

ООО "ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ "

ПЛМ-2004

Руководство по программированию v 3.4

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

08.04.2022

СОДЕРЖАНИЕ

| | |
|--|----|
| АННОТАЦИЯ | 8 |
| 1 Назначение и условия применения среды разработки..... | 9 |
| 2 Характеристики среды разработки..... | 10 |
| 3 Обобщенная схема работы | 11 |
| 4 Основные термины и определения..... | 13 |
| 5 Основные компоненты среды разработки | 15 |
| 5.1 Главное меню | 15 |
| 5.2 Панель инструментов..... | 17 |
| 5.2.1 Кнопки управления проектом | 17 |
| 5.2.2 Кнопки сборки проекта и установления связи с целевым устройством | 18 |
| 5.3 Дерево проекта..... | 19 |
| 5.3.1 Добавление элемента в дерево проекта | 20 |
| 5.3.2 Копирование элемента в дерево проекта..... | 20 |
| 5.3.3 Удаление элемента из дерева проекта | 21 |
| 5.3.4 Переименование элемента в дереве проекта | 22 |
| 5.3.5 Изменение типа РОУ элемента в дереве проекта | 22 |
| 5.3.6 Неиспользуемые элементы в дереве проекта..... | 23 |
| 5.4 Панель списка констант и переменных..... | 23 |
| 5.4.1 Панель списка глобальных констант и переменных | 24 |
| 5.4.2 Панель списка локальных констант и переменных..... | 25 |
| 5.4.3 Панель списка переменных функциональных блоков..... | 26 |
| 5.4.4 Имя переменной | 27 |
| 5.4.5 Класс переменной | 29 |
| 5.4.6 Тип переменной | 29 |
| 5.4.7 Адрес переменной | 29 |
| 5.4.8 Квалификатор переменной | 30 |
| 5.4.9 Фильтрация списка переменных..... | 31 |
| 5.4.10 Копирование и вставка переменных | 31 |
| 5.4.11 Сортировка переменных..... | 32 |
| 5.4.12 Добавление и удаление переменных | 33 |
| 5.5 Панель настройки проекта | 34 |
| 5.5.1 Конфигурационные переменные..... | 34 |
| 5.5.2 Группа переменной | 36 |

| | | |
|-------|---|----|
| 5.5.3 | Добавление и удаление групп переменных | 36 |
| 5.5.4 | Свойства проекта | 38 |
| 5.5.5 | Конфигурация | 40 |
| 5.5.6 | МЭК-код..... | 41 |
| 5.5.7 | Файлы проекта..... | 41 |
| 5.6 | Текстовые редакторы языков ST и IL | 42 |
| 5.7 | Графические редакторы языков FBD, SFC и LD | 43 |
| 5.7.1 | Редактор языка FBD..... | 43 |
| 5.7.2 | Редактор языка LD | 50 |
| 5.7.3 | Редактор языка SFC | 54 |
| 5.8 | Панель редактирования ресурса | 68 |
| 5.9 | Панель редактирования типа данных | 69 |
| 5.9.1 | Прямое наследование | 70 |
| 5.9.2 | Поддиапазон существующего типа..... | 70 |
| 5.9.3 | Перечисляемый тип | 71 |
| 5.9.4 | Массив | 71 |
| 5.9.5 | Структура | 72 |
| 5.10 | Панель экземпляров проекта | 73 |
| 5.11 | Дерево переменных..... | 75 |
| 5.12 | Панель библиотеки функций и функциональных блоков | 77 |
| 5.13 | Отладочная консоль..... | 79 |
| 5.14 | Поиск элементов в проекте | 80 |
| 5.15 | Панель отладки..... | 81 |
| 5.16 | Панель графика изменения значения переменной в режиме отладки | 83 |
| 6 | Основные компоненты среды разработки | 84 |
| 6.1 | Связывание регистров внешних модулей с переменными проекта | 84 |
| 6.2 | Адреса регистров внешних модулей | 86 |
| 7 | Основные компоненты среды разработки | 88 |
| 7.1 | Создание нового проекта | 88 |
| 7.2 | Настройка проекта | 89 |
| 7.2.1 | Глобальные переменные проекта | 90 |
| 7.2.2 | Настройки сборки проекта и соединения с целевым устройством | 91 |
| 7.2.3 | Данные о проекте..... | 92 |
| 7.3 | Программные модули..... | 93 |

| | | |
|-----------------------------------|--|-----|
| 7.3.1 | Программа | 94 |
| 7.3.2 | Функция | 100 |
| 7.3.3 | Функциональный блок..... | 104 |
| 7.4 | Ресурс | 106 |
| 7.4.1 | Глобальные переменные ресурса | 107 |
| 7.4.2 | Задачи и экземпляры ресурса | 107 |
| 7.4.3 | Задачи по событию..... | 111 |
| 7.5 | Типы данных | 114 |
| 7.6 | Расширения..... | 117 |
| 7.6.1 | Язык Си | 117 |
| 8 | СБОРКА, загрузка И ОТЛАДКА прикладной программы | 120 |
| 8.1 | Сборка проекта | 120 |
| 8.2 | Соединение с целевым устройством и передача исполняемого файла | 121 |
| 8.3 | Отладка текстовых языков..... | 124 |
| 8.4 | Отладка FBD диаграмм | 126 |
| 8.5 | Отладка LD диаграммы..... | 127 |
| 8.6 | Отладка SFC диаграммы | 128 |
| 8.7 | График изменения значения переменной..... | 130 |
| 8.8 | Эмуляция целевой программы..... | 132 |
| 8.8.1 | Главное меню. | 132 |
| 8.8.2 | Панель инструментов..... | 133 |
| 8.8.3 | Библиотека элементов..... | 134 |
| 8.8.4 | Монитор СОМ-порта. | 135 |
| 8.8.5 | Редактор эмулятора. | 135 |
| 8.8.6 | Конфигурация сервера PYRO. | 137 |
| 8.8.7 | Конфигурация клиента PYRO. | 138 |
| 8.8.8 | Добавление пользовательских элементов в эмулятор..... | 138 |
| PРИЛОЖЕНИЕ 1 | | 139 |
| УСТАНОВКА BEREMIZ | | 139 |
| ПРИЛОЖЕНИЕ 2 | | 142 |
| СТАНДАРТНАЯ БИБЛИОТЕКА АЛГОРИТМОВ | | 142 |
| ПРИЛОЖЕНИЕ 3 | | 152 |
| ОПИСАНИЕ ЯЗЫКА ST | | 152 |
| ПРИЛОЖЕНИЕ 4 | | 162 |

| | |
|--|-----|
| ОПИСАНИЕ ЯЗЫКА IL..... | 162 |
| ПРИЛОЖЕНИЕ 5 | 166 |
| ОПИСАНИЕ ЯЗЫКА FBD..... | 166 |
| ПРИЛОЖЕНИЕ 6 | 170 |
| ОПИСАНИЕ ЯЗЫКА LD | 170 |
| ПРИЛОЖЕНИЕ 7 | 175 |
| ОПИСАНИЕ ЯЗЫКА SFC | 175 |
| ПРИЛОЖЕНИЕ 8 | 183 |
| БИБЛИОТЕКА АЛГОРИТМОВ ФИЛЬТРАЦИИ СИГНАЛОВ | 183 |
| Функциональный блок FiltrD | 183 |
| Функциональный блок FiltrA | 183 |
| ПРИЛОЖЕНИЕ 9 | 185 |
| БИБЛИОТЕКА АЛГОРИТМОВ МАСШТАБИРОВАНИЯ СИГНАЛОВ | 185 |
| Функция ScaleA_Ka | 185 |
| Функция ScaleA_Kb | 186 |
| Функция ScaleA | 187 |
| ПРИЛОЖЕНИЕ 10 | 188 |
| БИБЛИОТЕКА ПИД-РЕГУЛЯТОРОВ | 188 |
| Функциональный блок PID1 | 188 |
| Функциональный блок PID2 | 192 |
| Функциональный блок PIDtun..... | 194 |
| ПРИЛОЖЕНИЕ 11 | 197 |
| БИБЛИОТЕКА РЕГУЛЯТОРОВ НЕЧЕТКОЙ ЛОГИКИ | 197 |
| Функциональный блок FLR1 | 197 |
| Функциональный блок FLR2 | 199 |
| Функциональный блок FLRtun..... | 200 |
| ПРИЛОЖЕНИЕ 12 | 202 |
| БИБЛИОТЕКА КАНАЛОВ ДИСКРЕТНОГО ВВОДА..... | 202 |
| Функциональный блок DIMode..... | 202 |
| Функциональный блок DIStatus | 203 |
| Функциональный блок DIVal | 204 |
| Функциональный блок DICntRst..... | 205 |
| Функциональный блок DICntT..... | 206 |
| Функциональный блок DISetCfgCntT..... | 207 |

| | |
|--|------------|
| Функциональный блок DIStateCfgCntT | 208 |
| Функциональный блок DICnt | 208 |
| Функциональный блок DISetCfgCnt..... | 209 |
| Функциональный блок DIStateCfgCnt | 210 |
| Функциональный блок DIEnc..... | 211 |
| Функциональный блок DISetCfgEnc | 212 |
| Функциональный блок DIStateCfgEnc | 214 |
| ПРИЛОЖЕНИЕ 13 | 216 |
| БИБЛИОТЕКА КАНАЛОВ АНАЛОГОВОГО ВВОДА | 216 |
| Функциональный блок AIMode | 216 |
| Функциональный блок AIStatus..... | 217 |
| Функциональный блок AIVal..... | 218 |
| ПРИЛОЖЕНИЕ 14 | 219 |
| БИБЛИОТЕКА КАНАЛОВ РЕЗИСТИВНЫХ ДАТЧИКОВ | 219 |
| Функциональный блок PtMode | 219 |
| Функциональный блок PtStatus | 220 |
| Функциональный блок PtVal | 221 |
| ПРИЛОЖЕНИЕ 15 | 223 |
| БИБЛИОТЕКА КАНАЛОВ ДАТЧИКОВ DS18N20 ШИНЫ 1-WIRE | 223 |
| Функциональный блок DtMode..... | 223 |
| Функциональный блок DtStatus | 224 |
| Функциональный блок DtGetID..... | 225 |
| Функциональный блок DtSetID..... | 226 |
| Функциональный блок DtVal | 227 |
| ПРИЛОЖЕНИЕ 16 | 229 |
| БИБЛИОТЕКА КАНАЛОВ ДИСКРЕТНОГО ВЫВОДА | 229 |
| Функциональный блок DOMode | 229 |
| Функциональный блок DOStatus..... | 230 |
| Функциональный блок DOVal..... | 231 |
| Функциональный блок DOFast | 232 |
| Функциональный блок DOPwm..... | 234 |
| Функциональный блок DOSafe | 235 |
| ПРИЛОЖЕНИЕ 17 | 237 |
| БИБЛИОТЕКА КАНАЛОВ АНАЛОГОВОГО ВЫВОДА | 237 |

| | |
|--|------------|
| Функциональный блок AOMode | 237 |
| Функциональный блок AOStatus | 238 |
| Функциональный блок AOVal | 239 |
| Функциональный блок AOFast | 240 |
| Функциональный блок AOSafe | 241 |
| ПРИЛОЖЕНИЕ 18 | 243 |
| Библиотека Modbus RTU master..... | 243 |
| Коды процесса исполнения..... | 243 |
| Коды ошибок формирования запроса | 243 |
| Коды ошибок обработки ответа..... | 244 |
| Общие принципы работы и использования блоков библиотеки. | 245 |
| Доступные интерфейсные порты для поддерживаемых целевых устройств..... | 246 |
| Вход блоков Сом. | 246 |
| Функциональный блок MbRtuReadBool | 246 |
| Функциональный блок MbRtuReadInt | 248 |
| Функциональный блок MbRtuReadDInt..... | 249 |
| Функциональный блок MbRtuReadWord..... | 250 |
| Функциональный блок MbRtuReadDWord | 251 |
| Функциональный блок MbRtuReadReal..... | 252 |
| Функциональный блок MbRtuWriteBool | 254 |
| Функциональный блок MbRtuWriteInt | 255 |
| Функциональный блок MbRtuWriteDInt..... | 257 |
| Функциональный блок MbRtuWriteWord..... | 258 |
| Функциональный блок MbRtuWriteDWord | 259 |
| Функциональный блок MbRtuWriteReal..... | 261 |
| Функциональный блок MbRtuDiag..... | 262 |
| Функциональный блок MbRtuSetTimeOutValue | 263 |
| Функциональный блок MbRtuGetTimeOutValue..... | 264 |
| ПРИЛОЖЕНИЕ 19 | 265 |
| БИБЛИОТЕКА EEPROM..... | 265 |
| Скрытое автоматическое назначение адресов..... | 265 |
| Структура типов регистров EEPROM | 265 |
| Функции и функциональные блоки | 268 |
| ПРИЛОЖЕНИЕ 20 | 277 |

| | |
|--|-----|
| БИБЛИОТЕКА СИСТЕМНЫХ ФУНКЦИЙ | 277 |
| Функция SoftReset..... | 277 |
| ПРИЛОЖЕНИЕ 21 | 278 |
| БИБЛИОТЕКА УПРАВЛЕНИЯ ШАГОВЫМ ДВИГАТЕЛЕМ | 278 |
| Функциональный блок StepStatus..... | 278 |
| Функциональный блок StepMotor | 279 |
| ПРИЛОЖЕНИЕ 22 | 282 |
| БИБЛИОТЕКА ЭЛЕМЕНТОВ ЭМУЛЯТОРА..... | 282 |
| Encoder | 282 |
| Slider..... | 283 |
| Stepper | 284 |
| Энкодер | 285 |
| ПРИЛОЖЕНИЕ 23 | 286 |
| ДОБАВЛЕНИЕ ПОЛЬЗОВАТЕЛЬСКОГО БЛОКА В ЭМУЛЯТОР..... | 286 |

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. В конце даётся перечень ссылочной документации.

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 3.1.1.

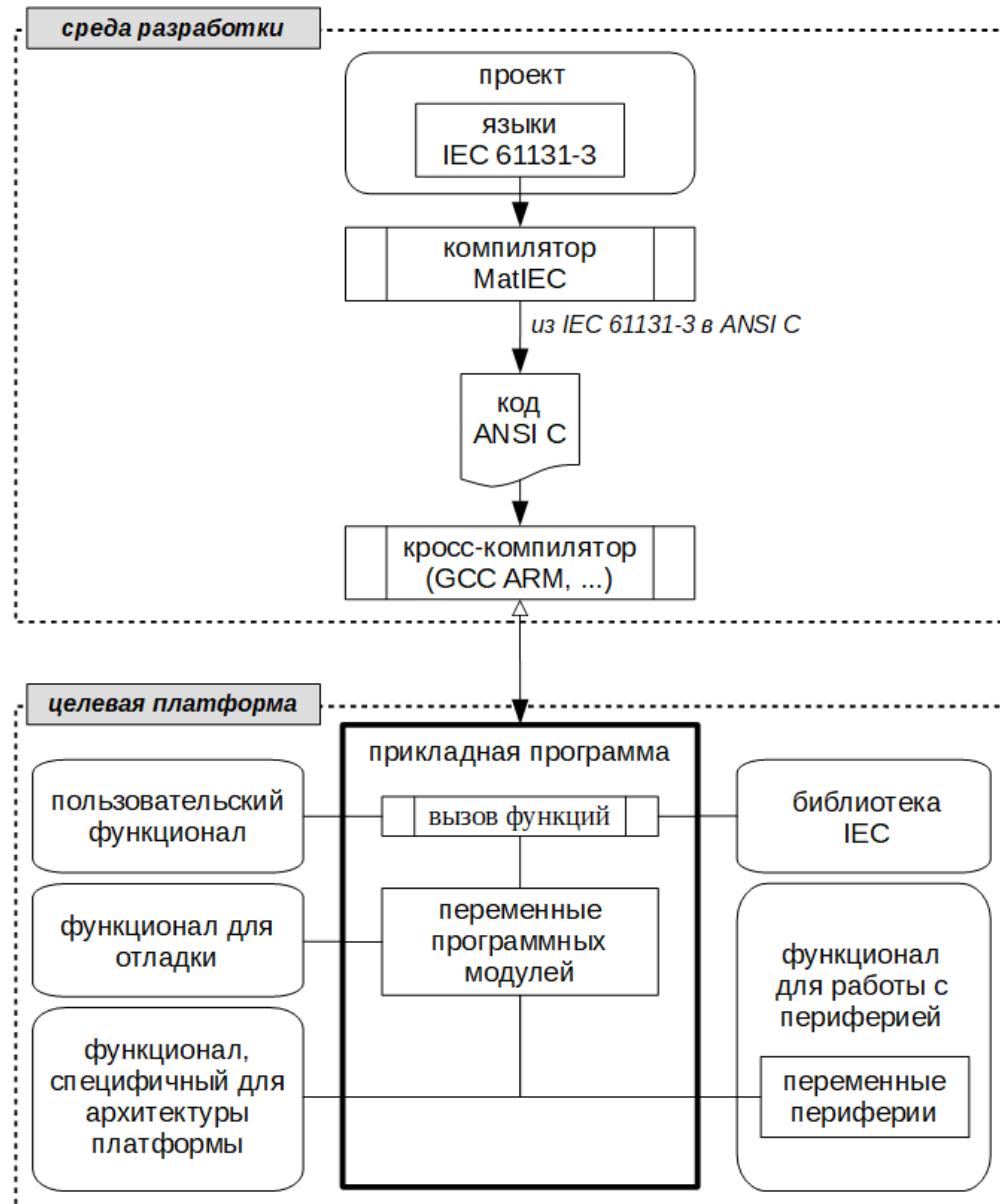


Рисунок 3.1.1 - Обобщенная схема работы

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 – содержит 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 Bergemiz 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 Bergemiz, может быть размещен на целевом устройстве различными коммуникационными средствами (например, загрузка через СОМ-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 высокуровневых 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 программного кода 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 также 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;
- дерево переменных;
- 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 5.1) содержит следующие пункты:

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

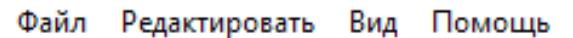


Рисунок 5.1 – Главное меню

Часть операций, выполняемых с помощью выбора определённого пункта меню мышью, могут быть выполнены 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 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 Veremiz. 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 5.2) и их функции приведены в таблице 1.



Рисунок 5.2 – Кнопки управления проектом

Таблица 1 - Функции кнопок управления проектом

| Кнопка | Функция |
|--------|----------------------|
| | Новый проект |
| | Открыть проект |
| | Сохранить проект |
| | Сохранить проект как |
| | Печать |
| | Отменить изменение |
| | Повторить изменение |
| | Вырезать выделение |

| Кнопка | Функция |
|--------|---------------------------|
| | 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 5.3) и их функции приведены в таблице 2.



Рисунок 5.3 – Кнопки сборки проекта и установления связи с целевым устройством

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

| Кнопка | Функция |
|--------|---|
| | Собрать проект <i>компиляция и компоновка проекта в директории сборки «build»</i> |
| | Очистить директорию сборки |
| | Соединиться с целевым устройством <i>по адресу URI, заданному в настройках проекта</i> |
| | 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 5.4) и отображает структуру проекта.

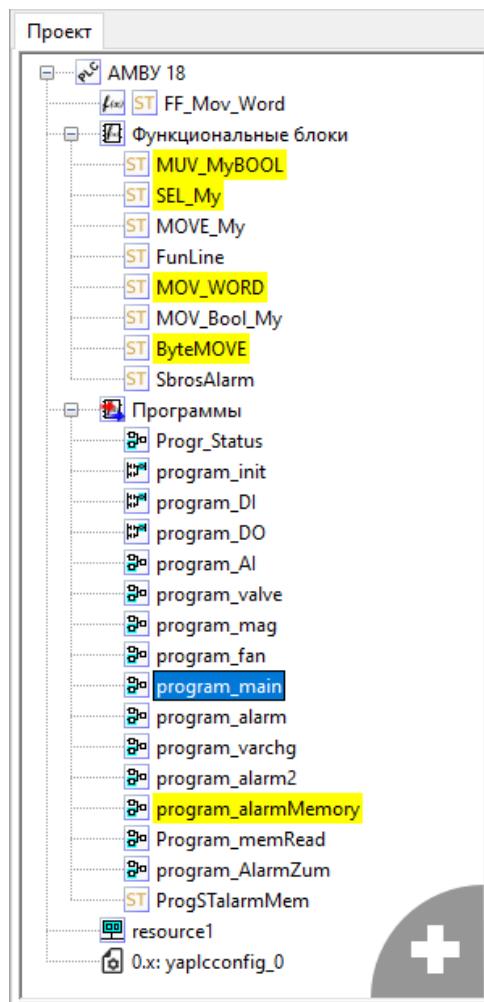


Рисунок 5.4 – Дерево проекта

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 доступны только 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 5.5).

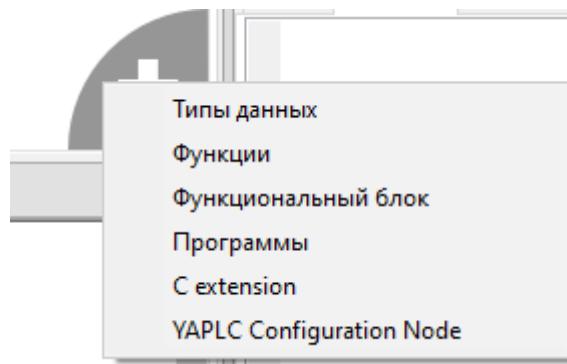


Рисунок 5.5 – Меню выбора добавляемого элемента в дерево проекта

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 5.6), где можно выбрать: «Добавить POU» или «Вставить POU» (если он был скопирован в буфер обмена).

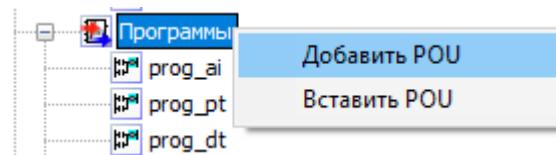


Рисунок 5.6 – Меню добавления программного компонента

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 5.7);
- 4) после выделения объекта нажать комбинацию клавиш Ctrl+C.

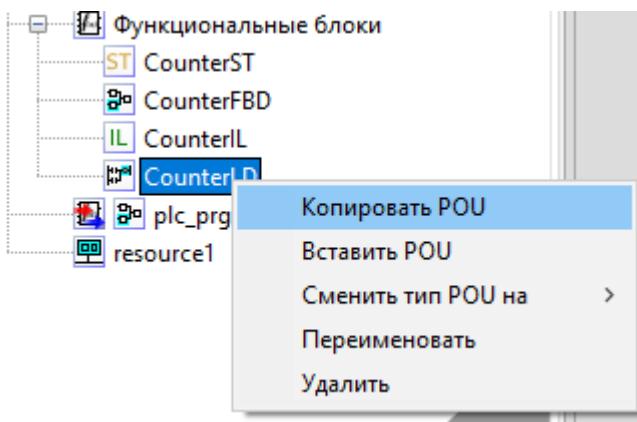


Рисунок 5.7 – Копирование элемента дерева проекта

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 5.8);
- 3) на группе элементов дерева проекта нажать комбинацию кнопок Ctrl+V.

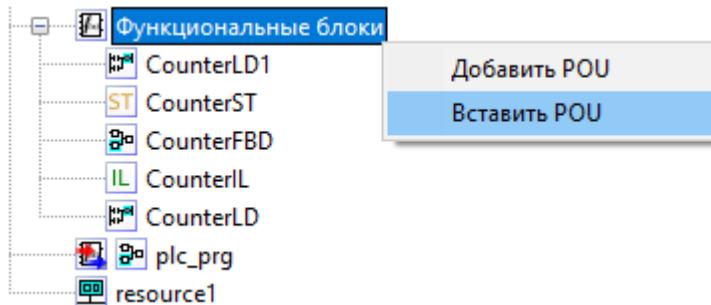


Рисунок 5.8 – Вставка скопированного элемента дерева проекта

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 5.9);
- 4) после выделения необходимого элемента нажать кнопку Del.

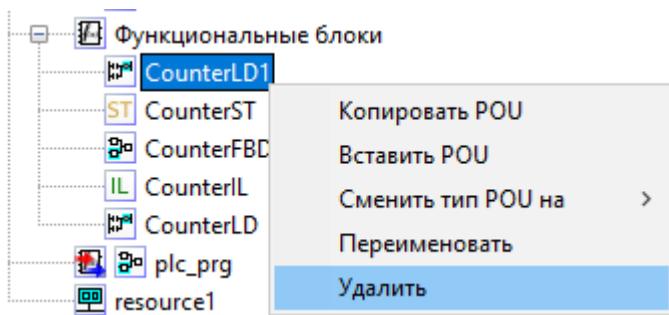


Рисунок 5.9 – Удаление элемента из дерева проекта

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 5.10).

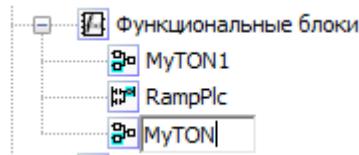


Рисунок 5.10 – Режим переименования элемента в дереве проекта

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 5.11).

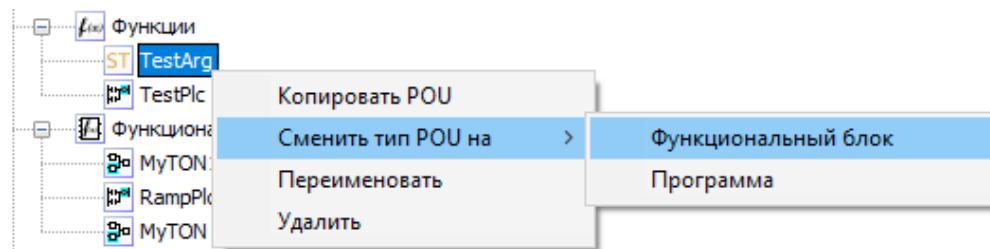


Рисунок 5.11 – Изменение типа POU элемента в дереве проекта

5.3.6 Неиспользуемые элементы в дереве проекта

Для функций, функциональных блоков, POU предусмотрено выделение цветом, если элемент не используется в программах или ресурсах.

Если элемент используется в проекте – подсветки нет, если не используется – подсвечивается желтым цветом.

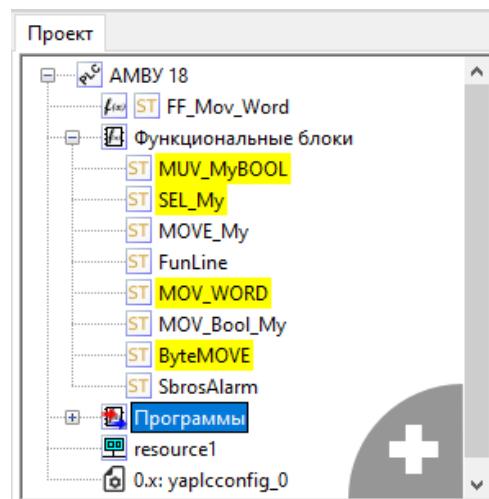


Рисунок 5.12 – Выделение неиспользуемых функциональных блоков проекта

5.4 Панель списка констант и переменных

Данная панель находится в верхней части редактора, и различается по назначению образом:

- для конфигурационных переменных (глобальных переменных),
- для внешних переменных программ и функциональных блоков,
- для локальных переменных,

- для переменных имеющих тип данных функционального блока.

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 Панель списка глобальных констант и переменных

Панель списка констант и переменных входит в состав редакторов:

- функциональные блоки (внешние глобальные переменные функциональных блоков),
- программы (внешние глобальные переменные программ)

Данная панель выглядит следующим образом:

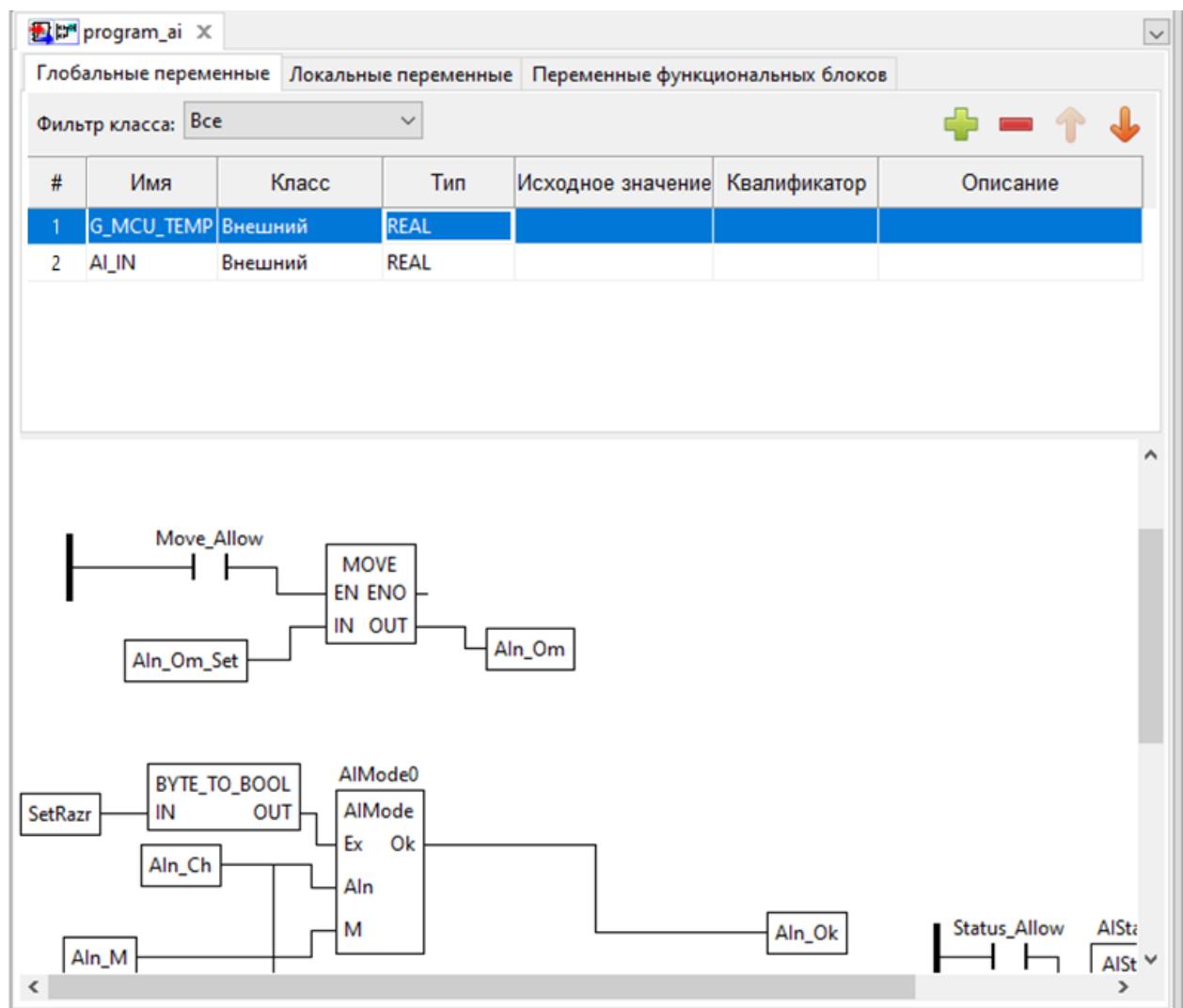


Рисунок 5.13 – Редактор глобальных переменных программ и функциональных блоков

5.4.2 Панель списка локальных констант и переменных

Панель списка констант и переменных входит в состав редакторов:

- функции,
- функциональные блоки (локальные переменные блока),
- программы (локальные переменные программы).

Данная панель выглядит следующим образом:

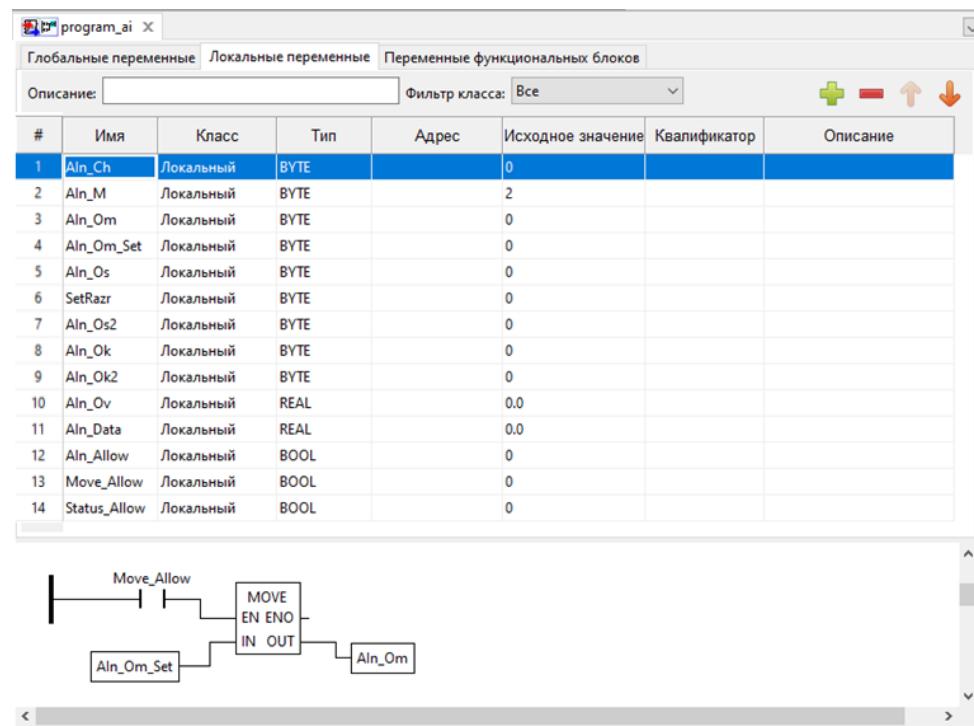


Рисунок 5.14 – Редактор локальных переменных программ и функциональных блоков

5.4.3 Панель списка переменных функциональных блоков

Панель списка констант и переменных входит в состав редакторов:

- функциональные блоки (локальные переменные блока),
- программы (локальные переменные программы)

Данная панель выглядит следующим образом:

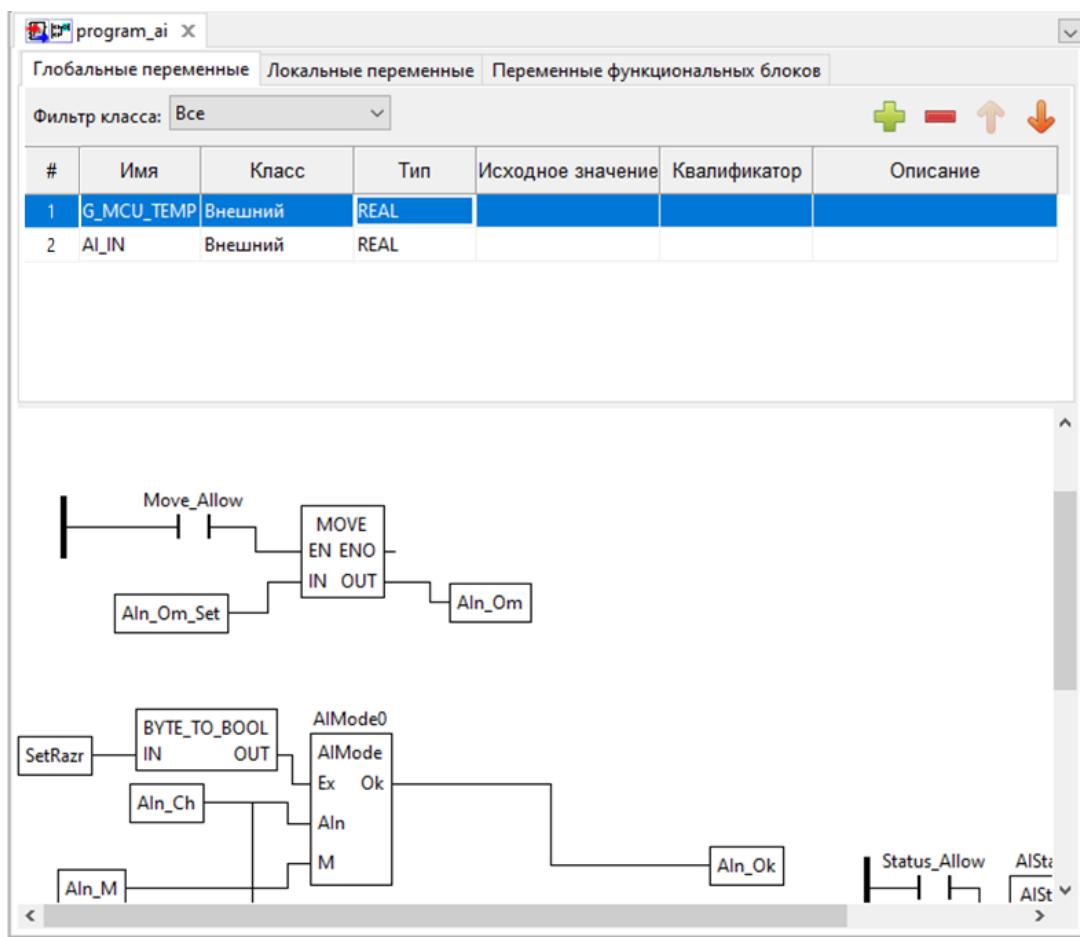


Рисунок 5.15 – Редактор переменных функциональных блоков программ и функциональных блоков.

5.4.4 Имя переменной

Первый символ имени должен 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 5.16).

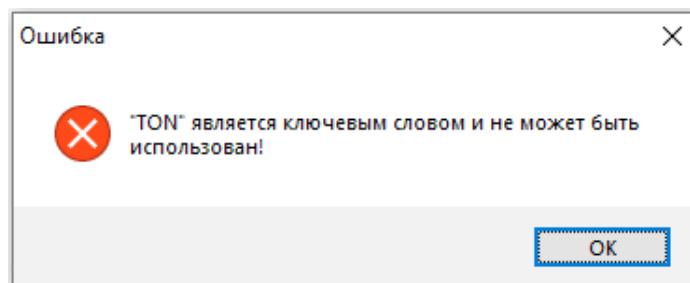


Рисунок 5.16 – Диалог-предупреждение о попытке использования зарезервированного имени «TON»

Переход в режим редактирования имени выполняется следующим образом - щелкнуть два раза левой клавишей мыши на поле с нужным именем переменной.

5.4.4.1 Имя внешней переменной

Имя внешней переменной выбирается из списка (рисунок 5.17), который отображается после двойного нажатия. Все переменные берутся из уже созданных конфигурационных переменных (рисунок 5.18).

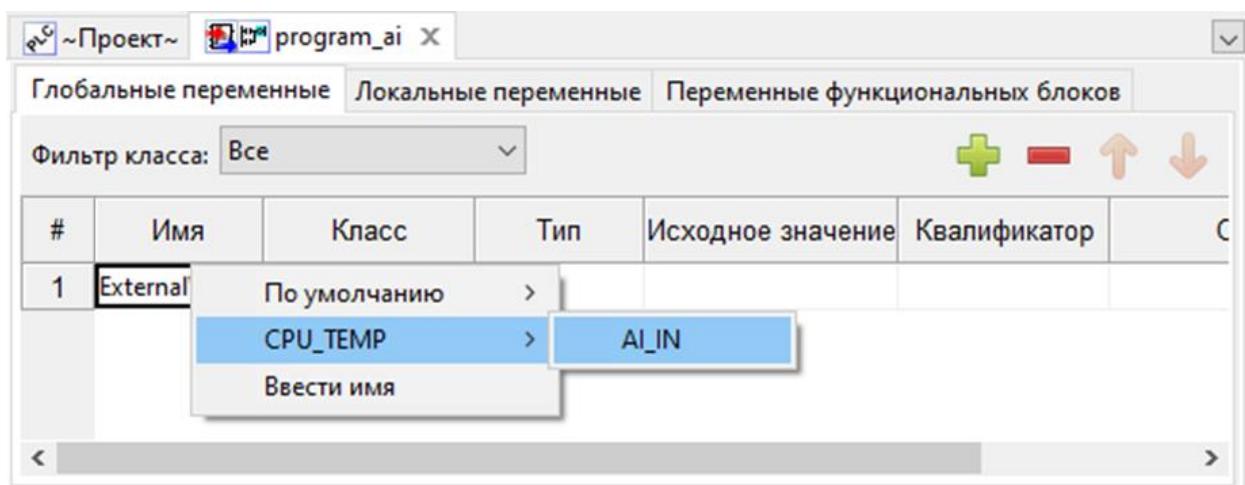


Рисунок 5.17 – Выбор глобальной переменной программы (внешней).

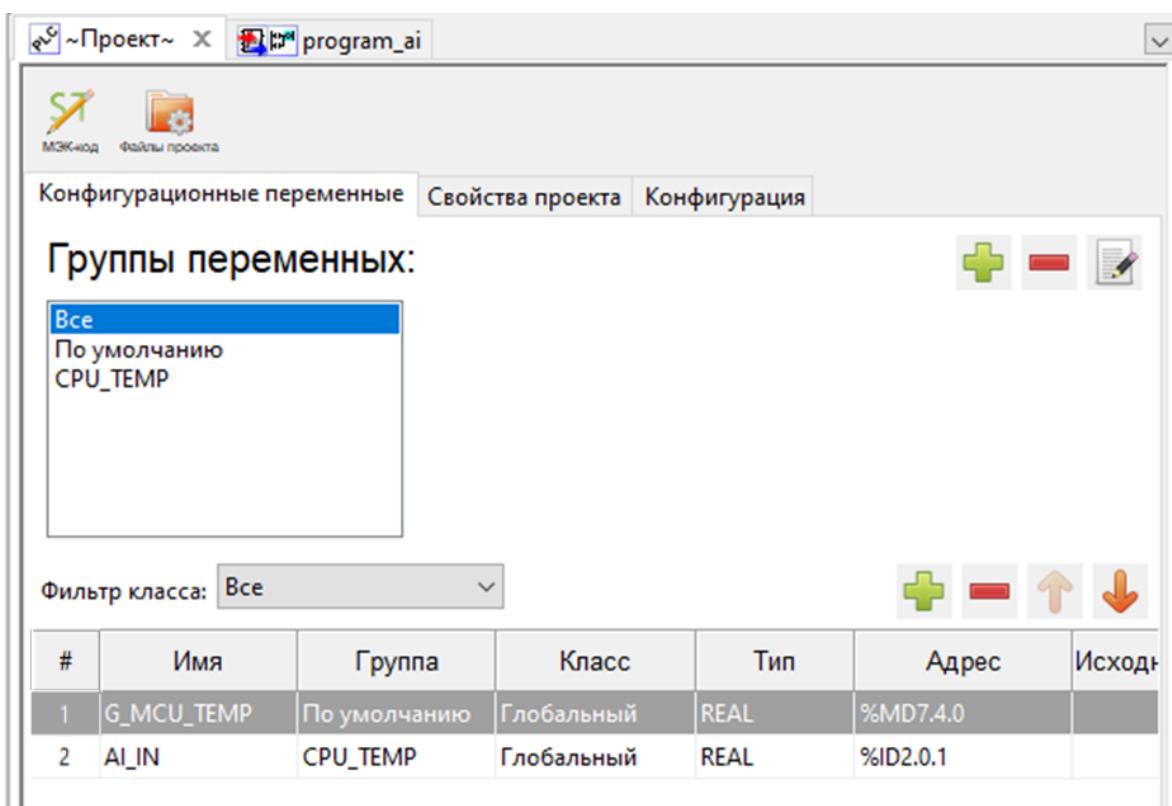


Рисунок 5.18 – Конфигурационные переменные доступные для выбора.

5.4.5 Класс переменной

Класс выбирается из списка (рисунок 5.19), который отображается после двойного нажатия левой клавишей мыши по данному полю.

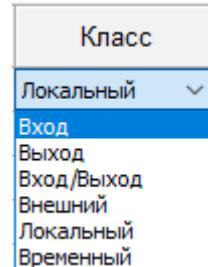


Рисунок 5.19 – Список выбора Класса

5.4.6 Тип переменной

Тип выбирается из списка (рисунок 5.20), который отображается после двойного нажатия.

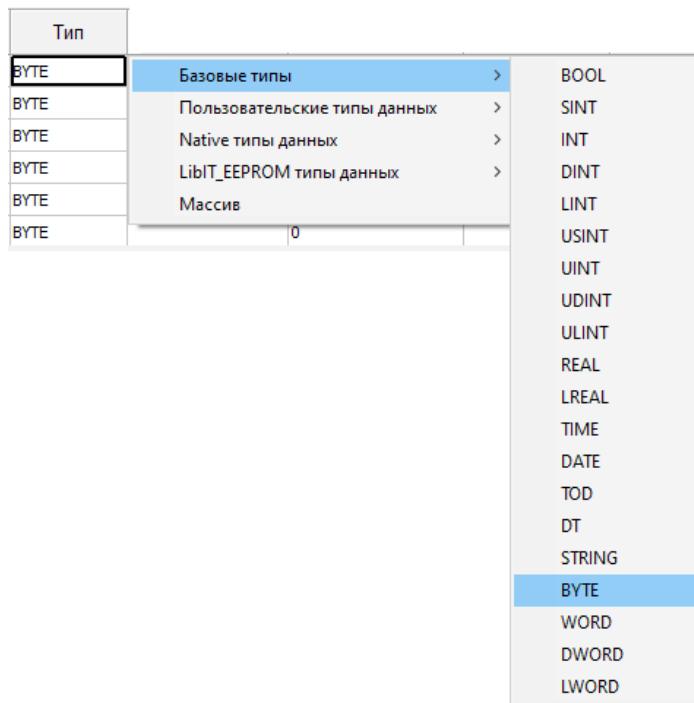


Рисунок 5.20 – Выбор базового Типа

5.4.7 Адрес переменной

Переменную или константу можно связать с регистром модуля УСО. Двойное нажатие на поле «Адрес» вызывает появление мигающего курсора и кнопки «...» (рисунок 5.21).

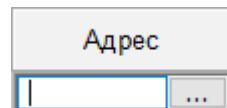


Рисунок 5.21 – Режим редактирования Адреса

Нажатие на кнопку «...» приводит к появлению диалога «Просмотр адресов» (рисунок 5.22) - списка регистров модулей УСО, которые могут быть связаны с переменной 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 в позиции мигающего курсора.

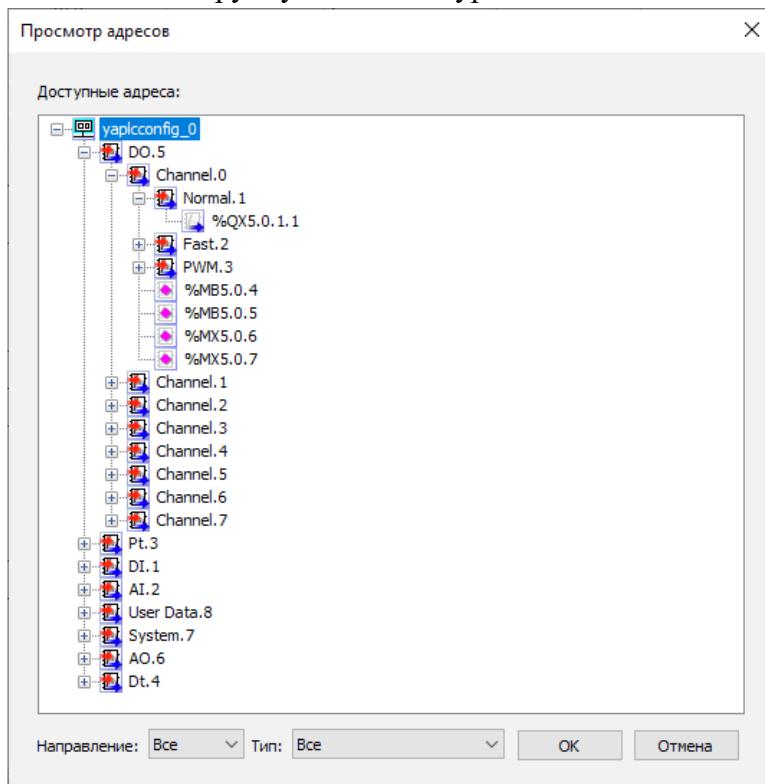


Рисунок 5.22 – Диалог просмотра адресов регистров УСО

5.4.8 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.

5.4.9 Фильтрация списка переменных

Панель переменных и констант предоставляет возможность фильтровать отображаемые переменные по классам - выполняется с помощью функции «Фильтр по классам» (рисунок 5.23).

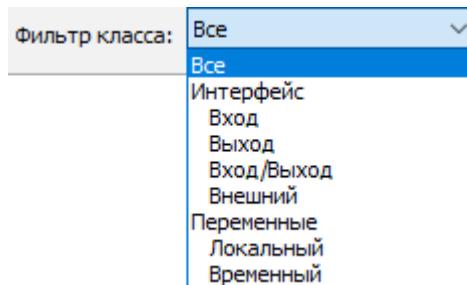


Рисунок 5.23 – Фильтр переменных по классу

5.4.10 Копирование и вставка переменных

Копирование/вставка переменных из списка переменных осуществляется путем выбора строки или нескольких строк (см рисунок 5.24). Копировать/вставлять переменную можно 2-я способами:

- Способ 1. Нажать правой кнопкой на поле таблицы. В появившемся списке выбрать «Копировать»/«Вставить».
- Способ 2. С помощью комбинации клавиш. Для копирования «Ctrl+c» для вставки «Ctrl+v»

| # | Имя | Группа | Класс | Тип | Адрес | Исходное значение | Квалификатор | Описание |
|---|-------------|--------------|------------|------|----------|-------------------|--------------|----------|
| 1 | G MCU TEMP | По умолчанию | Глобальный | REAL | | | | |
| 2 | G MCU TEMPO | По умолчанию | Глобальный | REAL | | | | |
| 3 | G MCU TEMP1 | Копировать | Глобальный | REAL | | | | |
| 4 | AI IN | Вставить | Глобальный | REAL | %ID2.0.1 | | | |
| | | Удалить | | | | | | |

Рисунок 5.24 – Копирование переменной.

Копирование возможно только для переменных одного класса. Исключением являются переменные, имеющие внешний класс. Их можно скопировать во внешние и глобальные переменные в глобальные переменные. При попытке копирования данных не относящихся к переменным выдается предупреждение (рисунок 5.25).

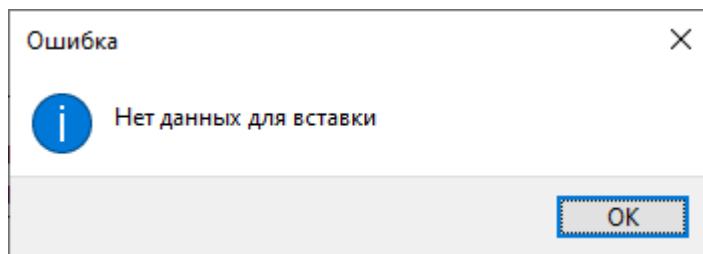


Рисунок 5.25 – Отсутствие данных при копировании.

5.4.11 Сортировка переменных

Сортировка переменных в списке переменных происходит при двойном клике по заголовку столбца (Рисунок 5.26, Рисунок 5.27).

| # | Имя | Класс | Тип | Адрес | Исходное значение | Квалификатор | Описание |
|----|---------|-----------|------|-------|-------------------|--------------|----------|
| 1 | PTn_Ch | Локальный | BYTE | | 0 | | |
| 2 | Ex | Локальный | BOOL | | 0 | | |
| 3 | PTn_M | Локальный | BYTE | | 2 | | |
| 4 | PTn_STy | Локальный | BYTE | | 0 | | |
| 5 | PTn_Om | Локальный | BYTE | | 0 | | |
| 6 | PTn_Os | Локальный | BYTE | | 0 | | |
| 7 | PTn_Os2 | Локальный | BYTE | | 0 | | |
| 8 | PTn_Ok | Локальный | BYTE | | 0 | | |
| 9 | PTn_Ok2 | Локальный | BYTE | | 0 | | |
| 10 | PTn_Ov | Локальный | REAL | | 0.0 | | |

Рисунок 5.26 – Несортированные элементы списка переменных.

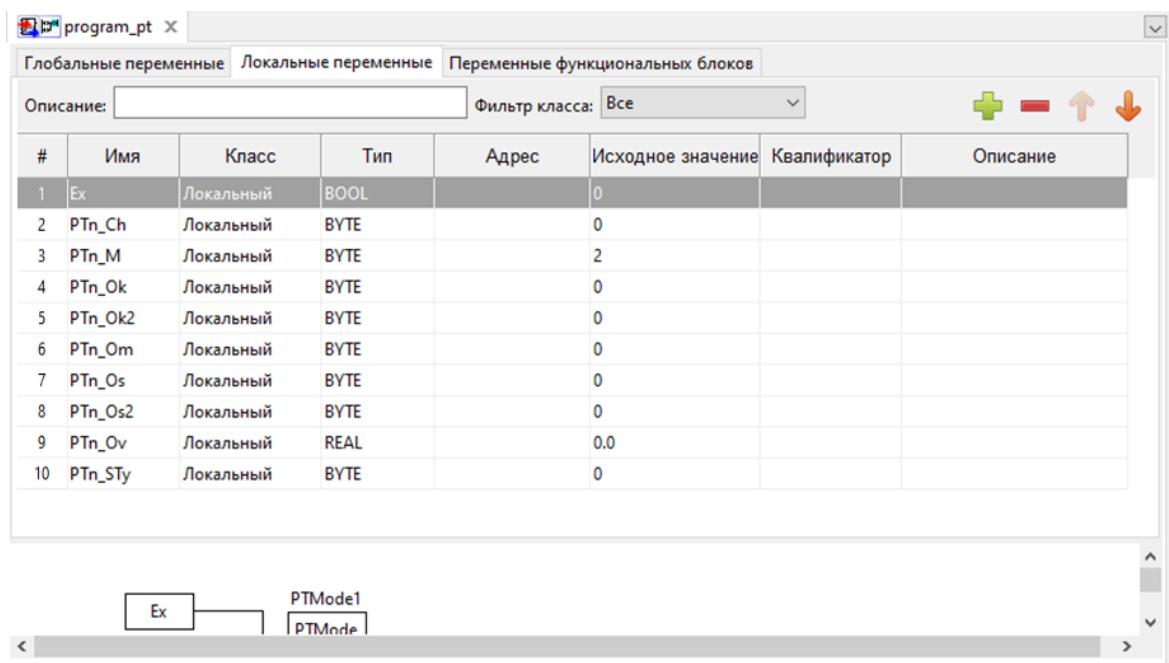


Рисунок 5.27 – Список переменных отсортированный по имени.

5.4.12 Добавление и удаление переменных

Добавление, удаление, а также перемещение переменных происходит с помощью специальных кнопок на панели переменных и констант (рисунок 5.28). Описание функций этих кнопок приведено в таблице 3.



Рисунок 5.28 – Кнопки работы с переменными

Таблица 3 - Функции кнопок работы с переменными

| Кнопка | Функция |
|--------|---|
| | Добавить переменную <i>новая переменная добавляется со значениями по умолчанию</i> |
| | 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 - в дереве проекта дважды щелкнуть левой клавишей мыши на корневом элементе с именем проекта.

Данная панель состоит из следующих компонентов (рисунок 5.29):

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

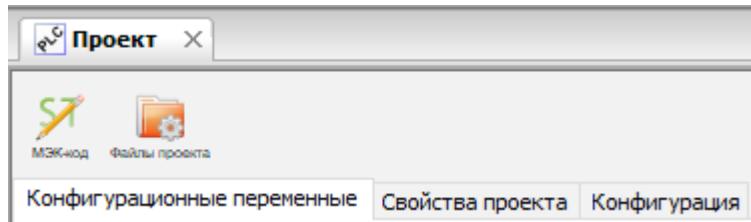


Рисунок 5.29 – Компоненты панели настройки проекта

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 5.5.2 и 5.5.3:

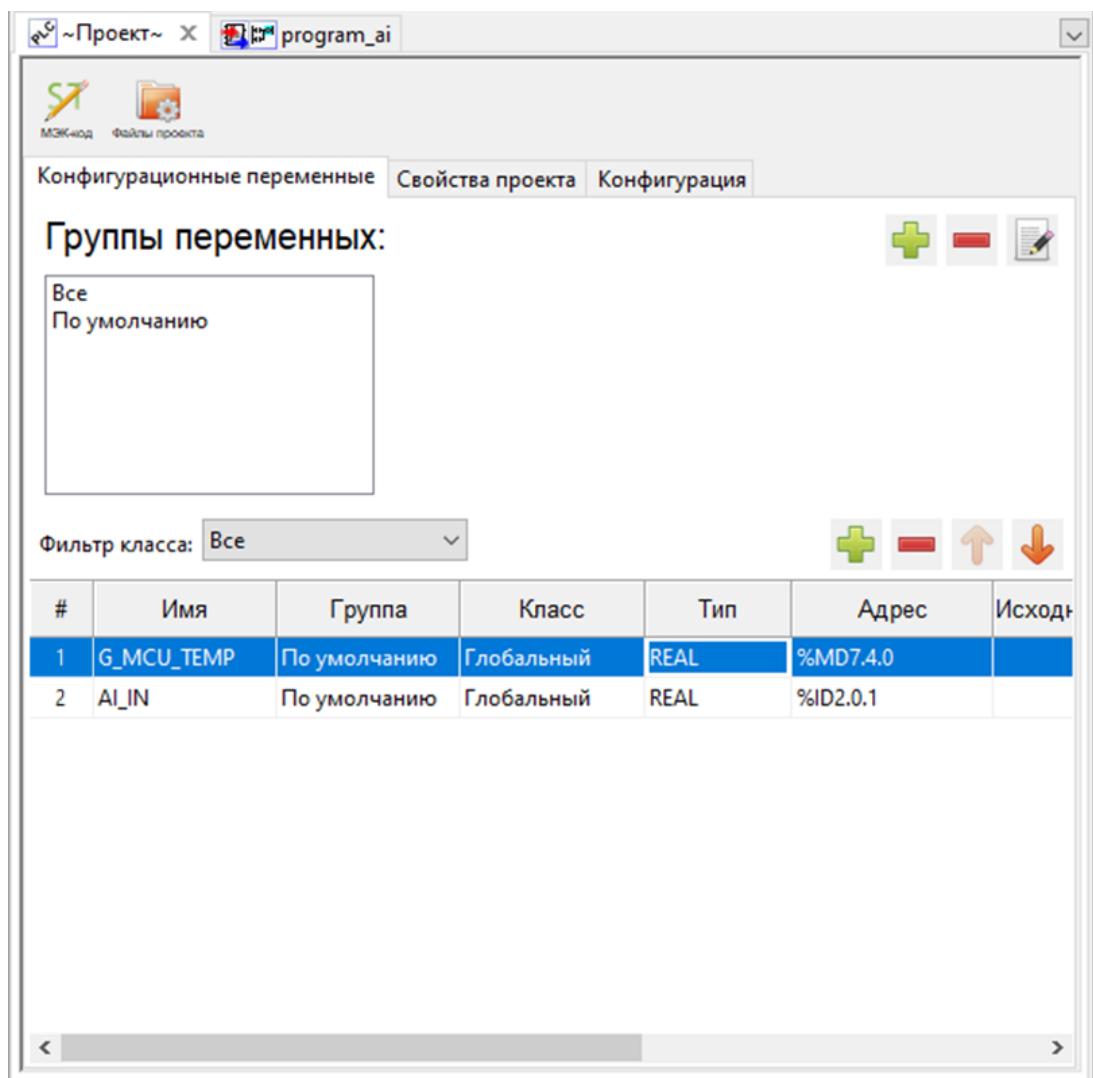


Рисунок 5.30 – Пример конфигурационных (глобальных) переменных

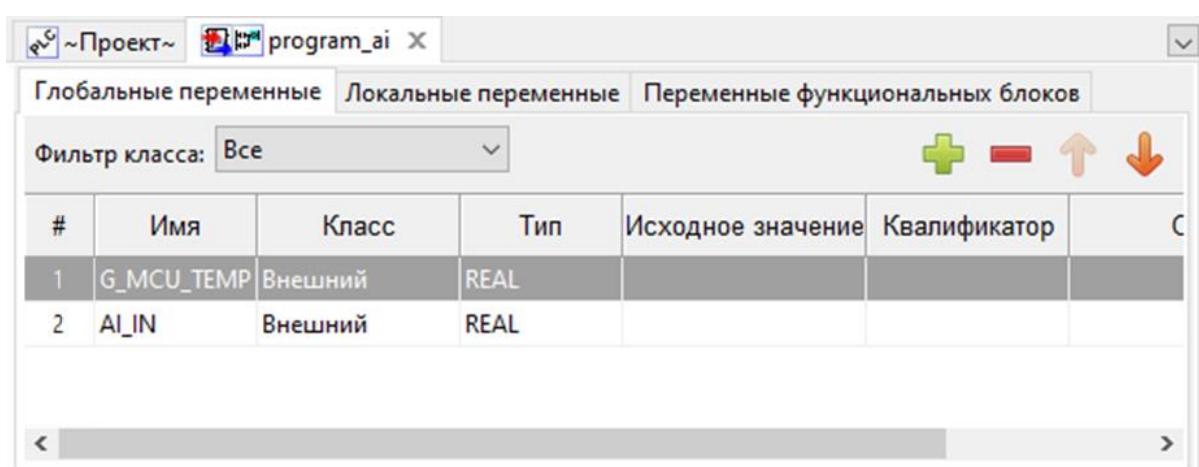


Рисунок 5.31 – Пример использования глобальных переменных из программы «prog_main»

5.5.2 Группа переменной

Группа выбирается из списка (рисунок 5.32), который задается в панели конфигурационных переменных (рисунок 5.33). При двойном клике по группе в списке переменных отображаются те переменные, которые принадлежат к выбранной группе.

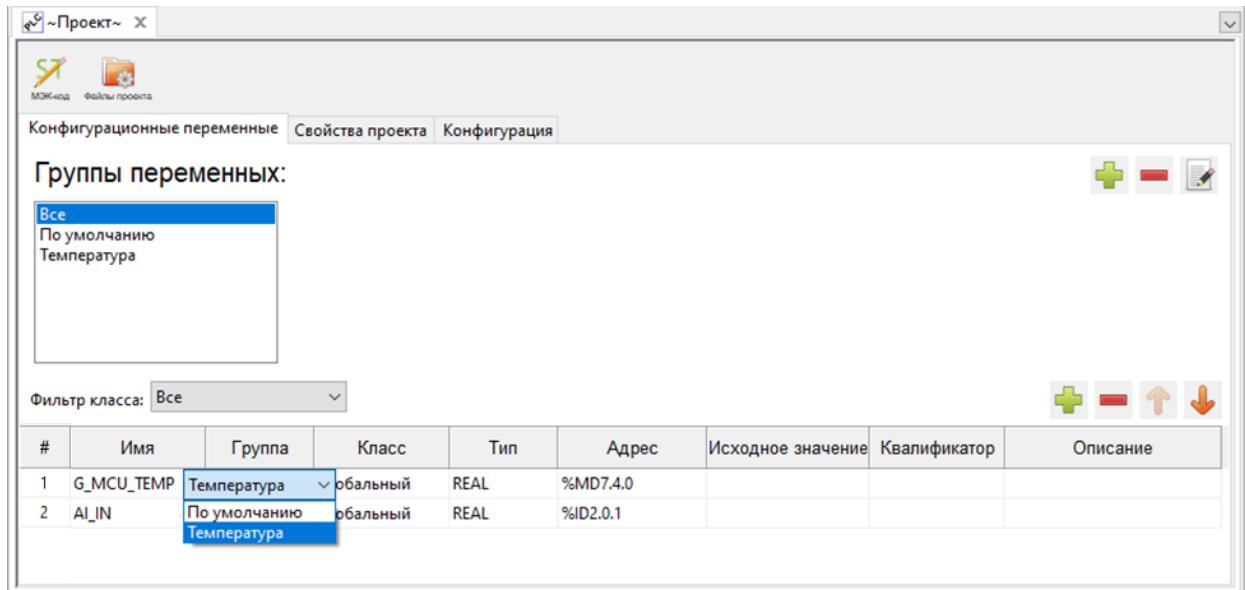


Рисунок 5.32 – Выбор группы для переменной.

Группы переменных:

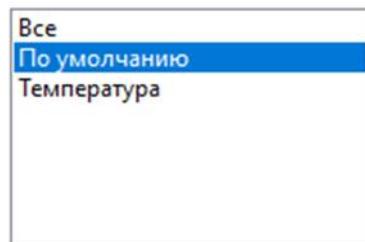


Рисунок 5.33 – Окно выбора группы.

5.5.3 Добавление и удаление групп переменных

Добавление, удаление, а также перемещение переменных происходит с помощью специальных кнопок представленных на рисунке ниже (рисунок 5.34). Назначение кнопок указано в таблице 4.

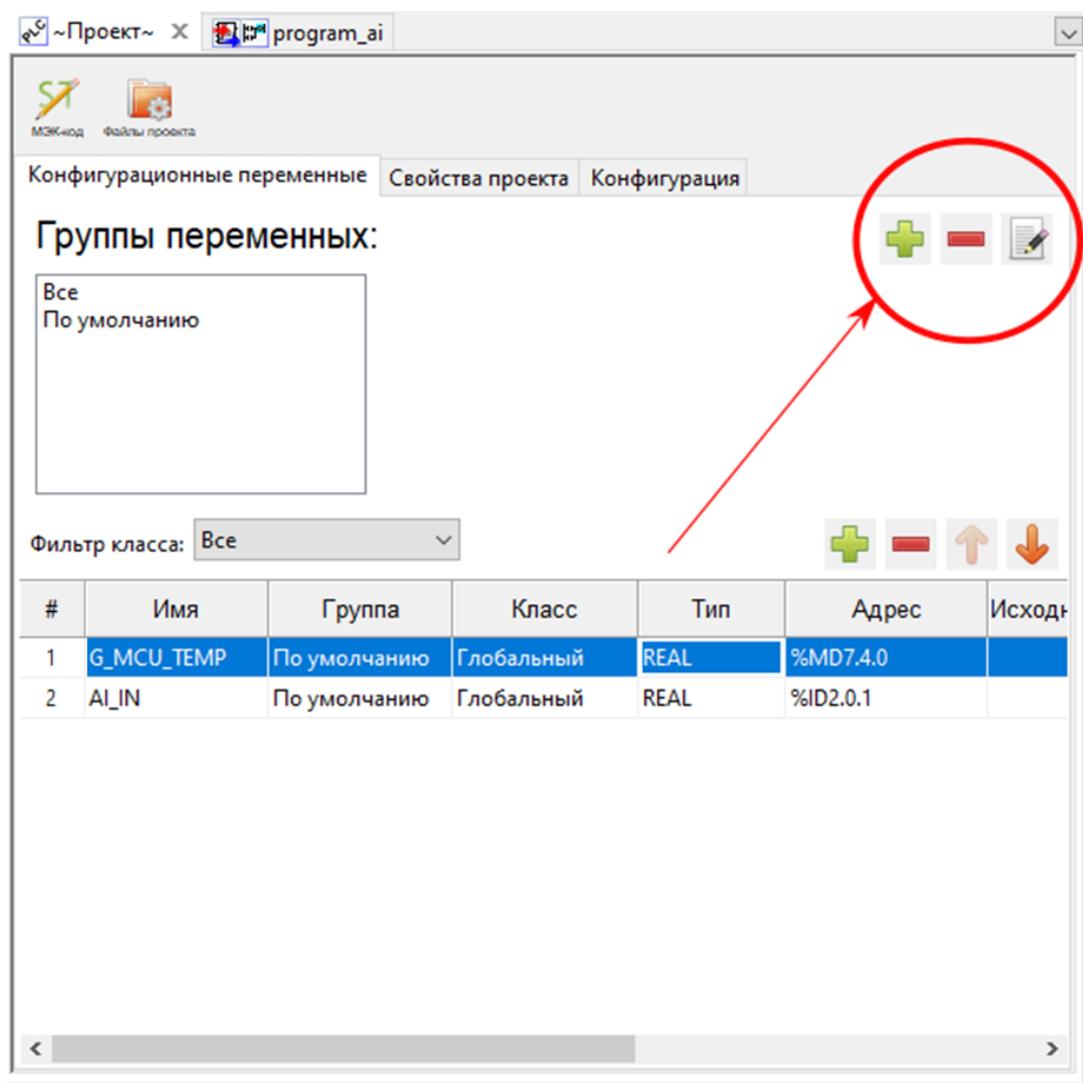


Рисунок 5.34 – Кнопки управления группами переменных

Таблица 4

| Кнопка | Функция |
|--------|--|
| | Добавить группу переменных (рисунок 5.35) |
| | Удалить выбранную группу переменных |
| | Переименовать группу переменных (рисунок 5.36) |

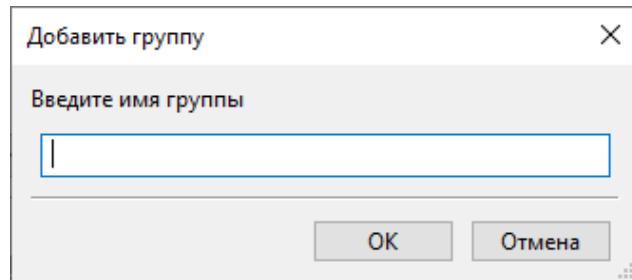


Рисунок 5.35 – Добавление группы

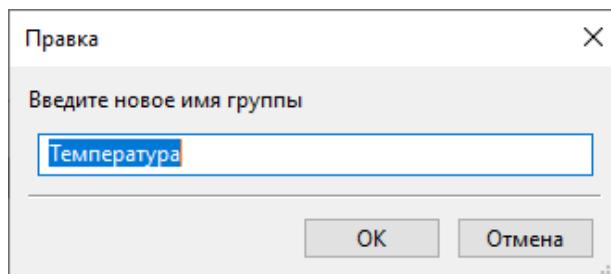


Рисунок 5.36 – Переименование группы

5.5.4 Свойства проекта

Данная панель содержит набор сгруппированных полей с информацией о проекте (рисунок 5.37). Поля сгруппированы по вкладкам.

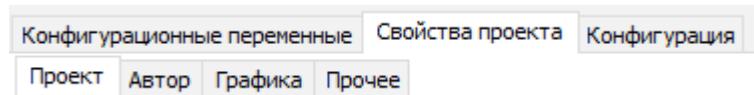


Рисунок 5.37 – Вкладки панели «Свойства проекта»

Вкладка «Проект» позволяет задать общую информацию о проекте (рисунок 5.38):

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

| | |
|--------------------------------|---------------|
| Имя проекта (обязательно): | Test |
| Версия проекта (опционально): | 1 |
| Имя продукта (обязательно): | PID regulator |
| Версия продукта (обязательно): | 2 |
| Релиз продукта (опционально): | 3 |

Рисунок 5.38 – Вкладка «Проект»

Вкладка «Автор» позволяет задать информацию об авторе проекта (рисунок 5.39):

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

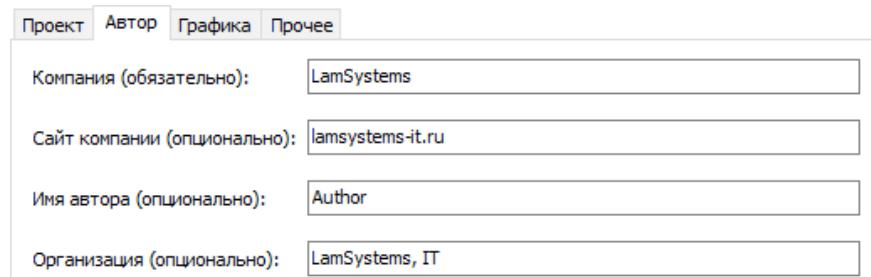


Рисунок 5.39 – Вкладка «Автор»

Вкладка «Графика» позволяет задать размеры страницы, шаг сетки для редакторов диаграмм графических элементов (рисунок 5.40).

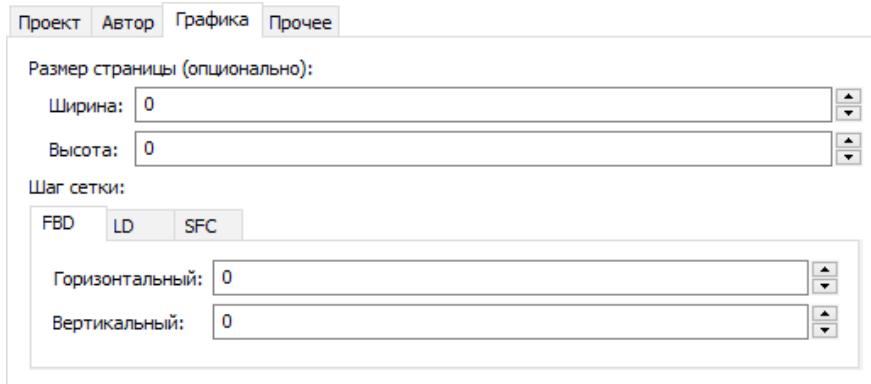


Рисунок 5.40 – Вкладка «Графика»

Вкладка «Прочее» позволяет задать язык интерфейса для среды разработки Beremiz и указать дополнительное текстовое описание проекта (рисунок 5.41).

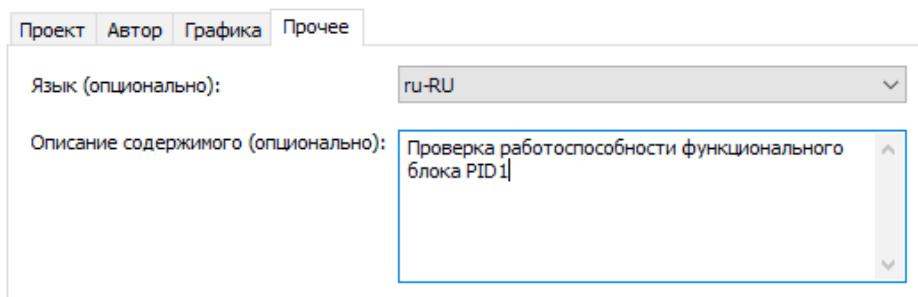


Рисунок 5.41 – Вкладка «Прочее»

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.

5.5.5 Конфигурация

Данная панель содержит набор элементов (рисунок 5.42):

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

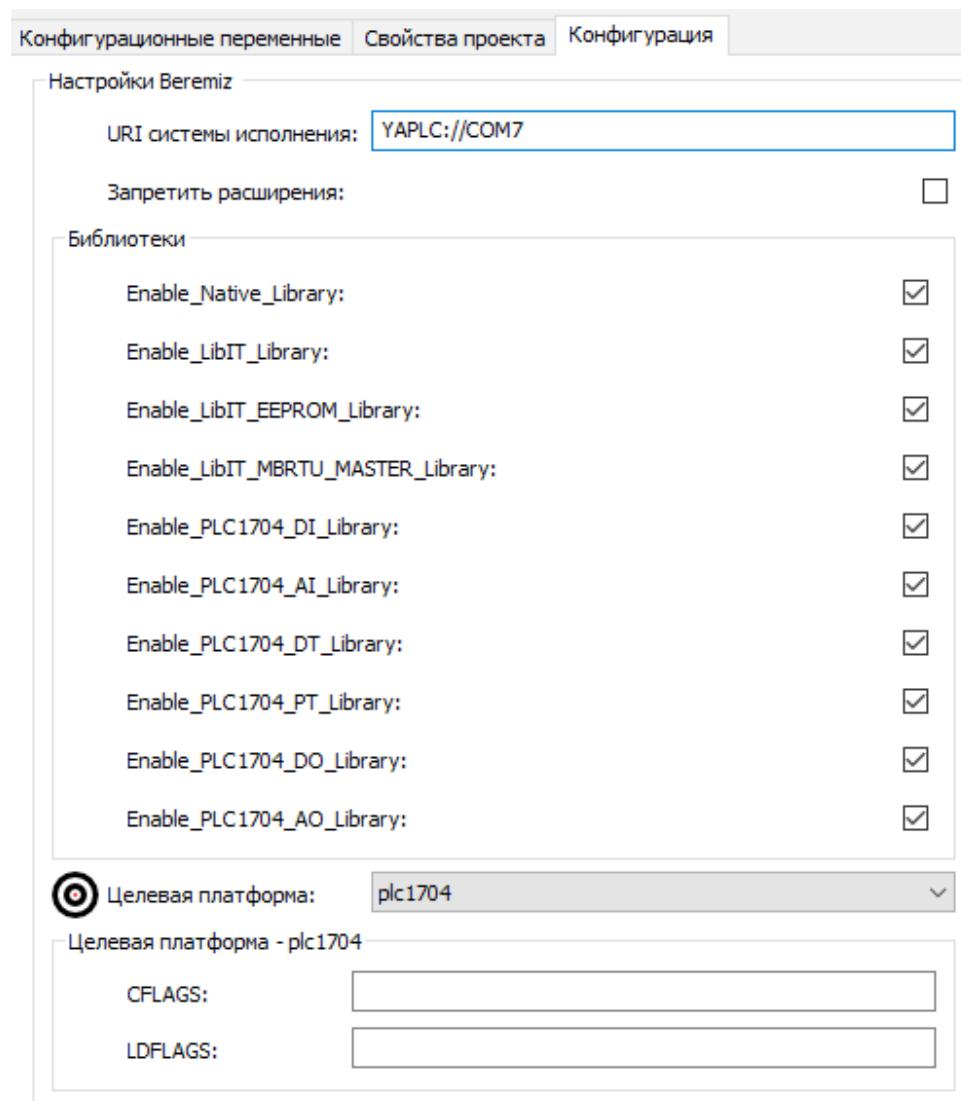


Рисунок 5.42 – Панель конфигурации проекта

URI системы исполнения – строка, описывающая подключение к целевому устройству для режима отладки (см. п. [7.2.2. Настройки сборки проекта и соединения с целевым устройством](#)).

Библиотеки - подключаемые дополнительные библиотеки.

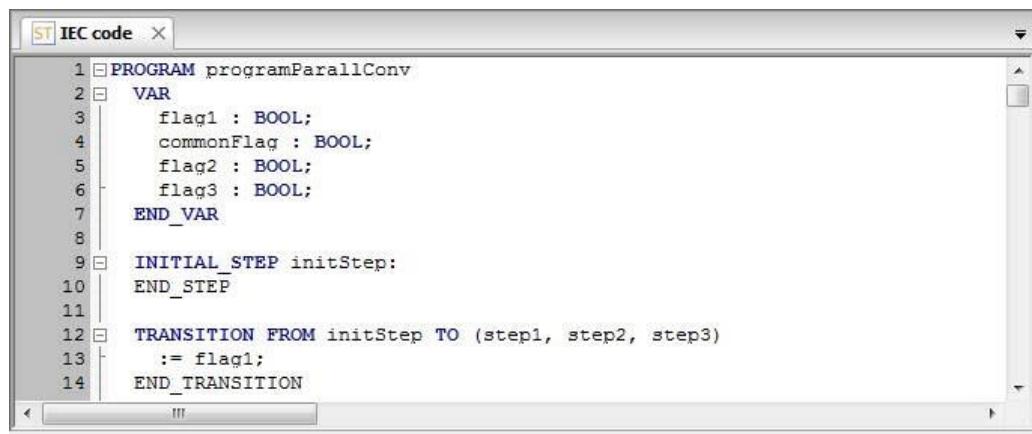
Целевая платформа – выбор целевой платформы (из списка).

CFLAGS – дополнительные флаги Си-компилятора.

LDFLAGS – дополнительные флаги компоновщика.

5.5.6 МЭК-код

При нажатии на кнопку «МЭК-код» отображается панель с текстовым редактором (рисунок 5.43), отображающим промежуточный МЭК-код на языке ST, доступный только для чтения, без возможности редактирования.



```
ST IEC code X
1 PROGRAM programParallelConv
2   VAR
3     flag1 : BOOL;
4     commonFlag : BOOL;
5     flag2 : BOOL;
6     flag3 : BOOL;
7   END_VAR
8
9   INITIAL_STEP initStep:
10  END_STEP
11
12  TRANSITION FROM initStep TO (step1, step2, step3)
13    := flag1;
14  END_TRANSITION
```

Рисунок 5.43 – Панель промежуточного МЭК-кода

5.5.7 Файлы проекта

Панель файлов проекта (рисунок 5.44) содержит встроенный проводник файлов, с помощью которого можно: добавлять дополнительные файлы в проект, удалять дополнительные файлы из проекта.

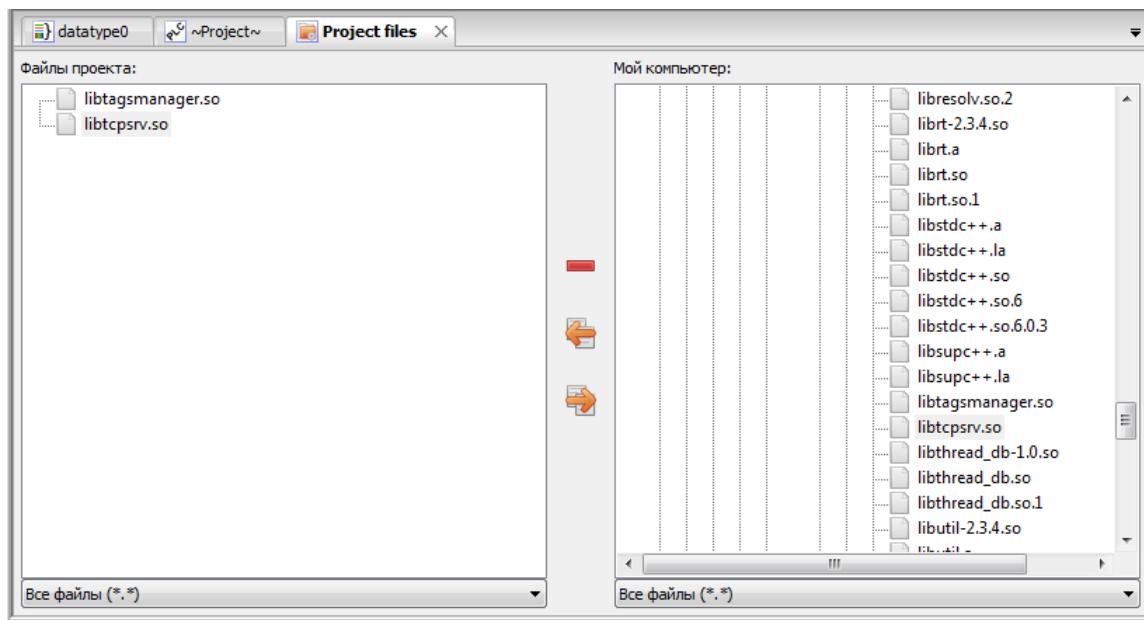


Рисунок 5.44 – Встроенный проводник

Все манипуляции с файлами осуществляются с помощью кнопок, расположенных в середине данной панели. Их описание приведено в таблице 5.

Таблица 5 - 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 5.45) позволяют создавать и редактировать алгоритмы и логику выполнения программных модулей на языках ST и IL.

The screenshot shows a software interface for programming logic. At the top, there's a toolbar with icons for file operations. Below it is a table with columns: #, Имя (Name), Класс (Class), Тип (Type), Исходное значение (Initial value), Квалификатор (Qualifier), and Описание (Description). The table contains five rows with variable definitions. Below the table is a code editor window displaying the following ST (Structured Text) code:

```

1 IF Ex = TRUE THEN
2   ScaleA:= (Ka * V) + Kb;
3 ELSE
4   ScaleA:= Y;
5 END_IF;
6

```

Рисунок 5.45 – 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 шрифта 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 5.46):



Рисунок 5.46 – Панель редактора FBD диаграмм

С помощью данной панели можно добавить все элементы языка FBD. Функциональное назначение каждой кнопки описано в таблице 6.

Таблица 6 - 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 5.47).

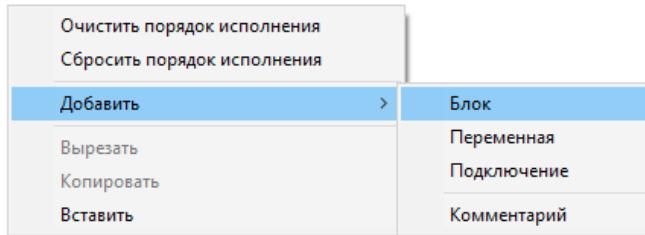


Рисунок 5.47 – Контекстное меню редактора FBD

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 5.48).

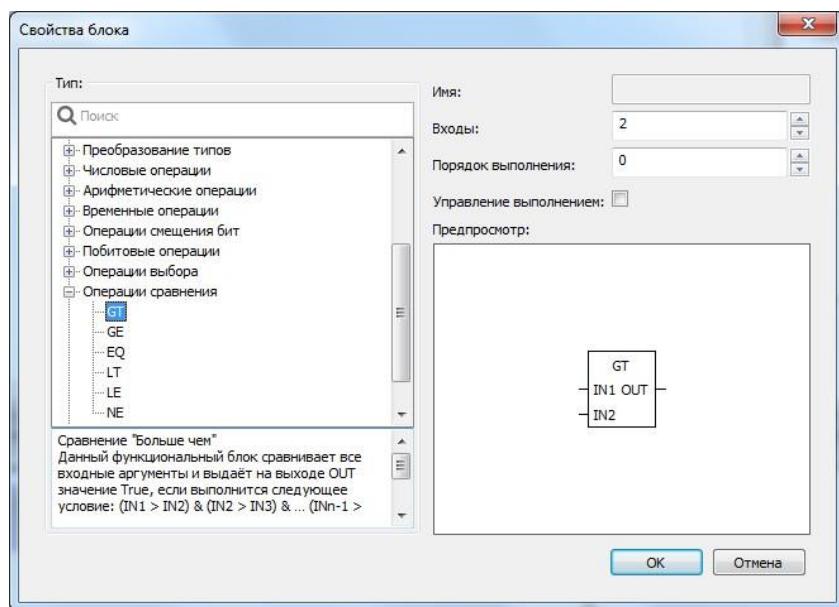


Рисунок 5.48 – Свойства функционального блока

В данном диалоге приведено краткое описание функционального блока и представлена 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

Для добавления переменной из панели переменных и констант с помощью перетаскивания (Drag&Drop) необходимо зажать левой кнопкой мыши за первый столбец (который имеет заголовок #) переменную, удовлетворяющую выше критериям в область редактирования диаграмм (рисунок 5.49).

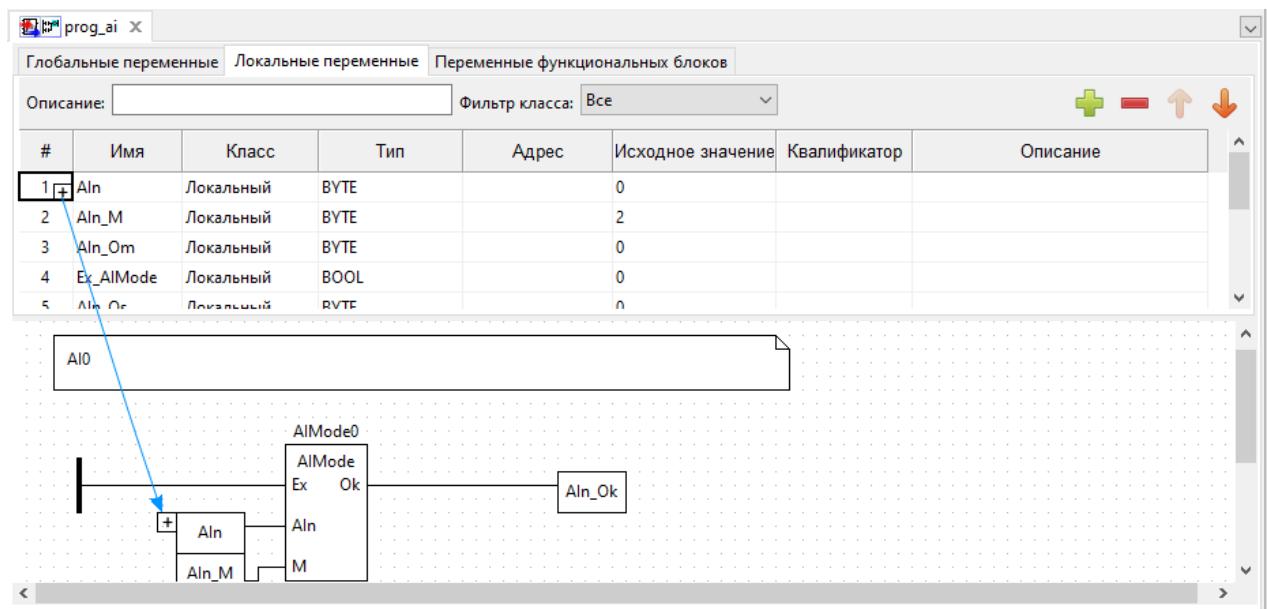


Рисунок 5.49 – Добавление переменной из панели переменных и констант

Изменить параметры переменной можно в диалоге «Свойства переменной» (рисунок 5.50), нажав на неё два раза левой клавишей мыши.

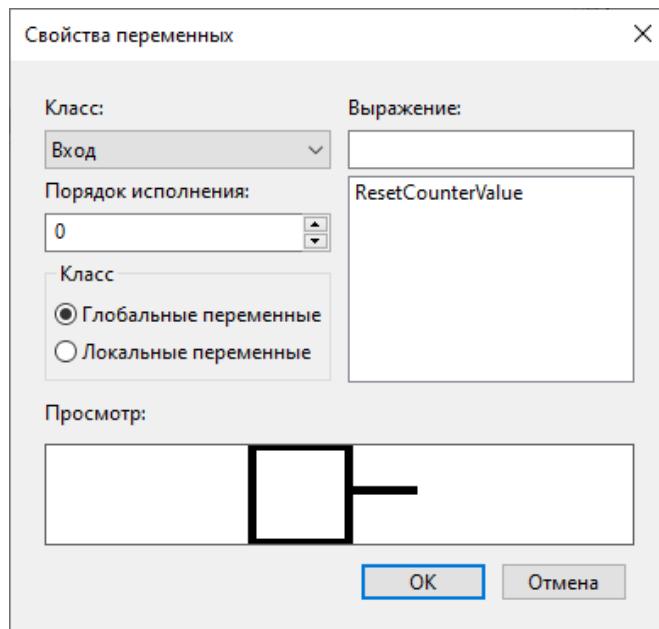


Рисунок 5.50 – Свойства переменной

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

Другой способ добавления переменной в область диаграммы – нажать кнопку 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 приводят к тому, что схема становится менее понятной.

После выбора добавления элемента «Соединение» появится диалог «Свойства соединения» (рисунок 5.51).

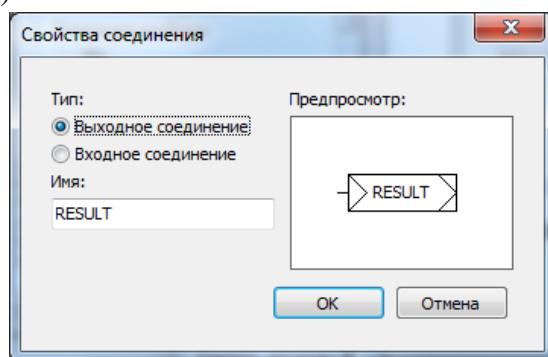


Рисунок 5.51 – Диалог добавления соединения для FBD

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. Использование соединений представлено ниже (рисунок 5.52).

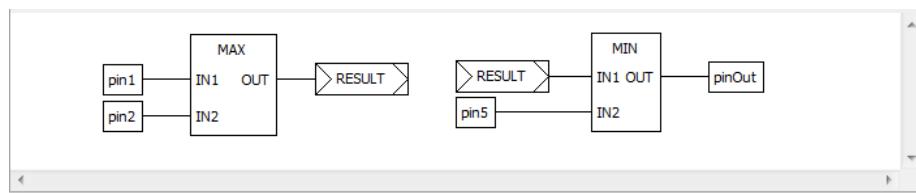


Рисунок 5.52 – Пример FBD диаграммы с использованием соединений

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 5.53) для ввода текста комментария.

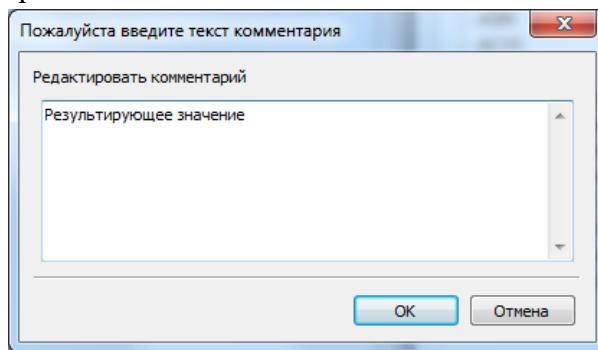


Рисунок 5.53 – Диалог добавления комментария

После нажатия кнопки OK комментарий появится на диаграмме (рисунок 5.54).

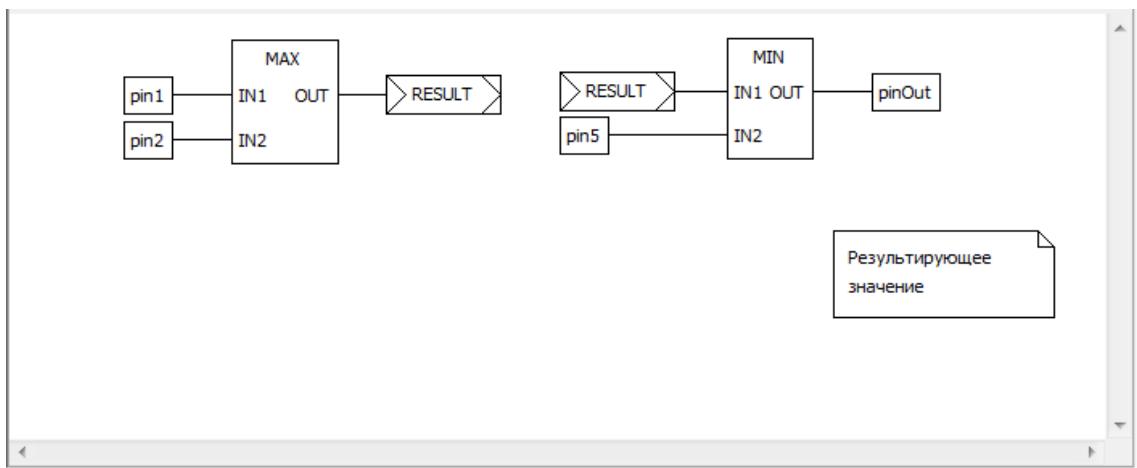


Рисунок 5.54 – Добавленный комментарий к FBD диаграмме

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.

В примере, представленном ниже (рисунок 5.55) порядок выполнения функций следующий: 1 – SUB; 2 – MUL; 3 – ADD.

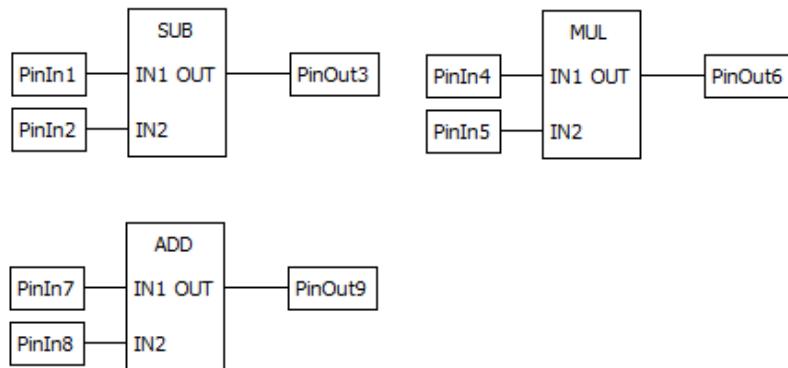


Рисунок 5.55 – Схема, содержащая функции с порядком выполнения (обсчета) по расположению

Порядок выполнения может быть изменён вручную с помощью диалога свойств. Данная опция «Порядок выполнения» выделена красным цветом (рисунок 5.56).

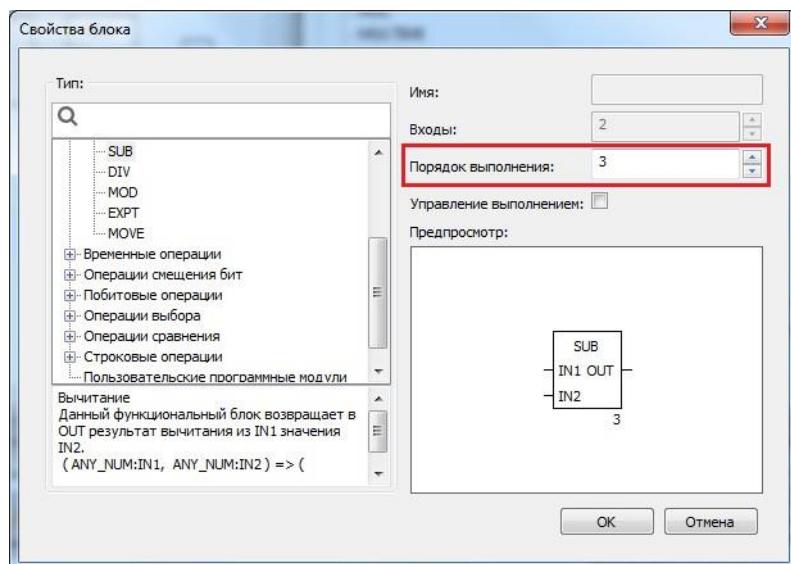


Рисунок 5.56 – Свойство порядок выполнения функции или функционального блока

После задания порядка выполнения для каждой функции или функционального блока на схеме в правом нижнем углу будет указан его порядковый номер выполнения (рисунок 5.57).

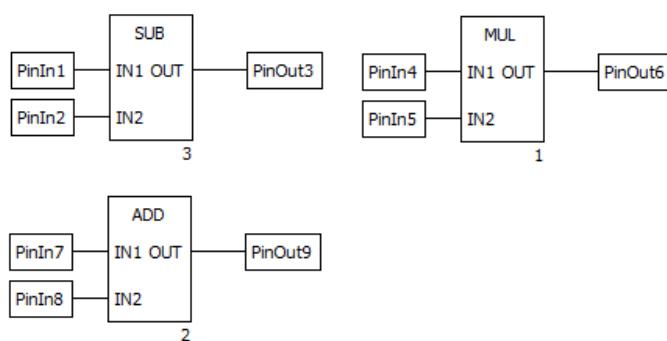


Рисунок 5.57 – Схема, содержащая функции с порядком выполнения заданным вручную

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 5.58) с элементами языка LD.

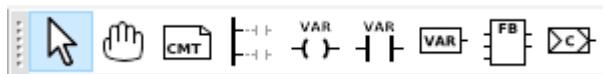


Рисунок 5.58 – Панель редактирования LD диаграмм

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 7 приведено описание кнопок данной панели.

Таблица 7 - 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 5.59).

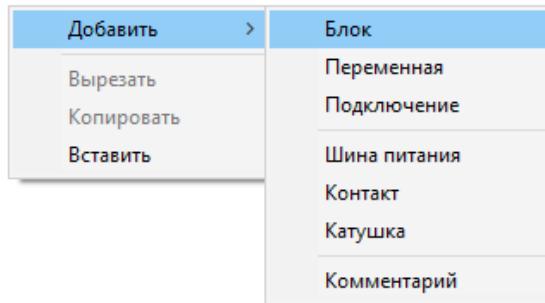


Рисунок 5.59 – Контекстное меню редактора LD

5.7.2.1 Добавление шины питания

При добавлении шины питания, одним из описанных выше способов, появится диалог «Свойства шины питания» (рисунок 5.60).

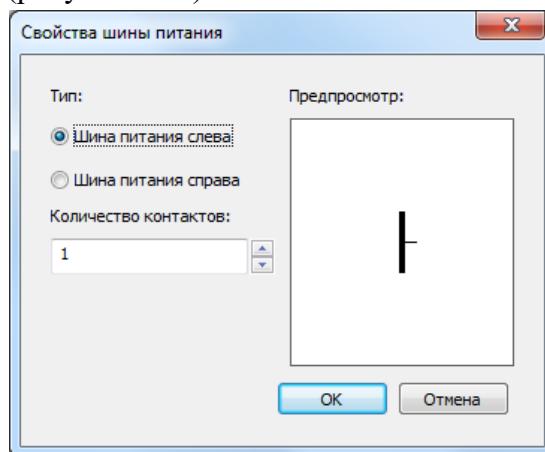


Рисунок 5.60 – Свойство шины питания

В данном диалоге указываются следующие свойства:

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

В примере ниже (рисунок 5.61) приведены две добавленные шины питания: левая с тремя контактами и правая с одним контактом.

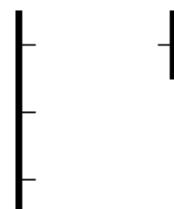


Рисунок 5.61 – Шины питания на LD диаграмме

5.7.2.2 Добавление контакта

При добавлении контакта на LD диаграмму появится диалог «Редактирование значения контакта» (рисунок 5.62).

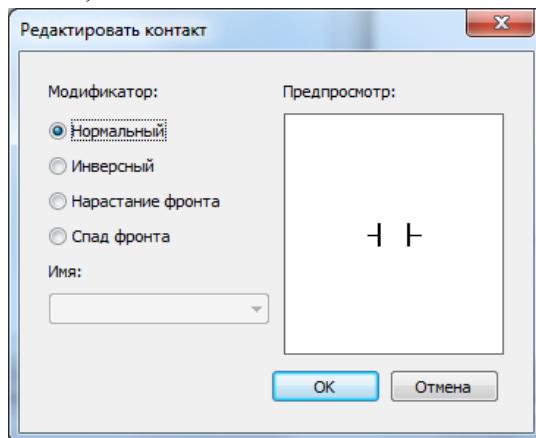


Рисунок 5.62 – Редактирование контакта

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 5.63).

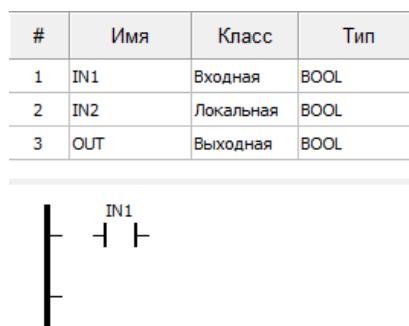


Рисунок 5.63 – Добавление контакт на диаграмму из панели переменных и констант

5.7.2.3 Добавление катушки

При добавлении катушки на LD диаграмму появится диалог «Редактирование значения катушки» (рисунок 5.64).

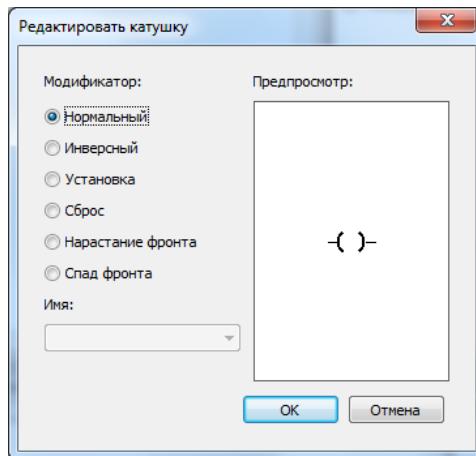


Рисунок 5.64 – Редактирование катушки

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 5.65).

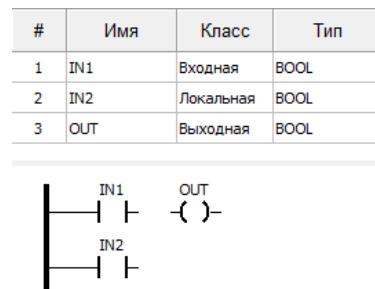


Рисунок 5.65 – Добавление катушки на диаграмму из панели переменных и констант

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 5.66).



Рисунок 5.66 – Панель редактора SFC диаграмм

С помощью данной панели можно добавить все элементы языка SFC. Функциональное назначение каждой кнопки описано в таблице 8.

Таблица 8 - Функции кнопок панели редактора SFC диаграмм

| Кнопка | Функция |
|--------|---|
| | Режим выделения 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 5.67).

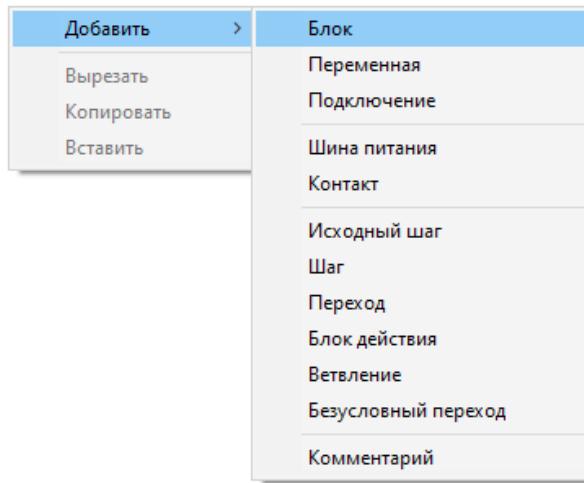


Рисунок 5.67 – Контекстное меню редактора SFC

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 5.68).

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 5.69).

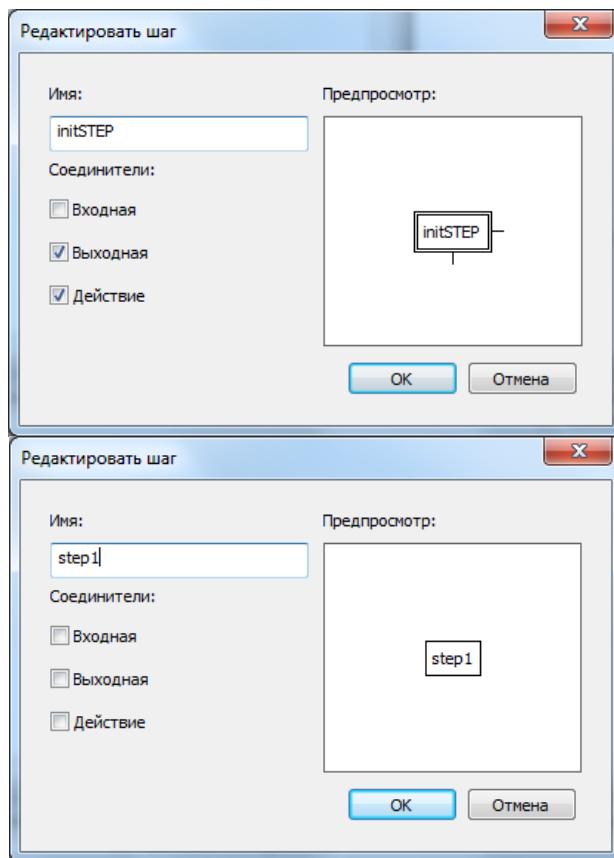


Рисунок 5.68 – Диалоги редактирования шага инициализации и обычного шага SFC
диаграммы



Рисунок 5.69 – Шаг инициализации языка SFC

В случае, если при добавлении шага не указано его имя – будет выдана ошибка (рисунок 5.70).

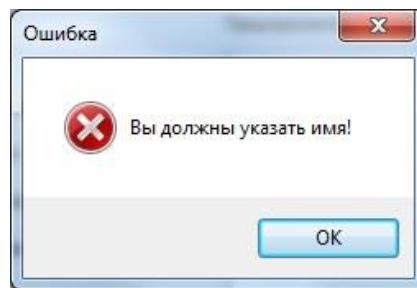


Рисунок 5.70 – Ошибка отсутствия имени шага при его добавлении

При добавлении шага появляется диалог, в котором можно указать, с помощью галочек его соединители (рисунок 5.71):

- Входная;
- Выходная;
- Действие.

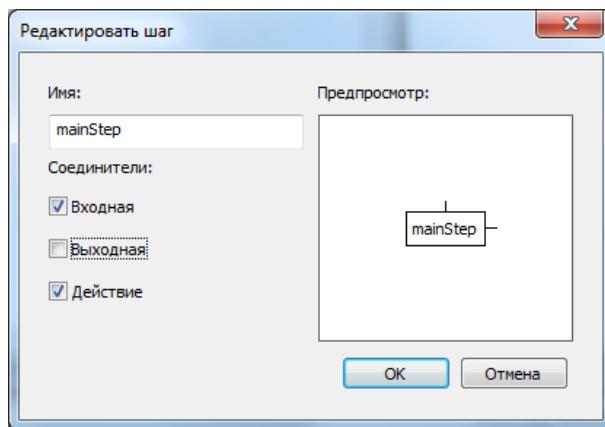


Рисунок 5.71 – Добавление шага на SFC диаграмму

«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 5.72).



Рисунок 5.72 – 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 5.73).

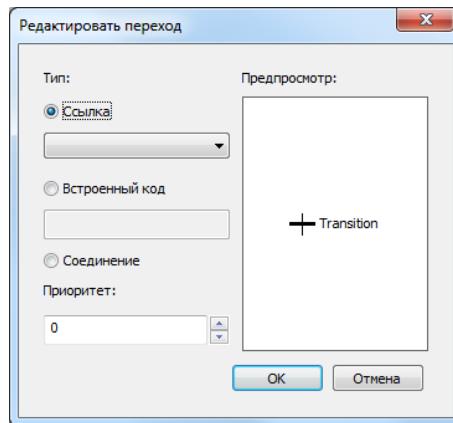


Рисунок 5.73 – Диалог редактирования перехода для SFC диаграммы

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

- Ссылка;
- Встроенный код;
- Соединение.

При выборе типа перехода «Ссылка» в открывшемся списке (рисунок 5.74) 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.

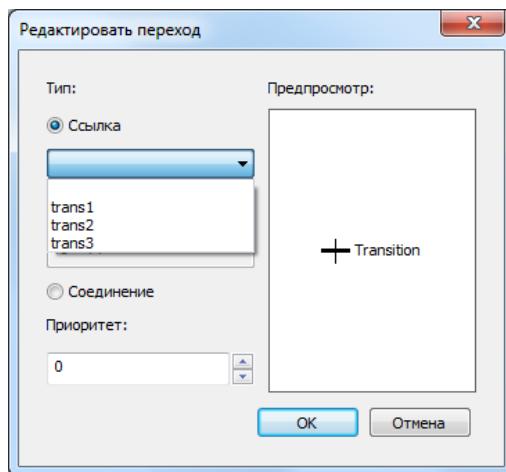


Рисунок 5.74 – Всплывающий список с доступными предопределёнными переходами

При выборе типа перехода «Встроенный код» (рисунок 5.75), условие перехода можно написать в виде выражения на языке ST.

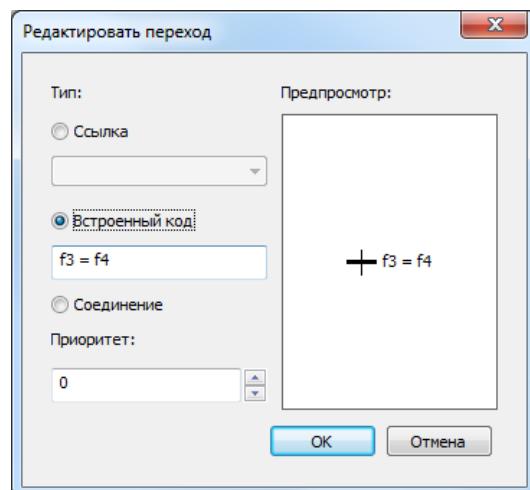


Рисунок 5.75 – Условие перехода в виде встроенного кода, написанного на языке ST

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» (рисунок 5.76), в качестве условия перехода можно использовать выходные значения элементов языка FBD или LD.

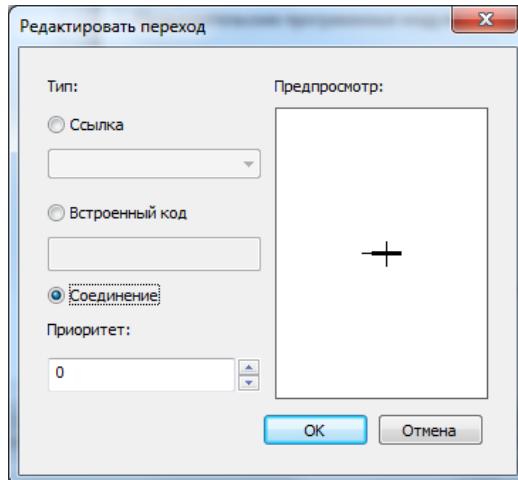


Рисунок 5.76 – Выбор условия перехода как соединение с элементами других графических языков IEC 61131-3

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. В примере ниже (рисунок 5.77) красным цветом выделен переход, условия которого заданы с помощью языка LD – двух последовательно соединённых контактов языка LD.

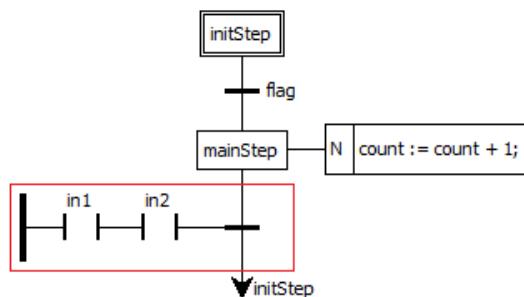


Рисунок 5.77 – Пример SFC диаграммы, в которой один из переходов задан с помощью языка LD

5.7.3.3 Добавление блока действий

При добавлении блока действий на диаграмму появится диалог «Редактировать свойство блока действий» (рисунок 5.78).

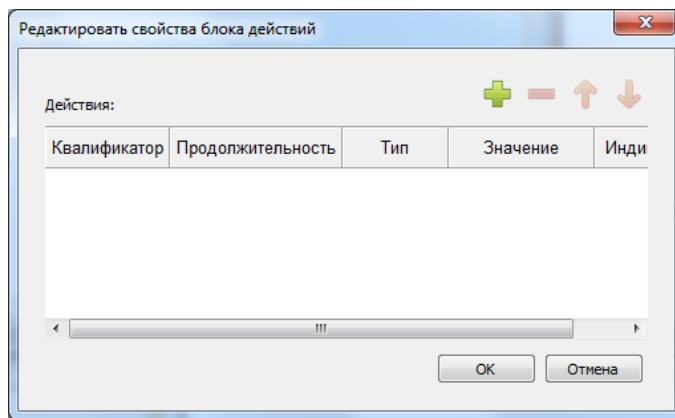


Рисунок 5.78 – Диалог «Редактировать свойство блока действий»

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 5.79).

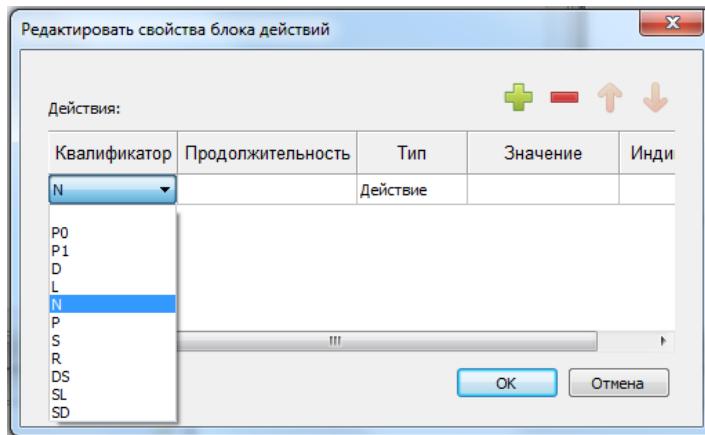


Рисунок 5.79 – Меню выбора квалификатора для действия в диаграмме SFC

Подробное описание квалификаторов, которые выбираются из предлагаемого списка при добавлении действия приведено в таблице 9.

Таблица 9 – Квалификиаторы 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 выполняяться через некоторое заданное время, только в том случае если шаг еще активен |
| SL | Действие активно в течении некоторого, заданного интервала |
| SD | Действие начинается выполняяться 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 5.80).

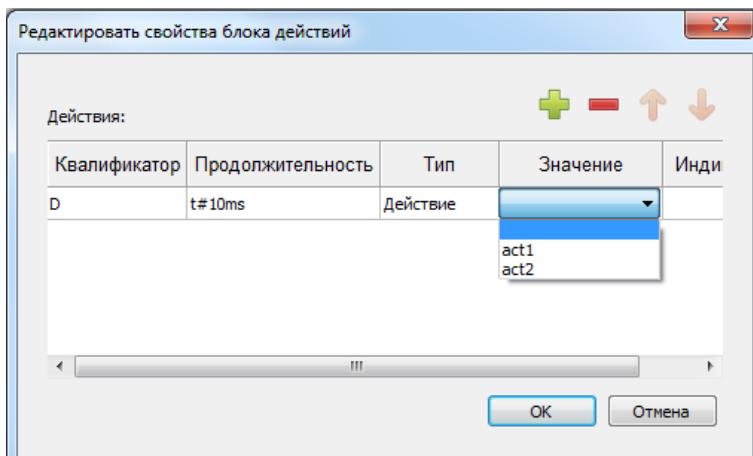


Рисунок 5.80 – Выбор предопределённого действия

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 5.81), относящиеся к данному программному модулю.

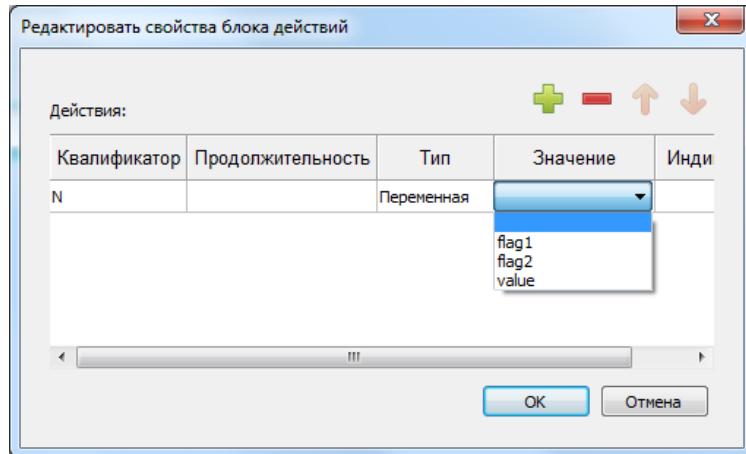


Рисунок 5.81 – Выбор предопределённой переменной

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 5.82).

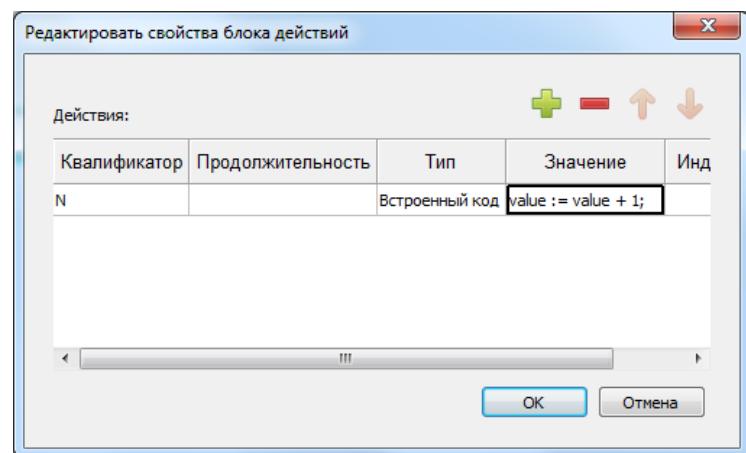


Рисунок 5.82 – Написание встроенного кода для действия

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 5.83).

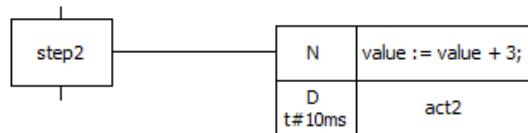


Рисунок 5.83 – Пример ассоциирование шага step2 блоком действия, содержащим два действия

5.7.3.4 Добавление дивергенции/конвергенции

При добавлении дивергенции, появится диалог «Создать новую дивергенцию и конвергенцию» (рисунок 5.84).

В первую очередь следует выбрать тип дивергенции:

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

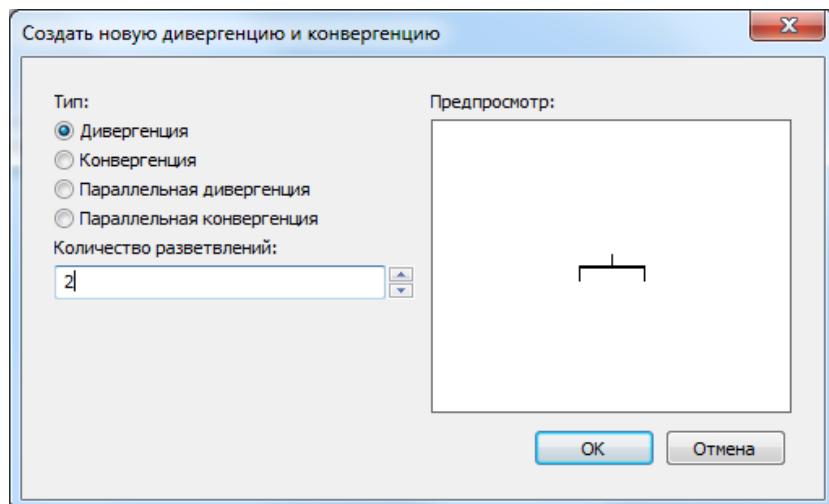


Рисунок 5.84 – Добавление дивергенции

Вторым параметром является количество разветвлений, которое определяет на сколько ветвей будет либо расходиться (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 ниже (рисунок 5.85) и выделен красным цветом.

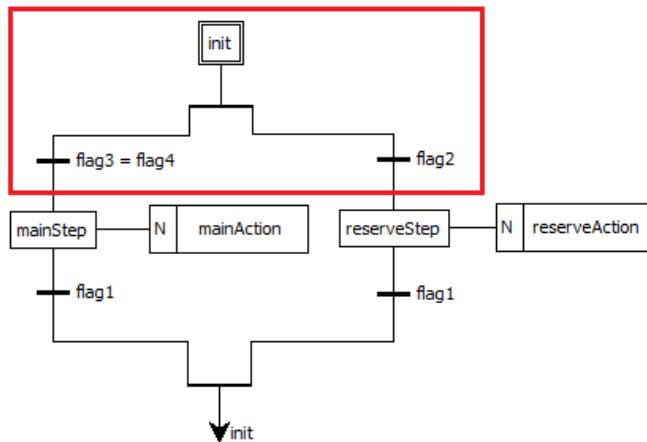


Рисунок 5.85 – Пример SFC диаграммы, содержащей дивергенцию

Пример конвергенции выделен красным цветом на рисунке ниже (рисунок 5.86).

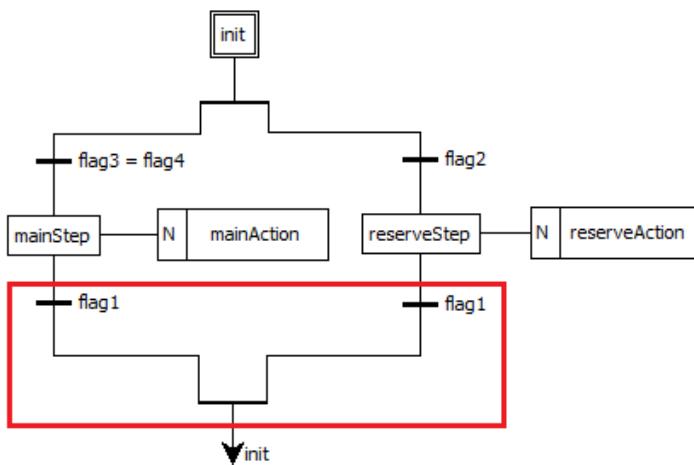


Рисунок 5.86 – Пример SFC диаграммы, содержащей конвергенцию

Пример параллельной конвергенции показан на рисунке ниже (рисунок 5.87) и выделен красным цветом.

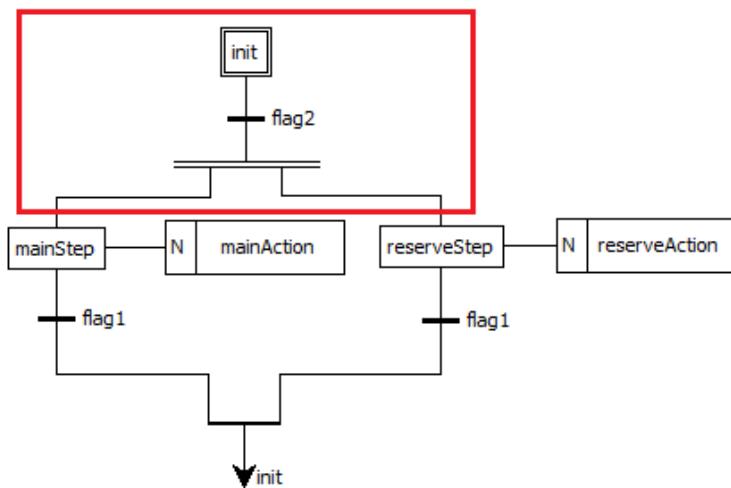


Рисунок 5.87 – Пример SFC диаграммы, содержащей параллельную дивергенцию

5.7.3.5 Добавление «прыжка»

Элемент «прыжка» 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 5.88), в котором необходимо выбрать из списка шаг, к которому будет происходить «прыжок» – переход от одного шага SFC диаграммы к другому.

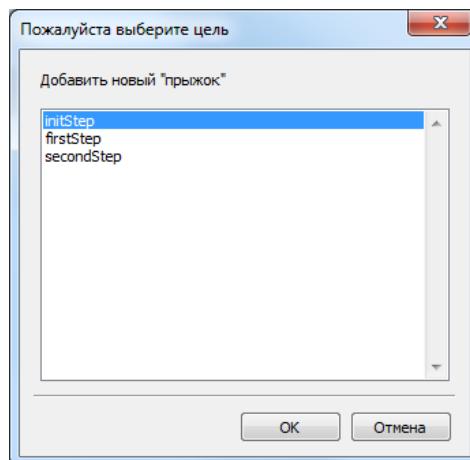


Рисунок 5.88 – Диалог добавления «прыжка»

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 5.89). Справа от стрелочки находится имя шага, к которому осуществляется переход в случае выполнения условия перехода, находящегося выше и соединённого с ней.

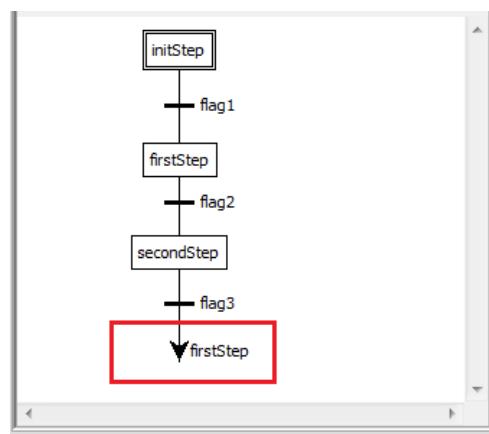


Рисунок 5.89 – «Прыжок» с шага «secondStep» на «firstStep»

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 5.90).

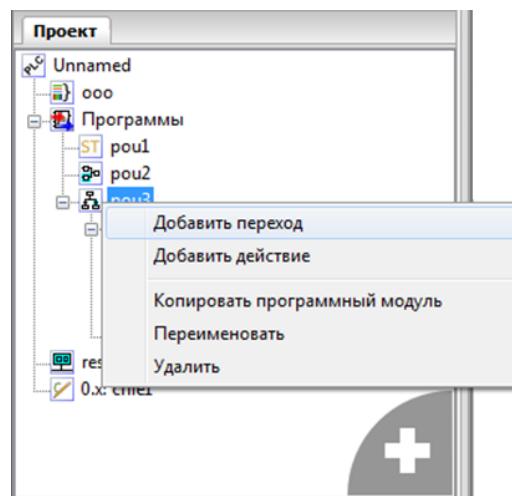


Рисунок 5.90 – Всплывающее меню SFC диаграммы в панели проекта

Далее появится диалог под названием «Создать новый переход» (рисунок 5.91). В нём необходимо выбрать уникальное имя перехода и язык, в котором будет описано данное условие.

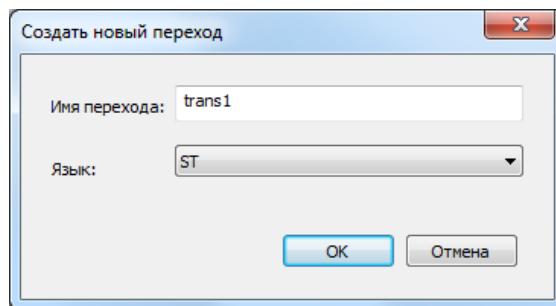


Рисунок 5.91 – Диалог «Создать новый переход»

В случае, если переходы с введённым именем уже существуют, то будет выведено сообщение об ошибке (рисунок 5.92).

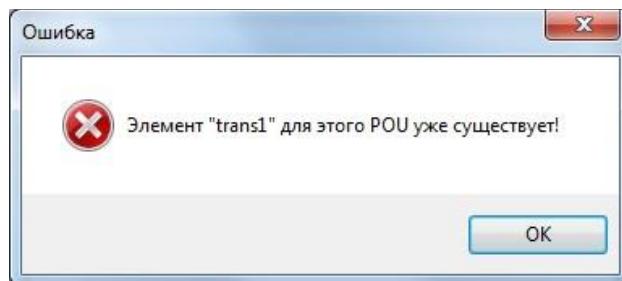


Рисунок 5.92 – Сообщение об ошибке добавления существующего программного модуля

Добавление действия в структуру SFC диаграммы (рисунок 5.93) происходит аналогично добавлению перехода в данную структуру.

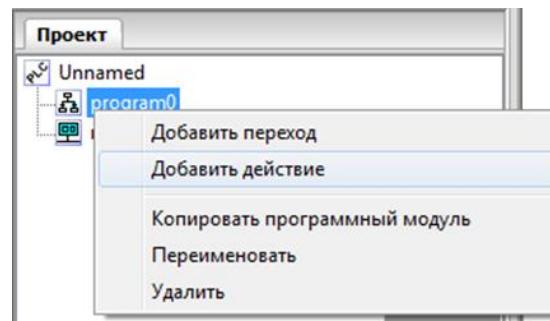


Рисунок 5.93 – Всплывающее меню SFC для структуры диаграммы

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 5.94).

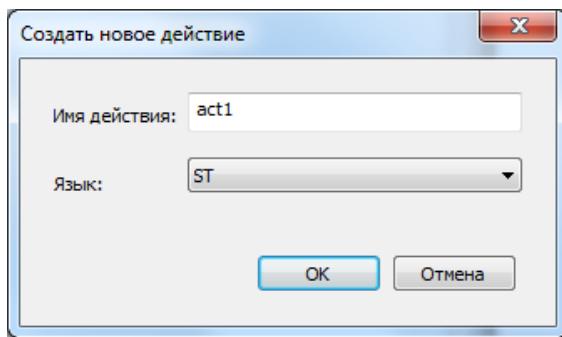


Рисунок 5.94 – Диалог «Создать новое действие»

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 5.95).

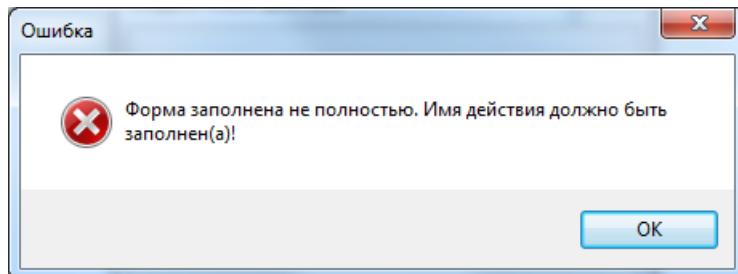


Рисунок 5.95 – Ошибка не заполнения имени действия при его добавлении

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 5.94). 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 5.96) содержит панель переменных и констант, которая позволяет определять глобальные переменные на уровне ресурса и панели, содержащие задачи и экземпляры.

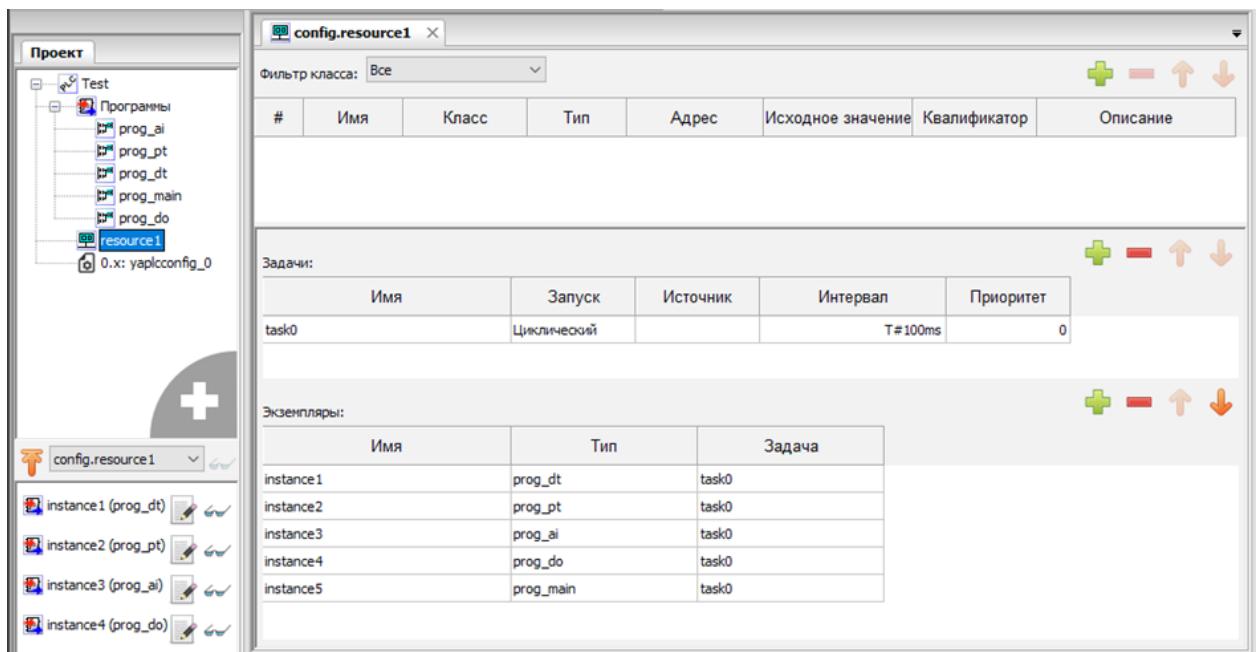


Рисунок 5.96 – Вкладка ресурс главной рабочей области

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 5.97) позволяет определить различные параметры создаваемого пользовательского типа данных.

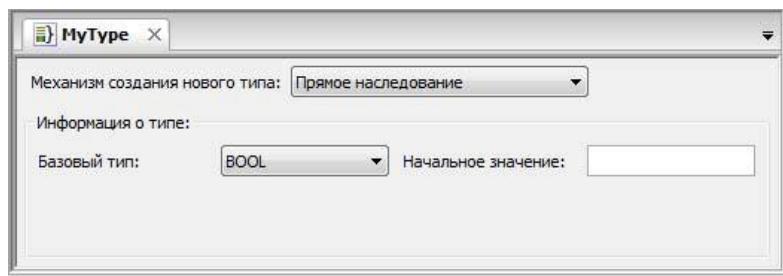


Рисунок 5.97 – Вкладка создания нового типа данных

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 нескольких типов.

Далее рассмотрены подробнее параметры для каждого из вышеперечисленных типов.

5.9.1 Прямое наследование

При выборе «Прямое наследование» (рисунок 5.98), из списка указывается базовый тип и его начальное значение.

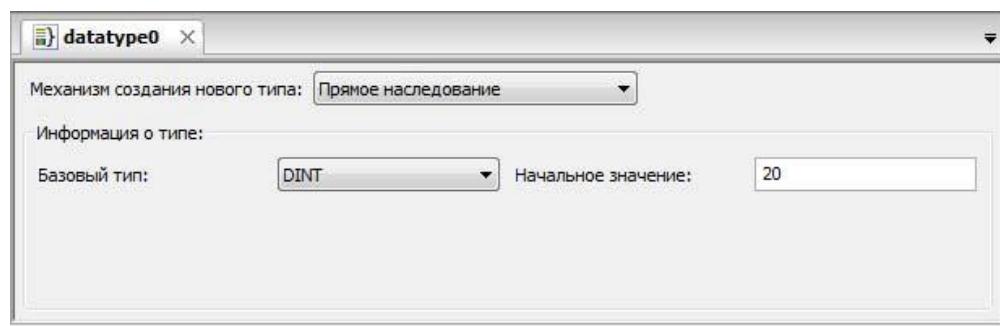


Рисунок 5.98 – Добавление псевдонима типа данных

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 5.99), т.е. соответственно минимального и максимального значения, которое может принимать создаваемый тип данных.

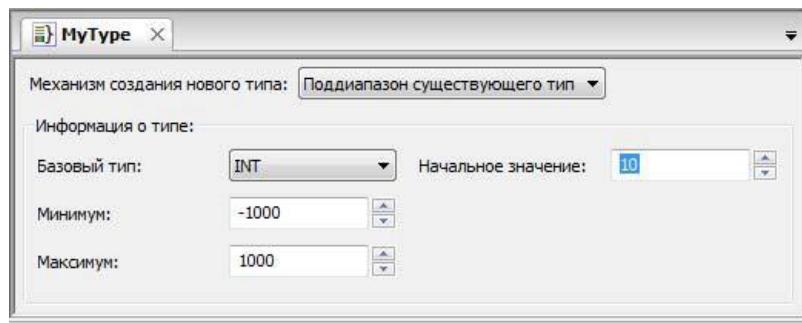


Рисунок 5.99 – Добавление нового типа данных, представляющего поддиапазон существующего типа

5.9.3 Перечисляемый тип

При выборе механизма создания нового типа «Перечисляемый тип» (рисунок 5.100), появится панель, содержащая таблицу, в которой можно задать список возможных значений данного перечисляемого типа.

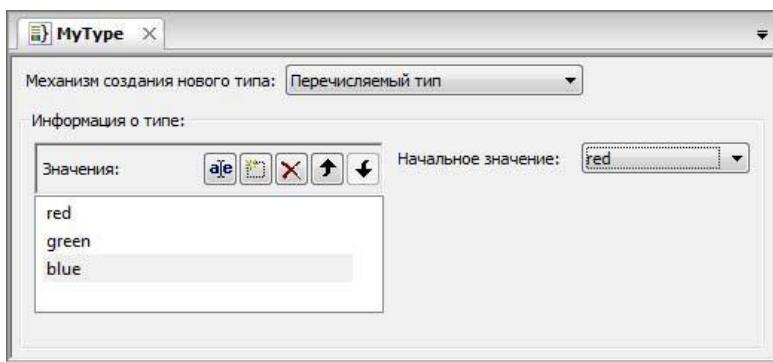


Рисунок 5.100 – Добавление перечисляемого типа данных

Добавление, редактирование, удаление, перемещение данных значений осуществляется с помощью кнопок, описание которых приведено в таблице 10.

Таблица 10 - Функции кнопок редактирования значений перечисляемого типа

| Кнопка | Функция |
|--------|---|
| | Редактировать выделенное поле |
| | Добавить новое поле |
| | Удалить выделенное поле |
| | Переместить выделенное поле на одну позицию вверх |
| | 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 5.101) появится панель, в которой необходимо указать базовый тип, начальное значение, а так же размерность массива.

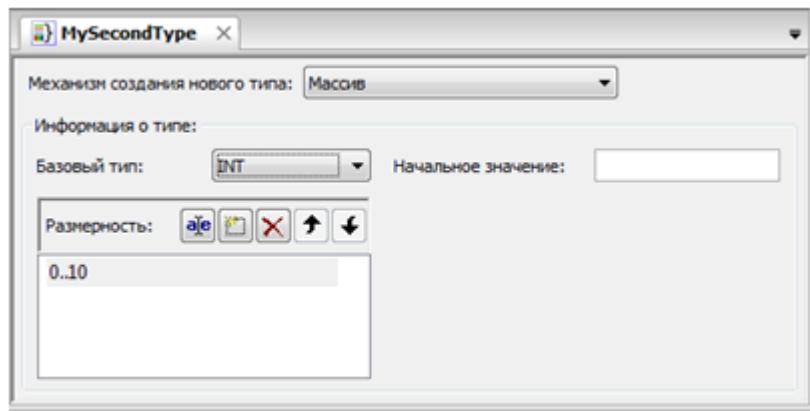


Рисунок 5.101 – Добавление типа данных «Массив»

Размерность массива задаётся в следующем формате:

<начальное значение>..<конечное значение>

5.9.5 Структура

При выборе механизма создания нового типа «Структура» (рисунок 5.102), в появившейся таблице необходимо добавить необходимое количество полей структуры. Каждое поле имеет своё имя, тип и начальное значение.

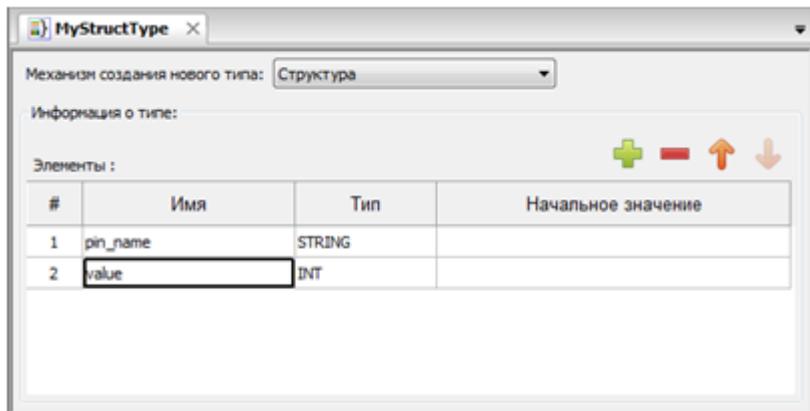


Рисунок 5.102 – Добавление типа данных «Структура»

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 5.103) обычно располагается слева в среде разработки Begetiz и отображаемые в ней экземпляры зависят от выбранного элемента в дереве проекта.

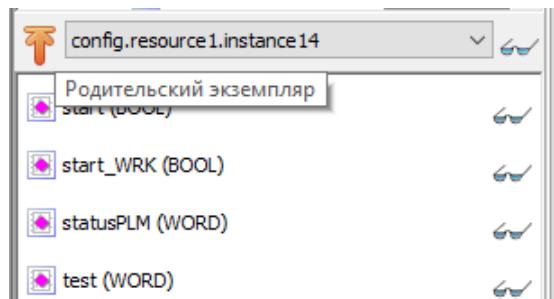


Рисунок 5.103 – Панель экземпляров проекта

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 ниже (рисунок 5.104) показано, как в панели редактирования ресурса определена глобальная переменная «globalValue» и два экземпляра для программных модулей «program0» и «program1»:

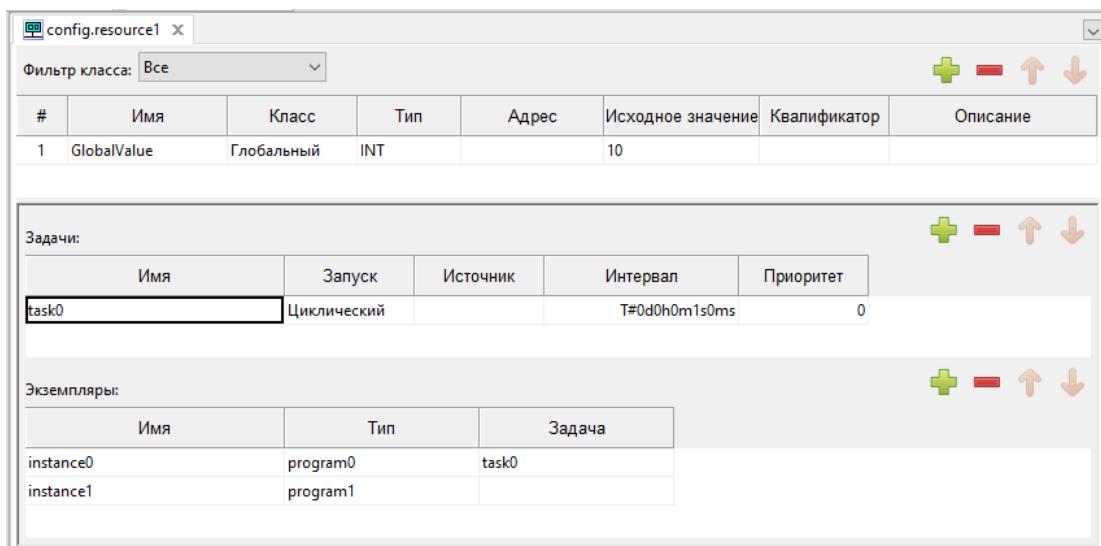


Рисунок 5.104 – Пример с двумя экземплярами проекта в панели редактирования ресурса

Соответственно, при выборе в дереве проекта ресурса, в котором определены экземпляры (описанные выше) и глобальная переменная, панель экземпляров будет выглядеть, как показано на рисунке ниже (рисунок 5.105):

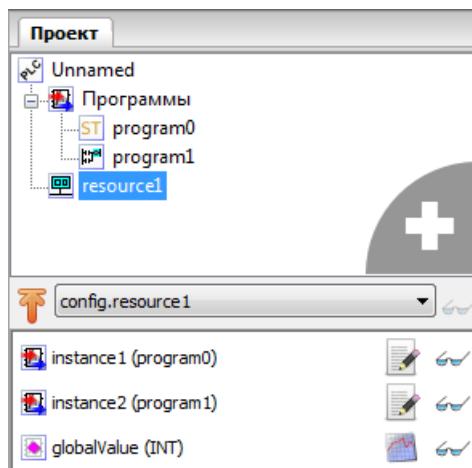


Рисунок 5.105 – Панель экземпляров при выборе элемента ресурса в дереве проекта

При выборе в дереве проекта элемента, соответствующего программным модулям «Программа» и «Функциональный блок» в панели экземпляров будут отображены переменные, определённые в них. На рисунке ниже (рисунок 5.106) приведён пример программного модуля типа «Программа» с именем «program0», в котором определено 6 переменных различных классов, одна из которых глобальная.

| # | Имя | Класс | Тип | Адрес | Исходное значение | Квалификатор | Описание |
|---|-------------|------------|--------|-------|-------------------|--------------|----------|
| 1 | pin_in | Вход | REAL | | | | |
| 2 | pin_out | Выход | USINT | | | | |
| 3 | pin_in_out | Вход/Выход | INT | | | | |
| 4 | pin_local | Локальный | INT | | | | |
| 5 | temp_string | Временный | STRING | | | | |

Рисунок 5.106 – Программный модуль типа «Программа»

Соответственно, при выборе в дереве проекта данного программного модуля в панели экземпляров будут отображены, определённые выше переменные (рисунок 5.107).

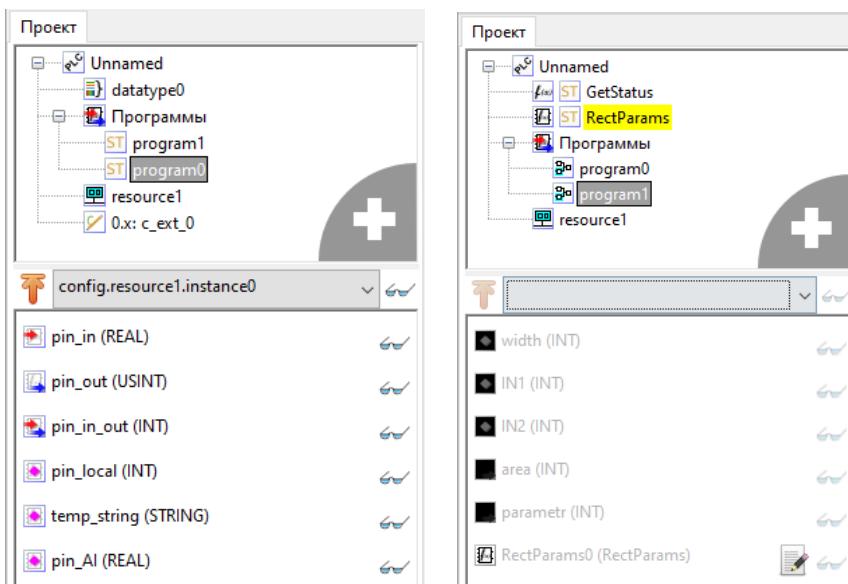


Рисунок 5.107 – Панель экземпляров проекта при выборе в дереве проекта программного модуля типа «Программа» при соединении с устройством и при отсутствии соединения с устройством

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 11.

Таблица 11 - Функции кнопок панели экземпляра проекта

| Кнопка | Функция |
|--------|--|
| | Запуск режима отладки для экземпляра при наличие соединения с устройством |
| | Запуск режима отладки для экземпляра невозможен, из-за отсутствия соединения с устройством |

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 Дерево переменных.

Дерево переменных (рисунок 5.108) располагается справа в среде разработки Beremiz. Оно содержит все переменные, использующиеся в проекте, а так же их типы данных. Переменные разделены по программным модулям, в которых они используются.

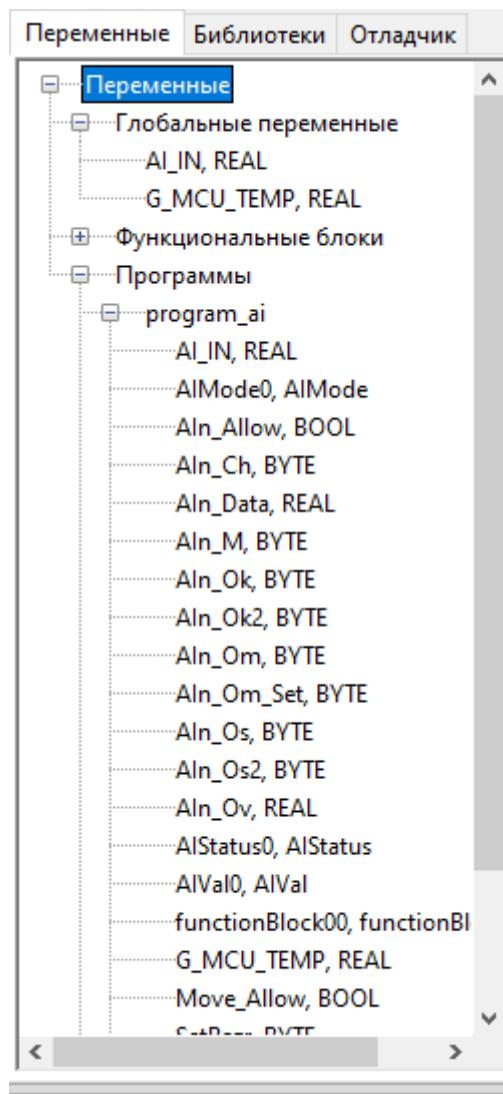


Рисунок 5.108 – Дерево переменных.

При двойном щелчке по переменной открывается список переменных выбранной программы. Выбранная переменная отмечается цветом (см. рисунок 5.109).

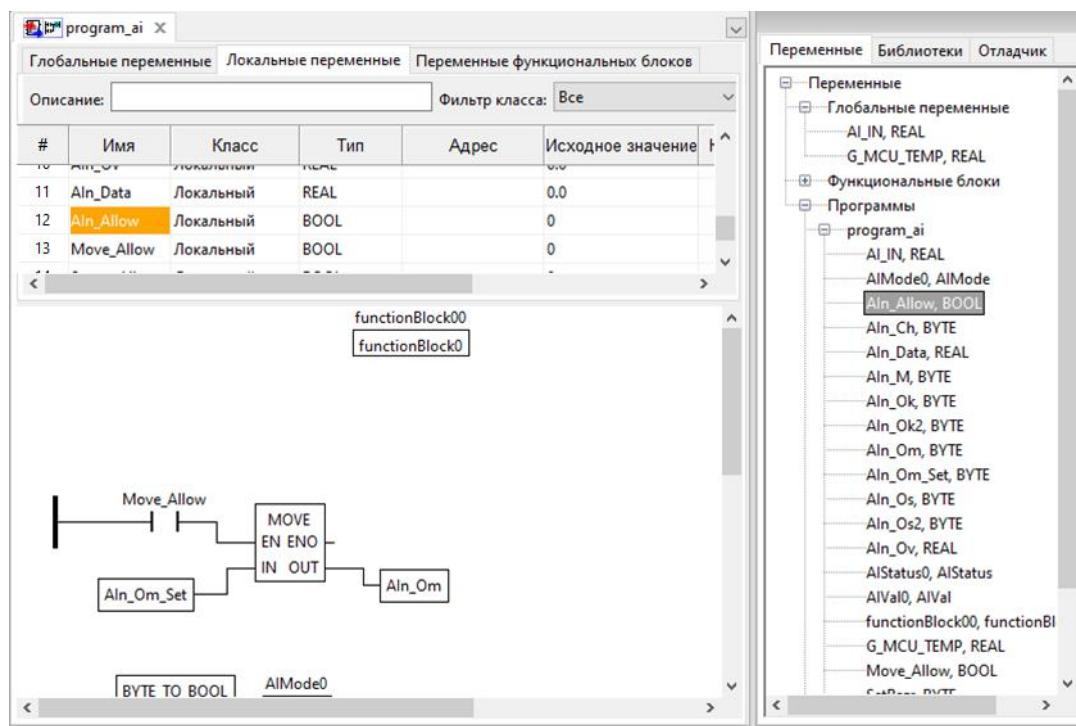


Рисунок 5.109 – Переход к выбранной переменной

5.12 Панель библиотеки функций и функциональных блоков

Панель библиотеки функций и функциональных блоков (рисунок 5.110), 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.

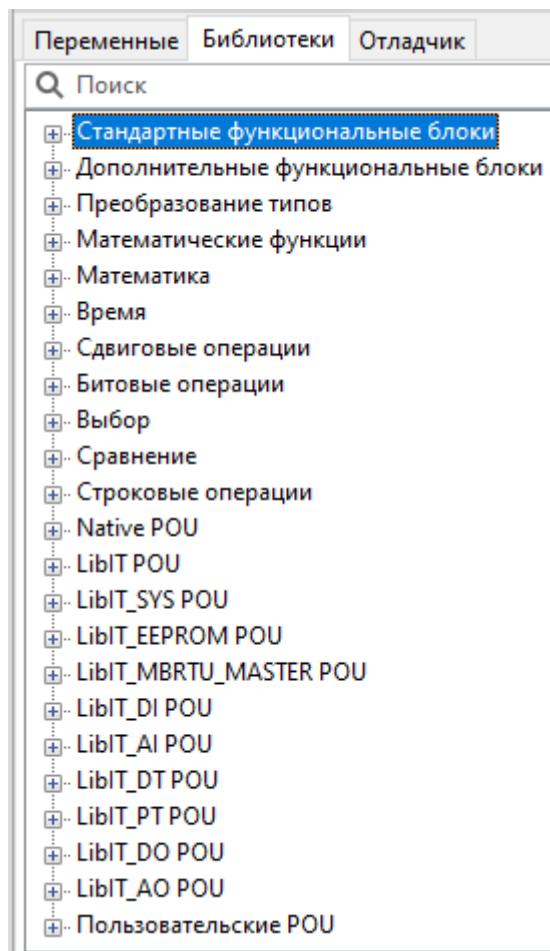


Рисунок 5.110 – Панель библиотеки функций и функциональных блоков

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.

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](#).

5.13 Отладочная консоль

Панель, содержащая отладочную консоль (рисунок 5.111), как правило, располагается в правом нижнем углу среды разработки Beremiz.

```

Поиск Консоль Лог ПЛК
Сборка запущена в D:\Download\Микроволновка\V2.02_Python3\Beremiz_V2.02\build
ld
Генерация МЭК-61131 ST/IL/SFC кода ПЛК...
Компиляция МЭК-программы в С-код...
Экспорт локальных переменных...
0 -> Nothing to do
С-код успешно сгенерирован.

ПЛК:
[CC] plc_main.c -> plc_main.o
[CC] plc_debugger.c -> plc_debugger.o

ПЛК:
[CC] config.c -> config.o
[CC] resourcel.c -> resourcel.o

Линковка:
[CC] plc_main.o plc_debugger.o config.o resourcel.o -> Beremiz_V2.02.elf
if
[OBJCOPY] Beremiz_V2.02.elf -> Beremiz_V2.02.elf.hex
Output size:
text data bss dec hex filename
131840 2576 18064 152480 253a0 D:\Download\Микроволновка\V2.02_Py
ton3\Beremiz_V2.02\build\Beremiz_V2.02.elf
Сборка прошла успешно.

```

Рисунок 5.111 – 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 5.112).

```

Поиск Консоль Лог ПЛК
Сборка запущена в D:\Download\Микроволновка\V2.02_Python3\Beremiz_V2.02\build
Генерация МЭК-61131 ST/IL/SFC кода ПЛК...
Компиляция МЭК-программы в С-код...
"C:\Program Files (x86)\Beremiz-IT03.399.22\matiec\iec2c.exe" -f -l -p -I "C
:\Program Files (x86)\Beremiz-IT03.399.22\matiec\lib" -T "D:\Download\Микрово
лновка\V2.02_Python3\Beremiz_V2.02\build" "D:\Download\Микроволновка\V2.02_Pyt
on3\Beremiz_V2.02\build\plc.st"
завершился с кодом 1 (pid 15244)
D:\Download\Микроволновка\V2.02_Python3\Beremiz_V2.02\build\plc.st:2667-3..266
7-15: error: invalid variable before ':' in ST assignment statement.

In section: PROGRAM program_DO
2667: mag_1_PLK_Out := DOVal4.Ov;

1 error(s) found. Bailing out!

Ошибка: компилятор МЭК в С вернул код ошибки 1.
Неудачная генерация кода!

```

Рисунок 5.112 – Ошибка сборки проекта в отладочной консоли

5.14 Поиск элементов в проекте

Для поиска элемента в проекте используется диалог «Поиск в проекте» (рисунок 5.113), вызов которого возможен с помощью главного меню программы (п. 5.1) или панели инструментов (п. 5.2).

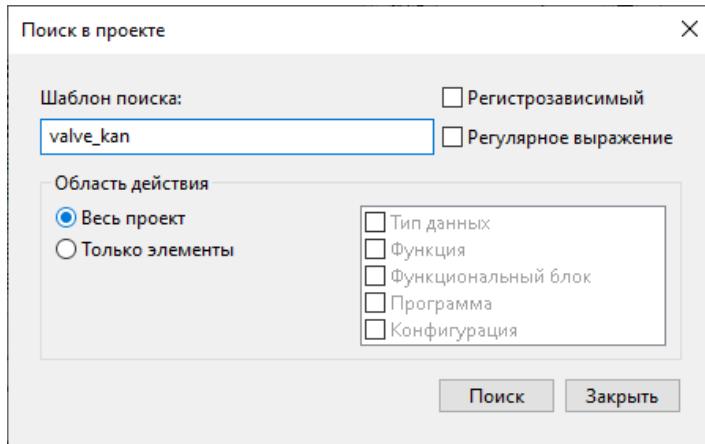


Рисунок 5.113 – Диалог поиска в проекте

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. На рисунке ниже (рисунок 5.114) приведен пример поиска элемента с именем «valve_kan».

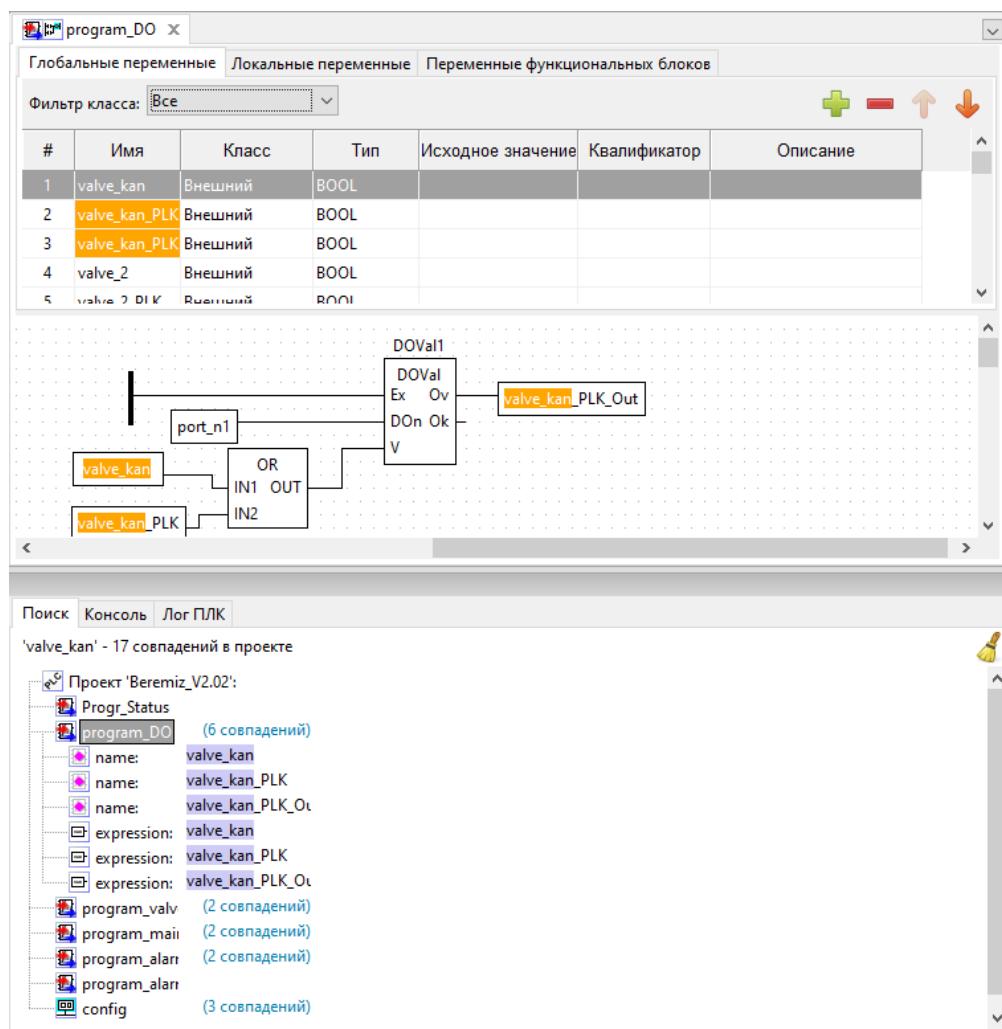


Рисунок 5.114 – Результат примера поиска элемента в проекте

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

5.15 Панель отладки

Панель отладки располагается в правой части среды разработки Beremiz (рисунок 5.115).

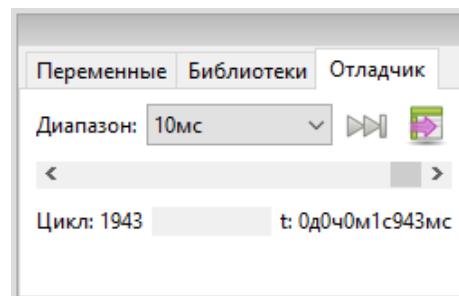


Рисунок 5.115 – Панель отладки

Добавление в панель переменных осуществляется с помощью панели экземпляров проекта (см. п. 5.10). Изменение значений переменной во время отладки прикладной программы осуществляется кликом левой клавишей мыши по значку 1 (рисунок 5.116.б).

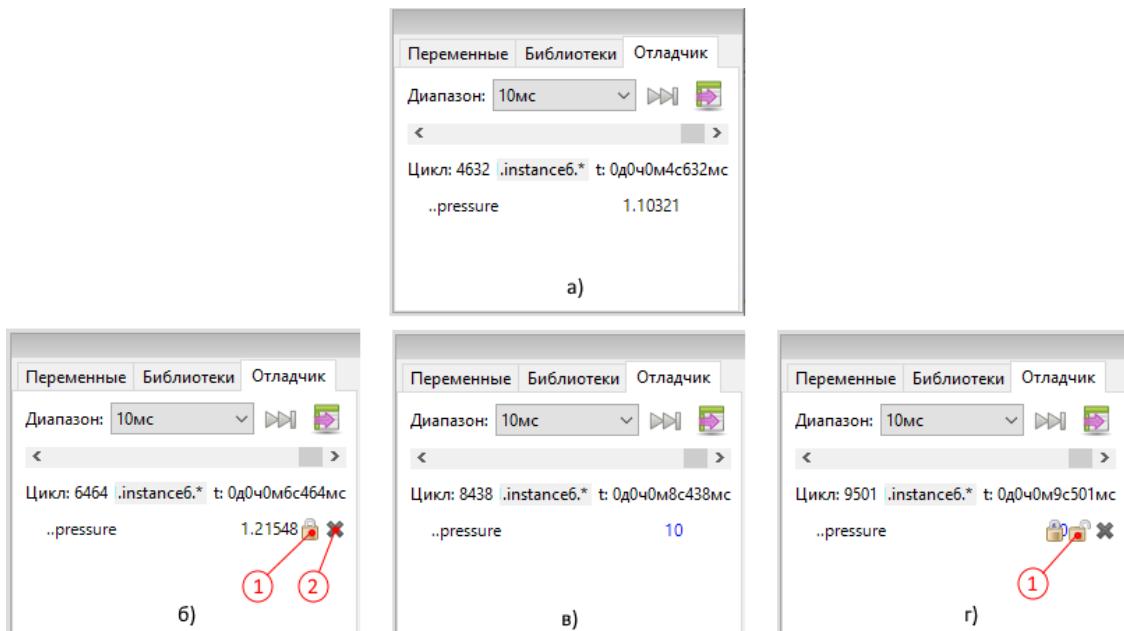


Рисунок 5.116 – Контекстное меню управления значением

Далее появится диалог ввода значения для выбранной переменной (рисунок 5.117).

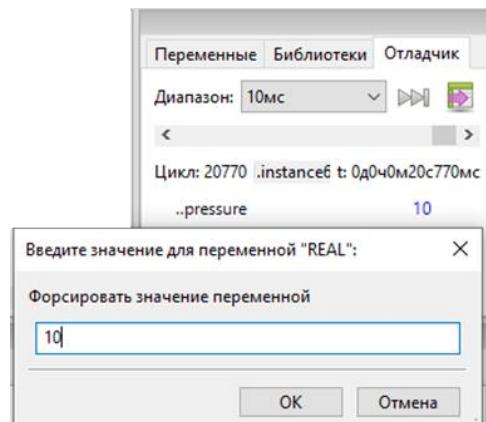


Рисунок 5.117 – Установка значения переменной во время отладки

После изменения значения переменной она переводится в отладочный режим (в панели отладчика она при этом отображается синим цветом (рисунок 5.116.в)), т.е. ее значение не может быть изменено действиями из отлаживаемой программы, а только вручную.

Для того, чтобы вернуть переменную под управление программы пользователя, ее нужно освободить. Для этого надо навести курсор мыши на строку переменной в панели отладчика (рисунок 5.116.г) и кликнуть левой кнопкой мыши по значку 1 (рисунок 5.116.г).

Для удаления переменной из панели отладки нужно кликнуть левой кнопкой мыши по значку 2 (рисунок 5.116.б).

5.16 Панель графика изменения значения переменной в режиме отладки

Двойной клик левой кнопкой мыши по строке переменной в панели отладчика переводит ее в режим отображения в виде графика (рисунок 5.118).

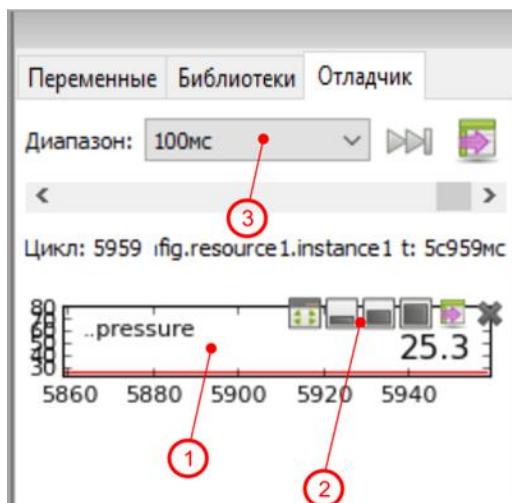


Рисунок 5.118 – График изменения значения переменной во время отладки

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 12.

Таблица 12 - Функции кнопок управления графиком изменения значения переменной

| Кнопка | Функция |
|--------|---|
| | Перейти к последней точке графика справа |
| | Сбросить масштаб в исходное состояние |
| | Отображение поля графика в четверть размера |
| | Отображение поля графика в половину размера |
| | Отображение поля графика в полный размер |
| | 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 6.1);

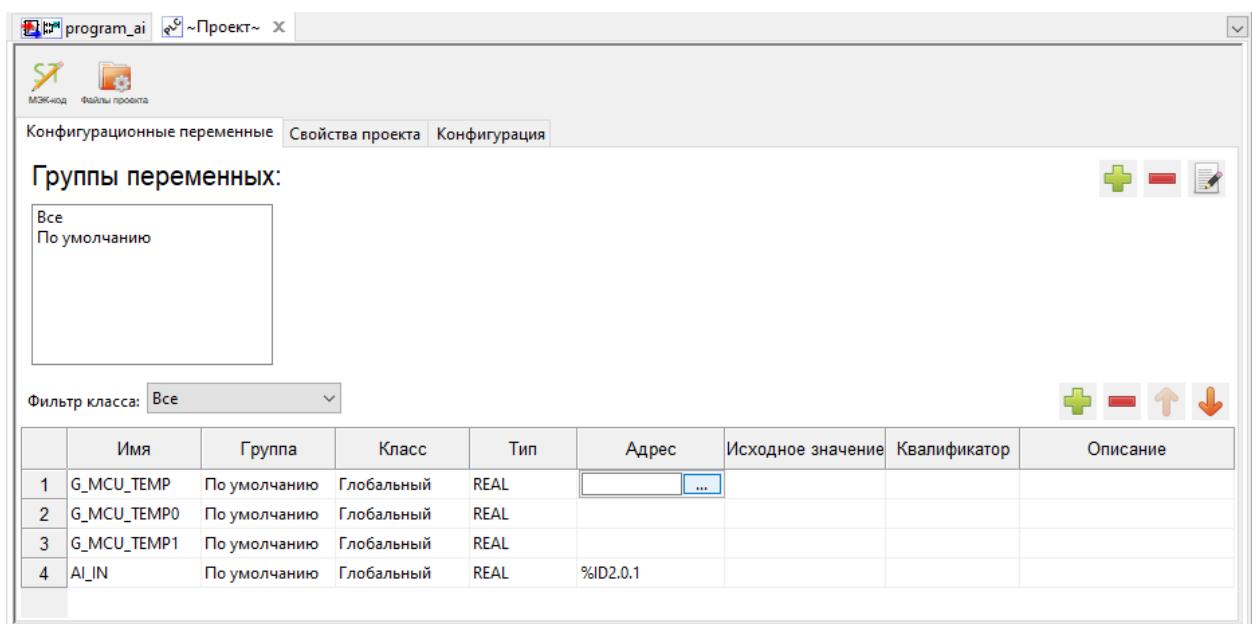


Рисунок 6.1 – Поле «Адрес» панели переменных и констант

- нажав появившуюся кнопку «...» в поле «Адрес» появится диалог «Просмотр адресов» (рисунок 6.2), в котором отображаются доступные для связывания регистры внешних модулей;

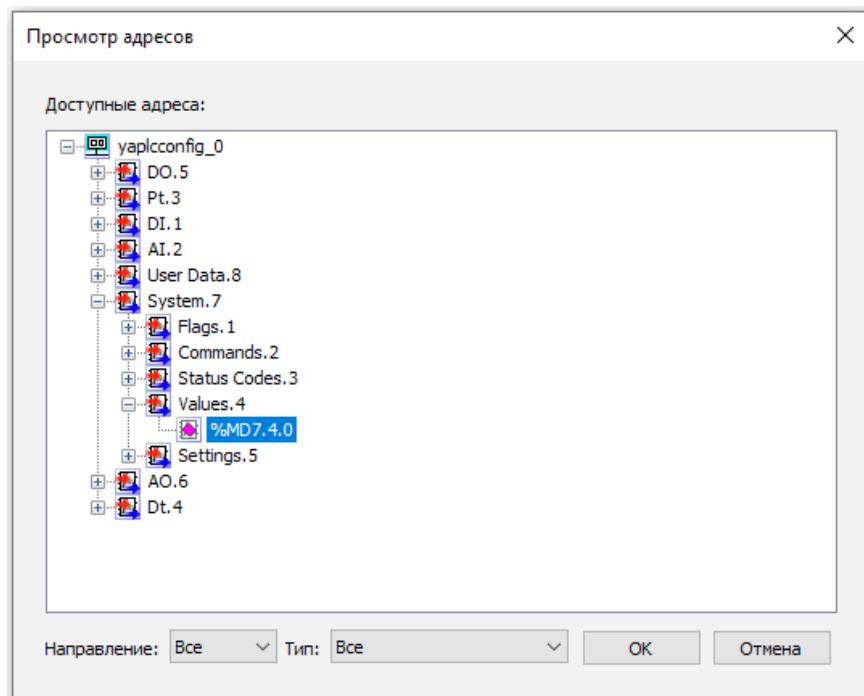


Рисунок 6.2 – Диалог просмотра адресов

- после выбора необходимой переменной и нажатия кнопки «OK» будет выдано диалоговое окно выбора класса переменной: Вход, Выход, Память (рисунок 6.3);

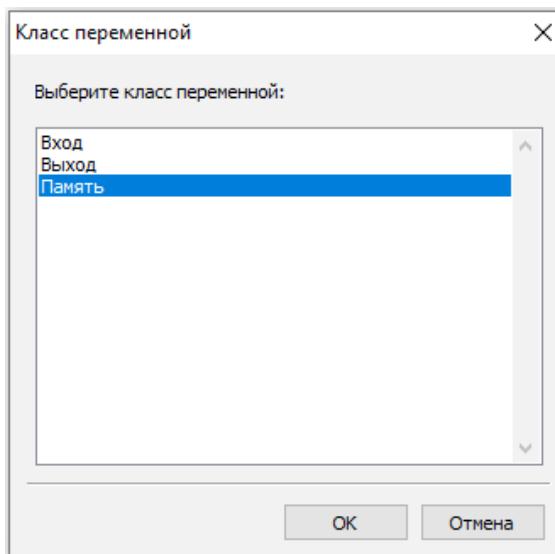


Рисунок 6.3 – Диалог выбора класса переменной

- после выбора класса и нажатия кнопки «OK» поле адреса соответствующей переменной будет заполнено адресом внешней переменной (рисунок 6.4).

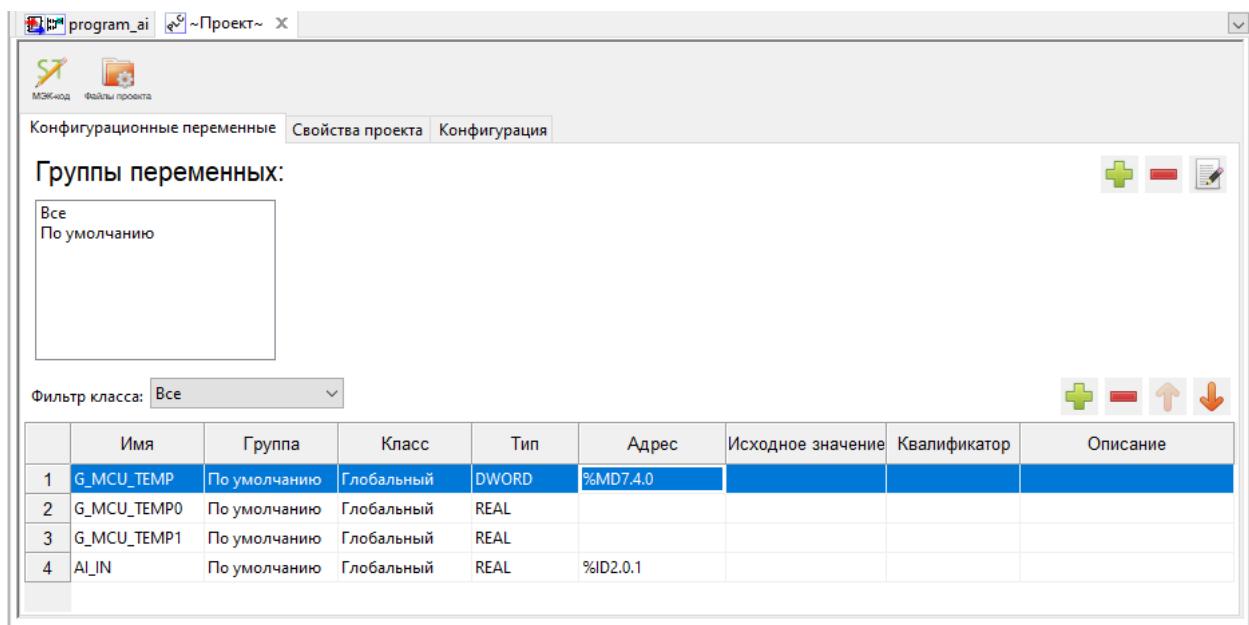


Рисунок 6.4 – Добавленный адрес переменной

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 13).

Таблица 13 – Коды поддерживаемых типов данных

| Код | Тип | | Диапазон значений | Размер | |
|-----|-------|--------|-------------------|--------|-------|
| | IEC | Cи | | биты | байты |
| X | BOOL | uint8 | 0 ... 255 (0, 1) | 8 | 1 |
| B | BYTE | uint8 | 0 ... 255 | 8 | 1 |
| | USINT | int8 | -128 ... 127 | | |
| W | WORD | uint16 | 0 ... 65535 | 16 | 2 |
| | UINT | int16 | -32768 ... 32767 | | |
| | INT | | | | |

| Код | Тип | | Диапазон значений | Размер | |
|-----|---------------------------------|---------------------------|--|--------|-------|
| | IEC | Си | | биты | байты |
| D | DWORD UDINT DINT REAL | uint32 int32 float | 0 ... 4294967295 -2147483648 ... 2147483647 $3.4 \cdot 10^{-38}$... $3.4 \cdot 10^{38}$ | 32 | 4 |
| L | LWORD ULINT LINT LREAL | uint64 int64 double | 0 ... $(2^{64}-1)$ $-(2^{63}-1)$... $(2^{63}-1)$ $1.7 \cdot 10^{-308}$... $1.7 \cdot 10^{308}$ | 64 | 8 |

Например, на Рисунок 6.4 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 7.1), либо с помощью кнопки «Новый» на панели управления (рисунок 7.2).

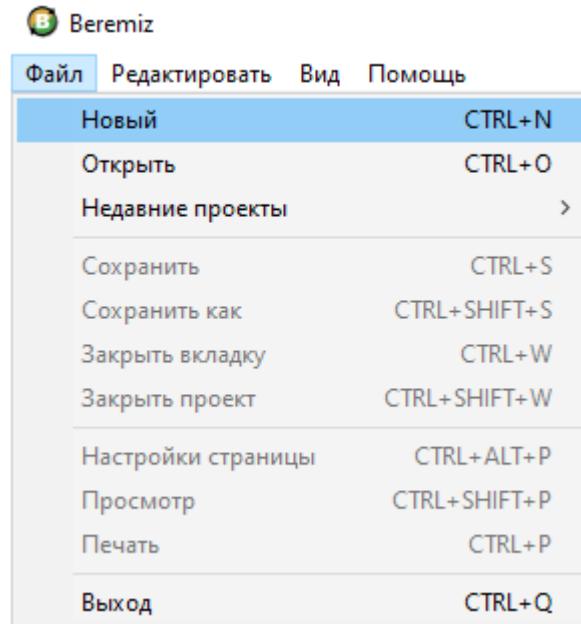


Рисунок 7.1 – Создание нового проекта через меню «Файл»

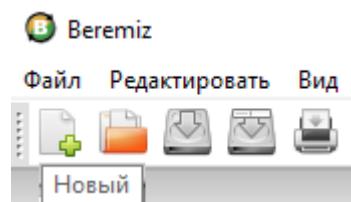


Рисунок 7.2 – Создание нового проекта через кнопку «Новый» панели управления

Далее появится диалог (рисунок 7.3), в котором необходимо выбрать папку, где будет храниться данный проект.

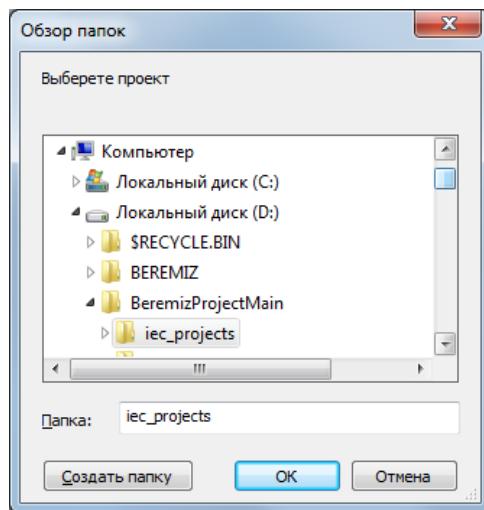


Рисунок 7.3 – Диалог выбора папки для нового проекта

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 ниже (рисунок 7.4):

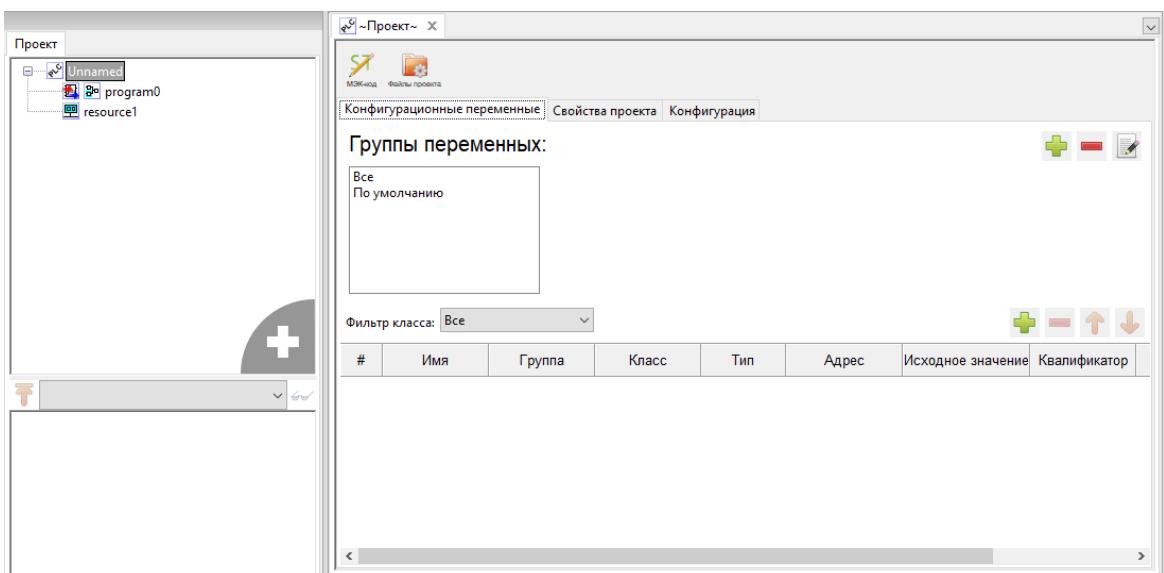


Рисунок 7.4 – Вызов панели настройки проекта с помощью корневого элемента дерева проекта

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, рисунок 7.5, в панели переменных и констант создана глобальная переменная «globalValue» с начальным значением 10, с помощью кнопки «Добавить переменную» (таблица 3).

| # | Имя | Группа | Класс | Тип | Адрес | Исходное значение | Квалификатор | Описание |
|---|-------------|--------------|------------|-----|-------|-------------------|--------------|----------|
| 1 | globalValue | По умолчанию | Глобальный | INT | | 10 | | |

Рисунок 7.5 – Объявление глобальной 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 (выбрать из списка), как и ранее объявленная глобальная, и установить её класс «Внешняя». На рисунке ниже (рисунок 7.6) приведён пример объявления в программном модуле «program0» переменной «globalValue» класса «Внешняя», типа INT и изменение её значения с помощью языка ST.

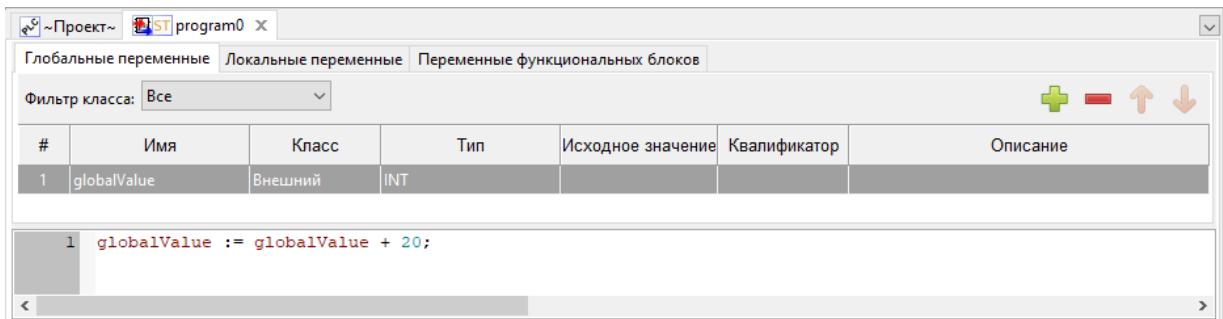


Рисунок 7.6 – Использование глобальной переменной в программном модуле «program0»

На рисунке ниже (рисунок 7.7) в программном модуле «program1» также определена переменная «globalValue» класса «Внешняя» и изменение её значения с помощью языка ST.

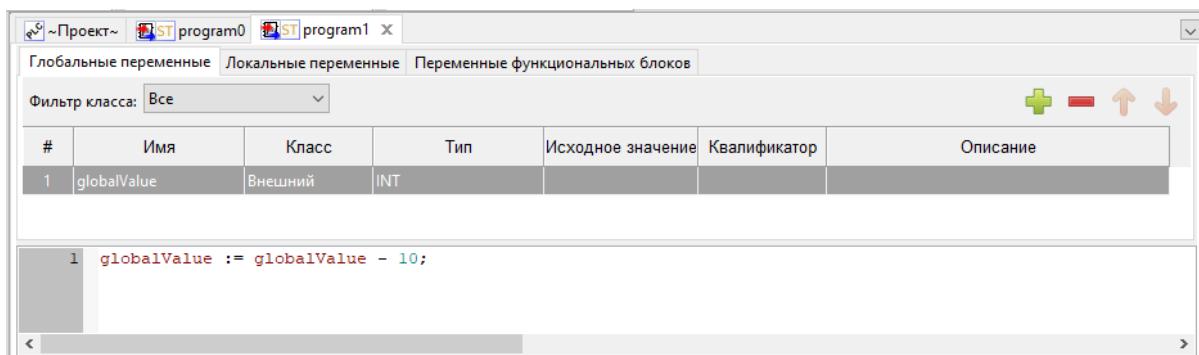


Рисунок 7.7 – Использование глобальной переменной в программном модуле «program1»

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 рисунке ниже (рисунок 7.8) данные параметры выделены красным цветом.

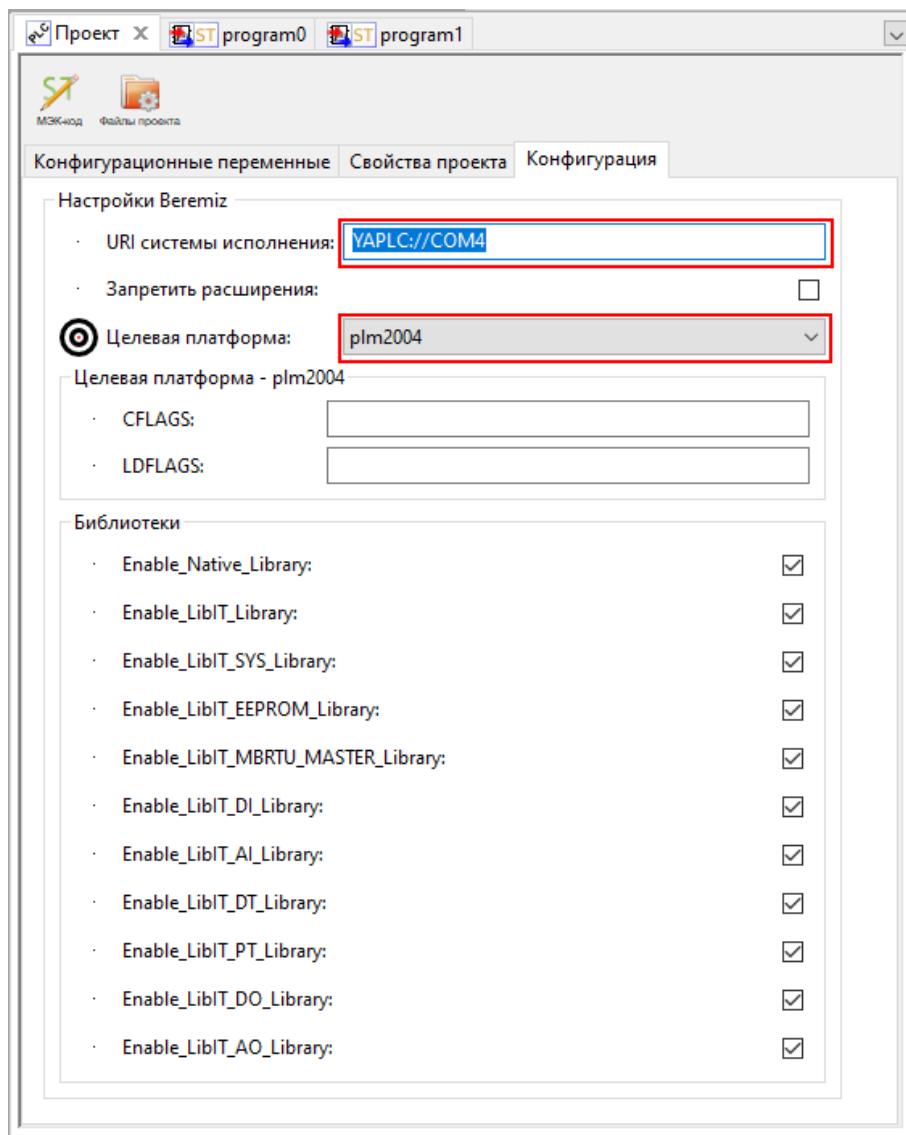


Рисунок 7.8 – Задание URI-адреса системы исполнения и типа целевой платформы

«URI адрес» указывается в формате:

$$YAPLC://<\text{номер COM-порта}\rangle$$

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 7.9), позволяющую удобным образом различать проекты.

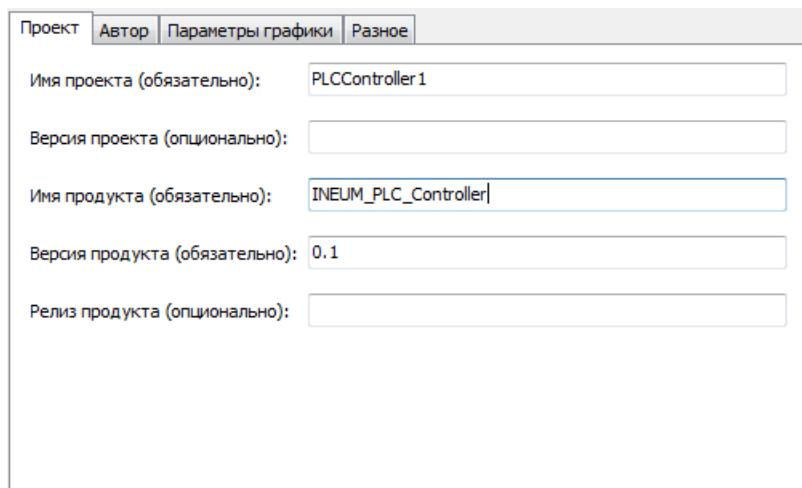


Рисунок 7.9 – Указание данных о проекте

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 POU» (рисунок 7.10).

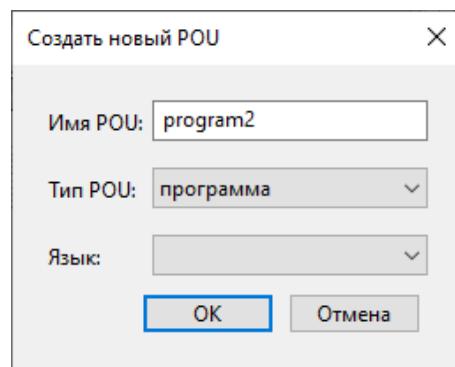


Рисунок 7.10 – Диалог добавления программного модуля

В данном диалоге три поля:

- Имя POU;
- Тип POU;
- Язык.

Имя, присвоенное по умолчанию, может быть заменено на имя, соответствующее назначению данного POU. В зависимости от того, какой программный модуль был выбран во всплывающем меню, в поле «Тип POU» будет подставлено именование данного программного модуля. В поле «Язык» необходимо выбрать из списка (рисунок 7.11) один из языков стандарта IEC 61131-3 (IL, ST, LD, FBD, SFC), на котором будет реализованы алгоритмы и логика работы данного добавляемого программного модуля.

Нужно отметить, что если есть потребность использования языка FBD, то лучше выбрать LD, т.к. в этом случае будет возможность в программном модуле использовать помимо релейной логики и FBD тоже.

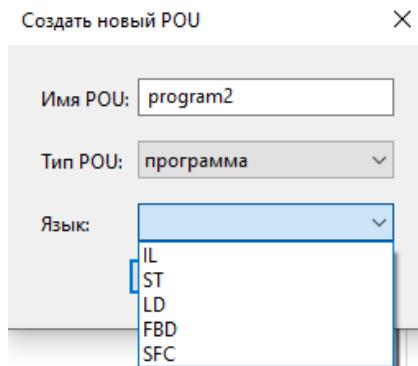


Рисунок 7.11 – Выбор языка для программного модуля

Далее рассмотрено 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 7.12):

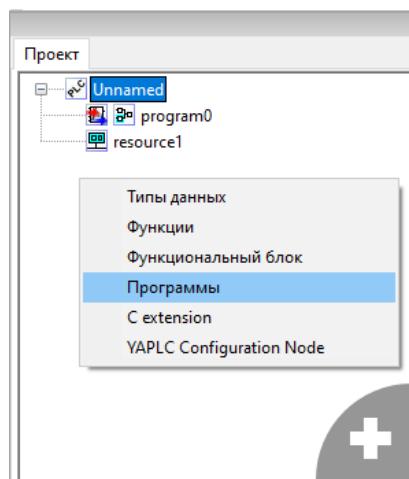


Рисунок 7.12 – Выбор в дереве проекта добавления программы

В появившемся диалоге (рисунок 7.13) выбирается язык FBD и нажимается кнопка «OK».

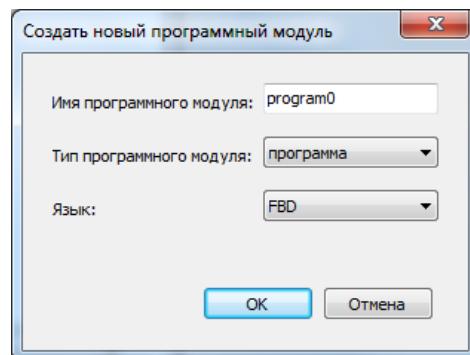


Рисунок 7.13 – Диалог добавления программы

Далее в открывшейся вкладке с панелью редактирования данного программного модуля в панели переменных и констант (рисунок 7.14) добавляются переменные: «globalValue» типа REAL, класса «Внешняя» и «globalLevel» типа INT, класса «Внешняя».

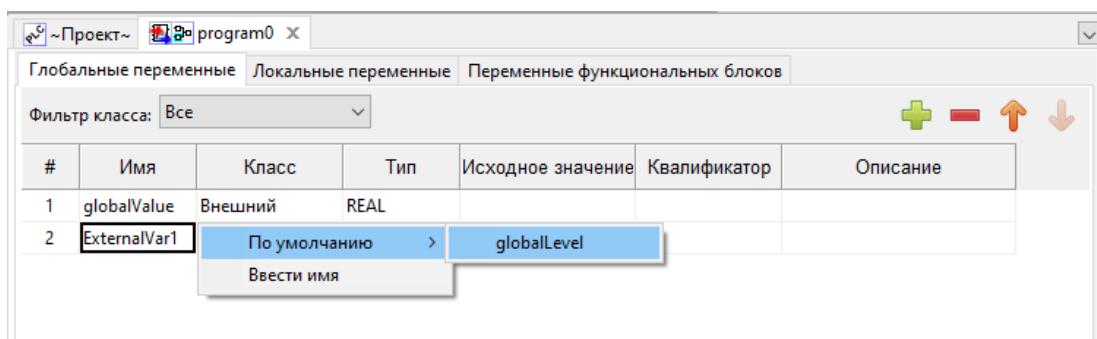


Рисунок 7.14 – Объявление в программе внешних переменных

Предполагается, что эти переменные уже определены глобальными переменными проекта в панели редактирования проекта (рисунок 7.15), как описывается в п. 7.2.1.

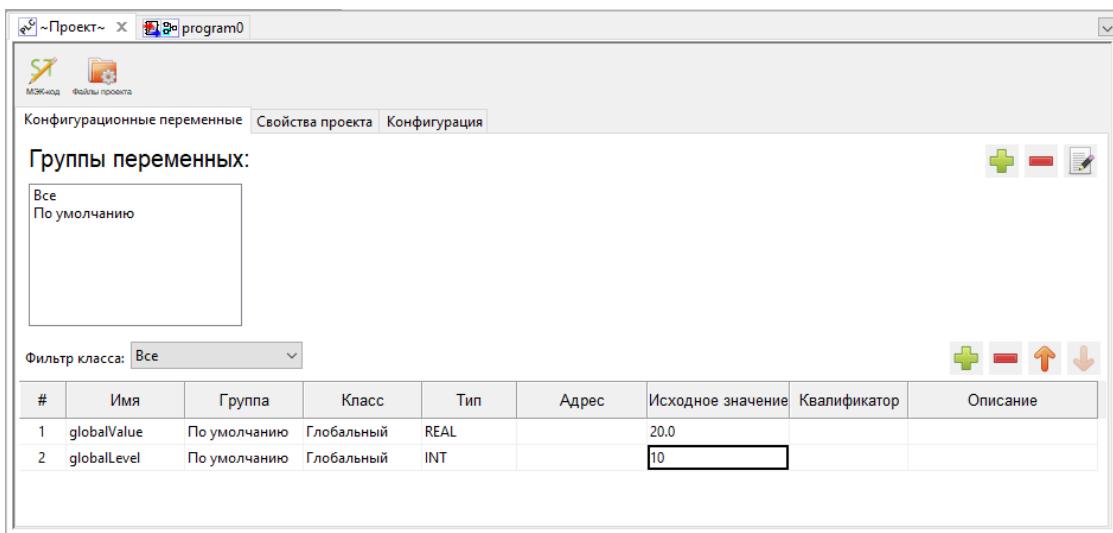


Рисунок 7.15 – Глобальные переменные проекта

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 7.16):

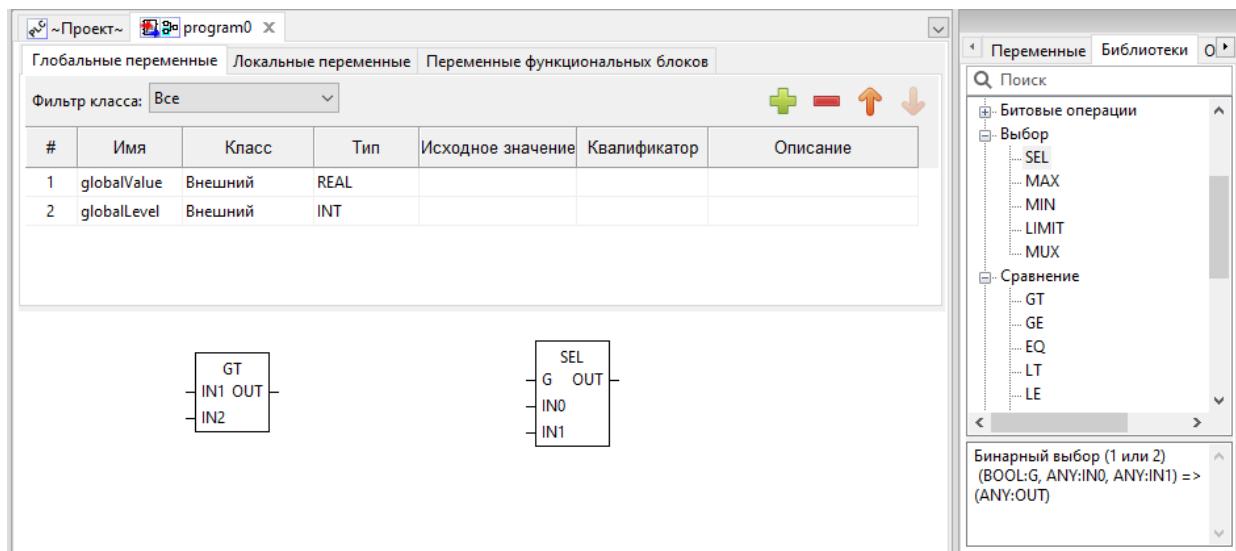


Рисунок 7.16 – Добавление двух функций на FBD диаграмму

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 7.17).

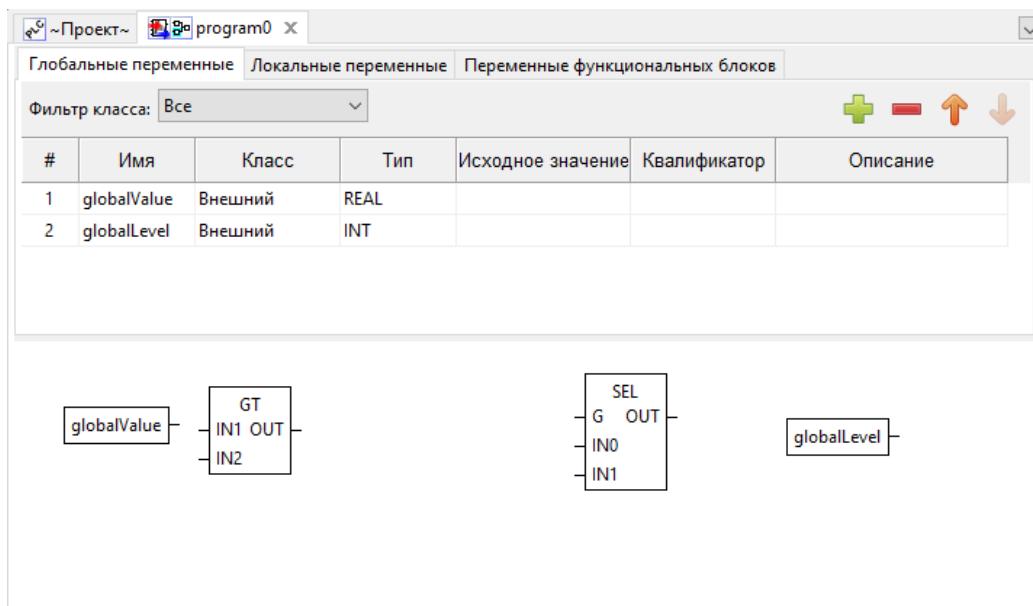


Рисунок 7.17 – Перенесённые переменные на FBD диаграмму

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 ниже (рисунок 7.18):

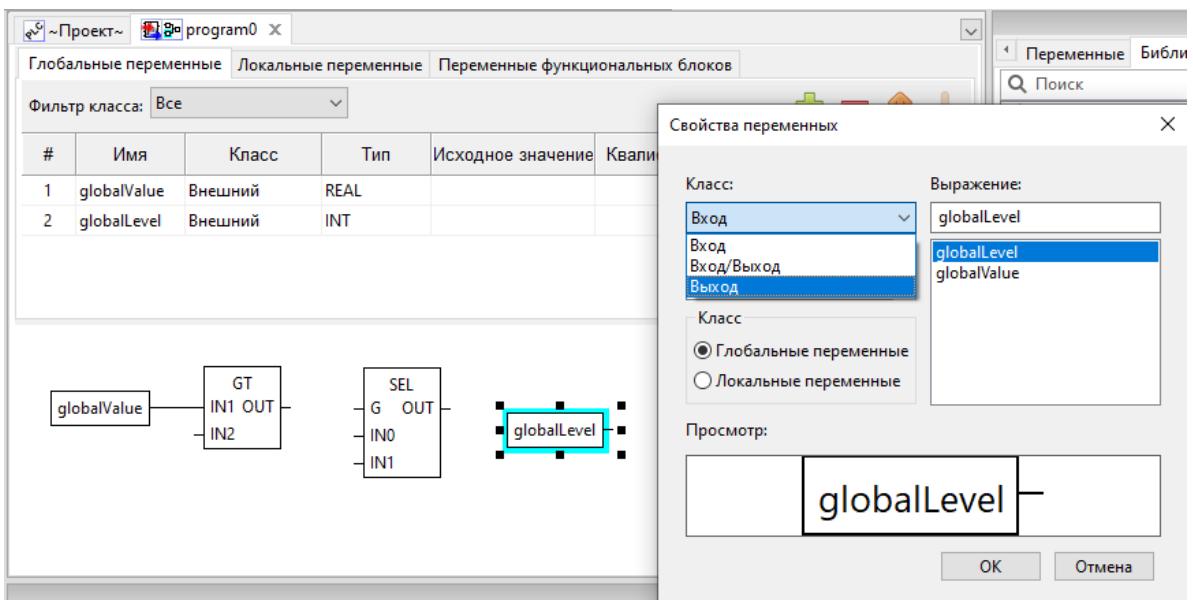


Рисунок 7.18 – Указание класса «Выходная» для переменной

После этого переменную «globalValue» можно соединить с выходом «OUT» функции «SEL» и выход «OUT» функции «GT» с входом «G» функции «SEL» (рисунок 7.19).

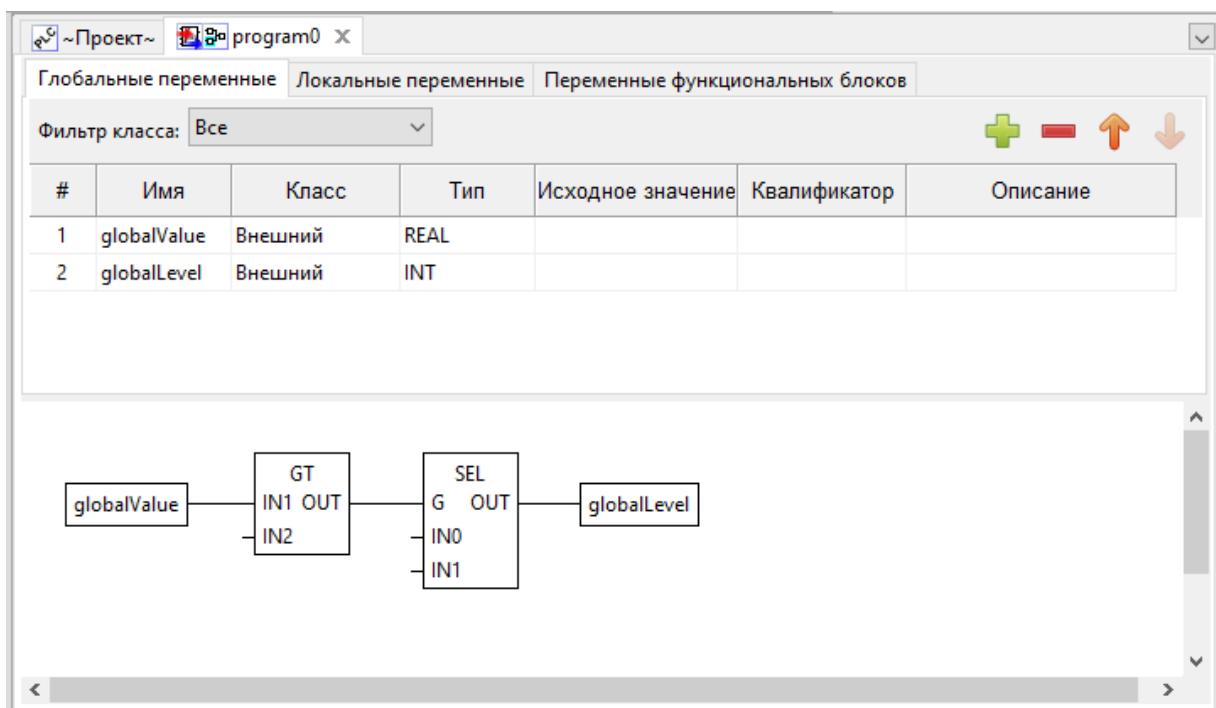


Рисунок 7.19 – Соединение входов и выходом функций на диаграмме FBD

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 7.20) в поле выражения пишется «10.0». Нажимается кнопка «OK».

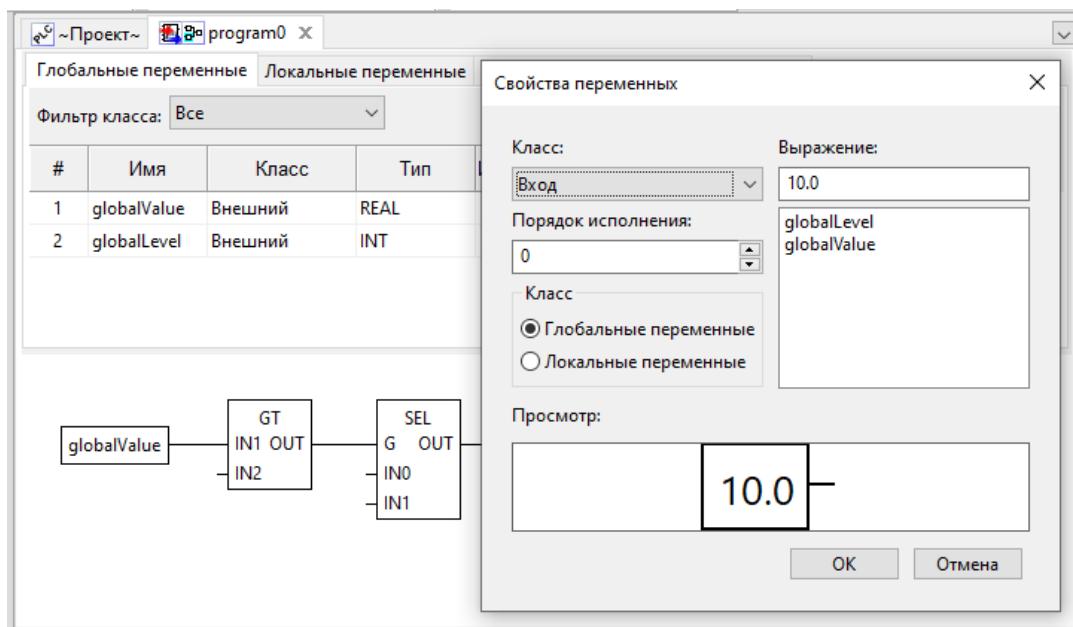


Рисунок 7.20 – Добавление переменной на FBD диаграмму

Добавленный литерал соединяется с входом «IN2» функции «GT», как показано на рисунке ниже (рисунок 7.21):

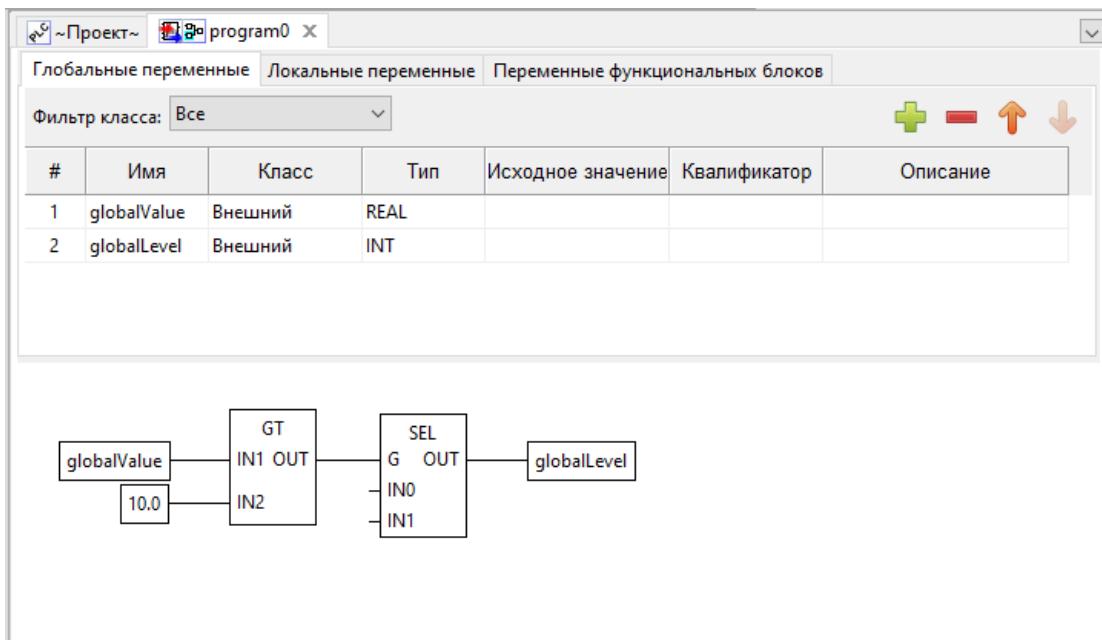


Рисунок 7.21 – Соединение добавленной переменной с выходом функции

Аналогичным образом создаются литералы со значениями 50 и 100 для выходов «IN0» и «IN1» функции «SEL» и соединяются с ними (рисунок 7.22).

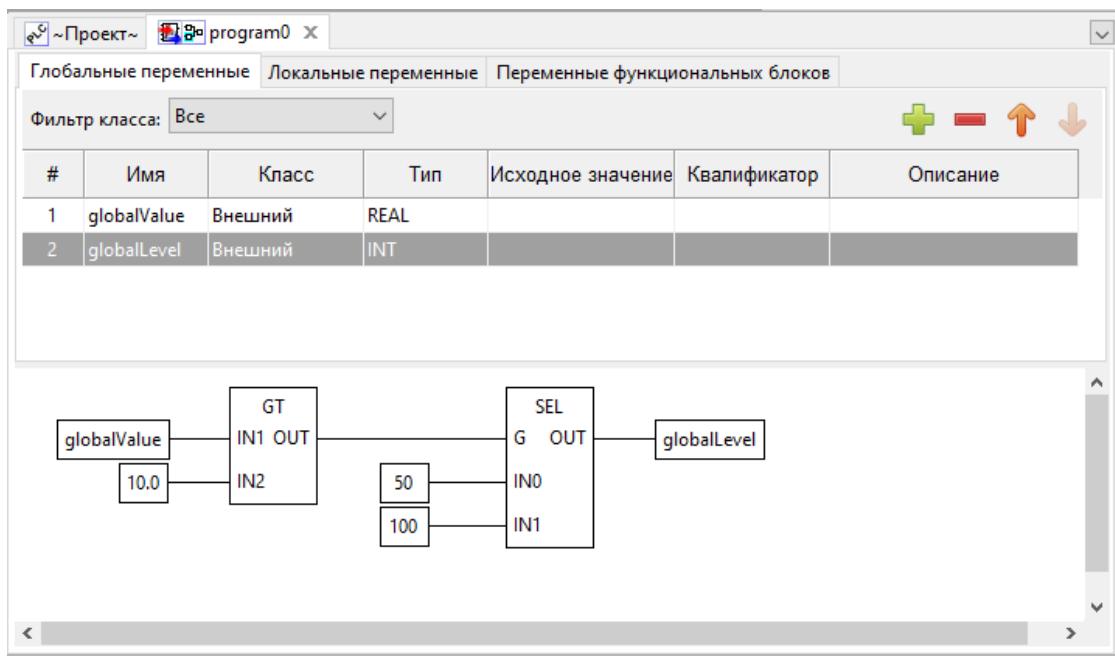


Рисунок 7.22 – Пример программного модуля, написанного на языке FBD

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 7.23):

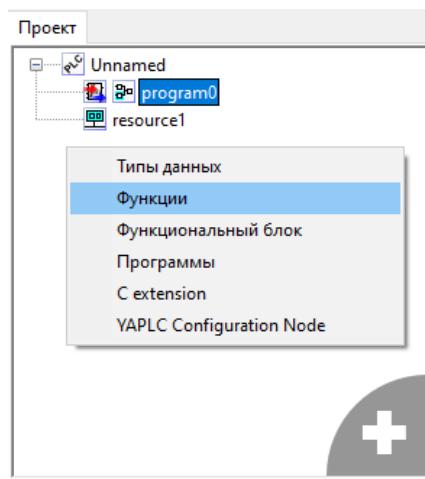


Рисунок 7.23 – Выбор в дереве проекта добавления функции

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 7.24). Язык SFC не может быть использован для описания алгоритма и логики работы функции.

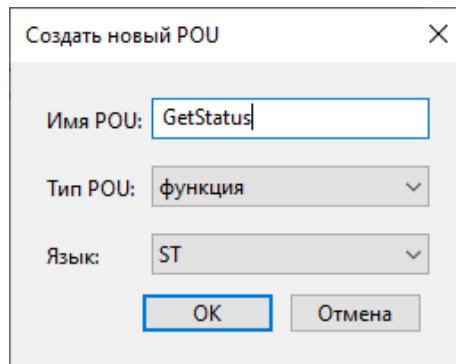


Рисунок 7.24 – Диалог добавления пользовательской функции

В появившейся панели редактирования функции выберем тип возвращаемого значения функции – STRING (рисунок 7.25).

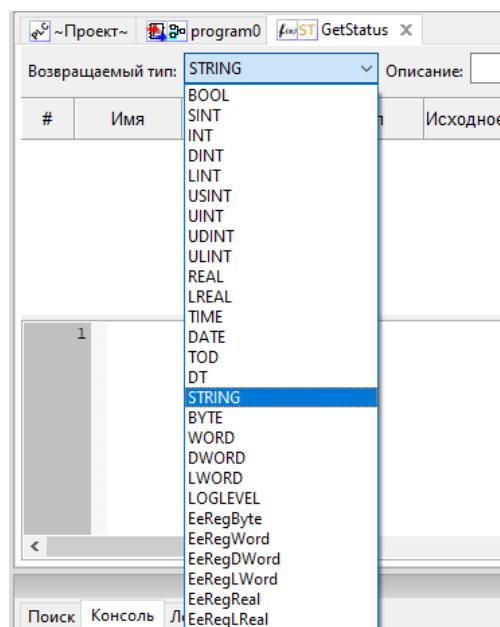


Рисунок 7.25 – Выбор значения, возвращаемого функцией

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 ниже (рисунок 7.26):

| # | Имя | Класс | Тип | Исходное значение | Квалификатор | Описание |
|---|-------|-------|-----|-------------------|--------------|----------|
| 1 | value | Вход | INT | | | |

```

1 CASE value OF
2   0..10: GetStatus := 'Low Level';
3   11..23: GetStatus := 'Middle Level';
4   11..23: GetStatus := 'High Level';
5   ELSE
6     GetStatus := 'Error';
7   END_CASE;
  
```

Рисунок 7.26 – 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 ниже (рисунок 7.27):

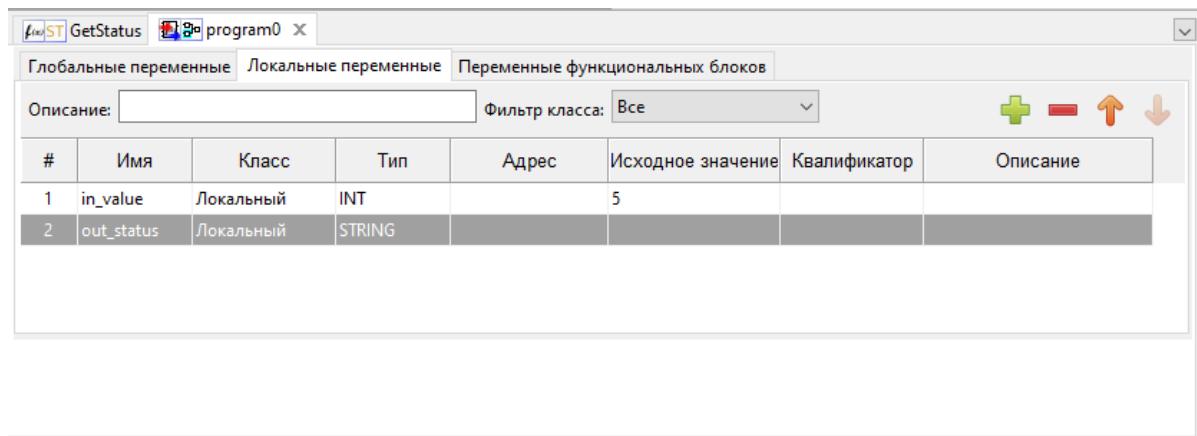


Рисунок 7.27 – Добавленный программный модуль «Программа» на языке FBD

На панели библиотеки функций и функциональных блоков в разделе «Пользовательские POU» 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 7.28).

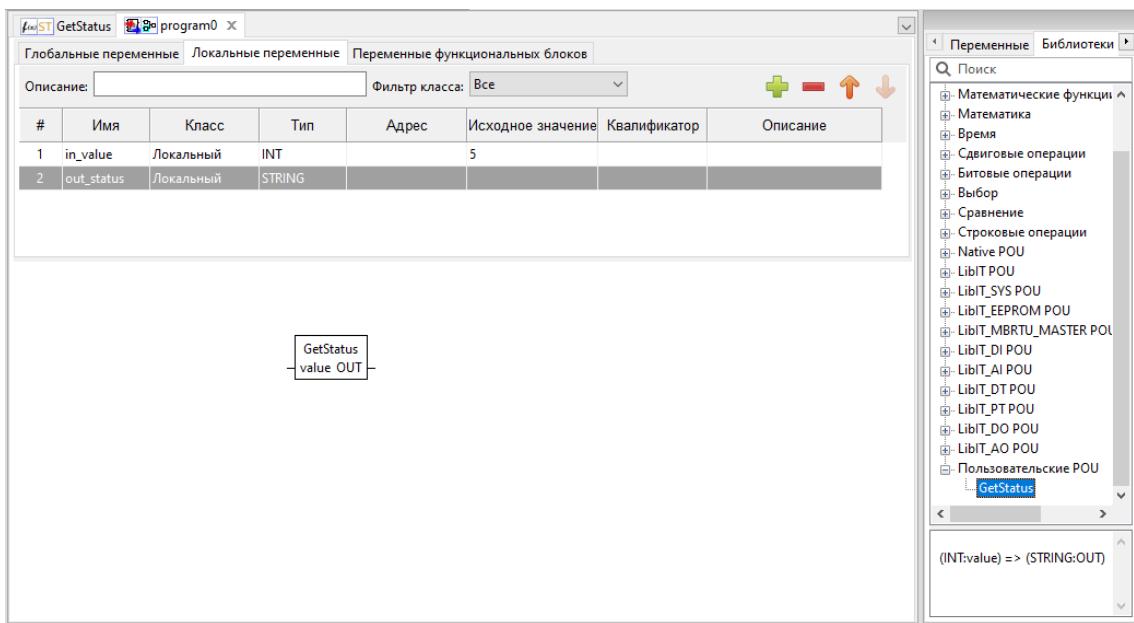


Рисунок 7.28 – Добавление на FBD диаграмму пользовательской функции

Аналогичным образом «перетаскиваются» переменные из панели переменных и констант в область редактирования FBD диаграммы (рисунок 7.29).

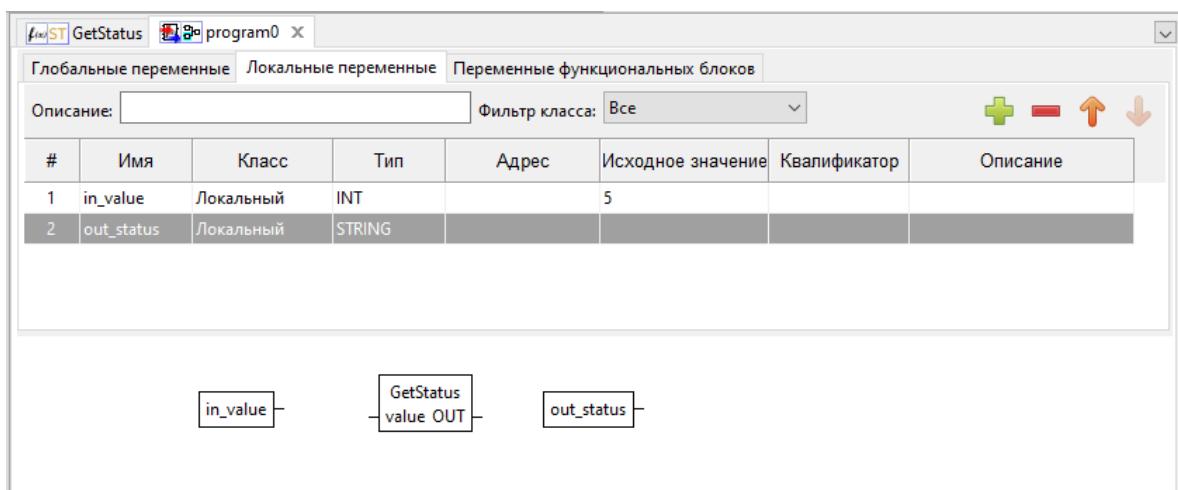


Рисунок 7.29 – Добавление на FBD диаграмму переменных из панели переменных и констант

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 7.30).

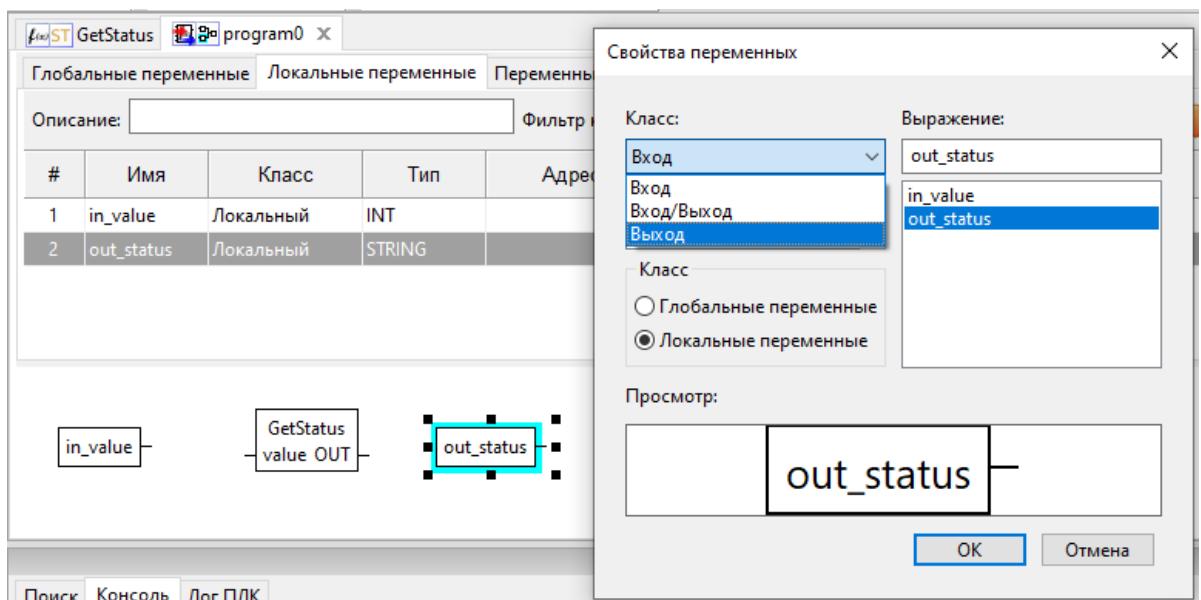


Рисунок 7.30 – Изменение «класса» переменной FBD диаграммы

Теперь переменные «in_value» и «out_status» можно соединить, соответственно, с «входом» и «выходом» функции «GetStatus» (рисунок 7.31).

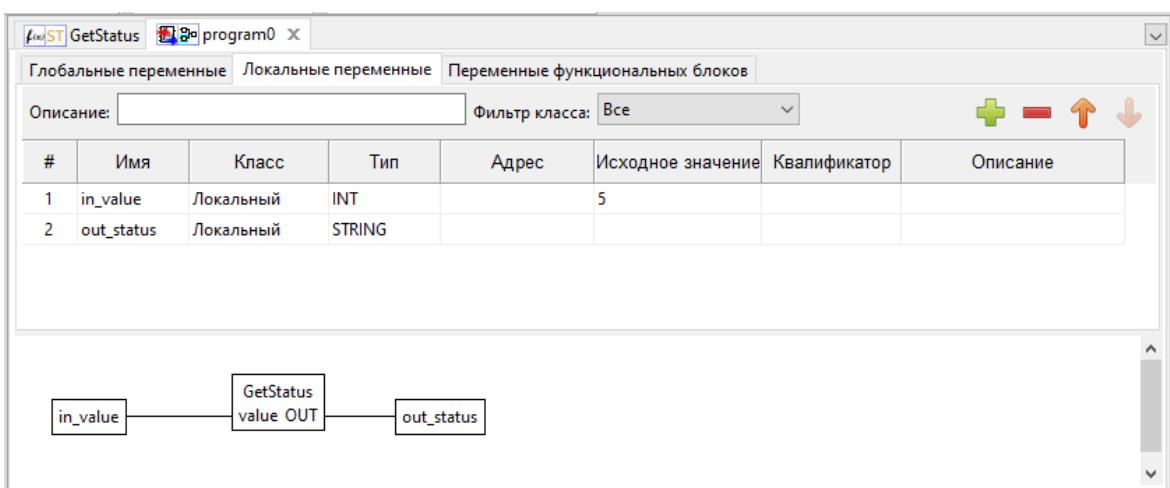


Рисунок 7.31 – Пример FBD диаграммы с использованием пользовательской функции

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» (рисунок 7.32), который будет считать площадь и периметр прямоугольника с заданными сторонами.

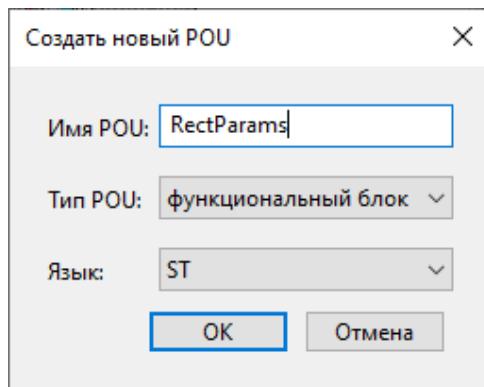


Рисунок 7.32 – Диалог добавления пользовательского функционального блока

В отличие от функции, функциональный блок может быть описан на любом языке стандарта IEC 61131-3, включая язык SFC. На рисунке ниже (рисунок 7.33) показана реализация данного функционального блока на языке ST.

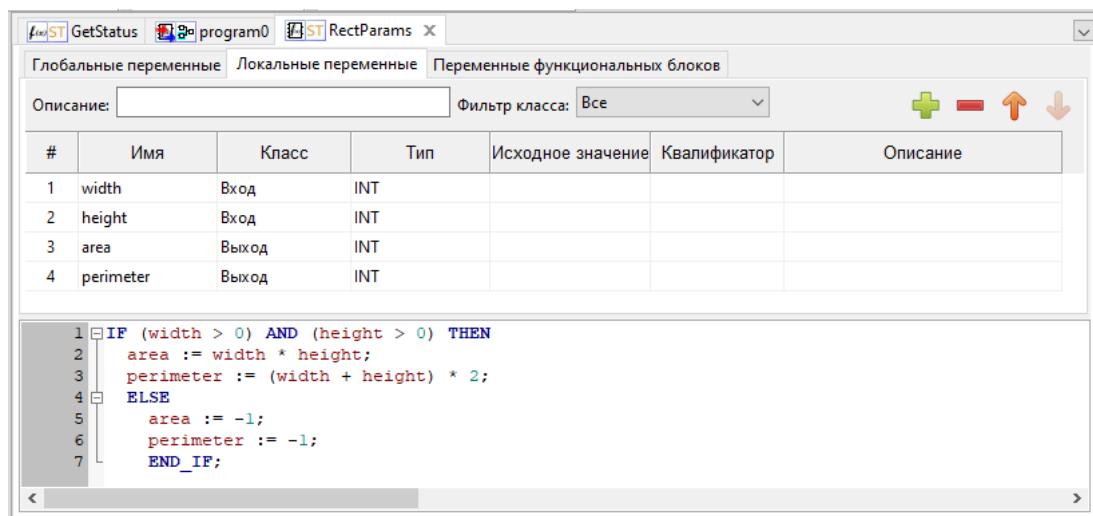


Рисунок 7.33 – Описание пользовательского функционального блока на языке ST

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 ниже (рисунок 7.34) показано использование созданного функционального блока «RectParams» в FBD диаграмме.

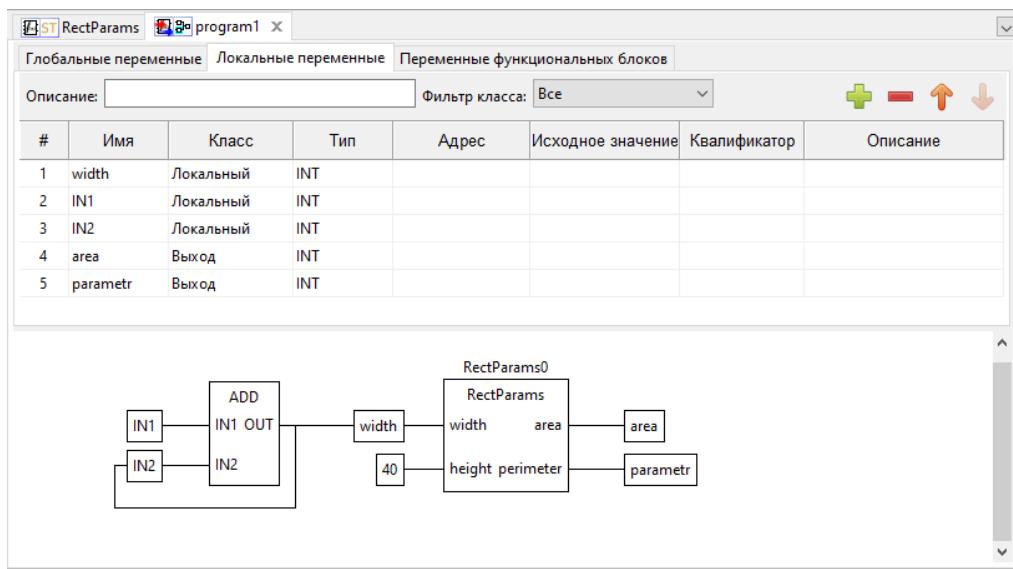


Рисунок 7.34 – Использование созданного функционального блока в FBD диаграмме

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 7.35), где эти добавленные программные модули уже используются, будет выдана ошибка.

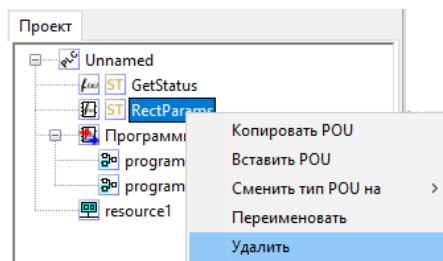


Рисунок 7.35 – Удаление функционального блока

Пример сообщения об ошибке удаления программного модуля приведен на ниже (рисунок 7.36).

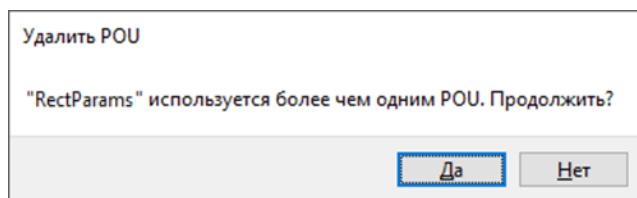


Рисунок 7.36 – Сообщение об ошибке при удалении функционального блока

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

определенной 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 7.37) выбранного ресурса с использованием кнопки «Добавить переменную», либо «Добавить переменные».

| # | Имя | Класс | Тип | Адрес | Исходное значение | Квалификатор | Описание |
|---|------------|------------|------|-------|-------------------|--------------|----------|
| 1 | GlobalVar0 | Глобальный | INT | | 10 | | |
| 2 | GlobalFlag | Глобальный | BOOL | | True | | |

Рисунок 7.37 – 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 (рисунок 7.38), показано, как в программном модуле «program0» добавлено две переменных класса «Внешний» с такими же именами, как и глобальными переменными, объявленные выше для ресурса.

| # | Имя | Класс | Тип | Исходное значение | Квалификатор | Описание |
|---|------------|---------|------|-------------------|--------------|----------|
| 1 | GlobalVar0 | Внешний | INT | | | |
| 2 | GlobalFlag | Внешний | BOOL | | | |

```

1 IF GlobalFlag = False THEN
2   GlobalVar0 := 150;
3 ELSE
4   GlobalVar0 := -150;
5 END_IF;

```

Рисунок 7.38 – Объявление и использование глобальных переменных ресурса

В редакторе ST кода написан алгоритм работы программного модуля с использованием указанных выше 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 7.39):

- «Циклическое» – выполнение программного модуля типа «Программа» через заданный интервал времени, указанный в поле «Интервал»;
- «Прерывание» – 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 «Источник».

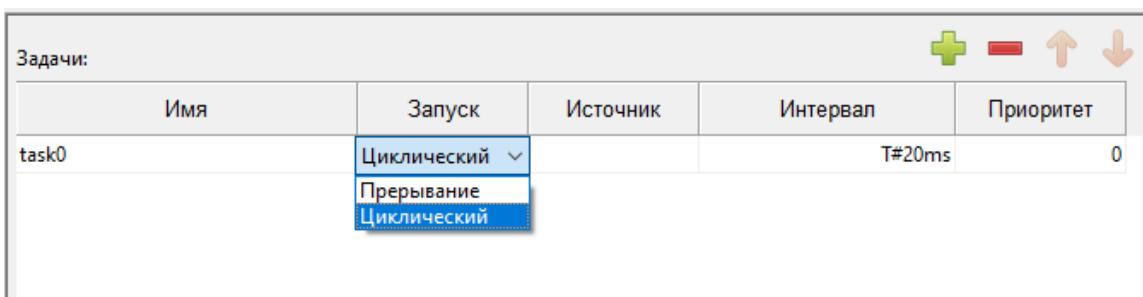


Рисунок 7.39 – Выбор типа выполнения задачи

В случае выбора типа выполнения «Циклическое», 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 7.40).

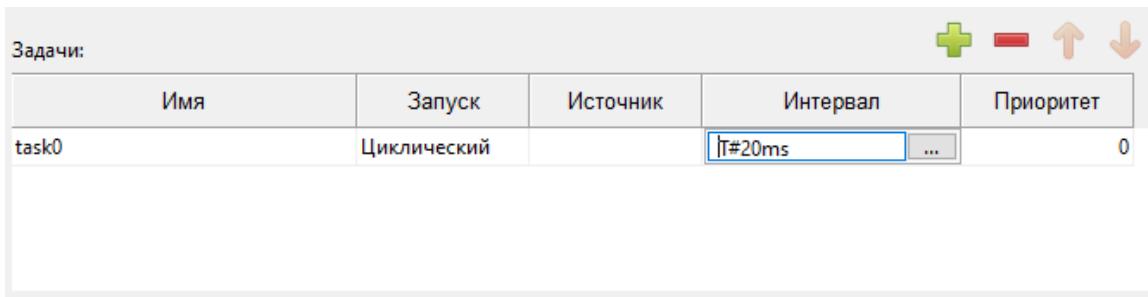


Рисунок 7.40 – Добавление задачи с циклическим режимом выполнения

Нажатие данной кнопки вызывает диалог «Редактировать длительность» (рисунок 7.41), в котором можно указать время, используя микросекунды, миллисекунды, секунды, минуты, часы и дни.

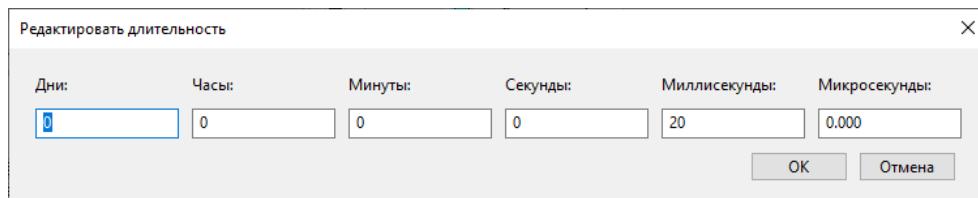


Рисунок 7.41 – Диалог редактирования продолжительности задачи

Завершение ввода времени кнопкой «OK» приводит к закрытию диалога и добавлению данного интервала времени в поле «Интервал» добавляемой задачи (рисунок 7.42).

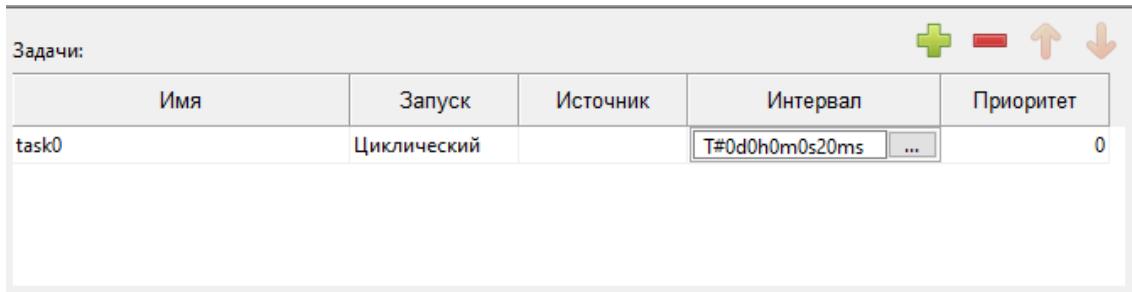


Рисунок 7.42 – Добавленный интервал выполнения

В случае выбора типа выполнения «Прерывание» в поле «Источник» 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 ниже (рисунок 7.43) выбирается переменная «globalFlag», определённая в данном ресурсе.

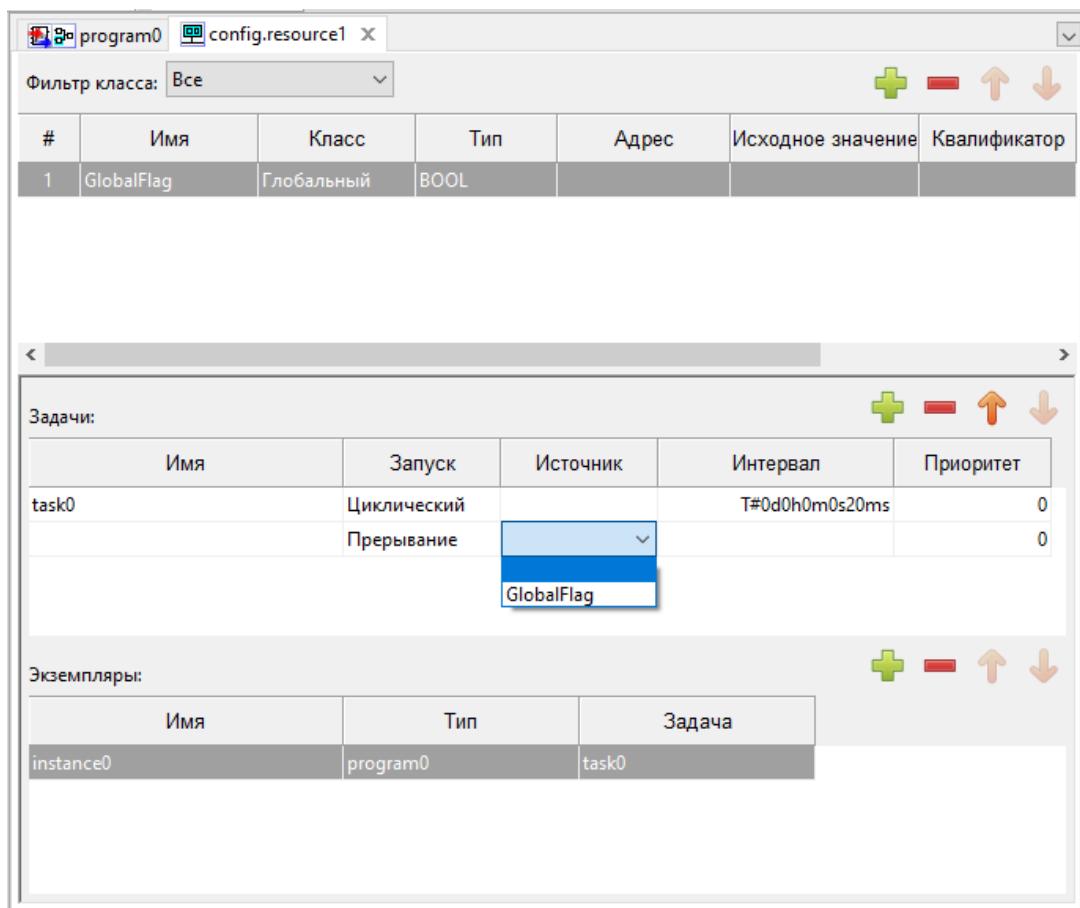


Рисунок 7.43 – Выбор переменной типа BOOL для определения условия выполнения задачи

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 «Циклический», 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 7.44).

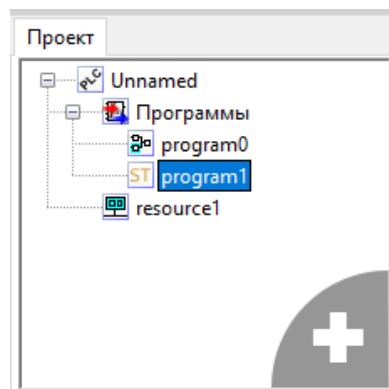


Рисунок 7.44 – Проект, содержащий два программными модулями типа «Программа»

Соответственно, при создании экземпляра в поле «Тип» оба этих программных модуля будут доступны (рисунок 7.45).

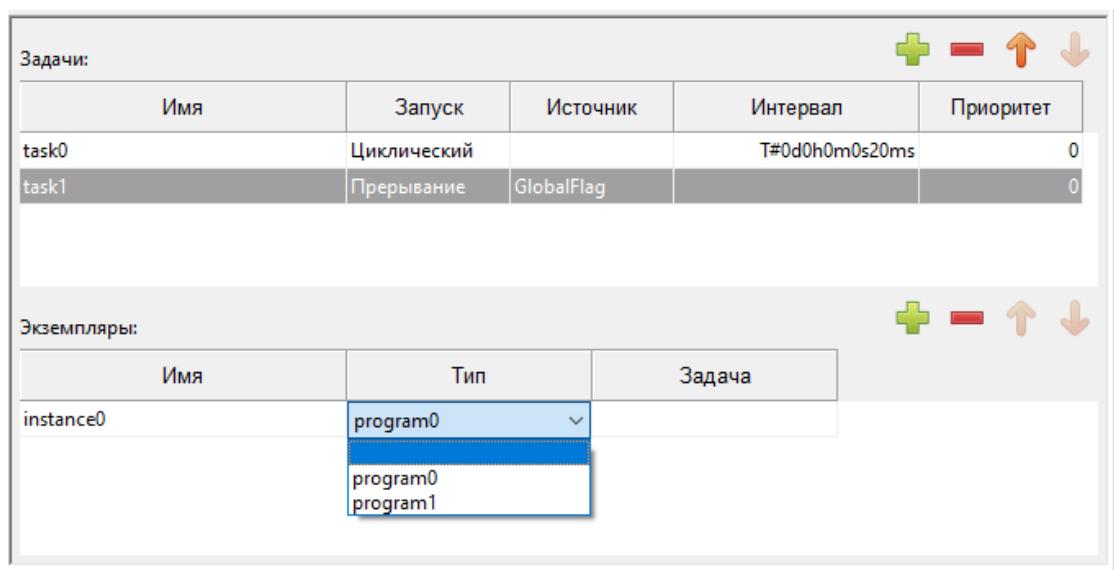


Рисунок 7.45 – Выбор программного модуля типа «Программа» для экземпляра

Аналогичным образом выбирается задача из списка, в котором будут отображены определённые ранее задачи (рисунок 7.46).

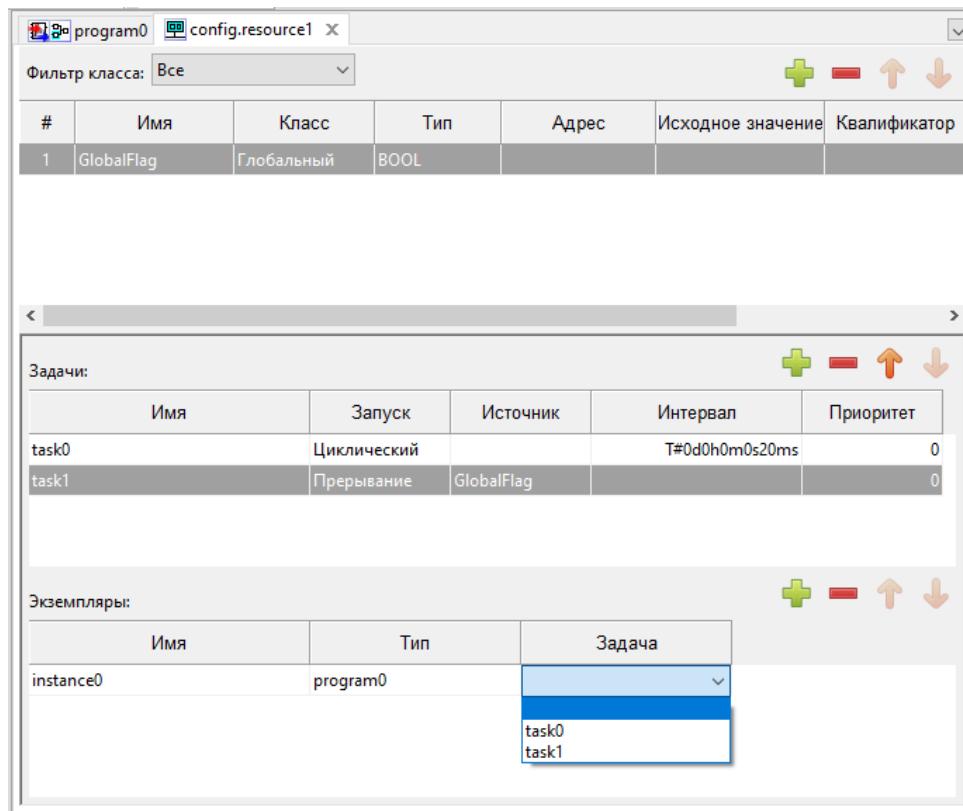


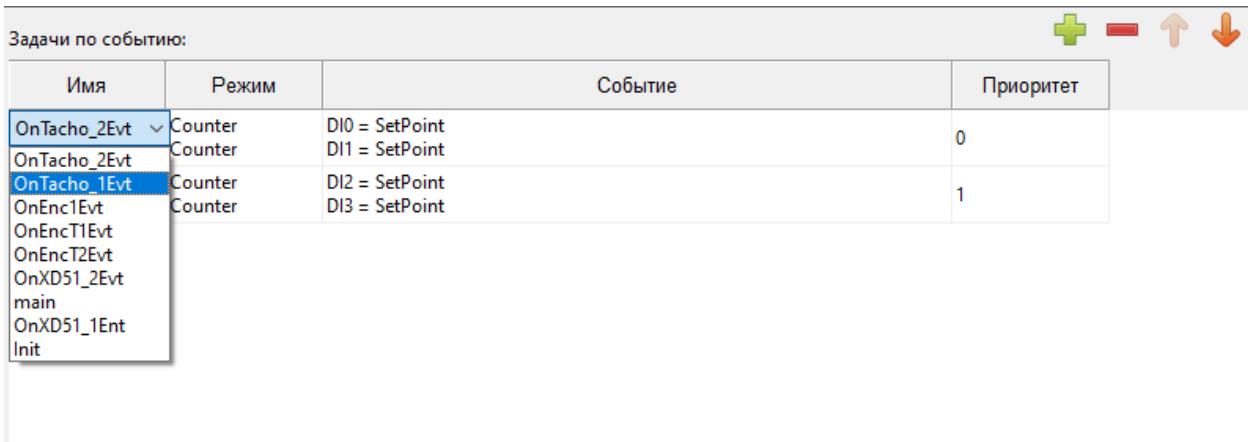
Рисунок 7.46 – Выбор задачи для экземпляра

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).

7.4.3 Задачи по событию.

Для создания задачи необходимо наличие как минимум одного программного модуля типа «Программа» в проекте.

После добавления задачи с помощью кнопки «Добавить» (данная кнопка аналогична кнопке «Добавить» на панели переменных и констант), необходимо выбрать её имя из предложенного списка (поле «Имя»), в данном списке отображаются программы созданные пользователем (рисунок 7.47).



| Задачи по событию: | | | |
|--------------------|---------|----------------|-----------|
| Имя | Режим | Событие | Приоритет |
| OnTacho_2Evt | Counter | DIO = SetPoint | 0 |
| OnTacho_2Evt | Counter | DI1 = SetPoint | |
| OnTacho_1Evt | Counter | DI2 = SetPoint | 1 |
| OnEnc1Evt | | DI3 = SetPoint | |
| OnEncT1Evt | | | |
| OnEncT2Evt | | | |
| OnXD51_2Evt | | | |
| main | | | |
| OnXD51_1Ent | | | |
| Init | | | |

Рисунок 7.47 – Добавление задачи по событию

При двойном клике по столбцам «Режим» или «Событие» открывается окно для конфигурации источника события (рисунок 7.48). Конфигурация источника происходит при помощи таблицы. Ниже описаны столбцы таблицы выбора источника:

- «Источник» – выбор источника сигнала. Это может быть дискретный вход, аналоговый вход, датчик температуры и т.д. Значение задается из выпадающего списка (рисунок 7.49).
- «Режим» – данный параметр зависит от выбранного входа. Здесь можно выбрать нужный режим входа, так же для некоторых входов этот параметр может оставаться пустым. Параметр задается из выпадающего списка.
- «Условие» – условие запуска задачи. Возможные условия:
 - Дискретные входы: «=» (равно).
 - Аналоговые входы: «<» или «>» (меньше или больше)
- «Значение» – значение, при котором запустится задача. Значение «SetPoint» означает, что оно задается в пользовательской программе при помощи функционального блока. Для дискретных входов существуют режимы работы по уставке. Когда уставка установлена, задача может запускаться. Для аналоговых входов доступна установка значения.

Конфигурация входов PLM для работы задач по событию происходит при помощи функциональных блоков.

- Для дискретных входов необходимо выбрать режим работы, а также сконфигурировать работу по уставке (блоки конфигурации счетчика или тахометра). Сработка задачи по событию происходит один раз при достижении уставки (SetPoint). Если вход настроен как нормальный, то событие будет срабатывать каждый раз при изменении уровня сигнала (фронт или спад). Для изменения условия запуска задачи по событию, необходимо переконфигурировать дискретный вход при помощи функциональных блоков.
- Для работы задач по событию от аналоговых входов, необходимо установить необходимый режим работы входа. Задача запустится только один раз при истинном условии установленного события. Повторный запуск повториться только после ложного условия события.

- Например: при условии AI0 > 3 задача запустится один раз. Дальнейшее увеличение уровня сигнала не приведет к вызову задачи. Повторно задача вызовется только тогда, когда уровень сигнала упадет до AI0 < 3 и снова поднимется до AI0 > 3.

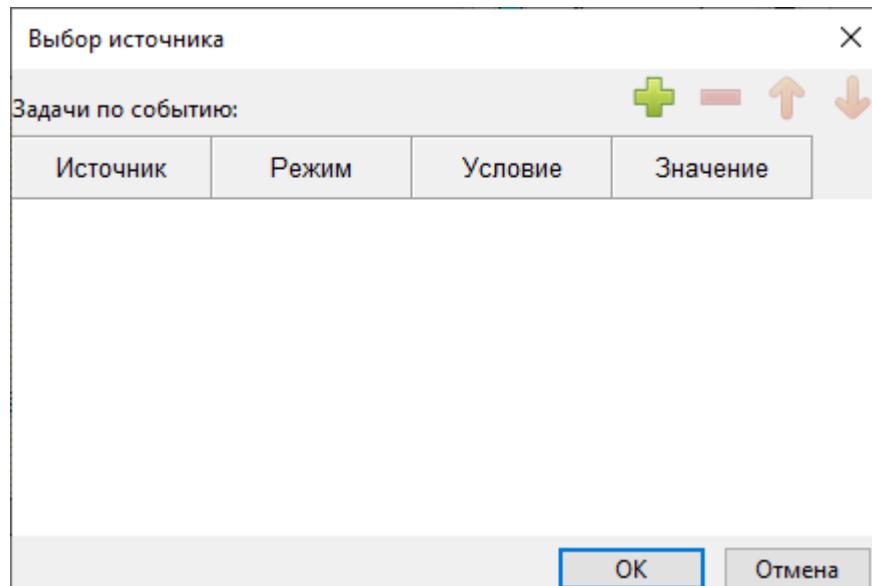


Рисунок 7.48 – Окно выбора источника события.

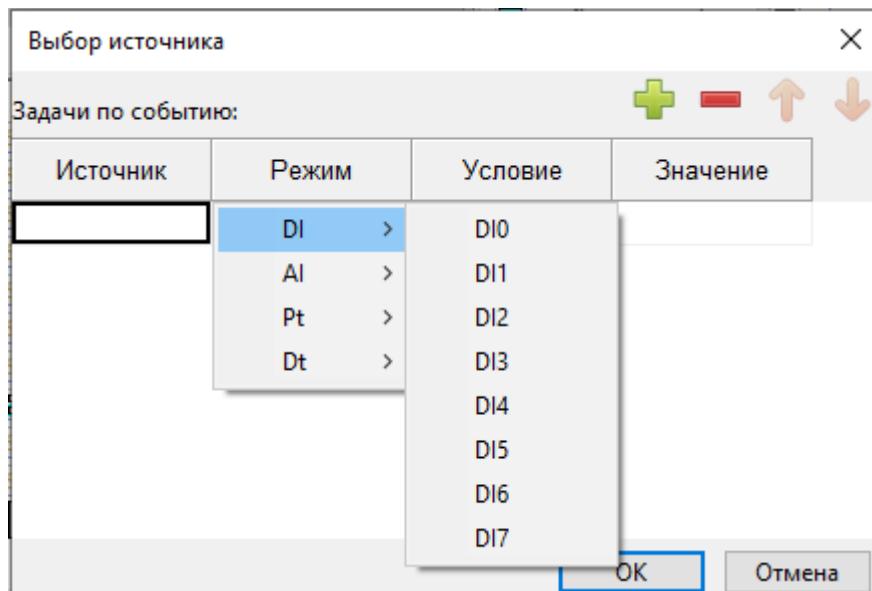


Рисунок 7.49 – выбор источника

Для правильной работы задач по событию необходимо сконфигурировать входы в соответствии с установленными значениями источников событий («Источник», «Режим»), если вход сконфигурирован неправильно, то задача по событию не запустится.

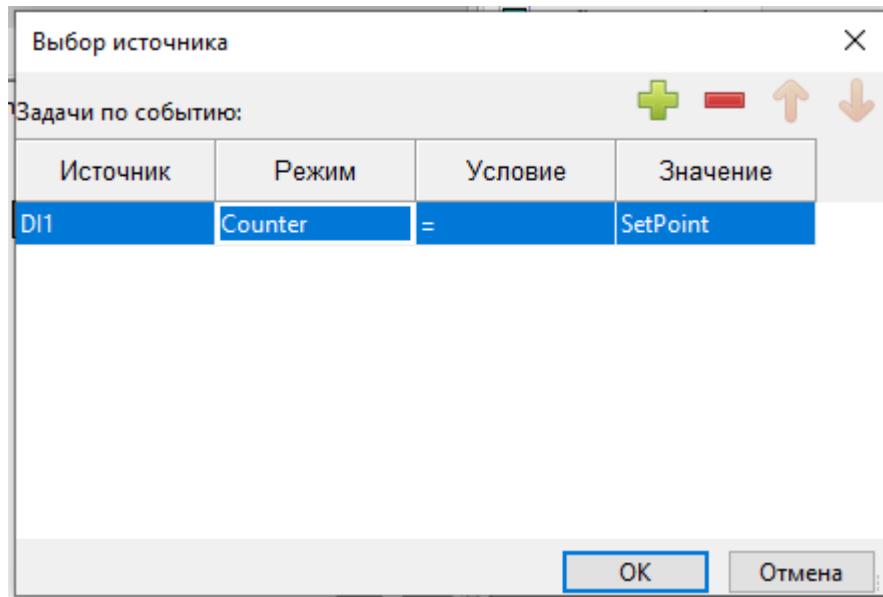


Рисунок 7.50 – Выбор типа выполнения задачи

Для одной задачи можно выбрать несколько источников.

После сохранения выбранных источников список задач будет выглядеть следующим образом (рисунок 7.51).

| Задачи по событию: | | | |
|--------------------|---------|----------------|-----------|
| Имя | Режим | Событие | Приоритет |
| OnEnc1Evt | Counter | DI1 = SetPoint | 0 |
| | | DI0 = SetPoint | |

Рисунок 7.51 – Задача по событию

Так как задачи запускается по событию, для увеличения скорости работы данные задачи не используют работу с памятью EEPROM, а так же с регистрами, описанными в п.6.

7.5 Типы данных

Добавление типа данных происходит выбором пункта «Типа данных» в меню дерева проекта (рисунок 7.52) в уже созданный проект, содержащий программный модуль типа «Программа» – «program0».

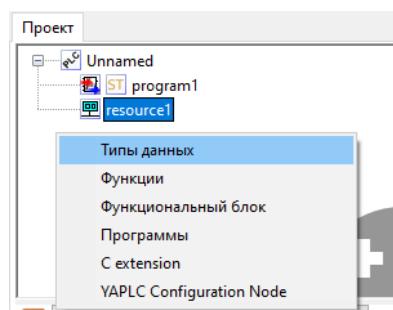


Рисунок 7.52 – Выбор пункта меню добавления пользовательского типа данных в дереве проекта

Будет создан массив типа BOOL размерностью 11 элементов. В дереве проекта появляется панель редактирования добавленного типа данных с именем «datatype0» (п. 5.9). В поле «Механизм создания типа» необходимо выбрать «Массив» и указать тип INT, как показано на рисунке ниже (рисунок 7.53):

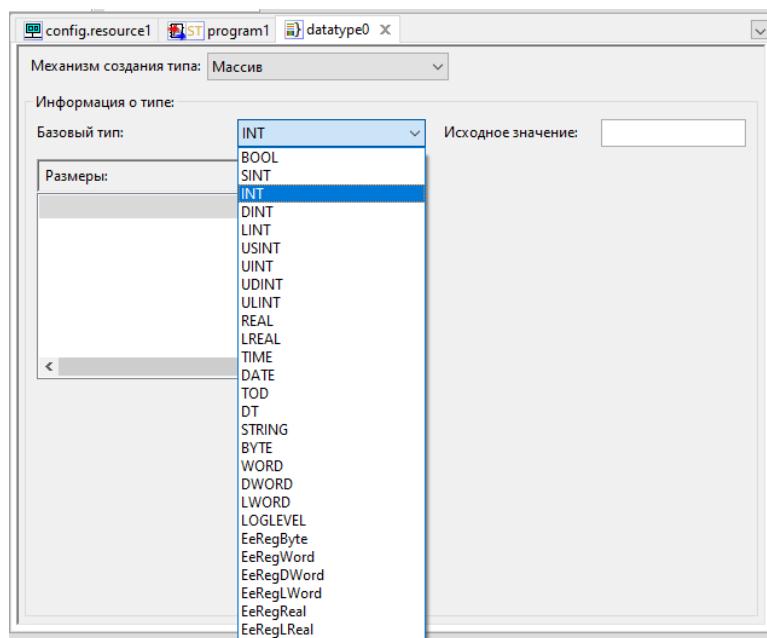


Рисунок 7.53 – Выбор базового типа для массива

С помощью кнопки «Добавить» создаётся поле для массива с указанием его размерности (рисунок 7.54) в соответствующем формате.

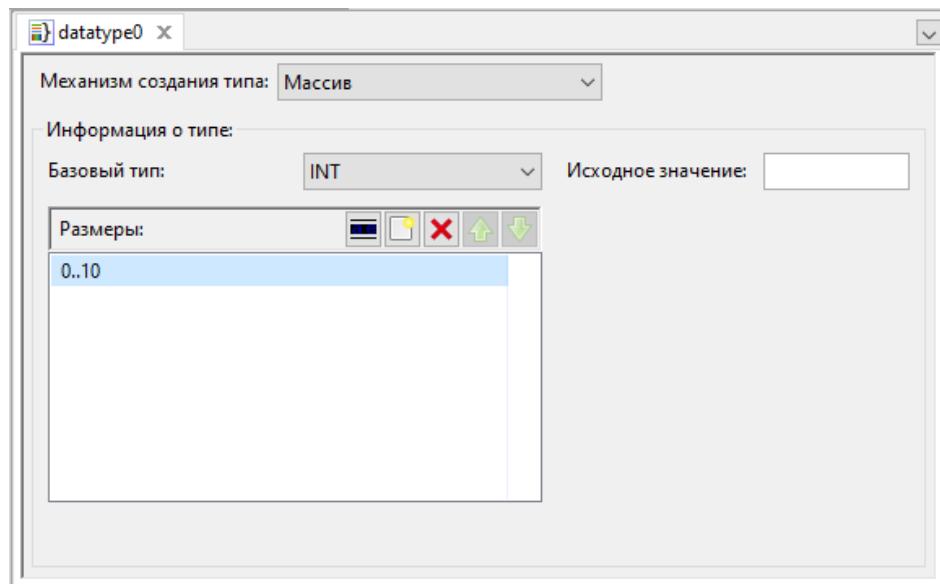


Рисунок 7.54 – Задание размерности для массива

После выполнения вышеперечисленных операций тип «datatype0» может быть использован для определения переменных в программных модулях, так же как и базовые типы данных (рисунок 7.55).

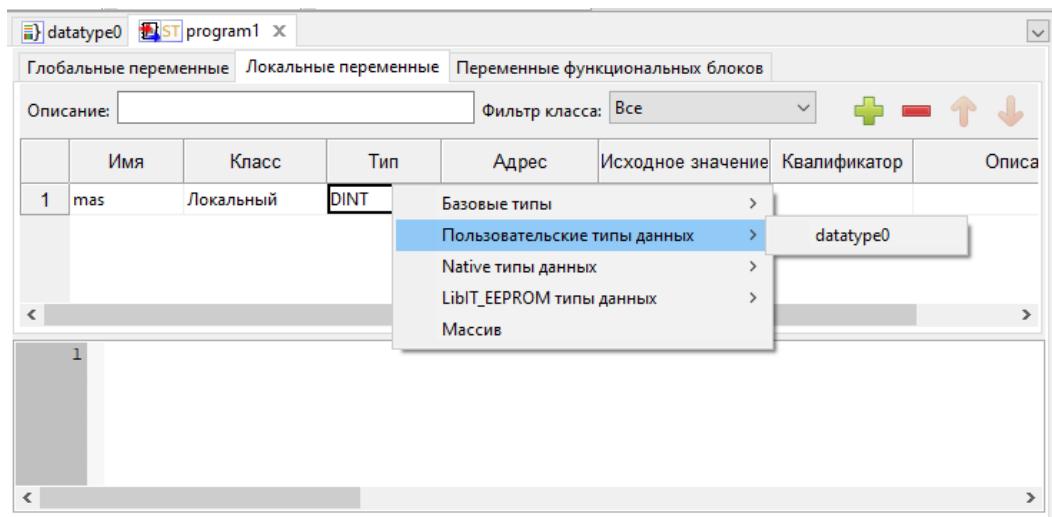


Рисунок 7.55 – Выбор добавленного типа данных в панели переменных и констант программного модуля

Добавим две переменные «temp_value» и «out_value» типа INT в панель переменных и констант. На рисунке ниже (рисунок 7.56) 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.

The screenshot shows the ST (Structured Text) code editor interface. At the top, there are tabs for 'Глобальные переменные' (Global variables), 'Локальные переменные' (Local variables), and 'Переменные функциональных блоков' (Functional block variables). Below these tabs is a search bar with 'Описание:' and a filter dropdown set to 'Все'. To the right are icons for adding (+), deleting (-), and navigating (up, down, left, right).

| # | Имя | Класс | Тип | Адрес | Исходное значение | Квалификатор | Описание |
|---|------------|-----------|-----------|-------|-------------------|--------------|----------|
| 1 | mas | Локальный | datatype0 | | | | |
| 2 | temp_value | Локальный | INT | | | | |
| 3 | out_value | Локальный | INT | | | | |

Below the table is a code editor window containing the following ST code:

```

1 temp_value := 10;
2 mas[0] := temp_value;
3 mas[1] := 20;
4 out_value := mas[0];

```

Рисунок 7.56 – Пример использования массива в коде на языке ST

7.6 Расширения

7.6.1 Язык Си

Добавление си расширения в проект происходит при помощи добавление элемента в дереве проекта (рисунок 7.57):

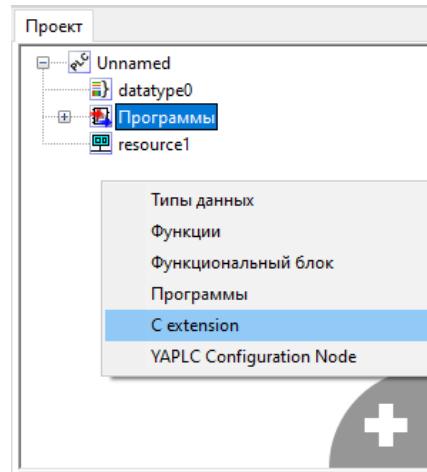


Рисунок 7.57 – Добавление расширения С extension

Окно редактора представлено на рисунке ниже (рисунок 7.58). Данное окно имеет следующие элементы:

- Список переменных. Данные переменные в проекте используются как глобальные.
- Текстовый редактор языка Си.

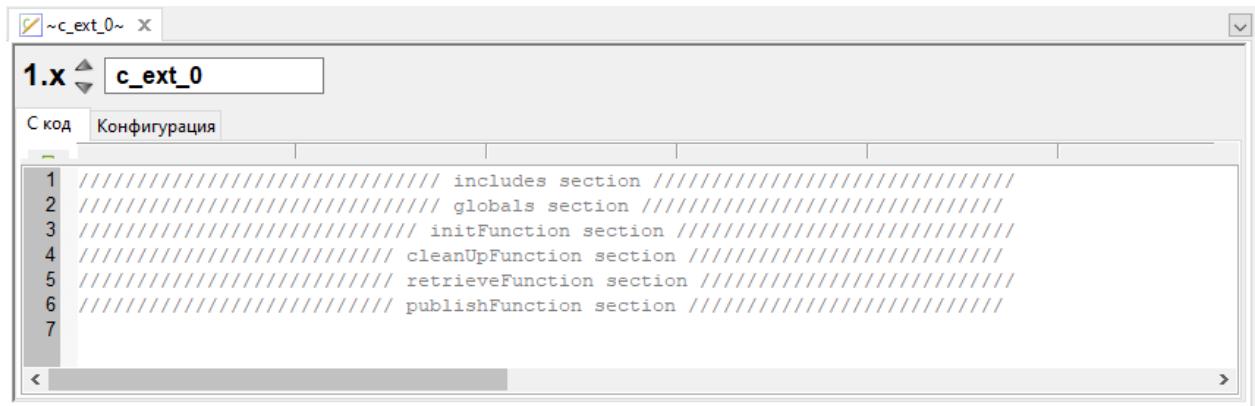


Рисунок 7.58 – Окно редактора расширения Си

Текстовый редактор представленный на рисунке разделен на следующие секции:

- *includes section* – Раздел подключения пользовательских библиотек. Данный раздел находится в начале генерируемого .с файла
- *globals section* – Глобальный раздел программы Си. Этот сектор предназначен для написания пользовательских функций.
- *initFunction section* – Данный раздел записывается в функцию __init_. Вызывается перед запуском программы ПЛК
- *cleanUpFunction section* – Данный раздел записывается в функцию __cleanup_. Вызывается после завершения программы ПЛК
- *retrieveFunction section* – Данный раздел записывается в функцию __retrieve_. Вызывается перед каждым циклом ПЛК (получает значения входных переменных)
- *publishFunction section* – Данный раздел записывается в функцию __publish_. Вызывается после каждого цикла ПЛК (устанавливает значения входных переменных).

Пример функции на Си представлен ниже (Рисунок 7.59):

The screenshot shows the ST3 software interface with the following details:

- Configuration Table:** A table titled "1.x" with "c_ext_0" selected. It has columns for "Имя" (Name), "Тип" (Type), "Исходное значение" (Initial value), "Описание" (Description), "При изменении" (On change), and "Настройки" (Settings). It lists four global variables: C_GLOBAL_VAR0, C_GLOBAL_VAR1, C_GLOBAL_VAR2, and C_GLOBAL_VAR3, all of type DINT with initial values of 0.
- C Code:** Below the table is a code editor window containing the following C code:

```

1 ////////////////////////////////////////////////////////////////// includes section //////////////////////////////
2 ////////////////////////////////////////////////////////////////// globals section //////////////////////////////
3 void C_Func_Increment()
4 {
5     C_GLOBAL_VAR0++;
6
7 }
8
9 void Func_Set_TStart()
10 {
11     C_GLOBAL_VAR1 += 50;
12 }
13
14 void Func_Set_T12()
15 {
16     C_GLOBAL_VAR2 += 100;
17 }
18
19 void Func_Set_TEnd()
20 {
21     C_GLOBAL_VAR3 -= 50;
22 }
23 ////////////////////////////////////////////////////////////////// initFunction section //////////////////////////////
24 Func_Set_TStart();
25
26 ////////////////////////////////////////////////////////////////// cleanUpFunction section //////////////////////////////
27 ////////////////////////////////////////////////////////////////// retrieveFunction section //////////////////////////////
28 Func_Set_T12();
29
30 ////////////////////////////////////////////////////////////////// publishFunction section //////////////////////////////
31 Func_Set_TEnd();

```

Рисунок 7.59 – Пример функций.

Для использования расширения необходимо создать программу или функциональный блок на языке ST, вызов из языка ST представлен ниже (рисунок 7.60).

The screenshot shows the ST3 software interface with the following details:

- Table of Global Variables:** A table titled "Глобальные переменные" with columns for "#", "Имя" (Name), "Класс" (Class), "Тип" (Type), "Исходное значение" (Initial value), "Квалификатор" (Qualifier), and "Описание" (Description). It lists four global variables: C_GLOBAL_VAR, all of type DINT with initial values of 0.
- C Code:** Below the table is a code editor window containing the following C code:

```

1 {{{
2 |   extern void C_Func_Increment();
3 |   C_Func_Increment();
4 }}}

```

Рисунок 7.60 – Вызов Си функции из 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 Veremiz).

Среда разработки Veremiz 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 8.1 – помечено цветом):



Рисунок 8.1 – Очистка директории сборки

Далее запускается сборка проекта, для чего необходимо нажать на кнопку «Сборка» в панели инструментов (рисунок 8.2 – помечено цветом):



Рисунок 8.2 – Сборка проекта

Результат сборки (успешно или неуспешно) выводится Консоль сообщений (рисунок 8.3):

```

Поск Консоль Лог ПЛК
Сборка запущена в D:\User1\workspace\plc1704-app\projects\test\test-di-fb\build
Генерация МЭК-61131 ST/IL/SFC кода ПЛК...
Компиляция МЭК-программы в С-код...
Экспорт локальных переменных...
0 -> Nothing to do
С-код успешно сгенерирован.
ПЛК:
[CC] plc_main.c -> plc_main.o
[CC] plc_debugger.c -> plc_debugger.o
ПЛК:
[CC] config.c -> config.o
[CC] resource1.c -> resource1.o
Линковка:
[CC] plc_main.o plc_debugger.o config.o resource1.o -> test-di-fb.elf
[OBJCOPY] test-di-fb.elf -> test-di-fb.elf.hex
Output size:
text     data      bss      dec      hex filename
15416    1124    4624    21164   52ac D:\User1\workspace\plc1704-app\projects\test\test-di-fb\build\test-di-fb.elf
Сборка прошла успешно.

```

Рисунок 8.3 – 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 рисунок 8.4 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:

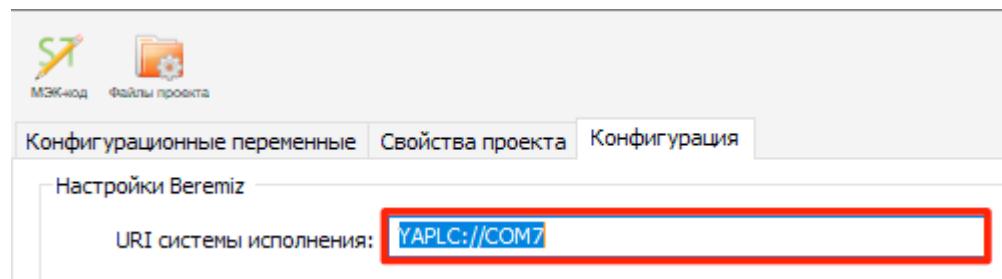


Рисунок 8.4 – URI адреса целевого устройства, подключенного к COM7

На рисунке ниже (рисунок 8.4) показан тип целевого устройства «plm2004», который выбирается из списка в панели настроек проекта:



Рисунок 8.5 – 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 8.5 – выделено цветом):



Рисунок 8.6 – Подключение к целевому устройству

Информация о результате подключения выводится в Консоль сообщений (рисунок 8.6) и в строке статуса (нижняя часть окна ICP Beremiz):

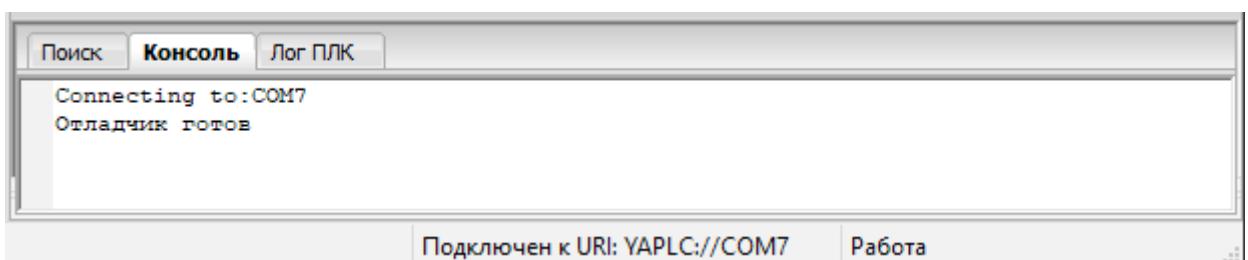


Рисунок 8.7 – Результат подключения в Консоли сообщений и строке статуса

После успешного подключения на панели инструментов будут доступны кнопки: «Остановить/Запустить исполнение прикладной программы на целевом устройстве», «Передать прикладную программу на целевое устройство» (рисунок 8.7):



Рисунок 8.8 – 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 8.9):

```
stm32flash 0.5
http://stm32flash.sourceforge.net/

Using Parser : Intel HEX
Interface serial_w32: 115200 8E1
Version      : 0x31
Option 1     : 0x00
Option 2     : 0x00
Device ID   : 0x0411 (STM32F2xxxx)
- RAM        : 128KiB (8192b reserved by bootloader)
- Flash      : 1024KiB (size first sector: 1x16384)
- Option RAM : 16b
- System RAM : 30KiB
Write to memory
Erasing memory
Wrote and verified address 0x08022200 (52.60%)
```

Рисунок 8.9 – Работа утилиты загрузчика прикладной программы

По окончании загрузки, утилита автоматически завершит свою работу (окно закроется) и в Консоль сообщений выведется информация о результате (рисунок 8.10). После успешной загрузки, прикладная программа автоматически будет запущена на целевом устройстве.

```
ПЛК Stopped
Передача успешно выполнена.
ПЛК запущен на выполнение
```

Рисунок 8.10 – 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 ниже (рисунок 8.11), добавить переменных панель отладки (п. 5.14):

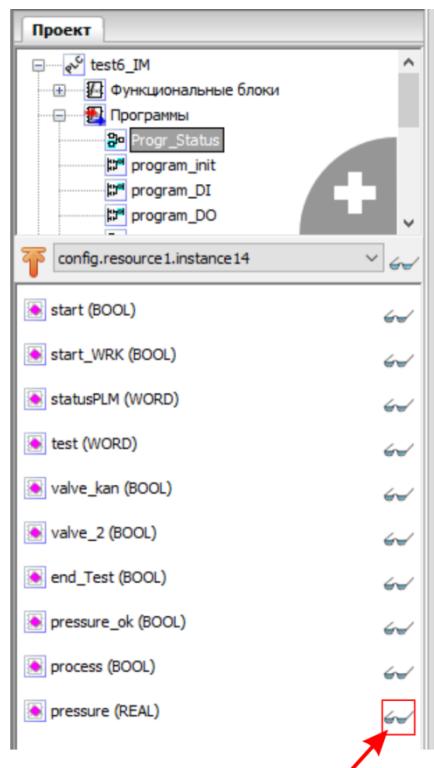


Рисунок 8.11 – Добавление переменных для отладки из панели экземпляров

На панели отладки сразу же отобразятся текущие значения добавленных переменных (рисунок 8.12).

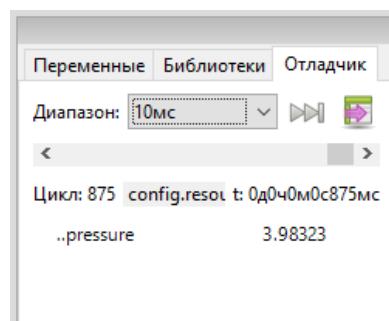


Рисунок 8.12 – Панель отладки значений переменных

Для того чтобы изменить значение переменной во время исполнения проекта, необходимо кликнуть левой клавишей мыши на значок 1 интересующей переменной (рисунок 8.13).

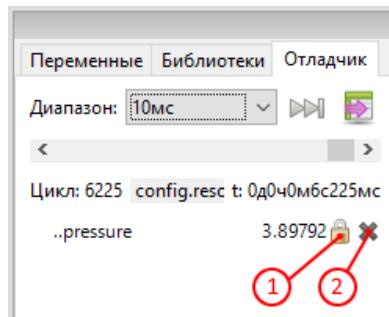


Рисунок 8.13 – Всплывающее меню манипуляций со значением переменной

Появится диалог, показанный на рисунке рисунок 8.14.а.

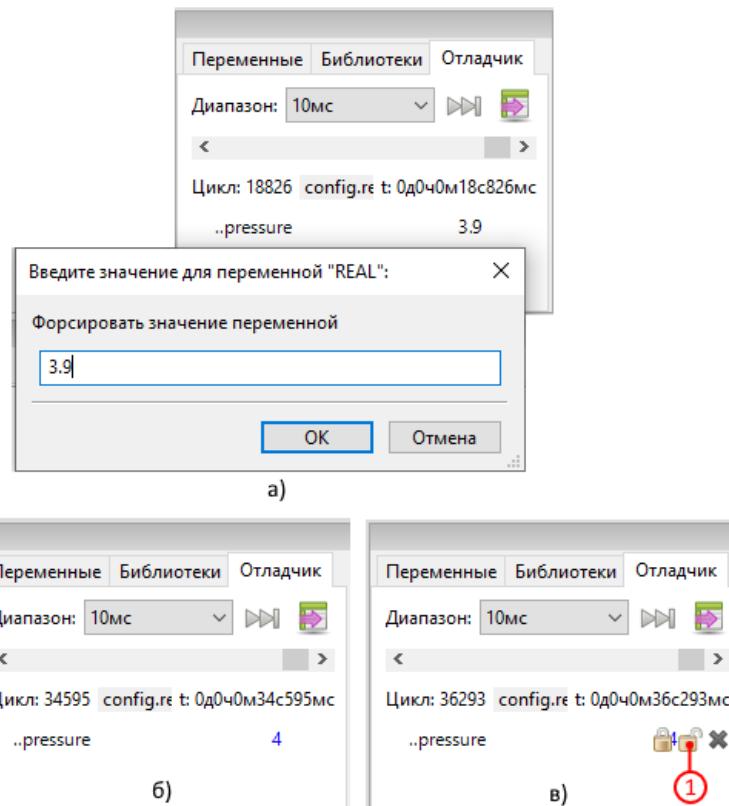


Рисунок 8.14 – Установка значения переменной во время отладки

Для корректного изменения, вводимое значение должно соответствовать типу переменной, иначе будет выведено сообщение об ошибке (Рисунок 8.15).

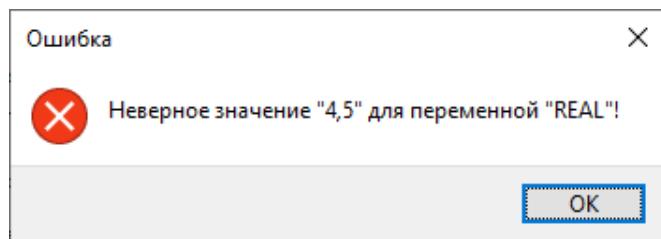


Рисунок 8.15 – Ошибка при вводе недопустимого значения изменяемой переменной в режиме отладки

После изменения значения переменной она переводится в отладочный режим (в панели отладчика она при этом отображается синим цветом (рисунок 8.14.б)), т.е. ее значение не может быть изменено действиями из отлаживаемой программы, а только вручную.

Для того чтобы вернуть переменную под управление программы пользователя, ее нужно освободить. Для этого надо навести курсор мыши на строку переменной в панели отладчика (рисунок 8.14.в) и кликнуть левой кнопкой мыши по значку 1 (рисунок 8.14.в).

Для того чтобы удалить переменную из списка отладчика нужно кликнуть левой кнопкой мыши по значку 2 (Рисунок 8.13).

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 (рисунок 8.16 – выделено красным цветом) для экземпляра программы, написанной на одном из графических языков, открывается вкладка с панелью диаграммы в режиме отладки.

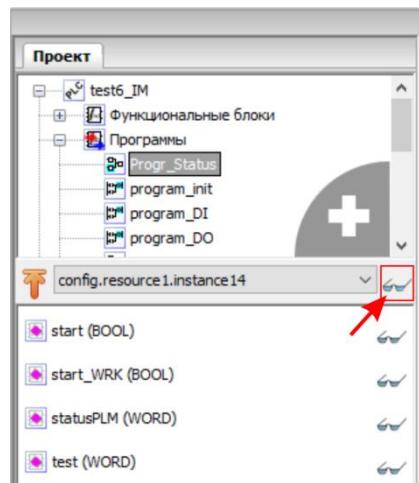


Рисунок 8.16 – Панель экземпляров проекта

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 8.17).

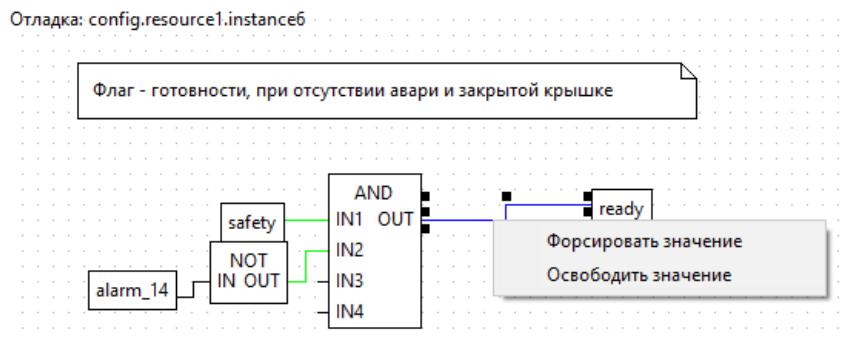


Рисунок 8.17 – Пример отладки FBD диаграммы

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 становится исходного цвета.

Кроме этого, значения переменных во время выполнения FBD программы могут быть изменены аналогично отладке программы ST.

8.5 Отладка LD диаграммы

Отладка LD диаграммы осуществляется аналогично отладке FBD диаграммы. Для вызова всплывающего меню (рисунок 8.18), в котором можно установить желаемое значение для контакта или катушки необходимо нажать правую клавишу мыши.

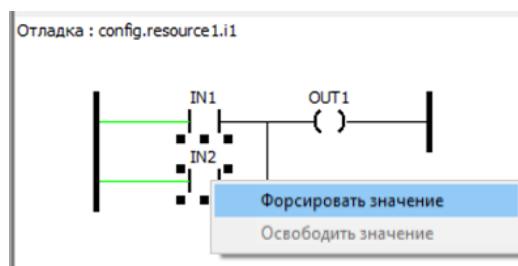


Рисунок 8.18 – Пример отладки LD диаграммы

Появится диалог (рисунок 8.19), в котором нужно ввести значение типа BOOL: TRUE – контакт «ON», FALSE – контакт «OFF».

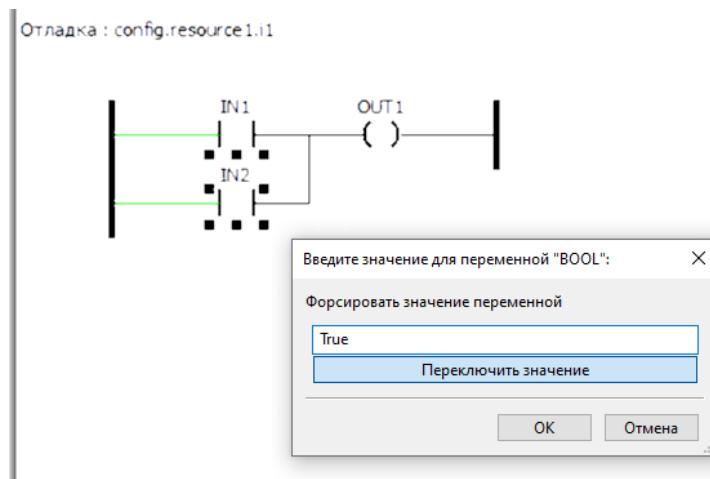


Рисунок 8.19 – Отладка LD диаграммы

После установки в режиме отладки, для данного примера, контакта «IN2» (рисунок 8.20), в значение TRUE катушка «OUT1» приняла значение TRUE, т.к. данная схема представляет собой реализацию «Логического ИЛИ» для «IN1» и «IN2».

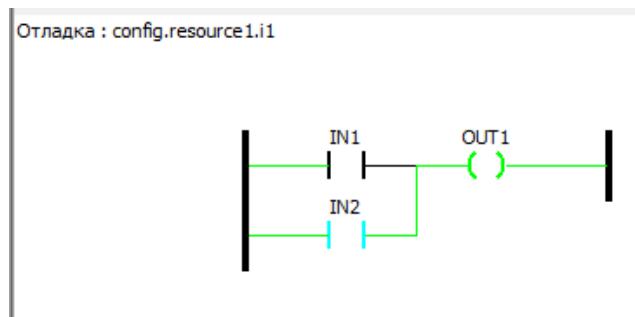


Рисунок 8.20 – Отладка LD диаграммы

Кроме этого, значения переменных во время выполнения FBD программы могут быть изменены аналогично отладке программы ST.

8.6 Отладка SFC диаграммы

Отладка SFC диаграммы происходит аналогично отладке диаграмм FBD и LD. С помощью всплывающего меню (рисунок 8.21), есть возможность устанавливать активность для шагов и переходов.

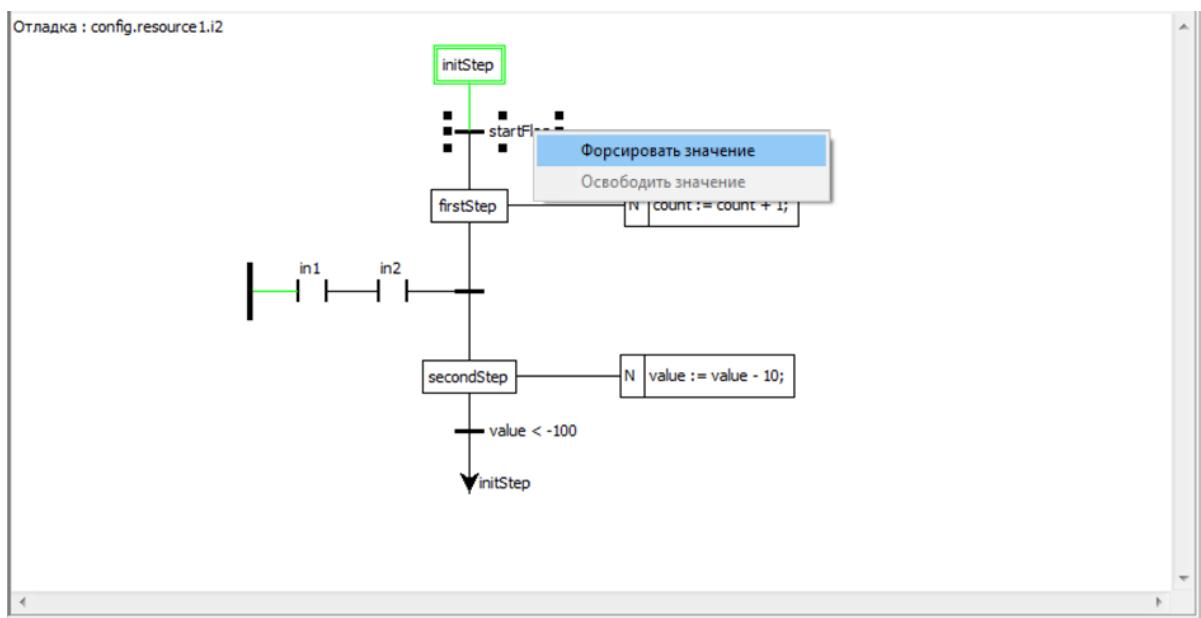


Рисунок 8.21 – Пример отладки SFC диаграммы

На рисунке ниже (рисунок 8.22) показано, как устанавливается значение (после выбора «Форсировать значение», появится диалог) TRUE для шага «firstStep».

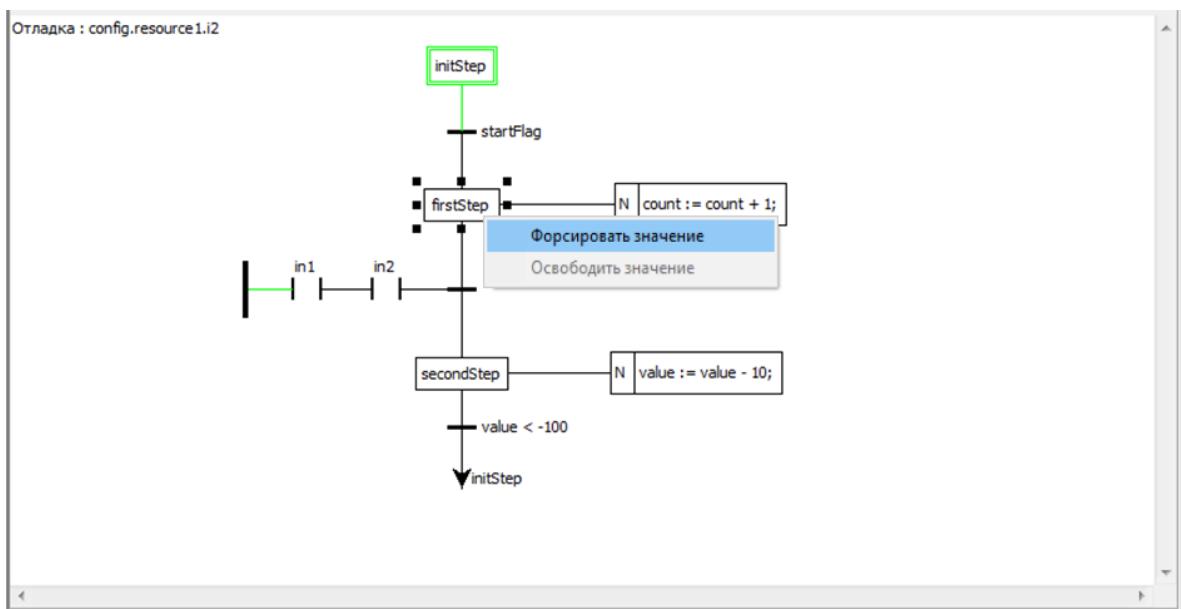


Рисунок 8.22 – Пример отладки SFC диаграммы

После установки значения шага в TRUE с помощью режима отладки, шаг будет выделен голубым цветом. Как можно заметить по рисунку ниже (рисунок 8.23), 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.
```

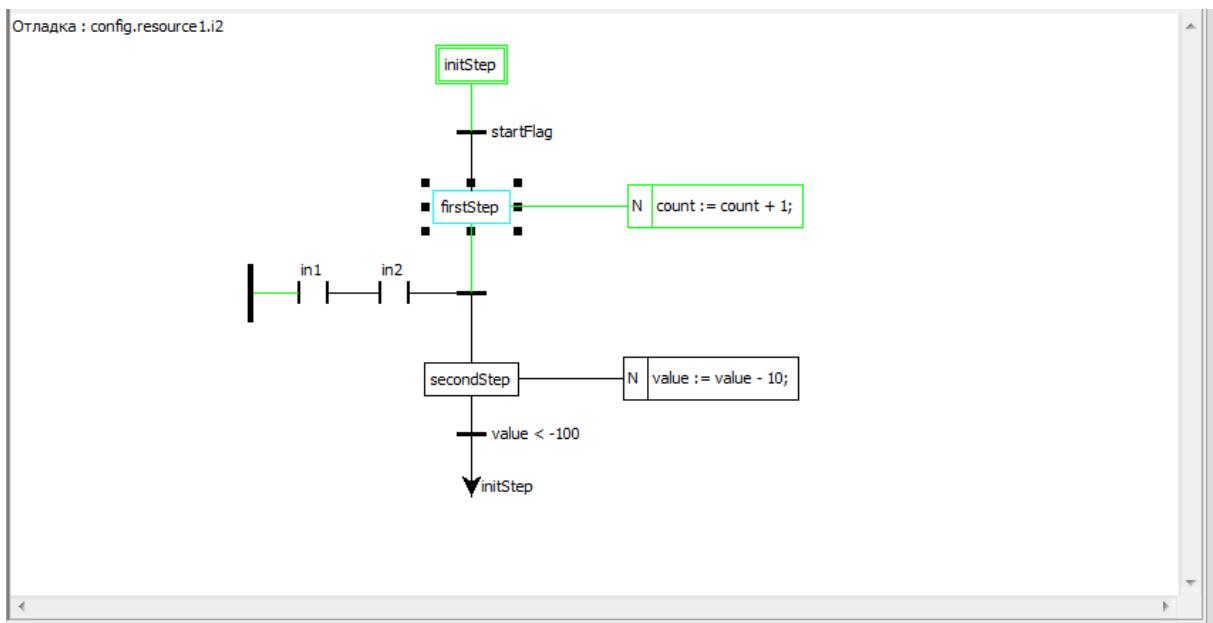


Рисунок 8.23 – Пример отладки SFC диаграммы

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

Кроме этого, значения переменных во время выполнения FBD программы могут быть изменены аналогично отладке программы ST.

8.7 График изменения значения переменной

Среда разработки Begetiz так же позволяет отображать в виде графика изменение значения переменной в режиме отладки. Для вывода панели с графиком, необходимо двойным кликом левой кнопки мыши по строке переменной в отладчике перевести ее в режим отображения графика изменения значения (рисунок 8.24):

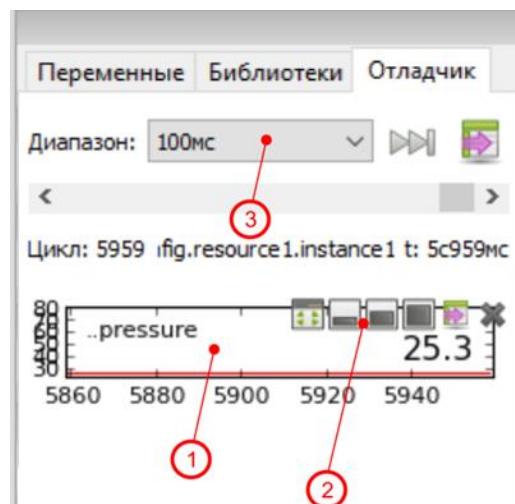


Рисунок 8.24 – Выбор кнопки на панели экземпляров для отображения графика изменения значения переменной

В поле 1 отображается график изменения значения переменной. С помощью выбора значения поля 3 можно задать период обновления значения переменной.

С помощью элементов управления 3, можно изменить вид отображаемого поля 1 (рисунок 8.25).

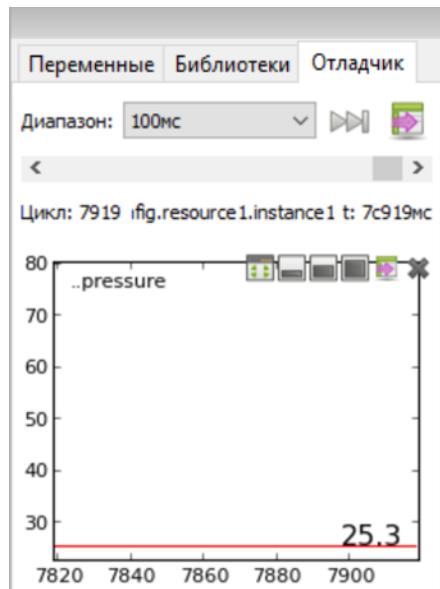


Рисунок 8.25 – График изменения переменной

Двойной клик левой кнопкой мыши по полю 1, вернет режим отображения переменной в текстовую форму.

График, изображенный на рисунке выше (рисунок 8.25), соответствует изменению переменной из программного модуля (рисунок 8.26):

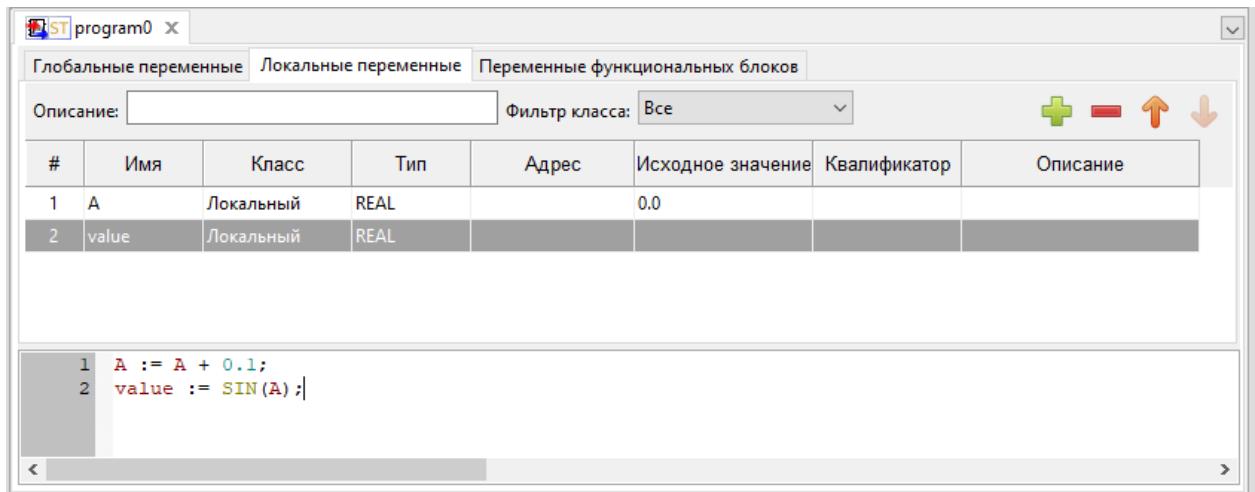


Рисунок 8.26 – Пример программного модуля тип «Программа» на языке ST

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

8.8 Эмуляция целевой программы.

Для отладки программы так же доступна эмуляция целевого устройства. Для запуска эмулятора необходимо нажать на панели инструментов кнопку запуска эмуляции (рисунок 8.27). При успешной сборке проекта эмулятор запустится (рисунок 8.28).



Рисунок 8.27 – Кнопка запуска эмулятора.

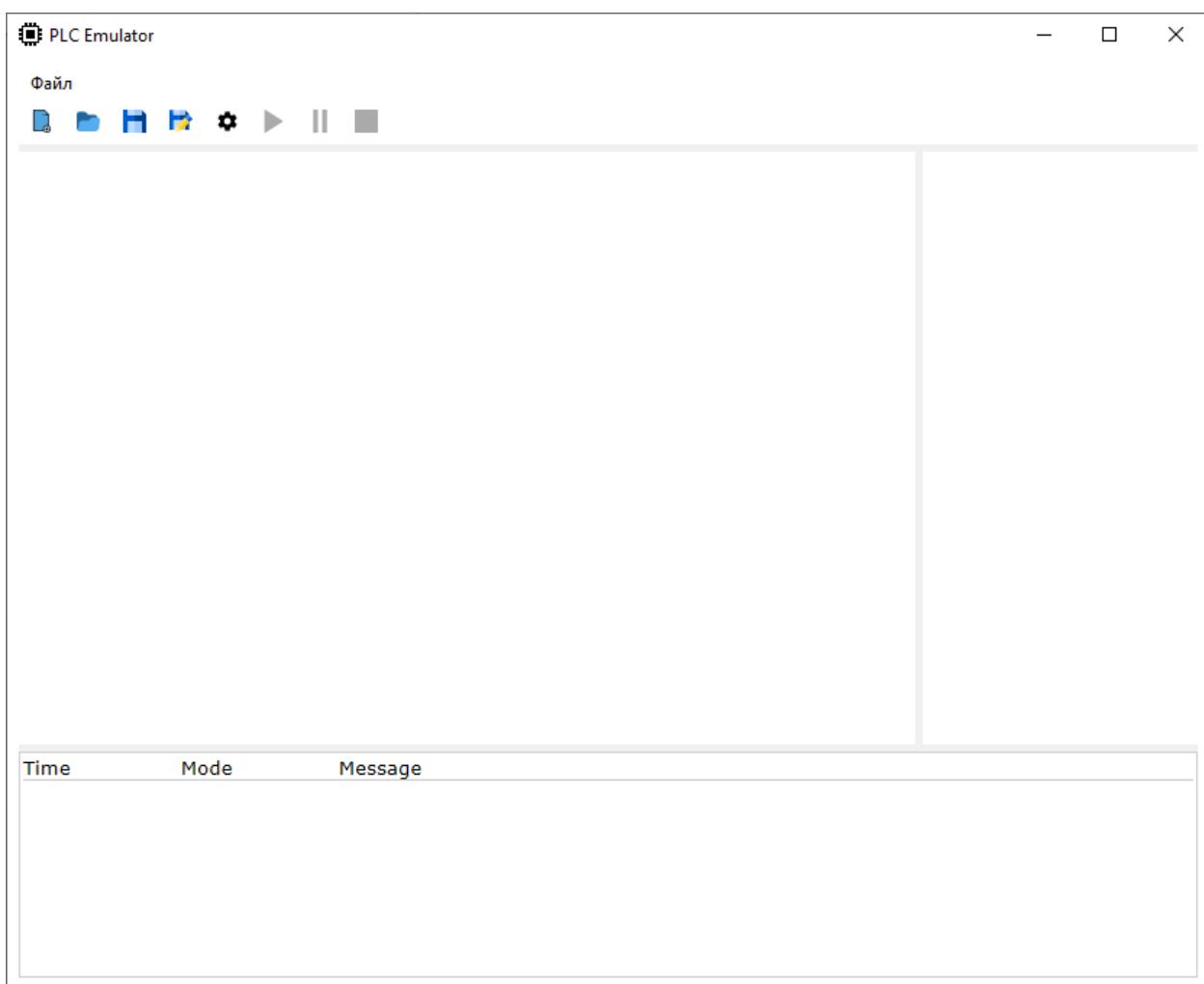


Рисунок 8.28 – PLC Emulator

8.8.1 Главное меню.

Главное меню программы содержит следующие пункты:

- Файл.

Далее будет подробно описан каждый пункт.

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

- «Новый» – создание нового проекта;
- «Открыть» – открытие существующего проекта;
- «Сохранить» – сохранение текущего проекта пункт;
- «Сохранить как» – сохранение текущего проекта в папку отличную от той, в которой он сохранён на данный момент;

8.8.2 Панель инструментов.

Панель инструментов представляет собой панель с кнопками для быстрого обращения к часто используемым функциям эмулятора. Она состоит из нескольких панелей, содержащих кнопки: управления проектом и установки связи с целевой программой с помощью PYRO сервера. Описание панелей представлено ниже.

8.8.2.1 Кнопки управления эмулятором.

Список кнопок панели инструментов (рисунок 8.29) и их функции приведены в таблице 1.



Рисунок 8.29 – Кнопки управления проектом

Таблица 14 - Функции кнопок управления проектом

| Кнопка | Функция |
|--------|----------------------|
| | Новый проект |
| | Открыть проект |
| | Сохранить проект |
| | Сохранить проект как |

8.8.2.2 Кнопки установления связи с целевой программой.

Список кнопок панели инструментов (рисунок 8.30) и их функции приведены в Таблица 15.



Рисунок 8.30 – Кнопки установления связи с эмулируемой программой

Таблица 15 - Функции кнопок установления связи с эмулируемой программой.

| Кнопка | Функция |
|--------|--------------------------------------|
| | Настройки подключения к PYRO серверу |
| | Запуск эмуляции |
| | Приостановить эмуляцию |
| | Остановить эмуляцию |

Подробнее о настройке PYRO сервера см. п.8.8.6 – 8.8.7.

В зависимости от того, выполняется ли прикладная программа, появляются и скрываются определенный набор кнопок данной панели.

8.8.3 Библиотека элементов.

Панель библиотеки элементов располагается справа в окне эмулятора (рисунок 8.31). Она содержит коллекцию элементов, которые доступны при построении схемы для эмуляции устройства.

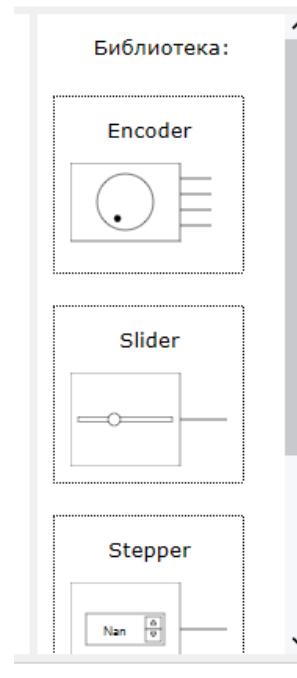


Рисунок 8.31 – Библиотека элементов.

Использование данных элементов осуществляется перетаскиванием необходимого блока с помощью зажатой левой кнопки мыши (Drag&Drop) в область редактирования.

Более подробное описание каждого элемента приведено в [Приложении 22](#).

8.8.4 Монитор СОМ-порта.

Монитор СОМ-порта расположен в нижней части окна. Данный монитор отображает сообщения протокола Modbus RTU (рисунок 8.32).

Modbus – открытый коммуникационный протокол, основанный на архитектуре ведущий — ведомый (англ. master-slave; в стандарте Modbus используются термины client-server). Широко применяется в промышленности для организации связи между электронными устройствами. Подробнее [см.: [Modbus Application Protocol V1.1b](#)]

| Time | Mode | Message |
|----------|------|----------------------------|
| 08:14:20 | RX | 01 03 04 00 00 00 00 FA 33 |
| 08:14:19 | TX | 01 10 00 2A 00 01 02 00 |
| 08:14:19 | RX | 01 03 04 00 00 00 00 FA |
| 08:14:19 | TX | 01 10 00 2A 00 01 02 00 |
| 08:14:19 | RX | 01 03 04 00 00 00 00 FA |
| 08:14:20 | TX | 01 10 00 2A 00 01 02 00 |
| 08:14:20 | RX | 01 03 04 00 00 00 00 FA 33 |

Рисунок 8.32 – Монитор СОМ-порта

По рисунку выше можно проанализировать обмен сообщениями. Столбец «Mode» отображает режим работы эмулируемой целевой программы.

- «RX» – режим приемника.
- «TX» – режим передатчика.

Столбец «Message» отображает сообщения в шестнадцатеричном формате.

Для работы с Modbus RTU необходимо подключить преобразователь USB-RS485 к ПК.

8.8.5 Редактор эмулятора.

Для запуска редактора эмулятора необходимо создать проект, или открыть ранее созданный. При создании нового проекта выводится диалоговое окно с выбором целевой платформы (рисунок 8.33).

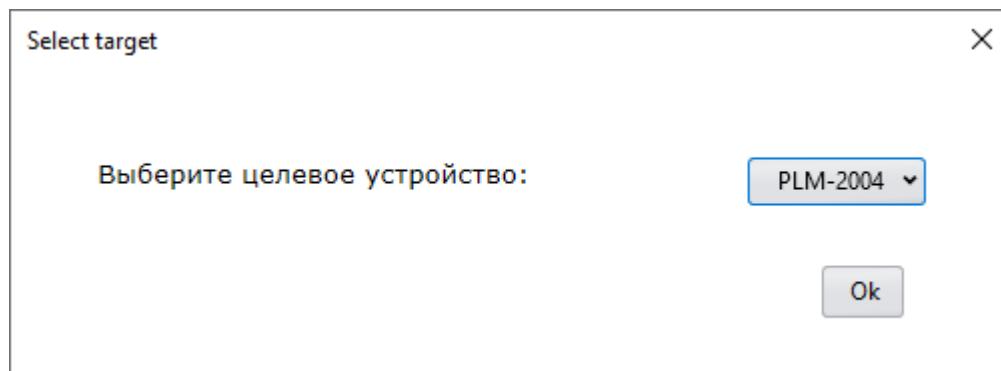


Рисунок 8.33 – Выбор целевого устройства

Редактор эмулятора представляет собой графическую область. Основными элементами являются ПЛК, подключаемые к нему компоненты и линии связи элементов. Редактор элементов представлен ниже (рисунок 8.34).

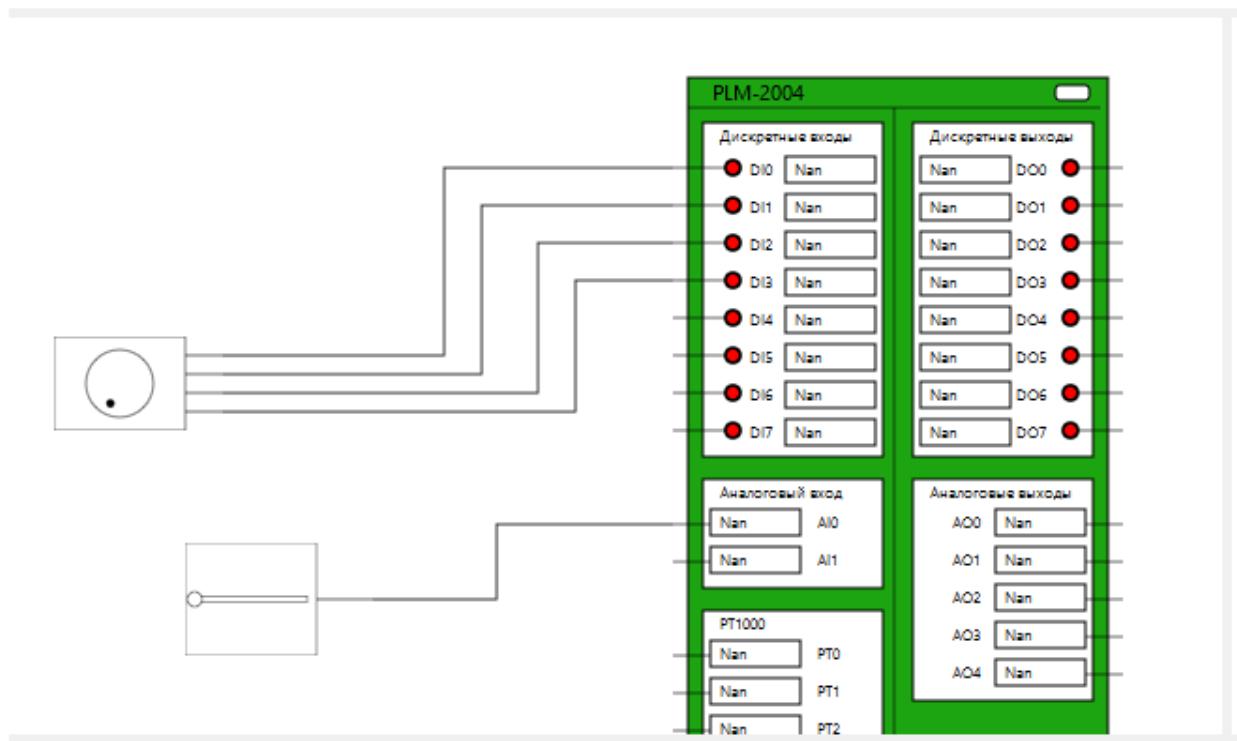


Рисунок 8.34 – Редактор элементов

В редакторе доступно добавление, удаление, копирование элементов, перемещение, а так же соединение элементов линиями связи.

Для увеличения поля редактора необходимо зажать Ctrl и с помощью колеса прокрутки мыши увеличить или уменьшить поле. Для перемещения по полю редактора необходимо зажать колесико мыши и перемещать поле в нужном направлении.

При двойном клике по элементу открывается окно конфигурации элемента. Ниже представлен рисунок конфигурации ПЛК (рисунок 8.35)

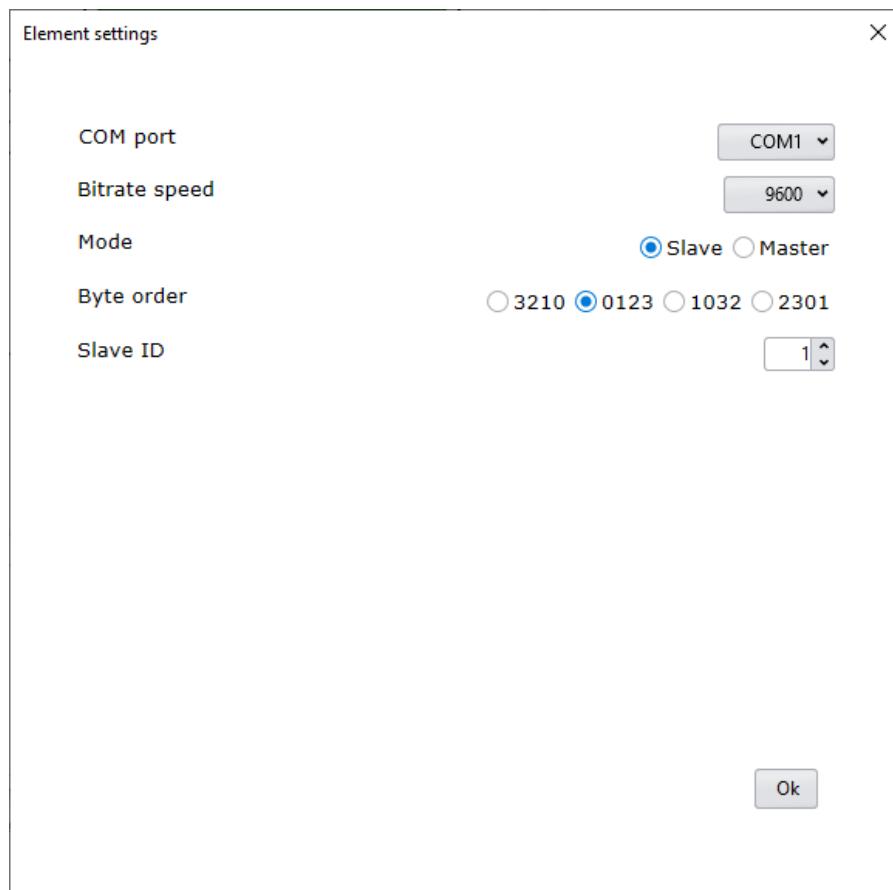


Рисунок 8.35 – Конфигурация ПЛК.

Как видно из рисунка выше конфигурация ПЛК сводится к конфигурации Modbus RTU.

8.8.6 Конфигурация сервера PYRO.

При запуске целевой программы на ПК запускается программа «Beremiz_service». Сервер необходим для связи запущенной целевой программы с ИСР Beremiz и эмулятором, он получает значения для отладки и эмуляции целевой программы. PYRO сервер является сокетом, и для передачи между процессами использует протокол TCP/IP. Данная программа запускается в фоновом режиме при запуске эмулятора, настраивает TCP/IP соединение и запускает целевую программу. При запуске «Beremiz_service» в панели задач отобразится иконка (рисунок 8.36).



Рисунок 8.36 – Иконка программы «Beremiz_service» в панели задач

При нажатии правой кнопкой на иконку «Beremiz_service» появляется меню настроек сервера (рисунок 8.37). Ниже описаны все параметры конфигурации «Beremiz_service»:

- «Start PLC» – запуск целевой программы.
- «Stop PLC» – остановка целевой программы.

- «Change Name» – сменить имя сервера
- «Change IP of interface to bind» – изменить IP адрес. Целевая программа запускается на локальном ПК что соответствует IP адресу «localhost».
- «Change Port Number» – смена номера порта сервера. При запуске эмулятора порт задается автоматически.
- «Change working directory» – сменить рабочую директорию. При запуске эмулятора создается временная папка для хранения и запуска собранной целевой программы. После прекращения работы программы временная папка удаляется.
- «Launch a live Python shell» – Python shell необходим для отладки расширений написанных на python.
- «Launch WX GUI inspector» – отладчик для wxWidgets.
- «Quit» – Прекращение работы программы.

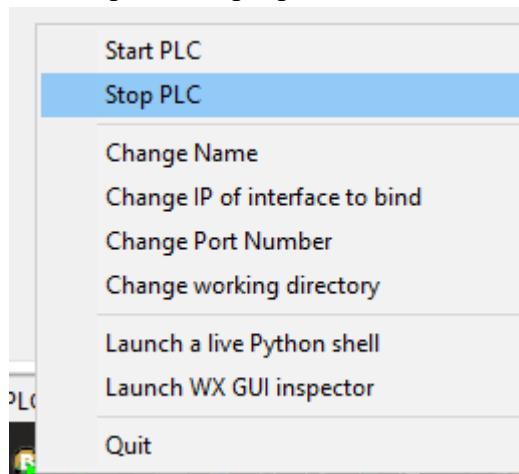


Рисунок 8.37 – Меню настроек «Beremiz_service»

8.8.7 Конфигурация клиента PYRO.

Эмулятор и среда Беремиз являются клиентами.

При нажатии на кнопку запуска эмулятора сборка программы, запуск сервера, а так же конфигурация клиента и сервера происходит автоматически. Так же доступно установка пользовательской конфигурации. Для изменения конфигурации PYRO соединения с сервером необходимо нажать на кнопку конфигурации в панели инструментов (см. п. 8.8.2.2).

Эмулятор способен эмулировать работу ПЛК при помощи графического интерфейса ПК, а так же поддерживает работу с памятью и протокол Modbus.

8.8.8 Добавление пользовательских элементов в эмулятор.

Добавление пользовательских элементов производится при помощи добавления модулей, написанных на языке javascript, в папку «beremiz/emulator/Sciter/editor/elements». Подробное руководство о добавлении пользовательского элемента описано в [приложении 23](#).

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.)

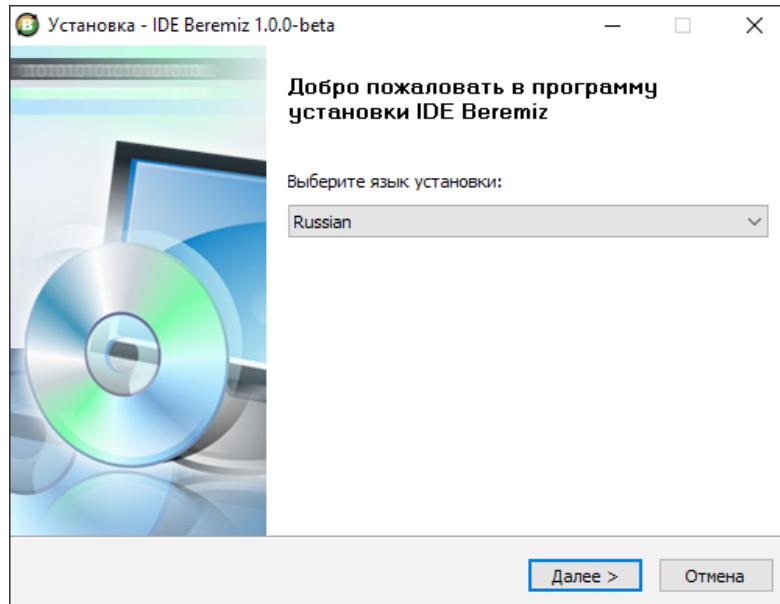


Рисунок 1.1 – Выбор языка установки

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

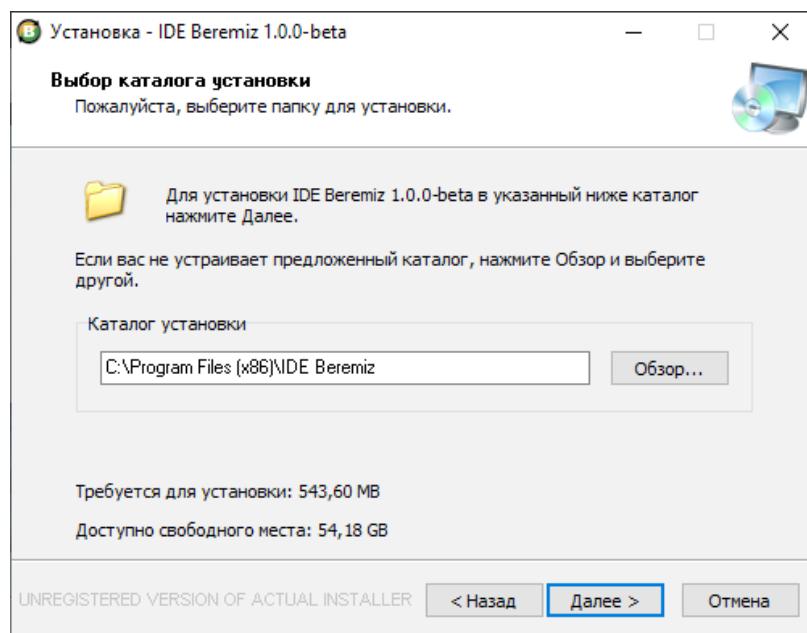


Рисунок 1.2 – Выбор директории установки

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

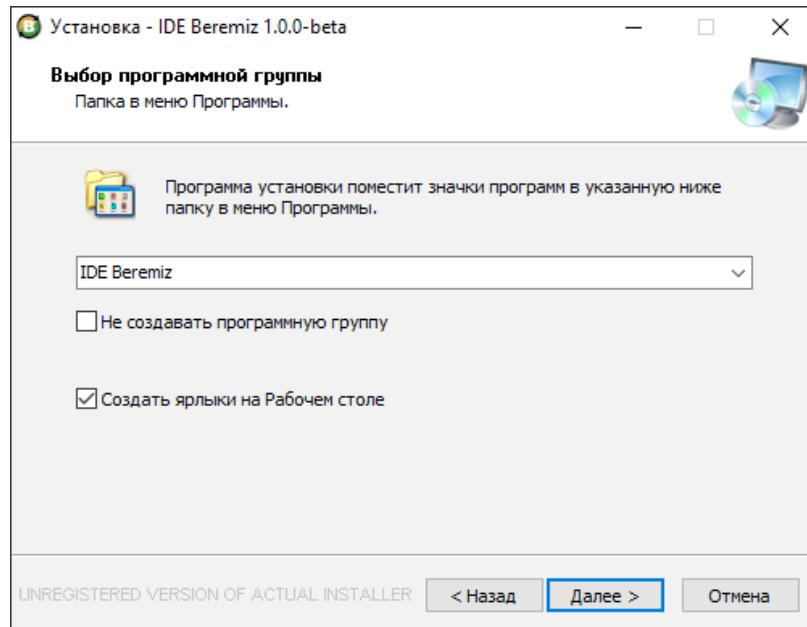


Рисунок 1.3 – Выбор группы в меню Пуск и разрешение на создание ярлыка на Рабочем столе

- 5) Подтвердить установку (рисунок 1.4)

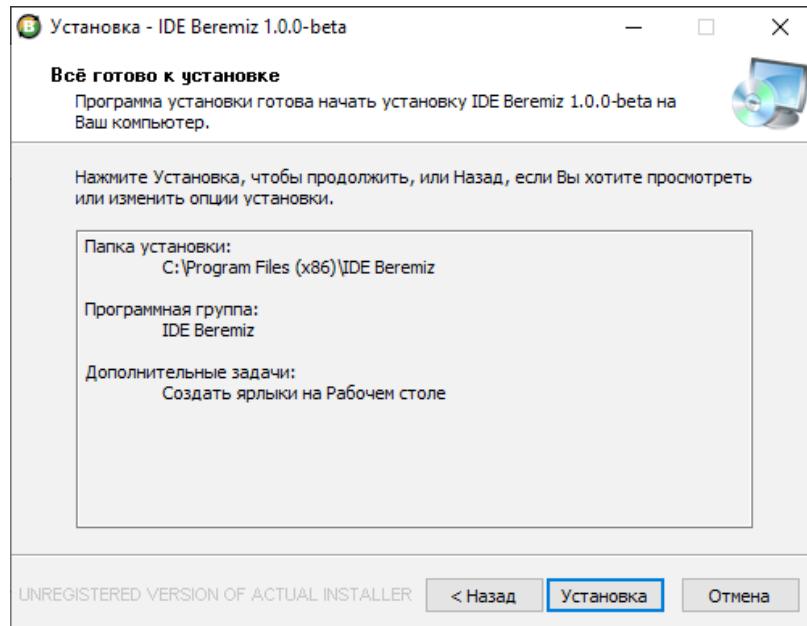


Рисунок 1.4 – Подтверждение установки

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

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

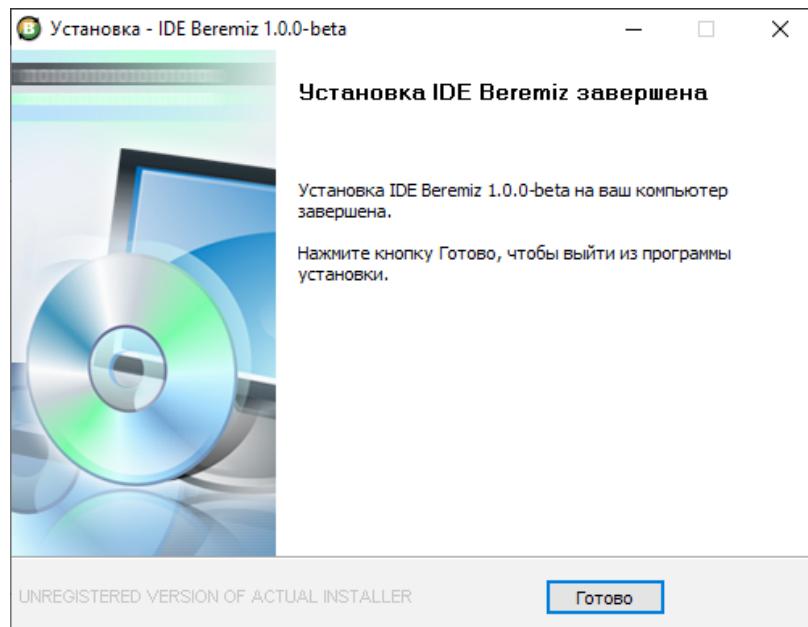


Рисунок 1.5 – Подтверждение установки

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".

СТУ – 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 \geq 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.

СТД – 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 \leq 0, выход Q принимает значение "1".

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

СТUD – 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 \geq PV, выход QU имеет значение "1".

Когда CV \leq 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.

ТР – 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) повторяет 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 – возвведение 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 IN1, иначе – OUT равно IN2.

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 выходе 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.

Для побитного обращения (установить/снять бит) к битовым строкам типа WORD и LWORD можно обращаться с помощью функций из библиотеки LibIT POU, указанных ниже. Для выполнения побитного обращения к переменным другого типа необходимо выполнить преобразования типов с помощью соответствующих функций или программных блоков.

SetBitWo – установить значение бита в указанной позиции переменной типа WORD

Данная функция возвращает в OUT значение измененной переменной Wo, в которой в бите с позицией, указанной в параметре Pos, установлено значение, указанное в параметре Val. Параметрам можно присваивать переменные следующих типов:

- Wo [WORD]
- Pos [BYTE] Допустимое значение: 0...15
- Val [BOOL]

Выходная переменная должна иметь тип WORD.

GetBitWo – считать значение бита в указанной позиции переменной типа WORD

Данная функция возвращает в OUT значение бита переменной Wo, расположенного в позиции, указанной в параметре Pos. Параметрам можно присваивать переменные следующих типов:

- Wo [WORD]
- Pos [BYTE] Допустимое значение: 0...15

Выходная переменная должна иметь тип BOOL.

SetBitLWo – установить значение бита в указанной позиции переменной типа LWORD

Данная функция возвращает в OUT значение измененной переменной Wo, в которой в бите с позицией, указанной в параметре Pos, установлено значение, указанное в параметре Val. Параметрам можно присваивать переменные следующих типов:

- Wo [LWORD]
- Pos [BYTE] Допустимое значение: 0...63
- Val [BOOL]

Выходная переменная должна иметь тип LWORD.

GetBitLWo – считать значение бита в указанной позиции переменной типа WORD

Данная функция возвращает в OUT значение бита переменной Wo, расположенного в позиции, указанной в параметре Pos. Параметрам можно присваивать переменные следующих типов:

- Wo [LWORD]
- Pos [BYTE] Допустимое значение: 0...63

Выходная переменная должна иметь тип BOOL.

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

| ANY | | | | |
|--|--|--|---|--|
| ANY_BIT | ANY_NUM | | ANY_DATE | TIME STRING и другие типы данных |
| BOOL BYTE WORD DWORD LWORD | ANY_INT INT SINT DINT LINT | | ANY_REAL UINT USINT UDINT ULINT | |
| | | | DATE TIME_OF_DAY DATE_AND_TIME | |

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 REPEAT 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»;

- \diamond – сравнение на «неравенство»;
- \gg – сравнение на «больше»;
- $\geq\gg$ – сравнение на «не меньше»;
- \ll – сравнение на «меньше»;
- $\leq\ll$ – сравнение на «не больше».

В качестве результата сравнения всегда используется значение типа BOOL.

Присвоение

Для обозначения присвоения используется парный знак \coloneqq (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 выполнится.

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. Стока 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 |
|---|---|
| <pre> 01: CASE k OF 02: 1: 03: k := k * 10; 04: 2..5: 05: k := k * 5; 06: i := 0; 07: 5, 9..20: 08: k := k - 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE; </pre> <p style="text-align: center;">Диапазоны в строках 04 и 07 пересекаются</p> | <pre> 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; </pre> |

| Неправильная запись | Правильная запись |
|--|---|
| <pre> 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; </pre> <p style="text-align: center;">В строке 07 диапазон значений задан неправильно</p> | <pre> 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; </pre> |

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 operand |
| R | Если значение аккумулятора FALSE, сбросить логический operand |
| 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 operand, результат записывается в аккумулятор |
| DIV | Деление аккумулятора на operand, результат записывается в аккумулятор |
| GT | Значение аккумулятора сравнивается со значением операнда(>(greater than)). Значение (TRUE или FALSE) записывается в аккумулятор |
| GE | Значение аккумулятора сравнивается со значением операнда(>=greater than or equal)). Значение (TRUE или FALSE) записывается в аккумулятор |
| EQ | Значение аккумулятора сравнивается со значением операнда (=equal)). Значение (TRUE или FALSE) записывается в аккумулятор |
| NE | Значение аккумулятора сравнивается со значением операнда (<>(not equal)). Значение (TRUE или FALSE) записывается в аккумулятор |
| LE | Значение аккумулятора сравнивается со значением операнда (<=(less than or equal to)). Значение (TRUE или FALSE) записывается в аккумулятор |
| LT | Значение аккумулятора сравнивается со значением операнда (<(less than)). Значение (TRUE или FALSE) записывается в аккумулятор |

| Оператор | Описание |
|----------|--|
| 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 $C = A \text{ AND NOT } B$:

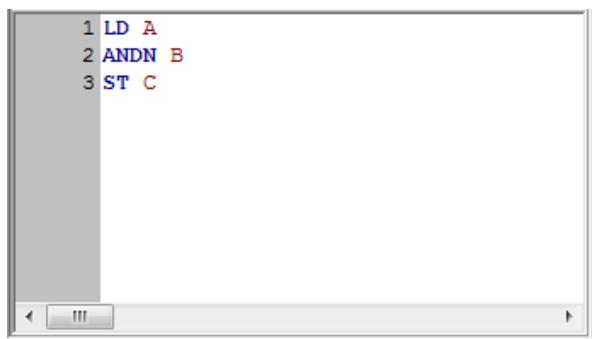


Рисунок 4.1 – Пример программы на языке IL

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 С 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 булевые 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*».

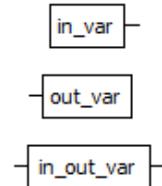


Рисунок 5.1 – Изображение переменной в языке FBD

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

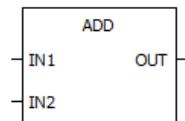


Рисунок 5.2 – Изображение функции в языке FBD

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

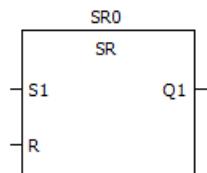


Рисунок 5.3 – Изображение функционального блока в языке FBD

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.

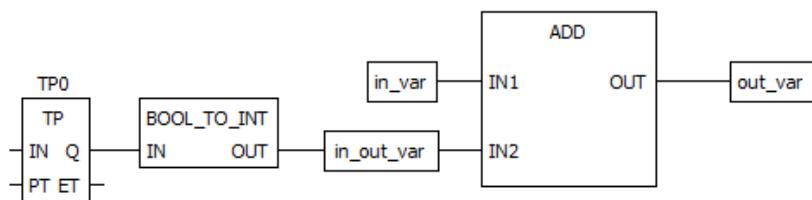


Рисунок 5.4 – Пример соединения переменных, функций и функциональных блоков

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.

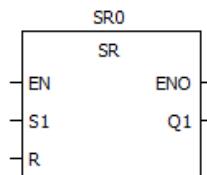


Рисунок 5.5 – Изображение элементарного функционального блока с параметрами EN/ENO

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:

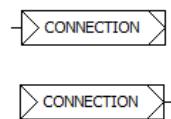


Рисунок 5.6 – Изображение соединений в языке FBD

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:

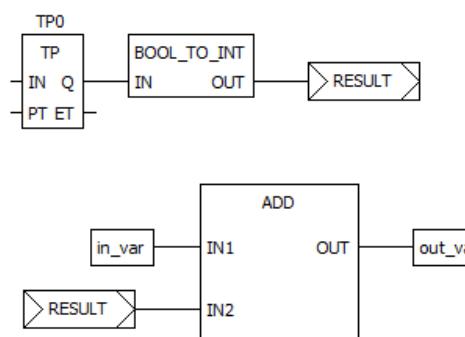


Рисунок 5.7 – Пример использования соединения на FBD диаграмме

Пример программы на языке FBD

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

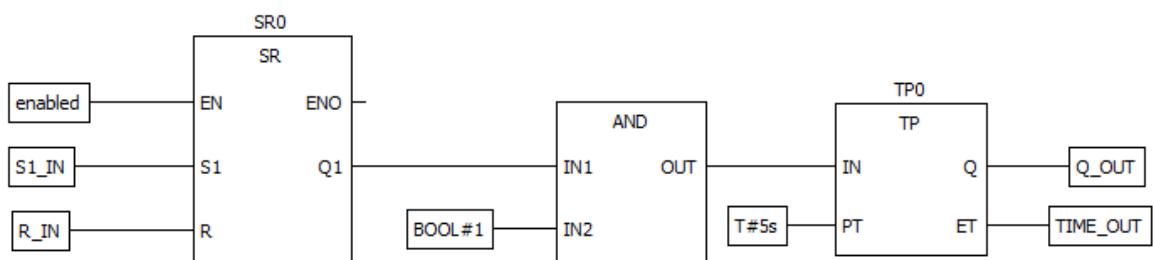


Рисунок 5.8 – пример FBD диаграммы

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).

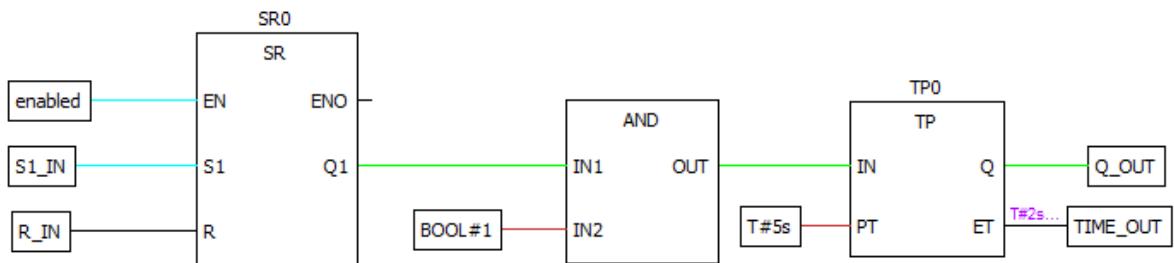


Рисунок 5.9 – Выполнение FBD диаграммы

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).

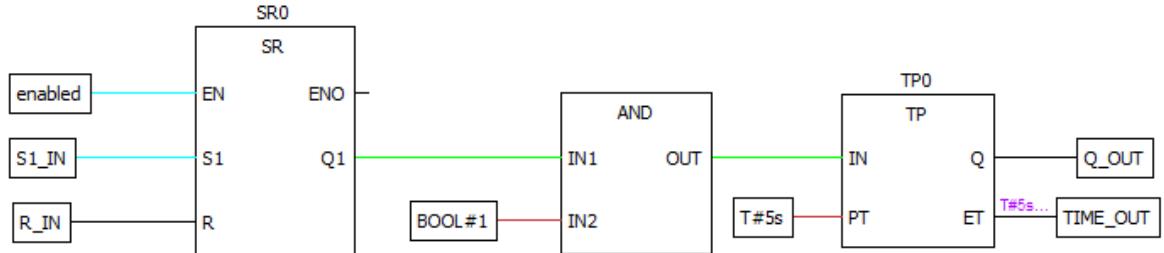


Рисунок 5.10 – Выполнение FBD диаграммы

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).

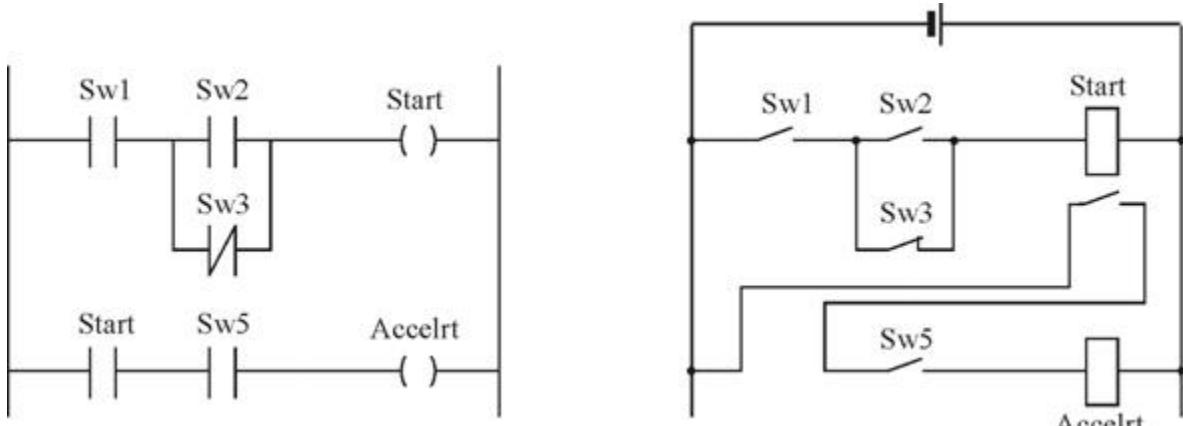


Рисунок 6.1 – Программа на языке LD (слева) и ее эквивалент в виде электрической (справа)

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 $\mid\lrcorner$ 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 переменной 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.

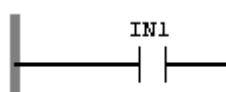


Рисунок 6.2 – Нормальный контакт

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.

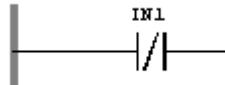


Рисунок 6.3 – Инверсный контакт

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.

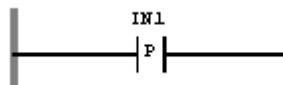


Рисунок 6.4 – Контакт для обнаружения нарастания фронта

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.

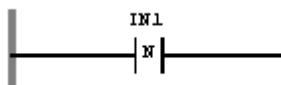


Рисунок 6.5 – Контакт для обнаружения спада фронта

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.

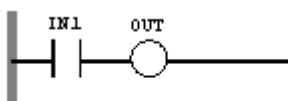


Рисунок 6.6 – Нормальные контакт и катушка

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.

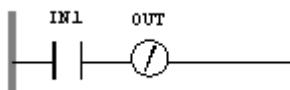


Рисунок 6.7 – Нормальный контакт и инвертированная катушка

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.

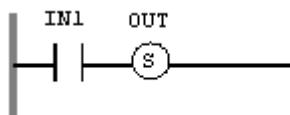


Рисунок 6.8 – Нормальный контакт и катушка установки

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.

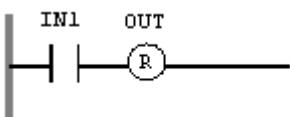


Рисунок 6.9 – Нормальный контакт и катушка сброса

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.

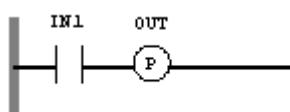


Рисунок 6.10 – Нормальный контакт и катушка обнаружения нарастания фронта

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.

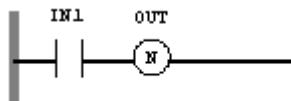


Рисунок 6.11 – Нормальный контакт и катушка обнаружения спада фронта

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:

$C = A \text{ AND NOT } B$

При создании LD диаграмм можно использовать только переменные типа BOOL. Добавим новый контакт и привяжем его к имени А (имени переменной). Далее добавляется шина питания слева, шина питания справа, нормальный контакт, инверсный контакт и нормальная катушка. Нормальный контакт ассоциируется с переменной А, инверсный контакт с переменной В, нормальная катушка с переменной С. Далее это всё последовательно соединяется (рисунок 6.12), и результатом является программа, написанная на языке LD, реализующая логическое выражение:

$C = A \text{ AND NOT } B$

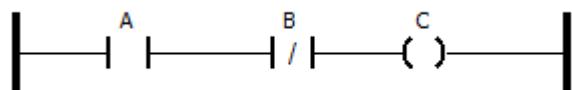


Рисунок 6.12 – Пример LD диаграммы, реализующей логическое выражение
 $C = A \text{ AND NOT } B$

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).

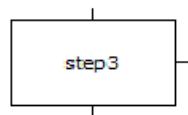


Рисунок 7.1 – Графическое представление «Шага» языка SFC

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).

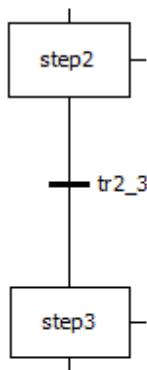


Рисунок 7.2 – Шаги «step2» и «step3», соединённые переходом «tr2_3»

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».

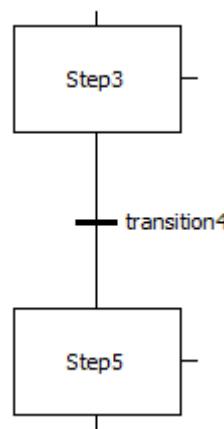


Рисунок 7.3 – Переход между шагами «Step3» и «Step5» с предопределённым условием «transition4»

В данном случае «transition4» это имя для предопределенного 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: $level > 10$

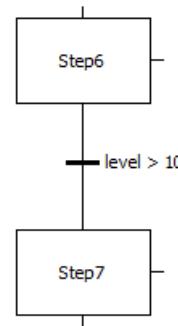


Рисунок 7.4 – Переход между шагами «Step6» и «Step7» с предопределенным условием
« $level > 10$ »

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

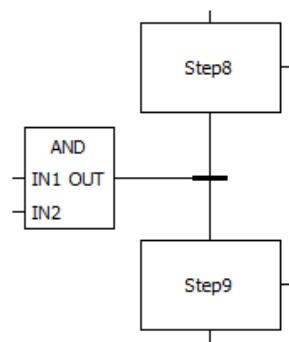


Рисунок 7.5 – Переход между шагами «step8» и «step9», заданный «логическим И» на языке FBD

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.

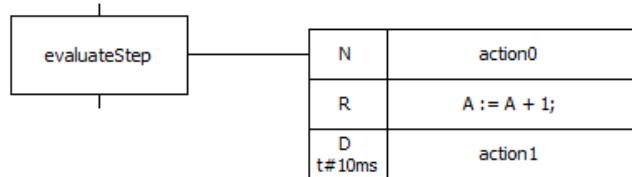


Рисунок 7.6 – Шаг «evaluateStep» и связанный с ним блок действий, содержащий 3 действия

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».

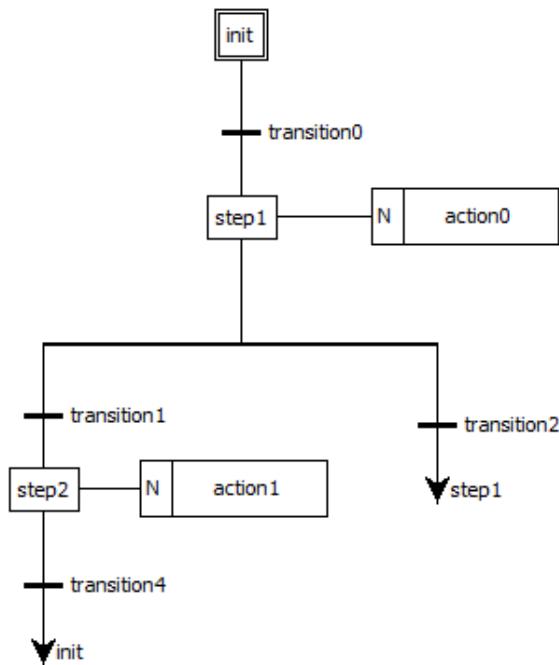


Рисунок 7.7 – SFC диаграмма, содержащая «прыжки»

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:

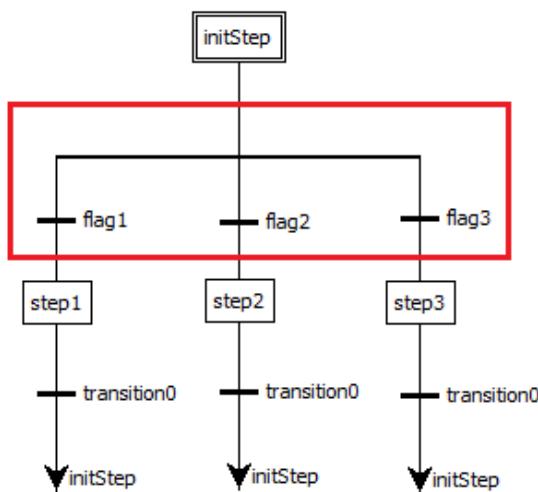


Рисунок 7.8 – Дивергенция на SFC диаграмме

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:

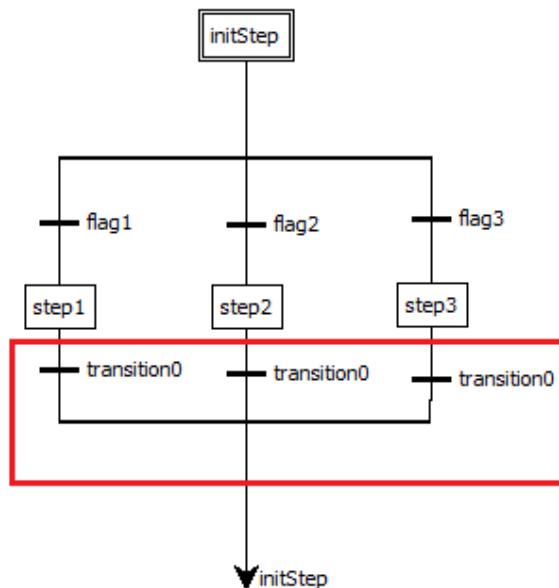


Рисунок 7.9 – Конвергенция на SFC диаграмме

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:

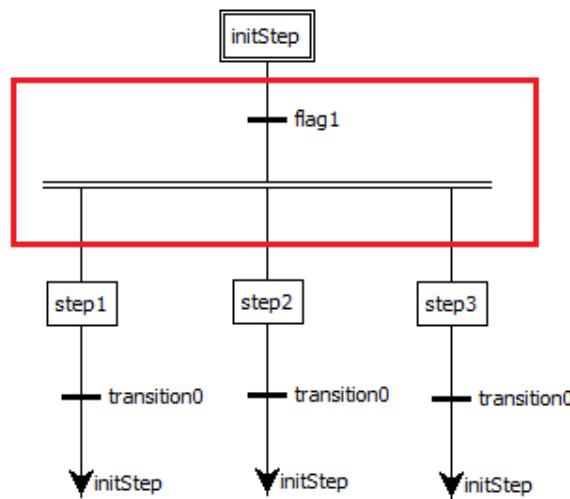


Рисунок 7.10 – Параллельная дивергенция на SFC диаграмме

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:

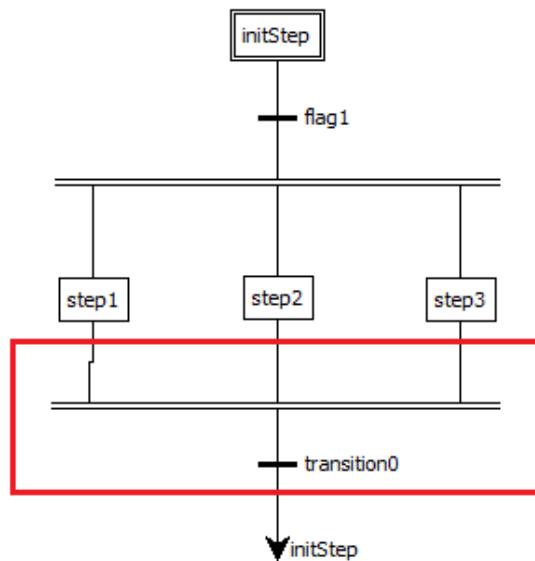


Рисунок 7.11 – Параллельная конвергенция на SFC диаграмме

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.

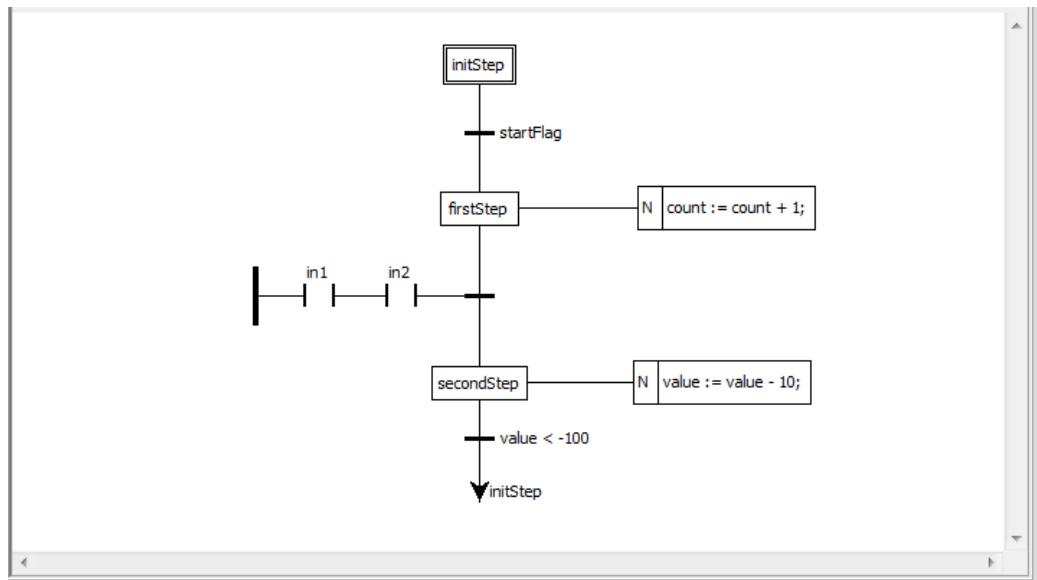


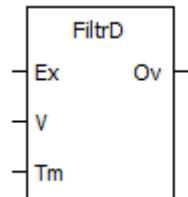
Рисунок 7.12 – SFC диаграмма

ПРИЛОЖЕНИЕ 8

БИБЛИОТЕКА АЛГОРИТМОВ ФИЛЬТРАЦИИ СИГНАЛОВ

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы фильтрации сигналов. Алгоритмы аппаратно-независимые.

Функциональный блок FiltrD



Функциональный блок «FiltrD» осуществляет фильтрацию входного дискретного значения.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

V — исходное значение [BOOL]

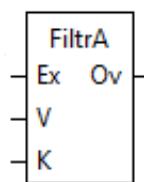
Tm — время выдержки исходного значения [TIME]

Выходы:

Ov — отфильтрованное значение [BOOL]

Если Ex равно FALSE, то возвращается предыдущее отфильтрованное значение (по умолчанию, FALSE).

Функциональный блок FiltrA



Функциональный блок «FiltrA» осуществляет фильтрацию входного аналогового значения.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

V — исходное значение [REAL]

Tm — коэффициент Калмана (степень сглаживания сигнала) [REAL]

= 0,0 ... 1,0 (чем меньше, тем сильнее сглаживание)

Выходы:

Ov – отфильтрованное значение [REAL]

Фильтрация осуществляется по формуле:

$$Ov = K \cdot V + (1 - K) \cdot Ov_{-1}$$

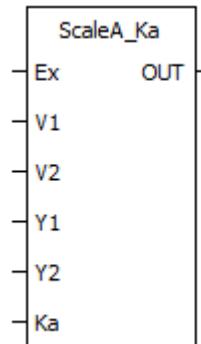
где,

Ov_{-1} – отфильтрованное значение на предыдущем цикле.

Если Ex равно FALSE, то возвращается предыдущее отфильтрованное значение (по умолчанию, 0.0).

ПРИЛОЖЕНИЕ 9**БИБЛИОТЕКА АЛГОРИТМОВ МАСШТАБИРОВАНИЯ СИГНАЛОВ**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы масштабирования сигналов. Алгоритмы аппаратно-независимые.

Функция ScaleA_Ka

Функция «ScaleA_Ka» вычисляет коэффициент масштабирования.

Входы:

Ex — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть

V1 — значение нижней точки [REAL]
V2 — значение верхней точки [REAL]
Y1 — калибровочное значение нижней точки [REAL]
Y2 — калибровочное значение верхней точки [REAL]
Ka — исходный коэффициент [REAL]

Выходы:

Out — коэффициент масштабирования (угловой коэффициент) [REAL]

Коэффициент масштабирования вычисляется по формуле:

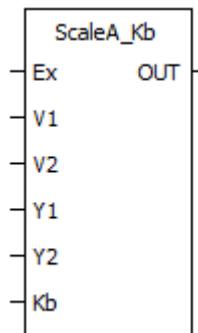
$$Out = \frac{Y2 - Y1}{V2 - V1}$$

Если Ex равно FALSE, то возвращается значение Ka.

Значение нижней или верхней точки — например, код АЦП от 0 до 4095.

Калибровочное значение нижней и верхней точки — например, от 0 до 10 В.

Функция ScaleA_Kb



Функция «ScaleA_Kb» вычисляет смещение характеристики.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

V1 — значение нижней точки [REAL]

V2 — значение верхней точки [REAL]

Y1 — калибровочное значение нижней точки [REAL]

Y2 — калибровочное значение верхней точки [REAL]

Kb — исходный коэффициент [REAL]

Выходы:

Out — коэффициент смещения [REAL]

Смещение характеристики вычисляется по формуле:

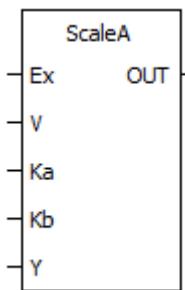
$$Out = \frac{Y1 \cdot V2 - Y2 \cdot V1}{V2 - V1}$$

Если Ex равно FALSE, то возвращается значение Kb.

Значение нижней или верхней точки — например, код АЦП от 0 до 4095.

Калибровочное значение нижней и верхней точки — например, от 0 до 10 В.

Функция ScaleA



Функция «ScaleA» выполняет масштабирование аналогового входного сигнала к значению определенной физической величины (например, входное значение в мА преобразует в кПа).

Входы:

Ex — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть

V — значение текущей точки [REAL]

Ka — угловой коэффициент масштабирования [REAL]

Kb — коэффициент смещения масштабирования [REAL]

Y — исходное отмасштабированное значение [REAL]

Выходы:

Out — отмасштабированное значение [REAL]

Масштабирование производится по формуле:

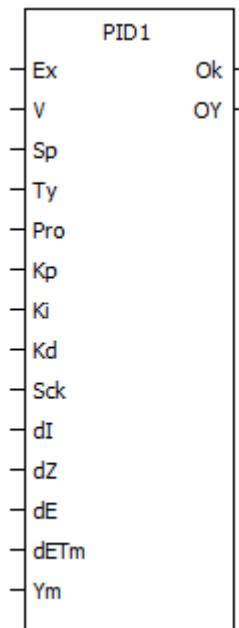
$$Out = Ka \cdot V + Kb$$

Если Ex равно FALSE, то возвращается значение Y.

Значение текущей точки — например, код АЦП от 0 до 4095.

ПРИЛОЖЕНИЕ 10**БИБЛИОТЕКА ПИД-РЕГУЛЯТОРОВ**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы ПИД-регуляторов. Алгоритмы аппаратно-независимые.

Функциональный блок PID1

Функциональный блок «PID1» реализует ПИД-регулятор и выдает в качестве управляющего воздействия значение (%) в зависимости от величины ошибки рассогласования на входе регулятора. Квантование регулятора выполняется внешним дискретным сигналом.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

V — входное значение с источника сигнала [REAL]

= 0.0 (по умолчанию)

Sp — уставка (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

Ty — тип регулятора [BYTE]

= 0 — ПИД (по умолчанию),

= 1 — ПИ,

= 2 — ПД,

= 3 — П;

Pro — тип процесса [BYTE]

= 0 — увеличение (например, нагрев) (по умолчанию),

= 1 — уменьшение (например, охлаждение).

Kp – коэффициент пропорции [REAL]
 = 0.0 (по умолчанию)

Ki – коэффициент интеграла [REAL]
 = 0.0 (по умолчанию)

Kd – коэффициент дифференциала [REAL]
 = 0.0 (по умолчанию)

Sck — внешняя команда квантования регулятора [BOOL]
 = FALSE — нет (по умолчанию)
 = TRUE — есть

dI — зона работы интегральной составляющей (в единицах значения V) [REAL]
 = 0.0 (по умолчанию)

dZ — зона нечувствительности (в единицах значения V) [REAL]
 = 0.0 (по умолчанию)

dE — зона определения обрыва контура управления (в единицах значения V) [REAL]
 = 0.0 (по умолчанию)

dETm — время определения обрыва управления (мс) [DWORD]
 = 5000 (по умолчанию). Если значение параметра меньше 100, контроль обрыва управления не выполняется.

Ym — тип выхода регулятора [BYTE]
 = 0 - униполярный (от 0.0 до 100.0 %) (по умолчанию)
 = 1 – биполярный (от -100.0 до 100.0 %).

Выходы:

Ok – статус работы регулятора [BYTE]
 = 0 – остановлен,
 = 1 – работает,
 = 2 – работает в зоне нечувствительности,
 = 3 – работает в зоне интегральной составляющей,
 = 4 – ошибка! обрыв контура управления,
 = 5 – ошибка! задан некорректный тип регулятора («Ту»).

OY – выходной уровень регулятора (%) [REAL]
 = 0.0 (по умолчанию)

Разрешение работы регулятора выполняется подачей логической «1» (TRUE) на вход Ex.

При «Ex» равном логическому 0:

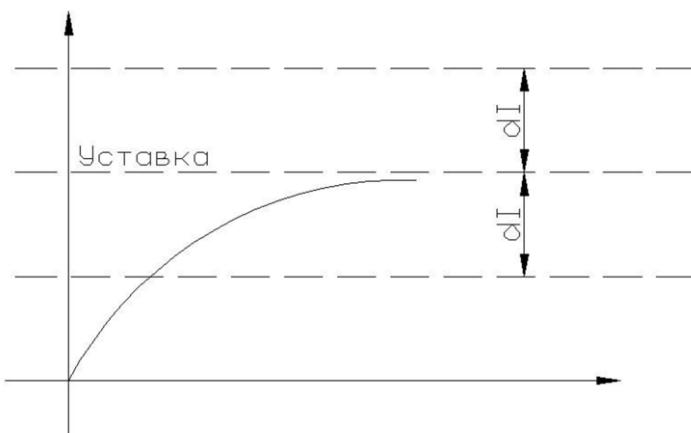
- «Ok» = 0 (регулятор выключен, по умолчанию),
- «OY» = 0.0 %.

При «Ex» равном логической 1:

- «Ok» = 1, 2, 3, 4 5,
- «OY» = 0.0 % (при ошибке) или высчитанному значению регулятора.

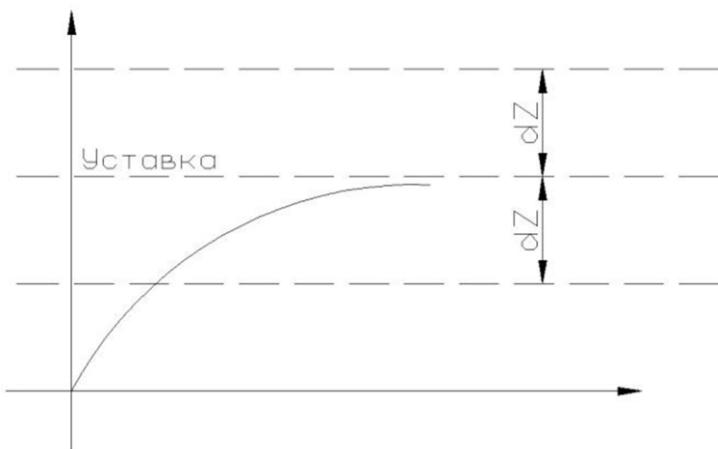
Квантование регулятора выполняется от внешнего генератора – по переднему фронту сигнала на входе «Sck» (например, квантование от блока таймера TON.Q).

Состояние работы алгоритма контролируется по значению выхода Ok.



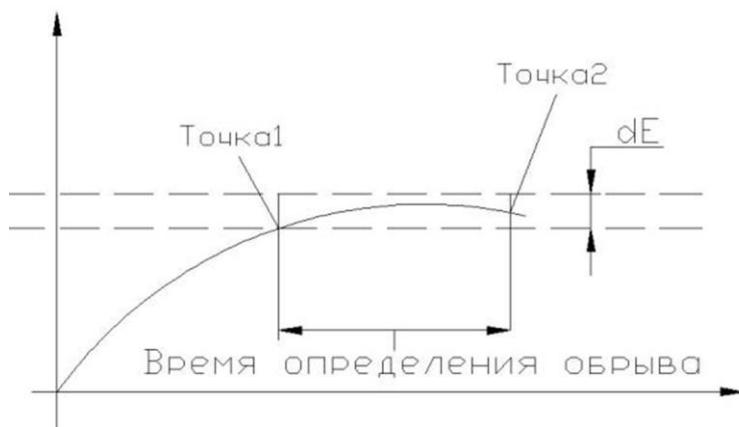
Зона интегрирования

Параметр dI определяет диапазон (в единицах источника сигнала), в котором интегральная составляющая ПИД/ПИ регулятора будет задействована. Для ПИД/ПИ-регуляторов, если dI равно нулю, то интегрирование будет выполняться во всем диапазоне, иначе — только в заданной зоне. Отключение интегральной составляющей возможно путем выбора другого типа регулятора («Ту») или заданием нулевого коэффициента интегрирования (« Ki »).



Зона нечувствительности

Параметр dZ определяет зону нечувствительности в которой управляющее воздействие остаётся равным тому значению, которое было на момент вхождения в эту зону.



Зона окна и время определения обрыва контура управления

Обрыв контура управления определяется, если абсолютное значение управляющего воздействия (ABS(OY)) достигло 100.0 %:

- 1) производится засечка регулируемой величины в точке 1,
- 2) начинается отсчёт времени,
- 3) когда отсчитываемое время равно времени определения обрыва управления «dETm», то производится сравнение:

$$(Точка2 - Точка1) < de \text{ ИЛИ } (Точка2 - Точка1) < 0$$

- 4) если результат сравнения истина, то:

- фиксируется аварийная ситуация «Обрыв контура управления»,
- регулятор принудительно останавливается,
- на выходе «Ok» код 4 - «Ошибка! Обрыв контура управления».

При появлении ошибок регулятор останавливается:

- фиксируется и выводится код состояния «Ok» (4 или 5),
- выходное значение «OY» равно 0.0 %.

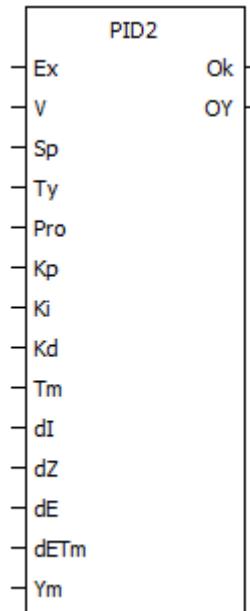
Сброс ошибок выполняется перезапуском регулятора: на вход «Ex» подать логический 0 и через один цикл выполнения приложения подать логическую «1».

При перезапуске регулятора:

- сбрасываются ошибки,
- обнуляются составляющие регулятора (П, И, Д).

Для управления дискретным выходом методом ШИМ, за период ШИМ можно принимать время квантования регулятора, а выход регулятора (OY) за коэффициент заполнения.

Функциональный блок PID2



Функциональный блок «PID2» реализует ПИД-регулятор и выдает в качестве управляющего воздействия значение (%) в зависимости от величины ошибки рассогласования на входе регулятора. Квантование регулятора выполняется от внутреннего таймера. Логика работы полностью совпадает с логикой работы блока «PID1».

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

V — входное значение с источника сигнала [REAL]

= 0.0 (по умолчанию)

Sp — уставка (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

Ty — тип регулятора [BYTE]

= 0 — ПИД (по умолчанию),

= 1 — ПИ,

= 2 — ПД,

= 3 — П;

Pro — тип процесса [BYTE]

= 0 — увеличение (например, нагрев) (по умолчанию),

= 1 — уменьшение (например, охлаждение).

Kp — коэффициент пропорции [REAL]

= 0.0 (по умолчанию)

Ki — коэффициент интеграла [REAL]

= 0.0 (по умолчанию)

Kd — коэффициент дифференциала [REAL]

= 0.0 (по умолчанию)

Tm — время квантования регулятора (мсек) [BYTE]

= 1000 (по умолчанию)

dI — зона работы интегральной составляющей (в единицах значения V) [REAL]
= 0.0 (по умолчанию)

dZ — зона нечувствительности (в единицах значения V) [REAL]
= 0.0 (по умолчанию)

dE — зона определения обрыва контура управления (в единицах значения V)
[REAL]
= 0.0 (по умолчанию)

dETm — время определения обрыва управления (мс) [DWORD]
= 5000 (по умолчанию). Если значение параметра меньше 100, контроль
обрыва управления не выполняется.

Ym — тип выхода регулятора [BYTE]
= 0 - униполярный (от 0.0 до 100.0 %) (по умолчанию)
= 1 – биполярный (от -100.0 до 100.0 %).

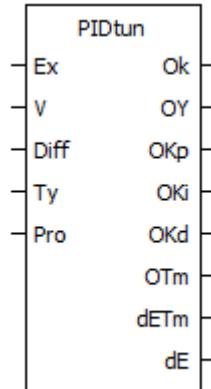
Выходы:

Ok – статус работы регулятора [BYTE]
= 0 – остановлен,
= 1 – работает,
= 2 – работает в зоне нечувствительности,
= 3 – работает в зоне интегральной составляющей,
= 4 – ошибка! обрыв контура управления,
= 5 – ошибка! задан некорректный тип регулятора («Ту»).

OY – выходной уровень регулятора (%) [REAL]
= 0.0 (по умолчанию)

При выборе времени квантования регулятора, возможно, потребуется учитывать время
цикла выполнения пользовательского приложения. Время цикла задается вручную в
настройках пользовательского проекта ICP Beremiz.

Функциональный блок PIDtun



Функциональный блок «PIDtun» выполняет автоматическую настройку ПИД-регулятора.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

V — входное значение с источника сигнала [REAL]

= 0.0 (по умолчанию)

Diff — разностная уставка (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

Ty — тип регулятора [BYTE]

= 0 — ПИД (по умолчанию),

= 1 — ПИ,

= 2 — ПД,

= 3 — П;

Pro — тип процесса [BYTE]

= 0 — увеличение (например, нагрев) (по умолчанию),

= 1 — уменьшение (например, охлаждение).

Выходы:

Ok — статус работы регулятора [BYTE]

= 0 — остановлен,

= 1 — работает,

= 2 — ошибка! задан некорректный тип регулятора («Ty»),

= 3 — работа завершена успешно.

OY — выходной уровень регулятора (%) [REAL]

= 0.0 (по умолчанию),

= 100.0 (на всем протяжении настройки).

OKp — вычисленный П-коэффициент [REAL]

= 0.0 (по умолчанию)

OKi — вычисленный И-коэффициент [REAL]

= 0.0 (по умолчанию)

OKd — вычисленный Д-коэффициент [REAL]

= 0.0 (по умолчанию)

OTm — вычисленный период квантования (мсек) [DWORD]

= 0 (по умолчанию)

dETm – вычисленный период квантования (мсек) [DWORD]
= 0 (по умолчанию)

dE – значение зоны определения обрыва контура управления [REAL]
= 0 (по умолчанию)

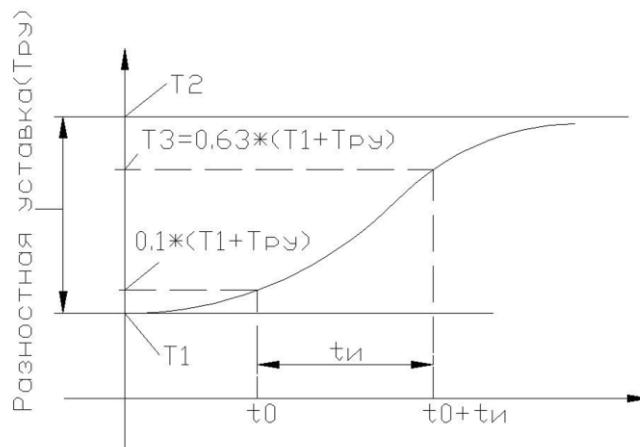


Схема автонастройки ПИД-регулятора

Автонастройка регулятора организована по схеме «Реакция системы на единичное воздействие»:

- 1) на управляющий элемент подаётся максимальное управляющее воздействие 100.0 %,
- 2) определяется величина t_0 (время транспортного запаздывания), когда:

$$V = (0.1 \cdot Diff) + V_1$$

где,

V – текущее значение с источника сигнала,

V_1 – значение с источника сигнала в момент запуска настройки.

- 3) определяется величина t_{ii} (постоянная времени системы), когда:

$$V = (0.63 \cdot Diff) + V_1$$

- 4) рассчитываются коэффициенты регулятора и время его квантования:

время квантования:

$$OT_m = t_0 \cdot 1.875 \text{ (мс)}$$

коэффициенты ПИД-регулятора:

$$OK_p = \frac{1.2}{R \cdot \frac{t_0}{1000}}; OK_i = 2 \cdot \frac{t_0}{1000}; OK_d = 0.4 \cdot \frac{t_0}{1000}$$

коэффициенты ПИ-регулятора:

$$OK_p = \frac{0.8}{R \cdot \frac{t_0}{1000}}; OK_i = 2 \cdot \frac{t_0}{1000}$$

коэффициенты ПД-регулятора:

$$OK_p = \frac{1}{R \cdot \frac{t_0}{1000}}; OK_d = 0.25 \cdot \frac{t_0}{1000}$$

коэффициенты П-регулятора:

$$OK_p = \frac{1}{R \cdot \frac{t_0}{1000}}$$

где,

$$R = \frac{Ref}{\frac{t - t_0}{1000}}$$

$$t = t_0 + t_i$$

t_0 - время транспортного запаздывания (мс),

t_i - постоянная времени системы (мс),

1000 – делитель для приведения временной шкалы к секундам (мс/1000 = сек).

Нужно учитывать, что перед запуском автонастройки система должна быть выведена в штатный режим работы по остальным параметрам. Например, для вентиляционных установок перед автонастройкой регулятора секции нагрева, установка должна быть выведена в штатный режим по воздуху.

Запуск автонастройки ПИД-регулятора выполняется подачей логической «1» (TRUE) на вход Ex, а подачей логического «0» - выключение после успешного завершения операции или прерывание во время работы.

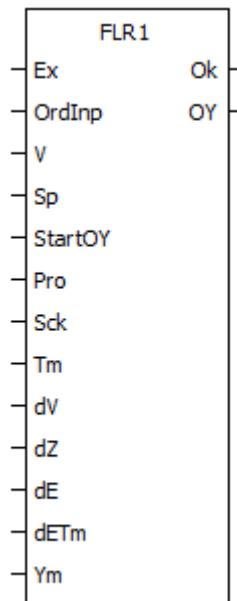
Состояние работы алгоритма контролируется по значению выхода Ok.

ПРИЛОЖЕНИЕ 11

БИБЛИОТЕКА РЕГУЛЯТОРОВ НЕЧЕТКОЙ ЛОГИКИ

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы регуляторов нечеткой логики. Алгоритмы аппаратно-независимые.

Функциональный блок FLR1



Функциональный блок «FLR1» реализует регулятор нечеткой логики и выдает в качестве управляющего воздействия скорость нарастания/спада управляющего сигнала в зависимости от величины ошибки рассогласования на входе регулятора. Чем больше разница между заданным значением и текущим, тем быстрее будет нарастать/убывать (в зависимости от знака разности) управляющий сигнал.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

OrdInp — порядок величин входного сигнала [BOOL]

= FALSE — величины первого порядка (до десятков)

= TRUE (по умолчанию) — величины второго и более порядков (от десятков и выше)

V — входное значение с источника сигнала [REAL]

= 0.0 (по умолчанию)

Sp — уставка (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

StartOY — значение на выходе регулятора с которого начнется регулирование при старте регулятора (в %) [REAL]

= 0.0 (по умолчанию)

Pro – тип процесса [BYTE]
 = 0 – увеличение (например, нагрев) (по умолчанию),
 = 1 – уменьшение (например, охлаждение).

Sck — внешняя команда квантования регулятора [BOOL]
 = FALSE — нет (по умолчанию)
 = TRUE — есть

Tm — период квантования регулятора (мсек) [DWORD]

dV — шаг приращения/уменьшения OY (%) [REAL]

dZ — зона нечувствительности (в единицах значения V) [REAL]
 = 0.0 (по умолчанию)

dE — зона определения обрыва контура управления (в единицах значения V) [REAL]
 = 0.0 (по умолчанию)

dETm — время определения обрыва управления (мс) [DWORD]
 = 5000 (по умолчанию). Если значение параметра меньше 100, контроль обрыва управления не выполняется.

Ym — тип выхода регулятора [BYTE]
 = 0 - униполярный (от 0.0 до 100.0 %) (по умолчанию)
 = 1 – биполярный (от -100.0 до 100.0 %).

Выходы:

Ok – статус работы регулятора [BYTE]
 = 0 – остановлен,
 = 1 – работает,
 = 2 – работает в зоне нечувствительности,
 = 4 – ошибка! обрыв контура управления,

OY – выходной уровень регулятора (%) [REAL]
 = 0.0 (по умолчанию)

Разрешение работы регулятора выполняется подачей логической «1» (TRUE) на вход Ex.

При «Ex» равном логическому 0:

- «Ok» = 0 (регулятор выключен, по умолчанию),
- «OY» = 0.0 %.

При «Ex» равном логической 1:

- «Ok» = 1, 2, 4,
- «OY» = 0.0 % (при ошибке) или высчитанному значению регулятора.

Квантование регулятора выполняется от внешнего генератора – по переднему фронту сигнала на входе «Sck» (например, квантование от блока таймера TON.Q).

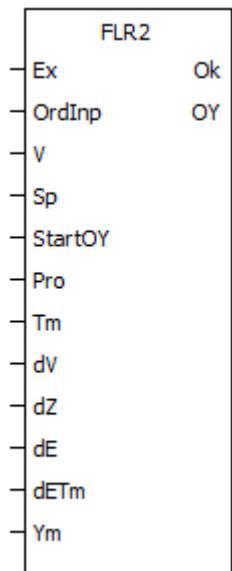
Состояние работы алгоритма контролируется по значению выхода Ok.

Параметр dV определяет шаг увеличения/уменьшения выхода регулятора «OY». Скорость изменения «OY» определяется временем задержки, максимальное значение которого равно периоду квантования регулятора «Tm» - это минимальная скорость.

Параметр **dZ** определяет зону нечувствительности, в которой управляющее воздействие остаётся равным тому значению, которое было на момент вхождения в эту зону (см. [приложение 9](#): функциональный блок «PID1» для пояснения).

Параметр **dE** определяет зону обрыва контура управления (см. [приложение 9](#): функциональный блок «PID1» для пояснения).

Функциональный блок FLR2



Функциональный блок «FLR2» реализует регулятор нечеткой логики (полностью наследует логику функционального блока «FLR1»). Квантование регулятора выполняется от внутреннего таймера.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

OrdInp — порядок величин входного сигнала [BOOL]

= FALSE — величины первого порядка (до десятков)

= TRUE (по умолчанию) — величины второго и более порядков (от десятков и выше)

V — входное значение с источника сигнала [REAL]

= 0.0 (по умолчанию)

Sp — уставка (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

StartOY — значение на выходе регулятора с которого начнется регулирование при старте регулятора (в %) [REAL]

= 0.0 (по умолчанию)

Pro — тип процесса [BYTE]

= 0 — увеличение (например, нагрев) (по умолчанию),

= 1 — уменьшение (например, охлаждение).

Tm — период квантования регулятора (мсек) [DWORD]

dV — шаг приращения/уменьшения OY (%) [REAL]

dZ — зона нечувствительности (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

dE — зона определения обрыва контура управления (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

dETm — время определения обрыва управления (мс) [DWORD]

= 5000 (по умолчанию). Если значение параметра меньше 100, контроль обрыва управления не выполняется.

Ym — тип выхода регулятора [BYTE]

= 0 - униполярный (от 0.0 до 100.0 %) (по умолчанию)

= 1 – биполярный (от -100.0 до 100.0 %).

Выходы:

Ok – статус работы регулятора [BYTE]

= 0 – остановлен,

= 1 – работает,

= 2 – работает в зоне нечувствительности,

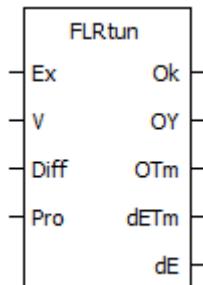
= 4 – ошибка! обрыв контура управления,

OY – выходной уровень регулятора (%) [REAL]

= 0.0 (по умолчанию)

При выборе времени квантования регулятора, возможно, потребуется учитывать время цикла выполнения пользовательского приложения. Время цикла задается вручную в настройках пользовательского проекта ИСР Beremiz.

Функциональный блок FLRtun



Функциональный блок «FLRtun» выполняет автоматическую настройку регулятора нечеткой логики.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

V – входное значение с источника сигнала [REAL]

= 0.0 (по умолчанию)

Diff – разностная уставка (в единицах значения V) [REAL]

= 0.0 (по умолчанию)

Pro – тип процесса [BYTE]

= 0 – увеличение (например, нагрев) (по умолчанию),

= 1 – уменьшение (например, охлаждение).

Выходы:

Ok – статус работы регулятора [BYTE]

= 0 – остановлен,

= 1 – работает,

= 3 – работа завершена успешно.

OY – выходной уровень регулятора (%) [REAL]

= 0.0 (по умолчанию),

= 100.0 (на всем протяжении настройки).

OTm – вычисленный период квантования (мсек) [DWORD]

= 0 (по умолчанию)

dETm – вычисленный период квантования (мсек) [DWORD]

= 0 (по умолчанию)

dE – значение зоны определения обрыва контура управления [REAL]

= 0 (по умолчанию)

Автонастройка регулятора организована по схеме «Реакция системы на единичное воздействие» (см. описание функционального блока PIDtun для пояснения):

- 1) на управляющий элемент подаётся максимальное управляющее воздействие 100.0 %,
- 2) определяется величина t_0 (время транспортного запаздывания), когда:

$$V = (0.1 \cdot Diff) + V_1$$

где,

V – текущее значение с источника сигнала,

V_1 – значение с источника сигнала в момент запуска настройки.

- 3) рассчитываются коэффициенты регулятора и время его квантования:

время квантования:

$$OT_m = t_0 \cdot 1.875 \text{ (мс)}$$

Нужно учитывать, что перед запуском автонастройки система должна быть выведена в штатный режим работы по остальным параметрам. Например, для вентиляционных установок перед автонастройкой регулятора секции нагрева, установка должна быть выведена в штатный режим по воздуху.

Запуск автонастройки FLR-регулятора выполняется подачей логической «1» (TRUE) на вход Ex, а подачей логического «0» - выключение после успешного завершения операции или прерывание во время работы.

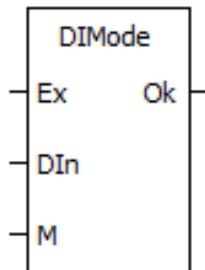
Состояние работы алгоритма контролируется по значению выхода Ok.

ПРИЛОЖЕНИЕ 12

БИБЛИОТЕКА КАНАЛОВ ДИСКРЕТНОГО ВВОДА

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_DI» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с каналами дискретного ввода. Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Функциональный блок DIMode



Функциональный блок «DIMode» выполняет автоматическую настройку канала дискретного ввода на определенный режим работы.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DIIn — номер канала дискретного ввода [BYTE]

= 0 ... N

M — режим работы [BYTE]

= 0 — нормальный,

= 1 — счетный,

= 2 — тахометр,

= 3 — инкрементальный энкодер (только счет),

= 4 — инкрементальный энкодер (счет и тахометр).

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — задан неверный режим работы канала,

= 4 — попытка задать режим «Инкрементальный энкодер» для недопустимого канала.

Для ПЛМ2004:

1. DIIn: 0 .. 7
2. M: 0 .. 4

Так как значение режима работы является энергонезависимым (сохраняется в EEPROM при изменении), то не рекомендуется частое изменение этого параметра. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) изменение режима не происходит,
- 2) возвращается код результата («Ok» = 0).

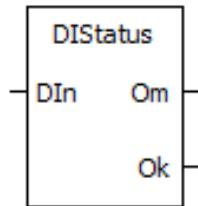
Если выполнение блока завершилось некорректно, то:

- 1) изменение режима не происходит,
- 2) в «Ok» возвращается код ошибки.

Изменение режима работы канала выполняется по окончании цикла работы пользовательского приложения. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Режим «Инкрементальный энкодер» задается только для каналов: 0, 2, 4, 6.

Функциональный блок DIStatus



Функциональный блок «DIStatus» возвращает текущий режим работы дискретного ввода.

Поддерживаемые устройства: ПЛМ2004.

Входы:

DIn – номер канала дискретного ввода [BYTE]
= 0 ... N

Выходы:

Om – установленный режим работы [BYTE]
= 0 — нормальный,
= 1 — счетный,
= 2 — тахометр,
= 3 — инкрементальный энкодер (только счет),
= 4 — инкрементальный энкодер (счет и тахометр).

Ok – код результата выполнения [BYTE]
= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала,
= 3 – задан неверный режим работы канала,
= 4 – попытка задать режим «Инкрементальный энкодер» для недопустимого канала.

Для ПЛМ2004:

3. Din: 0 .. 7

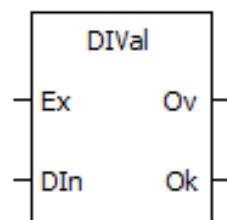
Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) изменение режима не происходит,
- 2) возвращает последнее переданное через блок значение («Om»),
- 3) возвращается код ошибки.

Если выполнение блока завершилось некорректно, то:

- 1) изменение режима не происходит,
- 2) возвращает значение по умолчанию («Om» = 0),
- 3) в «Ok» возвращается код ошибки.

Функциональный блок DIVal



Функциональный блок «DIVal» возвращает значение канала дискретного ввода, настроенного на режим «Нормальный».

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 = FALSE — нет
 = TRUE — есть
- DIn** — номер канала дискретного ввода [BYTE]
 = 0 ... N

Выходы:

- Ov** — значение дискретного ввода [BOOL]
 = FALSE — низкий уровень,
 = TRUE — высокий уровень.

- Ok** — код результата выполнения [BYTE]
 = 0 — выполнение блока запрещено (Ex = FALSE),
 = 1 — нет ошибок,
 = 2 — задан неверный номер канала,
 = 3 — режим канала не «Нормальный».

Для ПЛМ2004:

4. Din: 0 .. 7

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) на Ov установлено последнее переданное через блок значение.

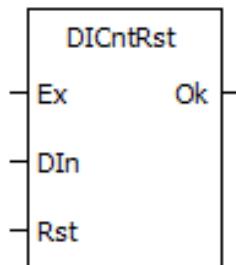
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) возвращается значение ввода по умолчанию («Ov» = FALSE),
- 2) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользователяского приложения.

Функциональный блок DICntRst



Функциональный блок «DICntRst» обнуляет значения счетчиков канала дискретного ввода, настроенного на режим «Счетный», «Тахометр» или «Инкрементный энкодер».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DIn — номер канала дискретного ввода [BYTE]

= 0 ... N

Rst — разрешение обнуления счетчиков канала [BOOL]

= FALSE — нет

= TRUE — есть

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — режим канала не «Счетный», не «Тахометр»,
не «Инкрементальный энкодер».

Для ПЛМ2004:

5. Din: 0 .. 7

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) ничего не выполняется,
- 2) возвращается код результата («Ok» = 0).

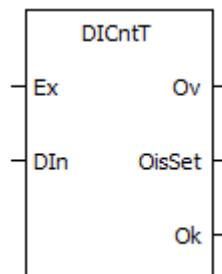
Если выполнение блока завершилось некорректно, то:

- 1) ничего не выполняется,

- 2) в «Ok» возвращается код ошибки.

Обнуление счетчиков для канала ввода и обновление выходов блока выполняется перед началом цикла работы пользовательского приложения. Пока вход «Rst» блока имеет значение TRUE, счетчик канала инкрементироваться не будет.

Функциональный блок DICntT



Функциональный блок «DICntT» возвращает значение канала дискретного ввода, настроенного на режим «Тахометр», а также управляет работой счетчика по уставке.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DIn — номер канала дискретного ввода [BYTE]

= 0 ... N

Выходы:

Ov — значение счетчика [WORD]

OisSet — признак достижения уставки [BOOL]

= FALSE — нет

= TRUE — есть

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — режим канала не «Тахометр».

Для ПЛМ2004:

6. DIn: 0 .. 7
7. Макс. частота подаваемых импульсов – 1 кГц.

Если нет разрешения выполнения («Ex» равно FALSE), то:

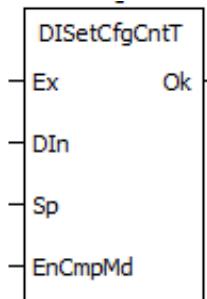
- 1) на Ov и OisSet возвращается последнее, переданное через блок значение.
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) возвращается значения по умолчанию («Ov» = 0, «OisSet» = FALSE),
- 2) в «Ok» возвращается код ошибки.

Счет импульсов выполняется по прерыванию и не зависит от цикла работы пользовательского приложения, поэтому значение счетчика доступно быстро. В этом же прерывании обрабатывается и условие равности уставке.

Функциональный блок DISetCfgCntT



Функциональный блок «DISetCfgCntT» задает параметры канала дискретного ввода, настроенного на режим «Тахометр». Данных блок так же необходим для конфигурации задачи по событию с тахометра.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- DIn** — номер канала дискретного ввода [BYTE]
 - = 0 ... N
- Sp** — уставка тахометра [WORD]
 - = 0 ... 65535 имп/сек
- EnCmpMd** — разрешение работы тахометра по уставке, запуск задач по событию при достижении уставки тахотемпа [BOOL]
 - = FALSE — нет
 - = TRUE — есть

Выходы:

- Ok** — код ошибки [BYTE]
 - = 0 — выполнение блока запрещено (Ex = FALSE),
 - = 1 — нет ошибок,
 - = 2 — задан неверный номер канала,
 - = 3 — режим канала не «Тахометр».

Для ПЛМ2004:

8. DIn: 0 .. 7
9. Макс. частота подаваемых импульсов – 1 кГц.

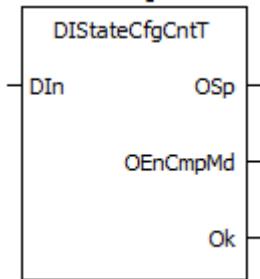
Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) в «Ok» возвращается код ошибки.

Функциональный блок DIStateCfgCntT



Функциональный блок «DIStateCfgCntT» возвращает значения параметров канала дискретного ввода, настроенного на режим «Тахометр».

Поддерживаемые устройства: ПЛМ2004.

Входы:

DIn – номер канала дискретного ввода [BYTE]
= 0 ... N

Выходы:

OSp – значение уставки тахометра [WORD]
OEnCmpMd – разрешение работы по уставке [BOOL]
 = FALSE — не разрешена
 = TRUE — разрешена
Ok – код ошибки [BYTE]
 = 0 – выполнение блока запрещено (Ex = FALSE),
 = 1 – нет ошибок,
 = 2 – задан неверный номер канала,
 = 3 – режим канала не «Тахометр».

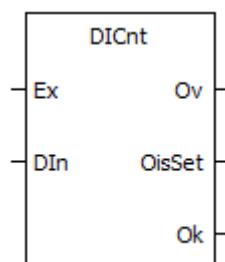
Для ПЛМ2004:

10. DIn: 0 .. 7

Если выполнение блока завершилось некорректно, то:

- 1) возвращается значения по умолчанию («OSp» = 0, «OEnCmpEn» = FALSE),
- 2) в «Ok» возвращается код ошибки.

Функциональный блок DICnt



Функциональный блок «DICnt» возвращает значение канала дискретного ввода, настроенного на режим «Счетный», а также управляет работой счетчика по уставке.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- DIn** — номер канала дискретного ввода [BYTE]
 - = 0 ... N

Выходы:

- Ov** — значение счетчика [DWORD]
- OisSet** — признак достижения уставки [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- Ok** — код результата выполнения [BYTE]
 - = 0 — выполнение блока запрещено (Ex = FALSE),
 - = 1 — нет ошибок,
 - = 2 — задан неверный номер канала,
 - = 3 — режим канала не «Счетный».

Для ПЛМ2004:

11. DIn: 0 .. 7
12. Макс. частота подаваемых импульсов – 1 кГц.

Если нет разрешения выполнения («Ex» равно FALSE), то:

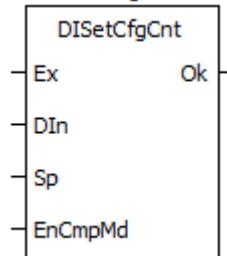
- 1) на Ov и OisSet возвращается последнее, переданное через блок значение.
- 2) в «Ok» возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) возвращаются значения по умолчанию («Ov» = 0, «OisSet» = FALSE),
- 2) в «Ok» возвращается код ошибки.

Счет импульсов выполняется по прерыванию и не зависит от цикла работы пользовательского приложения, поэтому значение счетчика доступно быстро. В этом же прерывании обрабатывается и условие равности уставке.

Функциональный блок DISetCfgCnt



Функциональный блок «DISetCfgCnt» задает параметры канала дискретного ввода, настроенного на режим «Счетный». Данных блок так же необходим для конфигурации задачи по событию со счетчика.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DIn — номер канала дискретного ввода [BYTE]

= 0 ... N

Sp — уставка счетчика [DWORD]

= 0 ... 4294967295 имп

EnCmpMd — разрешение работы счетчика по уставке, запуск задач по событию при достижении уставки счетчика [BOOL]

= FALSE — не разрешено

= TRUE — разрешено

Выходы:

Ok — код ошибки [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — режим канала не «Счетный».

Для ПЛМ2004:

13. DIn: 0 .. 7

14. Макс. частота подаваемых импульсов — 1 кГц.

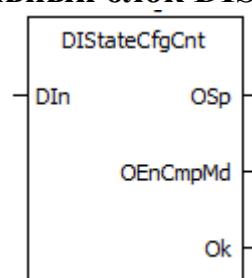
Если нет разрешения выполнения («Ex» равно FALSE), то:

1) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

1) в «Ok» возвращается код ошибки.

Функциональный блок DIStateCfgCnt



Функциональный блок «DIStateCfgCnt» возвращает значения параметров канала дискретного ввода, настроенного на режим «Счетный».

Поддерживаемые устройства: ПЛМ2004.

Входы:

DIn — номер канала дискретного ввода [BYTE]

= 0 ... N

Выходы:

OSp – значение уставки счетчика [DWORD]

OEnCmpMd – разрешение работы по уставке [BOOL]

= FALSE — не разрешена

= TRUE — разрешена

Ok – код ошибки [BYTE]

= 0 – выполнение блока запрещено (Ex = FALSE),

= 1 – нет ошибок,

= 2 – задан неверный номер канала,

= 3 – режим канала не «Счетный».

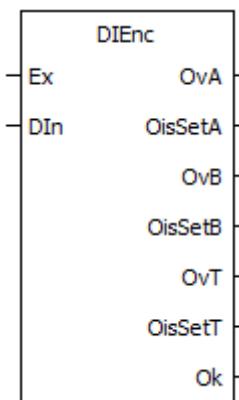
Для ПЛМ2004:

15. Din: 0 .. 7

Если выполнение блока завершилось некорректно, то:

- 1) возвращается значения по умолчанию («OSp» = 0, «OEnCmpEn» = FALSE),
- 2) в «Ok» возвращается код ошибки.

Функциональный блок DIEnc



Функциональный блок «DIEnc» возвращает значение канала дискретного ввода, настроенного на режим «Инкрементный энкодер», а также управляет работой счетчика по уставке.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DIn – номер группового канала дискретного ввода [BYTE]

= 0 – группа DI0-1,

= 2 – группа DI2-3,

= 4 – группа DI4-5,

= 6 – группа DI6-7

Выходы:

OvA – значение счетчика фазы «A» [DWORD]

OisSetA – признак достижения уставки счетчика фазы «A» [BOOL]

- = FALSE — нет
- = TRUE — есть

OvB – значение счетчика фазы «B» [DWORD]

OisSetB – признак достижения уставки счетчика фазы «B» [BOOL]

- = FALSE — нет
- = TRUE — есть

OvT – значение тахометра фазы «B» [WORD]

OisSetT – признак достижения уставки тахометра фазы «B» [BOOL]

- = FALSE — нет
- = TRUE — есть

Ok – код результата выполнения [BYTE]

- = 0 – выполнение блока запрещено (Ex = FALSE),
- = 1 – нет ошибок,
- = 2 – задан неверный номер канала,
- = 3 – режим канала не «Инкрементальный энкодер».

Для ПЛМ2004:

16. Din: 0 .. 7
17. Макс. частота подаваемых импульсов – 1 кГц.

Если нет разрешения выполнения («Ex» равно FALSE), то:

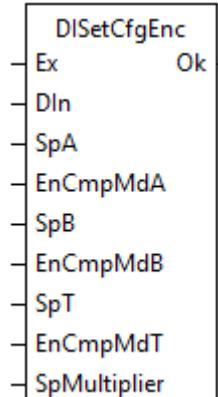
- 1) на OvX и OisSetX возвращается последнее, переданное через блок значение. (X – функциональное обозначение выхода).
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) возвращаются значения по умолчанию («OvX» = 0 «OisSetX» = FALSE, где X – функциональное обозначение выхода).
- 2) в «Ok» возвращается код ошибки.

Счет импульсов выполняется по прерыванию и не зависит от цикла работы пользовательского приложения, поэтому значение счетчика доступно быстро. В этом же прерывании обрабатывается и условие равности уставке.

Функциональный блок DISetCfgEnc



Функциональный блок «DIEnc» задает параметры канала дискретного ввода, настроенного на режим «Инкрементный энкодер». Данных блок так же необходим для конфигурации задачи по событию со счетчика и тахометра.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DIn — номер группового канала дискретного ввода [BYTE]

= 0 — группа DI0-1,

= 2 — группа DI2-3,

= 4 — группа DI4-5,

= 6 — группа DI6-7

SpA — уставка счетчика фазы «A» энкодера [DWORD]

= 0 ... 4294967295 имп.

EnCmpMdA — разрешение работы счетчика фазы «A» по уставке, запуск задач по событию при достижении уставки счетчика фазы «A» [BOOL]

= FALSE — не разрешена

= TRUE — разрешена

SpB — уставка счетчика фазы «B» энкодера [DWORD]

= 0 ... 4294967295 имп.

EnCmpMdB — разрешение работы счетчика фазы «B» по уставке, запуск задач по событию при достижении уставки счетчика фазы «B» [BOOL]

= FALSE — не разрешена

= TRUE — разрешена

SpT — уставка тахометра фазы «B» энкодера [WORD]

= 0 ... 65535 имп./сек

* для режима работы «Инкрементальный энкодер» (счетный и тахометр)

EnCmpMdT — разрешение работы тахометра фазы «B» по уставке, запуск задач по событию при достижении уставки тахометра [BOOL]

= FALSE — не разрешена

= TRUE — разрешена

SpMultiplier — множитель энкодера [BYTE]

= 1,

= 2,

= 4.

Выходы:

Ok — код ошибки [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — режим канала не «Инкрементальный энкодер».

Для ПЛМ2004:

18. Din: 0 .. 7

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) возвращается код результата («Ok» = 0).

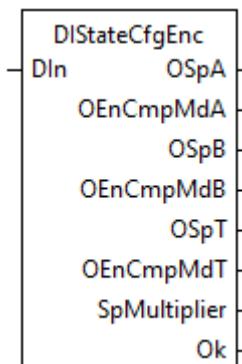
Если выполнение блока завершилось некорректно, то:

- 1) в «Ok» возвращается код ошибки.

Для конфигурации энкодера доступно 3 режима работы:

- 1x режим:
 - В этом случае счетчик, ведущий подсчет смещения энкодера будет прирастать и убавляться на 1 при каждом колебании так как фиксируется только фронты сигналов А и В.
- 2x режим:
 - В этом случае точность контроля удваивается. Ибо на каждый физически возможный импульс датчика фиксируется и момент перехода датчика А из 1 в 0 и из 0 в 1.
- 4x режим:
 - В этом режиме анализируются все 4 фронта (два фронта сигнала А и два фронта сигнала В). В этом случае точность измерений становится в 4 раза.

Функциональный блок DIStateCfgEnc



Функциональный блок «DIEnc» возвращает значения параметров канала дискретного ввода, настроенного на режим «Инкрементный энкодер».

Поддерживаемые устройства: ПЛМ2004.

Входы:

DIIn – номер группового канала дискретного ввода [BYTE]
 = 0 – группа DI0-1,
 = 2 – группа DI2-3,
 = 4 – группа DI4-5,
 = 6 – группа DI6-7

Выходы:

OSpA – значение уставки счетчика фазы «А» [DWORD]
OEnCmpMdA – разрешение работы по уставке счетчика фазы «А» [BOOL]
 = FALSE — не разрешена
 = TRUE — разрешена
OSpB – значение счетчика фазы «В» [DWORD]

OEnCmpMdB – разрешение работы по уставке счетчика фазы «B» [BOOL]
= FALSE — не разрешена
= TRUE — разрешена

OSpT – значение тахометра фазы «B» [WORD]

OEnCmpMdT – разрешение работы по уставке тахометра фазы «B» [BOOL]
= FALSE — не разрешена
= TRUE — разрешена

SpMultiplier – множитель энкодера [BYTE]

= 1,
= 2,
= 4.

Ok – код ошибки [BYTE]

= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала,
= 3 – режим канала не «Инкрементальный энкодер».

Для ПЛМ2004:

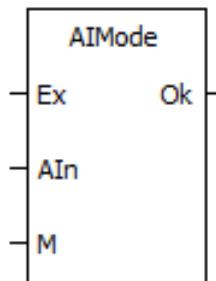
19. Din: 0 .. 7

Если выполнение блока завершилось некорректно, то:

- 1) возвращается значения по умолчанию («OSpX» = 0 «OEnCmpMdX» = FALSE, где X – функциональное обозначение выхода).
- 2) в «Ok» возвращается код ошибки.

ПРИЛОЖЕНИЕ 13**БИБЛИОТЕКА КАНАЛОВ АНАЛОГОВОГО ВВОДА**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_AI» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с каналами аналогового ввода. Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Функциональный блок AIMode

Функциональный блок «AIMode» позволяет установить режим работы канала, а также возвращает код результата исполнения блока.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

AIn — номер канала аналогового ввода [BYTE]

= 0 .. N

M — режим работы [BYTE]

= 0 — опрос,

= 2 — выключен.

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — задан неверный режим работы.

Для ПЛМ2004:

20. AIn: 0 .. 1

21. M: 0, 2

Если нет разрешения выполнения («Ex» равно FALSE), то:

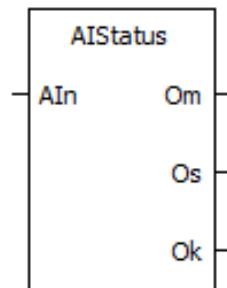
1) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

1) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок AIStatus



Функциональный блок «AIStatus» возвращает:

- код установленного режима
- код состояния входа
- код результата исполнения блока.

Поддерживаемые устройства: ПЛМ2004.

Входы:

AIn – номер канала аналогового ввода [BYTE]
= 0 .. N

Выходы:

Om – установленный режим работы [BYTE]
= 0 – опрос,
= 1 – калибровка (только чтение),
= 2 – выключен.

Os – код состояния входа: [BYTE]
= 0 – опрос,
= 1 – калибровка,
= 11 – калибровка завершена успешно,
= 2 – выключен.

Ok – код результата выполнения [BYTE]
= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала,
= 3 – задан неверный режим работы.

Для ПЛМ2004:

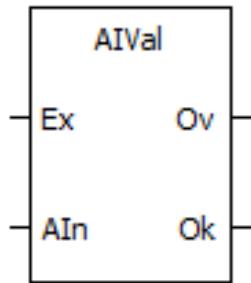
22. AIn: 0 .. 1
23. M: 0, 2

Если выполнение блока завершилось некорректно, то:

- 1) изменение режима не происходит,
- 2) возвращает значение по умолчанию («Om» = 0),
- 3) возвращает значение по умолчанию («Os» = 0),
- 4) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок AIVal



Функциональный блок «AI» возвращает значение канала аналогового ввода.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть

AIn — номер канала аналогового ввода [BYTE]
= 0 .. N

Выходы:

Ov — значение канала [REAL]
= 0.0 ... 10.0 В

Ok — код результата выполнения [BYTE]
= 1 — нет ошибок,
= 2 — задан неверный номер канала,
= 3 — задан неверный режим работы.

Для ПЛМ2004:

24. AIn: 0 .. 1

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) возвращает последнее, переданное через блок значение («Ov»),
- 2) возвращается код результата («Ok» = 0).

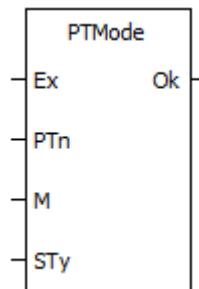
Если выполнение блока завершилось некорректно, то:

- 1) возвращает значение по умолчанию («Ov» = 0),
- 2) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

ПРИЛОЖЕНИЕ 14**БИБЛИОТЕКА КАНАЛОВ РЕЗИСТИВНЫХ ДАТЧИКОВ**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_PT» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с каналами резистивных датчиков. Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Функциональный блок PtMode

Функциональный блок «PtMode» устанавливает режим работы и тип чувствительного элемента датчика, а также возвращает код выполнения блока.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть

PTn — номер входа резистивного датчика [BYTE]
= 0 ... N

M — режим работы [BYTE]
= 0 — опрос,
= 2 — выключен.

STy — код типа чувствительного элемента [BYTE]
= 0 — PT3850 (температурный (альфа) коэффициент 0.00385 0C-1),
= 1 — PT3911 (температурный (альфа) коэффициент 0.003911 0C-1).

Выходы:

Ok — код результата выполнения [BYTE]
= 0 — выполнение блока запрещено (Ex = FALSE),
= 1 — нет ошибок,
= 2 — задан неверный номер канала,
= 3 — задан неверный режим работы,
= 5 — задан неверный тип чувствительного элемента.

Для ПЛМ2004:

- 25. PTn: 0 .. 3
- 26. M: 0, 2
- 27. STy: 0, 1

Так как настраиваемые параметры являются энергонезависимыми (сохраняются в EEPROM при изменении), то не рекомендуется их частое изменение. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

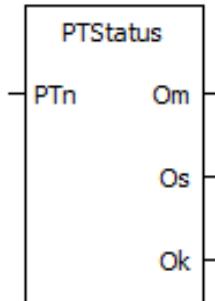
- 1) изменение режима не происходит,
- 2) изменение типа чувствительного элемента не происходит,
- 3) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) изменение режима не происходит,
- 2) изменение типа чувствительного элемента не происходит,
- 3) в «Ok» возвращается код ошибки.

Изменение режим и типа чувствительного элемента выполняется по окончании цикла работы пользовательского приложения. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок PtStatus



Функциональный блок «PtStatus» возвращает результат:

- код установленного режима работы
- код состояния
- код выполнения блока.

Поддерживаемые устройства: ПЛМ2004.

Входы:

PTn – номер входа резистивного датчика [BYTE]
= 0 ... N

Выходы:

Om – установленный режим работы [BYTE]
= 0 – опрос,
= 1 – калибровка (только чтение),
= 2 – выключен.

Os – код состояния входа: [BYTE]

- = 0 – опрос,
- = 1 – калибровка,
- = 11 – калибровка завершена успешно,
- = 2 – выключен,
- = 3 – измеренное значение ниже минимального предела,
- = 4 – измеренное значение выше максимального предела.

Ok – код результата выполнения [BYTE]

- = 1 – нет ошибок,
- = 2 – задан неверный номер канала,
- = 3 – задан неверный режим работы,
- = 5 – задан неверный тип чувствительного элемента.

Для ПЛМ2004:

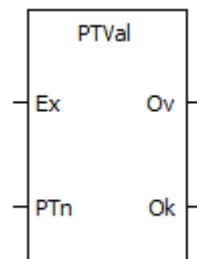
28. PTn: 0 .. 3
29. M: 0, 2

Если выполнение блока завершилось некорректно, то:

- 1) изменение режима не происходит,
- 2) изменение типа чувствительного элемента не происходит,
- 3) возвращает значение по умолчанию («Om» = 0),
- 4) возвращает значение по умолчанию («Os» = 0),
- 5) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок PtVal



Функциональный блок «PtVal» возвращает значение резистивного датчика.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

- = FALSE — нет
- = TRUE — есть

PTn — номер канала резистивного датчика [BYTE]

- = 0 ... N

Выходы:

Ov – значение канала [REAL]
= -60.0 ... 160.0 (опрос и нет ошибок),
= 0.0 (калибровка, калибровка завершена успешно).

Ok – код результата выполнения [BYTE]
= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала,
= 3 – задан неверный режим работы,
= 5 – задан неверный тип чувствительного элемента.

Для ПЛМ2004:

30. PTn: 0 .. 3

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) возвращает последнее, переданное через блок значение («Ov»),
- 2) возвращается код результата («Ok» = 0).

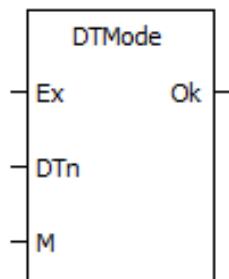
Если выполнение блока завершилось некорректно, то:

- 1) возвращает значение по умолчанию («Ov» = 0),
- 2) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

ПРИЛОЖЕНИЕ 15**БИБЛИОТЕКА КАНАЛОВ ДАТЧИКОВ DS18N20 ШИНЫ 1-WIRE**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_DT» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с каналами датчиков DS18N20 шины 1-Wire. Алгоритмы аппаратно-зависимые: поддерживаемые устройства и особенности указаны в описании.

Функциональный блок DtMode

Функциональный блок «DtMode» устанавливает режим работы датчика и код результата исполнения блока.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DTn — номер канала датчика [BYTE]

= 0 ... N

M — режим работы [BYTE]

= 0 — опрос,

= 1 — поиск,

= 2 — выключен.

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — задан неверный режим работы.

Для ПЛМ2004:

31. DTn: 0 .. 9

32. M: 0 .. 2

Так как настраиваемые параметры является энергонезависимыми (сохраняются в EEPROM при изменении), то не рекомендуется их частое изменение. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

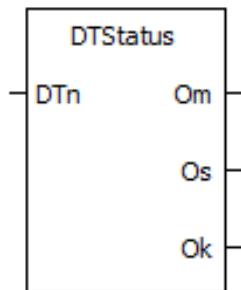
- 1) изменение режима и/или идентификатора не происходит,
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) никаких изменений не происходит,
- 2) в «Ok» возвращается код ошибки.

Изменение настроек выполняется по окончании цикла работы пользовательского приложения. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок DtStatus



Функциональный блок «DtStatus» возвращает установленные режим работы, код состояния входа и код результата исполнения блока.

Поддерживаемые устройства: ПЛМ2004.

Входы:

DTn – номер канала датчика [BYTE]
= 0 ... N

Выходы:

Om – установленный режим работы [BYTE]
= 0 – опрос,
= 1 – поиск,
= 2 – выключен.

Os – код состояния входа: [BYTE]
= 0 – опрос,
= 1 – поиск,
= 11 – поиск завершен успешно,
= 2 – выключен,
= 3 – нет ни одного датчика на шине 1-wire,
= 4 – нет датчика с заданным ID на шине 1-wire,
= 5 – попытка опроса датчика без ID,
= 6 – неисправность питания датчика.

Ok – код результата выполнения [BYTE]
= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала,
= 3 – задан неверный режим работы.

Для ПЛМ2004:

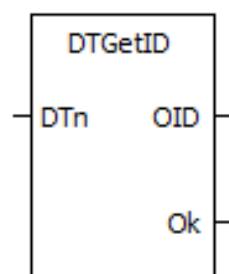
- 33. DTn: 0 .. 9
- 34. M: 0 .. 2

Если выполнение блока завершилось некорректно, то:

- 1) никаких изменений не происходит,
- 2) возвращает значение по умолчанию ($\ll O_m \gg = 0$),
- 3) возвращает значение по умолчанию ($\ll O_s \gg = 0$),
- 4) в $\ll Ok \gg$ возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок DtGetID



Функциональный блок «DtGetID» возвращает установленный идентификатор датчика.

Поддерживаемые устройства: ПЛМ2004.

Входы:

DTn – номер канала датчика [BYTE]
= 0 ... N

Выходы:

OID – код установленного идентификатора датчика [LWORD]

Ok – код результата выполнения [BYTE]
= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала.

Для ПЛМ2004:

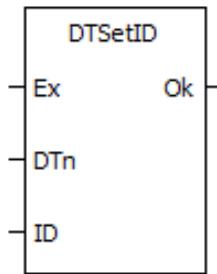
- 35. DTn: 0 .. 9

Если выполнение блока завершилось некорректно, то:

- 1) возвращает значение по умолчанию ($\ll OID \gg = 0$),
- 2) в $\ll Ok \gg$ возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок DtSetID



Функциональный блок «DtSetID» устанавливает идентификатор датчика.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DTn — номер канала датчика [BYTE]

= 0 ... N

ID — код идентификатора датчика [LWORD]

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала.

Для ПЛМ2004:

36. DTn: 0 .. 9

Если нет разрешения выполнения («Ex» равно FALSE), то:

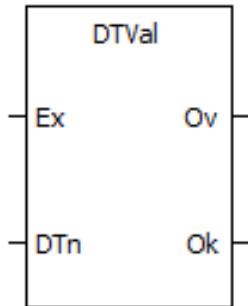
- 1) изменение идентификатора не происходит,
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) изменение идентификатора не происходит,
- 2) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок DtVal



Функциональный блок «DtVal» возвращает значение температуры датчика, код состояния и код результата исполнения блока.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DTn — номер канала датчика [BYTE]

= 0 ... N

Выходы:

Ov — значение температуры (C°) [REAL]

= -55.0 ... 125.0 (опрос и нет ошибок),

= 0.0 (поиск, поиск завершен успешно, канал датчика выключен),

= -500.0 (нет ни одного датчика на шине 1-wire),

= -600.0 (нет датчика с заданным ID на шине 1-wire),

= -700.0 (попытка опроса датчика без ID),

= 127.0 (неисправность питания датчика).

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — задан неверный режим работы.

Для ПЛМ2004:

37. DTn: 0 .. 9

Если при опросе возникла ошибка, то в значении температуры (C°) будет записано число больше диапазона измерения датчика.

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) возвращает последнее, переданное через блок значение («Ov»),
- 2) возвращается код результата («Ok» = 0).

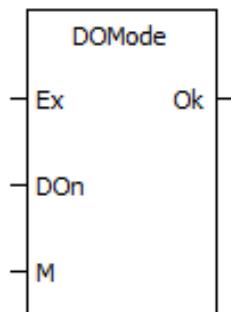
Если выполнение блока завершилось некорректно, то:

- 1) возвращает значение по умолчанию («Ov» = 0),
- 2) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

ПРИЛОЖЕНИЕ 16**БИБЛИОТЕКА КАНАЛОВ ДИСКРЕТНОГО ВЫВОДА**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_DO» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с каналами дискретного вывода. Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Функциональный блок DOMode

Функциональный блок «DOMode» устанавливает режим работы канала дискретного вывода.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DOn — номер канала [BYTE]

= 0 ... N

M — режим работы [BYTE]

= 0 — нормальный,

= 1 — быстрый,

= 2 — ШИМ,

= 3 — шаг ШД (step),

= 4 — направление ШД (dir),

= 5 — выключен.

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — задан неверный режим работы.

Для ПЛМ2004:

38. DOn: 0 .. 7

39. M: 0 .. 3

Так как настраиваемые параметры являются энергонезависимыми (сохраняются в EEPROM при изменении), то не рекомендуется их частое изменение. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

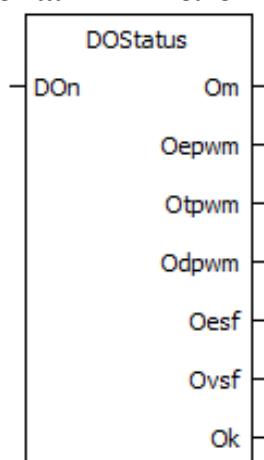
- 1) изменение режима не происходит,
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) никаких изменений не происходит,
- 2) в «Ok» возвращается код ошибки.

Изменение режима работы канала выполняется по окончании цикла работы пользовательского приложения. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок DOStatus



Функциональный блок «DOStatus» возвращает установленные режим работы.

Поддерживаемые устройства: ПЛМ2004.

Входы:

DOn – номер канала [BYTE]
= 0 ... N

Выходы:

Om – установленный режим работы [BYTE]
= 0 – нормальный,
= 1 - быстрый,
= 2 – ШИМ,
= 3 – шаг ШД (step),
= 4 – направление ШД (dir),
= 5 – выключен.

Oerwm – разрешение работы ШИМ [BOOL]
= FALSE — не разрешено
= TRUE — разрешено

Otpwm – период ШИМ (мс) [DWORD]

Odpwm – заполнение ШИМ (%) [REAL]

Oesf – разрешение монитора безопасного состояния [BOOL]

= FALSE — не разрешено

= TRUE — разрешено

Ovsf – значение уровня безопасного состояния [BOOL]

= FALSE — лог. 0

= TRUE — лог. 1

Ok – код результата выполнения [BYTE]

= 0 – выполнение блока запрещено (Ex = FALSE),

= 1 – нет ошибок,

= 2 – задан неверный номер канала,

= 3 – задан неверный режим работы.

Для ПЛМ2004:

40. DOn: 0 .. 7

41. M: 0 .. 3

Если нет разрешения выполнения («Ex» равно FALSE), то:

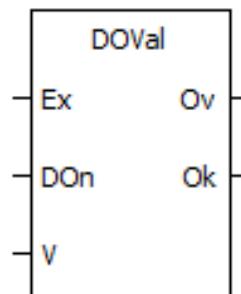
- 1) изменение режима не происходит,
- 2) возвращает последние, переданные через блок значения,
- 3) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) никаких изменений не происходит,
- 2) возвращаются значения по умолчанию (= 0),
- 3) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользователя приложения.

Функциональный блок DOVal



Функциональный блок «DOVal» устанавливает уровень выходного канала для режима работы «Нормальный» и возвращает результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

DOn — номер канала [BYTE]

= 0 ... N

V — уровень канала [BOOL]

= FALSE — низкий

= TRUE — высокий

Выходы:

Ov — установленный уровень канала [BOOL]

= FALSE — низкий

= TRUE — высокий

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — режим канала не «Нормальный»,

= 4 — канал переведен в безопасное состояние.

Для ПЛМ2004:

42. DOn: 0 .. 7

Если нет разрешения выполнения («Ex» равно FALSE), то:

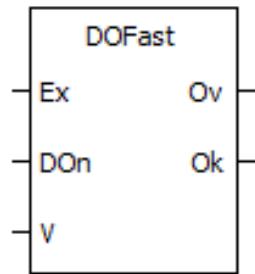
- 1) ничего не выполняется,
- 2) возвращает последнее, переданное через блок значение («Ov»),
- 3) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) ничего не выполняется,
- 2) возвращается значение по умолчанию («Ov» = FALSE),
- 3) в «Ok» возвращается код ошибки.

Выдача уровня на физический канал вывода выполняется по завершении цикла работы пользовательского приложения. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок DOFast



Функциональный блок «DOFast» устанавливает уровень выходного канала для режима работы «Быстрый» и возвращает результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 = FALSE — нет команды на применение быстрого выхода
 = TRUE — есть команда на применение быстрого выхода
- DOn** — номер канала [BYTE]
 = 0 ... N
- V** — уровень канала [BOOL]
 = FALSE — низкий
 = TRUE — высокий

Выходы:

- Ov** — установленный уровень канала [BOOL]
 = FALSE — низкий
 = TRUE — высокий
- Ok** — код результата выполнения [BYTE]
 = 0 – выполнение блока запрещено (Ex = FALSE),
 = 1 – нет ошибок,
 = 2 – задан неверный номер канала,
 = 3 – режим канала не «Быстрый»,
 = 4 – канал переведен в безопасное состояние.

Для ПЛМ2004:

43. DOn: 0 .. 7

Если нет разрешения выполнения («Ex» равно FALSE), то:

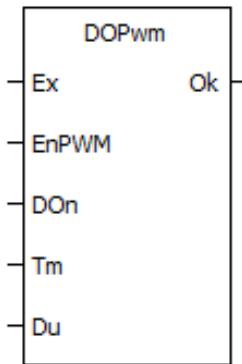
- 1) ничего не выполняется,
- 2) возвращает последнее, переданное через блок значение («Ov»),
- 3) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) ничего не выполняется,
- 2) возвращается значение по умолчанию («Ov» = FALSE),
- 3) в «Ok» возвращается код ошибки.

Алгоритм быстрых выходов выполняется по прерыванию системного таймера и не зависит от цикла работы пользовательского приложения, поэтому выдача уровня на физический канал вывода происходит быстро. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок DOPwm



Функциональный блок «DOPwm» устанавливает настройки выходного канала для режима работы «ШИМ» и возвращает результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- EnPWM** — управление каналом ШИМ [BOOL]
 - = FALSE — выключен
 - = TRUE — включен
- DOn** — номер канала [BYTE]
 - = 0 ... N
- Tm** — период ШИМ [DWORD]
 - = 100 ... 4294967295 мс
- Du** — коэффициент заполнения ШИМ [REAL]
 - = 0.0 ... 100.0 %

Выходы:

- Ok** — код результата выполнения [BYTE]
 - = 0 — выполнение блока запрещено (Ex = FALSE),
 - = 1 — нет ошибок,
 - = 2 — задан неверный номер канала,
 - = 3 — канал не настроен на режим «ШИМ»,
 - = 4 — канал переведен в безопасное состояние.

Для ПЛМ2004:

44. DOn: 0 .. 7

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) блок не исполняется,
- 2) ШИМ остается в ранее настроенном режиме работы.
- 3) возвращается код результата («Ok» = 0).

Если ШИМ отключен («EnPWM» равно FALSE), то:

- 1) ШИМ для заданного канала останавливается,
- 2) выходное значение канала устанавливается в логический «0».

3) возвращается код результата («Ok» = 1).

Если канал переведен системой в безопасное состояние, то:

- 1) ничего не выполняется,
- 2) ШИМ выключается,
- 3) возвращается код результата («Ok» = 4).

Если выполнение блока завершилось некорректно, то:

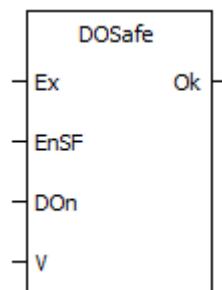
- 1) ничего не выполняется,
- 2) ШИМ выключается,
- 3) в «Ok» возвращается код ошибки.

Алгоритм ШИМ выполняется по прерыванию системного таймера и не зависит от цикла работы пользовательского приложения, поэтому выдача уровня на физический канал вывода происходит быстро. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Данный блок можно использовать в связке с блоком регулятора «PID» или «FLR»:

45. выход «OY» блока регулятора подключать на вход «D» блока «DoPwm»,
46. период квантования регулятора (вход «Tm») подключать на вход «Tm» блока «DoPwm».

Функциональный блок DOSafe



Функциональный блок «DOSafe» настраивает безопасное состояние выходного канала и возвращает результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

EnSF — разрешение безопасного состояния для канала [BOOL]

= FALSE — монитор безопасного состояния выключен

= TRUE — монитор безопасного состояния включен

DOn — номер канала [BYTE]

= 0 ... N

V — уровень безопасного состояния канала [BOOL]

= FALSE — низкий

= TRUE — высокий

Выходы:

- Ok** – код результата выполнения [BYTE]
= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала,
= 3 – задан неверный режим работы канала.

Для ПЛМ2004:

47. DOn: 0 .. 7

Так как настраиваемые параметры являются энергонезависимыми (сохраняются в EEPROM при изменении), то не рекомендуется их частое изменение. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) ничего не выполняется,
- 2) возвращается код результата («Ok» = 0).

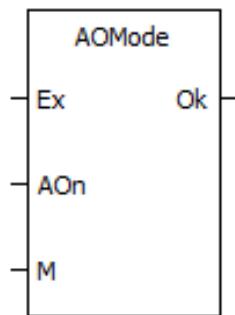
Если выполнение блока завершилось некорректно, то:

- 1) никаких изменений не происходит,
- 2) в «Ok» возвращается код ошибки.

Изменение настроек безопасного состояния канала выполняется по окончании цикла работы пользовательского приложения. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

ПРИЛОЖЕНИЕ 17**БИБЛИОТЕКА КАНАЛОВ АНАЛОГОВОГО ВЫВОДА**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_AO» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с каналами аналогового вывода. Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Функциональный блок AOMode

Функциональный блок «AOMode» устанавливает режим работы канала дискретного вывода и возвращает код результата

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

AOn — номер канала [BYTE]

= 0 ... N

M — режим работы [BYTE]

= 0 — нормальный,

= 1 - быстрый,

= 3 — выключен.

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — задан неверный режим работы.

Для ПЛМ2004:

48. AOn: 0 .. 4

49. M: 0, 1, 3

Так как настраиваемые параметры являются энергонезависимыми (сохраняются в EEPROM при изменении), то не рекомендуется их частое изменение. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

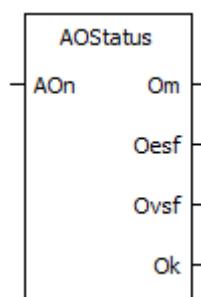
- 1) изменение режима не происходит,
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) никаких изменений не происходит,
- 2) в «Ok» возвращается код ошибки.

Изменение режима работы канала выполняется по окончании цикла работы пользовательского приложения. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок AOStatus



Функциональный блок «AOStatus» возвращает установленные режим работы.

Поддерживаемые устройства: ПЛМ2004.

Входы:

AOn – номер канала [BYTE]
= 0 ... N

Выходы:

Om – установленный режим работы [BYTE]
= 0 – нормальный,
= 1 - быстрый,
= 3 – выключен.

Oesf – разрешение монитора безопасного состояния [BOOL]
= FALSE — не разрешено
= TRUE — разрешено

Ovsf – значение уровня безопасного состояния [REAL]

Ok – код результата выполнения [BYTE]
= 0 – выполнение блока запрещено (Ex = FALSE),
= 1 – нет ошибок,
= 2 – задан неверный номер канала,
= 3 – задан неверный режим работы.

Для ПЛМ2004:

50. AOn: 0 .. 4
51. M: 0, 1, 3

Так как настраиваемые параметры являются энергонезависимыми (сохраняются в EEPROM при изменении), то не рекомендуется их частое изменение. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

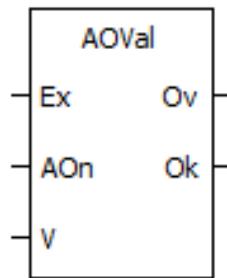
- 3) изменение режима не происходит,
- 4) возвращает последние, переданные через блок значения,
- 5) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 3) никаких изменений не происходит,
- 4) возвращаются значения по умолчанию (= 0),
- 5) в «Ok» возвращается код ошибки.

Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок AOVal



Функциональный блок «AOVal» устанавливает уровень выходного канала для режима работы «Нормальный» и возвращает результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

AOn — номер канала [BYTE]

= 0 ... N

V — уровень канала [REAL]

Выходы:

Ov — установленный уровень канала [REAL]

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — режим работы не «Нормальный»,

= 4 — канал переведен в безопасное состояние.

Для ПЛМ2004:

52. AOn: 0 .. 4

53. V: 0.0 .. 10.0 В

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) ничего не выполняется,
- 2) возвращает последнее, переданное через блок значение («Ov»),
- 3) возвращается код результата («Ok» = 0).

Если канал переведен системой в безопасное состояние, то:

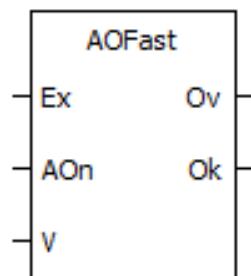
- 1) ничего не выполняется,
- 2) возвращается безопасное состояние («Ov»),
- 3) возвращается код результата («Ok» = 4).

Если выполнение блока завершилось некорректно, то:

- 1) ничего не выполняется,
- 2) возвращается значение по умолчанию («Ov» = 0.0),
- 3) в «Ok» возвращается код ошибки.

Выдача уровня на физический канал вывода выполняется по завершении цикла работы пользователяского. Обновление выходов блока производится перед началом цикла работы пользователяского приложения.

Функциональный блок AOFast



Функциональный блок «AOFast» устанавливает уровень выходного канала для режима работы «Быстрый» и возвращает результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 = FALSE — нет команды на применение быстрого выхода
 = TRUE — есть команда на применение быстрого выхода
- AOn** — номер канала [BYTE]
 = 0 ... N
- V** — уровень канала [REAL]

Выходы:

- Ov** — установленный уровень канала [REAL]
- Ok** — код результата выполнения [BYTE]
 = 0 — выполнение блока запрещено (Ex = FALSE),
 = 1 — нет ошибок,

- = 2 – задан неверный номер канала,
- = 3 – режим работы не «Быстрый»,
- = 4 – канал переведен в безопасное состояние.

Для ПЛМ2004:

54. AOn: 0 .. 4
55. V: 0.0 .. 10.0 В

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) ничего не выполняется,
- 2) возвращает последнее, переданное через блок значение («Ov»),
- 3) возвращается код результата («Ok» = 0).

Если канал переведен системой в безопасное состояние, то:

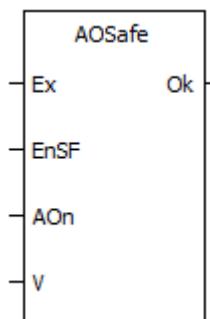
- 1) ничего не выполняется,
- 2) возвращается безопасное состояние («Ov»),
- 3) возвращается код результата («Ok» = 4).

Если выполнение блока завершилось некорректно, то:

- 1) ничего не выполняется,
- 2) возвращается значение по умолчанию («Ov» = 0.0),
- 3) в «Ok» возвращается код ошибки.

Алгоритм быстрых выходов выполняется по прерыванию системного таймера и не зависит от цикла работы пользовательского приложения, поэтому выдача уровня на физический канал вывода происходит быстро. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

Функциональный блок AOSafe



Функциональный блок «AOSafe» настраивает безопасное состояние выходного канала и возвращает результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
= FALSE — нет

= TRUE — есть

EnSF — разрешение безопасного состояния для канала [BOOL]

= FALSE — монитор безопасного состояния выключен

= TRUE — монитор безопасного состояния включен

AOn — номер канала [BYTE]

= 0 ... N

V — уровень канала [REAL]

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE),

= 1 — нет ошибок,

= 2 — задан неверный номер канала,

= 3 — задан неверный режим работы канала.

Для ПЛМ2004:

56. AOn: 0 .. 4

57. V: 0.0 .. 10.0 В

Так как настраиваемые параметры является энергонезависимыми (сохраняются в EEPROM при изменении), то не рекомендуется их частое изменение. Например, можно вызывать этот блок в момент инициализации (на 1-м цикле работы приложения).

Если нет разрешения выполнения («Ex» равно FALSE), то:

- 1) ничего не выполняется,
- 2) возвращается код результата («Ok» = 0).

Если выполнение блока завершилось некорректно, то:

- 1) никаких изменений не происходит,
- 2) в «Ok» возвращается код ошибки.

Изменение настроек безопасного состояния канала выполняется по окончании цикла работы пользовательского. Обновление выходов блока производится перед началом цикла работы пользовательского приложения.

ПРИЛОЖЕНИЕ 18**Библиотека Modbus RTU master**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_MbRtu_Master» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с ModBus RTU в режиме «Master». Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Нужно отметить, что блоки, входящие в данную библиотеку отрабатываются в несколько проходов цикла выполнения программы пользователя. Т.е. они могут быть использованы только в циклических задачах.

Каждый из входящих в библиотеку блоков может возвращать следующие коды на выходе Ok:

Коды процесса исполнения

Табл. 18.1

| Значение кода (DEC) | Описание | Комментарий |
|---------------------|---|---|
| 0 | Выполнение блока запрещено | На Ex подан 0, поэтому выполнение запрещено и запрос не формируется |
| 1 | Запрос ожидает выполнения | |
| 2 | Запрос добавлен в очередь на выполнение | |
| 3 | Запрос выполняется | |
| 4 | Запрос выполнен успешно | |

Коды ошибок формирования запроса

Данные коды могут возникнуть только на этапе формирования запроса от ведущего устройства, соответственно в случае их возникновения запрос ведомому устройству не отправляется.

Табл. 18.2

| Значение кода (DEC) | Описание | Комментарий |
|---------------------|-------------------------------|--|
| 10 | Задан неверный СОМ-порт | Выбранное целевое устройство не поддерживает СОМ порт с таким номером. |
| 11 | СОМ-порт не в режиме «Master» | Указанный СОМ-порт целевого устройства не в режиме «Master» |
| 12 | Задан неверный код функции | Указанный код функции протокола ModBus не соответствует назначению блока или не поддерживается целевым устройством |

| Значение кода (DEC) | Описание | Комментарий |
|----------------------------|--|--|
| 13 | Неверное значение числа обрабатываемых регистров. | Указанное значение выходит за рамки допустимого диапазона. |
| 14 | Ведомое устройство не отвечает в течении заданного таймаута ожидания ответа. | |

Коды ошибок обработки ответа

Данные коды могут возникнуть только на этапе обработки ответа от ведомого устройства на запрос, т.е. если от ведомого получена некая последовательность байт до истечения таймаута ожидания ответа.

Значения кодов группы 2x формируются ведомым устройством как ответ на распознанную ошибку в соответствии со спецификацией на протокол ModBus.

Значения кодов группы 2xx формируются мастером в том случае если полученную от ведомого устройства последовательность байт не удалось интерпретировать как стандартный ответ или возникла какая-то внутренняя системная ошибка выполнения на стороне мастера.

Табл. 18.3

| Значение кода (DEC) | Описание | Комментарий |
|----------------------------|---|--|
| 21 | Принятый код функции не может быть обработан. | Соответствует коду исключений протокола ModBus: 01 ILLEGAL FUNCTION |
| 22 | Адрес регистра, указанный в запросе, недоступен. | Соответствует коду исключений протокола ModBus: 02 ILLEGAL DATA ADDRESS |
| 23 | Значение, содержащееся в поле данных запроса, является недопустимой величиной. | Соответствует коду исключений протокола ModBus: 03 ILLEGAL DATA VALUE |
| 24 | Неверное значение числа обрабатываемых регистров. | Соответствует коду исключений протокола ModBus: 04 SERVER DEVICE FAILURE |
| 25 | Ведомое устройство приняло запрос и обрабатывает его, но это требует много времени. | Соответствует коду исключений протокола ModBus: 05 ACKNOWLEDGE |
| 26 | Ведомое устройство занято обработкой команды. | Соответствует коду исключений протокола ModBus: 06 SERVER DEVICE BUSY |

| Значение кода (DEC) | Описание | Комментарий |
|---------------------|---|--|
| 200 | В поле Slave ID (сетевой адрес ведомого устройства) ответа находится некорректное значение | |
| 201 | Зарезервировано | |
| 202 | Зарезервировано | |
| 203 | В поле Quantity of Data ответа находится некорректное значение | |
| 204 | В поле Function ответа находится некорректное значение | |
| 205 | Не совпадают типы регистров в запросе и ответе | |
| 206 | В поле Starting Address ответа находится некорректное значение | |
| 250 | Неверное значение контрольной суммы в ответе | |
| 251 | Переполнение приемного буфера | Количество байт в полученной последовательности превышает емкость приемного буфера |
| 254 | Слишком мало байт в ответе | Количество байт в полученной последовательности меньше минимально возможной для фрейма ответа протокола ModBus |
| 255 | Внутренняя системная ошибка обработки данных на стороне мастера | |

Общие принципы работы и использования блоков библиотеки.

Любой из блоков формирования запросов по протоколу ModBus библиотеки может использоваться в одном из двух режимов работы задаваемых уровнем на входе Mode блока.

Mode = 0 (по умолчанию): Циклический режим формирования запроса:

На управляющий вход блока должен быть подан уровень TRUE. Будет выполнено формирование запроса и его отправка в физическую линию обмена данными с ведомым устройством.

После формирования на выходе Ok блока кода результата выполнения большего или равного 4, цикл будет повторен вновь.

Цикл формирования и отправки будет повторяться до тех пор, пока на входе Ex блока присутствует уровень TRUE.

Mode = 1: Одиночный режим формирования запроса:

На управляющий вход блока, находящийся в FALSE, должен быть подан уровень TRUE.

Будет выполнено формирование запроса и его отправка в физическую линию обмена данными с ведомым устройством.

После формирования на выходе Ok блока кода результата выполнения большего или равного 4, состояние блока будет находиться в неизменном состоянии до тех пор, пока на вход Ex блока не будет подано FALSE, а затем TRUE с удержанием его в этом состоянии.

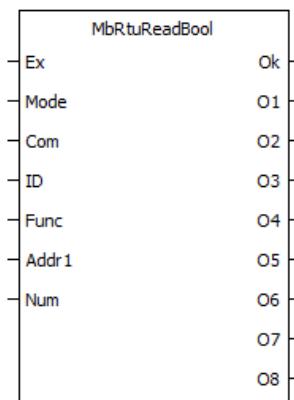
Необходимо отметить, что обстоятельство, что выполнение действий внутри блоков выполнения запросов от инициации запроса до получения кода результата его выполнения, производится в несколько проходов пользовательского приложения, так как каждый из сформированных запросов ставится в очередь и выполняется по мере освобождения интерфейсной шины данных.

При установленном на входе Ex состоянии FALSE, на выходе Ok формируется и удерживается значение 0, так же как и на выходах блока, предназначенных для хранения полученных от ведомого устройства данных.

Доступные интерфейсные порты для поддерживаемых целевых устройств. Вход блоков Com.

ПЛМ2004: COM1 (RS-485) = 1

Функциональный блок MbRtuReadBool



Блок считывает значение одного или группы дискретных регистров (тип BOOL) из удаленного устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
= FALSE — нет (по умолчанию)
= TRUE — есть
- Mode** — Режим работы блока [BOOL]
= FALSE — циклическое формирование запроса (по умолчанию)
= TRUE — одиночный запрос
- Com** — номер последовательного порта [BYTE]
= 1 (по умолчанию) ... 255 (в зависимости от целевого устройства)
- ID** — идентификатор удаленного устройства [BYTE]
= 0 (по умолчанию) ... 255
- Func** — код функции чтения [BYTE]
= 1 (0x01) — чтение из таблицы «Coils»,
= 2 (0x02) — чтение из таблицы «Discrete Inputs».
- Addr1** — адрес начального элемента данных [WORD]
= 0 (по умолчанию) ... 65535
- Num** — количество считываемых элементов данных [BYTE]
= 1 (по умолчанию) ... 8

Выходы:

- Ok** — код результата [BYTE] (см. Табл. 18.2/3)
- O1** — значение 1-го элемента данных [BOOL]
O2 — значение 2-го элемента данных [BOOL]
...
O8 — значение 8-го элемента данных [BOOL]

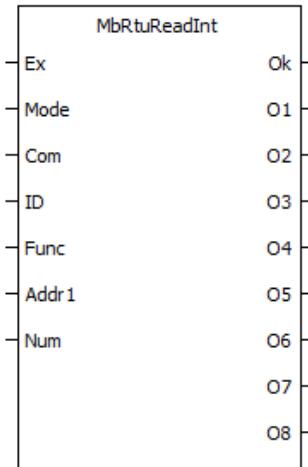
Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) чтение данных не выполняется,
- 2) возвращает код результата («Ok» = 0),
- 3) в On возвращаются последние, прочитанные через блок, значения.

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) чтение данных выполняется,
- 2) возвращает код результата («Ok» > 0),
- 3) возвращает значения в On (при ошибке не изменяет их).

Функциональный блок MbRtuReadInt



Блок считывает значение одного или группы числовых регистров (тип INT) из удаленного устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Mode — Режим работы блока [BOOL]

= FALSE — циклическое формирование запроса (по умолчанию)

= TRUE — одиночный запрос

Com — номер последовательного порта [BYTE]

= 1 (по умолчанию)...255 (в зависимости от целевого устройства)

ID — идентификатор удаленного устройства [BYTE]

Func — код функции чтения [BYTE]

= 3 (0x03) — чтение из таблицы «Holding Registers»,

= 4 (0x04) — чтение из таблицы «Input Registers».

Addr1 — адрес начального регистра группы [WORD]

= 0 ... 65535

Num — количество считываемых регистров [BYTE]

= 1 ... 8

Выходы:

Ok — код результата [BYTE] (см. Табл. 18.2/3)

O1 — значение 1-го регистра группы [INT]

O2 — значение 2-го регистра группы [INT]

...

O8 — значение 8-го регистра группы [INT]

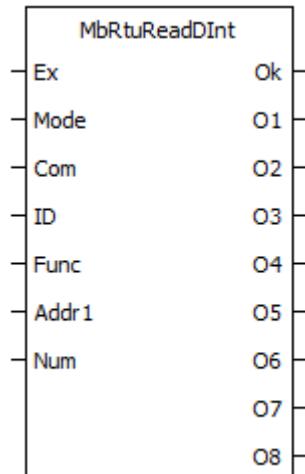
Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) чтение данных не выполняется,
- 2) возвращает код результата («Ok» = 0),
- 3) в On возвращаются последние, прочитанные через блок, значения.

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) чтение данных выполняется,
- 2) возвращает код результата («Ok» > 0),
- 3) возвращает значения в On (при ошибке не изменяет их).

Функциональный блок MbRtuReadDInt



Блок считывает значение одного или группы числовых регистров (тип DINT) из удаленного устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть

Mode — Режим работы блока [BOOL]
= FALSE — циклическое формирование запроса (по умолчанию)
= TRUE — одиночный запрос

Com — номер последовательного порта [BYTE]
= 1 (по умолчанию)...255 (в зависимости от целевого устройства)

ID — идентификатор удаленного устройства [BYTE]

Func — код функции чтения [BYTE]
= 3 (0x03) — чтение из таблицы «Holding Registers»,
= 4 (0x04) — чтение из таблицы «Input Registers».

Addr1 — адрес начального регистра группы [WORD]
= 0 ... 65535

Num — количество считываемых регистров [BYTE]
= 1 ... 8

Выходы:

Ok — код результата [BYTE] (см. Табл. 18.2/3)

O1 — значение 1-го регистра группы [DINT]

O2 — значение 2-го регистра группы [DINT]

...

O8 — значение 8-го регистра группы [DINT]

Если нет разрешения выполнения (`«Ex» = FALSE`), то:

- 1) чтение данных не выполняется,
- 2) возвращает код результата (`«Ok» = 0`),
- 3) в On возвращаются последние, прочитанные через блок, значения.

Если есть разрешение на выполнение (`«Ex» = TRUE`), то:

- 1) чтение данных выполняется,
- 2) возвращает код результата (`«Ok» > 0`),
- 3) возвращает значения в On (при ошибке не изменяет их).

Функциональный блок MbRtuReadWord

| MbRtuReadWord | |
|---------------|----|
| Ex | Ok |
| Mode | 01 |
| Com | 02 |
| ID | 03 |
| Func | 04 |
| Addr1 | 05 |
| Num | 06 |
| | 07 |
| | 08 |

Блок считывает значение одного или группы числовых регистров (тип WORD) из удаленного устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 = FALSE — нет
 = TRUE — есть
- Mode** — Режим работы блока [BOOL]
 = FALSE — циклическое формирование запроса (по умолчанию)
 = TRUE — одиночный запрос
- Com** — номер последовательного порта [BYTE]
 = 1 (по умолчанию)...255 (в зависимости от целевого устройства)
- ID** — идентификатор удаленного устройства [BYTE]
- Func** — код функции чтения [BYTE]
 = 3 (0x03) — чтение из таблицы «Holding Registers»,
 = 4 (0x04) — чтение из таблицы «Input Registers».
- Addr1** — адрес начального регистра группы [WORD]
 = 0 ... 65535
- Num** — количество считываемых регистров [BYTE]
 = 1 ... 8

Выходы:

Ok – код результата [BYTE] (см. Табл. 18.2/3)

O1 – значение 1-го регистра группы [WORD]

O2 – значение 2-го регистра группы [WORD]

...

O8 – значение 8-го регистра группы [WORD]

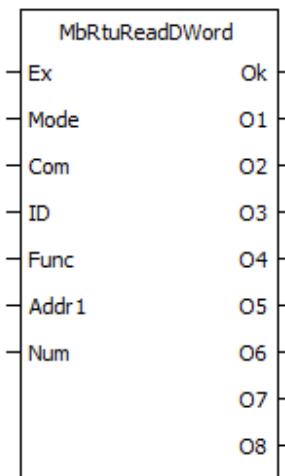
Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) чтение данных не выполняется,
- 2) возвращает код результата («Ok» = 0),
- 3) в On возвращаются последние, прочитанные через блок, значения.

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) чтение данных выполняется,
- 2) возвращает код результата («Ok» > 0),
- 3) возвращает значения в On (при ошибке не изменяет их).

Функциональный блок MbRtuReadDWord



Блок считывает значение одного или группы числовых регистров (тип DWORD) из удаленного устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Mode — Режим работы блока [BOOL]

= FALSE — циклическое формирование запроса (по умолчанию)

= TRUE — одиночный запрос

Com — номер последовательного порта [BYTE]

= 1 (по умолчанию)...255 (в зависимости от целевого устройства)

ID — идентификатор удаленного устройства [BYTE]

Func — код функции чтения [BYTE]

- = 3 (0x03) – чтение из таблицы «Holding Registers»,
- = 4 (0x04) – чтение из таблицы «Input Registers».

Addr1 – адрес начального регистра группы [WORD]
 = 0 ... 65535

Num – количество считываемых регистров [BYTE]
 = 1 ... 8

Выходы:

Ok – код результата [BYTE] (см. Табл. 18.2/3)

O1 – значение 1-го регистра группы [DWORD]

O2 – значение 2-го регистра группы [DWORD]

...

O8 – значение 8-го регистра группы [DWORD]

Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) чтение данных не выполняется,
- 2) возвращает код результата («Ok» = 0),
- 3) в On возвращаются последние, прочитанные через блок, значения.

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) чтение данных выполняется,
- 2) возвращает код результата («Ok» > 0),
- 3) возвращает значения в On (при ошибке не изменяет их).

Функциональный блок MbRtuReadReal

| MbRtuReadReal | |
|---------------|----|
| Ex | Ok |
| Mode | O1 |
| Com | O2 |
| ID | O3 |
| Func | O4 |
| Addr1 | O5 |
| Num | O6 |
| | O7 |
| | O8 |

Блок считывает значение одного или группы числовых регистров (тип REAL) из удаленного устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
 = FALSE — нет
 = TRUE — есть

Mode – Режим работы блока [BOOL]

= FALSE — циклическое формирование запроса (по умолчанию)
= TRUE — одиночный запрос

Com — номер последовательного порта [BYTE]

= 1 (по умолчанию)...255 (в зависимости от целевого устройства)

ID — идентификатор удаленного устройства [BYTE]

Func — код функции чтения [BYTE]

= 3 (0x03) — чтение из таблицы «Holding Registers»,

= 4 (0x04) — чтение из таблицы «Input Registers».

Addr1 — адрес начального регистра группы [WORD]

= 0 ... 65535

Num — количество считываемых регистров [BYTE]

= 1 ... 8

Выходы:

Ok — код результата [BYTE] (см. Табл. 18.2/3)

O1 — значение 1-го регистра группы [REAL]

O2 — значение 2-го регистра группы [REAL]

...

O8 — значение 8-го регистра группы [REAL]

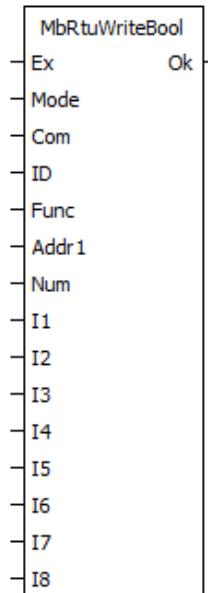
Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) чтение данных не выполняется,
- 2) возвращает код результата («Ok» = 0),
- 3) в On возвращаются последние, прочитанные через блок, значения.

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) чтение данных выполняется,
- 2) возвращает код результата («Ok» > 0),
- 3) возвращает значения в On (при ошибке не изменяет их).

Функциональный блок MbRtuWriteBool



Блок передает (записывает) значение одного или группы дискретных регистров (тип BOOL) на удаленное устройство.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Mode — Режим работы блока [BOOL]

= FALSE — циклическое формирование запроса (по умолчанию)

= TRUE — одиночный запрос

Com — номер последовательного порта [BYTE]

= 1 (по умолчанию)...255 (в зависимости от целевого устройства)

ID — идентификатор удаленного устройства [BYTE]

Func — код функции чтения [BYTE]

= 5 (0x05) — запись в таблицу «Coils» в режиме «single»,

= 15 (0x0F) — запись в таблицу «Coils» в режиме «multiple».

Addr1 — адрес начального элемента данных [WORD]

= 0 ... 65535

Num — Func 0F: количество передаваемых элементов данных [BYTE]

Func 05: определяет какое значение Ix будет передано в элемент Addr1

= 1 ... 8

I1 — значение 1-го элемента данных [BOOL]

I2 — значение 2-го элемента данных [BOOL]

...

I8 — значение 8-го элемента данных [BOOL]

Выходы:

Ok — код результата [BYTE] (см. Табл. 18.2/3)

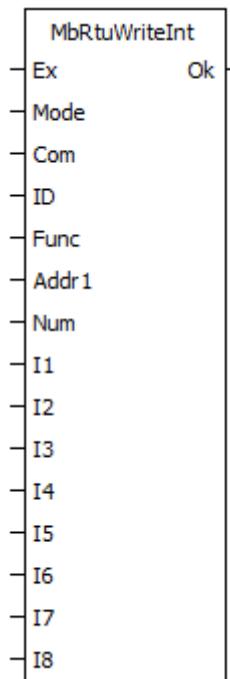
Если нет разрешения выполнения ($\text{«Ex»} = \text{FALSE}$), то:

- 1) передача данных не выполняется,
- 2) возвращает код результата ($\text{«Ok»} = 0$).

Если есть разрешение на выполнение ($\text{«Ex»} = \text{TRUE}$), то:

- 1) передача данных выполняется,
- 2) возвращает код результата ($\text{«Ok»} > 0$).

Функциональный блок MbRtuWriteInt



Блок передает (записывает) значение одного или группы числовых регистров (тип INT) на удаленное устройство.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- Mode** — Режим работы блока [BOOL]
 - = FALSE — циклическое формирование запроса (по умолчанию)
 - = TRUE — одиночный запрос
- Com** — номер последовательного порта [BYTE]
 - = 1 (по умолчанию)...255 (в зависимости от целевого устройства)
- ID** — идентификатор удаленного устройства [BYTE]
- Func** — код функции чтения [BYTE]
 - = 6 (0x06) — запись в таблицу «Holding Registers» в режиме «single»,
 - = 16(0x10) — запись в таблицу «Holding Registers» в режиме «multiple».
- Addr1** — адрес начального регистра группы [WORD]
 - = 0 ... 65535

Num – Func 16(0x10): количество передаваемых регистров[BYTE]

Func 6 (0x06): определяет какое значение Ix будет передано в регистр

Addr1

= 1 ... 8

I1 – значение 1-го регистра группы [INT]

I2 – значение 2-го регистра группы [INT]

...

I8 – значение 8-го регистра группы [INT]

Выходы:

Ok – код результата [BYTE] (см. Табл. 18.2/3)

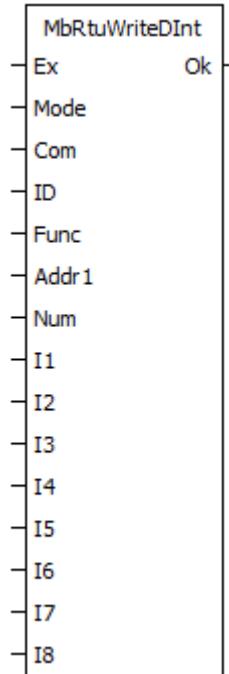
Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) передача данных не выполняется,
- 2) возвращает код результата («Ok» = 0).

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) передача данных выполняется,
- 2) возвращает код результата («Ok» > 0).

Функциональный блок MbRtuWriteDInt



Блок передает (записывает) значение одного или группы числовых регистров (тип DINT) на удаленное устройство.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- Mode** — Режим работы блока [BOOL]
 - = FALSE — циклическое формирование запроса (по умолчанию)
 - = TRUE — одиночный запрос
- Com** — номер последовательного порта [BYTE]
 - = 1 (по умолчанию)...255 (в зависимости от целевого устройства)
- ID** — идентификатор удаленного устройства [BYTE]
- Func** — код функции чтения [BYTE]
 - = 16 (0x10)—запись в таблицу «Holding Registers» в режиме «multiple».
- Addr1** — адрес начального регистра группы [WORD]
 - = 0 ... 65535
- Num** — Func 16 (0x10): количество передаваемых регистров[BYTE]

- I1** — значение 1-го регистра группы [DINT]
- I2** — значение 2-го регистра группы [DINT]
- ...
- I8** — значение 8-го регистра группы [DINT]

Выходы:

- Ok** — код результата [BYTE] (см. Табл. 18.2/3)

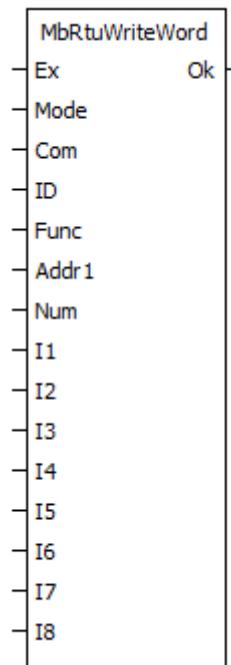
Если нет разрешения выполнения ($\text{«Ex»} = \text{FALSE}$), то:

- 1) передача данных не выполняется,
- 2) возвращает код результата ($\text{«Ok»} = 0$).

Если есть разрешение на выполнение ($\text{«Ex»} = \text{TRUE}$), то:

- 1) передача данных выполняется,
- 2) возвращает код результата ($\text{«Ok»} > 0$).

Функциональный блок MbRtuWriteWord



Блок передает (записывает) значение одного или группы числовых регистров (тип WORD) на удаленное устройство.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- Mode** — Режим работы блока [BOOL]
 - = FALSE — циклическое формирование запроса (по умолчанию)
 - = TRUE — одиночный запрос
- Com** — номер последовательного порта [BYTE]
 - = 1 (по умолчанию)...255 (в зависимости от целевого устройства)
- ID** — идентификатор удаленного устройства [BYTE]
- Func** — код функции чтения [BYTE]
 - = 6 (0x06) — запись в таблицу «Holding Registers» в режиме «single»,
 - = 16 (0x10) — запись в таблицу «Holding Registers» в режиме «multiple».
- Addr1** — адрес начального регистра группы [WORD]
 - = 0 ... 65535

Num – Func 16 (0x10): количество передаваемых регистров[BYTE]

Func 6 (0x06): определяет какое значение Ix будет передано в регистр

Addr1

= 1 ... 8

I1 – значение 1-го регистра группы [WORD]

I2 – значение 2-го регистра группы [WORD]

...

I8 – значение 8-го регистра группы [WORD]

Выходы:

Ok – код результата [BYTE] (см. Табл. 18.2/3)

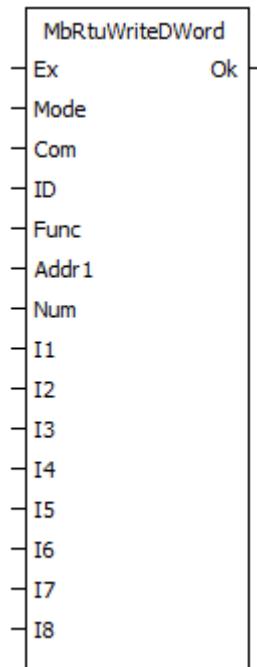
Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) передача данных не выполняется,
- 2) возвращает код результата («Ok» = 0).

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) передача данных выполняется,
- 2) возвращает код результата («Ok» > 0).

Функциональный блок MbRtuWriteDWord



Блок передает (записывает) значение одного или группы числовых регистров (тип DWORD) на удаленное устройство.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть
- Mode** — Режим работы блока [BOOL]
= FALSE — циклическое формирование запроса (по умолчанию)
= TRUE — одиночный запрос
- Com** — номер последовательного порта [BYTE]
= 1 (по умолчанию)...255 (в зависимости от целевого устройства)
- ID** — идентификатор удаленного устройства [BYTE]
- Func** — код функции чтения [BYTE]
= 16 (0x10)—запись в таблицу «Holding Registers» в режиме «multiple».
- Addr1** — адрес начального регистра группы [WORD]
= 0 ... 65535
- Num** — Func 16 (0x10): количество передаваемых регистров[BYTE]
- I1** — значение 1-го регистра группы [DWORD]
- I2** — значение 2-го регистра группы [DWORD]
- ...
- I8** — значение 8-го регистра группы [DWORD]

Выходы:

- Ok** — код результата [BYTE] (см. Табл. 18.2/3)

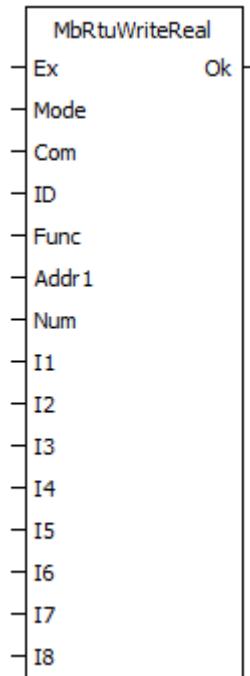
Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) передача данных не выполняется,
- 2) возвращает код результата («Ok» = 0).

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) передача данных выполняется,
- 2) возвращает код результата («Ok» > 0).

Функциональный блок MbRtuWriteReal



Блок передает (записывает) значение одного или группы числовых регистров (тип REAL) на удаленное устройство.

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть
- Mode** — Режим работы блока [BOOL]
= FALSE — циклическое формирование запроса (по умолчанию)
= TRUE — одиночный запрос
- Com** — номер последовательного порта [BYTE]
= 1 (по умолчанию)...255 (в зависимости от целевого устройства)
- ID** — идентификатор удаленного устройства [BYTE]
- Func** — код функции чтения [BYTE]
= 16 (0x10)—запись в таблицу «Holding Registers» в режиме «multiple».
- Addr1** — адрес начального регистра группы [WORD]
= 0 ... 65535
- Num** — Func 16 (0x10): количество передаваемых регистров[BYTE]
- I1** — значение 1-го регистра группы [REAL]
- I2** — значение 2-го регистра группы [REAL]
- ...
- I8** — значение 8-го регистра группы [REAL]

Выходы:

- Ok** — код результата [BYTE] (см. Табл. 18.2/3)

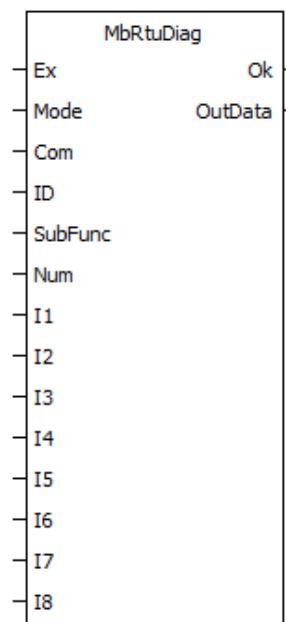
Если нет разрешения выполнения (**«Ex» = FALSE**), то:

- 1) передача данных не выполняется,
- 2) возвращает код результата (**«Ok» = 0**).

Если есть разрешение на выполнение (**«Ex» = TRUE**), то:

- 1) передача данных выполняется,
- 2) возвращает код результата (**«Ok» > 0**).

Функциональный блок MbRtuDiag



Блок передает функцию «0x08» и код диагностики на удаленное устройство, получает и выводит результат.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Mode — Режим работы блока [BOOL]

= FALSE — циклическое формирование запроса (по умолчанию)

= TRUE — одиночный запрос

Com — номер последовательного порта [BYTE]

= 1 (по умолчанию)...255 (в зависимости от целевого устройства)

ID — идентификатор удаленного устройства [BYTE]

SubFunc — код функции диагностики (см. описание ModBus RTU) [BYTE]

= 0 — «Echo»,

= ...

Num — определяет какое значение Ix будет использовано как значащее

= 1 ... 8

I1 — значение 1-го регистра группы [WORD]

I2 — значение 2-го регистра группы [WORD]

...

I8 – значение 8-го регистра группы [WORD]

Выходы:

Ok – код результата [BYTE] (см. Табл. 18.2/3)

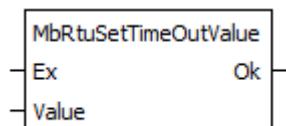
OutData – значение принятное обратно [WORD]

Если нет разрешения выполнения (**«Ex» = FALSE**), то:

- 1) передача данных не выполняется,
- 2) возвращает код результата (**«Ok» = 0**).

Если есть разрешение на выполнение (**«Ex» = TRUE**), то:

- 1) передача данных выполняется,
- 2) возвращает код результата (**«Ok» > 0**).

Функциональный блок MbRtuSetTimeOutValue

Блок устанавливает значение таймаута ожидания ответа от ведомого устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Value — новое значение таймаута (мс) [WORD]

Выходы:

Ok – код результата [BYTE]

= 0 – ожидание инициации команды (**Ex = FALSE**)

= 2 – операция чтения или записи завершена успешно

= 3 – неверный код операции

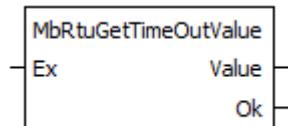
Если нет разрешения выполнения (**«Ex» = FALSE**), то:

- 2) ожидание инициации команды,
- 3) возвращает код результата (**«Ok» = 0**).

Если есть разрешение на выполнение (**«Ex» = TRUE**), то:

- 1) инициируется выполнение команды,
- 2) возвращает код результата (**«Ok» > 0**).

Функциональный блок MbRtuGetTimeOutValue



Блок возвращает текущее значение таймаута ожидания ответа от ведомого устройства.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
 = FALSE — нет
 = TRUE — есть

Выходы:

Value — текущее значение таймаута (мс) [WORD]

Ok — код результата [BYTE]
 = 0 — ожидание инициации команды (Ex = FALSE)
 = 2 — операция чтения или записи завершена успешно
 = 3 — неверный код операции

Если нет разрешения выполнения («Ex» = FALSE), то:

- 1) ожидание инициации команды,
- 2) возвращает код результата («Ok» = 0).

Если есть разрешение на выполнение («Ex» = TRUE), то:

- 1) инициируется выполнение команды,
- 2) возвращает код результата («Ok» > 0).

ПРИЛОЖЕНИЕ 19

БИБЛИОТЕКА EEPROM

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_EEPROM» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с EEPROM. Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Нужно отметить, что блоки, входящие в данную библиотеку отрабатываются в несколько проходов цикла выполнения программы пользователя. Т.е. они могут быть использованы только в циклических задачах.

Скрытое автоматическое назначение адресов

Реализован следующий алгоритм автоматического назначения адресов ячеек EEPROM для записи/чтения данных:

- номер адреса начинается с 0 (реальный адрес EEPROM = смещение + выделенный адрес);
- при назначении адреса учитывается тип данных (сколько байт занимают данные);
- номер адреса должен лежать в границах области памяти, выделенной для пользовательских данных;
- если нет свободной памяти, то адрес не назначается (возвращается код ошибки -1);
- разработчик пользовательского приложения в ИСР Beremiz имеет возможность только просматривать автоматически назначенные адреса ячеек, редактирование запрещено;
- назначенные адреса ячеек хранятся в виде скрытого списка в файле проекта ИСР Beremiz (считываются при открытии, сохраняются при закрытии, компиляции);
- адрес назначается при добавлении определенной переменной (регистр) в проект без перераспределения ранее назначенных адресов – выполняется перебор списка адресов, где ищется свободная область для необходимого количества байт и занимается;
- адрес освобождается при удалении определенной переменной (регистр) из проекта без перераспределения ранее назначенных адресов – область памяти освобождается и становится доступной для назначения.

Структура типов регистров EEPROM

Структура регистра EeRegByte

Структура «EeRegByte» описывает один регистр данных типа BYTE для работы с EEPROM. Данная структура входит в состав подключаемой аппаратно-зависимой библиотеки и доступна в списке «LibIT_EEPROM типы данных» ИСР Beremiz. Список полей структуры приведен в таблице:

| N п.п. | Имя | Тип | Значение по умолчанию | Описание |
|--------|------|------|-----------------------|-----------------|
| 1 | Addr | DINT | -1 | адрес EEPROM |
| 2 | Ту | BYTE | 1 | код типа данных |

Поле «Addr» содержит автоматически назначенный адрес данных в EEPROM:

- 1 – нет адреса,
- ≥ 0 – есть адрес.

Поле «Ту» содержит код (1) для типа BYTE (1 байт).

Структура регистра EeRegWord

Структура «EeRegWord» описывает один регистр данных типа WORD для работы с EEPROM. Данная структура входит в состав подключаемой аппаратно-зависимой библиотеки и доступна в списке «LibIT_EEPROM типы данных» ИСР Beremiz. Список полей структуры приведен в таблице:

| N п.п. | Имя | Тип | Значение по умолчанию | Описание |
|--------|------|------|-----------------------|----------------|
| 1 | Addr | DINT | -1 | адрес EEPROM |
| 2 | Ту | BYTE | 2 | код тип данных |

Поле «Addr» содержит автоматически назначенный адрес данных в EEPROM:

- 1 – нет адреса,
- ≥ 0 – есть адрес.

Поле «Ту» содержит код (2) для типа WORD (2 байта).

Структура регистра EeRegDWord

Структура «EeRegDWord» описывает один регистр данных типа DWORD для работы с EEPROM. Данная структура входит в состав подключаемой аппаратно-зависимой библиотеки и доступна в списке «LibIT_EEPROM типы данных» ИСР Beremiz. Список полей структуры приведен в таблице:

| N п.п. | Имя | Тип | Значение по умолчанию | Описание |
|--------|------|------|-----------------------|----------------|
| 1 | Addr | DINT | -1 | адрес EEPROM |
| 2 | Ту | BYTE | 3 | код тип данных |

Поле «Addr» содержит автоматически назначенный адрес данных в EEPROM:

- 1 – нет адреса,
- ≥ 0 – есть адрес.

Поле «Ту» содержит код (3) для типа DWORD (4 байта).

Структура регистра EeRegLWord

Структура «EeRegLWord» описывает один регистр данных типа LWORD для работы с EEPROM. Данная структура входит в состав подключаемой аппаратно-зависимой библиотеки и доступна в списке «LibIT_EEPROM типы данных» ИСР Beremiz. Список полей структуры приведен в таблице:

| N п.п. | Имя | Тип | Значение по умолчанию | Описание |
|--------|------|------|-----------------------|----------------|
| 1 | Addr | DINT | -1 | адрес EEPROM |
| 2 | Ту | BYTE | 4 | код тип данных |

Поле «Addr» содержит автоматически назначенный адрес данных в EEPROM:

- 1 – нет адреса,
- ≥ 0 – есть адрес.

Поле «Ту» содержит код (4) для типа LWORD (8 байт).

Структура регистра EeRegReal

Структура «EeRegReal» описывает один регистр данных типа REAL для работы с EEPROM. Данная структура входит в состав подключаемой аппаратно-зависимой библиотеки и доступна в списке «LibIT_EEPROM типы данных» ИСР Beremiz. Список полей структуры приведен в таблице:

| N п.п. | Имя | Тип | Значение по умолчанию | Описание |
|--------|------|------|-----------------------|----------------|
| 1 | Addr | DINT | -1 | адрес EEPROM |
| 2 | Ту | BYTE | 5 | код тип данных |

Поле «Addr» содержит автоматически назначенный адрес данных в EEPROM:

- 1 – нет адреса,
- ≥ 0 – есть адрес.

Поле «Ту» содержит код (5) для типа REAL (4 байта).

Структура регистра EeRegLReal

Структура «EeRegLReal» описывает один регистр данных типа LREAL для работы с EEPROM. Данная структура входит в состав подключаемой аппаратно-зависимой библиотеки и доступна в списке «LibIT_EEPROM типы данных» ИСР Beremiz. Список полей структуры приведен в таблице:

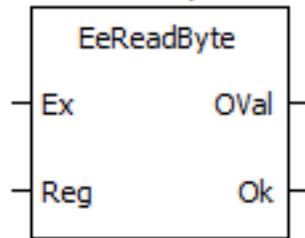
| N п.п. | Имя | Тип | Значение по умолчанию | Описание |
|--------|------|------|-----------------------|----------------|
| 1 | Addr | DINT | -1 | адрес EEPROM |
| 2 | Ту | BYTE | 6 | код тип данных |

Поле «Addr» содержит автоматически назначенный адрес данных в EEPROM:
 -1 – нет адреса,
 >= 0 – есть адрес.

Поле «Ту» содержит код (6) для типа LREAL (8 байт).

Функции и функциональные блоки

Функциональный блок EeReadByte



Функциональный блок «EeReadByte» считывает значение типа BYTE (1 байт) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
 = FALSE — нет
 = TRUE — есть

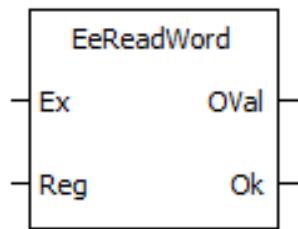
Reg — регистр [EeRegByte]

Выходы:

OVal — значение [BYTE]

Ok — код результата выполнения [BYTE]
 = 0 – выполнение блока запрещено (Ex = FALSE)
 = 1 – выполняется операция чтения или записи
 = 2 – операция чтения или записи завершена успешно
 = 3 – неверный код операции
 = 4 – неверный адрес памяти данных
 = 5 – неверный код типа данных
 = 6 – ошибка I2C
 = 7 – системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
 = 8 – системная ошибка: переполнение очереди-результата

Функциональный блок EeReadWord



Функциональный блок «EeReadWord» считывает значение типа WORD (2 байта) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Reg — регистр [EeRegWord]

Выходы:

OVal — значение [WORD]

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE)

= 1 — выполняется операция чтения или записи

= 2 — операция чтения или записи завершена успешно

= 3 — неверный код операции

= 4 — неверный адрес памяти данных

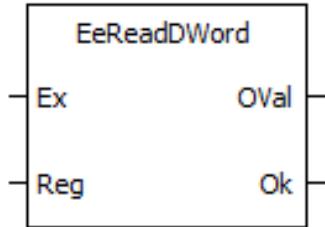
= 5 — неверный код типа данных

= 6 — ошибка I2C

= 7 — системная ошибка: нулевой указатель на интерфейс I2C или буфер данных

= 8 — системная ошибка: переполнение очереди-результата

Функциональный блок EeReadDWord



Функциональный блок «EeReadDWord» считывает значение типа DWORD (4 байта) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

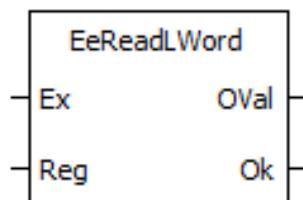
Входы:

Ex — разрешение выполнения [BOOL]
 = FALSE — нет
 = TRUE — есть
Reg — регистр [EeRegDWord]

Выходы:

OVal — значение [DWORD]
Ok — код результата выполнения [BYTE]
 = 0 — выполнение блока запрещено (Ex = FALSE)
 = 1 — выполняется операция чтения или записи
 = 2 — операция чтения или записи завершена успешно
 = 3 — неверный код операции
 = 4 — неверный адрес памяти данных
 = 5 - неверный код типа данных
 = 6 — ошибка I2C
 = 7 — системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
 = 8 — системная ошибка: переполнение очереди-результата

Функциональный блок EeReadLWord



Функциональный блок «EeReadLWord» считывает значение типа LWORD (8 байт) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]
 = FALSE — нет
 = TRUE — есть
Reg — регистр [EeRegLWord]

Выходы:

OVal — значение [LWORD]
Ok — код результата выполнения [BYTE]
 = 0 — выполнение блока запрещено (Ex = FALSE)
 = 1 — выполняется операция чтения или записи
 = 2 — операция чтения или записи завершена успешно
 = 3 — неверный код операции
 = 4 — неверный адрес памяти данных
 = 5 - неверный код типа данных
 = 6 — ошибка I2C

- = 7 – системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
- = 8 – системная ошибка: переполнение очереди-результата

Функциональный блок EeReadReal



Функциональный блок «EeReadReal» считывает значение типа REAL (4 байта) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- Reg** — регистр [EeRegReal]

Выходы:

- OVal** — значение [REAL]
- Ok** — код результата выполнения [BYTE]
 - = 0 – выполнение блока запрещено (Ex = FALSE)
 - = 1 – выполняется операция чтения или записи
 - = 2 – операция чтения или записи завершена успешно
 - = 3 – неверный код операции
 - = 4 – неверный адрес памяти данных
 - = 5 – неверный код типа данных
 - = 6 – ошибка I2C
 - = 7 – системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
 - = 8 – системная ошибка: переполнение очереди-результата

Функциональный блок EeReadLReal



Функциональный блок «EeReadLReal» считывает значение типа LREAL (8 байт) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Reg — регистр [EeRegLReal]

Выходы:

OVal — значение [LREAL]

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE)

= 1 — выполняется операция чтения или записи

= 2 — операция чтения или записи завершена успешно

= 3 — неверный код операции

= 4 — неверный адрес памяти данных

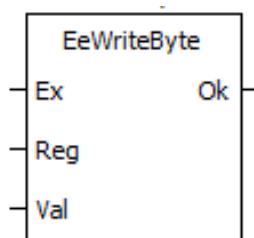
= 5 — неверный код типа данных

= 6 — ошибка I2C

= 7 — системная ошибка: нулевой указатель на интерфейс I2C или буфер данных

= 8 — системная ошибка: переполнение очереди-результата

Функциональный блок EeWriteByte



Функциональный блок «EeWriteByte» записывает значение типа BYTE (1 байт) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Reg — регистр [EeRegByte]

Val — значение [BYTE]

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE)

= 1 — выполняется операция чтения или записи

= 2 — операция чтения или записи завершена успешно

= 3 — неверный код операции

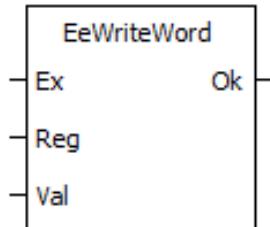
= 4 — неверный адрес памяти данных

= 5 — неверный код типа данных

= 6 — ошибка I2C

- = 7 – системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
- = 8 – системная ошибка: переполнение очереди-результата

Функциональный блок EeWriteWord



Функциональный блок «EeWriteWord» записывает значение типа WORD (2 байта) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

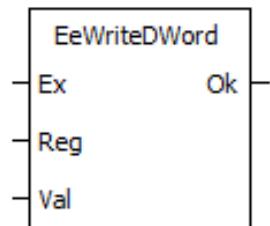
Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- Reg** — регистр [EeRegWord]
- Val** — значение [WORD]

Выходы:

- Ok** — код результата выполнения [BYTE]
 - = 0 – выполнение блока запрещено (Ex = FALSE)
 - = 1 – выполняется операция чтения или записи
 - = 2 – операция чтения или записи завершена успешно
 - = 3 – неверный код операции
 - = 4 – неверный адрес памяти данных
 - = 5 – неверный код типа данных
 - = 6 – ошибка I2C
 - = 7 – системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
 - = 8 – системная ошибка: переполнение очереди-результата

Функциональный блок EeWriteDWord



Функциональный блок «EeWriteDWord» записывает значение типа DWORD (4 байта) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Reg — регистр [EeRegDWord]

Val — значение [DWORD]

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE)

= 1 — выполняется операция чтения или записи

= 2 — операция чтения или записи завершена успешно

= 3 — неверный код операции

= 4 — неверный адрес памяти данных

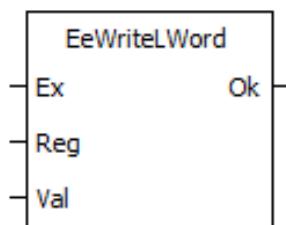
= 5 — неверный код типа данных

= 6 — ошибка I2C

= 7 — системная ошибка: нулевой указатель на интерфейс I2C или буфер данных

= 8 — системная ошибка: переполнение очереди-результата

Функциональный блок EeWriteLWord



Функциональный блок «EeWriteLWord» записывает значение типа LWORD (8 байт) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Reg — регистр [EeRegLWord]

Val — значение [LWORD]

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE)

= 1 — выполняется операция чтения или записи

= 2 — операция чтения или записи завершена успешно

= 3 — неверный код операции

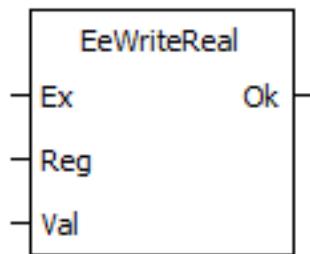
= 4 — неверный адрес памяти данных

= 5 — неверный код типа данных

= 6 — ошибка I2C

- = 7 – системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
- = 8 – системная ошибка: переполнение очереди-результата

Функциональный блок EeWriteReal



Функциональный блок «EeWriteReal» записывает значение типа REAL (4 байта) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

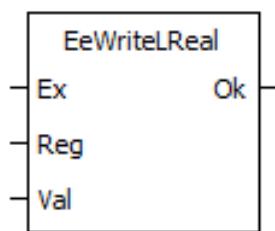
Входы:

- Ex** — разрешение выполнения [BOOL]
 - = FALSE — нет
 - = TRUE — есть
- Reg** — регистр [EeRegReal]
- Val** — значение [REAL]

Выходы:

- Ok** — код результата выполнения [BYTE]
 - = 0 – выполнение блока запрещено (Ex = FALSE)
 - = 1 – выполняется операция чтения или записи
 - = 2 – операция чтения или записи завершена успешно
 - = 3 – неверный код операции
 - = 4 – неверный адрес памяти данных
 - = 5 – неверный код типа данных
 - = 6 – ошибка I2C
 - = 7 – системная ошибка: нулевой указатель на интерфейс I2C или буфер данных
 - = 8 – системная ошибка: переполнение очереди-результата

Функциональный блок EeWriteLReal



Функциональный блок «EeWriteLReal» записывает значение типа LREAL (8 байт) из EEPROM по адресу, заданному в регистре «Reg».

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex — разрешение выполнения [BOOL]

= FALSE — нет

= TRUE — есть

Reg — регистр [EeRegLReal]

Val — значение [LREAL]

Выходы:

Ok — код результата выполнения [BYTE]

= 0 — выполнение блока запрещено (Ex = FALSE)

= 1 — выполняется операция чтения или записи

= 2 — операция чтения или записи завершена успешно

= 3 — неверный код операции

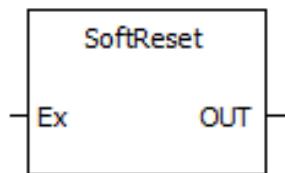
= 4 — неверный адрес памяти данных

= 5 — неверный код типа данных

= 6 — ошибка I2C

= 7 — системная ошибка: нулевой указатель на интерфейс I2C или буфер данных

= 8 — системная ошибка: переполнение очереди-результата

ПРИЛОЖЕНИЕ 20**БИБЛИОТЕКА СИСТЕМНЫХ ФУНКЦИЙ****Функция SoftReset**

Функция «SoftReset» выполняет перезагрузку целевого устройства.

Входы:

Ex — разрешение выполнения [BOOL]
= FALSE — нет
= TRUE — есть

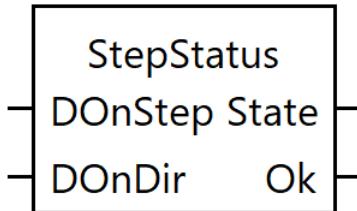
Выходы:

Ok – результат [BOOL]
= FALSE – не выполнено
= TRUE – выполнено

ПРИЛОЖЕНИЕ 21**БИБЛИОТЕКА УПРАВЛЕНИЯ ШАГОВЫМ ДВИГАТЕЛЕМ**

Данная библиотека представляет собой внешнюю подключаемую библиотеку «LibIT_Stepper» для ИСР Beremiz и предоставляет набор функций и функциональных блоков, реализующих алгоритмы для работы с шаговыми двигателями. Алгоритмы аппаратно-зависимые: поддерживающие устройства и особенности указаны в описании.

Библиотека не поддерживается эмулятором.

Функциональный блок StepStatus

Функциональный блок «StepStatus» возвращает текущий режим работы шагового двигателя.

Поддерживаемые устройства: ПЛМ2004.

Входы:

DOnStep – номер канала дискретного вывода подключенного ко входу Step драйвера двигателя [BYTE]
= 0 ... N

DOnDir – номер канала дискретного вывода подключенного ко входу Dir драйвера двигателя [BYTE]
= 0 ... N

Выходы:

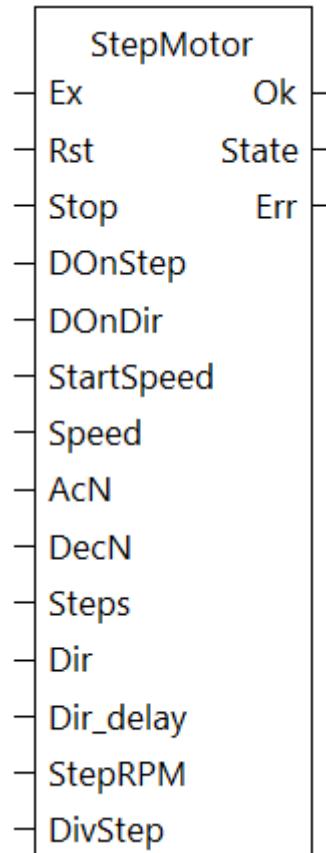
State – режим работы шагового двигателя [BYTE]
= 0 – двигатель остановлен,
= 1 – двигатель в режиме «Ускорение»,
= 2 – двигатель в режиме «Остановка»,
= 3 – двигатель в режиме «Работа»,

Ok – код результата выполнения [BYTE]
= 1 – нет ошибок,
= 3 – задан неверный номер канала.

Для ПЛМ2004:

58. DOn: 0 .. 7

Функциональный блок StepMotor



Функциональный блок «StepMove» запускает шаговый двигатель. Данный блок управляет двигателем при помощи сигналов STEP/DIR. За один импульс ротор двигателя поворачивается на один микрошаг. Чем выше частота импульсов, тем выше скорость вращения ротора. Выход DIR предназначен для выбора направления вращения двигателя.

Для правильной работы блока каналы дискретного вывода необходимо настроить на режим шаг ШД (step) и направление ШД (dir) см. функциональный блок DOMode Приложение 16.

Поддерживаемые устройства: ПЛМ2004.

Входы:

Ex – разрешение выполнения [BOOL],
 = FALSE — нет,
 = TRUE — есть.

Rst – сброс значения счетчика шагов двигателя [BOOL],
 = FALSE – нет,
 = TRUE – есть.

Stop – остановка двигателя [BYTE],
 = 0 – двигатель не останавливается,
 = 1 – плавная остановка,
 = 2 – аварийная остановка.

DOnStep – номер канала дискретного вывода подключенного ко входу Step драйвера двигателя [BYTE],
 = 0 ... N.

DOnDir – номер канала дискретного вывода подключенного ко входу Dir драйвера двигателя [BYTE],
 = 0 ... N.

StartSpeed – начальная скорость двигателя [WORD],
 = 0 ... 1000 об/мин.

Speed – скорость двигателя [WORD],
 = 0 ... 3000 об/мин.

AcN – количество шагов для ускорения двигателя до скорости **Speed** [WORD],
 = 10 ... N.

DecN – количество шагов для остановки двигателя до скорости **StartSpeed** [WORD],
 = 10 ... N.

Steps – количество шагов двигателя [DWORD],
 = 0 ... N.

Dir – направление вращения двигателя [BOOL],
 = FALSE,
 = TRUE.

Dir_delay – задержка смены направления вращения [WORD],
 = 0 ... N, мс.

StepRPM – количество шагов двигателя на оборот [WORD],
 = 0 ... N.

DivStep – деление шага (микрошаг) [WORD]
 = 1, 2, 4, 8, 16, 32, 64, 128, 256, ... N.

Выходы:

Ok – установленный режим работы [BYTE]
 = 0 – выполнение блока запрещено (Ex = FALSE),
 = 1 – нет ошибок,
 = 2 – задан неверный номер канала,
 = 3 – задан неверный режим работы канала,

State – режим работы шагового двигателя [BYTE]
 = 0 – двигатель остановлен,
 = 1 – двигатель в режиме «Ускорение»,
 = 2 – двигатель в режиме «Остановка»,
 = 3 – двигатель в режиме «Работа»,

Err – код ошибки [BYTE]
 = 0 – нет ошибок,
 = 1 – слишком высокая скорость вращения двигателя (**Speed**),
 = 2 – частота импульсов **Step** превышает 30 kHz,
 = 3 – неверно задан делитель шага (**DivStep**)
 = 4 – сумма шагов ускорения (**AcN**) и остановки (**DecN**) превышает количество шагов (**Steps**) двигателя,
 = 5 – количество шагов ускорения (**AcN**) меньше 10,
 = 6 – количество шагов остановки (**DecN**) меньше 10.

Для ПЛМ2004:

59. Din: 0 .. 7.

При конфигурации данного блока не учитывается инерция установки, исходя из этого параметры **StartSpeed**, **Speed**, **AcN**, **DecN**, **Dir_delay** индивидуальны для каждой системы.

Параметр **Dir_delay** необходим только при смене направления.

Параметры **AcN**, **DecN**, **Speed** и **StartSpeed** задаются в тот момент, когда двигатель остановлен. При попытке установить новое значение ПЛК проигнорирует их.

Для остановки двигателя используется 2 сценария остановки:

1. (**Stop** = 1) шаговый двигатель останавливается за заданное количество шагов (**DecN**)
2. (**Stop** = 2) ШИМ сигнал с выхода **DOnStep** перестает подаваться, что при большой скорости двигателя может привести к пропуску шагов.

Движение двигателя происходит при параметре остановки (**Stop**) равном 0. При параметре **Stop** отличном от нуля запуск двигателя не происходит.

При **Rst** = True счетчик количества шагов сбрасывается.

При смене направления изменяется уровень сигнала на канале **DOnDir**.

Количество шагов на один оборот двигателя (**StepRPM**) обычно указывается в документации на двигатель (для шагового двигателя NEMA17 этот параметр равен 200). Делителя шага(**DivStep**) устанавливается драйвером (для драйвера A4988 доступны следующие значения деления шага: 1, 2, 4, 8, 16). Стоит учитывать то, что при изменении параметров описанных выше (**StepRPM**, **DivStep**) угол оборота шкива так же изменится, так как произойдет деление шагов.

Например:

При вызове блока StepMove со следующими параметрами:

- **Ex** = True;
- **Rst** = False;
- **Stop** = 0;
- **DOnStep** = 0;
- **DOnDir** = 1;
- **StartSpeed** = 100;
- **Speed** = 1000;
- **AcN** = 200;
- **DecN** = 100;
- **Steps** = 500;
- **Dir** = True;
- **Dir_delay** = 100;
- **StepRPM** = 200;
- **DivStep** = 2;

двигатель запустится с начальной скорости **StartSpeed**, через **AcN** шагов двигатель разгонится до скорости **Speed**, будет продолжать двигаться до **Steps** – **DecN** шага, удерживая скорость **Speed**, после чего остановится до **StartSpeed** за **DecN** шагов. Количество шагов, которое пройдет двигатель равно **Steps**. Зная количество пройденных шагов и размер шкива можно рассчитать расстояние

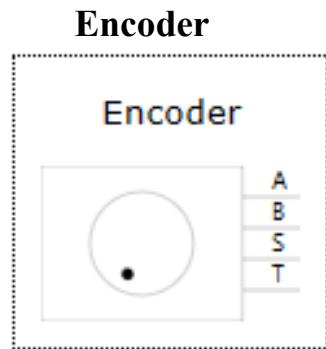
$$S = \frac{2\pi r}{StepRPM \cdot DivStep} \cdot Steps = \frac{2\pi \cdot 20,25 \text{ мм}}{200 \cdot 2} \cdot 500 \approx 159 \text{ мм}$$

Из формулы видно, что изменение параметров **StepRPM** или **DivStep** приведет к изменению расстояния **S**.

Для непрерывного вращения двигателя параметр количества шагов необходимо выставить в ноль (**Steps** = 0). Для остановки двигателя необходимо использовать параметр **Stop**.

ПРИЛОЖЕНИЕ 22**БИБЛИОТЕКА ЭЛЕМЕНТОВ ЭМУЛЯТОРА**

Данная библиотека представляет собой набор стандартных элементов для создания эмуляции устройств.



Элемент «Encoder» имеет 4 выхода:

- Канал А и В – генерирует импульсы при вращении энкодера.
- Switch – генерирует импульсы при нажатии на вал энкодера.
- Turn_switch – генерирует импульс на каждый оборот.

Для взаимодействия с элементом во время эмуляции доступны следующие события:

- Вращение колесика мыши – вращает вал энкодера.
- Клик по валу энкодера – генерирует сигнал на выходе «Switch».

При двойном клике по элементу, в режиме редактирования, открывается окно настроек (рисунок 23.1). Элемент имеет следующие настройки:

- «Step» – шаг энкодера при срабатывании события прокрутки колесика мыши. Этот параметр определяет скорость вращения энкодера.

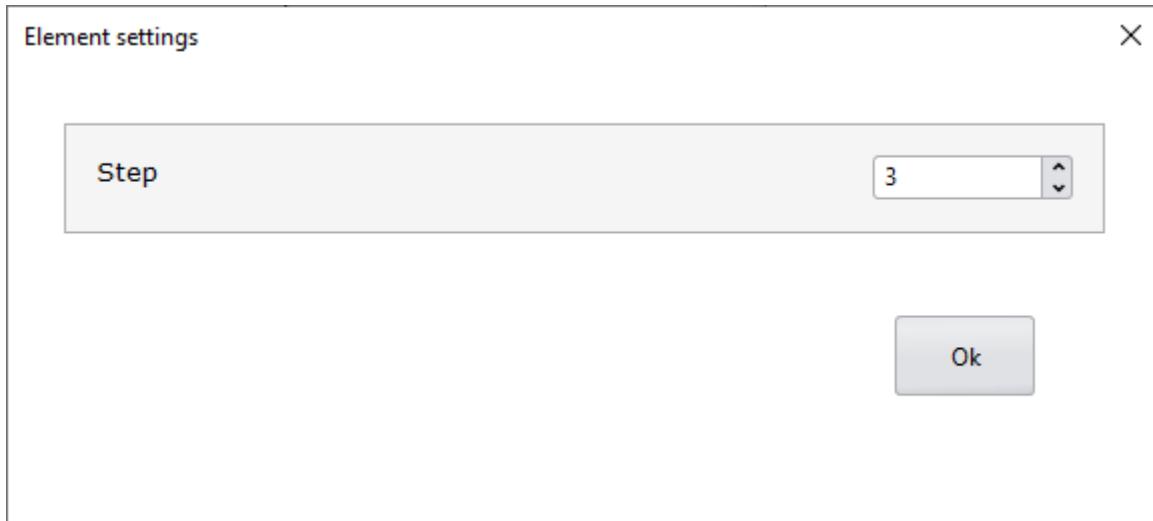
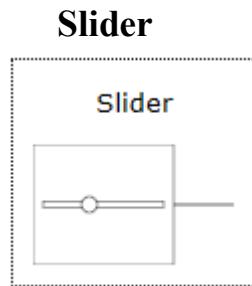


Рисунок 22.1 – Окно настроек элемента.

Данный элемент подключается на вход эмулируемого ПЛК.



Элемент «Slider» имеет 1 выход:

- Slider – генерирует аналоговый сигнал.

Для взаимодействия с элементом во время эмуляции доступны следующие события:

- Вращение колесика мыши – перемещает слайдер, изменяя напряжение на выходе.

При двойном клике по элементу, в режиме редактирования, открывается окно настроек (рисунок 23.2). Элемент имеет следующие настройки:

- «Step» – шаг изменения напряжения.
- «Minimum voltage» – минимальное напряжение. Это напряжение при установке слайдера в крайнее левое положение.
- «Maximum voltage» – максимальное напряжение. Это напряжение при установке слайдера в крайнее правое положение.

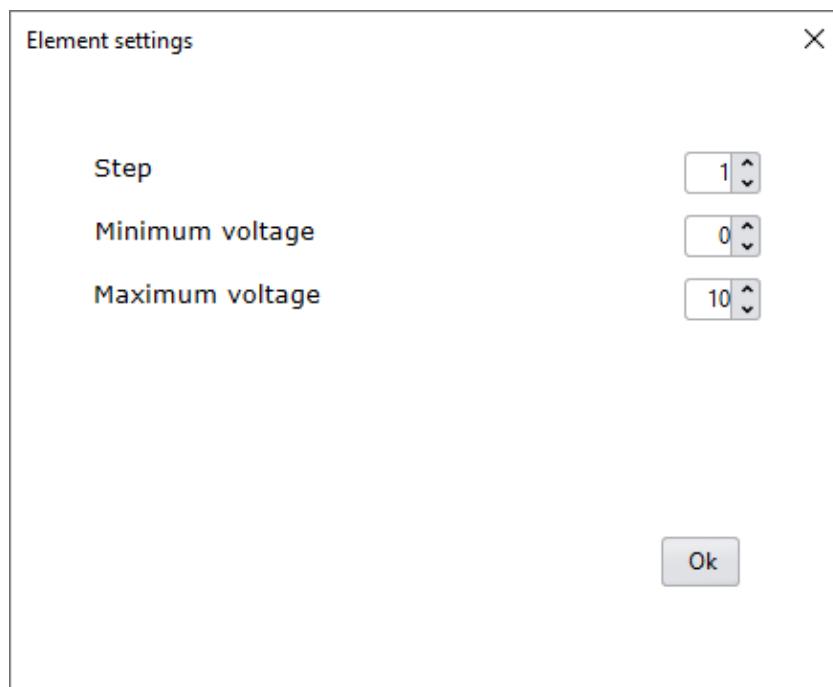
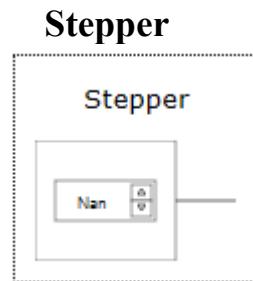


Рисунок 22.2 – Окно настроек элемента.



Элемент «Stepper» имеет 1 выход:

- Stepper – генерирует аналоговый сигнал.

Для взаимодействия с элементом во время эмуляции доступны следующие события:

- Вращение колесика мыши – изменяет напряжение на выходе.
- Клик по стрелке вверх – увеличивает напряжение на выходе.
- Клик по стрелке вниз – уменьшает напряжение на выходе.

При двойном клике по элементу, в режиме редактирования, открывается окно настроек (рисунок 23.3). Элемент имеет следующие настройки:

- «Step» – шаг изменения напряжения.
- «Minimum voltage» – минимальное напряжение.
- «Maximum voltage» – максимальное напряжение.

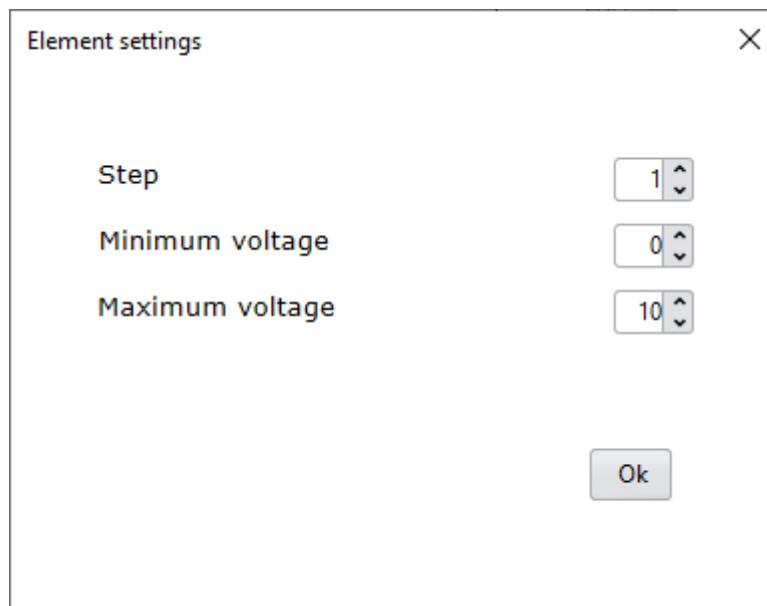
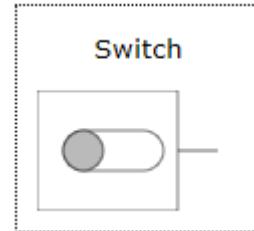


Рисунок 22.3 – Окно настроек элемента.

Энкодер



Элемент «Switch» имеет 1 выход:

- Switch – генерирует дискретный сигнал.

Для взаимодействия с элементом во время эмуляции доступны следующие события:

- Клик по переключателю – изменяет уровень сигнала на выходе элемента.

ПРИЛОЖЕНИЕ 23

ДОБАВЛЕНИЕ ПОЛЬЗОВАТЕЛЬСКОГО БЛОКА В ЭМУЛЯТОР

Все блоки написаны на языке javascript. Каталог, в котором хранятся все элементы: «beremiz\emulator\Sciter\editor\elements». Ниже представлен пример добавления элемента в библиотеку:

```
// Класс реализующий логику работы элементов в эмуляторе
import { SimElement } from "../sim_elem";

// конфигурация I/O
const pins = [{name: "A", type: "output", x: 45, y: 5, index: 0},
               {name: "B", type: "output", x: 45, y: 10, index: 1},
               {name: "Switch", type: "output", x: 45, y: 15, index: 2},
               {name: "Turn_switch", type: "output", x: 45, y: 20, index: 3}]

// Конфигурация настроек элемента
const SETTINGS = [ {type: "number", parameter_name: "Step", max_value: 100,
min_value: 1, step: 1} ]

// Значения настроек элемента по умолчанию
const DEFAULT_ATRGS = {"Step": 3}

/** Класс энкодера наследуется от SimElement */
class Encoder extends SimElement{
    // Константа для установки положения энкодера
    default_rotate = 115;
    /** Конструктор класса параметры передаются в reactor
     * см. https://github.com/c-smile/sciter-js-sdk/tree/main/docs/md/reactor
     */
    constructor(props, kids){
        // Вызов конструктора родителей
        super(props, kids);

        // Установка I/O
        this.set_connectors(pins);
        // Проверка заданных атрибутов
        if (this.attrs.length == 0){
            // Установка значений конфигурации по умолчанию
            this.attrs = DEFAULT_ATRGS;
        }

        // Установка параметров конфигурации окна настроек
        this.settings = SETTINGS;
        // Переменная определяющая положение вала энкодера
```

```
    this.rotate = this.default_rotate;
    // Счетчик оборотов вала энкодера
    this.turns = 0;
}
/** @brief Статическая функция для генерации изображения элемента.
 * Эта функция должна вызываться без конструктора класса.
 * Документация на svg формат
 * см. https://developer.mozilla.org/en-US/docs/Web/SVG
 * @param width - ширина svg рисунка.
 * @param height - высота svg рисунка.
 * @param className - зарезервированная переменная для изменения стиля
 *                     css
 * @param rotate - угол поворота энкодера. Данная переменная является
 *                 опциональной служит для конфигурации энкодера
 * @returns JSX - виртуальный элемент DOM дерева.
 *               См. https://sciter.com/tutorials/reactor-jsx/
*/
static get_svg(width, height, className = '', rotate=115){
    // Переменные для установки положения вала энкодера
    let cx_rotate, cy_rotate;
    // Переменные указывающие

    // центр вала энкодера
    let cx = 17.5;
    let cy = 12.5;

    // радиус вала энкодера
    let r = 9;

    // вычисление положения вала по углу поворота
    cx_rotate = cx + r * 0.65 * Math.cos(rotate * Math.PI / 180);
    cy_rotate = cy + r * 0.65 * Math.sin(rotate * Math.PI / 180);
    return (
        <svg style={`background:transparent;`}
            width={width}
            height={height}
            class={className}
            xmlns="http://www.w3.org/2000/svg">
            <g>
                <rect
                    style="fill:#ffffff;
                           fill-opacity:1;
                           stroke:#000000;
                           stroke-width:0.257106;">
                    width="35"
                    height="25"
                    x="0"
                    y="0" />
                <line style="stroke-width: 0.3;" x1="35" y1="5" />
            </g>
        </svg>
    );
}
```

```

                x2="45" y2="5" stroke="black" />
            <line style="stroke-width: 0.3;" x1="35" y1="10"
                  x2="45" y2="10" stroke="black" />
            <line style="stroke-width: 0.3;" x1="35" y1="15"
                  x2="45" y2="15" stroke="black" />
            <line style="stroke-width: 0.3;" x1="35" y1="20"
                  x2="45" y2="20" stroke="black" />
        </g>
        <g id="rotate">
            <circle
                style={`fill:#ffffff;
                        fill-opacity:1;
                        stroke:#000000;
                        stroke-width:0.264583;`}
                cx={cx}
                cy={cy}
                r={r}/>
            <circle style={`fill:#000000;
                        fill-opacity:1;
                        stroke:#000000;
                        stroke-width:0.264583;`}
                cx={cx_rotate}
                cy={cy_rotate}
                r="1"/>
        </g>
    </svg>);
}

/** @brief Метод вызывается при обновлении элемента.
 * Обновление элемента происходит при событиях
 * от пользователя (перенос элемента, клик по элемента и т.д.)
 * @returns JSX - виртуальный элемент DOM дерева. См.
https://sciter.com/tutorials/reactor-jsx/
 */
render(){
    return (<div class="draggable unselectable"
        id={this.target_name} titleid="elements"
        style={`top: ${this.top}; left: ${this.left};`}>
        {this.connectors}
        {Encoder.get_svg(this.width, this.height, "", this.rotate)}
    </div>)
}

/** @brief Метод вызывается при событии прокрутки колеса мыши в режиме
 * эмуляции
 * Здесь реализован поворот энкодера
 * @param evt - см. https://developer.mozilla.org/en-US/docs/Web/API/Element/wheel\_event
 * @returns None
 */
Wheel(evt){

```

```
let wheel;
// Получить пин с именем "A". Имена пинов описаны в константе pins выше
let connectorA = this.$("div#A");
// Получить пин с именем "B"
let connectorB = this.$("div#B");

// Определение вращение колеса мыши
if(evt.deltaY < 0){
    wheel = this.attrs.Step;
    // Приращение угла на значение заданное в конфигурации
    this.rotate += wheel;
    // вызов функции установки значения на пин "A"
    encoder_step(connectorA)
} else{
    wheel = -this.attrs.Step;
    this.rotate += wheel;
    encoder_step(connectorB)
}
// вычисление количества оборотов
let turn = Math.floor(this.rotate / 360)
if (turn !== this.turns){
    this.turns = turn;
    // Получить пин с именем "Turn_switch"
    let connector = this.$("div#Turn_switch")
    // Установить значение на выход пина "Turn_switch"
    connector.set_out(1);
    connector.set_out(0);
}

/** @brief Функция реализует эмуляцию работы энкодера */
function encoder_step(conn){
    conn.set_out(1)
    conn.set_out(0)
}
}

/** @brief Событие вызывающееся при нажатии на левую кнопку мыши
 * @param evt - см. https://developer.mozilla.org/en-US/docs/Web/API/Element/mousedown\_event
 * @returns None
*/
MouseDown(evt){
    let connector = this.$("div#Switch")
    connector.set_out(1);
}

/** @brief Событие вызывающееся при отжатии левой кнопки мыши
 * @param evt - см. https://developer.mozilla.org/en-US/docs/Web/API/Element/mouseup\_event
```

```
* @returns None
*/
MouseUp(evt){
    let connector = this.$("div#Switch")
    connector.set_out(0);
}
/** Привязка функции к глобальной переменной для вызова из эмулятора
 */
globalThis["Encoder"] = function(param){
    return <Encoder tagname={param.tagname} x={param.x} y={param.y} height={25}
width={45} editor={param.editor} attrs={param.attrs}></Encoder>
}
globalThis["Encoder"].get_svg = Encoder.get_svg;
```

Ниже приведено описание кода пользовательского элемента.

Константа `pins` служит для конфигурации интерфейсов входа/выхода элемента. Ниже рассмотрим все поля данной константы:

- `name` – имя контакта. Все имена контактов добавляемого элемента должны быть уникальны;
- `type` – тип контакта (input/output);
- `x, y` – положение контакта для svg рисунка;
- `index` – порядковый номер контакта.

Константа `SETTINGS` необходима для конфигурации элемента см. п.8.8.5. Для конфигурации доступно несколько видов отображения изменяемых параметров:

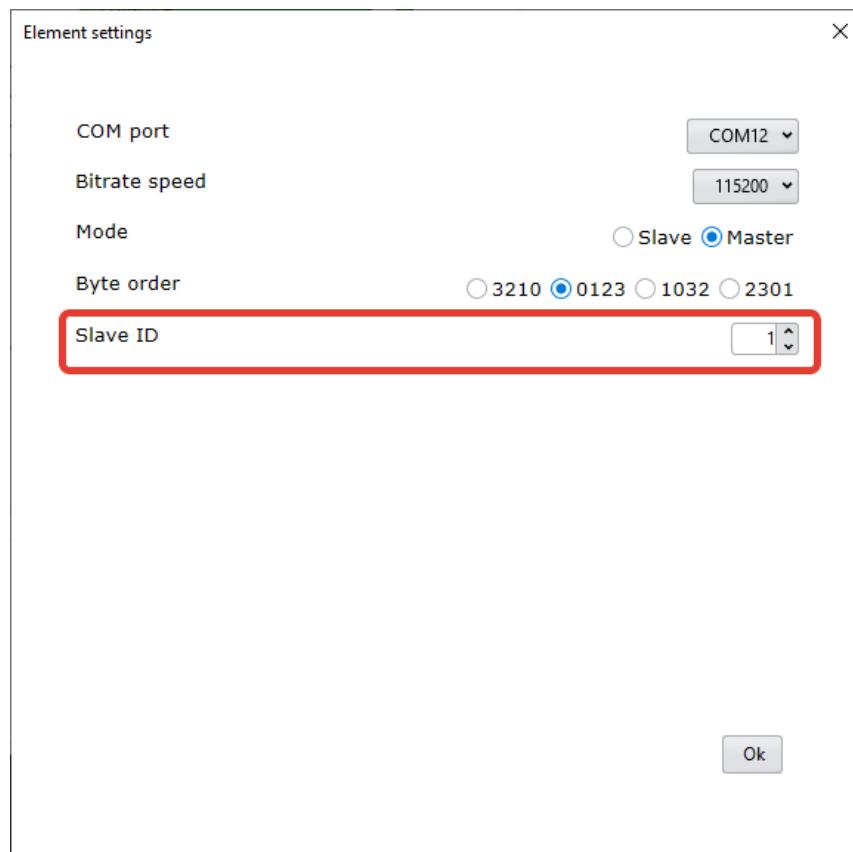


Рисунок 23.1 – Вид конфигурации типа “*number*”.

- type: “*number*” – изменение числового значения:
 - parameter_name: “Name” – имя параметра, которое будет отображаться в окне конфигурации элемента;
 - value: 1 – значение параметра по умолчанию;
 - max_value: 247 – максимальное значение параметра;
 - min_value: 1 – минимальное значение параметра;
 - step: 1 – шаг изменения параметра.

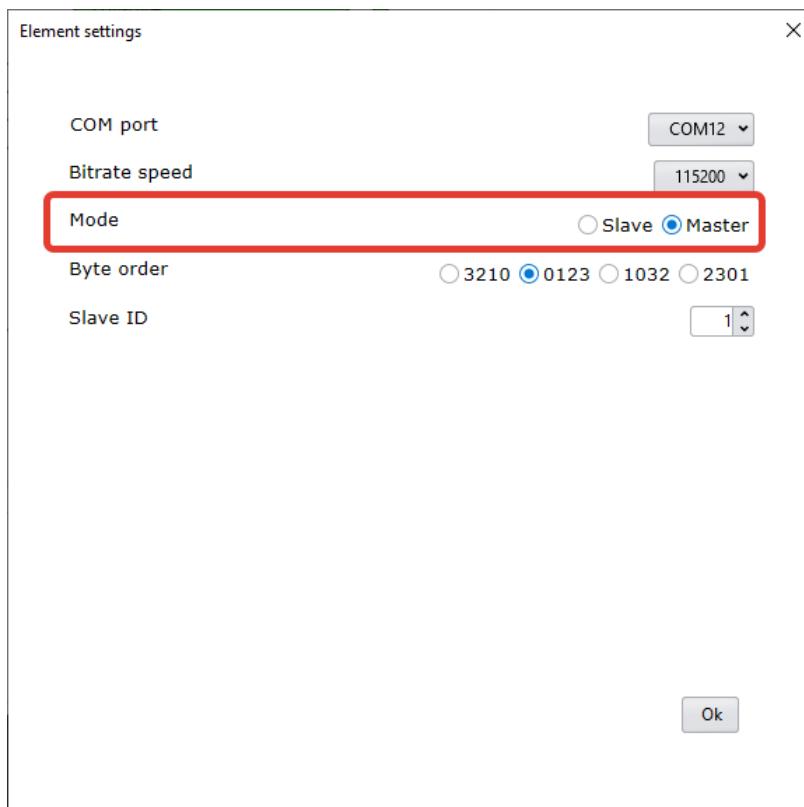


Рисунок 23.2 – Вид конфигурации типа “*radio button*”.

- type: “*radio button*” – выбор одного из предопределенных параметров:
 - parameter_name: “Name” – имя параметра, которое будет отображаться в окне конфигурации элемента;
 - options : [] – список параметров;

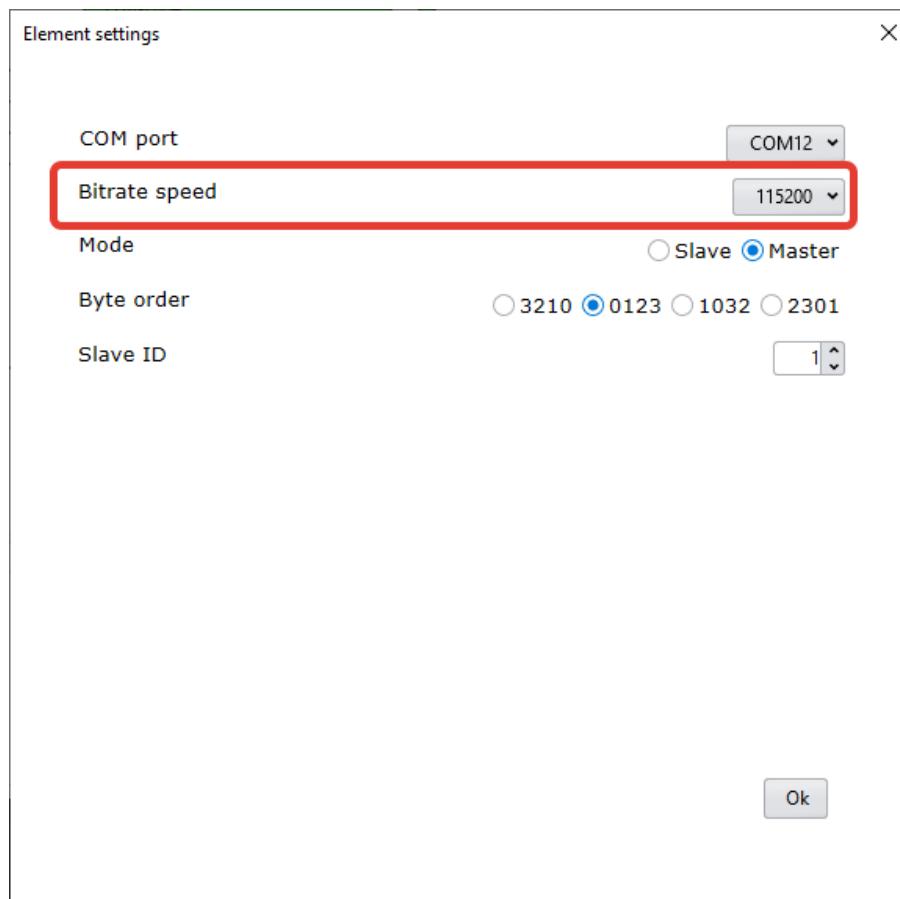


Рисунок 23.3 – Вид конфигурации типа “*drop list*”.

- type: “*drop list*” – выбор предопределенного параметра из выпадающего списка:
 - parameter_name: “Name” – имя параметра, которое будет отображаться в окне конфигурации элемента;
 - options : [] – список параметров

Константа `DEFAULT_ATTRS` необходима для установки параметров конфигурации по умолчанию.

Для создания элемента необходимо создать класс, наследовавшись от класса `SimElement`, который находится в директории «`beremiz/emulator/Sciter/editor/sim_elem.js`». В данном классе реализованы основные методы работы в эмуляторе. Для обработки событий необходимо переопределить методы класса `SimElement`. Ниже перечислены методы, которые доступны в эмуляторе. Все методы имеют аргумент типа `event`, для получения дополнительной информации см. документацию javascript.

- `Wheel(evt)` – событие, вызывающееся после прокрутки колеса мыши;
- `MouseDown(evt)` – событие, вызывающееся после нажатия кнопки мыши;
- `MouseUp(evt)` – событие, вызывающееся после отпускания кнопки мыши;
- `MouseDragRequest(evt)` – событие, вызывающееся при drag and drop;
- `DblClick(evt)` – событие, вызывающееся при двойном клике.

Стоит помнить, что данные события обрабатываются только при запуске эмуляции.

У каждого класса должен быть статический метод для вывода svg изображения. Этот метод используется для отображения элемента в библиотеке элементов эмулятора.

Вся логика работы элемента задается при помощи событий описанных выше.

После реализации класса необходимо добавить функции создания элемента в окне редактора, а так же функцию отображение элемента в библиотеке. Добавление этих функций происходит при помощи глобальной переменной `globalThis`.

Так как для элементов используется глобальная переменная, повторение имен недопустимо. Имя файла элемента должно соответствовать имени класса, а так же ключу элемента `globalThis` (из листинга выше `globalThis["Encoder"]` - `globalThis` имеет ключ `"Encoder"`, имя файла `Encoder.js`).

Метод `render` класса `Encoder` обеспечивает отображение в окне редактора. При срабатывании события прокрутки колеса мыши, метод, который отрабатывает это событие, вращает энкодер. Таким образом, можно реализовать анимацию движения элементов.

При создании элемента и его настройке доступна отладка. Для отображения отладки откройте файл `inspector.exe` в директории «`beremiz/emulator/`». После запуска приложение `inspector` ожидает запуска эмулятора. Как только программа эмулятор запустится, `inspector` отобразит отладочную информацию (рисунок 23.4)

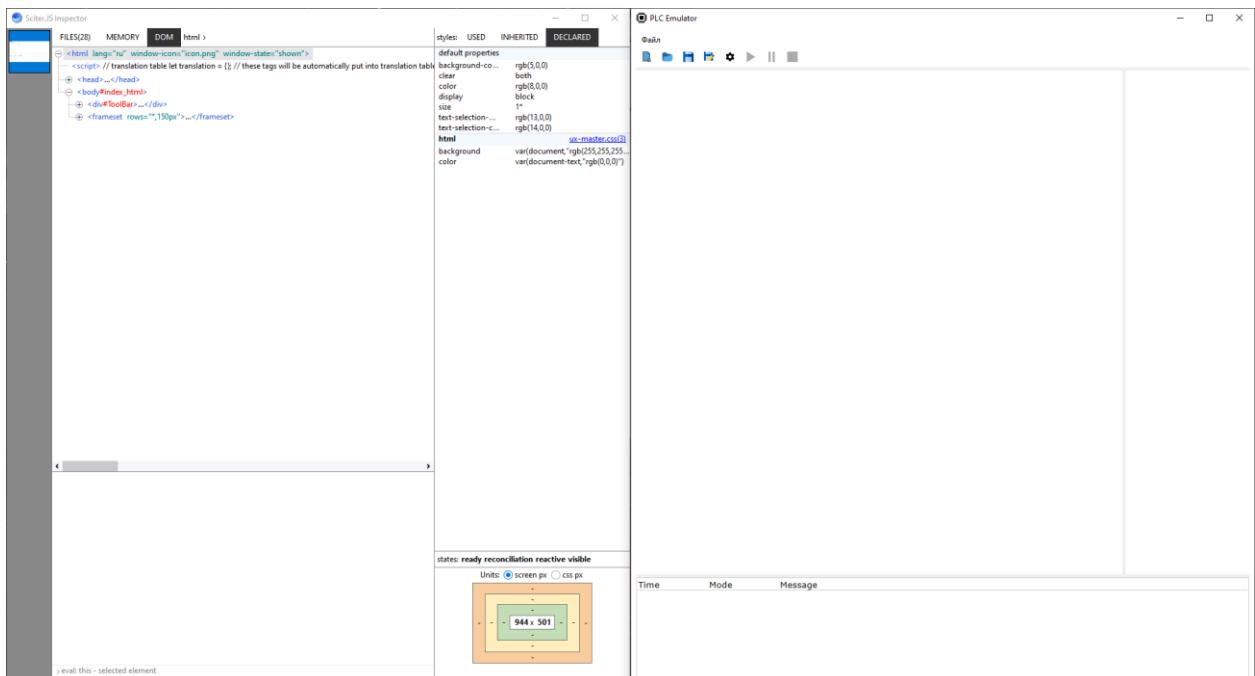


Рисунок 23.4 – Отладка приложения PLC Emulator.

Для отладки можно использовать DOM дерево (рисунок 23.5), а так же `console.log` функцию в javascript (рисунок 23.6).



Рисунок 23.5 – DOM дерево.

```
▶ Element(div#Encoder347.draggable.unselectable)
  ▼ Element(div#Encoder347.draggable.unselectable)
    editor : ▶ Element(div#editor_app)
    target : undefined
    kids : ▶ Array(0)
      top : 155
      left : 290
      props : ▶ Object
      height : 25
      width : 45
    target_name : "Encoder347"
    connectors : ▶ Array(4)
      attrs : ▶ Object
    lines_path : ▶ Array(0)
      undo : ▶ Array(0)
      redo : ▶ Array(0)
    default_rotate : 115
    settings : ▶ Array(1)
      rotate : 115
      turns : 0
  Symbol.factory : ▶ class Encoder extends SimElement{
    // Константа для установки положения энкодера
    default_rotate = 115;
    /** Конструктор класса параметры передаются в reactor
     * см. https://github.com/c-smile/sciter-js-sdk/tree/main.
     */
    constructor(props, kids){
      // Вызов конструктора родителей
      super(props, kids);

      // Установка I/O
      this.set_connectors(pins);
      // Проверка заданных атрибутов
      if (this.attrs.length == 0){
        // Установка значений конфигурации по умолчанию
        this.attrs = DEFAULT_ATTRS;
      }

      // Установка параметров конфигурации окна настроек
      this.settings = SETTINGS;
      // Переменная определяющая положение вала энкодера
      this.rotate = this.default_rotate;
      // Счетчик оборотов вала энкодера
      this.turns = 0;
    }
  }
```

Рисунок 23.6 – Использование функции console.log.