «Coils»; «Disc» — Чтение битовых полей типа «Discretes Input».
«Data type»	Тип данных. В данной версии среды доступен только тип WORD.
«Len»	Количество запрашиваемых элементов данных (в зависимости от типа запроса: регистров или битовых полей)
«Transfer method»	Определяет тип запуска исполнения запроса: «Periodic» — периодический запуск; «One shot» — однократный запрос (запуск осуществляется присвоением переменной «run» запроса значения TRUE, подробнее см. ниже).
«Period (ms)»	Период запуска исполнения запроса, если выбран «Transfer method» = «Periodic».

По имени запроса в проекте создается глобальная переменная с типом соответствующим запросу: тип «WORD» — для запросов длиной 1 и массив типа «ARRAY [0..n] OF WORD», где n = «Len» – 1 (значение столбца «Len» таблицы запросов минус 1). В связи с этим имя запроса должно быть уникальным относительно всего проекта, а не только в рамках одного сокета.

На рисунке 71 представлен пример создания запросов в плагине «ModbusTCP master». В данном примере созданы три запроса: request0, request1 и request2. Первые два на чтение, третий на запись holding регистров

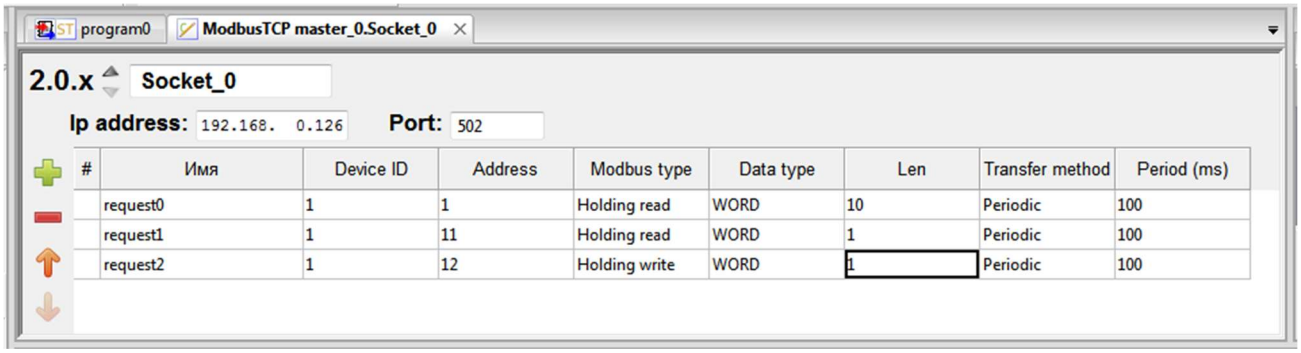


Рисунок 71 – Пример создания запросов в плагине «ModbusTCP master»

Для использования данных запросов нужно в программе объявить переменные с именем, совпадающим с именем запроса, указав класс переменной «Внешний» (рис. 72).

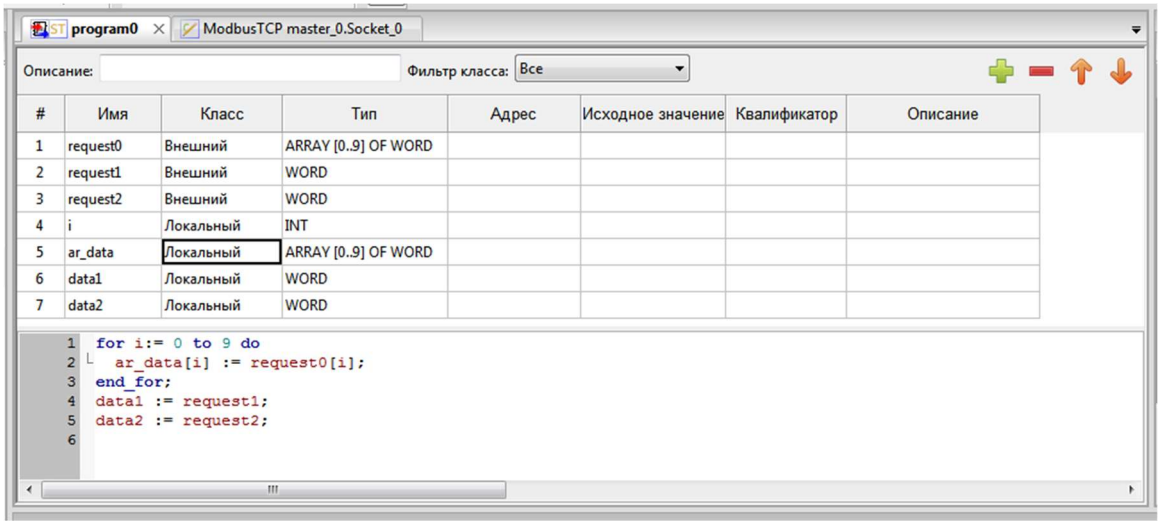


Рисунок 72 – Пример использования запросов плагина «ModbusTCP master»

Запросы, у которых параметр «Transfer method» выставлен в значение «Periodic», будут отправляться с периодичностью равной параметру «Period (ms)». В случае, если это запросы на чтение («Holding read», «Input», «Coil read», «Disc»), считанные в этих запросах данные будут синхронизироваться с соответствующими им переменными перед каждой итерацией задачи пользователя. В случае, если это запросы на запись («Holding write», «Coil write»), данные из переменных будут синхронизироваться с данными в запросе после каждой итерации задачи пользователя.

В таблице 22 приведено описание МЭК адресов плагина «ModbusTCP master». В таблице используются следующие обозначения:

- [x] — префикс адресов плагина, задается в панели редактирования плагина;
- [s] — префикс адреса сокета, задается в панели редактирования сокета;
- [r] — номер запроса, задается автоматически порядком расположения запросов в таблице запросов;
- <Socket name> — имя сокета в плагине «ModbusTCP master»;
- <Request name> — имя запроса, в сокете с именем <Socket name>.

Таблица 22 – МЭК-адреса плагина «ModbusTCP master»

Наименование в дереве	Адрес	Описание	Тип
<Socket name> └ <Request name> ├ run ├ status └ error	 %QX[x].[s].[r].0 %QW[x].[s].[r].1 %QW[x].[s].[r].2	Диагностические переменные запроса run — запуск исполнения запроса; status — статус запроса; error — код ошибки при неудачном запросе.	 BOOL WORD WORD
<Socket name> └ Connection Status	 %QW[x].[s].0	Статус подключения сокета	WORD

Расшифровка значений переменной «status» (см. табл. 22) приведена в таблице 23.

Таблица 23 – Коды статуса работы по протоколу Modbus (TCP/RTU) в режиме ведущего (клиента)

Код статуса	Описание
0	Запрос выполнен без ошибок
1	Запрос выполняется в данный момент
2	Запрос выполнен с ошибкой, код ошибки в переменной error

Расшифровка значений переменной «error» (см. табл. 22) приведена в таблице 24.

Таблица 24 – Коды ошибок работы по протоколу Modbus (TCP/RTU) в режиме ведущего

Код ошибки	Описание
0	Нет шибки
1	Неверная функция
2	Неверный адрес данных
3	Неверное значение данных
4	Общий сбой устройства сервера
5	Ведомое устройство приняло запрос и обрабатывает его, но это требует много времени. Этот ответ предохраняет ведущее устройство от генерации ошибки тайм-аута.
6	Устройство сервера занято
9	Неопределенная ошибка
12	Контрольная сумма не совпала
13	Данные ответа не соответствует запросу (число байт в ответе не соответствует ожидаемому)
14	Неизвестный код ошибки
15	Код ошибки не соответствует запросу
16	При правильной CRC данных в ответе больше, чем 125 для регистров и 2000 для битов
17	Ответ от сервера (ведомого) с другим адресом
20	Таймаут запроса
21	Неправильно заполнены поля структуры запроса
22	Не удалось инициализовать системную библиотеку Modbus
23	Не удалось соединиться (по ModbusTCP) с сервером

24	Передано/принято данных меньше, чем следовало
25	Адрес/длина принимаемых/передаваемых данных выходят за пределы связанных с Modbus массивами переменных
254	Запрос обрабатывается
255	Запрос ещё ни разу не обрабатывался

Расшифровка значений переменной «Connection Status» (см. табл. 22) приведена в таблице 25.

Таблица 25 – Коды статуса подключения сокета по протоколу ModbusTCP в режиме ведущего (клиента)

Код статуса	Описание
0	Подключен
1	Не подключен
2	Ошибка выделения памяти для создания сокета
3	Таймаут при попытке подключения

Подробнее о привязке внутренних данных плагина с переменными программ см. в разделе 3.10.6.

3.10.4 Плагин «MK200 Modbus master request»

Для задания запросов к внешним устройствам по протоколу Modbus RTU используется плагин «MK200 Modbus master request». В проект данный плагин добавляется аналогично другим плагинам.

Панель редактирования плагина «ModbusTCP master» содержит следующие элементы: имя плагина, префикс его МЭК адресов, поле выбора логического ведущего устройства, а также таблицу запросов (рис. 73).

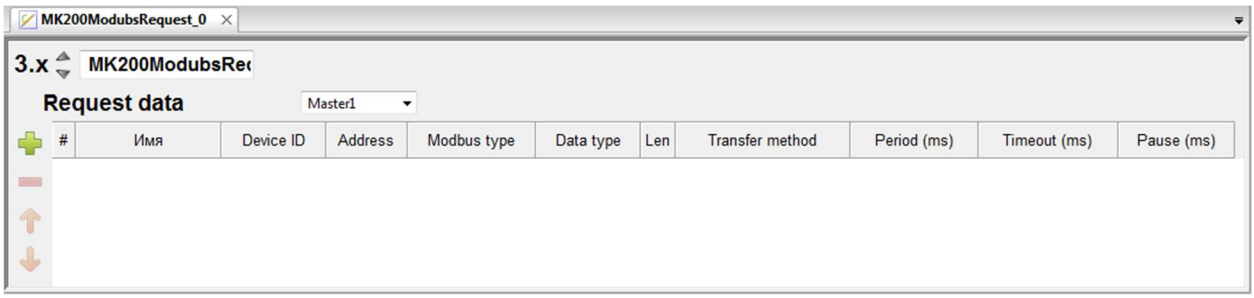


Рисунок 73 – Панель редактирования плагина «MK200 Modbus master request»

В Контроллере реализовано три логических ведущих Modbus RTU устройства с условными именами: «Master 0», «Master 1» и «Master 2». В проект можно добавить три плагина «MK200 Modbus master request», каждый из которых должен быть привязан к одному логическому ведущему устройству Контроллера. Логическое ведущее устройство, в свою очередь, привязывается к физическому порту RS-485 (см. раздел 3.12.3).

Взаимосвязи плагинов, логических ведущих устройств и физических портов показаны на рис. 74.

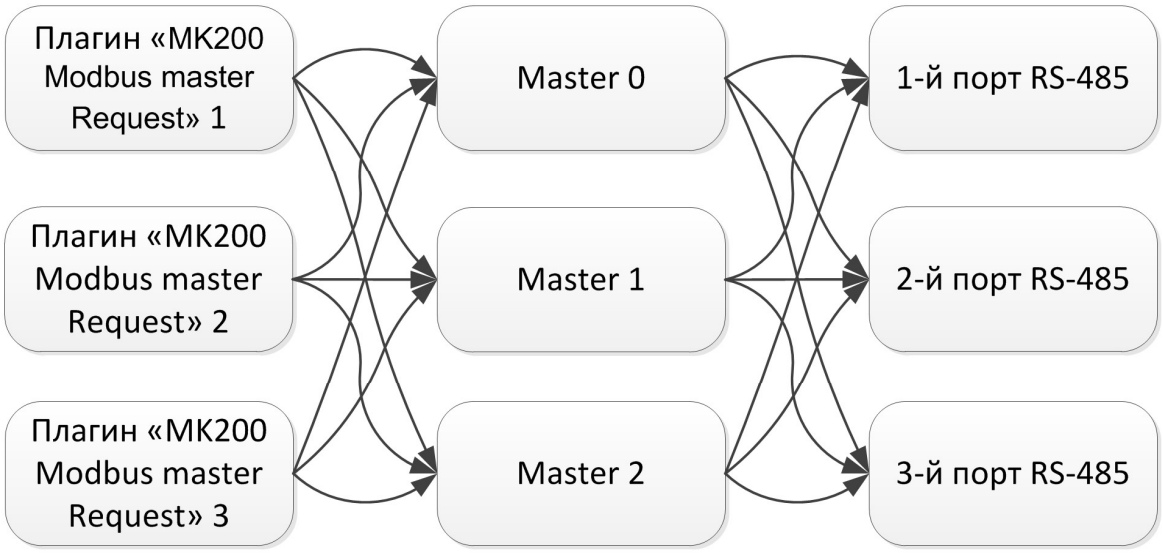


Рисунок 74 – Взаимосвязи плагина «MK200 Modbus master request» с логическими ведущими устройствами и физическими портами RS-485

Примечание: Недопустимо, чтобы несколько плагинов было привязано к одному логическому ведущему устройству. Так же недопустимо, чтобы одно логическое ведущее устройство было привязано к нескольким физическим портам. Такие варианты привязки могут иметь непредсказуемые результаты.

На рисунке приведен пример корректной привязки трех плагинов в проекте к логическим ведущим устройствам и физическим портам.

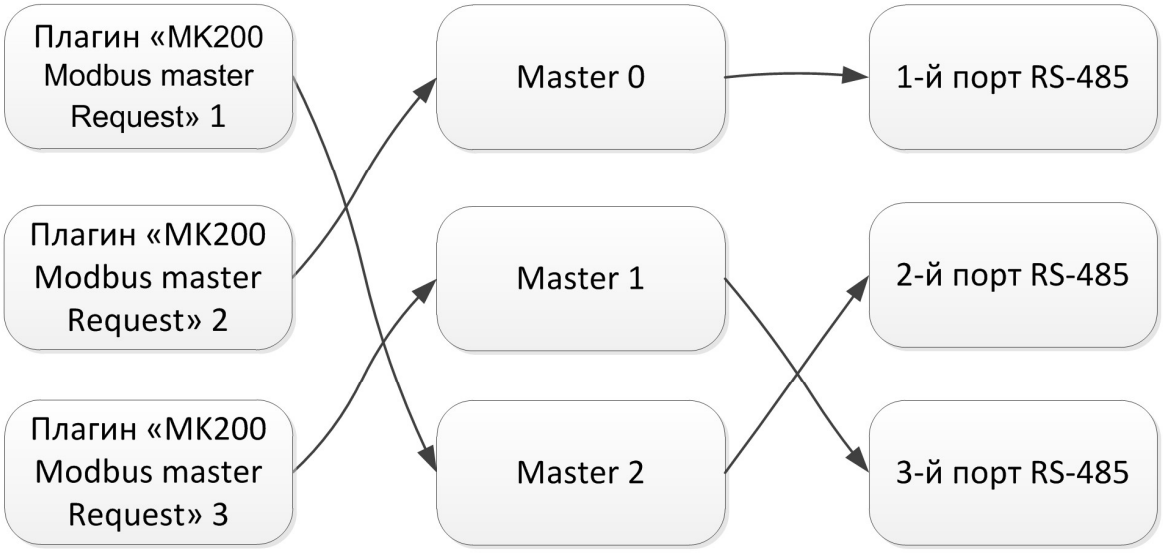


Рисунок 75 – Пример привязки плагина «MK200 Modbus master request» к логическими ведущими устройствами и физическими портами RS-485

В таблице запросов задаются Modbus запросы ведомому устройству. Описание столбцов таблицы приведено в таблице 21.

Таблица 26 – Назначение столбцов таблицы запросов плагина «ModbusTCP master»

Столбец	**Назначение**
«#»	Номер запроса
«Имя»	Уникальное имя запроса
«Device ID»	Modbus адрес ведомого устройства
«Address»	Адрес начальных данных Modbus (в зависимости от типа запроса: регистра или битового поля)
«Modbus type»	Тип запроса протокола Modbus: «Holding read» — Чтение регистров «Holding registers»; «Holding write» — Запись регистров «Holding registers»; «Input» — Чтение регистров «Input registers»; «Coil read» — Чтение битовых полей типа «Coils»; «Coil write» — Запись битовых полей типа «Coils»; «Disc» — Чтение битовых полей типа «Discretes Input».
«Data type»	Тип данных. В данной версии среды доступен только тип WORD.
«Len»	Количество запрашиваемых элементов данных (в зависимости от типа запроса: регистров или битовых полей)
«Transfer method»	Определяет тип запуска исполнения запроса: «Periodic» — периодический запуск; «One shot» — однократный запрос (запуск осуществляется присвоением переменной «run» запроса значения TRUE, подробнее см. ниже).
«Period (ms)»	Период запуска исполнения запроса в миллисекундах, если выбран «Transfer method» = «Periodic».
«Timeout (ms)»	Время ожидания ответа от ведомого устройства в миллисекундах
«Pause (ms)»	Задержка перед выполнением запроса в миллисекундах (может потребовать при работе с устройствами

В таблице 27 приведено описание МЭК адресов плагина «MK200 Modbus master request». В таблице используются следующие обозначения:

- [x] — префикс адресов плагина, задается в панели редактирования плагина;
- [r] — номер запроса, задается автоматически порядком расположения запросов в таблице запросов;
- <Request name> — имя запроса в таблице запросов.

Таблица 27 – МЭК-адреса плагина «MK200 Modbus master request»

Наименование в дереве	Адрес	Описание	Тип
<Request name> ├ run ├ status └ error	 %QX[x].[r].0 %QW[x].[r].1 %QW[x].[r].2	Диагностические переменные запроса run — запуск исполнения запроса; status — статус запроса; error — код ошибки при неудачном запросе.	 BOOL WORD WORD

Переменные «status» и «error» (см. табл. 27) имеют такие же значения как в плагине «ModbusTCP master», расшифровка их значений приведены в таблицах 23 и 24 соответственно.

Подробнее о привязке внутренних данных плагина с переменными программ см. в разделе 3.10.6.

3.10.5 Особенности реализации протокола Modbus (TCP/RTU) в режиме ведомого

При работе Контроллера по протоколу Modbus в режиме ведомого поддерживаются все стандартные типы данных (holding registers, input registers, coils и discrete inputs), а также функции чтения идентификационной информации. Режим ведомого устройства (сервера) по протоколу Modbus доступен по всем четырем физическим каналам: порт Ethernet и три порта RS-485.

Важной особенностью работы по протоколу Modbus в режиме ведомого устройства (сервера) является общая карта памяти всех данных Modbus для всех четырех каналов. При этом зарезервирована память сразу под все возможные адреса данных протокола.

На рис. 76 представлена общая схема организации работы по протоколу Modbus в Контроллера. Большинство данных протокола Modbus хранятся в ОЗУ и при инициализации контроллера равны 0. Исключением является holding registers с адресами с 40000 по 65535 — эти данные хранятся в энергонезависимой памяти Контроллера и при инициализации восстанавливают последние записанные значения.

Наполнение пользовательским содержимым карты памяти Modbus производится в программе пользователя при помощи функций расширений Modbus в библиотеке функций и функциональных блоков (подробнее см. раздел 3.10.8.2).

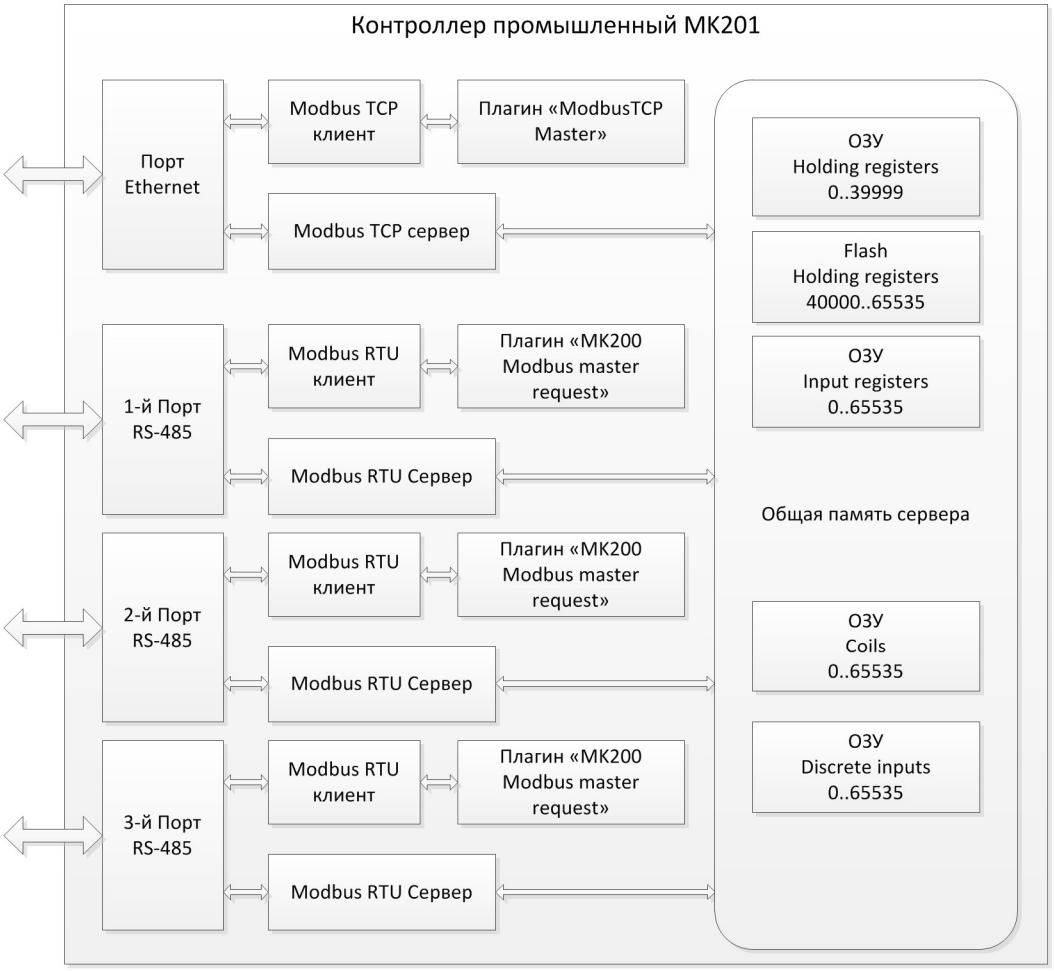


Рисунок 76 – Организация работы по протоколу Modbus в КОНТРОЛЛЕР

3.10.6 Привязка каналов ввода-вывода Контроллера к переменным в программе пользователя

Привязка каналов ввода-вывода Контроллера к переменным осуществляется после того, как в проект был добавлен и настроен хотя бы один из описанных выше плагинов. Привязка осуществляется из панели редактирования программного модуля.

Для привязки каналов необходимо в панели переменных и констант программного модуля выбрать нужную строку с переменной и двойным щелчком мыши по полю Адрес этой переменной перевести ее в режим редактирования (рис. 78).

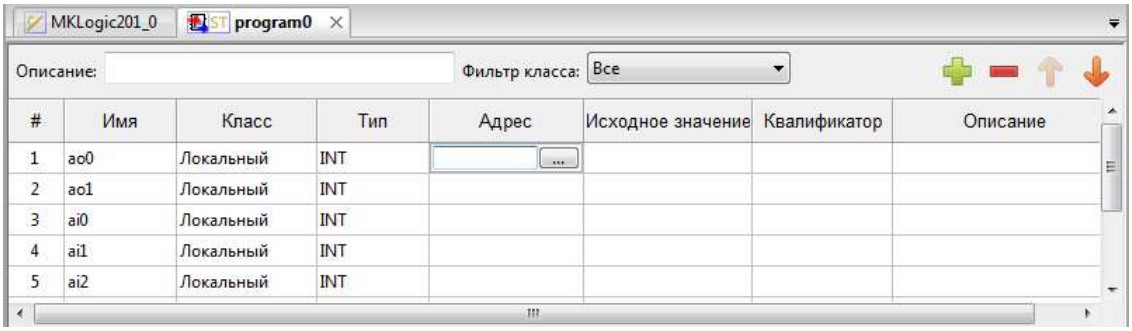


Рисунок 77 – Выбор переменной для привязки каналов ввода вывода к переменным программного модуля

Далее необходимо либо ввести адрес канала ввода-вывода вручную, либо, нажав появившуюся в поле «Адрес» кнопку [...], выбрать канал из списка в окне «Просмотр доступных МЭК-адресов» (рис. 78).

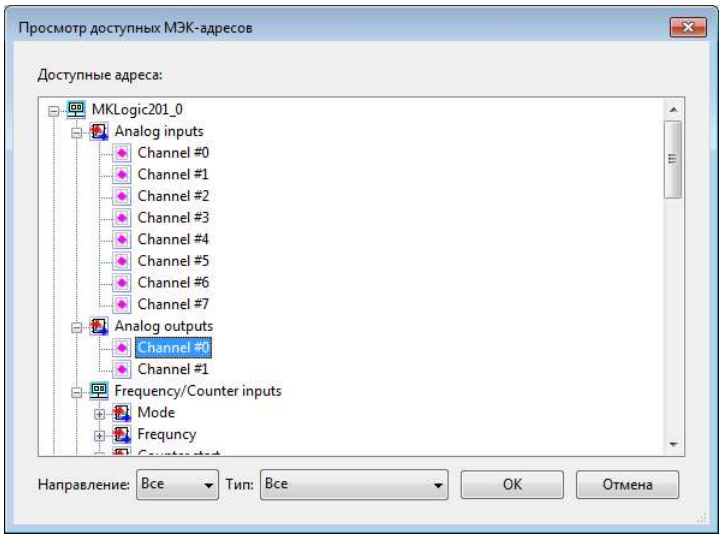


Рисунок 78 – Окно «Просмотр доступных МЭК-адресов»

В окне «Просмотр доступных МЭК-адресов» нужно выбрать интересующий канал ввода вывода (см. соответствие каналов и МЭК адресов в таблицах с 14 по 20, а также 22 и 27) и нажать кнопку «OK».

При этом панели переменных и констант программного модуля в строке с переменной в поле «Адрес» будет отображен адрес канала ввода-вывода (рис. 79).

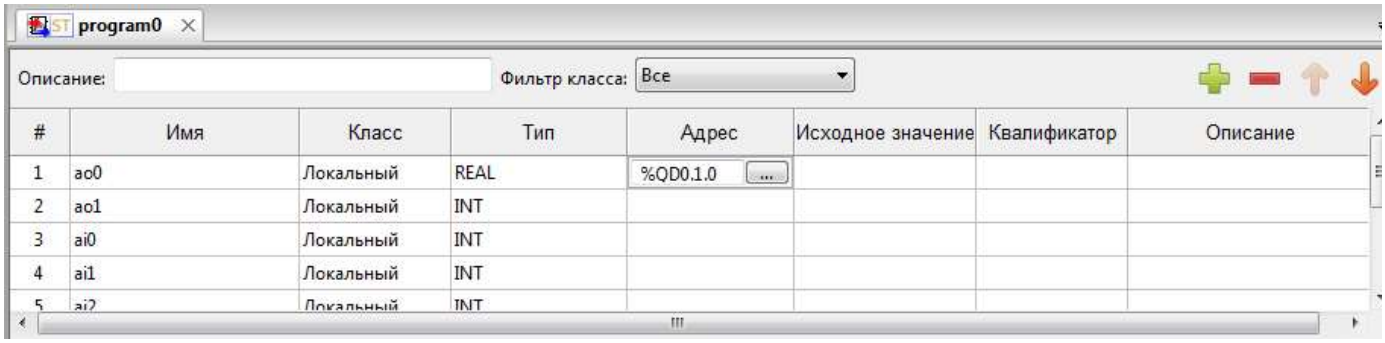


Рисунок 79 – Привязанный канал ввода-вывода к переменной программного модуля

Таким образом, этот канал будет привязан к переменной программы пользователя. При этом тип переменной автоматически сменится на тип, ассоциированный с данным каналом.

3.10.7 Плагин Си-расширений

В проект можно добавить Си-расширения. Это программы, написанные на языке Си, которые можно также исполнять в задачах.

ВНИМАНИЕ! Функция добавлена для будущих применений. В данной версии использовать Си-расширения не допускается.

3.10.8 Использование специальных функций расширения

Наряду со стандартными функциями и функциональными блоками (см. Приложение А) в библиотеке доступны специфические для Контроллера функции. В библиотеке эти функции объединены в группы, имя каждой группы начинается с префикса «МК200…».

При выборе соответствующей функции в нижней части панели библиотеки отображается краткое описание данной функции и ее формат: ее входные и выходные параметры.

Например, запись

```
(ANY:IN1, ANY:IN2) => (BOOL:OUT)
```

означает, что функция принимает два входных параметра (IN1 и IN2) любого типа (ключевое слово ANY) и возвращает значение типа BOOL.

3.10.8.1 Функции MK200 POUs

Описание функций группы «MK200 POUs» представлено в таблице 28.

Таблица 28 – Описание функций группы «MK200 POUs»

Имя функции	Описание
WORD_AS_REAL	Представление числа содержащегося в двух 16-тибитных числах без знака в виде числа с плавающей точкой (тип REAL) Формат функции: `(WORD:inHigh, WORD:inLow) => (REAL:out)`
REAL_AS_WORD	Представление числа с плавающей точкой (тип REAL) в виде двух 16-тибитных чисел без знака. Формат функции: `(REAL:in) => (WORD:outHigh, WORD:outLow)`

3.10.8.2 Функции MK200Modbus POUs

В группе «MK200Modbus POUs» представлены функции для работы с картой протокола Modbus в режиме ведомого (сервера). Описание функций группы «MK200Modbus POUs» представлено в таблице 29.

Таблица 29 – Описание функций группы «MK200Modbus POUs»

Имя функции	Описание
SET_HOLDING_REAL	Задает значение двух holding регистров, расположенных по адресам ADDR и ADDR+1, в виде числа с плавающей точкой (тип REAL) Формат функции: `(UINT:ADDR, REAL:IN_REAL) => (INT:OUT)`
SET_HOLDING_WORD	Задает значение holding регистра, расположенного по адресу ADDR Формат функции: `(UINT:ADDR, WORD:IN_WORD) => (INT:OUT)`

SET_HOLDING_LONG	Задает значение двух holding регистров, расположенных по адресам ADDR и ADDR+1, в виде 32-битного числа без знака (тип DWORD) Формат функции: (UINT:ADDR, DWORD:IN_DWORD) => (INT:OUT)
GET_HOLDING_REAL	Возвращает значение двух holding регистров, расположенных по адресам ADDR и ADDR+1, в виде числа с плавающей точкой (тип REAL) Формат функции: (UINT:ADDR) => (REAL:OUT)
GET_HOLDING_WORD	Возвращает значение holding регистра, расположенного по адресу ADDR Формат функции: (UINT:ADDR) => (REAL:OUT)
GET_HOLDING_LONG	Возвращает значение двух holding регистров, расположенных по адресам ADDR и ADDR+1, в виде 32-битного числа без знака (тип DWORD) Формат функции: (UINT:ADDR) => (DWORD:OUT)
SET_INPUT_WORD	Задает значение input регистра, расположенного по адресу ADDR Формат функции: (UINT:ADDR, WORD:IN_WORD) => (INT:OUT)
GET_INPUT_WORD	Возвращает значение input регистра, расположенного по адресу ADDR Формат функции: (UINT:ADDR) => (WORD:OUT)
SET_COIL	Задает значение битового поля coil, расположенного по адресу ADDR Формат функции: (UINT:ADDR, BOOL:IN_WORD) => (INT:OUT)
GET_COIL	Возвращает значение битового поля coil, расположенного по адресу ADDR Формат функции: (UINT:ADDR) => (BOOL:OUT)
SET_DISC	Задает значение битового поля discrete input, расположенного по адресу ADDR Формат функции: (UINT:ADDR, BOOL:IN_WORD) => (INT:OUT)
GET_DISC	Возвращает значение битового поля discrete input, расположенного по адресу ADDR Формат функции: (UINT:ADDR) => (BOOL:OUT)

3.10.8.3 Функции MK200Time POUs

Описание функций группы «MK200Time POUs» представлено в таблице

Таблица 30 – Описание функций группы «MK200Time POUs»

Имя функции	Описание

US_TIME	Возвращает количество микросекунд прошедшее с момента последней перезагрузки контроллера Формат функции: `(INT:NO_MATTER) => (UDINT:OUT)`

3.10.8.4 Функции MK200Service POUs

Таблица 31 – Описание функций группы «MK200Service POUs»

Имя функции	Описание
GET_APP_CRC	Возвращает контрольную сумму контроллера (CRC16) Формат функции: `(INT:NO_MATTER) => (WORD:OUT)`

3.10.8.5 Функции MK200CustomRetain POUs

Таблица 32 – Описание функций группы «MK200CustomRetain POUs»

Имя функции	Описание
GET_CUSTOM_RETAIN	Возвращает значение специальной энергонезависимой 16-битной переменной по адресу (ADDR). Допустимые значения адреса от 0 до 65535. Формат функции: `(INT:ADDR) => (WORD:OUT)`
SET_CUSTOM_RETAIN	Записывает значение специальной энергонезависимой 16-битной переменной по адресу (ADDR). Допустимые значения адреса от 0 до 65535. Формат функции: `(INT:ADDR, WORD:VAL) => (BYTE:OUT)`

3.11 Сборка проекта

После создания основных элементов проекта выполняется его сборка (компиляция и компоновка).

Сборка проекта осуществляется с помощью соответствующих кнопок на панели инструментов (см. 3.1.2). Как уже отмечалось, каждый проект должен иметь один ресурс. В ресурсе должна быть определена, как минимум, одна задача циклического типа и, как минимум, один экземпляр. Соответственно, проект обязан содержать, как минимум, один программный модуль типа «Программа», причём тело, т.е. алгоритм и логика его выполнения, не может быть пустым (в противном случае будет ошибка компиляции).

На рис. 80 представлен сообщения Консоли в случае успешной сборки проекта.

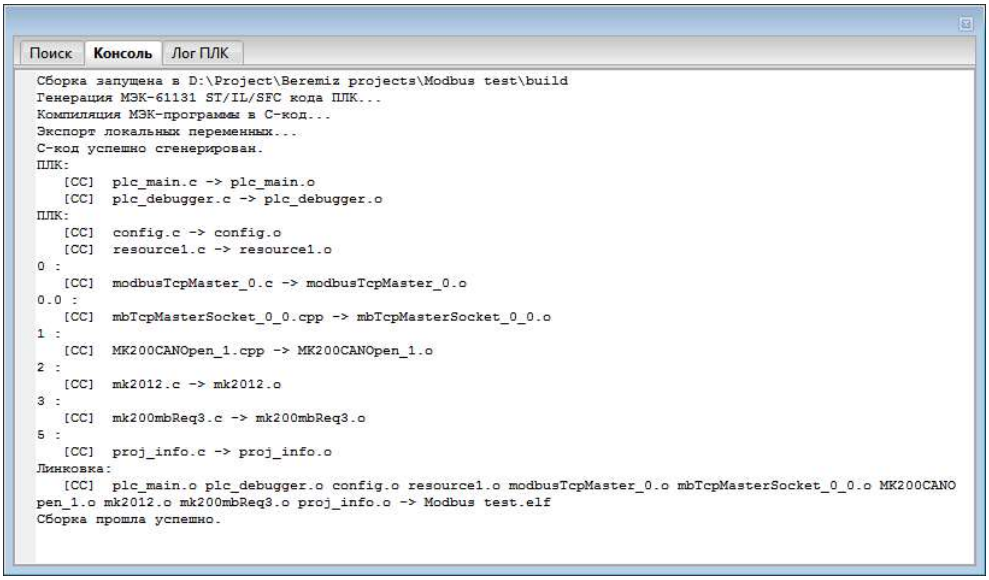


Рисунок 80 – Сообщения в Консоли при удачной сборке проекта

В случае если во время сборки произойдут ошибки, в Консоль будут выведены соответствующие ошибки с указанием мест в программных модулях, которые вызывают ошибки. Сообщения об ошибках подсвечиваются красным цветом (рис. 81).

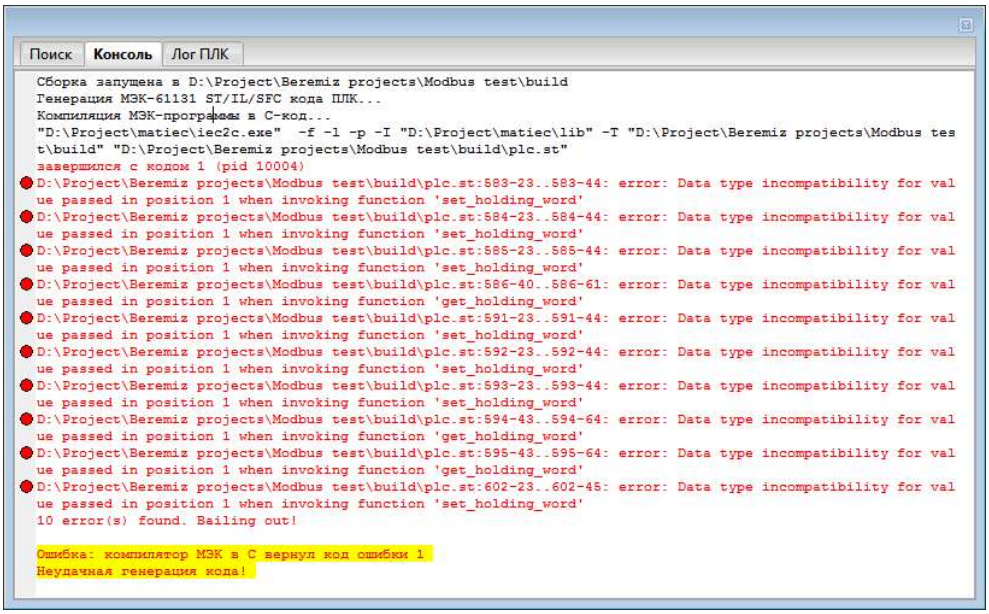


Рисунок 81 – Сообщения об ошибках в Консоли при сборке проекта

После удачной сборки программа пользователя готова к использованию в Контроллере. Программу пользователя необходимо загрузить в Контроллер, после чего она будет запущена на выполнение.

3.12 Работа с Контроллером в среде Beremiz

Работа с Контроллером в среде Beremiz заключается в загрузке готовой программы пользователя, а также настройки конфигурации его коммуникационных портов. Для работы с Контроллером необходимо осуществить подключение к Контроллеру в среде Beremiz.

3.12.1 Подключение к Контроллеру

Подключение Контроллера к среде Beremiz осуществляется при помощи:
- сервисного порта;
- одного из трех последовательных портов по протоколу Modbus RTU;
- Ethernet порта по протоколу Modbus TCP.

Перед подключением необходимо задать настройки подключения.

3.12.1.1 Настройки подключения

Настройки подключения задаются в специально окне, которое вызывается кнопкой  на панели инструментов (рис. 82).

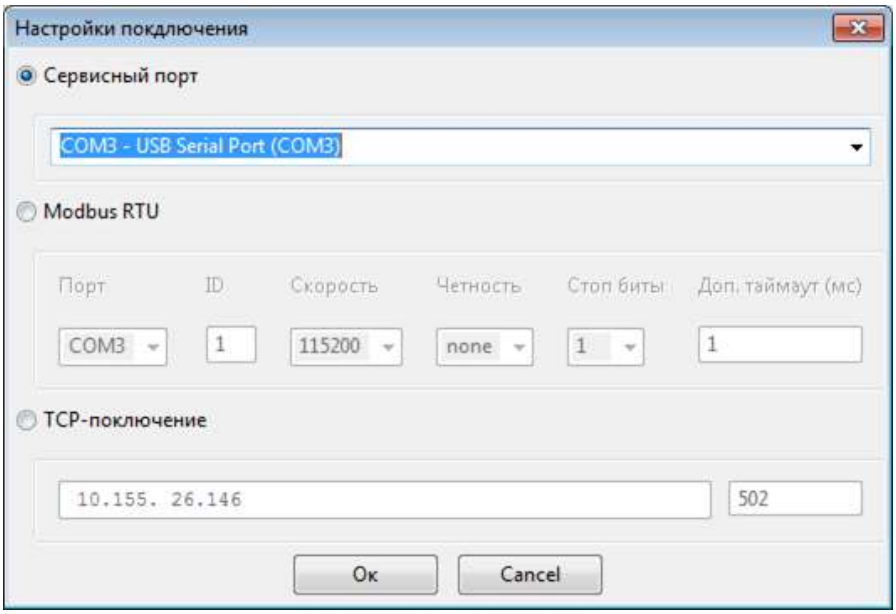


Рисунок 82 – Окно настроек подключения Контроллера к среде Beremiz

В окне настройки подключения следует выбрать нужный вариант подключения и ввести параметры этого подключения:
- для сервисного порта нужно указать номер COM-порта в системе, под которым определился сервисный порт контроллера;
- для связи по протоколу Modbus RTU необходимо указать номер COM-порта, к которому подключен контроллер, а также адрес ведомого устройства («ID»), скорость, четность, количество стоп бит и дополнительный таймаут (эти параметры должны быть заранее настроены для подключаемого последовательного порта контроллера);
- для TCP-подключения следует указать IP-адрес Контроллера, к которому производится подключение и TCP-порт подключения.

3.12.1.2 Подключение к контроллеру

Подключение к Контроллеру осуществляется нажатием кнопки подключения к целевому ПЛК  на панели статуса панели инструментов.

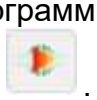
При удачном подключении к консоли среды разработки появится сообщение: «Отлаживаемая программа не соответствует программе в ПЛК - остановите/загрузите/запустите, чтобы разрешить отладку».

При этом панель статуса изменит вид, показанный на рис. 83:

Рисунок 83 – Панель статуса при удачном подключении

После подключения контроллера становятся доступными два действия: загрузка программы пользователя в контроллер и настройка его конфигурации.

3.12.2 Загрузка программы пользователя в Контроллер

Для загрузки программы пользователя необходимо на панели статуса нажать кнопку передачи в ПЛК  . Начнется процесс загрузки программы пользователя в Контроллер (рис. 84)

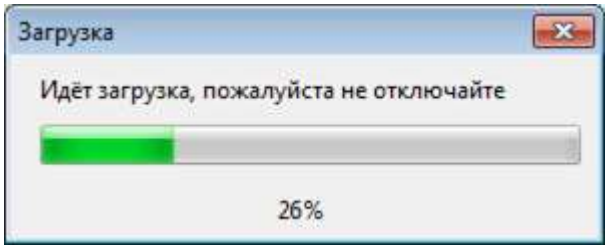


Рисунок 84 – Окно индикации процесса загрузки

Во время загрузки кнопки панели статуса недоступны в целях защиты от ошибок.

3.12.3 Настройка конфигурации Контроллера

Настройка конфигурации контроллера осуществляется по кнопке  . При этом вызывается окно конфигурации «MKSetup 0.0.1» (рис. 85).

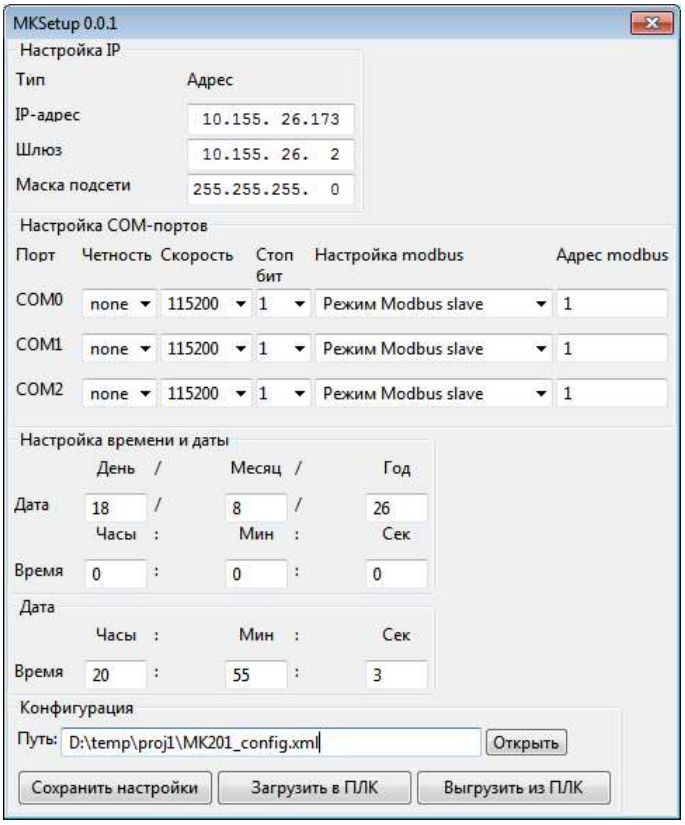


Рисунок 85 – Окно настройки конфигурации

В случае необходимости изменения существующих настроек контроллера в окне настройки конфигурации необходимо нажать кнопку «Выгрузить из ПЛК». Поля обновятся выгруженными из контроллера данными. Далее можно отредактировать параметры и загрузить настройки в контроллер, нажав кнопку «Загрузить в ПЛК».

В окне настройки конфигурации осуществляется настройка следующих параметров контроллера:

- IP адресов;
- последовательных портов;
- времени и даты;

В группе «Настройка IP» должны быть указаны:

- IP-адрес;
- адрес шлюза;
- адрес маски.

В группе «Настройка COM-портов» для каждого из портов указываются:

- четность (none/even/odd);
- скорость (1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200 кбит/с);
- количество стоп битов (1, 1.5, 2);
- настройки и адрес Modbus.

В настройках Modbus для каждого из портов может быть указан один из режимов:

- «Режим Modbus Master N» режим ведущего (клиента), где N = 0, 1, 2 — номер логического ведущего устройства (см. рис. 74);
- «Режим Modbus slave» режим ведомого (сервера);
- «Выключен» — данный COM-порт не работает.

В настройках времени и даты указывается внутреннее время Контроллера:

- дата в формате день/месяц/год;
- время в формате часы/минуты/секунды.

Окно настроек конфигурации может быть запущено и без подключения среды к контроллеру. В этом случае кнопки «Загрузить в ПЛК» и «Выгрузить из ПЛК» будут

неактивны. Пользователь может ввести нужные настройки и сохранить их в файл для загрузки в контроллер в будущем. Для сохранения настроек конфигурации применяется кнопка «Сохранить настройки». Путь сохранения файла конфигурации целевого устройства указывается в поле «Путь». Позже сохраненную конфигурацию можно загрузить по кнопке «Открыть».

ПРИЛОЖЕНИЕ А Описание библиотеки функций и функциональных блоков

Функциями и функциональными блоками называются предопределённые элементы, которые используются при реализации алгоритмов и логики программных модулей типа «Функциональный блок» и «Программа» на языках стандарта IEC 61131-3.

Данные элементы имеют параметры на входе и на выходе. Как правило, каждый параметр имеет имя и своё назначение.

1 Стандартные функциональные блоки

1.1 SR-триггер

Данный функциональный блок представляет собой SR-триггер с двумя устойчивыми состояниями, с управляющим входом S (Set). Выход Q1 переходит в состояние «1», когда вход S1 переходит в состояние «1». Это состояние сохраняется, даже если S1 возвращается обратно в «0». Выход Q1 переходит в состояние «0», когда вход R переходит в состояние «1». Если входы S1 и R находятся в состоянии «1» одновременно, управляющий вход S1 установит выход Q1 в состояние «1». Когда функциональный блок вызывается первый раз, выход Q1 находится в состоянии «0».

1.2 RS-триггер

Данный функциональный блок представляет собой RS-триггер с двумя устойчивыми состояниями, с управляющим входом R (Reset). Выход Q1 переходит в состояние «1», когда вход S переходит в состояние «1». Состояние сохраняется, даже если S возвращается обратно в состояние «0». Выход Q1 возвращается в состояние «0», когда вход R1 переходит в состояние «1». Если входы S и R1 находятся в состоянии «1» одновременно, управляющий вход R1 установит выход Q1 в «0». Когда функциональный блок вызывается первый раз, выход Q1 находится в состоянии «0».

1.3 SEMA – семафор

Данный функциональный блок представляет собой семафор, определяющий механизм, позволяющий элементам программы иметь взаимоисключающий доступ к ресурсам.

1.4 R_TRIG – индикатор нарастания фронта

Функциональный блок, представляющий собой индикатор нарастания фронта, который генерирует на выходе одиночный импульс при нарастании фронта сигнала. Выход Q переходит в состояние «1», если вход CLK из состояния «0» переходит в «1». Выход остается в состоянии «1» в течение одного цикла, а затем выход возвращается в «0».

1.5 F_TRIG – индикатор спада фронта

Функциональный блок, представляющий собой индикатор спада фронта, который генерирует на выходе одиночный импульс при спаде фронта сигнала.

Выход Q переходит в состояние «1», если вход CLK из состояния «1» переходит в «0». Выход остается в состоянии "1" в течение одного цикла, а затем возвращается в «0».

1.6 CTU – инкрементный счётчик

Функциональный блок, представляющий собой инкрементный счётчик. В случае, когда вход R находится в состоянии «1», выход CV переходит в состояние «0». При каждом переходе входа CU из «0» в «1» значение на выходе CV увеличивается на 1. Когда значение на выходе CV ≥ PV, выход Q перейдет в состояние «1».

Внимание! Счетчик работает только до достижения максимального значения используемого типа данных. Переполнения не происходит.

Входы CU, RESET и выход Q имеют тип BOOL, вход PV и выход CV - тип WORD.

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

1.7 CTD – декрементный счётчик

Функциональный блок, представляющий собой декрементный счётчик. Когда на вход LD поступает «1», значение на входе PV присваивается выходу CV . При каждом переходе из входа CD из состояния «0» в «1» значение на выходе CV уменьшается на 1.

Когда значение на выходе CV ≤ 0, выход Q переходит в состояние «1».

Внимание! Счетчик работает только до достижения минимального значения используемого типа данных. Переполнения не происходит.

1.8 CTUD – реверсивный счётчик

Функциональный блок, представляющий собой реверсивный счётчик. Когда на вход R поступает «1», на выходе CV устанавливается «0». Когда на вход LD поступает сигнал «1», значение на входе PV присваивается выходу CV. При каждом переходе входа CU из «0» в «1» значение на выходе CV увеличивается на 1. При каждом переходе входа CD из «0» в «1» значение на выходе CV уменьшается на 1.

Если «1» поступает одновременно на входы R и LD, вход R обрабатывается первым.

Когда значение на выходе CV ≥ значения на входе PV, выход QU принимает значение «1».

Когда значение на выходе CV ≤ 0, выход QD принимает значение «1».

Внимание! Вычитающий счетчик работает только до достижения минимального значения используемого типа данных, суммирующий счетчик работает только до достижения максимального значения используемого типа данных. Переполнения не происходит.

1.9 TP – повторитель импульсов

Функциональный блок, представляющий собой повторитель импульсов и использующийся для генерации импульсов с заданной продолжительностью. Если IN переходит в состояние «1», Q также переходит в состояние «1», и начинается отсчет внутреннего времени (ET). Если внутреннее время достигает значения PT, Q переходит в состояние «0» (независимо от состояния IN). Отсчет внутреннего времени останавливается/сбрасывается, если IN переходит в состояние «0». Если внутреннее время не достигло значения PT, импульс IN не влияет на внутреннее время. Если внутреннее время достигло значения PT, и IN находится в состоянии «0», отсчет внутреннего времени останавливается/ сбрасывается, и Q переходит в состояние «0».

1.10 TON – таймер с задержкой включения

Функциональный блок, представляющий собой таймер с задержкой включения. Он запускается, когда состояние сигнала на входе переходит из состояния «0» в состояние «1» и устанавливает на выходе «1» по истечении заданного времени.

Если IN переходит в состояние «1», запускается отсчет внутреннего времени (ET). Если внутреннее время достигает значения PT, Q переходит в состояние «1». Если IN переходит в состояние «0», Q также переходит в состояние «0», а подсчет внутреннего времени останавливается/ сбрасывается. Если IN переходит в состояние «0» до того, как внутреннее время достигло значения PT, подсчет внутреннего времени останавливается/ сбрасывается, а выход Q не переходит в состояние «0».

1.11 TOF – таймер с задержкой отключения

Функциональный блок, представляющий собой таймер с задержкой отключения. Он запускается, когда состояние сигнала на входе переходит из «1» в «0» и устанавливает на выходе состояние «0» по истечении заданного времени.

Если IN переходит в состояние «1», Q также переходит в состояние «1».

Если IN переходит в состояние «0», запускается отсчет внутреннего времени (ET).

Если внутреннее время достигает значения PT, Q переходит в состояние «0».

Если IN переходит в состояние «1», Q также переходит в состояние «1», а подсчет внутреннего времени останавливается/сбрасывается.

Если IN переходит в состояние «1» до того, как внутреннее время достигло значения PT, подсчет внутреннего времени останавливается/сбрасывается, а выход Q не устанавливается в «0».

2 Дополнительные функциональные блоки

2.1 PID – Пропорционально-интегрально-дифференциальный регулятор

Функциональный блок, представляющий собой устройство в цепи обратной связи, используемое в системах автоматического управления для формирования управляющего сигнала. ПИД-регулятор формирует управляющий сигнал, являющийся суммой трёх слагаемых, первое из которых пропорционально входному сигналу, второе – интеграл входного сигнала, третье – производная входного сигнала.

2.2 HYSTERESIS – гистерезис

Функциональный блок, предоставляющий собой выходное гистерезисное булево значение, которое определяется разницей вводных параметров XIN1 и XIN2 (типа REAL с плавающей точкой).

2.3 Преобразования типов

Набор функций, предназначенный для преобразования между типами данных, согласно стандарту IEC 61131-3.

2.4 Числовые операции

2.4.1 ABS – модуль числа

Функция возвращает на выходе OUT значение входного числа IN, взятое по модулю.

2.4.2 SQRT – квадратный корень

Функция возвращает на выходе OUT квадратный корень входного числа IN.

2.4.3 LN – натуральный логарифм

Функция возвращает на выходе OUT значение натурального логарифма от входного числа IN.

2.4.4 LOG – логарифм по основанию 10

Функция возвращает на выходе OUT значение логарифма по основанию 10 от входного числа IN.

2.4.5 EXP – возведение в степень экспоненты

Функция возвращает на выходе OUT значение экспоненты, возведённой в степень IN.

2.4.6 SIN – синус

Функция возвращает на выходе OUT значение синуса IN.

2.4.7 COS – косинус

Функция возвращает на выходе OUT значение косинуса IN.

2.4.8 TAN – тангенс

Функция возвращает на выходе OUT значение тангенса IN.

2.4.9 ASIN – арксинус

Функциональный блок возвращает на выходе OUT значение арксинуса IN.

2.4.10 ACOS – арккосинус

Функция возвращает на выходе OUT значение арккосинуса IN.

2.4.11 ATAN – арктангенс

Функция возвращает на выходе OUT значение арктангенса IN.

2.5 Арифметические операции

2.5.1 ADD – сложение

Функция возвращает на выходе OUT результат сложения IN1 и IN2.

2.5.2 MUL – умножение

Функция возвращает на выходе OUT результат умножения IN1 и IN2.

2.5.3 SUB – вычитание

Функция возвращает на выходе OUT результат вычитания из IN1 значения IN2.

2.5.4 DIV – деление

Функция возвращает на выходе OUT результат деления IN1 на IN2.

2.5.5 MOD – остаток от деления

Функция возвращает на выходе OUT остаток от деления IN1 на IN2.

2.5.6 EXPT – возведение в степень

Функция возвращает на выходе OUT значение IN1, возведённое в степень IN2.

2.5.7 MOVE – присвоение

Функция присваивает выходу OUT входное значение IN.

2.6 Временные операции

2.6.1 ADD_TIME – сложение переменных типа TIME

Функция складывает входные значения IN(k) типа TIME и возвращает результат в OUT типа TIME. Количество входов IN(n) изменяемое - от 2 до 20. По умолчанию 2.

2.6.2 ADD_TOD_TIME – сложение времени дня TOD с интервалом времени TIME

Функция складывает входную переменную IN1 типа TOD (TIME_OF_DAY) с переменной IN2 типа TIME. Возвращаемая величина OUT имеет тип TIME_OF_DAY.

2.6.3 ADD_DT_TIME – прибавление промежутка времени TIME к моменту времени DT

Функция ADD_DT_TIME прибавляет промежуток времени (формат TIME) к моменту времени (формат DT) и поставляет в качестве результата новый момент времени (формат DT). Момент времени (параметр T) должен лежать в диапазоне от DT#1990-01-01-00:00:00.000 до DT#2089-12-31-23:59:59.999.

Функция не выполняет входной проверки. Если результат сложения не лежит внутри допустимого диапазона, то результат ограничивается соответствующим значением и бит двоичного результата (BR) слова состояния устанавливается в «0».

Для входного параметра T и выходного параметра можно ставить в соответствие только символически определенную переменную.

2.6.4 MULTIME – умножение времени TIME на число

Функция выполняет умножение входного значения IN1 типа TIME на число IN2 типа ANY_NUM и возвращает результат в OUT типа TIME.

2.6.5 SUB_TIME – разность двух значений типа TIME

Данная функция вычитает из входного значения IN1 типа TIME значение на входе IN2 типа TIME и возвращает результат в OUT типа TIME.

2.6.6 SUB_DATE_DATE – разность двух значений типа DATE

Данная функция вычитает из входного значения IN1 типа DATE входное значение IN2 типа DATE и возвращает в OUT их разницу типа TIME.

2.6.7 SUB_TOD_TIME – вычитание из времени дня TOD интервала времени TIME

Данная функция вычитает из входного значения IN1 типа TOD (TIME_OF_DAY) входное значение IN2 типа TIME и возвращает результат в OUT типа TIME_OF_DAY.

2.6.8 SUB_DT_TIME – вычитание из момента времени DT промежутка времени TIME

Данная функция вычитает промежуток времени (формат TIME) из момента времени (формат DT) и поставляет в качестве результата новый момент времени (формат DT). Момент времени (параметр T) должен лежать в диапазоне от DT#1990-01-01-00:00:00.000 до DT#2089-12-31-23:59:59.999. Функция не выполняет входной проверки. Если результат вычитания не лежит внутри допустимого диапазона, то результат ограничивается соответствующим значением и бит двоичного результата (BR) слова состояния устанавливается в "0".

Для входного параметра T и выходного параметра можно ставить в соответствие только символически определенную переменную.

2.6.9 DIVTIME – деление времени TIME на число

Данная функция выполняет деление входного значения IN1 типа TIME на число IN2 типа ANY_NUM и возвращает результат в OUT типа TIME.

2.7 Операции смещения бит

2.7.1 SHL – арифметический сдвиг влево

Данная функция возвращает в OUT арифметический сдвиг аргумента IN на N бит влево с заполнением битов справа нулями.

2.7.2 SHR – арифметический сдвиг вправо

Данная функция возвращает в OUT арифметический сдвиг аргумента IN на N бит вправо с заполнением битов слева нулями.

2.7.3 ROR – циклический сдвиг направо

Данная функция возвращает в OUT циклический сдвиг аргумента IN на N бит влево.

2.7.4 ROL – циклический сдвиг влево

Данная функция возвращает в OUT циклический сдвиг аргумента IN на N бит вправо.

2.8 Побитовые операции

2.8.1 AND – побитовое «И»

Данный функциональный блок представляет собой организацию «логического И» для всех входных аргументов IN1…INn.

2.8.2 OR – побитовое ИЛИ

Данная функция представляет собой организацию «логического ИЛИ» для всех входных аргументов IN1…INn.

2.8.3 XOR – побитовое исключающее ИЛИ

Данная функция представляет собой организацию «логического исключающего ИЛИ» для всех входных аргументов IN1…INn.

2.8.4 NOT – побитовая инверсия

Данная функция представляет собой организацию «логической инверсии» для входного аргумента IN.

2.9 Операции выбора

2.9.1 SEL – выбор из двух значений

Данная функция возвращает в OUT один из двух аргументов IN1 или IN2 в зависимости от значения аргумента G. Если G = 0, то OUT равно X1, иначе – OUT равно X2.

2.9.2 MAX – максимум

Данная функция возвращает в OUT максимум из входных аргументов IN1 и IN2.

2.9.3 MIN – минимум

Данная функция возвращает в OUT минимум из входных аргументов IN1 и IN2.

2.9.4 LIMIT – ограничитель значения

Данная функция возвращает в OUT значение входного аргумента IN, в случае превышения им значения MX – в OUT возвращается MX, в случае если IN меньше MN – в OUT возвращается MN.

2.9.5 MUX – Мультиплексор (выбор 1 из N)

Данная функция возвращает в OUT значение на входе IN(K), в зависимости от входного K. Количество входов IN(n) изменяемое – от 2 до 20. По умолчанию 2.

2.10 Операции сравнения

2.10.1 GT – больше чем

Данная функция сравнивает все входные аргументы и выдаёт на выходе OUT значение True, если выполнится следующее условие: (IN1 > IN2) & (IN2 > IN3) & … (INn-1 > INn), в противном случае в OUT выдаётся False. Количество входов IN(n) изменяемое – от 2 до 20. По умолчанию 2.

2.10.2 GE – больше чем или равно

Данная функция сравнивает все входные аргументы и выдаёт на выходе OUT значение True, если выполнится следующее условие: (IN1 >= IN2) & (IN2 >= IN3) & … (INn-1 >= INn), в противном случае в OUT выдаётся False. Количество входов IN(n) изменяемое – от 2 до 20. По умолчанию 2.

2.10.3 EQ – равенство

Данная функция сравнивает все входные аргументы и выдаёт на выходе OUT значение True, если выполнится следующее условие: (IN1 = IN2) & (IN2 = IN3) & … (INn-1 = INn), в противном случае в OUT выдаётся False. Количество входов IN(n) изменяемое – от 2 до 20. По умолчанию 2.

2.10.4 LT – меньше чем

Данная функция сравнивает все входные аргументы и выдаёт на выходе OUT значение True, если выполнится следующее условие: (IN1 < IN2) & (IN2 < IN3) & … (INn-1 < INn), в противном случае в OUT выдаётся False. Количество входов IN(n) изменяемое – от 2 до 20. По умолчанию 2.

2.10.5 LE – меньше чем или равно

Данная функция сравнивает все входные аргументы и выдаёт на выходе OUT значение True, если выполнится следующее условие: (IN1 <= IN2) & (IN2 <= IN3) & … (INn-1 <= INn), в противном случае в OUT выдаётся False. Количество входов IN(n) изменяемое – от 2 до 20. По умолчанию 2.

2.10.6 NE – не равно

Данная функция сравнивает все входные аргументы и выдаёт на выходе OUT значение True, если выполнится следующее условие: (IN1 <> IN2) & (IN2 <> IN3) & … (INn-1 <> INn), в противном случае в OUT выдаётся False. Количество входов IN(n) изменяемое - от 2 до 20. По умолчанию 2.

2.11 Строковые операции с переменными типа

2.11.1 STRING LEN – длина строки

Данная функция возвращает в OUT длину строки IN. Входному параметру можно ставить в соответствие только символически определенную переменную.

2.11.2 LEFT – левая часть строки

Данная функция возвращает в OUT из строки IN первые L символов. Если L больше, чем текущая длина переменной типа STRING, то возвращается входное значение. При L = 0 и при пустой строке в качестве входного значения возвращается пустая строка. Если число L отрицательно, то выводится пустая строка. Параметру IN и возвращаемому значению можно ставить в соответствие только символически определенную переменную.

2.11.3 RIGHT – правая часть строки

Данная функция возвращает в OUT из строки IN последние L символов. Если L больше, чем текущая длина переменной STRING, то возвращается входное значение. При L = 0 и при пустой строке в качестве входного значения возвращается пустая строка. Если число L отрицательно, то выводится пустая строка. Параметру IN и возвращаемому значению можно ставить в соответствие только символически определенную переменную.

2.11.4 MID – середина строки

Данная функция возвращает в OUT из строки IN L-символов, начиная с позиции P. Если сумма L и (P–1) превосходит текущую длину переменной типа STRING, то возвращается строка символов, начиная с P-го символа входной строки до ее конца. Во всех остальных случаях (P находится вне текущей длины, P и/или L равны нулю или отрицательны) выводится пустая строка. Параметру IN и возвращаемому значению можно ставить в соответствие только символически определенную переменную.

2.11.5 CONCAT – объединение двух переменных STRING

Данная функция возвращает в OUT объединение (конкатенацию) строк IN1 и IN2.

2.11.6 CONCAT_DAT_TOD – объединение (конкатенация) времени

Данная функция возвращает в OUT типа DT конкатенацию входных значений типов DATE и TOD, соответственно IN1 и IN2.

2.11.7 INSERT – вставка в переменной STRING

Данная функция возвращает в OUT строку IN1, в которую вставлена строка IN2, начиная с позиции P. Если P равно нулю, то вторая строка символов вставляется перед первой строкой символов. Если P больше, чем текущая длина первой строки символов, то вторая строка символов присоединяется к первой. Если P отрицательно, то выводится пустая строка. Входным параметрам IN1 и IN2 и выходному параметру можно ставить в соответствие только символически определенную переменную.

2.11.8 DELETE – удаление в переменной STRING

Данная функция возвращает в OUT строку IN1, в которой удалено L символов, начиная с позиции P. Если L и/или P равны нулю или P больше, чем текущая длина входной строки, то возвращается входная строка. Если сумма L и P больше, чем входная строка символов, то строка символов удаляется до конца. Если L и/или P имеют отрицательное значение, то выводится пустая. Входному параметру IN и выходному параметру можно ставить в соответствие только символически определенную переменную.

2.11.9 REPLACE – замена в переменной STRING

Данная функция возвращает в OUT строку IN1, в которой символы, начиная с позиции P, заменены L первыми символами строки IN2. Если L равно нулю, то возвращается первая строка символов. Если P равно нулю или единице, то замена происходит, начиная с 1-го символа (включительно). Если P лежит вне первой строки символов, то вторая строка присоединяется к первой строке. Если L и/или P отрицательны, то возвращается пустая строка. Входным параметрам IN1 и IN2 и выходному параметру можно ставить в соответствие только символически определенную переменную.

2.11.10 FIND – поиск в переменной STRING

Данная функция возвращает в OUT номер позиции, в которой находится строка IN2 в строке IN1. Поиск начинается слева, сообщается о первом появлении строки символов. Если вторая строка символов не содержится в первой, то возвращается нуль. Входным параметрам IN1 и IN2 можно ставить в соответствие только символически определенную переменную.

ПРИЛОЖЕНИЕ Б Описание языка ST

1 Общие сведения о языке ST

ST (Structured Text) – это текстовый язык высокого уровня общего назначения, по синтаксису схожий с языком Pascal. Удобен для программ, включающих числовой анализ или сложные алгоритмы. Может использоваться в программах, в теле функции или функционального блока, а также для описания действия и перехода внутри элементов SFC. Согласно IEC 61131-3 ключевые слова должны быть введены в символах верхнего регистра. Пробелы и метки табуляции не влияют на синтаксис, они могут использоваться везде.

Выражения в ST выглядят точно так же, как и в языке Pascal:

[variable] := [value];

Порядок их выполнения – справа налево. Выражения состоят из операндов и операторов. Операндом является литерал, переменная, структурированная переменная, компонент структурированной переменной, обращение к функции или прямой адрес.

2 Типы данных

Согласно стандарту IEC 61131-3, язык ST поддерживает весь необходимый набор типов, аналогичный классическим языкам программирования. Целочисленные типы: SINT (char), USINT (unsigned char), INT (short int), UINT (unsigned int), DINT (long), UDINT (unsigned long), LINT (64 бит целое), ULINT (64 бит целое без знака). Действительные типы: REAL (float), LREAL (double). Специальные типы BYTE, WORD, DWORD, LWORD представляют собой битовые строки длиной 8, 16, 32 и 64 бит соответственно. Битовых полей в ST нет. К битовым строкам можно непосредственно обращаться побитно. Например:

a.3 := 1; (* Установить бит 3 переменной a *)

Логический тип BOOL может иметь значение TRUE или FALSE. Физически переменная типа BOOL может соответствовать одному биту. Строка STRING является именно строкой, а не массивом. Есть возможность сравнивать и копировать строки стандартными операторами. Например:

strA := strB;

Для работы со строками есть стандартный набор функций (см. приложение А, раздел 2.11).

Специальные типы в стандарте IEC определены для длительности (TIME), времени суток (TOD), календарной даты (DATE) и момента времени (DT).

В таблице Б.1 приведены значения по умолчанию, соответствующие описанным выше типам.

Таблица Б.1 – Значения по умолчанию для типов данных IEC 61131-3

Тип(ы) данных	Значение
BOOL, SINT, INT, DINT, LINT	0
USINT, UINT, UDINT, ULINT	0
BYTE, WORD, DWORD, LWORD	0
REAL, LREAL	0.0
TIME	T#0S
DATE	D#0001-01-01

TIME_OF_DAY	TOD#00:00:00
DATE_AND_TIME	DT#0001-01-01-00:00:00
STRING	‘’ (пустая строка)

По умолчанию, все переменные инициализируются нулем. Иное значение переменной можно указать явно при ее объявлении. Например:

```
str1: STRING := 'Hello world';
```

В определённых ситуациях при разработке программных модулей удобно использовать обобщения типов, т.е. общее именование группы типов данных. Данные обобщения приведены в таблице Б.2.

Таблица Б.2 – Обобщения типов данных IEC 61131-3

<table>
<tr><td colspan="6">ANY</td></tr>
<tr><td>ANY_BIT</td><td colspan="3">ANY_NUM</td><td>ANY_DATE</td><td rowspan="2">TIME
STRING
и
другие
типы
данных</td></tr>
<tr><td rowspan="5">BOOL
BYTE
WORD
DWORD
LWORD</td><td colspan="2">ANY_INT</td><td>ANY_REAL</td><td rowspan="4">DATE
TIME_OF_DAY
DATE_AND_TIME</td></tr>
<tr><td>INT
SINT
DINT
LINT</td><td>UINT
USINT
UDINT
ULINT</td><td rowspan="3">REAL
LREAL</td></tr>
</table>

3 Конструкции языка

К конструкциям языка ST относятся:
- арифметические операции;
- логические (побитовые) операции;
- операции сравнения;
- операция присвоения;
- конструкция IF – ELSEIF – ELSE;
- цикл FOR;
- цикл WHILE;
- цикл REPAET UNTIL;
- конструкция CASE.

При записи арифметических выражений допустимо использование скобок для указания порядка вычислений. При записи выражений допустимо использовать переменные (локальные и глобальные) и константы.

3.1 Арифметические операции

К арифметическим операциям относятся:
- «+» – сложение;
- «-» – вычитание;
- «*» – умножение;
- «/» – деление;
- «mod» – остаток от целочисленного деления.

3.2 Логические (побитовые) операции

К данным операциям относятся:

- «OR» – логическое (побитовое) сложение;
- «AND» – логическое (побитовое) умножение;
- «XOR» – логическое (побитовое) «исключающее ИЛИ»;
- «NOT» – логическое (побитовое) отрицание.

3.3 Операторы сравнения

Поддерживаются следующие операторы сравнения:

- «=» — оператор «равенство»;
- «<>» — оператор «неравенство»;
- «>» — оператор «больше»;
- «>=» — оператор «больше или равно»;
- «<» — оператор «меньше»;
- «<=» — оператор «меньше или равно».

Результат сравнения - значение типа BOOL.

3.4 Оператор присвоения

Оператор присвоения обозначается знаком «:=». В правой и левой части выражения должны быть операнды одного типа (автоматического приведения типов не предусмотрено). В левой части выражения (принимающая сторона) может быть использована только переменная. Правая часть может содержать выражение или константу.

В таблице Б.3 приведены приоритеты при выполнении описанных выше операций.

Таблица Б.3 – Приоритеты операций

Операция	**Приоритет**
сравнение	1
сложение, вычитание	2
умножение, деление	3
OR	4
AND, XOR	5
NOT	6
унарный минус	7
вызов функции	8

3.5 Конструкция IF – ELSEIF – ELSE

Для описания некоторых конструкций языка удобно использовать фигурные и квадратные скобки. Считается, что:

- выражение в фигурных скобках может использоваться ноль или больше раз подряд;
- выражение в квадратных скобках не обязательно к использованию.

Конструкция IF-ELSEIF-ELSE имеет следующий формат:

```
IF <boolean expression> THEN
<statement list>
[ELSEIF <boolean expression> THEN <statement list>]
```

```
[ELSE <statement list>]
END_IF;
```

Например:

```
IF Var <> 0
THEN Var := 1
ELSEIF Var > 0
THEN Var := 0; ELSE Var := 10;
END_IF;
```

Конструкция допускает вложенность, т.е. внутри одного IF может быть еще один и т.д.

Например:

```
IF Var > 10 THEN
IF Var < Var2 + 1
THEN Var := 10; ELSE Var := 0;
END_IF; END_IF;
```

3.6 Оператор цикла FOR

Оператор цикла с фиксированным количеством итераций. Формат конструкции следующий:

```
FOR <Control Variable> := <expression1> TO <expression2>
[BY <expression3>] DO
<statement list>
END_FOR;
```

При задании условий цикла считается, что <Control Variable>, <expression1> … <expression3> имеют тип INT. Выход из цикла будет произведен в том случае, если значение переменной цикла превысит значение <expression2>.

Например:

```
FOR i := 1 TO 10 BY 2 DO
k := k * 2;
END_FOR;
```

Оператор BY задает приращение переменной цикла (в данном случае i будет увеличиваться на 2 при каждом проходе по циклу). Если оператор BY не указан, то приращение равно 1. Например:

```
FOR i := 1 TO k / 2 DO
 var := var + k;
 k := k – 1;
END_FOR;
```

Внутри цикла могут использоваться другие циклы, операторы IF и CASE. Для выхода из цикла (любого типа) может использоваться оператор EXIT. Например:

```
FOR i := 1 TO 10 BY 2 DO
```

```
 k := k * 2;
 IF k > 20 THEN
 EXIT;
 END_IF;
END_FOR;
```

Примечание 1: Выражения <expression1> … <expression3> вычисляются до входа в цикл, поэтому изменения значений переменных, входящих в любое из этих выражений не приведет к изменению числа итераций. Например:

```
01: k := 10;
02: FOR I := 1 TO k / 2 DO
03: k := 20;
04: END_FOR;
```

В строке 3 производится изменение переменной k, но цикл все равно выполнится только пять раз. Примечание 2: Значение переменной цикла может изменяться внутри тела цикла, но в начале очередной итерации значение данной переменной будет выставлено в соответствие с условиями цикла. Например:

```
01: FOR I := 1 TO 5 DO
02: I := 55;
03: END_FOR;
```

При первом проходе значение I будет равно 1, потом в строке 2 изменится на 55, но на втором проходе значение I станет равно 2 – следующему значению по условиям цикла.

3.7 Оператор цикла WHILE

Оператор цикла с предусловием. Цикл будет исполняться до тех пор, пока выражение в предложении WHILE возвращает TRUE. Формат конструкции следующий:

```
WHILE <Boolean-Expression> DO
 <Statement List>
END_WHILE;
```

Значение <Boolean-Expression> проверяется на каждой итерации. Завершение цикла произойдет, если выражение <Boolean-Expression> вернет FALSE. Например:

```
k := 10;
WHILE k > 0 DO
 i := I + k;
 k := k -1;
END_WHILE;
```

Внутри цикла могут использоваться другие циклы, операторы IF и CASE. Для досрочного завершения цикла используется оператор EXIT (см. пример в описание цикла FOR).

3.8 Оператор цикла REPEAT UNTIL

Оператор цикла с постусловием. Завершение цикла произойдет тогда, когда выражение в предложении UNTIL вернет FALSE. Другими словами: цикл будет выполняться, пока условие в предложении UNTIL не выполнятся. Формат конструкции следующий:

```
REPEAT
 <Statement List>
UNTIL <Boolean Expression>;
END_REPEAT;
```

Например:

```
k := 10;
REPEAT
 i := i + k;
 k := k - 1;
UNTIL k = 0;
END_REPEAT;
```

Внутри цикла могут использоваться другие циклы, операторы IF и CASE. Для досрочного завершения цикла используется оператор EXIT (см. пример в описании цикла FOR).

3.9 Конструкция CASE

Данная конструкция служит для организации выбора из диапазона значений. Формат конструкции следующий:

```
CASE <Expression> OF
CASE_ELEMENT {CASE_ELEMENT} [ELSE <Statement List>]
END_CASE;
```

CASE_ELEMENT — это список значений, перечисленных через запятую. Элементом списка может быть целое число или диапазон целых чисел. Диапазон задается следующим образом BEGIN_VAL .. END_VAL.

Если текущее значение <Expression> не попало ни в один CASE_ELEMENT, то управление будет передано на предложение ELSE. Если предложение ELSE не указано, то никаких действий выполнено не будет.

Значение <Expression> может быть только целым. Например:

```
01: CASE k OF
02: 1:
03: k := k * 10;
04: 2..5:
05: k := k * 5;
06: i := 0;
07: 6, 9..20:
08: k := k - 1;
09: ELSE
10: k := 0;
11: i := 1;
12: END_CASE;
```

Строка 4 содержит диапазон значений. Если значение k принадлежит числовому отрезку [2, 5], то будут выполнены строки 5 и 6.

В строке 7 использован список значений. Строка 8 выполнится, если значение k будет равно 6 или будет принадлежать числовому отрезку [9, 20].

Строки 10 и 11 будут выполнены в том случае, если k < 1, или 6 < k < 9, или k > 20 (в данном случае сработает предложение ELSE).

При задании списка значений необходимо выполнять следующие условия:

- наборы значений внутри одного CASE не должны пересекаться;
- при указании диапазона значений начало диапазона должно быть меньше его конца.

В таблице Б.4 приведены примеры кода записи правильной и неправильной записи конструкции CASE.

Действия, предусмотренные для обработки каждого из случаев CASE, могут использовать циклы, операторы IF и CASE.

Таблица Б.4 – Запись конструкции CASE

Неправильная запись	Правильная запись
01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 5, 9..20: 08: k := k – 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE; Диапазоны в строках 4 и 7 пересекаются	01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 6, 9..20: 08: k := k – 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;
01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 6, 20..9: 08: k := k – 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE; В строке 7 диапазон значений задан неправильно	01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 6, 9..20: 08: k := k – 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;

При написании программ на ST возможно использование стандартных и пользовательских функций и функциональных блоков.

ПРИЛОЖЕНИЕ В Описание языка IL

1 Общие сведения о языке IL

IL (Instruction List) представляет собой текстовый язык программирования низкого уровня, схожий с Assembler'ом, но, в отличие от него, не привязанный к конкретной архитектуре процессора. Он позволяет описывать функции, функциональные блоки и программы, а также шаги и переходы в языке SFC. Одним из ключевых преимуществ IL является его простота и возможность добиться оптимизированного кода для реализации критических секторов программ. Особенности IL делают его неудобным для описания сложных алгоритмов с большим количеством ветвлений.

2 Операторы языка

Основные операторы языка программирования IL, как и языка Assembler - переходы по меткам и аккумулятор. В аккумулятор загружаются значения переменной, а дальнейшее выполнение алгоритма представляет собой извлечение значения из аккумулятора и совершение над ним операций. Далее в таблице В.1 приведены операторы языка IL.

Таблица В.1 – Операторы языка IL

Оператор	Описание
LD	Загрузить значение операнда в аккумулятор
LDN	Загрузить обратное значение операнда в аккумулятор
ST	Присвоить значение аккумулятора операнду
STN	Присвоить обратное значение аккумулятора операнду
S	Если значение аккумулятора TRUE, установить логический операнд
R	Если значение аккумулятора FALSE, сбросить логический операнд
AND	«Поразрядное И» аккумулятора и операнда
ANDN	«Поразрядное И» аккумулятора и обратного операнда
OR	«Поразрядное ИЛИ» аккумулятора и операнда
ORN	«Поразрядное ИЛИ» аккумулятора и обратного операнда
XOR	«Поразрядное разделительное ИЛИ» аккумулятора и операнда
XORN	«Поразрядное разделительное ИЛИ» аккумулятора и обратного операнда
NOT	«Поразрядная инверсия» аккумулятора
ADD	Сложение аккумулятора и операнда, результат записывается в аккумулятор
SUB	Вычитание операнда из аккумулятора, результат записывается в аккумулятор
MUL	Умножение аккумулятора на операнд, результат записывается в аккумулятор
DIV	Деление аккумулятора на операнд, результат записывается в аккумулятор
GT	Значение аккумулятора сравнивается со значением операнда(>(greater than)). Значение (TRUE или FALSE) записывается в аккумулятор
GE	Значение аккумулятора сравнивается со значением

	операнда(>=greater than or equal)). Значение (TRUE или FALSE) записывается в аккумулятор
EQ	Значение аккумулятора сравнивается со значением операнда (=(equal)). Значение (TRUE или FALSE) записывается в аккумулятор
NE	Значение аккумулятора сравнивается со значением операнда (<>(not equal). Значение (TRUE или FALSE) записывается в аккумулятор
LE	Значение аккумулятора сравнивается со значением операнда (<=(less than or equal to)). Значение (TRUE или FALSE) записывается в аккумулятор
LT	Значение аккумулятора сравнивается со значением операнда (<(less than)). Значение (TRUE или FALSE) записывается в аккумулятор
JMP	Переход к метке
JMPC	Переход к метке при условии, что значение аккумулятора TRUE
JMPCN	Переход к метке при условии, что значение аккумулятора FALSE
CAL	Вызов программного или функционального блока
CALC	Вызов программного или функционального блока при условии, что значение аккумулятора TRUE
CALCN	Вызов программного или функционального блока при условии, что значение аккумулятора FALSE
RET	Выход из POU и возврат в вызывающую программу
RETC	Выход из POU и возврат в вызывающую программу при условии, что значение аккумулятора TRUE
RETCN	Выход из POU и возврат в вызывающую программу при условии, что значение аккумулятора FALSE

3 Пример программы на языке IL

Ниже приведён пример программы на языке IL, которая эквивалентна следующему логическому выражению C = A AND NOT B:

```
LD A
ANDN B
ST C
```

Первый оператор примера LD помещает значение переменной A в аккумулятор, способный хранить значения любого типа. Второй оператор ANDN выполняет «побитовое И» аккумулятора и обратного значения операнда, результат всегда помещается в аккумулятор. Последний оператор примера ST присваивает переменной C значение аккумулятора.

ПРИЛОЖЕНИЕ Г Описание языка FBD

1 Общие сведения о языке FBD

FBD (Function Block Diagram) – это графический язык программирования высокого уровня, обеспечивающий управление потоком данных всех типов. Позволяет использовать мощные алгоритмы простым вызовом функций и функциональных блоков. Удовлетворяет непрерывным динамическим процессам. Подходит для небольших приложений и удобен для реализации сложных вещей подобно ПИД регуляторам, массивам и т. д. Данный язык может использовать большую библиотеку блоков, описание которых приведено в Приложении А. FBD заимствует символику булевой алгебры и, так как булевы символы имеют входы и выходы, которые могут быть соединены между собой, FBD является более эффективным для представления структурной информации, чем язык релейно-контактных схем.

2 Основные понятия и конструкции языка

Согласно IEC 61131-3, основными элементами языка FBD являются: переменные, функции, функциональные блоки и соединения.

Переменные делятся на входные, выходные и входные/ выходные. На рис. Г.1 показаны: входная переменна — «in_var», выходная переменная — «out_var» и входная/выходная переменная — «in_out_var».

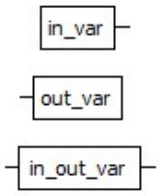


Рисунок Г.1 – Изображение переменной в языке FBD

Графическое изображение функции приведено на рис. Г.2. С левой стороны располагаются входы (IN1 и IN2), с правой стороны выходы (OUT).

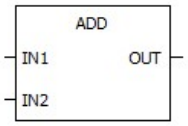


Рисунок Г.2 – Изображение функции в языке FBD

Аналогично, изображение функционального блока, приведённое на рис. Г.3, имеет с левой стороны входы (S1 и R), с правой стороны выход (Q1).

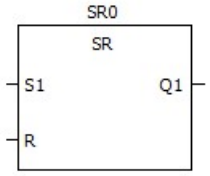


Рисунок Г.3 – Изображение функционального блока в языке FBD

Соответственно, переменные соединяются с входными и выходными параметрами функций и функциональных блоков (рис. Г.4). Входные переменные могут быть соединены только с входными параметрами функции или функционального блока, выходные переменные – только с выходными параметрами функции или функционального блока, входные/выходные переменные – как входами, так и с выходами функции или функционального блока. Также выходной параметр одной функции или функционального блока может быть напрямую соединён с входным параметром другого.

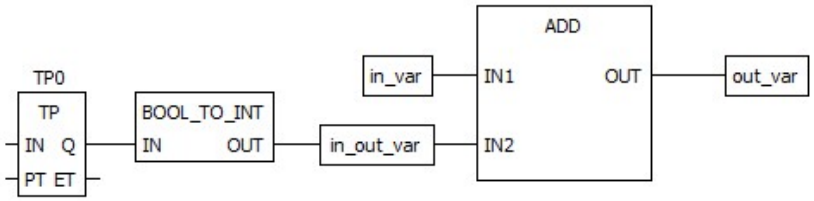


Рисунок Г.4 – Пример соединения переменных, функций и функциональных блоков

Все функциональные блоки могут быть вызваны с дополнительными (необязательными) формальными параметрами: EN (входом) и ENO (выходом). Пример такого функционального блока приведен на рис. Г.5.

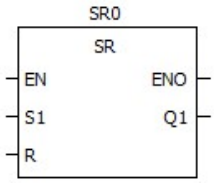


Рисунок Г.5 – Изображение элементарного функционального блока с параметрами EN/ENO

Если функциональный блок вызывается с параметрами EN/ENO и при этом значение EN равно нулю, то алгоритмы, определяемые в функциональном блоке, не будут выполняться. В этом случае значение ENO автоматически устанавливается равным 0. Если же значение EN равно 1, то алгоритмы, определяемые функциональным блоком, будут выполнены. После выполнения этих алгоритмов без ошибок значение ENO автоматически устанавливается равным 1. Если же возникает ошибка во время выполнения этих алгоритмов, то значение ENO будет установлено равным 0. Поведение функционального блока одинаково как в случае вызова функционального блока с EN = 1, так и при вызове без параметров EN/ENO.

Для более компактного соединения входов и выходов различных функций и функциональных блоков используются элементы «Соединение», показанные на рис. Г.6.

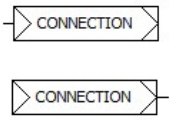


Рисунок Г.6 – Изображение соединений в языке FBD

Они бывают двух видов: входное соединение и выходное выходные соединение. Основная задача соединений – передать значение из одного выхода на другой вход без прямого соединения выхода и входа. На рис. Г.7 показан пример, в котором выходное значение OUT функции BOOL_TO_INT передаётся на вход IN2 функции ADD.

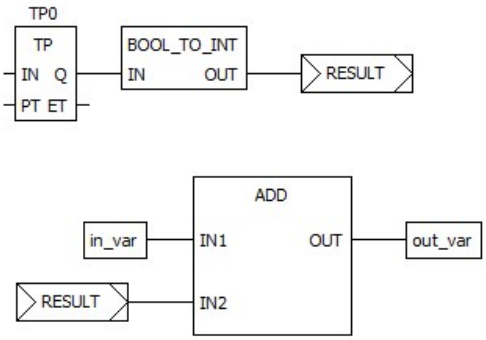


Рисунок Г.7 – Пример использования соединения на FBD диаграмме

На рис. Г.8 приведена FBD диаграмма, состоящая из следующих функциональных блоков: SR0, AND, TP0.

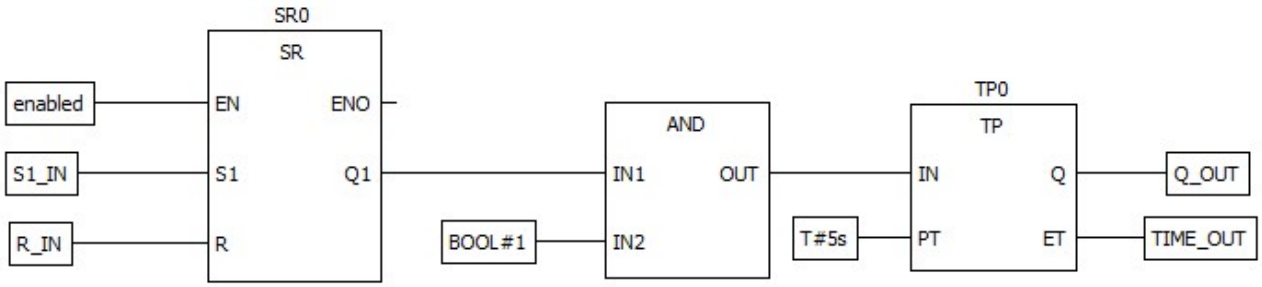


Рисунок Г.8 – Пример FBD диаграммы

Функциональный блок SR0 представляет собой Бистабильный SR-триггер. У него имеются входы S1, R1 и выход Q1, а так же дополнительный вход EN и выход ENO, позволяющие включать и выключать выполнение SR0. Выход Q1 с помощью соединён с входом IN1 блока AND, представляющий собой «Логическое И». Вход IN2 типа BOOL соединён с литералом «BOOL#1», который всегда положительный. Выход OUT блока AND соединён с входом IN функционального блока TP0, представляющий собой повторитель импульсов. Вход PT типа TIME, соединён с литералом «T#5s», который задаёт значение 5 секунд.

Если после запуска выполнения данного функционального блока enabled равно True и переменная S1_IN тоже True, функциональный блок SR0 начинает выполняться. На выходе OUT функционального блока AND будет значение True как только Q1 у SR0 будет равен True. Следовательно, как только OUT становится True вход IN функционального блока TP0 принимает тоже True и начинается отсчёт таймера ET (рис. Г.9).

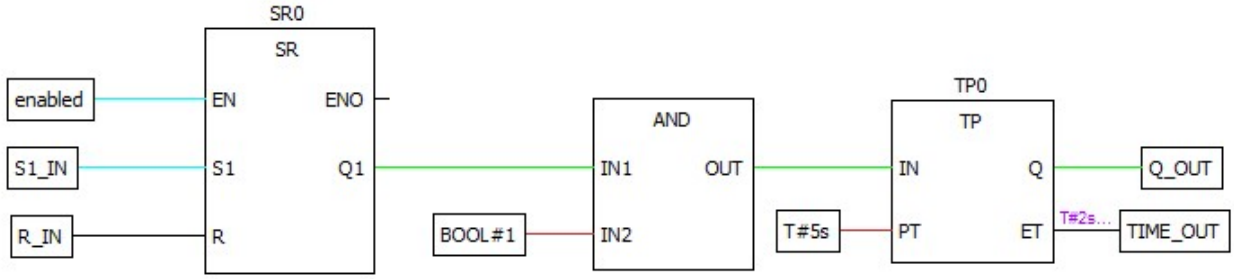


Рисунок Г.9 – Выполнение FBD диаграммы

Пока данный таймер не достигнет значения PT выход Q у функционального блока TP0 будет равен True. При достижении таймером ET значения PT, т.е. через 5 секунд выход Q становится False (рис. Г.10).

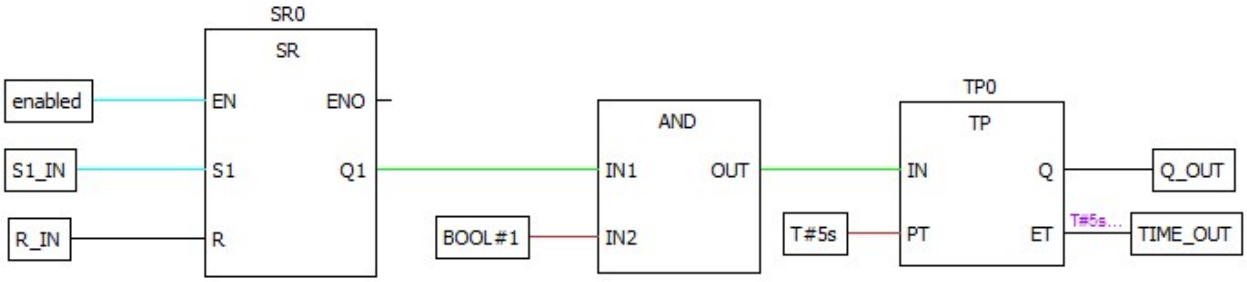


Рисунок Г.10 – Выполнение FBD диаграммы (продолжение)

Как только вход IN функционального блока TP0 становится значения FALSE, счётчик ET сбрасывается в T#0s.

ПРИЛОЖЕНИЕ Д Описание языка LD

1 Общие сведения о языке LD

LD (Ladder Diagram) – графический язык, основанный на принципах релейно-контактных схем (элементами релейно-контактной логики являются: контакты, обмотки реле, вертикальные и горизонтальные перемычки и др.) с возможностью использования большого количества различных функциональных блоков. Достоинствами языка LD являются: представление программы в виде электрического потока (близко специалистам по электротехнике), наличие простых правил, использование только булевых выражений. На рис. Д.1 приведён пример программы на языке LD (слева) и ее эквивалент в виде электрической цепи с реле и выключателями (справа).

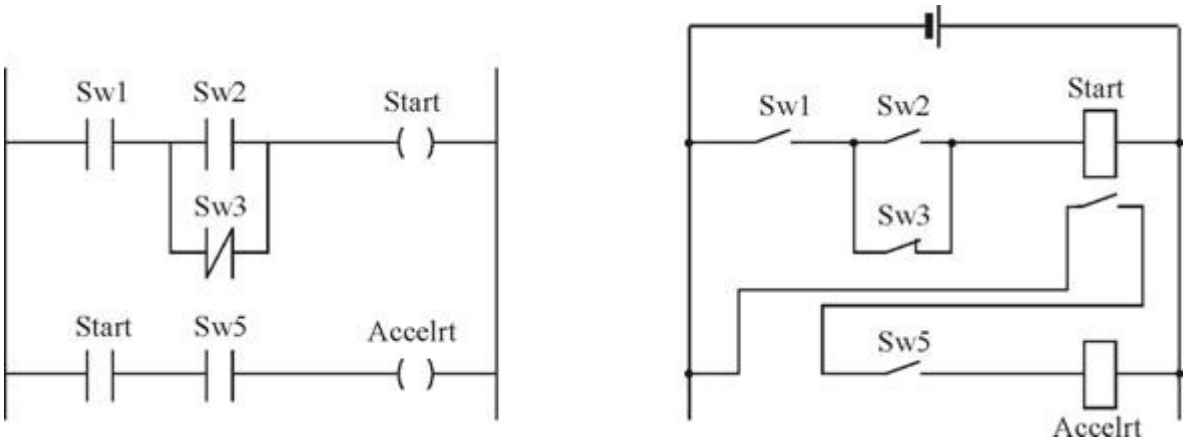


Рисунок Д.1 – Программа на языке LD (слева) и ее эквивалент в виде электрической (справа)

Схемы, реализованные на данном языке, называются многоступенчатыми. Они представляют собой набор горизонтальных цепей, напоминающих ступеньки лестницы, соединяющих вертикальные шины питания.

Объекты языка программирования LD обеспечивают средства для структурирования программного модуля в некоторое количество контактов, катушек. Эти объекты взаимосвязаны через фактические параметры или связи.

Порядок обработки индивидуальных объектов в LD-секции определяется потоком данных внутри секции. Ступени, подключённые к левой шине питания, обрабатываются сверху вниз (соединение к левой шине питания). Ступени внутри секции, которые не зависят друг от друга, обрабатываются в порядке размещения.

2 Основные конструкции языка

Слева и справа схема на языке LD ограничена вертикальными линиями – шинами питания. Между ними расположены цепи, образованные контактами и катушками реле, по аналогии с обычными электронными цепями. Слева любая цепь начинается набором контактов, которые посылают слева направо состояние «ON» или «OFF», соответствующие логическим значениям TRUE или FALSE. Каждому контакту соответствует логическая переменная (типа BOOL). Если переменная имеет значение TRUE, то состояние передается через контакт. Иначе – правое соединение получает значение выключено ("OFF").

Контакты могут быть соединены параллельно, тогда соединение передаёт состояние «логическое ИЛИ». Если контакты соединены последовательно, то соединение передаёт «логическое И».

Контакт может быть инвертируемым. Такой контакт обозначается с помощью символа |/| и передаёт состояние "ON", если значение переменной FALSE.

Язык LD позволяет:

- выполнять последовательное соединение контактов;
- выполнять параллельное соединение контактов;
- применять нормально разомкнутые или замкнутые контакты;
- использовать переключаемые контакты;
- записывать комментарии;
- включать Set/Reset-выходы (Установка/Сброс);
- переходы;
- включать в диаграмму функциональные блоки;
- управлять работой блоков по входам EN.

2.1 Контакт

Контактом является LD-элемент, который передаёт состояние горизонтальной связи левой стороны горизонтальной связи на правой стороне. Это состояние – результат булевой AND- операции состояния горизонтальной связи с левой стороны с состоянием ассоциированной переменной или прямого адреса. Контакт не изменяет значения связанной переменой или прямого адреса.

Для нормальных контактов (рис. Д.2) состояние левой связи передается в правую связь, если состояние связанного логического фактического параметра TRUE. Иначе, состояние правой связи FALSE.

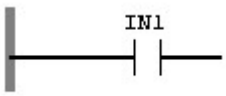


Рисунок Д.2 – Нормальный контакт

Для инверсных контактов (рис. Д.3) состояние левой связи передается в правую связь, если состояние связанного логического фактического параметра FALSE. Иначе, состояние правой связи TRUE.

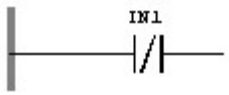


Рисунок Д.3 – Инверсный контакт

В контактах для обнаружения нарастания фронта (рис. Д.4) правая связь устанавливается в состояние TRUE, если переход связанного фактического параметра происходит из FALSE в TRUE, и в то же время состояние левой связи TRUE. Иначе, состояние правой связи FALSE.

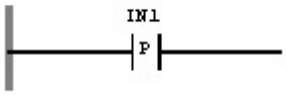


Рисунок Д.4 – Контакт для обнаружения нарастания фронта

В контактах для обнаружения спада фронта (рис. Д.5) правая связь устанавливается в состояние TRUE, если переход связанного фактического параметра происходит из True в False, и состояние левой связи True в то же время. Иначе, состояние правой связи FALSE.

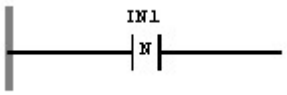


Рисунок Д.5 – Контакт для обнаружения спада фронта

2.2 Катушка

Катушка является LD-элементом, который передаёт состояние горизонтальной связи на левой стороне неизменяемым горизонтальной связи на правой стороне. В этом процессе состояние связанной переменной или прямого адреса будет сохранено.

В нормальных катушках (рис. Д.6) состояние левой связи передается в связанный логический фактический параметр и в правую связь.

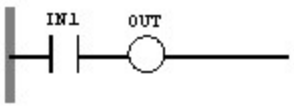


Рисунок Д.6 – Нормальная катушка

В инвертирующей катушке (рис. Д.7) состояние левой связи копируется в правую связь. Инвертированное состояние левой связи копируется в связанный логический фактический параметр. Если связь находится в состоянии FALSE, тогда правая связь тоже будет находиться в состоянии FALSE, и связанный логический фактический параметр будет находиться в состоянии TRUE.

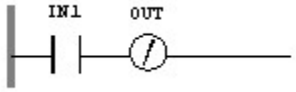


Рисунок Д.7 – Инвертирующая катушка

В катушке установки (рис. Д.8) состояние левой связи копируется в правую связь. Связанный логический фактический параметр устанавливается в состояние TRUE, если левая связь имеет состояние TRUE, иначе он не изменяется. Связанный логический фактический параметр может сбрасываться только катушкой сброса.

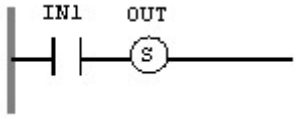


Рисунок Д.8 – Катушка установки

В катушке сброса (рис. Д.9) состояние левой связи копируется в правую связь. Связанный логический фактический параметр устанавливается в состояние FALSE, если левая связь имеет состояние TRUE, иначе он не изменяется. Связанный логический фактический параметр может устанавливаться только катушкой установки.

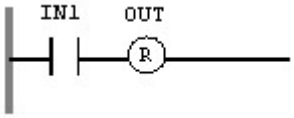


Рисунок Д.9 – Катушка сброса

В катушке обнаружения нарастания фронта (рис. Д.10) состояние левой связи копируется в правую связь. Связанный фактический параметр типа данных BOOL будет установлен в состояние TRUE для цикла программы, если произошел переход левой связи из FALSE в TRUE.

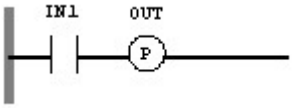


Рисунок Д.10 – Катушка обнаружения нарастания фронта

В катушке обнаружения спада фронта (рис. Д.11) состояние левой связи копируется в правую связь. Связанный фактический параметр типа данных BOOL будет установлен в состояние TRUE для цикла программы, если произошел переход левой связи из TRUE в FALSE.

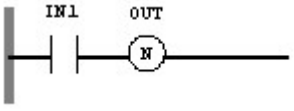


Рисунок Д.11 – Катушка обнаружения спада фронта

Слово «катушка» имеет обобщенный образ исполнительного устройства, поэтому в русскоязычной документации обычно говорят о выходе цепочки, хотя можно встретить и частные значения термина, например катушка реле.

2.3 Шина питания

Левая шина питания соответствует единичному сигналу. Ступени, подключённые к левой шине питания, обрабатываются сверху вниз (соединение к левой шине питания).

3 Пример программы на языке LD

Пример представляет собой реализацию логического выражения: C = A AND NOT B.

При создании LD диаграмм можно использовать только переменные типа BOOL. Добавим новый контакт и привяжем его к имени A (имени переменной). Далее добавляется шина питания слева, шина питания справа, нормальный контакт, инверсный контакт и нормальная катушка. Нормальный контакт ассоциируется с переменной A, инверсный контакт с переменой B, нормальная катушка с переменной C. Далее это всё последовательно соединяется (рис. Д.12), и результатом является программа, написанная на языке LD, реализующая логическое выражение: C = A AND NOT B.

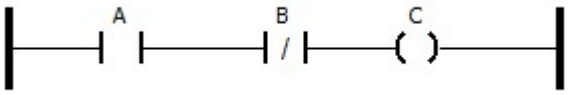


Рисунок Д.12 – Пример LD диаграммы, реализующей логическое выражение C = A AND NOT B

ПРИЛОЖЕНИЕ Е Описание языка SFC

1 Общие сведения о языке SFC

SFC (Sequential Function Chart) расшифровывается как «Последовательность функциональных диаграмм», и является одним из языков стандарта IEC 61131-3. SFC позволяет легко описывать последовательность протекания процессов в системе.

SFC осуществляет последовательное управление процессом, базируясь на системе условий, передающих управления с одной операции на другую. Язык SFC состоит из конечного числа базовых элементов, которые используются как блоки для построения целостного алгоритма протекания программы.

2 Основные понятия языка SFC

Язык SFC использует следующие структурные элементы для создания программы: шаг (и начальный шаг), переход, блок действий, прыжок и связи типа дивергенция и конвергенция.

После вызова программного модуля, описанного языком SFC, первым выполняется начальный шаг. Шаг, выполняемый в данный момент, называется активным. Действия, связанные с активным шагом, выполняются один раз в каждом управляющем цикле. В режиме выполнения активные шаги выделяются салатовым цветом. Следующий за активным шагом шаг станет активным, только если в переходе между этими шагами условие будет истинно.

В каждом управляющем цикле будут выполнены действия, содержащиеся в активных шагах. Далее проверяются условия перехода, и, возможно, уже другие шаги становятся активными, но выполняться они будут уже в следующем цикле.

Далее описывается каждый элемент SFC диаграммы.

2.1 Шаг

Наиболее важным элементом языка SFC является шаг, который описывает одну операцию. Шаг изображается в виде прямоугольника с собственным именем внутри (рис. Е.1).

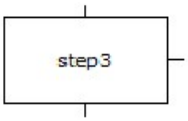


Рисунок Е.1 – Графическое представление «Шага» языка SFC

У каждого шага может быть 3 контакта. Сверху и снизу для соединения с переходом и справа для соединения с блоком действий. Шаг предваряется переходом, который определяет условие для активации данного шага в процессе выполнения программы и отображается в виде горизонтальной черты на ветви диаграммы процесса с указанием имени и условия. Два шага никогда не могут быть соединены непосредственно, они должны всегда отделяться переходом (рис. Е.2).

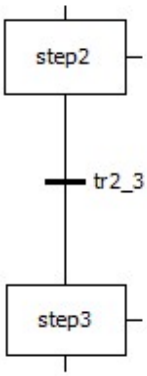


Рисунок Е.2 – Шаги «step2» и «step3», соединённые переходом «tr2_3»

Любая SFC диаграмма должна содержать начальный шаг (шаг, выделенный двойной рамкой), с которого начинается выполнение диаграммы.

2.2 Переход

Между шагами находятся так называемые переходы. Условием перехода может быть логическая переменная или константа, логический адрес или логическое выражение, описанное на любом языке. Условие может включать серию инструкций, образующих логический результат, в виде ST выражения, например:

```
(i<= 100) AND b
```

либо на любом другом языке.

На рис. Е.3 приведён пример перехода между шагом «Step3» и «Step5» с именем «transition4».

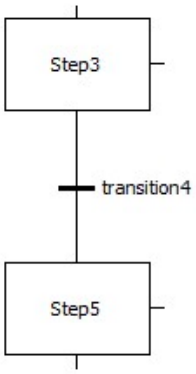


Рисунок Е.3 – Переход между шагами «Step3» и «Step5» с предопределённым условием «transition4»

В данном случае «transition4» это имя для предопределённого перехода, который может использоваться многократно на SFC диаграмме для определения переходов между несколькими шагами. Код для него может быть представлен, например, на языке ST:

```
:= (flag = True AND level > 10);
```

На рис. Е.4 представлен переход между шагами «Step6» и «Step7» в виде обычного условия: level > 10

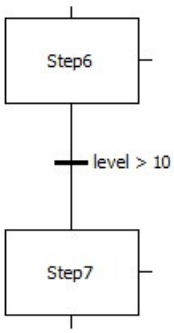


Рисунок Е.4 – Переход между шагами «Step6» и «Step7» с предопределённым условием «level > 10»

На рис. Е.5 представлен переход между шагами «Step8» и «Step9» в виде значения логического выражения «AND» на языке FBD:

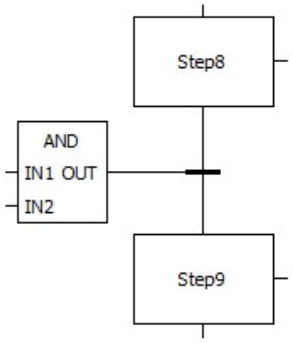


Рисунок Е.5 – Переход между шагами «step8» и «step9», заданный «логическим И» на языке FBD

Условие не должно содержать присваивания, вызов программ и экземпляров функциональных блоков.

2.3 Блок действий

Каждый шаг имеет нулевое или большее количеством действий, объединённых, как правило, на диаграмме, в блок действий. На рис. Е.6 показан примера шага «evaluateStep» и связанный с ним блок действий.

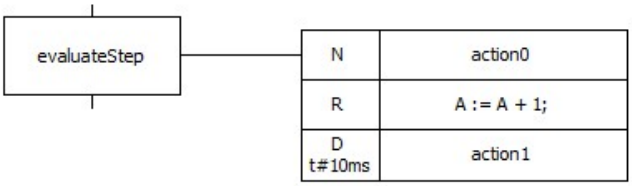


Рисунок Е.6 – Шаг «evaluateStep» и связанный с ним блок действий, содержащий 3 действия

Блок действий определяет операции, которые должны выполняться при активации (выполнении) шага. Шаги без связанного блока действий идентифицируются как ждущий

шаг. Блок действий может состоять из предопределённых действий. Каждому предопределённому действию присваивается имя (на рис. Е.6 это «action0» и «action1»). Одно действие может использоваться сразу в нескольких шагах. Действие может выполняться непрерывно, пока активен шаг, либо единожды. Это определяется специальными квалификаторами, описание которых приведено в таблице 13. Квалификаторы также могут ограничивать время выполнения каждого действия в шаге.

2.4 Безусловный переход (прыжок)

Шаг может быть также заменён безусловным переходом. Последовательности шагов всегда ассоциируются с прыжком к другому шагу той же самой последовательности шагов. Это означает, что они выполняются циклически. Переход на произвольный шаг – это соединение на шаг, имя которого указано под знаком безусловного перехода. Такие переходы нужны для того, чтобы избежать пересекающихся и идущих вверх соединений. На рис. Е.7 показана SFC диаграмма, содержащая два безусловных перехода.

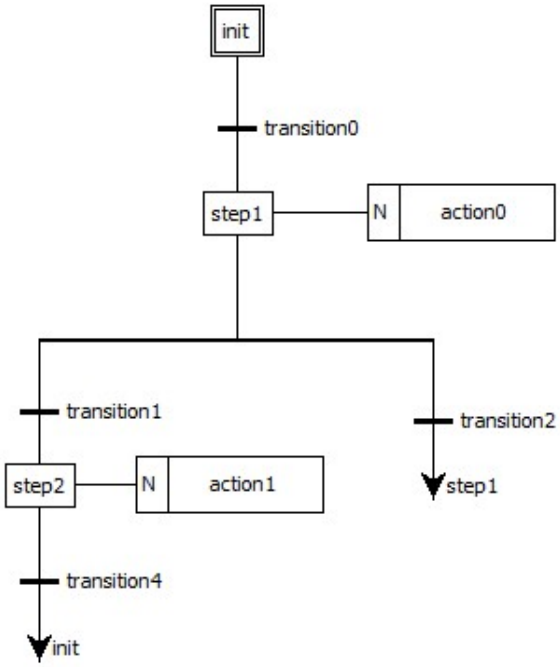


Рисунок Е.7 – SFC диаграмма, содержащая безусловные переходы

Первый делает переход к шагу «init» в случае выполнения условия «transition4», второй делает переход к шагу «step1», в случае выполнения условия «transition2».

2.5 Ветвление (дивергенция) и объединение (конвергенция)

Ветвление (дивергенция) – это множественное соединение в направлении от одного шага к нескольким переходам. Активируется только одна из ветвей. Условия, связанные с различными переходами в начале ветвления, не являются взаимоисключающими по умолчанию. Взаимоисключение должно быть явно задано в условии перехода, чтобы гарантировать, что во время выполнения программы активируется одна конкретная ветвь. Пример дивергенции на SFC диаграмме приведён на рис. Е.8 и выделен красным цветом:

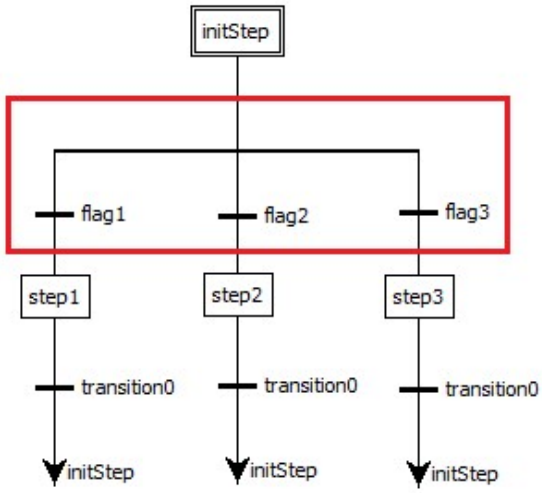


Рисунок Е.8 – Ветвление на SFC диаграмме

Объединение (конвергенция) – это множественное соединение, направленное от нескольких переходов к одному и тому же шагу. Она обычно используется для группировки ветвей SFC – программы, которые берут начало из одинарной дивергенции. Пример конвергенции на SFC диаграмме приведён на рис. Е.9 и выделен красным цветом:

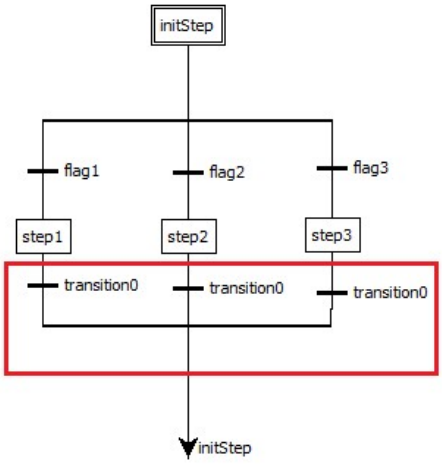


Рисунок Е.9 – Объединение на SFC диаграмме

Параллельное ветвление – это множественное соединение, направленное от одного перехода к нескольким шагам. Оно соответствует параллельному выполнению операций процесса. Пример параллельного ветвления на SFC диаграмме приведён на рис. Е.10 и выделен красным цветом:

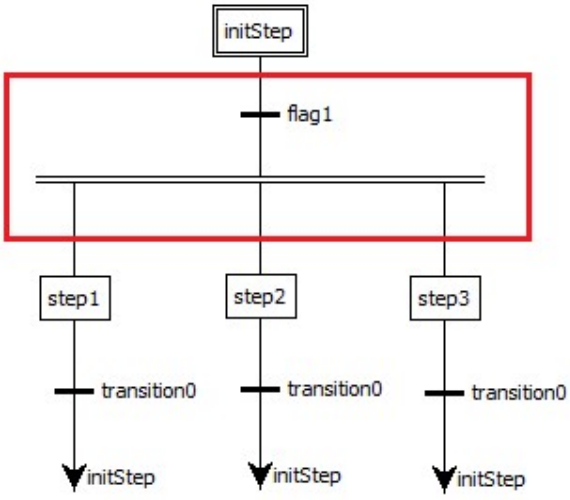


Рисунок Е.10 – Параллельное ветвление на SFC диаграмме

Параллельная конвергенция – это соединение нескольких шагов к одному и тому же переходу. Обычно она используется для группирования ветвей, взявших начало дивергенции. Пример параллельной конвергенции на SFC диаграмме приведён на рис. Е.11 и выделен красным цветом:

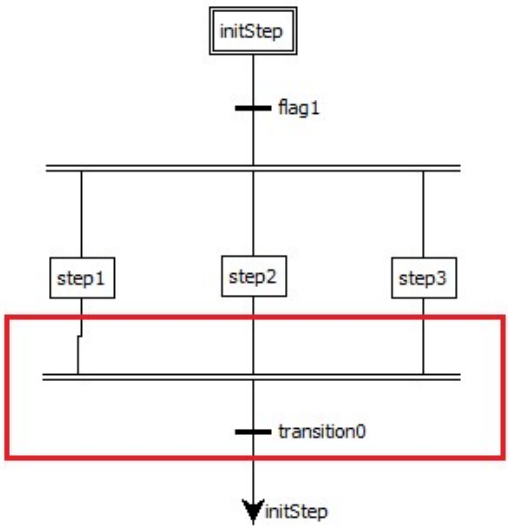


Рисунок Е.11 – Параллельное объединение на SFC диаграмме

3 Пример программы на языке SFC

На рис. Е.12 приведен пример SFC диаграммы состоящей из начального шага «initStep», шагов «firstStep» и «secondStep» и 3 перехода.

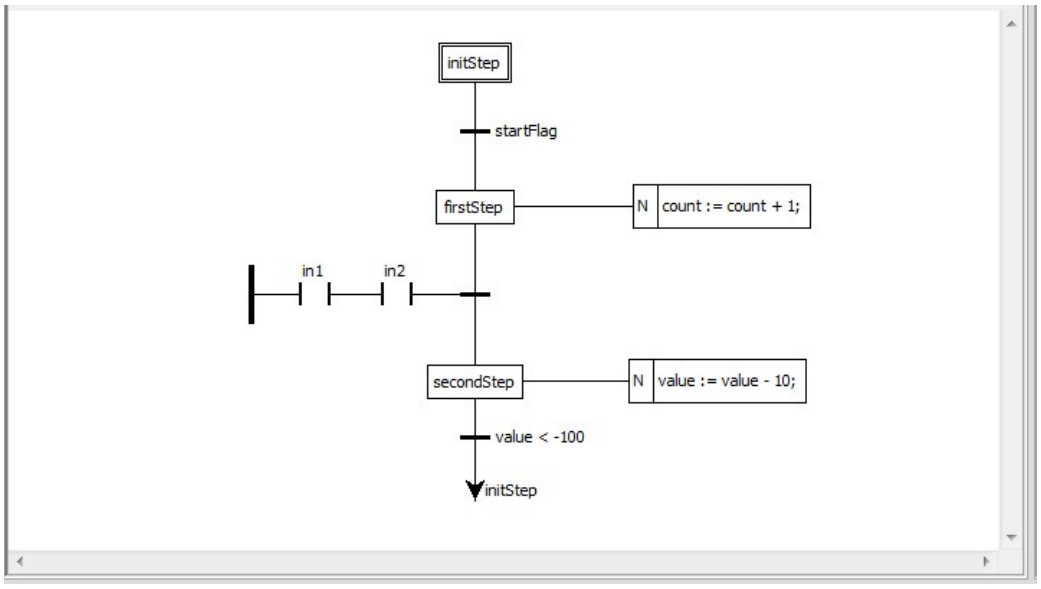


Рисунок Е.12 – SFC диаграмма

Переход «startFlag» представляет обычную переменную типа BOOL и полностью зависит от её значения. Переход между «firstStep» и «secondStep» зависит от LD диаграммы с двумя катушками, ассоциированными с переменными типа BOOL: «in1» и «in2». Переход активируется только в том случае, если «in1» и «in2» будут TRUE. Переход между «secondStep» и прыжком на «initStep» активирован, когда значение переменной «value» меньше –100. Во время действия «firstStep» выполняется увеличение переменной count на 1. Во время действия «secondStep» из переменной «value» вычитается 10.

Перечень принятых сокращений

ПЛК – Контроллер программируемый логический
РП – руководство программиста
ST – Structured Text
IL – Instruction List
FBD – Function Block Diagram
LD – Ladder Diagram
SFC – Sequential Function Chart

Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ докумен-та	Входящий № сопроводительного документа и дата	Подпись	Дата
	Измененных	Замененных	Новых	Аннулированных					