ООО «Научно-техническая компания ПРИБОРЭНЕРГО»

Руководство по программированию
РП.ПЛК.01.

Среда разработки Beremiz

Программные средства поддержки языков программирования высокого уровня стандарта IEC 61131-3

Версия 1.0
Чебоксары, 2023 г.

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном руководстве программиста представлено описание порядка работы со средой разработки Beremiz. Документ содержит информацию о назначении программы, условиях выполнения, элементах пользовательского интерфейса, порядке разработки прикладных программ, работе с внешними модулями – плагинами. Рассмотрены основные её компоненты и их назначение с приведёнными примерами. Описан процесс работы с редакторами языков стандарта IEC 61131-3 и режимом отладки созданных прикладных программ. В документе приведены тексты сообщений, выдаваемых в ходе выполнения программы и описание их содержания.

В приложениях приведены: порядок установки среды Beremiz под операционную систему Windows, описание стандартных функциональных блоков и общие сведения о языках стандарта IEC 61131-3.

1 НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ СРЕДЫ РАЗРАБОТКИ

Среда разработки Beremiz предназначена для создания и отладки прикладных программ на языках стандарта IEC 61131-3. В качестве языков описания алгоритмов и логики работы данных программ, могут выступать как текстовые Structured Text (ST) и Instruction List (IL), так и графические Function Block Diagram (FBD), Ladder Diagram (LD), Sequential Function Chart (SFC).

Техническими требованиями для работы среды разработки Beremiz является персональный компьютер с тактовой частотой процессора от 1000 МГц (поддерживаются как 32-битные, так и 64- битные), 1 Гб оперативной памяти и установленная операционная система Windows XP/Vista/7/8/10. Необходимо наличие монитора (для оптимальной работы не меньше 17”), клавиатуры, мыши (или устройства, полноценно заменяющего мышь).

Среда разработки Beremiz может выполняться на различных операционных системах: Windows, Linux. Она написана с использованием кроссплатформенных языков Python, C, C++ и дополнительных библиотек к ним. Для запуска и работы Beremiz, необходим интерпретатор Python с определённым набором установленных пакетов (библиотек) – поставляется в составе дистрибутива для операционной системы Windows.

Инструкция по установке для операционной системы Windows XP представлена в Приложении 1.

2 ХАРАКТЕРИСТИКИ СРЕДЫ РАЗРАБОТКИ

Среда разработки Beremiz позволяет работать в конфигурационном режиме и в режиме исполнения прикладной программы.

В конфигурационном режиме происходит создание прикладной программы, написание алгоритмов и логики её основных программных модулей и их связывание с внешними модулями – устройствами связи с объектами (УСО).

В режиме исполнения прикладная программа передаётся на целевое устройство и может быть запущена с режимом отладки и без отладки.

Контроль правильности выполнения осуществляется средствами среды:

- редакторами языков IEC 61131-3;
- компиляторами языков IEC 61131-3;
- компилятором GCC для целевой платформы.

Соответствующие предупреждения и ошибки выводятся либо в виде сообщения в диалоге, либо в виде текстовой информации в отладочную консоль.

3 ОБОБЩЕННАЯ СХЕМА РАБОТЫ

Общая схема по созданию прикладной программы в среде разработки Beremiz представлена на рисунке 1.

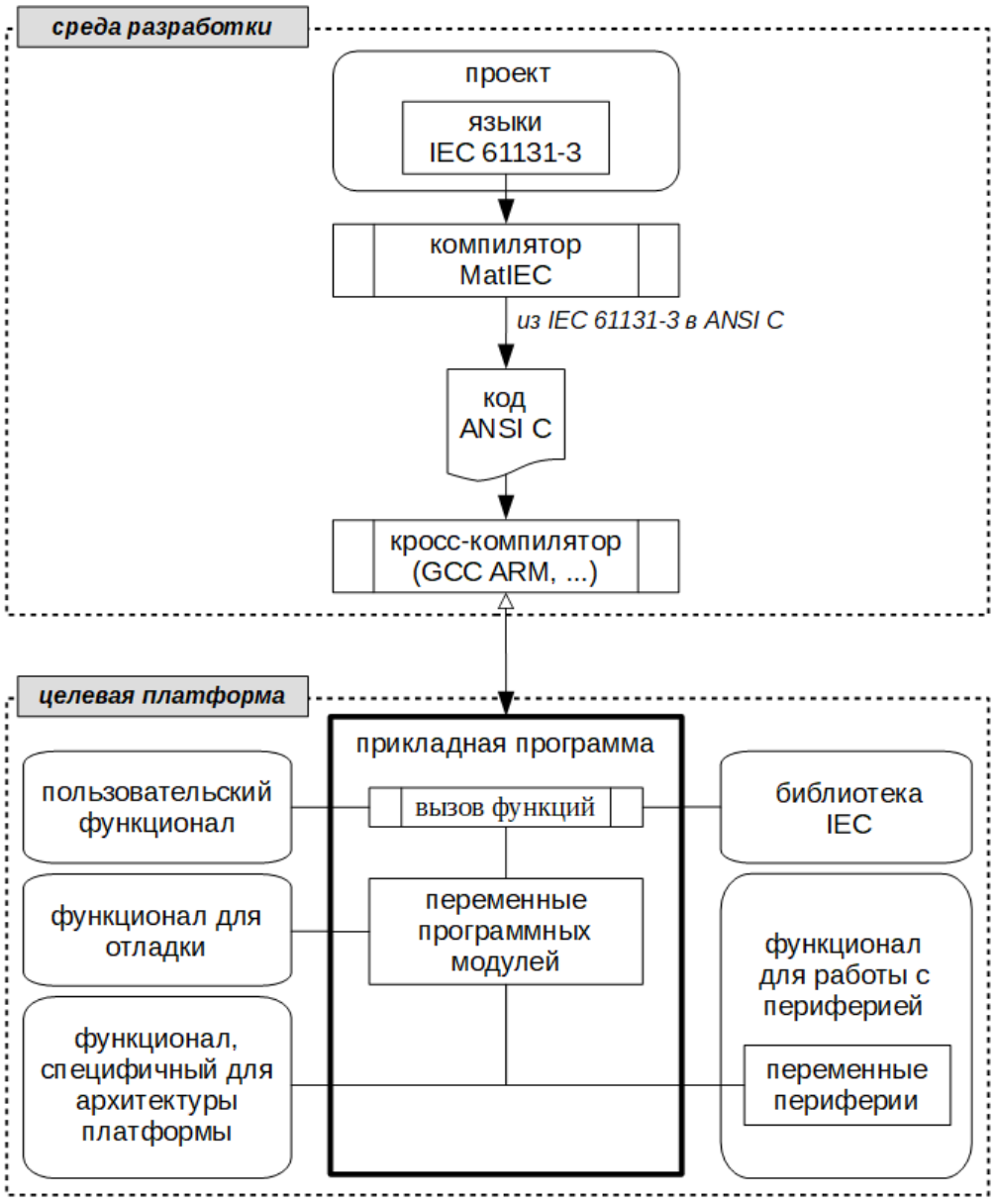


Рисунок 1 - Обобщенная схема работы

Входными данными являются программные модули, написанные пользователем (в большинстве случаев инженером по автоматизации) на текстовых (ST, IL) и/или графических (FBD, SFC, LD) языках в соответствии со стандартом IEC 61131-3, объединённые в проект.

Каждый такой проект представлен в формате XML и хранится в отдельной папке.

Выходными данными является сгенерированный исходный код и исполняемый файл:

1) файл <название проекта>.st, содержащий промежуточный код на языке ST, сгенерированный для всех программных модулей и ресурсов, транслируемый в язык C;
2) файлы: config.c config.h, POUS.h, POUS.c и файлы, соответствующие ресурсам – содержат код (на языке C) реализации алгоритмов и логики работы программных модулей и ресурсов проекта;
3) файлы plc_common_main.c и plc_debugger.c содержат код специфичный для целевой архитектуры и код для отладки прикладной программы на целевом устройстве из среды разработки Beremiz соответственно;
4) файлы, содержащие код драйверов на языке C для взаимодействия с внешними модулями;
5) исполняемый файл (*.bin, *.hex, *.so), компилируемый из этих вышеперечисленных C файлов.

Исполняемый файл, благодаря средствам Beremiz, может быть размещен на целевом устройстве различными коммуникационными средствами (например, загрузка через COM-порт – RS-232, RS-485).

На целевом устройстве исполняемый файл запускается и в процессе работы выполняет следующие действия:

- с помощью драйверов модулей УСО обменивается данными с внешними модулями;
- исполняет алгоритмы и логику, определенную пользователем в программных модулях проекта;
- предоставляет данные для трансляции в системы верхнего уровня;
- сохраняет и транслирует информацию для отладки прикладных программ.

4 ОСНОВНЫЕ ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

IEC 61131-3 – раздел международного стандарта МЭК 61131 (также существует соответствующий европейский стандарт EN 61131), описывающий языки программирования для программируемых логических контроллеров.

Среда разработки для языков стандарта IEC 61131-3 – система программных средств, используемая инженерами по автоматизации, для разработки прикладного программного обеспечения на высокоуровневых языках стандарта IEC 61131-3 под различные целевые платформы, которая включает в себя:

- текстовые и графические редакторы языков стандарта IEC 61131-3;
- транслятор диаграмм графических языков в текстовый язык;
- транслятор текстового языка в язык C;
- механизмы плагинов для взаимодействия с модулями УСО;
- механизмы добавления компиляторов под целевую платформу;
- механизмы соединений с целевыми устройствами;
- отладчик.

Модули УСО – модули ввода/вывода, обеспечивающие подключение датчиков и исполнительных механизмов.

Целевое устройство – аппаратное средство с определённой архитектурой процессора, на котором могут исполняться различные исполняемые файлы, обращающиеся с помощью него к модулям УСО.

Прикладная программа (исполняемый файл) для целевого устройства – скомпилированный и скомпонованный файл (*.bin, *.hex, *.so), который будет выполняться на целевом устройстве.

Плагин для модуля УСО – интерфейс, состоящий из специальных драйверов и элементов пользовательского интерфейса для среды разработки Beremiz, позволяющий связывать переменные модулей УСО с переменными программных модулей, из которых состоит проект.

Проект – совокупность программных модулей (программ, функциональных блоков, функций), плагинов внешних модулей УСО, ресурсов, пользовательских типов данных, сборка (компиляция и компоновка) которых, представляет собой прикладную программу для целевого устройства. Каждый проект сохраняется в отдельном файле.

Переменная – область памяти, в которой находятся данные, с которыми оперирует программный модуль.

Ресурс – элемент, отвечающий за конфигурацию проекта: глобальные переменные и экземпляры проекта, связываемыми с программными модулями типа «Программа» и задачами.

Программный модуль – элемент, представляющий собой функцию, функциональный блок или программу. Каждый программный модуль состоит из раздела объявлений и кода. Для написания всего кода программного используется только один из языков программирования стандарта IEC 61131-3.

Функция – программный модуль, который возвращает только единственное значение, которое может состоять из одного и нескольких элементов (если это битовое поле или структура).

Функциональный блок – программный модуль, который принимает и возвращает произвольное число значений, а так же позволяет сохранять своё состояние (подобно классу в различных объектно-ориентированных языках). В отличие от функции функциональный блок не формирует возвращаемое значение.

Программа – программный модуль, представляющий собой единицу исполнения, как правило, связывается (ассоциируется) с задачей.

Задача – элемент представляющий время и приоритет выполнения программного модуля типа «Программа» в рамках экземпляра проекта.

Экземпляр – представляет собой программу, как единицу исполнения, связанную (ассоциированную) с определённой задачей. Так же, как экземпляр, рассматриваются переменные, определённые в программных модулях: программа и функциональный блок.

Пользовательский тип данных – тип данных, добавленный в проект и представляющий собой: псевдоним существующего типа, поддиапазон существующего типа, перечисление, массив или структуру.

POU (Program Organization Unit) – программный компонент, к которому относятся: функции, функциональные блоки, программы.

5 ОСНОВНЫЕ КОМПОНЕНТЫ СРЕДЫ РАЗРАБОТКИ

Пользовательский интерфейс среды разработки Beremiz состоит из следующих компонент:

- главное меню программы;
- панель инструментов;
- дерево проекта;
- панель списка переменных и констант;
- панель настроек проекта;
- панель файлов проекта;
- панель отображения промежуточного кода;
- текстовые редакторы языков ST и IL;
- графические редакторы языков FBD, SFC, LD;
- панель редактирования ресурса;
- панель экземпляров проекта;
- панель библиотеки функций и функциональных блоков;
- отладочная консоль;
- поиск элементов в проекте;
- панель отладки;
- панель графика изменения значения переменной в режиме отладки.

Далее подробно рассказано про каждый компонент среды разработки Beremiz в отдельности.

5.1 Главное меню

Главное меню программы (рисунок 2) содержит следующие пункты:

- Файл;
- Редактировать;
- Вид;
- Помощь.

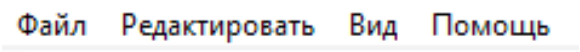


Рисунок 2 – Главное меню

Часть операций, выполняемых с помощью выбора определённого пункта меню мышью, могут быть исполнены с помощью «горячей клавиши». Далее будет подробно описан каждый пункт меню и соответствующая ему (если определена) «горячая клавиша».

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

- Новый – создание нового проекта (CTRL + N);
- Открыть – открытие существующего проекта (CTRL + O);
- Недавние проекты – быстрое открытие одного из десяти последних, недавно редактированных проектов;
- Сохранить – сохранение текущего проекта пункт (CTRL + S);
- Сохранить как – сохранение текущего проекта в папку отличную от той, в которой он сохранён на данный момент (CTRL + SHIFT + S);
- Закрыть вкладку – закрытие активной вкладки (например, вкладки переменных плагина, конфигурации и т.д.) для открытого проекта (CTRL + W);
- Закрыть проект – закрыть текущий, открытый проект (CTRL + SHIFT + W);
- Настройки страницы – настройка параметров страницы для печати на принтере активной программы, представленной в виде диаграммы (CTRL + ALT + P);
- Просмотр – предпросмотр перед печатью на принтере активной программы
- (CTRL + SHIFT + P);
- Печать – печать на принтере активной программы (CTRL + P);
- Выход – закрытие текущего проекта и выход из программы Beremiz (CTRL+ Q).

Меню «Редактировать» предназначено для работы с редакторами языков и предоставляет следующие возможности:

- Отменить – отмена последней манипуляции в редакторе (CTRL + Z);
- Повторить - повтор отменённой манипуляции в редакторе (CTRL + Y);
- Вырезать – удалить в буфер обмена выделенный(е) элемент(ы) в редакторе (CTRL
- + X);
- Копировать – копировать в буфер обмена выделенный(е) элемент(ы) в редакторе
- (CTRL + C);
- Вставить – вставить из буфера обмена находящиеся там элемент(ы) в редактор
- (CTRL + V);
- Поиск – вызов диалога поиска данных в редакторе (CTRL + F);
- Поиск следующего – поиск вперед в редакторе (CTRL + K);
- Поиск предыдущего – поиск назад в редакторе (CTRL + SHIFT + K);
- Поиск в проекте – вызов диалога поиска данных в проекте (CTRL + SHIFT + F);
- Добавить элемент – добавление одного из следующих элемента в текущий проект:
- Выделить все – выделение всех данных в открытой вкладке редактора;
- Удалить – удаление выделенных данных.

Меню «Вид» предназначено для работы с редакторами языков и предоставляет следующие возможности:

- Обновить – обновление данных и снятие выделения в редакторе (CTRL + R);
- Очистить ошибки – очистка указателей ошибок в редакторе (CTRL + K);
- Приближение – пункт, в котором можно выбрать в процентах величину масштаба;
- Сменить представление – скрывает/отображает панели среды разработки;
- Сбросить представление – восстановление расположения панелей в исходное состояние.

Меню «Помощь» предназначено для обращения к выводу информации в виде диалога о создателях данной среды – пункт «О программе».

5.2 Панель инструментов

Панель инструментов представляет собой панель с кнопками для быстрого обращения к часто используемым функциям среды разработки Beremiz. Она состоит из нескольких панелей, содержащих кнопки: управления проектом, сборки проекта и установки связи с целевым устройством. Подробнее об этих панелях рассказано ниже. При редактировании программных модулей, написанных на графических языках, появляются дополнительные панели с кнопками. Они рассмотрены при описании редакторов графических языков стандарта IEC 61131-3.

5.2.1 Кнопки управления проектом

Список кнопок панели инструментов (рисунок 3) и их функции приведены в таблице 1.

Рисунок 3 – Кнопки управления проектом

Таблица 1 - Функции кнопок управления проектом

Кнопка	Функция
	Новый проект
	Открыть проект
	Сохранить проект
	Сохранить проект как

Кнопка	Функция
	Печать
	Отменить изменение
	Повторить изменение
	Вырезать выделение
	Копировать выделение
	Вставить из буфера обмена
	Поиск в проекте

5.2.2 Кнопки сборки проекта и установления связи с целевым устройством

Список кнопок панели инструментов (рисунок 4) и их функции приведены в таблице 2.

Рисунок 4 – Кнопки сборки проекта и установления связи с целевым устройством

Таблица 2 - Функции кнопок сборки и установления связи с целевым устройством

Кнопка	Функция
	Собрать проект *компиляция и компоновка проекта в директории сборки «build»*
	Очистить директорию сборки
	Соединиться с целевым устройством *по адресу URI, заданному в настройках проекта*
	Отключиться от целевого устройства
	Показать сгенерированный код проекта в формате ST
	Загрузить программу в целевое устройство
	Запустить программу на целевом устройстве
	Остановить программу на целевом устройстве

В зависимости от того, произведено в настоящий момент времени соединение с целевым устройством или выполняется ли прикладная программа на нём, появляются и скрываются определенный набор кнопок данной панели.

5.3 Дерево проекта

Дерево проекта обычно расположено в левой части окна среды разработки Beremiz (рисунок 5) и отображает структуру проекта.

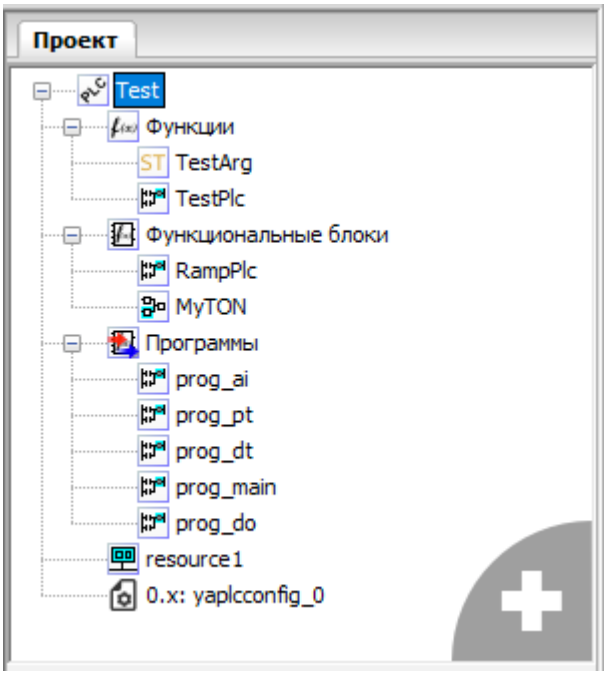


Рисунок 5 – Дерево проекта

В роли элементов могут выступать:

- программные модули (функции, функциональные блоки и программ) и их составные части;
- ресурсы;
- типы данных;
- плагины модулей УСО.

Дерево проекта позволяет добавлять, удалять элементы. Операции копирования и вставки только доступны для программных модулей.

5.3.1 Добавление элемента в дерево проекта

В правом нижнем углу дерева проекта находится кнопка «+», при нажатии на которую появляется меню для выбора добавляемого элемента в проект (рисунок 6).

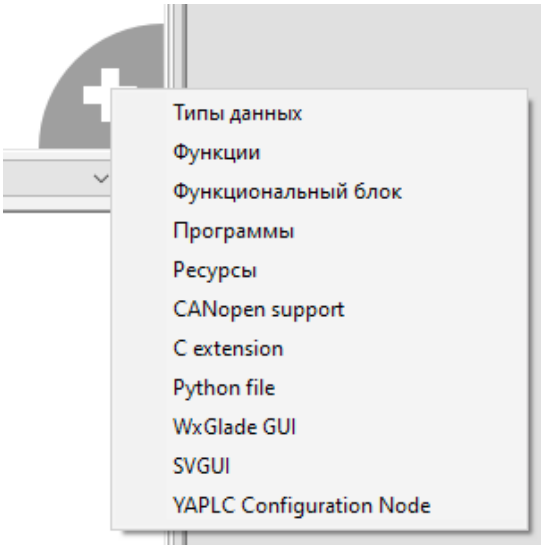


Рисунок 6 – Меню выбора добавляемого элемента в дерево проекта

Добавление нескольких элементов одного типа, например нескольких программ, функций, функциональных блоков приводит к их группировке в дереве проекта.

Еще одним способом добавления нового элемента в дерево проекта является нажатие правой клавиши мыши по определённому разделу в дереве проекта. Например, при нажатии на «Программы», появится всплывающее меню (рисунок 7), где можно выбрать: «Добавить POU» или «Вставить POU» (если он был скопирован в буфер обмена).

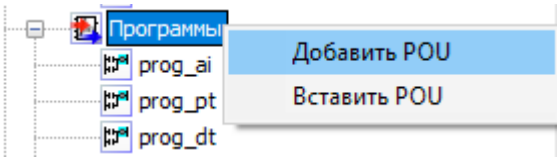


Рисунок 7 – Меню добавления программного компонента

Добавление нового элемента или выбор существующего в дереве проекта приводит к появлению панели редактирования и настроек соответствующего элемента. Каждая панель будет рассмотрена в последующих пунктах.

5.3.2 Копирование элемента в дерево проекта

Элемент дерева проекта (кроме ресурса) можно скопировать:

1) выделить элемент дерева проекта;
2) нажать на выделенном элементе правой клавишей мыши;
3) в появившемся контекстном меню выбрать «Копировать» (рисунок 8).

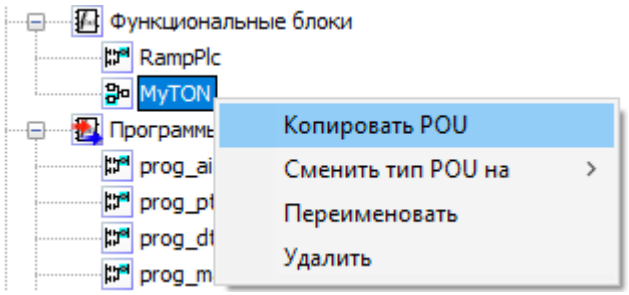


Рисунок 8 – Копирование элемента дерева проекта

Так выбранный элемент копируется в буфер обмена.

Вставить копию элемента из буфера обмена можно следующим образом:

1) на группе элементов дерева проекта нажать правой клавишей мыши;
2) в появившемся контекстном меню выбрать «Вставить» (рисунок 9).

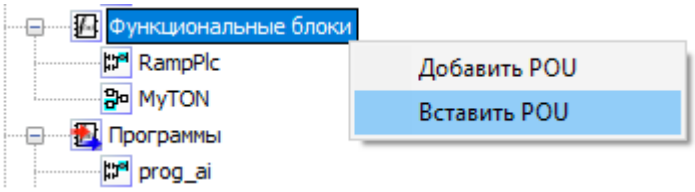


Рисунок 9 – Вставка скопированного элемента дерева проекта

Так скопированный элемент из буфера обмена будет вставлен в выбранную группу элементов дерева проекта. Копия элемента будет носить имя исходного элемента с добавлением окончания «1» (например, при копировании функционального блока «MyTON» его копия будет называться «MyTON1»).

5.3.3 Удаление элемента из дерева проекта

Удаление элемента из дерева проекта осуществляется следующим образом:

1) выделить элемент дерева проекта;
2) нажать на выделенном элементе правой клавишей мыши;
3) в появившемся контекстном меню выбрать «Удалить» (рисунок 10).

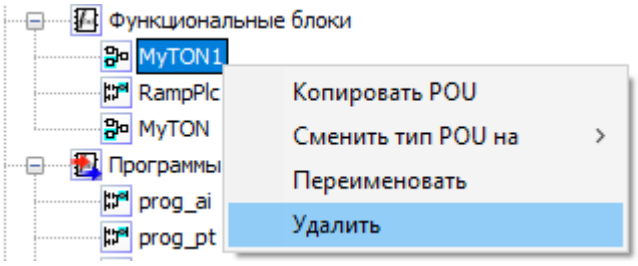


Рисунок 10 – Удаление элемента из дерева проекта

5.3.4 Переименование элемента в дереве проекта

Изменить имя элемента в дереве проекта можно следующим образом:

1) выделить элемент дерева проекта;
2) нажать на выделенном элементе правой клавишей мыши;
3) в появившемся контекстном меню выбрать «Переименовать»

ИЛИ

2) нажать на выделенном элементе левой клавишей мыши.

Режим переименования – когда вокруг имени элемента в дереве проекта появляется рамка, а внутри рамки мигающий курсор (рисунок 11).

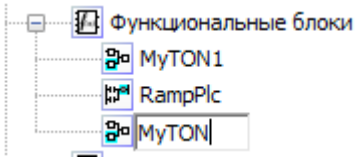


Рисунок 11 – Режим переименования элемента в дереве проекта

Выход из режима переименования осуществляется:

- нажатием клавиши «ENTER» клавиатуры (новое имя применяется)
- нажатием клавиши «ESC» клавиатуры (остается старое имя),
- нажатием клавиши мыши вне области выбранного элемента дерева проекта.

5.3.5 Изменение типа POU элемента в дереве проекта

Для функций и функциональных блоков имеется возможность изменения их типа представления.

Функцию можно привести к типу представления: «Функциональный блок», «Программа».

Функциональный блок можно привести к типу представления «Программа».

Изменение типа представления для данных элементов дерева проекта осуществляется следующим образом:

1) выбрать элемент функции или функционального блока в дереве проекта;
2) нажать на выбранном элементе правой клавишей мыши;
3) в появившемся контекстном меню выбрать «Сменить тип POU на»;
4) выбрать необходимый тип POU (рисунок 12).

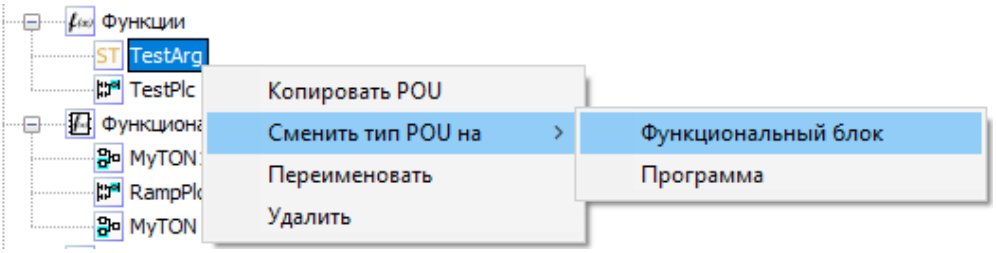


Рисунок 12 – Изменение типа POU элемента в дереве проекта

5.4 Панель списка констант и переменных

Панель списка констант и переменных входит в состав редакторов:

- конфигурационные переменные проекта (глобальные переменные),
- ресурсы (глобальные переменные),
- функции,
- функциональные блоки,
- программы

Данная панель находится в верхней части редактора и представляет собой таблицу (рисунок 13).

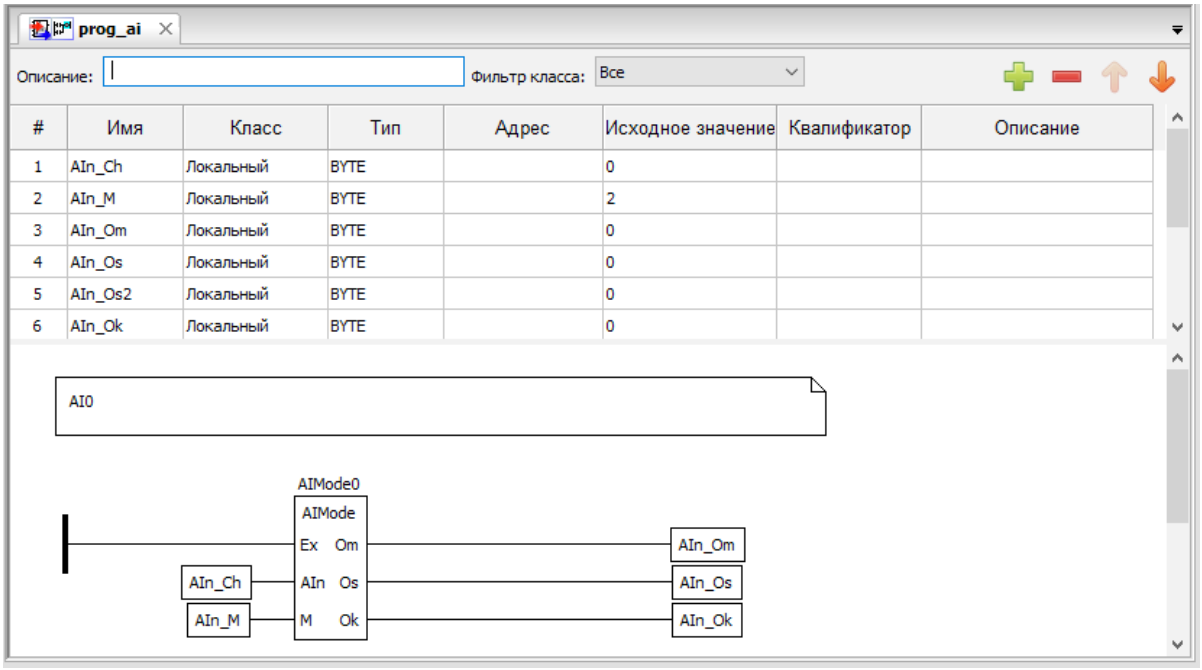


Рисунок 13 – Редактор программы с панелью констант и переменных

Каждая константа или переменная имеет следующие параметры:

- Имя - уникальный идентификатор в пределах её области видимости и действия;
- Класс:
  - Вход,
  - Выход,
  - Вход/Выход,
  - Внешний (глобальная переменная),
  - Локальный,
  - Временный;
- Тип – принадлежность к базовому типу (по стандарту IEC 61131-3), пользовательскому типу (псевдониму и поддиапазону существующего типа, перечислению, массиву, структуре) или типу функционального блока (стандартному или пользовательскому);
- Адрес - идентификатор, необходимый для связывания данной переменной с переменной (регистром) плагина модуля УСО;
- Исходное значение – инициализация переменной некоторым начальным значением;
- Квалификатор:
  - Константа,
  - Retain (сохраняемая в энергонезависимой памяти),
  - Не-Retain (не сохраняемая);
- Описание – комментарий.

5.4.1 Имя переменной

Первый символ имени быть буквой, или символом подчеркивания, далее могут следовать цифры, буквы латинского алфавита (строчные или прописные) и символы подчеркивания. Имеется ряд зарезервированных имен, использование которых запрещено – будет выдан диалог-предупреждение. Например, «TON» - зарезервированное имя функционального блока стандартной библиотеки (рисунок 14).

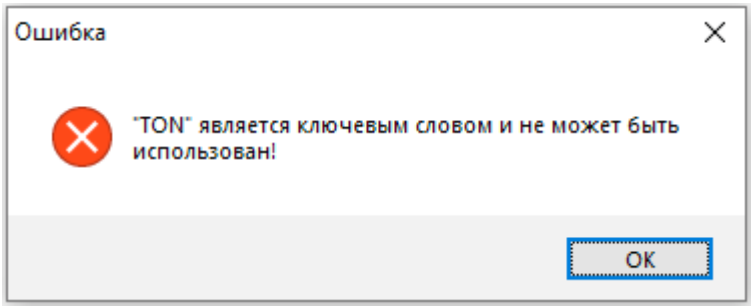


Рисунок 14 – Диалог-предупреждение о попытке использования зарезервированного имени «TON»

Переход в режим редактирования имени выполняется следующим образом:

1) щелкнуть два раза левой клавишей мыши на поле с нужным именем переменной.

5.4.2 Класс переменной

Класс выбирается из списка (рисунок 15), который отображается после двойного нажатия левой клавишей мыши по данному полю.

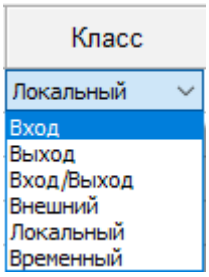


Рисунок 15 – Список выбора Класса

5.4.3 Класс переменной

Тип выбирается из списка (рисунок 16) , который отображается после двойного нажатия.

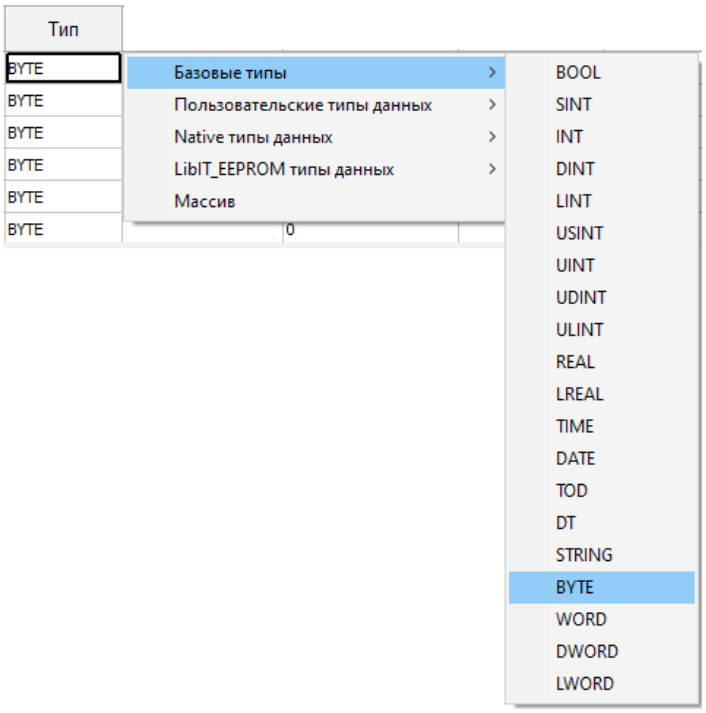


Рисунок 16 – Выбор базового Типа

5.4.4 Адрес переменной

Переменную или константу можно связать с регистром модуля УСО. Двойное нажатие на поле «Адрес» вызывает появление мигающего курсора и кнопки «…» (рисунок 17).

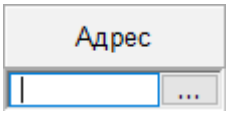


Рисунок 17 – Режим редактирования Адреса

Нажатие на кнопку «…» приводит к появлению диалога «Просмотр адресов» (рисунок 18) - списка регистров модулей УСО, которые могут быть связанны с переменой в панели переменных и констант. При выборе в данном диалоге определённого элемента и нажатии клавиши «OK» в поле «Адрес» будет добавлен адрес регистра внешнего модуля УСО. Адрес можно также ввести вручную с клавиатуры в позиции мигающего курсора.

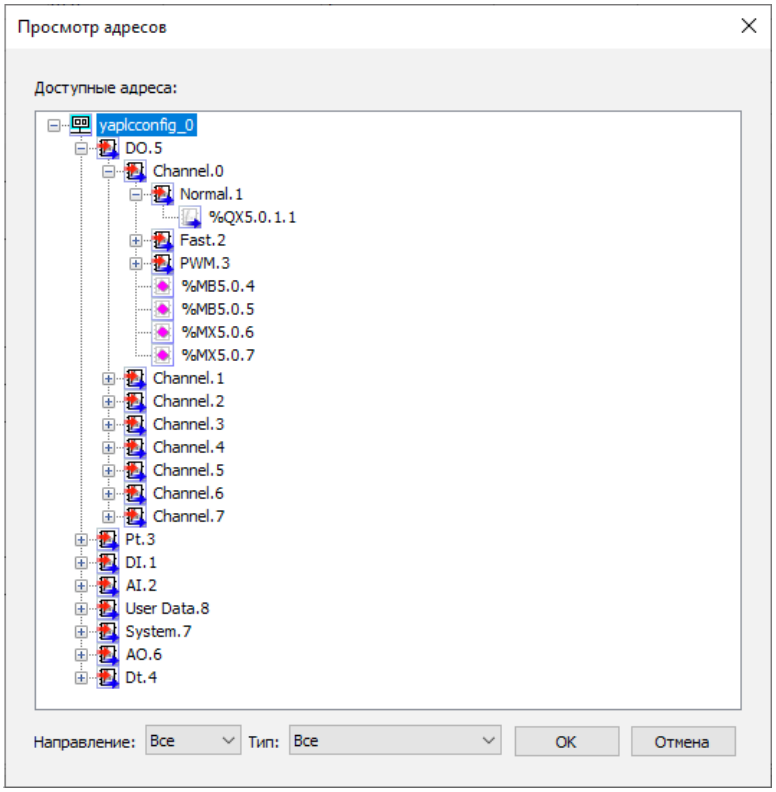


Рисунок 18 – Диалог просмотра адресов регистров УСО

5.4.5 Квалификатор переменной

Поле «Квалификатор» позволяет определить переменную как константу. Соответственно, если компилятор обнаружит в коде фрагмент, в котором происходит изменение такой переменной – будет выведена ошибка «Assignment to CONSTANT variables is not be allowed» в «Отладочной консоли». Квалификатор «Константа» не может быть использован в объявлении функциональных блоков.

5.4.6 Фильтрация списка переменных

Панель переменных и констант предоставляет возможность фильтровать отображаемые переменные по классам - выполняется с помощью функции «Фильтр по классам» (рисунок 19).

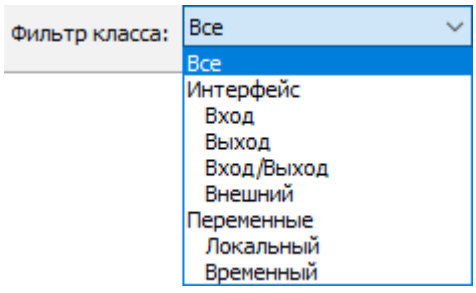


Рисунок 19 – Фильтр переменных по классу

5.4.7 Добавление и удаление переменных

Добавление, удаление, а также перемещение переменных происходит с помощью специальных кнопок на панели переменных и констант (рисунок 20). Описание функций этих кнопок приведено в таблице 3.

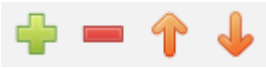


Рисунок 20 – Кнопки работы с переменными

Таблица 3 - Функции кнопок работы с переменными

Кнопка	Функция
	Добавить переменную *новая переменная добавляется со значениями по умолчанию*
	Удалить выбранную переменную из списка переменных
	Переместить выбранную переменную вверх на одну позицию
	Переместить выбранную переменную вниз на одну позицию

5.5 Панель настройки проекта

Панель настройки проекта отображается следующим образом:

1) в дереве проекта дважды щелкнуть левой клавишей мыши на корневом элементе с именем проекта.

Данная панель состоит из следующих компонентов (рисунок 21):

- панель «Конфигурационные переменные»,
- панель «Свойства проекта»,
- панель «Конфигурация»,
- кнопка «МЭК-код»,
- кнопка «Файлы проекта».

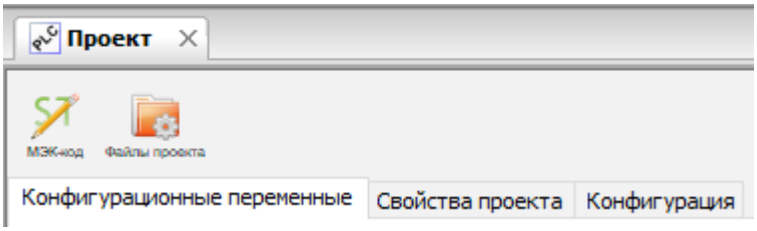


Рисунок 21 – Компоненты панели настройки проекта

5.5.1 Конфигурационные переменные

Данная панель содержит редактор глобальных переменных проекта. Работа с переменными выполняется в соответствие с п. 5.4.

Чтобы пользоваться глобальными переменными в функциях, функциональных блоках или программах, необходимо в редакторе переменных функции, функционального блока или программы создать переменную со следующими атрибутами:

- Имя соответствует имени глобальной переменной,
- Класс «Внешний»,
- Тип соответствует типу глобальной переменной.

Пример связи с глобальными переменными приведен на рисунках 22 и 23:

Конфигурационные переменные | Свойства проекта | Конфигурация

Фильтр класса: Все

#	Имя	Класс	Тип	Адрес	Исходное значение	Квалификатор
1	G_MCU_TEMP	Глобальный	REAL	%MD7.4.0	0.0	
2	G_PT0_TEMP	Глобальный	REAL		0.0	
3	G_DO0_PWM_EN	Глобальный	BOOL		TRUE	
4	G_DO0_PWM_DU	Глобальный	REAL		0.0	

Рисунок 22 – Пример конфигурационных (глобальных) переменных

prog_main

Описание: | Фильтр класса: Все

#	Имя	Класс	Тип	Адрес	Исходное значение	Квалификатор
28	PID_I	Локальный	REAL		0.0	
29	PID_D	Локальный	REAL		0.0	
30	G_DO0_PWM_EN	Внешний	BOOL			
31	G_DO0_PWM_DU	Внешний	REAL			
32	G_PT0_TEMP	Внешний	REAL			

Рисунок 23 – Пример использования глобальных переменных
из программы «prog_main»

5.5.2 Свойства проекта

Данная панель содержит набор сгруппированных полей с информацией о проекте (рисунок 24). Поля сгруппированы по вкладкам.

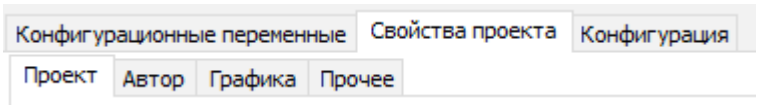


Рисунок 24 – Вкладки панели «Свойства проекта»

Вкладка «Проект» позволяет задать общую информацию о проекте (рисунок 25):

- Имя проекта,
- Версию проекта,
- Имя продукта,
- Версию продукта,
- Релиз продукта.

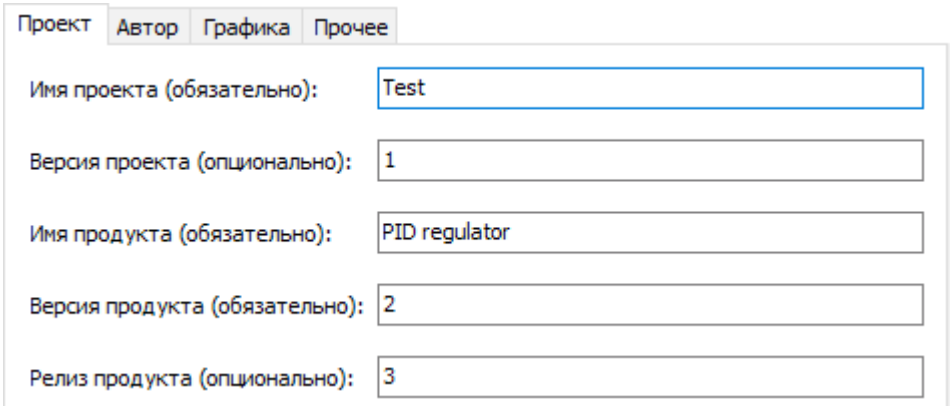


Рисунок 25 – Вкладка «Проект»

Вкладка «Автор» позволяет задать информацию об авторе проекта (рисунок 26):

- Компания,
- Сайт компании,
- Имя автора,
- Организация.

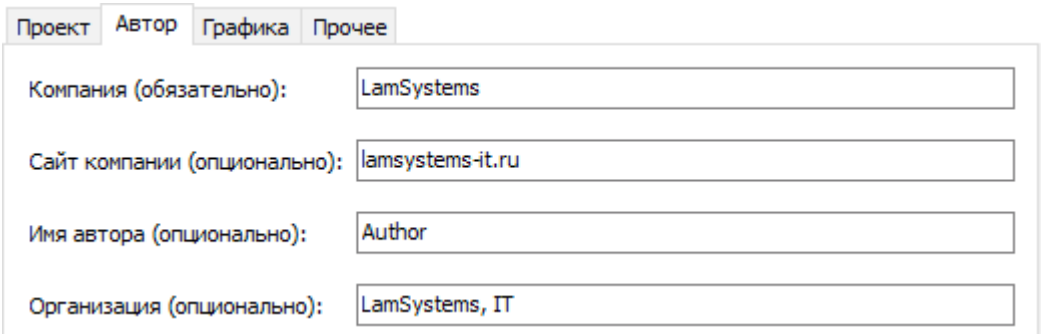


Рисунок 26 – Вкладка «Автор»

Вкладка «Графика» позволяет задать размеры страницы, шаг сетки для редакторов диаграмм графических элементов (рисунок 27).

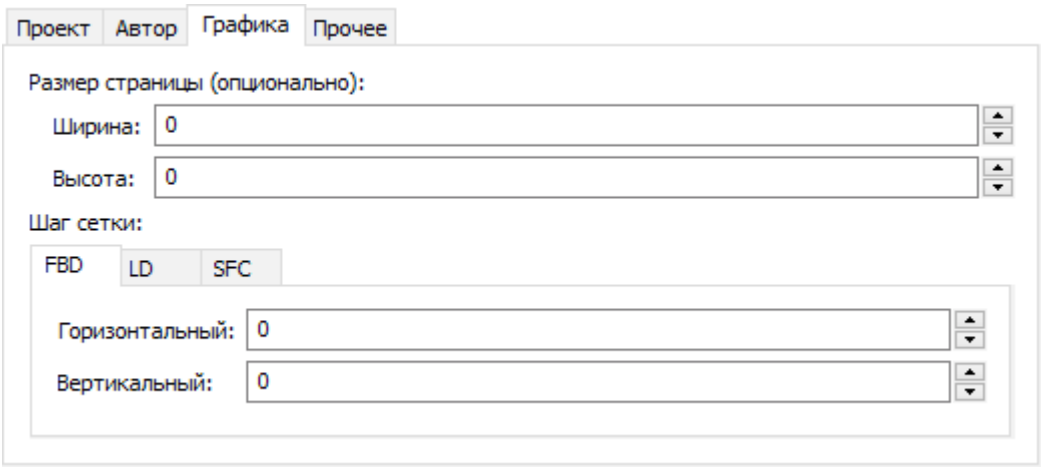


Рисунок 27 – Вкладка «Графика»

Вкладка «Прочее» позволяет задать язык интерфейса для среды разработки Beremiz и указать дополнительное текстовое описание проекта (рисунок 28).

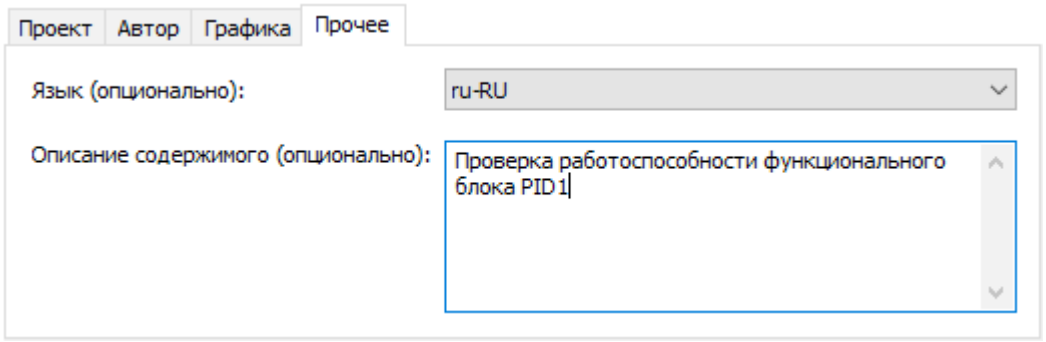


Рисунок 28 – Вкладка «Прочее»

При запуске среды разработки Beremiz языком по умолчанию является язык, соответствующий текущей локали операционной системы, если файл для данной локали присутствуют. В случае отсутствия данных файлов, устанавливается английская локаль, которая доступна всегда. Файлы доступных локалей располагаются в папке beremiz/locale.

5.5.3 Конфигурация

Данная панель содержит набор элементов (рисунок 29):

- для настройки системы отладки в режиме реального времени,
- для выбора внешних подключаемых библиотек,
- для выбора целевой платформы,
- для настройки параметров сборки проекта.

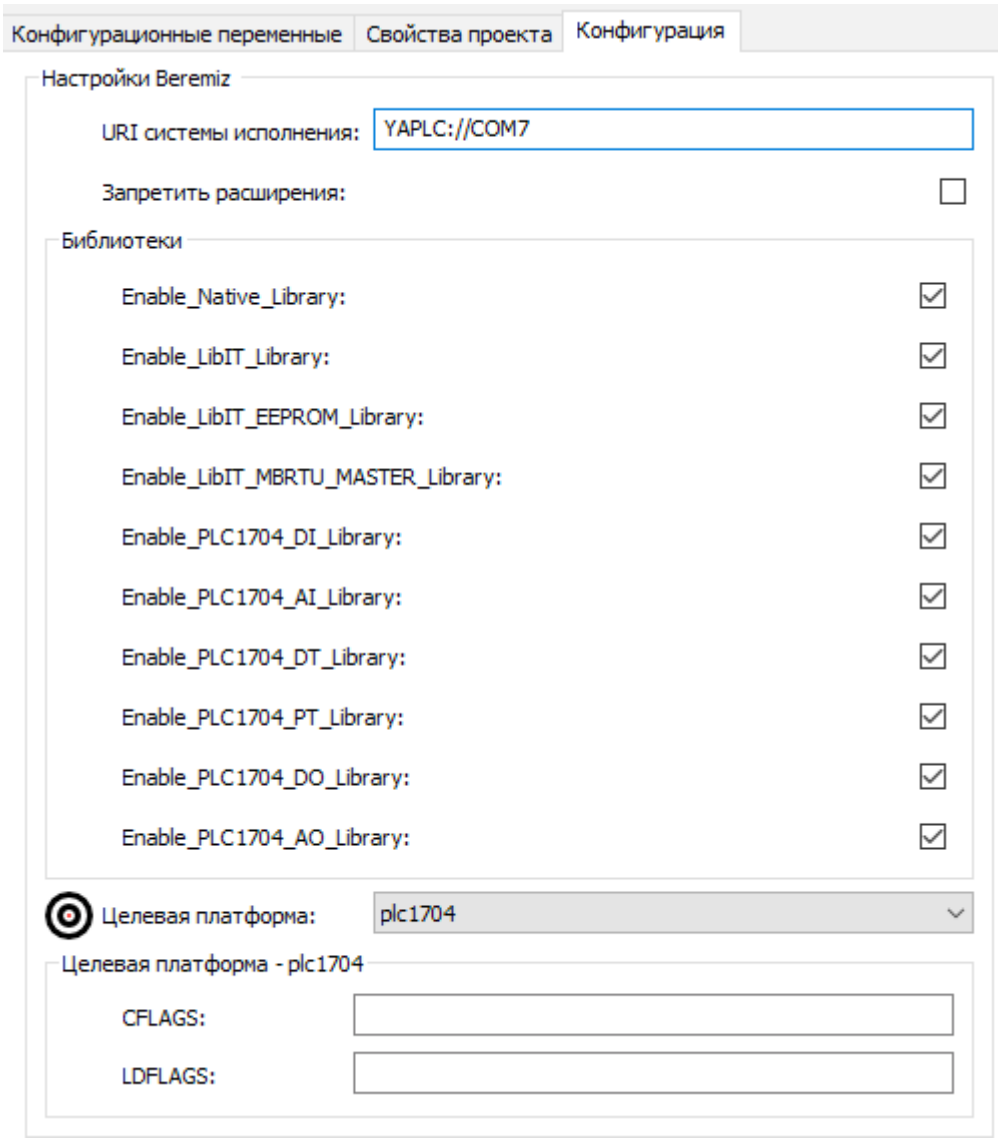


Рисунок 29 – Панель конфигурации проекта

URI системы исполнения – строка, описывающая подключение к целевому устройству для режима отладки (см. п. Режим отладки).

Библиотеки - подключаемые дополнительные библиотеки.

Целевая платформа – выбор целевой платформы (из списка).

CFLAGS – дополнительные флаги Си-компилятора.
LDFLAGS – дополнительные флаги компоновщика.

5.5.4 МЭК-код

При нажатии на кнопку «МЭК-код» отображается панель с текстовым редактором (рисунок 30), отображающим промежуточный МЭК-код на языке ST, доступный только для чтения, без возможности редактирования.

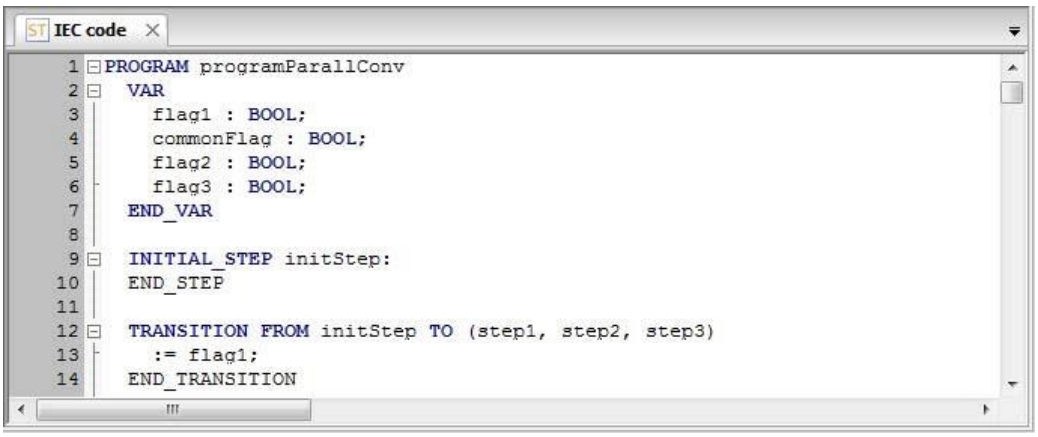


Рисунок 30 – Панель промежуточного МЭК-кода

5.5.5 Файлы проекта

Панель файлов проекта (рисунок 31) содержит встроенный проводник файлов, с помощью которого можно: добавлять дополнительные файлы в проект, удалять дополнительные файлы из проекта.

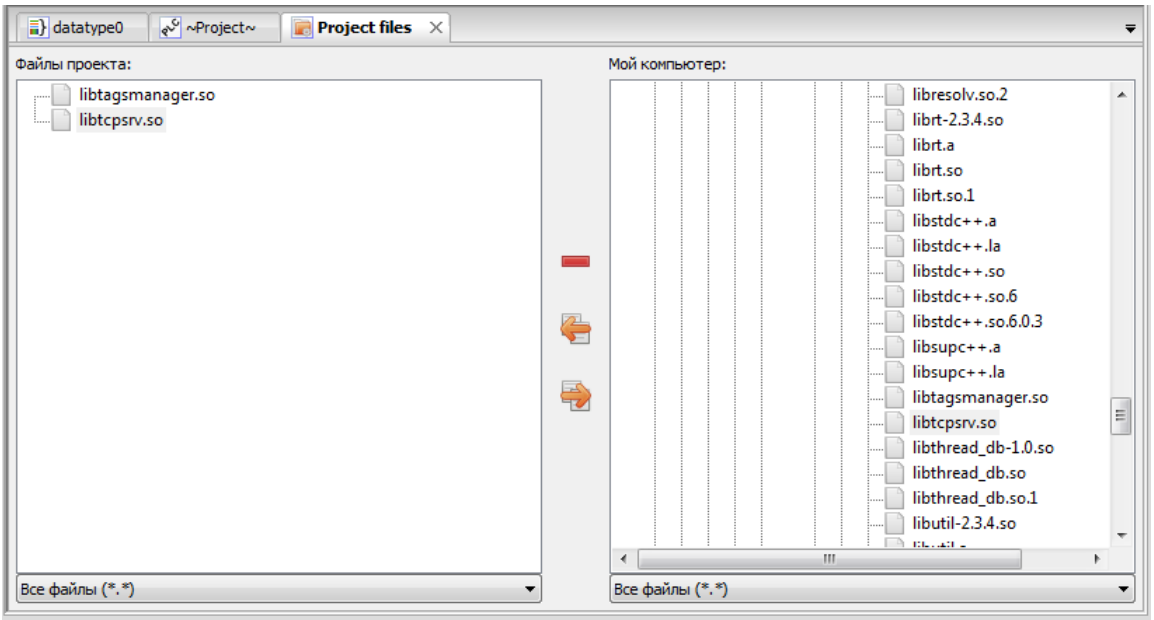


Рисунок 31 – Встроенный проводник

Все манипуляции с файлами осуществляются с помощью кнопок, расположенных в середине данной панели. Их описание приведено в таблице 4.

Таблица 4 - Функции кнопок встроенного проводника

Кнопка	Функция
	Удалить выбранный файл из проекта
	Добавить выбранный файл из файловой системы в проект
	Переместить выбранный файл из проекта в файловую систему

Дополнительные файлы, добавленные в проект, будут переданы на целевое устройство вместе с исполняемым файлом. Как правило, этими дополнительными файлами проекта являются сторонние библиотеки, необходимые для корректной работы плагинов модулей УСО.

5.6 Текстовые редакторы языков ST и IL

Текстовые редакторы языков ST и IL (рисунок 32) позволяют создавать и редактировать алгоритмы и логику выполнения программных модулей на языках ST и IL.

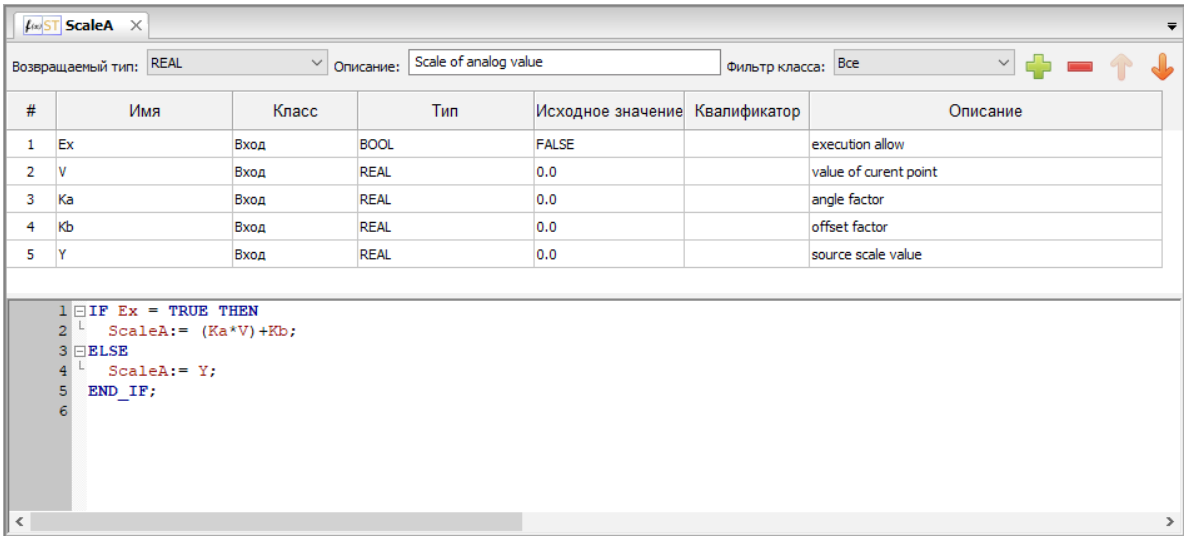


Рисунок 32 – Редактор языков ST и IL

Редактор предоставляет следующие возможности:

- подсветку синтаксиса кода, написанного пользователем, т.е. выделения особыми параметрами шрифта ключевых слов данных языков;
- нумерации строк, что может быть полезным при возникновении ошибок в программе, т.к. транслятор кода ST в C выдаёт номер строки, в которой найдена ошибка;

- сворачивание кода структурных элементов языка: определения функции, определение типа и т.д.
- увеличение или уменьшение размера шрифт выполняется с помощью Ctrl + <колесо мыши>.

Описание языка ST приведено в Приложении 3, а языка IL – в Приложении 4.

5.7 Графические редакторы языков FBD, SFC и LD

Данные редакторы позволяют создавать и редактировать алгоритмы и логику выполнения программных модулей, написанных на языках FBD, SFC и LD.

5.7.1 Редактор языка FBD

Основными элементами языка FBD являются: переменные, функциональные блоки и соединения. При редактировании FBD диаграммы, в панели инструментов появляется следующая панель (рисунок 33):

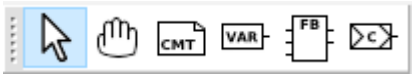


Рисунок 33 – Панель редактора FBD диаграмм

С помощью данной панели можно добавить все элементы языка FBD. Функциональное назначение каждой кнопки описано в таблице 5.

Таблица 5 - Функции кнопок панели редактора FBD диаграмм

Кнопка	Функция
	Режим выделения элементов диаграммы
	Режим перемещения выделенных элементов диаграммы
	Добавить элемент комментария
	Добавить элемент переменной
	Добавить элемент функционального блока
	Добавить элемент соединения

Добавление элементов возможно также через контекстное меню, вызываемое нажатием правой кнопки мыши в области редактора диаграммы (рисунок 34).

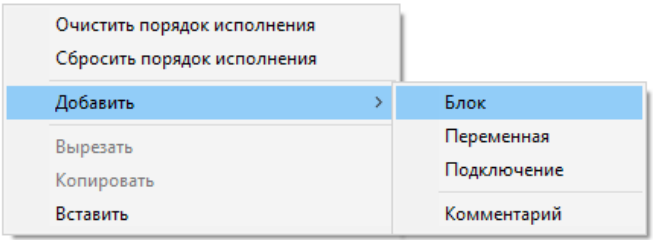


Рисунок 34 – Контекстное меню редактора FBD

5.7.1.1 Добавление функционального блока

Добавить функциональный блок в диаграмму возможен с помощью кнопок панели редактора диаграммы или через контекстное меню.

При добавлении функционального блока одним из описанных выше способов, появится диалог «Свойства блока» (рисунок 35).

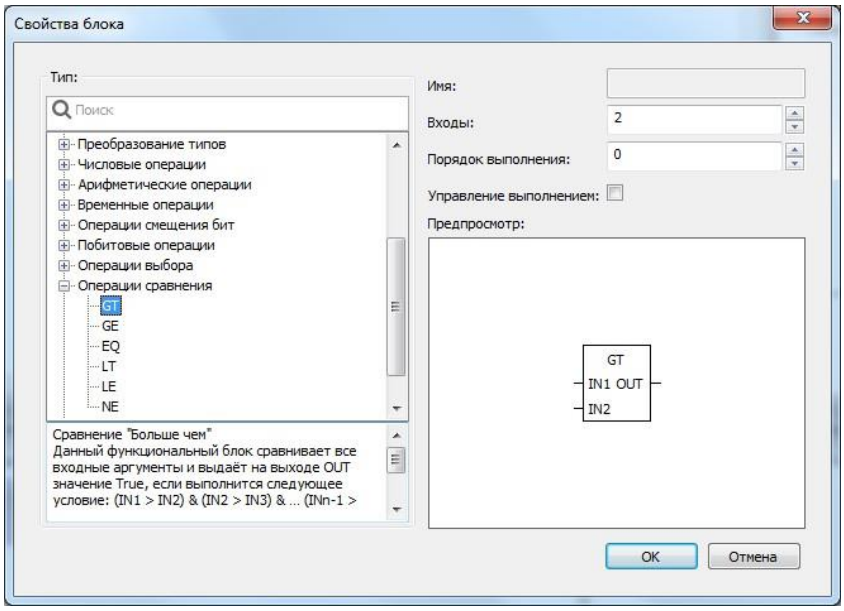


Рисунок 35 – Свойства функционального блока

В данном диалоге приведено краткое описание функционального блока и предоставлена возможность задать некоторые свойства (имя, количество входов, порядок выполнения и т.д.).

Опция «Управление выполнением» добавляет в функциональный блок дополнительные параметры EN/ENO, о которых подробнее рассказано в Приложении 5. Для сохранения изменений необходимо нажать «OK». Одним из свойств является «Порядок выполнения», его назначение рассматривается в п.5.7.1.5.

Добавление (путем копирования существующего блока), удаление и переименование функционального блока осуществляется при помощи команд меню «Редактирование» в главном меню или с помощью всплывающего меню диаграммы.

Следует отметить, что функциональный блок может быть так же добавлен из «Панели библиотеки функций и функциональных блоков» (см. п.5.11), перетаскиванием мыши (Drag&Drop) выбранного блока на панель редактирования диаграммы FBD.

5.7.1.2 Добавление переменной

Переменные добавляются из панели переменных и констант с помощью перетаскивания (Drag&Drop) левой клавишей мыши за область, выделенную красным цветом, в область редактирования диаграмм (рисунок 36).

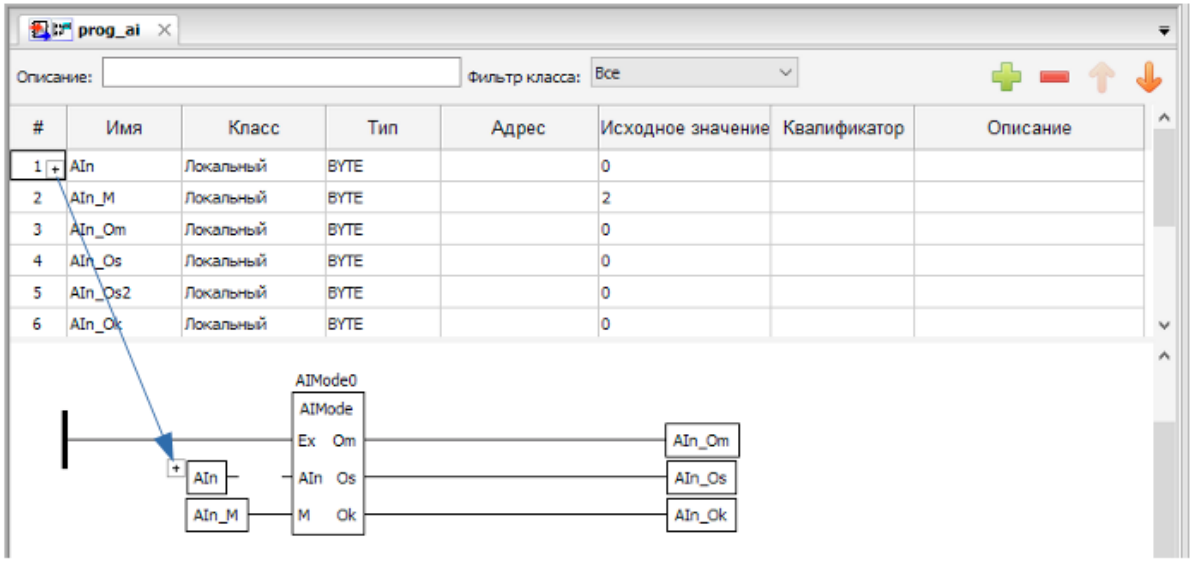


Рисунок 36 – Добавление переменной из панели переменных и констант

Изменить параметры переменной можно в диалоге «Свойства переменной» (рисунок 37), нажав на неё два раза левой клавишей мыши.

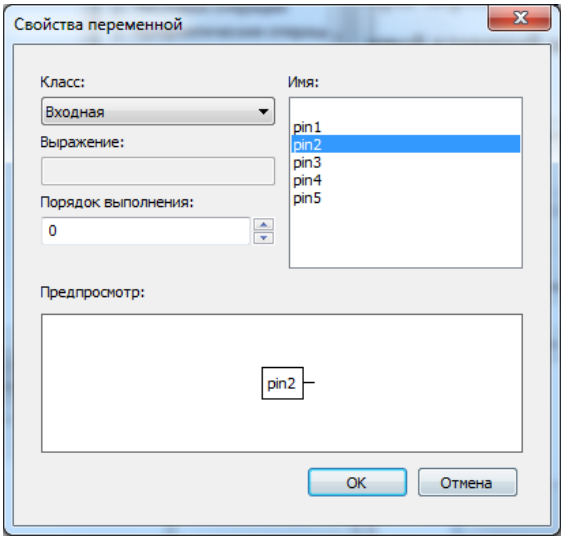


Рисунок 37 – Свойства переменной

В данном диалоге можно задать порядок выполнения переменной и изменить её класс («Входная», «Выходная», «Входная/Выходная»), а также выбрать другую переменную – имя.

Другой способ добавления переменной в область диаграммы – нажать кнопку VAR в панели редактора и щелкнуть левой кнопкой мыши на свободной области диаграммы. При этом автоматически будет вызвано диалоговое окно «Свойства переменной», где необходимо выбрать переменную и задать ее свойства.

5.7.1.3 Добавление соединения

В тех случаях, когда необходимо передать выходное значение одного функционального блока на один из входов другого, удобно использовать элемент «Соединение». При прямом соединении с помощью перетаскивания выхода одного функционального блока к входу другого получится прямое соединение с помощью чёрной соединительной линии. На схемах с большим количеством функциональных блоков элемент «Соединение» позволяет избежать пересечения прямых соединений, которые приводит к тому, что схема становится менее понятной.

После выбора добавления элемента «Соединение» появится диалог «Свойства соединения» (рисунок 38).

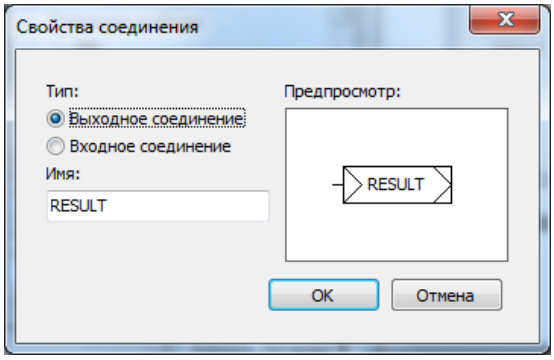


Рисунок 38 – Диалог добавления соединения для FBD

В данном диалоге можно выбрать тип соединения: «Выходное соединение» – для выходного значения, «Входное соединение» – для входного значения, а так же необходимо указать имя данного соединения. На рисунке 39 представлен пример использования соединений.

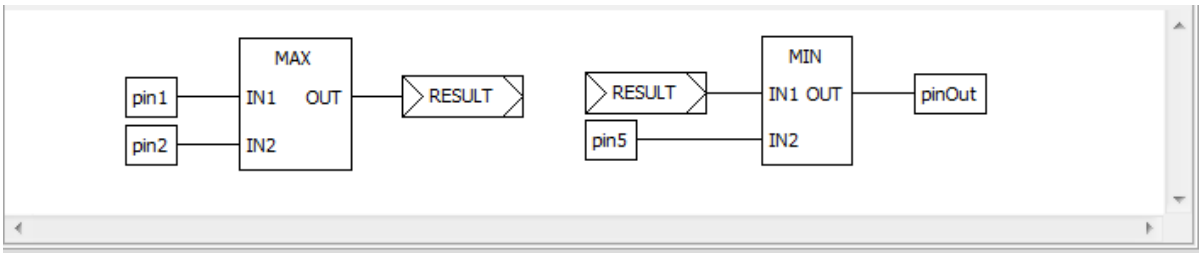


Рисунок 39 – Пример FBD диаграммы с использованием соединений

Функция «MAX» на выходе «OUT» имеет некоторое значение, которое с помощью соединения «RESULT» передаётся на вход «IN1» в функцию «MIN». В функции «MAX» используется соединение типа «Выходное соединение», в функции «MIN» – типа «Входное соединение». Имена у этих соединений, соответственно, одинаковые.

5.7.1.4 Добавление комментариев

Редактор FBD диаграмм (и остальные редакторы, о которых будет рассказано ниже) позволяют добавлять комментарии на диаграмму. После выбора на панели редактирования комментария и добавления его в область редактирования появится диалог (рисунок 40) для ввода текста комментария.

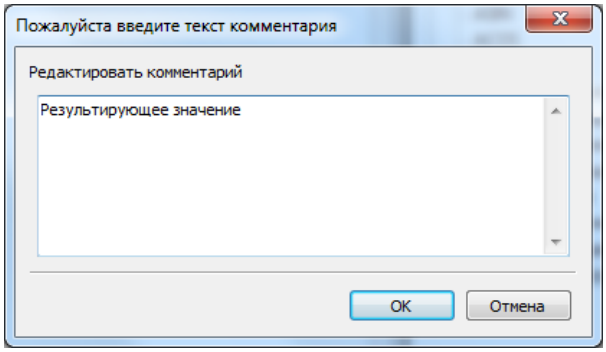


Рисунок 40 – Диалог добавления комментария

После нажатия кнопки OK комментарий появится на диаграмме (рисунок 41).

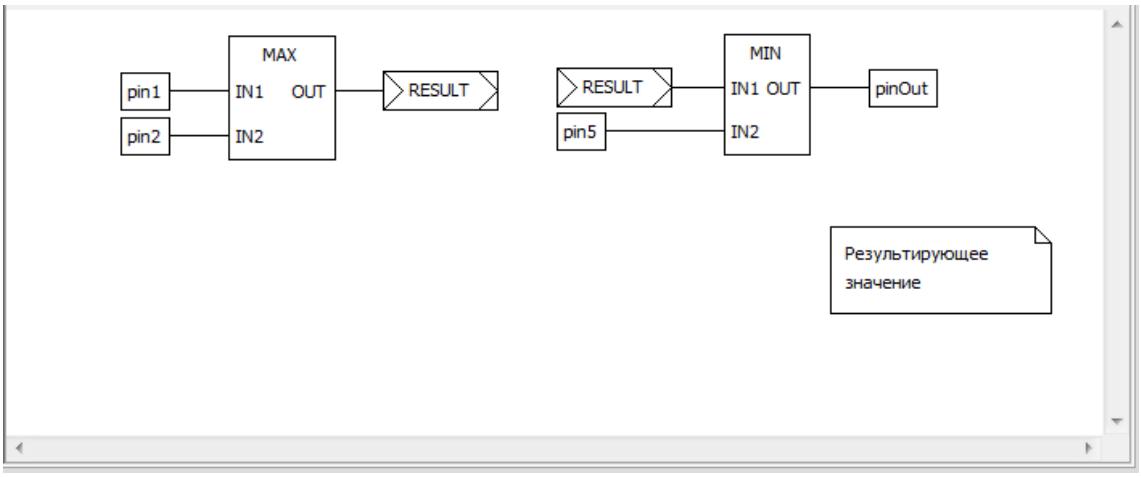


Рисунок 41 – Добавленный комментарий к FBD диаграмме

5.7.1.5 Порядок выполнения функций и функциональных блоков

Последовательность исполнения функций и функциональных блоков определяется порядком их выполнения. Автоматически он регламентируется следующим образом: чем выше и левее расположен верхний левый угол, описывающего функцию или функциональный блок прямоугольника, тем раньше данная функция или функциональный будет выполнен.

Например, на рисунке 42 порядок выполнения функций следующий: 1 – SUB; 2 – MUL; 3 – ADD.

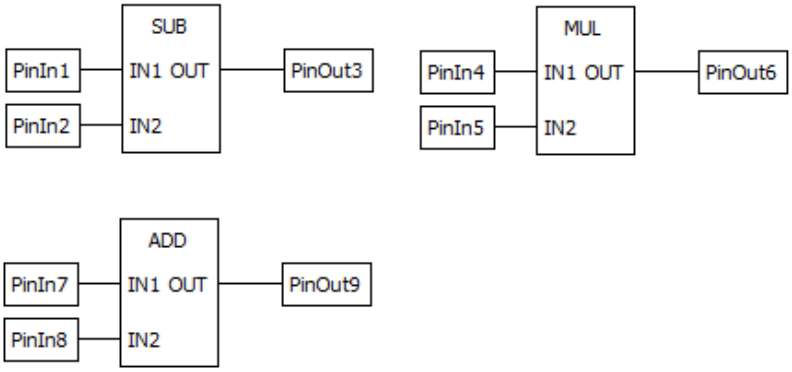


Рисунок 42 – Схема, содержащая функции с порядком выполнения (обсчета) по расположению

Порядок выполнения может быть изменён вручную с помощью диалога свойств. Данная опция «Порядок выполнения» выделена красным цветом на рисунке 43.

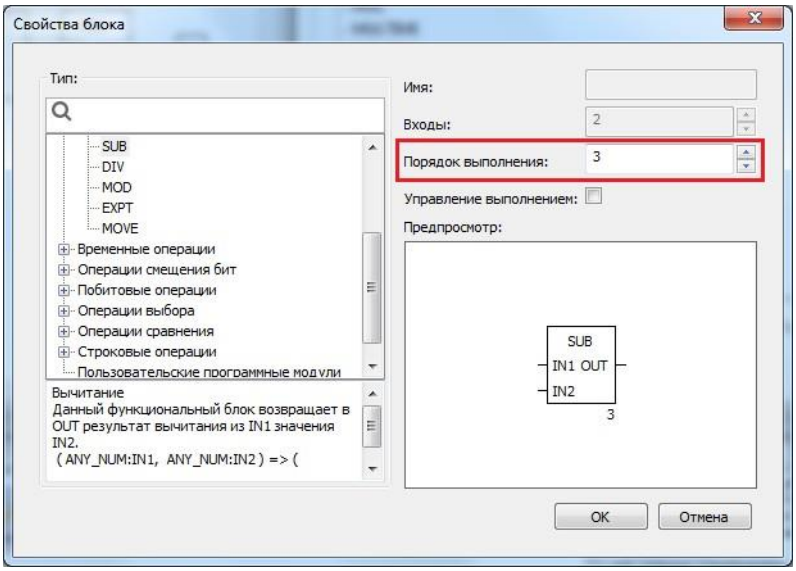


Рисунок 43 – Свойство порядок выполнения функции или функционального блока

После задания порядка выполнения для каждой функции или функционального блока на схеме в правом нижнем углу будет указан его порядковый номер выполнения. Пример представлен на рисунке 44.

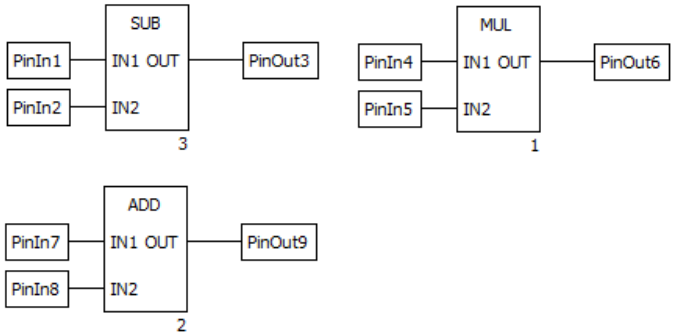


Рисунок 44 – Схема, содержащая функции с порядком выполнения заданным вручную

Описание языка FBD, основных его конструкций и пример использования приведены в Приложении 5.

5.7.2 Редактор языка LD

Язык LD или РКС (Релейно-Контактные Схемы) представляет собой графическую форму записи логических выражений в виде контактов и катушек реле. Основными элементами языка LD являются: шина питания, катушка, контакт. Добавить данные элементы, так же как и элементы языка FBD, можно несколькими способами.

Как только активной становится вкладка с редактированием LD диаграммы, в панели инструментов появляется панель (рисунок 45) с элементами языка LD.

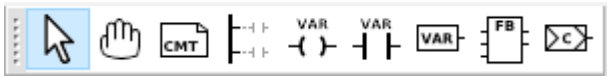


Рисунок 45 – Панель редактирования LD диаграмм

Аналогично редактору языка FBD с помощью данной панели можно добавить все элементы языка LD, а так же и FBD, т.к. есть возможность комбинированного применения языков на одной диаграмме. В таблице 6 приведено описание кнопок данной панели.

Таблица 6 - Функции кнопок панели редактора LD диаграмм

Кнопка	Функция
	Режим выделения элементов диаграммы
	Режим перемещения выделенных элементов диаграммы
	Добавить элемент комментария
	Добавить элемент шины питания (левую, правую)
	Добавить элемент катушки
	Добавить элемент контакта
	Добавить элемент переменной
	Добавить элемент функционального блока
	Добавить элемент соединения

Добавление элементов возможно также через контекстное меню, вызываемое нажатием правой кнопки мыши в области редактора диаграммы (рисунок 46).

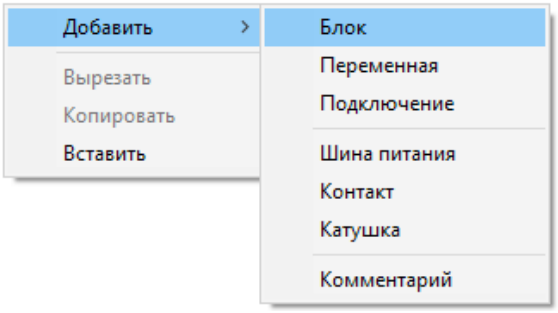


Рисунок 46 – Контекстное меню редактора LD

5.7.2.1 Добавление шины питания

При добавлении шины питания, одним из описанных выше способов, появится диалог «Свойства шины питания» (рисунок 47).

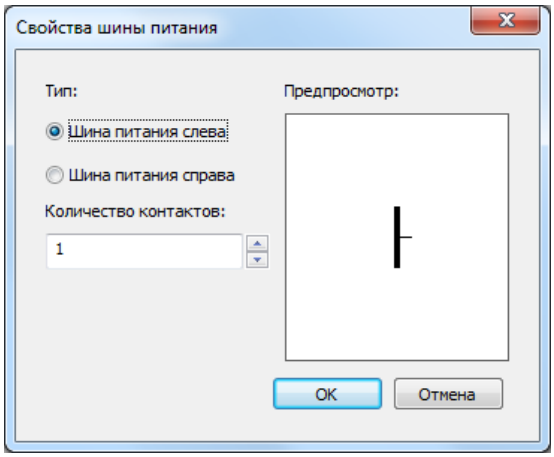


Рисунок 47 – Свойство шины питания

В данном диалоге указываются следующие свойства:

- тип шины питания: шина питания слева или шина питания справа;
- количество контактов на добавляемой шине питания.

Например, на рисунке 48 приведены две добавленные шины питания: левая с тремя контактами и правая с одним контактом.

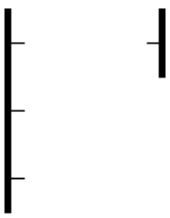


Рисунок 48 – Шины питания на LD диаграмме

5.7.2.2 Добавление контакта

При добавлении контакта на LD диаграмму появится диалог «Редактирование значения контакта» (рисунок 48).

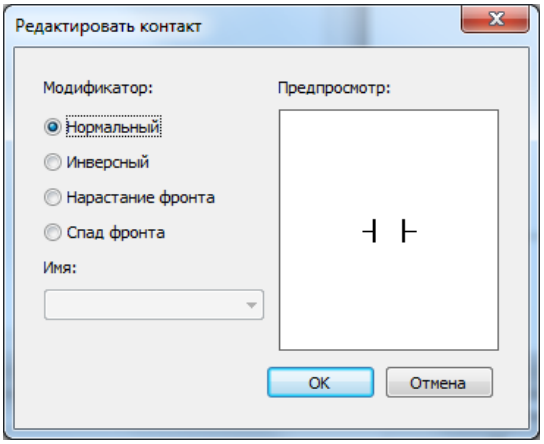


Рисунок 49 – Редактирование контакта

Данный диалог позволяет определить модификатор данного контакта:

- Нормальный;
- Инверсный;
- Нарастание фронта;
- Спад фронта.

Кроме того, диалог позволяет выбрать из списка «Имя» переменную, связываемую с данным контактом. Следует отметить, что «связываемые» переменные должны быть определены в панели переменных и констант для данного программного модуля типом BOOL.

Еще одним способом добавления контакта на диаграмму является метод Drag&Drop из панели переменных и констант переменной типа BOOL и класса: «Входная», «Входная/Выходная», «Внешняя», «Локальная», «Временная». Для этого необходимо зажать левой кнопкой мыши за первый столбец (который имеет заголовок #) переменную, удовлетворяющую описанным выше критериям и перенести в область редактирования диаграммы (рисунок 50).

Рисунок 50 – Добавление контакт на диаграмму из панели переменных и констант

5.7.2.3 Добавление катушки

При добавлении катушки на LD диаграмму появится диалог «Редактирование значения катушки» (рисунок 51).

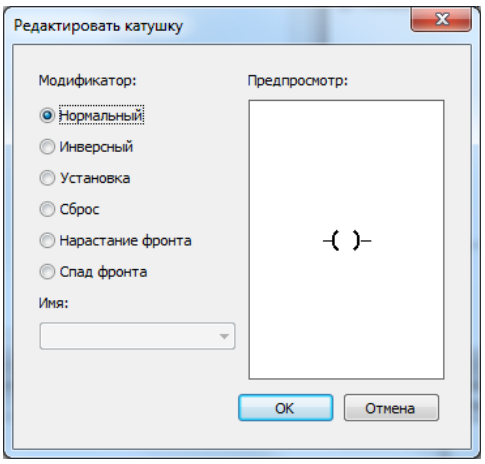


Рисунок 51 – Редактирование катушки

В данном диалоге можно определить модификатор данного контакта:

- Нормальный;
- Инверсный;
- Установка;
- Сброс;
- Нарастание фронта;
- Спад фронта.

Кроме того, производится выбор из списка «Имя» переменной, «связываемой» с данным контактом. Эти переменные, как и для контактов, должны быть определены в панели переменных и констант для данного программного модуля типом BOOL.

Аналогично добавлению контакта с помощью Drag&Drop можно добавить и катушки, но в данном случае переменная должна относиться к классу «Выходная» (рисунок 52).

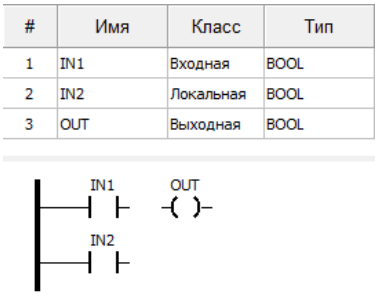


Рисунок 52 – Добавление катушки на диаграмму из панели переменных и констант

Описание языка LD, основных конструкций и примера его использования приведены в Приложении 6.

5.7.3 Редактор языка SFC

Основными элементами языка SFC являются: начальный шаг, шаг, переход, блок действий, дивергенции, «прыжок». Программа на языке SFC состоит из набора шагов, связанных переходами.

Как только активной становится вкладка с редактированием SFC диаграммы, в панели инструментов появляется следующая панель (рисунок 53).

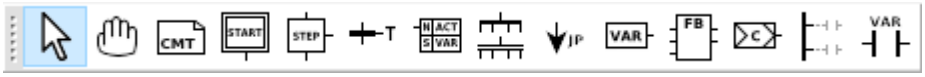


Рисунок 53 – Панель редактора SFC диаграмм

С помощью данной панели можно добавить все элементы языка SFC. Функциональное назначение каждой кнопки описано в таблице 7.

Таблица 7 - Функции кнопок панели редактора SFC диаграмм

Кнопка	Функция
	Режим выделения элементов диаграммы
	Режим перемещения выделенных элементов диаграммы
	Добавить элемент комментария
	Добавить элемент начального шага
	Добавить элемент шага
	Добавить элемент перехода
	Добавить элемент блока действий
	Добавить элемент дивергенции/конвергенции
	Добавить элемент прыжка
	Добавить элемент переменной
	Добавить элемент функционального блока
	Добавить элемент соединения

Добавление элементов возможно также через контекстное меню, вызываемое нажатием правой кнопки мыши в области редактора диаграммы (рисунок 54).

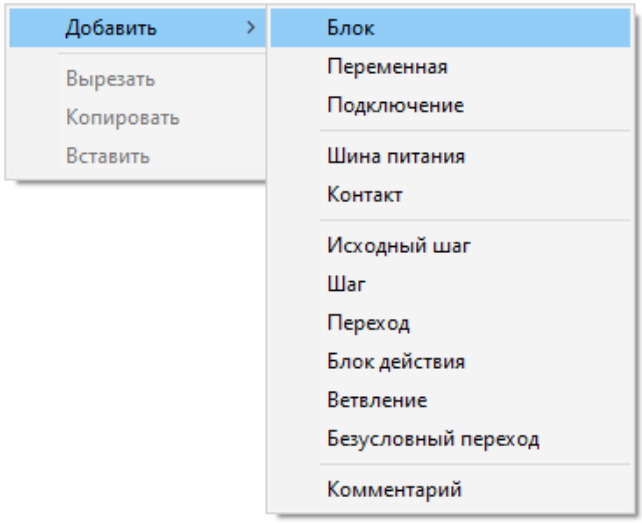


Рисунок 54 – Контекстное меню редактора SFC

5.7.3.1 Добавление шага инициализации и шага

Процедура добавления шага инициализации и обычного шага ничем не отличается. В обоих случаях вызывается диалог «Редактировать шаг» (рисунок 55).

Согласно стандарту IEC 61131-3, на SFC диаграмме должен быть один шаг инициализации, который характеризует начальное состояние SFC-диаграммы и отображается со сдвоенными линиями на границах (рисунок 56).

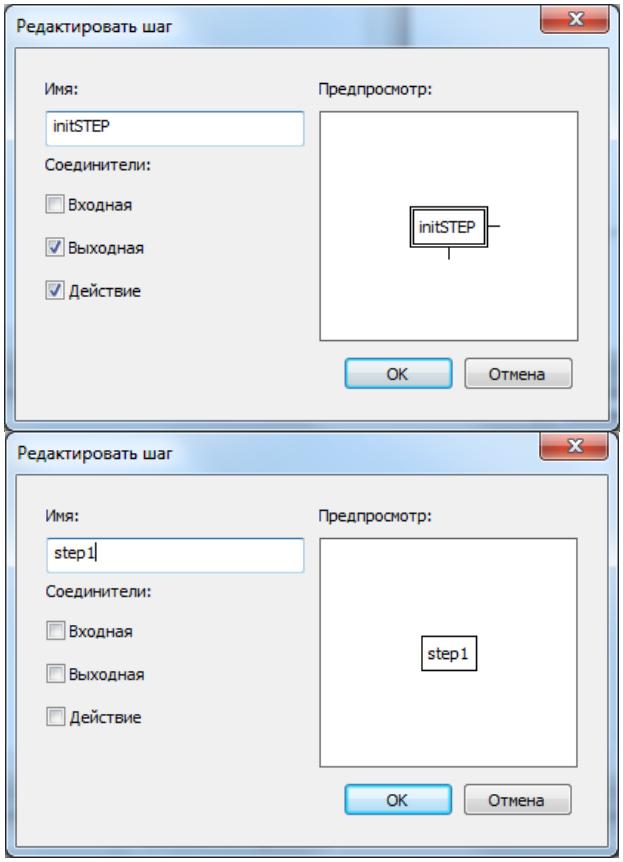


Рисунок 55 – Диалоги редактирования шага инициализации и обычного шага SFC диаграммы

Рисунок 56 – Шаг инициализации языка SFC

В случае, если при добавлении шага не указано его имя – будет выдана ошибка (рисунок 57).

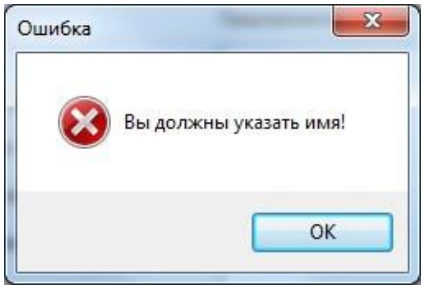


Рисунок 57 – Ошибка отсутствия имени шага при его добавлении

При добавлении шага появляется диалог, в котором можно указать, с помощью галочек его соединители (рисунок 58):

- Входная;
- Выходная;
- Действие.

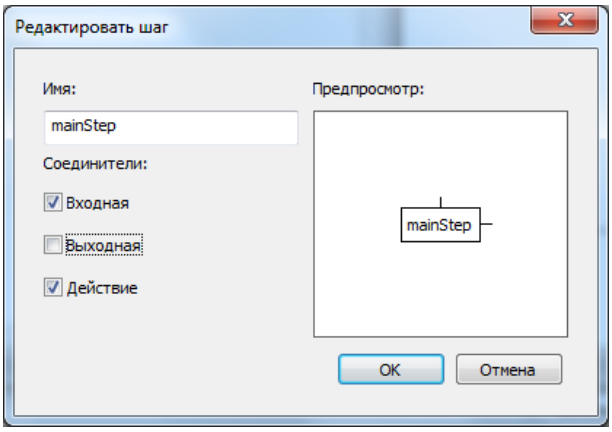


Рисунок 58 – Добавление шага на SFC диаграмму

«Действие» добавляет соединитель для связывания данного шага с блоком действий. «Входной» и «Выходной» соединители, как правило, соединены с переходом. Соответственно, после нажатия кнопки OK, на диаграмму будет добавлен шаг с указанными соединителями (рисунок 59).

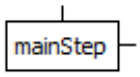


Рисунок 59 – Шаг SFC диаграммы с соединителями входа и действия

5.7.3.2 Добавление перехода

При добавлении на SFC диаграмму перехода, появится диалог «Редактировать переход» (рисунок 60).

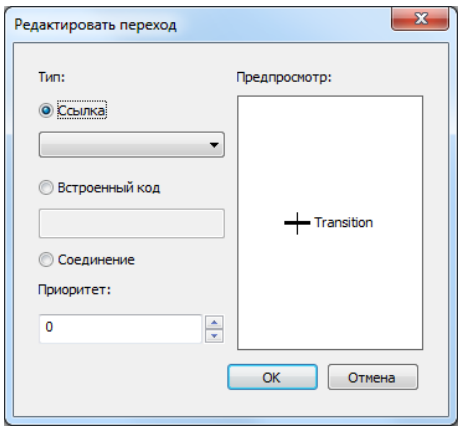


Рисунок 60 – Диалог редактирования перехода для SFC диаграммы

В данном диалоге необходимо выбрать тип перехода и его приоритет. Тип перехода может быть:

- Ссылка;
- Встроенный код;
- Соединение.

При выборе типа перехода «Ссылка» в открывающемся списке (рисунок 61) будут доступны переходы, предопределённые в дереве проекта для данного программного модуля, написанного на языке SFC. Добавление предопределённого перехода описывается ниже после описания всех добавляемых элементов языка SFC.

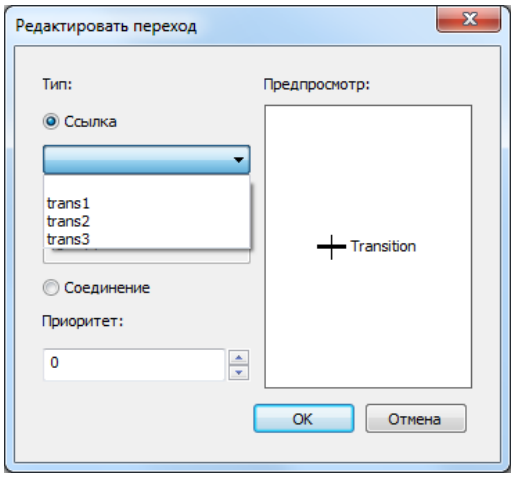


Рисунок 61 – Всплывающий список с доступными предопределёнными переходами

При выборе типа перехода «Встроенный код» (рисунок 62), условие перехода можно написать в виде выражения на языке ST.

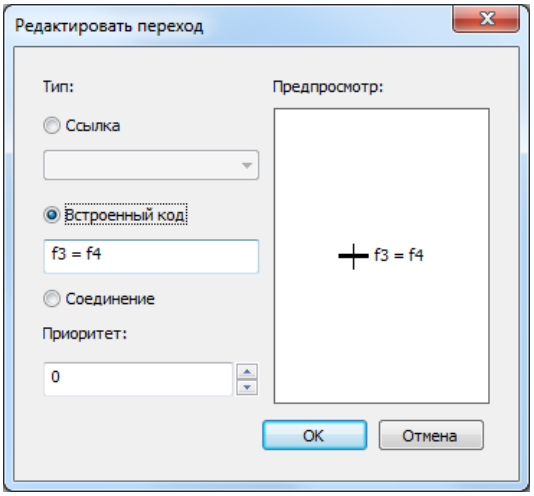


Рисунок 62 – Условие перехода в виде встроенного кода, написанного на языке ST

Реализация перехода таким способом удобна в случае, когда необходимо короткое условие.

Например: переменные «f3» и «f4» типа INT равны. Встроенный код для такого условия выглядит следующим образом: f3 = f4.

Также, например, можно в качестве условия просто указать переменную. В случае её значения равного 0 – будет означать FALSE, все остальные значения – TRUE.

При выборе типа перехода «Соединение» (см. рис. 59), в качестве условия перехода можно использовать выходные значения элементов языка FBD или LD.

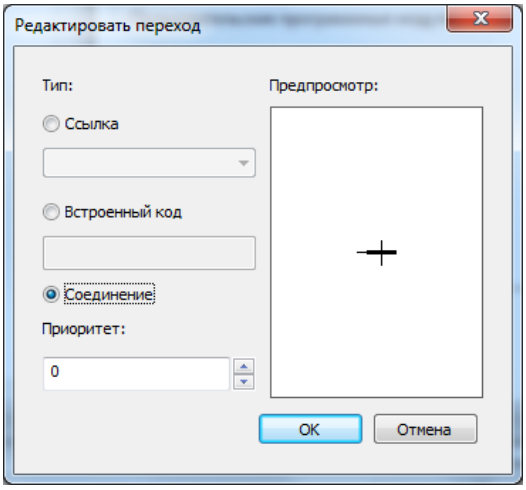


Рисунок 63 – Выбор условия перехода как соединение с элементами других графических языков IEC 61131-3

При выборе типа перехода «Соединение», у добавленного перехода появится слева контакт, который необходимо соединить с выходным значением, например, функционального блока языка FBD или катушки LD диаграммы. Стоит отметить, что данное выходное значение должно быть типа BOOL. Например, на рисунке 64 красным цветом выделен переход, условия которого заданы с помощью языка LD – двух последовательно соединённых контактов языка LD.

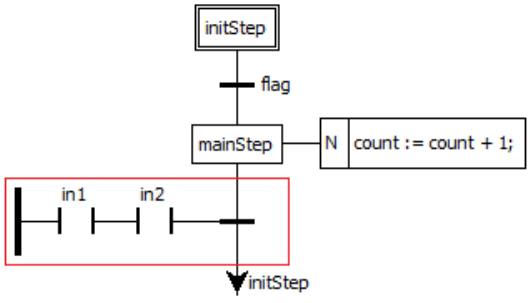


Рисунок 64 – Пример SFC диаграммы, в которой один из переходов задан с помощью языка LD

5.7.3.3 Добавление блока действий

При добавлении блока действий на диаграмму появится диалог «Редактировать свойство блока действий» (рисунок 65).

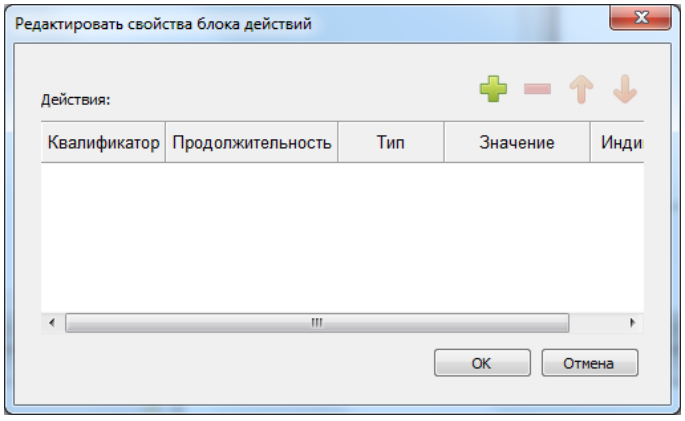


Рисунок 65 – Диалог «Редактировать свойство блока действий»

Данный блок действий может содержать набор действий. Добавить новое действие можно нажав кнопку «Добавить» и установив необходимые параметры:

- Квалификатор;
- Продолжительность;
- Тип:
  - Действие,
  - Переменная,
  - Встроенный код;
- Значение;
- Индикатор.

Поле «Квалификатор» определяет момент времени, когда действие начинается, сколько времени продолжается и когда заканчивается. Выбрать квалификатор можно из списка (рисунок 66).

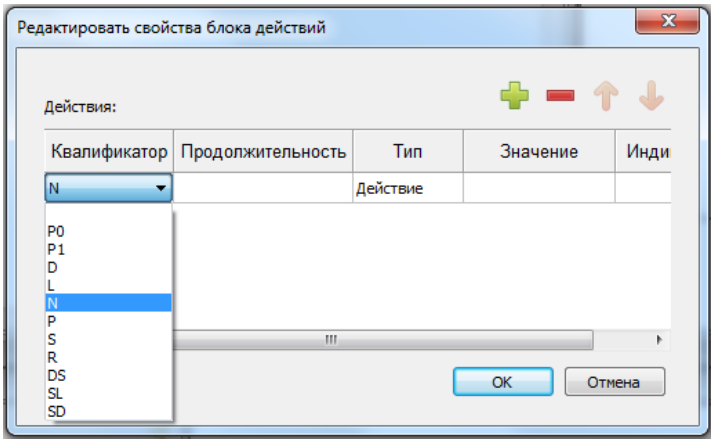


Рисунок 66 – Меню выбора квалификатора для действия в диаграмме SFC

Подробное описание квалификаторов, которые выбираются из предлагаемого списка при добавлении действия приведено в таблице 8.

Таблица 8 – Квалификаторы действий SFC диаграммы

Квалификатор	Поведение блока действий
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 деактивируется
DS	Действие начинается выполняться через некоторое заданное время, только в том случае если шаг еще активен
SL	Действие активно в течении некоторого, заданного интервала
SD	Действие начинается выполняться через некоторое время, даже в том случае если шаг уже не активен

Поле «Продолжительность» необходимо для установки интервала времени необходимого для некоторых квалификаторов, описанных выше в таблице.

«Тип» определяет код или конкретную манипуляцию, которая будет выполняться во время активации действия. В случае выбора «Действия» появляется возможность, как и в случае с переходом, использовать предопределённые действия в дереве проекта для данного программного модуля, написанного на языке SFC (рисунок 67).

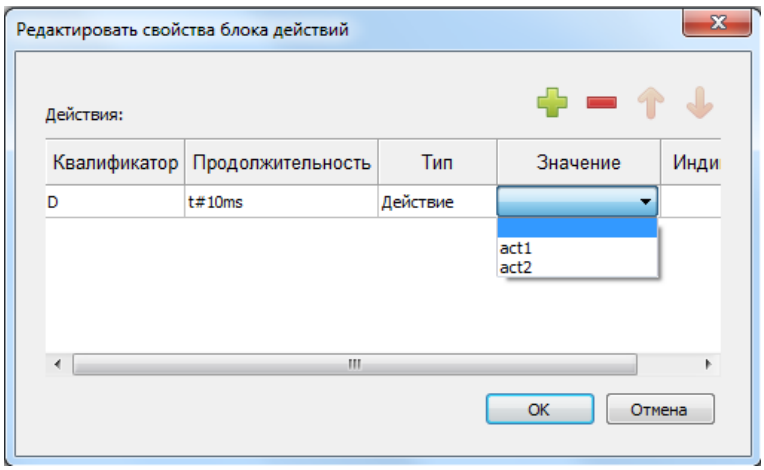


Рисунок 67 – Выбор предопределённого действия

Добавление предопределённого действия также как добавление предопределённого перехода описывается ниже после описания всех добавляемых элементов языка SFC.

В случае выбора типа действия «Переменная» в поле «Значение» появляется возможность выбрать переменные (рисунок 68), относящиеся к данному программному модулю.

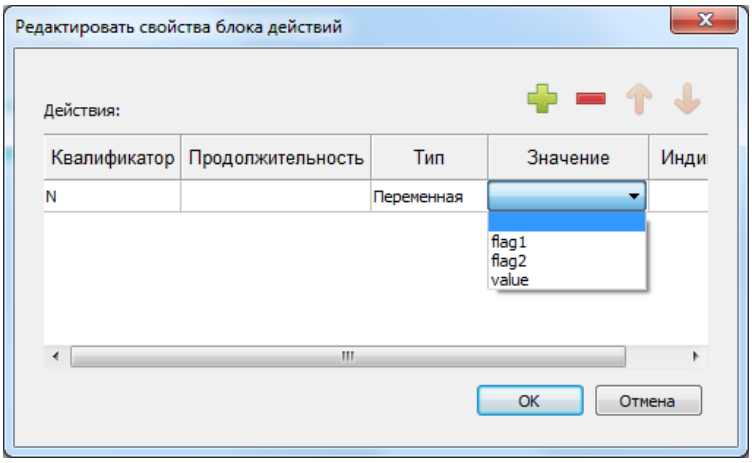


Рисунок 68 – Выбор предопределённой переменной

Как только шаг становится активным, данная переменная в зависимости от своего типа принимает значение 0, 0.0, FALSE и другие нулевые значения типов. Как только действие начинает выполняться, переменная принимает значение 1, 1.0, TRUE и другие единичные значения типов. В случае если действие прекратило своё выполнение переменная снова принимает значение 0, 0.0, FALSE и другое нулевое значение, в зависимости от своего типа.

В случае выбора «Встроенный код», появляется возможность в поле «Значение» написать на языке ST код, который будет выполняться, когда действие становится активным (рисунок 69).

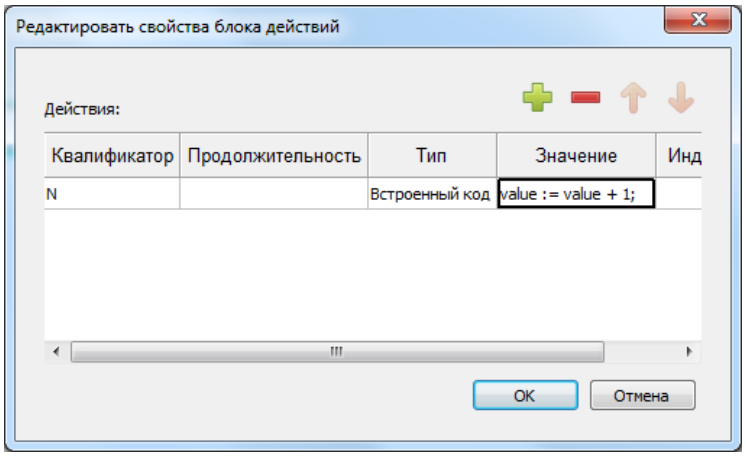


Рисунок 69 – Написание встроенного кода для действия

Следует отметить, что в конце встроенного кода для действия необходимо поставить «;», в отличие от встроенного кода для перехода.

После добавления блока действия на диаграмму необходимо его ассоциировать с конкретным шагом. Данная операция выполняется обычным соединением правого контакта у шага и левого контакта у действия (рисунок 70).

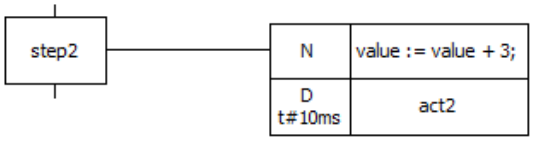


Рисунок 70 – Пример ассоциирование шага step2 блоком действия, содержащим два действия

5.7.3.4 Добавление дивергенции/конвергенции

При добавлении дивергенции, появится диалог «Создать новую дивергенцию и конвергенцию» (рисунок 71).

В первую очередь следует выбрать тип дивергенции:

- Дивергенция;
- Конвергенция;
- Параллельная дивергенция;
- Параллельная конвергенция.

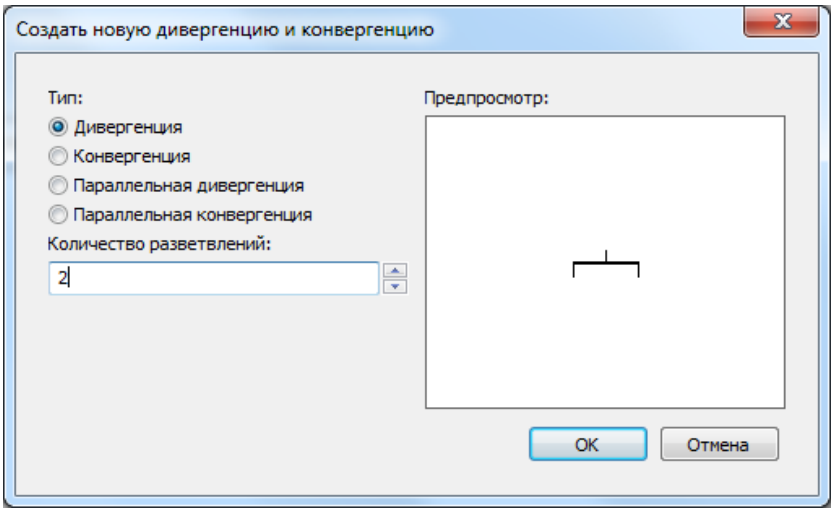


Рисунок 71 – Добавление дивергенции

Вторым параметром является количество разветвлений, которое определяет на сколько ветвей будет либо расходится (в случае выбора типа дивергенции «Дивергенции» или «Параллельной дивергенции») одна ветвь, либо сколько ветвей будет сходиться в одну ветвь (в случае выбора типа дивергенции «Конвергенция» или «Параллельная конвергенция»).

Пример дивергенции с двумя разветвлениями показан на рисунке 72 и выделен красным цветом.

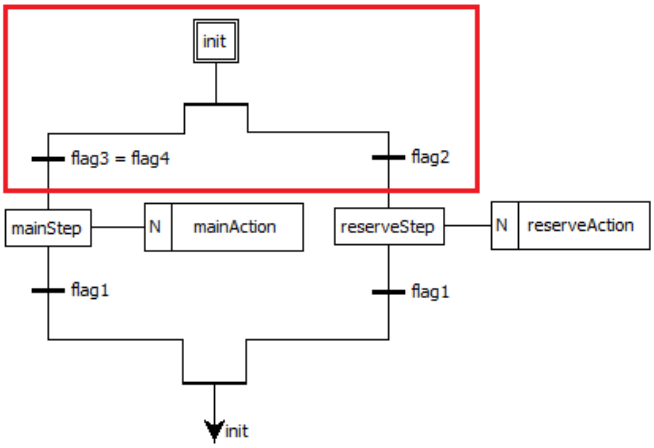


Рисунок 72 – Пример SFC диаграммы, содержащей дивергенцию

Пример конвергенции выделен красным цветом на рисунке 73.

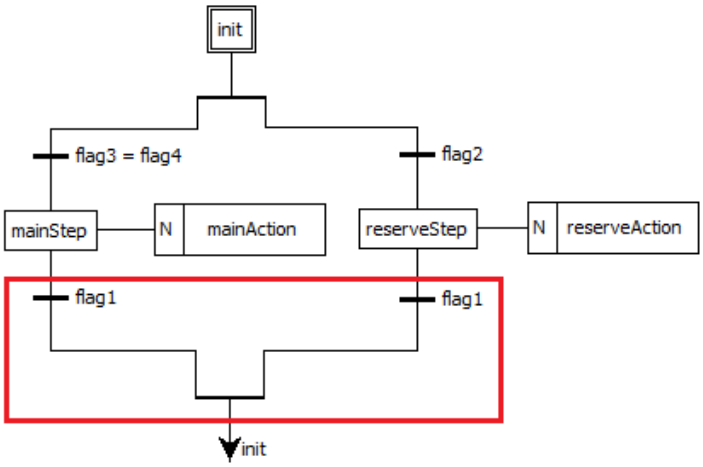


Рисунок 73 – Пример SFC диаграммы, содержащей конвергенцию

Пример параллельной конвергенции показан на рисунке 74 и выделен красным цветом.

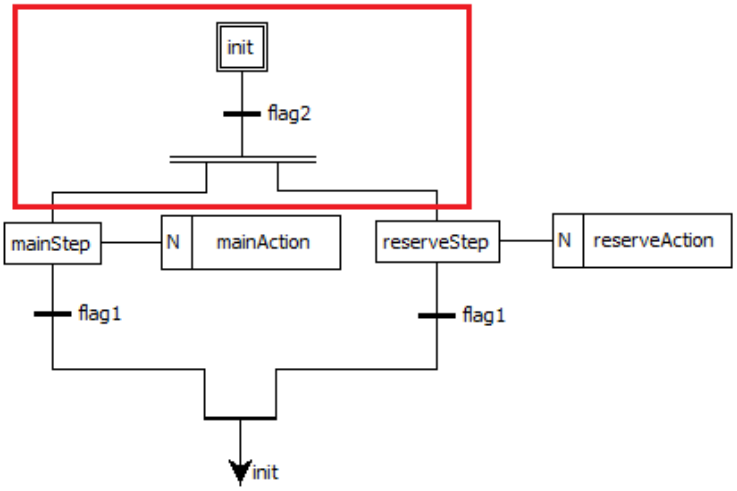


Рисунок 74 – Пример SFC диаграммы, содержащей параллельную дивергенцию

5.7.3.5 Добавление «прыжка»

Элемент «прыжок» на SFC диаграмме подобен выполнению оператора GOTO при переходе на определённую метку в коде в различных языках программирования. После выбора добавления «прыжка» на SFC диаграмму, появится диалог (рисунок 75), в котором необходимо выбрать из списка шаг, к которому будет происходить «прыжок» – переход от одного шага SFC диаграммы к другому.

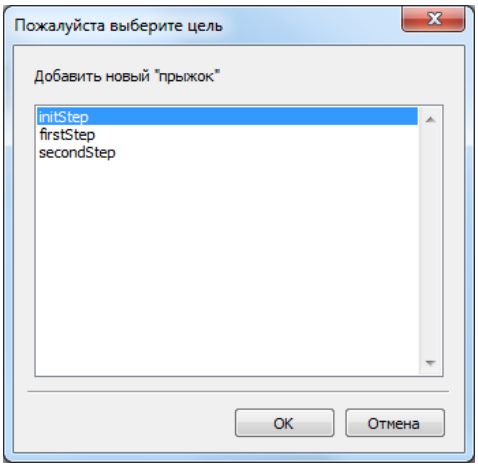


Рисунок 75 – Диалог добавления «прыжка»

В данном диалоге также присутствует и шаг инициализации (начальный шаг). После выбора шага и нажатия кнопки OK. На SFC диаграмме появится стрелочка, которую нужно соединить с переходом (рисунок 76). Справа от стрелочки находится имя шага, к которому осуществляется переход в случае выполнения условия перехода, находящегося выше и соединённого с ней.

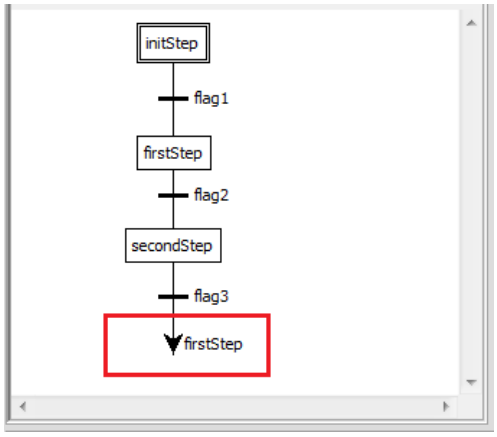


Рисунок 76 – «Прыжок» с шага «secondStep» на «firstStep»

Согласно стандарту IEC 61131-3, между шагом и «прыжком» должен обязательно быть определён переход.

5.7.3.6 Предопределённые условия перехода и действия в дереве проекта

В случае, если необходимо использовать определённое условие перехода между множеством шагов, есть возможность определить данное условие перехода в структуре SFC диаграммы. Данная операция выполняется нажатием на данную SFC диаграмму на дереве проекта (п. 5.3) правой клавишей мыши и выбором «Добавить переход» (рисунок 77).

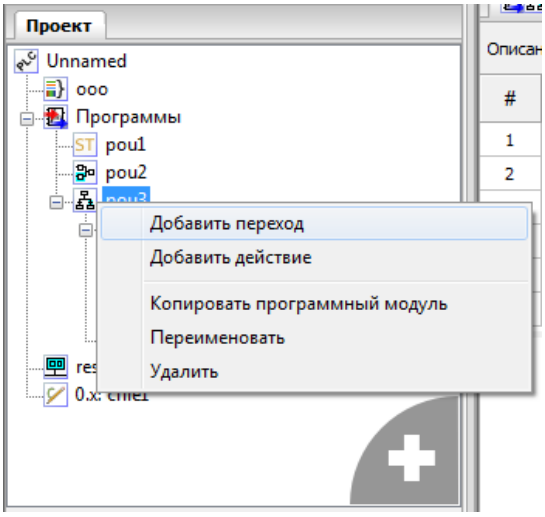


Рисунок 77 – Всплывающее меню SFC диаграммы в панели проекта

Далее появится диалог под названием «Создать новый переход» (рисунок 78). В нём необходимо выбрать уникальное имя перехода и язык, в котором будет описанно данное условие.

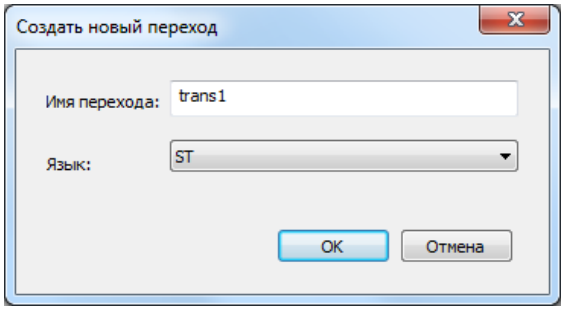


Рисунок 78 – Диалог «Создать новый переход»

В случае, если переходы с введённым именем уже существуют, то будет выведено сообщение об ошибке (рисунок 79).

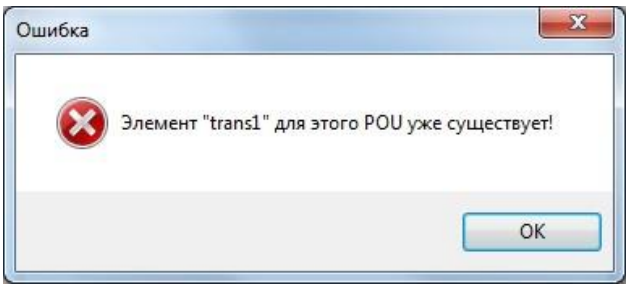


Рисунок 79 – Сообщение об ошибке добавления существующего программного модуля

Добавление действия в структуру SFC диаграммы (рисунок 80) происходит аналогично добавлению перехода в данную структуру.

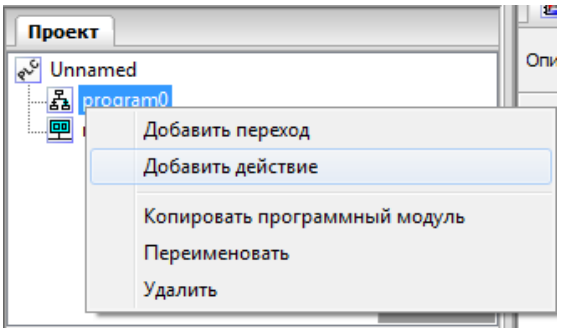


Рисунок 80 – Всплывающее меню SFC для структуры диаграммы

После выбора «Добавить действие» во всплывающем меню, вызванном с помощью нажатия правой клавиши мыши по программному модулю, написанном с помощью языка SFC, появится диалог «Создать новое действие» (рисунок 81).

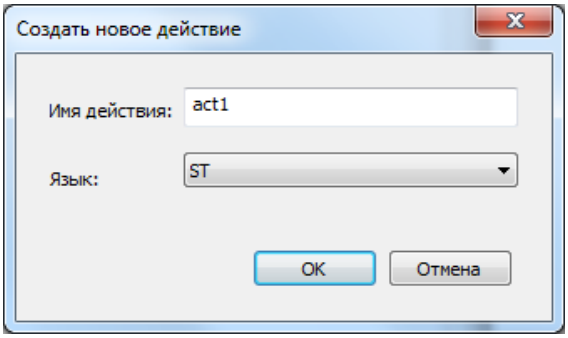


Рисунок 81 – Диалог «Создать новое действие»

В данном диалоге необходимо указать «Имя действия» (должно быть уникальным) и выбрать язык (ST, IL, FBD, LD), на котором будет написано данное действие. Если имя действия не заполнено будет выведено сообщение об ошибке (рисунок 82).

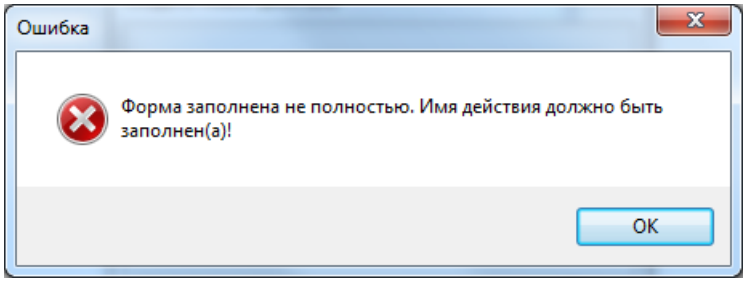


Рисунок 82 – Ошибка не заполнения имени действия при его добавлении

После того как действие добавлено, необходимо реализовать его код на текстовом или графическом языке, в зависимости от языка, который был выбран в диалоге «Создать новое действие» (рисунок 81). После добавления переходов и действий в дерево проекта они будут доступны для множественного использования.

Описание языка SFC, основных конструкций и примера его использования приведены в Приложении 7.

5.8 Панель редактирования ресурса

Панель редактирования ресурса (рисунок 83) содержит панель переменных и констант, которая позволяет определять глобальные переменные на уровне ресурса и панели, содержащие задачи и экземпляры.

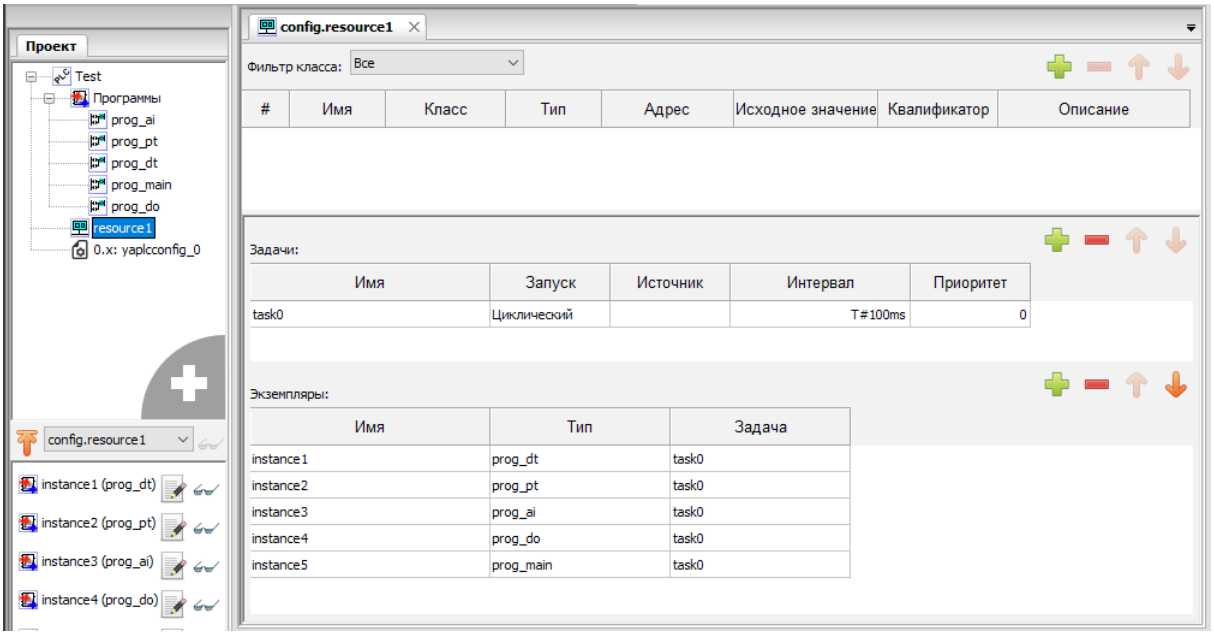


Рисунок 83 – Вкладка ресурс главной рабочей области

Добавление переменных в ресурс ничем не отличается от добавления переменных в программные модули, единственное исключение – переменные могут быть только класса «Глобальная».

Основной задачей данной панели является возможность добавить экземпляр, указать для него программный модуль типа «Программа», из ранее определённых в проекте, для поля «Тип» и выбрать задачу из добавленных в список «Задачи».

5.9 Панель редактирования типа данных

Панель редактирования типа данных (рисунок 84) позволяет определить различные параметры создаваемого пользовательского типа данных.

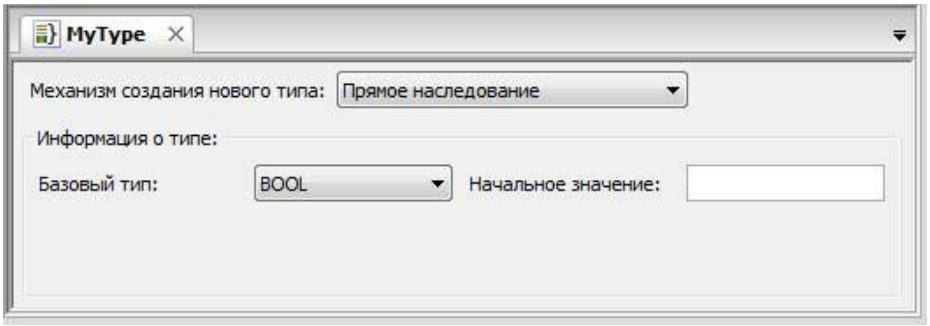


Рисунок 84 – Вкладка создания нового типа данных

Главным параметром является список под названием «Механизм создания нового типа», позволяющим выбрать следующие типы:

- прямое наследование;

- поддиапазон существующего типа (выделение диапазона значений стандартного типа);
- перечисляемый тип;
- массив;
- структура, позволяющая определять тип, основанный на объединении несколько типов.

Далее рассмотрены подробнее параметры для каждого из вышеперечисленных типов

5.9.1 Прямое наследование

При выборе «Прямое наследование» (рисунок 85), из списка указывается базовый тип и его начальное значение.

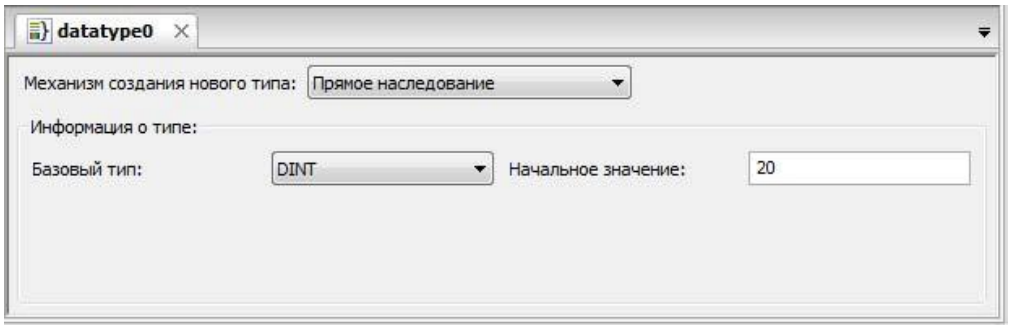


Рисунок 85 – Добавление псевдонима типа данных

Созданный тип представляет собой псевдоним (например, аналогично использованию typedef в языке C) уже существующего типа.

5.9.2 Поддиапазон существующего типа

В случае выбора механизма создания нового типа «Поддиапазон существующего типа», помимо базового типа и начального значения производится установка параметров «Минимум» и «Максимум» (рисунок 86), т.е. соответственно минимального и максимального значения, которое может принимать создаваемый тип данных.

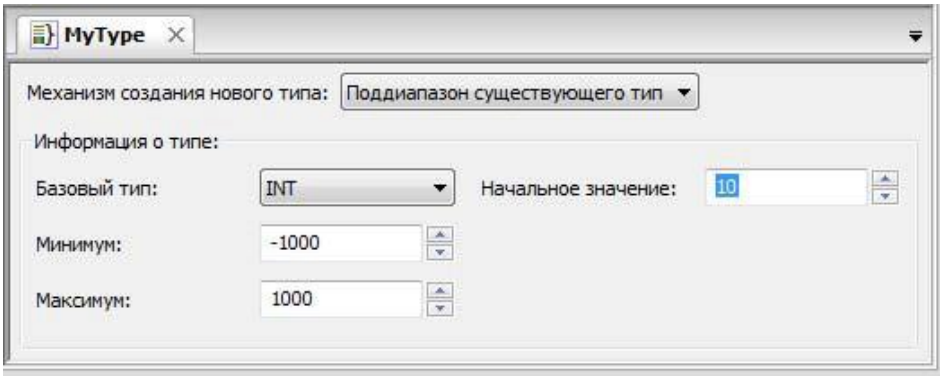


Рисунок 86 – Добавление нового типа данных, представляющего поддиапазон существующего типа

5.9.3 Перечисляемый тип

При выборе механизма создания нового типа «Перечисляемый тип» (рисунок 87), появится панель, содержащая таблицу, в которой можно задать список возможных значений данного перечисляемого типа.

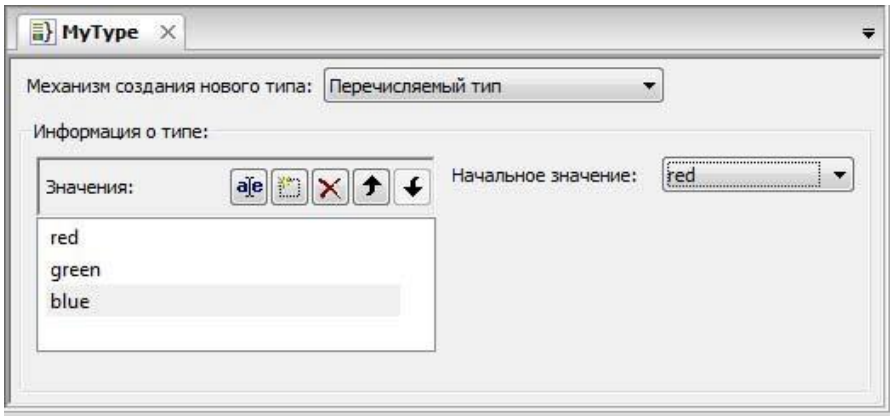


Рисунок 87 – Добавление перечисляемого типа данных

Добавление, редактирование, удаление, перемещение данных значений осуществляется с помощью кнопок, описание которых приведено в таблице 9.

Таблица 9 - Функции кнопок редактирования значений перечисляемого типа

Кнопка	Функция
	Редактировать выделенное поле
	Добавить новое поле
	Удалить выделенное поле
	Переместить выделенное поле на одну позицию вверх

Кнопка	Функция
	Переместить выделенное поле на одну позицию вниз

Также есть возможность задать начальное значение данного перечисляемого типа в поле «Начальное значение».

5.9.4 Массив

При выборе механизма создания нового типа «Массив» (рисунок 88) появится панель, в которой необходимо указать базовый тип, начальное значение, а так же размерность массива.

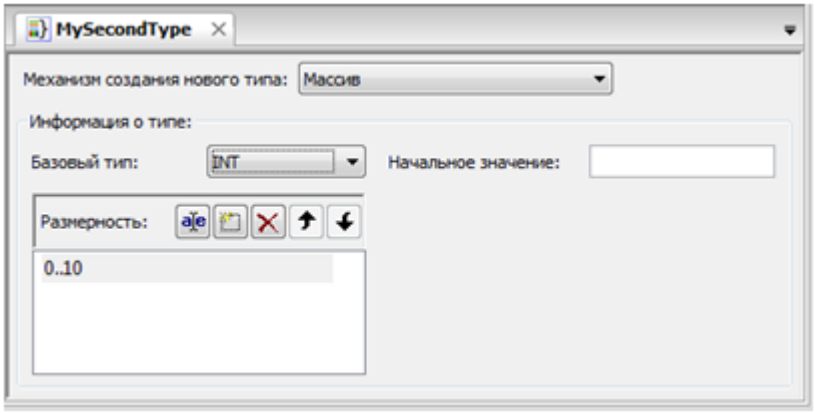


Рисунок 88 – Добавление типа данных «Массив»

Размерность массива задаётся в следующем формате:

<начальное значение>..<конечное значение>

5.9.5 Структура

При выборе механизма создания нового типа «Структура» (рисунок 89), в появившейся таблице необходимо добавить необходимое количество полей структуры. Каждое поле имеет своё имя, тип и начальное значение.

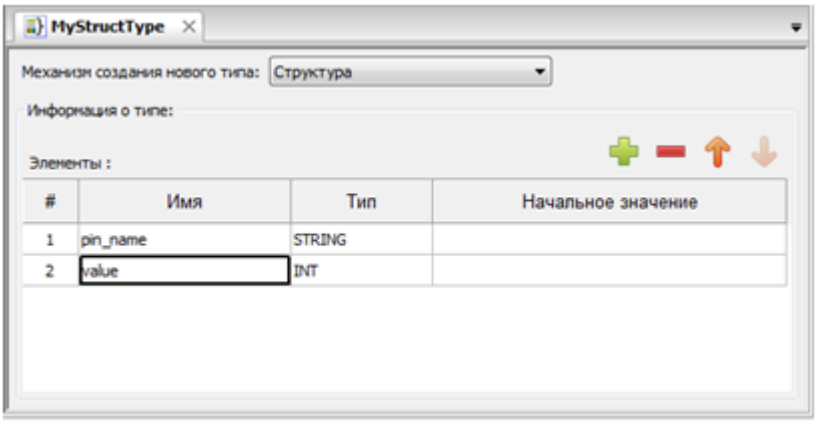


Рисунок 89 – Добавление типа данных «Структура»

Добавленные типы данных могут использоваться также как и стандартные при реализации алгоритмов и логики выполнения программных модулей.

Выше были рассмотрены варианты редактирования различных элементов, из которых состоит проект, согласно стандарту IEC 61131-3. Далее будет продолжено рассмотрение остальных компонент среды разработки Beremiz.

5.10 Панель экземпляров проекта

Панель экземпляров проекта (рисунок 90) обычно располагается слева в среде разработки Beremiz и отображаемые в ней экземпляры зависят от выбранного элемента в дереве проекта.

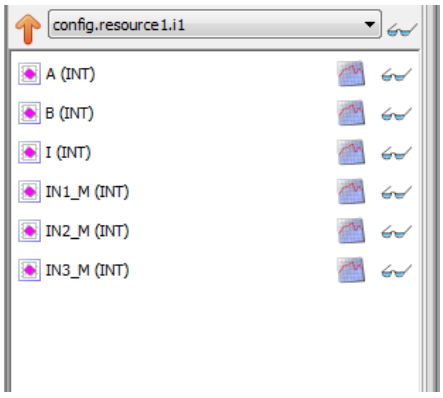


Рисунок 90 – Панель экземпляров проекта

При выборе в дереве проекта элемента, соответствующего ресурсу, в панели экземпляров проекта будут отображены экземпляры (см. п. 5.8), определённые в данном ресурсе, а так же глобальные переменные ресурса.

На рисунке 91 показано, как в панели редактирования ресурса определена глобальная переменная «globalValue» и два экземпляра для программных модулей «program0» и «program1»:

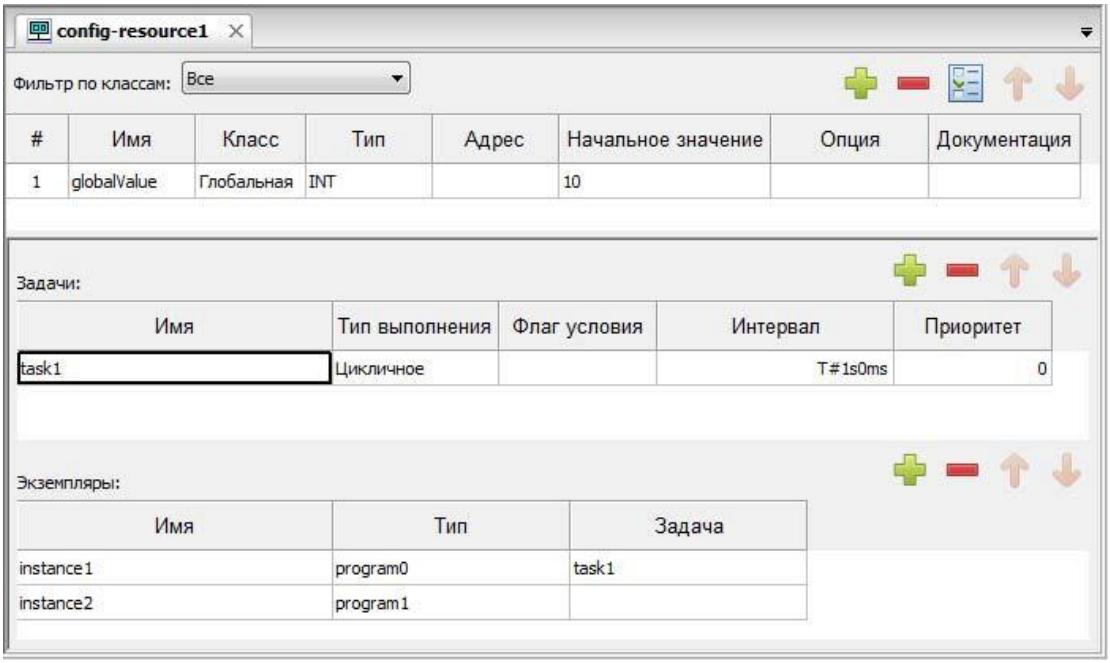


Рисунок 91 – Пример с двумя экземплярами проекта в панели редактирования ресурса

Соответственно, при выборе в дереве проекта ресурса, в котором определены экземпляры (описанные выше) и глобальная переменная, панель экземпляров будет выглядеть, как показано на рисунке 92:

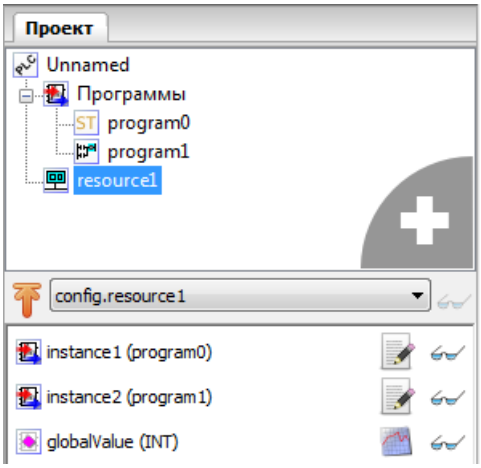
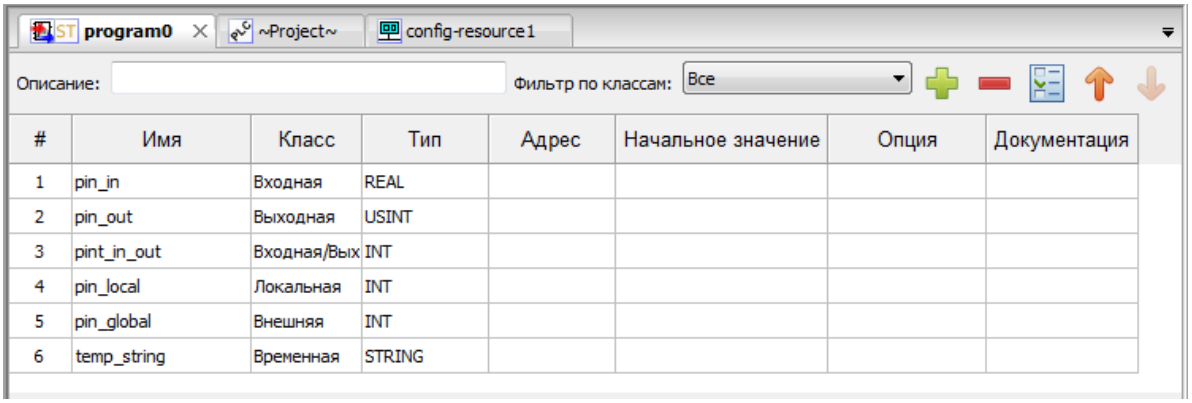


Рисунок 92 – Панель экземпляров при выборе элемента ресурса в дереве проекта

При выборе в дереве проекта элемента, соответствующего программным модулям «Программа» и «Функциональный блок» в панели экземпляров будут отображены переменные, определённые в них. Ниже на рисунке 93 приведён пример программного модуля типа «Программа» с именем «program0», в котором определено 6 переменных различных классов.



#	Имя	Класс	Тип	Адрес	Начальное значение	Опция	Документация
1	pin_in	Входная	REAL				
2	pin_out	Выходная	USINT				
3	pint_in_out	Входная/Вых	INT				
4	pin_local	Локальная	INT				
5	pin_global	Внешняя	INT				
6	temp_string	Временная	STRING				

Рисунок 93 – Программный модуля типа «Программа»

Соответственно, при выборе в дереве проекта данного программного модуля в панели экземпляров будут отображены, определённые выше переменные (рисунок 94).

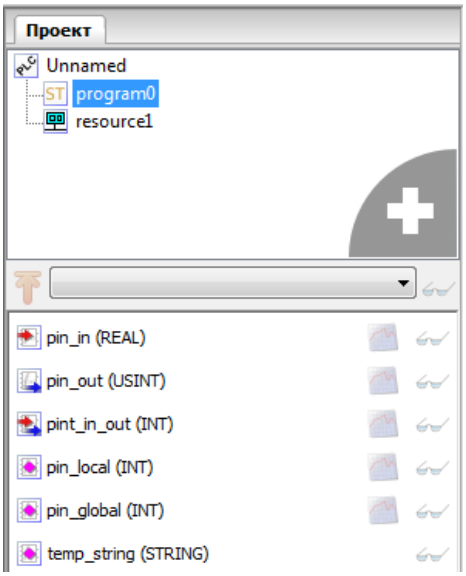


Рисунок 94 – Панель экземпляров проекта при выборе в дереве проекта программного модуля типа «Программа»

В случае выбора других элементов в дереве проекта, панель отладки будет пустой. Как можно заметить, с правой стороны от каждого элемента в панели отладки располагаются кнопки, назначение которых описано в таблице 10.

Таблица 10 - Функции кнопок панели экземпляра проекта

Кнопка	Функция
	Отображение графика изменения значения переменной во времени в режиме отладки
	Запуск режима отладки для экземпляра

В случае нажатия кнопки запуска режима отладки, для экземпляра программы, написанной на одном из графических языков (FBD, LD или SFC), откроется вкладка с панелью, на которой диаграмма будет отображена в режиме отладки (см. п. 8). Если кнопка запуска режима отладки нажимается для элемента переменной, то переменная будет добавлена в панель отладки (см. п. 5.14).

Описанные выше кнопки доступны только в режиме отладки прикладной программы. Про данный режим подробнее рассказывается в п. 8.

5.11 Панель библиотеки функций и функциональных блоков

Панель библиотеки функций и функциональных блоков (рисунок 95), как правило, располагается справа в среде разработки Beremiz. Она содержит коллекцию стандартных функций и функциональных блоков, разделённых по разделам в соответствии с их назначением, которые доступны при написании алгоритмов и логики работы программных модулей.

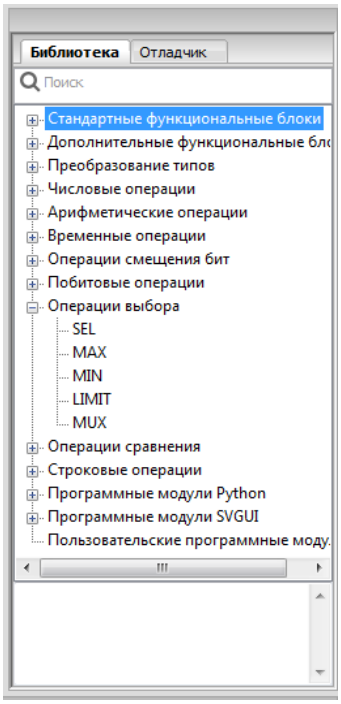


Рисунок 95 – Панель библиотеки функций и функциональных блоков

Выделены следующие разделы для функций и функциональных блоков: стандартные, дополнительные, преобразования типов данных, операций с числовыми данными, арифметических операций, временных операций, побитовых и смещения бит, операций выбора, операций сравнения, строковых операций, модулей «Python» и «SVGUI».

Помимо стандартных функций и функциональных блоков, данная панель содержит раздел «пользовательские программные модули». В него попадают функции и

функциональные блоки, добавленные в конкретный проект, т.е. содержащиеся в дереве проекта.

Использование данных функций и функциональных блоков осуществляется перетаскиванием необходимого блока с помощью зажатой левой кнопки мыши (Drag&Drop) в область редактирования: либо текстовый редактор, либо графический редактор.

Имеется специальное поле поиска функционального блока по имени. Более подробное описание каждого функционального блока приведено в Приложении 2.

5.12 Отладочная консоль

Панель, содержащая отладочную консоль (рисунок 96), как правило, располагается в правом нижнем углу среды разработки Beremiz.

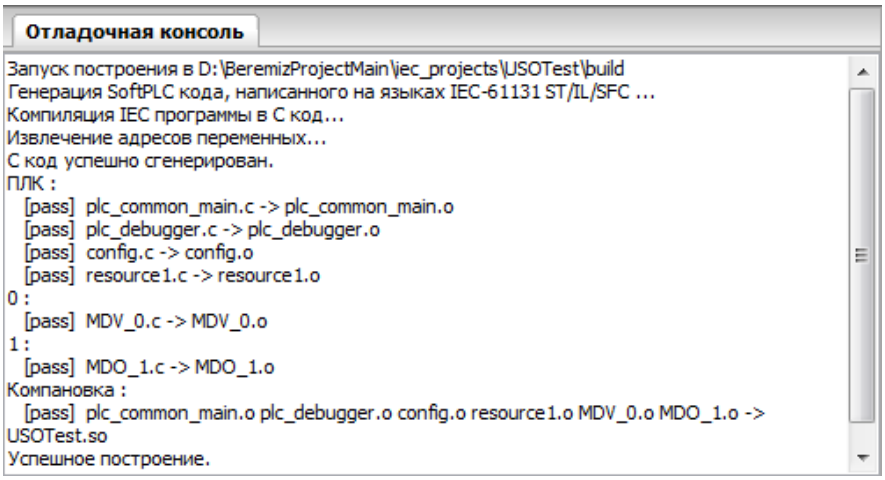


Рисунок 96 – Успешная сборка в отладочной консоли

Консоль служит для отображения:

- результатов генерации ST и C кода;
- результатов компиляции и компоновки прикладной программы;
- процесса соединения и передачи прикладной программы на целевое устройство;
- различных промежуточных манипуляций в процессы создания прикладной программы.

Цвет предупреждений – красный, критических ошибок – красный с желтым выделением (рисунок 97).

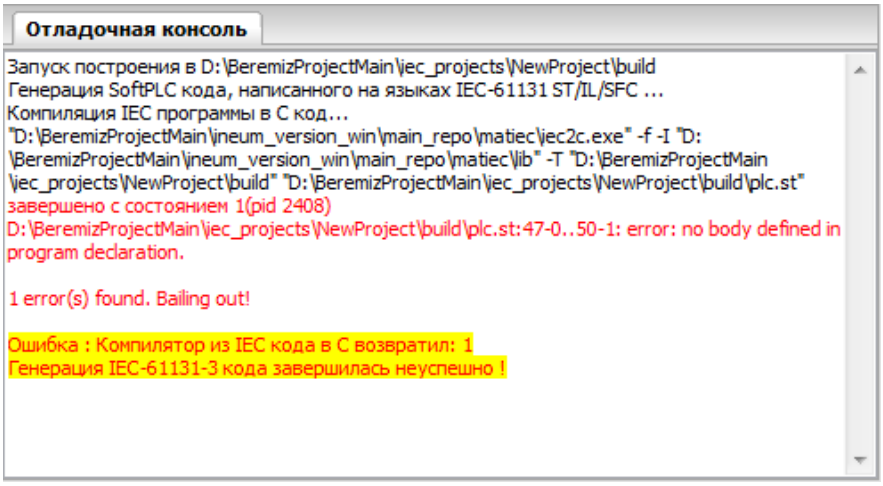


Рисунок 97 – Ошибка сборки проекта в отладочной консоли

5.13 Поиск элементов в проекте

Для поиска элемента в проекте используется диалог «Поиск в проекте» (рисунок 98), вызов которого возможен с помощью главного меню программы (п. 5.1) или панели инструментов (п. 5.2).

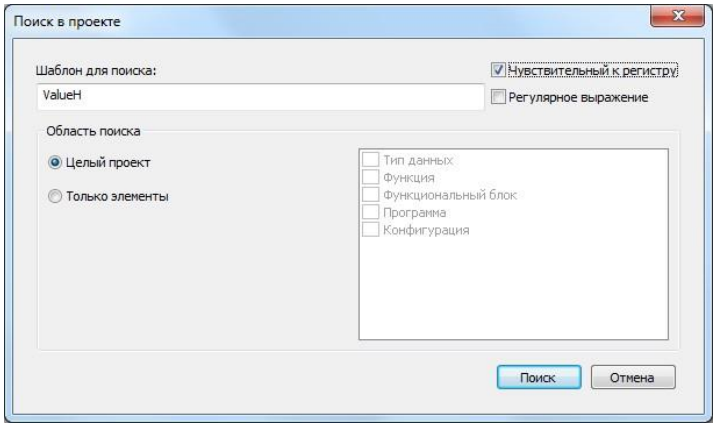


Рисунок 98 – Диалог поиска в проекте

В появившемся диалоге можно установить различные параметры поиска: шаблон поиска, область поиска, чувствительность к регистру при поиске, а так же записать шаблон поиска в виде регулярного выражения. После того как все параметры установлены, необходимо нажать кнопку «Поиск» в этом диалоге. Ниже на рисунке 99 приведен пример поиска элемента с именем «ValueH».

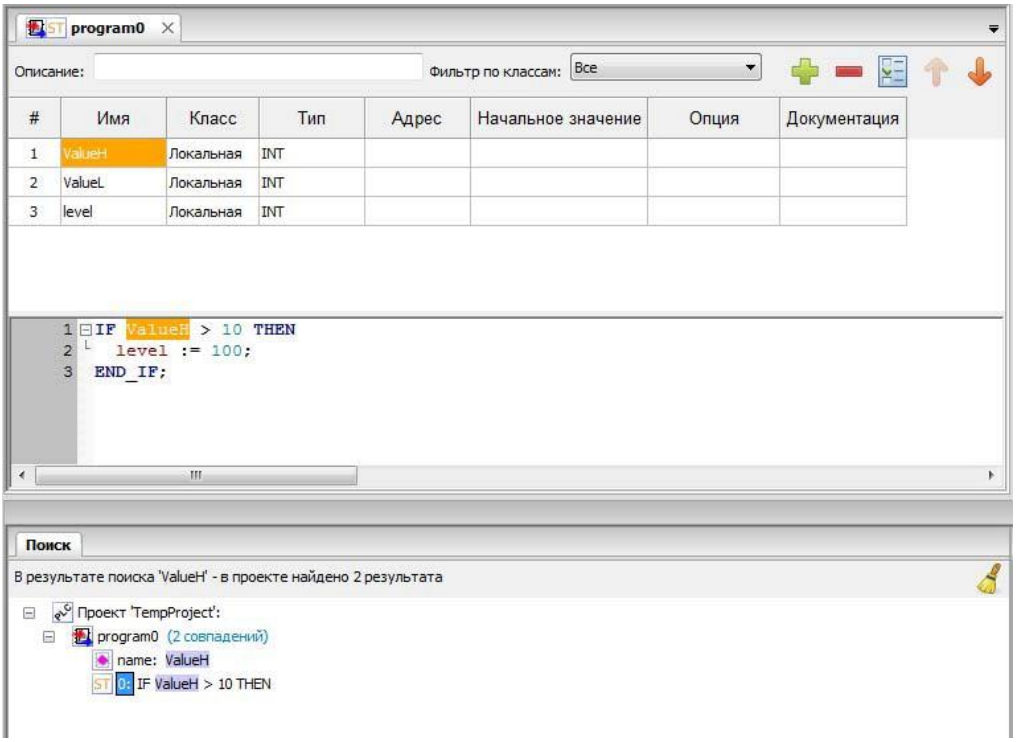


Рисунок 99 – Результат примера поиска элемента в проекте

Результат поиска выводится в иерархической структуре. При двойном щелчке по одному из результатов – данный элемент будет выделен в проекте оранжевым цветом.

5.14 Панель отладки

Панель отладки располагается в правой части среды разработки Beremiz (рисунок 100).

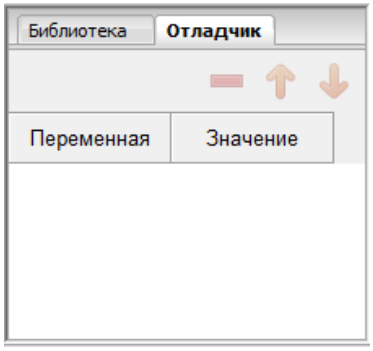


Рисунок 100 – Панель отладки

Данная панель представляет собой таблицу с двумя столбцами «Переменная» и «Значение». Соответственно, столбец «Переменная» содержит экземпляры переменных, значения которых во время исполнения, отображаются в поле «Значение» и могут изменяться. Добавление переменных осуществляется с помощью панели экземпляров проекта (см. п. 5.10). Изменение значений переменной во время отладки прикладной

программы осуществляется нажатием правой клавишей мыши в поле «Значение» интересующей переменной и в появившемся всплывающем меню (рисунок 101) выбирается «Установить значение».

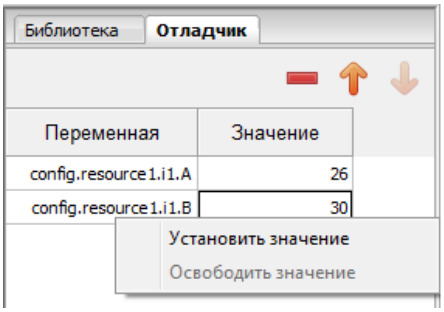


Рисунок 101 – Контекстное меню управления значением

Далее появится диалог ввода значения для выбранной переменной (рисунок 102).

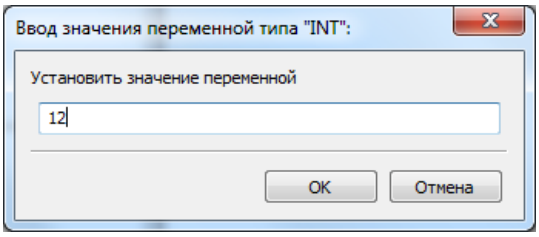


Рисунок 102 – Установка значения переменной во время отладки

Для того чтобы отменить установленное значение для переменной, необходимо нажать во всплывающем меню «Освободить значение».

На данной панели так же присутствуют кнопки, подобные кнопкам в панели переменных и констант, позволяющие перемещать и удалять добавленные переменные.

5.15 Панель графика изменения значения переменной в режиме отладки

Данная панель (рисунок 103) открывается в отдельной вкладке в случае, если в панели экземпляров проекта (п. 5.10) для переменной была нажата кнопка отображения графика изменения значения переменной во времени в режиме отладки.

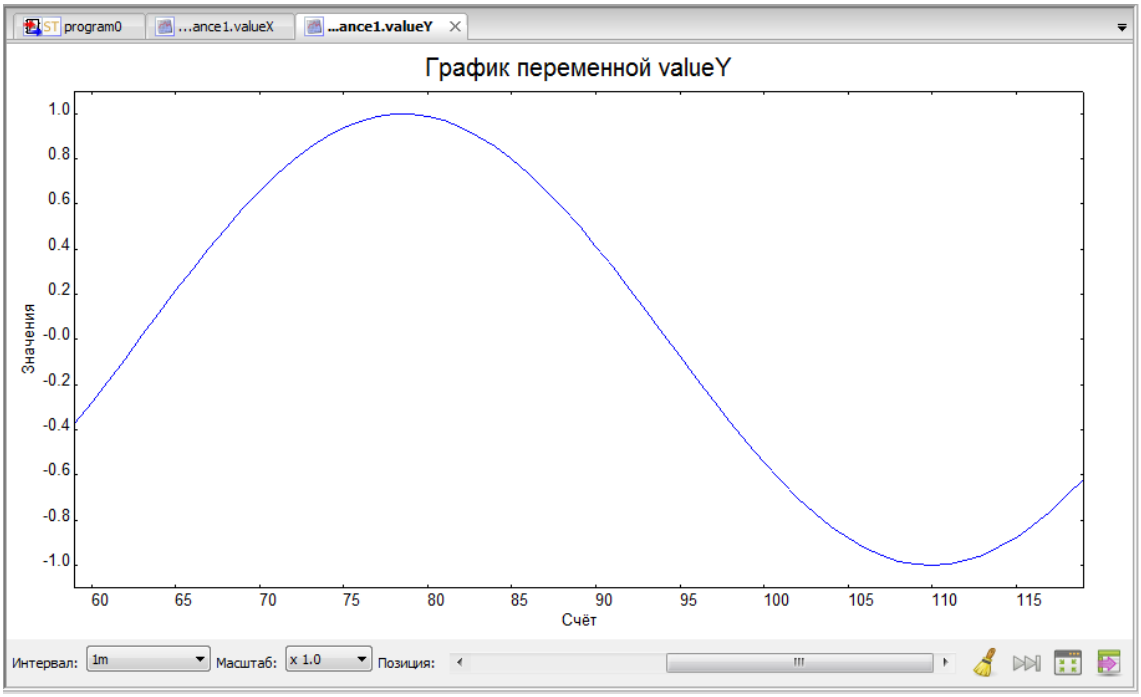


Рисунок 103 – График изменения значения переменной во время отладки

На данной панели можно установить:

- Интервал – временной отрезок, за который отображается изменений графика;
- Масштаб – задание приближения отображения графика;
- Позиция – перемещение по отображению графика, от начала и до конца.

Также на данной панели в правом нижнем углу располагаются вспомогательные кнопки. Описание данных кнопок приведено в таблице 11.

Таблица 11 - Функции кнопок управления графиком изменения значения переменной

Кнопка	Функция
	Очистить график
	Перейти к последней точке графика справа
	Сбросить масштаб в исходное состояние
	Экспортировать график в буфер обмена

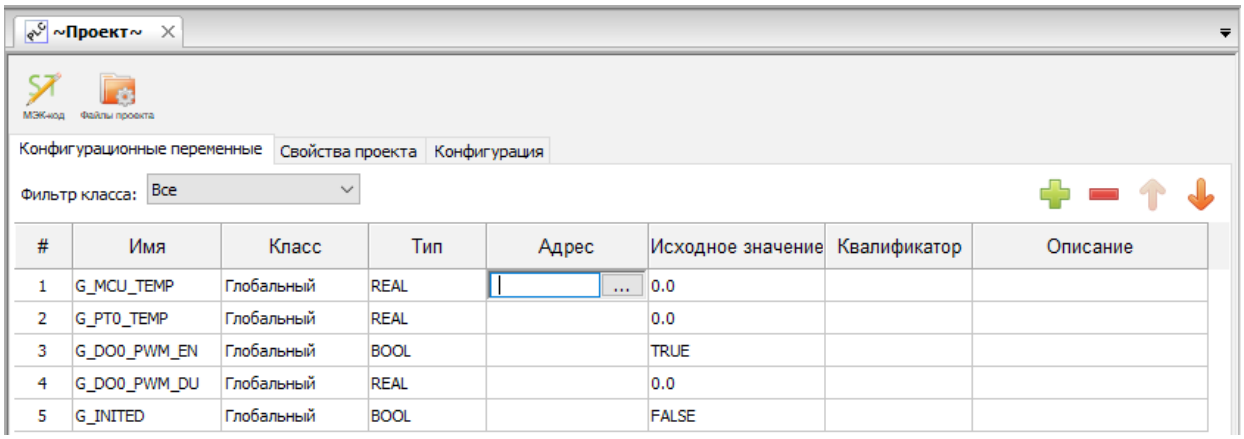
6 ОСНОВНЫЕ КОМПОНЕНТЫ СРЕДЫ РАЗРАБОТКИ

Среда разработки Beremiz предоставляет интерфейс, позволяющий связывать внешние источники данных, такие как модули УСО (их параметры, состояния) с программными модулями (в частности с их переменными), написанными на языках высокого уровня (IEC 61131-3), из которых состоит прикладная программа.

6.1 Связывание регистров внешних модулей с переменными проекта

Связывание внешних переменных с переменными программных модулей осуществляется с помощью адресов переменных (п. 6.2). Данная операция осуществляется следующим образом:

- выбирается программа, переменные которой необходимо связать с регистрами внешних модулей;
- на панели переменных и констант (п. 5.4) выбирается переменная, которую необходимо связать;
- выполняется двойной щелчок мышью по полю «Адрес» данной переменной для того, чтобы появилась кнопка «…» (рисунок 104);



#	Имя	Класс	Тип	Адрес	Исходное значение	Квалификатор	Описание
1	G_MCU_TEMP	Глобальный	REAL		0.0		
2	G_PT0_TEMP	Глобальный	REAL		0.0		
3	G_DO0_PWM_EN	Глобальный	BOOL		TRUE		
4	G_DO0_PWM_DU	Глобальный	REAL		0.0		
5	G_INITED	Глобальный	BOOL		FALSE		

Рисунок 104 – Поле «Адрес» панели переменных и констант

- нажав появившуюся кнопку «…» в поле «Адрес» появится диалог «Просмотр адресов» (рисунок 105), в котором отображаются доступные для связывания регистры внешних модулей;

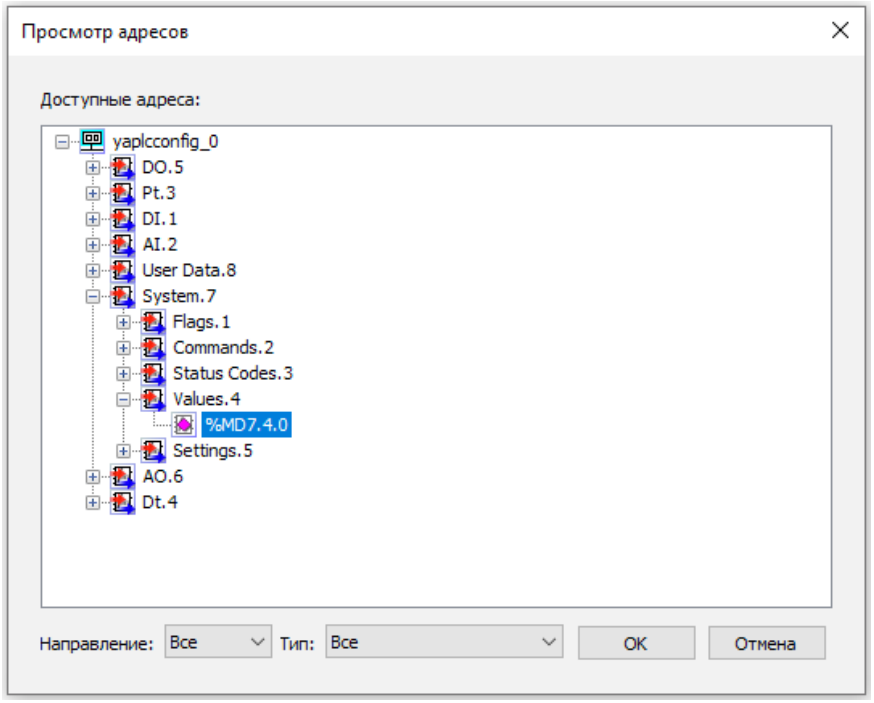


Рисунок 105 – Диалог просмотра адресов

- после выбора необходимой переменной и нажатия кнопки «OK» будет выдано диалоговое окно выбора класса переменной: Вход, Выход, Память (п. 6.2, рисунок 106);

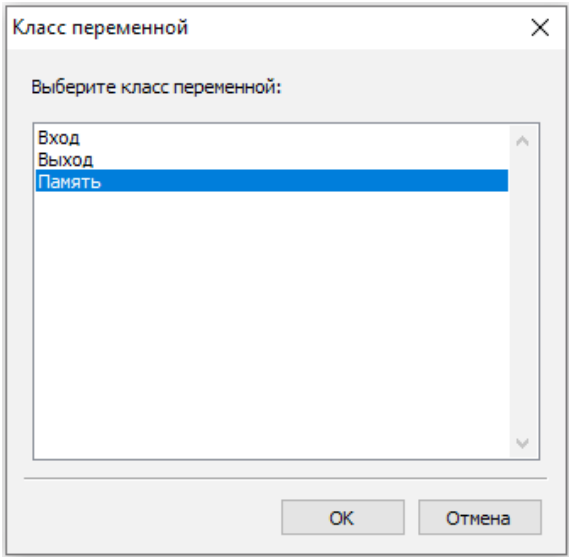


Рисунок 106 – Диалог выбора класса переменной

- после выбора класса и нажатия кнопки «OK» поле адреса соответствующей переменной будет заполнено адресом внешней переменной (рисунок 107).

#	Имя	Класс	Тип	Адрес	Исходное значение	Квалификатор	Описание
1	G_MCU_TEMP	Глобальный	DWORD	%MD7.4.0	0.0		
2	G_PT0_TEMP	Глобальный	REAL		0.0		
3	G_DO0_PWM_EN	Глобальный	BOOL		TRUE		
4	G_DO0_PWM_DU	Глобальный	REAL		0.0		
5	G_INITED	Глобальный	BOOL		FALSE		

Рисунок 107 – Добавленный адрес переменной

Адрес регистра внешнего модуля также можно задать вручную. Список адресов регистров приводится в руководстве по эксплуатации модуля или целевой платформы.

6.2 Адреса регистров внешних модулей

Адрес регистра задается в следующем формате:

%<код Класса><код Типа><код Группы регистров>.<Адрес>

Коды классов:

- I –вход,
- Q – выход,
- M – память.

В пределах класса адреса распределяются по типам данных и группам (таблица 12).

Таблица 12 – Коды поддерживаемых типов данных

Код	**Тип**		**Диапазон значений**	**Размер**	
	IEC	**Си**		**биты**	**байты**
X	BOOL	uint8	0 … 255 (0, 1)	8	1
B	BYTE USINT SINT	uint8 int8	0 … 255 -128 … 127	8	1
W	WORD UINT INT	uint16 int16	0 … 65535 -32768 … 32767	16	2
D	DWORD UDINT DINT REAL	uint32 int32 float	0 … 4294967295 -2147483648 … 2147483647 $3.4\cdot10^{-38}$ … $3.4\cdot10^{38}$	32	4
L	LWORD ULINT LINT LREAL	uint64 int64 double	0 … $(2^{64}-1)$ $-(2^{63}-1)$ … $(2^{63}-1)$ $1.7\cdot10^{-308}$ ... $1.7\cdot10^{308}$	64	8

Например, на рисунке 107 адрес %MD7.4.0 – «Внутренняя температура ПЛК, °C», где:

- M – код класса «Память»,
- D – код типа REAL (из руководства на целевую платформу),
- 7 – код группы «Системные регистры»,
- 4 – код подгруппы «Значения/Показания»,
- 0 – адрес регистра.

Другой пример, адрес %IX1.3.1.1 – «Нормальный дискретный вход 3 (выкл./вкл.)», где:

- I – код класса «Вход»,
- X – код типа BOOL,
- 1 – код группы «Дискретные входы»,
- 3 – номер канала (входа),
- 1 – код подгруппы «Нормальный дискретный вход»,
- 1 – адрес регистра.

7 ОСНОВНЫЕ КОМПОНЕНТЫ СРЕДЫ РАЗРАБОТКИ

В данном разделе рассмотрены основные приёмы работы в среде разработки Beremiz, которые необходимы при создании прикладной программы.

7.1 Создание нового проекта

Новый проект создаётся с помощью главного меню «Файл» – «Новый» (рисунок 108), либо с помощью кнопки «Новый» на панели управления (рисунок 109).

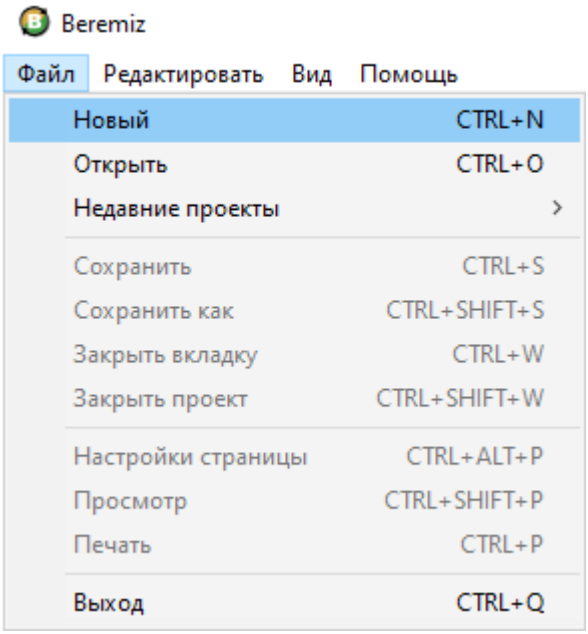


Рисунок 108 – Создание нового проекта через меню «Файл»

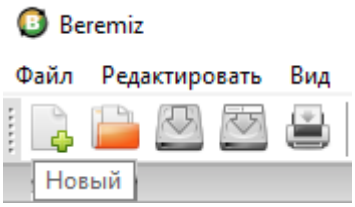


Рисунок 109 – Создание нового проекта через кнопку «Новый» панели управления

Далее появится диалог (рисунок 110), в котором необходимо выбрать папку, где будет храниться данный проект.

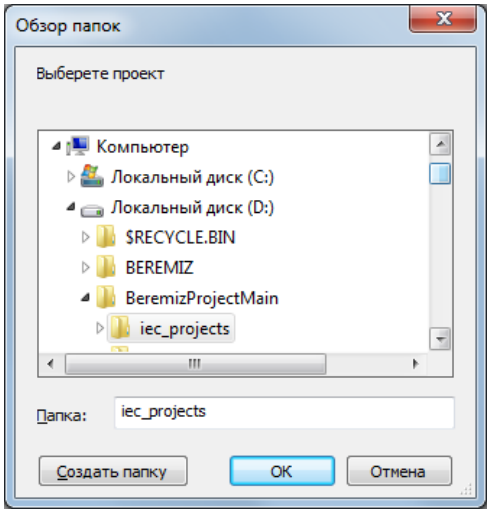


Рисунок 110 – Диалог выбора папки для нового проекта

Папка должна быть обязательно пустой и не защищена от записи. Если в папке уже есть файлы, будет выдана соответствующая ошибка. В созданной папке будут сохранены следующие файлы и папки:

- beremiz.xml – в данном XML файле сохраняются настройки специфичные для среды разработки Beremiz относительно проекта;
- plc.xml – в данном XML файле сохраняется полное описание проекта: всех программных модулей, ресурсов, пользовательских типов данных, данных о проекте, настроек редакторов графических языков IEC 61131-3;
- директория с настройками плагинов внешних модулей УСО;
- директория «build», которая хранит генерируемый ST и C код, а также получаемый исполняемый бинарный файл.

7.2 Настройка проекта

Как правило, первым шагом после создания проекта является его настройка, включающая в себя задание глобальных переменных, установку параметров компиляции и компоновки, а также заполнение данных о проекте. Вызов панели настройки проекта осуществляется при выборе (двойном щелчке левой кнопкой мыши) корневого элемента дерева проекта, который по умолчанию, сразу после создания проекта называется «Unnamed», как показано на рисунке 111:

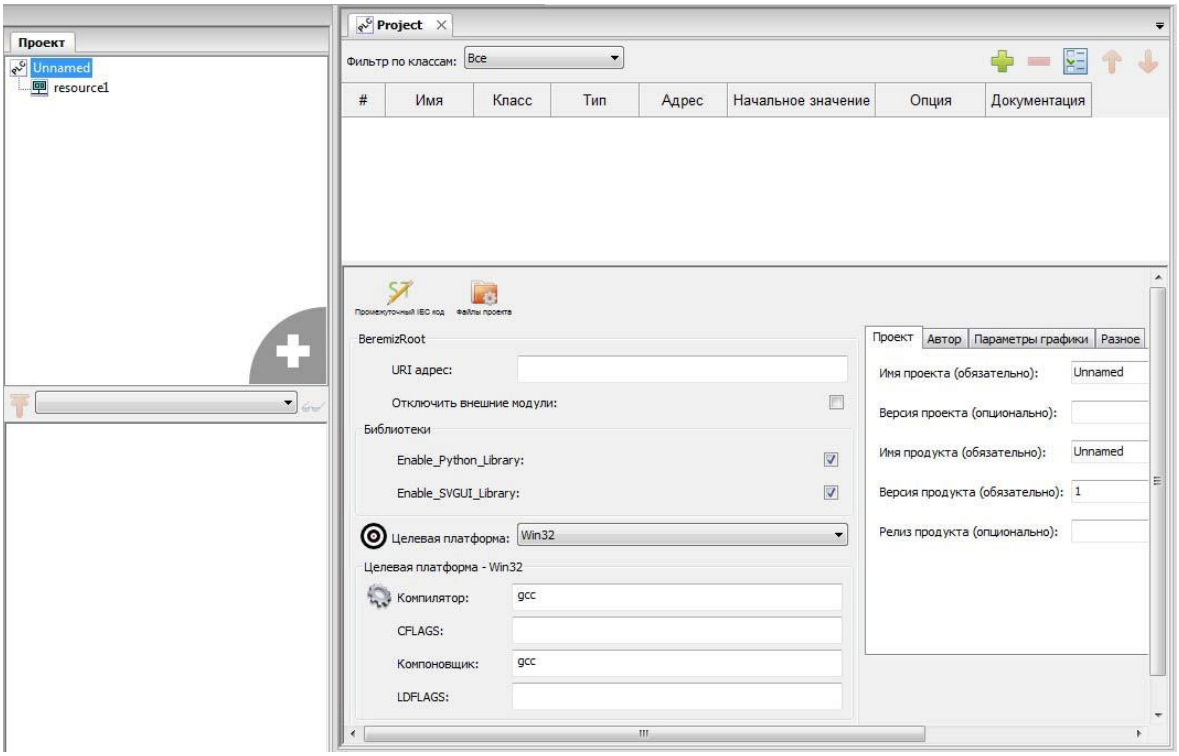


Рисунок 111 – Вызов панели настройки проекта с помощью корневого элемента дерева проекта

7.2.1 Глобальные переменные проекта

Глобальные переменные позволяют программным модулям типа «Программа» и «Функциональный блок» использовать общие переменные, которые будут определены в глобальной области видимости проекта.

Ниже, на рисунок 112, в панели переменных и констант создана глобальная переменная «globalValue» с начальным значением 10, с помощью кнопки «Добавить переменную» (таблица 3).

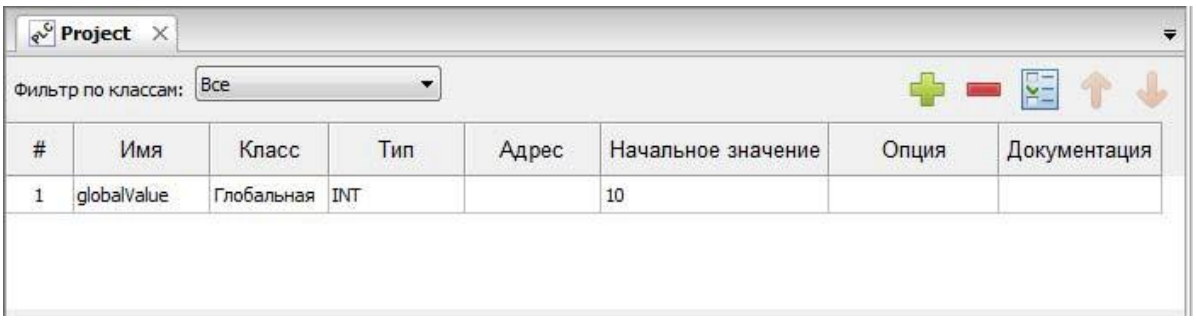


Рисунок 112 – Объявление глобальный переменной проекта

Для того чтобы к данной глобальной переменной можно было обращаться из программных модулей типа «Программа» или «Функциональный блок» необходимо в их панели редактирования в панели переменных и констант создать переменную с таким же

именем, как и ранее объявленная глобальная, и установить её класс «Внешняя». На рисунке 113 приведён пример объявления в программном модуле «program0» переменной «globalValue» класса «Внешняя», типа INT и изменение её значения с помощью языка ST.

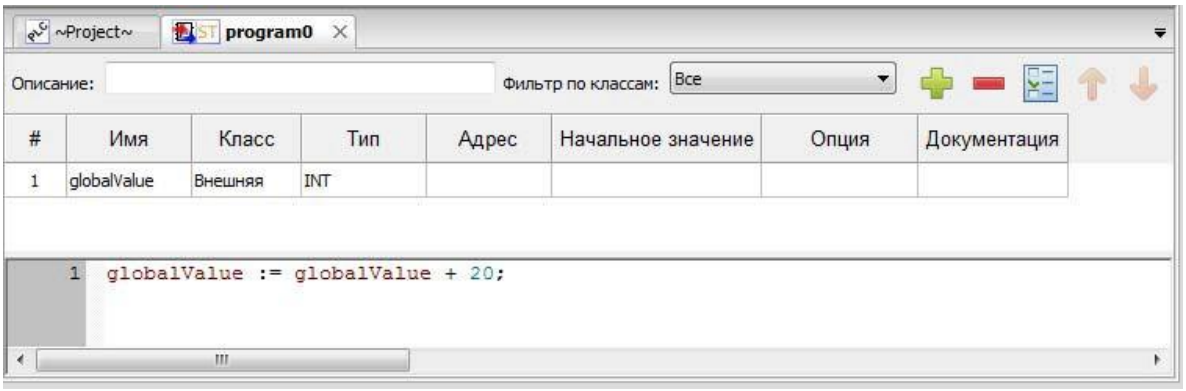


Рисунок 113 – Использование глобальной переменной в программном модуле «program0»

На рис. 114 в программном модуле «program1» также определена переменная «globalValue» класса «Внешняя» и изменение её значения с помощью языка ST.

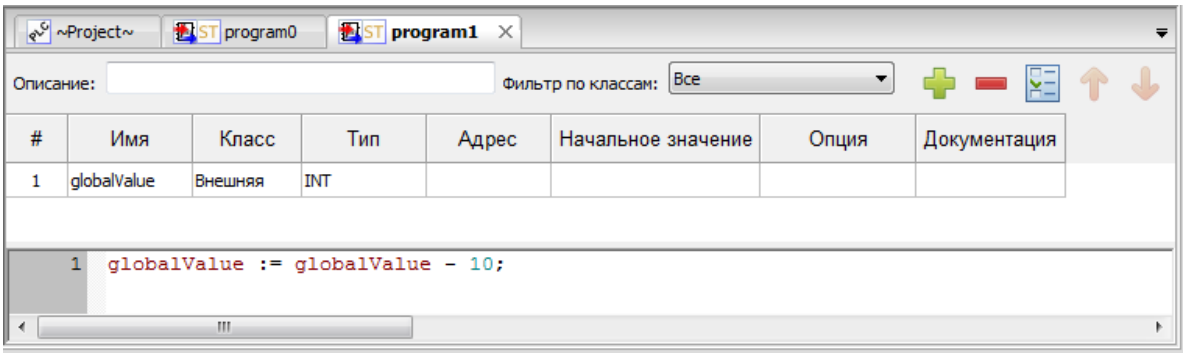


Рисунок 114 – Использование глобальной переменной в программном модуле «program1»

Соответственно, переменная «globalValue» будет изменяться во время выполнения в двух программных модулях: «program0» и «program1».

7.2.2 Настройки сборки проекта и соединения с целевым устройством

Для использования написанной прикладной программы необходимо её собрать (скомпилировать и скомпоновать), т.е. получить исполняемый файл и передать на целевое устройство для отладки или просто исполнения. В связи с этим основными настройками являются: «URI адрес» целевого устройства и целевая платформа, указывающая архитектуру платформы целевого устройства. На рис. 115 данные параметры выделены красным цветом.

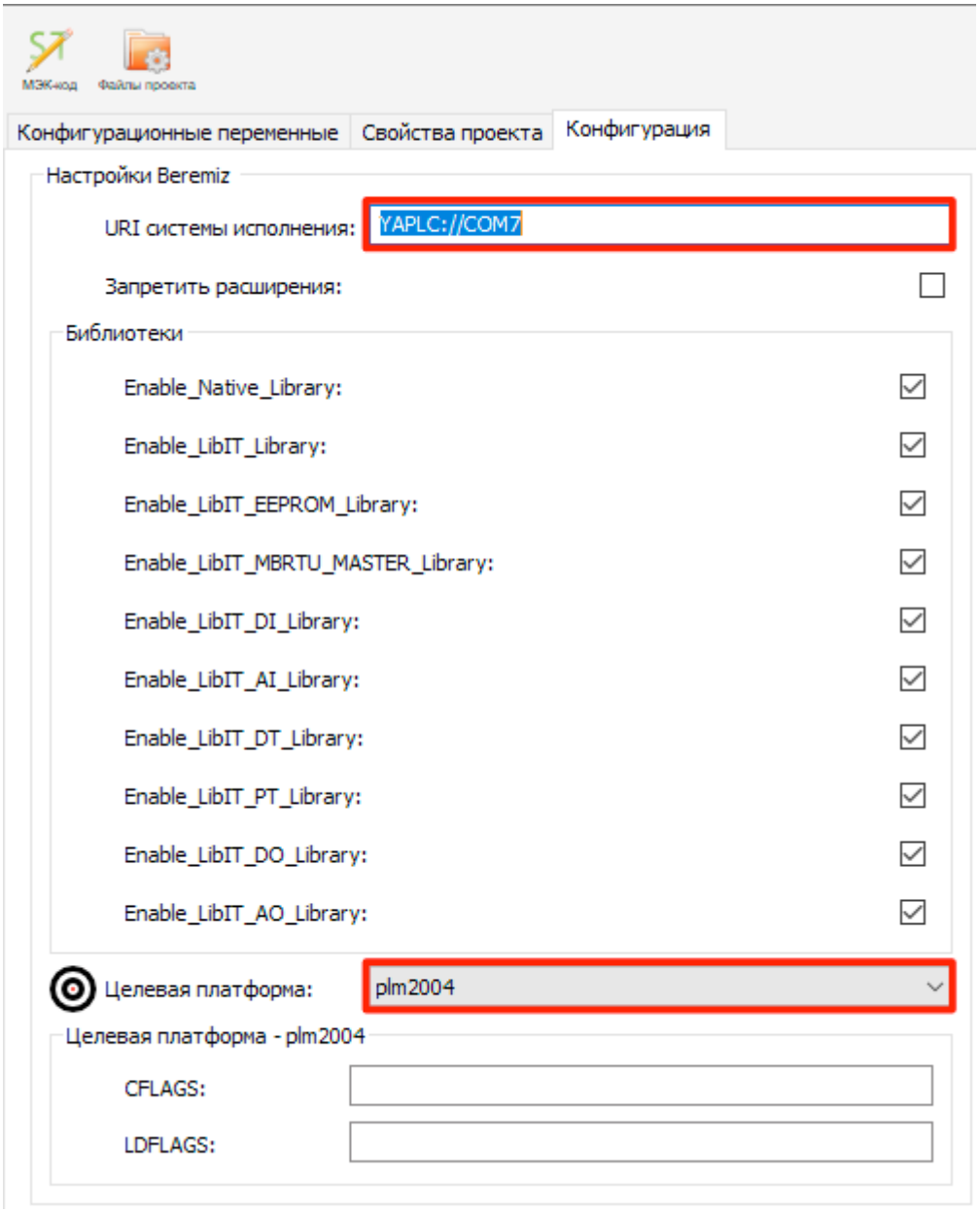


Рисунок 115 – Задание URI-адреса системы исполнения и типа целевой платформы

«URI адрес» указывается в формате:

YAPLC://<номер COM-порта>

где, YAPLC – это драйвер, входящий в состав ИСР Beremiz и предоставляющий работу из ИСР с целевым устройством по последовательному порту: загрузка проекта в целевое устройство, отладка проекта в режиме реального времени; <номер COM-порта> - символьный номер COM-порта.

Целевая платформа (например, plm2004) выбирается из списка «Целевая платформа».

7.2.3 Данные о проекте

При создании нового проекта, все обязательные поля в настройках информации о проекте заполняются значениями по умолчанию. Рекомендуется заменить данные настройки по умолчанию на релевантную информацию (рисунок 116), позволяющую удобным образом различать проекты.

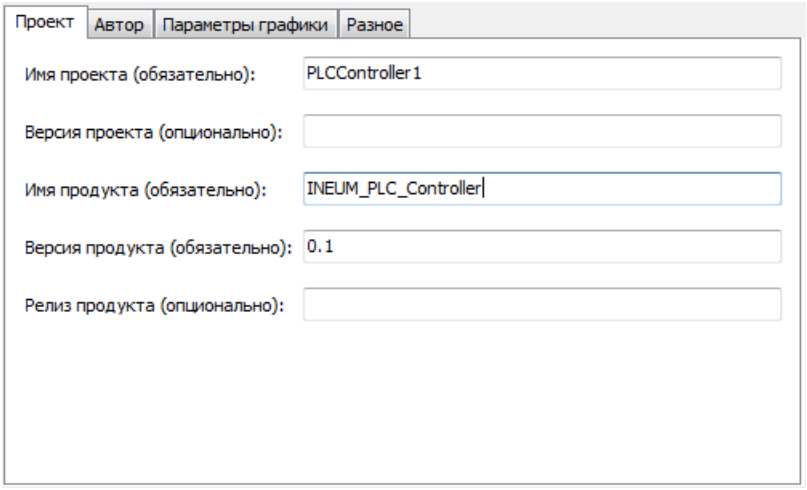


Рисунок 116 – Указание данных о проекте

Большая часть данных в информации проекте являются необязательным для заполнения, но некоторые должны быть всегда заполнены. Это указывается в подсказках в именовании каждого пункта.

После задания настроек проекта, как правило, следует добавление в проект необходимых программных модулей (функций, функциональных блоков и программ), реализация их алгоритмов и логики работы с помощью текстовых и графических языков стандарта IEC 61131-3.

7.3 Программные модули

Добавление программных модулей (программ, функций, функциональных блоков) осуществляется с помощью всплывающего меню дерева проекта, в котором необходимо выбрать пункт «Функция», «Функциональный блок» или «Программа». Далее появится диалог «Создать новый программный модуль» (рисунок 117).

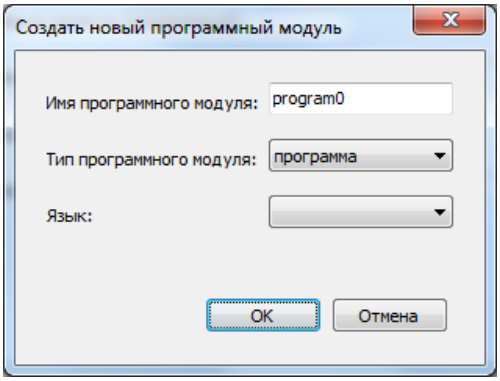


Рисунок 117 – Диалог добавления программного модуля

В данном диалоге три поля:

- Имя программного модуля;
- Тип программного модуля;
- Язык.

Имя, присвоенное по умолчанию, может быть заменено на имя, соответствующее назначению данного программного модуля. В зависимости от того, какой программный модуль был выбран во всплывающем меню, в поле «Тип программного модуля» будет подставлено именование данного программного модуля. В поле «Язык» необходимо выбрать из списка (рисунок 118) один из языков стандарта IEC 61131-3 (IL, ST, LD, FBD, SFC), на котором будет реализованы алгоритмы и логика работы данного добавляемого программного модуля.

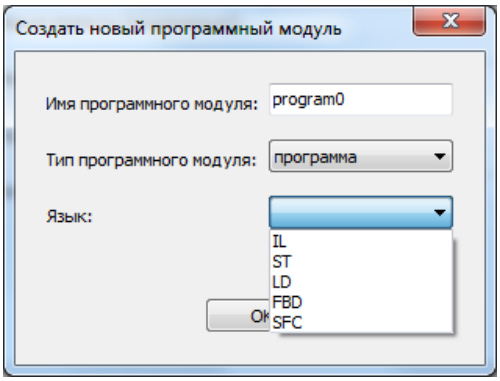


Рисунок 118 – Выбор языка для программного модуля

Далее расмотрено добавление каждого программного модуля в отдельности.

7.3.1 Программа

Ниже будет приведён пример добавления в проект программы, написанной на языке FBD. Логика и алгоритм работы данного программного модуля следующие: определены две глобальные переменные «globalValue» и «globalLevel», если значение «globalValue»

больше 10.0, то присвоить переменной «globalLevel» значение 100, в противном случае присвоить «globalLevel» значение 50.

Сначала следует добавление программы в проект, осуществляемое с помощью меню дерева проекта, выбором пункта «Программа» (рисунок 119):

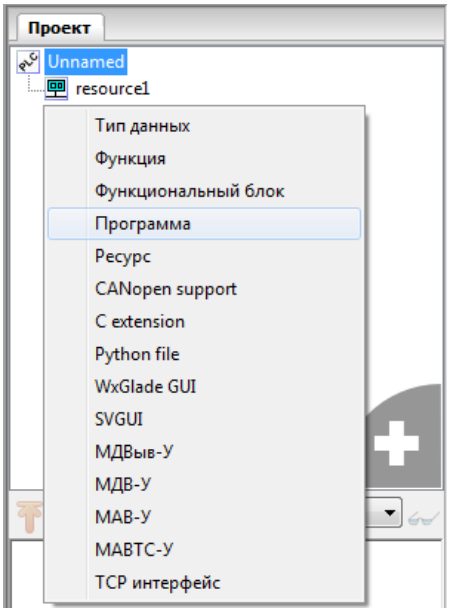


Рисунок 119 – Выбор в дереве проекта добавления программы

В появившемся диалоге (рисунок 120) выбирается язык FBD и нажимается кнопка «OK».

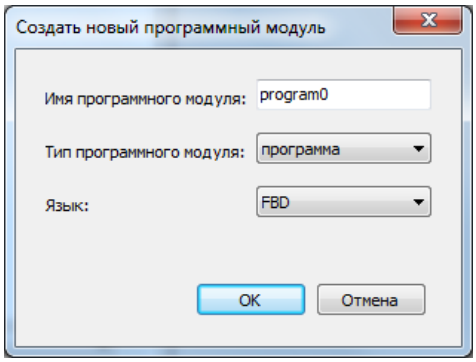


Рисунок 120 – Диалог добавления программы

Далее в открывшейся вкладке с панелью редактирования данного программного модуля в панели переменных и констант (рисунок 121) добавляются переменные: «globalValue» типа REAL, класса «Внешняя» и «globalLevel» типа INT, класса «Внешняя».

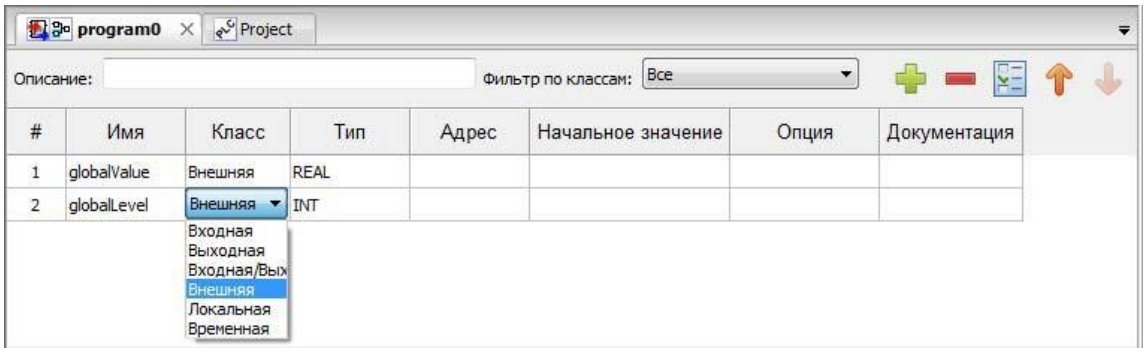


Рисунок 121 – Объявление в программе внешних переменных

Предполагается, что эти переменные уже определены глобальными переменными проекта в панели редактирования проекта (рисунок 122), как описывается в п. 7.2.1.

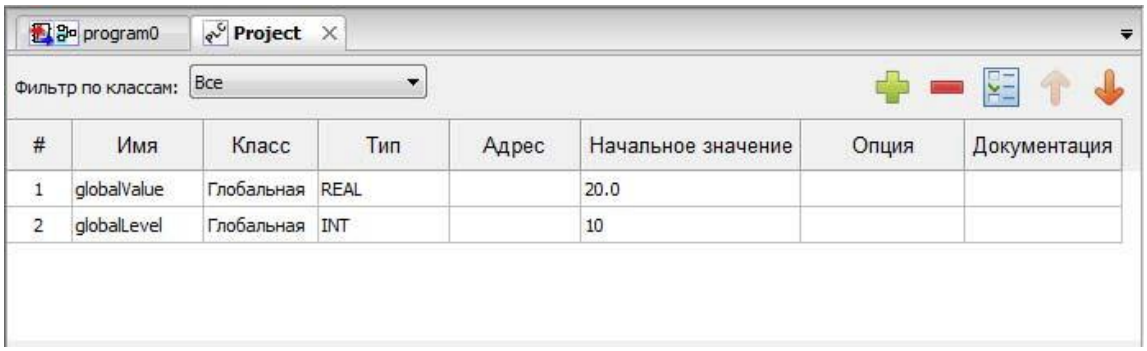


Рисунок 122 – Глобальные переменные проекта

Далее необходимо обратиться к редактору языка FBD. Для написания алгоритма и логики выполнения данной программы будут добавлены две функции: «GT» и «SEL». Функция «GT» обозначает сравнение «Больше чем» и находится во вкладке «Операции сравнения». Она может содержать от 2 до 20 входных значений (в данном примере их будет 2) и одно выходное значение «OUT». Если значение «IN1» больше значения «IN2», то на выходе «OUT» будет TRUE, в противном случае FALSE.

Функция «SEL» обозначает «Выбор одного из двух значений» и находится во вкладке

«Операции выбора». Она содержит три входных переменных «G», «IN0», «IN1» и одну выходную «OUT». Если «G» равно 0 (или FALSE), то выходной переменной «OUT» присваивается значение «IN0». Если «G» равно 1 (или TRUE), то выходной переменной «OUT» присваивается значение «IN1».

Добавление данных функций удобнее осуществить переносом соответствующей функции с помощью мыши (Drag&Drop) из панели библиотеки функций и

функциональных блоков в область редактирования FBD диаграммы данного программного модуля (рисунок 123):

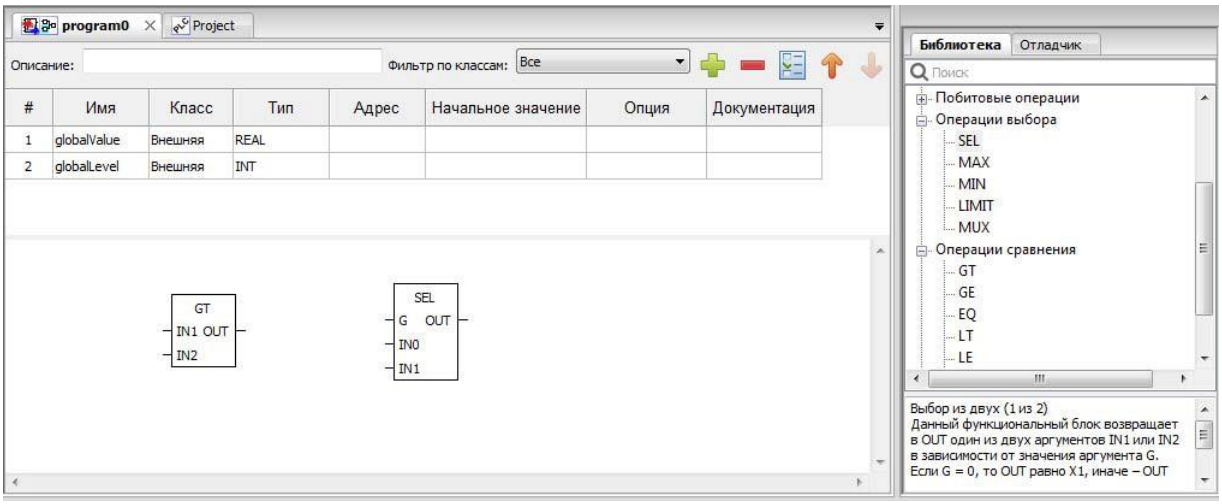


Рисунок 123 – Добавление двух функций на FBD диаграмму

Далее, так же используя мышь, переносятся переменные «globalValue» и «globalLevel» на FBD диаграмму. Как уже упоминалось ранее (см. п. 5.7.1.2), необходимо левой клавишей мыши зажать столбец «#» для переменной в панели переменных и констант, далее перенести указатель на область редактирования FBD диаграммы и отпустить кнопку мыши (Drag&Drop). Такую манипуляцию нужно произвести для обоих переменных (рисунок 124).

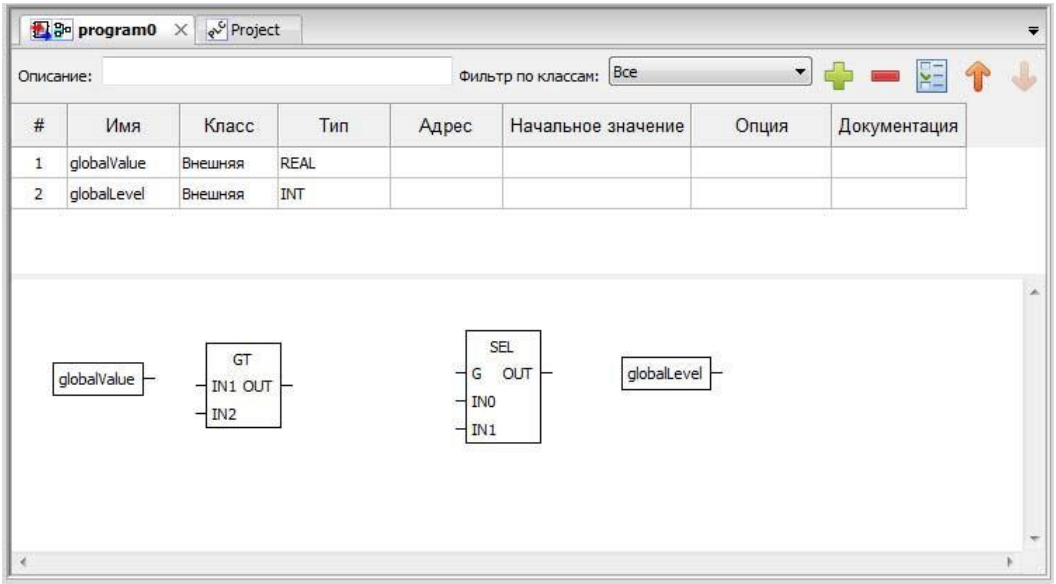


Рисунок 124 – Перенесённые переменные на FBD диаграмму

Переменную «globalValue» можно сразу соединить с входом «IN1» функции «GT». Чтобы переменную «globalValue» можно было соединить с выходом «OUT» функции «SEL», необходимо сделать двойной щелчок левой кнопкой мыши по «globalValue» на

FBD диаграмме и в появившемся диалоге «Свойства переменной» указать класс «Выходная», как показано на рисунке 125:

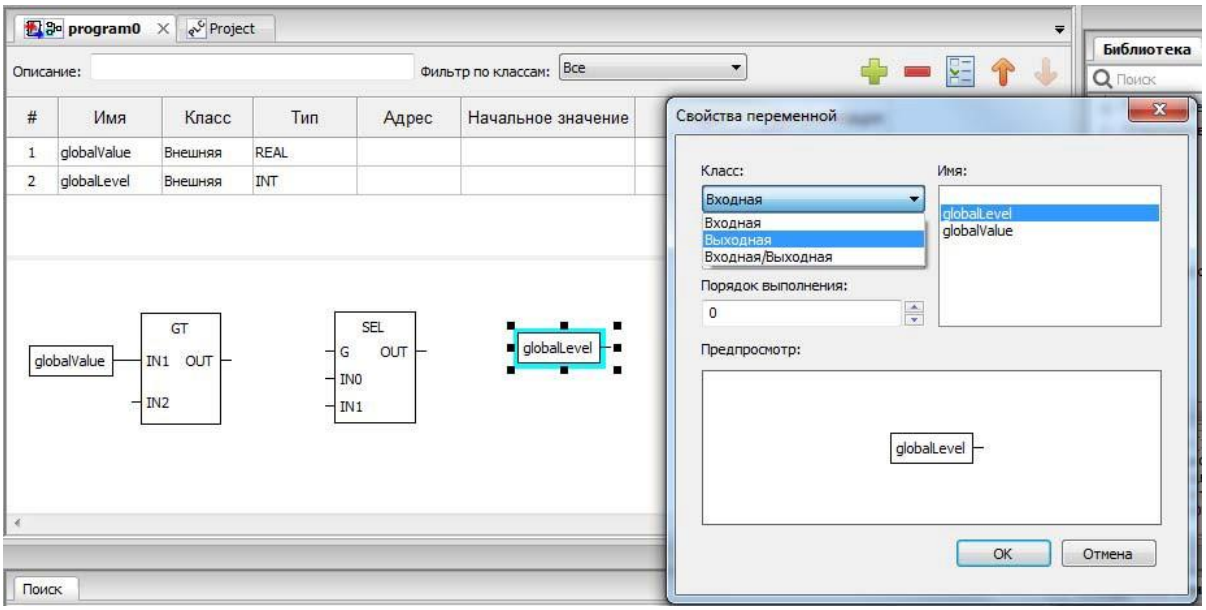


Рисунок 125 – Указание класса «Выходная» для переменной

После этого переменную «globalValue» можно соединить с выходом «OUT» функции «SEL» и выход «OUT» функции «GT» с входом «G» функции «SEL» (рисунок 126).

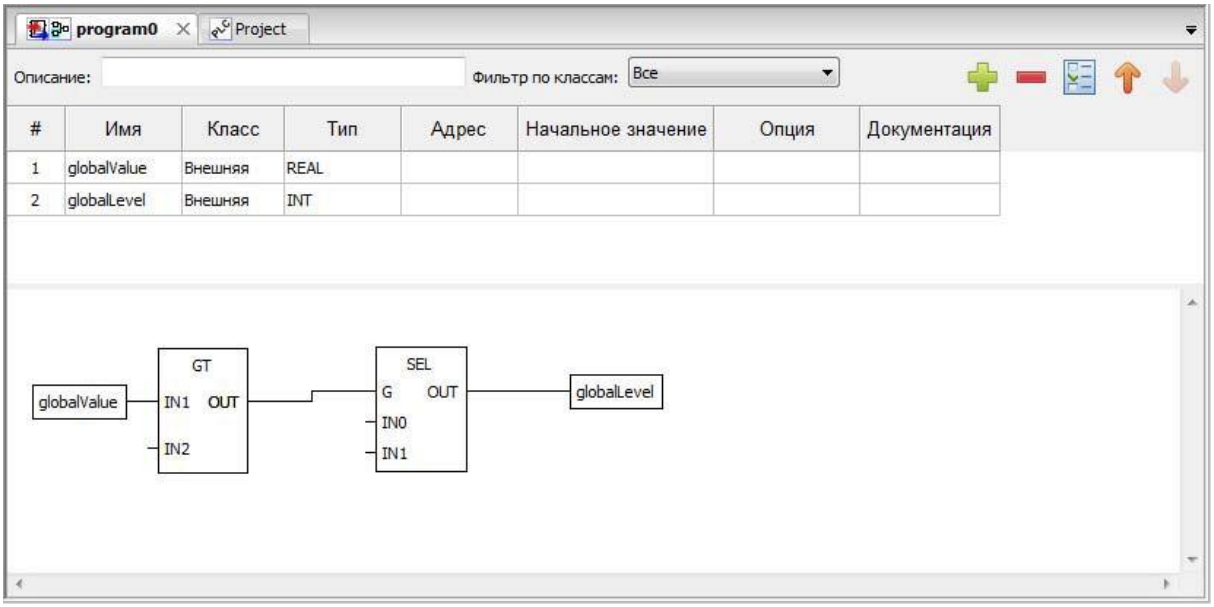


Рисунок 126 – Соединение входов и выходом функций на диаграмме FBD

Далее необходимо добавить 3 числовых литерала, которые будут напрямую соединены с входом «IN2» функции «GT» и входами «IN0» и «IN1» функции «SEL». В панели редактирования

FBD диаграммы выбирается кнопка «Добавить переменную» и в появившемся диалоге «Свойства переменной» (рисунок 127) в поле выражения пишется «10.0». Нажимается кнопка «OK».

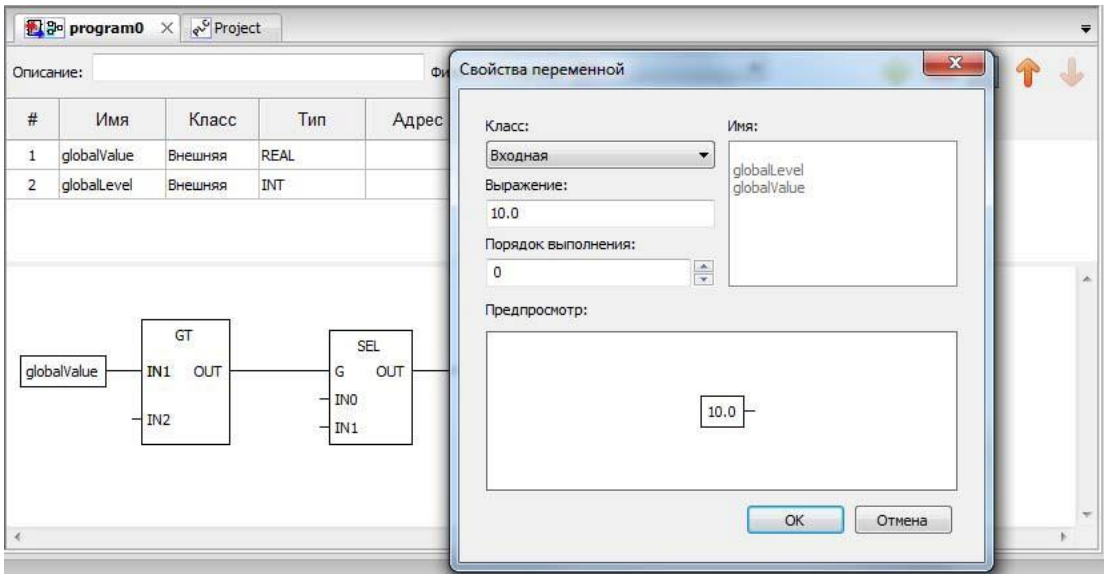


Рисунок 127 – Добавление переменной на FBD диаграмму

Добавленный литерал соединяется с входом «IN2» функции «GT», как показано на рисунке 128:

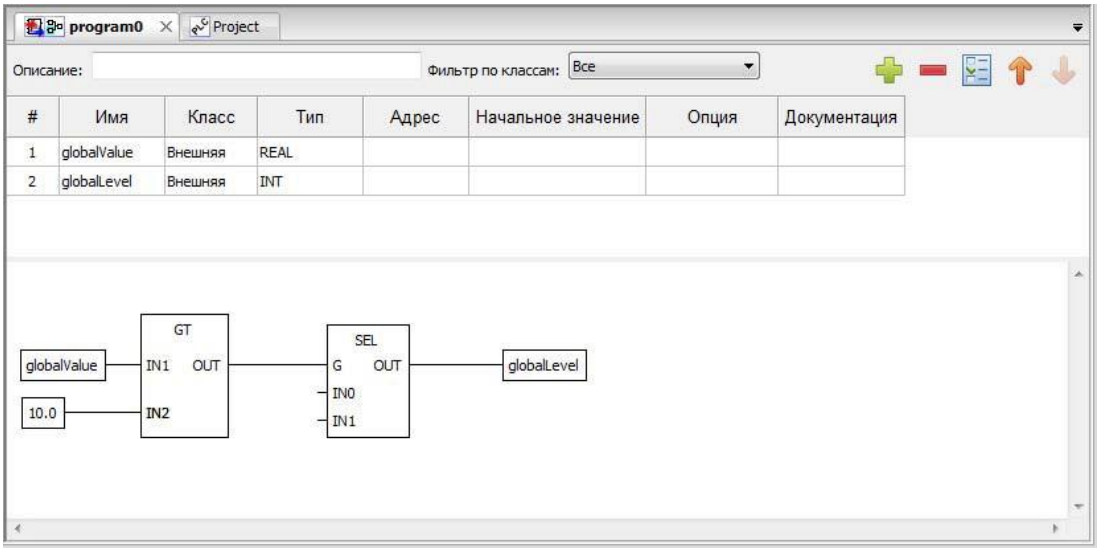


Рисунок 128 – Соединение добавленной переменной с выходом функции

Аналогичным образом создаются литералы со значениями 50 и 100 для входов «IN0» и «IN1» функции «SEL» и соединяются с ними (рисунок 129).

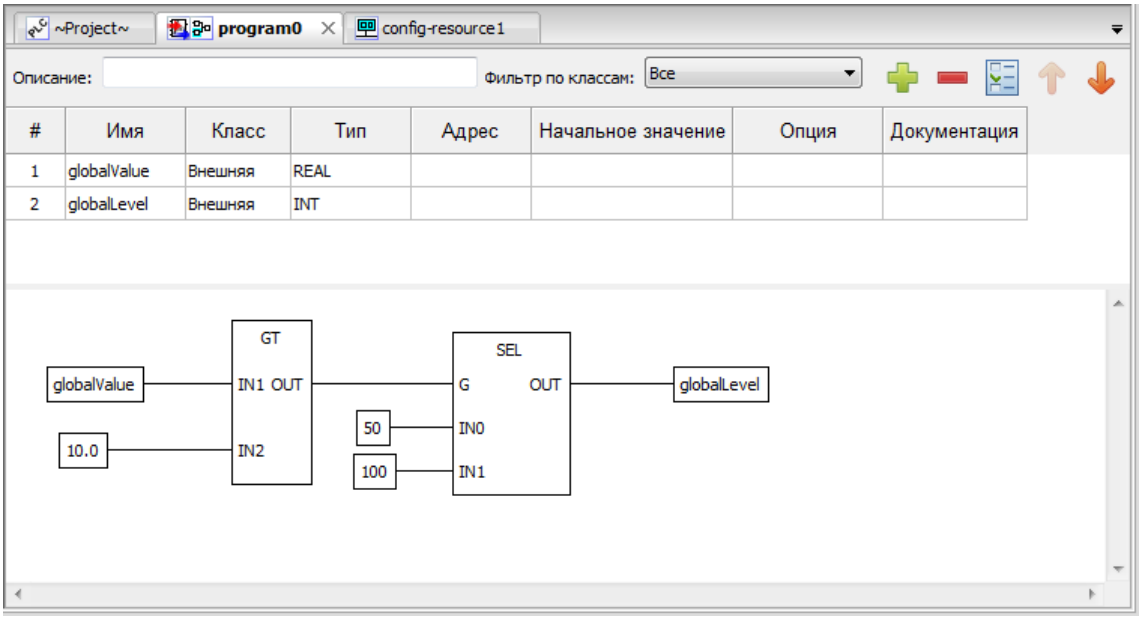


Рисунок 129 – Пример программного модуля, написанного на языке FBD

Соответственно, в случае если значение переменной «globalValue» больше 10.0, то на выходе «OUT» функции «GT» будет значение TRUE, тем самым на вход функции «SEL» поступит тоже True и выходное значение «OUT» будет равно «IN1», т.е. 100.

Далее будет рассмотрен пример добавления программного модуля типа «Функция».

7.3.2 Функция

Ниже будет приведен пример добавления в проект функции, и её использование в программном модуле типа «Программа». Добавление функции осуществляется с помощью меню дерева проекта, выбором пункта «Функция» (рисунок 130):

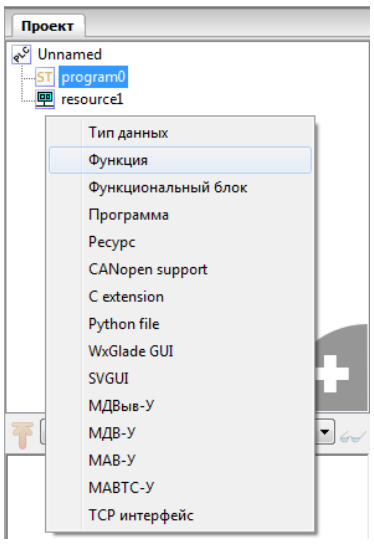


Рисунок 130 – Выбор в дереве проекта добавления функции

В появившемся диалоге «Создать новый программный модуль» в поле «Имя программного модуля» укажем имя «GetStatus» и выберем «Язык» ST из списка языков стандарта IEC 61131-3 (рисунок 131). Язык SFC не может быть использован для описания алгоритма и логики работы функции.

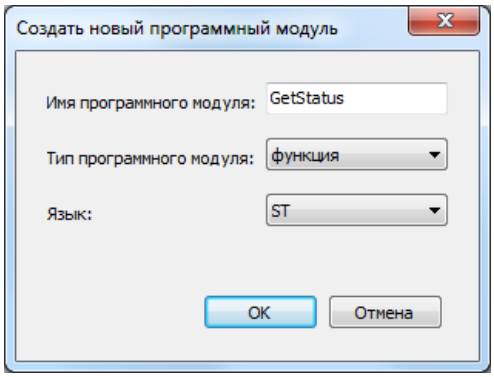


Рисунок 131 – Диалог добавления пользовательской функции

В появившейся панели редактирования функции выберем тип возвращаемого значения функции – STRING (рисунок 132).

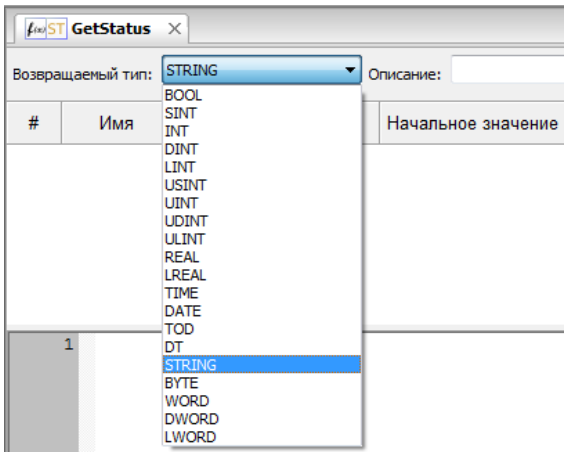


Рисунок 132 – Выбор значения, возвращаемого функцией

Далее в панели переменных и констант указываются переменная «value» (хотя переменных может быть несколько) класса «Входная» и в редакторе языка ST пишется алгоритм и логика работы данной функции, как показано на рисунке 133:

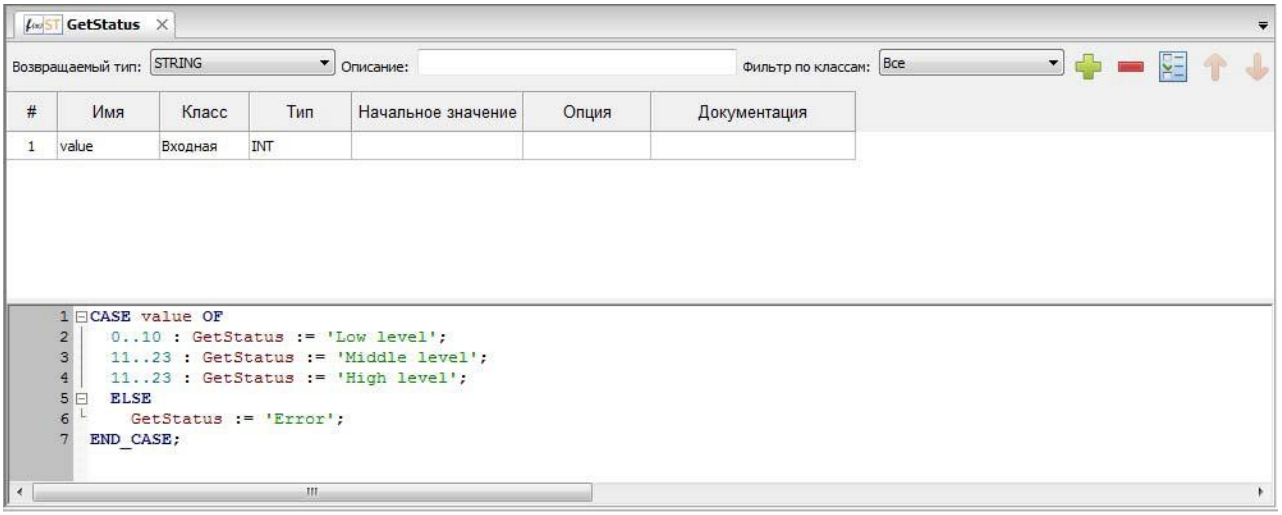


Рисунок 133 – Определение функции алгоритма и логики выполнения

С помощью блока CASE…OF определяются возвращаемые значения функции в зависимости от значения «value». Если ни одно из 3 условий не подходит, то возвращается «Error».

Далее необходимо создать программный модуль «Программа» на языке FBD и в появившейся панели его редактирования добавить две переменные «in_value» и «out_status», как показано на рисунке 134:

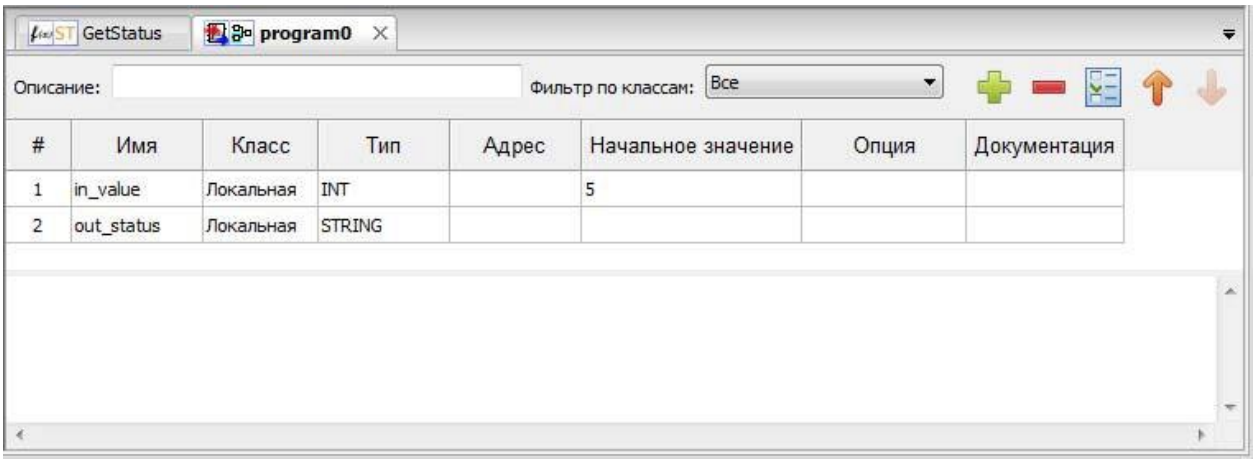


Рисунок 134 – Добавленный программный модуль «Программа» на языке FBD

На панели библиотеки функций и функциональных блоков в разделе «Пользовательские программные модули» необходимо выбрать функцию «GetStatus» и с помощью указателя мыши (зажав левую кнопку мыши) перенести данную функцию (Drag&Drop) в область редактирования FBD диаграммы программного модуля «program0» (рисунок 135).

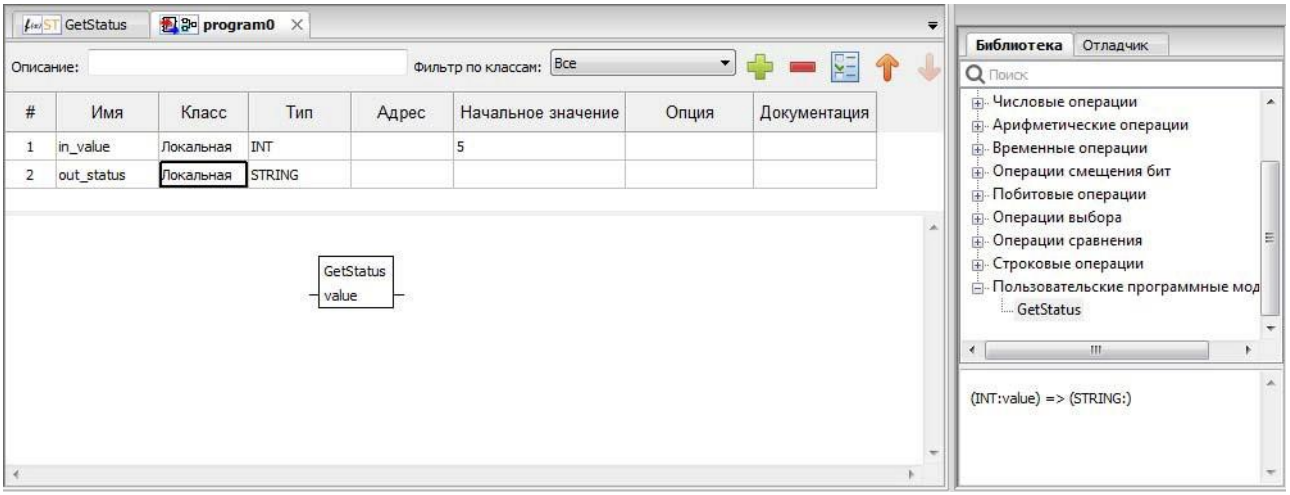


Рисунок 135 – Добавление на FBD диаграмму пользовательской функции

Аналогичным образом «перетаскиваются» переменные из панели переменных и констант в область редактирования FBD диаграммы (рисунок 136).

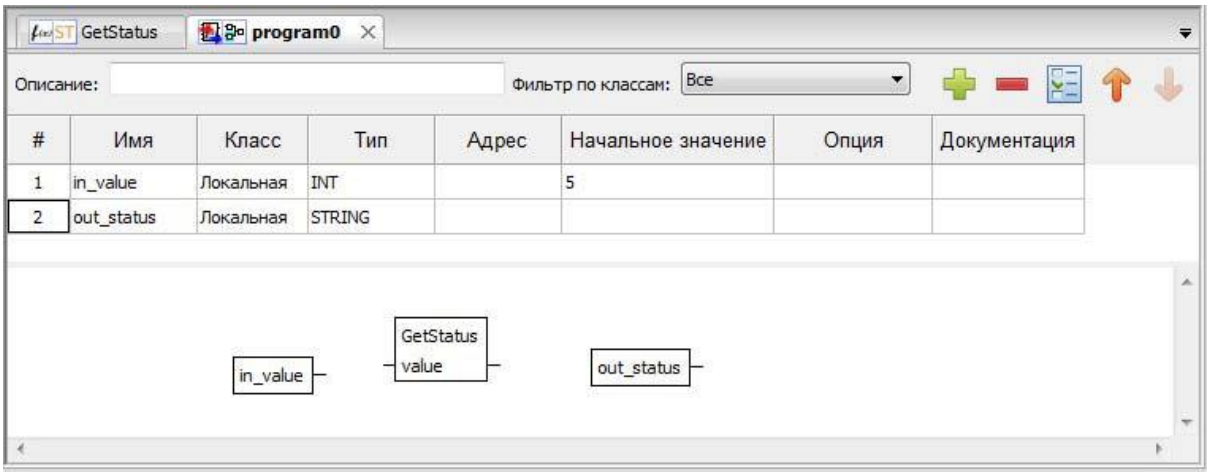


Рисунок 136 – Добавление на FBD диаграмму переменных из панели переменных и констант

Для того чтобы переменную «out_status» можно было соединить с «выходом» функции

«GetStatus», необходимо в диалоге свойств данной переменной (который вызывается двойным щелчком левой кнопки мыши по переменной в области редактирования FBD диаграммы) указать класс «Выходная» (рисунок 137).

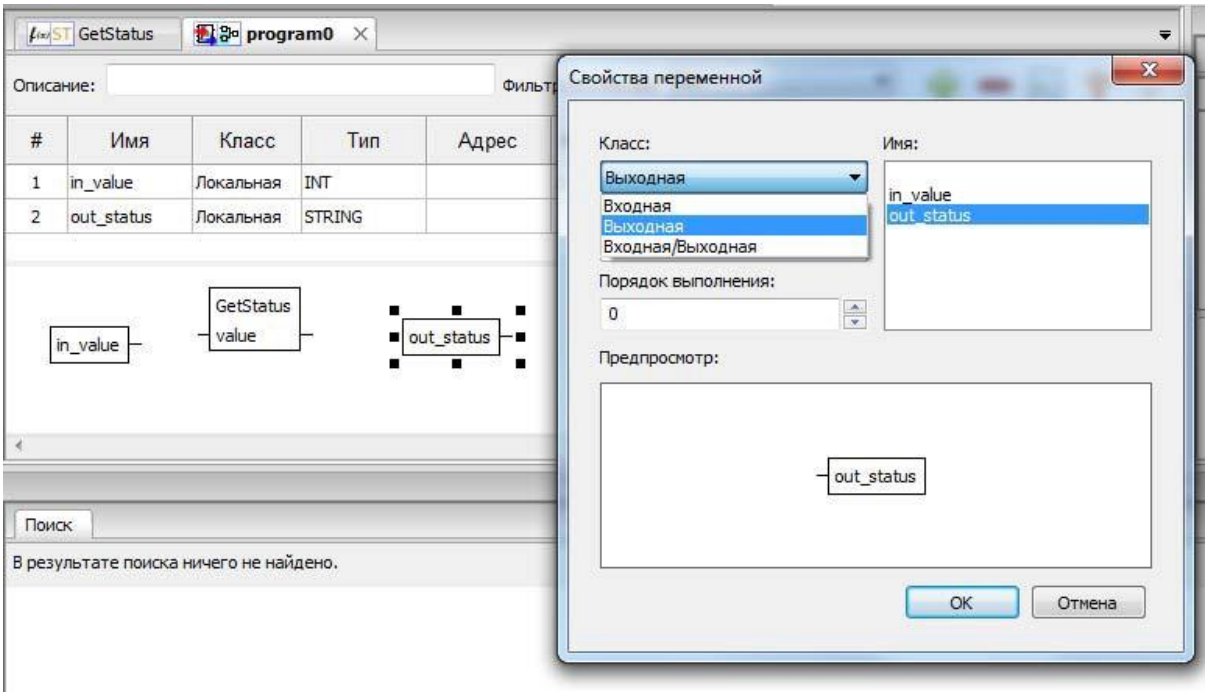


Рисунок 137 – Изменение «класса» переменной FBD диаграммы

Теперь переменные «in_value» и «out_status» можно соединить, соответственно, с «входом» и «выходом» функции «GetStatus» (рисунок 138).

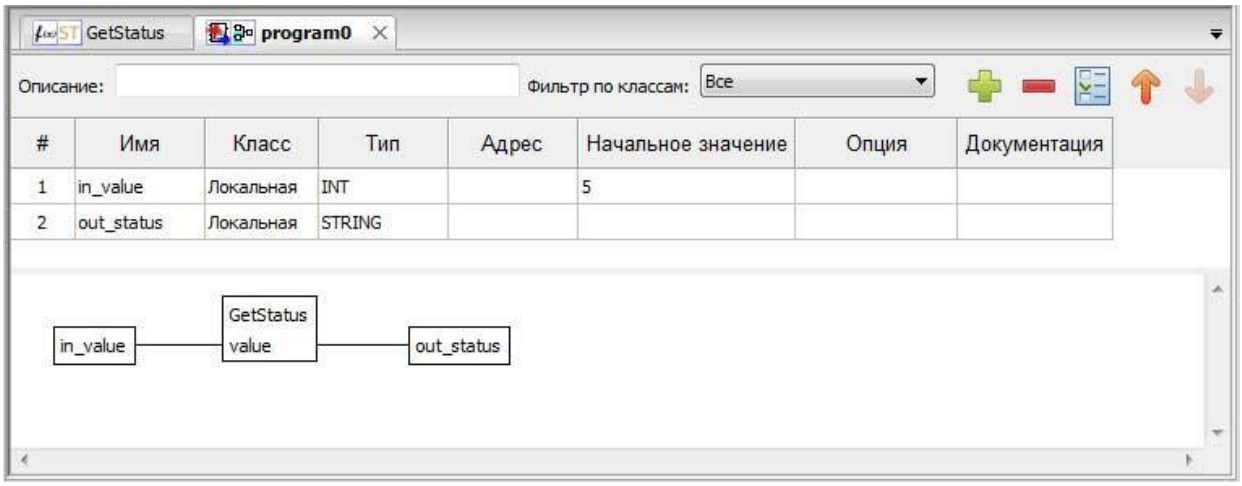


Рисунок 138 – Пример FBD диаграммы с использованием пользовательской функции

В результате созданная функция будет возвращать текстовое значение в переменную «out_status» в зависимости от значения переменной «in_value». Стоит отметить, что в данном примере значение переменной «in_value» постоянно равно 5 (для упрощения примера), но она также может зависеть от других переменных или быть связана, используя поле «Адрес», например с переменными внешних модулей УСО.

7.3.3 Функциональный блок

Добавление пользовательского функционального блока происходит аналогично добавлению функции. Ниже приведён пример создания функционального блока и его использования. После выбора в дереве проекта добавить «Функциональный блок», создаётся функциональный блок с именем «RectParams» (рис 139), который будет считать площадь и периметр прямоугольника с заданными сторонами.

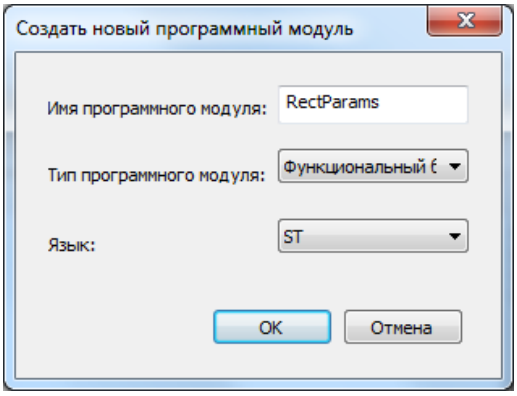


Рисунок 139 – Диалог добавления пользовательского функционального блока

В отличие от функции, функциональный блок может быть описан на любом языке стандарта IEC 61131-3, включая язык SFC. На рисунке 140 показана реализация данного функционального блока на языке ST.

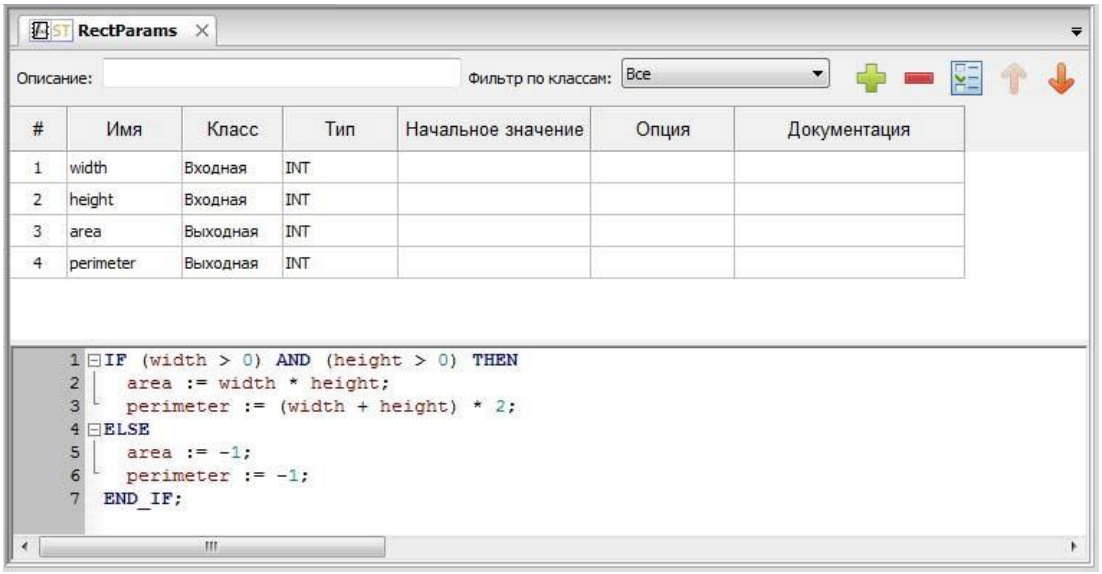


Рисунок 140 – Описание пользовательского функционального блока на языке ST

Возвращаемого значения у функционального блока нет. Добавленные переменные, обозначающие ширину – «width» и высоту – «height» имеют класс «Входная» и тип INT. Выходные значения: площадь – «area» и периметр – «perimeter» определены соответственно классом «Выходная» и так же типа INT. Ниже находится текст алгоритма расчёта площади и периметра на языке ST. Реализованный функциональный блок становится доступным в панели библиотеки функций и функциональных блоков (п. 5.11) и

может использоваться в программных модулях типа «Программа» и «Функциональный блок». На рисунке 141 показано использование созданного функционального блока «RectParams» в FBD диаграмме.

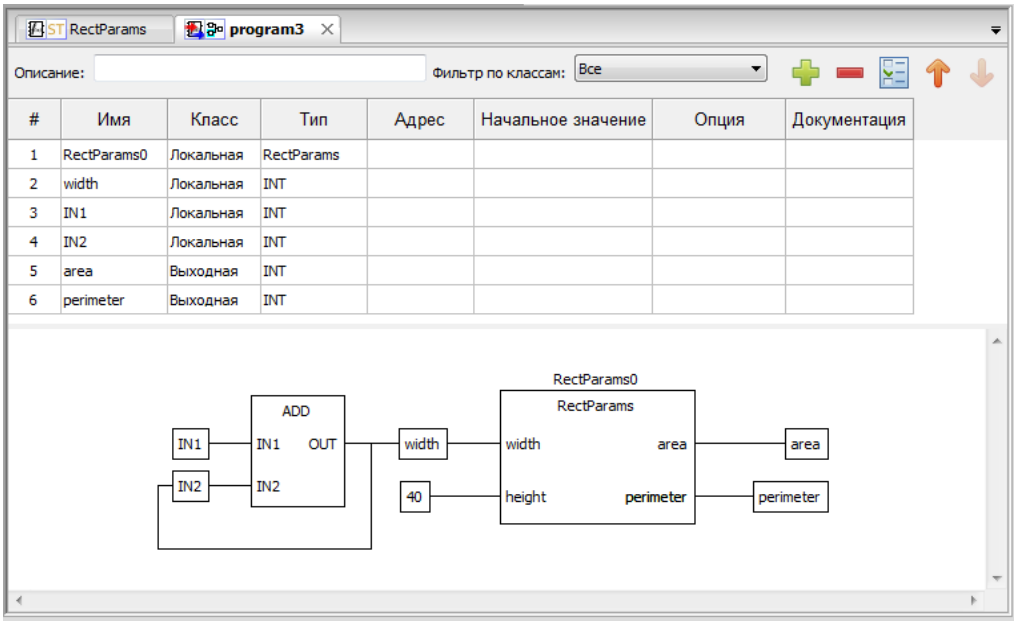


Рисунок 141 – Использование созданного функционального блока в FBD диаграмме

С входом «width» соединена переменная «width», а с входом «height» литерал «40».

Результат выполнения данного функционального блока также помещается, соответственно, в «area» и «perimeter». Следует отметить, что при попытке удаления функции или функционального блока из проекта (рисунок 142), где эти добавленные программные модули уже используются, будет выдана ошибка.

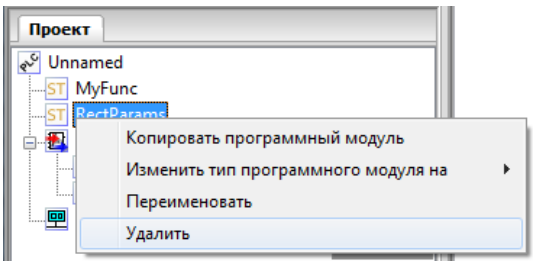


Рисунок 142 – Удаление функционального блока

Пример сообщения об ошибке удаления программного модуля приведен на рисунке 143.

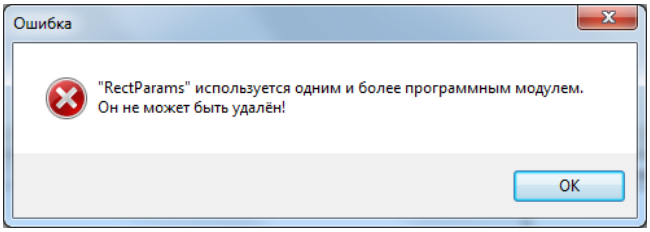


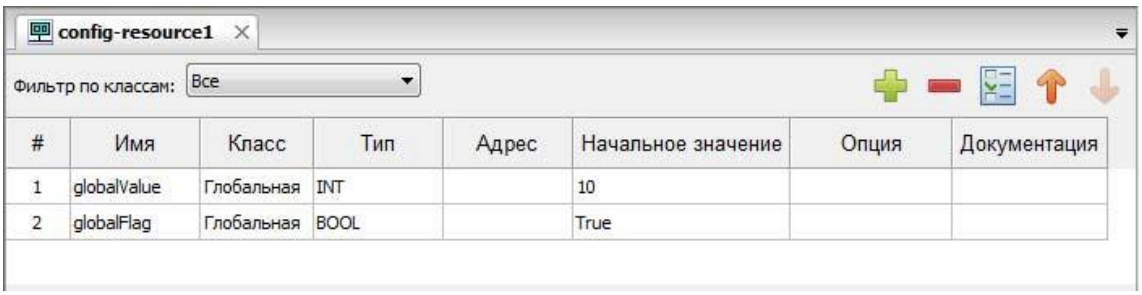
Рисунок 143 – Сообщение об ошибке при удалении функционального блока

7.4 Ресурс

Согласно стандарту IEC 61131-3, каждый проект должен иметь как минимум один ресурс, с определённым в нём как минимум одним экземпляром. Экземпляр представляет собой элемент, связанный с программным модулем типа «Программа» и одной определённой задачей. По умолчанию, среда разработки Beremiz создаёт для нового проекта один ресурс.

7.4.1 Глобальные переменные ресурса

Глобальные переменные ресурса объявляются аналогично глобальным переменным проекта (п. 7.2.1) на панели переменных и констант (рисунок 144) выбранного ресурса с использованием кнопки «Добавить переменную», либо «Добавить переменные».



#	Имя	Класс	Тип	Адрес	Начальное значение	Опция	Документация
1	globalValue	Глобальная	INT		10		
2	globalFlag	Глобальная	BOOL		True		

Рисунок 144 – Пример объявления глобальной на уровне проекта константы

Использование данных глобальных переменных на уровне ресурса также аналогично использованию глобальных переменных проекта в программных модулях. Ниже, на рисунке 145, показано, как в программном модуле «program0» добавлено две переменных класса «Внешняя» с такими же именами, как и глобальными переменные, объявленные выше для ресурса.

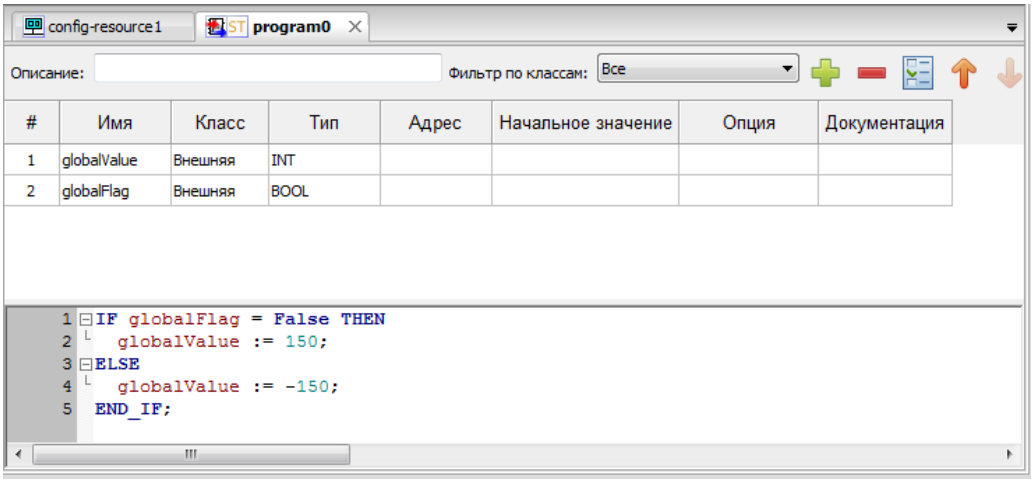


Рисунок 145 – Объявление и использование глобальных переменных ресурса

В редакторе ST кода написан алгоритм работы данного программного модуля с использованием данных глобальных переменных.

7.4.2 Задачи и экземпляры ресурса

Для создания экземпляра необходимо наличие как минимум одного программного модуля типа «Программа» в проекте и как минимум одной задачи, определённой в панели редактирования ресурса.

После добавления задачи с помощью кнопки «Добавить» (данная кнопка аналогична кнопки «Добавить» на панели переменных и констант), необходимо задать её уникальное имя (поле «Имя») и выбрать тип выполнения задачи (поле «Тип выполнения», рисунок 146):

- «Цикличное» – выполнение программного модуля типа «Программа» через заданный интервал времени, указанный в поле «Интервал»;
- «По условию» – выполнение программного модуля типа «Программа» один раз при наступлении значения TRUE глобальной переменной типа BOOL, определённой на уровне проекта, либо на уровне ресурса, указанной в поле «Флаг условия».

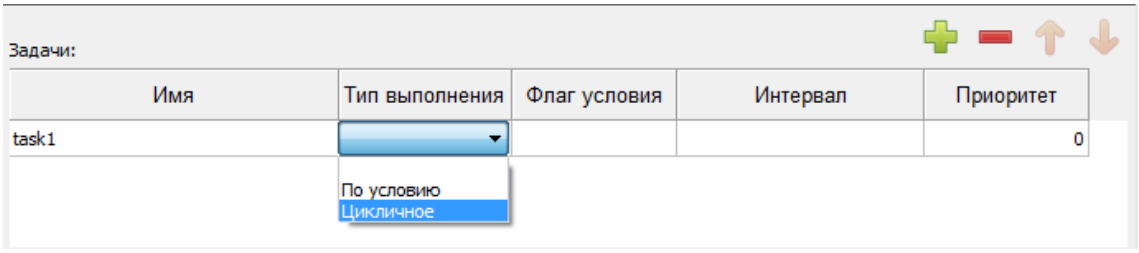


Рисунок 146 – Выбор типа выполнения задачи

В случае выбора типа выполнения «Цикличное», в поле «Интервал» необходимо указать интервал, с которым будет выполняться данная задача. Двойной щелчок левой кнопкой мыши по полю «Интервал» приводит к появлению кнопки «...» (рисунок 147).

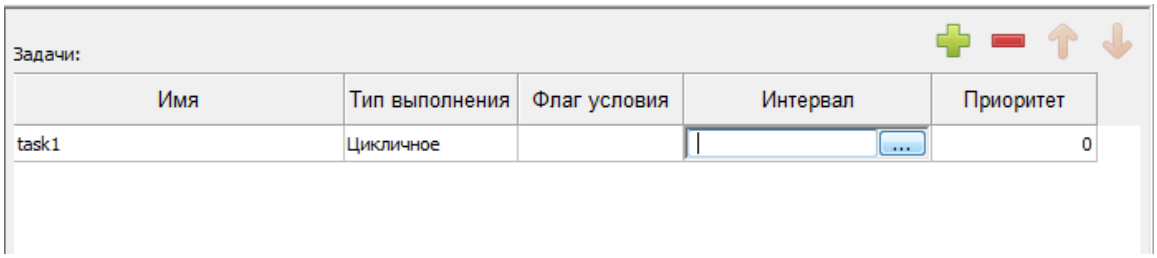


Рисунок 147 – Добавление задачи с цикличным режимом выполнения

Нажатие данной кнопки вызывает диалог «Редактировать продолжительность» (рисунок 148), в котором можно указать время, используя микросекунды, миллисекунды, секунды, минуты, часы и дни.

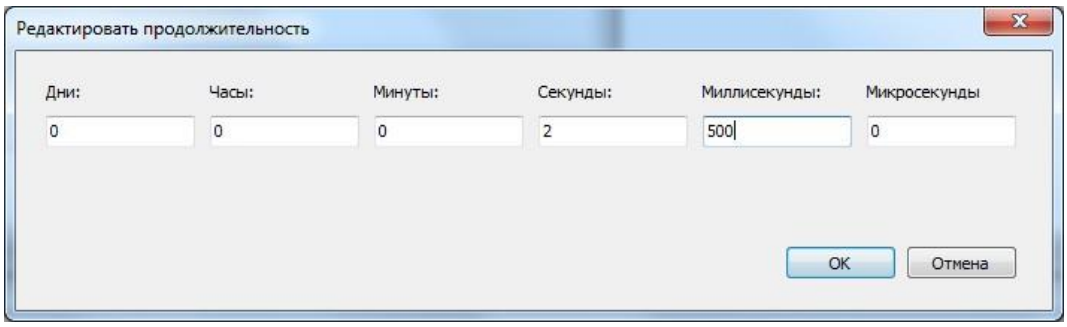


Рисунок 148 – Диалог редактирования продолжительности задачи

Завершение ввода времени кнопкой «OK» приводит к закрытию диалога и добавлению данного интервала времени в поле «Интервал» добавляемой задачи (рисунок 149).

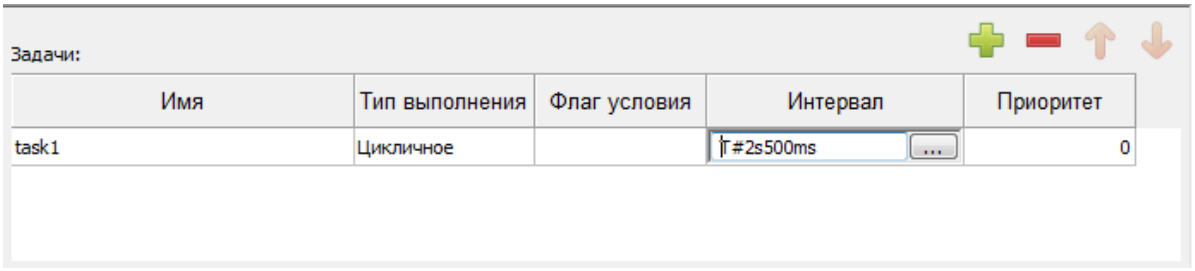


Рисунок 149 – Добавленный интервал выполнения

В случае выбора типа выполнения «По условию» в поле «Флаг условия» необходимо указать переменную типа BOOL, определённую глобально либо на уровне проекта (п. 7.2.1), либо на уровне ресурса (п. 7.5.1). На рисунке 150 выбирается переменная «globalFlag», определённая в данном ресурсе.

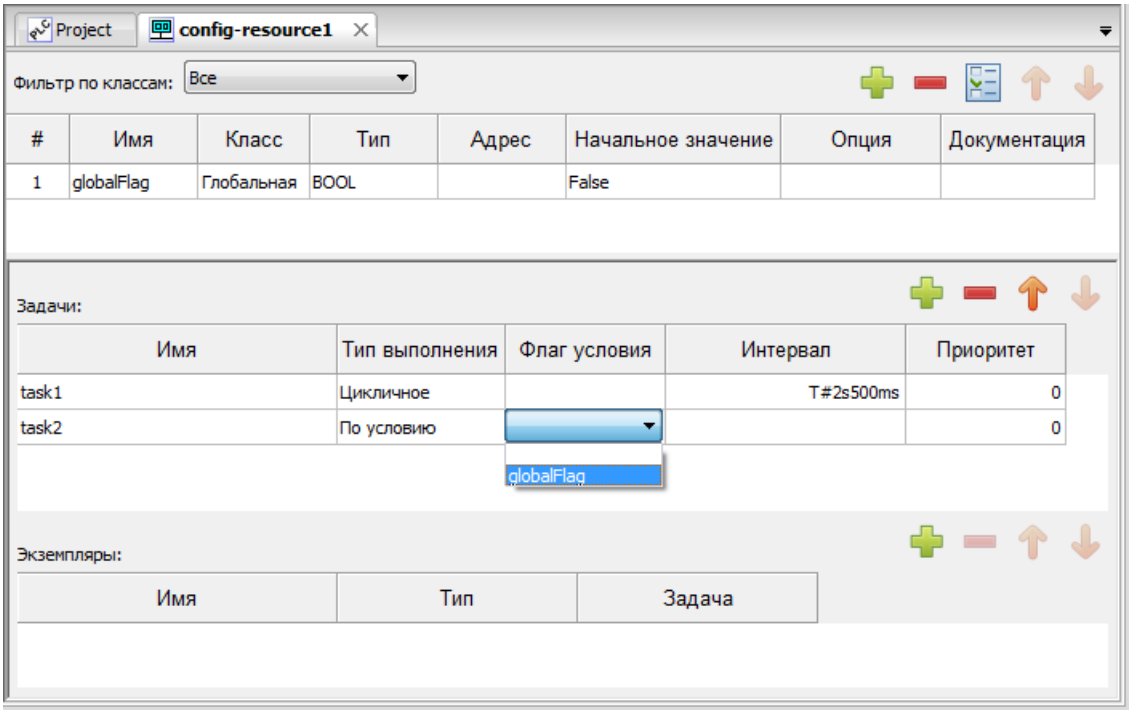


Рисунок 150 – Выбор переменной типа BOOL для определения условия выполнения задачи

Задача будет выполнена один раз, как только значение переменной, определённой в этом поле, будет TRUE.

Поле «Приоритет» позволяет указать приоритет выполнения задачи, по умолчанию все задачи имеют приоритет 0. Следует отметить, что в ресурсе должна быть определена как минимум одна задача с типом выполнения «Цикличное», в противном случае будет ошибка в компиляции в отладочной консоли (п. 5.12).

После того как задачи определены, их можно использовать в экземплярах. Создание экземпляра происходит аналогичным образом с помощью кнопки «Добавить». Необходимо выбрать уникальное имя экземпляра и далее указать программный модуль типа «Программа» в поле «Тип» и одну из задач в поле «Задача». Например, в проекте определено два программных модуля типа «Программа»: «program0» и «program1» (рисунок 151).

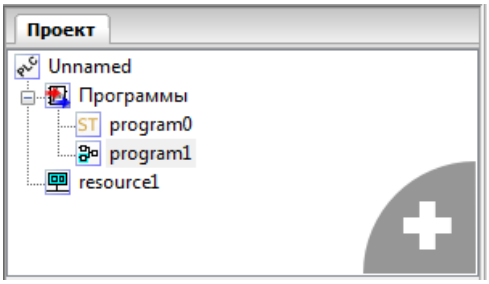


Рисунок 151 – Проект, содержащий два программными модулями типа «Программа»

Соответственно, при создании экземпляра в поле «Тип» оба этих программных модуля будут доступны (рисунок 152).

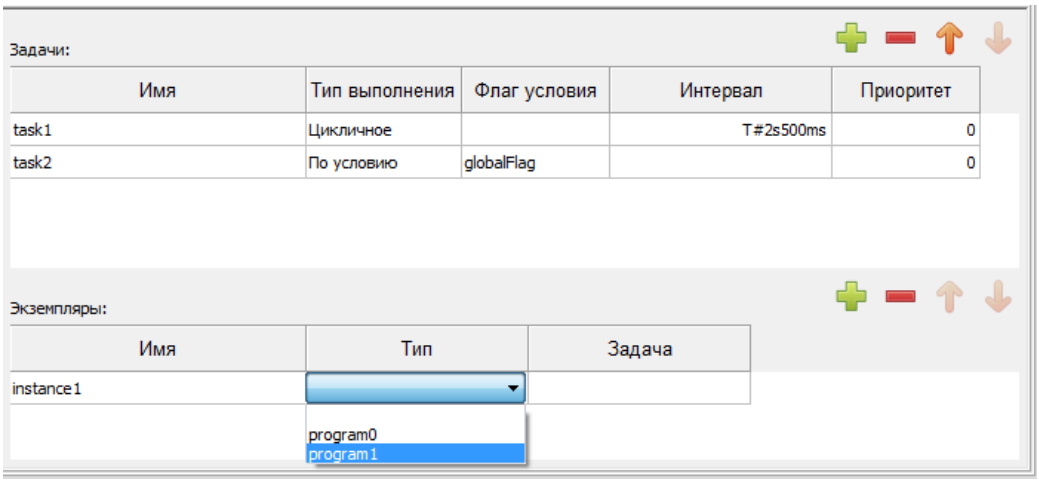


Рисунок 152 – Выбор программного модуля типа «Программа» для экземпляра

Аналогичным образом выбирается задача из списка, в котором будут отображены определённые ранее задачи (рисунок 153).

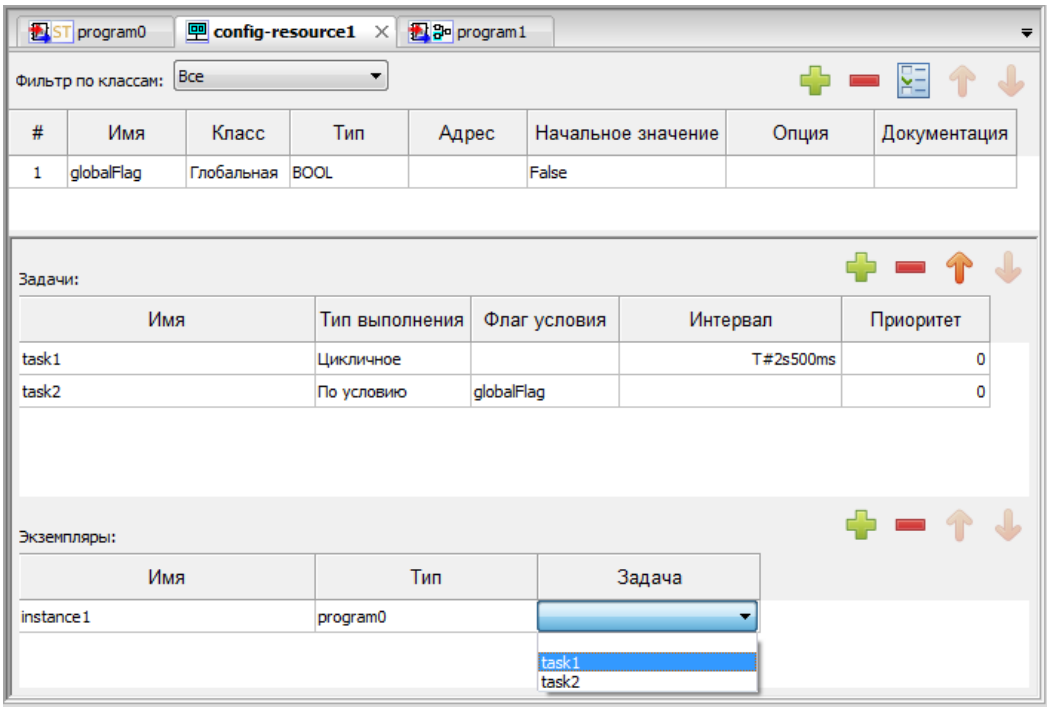


Рисунок 153 – Выбор задачи для экземпляра

В каждом проекте в ресурсе должен быть определен как минимум один экземпляр, в противном случае будет ошибка выдана компиляции в отладочной консоли (п. 5.12).

7.5 Типы данных

Добавление типа данных происходит выбором пункта «Типа данных» в меню дерева проекта (рисунок 154) в уже созданный проект, содержащий программный модуль типа «Программа» – «program0».

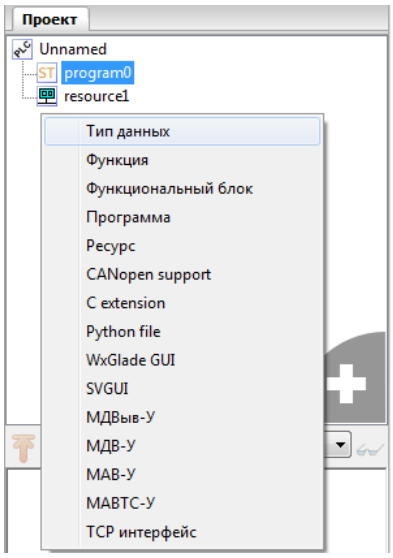


Рисунок 154 – Выбор пункта меню добавления пользовательского типа данных в дереве проекта

Будет создан массив типа INT размерностью 11 элементов. В появившемся диалоге указывается имя создаваемого типа данных, например «MyArrayType» (рисунок 155) и нажимается кнопка «OK».

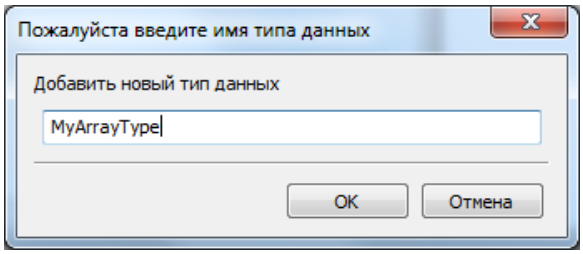


Рисунок 155 – Диалог ввода имени создаваемого пользовательского типа данных

В дереве проекта появляется панель редактирования добавленного типа данных с именем «MyArrayType (п. 5.9). В поле «Механизм создания нового типа» необходимо выбрать «Массив» и указать тип INT, как показано на рисунке 156:

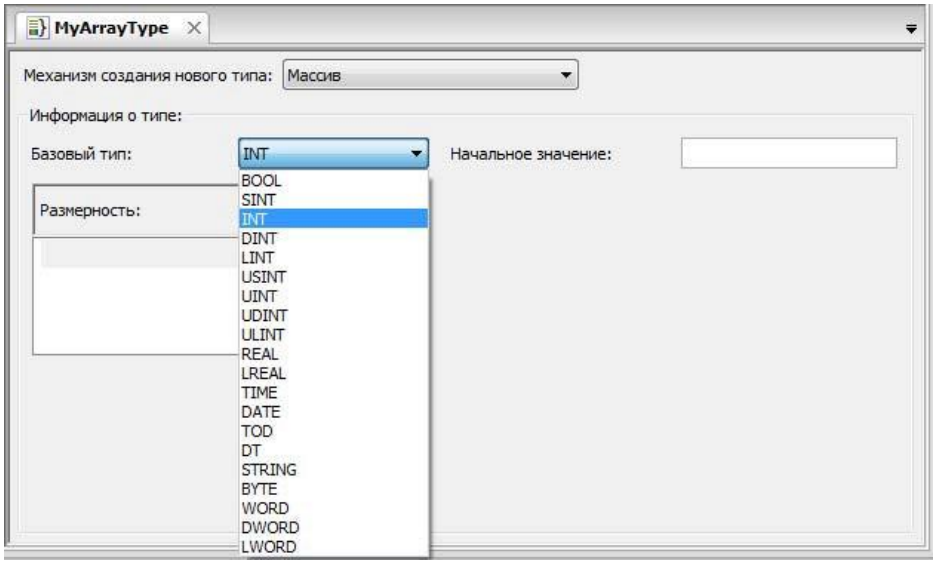


Рисунок 156 – Выбор базового типа для массива

С помощью кнопки «Добавить» создаётся поле для массива с указанием его размерности (рисунок 157) в соответствующем формате.

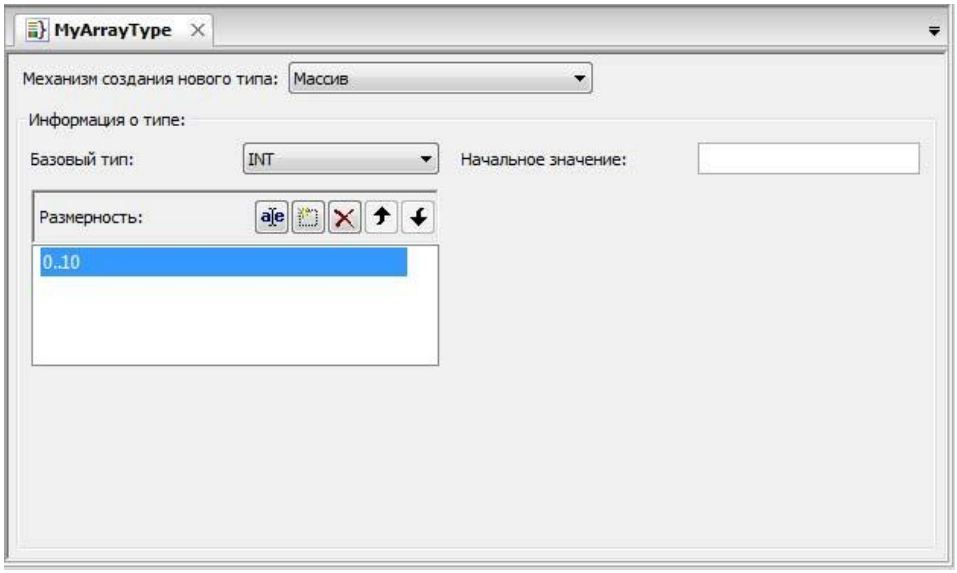


Рисунок 157 – Задание размерности для массива

После выполнения вышеперечисленных операций тип «MyArrayType» может быть использован для определения переменных в программных модулях, так же как и базовые типы данных (рисунок 158).

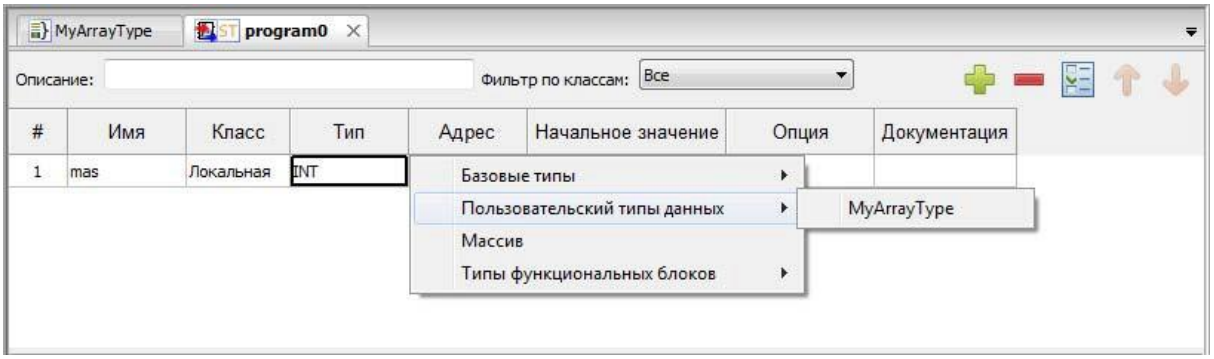


Рисунок 158 – Выбор добавленного типа данных в панели переменных и констант программного модуля

Добавим две переменные «temp_value» и «out_value» типа INT в панель переменных и констант. Ниже, на рисунке 159, пример кода на языке ST, в котором используется переменная «mas», тип которой массив, добавленный выше.

Сначала локальной переменной «temp_value» присваивается значение 10:

```
temp_value := 10;
```

Далее первому элементу массива «mas» (нумерация элементов в массиве начинается с 0) присваивается значение переменной «temp_value»:

```
mas[0] := temp_value;
```

Второму элементу массива «mas» присваивается значение 20:

```
mas[1] := 20;
```

И в конце второй локальной переменной присваивается первый элемент массива «mas»:

```
out_value := mas[0];
```

Далее будет рассмотрена процедура сборки и отладки проекта.

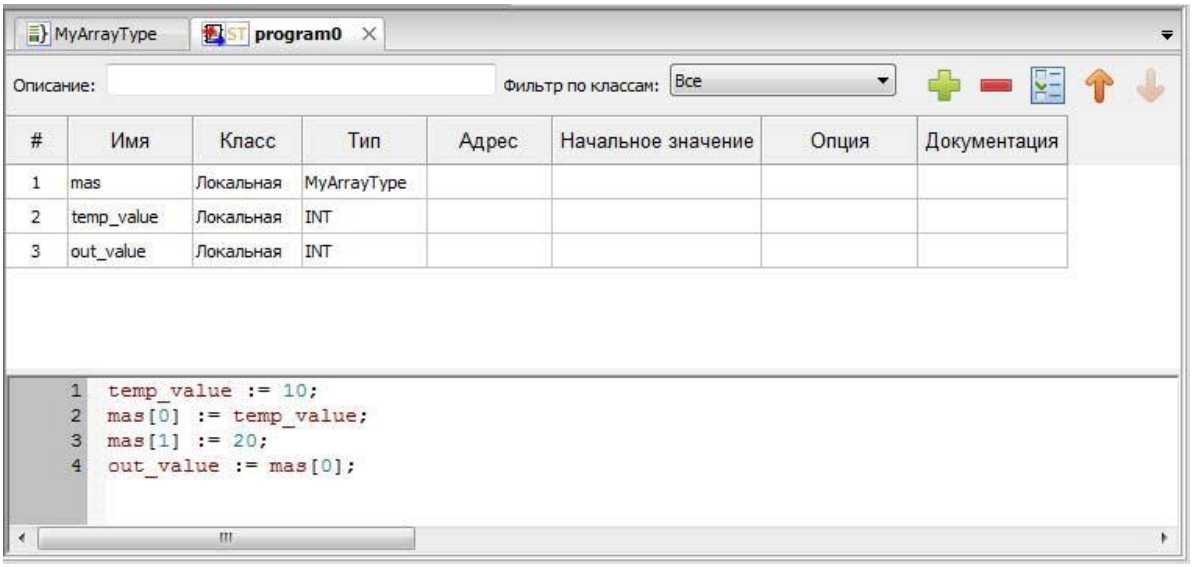


Рисунок 159 – Пример использования массива в коде на языке ST

8 СБОРКА, ЗАГРУЗКА И ОТЛАДКА ПРИКЛАДНОЙ ПРОГРАММЫ

Следующими шагами после создания основных элементов проекта является его сборка (компиляция и компоновка), передача полученного исполняемого файла на целевое устройство и отладка данной прикладной программы.

Сборка проекта осуществляется с помощью соответствующих кнопок, находящихся на панели инструментов (п. 5.2.2). Для успешного завершения данной операции каждый проект должен иметь как минимум один ресурс (как уже упоминалось, при создании проекта по умолчанию ресурс будет создан). 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).

Возможность передачи на целевое устройство прикладной программы и её отладки определяется наличием на целевом устройстве специальной программной части (поддержка отладочного протокола ИСР Beremiz).

Среда разработки Beremiz предоставляет следующие возможности отладки:

- просмотр и изменение значения всех переменных проекта, используя панель отладки, (п. 5.14);
- визуально отслеживание выполнения программ на графических языках и изменение значения различных графических элементов конкретного языка;
- отображение значений переменных в виде графика (п. 5.15).

Далее подробнее рассказывается про соединение с целевым устройством, передачи на него исполняемого файла и его отладке.

8.1 Сборка проекта

Перед сборкой проекта, необходимо очистить директорию сборки от файлов предыдущей сборки, нажав на кнопку «Очистка» в панели инструментов (рисунок 160 – помечено цветом):

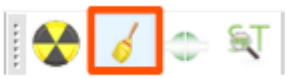


Рисунок 160 – Очистка директории сборки

Далее запускается сборка проекта, для чего необходимо нажать на кнопку «Сборка» в панели инструментов (рисунок 161 – помечено цветом):

Рисунок 161 – Сборка проекта

Результат сборки (успешно или неуспешно) выводится Консоль сообщений (рисунок 162):

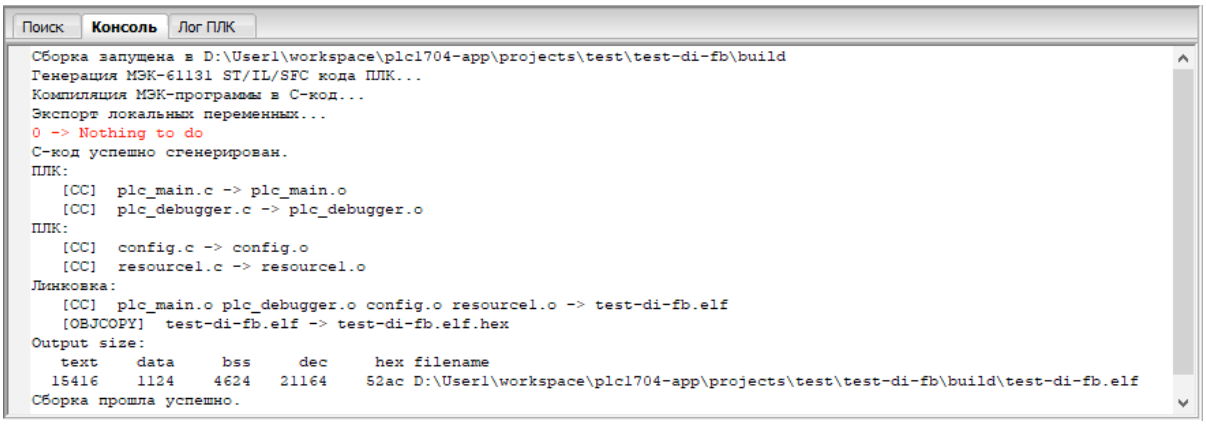


Рисунок 162 – Результат сборки приложения в Консоли сообщений

8.2 Соединение с целевым устройством и передача исполняемого файла

После того как проект собран, можно производить соединение с целевым устройством и загрузку исполняемого файла на целевое устройство. В панели настроек проекта (п. 5.5) необходимо указать URI-адрес целевого устройства:

YAPLC://<номер COM-порта>

где, YAPLC – это драйвер, входящий в состав ИСР Beremiz и предоставляющий работу из ИСР с целевым устройством по последовательному порту: загрузка проекта в целевое устройство, отладка проекта в режиме реального времени; <номер COM-порта> - символьный номер COM-порта.

На рисунке 163 показан URI адрес целевого устройства YAPLC://COM7 (целевое устройство подключено к последовательному порту COM7 компьютера, где производится разработка проекта), который задается в панели настроек проекта:

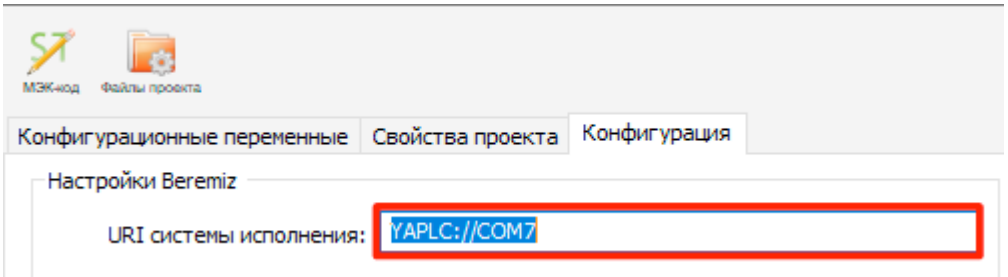


Рисунок 163 – URI адреса целевого устройства, подключенного к COM7

На рисунке 164 показан тип целевого устройства «plm2004», который выбирается из списка в панели настроек проекта:

Рисунок 164 – Выбор целевого устройства «plm2004»

После задания URI-адреса и типа целевого устройства можно соединяться с устройством, для этого необходимо нажать на кнопку «Подключиться к целевому устройству» в панели инструментов (рисунок 165 – выделено цветом):

Рисунок 165 – Подключение к целевому устройству

Информация о результате подключения выводится в Консоль сообщений (рисунок 166) и в строке статуса (нижняя часть окна ИСР Beremiz):

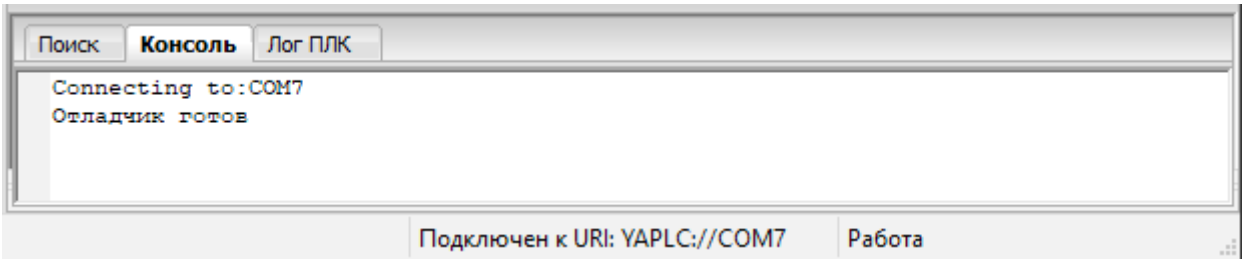


Рисунок 166 – Результат подключения в Консоли сообщений и строке статуса

После успешного подключения на панели инструментов будут доступны кнопки: «Остановить/Запустить исполнение прикладной программы на целевом устройстве», «Передать прикладную программу на целевое устройство» (рисунок 167):

Рисунок 167 – Кнопки «Остановить программу» и «Передать программу» в панели инструментов

Для передачи собранной прикладной программы на целевое устройство необходимо нажать на кнопку «Передать прикладную программу», при этом запустится утилита загрузчика (рисунок 168):

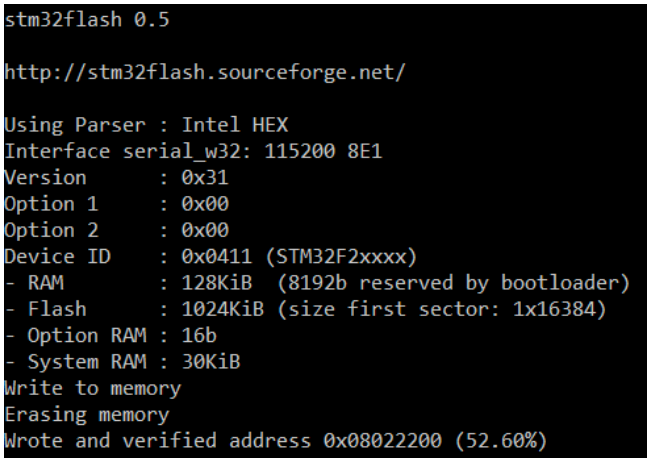


Рисунок 168 – Работа утилиты загрузчика прикладной программы

По окончании загрузки, утилита автоматически завершит свою работу (окно закроется) и в Консоль сообщений выведется информация о результате (рисунок 169). После успешной загрузки, прикладная программа автоматически будет запущена на целевом устройстве.

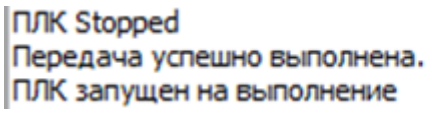


Рисунок 169 – Результат передачи прикладной программы на целевое устройство

Пример действий для загрузки прикладной программы в целевое устройство типа «plm2004»:

1) подключить преобразователь RS-485 к разъему «DBG» на плате устройства,
2) установить перемычку «ISP» на плате устройства,
3) в ИСР Beremiz задать настройки подключения и тип целевого устройства,
4) в ИСР Beremiz собрать проект,
5) в ИСР Beremiz подключиться к целевому устройству,
6) выключить устройство,
7) включить устройство,
8) в ИСР Beremiz запустить передачу собранной прикладной программы на целевое устройство,
9) дождаться завершения загрузки прикладной программы на целевое устройство,
10) в ИСР Beremiz отключиться от целевого устройства,
11) выключить устройство,
12) снять перемычку «ISP»,
13) отключить преобразователь RS-485 от разъема «DBG»,
14) включить устройство.

8.3 Отладка текстовых языков

После установки соединения с целевым устройством и запуском прикладной программы на выполнение, среда разработки Beremiz позволяет отслеживать и изменять значения переменных программных модулей, из которых состоит проект. Необходимо обратиться к панели экземпляров (п. 5.10) и используя соответствующие кнопки, выделенные красным цветом на рисунке 170, добавить переменных панель отладки (п. 5.14):

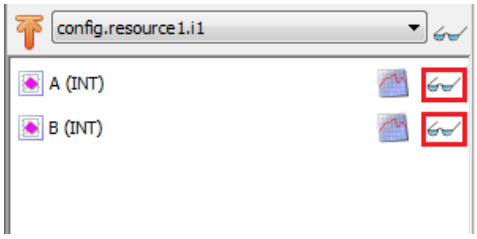


Рисунок 170 – Добавление переменных для отладки из панели экземпляров

На панели отладки сразу же отобразятся текущие значения добавленных переменных (рисунок 171).

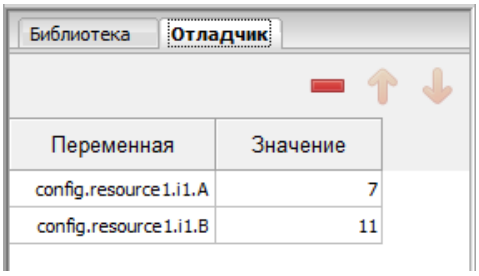


Рисунок 171 – Панель отладки значений переменных

Для того чтобы изменить значение переменной во время исполнения проекта, необходимо нажать правой клавишей мыши в поле «Значение» интересующей переменной и в появившемся всплывающем меню (рисунок 172) выбрать «Установить значение».

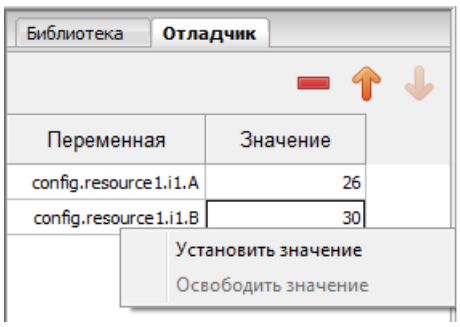


Рисунок 172 – Всплывающее меню манипуляций со значением переменной

Появится диалог, показанный на рисунке 173.

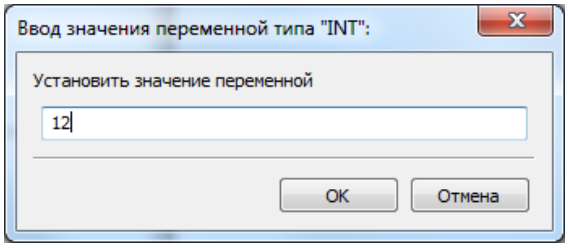


Рисунок 173 – Установка значения переменной во время отладки

Для корректного изменения, вводимое значение должно соответствовать типу переменной, иначе будет выведено сообщение об ошибке (рисунок 174).

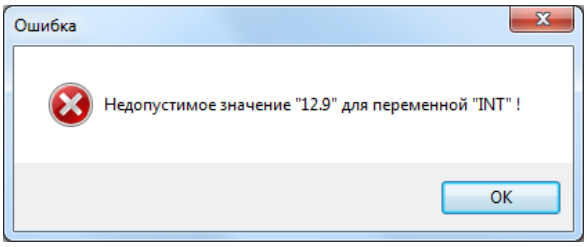


Рисунок 174 – Ошибка при вводе недопустимого значения изменяемой переменной в режиме отладки

После изменения значения переменной, она будет отображаться синим цветом (рисунок 175) в таблице переменных и их значений во вкладке «Отладка»:

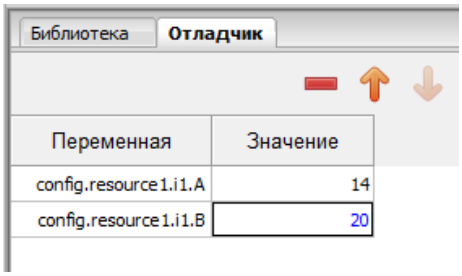


Рисунок 175 – Переменные в режиме отладки

Для возврата переменной в «нормальный» режим (т.е. вывести из режима отладки), необходимо снова обратиться к всплывающему меню и выбрать «Освободить значение» (рисунок 176):

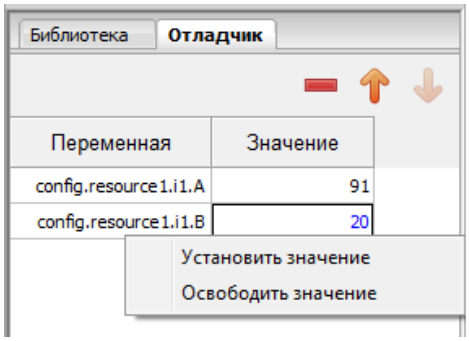


Рисунок 176 – Всплывающее меню манипуляций со значением переменной

Далее следует описание отладки прикладных программ, написанных на графических языках.

8.4 Отладка FBD диаграмм

Во время отладки прикладной программы, в которой часть программных модулей написано на графических языках, есть возможность видеть изменения всех значений на диаграмме и вносить необходимые изменения прямо на ней. Как уже упоминалось ранее в п. 5.10, в случае нажатия кнопки запуска режима отладки (на рисунке 177 выделены красным цветом) для экземпляра программы, написанной на одном из графических языков, откроется вкладка с панелью диаграммы в режиме отладки.

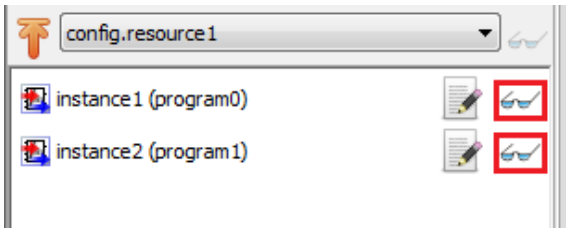


Рисунок 177 – Панель экземпляров проекта

В режиме отладки FBD диаграммы есть возможность устанавливать входные и выходные значения переменных (с помощью всплывающего меню, которые вызывается нажатием правой клавишей по соединению) для функциональных блоков, а также в целом видеть все остальные значения на входах и выходах элементов диаграммы (рисунок 178).

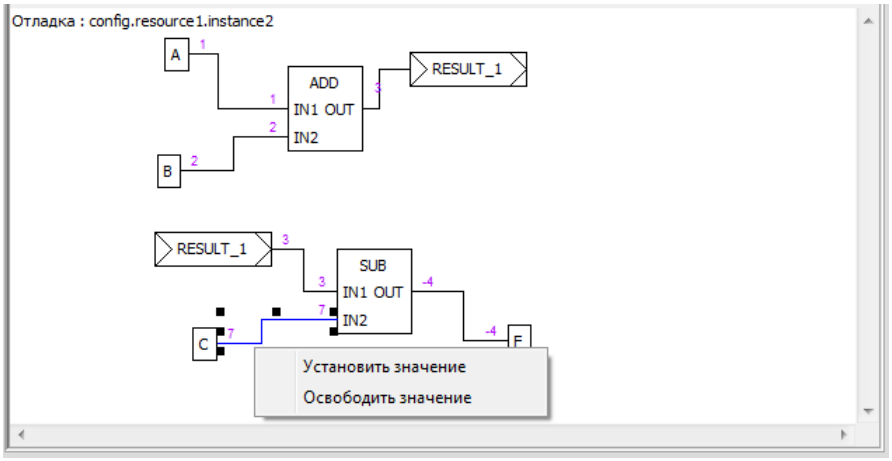


Рисунок 178 – Пример отладки FBD диаграммы

Изменённые значения в режиме отладки выделяются синим цветом. После выбора во всплывающем меню «Освободить значение» значение возвращается в то, которое получается в результате выполнения логики и алгоритма данного модуля на данном участке, а соединение на диаграмме становятся исходного цвета.

8.5 Отладка LD диаграммы

Отладка LD диаграммы осуществляется аналогично отладке FBD диаграммы. Для вызова всплывающего меню (рисунок 179), в котором можно установить желаемое значение для контакта или катушки необходимо нажать правую клавишу мыши.

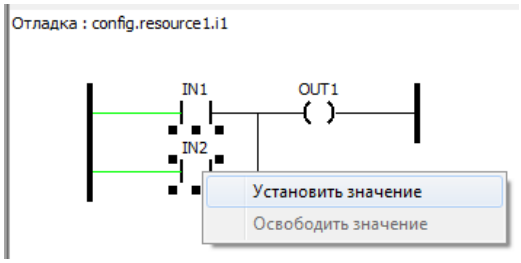


Рисунок 179 – Пример отладки LD диаграммы

Появится диалог (рисунок 180), в котором нужно ввести значение типа BOOL: TRUE – контакт «ON», FALSE – контакт «OFF».

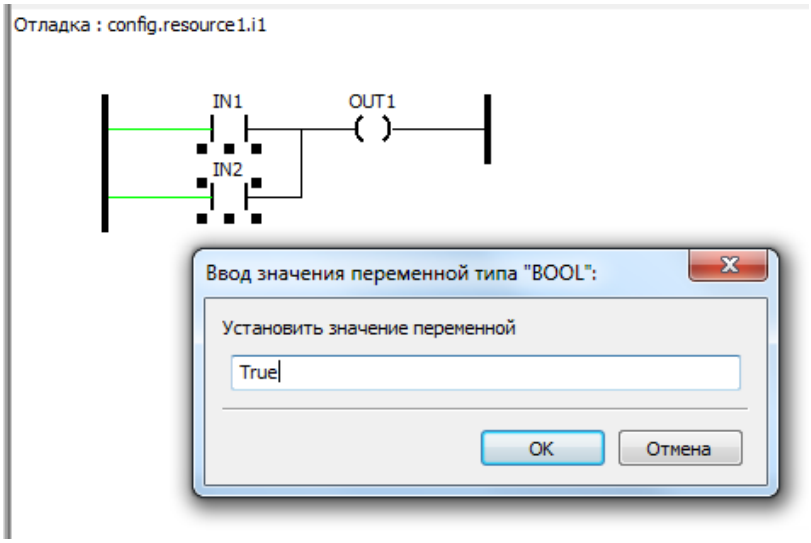


Рисунок 180 – Отладка LD диаграммы

После установки в режиме отладки, для данного примера, контакта «IN2» (рисунок 181), в значение TRUE катушка «OUT1» приняла значение TRUE, т.к. данная схема представляет собой реализацию «Логического ИЛИ» для «IN1» и «IN2».

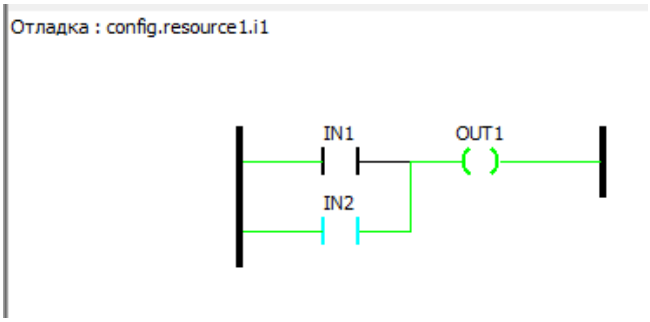


Рисунок 181 – Отладка LD диаграммы

8.6 Отладка SFC диаграммы

Отладка SFC диаграммы происходит аналогично отладке диаграмм FBD и LD. С помощью всплывающего меню (рисунок 182), есть возможность устанавливать активность для шагов и переходов.

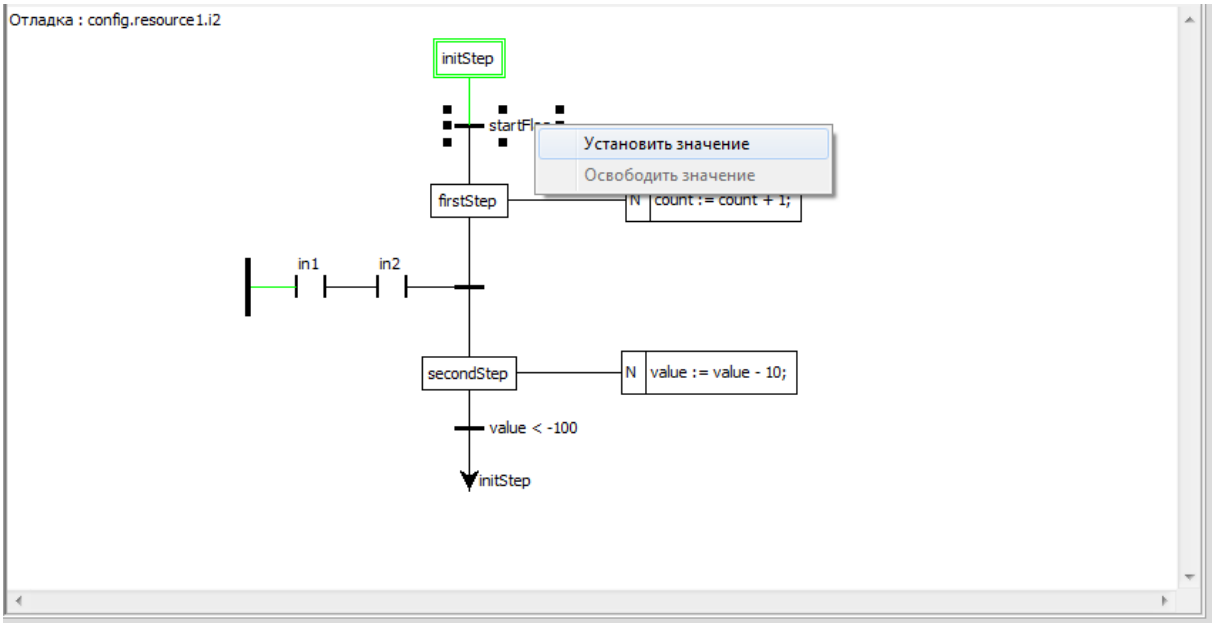


Рисунок 182 – Пример отладки SFC диаграммы

На рисунке 183 показано, как устанавливается значение (после выбора «Установить значение», появится диалог) TRUE для шага «firstStep».

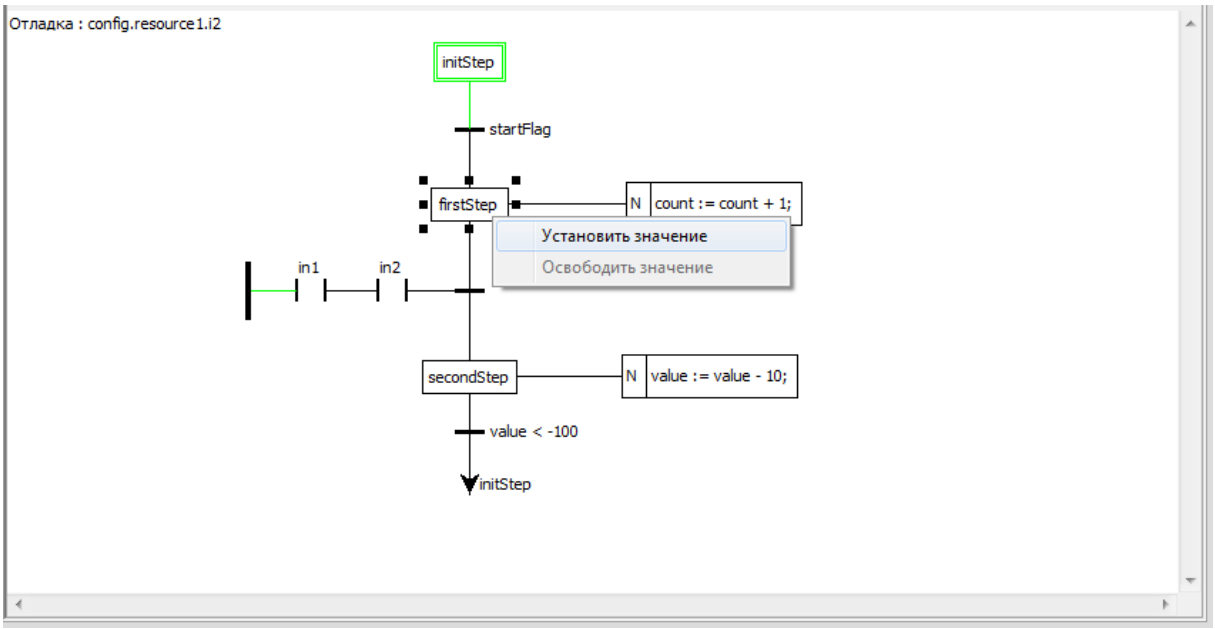


Рисунок 183 – Пример отладки SFC диаграммы

После установки значения шага в TRUE с помощью режима отладки, шаг будет выделен голубым цветом. Как можно заметить по рисунку 184, т.к. шаг «firstStep» стал активным, блок действий, ассоциированный с ним, так же стал активным (выделен зелёным цветом), а действия внутри него, в данном случае одно действие по увеличению переменной «count» на 1:

count := count + 1;

стало выполняться.

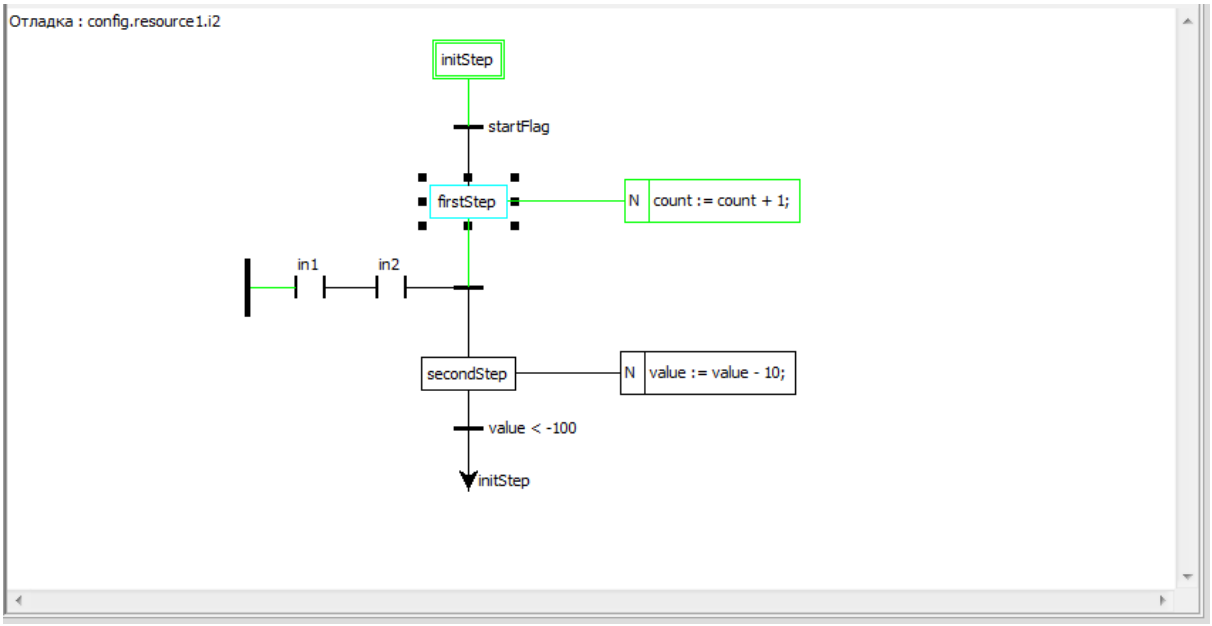


Рисунок 184 – Пример отладки SFC диаграммы

Так как квалификатор этого действия – N, то оно будет выполняться до тех пор, пока шаг активен.

8.7 График изменения значения переменной

Среда разработки Beremiz так же позволяет отображать в виде графика изменение значения переменной в режиме отладки. Для вывода панели с графиком, необходимо выбрать в панели экземпляра кнопку отображения графика изменения значения переменной в режиме отладки для необходимой переменной, выделенную красным цветом на рисунке 185:

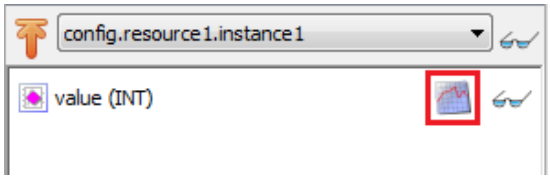


Рисунок 185 – Выбор кнопки на панели экземпляров для отображения графика изменения значения переменной

Появившееся панель графика изменения переменной (рисунок 186) позволяет отслеживать то, как значение определённой переменной изменяется в течение времени.

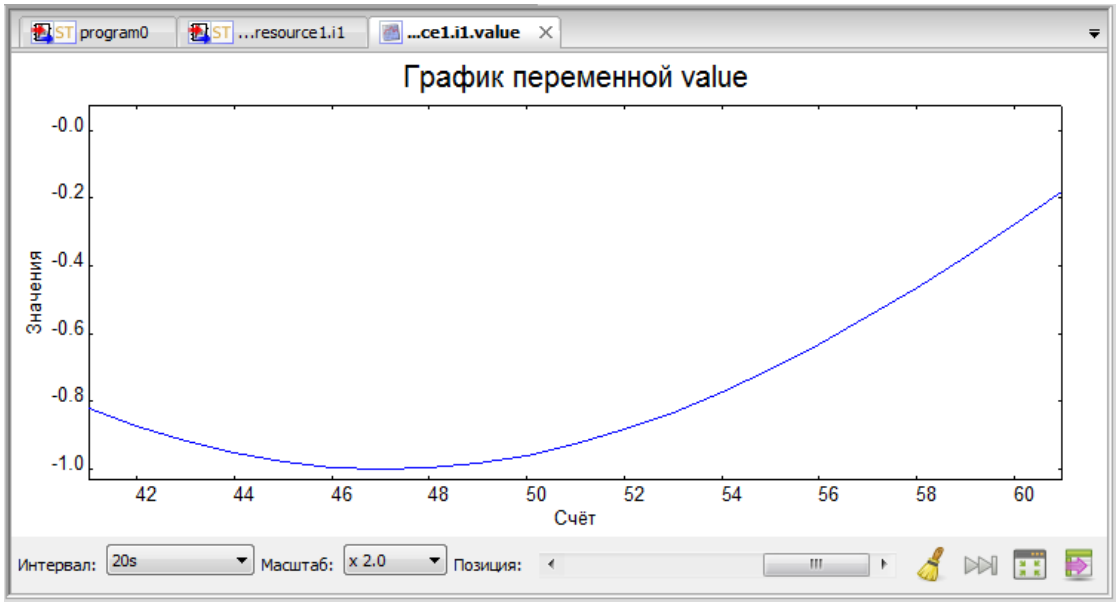


Рисунок 186 – График изменения переменной

На данной панели можно установить интервал обновления и масштаб отображения графика, а так же передвигать позицию графика.

График, изображенный на рисунке 186, соответствует изменению переменной из программного модуля на рисунке 187:

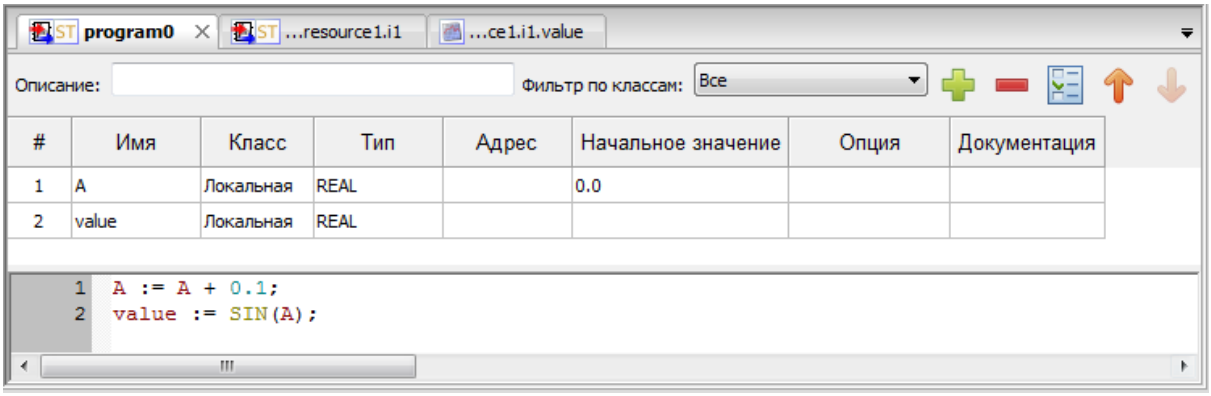


Рисунок 187 – Пример программного модуля тип «Программа» на языке ST

Как видно из графика, переменная «value» изменяется по закону синуса, что соответствует алгоритму, записанному на языке ST.

ПРИЛОЖЕНИЕ 1

УСТАНОВКА BEREMIZ

Установка среды разработки Beremiz под операционную систему Windows 7:

1) Запустить установку, дважды щелкнув левой клавишей мыши на файле «IDE Beremiz.exe»;
2) Выбрать язык и нажать на кнопку «Далее» (рисунок 1.1.)

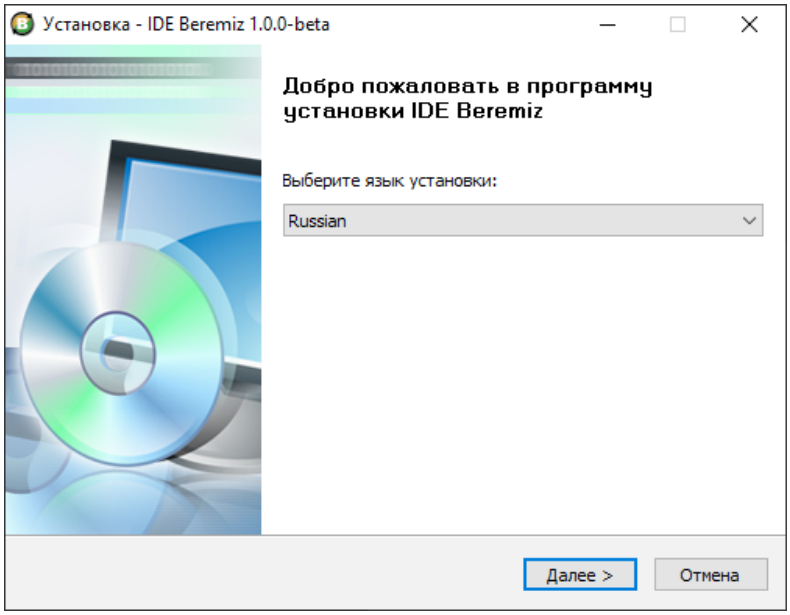


Рисунок 1.1 – Выбор языка установки

3) Указать директорию, куда будет произведена установка (рисунок 1.2)

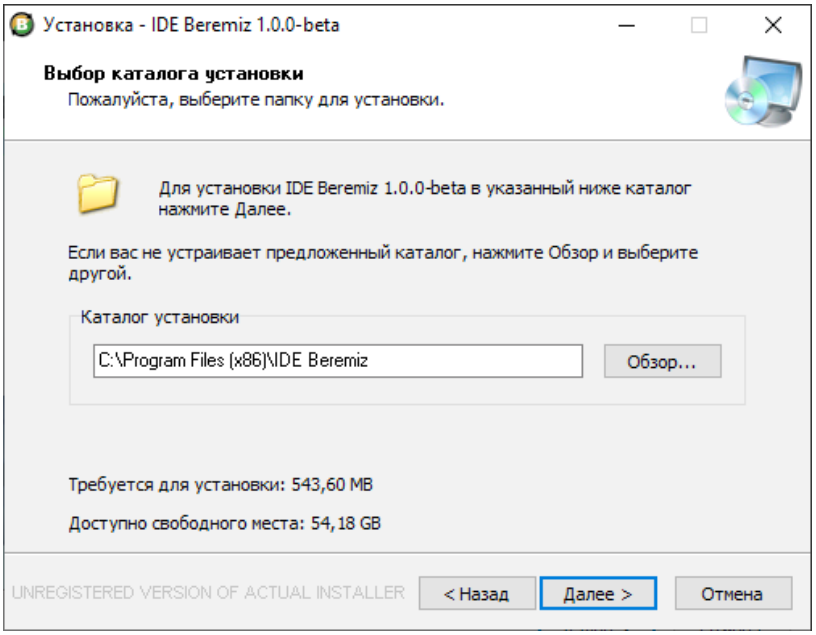


Рисунок 1.2 – Выбор директории установки

4) Выбрать группу в меню Программы и дать разрешение на создание ярлыка на Рабочем столе (рисунок 1.3)

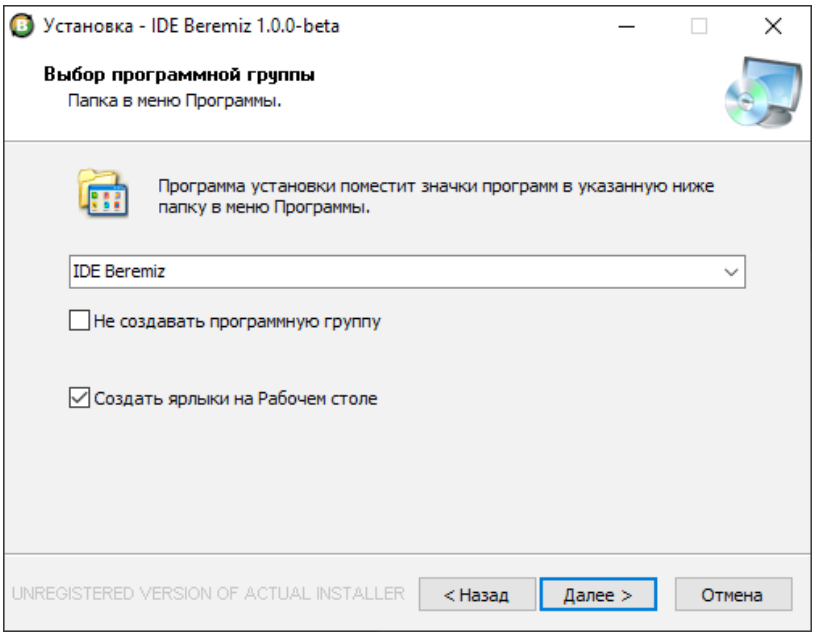


Рисунок 1.3 – Выбор группы в меню Пуск и разрешение на создание ярлыка на Рабочем столе

5) Подтвердить установку (рисунок 1.4)

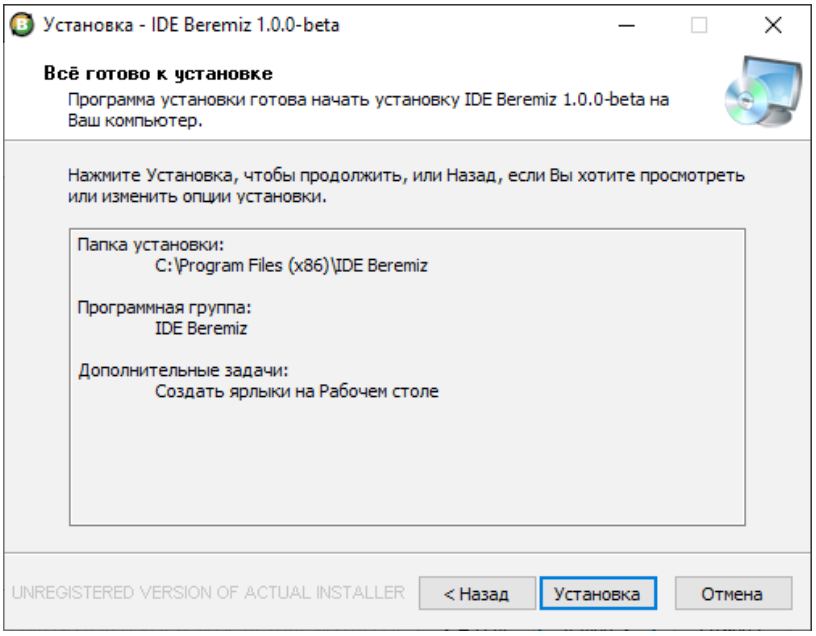


Рисунок 1.4 – Подтверждение установки

6) Дождаться завершения установки (может занять некоторое время).

7) Завершить работу установщика, щелкнув левой кнопкой мыши на кнопке «Готово» (рисунок 1.5)

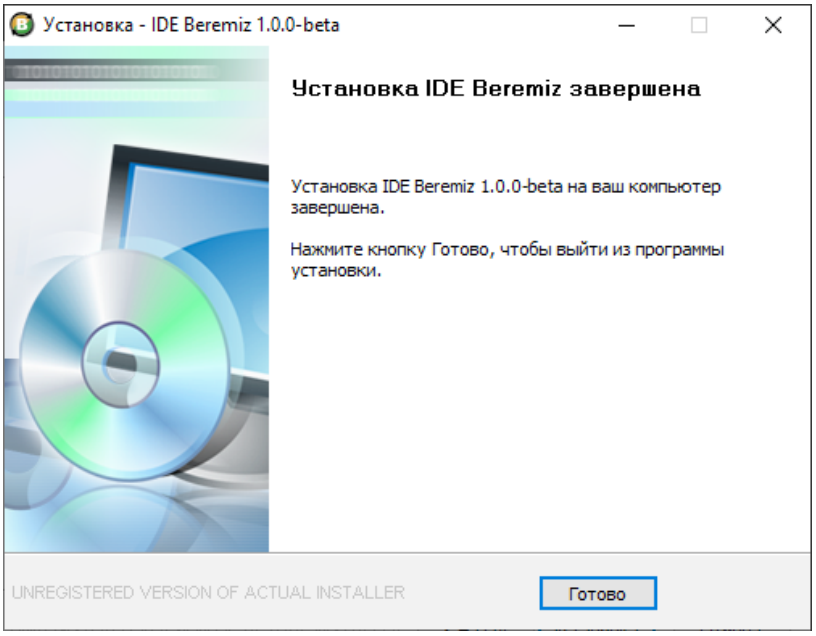


Рисунок 1.5 – Подтверждение установки

ПРИЛОЖЕНИЕ 2

СТАНДАРТНАЯ БИБЛИОТЕКА АЛГОРИТМОВ

Функции и функциональные блоки представляют собой предопределённые элементы, которые могут быть использованы при написании алгоритмов и логики программных модулей типа «Функциональный блок» и «Программ», как на текстовых, так и на графических языках стандарта IEC 61131-3.

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

Стандартные функциональные блоки

Бистабильный SR-триггер

Данный функциональный блок представляет собой бистабильный SR-триггер, с доминирующим входом S (Set). Выход Q1 становится "1", когда вход S1 становится "1". Это состояние сохраняется, даже если S1 возвращается обратно в "0". Выход Q1 возвращается в "0", когда вход R становится "1". Если входы S1 и R находятся в "1" одновременно, доминирующий вход S1 установит выход Q1 в "1". Когда функциональный блок вызывается первый раз, начальное состояние Q1 это "0".

Бистабильный RS-триггер

Данный функциональный блок представляет собой бистабильный RS-триггер, с доминирующим входом R (Reset). Выход Q1 становится "1", когда вход S становится "1". Это состояние сохраняется, даже если S возвращается обратно в "0". Выход Q1 возвращается в "0", когда вход R1 становится "1". Если входы S и R1 находятся в "1" одновременно, доминирующий вход R1 установит выход Q1 в "0". Когда функциональный блок вызывается первый раз, начальное состояние Q1 это "0".

SEMA – Семафор

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

R_TRIG – Индикатор нарастания фронта

Данный функциональный блок представляет собой индикатор нарастания фронта, который генерирует на выходе одиночный импульс при нарастании фронта сигнала. Выход Q становится "1", если происходит переход из "0" в "1" на входе CLK . Выход остается в состоянии "1" от одного выполнения блока до следующего (один цикл); затем выход возвращается в "0".

F_TRIG – Индикатор спада фронта

Данный функциональный блок представляет собой индикатор спада фронта, который генерирует на выходе одиночный импульс при спаде фронта сигнала.

Выход Q становится "1", если происходит переход из "1" в "0" на входе CLK . Выход будет оставаться в состоянии "1" от одного выполнения блока до следующего; затем выход возвращается в "0".

CTU – инкрементный счётчик

Данный функциональный блок представляет собой инкрементный счётчик. Сигнал "1" на входе R вызывает присваивание значения "0" выходу CV . При каждом переходе из "0" в "1" на входе CU значение CV увеличивается на 1. Когда CV >= PV, выход Q устанавливается в "1".

Примечание: Счетчик работает только до достижения максимального значения используемого типа данных. Переполнения не происходит.

Входы CU, RESET и выход Q типа BOOL, вход PV и выход CV типа WORD.

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

CTD – декрементный счётчик

Данный функциональный блок представляет собой декрементный счётчик. Сигнал "1" на входе LD вызывает присваивание значения на входе PV выходу CV . При каждом переходе из "0" в "1" на входе CD значение CV уменьшается на 1.

Когда CV <= 0, выход Q принимает значение "1".

Примечание: Счетчик работает только до достижения минимального значения используемого типа данных. Переполнения не происходит.

CTUD – реверсивный счётчик

Данный функциональный блок представляет собой реверсивный счётчик. Сигнал "1" на входе R вызывает присваивание значения "0" выходу CV . Сигнал "1" на входе LD вызывает присваивание значения на входе PV выходу CV . При каждом переходе из "0" в "1" на входе CU значение CV увеличивается на 1. При каждом переходе из "0" в "1" на входе CD значение CV уменьшается на 1.

Если сигнал "1" приходит одновременно на входы R и LD, вход R обрабатывается первым.

Когда CV >= PV, выход QU имеет значение "1".

Когда CV <= 0, выход QD принимает значение "1".

Примечание: 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.

TP – повторитель импульсов

Данный функциональный блок представляет собой повторитель импульсов и используется для генерирования импульса с заданной продолжительностью. Если IN становится "1", Q становится "1", и начинается отсчет внутреннего времени (ET). Если внутреннее время достигает значения PT, Q становится "0" (независимо от IN). Отсчет внутреннего времени останавливается/сбрасывается, если IN становится "0". Если внутреннее время не достигло значения PT, импульс IN не влияет на внутреннее время. Если внутреннее время достигло значения PT, и IN равен "0", отсчет внутреннего времени останавливается/сбрасывается, и Q становится "0".

TON – таймер с задержкой включения

Данный функциональный блок представляет собой таймер с задержкой включения. Он запускается, когда состояние сигнала на входе меняется от 0 к 1 и устанавливает на выходе 1 по истечении заданного времени.

Если IN становится "1", запускается отсчет внутреннего времени (ET). Если внутреннее время достигает значения PT, Q становится "1". Если IN становится "0", Q становится "0", а подсчет внутреннего времени останавливается/сбрасывается. Если IN становится "0" до того, как внутреннее время достигло значения PT, подсчет внутреннего времени останавливается/сбрасывается, а выход Q не устанавливается в "0".

TOF – таймер с задержкой отключения

Данный функциональный блок представляет собой таймер с задержкой отключения. Он запускается, когда состояние сигнала на входе меняется от 1 к 0 и устанавливает на выходе 0 по истечении заданного времени.

Если IN становится "1", Q становится "1".

Если IN становится "0", запускается отсчет внутреннего времени (ET).

Если внутреннее время достигает значения PT, Q становится "0".

Если IN становится "1", Q становится "1", а подсчет внутреннего времени останавливается/сбрасывается.

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

Дополнительные функциональные блоки

RTC – часы реального времени

Данный функциональный блок представляет собой часы реального времени и имеет много вариантов использования, включая добавление временных отметок, для установки даты и времени в формируемых отчетах, в аварийных сообщениях и т.д.

Вход PDT (Preset DT) предназначен для установки времени. Часы начинают отсчет времени от значения PDT. Выход Q (BOOL) повторят значение EN. Выход CDT (Current DT) дает текущее значение даты и времени.

INTEGRAL – Интеграл

Функциональный блок интеграл интегрирует входное значение XIN по времени.

DERIVATIVE – Производная

Функциональный блок производная выдаёт значение XOUT пропорционально скорости изменения входного параметра XIN.

PID

Пропорционально-интегрально-дифференциальный регулятор

Данный функциональный блок представляет 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.

HYSTERESIS – гистерезис

Функциональный блок гистерезис предоставляет выходное гистерезисное булевское значение, которое определяется разницей вводных параметров XIN1 и XIN2 (типа REAL с плавающей точкой).

Преобразования типов

Данный набор функций предназначен для всех возможных и корректных, согласно стандарту IEC 61131-3, преобразований между типами данных.

Числовые операции

ABS – модуль числа

Данная функция возвращает в OUT модуль входного числа IN.

SQRT – квадратный корень

Данная функция возвращает в OUT квадратный корень входного числа IN.

LN – натуральный логарифм

Данная функция возвращает в OUT значение натурального логарифма от IN.

LOG – логарифм по основанию 10

Данная функция возвращает в OUT значение логарифма по основанию 10 от IN.

EXP – возведение в степень экспоненты

Данная функция возвращает в OUT значение экспоненты, возведённой в степень IN.

SIN – синус

Данная функция возвращает в OUT значение синуса IN.

COS – косинус

Данная функция возвращает в OUT значение косинуса IN.

TAN – тангенс

Данная функция возвращает в OUT значение тангенса IN.

ASIN – арксинус

Данный функциональный блок возвращает в OUT значение арксинуса IN.

ACOS – арккосинус

Данная функция возвращает в OUT значение арккосинуса IN.

ATAN – арктангенс

Данная функция возвращает в OUT значение арктангенса IN.

Арифметические операции

ADD – сложение

Данная функция возвращает в OUT результат сложения IN1 и IN2.

MUL – умножение

Данная функция возвращает в OUT результат умножения IN1 и IN2.

SUB – вычитание

Данная функция возвращает в OUT результат вычитания из IN1 значения IN2.

DIV – деление

Данная функция возвращает в OUT результат деления IN1 на IN2.

MOD – остаток от деления

Данная функция возвращает в OUT остаток от деления IN1 на IN2.

EXPT – возведение в степень

Данная функция возвращает в OUT значение IN1 возведённое в степень IN2.

MOVE – присвоение

Данная функция возвращает в OUT значение IN.

Временные операции

ADD_TIME – сложение переменных типа TIME

Данная 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.

ADD_TOD_TIME – сложение времени дня TOD с интервалом времени TIME

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

ADD_DT_TIME – прибавление промежутка времени TIME к моменту времени DT

Данная 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.

MULTIME – умножение времени TIME на число

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

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.

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.

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.

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.

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.

Битовые операции

SHL – арифметический сдвиг влево

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

SHR – арифметический сдвиг вправо

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

ROR – циклический сдвиг направо

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

ROL – циклический сдвиг влево

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

AND – побитовое И

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

OR – побитовое ИЛИ

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

XOR – побитовое исключающее ИЛИ

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

NOT – побитовая инверсия

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

Операции выбора

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.

MAX – максимум

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

MIN – минимум

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

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.

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.

Операции сравнения

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.

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.

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.

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.

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.

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.

Строковые операции с переменными типа STRING

LEN – длина строки

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

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.

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.

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.

CONCAT – объединение двух переменных STRING

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

CONCAT_DAT_TOD – объединение (конкатенация) времени

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

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.

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.

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.

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 3

ОПИСАНИЕ ЯЗЫКА ST

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 также, 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.

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 2, раздел «Строковые операции с переменными типа STRING»).

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

В таблице 3.1 приведены значения по умолчанию, соответствующие описанным выше типам.

Таблица 3.1 – 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 3.2.

Таблица 3.2 – 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 colspan="2">ANY_DATE</td></tr>
<tr><td rowspan="2">BOOL
BYTE
WORD
DWORD
LWORD</td><td colspan="2">ANY_INT</td><td>ANY_REAL</td><td rowspan="2">DATE TIME_OF_DAY
DATE_AND_TIME</td><td rowspan="2">TIME
STRING
и другие типы данных</td></tr>
<tr><td>INT
SINT
DINT
LINT</td><td>UINT
USINT
UDINT
ULINT</td><td>REAL
LREAL</td></tr>
</table>

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.

Арифметические операции

К арифметическим операциям относятся:

- «+» – сложение;
- «-» – вычитание;
- «*» – умножение;
- «/» – деление;
- «mod» – остаток от целочисленного деления.

Приоритет операций в выражениях указан в таблице 3.4 (чем выше приоритет, тем раньше исполняется операция).

Логические (побитовые) операции

К данным операциям относятся:

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

Операции сравнения

Поддерживаются следующие операции сравнения:

- = – сравнение на «равенство»;
- <> – сравнение на «неравенство»;
- «>» – сравнение на «больше»;
- «>=» – сравнение на «не меньше»;
- «<» – сравнение на «меньше»;
- «<=» – сравнение на «не больше».

В качестве результата сравнения всегда используется значение типа BOOL.

Присвоение

Для обозначения присвоения используется парный знак «:=» (двоеточие равно). 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 3.4 приведены приоритеты при выполнении описанных выше операций.

Таблица 3.4 – 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

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;
```

Цикл FOR

Служит для задания 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.

Цикл WHILE

Служит для определения 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).

Цикл REPEAT UNTIL

Служит для определения 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).

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 3.5 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 3.5 – 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;	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;

Неправильная запись	Правильная запись
11: i := 1; 12: END_CASE;	11: i := 1; 12: END_CASE;
Диапазоны в строках 04 и 07 пересекаются	

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: 6, 20..9: 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, 9..20: 08: k := k - 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;
В строке 07 диапазон значений задан неправильно	

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

ПРИЛОЖЕНИЕ 4

ОПИСАНИЕ ЯЗЫКА IL

Общие сведения о языке IL

IL (Instruction List) представляет собой текстовый язык программирования низкого уровня, который очень похож на Assembler, но к конкретной архитектуре процессора не привязан. 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 разветвлений.

Операторы языка

Основа языка программирования IL, как и в случае Assembler, это переходы по меткам и аккумулятор. В аккумулятор загружается значения переменной, а дальнейшее выполнение алгоритма представляет собой извлечение значения из аккумулятора и совершение над ним операций. Далее в таблице 4.1 приведены операторы языка IL.

Таблица 4.1 – 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

Оператор	Описание
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

Оператор	Описание
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

Пример программы на языке IL

На рисунке 4.1 приведён пример программы на языке IL, которая эквивалентна следующему логическому выражению C = A AND NOT B:

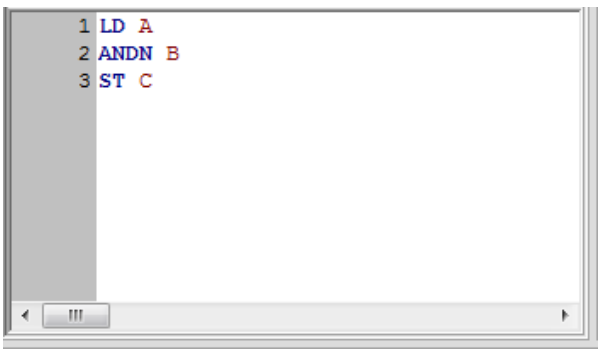


Рисунок 4.1 – Пример программы на языке IL

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 5

ОПИСАНИЕ ЯЗЫКА FBD

Общие сведения о языке FBD

FBD (Function Block Diagram) – это графический язык программирования высокого уровня, обеспечивающий управление потока данных всех типов. Позволяет использовать мощные алгоритмы простым вызовом функций и функциональных блоков. Удовлетворяет непрерывным динамическим процессам. Замечательно 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 2. 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.

Основные понятия и конструкции языка

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

Переменные бывают входные, выходные и входные/выходные. На рисунке 5.1 показаны: входная переменная – «in_var», выходная переменная – «out_var» и входная/выходная переменная – «in_out_var».

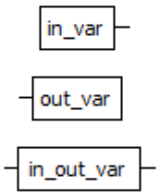


Рисунок 5.1 – Изображение переменной в языке FBD

Графическое изображение функции приведено на рисунке 5.2. С левой стороны располагаются входы (IN1 и IN2), с правой стороны выходы (OUT).

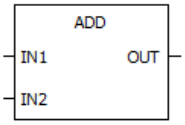


Рисунок 5.2 – Изображение функции в языке FBD

Аналогично, изображение функционального блока, приведённое на рисунке 5.3, имеет с левой стороны входы (S1 и R), с правой стороны выход (Q1).

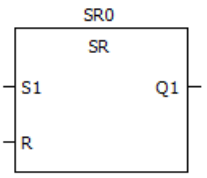


Рисунок 5.3 – Изображение функционального блока в языке FBD

Соответственно, переменные соединяются с входными и выходными параметрами функций и функциональных блоков. 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.

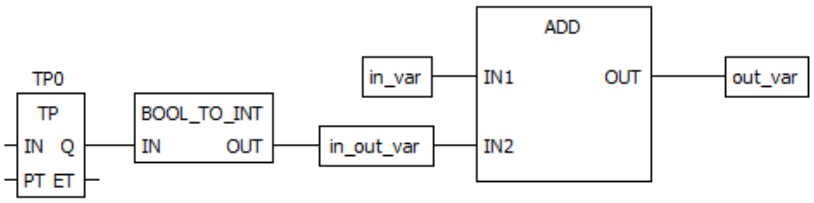


Рисунок 5.4 – Пример соединения переменных, функций и функциональных блоков

Все функциональные блоки могут быть вызваны с дополнительными (необязательными) формальными параметрами: EN (входом) и ENO (выходом). Пример такого функционального блока приведен на рисунке 5.5.

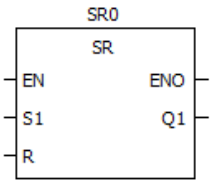


Рисунок 5.5 – Изображение элементарного функционального блока с параметрами EN/ENO

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 рисунке 5.6:

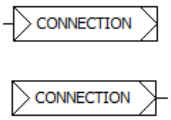


Рисунок 5.6 – Изображение соединений в языке FBD

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 рисунке 5.7 показан пример, в котором выходное значение OUT функции BOOL_TO_INT передаётся на вход IN2 функции ADD:

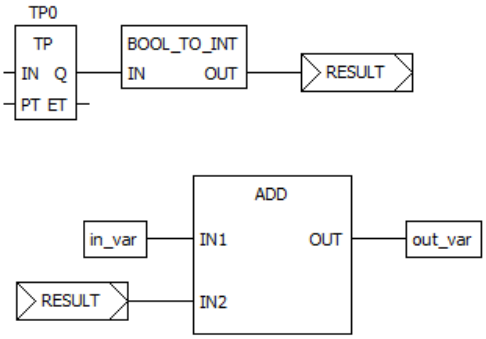


Рисунок 5.7 – Пример использования соединения на FBD диаграмме

Пример программы на языке FBD

На рисунке 5.8 приведена FBD диаграмма, состоящая из следующих функциональных блоков: SR0, AND, TP0.

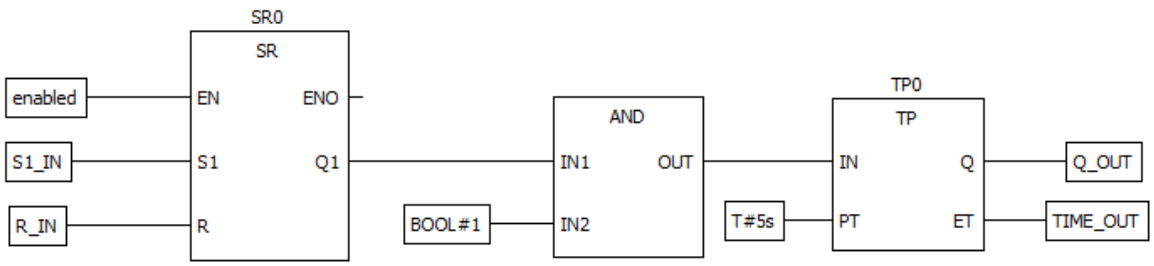


Рисунок 5.8 – пример FBD диаграммы

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 (рисунок 5.9).

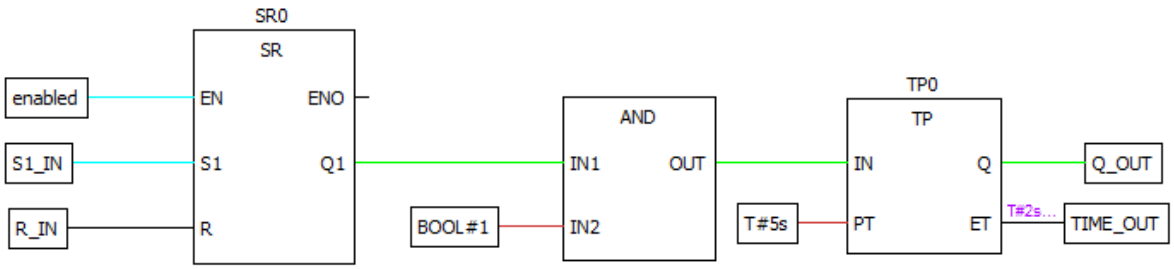


Рисунок 5.9 – Выполнение FBD диаграммы

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 (рисунок 5.10).

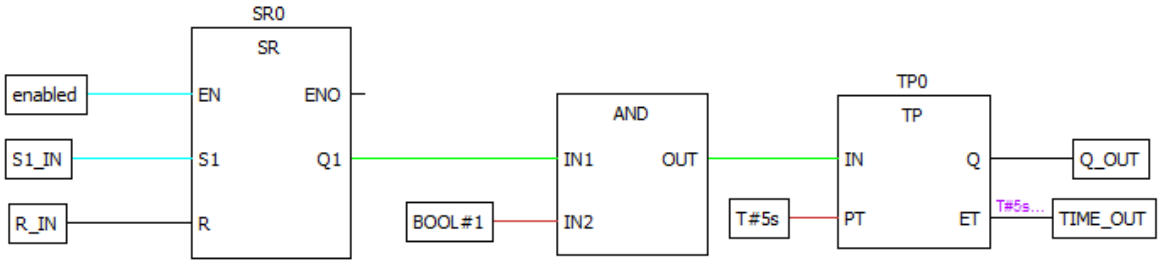


Рисунок 5.10 – Выполнение FBD диаграммы

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

ПРИЛОЖЕНИЕ 6

ОПИСАНИЕ ЯЗЫКА LD

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 рисунке 6.1 приведён пример программы на языке LD (слева) и ее эквивалент в виде электрической цепи с реле и выключателями (справа).

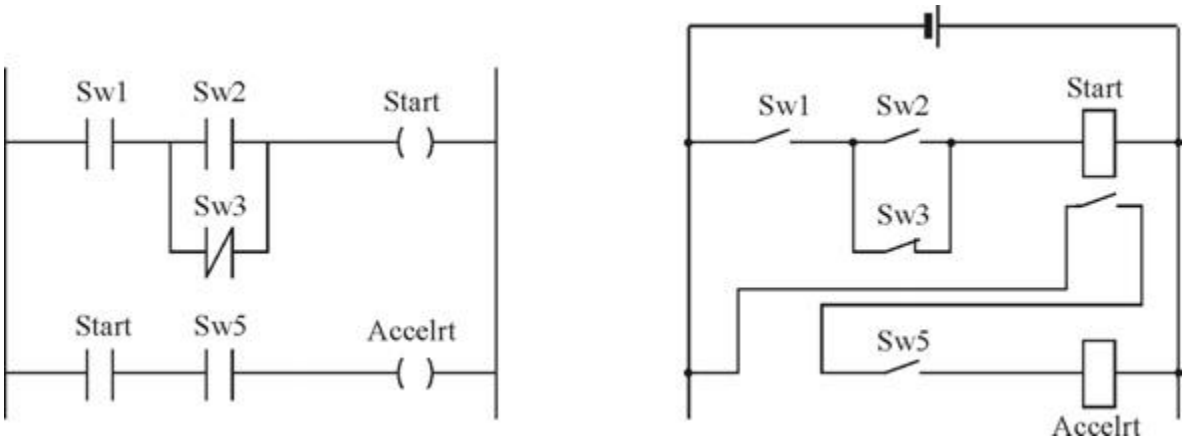


Рисунок 6.1 – Программа на языке LD (слева) и ее эквивалент в виде электрической (справа)

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, подключенные 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.

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 передает состояние «логическое ИЛИ». Если контакты соединены последовательно, то соединение передаёт «логическое И».

Контакт может быть инвертируемым. Такой контакт обозначается с помощью символа |/| и передает 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.

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 (рисунок 6.2) состояние левой связи передается в правую связь, если состояние связанного логического фактического параметра TRUE. Иначе, состояние правой связи FALSE.

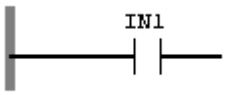


Рисунок 6.2 – Нормальный контакт

Для инверсных контактов (рисунок 6.3) состояние левой связи передается в правую связь, если состояние связанного логического фактического параметра FALSE. Иначе, состояние правой связи TRUE.

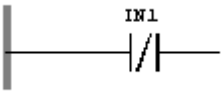


Рисунок 6.3 – Инверсный контакт

В контактах для обнаружения нарастания фронта (рисунок 6.4) 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.

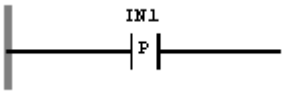


Рисунок 6.4 – Контакт для обнаружения нарастания фронта

В контактах для обнаружения спада фронта (рисунок 6.5) 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.

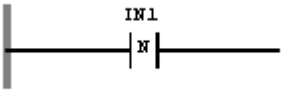


Рисунок 6.5 – Контакт для обнаружения спада фронта

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 (рисунок 6.6) состояние левой связи передается в связанный логический фактический параметр и в правую связь.

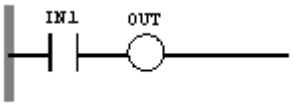


Рисунок 6.6 – Нормальные контакт и катушка

В инвертирующей катушке (рисунок 6.7) 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.

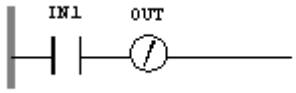


Рисунок 6.7 – Нормальный контакт и инвертированная катушка

В катушке установки (рисунок 6.8) 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.

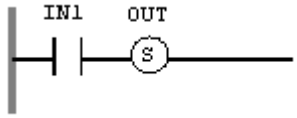


Рисунок 6.8 – Нормальный контакт и катушка установки

В катушке сброса (рисунок 6.9) 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.

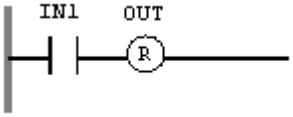


Рисунок 6.9 – Нормальный контакт и катушка сброса

В катушке обнаружения нарастания фронта (рисунок 6.10) 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.

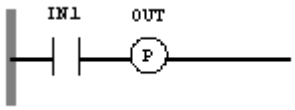


Рисунок 6.10 – Нормальный контакт и катушка обнаружения нарастания фронта

В катушке обнаружения спада фронта (рисунок 6.11) 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.

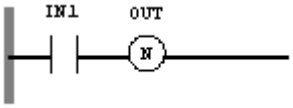


Рисунок 6.11 – Нормальный контакт и катушка обнаружения спада фронта

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.

Шина питания

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

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 (рисунок 6.12), и результатом является программа, написанная на языке LD, реализующая логическое выражение:

```
C = A AND NOT B
```

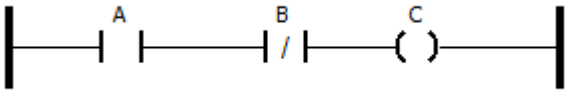


Рисунок 6.12 – Пример LD диаграммы, реализующей логическое выражение
C = A AND NOT B

ПРИЛОЖЕНИЕ 7

ОПИСАНИЕ ЯЗЫКА SFC

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.

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.

Шаг

Наиболее важным элементом языка SFC является шаг, который описывает одну операцию. Шаг изображается в виде прямоугольника с собственным именем внутри (рисунок 7.1).

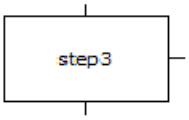


Рисунок 7.1 – Графическое представление «Шага» языка SFC

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 (рисунок 7.2).

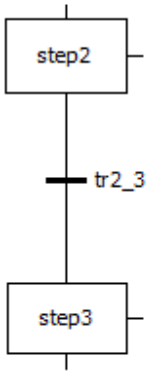


Рисунок 7.2 – Шаги «step2» и «step3», соединённые переходом «tr2_3»

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

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 рисунке 7.3 приведён пример перехода между шагом «Step3» и «Step5» с именем «transition4».

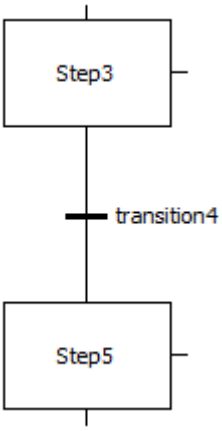


Рисунок 7.3 – Переход между шагами «Step3» и «Step5» с предопределённым условием «transition4»

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 рисунок 7.4 представлен переход между шагами «Step6» и «Step7» в виде обычного условия: level > 10

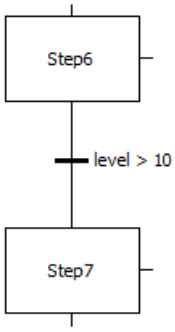


Рисунок 7.4 – Переход между шагами «Step6» и «Step7» с предопределённым условием
«level > 10»

На рисунок 7.5 представлен переход между шагами «Step8» и «Step9» в виде значения логического выражения «AND» на языке FBD:

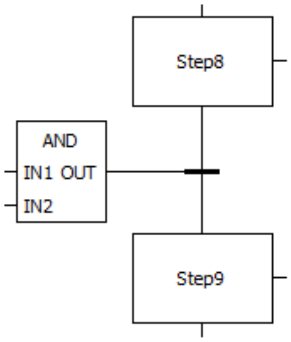


Рисунок 7.5 – Переход между шагами «step8» и «step9», заданный «логическим И» на языке FBD

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

Блок действий

Каждый шаг имеет нулевое или большее количеством действий, объединённых, как правило, на диаграмме, в блок действий. На рисунке 7.6 показан примера шага «evaluateStep» и связанный с ним блок действий.

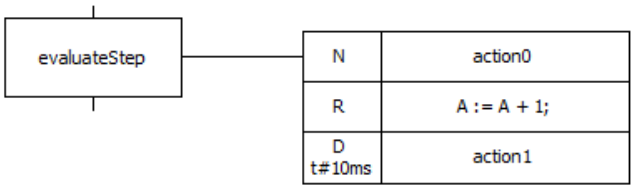


Рисунок 7.6 – Шаг «evaluateStep» и связанный с ним блок действий, содержащий 3 действия

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 (рисунок 7.6 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. Квалификаторы также могут ограничивать время выполнения каждого действия в шаге.

«Прыжок» – переход на произвольный шаг

Шаг может быть также заменён «прыжком». 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 «прыжка». Такие переходы нужны для того, чтобы избежать пересекающихся и идущих вверх соединений. На рисунок 7.7 показана SFC диаграмма, содержащая два «прыжка».

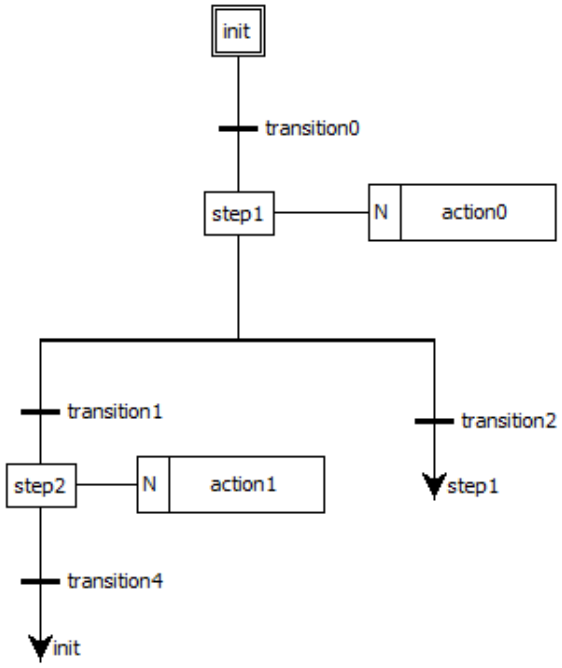


Рисунок 7.7 – SFC диаграмма, содержащая «прыжки»

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

Дивергенция и конвергенция

Дивергенция – это множественное соединение в направлении от одного шага к нескольким переходам. Активируется только одна из ветвей. Условия, связанные с различными переходами в начале дивергенции, не являются взаимоисключающими по умолчанию. Взаимоисключение должно быть явно задано в условиях переходов, чтобы гарантировать, что во время выполнения программы активируется одна конкретная ветвь. Пример дивергенции на SFC диаграмме приведён на рисунке 7.8 и выделен красным цветом:

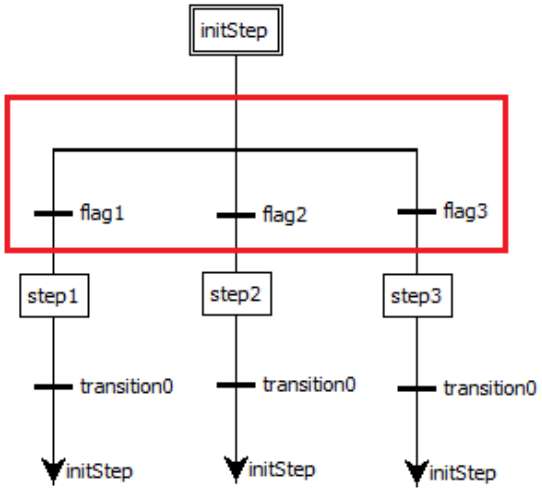


Рисунок 7.8 – Дивергенция на SFC диаграмме

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 рисунке 7.9 и выделен красным цветом:

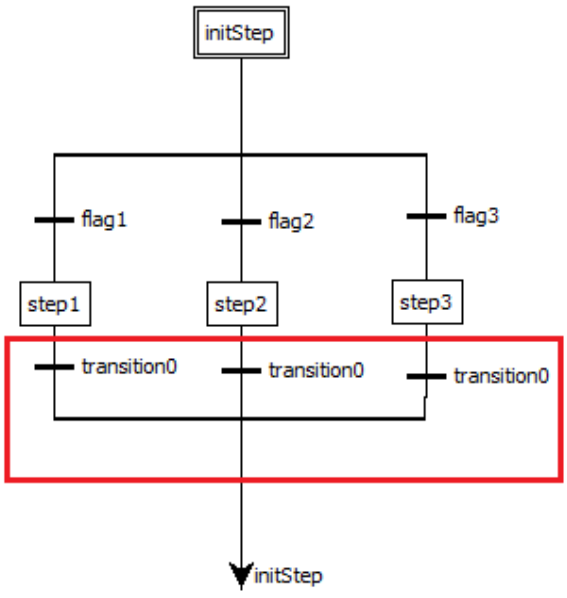


Рисунок 7.9 – Конвергенция на SFC диаграмме

Параллельная дивергенция – это множественное соединение, направленное от одного перехода к нескольким шагам. Она соответствует параллельному выполнению операций процесса. Пример параллельной дивергенции на SFC диаграмме приведён на рисунке 7.10 и выделен красным цветом:

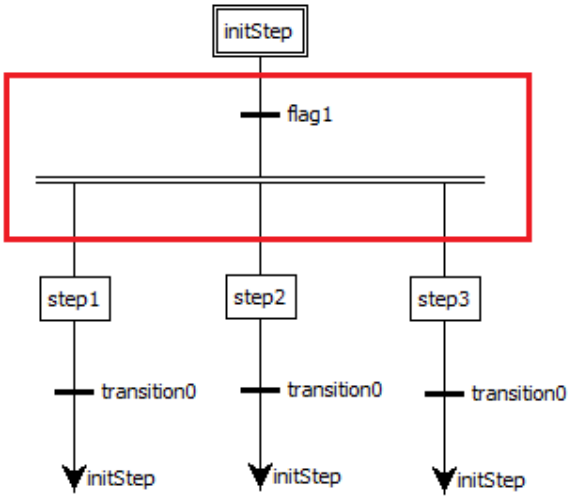


Рисунок 7.10 – Параллельная дивергенция на SFC диаграмме

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 рисунке 7.11 и выделен красным цветом:

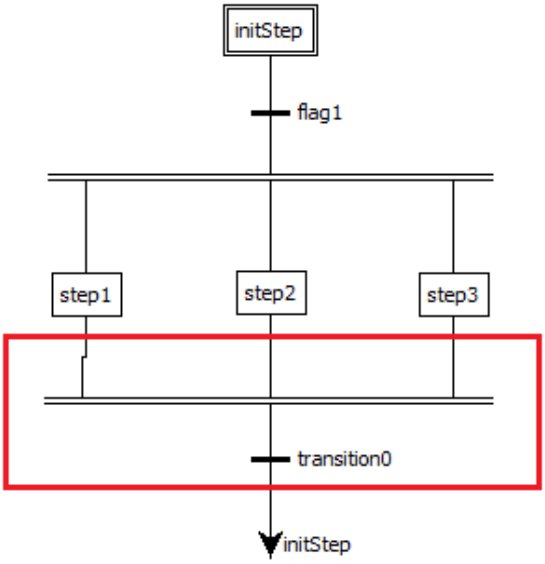


Рисунок 7.11 – Параллельная конвергенция на SFC диаграмме

Пример программы на языке SFC

На рисунке 7.12 приведен пример SFC диаграммы состоящей из начального шага «initStep»,

шагов «firstStep» и «secondStep» и 3 перехода.

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.

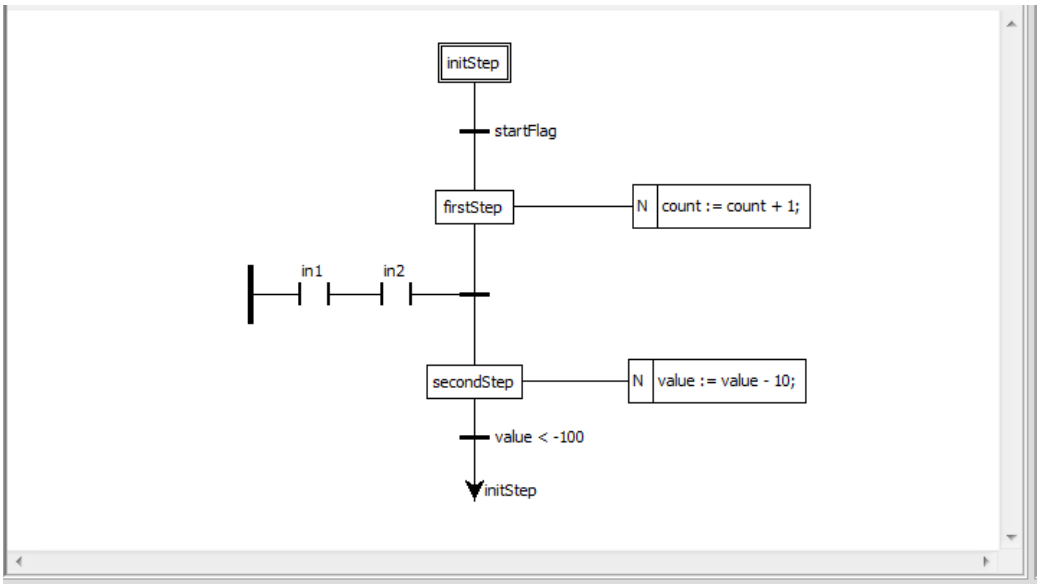


Рисунок 7.12 – SFC диаграмма