

Beremiz Documentation

1	Назначение и условия применения программы	3
2	Характеристики программы	5
2.1	Инструкции по установке	5
2.1.1	Установка среды разработки «Beremiz» под операционную систему Windows 7	5
2.2	Руководство программиста	8
2.2.1	Обращение к программе. Входные и выходные данные	8
2.2.2	Основные термины и определения	9
2.2.3	Основные компоненты среды Beremiz	11
2.2.4	Работа с проектом	63
2.2.5	Описание библиотеки функций и функциональных блоков	115
2.3	Языки стандарта МЭК 61131-3	124
2.3.1	Общие сведения о языке ST	124
2.3.2	Общие сведения о языке IL	132
2.3.3	Общие сведения о языке LD	134
2.3.4	Общие сведения о языке SFC	138
2.3.5	Общие сведения о языке FBD	145
2.4	Beremiz: Руководство разработчика	148
2.4.1	Документация softPLC	148
2.5	Участие в разработке документации	150
2.5.1	Быстрый старт	150
2.5.2	Настройка окружения и сборка документации	150
2.5.3	TODO	151
	Алфавитный указатель	153

Примечание: Документация находится в стадии разработки.

Устаревшая pdf-версия данного руководства [доступна](#) на сайте ИНЭУМ.

В данном руководстве 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 операционные системы Windows и Linux, описание стандартных функциональных блоков доступных для оператора и общие сведения о языках стандарта IEC 61131-3.

Назначение и условия применения программы

Среда разработки «Beremiz» предназначена для создания и отладки прикладных программ на языках стандарта ИЕС 61131-3 для целевых устройств (программируемых логических контроллеров) на базе СМ1820М. В качестве языков описания алгоритмов и логики работы данных программ, могут выступать как текстовые Structured Text (далее ST) и Instruction List (далее IL), так и графические Function Block Diagram (далее FBD), Ladder Diagram (далее LD), Sequential Function Chart (далее SFC).

Характеристики программы

Среда разработки «Beremiz» может выполняться на операционных системах Windows и Linux. Она написана с использованием кроссплатформенных языков Python, C, C++ и дополнительных библиотек к ним. Поэтому для запуска и работы «Beremiz», необходима сборка интерпретатора Python с определённым набором установленных пакетов (библиотек).

2.1 Инструкции по установке

2.1.1 Установка среды разработки «Beremiz» под операционную систему Windows 7

Для нормальной работы «Beremiz» под операционной системой Windows необходимо наличие следующих программных средств:

- интерпретатор Python версии 2.7 со всеми необходимыми библиотеками (wxPython-2.8, twisted, simplejson, Pyro, numpy, nevow);
- UNIX-подобная среда Cygwin с интерфейсом командной строки для Windows и установленным кросскомпилятором для целевой архитектуры x86.

Интерпретатор языка Python и UNIX-подобная среда Cygwin находятся в папке BeremizProject на DVD диске. Для установки необходимо:

1. скопировать папку BeremizProject в удобное для Вас место на жестком диске;
2. перенести папку python из BeremizProject, например, на диск C;
3. добавить путь к перенесённой папке python в «Переменных среды»

Добавление интерпретатора Python в переменные среды

Добавление интерпретатора языка Python осуществляется с помощью настроек операционной системы. Необходимо перейти в «Панель управления» и выбрать раздел «Система» (см. рис. 1.1).



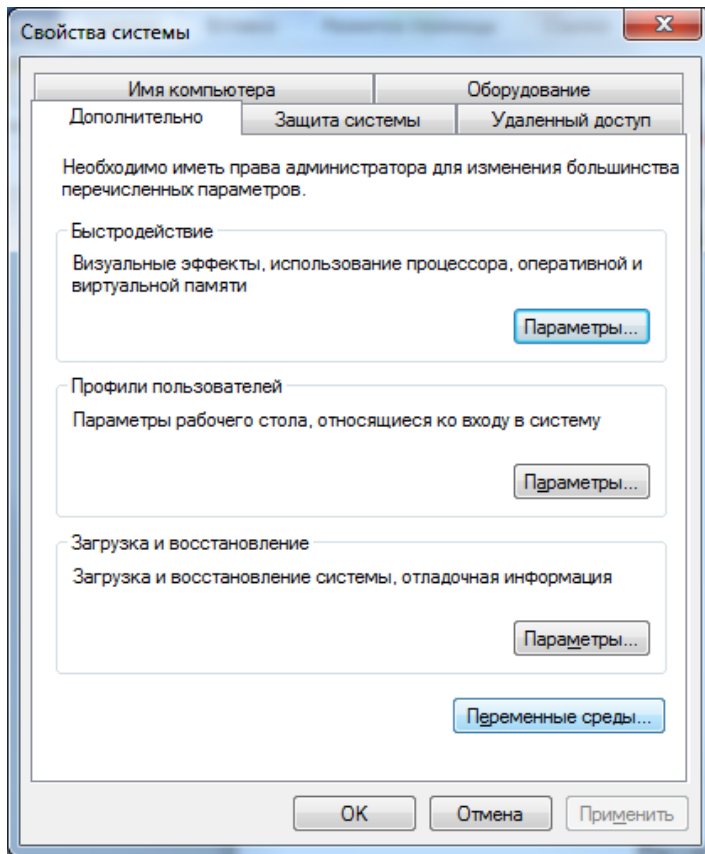


Рис. 1.3 – Диалог «Свойства системы» в Windows 7

Необходимо найти в разделе «Системные переменные» переменную «Path» (см. рис. 1.4) и нажать кнопку «Изменить».

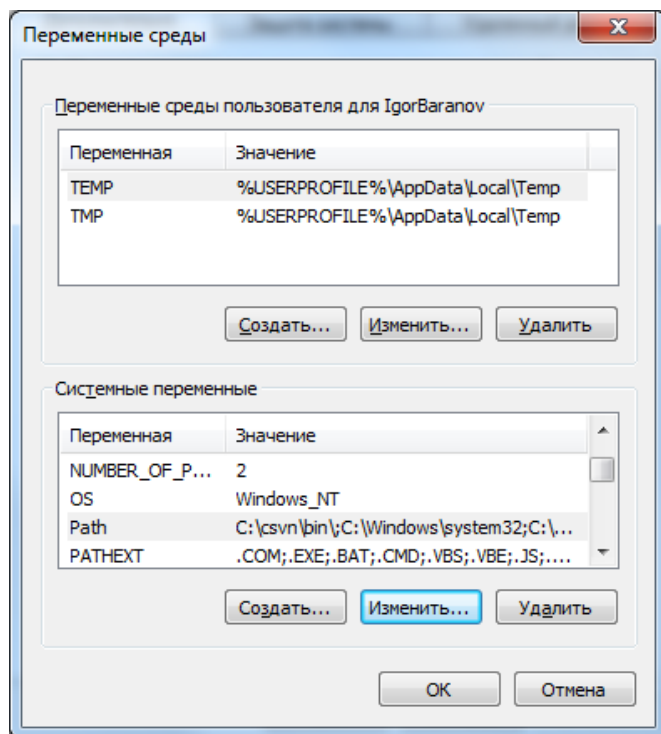


Рис. 1.4 – Диалог «Переменные среды» в Windows 7

В появившемся диалоге в поле «Значение переменной» добавляем, например, в середину строки, строку «C:\Python» (см. рис 1.5).

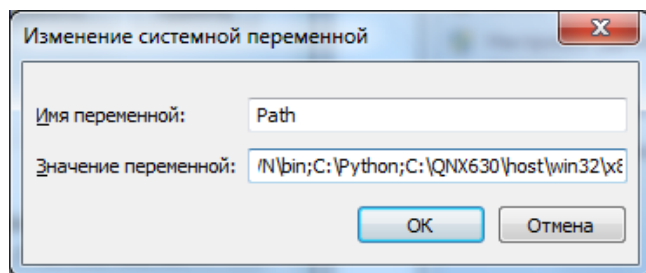


Рис. 1.5 – Диалог изменения системной переменной под операционной системой Windows

Далее во всех открытых диалогах, необходимо нажать «ОК», чтобы сохранить изменения. Теперь можно создать ярлык на рабочем столе со следующим адресом: “<путь с исполняемому файлу интерпретатора python >\pythonw.exe” “<путь к Beremiz>\beremiz\Beremiz.py”.

2.2 Руководство программиста

2.2.1 Обращение к программе. Входные и выходные данные

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

- Файл <название проекта> содержащий промежуточный код на языке ST, сгенерированный для всех программных модулей и ресурсов, транслируемый в язык C;
- Файлы: config.c config.h, POUS.h, POUS.c и файлы, соответствующие ресурсам - содержат код (на языке C) реализации алгоритмов и логики работы программных модулей и ресурсов проекта;
- Файлы plc_common_main.c и plc_debugger.c содержат код специфичный для целевой архитектуры и код для отладки прикладной программы на целевом устройстве из среды разработки Beremiz соответственно;
- Файлы, содержащие код драйверов на языке C для взаимодействия с внешними модулями УСО;
- Исполняемый файл в виде динамической библиотеки (с расширением so), компилируемый из этих вышеперечисленных C файлов.

Сгенерированный C код, с помощью кросскомпилятора, запущенного под UNIX-подобной оболочкой, компилируется в исполняемый бинарный файл, представленный в виде библиотеки.

Исполняемый файл, благодаря средствам Beremiz, может быть размещен на целевом устройстве через локальную сеть.

На целевом устройстве исполняемый файл запускается и в процессе работы выполняет следующие действия (см. [Рисунок 2.1](#)):

- 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.

2.2.2 Основные термины и определения

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

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

Текстовые и графические редакторы языков стандарта IEC 61131-3;

Транслятор диаграмм графических языков в текстовый язык;

Транслятор текстового языка в язык C;

Механизмы плагинов для взаимодействия с модулями УСО;

Механизмы добавления компиляторов под целевую платформу;

Механизмы соединений с целевыми устройствами;

Отладчик.

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

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

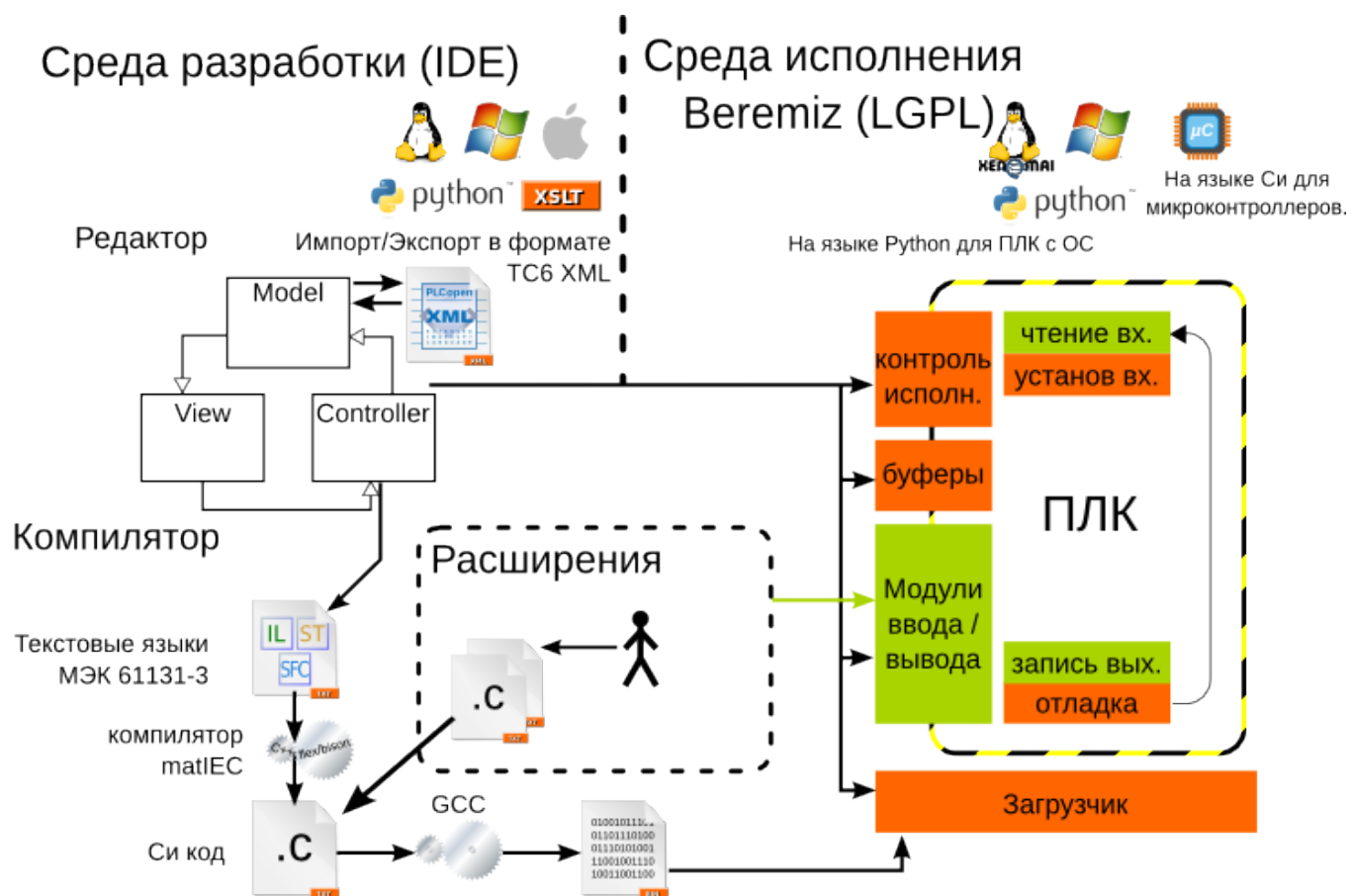


Рисунок 2.1: - Обобщенная схема инструментальной среды Beremiz

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

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

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

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

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

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

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

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

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

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

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

2.2.3 Основные компоненты среды Beremiz

Пользовательский интерфейс среды разработки Beremiz состоит из следующих компонент:

- Главное меню программы;
- Панель инструментов;
- Дерево проекта;
- Панель списка переменных и констант;
- Панель настроек проекта;
- Панель файлов проекта;
- Панель отображения промежуточного кода;
- Текстовые редакторы языков ST и IL;
- Графические редакторы языков FBD, SFC, LD;
- Панель редактирования ресурса;

- Панель экземпляров проекта;
- Панель библиотеки функций и функциональных блоков;
- Отладочная консоль;
- Поиск элементов в проекте;
- Панель отладки;
- Панель графика изменения значения переменной в режиме отладки.

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

Главное меню программы

Главное меню программы (см. [Рисунок 2.2](#)) содержит следующие пункты:

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

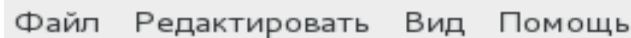


Рисунок 2.2: – Главное меню программы

Часть операций, выполняемых с помощью выбора определённого пункта меню мышью, могут быть исполнены с помощью «горячих клавиш». На выбор пользователя представлено два механизма работы горячих клавиш. Первый механизм использует первые буквы названия пунктов меню, для выбора пункта необходимо нажать (ALT + 'Клавиша первой буквы названия пункта в главном меню') затем можно выбрать операцию или подменю, нажав на клавиатуре первую букву названия соответствующего пункта. Второй механизм использует установленные клавиатурные сочетания, далее будет подробно описан каждый пункт меню и соответствующая ему (если определена) «горячая клавиша».

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

- «Новый» - создание нового проекта (CTRL + N);
- «Открыть» - открытие существующего проекта (CTRL + O);
- «Недавние проекты» - быстрое открытие одного из десяти последних, недавно редактированных проектов;
- «Сохранить» - сохранение текущего проекта пункт (CTRL + S);
- «Сохранить как» - сохранение текущего проекта в папку отличную от той, в которой он сохранён на данный момент (CTRL + SHIFT + S);
- «Закрыть вкладку» - закрытие активной вкладки (например, вкладки переменных плагина, конфигурации и т.д.) для открытого проекта (CTRL + W);
- «Закрыть проект» - закрыть текущий, открытый проект (CTRL + SHIFT + W);
- «Настройки страницы» - настройка параметров страницы для печати на принтере активной программы, представленной в виде диаграммы (CTRL + ALT + P);
- «Просмотр» - предварительный просмотр результата 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 стандарта МЭК 61131-3 и предоставляет следующие возможности:
- «Отмена» - 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» - поиск в текущем функциональном блоке (CTRL + F);
- «Поиск следующего» - подсветка следующего вхождения строки поиска (CTRL+K);
- «Поиск предыдущего» - подсветка предыдущего вхождения строки поиска (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:
- «Тип данных» - нового типа данных;
- «Функция» - новой функции;
- «Функциональный блок» - нового функционального блока;
- «Программа» - новую программу;
- «Ресурс» - новый ресурс;
- плагины для модулей УСО;
- «Выделить всё» - выделение всех элементов в активной вкладке редактора (CTRL + A);
- «Удалить» - удаление программного модуля, выделенного в дереве проекта.

Меню «Вид» предназначено для работы с редакторами языков стандарта IEC-61131 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» - убирает все панели, оставляя только рабочее поле(F12)
- «Сброс расположения панелей» - восстановление расположения панелей Beremiz в исходное состояние.

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

Панель инструментов

Панель инструментов представляет собой панель с кнопками для быстрого обращения к часто используемым функциям среды разработки Beremiz. Она состоит из нескольких панелей, содержащих кнопки: главного меню, 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 (см. п. 5.7).

Кнопки главного меню

Панель инструментов, содержащая кнопки главного меню представлена на [Рисунок 2.3](#).

Список кнопок и их функций описывается в таблице 1.

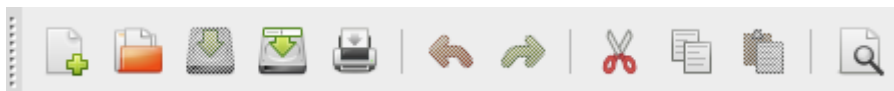


Рисунок 2.3: – Панель инструментов

Таблица 1 – Кнопки панели инструментов

Внешний вид кнопки	Название кнопки	Функция кнопки
	Новый проект	Создать новый проект
	Открыть проект	Открыть существующий проект
	Сохранить	Сохранить текущий проект
	Сохранить как	Сохранить текущий проект в определенную папку
	Печать	Печать на принтере текущей программы
	Отмена	Отмена последней манипуляции в редакторе
	Повторить	Повтор отмененной манипуляции в редакторе
	Вырезать	Удалить в буфер обмена выделенные в редакторе фрагменты
	Вставить	Вставить фрагменты из буфера обмена в редактор
	Поиск в проекте	Вызов диалога поиска данных в проекте

Кнопки сборки проекта и установки связи с целевым устройством

Панель, содержащая кнопки сборки проекта и соединения с целевым устройством, позволяет скомпилировать и скомпоновать текущий проект и, в случае, если эта операция завершилась успешно (данную информацию можно увидеть в отладочной консоли (см. п. 5.12.)), передать и запустить полученный исполняемый файл на целевом устройстве.

Часть кнопок данной панели показана на [Рисунок 2.4](#).

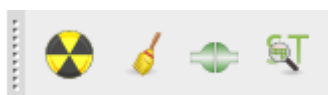









Рисунок 2.4: – Панель сборки проекта и соединения с целевым устройством

Таблица 2 – Кнопки сборки проекта и связи с целевым устройством на панели инструментов

Внешний вид кнопки	Название кнопки	Функция кнопки
	Сборка проекта в директории сборки	Полная сборка(компиляция и компоновка) текущего проекта в папку build, находящийся в папке, где хранится проект
	Очистить директорию сборки проекта	Удаление папки build, где был собран проект
	Подключиться к целевому ПЛК	Соединиться с целевым устройством по адресу URI, который был указан в настройках проекта
	Показать код, сгенерированный PLCGenerator	Показать код скомпилированного проекта языке ST
	Передать ПЛК	Перенести исполняемый файл, полученный в ходе сборки проекта, на целевое устройство
	Запустить ПЛК	Запустить на исполнение собранную прикладную программу на целевом устройстве
	Остановить запущенный ПЛК	Остановить исполнение прикладной программы 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 некоторые кнопки.

На [Рисунок 2.5](#) приведено состояние данной панели, когда соединение с целевым устройством установлено и на нём уже есть прикладная программа. Соответственно, можно запустить с помощью кнопки «Запуск прикладной программы» её или передать новую, используя кнопку «Передача прикладной программы».



Рисунок 2.5: - Панель инструментов сборки проекта и соединения с целевым устройством

В случае, когда при установке соединения произошли ошибки, данная информация будет выведена в отладочную консоль. Далее будет рассмотрен компонент «Дерево проекта», который представляет структуру элементов, составляющих проект.

Дерево проекта

Дерево проекта обычно расположено в левой части окна среды разработки Beremiz (см. [Рисунок 2.6](#)) и отображает структуру элементов, из которых состоит проект.

В роли элементов могут выступать:

- Ресурсы;

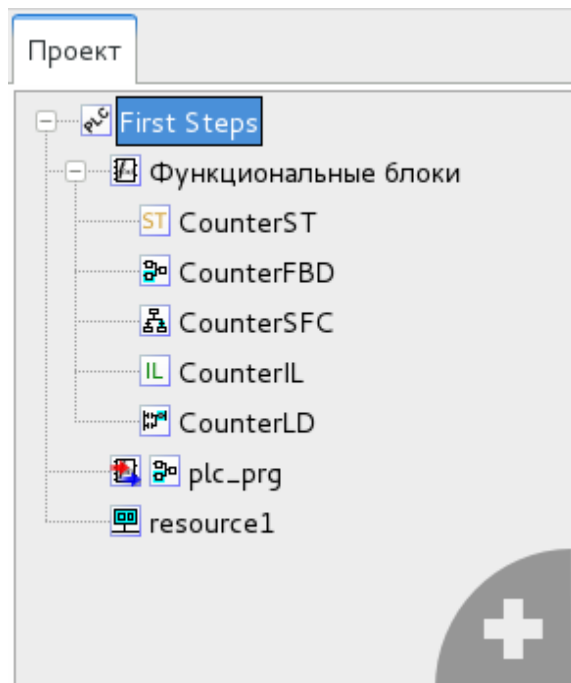


Рисунок 2.6: – Дерево проекта

- Программные модули (функции, функциональные блоки и программ) и их составные части;
- Типы данных;
- Плагины модулей УСО.

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

Добавление элемента в дерево проекта

В правом нижнем углу дерева проекта находится кнопка «+» (см. [Рисунок 2.7](#)), при нажатии на которую, появляется меню для выбора добавления необходимого элемента в проект.

В случае добавления программного модуля, т.е. выбора пункта «Функция», «Функциональный блок» или «Программа», появится диалог «Создать новый программный модуль» (см. [Рисунок 2.8](#)).

В данном диалоге три поля:

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

Имя, присвоенное по умолчанию, может быть заменено на имя, соответствующее назначению данного программного модуля. В зависимости от того, какой программный модуль был выбран во всплывающем меню, в поле «Тип программного модуля» будет подставлено именование данного программного модуля. В поле «Язык» необходимо выбрать из списка (см. [Рисунок 2.9](#)) один из языков стандарта IEC 61131-3 (IL, ST, LD, FBD, SFC), на котором будут реализованы алгоритмы и логика работы данного добавляемого программного модуля.

В случае выбора добавления типа данных, появится диалог (см. [Рисунок 2.10](#)), в котором необходимо указать механизм создания нового типа данных

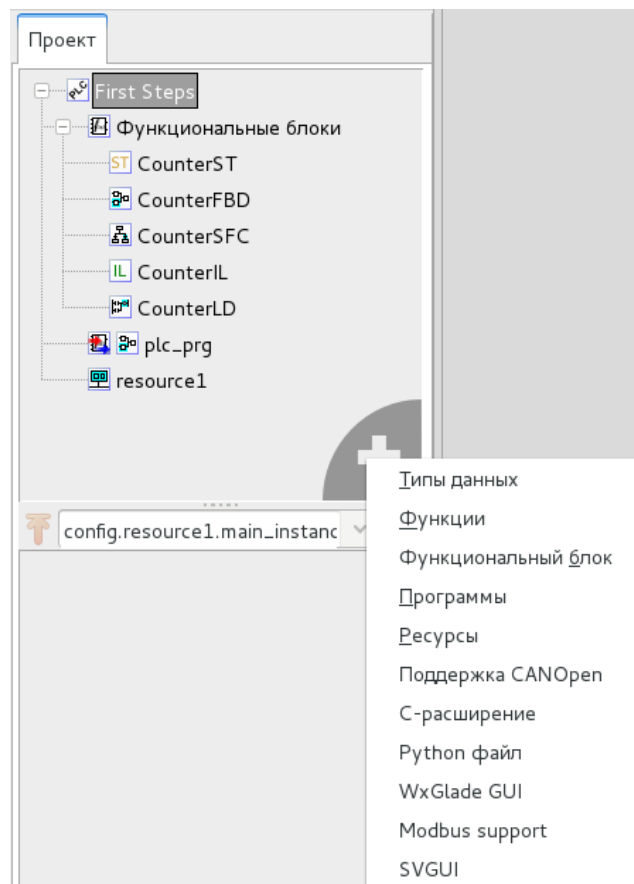


Рисунок 2.7: – Всплывающее меню добавления элементов проекта

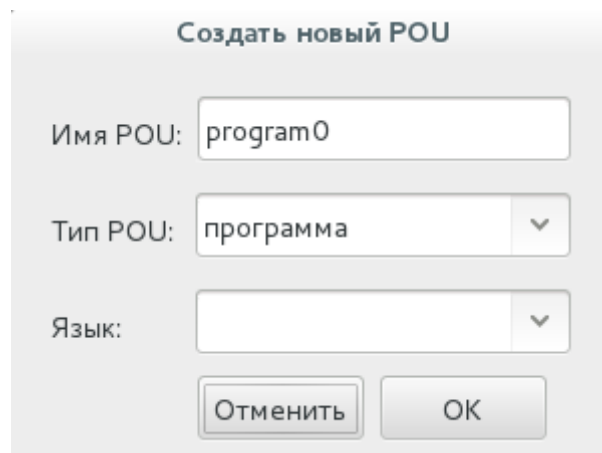


Рисунок 2.8: – Диалог добавления программного модуля

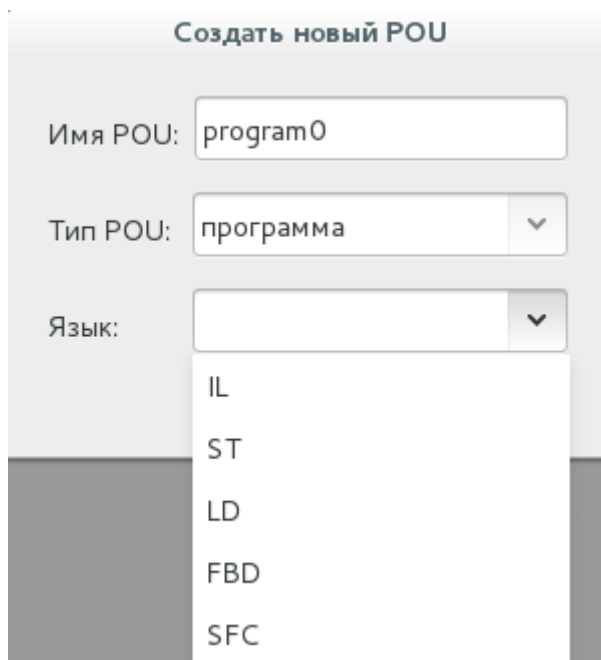


Рисунок 2.9: – Выбор языка программного модуля

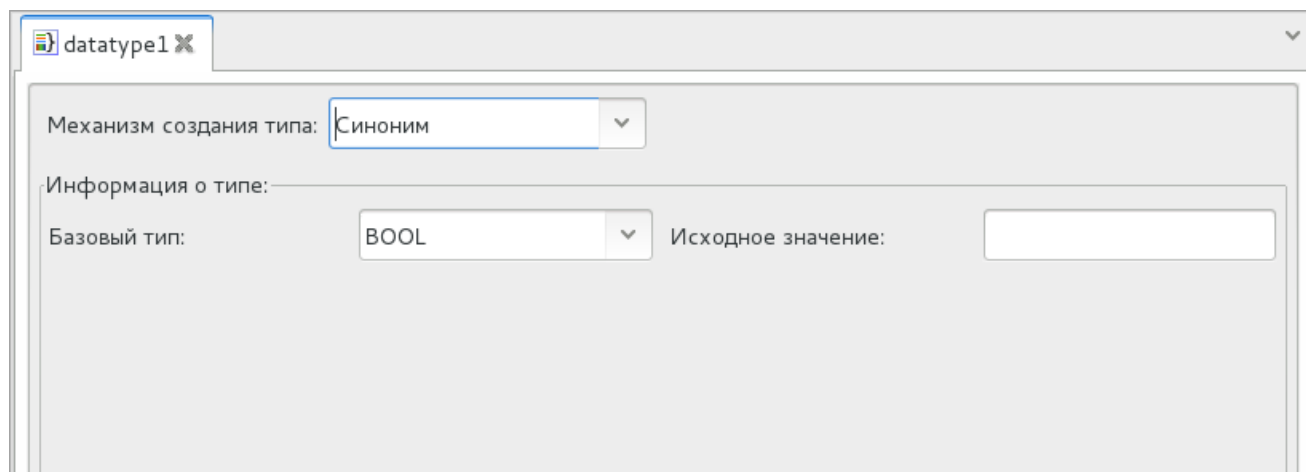


Рисунок 2.10: – Добавление пользовательского типа данных

Добавление нескольких элементов одного типа, например нескольких программ, функций, функциональных блоков приводит к их группировке в дереве проекта. Еще одним способом добавления нового элемента является нажатие правой клавиши мыши по определённому разделу в дереве проекта. Например, при нажатии на «Функциональные блоки», появится всплывающее меню (см. [Рисунок 2.11](#)). В данном меню можно выбрать «Добавить программный модуль» или «Вставить программный модуль», если он был скопирован в буфер обмена.

Добавление нового элемента или выбор существующего в дереве проекта приводит к появлению панели редактирования и настроек соответствующего элемента:

- Панель настроек проекта;
- Панель, содержащая текстовый редактор языков ST и IL;
- Панель, содержащая графические редакторы диаграмм языков FBD, SFC, LD;
- Панель настроек ресурса;
- Панель редактирование типа данных;
- Панели настроек плагинов модулей УСО.

Каждая вышеперечисленная панель редактирования будет рассмотрена в последующих пунктах.

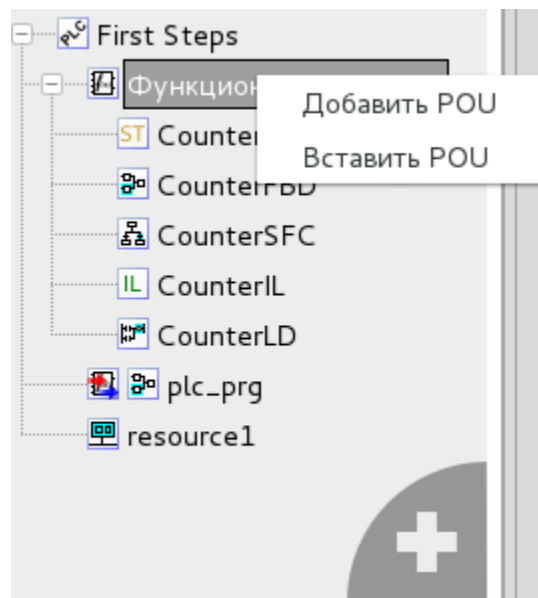


Рисунок 2.11: – Всплывающее меню добавления и вставки программного модуля

Удаление элемента в дереве проекта

Удаление осуществляется наведением на определённый элемент в дереве проекта и нажатием на него правой клавишей мыши, а далее в появившемся меню выбирается пункт «Удалить» (см. [Рисунок 2.12](#)).

Переименование, копирование и вставка программных модулей

Дерево проекта позволяет выполнять операции переименования, копирования и вставки для программных модулей. Копирование или переименование осуществляются с помощью нажатия правой клавиши мыши на элемент (см. [Рисунок 2.13](#)), соответствующий программному модулю в дереве проекта, и выбор соответствующего пункта появившегося меню.

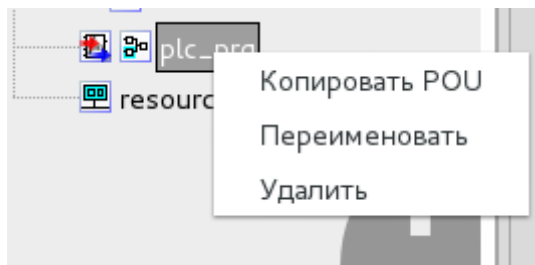


Рисунок 2.12: – Удаление элемента

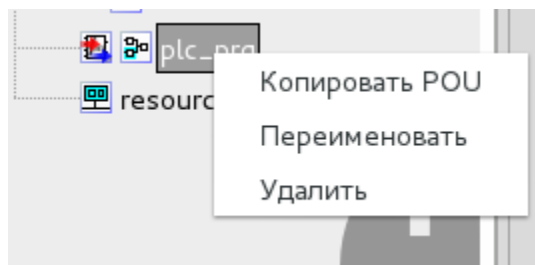


Рисунок 2.13: – Копирование и переименование элемента

Вставка программного модуля осуществляется в меню (нажатие правой клавишей мыши по данному элементу) корневого элемента дерева проекта, соответствующего проекту (см. [Рисунок 2.14](#)):

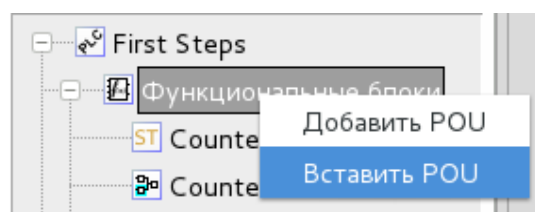


Рисунок 2.14: – Вставка программного модуля

Другим способом выполнения вышеописанной операции является вызов меню для элемента группировки программных модулей одного типа (см. [Рисунок 2.15](#)).

Далее приводится описание панели переменных и констант, которая присутствует при редактировании проекта, ресурса и программных модулей (функции, функционального блока, программы).

Панель списка переменных и констант

Панель списка переменных и констант (см. [Рисунок 2.16](#)) отображает с помощью таблицы переменные и константы, соответствующие выбранному программному модулю, ресурсу или в целом проекту.

Каждая переменная имеет следующие параметры:

- Имя, представляющее собой уникальный идентификатор переменной в пределах её области видимости и действия;
- Класс: «Глобальный», «Вход», «Выход», «Вход/Выход», «Локальный», «Внешний», «Временный»;

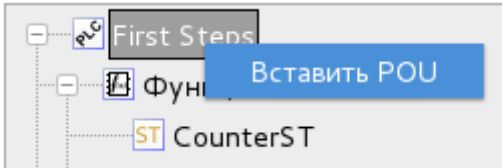


Рисунок 2.15: – Вставка программного модуля

test_main X							
Описание:		Фильтр класса: Все					
#	Имя	Класс	Тип	Размещение	Исходное значение	Настройка	Описание
1	PLC_OU	Локальный	INT	%QW0.0.2.8193.0			
2	PLC_IN	Локальный	INT	%IW0.0.2.8192.0			

Рисунок 2.16: – Панель переменных и констант

- Тип, определяющий тип переменной и может принадлежать базовому типу (в соответствии со стандартом IEC 61131-3), пользовательскому типу (псевдониму и поддиапазону существующего типа, перечислению, массиву, структуре) или типу функционального блока (стандартному или пользовательскому);
- Размещение - идентификатор, необходимый для связывания данной переменной с переменной плагина модуля УСО;
- Исходное значение значение - инициализация переменной некоторым начальным значением;
- Настройка - задание константности, ремаментности (сохранение её значения в энергонезависимой памяти) и неремаментности переменной;
- Описание - комментарий к назначению данной переменной или константы.

Первый символ имени переменной или константы должен быть буквой, или символом подчеркивания, далее могут следовать цифры, буквы латинского алфавита и символы подчеркивания. Набор возможных вариантов классов переменных зависит от типа элемента проекта, редактирования которого осуществляется. Двойной клик на полю «Размещение» вызывает появление кнопки «...», показанной на Рисунок 2.17 :

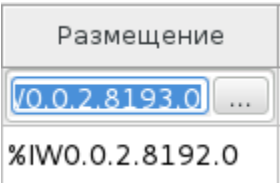


Рисунок 2.17: - Поле «Размещение» панели переменных и констант

Нажатие на данную кнопку приводит к появлению диалога «Просмотр директорий» (см. Рисунок 2.18), т.е. списка переменных модулей УСО, которые могут быть связаны с переменной в панели переменных и констант. При выборе в данном диалоге определённой переменной и нажатии клавиши «ОК» в поле «Адрес» будет добавлен адрес переменной внешнего модуля УСО.

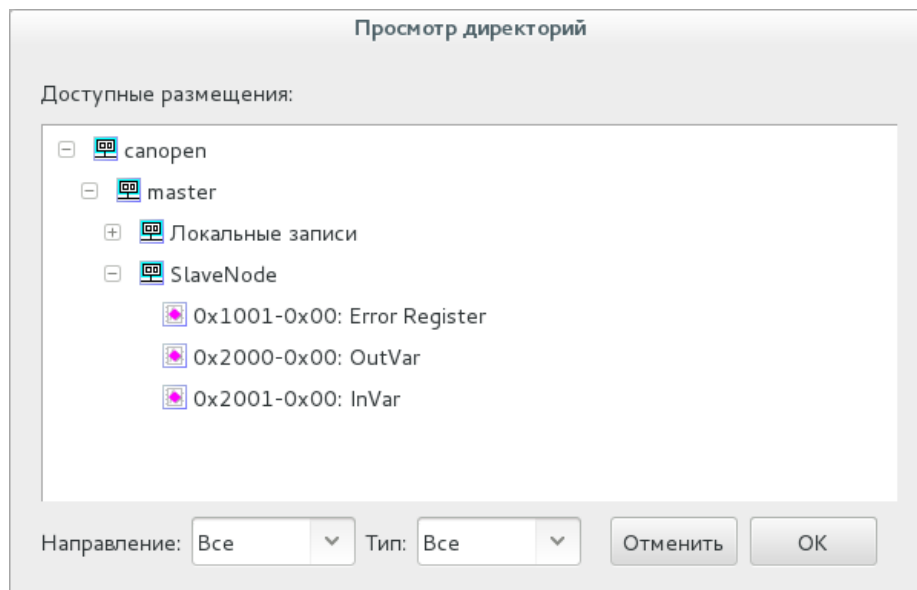


Рисунок 2.18: - Диалог «Просмотр адресов», вызываемый из поля «Адрес»

Поле опции позволяет определить переменную как константу. Соответственно, если компилятор обнаружит в коде фрагмент, в котором происходит изменение этой переменной - будет выведена ошибка компиляции «Assignment to constant variables is not be allowed» в «Отладочной консоли». Квалификатор «Константа» не может быть использован в объявлении функциональных блоков. Добавление, удаление и перемещение переменных происходит с помощью специальных кнопок на панели переменных и констант. Описания данных кнопок представлены в таблице 3.

Панель переменных и констант предоставляет возможность фильтровать отображаемые переменные по их конкретным классам («Вход», «Выход», «Вход/Выход»,

«Внешний», «Локальный», «Временный») или сгруппированным классам («Интерфейс» и «Переменные»). Данная операция выполняется с помощью функции «Фильтр класса» (см. [Рисунок 2.19](#)).

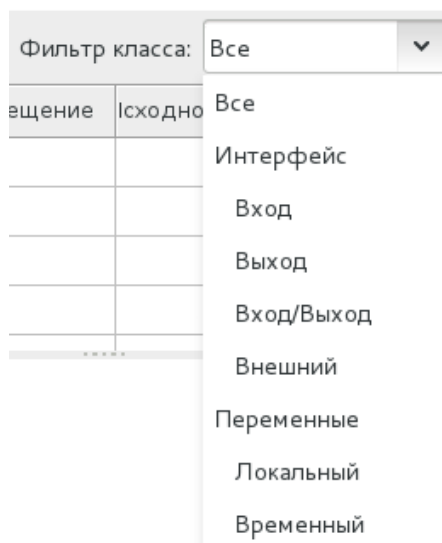






Рисунок 2.19: - Фильтрация отображения переменных в панели переменных и констант

Таблица 3 - Кнопки добавления, удаление и перемещения переменных на панели переменных и констант

Внешний вид кнопки	Название кнопки	Функция кнопки
	Добавить переменную	Добавить новую переменную в панель переменных и констант со значениями по умолчанию
	Удалить переменную	Удалить выделенную переменную или константу
	Переместить переменную вверх	Перемещение переменной в таблице переменных и констант вверх на одну позицию
	Переместить переменную вниз	Перемещение переменной в таблице переменных и констант вниз на одну позицию

Панель настройки проекта

Панель редактирования проекта (см. [Рисунок 2.20](#)) состоит из панели переменных и констант, а также настроек сборки проекта и данных о проекте.



Рисунок 2.20: - Панели настройки проекта

Настройки сборки проекта (см. [Рисунок 2.21](#)) позволяют задать следующие параметры:

- «URI системы исполнения» - унифицированный (единообразный) идентификатор ресурса, в данном случае под ресурсом понимается целевое устройство. Данный адрес необходим для режима отладки.
- «Запретить расширения» - установка данного флага позволяет не учитывать при сборке проекта внешние плагины;
- «Библиотеки» - подключаемые дополнительные библиотеки: «Native_Library», «Python_Library», «SVGUI_Library»;
- «Целевая платформа» - выбор из списка компилятора для архитектуры целевого устройства;
- «Компилятор» - имя исполняемого файла компилятора (если он определён в глобальных переменных среды), либо полный путь к нему;
- «CFLAGS» - указание флагов C компилятора;
- «Линковщик» - имя исполняемого файла компоновщика (если он определён в глобальных переменных среды), либо полный путь к нему;
- «LDFLAGS» - указание флагов компоновщика;

Также в настройках сборки проекта имеются две кнопки, описание которых приведено в таблице 4.

Таблица 4 - Кнопки в панели настройки сборки проекта

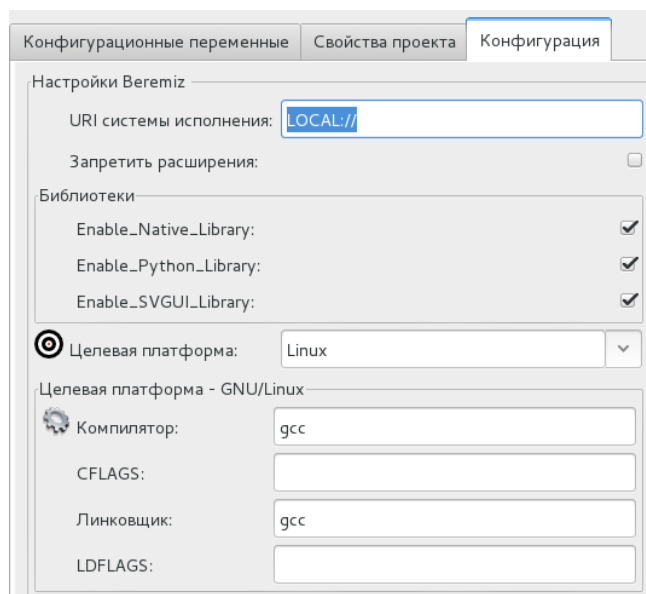


Рисунок 2.21: - Панель с настройками проекта

Внешний вид кнопки	Название кнопки	Функция кнопки
	МЭК - код	Вызов «Панели отображения промежуточного кода» (см. п. 5.5.1), для вывода кода, из которого генерируется ST код всего проекта
	Файлы проекта	Вызов «Панели файлов проекта», в которой можно выбрать файлы необходимые для передачи на целевое устройство вместе с исполняемым файлом (см. п. 5.5.2)

Вкладка «Проект» (см. [Рисунок 2.22](#)) позволяет задать: имя проекта, версию проекта, имя продукта, версию продукта и релиз продукта.

Рисунок 2.22: – Вкладка с данными проекта

Вкладка «Автор» (см. [Рисунок 2.23](#)) позволяет задать: Имя компании, URL-адрес компании, Имя

автора, Название организации.

Project Author Graphics Other

Company (mandatory): Avangard

Company website (optional):

Author name (optional):

Organization (optional):

Рисунок 2.23: – Вкладка данные об авторе проекта

Вкладка «Графика» (см. [Рисунок 2.24](#)) позволяет задать размеры страницы и разрешение сетки для редакторов диаграмм графических языков FBD, LD и SFC.

Project Author Graphics Other

Page size (optional):

Width: 0

Height: 0

Grid step:

FBD LD SFC

Horizontal: 5

Vertical: 0

Рисунок 2.24: – Вкладка графика

Вкладка «Прочее», изображенная на [Рисунок 2.25](#) , позволяет выбрать язык интерфейса для среды разработки Beremiz и указать дополнительное текстовое описание для проекта.

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

Панель отображения промежуточного кода

Данная панель (см. [Рисунок 2.26](#)) представляет собой текстовый редактор, отображающий с подсветкой синтаксиса и нумерацией строк код на языке ST, доступный только для чтения, без возможности редактирования.

Открытие данной панели доступно после сборки проекта с помощью соответствующей кнопки (см. таблицу 2).

Панель файлов проекта



Рисунок 2.25: – Вкладка с настройками языка и описанием проекта

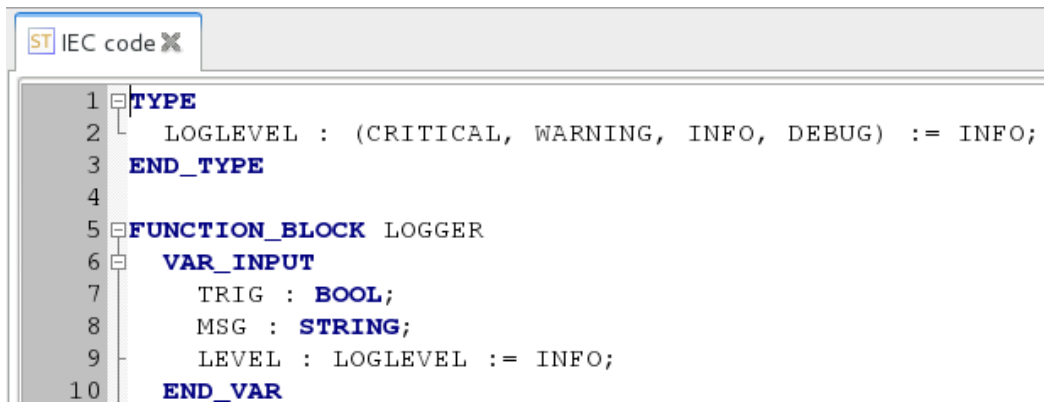


Рисунок 2.26: – Панель отображения промежуточного кода на языке ST

Панель файлов проекта (см. [Рисунок 2.27](#)) содержит встроенный проводник файлов (справа), в котором файлы могут быть выделены и перенесены в левую часть.

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

Таблица 5 - Кнопки в панели файлов проекта

Внешний вид кнопки	Название кнопки	Функция кнопки
	Удалить файл из левой директории	Удаление выделенного файла из левого списка добавленных файлов в проект
	Скопировать файл из правой директории в левую	Добавить выделенный файл из проводника файлов в проект
	Скопировать файл из левой директории в правую	Добавить в текущую папку проводника файлов слева выделенный файл в списке файлов проекта

Данные файлы 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.

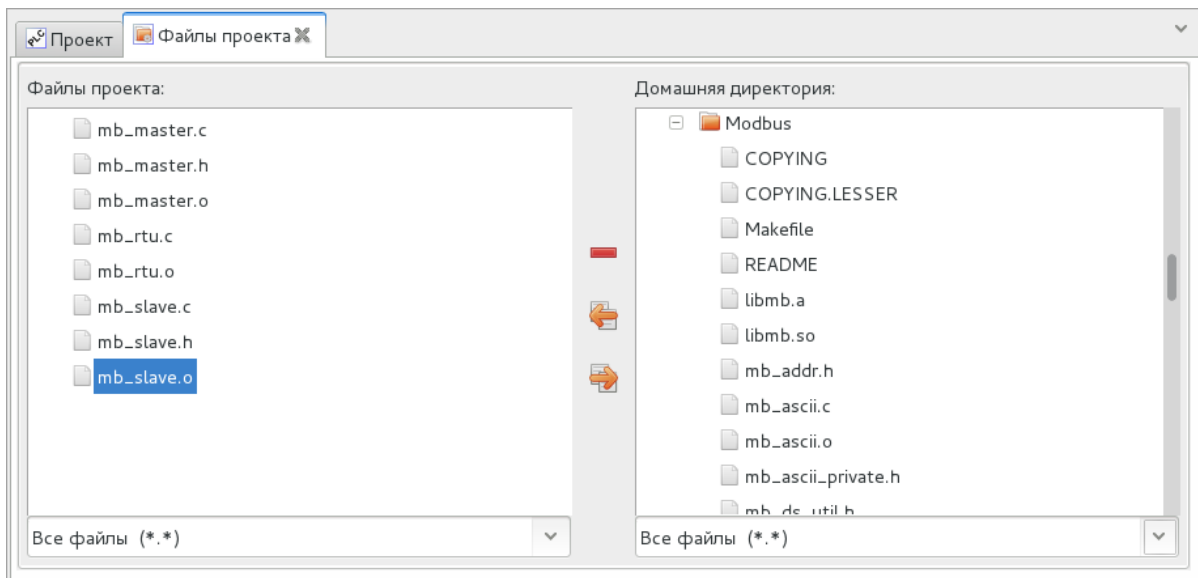


Рисунок 2.27: – Вкладка файлы проекта

Текстовый редактор языков ST и IL

Текстовый редактор языков ST и IL (см. [Рисунок 2.28](#)) позволяет создавать и редактировать алгоритмы и логику выполнения программных модулей на языках ST и IL.

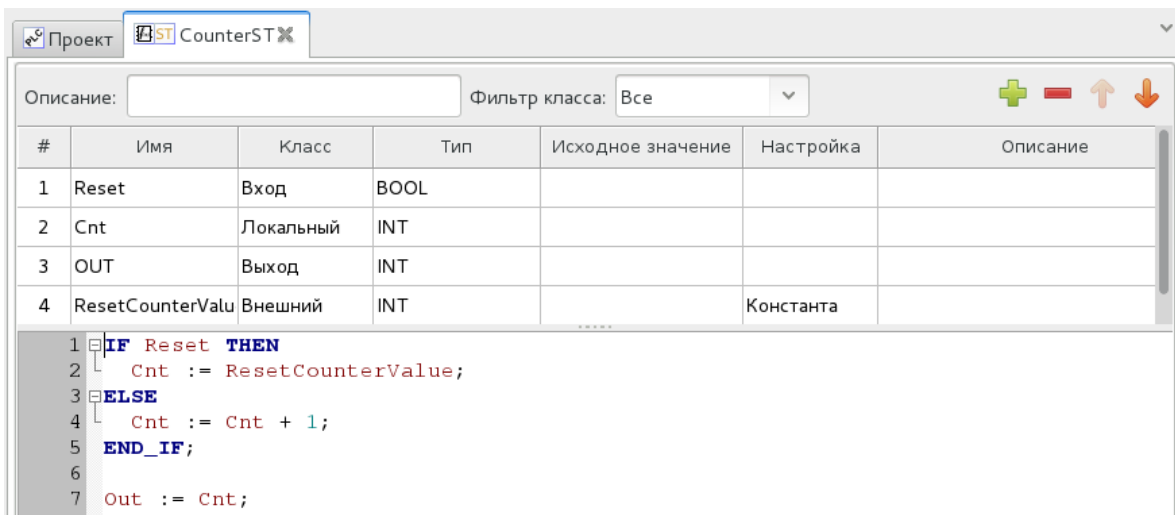


Рисунок 2.28: – Текстовый редактор языков ST и IL

Он обеспечивают 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** + <колёсико мыши>.

Описание синтаксиса, основных конструкций и примеров использования языков ST и IL приведены в описании языков стандарта МЭК 61131-3.

Графические редакторы диаграмм языков FBD, SFC, LD

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

Редактор языка FBD

Основными элементами языка FBD являются: переменные, функциональные блоки и соединения. При редактировании FBD диаграммы, в панели инструментов появляется следующая панель (см. [Рисунок 2.29](#)).



Рисунок 2.29: – Панель редактирования FBD диаграмм

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

Таблица 6 - Кнопки в панели файлов проекта

Внешний вид кнопки	Название кнопки	Функция кнопки
	Выделение объектов на диаграмме	Перевод указателя мыши в состояние, при котором можно осуществлять выделение объектов редакторе одного из графических языков
	Перемещение диаграммы	Перевод указателя мыши в состояние, при котором можно изменять размеры редактора одного из графических языков, с помощью его перемещения
	Создать новый комментарий	Вызов диалога создания комментария
	Добавить переменную	Вызов диалога добавления переменной
	Добавить функциональный блок	Вызов диалога добавления функционального блока
	Добавить подключение	Вызов диалога добавления соединения

Для этого необходимо указателем мыши выбрать необходимую кнопку и нажать на свободное место в области редактирования FBD диаграммы. В зависимости от выбранного элемента появятся определённые диалоги добавления данного элемента.

Аналогичные действия можно выполнить с помощью всплывающего меню в области редактирования FBD диаграмм. Вызов данного меню происходит нажатием правой клавишей мыши и выбором пункта «Добавить», в котором будет: «Блок», «Переменная», «Подключение», «Комментарий» (см. [Рисунок 2.30](#)).

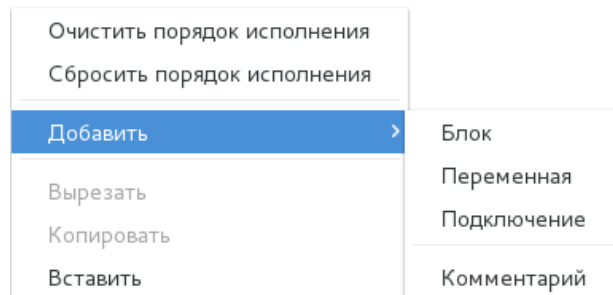


Рисунок 2.30: - Всплывающее меню редактора языка FBD

Далее рассмотрено добавление каждого элемента в отдельности.

Добавление функционального блока

При добавлении функционального блока одним из описанных выше способов, появится диалог «Свойства блока» (см. [Рисунок 2.31](#)).

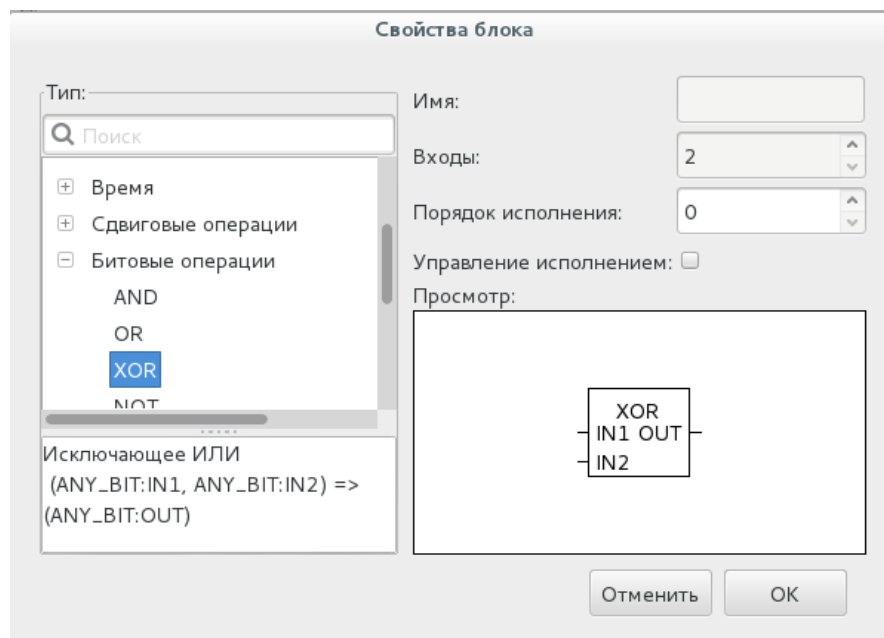


Рисунок 2.31: – Свойства функционального блока

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

Опция «Управление исполнением» добавляет в функциональный блок дополнительные параметры EN/ENO, о которых подробнее рассказано в приложении 5. Для сохранения изменений необходимо нажать «ОК». Одним из свойств является «Порядок исполнения».

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 (см. [Рисунок 2.31](#)).

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

Добавление переменной

Переменные добавляются из панели переменных и констант с помощью перетаскивания (Drag&Drop) левой клавишей мыши за область, выделенную красным цветом на [Рисунок 2.32](#), в область редактирования FBD диаграмм.

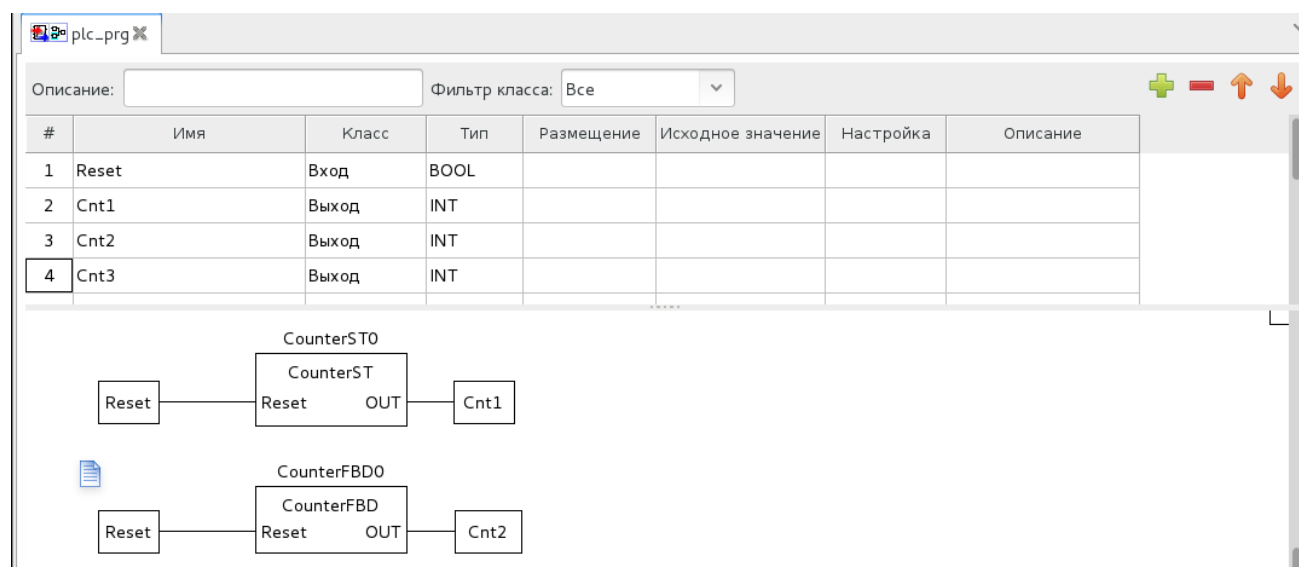


Рисунок 2.32: – Добавление переменной из панели переменных и констант

Изменить параметры переменной можно в диалоге «Свойства переменной» (см. [Рисунок 2.33](#)), нажав на неё два раза левой клавишей мыши.

В данном диалоге можно задать порядок исполнения переменной и изменить её класс («Вход», «Выход», «Вход/Выход»).

Добавление подключения

В тех случаях, когда необходимо передать выходное значение одного функционального блока на один из входов другого, удобно использовать элемент «Подключение». 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 «Подключение» позволяет избежать пересечения прямых соединений, которые приводит к тому, что схема становится менее понятной.

После выбора добавления элемента «Подключение» появится диалог «Свойства подключения» (см. [Рисунок 2.34](#)).

В данном диалоге можно выбрать тип соединения: «Выходное соединение» - для выходного значения, «Входное соединение» - для входного значения, а так же необходимо указать имя данного соединения. На [Рисунок 2.35](#) представлен пример использования 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 соеди-

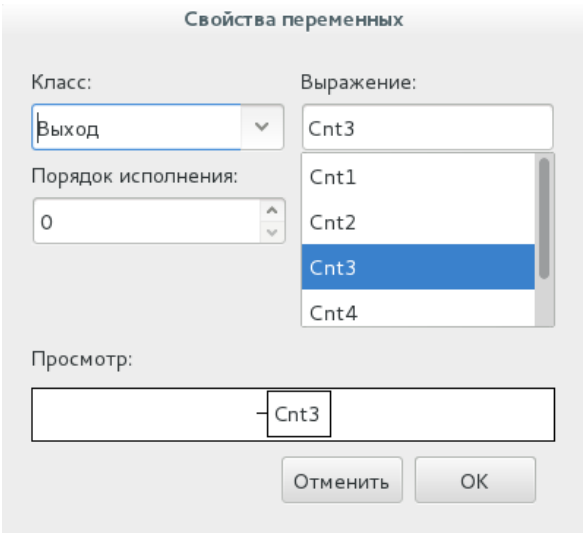


Рисунок 2.33: – Свойства переменной

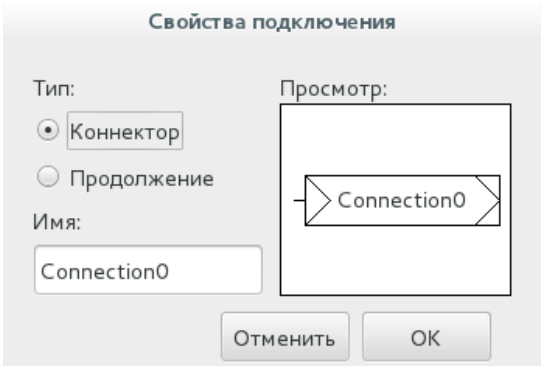


Рисунок 2.34: - Диалог добавления подключения для FBD

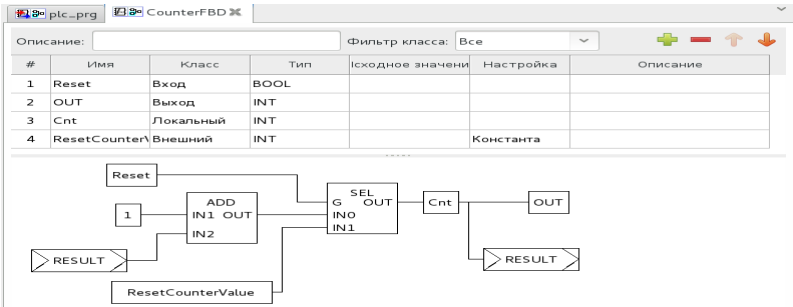


Рисунок 2.35: - Пример FBD диаграммы с использованием подключений

нение типа «Выходное соединение», в функции «MIN» - типа «Входное соединение». Имена у этих соединений, соответственно, одинаковые.

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 (см. [Рисунок 2.36](#)) для ввода текста комментария.

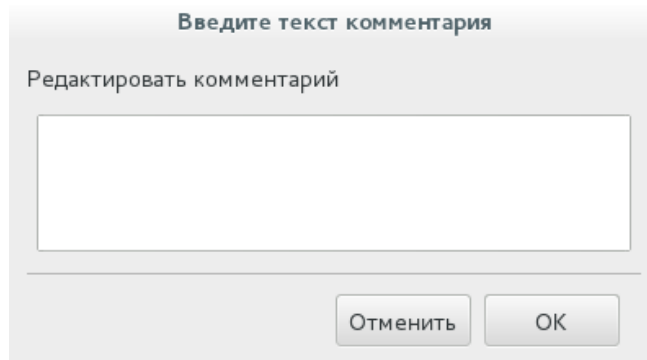


Рисунок 2.36: – Диалог добавления комментария

После нажатия кнопки «ОК» комментарий появится на диаграмме (см. [Рисунок 2.37](#))

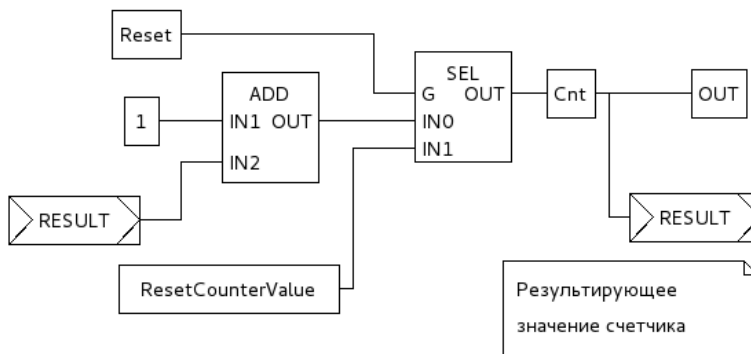


Рисунок 2.37: – Добавленный комментарий к FBD диаграмме

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.

Если обратиться к [Рисунок 2.38](#), то порядок выполнения функций будет следующим: 1 – CounterST0; 2 – CounterFBD0; 3 – CounterSFC0.

Данная опция «Порядок выполнения» выделена красным цветом на [Рисунок 2.39](#).

После задания порядка выполнения для каждой функции или функционального блока на схеме в правом нижнем углу будет указан его порядковый номер выполнения. Пример представлен на [Рисунок 2.40](#).

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

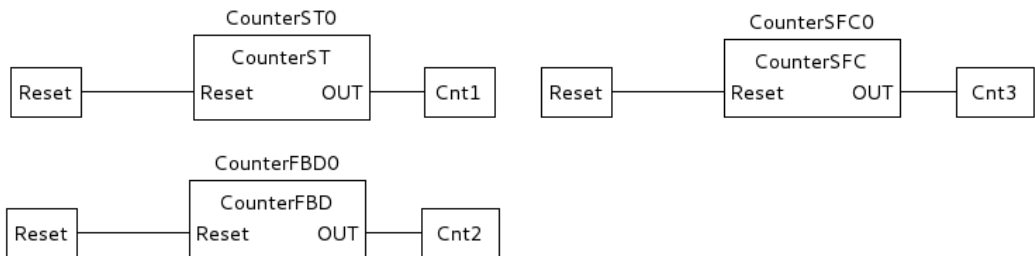


Рисунок 2.38: - Схема, содержащая функции с порядком выполнения (обсчета) по расположению

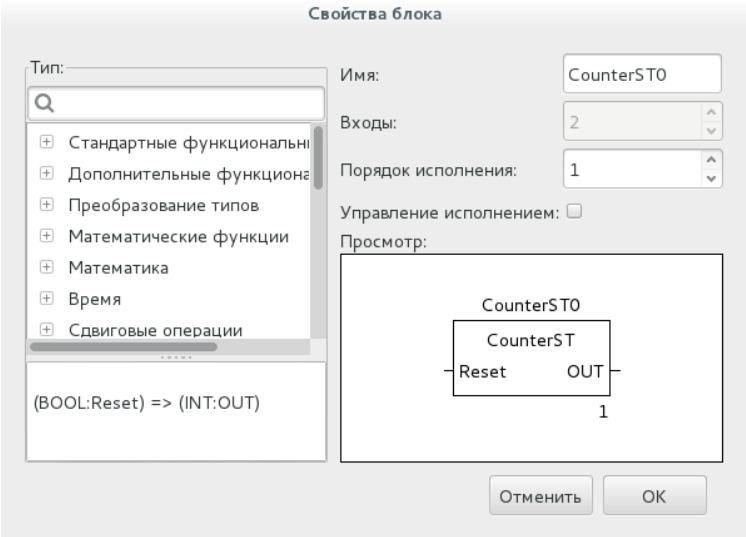


Рисунок 2.39: - Свойство порядок выполнения функции или функционального блока

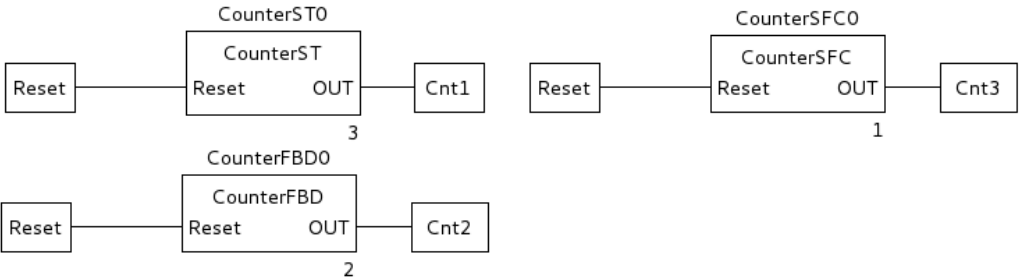


Рисунок 2.40: - Схема, содержащая функции с порядком выполнения заданным вручную

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 (см. [Рисунок 2.41](#)) с элементами языка LD.



Рисунок 2.41: - 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. Описание остальных кнопок, относящихся к языку FBD, находится в таблице 6.

Таблица 7 - Кнопки панели редактирования LD диаграммы

Внешний вид кнопки	Название кнопки	Функция кнопки
	Создать новую шину питания	Вызов диалога создания новой шины питания
	Создать новую катушку	Вызов диалога создания новой катушки
	Создать новый контакт	Вызов диалога создания нового контакта

Во всплывающем меню для редактора LD диаграмм (см. [Рисунок 2.42](#)), так же как и в панели инструментов помимо элементов LD языка, доступны элементы языка FBD.

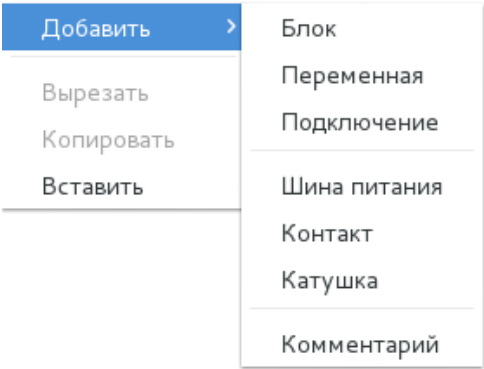


Рисунок 2.42: - Всплывающее меню редактора языка LD

Добавление шины питания

При добавлении шины питания, одним из описанных выше способов, появится диалог «Свойства шины питания» (см. [Рисунок 2.43](#)).

В данном диалоге указываются следующие свойства:

- Тип шины питания: шина питания слева или шина питания справа;
- Количество контактов на добавляемой шине питания.

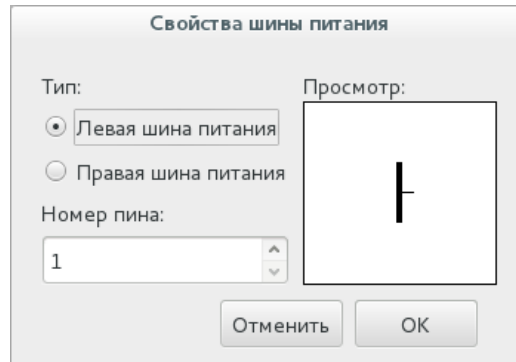


Рисунок 2.43: - Свойство шины питания

На [Рисунок 2.44](#) приведены две добавленные шины питания: левая с тремя контактами и правая с одним контактом.



Рисунок 2.44: - Шины питания на LD диаграмме

Добавление контакта

При добавлении контакта на LD диаграмму появится диалог «Редактирование значения контакта» (см. [Рисунок 2.45](#)).

Данный диалог позволяет определить модификатор данного контакта:

- «Обычный»;
- «Инверсия»;
- «Нарастающий фронт»;
- «Спадающий фронт».

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: «Вход»,

«Вход/Выход», «Внешний», «Локальный», «Временный». Для этого необходимо нажать левой кнопкой мыши за первый столбец (который имеет заголовок #) переменную, удовлетворяющую описанным выше критериям и перенести в область редактирования диаграммы (см. [Рисунок 2.46](#)).

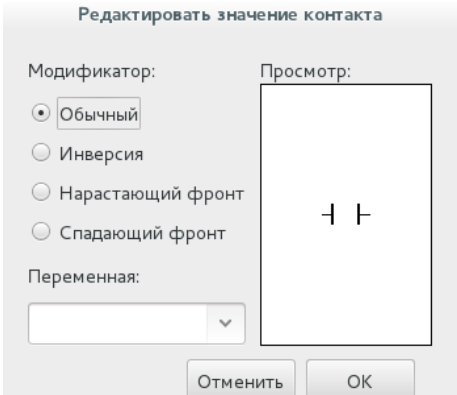


Рисунок 2.45: - Редактирование контакта

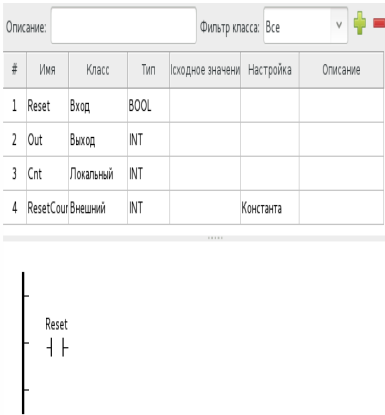


Рисунок 2.46: - Добавление контакт на диаграмму из панели переменных и констант

Добавление катушки

При добавлении катушки на LD диаграмму появится диалог «Редактирование значения катушки» (см. [Рисунок 2.47](#)).

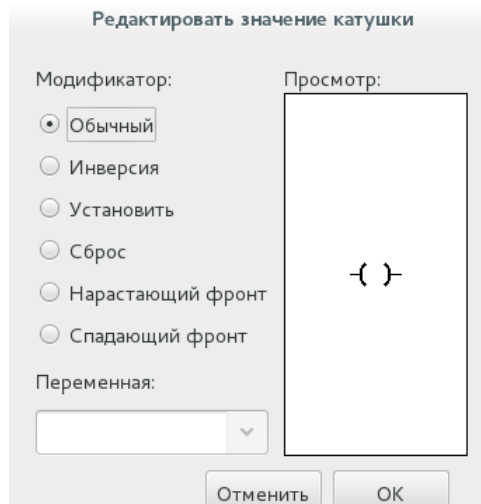


Рисунок 2.47: – Редактирование катушки

В данном диалоге можно определить модификатор данного контакта:

- «Обычный»;
- «Инверсия»;
- «Установить»;
- «Сброс»;
- «Нарастающий фронт»;
- «Спадающий фронт».

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 «Выход» (см. [Рисунок 2.48](#)).

Описание языка LD, основных конструкций и примера его использования приведены в описании языков стандарта МЭК 61131-3.

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 (см. [Рисунок 2.49](#)).

В таблице 8 приведено описание кнопок данной панели. Описание остальных кнопок, относящихся к языку FBD и LD (за исключением катушки) и так же находящихся на этой панели, приведены в таблицах 6 и 7 соответственно.

Таблица 8 - Кнопки панели редактирования LD диаграммы

#	Имя	Класс	Тип
1	Reset	Вход	BOOL
2	Out	Выход	BOOL
3	Cnt	Локальный	INT
4	ResetCour	Внешний	INT

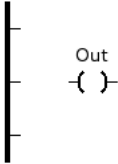








Рисунок 2.48: - Добавление катушки на диаграмму из панели переменных и констант



Рисунок 2.49: - Панель редактирования SFC диаграмм

Внешний вид кнопки	Название кнопки	Функция кнопки
	Создать новый начальный шаг	Вызов диалога редактирования шага
	Создать новый шаг	Вызов диалога редактирования шага
	Создать новый переход	Вызов диалога редактирования перехода
	Создать новый блок действий	Вызов диалога редактирования блока действий
	Создать новое ветвление	Вызов диалога создания новой дивергенции и конвергенции
	Создать новый безусловный переход	Вызов диалога создания «прыжка»

Далее даётся описание добавления приведённых в таблице 8 элементов языка SFC.

Добавление шага инициализации и шага

Процедура добавления шага инициализации и обычного шага ничем не отличается. В обоих случаях вызывается диалог «Редактировать шаг» (см. [Рисунок 2.53](#)).

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 (см. [Рисунок 2.51](#)).

В случае, если при добавлении шага не указано его имя - будет выдана ошибка (см. [Рисунок 2.52](#)).

При добавлении шага появляется диалог, в котором можно указать, с помощью галочек его соединители (см. [Рисунок 2.53](#)):

- «Вход»;

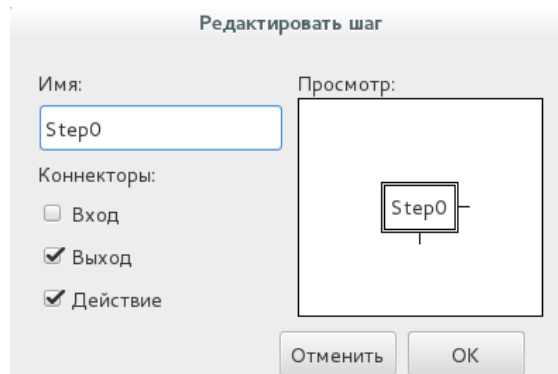


Рисунок 2.50: - Диалог редактирования шага инициализации SFC диаграммы

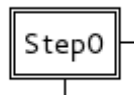


Рисунок 2.51: - Шаг инициализации языка SFC

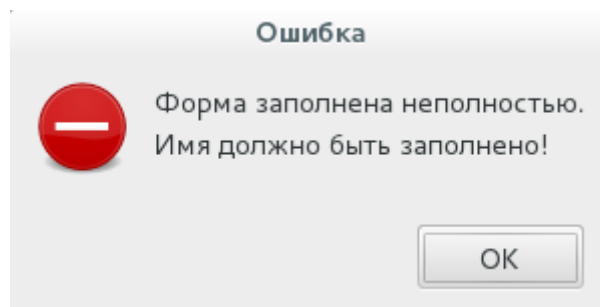


Рисунок 2.52: – Ошибка отсутствия имени шага при его добавлении в диаграмму

- «Выход»;
- «Действие».

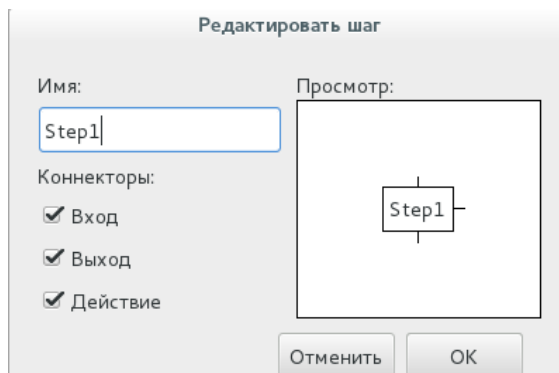


Рисунок 2.53: – Добавление шага SFC на диаграмму

«Действие» добавляет соединитель для связывания данного шага с блоком действий. «Вход» и «Выход» соединители, как правило, соединены с переходом. Соответственно, после нажатия кнопки ОК, на диаграмму будет добавлен шаг с указанными соединителями (см. [Рисунок 2.54](#)).



Рисунок 2.54: - Шаг SFC диаграммы с соединителями входа и действия

Добавление перехода

При добавлении на SFC диаграмму перехода, появится диалог «Редактировать переход» (см. [Рисунок 2.55](#)).

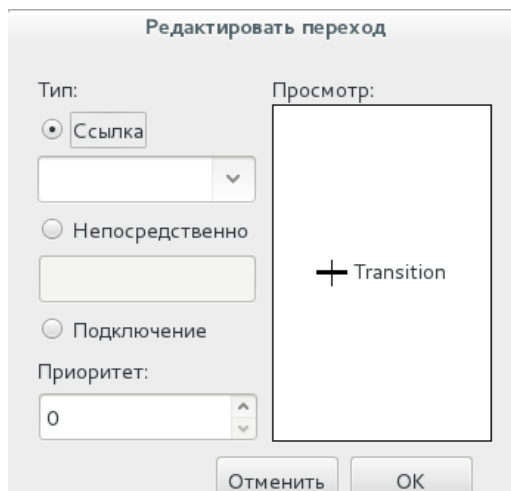


Рисунок 2.55: – Добавление нового перехода в диаграмму

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

- «Ссылка»;

- «Встроенный код»;
- «Соединение».

При выборе типа перехода «Ссылка» в открывающемся списке (см. [Рисунок 2.56](#)) 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.

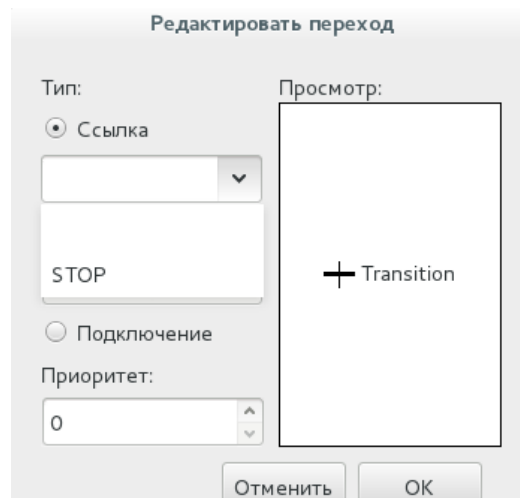


Рисунок 2.56: - Всплывающий список с доступными предопределёнными переходами

При выборе типа перехода «Непосредственно» (см. [Рисунок 2.57](#)), условие перехода можно написать в виде выражения на языке ST.

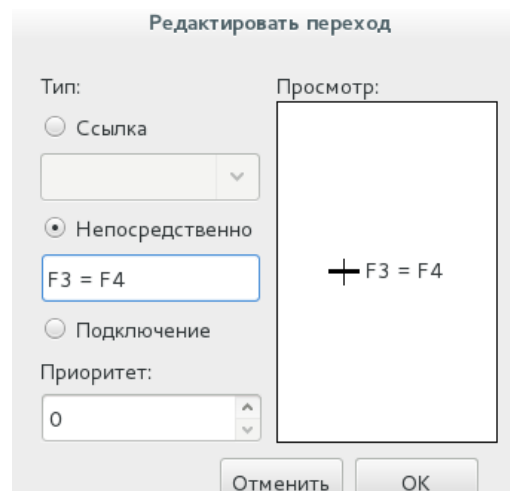


Рисунок 2.57: - Условие перехода в виде встроенного кода, написанного на языке ST

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 (см. [Рисунок 2.57](#)):

F3 = F4

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

При выборе типа перехода «Соединение» (см. [Рисунок 2.58](#)), в качестве условия перехода можно использовать выходные значения элементов языка FBD или LD.

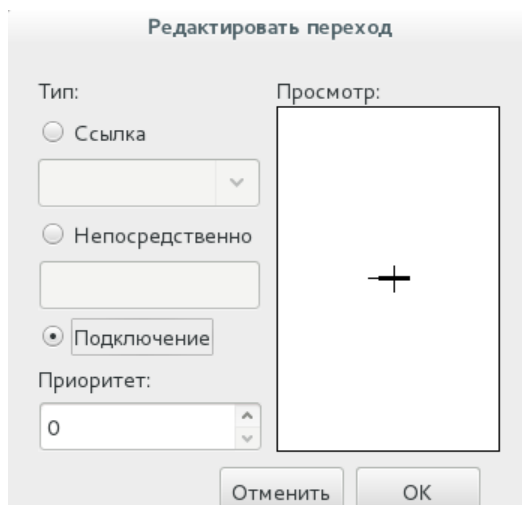


Рисунок 2.58: - Выбор условия перехода как соединение с элементами других графических языков IEC 61131-3

При выборе типа перехода «Подключение», 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. Ниже, на [Рисунок 2.59](#) красным цветом выделен пример перехода, условия которого задано с помощью языка LD.

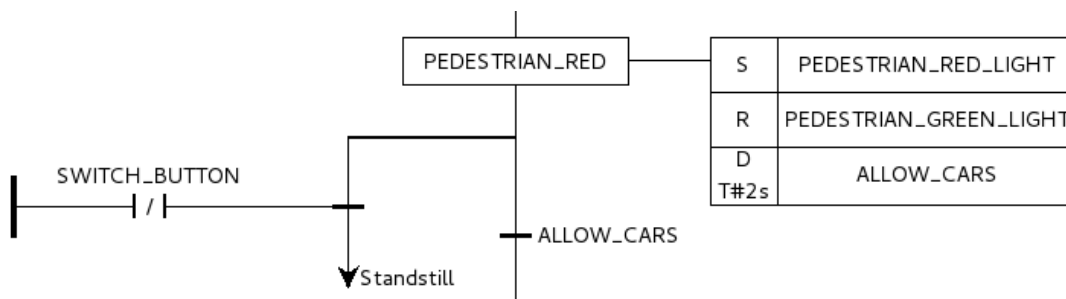


Рисунок 2.59: – Пример SFC диаграммы, в которой один из переходов задан с помощью языка LD

Добавление блока действий

При добавлении блока действий на диаграмму появится диалог «Редактировать свойство блока действий» (см. [Рисунок 2.60](#)).

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

- «Спецификатор»;
- «Длительность»;

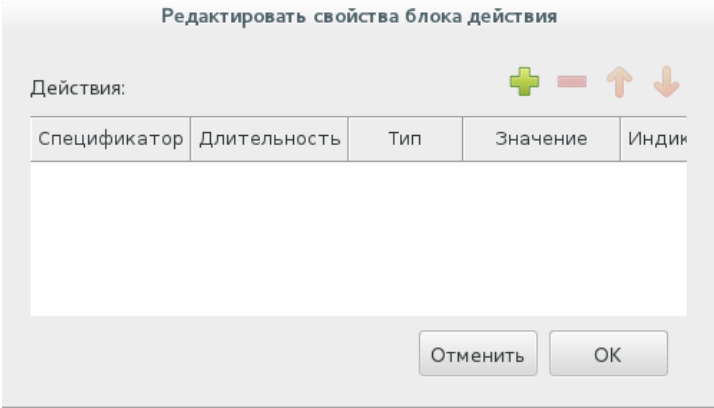


Рисунок 2.60: - Диалог «Редактировать свойство блока действий»

- «Тип»: «Действие», «Переменная», «Непосредственно»;
- «Значение»;
- «Индикатор».

Поле «Спецификатор» определяет момент времени, когда действие начинается, сколько времени продолжается и когда заканчивается. Выбрать квалификатор можно из списка (см. Рисунок 2.61).

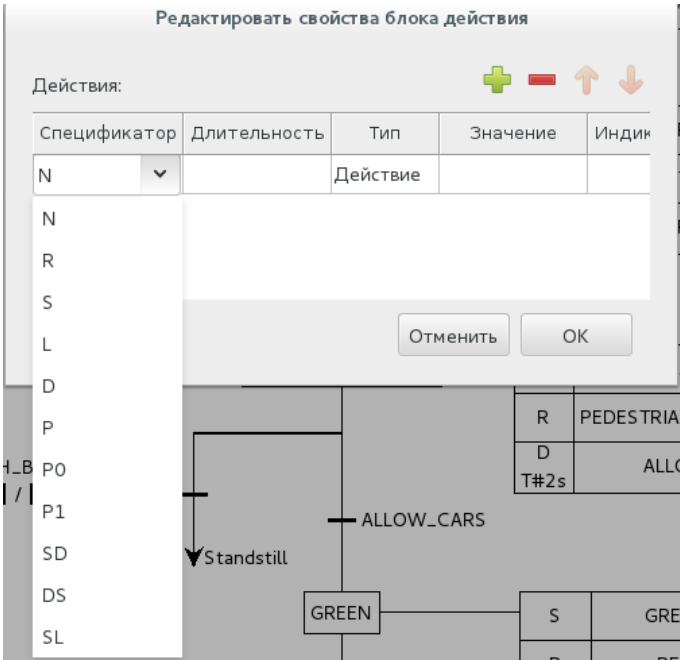


Рисунок 2.61: - Меню выбора спецификатора для действия в диаграмме SFC

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

Таблица 9 - Квалификаторы действий SFC диаграммы

Имя спецификатора	Поведение блока действия
D	Действие начинает выполняться через некоторое заданное время (если шаг еще активен) и выполняется до тех пор, пока данный шаг активен
L	Действие выполняется в течение некоторого заданного интервала времени, после чего выполнение действия останавливается
N	Действие выполняется, пока данный шаг активен
P	Действие выполняется один раз, как только шаг стал активен
P0	Действие выполняется один раз, как только шаг стал неактивен
P1	Действие выполняется один раз, как только шаг стал активен
S	Действие активируется и остается активным пока SFC диаграмма выполняется
R	Действие выполняется, когда диаграмма деактивируется
DS	Действие начинается выполняться через некоторое заданное время, только в том случае если шаг еще активен
SL	Действие активно в течении некоторого, заданного интервала
SD	Действие начинается выполняться через некоторое время, даже в том случае если шаг уже не активен

Поле «Длительность» необходимо для установки интервала времени необходимого для некоторых квалификаторов, описанных выше в таблице 9.

«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 (см. [Рисунок 2.62](#)).

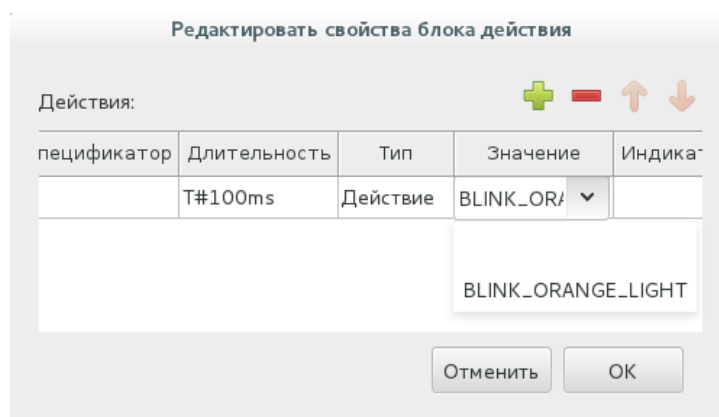


Рисунок 2.62: - Выбор предопределённого действия

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 (см. [Рисунок 2.63](#)), относящиеся к данному программному модулю.

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 «Непосредственно», появляется возможность в поле «Значение» написать на языке ST код, который будет выполняться, когда действие становится активным (см. [Рисунок 2.64](#)).

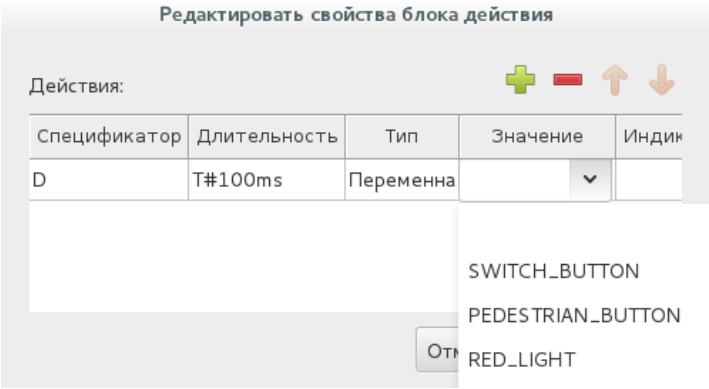


Рисунок 2.63: - Выбор predetermined variable

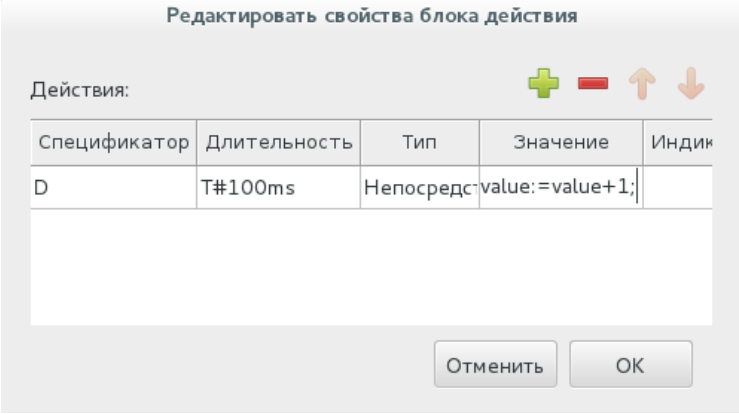


Рисунок 2.64: - Writing built-in code for action

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 (см. [Рисунок 2.65](#)).

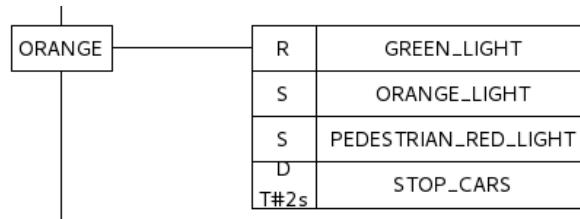


Рисунок 2.65: - Ассоциирование шага ORANGE блоком действия, содержащим четыре действия

Добавление дивергенции и конвергенции

При добавлении ветвления, появится диалог «Создать новое ветвление» (см. [Рисунок 2.66](#)).

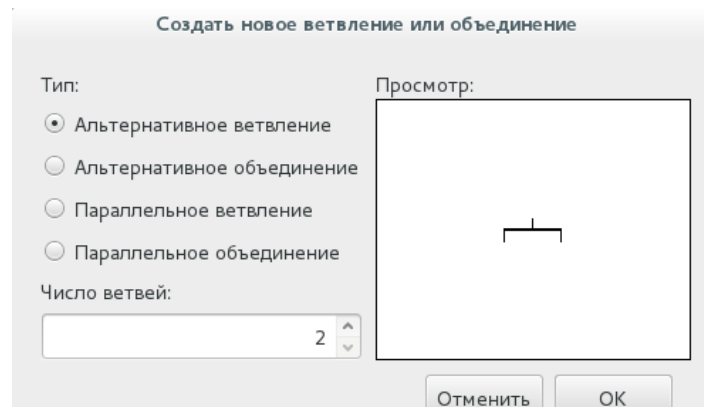


Рисунок 2.66: – Добавление альтернативного ветвления(дивергенции)

В первую очередь следует выбрать тип ветвления:

- «Альтернативное ветвление»;
- «Альтернативное объединения»;
- «Параллельное ветвление»;
- «Параллельное объединение».

Вторым параметром является количество разветвлений, которое определяет на сколько ветвей будет либо расходится (в случае выбора типа дивергенции «Альтернативное ветвление» или «Параллельное ветвление») одна ветвь, либо сколько ветвей будет сходиться в одну ветвь (в случае выбора типа дивергенции «Альтернативное объединения» или «Параллельное объединение»)

Пример дивергенции с двумя разветвлениями показан на [Рисунок 2.67](#) и выделен красным цветом.

Пример конвергенции выделен красным цветом на [Рисунок 2.68](#) .

Пример параллельного ветвления показан на [Рисунок 2.69](#) и выделен красным цветом.

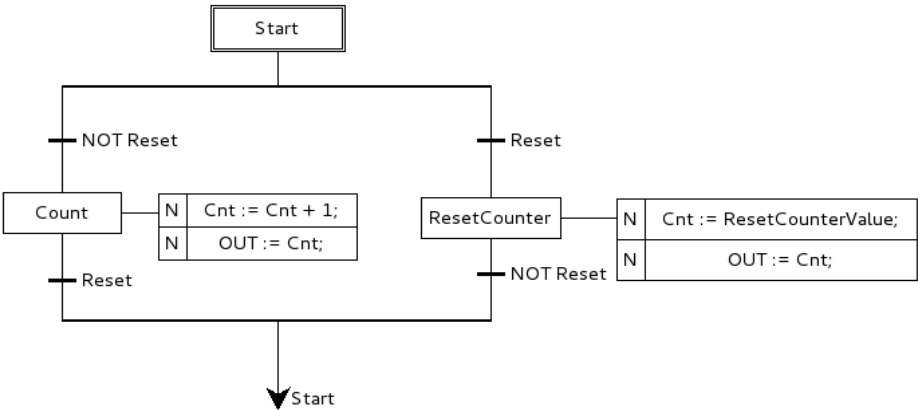


Рисунок 2.67: – Пример SFC диаграммы, содержащей альтернативное ветвление

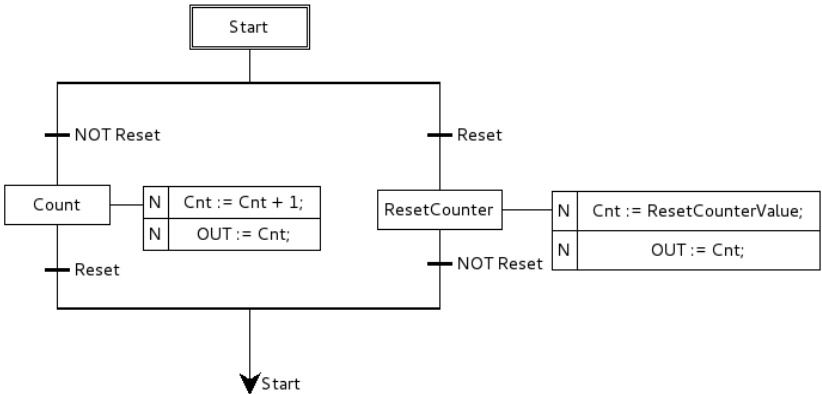


Рисунок 2.68: – Пример SFC диаграммы, содержащей альтернативное объединение

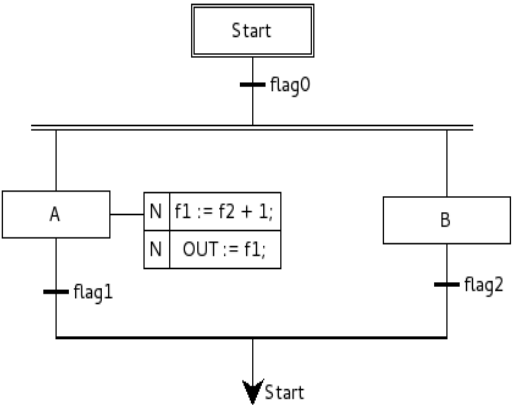


Рисунок 2.69: – Пример SFC диаграммы с параллельным ветвлением

Согласно стандарту IEC 61131-3, каждая ветвь альтернативного ветвления должна оканчиваться переходом, при альтернативном объединении переход должен быть перед каждой ветвью. При параллельном ветвлении переход должен быть перед ветвлением, а при параллельном объединении переход необходим после объединения.

Добавление безусловного перехода

Элемент «безусловный переход» 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 (см. [Рисунок 2.70](#)), в котором необходимо выбрать из списка шаг, к которому будет происходить «прыжок» - переход от одного шага SFC диаграммы к другому.

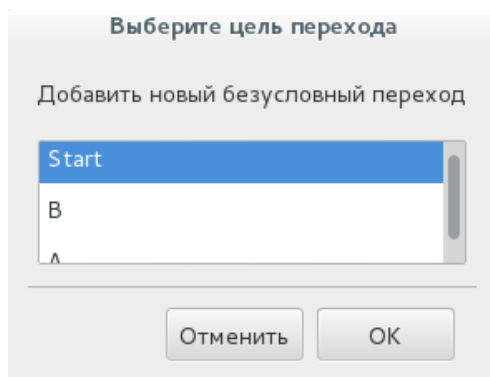


Рисунок 2.70: – Диалог добавления безусловного перехода

В данном диалоге также присутствует и шаг инициализации (начальный шаг). После выбора шага и нажатия кнопки ОК. На SFC диаграмме появится стрелочка, которую нужно соединить с переходом (см. [Рисунок 2.71](#)). Справа от стрелочки находится имя шага, к которому осуществляется переход в случае выполнения условия перехода, находящегося выше и соединённого с ней.

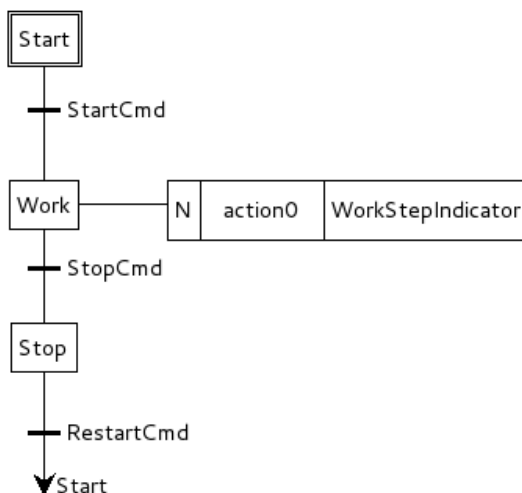


Рисунок 2.71: – Безусловный переход с шага Stop на начальный шаг Start

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

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 правой клавишей мыши и выбором «Добавить переход» (см. [Рисунок 2.72](#)).

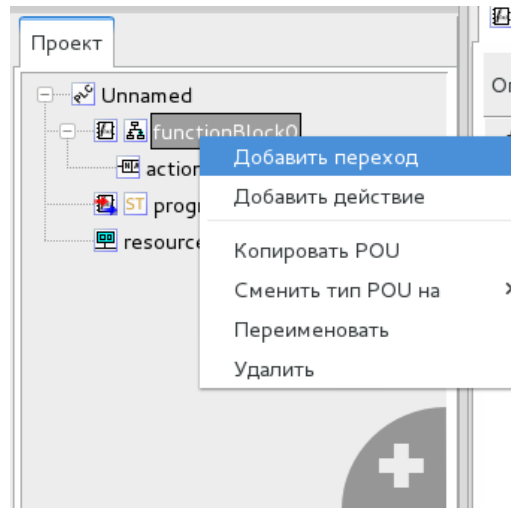


Рисунок 2.72: - Всплывающее меню SFC диаграммы в панели проекта

Далее появится диалог под названием «Создать новый переход» (см.:numref:image111). В нём необходимо выбрать уникальное имя перехода и язык, в котором будет описано данное условие.

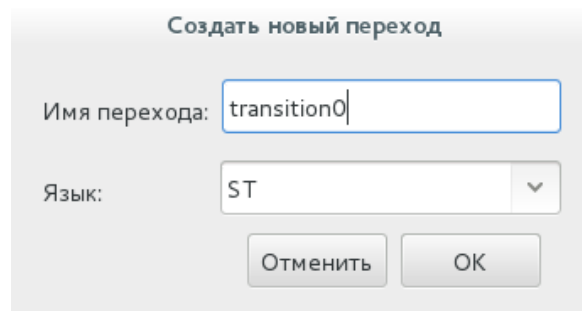


Рисунок 2.73: - Диалог «Создать новый переход»

В случае, если переходы с введённым именем уже существуют, то будет выведено сообщение об ошибке (см. [Рисунок 2.74](#)).

Добавление действия в структуру SFC диаграммы (см. [Рисунок 2.75](#)) происходит аналогично добавлению перехода в данную структуру.

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» (см. [Рисунок 2.76](#)).

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 (см. [Рисунок 2.77](#)).

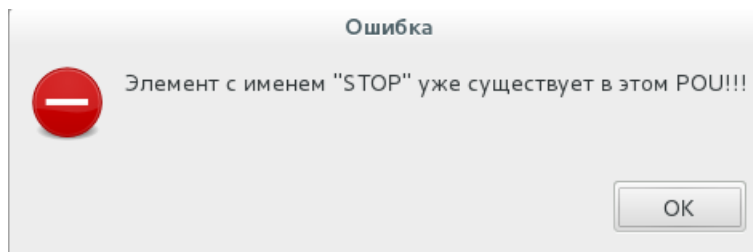


Рисунок 2.74: - Сообщение об ошибке добавления существующего программного модуля

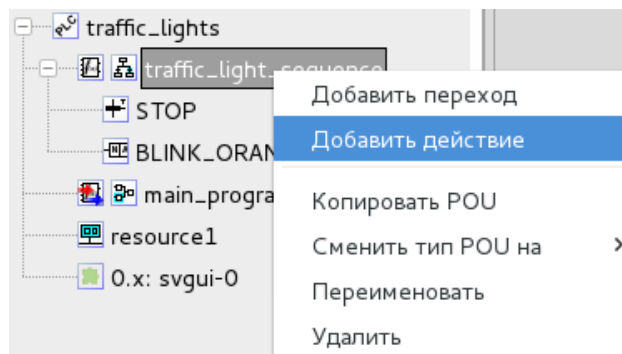


Рисунок 2.75: - Всплывающее меню SFC для структуры диаграммы

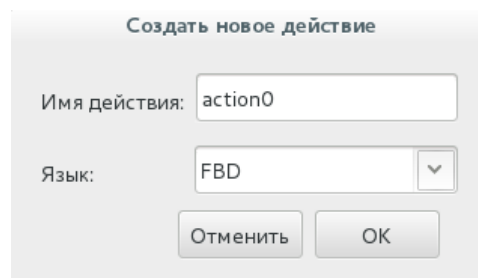


Рисунок 2.76: - Диалог «Создать новое действие»

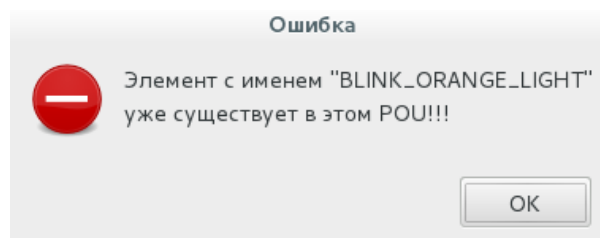


Рисунок 2.77: - Ошибка не заполнения имени действия при его добавлении

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» (см. [Рисунок 2.76](#)). 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.

Панель редактирования ресурса

Панель редактирования ресурса (см. [Рисунок 2.78](#)) содержит панель переменных и констант, которая позволяет определять глобальные переменные на уровне ресурса и панели, содержащие задачи и экземпляры.

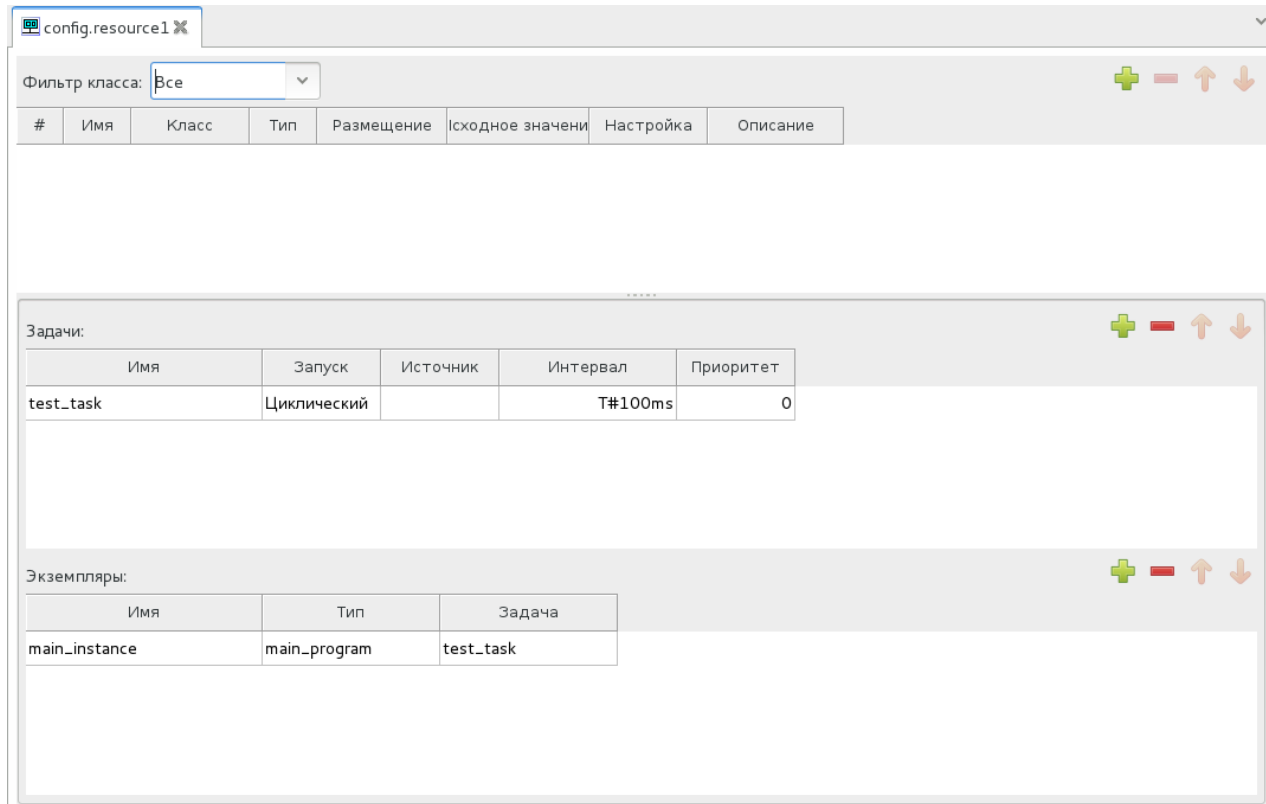


Рисунок 2.78: - Вкладка ресурс главной рабочей области

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».

Панель редактирования типа данных

Панель редактирования типа данных (см. [Рисунок 2.79](#)) позволяет определить различные параметры создаваемого пользовательского типа данных.

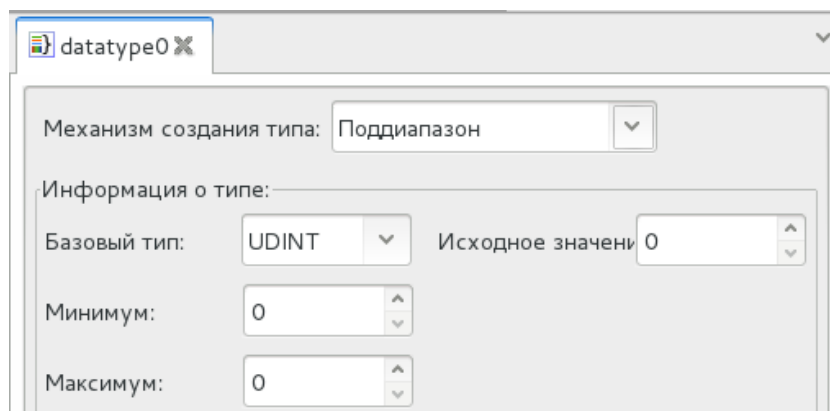


Рисунок 2.79: - Вкладка создания нового типа данных

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

- Синоним;
- Поддиапазон существующего типа (выделение диапазона значений стандартного типа);
- Перечисление (перечисляемый тип);
- Массив;
- Структура, позволяющая определять тип, основанный на объединении несколько типов.

Далее рассмотрены подробнее параметры для каждого из вышеперечисленных типов.

Синоним

При выборе «Синоним» (см. [Рисунок 2.80](#)), из списка указывается базовый тип и его начальное значение.

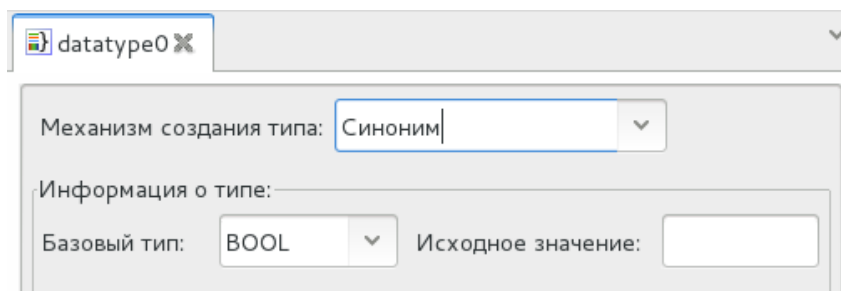


Рисунок 2.80: - Добавление псевдонима типа данных

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

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» (см. [Рисунок 2.81](#)), т.е. соответственно минимального и максимального значения, которое может принимать создаваемый тип данных.

Перечисляемый тип

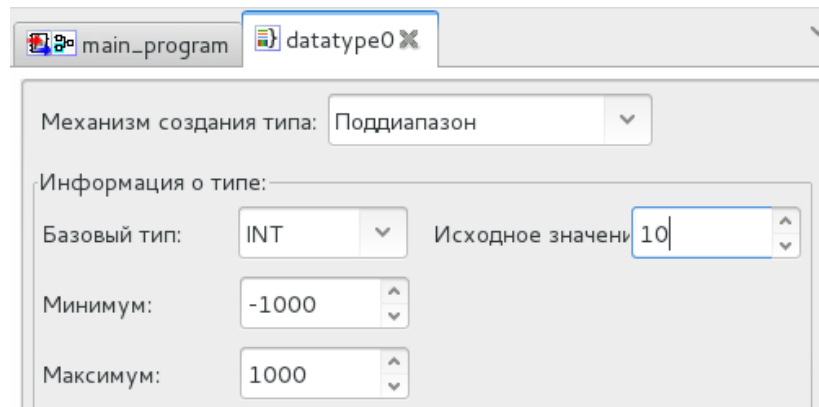


Рисунок 2.81: - Добавление нового типа данных, представляющего поддиапазон существующего типа

При выборе механизма создания нового типа «Перечисляемый тип» (см. [Рисунок 2.82](#)), появится панель, содержащая таблицу, в которой можно задать список возможных значений данного перечисляемого типа.

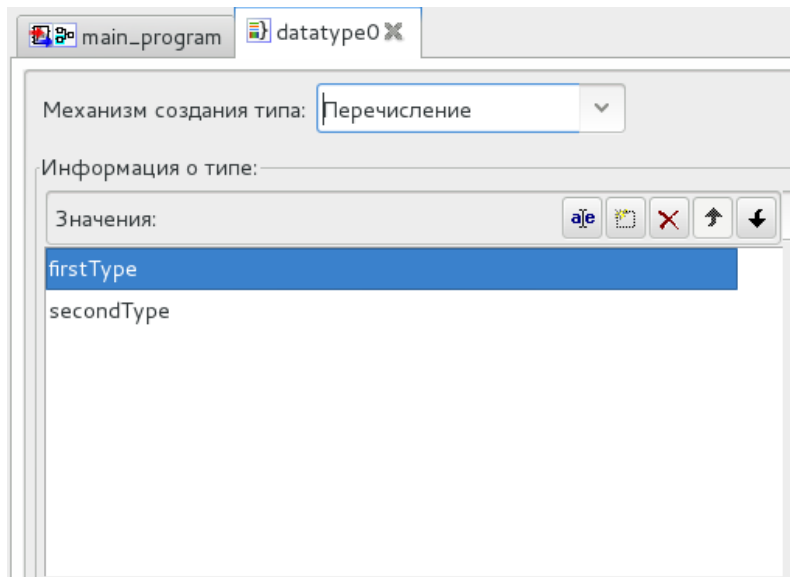






Рисунок 2.82: – Добавление перечисляемого типа данных

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

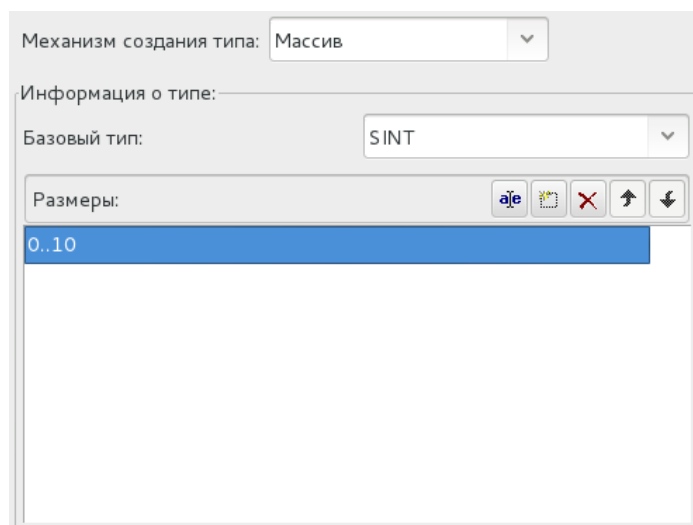
Таблица 10 - Кнопки редактирования значений перечисляемого типа

Внешний вид кнопки	Название кнопки	Функция кнопки
	Редактировать	Редактировать выделенное поле в таблице
	Добавить	Добавить новое поле в таблицу
	Удалить	Удалить выделенное поле в таблице
	Переместить вверх	Переместить вверх выделенное поле в таблице
	Переместить вниз	Переместить вниз выделенное поле в таблице

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

Массив

При выборе механизма создания нового типа «Массив» (см. [Рисунок 2.83](#)) появится панель, в которой необходимо указать базовый тип, начальное значение, а также размерность массива.



Механизм создания типа: Массив

Информация о типе:

Базовый тип: SINT

Размеры: 0..10

Рисунок 2.83: - Добавление типа данных - массива

Размерность массива задаётся в следующем формате: <начальное значение>..<конечное значение>

Структура

При выборе механизма создания нового типа «Структура» (см. [Рисунок 2.84](#)), в появившейся таблице необходимо добавить необходимое количество полей структуры. Каждое поле имеет своё имя, тип и начальное значение.

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 ИЕС 61131-3. Далее будет продолжено рассмотрение остальных компонент среды разработки Beremiz.

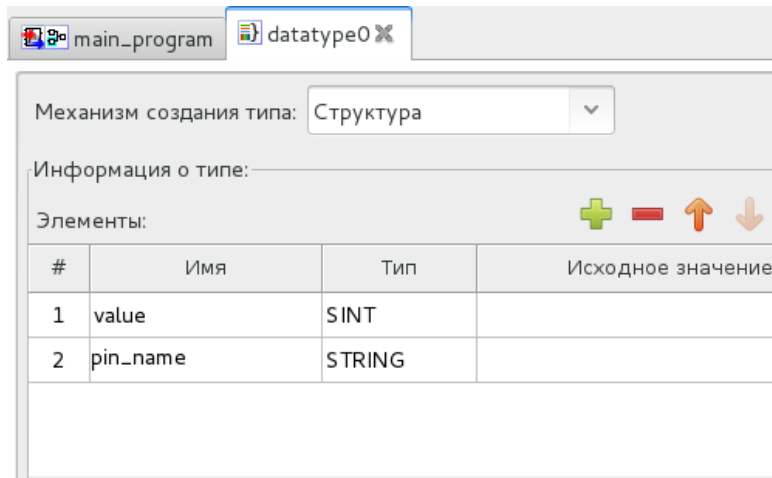


Рисунок 2.84: – Добавление типа данных – структура

Панель экземпляров проекта

Панель экземпляров проекта (см. [Рисунок 2.85](#)) обычно располагается слева в среде разработки Beremiz и отображаемые в ней экземпляры зависят от выбранного элемента в дереве проекта.

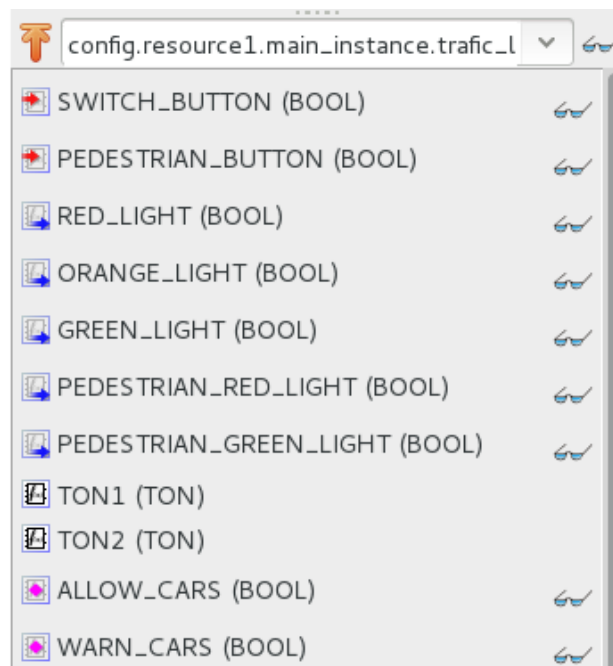


Рисунок 2.85: – Панель экземпляров проекта

При выборе в дереве проекта элемента, соответствующего ресурсу, в панели экземпляров проекта будут отображены экземпляры, определённые в данном ресурсе, а так же глобальные переменные ресурса. На [Рисунок 2.86](#) показано, как в панели редактирования ресурса определен экземпляр для программного модуля «main_program»:

Соответственно, при выборе в дереве проекта ресурса, в котором определены экземпляры (описанные выше) и глобальная переменная, панель экземпляров будет выглядеть, как показано на [Рисунок 2.87](#)

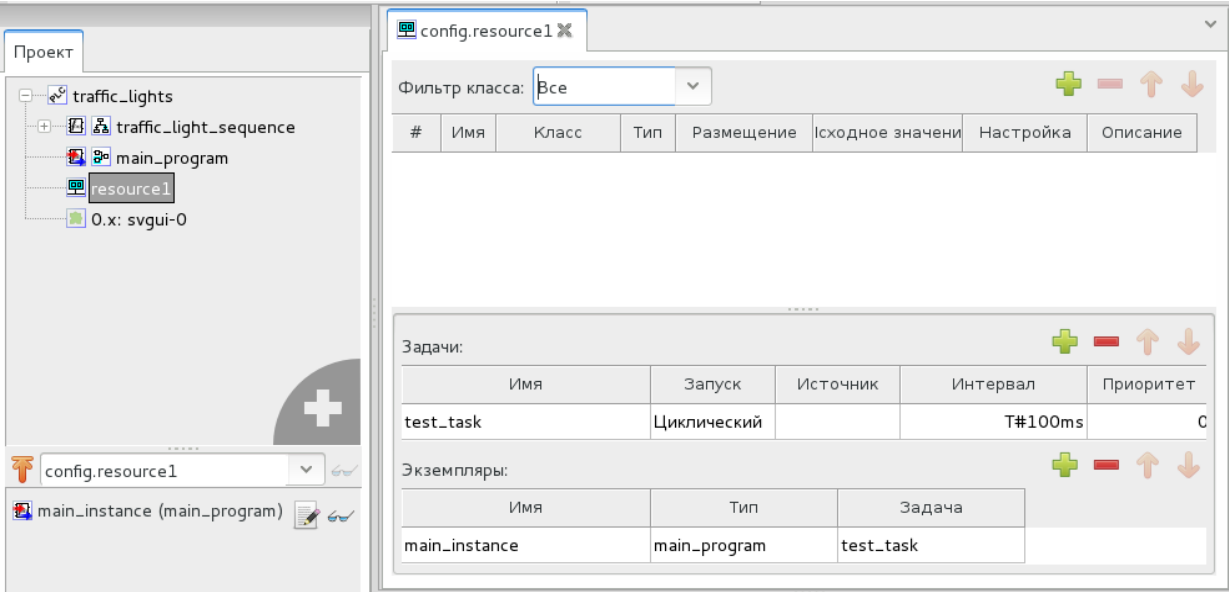


Рисунок 2.86: – Экземпляр main_instance для программного модуля main_program

:

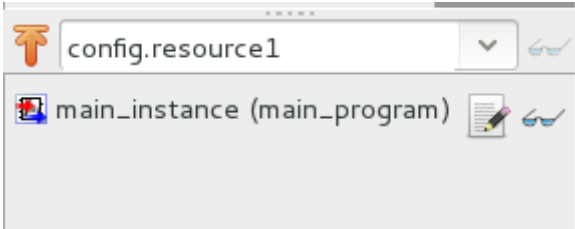


Рисунок 2.87: - Панель экземпляров при выборе элемента ресурса в дереве проекта

При выборе в дереве проекта элемента, соответствующего программным модулям «Программа» и «Функциональный блок» в панели экземпляров будут отображены переменные, определённые в них. Ниже на [Рисунок 2.88](#) приведён пример программного модуля типа «Программа» с именем «program0», в котором определено 8 переменных различных пользовательских классов.

Соответственно, при выборе в дереве проекта данного программного модуля в панели экземпляров будут отображены, определённые выше переменные (см. [Рисунок 2.89](#)).

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 - Кнопки на панели экземпляров проекта

main_program x

Описание: Фильтр класса: Все + - ↑ ↓

#	Имя	Класс	Тип	Размещение	Исходное значени	Настройка	Описание
1	traffic_light_se	Локальный	traffic_light_se				
2	SwitchButton	Локальный	Button				
3	PedestrianButt	Локальный	Button				
4	RedLight	Локальный	Led				
5	OrangeLight	Локальный	Led				
6	GreenLight	Локальный	Led				
7	PedestrianRed	Локальный	Led				
8	PedestrianGre	Локальный	Led				

Рисунок 2.88: - Программный модуль типа «Программа»

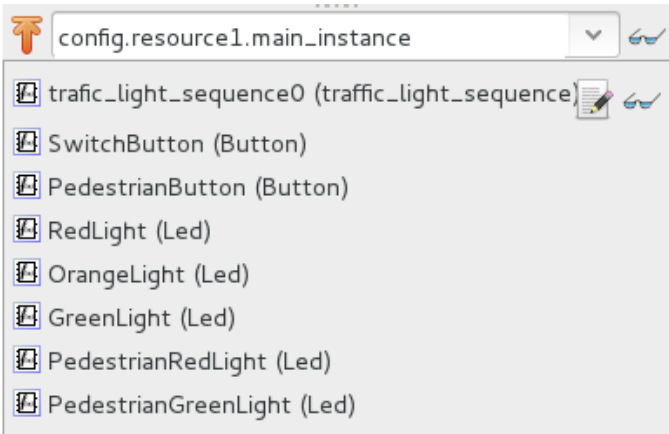





Рисунок 2.89: - Панель экземпляров проекта при выборе в дереве проекта программного модуля main_program

Внешний вид кнопки	Название кнопки	Функция кнопки
	Отладка экземпляра	Кнопка запуска режима отладки для экземпляра
	Двойной щелчок по кнопке «Отладка экземпляра»	Отображение графика изменения значения переменной в режиме отладки
	Родительский экземпляр	Переход к родительскому экземпляру и его локальным переменным

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. Если кнопка запуска режима отладки нажимается для элемента переменной, то переменная будет добавлена в панель отладки.

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

Панель библиотеки функций и функциональных блоков

Панель библиотеки функций и функциональных блоков (см. [Рисунок 2.90](#)), 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.

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

Помимо стандартных функций и функциональных блоков, данная панель содержит раздел «пользовательские программные модули». В него попадают функции и функциональные блоки, добавленные в конкретный проект, т. е. содержащиеся в дереве проекта.

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

Отладочная консоль

Панель, содержащая отладочную консоль (см. [Рисунок 2.91](#)), как правило, располагается в правом нижнем углу среды разработки Beremiz.

Она служит для отображения в виде текстовых сообщений:

- Результаты генерации ST и C кода;
- Результаты компиляции и компоновки прикладной программы;
- Процесса соединения и передачи прикладной программы на целевое устройство;
- Различных промежуточных манипуляций в процессы создания прикладной программы.

В случае, если необходимо вывести предупреждения среды разработки Beremiz или ошибки компиляторов (MatIEC или C кода) во время их работы цвет вывода текстовых сообщений становится красным.

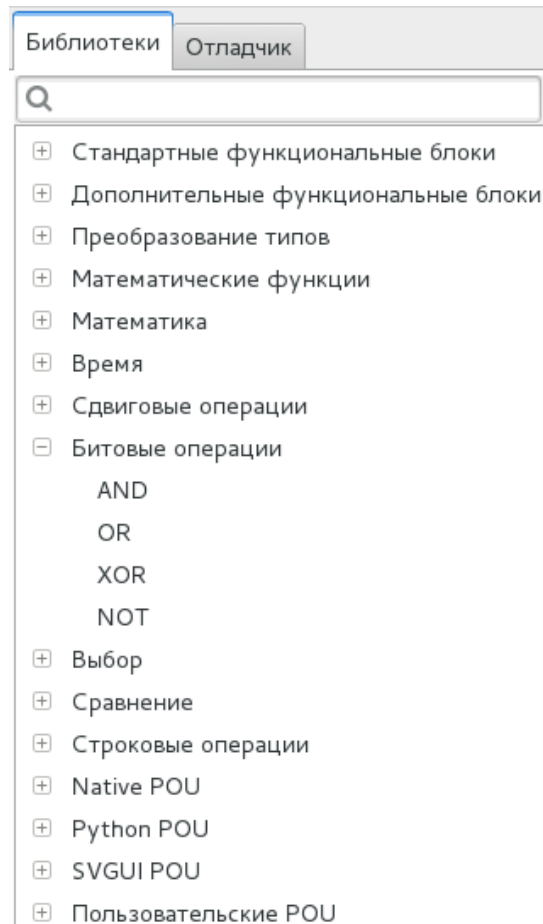


Рисунок 2.90: – Панель библиотеки функций и функциональных блоков

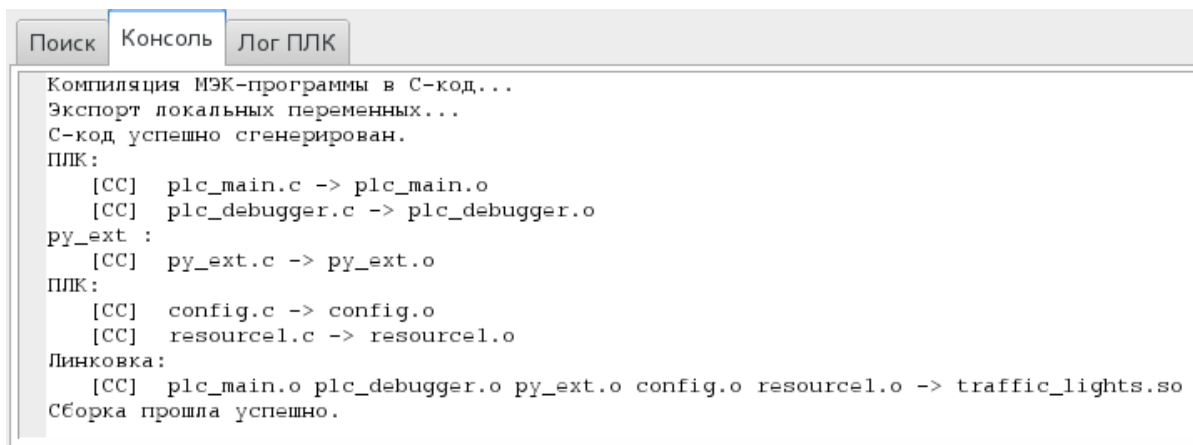


Рисунок 2.91: – Успешная сборка в отладочной консоли

Критические ошибки также выделяется красным цветом, но при этом еще желтым фоном (см. Рисунок 2.92).

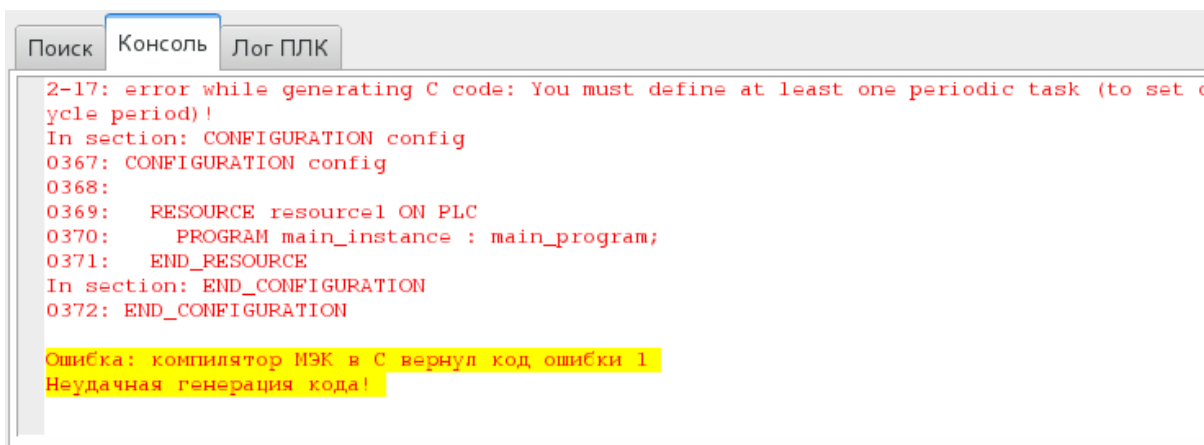


Рисунок 2.92: - Ошибка сборки проекта в отладочной консоли

Поиск элементов в проекте

Для поиска интересующего элемента в проекте используется диалог «Поиск в проекте» (см. Рисунок 2.93). Его вызов происходит с помощью главного меню программы или панели инструментов.

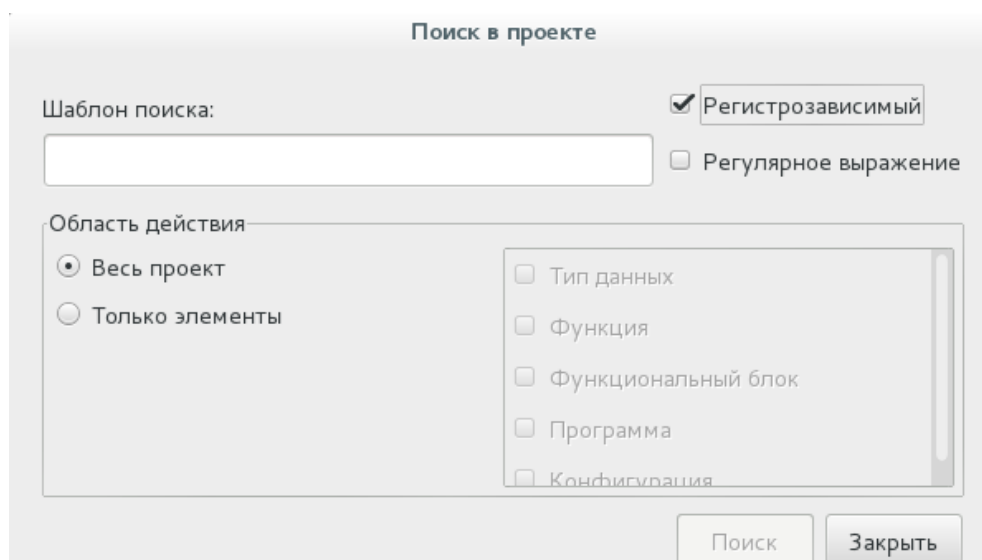


Рисунок 2.93: – Диалог поиска в проекте

В появившемся диалоге можно установить различные параметры поиска: шаблон поиска, область поиска, чувствительность к регистру при поиске, а так же записать шаблон поиска в виде регулярного выражения. После того как все параметры установлены, необходимо нажать кнопку «Поиск» в этом диалоге. Ниже на Рисунок 2.94 приведен пример поиска элемента с именем «LIGHT».

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

traffic_light_sequence

Описание:

Фильтр класса: Все

#	Имя	Класс	Тип	Исходное значени	Настройка	Описание
4	ORANGE_LIGHT	Выход	BOOL			
5	GREEN_LIGHT	Выход	BOOL			
6	PEDESTRIAN_RED_LIG	Выход	BOOL			

used to

Standstill

P

ORANGE_LIGHT := 1;

N

BLINK_ORANGE_LIGHT

R

PEDESTRIAN_RED_LIGHT

R

PEDESTRIAN_GREEN_LIGHT

R

RED_LIGHT

R

GREEN_LIGHT

SWITCH_BUTTON

ORANGE

R

GREEN_LIGHT

S

ORANGE_LIGHT

S

PEDESTRIAN_RED_LIGHT

D

T#2s

STOP_CARS

Поиск

Консоль

Лог ПЛК

'LIGHT' - 50 совпадений в проекте

Проект 'traffic_lights':

traffic_light_sequence (29)

name: traffic_light_sequen

name: RED_LIG

name: ORANGE_LIG

name: GREEN_LIG

name: PEDESTRIAN_RED_LIG

name: PEDESTRIAN_GREEN_LIG

Рисунок 2.94: – Результаты поиска элемента с именем LIGHT

Панель отладки

Панель отладки располагается в правой части среды разработки Beremiz (см. [Рисунок 2.95](#)).

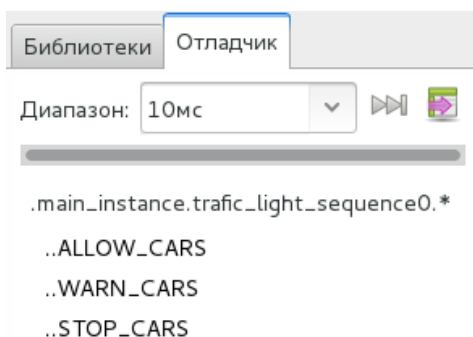


Рисунок 2.95: – Панель отладки

Данная панель представляет собой таблицу с двумя столбцами «Переменная» и «Значение». Соответственно, столбец «Переменная» содержит экземпляры переменных, значения которых во время исполнения, отображаются в поле «Значение» и могут изменяться. Добавление переменных осуществляется с помощью панели экземпляров проекта.

Изменение значений переменной во время отладки прикладной программы осуществляется нажатием левой клавишей мыши на иконку замка напротив интересующей переменной (см. [Рисунок 2.96](#)).

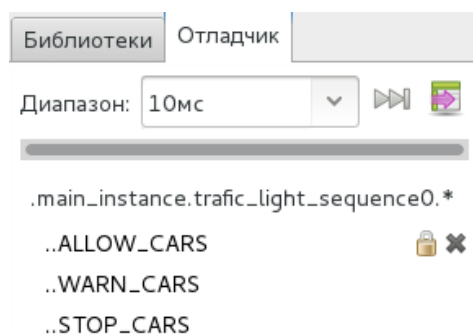


Рисунок 2.96: – Установка значения переменной

Далее появится диалог ввода значения для выбранной переменной (см. [Рисунок 2.97](#)).

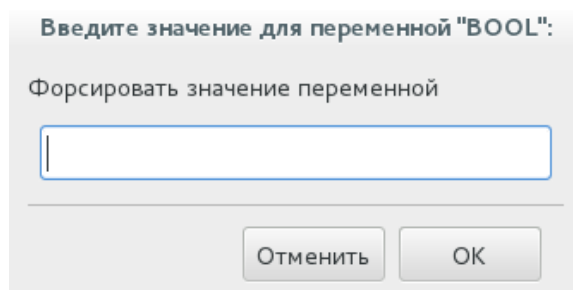


Рисунок 2.97: – Диалог установки значения для переменной

В режиме отладки форсированное значение переменной будет подсвечено синим цветом. Для того чтобы освободить значение переменной, необходимо нажать на иконку открытого замка (см. [Рисунок 2.98](#))

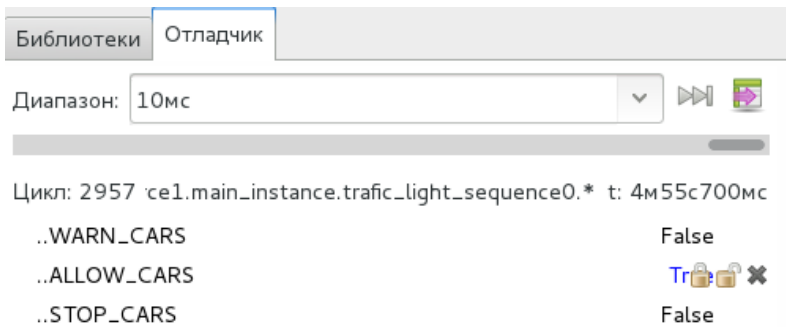


Рисунок 2.98: – Освобождение значения переменной

На данной панели присутствует кнопка удаления переменной из панели отладчика, перемещать и передавать ссылку на переменные можно в Drag&Drop режиме.

Панель графика изменения значения переменной в режиме отладки

Данная панель (см. [Рисунок 2.99](#)) открывается во вкладке отладчика напротив выбранной переменной в случае, если в панели отладчика нажать на переменную двойным щелчком мыши. Напротив переменной появляется график изменения значения переменной в режиме отладки (см. таблицу 11).

На данной панели есть возможность 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 - Кнопки на панели графика изменения значения переменной

Внешний вид кнопки	Функция кнопки
	Очистка отображения графика
	Переход к отображению текущего значения графика, т.е. сдвиг параметра «Позиция» максимально вправо
	Сброс настроек масштаба до настроек по умолчанию: x 1.0

2.2.4 Работа с проектом

Данный раздел отражает основные приемы работы в среде разработки Beremiz, необходимые при создании прикладной программы. Прикладная программа для целевой платформы является результатом сборки проекта с определенной конфигурацией.

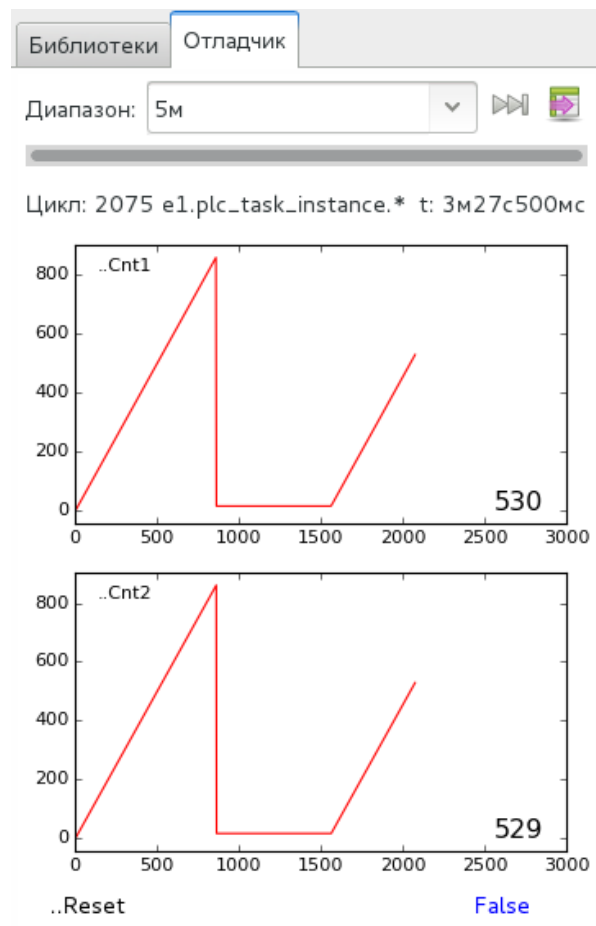


Рисунок 2.99: – Графики изменения значений счетчиков Cnt

Проект в Beremiz представляет собой именованную папку, в которой лежат исходные файлы. 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;
- папка «build», которая хранит генерируемый ST и C код, а также получаемый исполняемый бинарный файл прошивки.

Создание нового проекта

Новый проект создаётся с помощью главного меню «Файл» – «Новый» (см. Рисунок 2.100), либо с помощью кнопки «Новый» на панели управления.

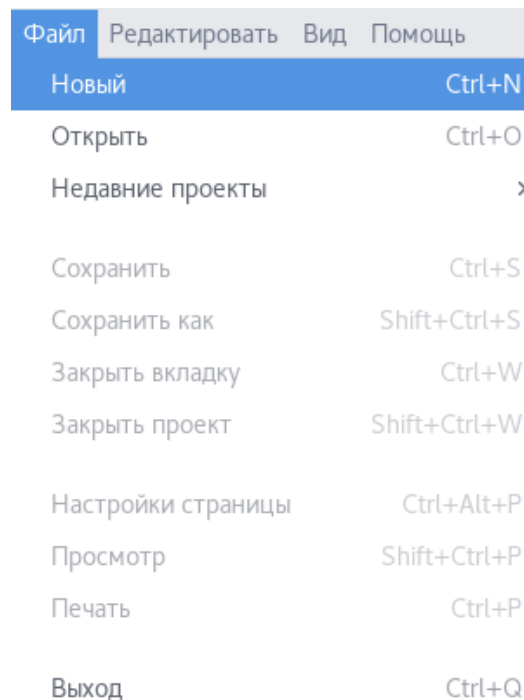


Рисунок 2.100: - Создание нового проекта с помощью главного меню

Далее появится диалог (см. Рисунок 2.101), в котором необходимо выбрать папку, где будет храниться данный проект.

В появившемся диалоге вам будет предложено настроить основной программный модуль проекта (см. Рисунок 2.102). В данном диалоге три поля:

- «Имя ROU»;
- «Тип ROU»;
- «Язык».

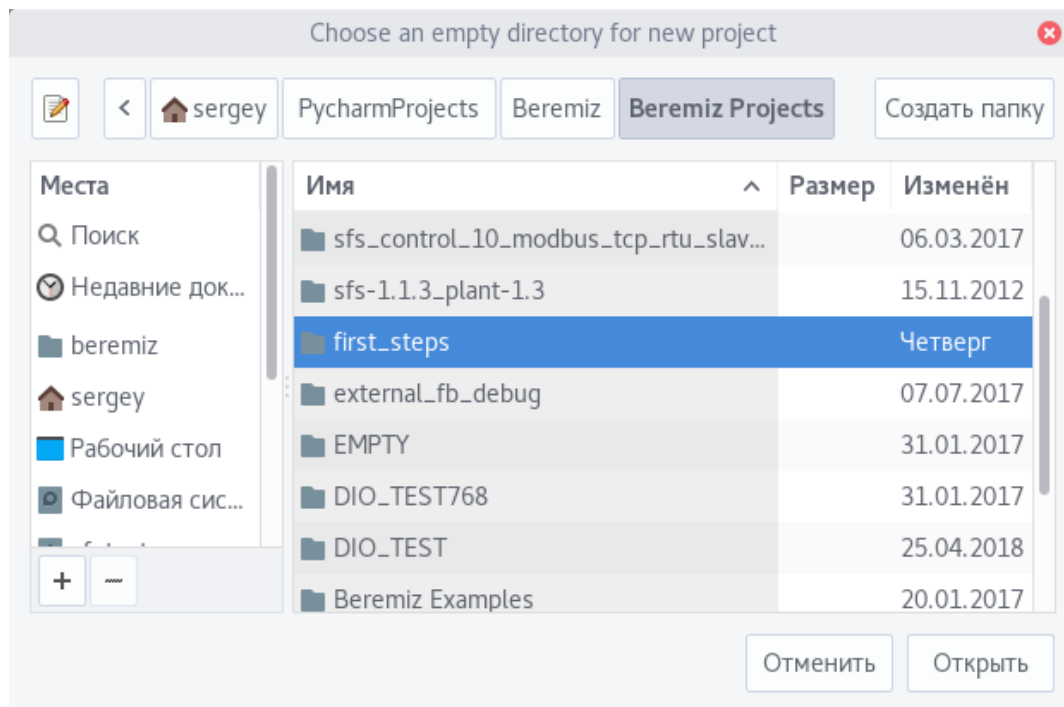


Рисунок 2.101: - Диалог выбора папки для нового проекта

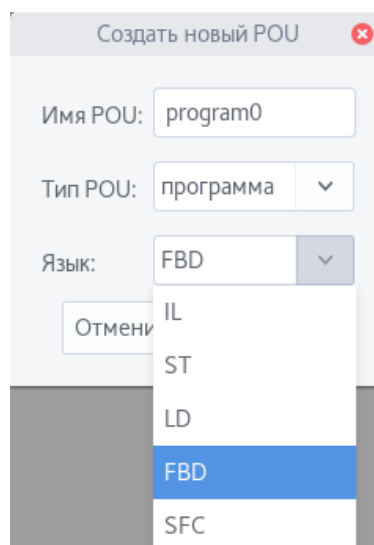


Рисунок 2.102: - Диалог добавления основного программного модуля

Имя программного модуля, присвоенное по умолчанию, может быть заменено на любое имя, соответствующее назначению данного программного модуля.

Тип основного программного модуля – «Программа», в дальнейшем в проект можно добавить дополнительные программные модули, функции и функциональные блоки.

В поле «Язык» необходимо выбрать из списка один из языков стандарта IEC 61131-3 (IL, ST, LD, FBD, SFC), на котором будут реализованы алгоритмы и логика работы данного добавляемого программного модуля.

При нажатии кнопки ОК в проект будет добавлен основной программный модуль с выбранными параметрами, ресурс проекта будет конфигурирован по умолчанию: добавлена одна задача циклического выполнения с интервалом 20 мс, и один экземпляр основной программы. При нажатии кнопки Отмена будет создан пустой проект без каких-либо настроек.

В рамках описания процесса создания нового проекта за основу выбран проект «First steps» из стандартного набора тестовых проектов IDE Beremiz. Основной программный модуль в этом проекте написан на языке FBD, соответственно, в диалоге необходимо выбрать язык FBD, в дальнейшем язык основного программного модуля возможно изменить.

Настройка проекта

Следующим шагом после создания проекта является его настройка, включающая в себя задание глобальных переменных, установку параметров компиляции и компоновки, и 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» (см. Рисунок 2.103).

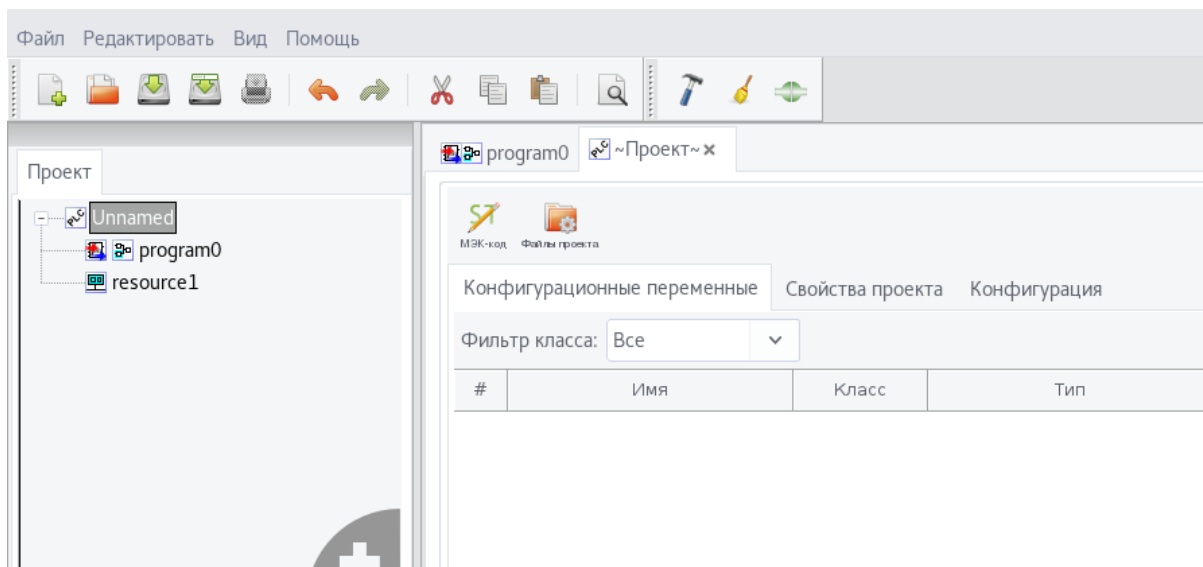


Рисунок 2.103: - Панель настройки проекта

На панели присутствуют три вкладки:

- Конфигурационные переменные;
- Свойства проекта;

- Конфигурация.

Конфигурационные переменные проекта

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

Ниже, на [Рисунок 2.104](#), в панели переменных и констант добавим конфигурационную константу «ResetCounterValue» типа INT с начальным значением 17, с помощью кнопки «Добавить переменную» (см. таблицу 3).

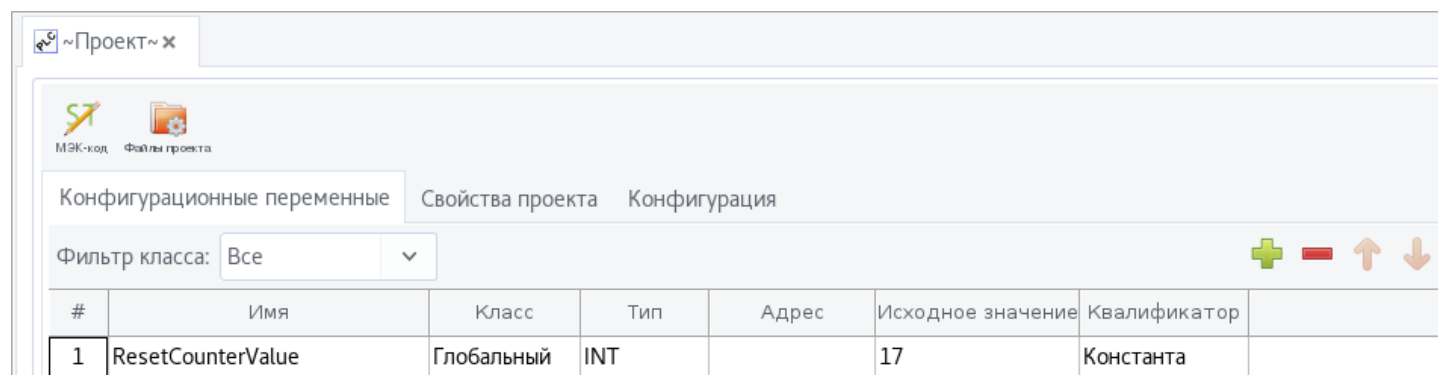


Рисунок 2.104: - Объявление конфигурационной переменной

Для того чтобы к данной конфигурационной переменной можно было обращаться из программных модулей типа «Программа» или «Функциональный блок» необходимо в их панели редактирования в панели переменных и констант создать переменную с таким же именем, как и ранее объявленная глобальная, и установить её класс «Внешний».

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 системы исполнения» - адрес целевого устройства, и целевая платформа - архитектура платформы целевого устройства (см. [Рисунок 2.105](#)).

Как правило, «URI системы исполнения» указывается в формате:

<Тип коннектора>://<Адрес последовательного порта подключения>:<битрейт>

Тип коннектора выбирается в зависимости от типа сервиса подключения к целевому устройству. Например, для отладки прикладной программы на локальной машине при помощи Soft PLC, целевым устройством является служба «Beremiz service» и тип коннектора следует выбрать «LOCAL». Для отладки прикладной программы вне локальной машины используется библиотека PYRO, в это случае «URI целевого устройства» указывается в формате:

PYRO://<IP-адрес целевого устройства>:<порт>

Если в проекте используются дополнительные библиотеки, их следует добавить в конфигурации проекта, нажав «checkbox» напротив добавляемой библиотеки в подменю «Библиотеки».

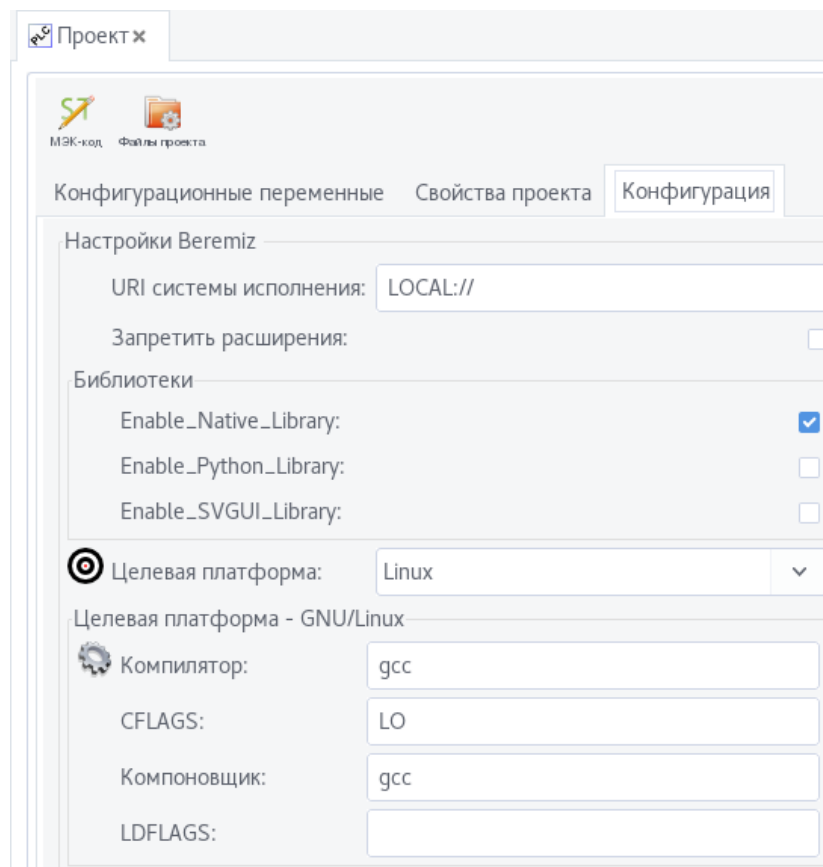


Рисунок 2.105: - Конфигурация проекта

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 (см.:numref:image152), позволяющую удобным образом различать проекты.

Рисунок 2.106: - Заполнение данных о проекте

Большая часть данных в информации проекте являются необязательным для заполнения, но обязательные должны быть заполнены, такие поля помечены 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.

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».

Проект «First steps» представляет собой основной программный модуль, написанный на языке FBD, в котором используются 5 функциональных блоков, написанных на пяти разных языках IEC 61131-3. Каждый функциональный блок это счетчик, увеличивающий значение выхода на единицу до тех пор, пока на входе Reset не будет установлено значение True. Инкрементация значения происходит в каждом цикле основной программы. Регулировать интервал цикла можно изменяя длительность задачи для экземпляра основной программы в панели ресурсов.

В созданный проект необходимо добавить программу program0, функцию и 5 функциональных блоков: CounterST, CounterLD, CounterFBD, CounterSFC, CounterIL. Если при создании проекта основной

программный модуль program0 не был добавлен, его следует добавить вручную. Далее рассмотрено добавление каждого программного модуля в отдельности.

Программа

Ниже будет приведён пример добавления в проект программы, написанной на языке FBD. Логика и алгоритм работы данного программного модуля следующие: определена переменная Reset типа BOOL, отвечающая за сброс каждого из пяти счетчиков, определены пять переменных Cnt1..Cnt5 типа INT, в них хранится значение каждого из пяти счетчиков, и добавлены пять функциональных блоков, представляющих собой инкрементирующий счетчик на пяти языках IEC 61131-3. При запуске программы начальное значение переменной Reset устанавливается по умолчанию False. Значения счетчиков начнут увеличиваться, начиная со значения по умолчанию (для типа INT равно 0). Для сброса счетчиков переменную Reset необходимо форсировать значением True, затем вернуть значение False. Переменным Cnt1..Cnt5 будет присвоено начальное значение конфигурационной константы ResetCounterValue, таким образом значения счетчиков сбросятся, и начнется отсчет начиная с 17.

Сначала следует добавление программы в проект, осуществляемое с помощью меню дерева проекта, выбором пункта «Программа» (см. [Рисунок 2.107](#)):

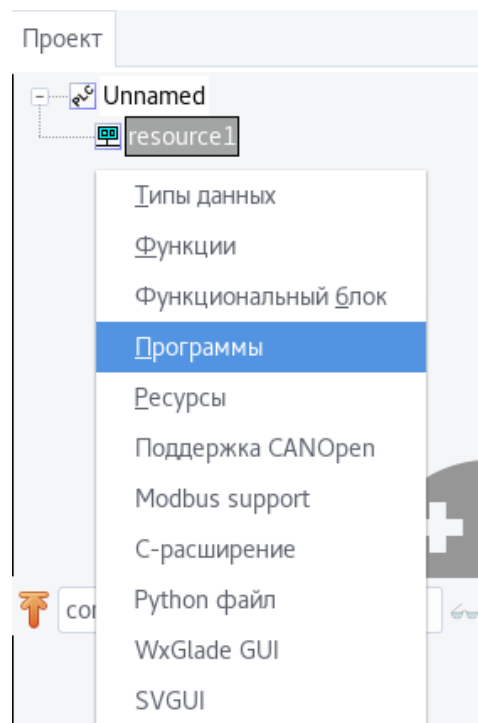


Рисунок 2.107: - Добавление программы в проект

В появившемся диалоге (см. [Рисунок 2.108](#)) выберем язык FBD и тип POU «программа».

Добавим в панели переменных и констант переменную Reset типа BOOL, отвечающую за сброс каждого из пяти счетчиков, а так же пять переменных Cnt1..Cnt5 типа INT, в которых будут храниться значения каждого из пяти счетчиков. Далее необходимо обратиться к редактору языка FBD. Для написания алгоритма и логики выполнения данной программы нам понадобятся функциональные блоки счетчиков, создание которых рассмотрено в п. 6.3.2.

Для удобства редактирования FBD диаграмм в редакторе существует функция Drag&Drop, необходимые функциональные блоки и переменные можно добавить в поле редактирования из библиотеки

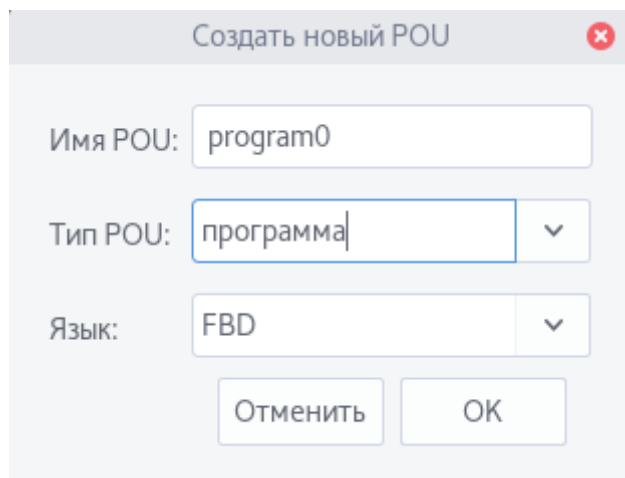


Рисунок 2.108: - Диалог добавления программы в проект

функций и функциональных блоков и таблицы переменных путем перетаскивания в поле редактирования. 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).

Перенесем 5 экземпляров переменной Reset и все переменные Cnt1..Cnt5 в поле редактирования диаграммы как показано на [Рисунок 2.109](#) :

Из библиотеки функций и функциональных блоков добавим пользовательские функциональные блоки. Добавление данных функциональных блоков 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 как показано на [Рисунок 2.110](#) :

Добавим связи между функциональными блоками и входными и выходными переменными.

Функциональный блок

Добавление пользовательского функционального блока происходит путем нажатия на пункт «Функциональный блок» во всплывающем меню дерева проекта . В диалоговом окне (см. [Рисунок 2.112](#)) задайте имя функционального блока в поле «Имя POU», в поле «Тип POU» выберите «функциональный блок», в поле «Язык» выберите язык, на котором будет написан алгоритм работы блока.

Функциональный блок на языке ST

Создайте функциональный блок с именем «CounterST» (см. [Рисунок 2.113](#)), в котором инструментами языка ST будет реализован счетчик , принимающий на вход переменную Reset типа BOOL, и возвращающий значение счетчика Out.

В отличие от функции, функциональный блок может быть описан на любом языке стандарта IEC 61131-3, включая язык SFC. На [Рисунок 2.114](#) показана реализация данного функционального блока на языке ST.

Возвращаемое значение «Out» имеет тип INT и класс «Выход». Добавленные локальная переменная «Cnt» и внешняя конфигурационная переменная «ResetCounterValue» имеют тип INT, входная переменная «Reset» имеет тип BOOL. Реализованный функциональный блок становится доступным в панели библиотеки функций и функциональных блоков и может использоваться в программных модулях

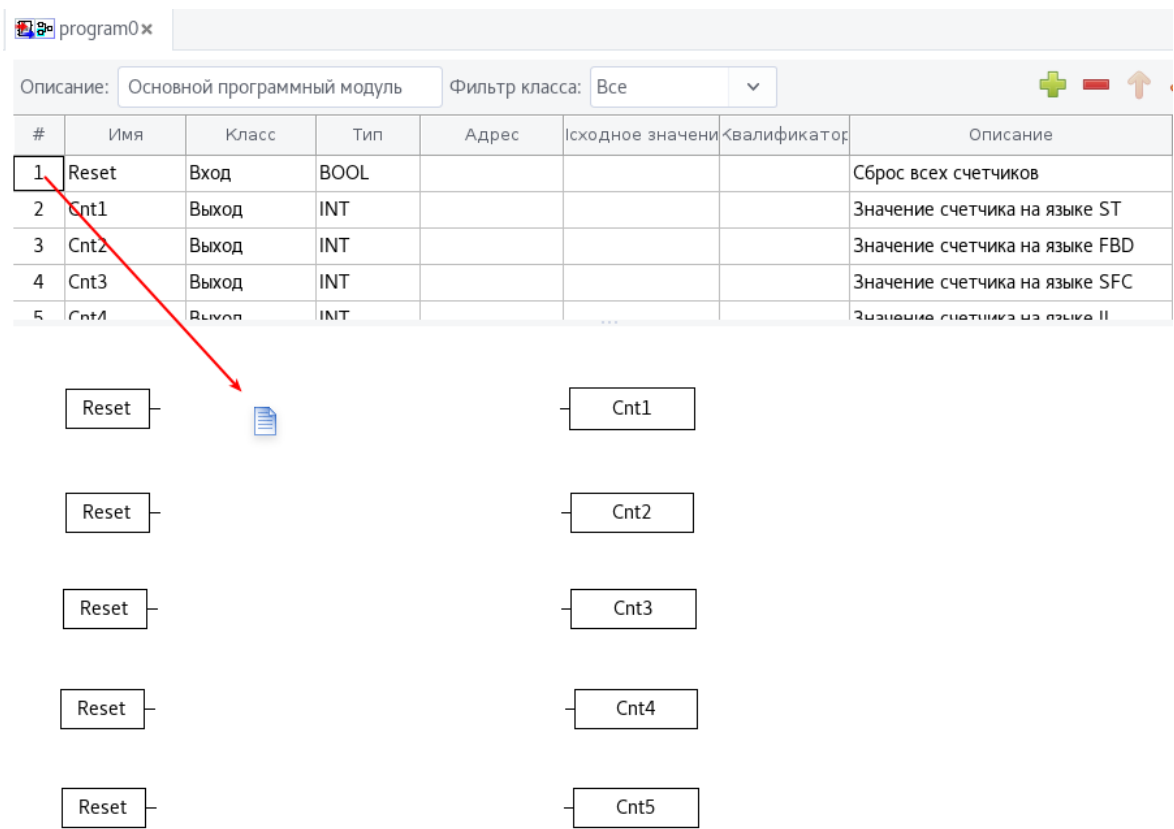


Рисунок 2.109: - Перенос переменных в поле редактирования

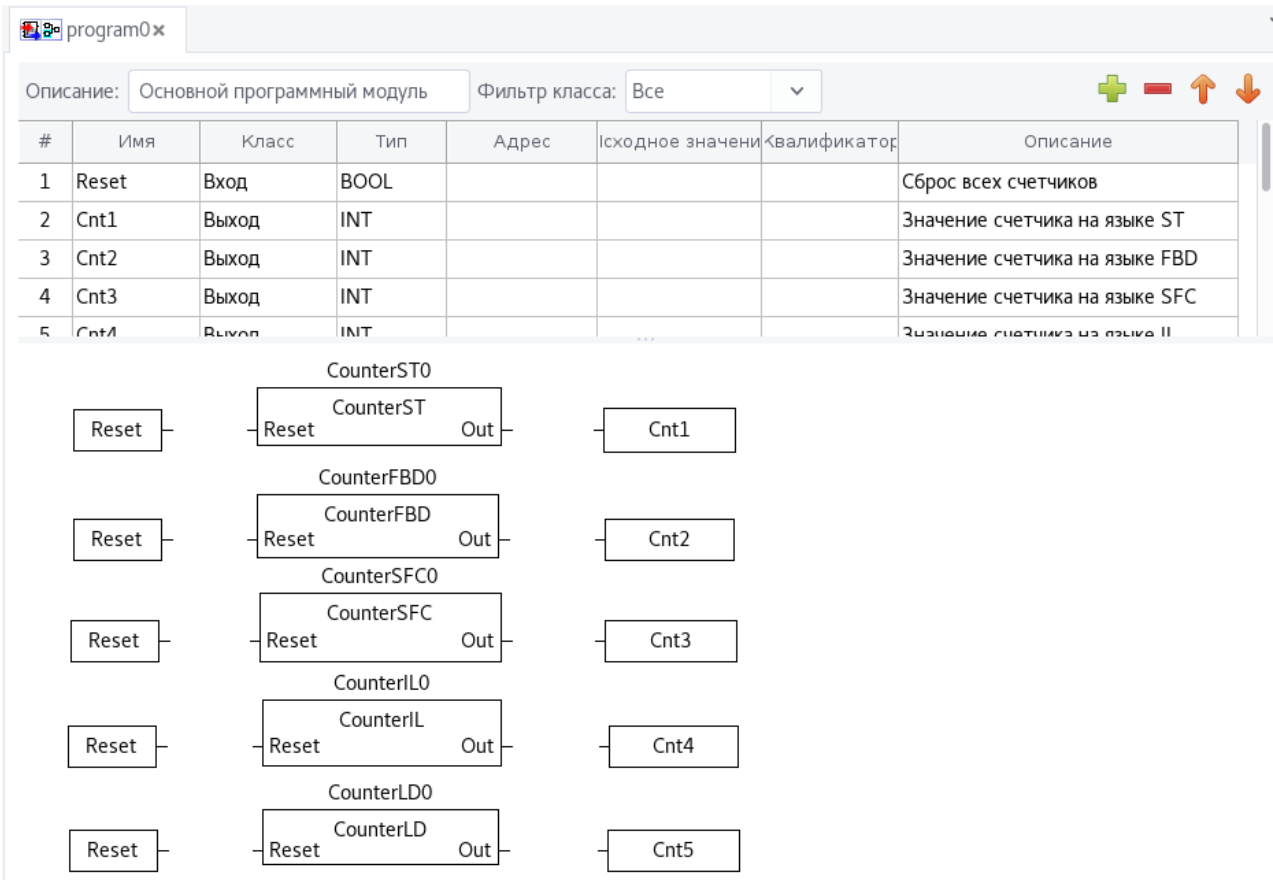


Рисунок 2.110: - Программа на языке FBD без связей

program0x

Описание: Основной программный модуль

Фильтр класса: Все

+

-

↑

#	Имя	Класс	Тип	Адрес	Исходное значение	Квалификатор	Описание
1	Reset	Вход	BOOL				Сброс всех счетчиков
2	Cnt1	Выход	INT				Значение счетчика на языке ST
3	Cnt2	Выход	INT				Значение счетчика на языке FBD
4	Cnt3	Выход	INT				Значение счетчика на языке SFC
5	Cnt4	Выход	INT		...		Значение счетчика на языке IL

CounterST0

Reset

CounterST

Out

Cnt1

CounterFBD0

Reset

CounterFBD

Out

Cnt2

CounterSFC0

Reset

CounterSFC

Out

Cnt3

CounterIL0

Reset

CounterIL

Out

Cnt4

CounterLD0

Reset

CounterLD

Out

Cnt5

Рисунок 2.111: - Основной программный модуль на языке FBD

Создать новый POU

Имя POU: CounterLANG

Тип POU: функциональный блок

Язык:

IL

ST

LD

FBD

SFC

Рисунок 2.112: - Диалог создания нового функционального блока

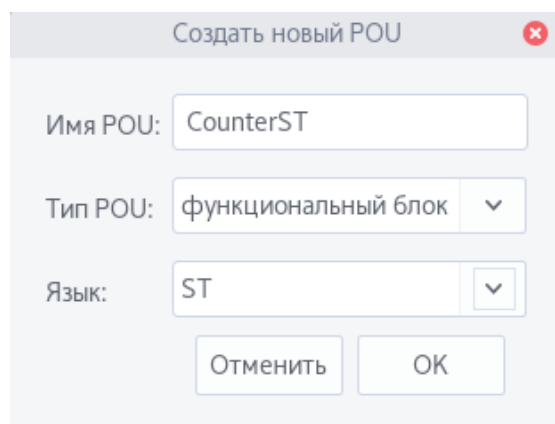
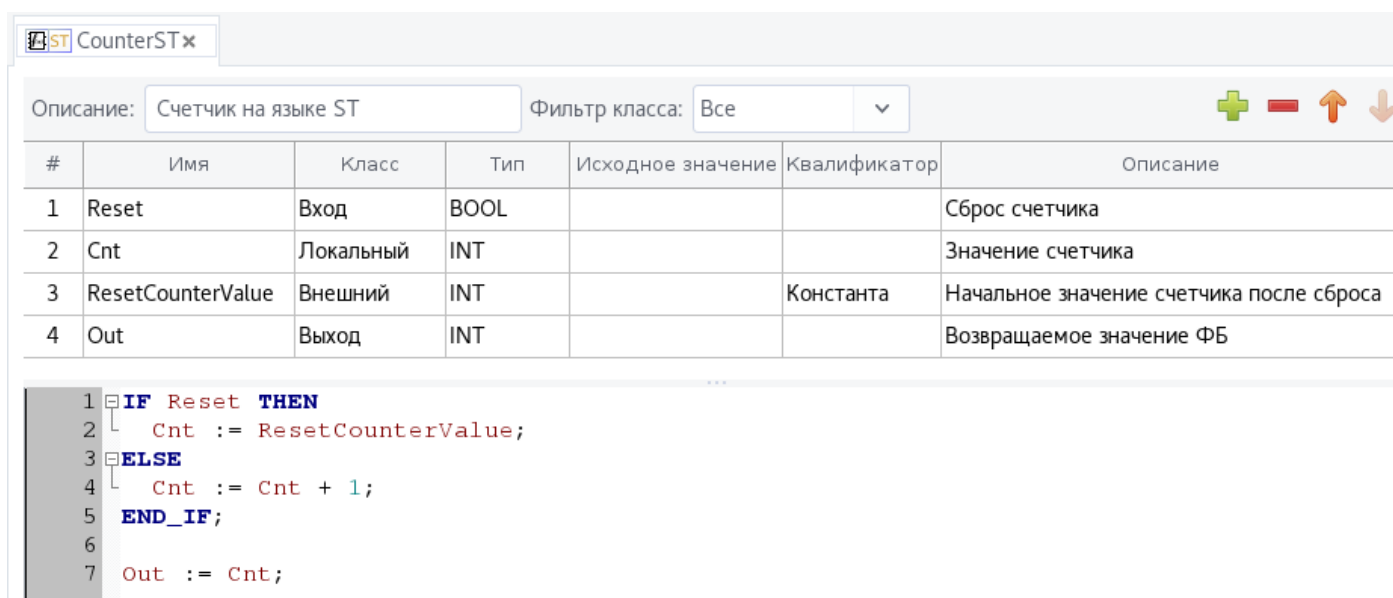


Рисунок 2.113: - Диалог добавления пользовательского функционального блока



#	Имя	Класс	Тип	Исходное значение	Квалификатор	Описание
1	Reset	Вход	BOOL			Сброс счетчика
2	Cnt	Локальный	INT			Значение счетчика
3	ResetCounterValue	Внешний	INT		Константа	Начальное значение счетчика после сброса
4	Out	Выход	INT			Возвращаемое значение ФБ

```
1 IF Reset THEN
2   Cnt := ResetCounterValue;
3 ELSE
4   Cnt := Cnt + 1;
5 END_IF;
6
7 Out := Cnt;
```

Рисунок 2.114: - описание пользовательского функционального блока на языке ST

типа «Программа» и «Функциональный блок». На [Рисунок 2.115](#) показано использование созданного функционального блока «CounterST» в основном программном модуле, написанном на языке FBD.

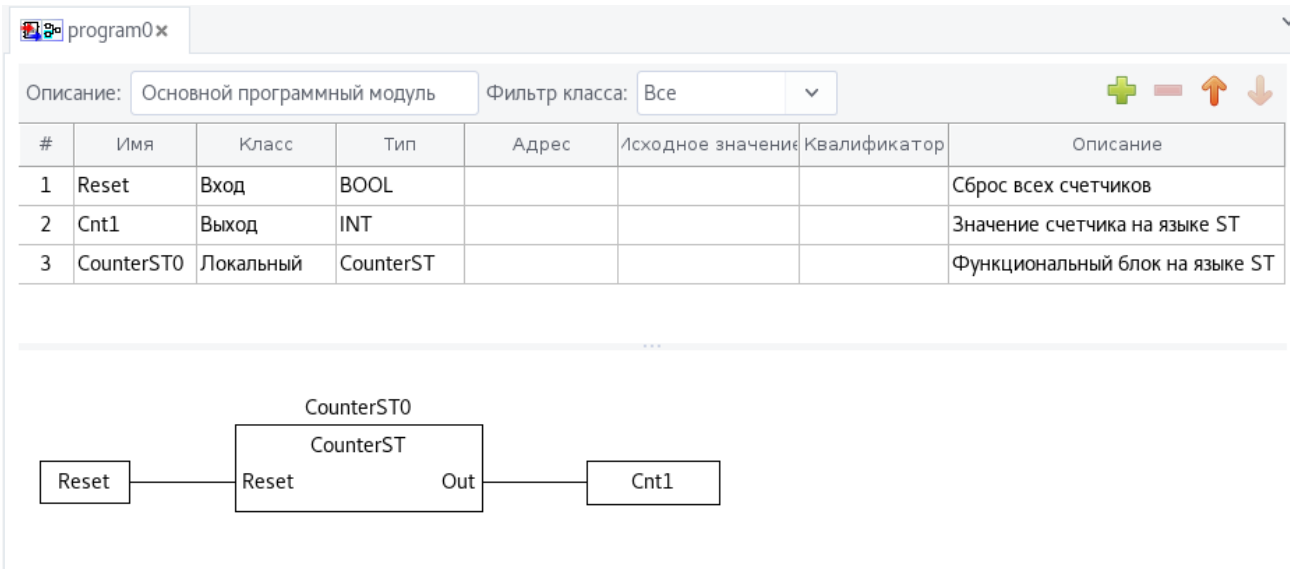


Рисунок 2.115: - Использование созданного функционального блока CounterST в основном программном модуле

С входом «Reset» соединена общая для всех счетчиков переменная «Reset» типа BOOL, результат выполнения помещается в переменную «Cnt1» типа INT. Следует отметить, что при попытке удаления функции или функционального блока из проекта (см. [Рисунок 2.116](#)), где эти добавленные программные модули уже используются, будет выдана ошибка.

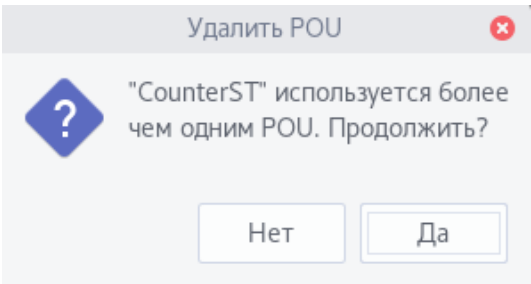


Рисунок 2.116: - Сообщение об ошибке при удалении функционального блока

Реализации на других языках полностью идентичны по набору входных, выходных и локальных переменных. Далее будут описаны примеры функциональных блоков на остальных четырех языках IEC 61131-3.

Функциональный блок на языке FBD

Создайте функциональный блок с именем «CounterFBD», в котором инструментами языка FBD будет реализован счетчик, принимающий на вход переменную «Reset» типа BOOL, и возвращающий значение счетчика «Out». Для удобства редактирования FBD диаграмм в редакторе существует функция Drag&Drop, необходимые функциональные блоки и переменные можно добавить в поле редактирования из библиотеки функций и функциональных блоков и таблицы переменных путем перетаскивания в поле редактирования (см. [Рисунок 2.117](#)). 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).

Добавим возвращаемое значение «Out» типа INT и класса «Выход», локальную переменную «Cnt» типа INT, внешнюю конфигурационную переменную «ResetCounterValue» типа INT, и входную переменную «Reset» типа BOOL.

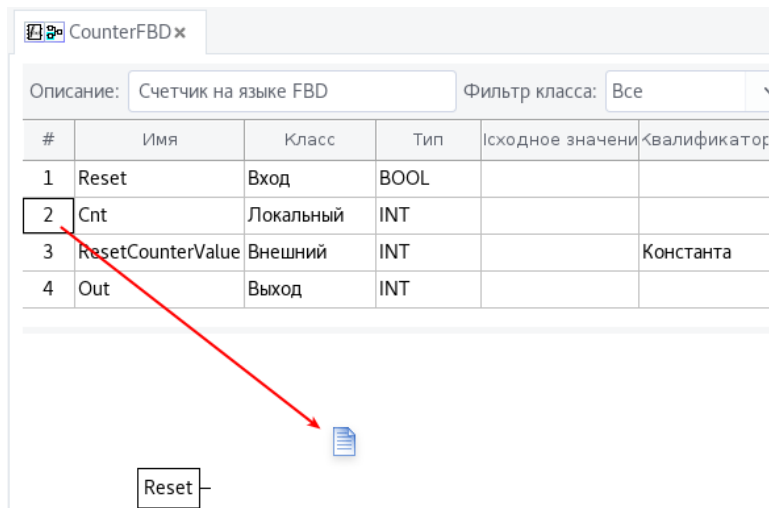


Рисунок 2.117: - Добавление переменной в поле редактирования

Перенесенные на поле редактирования переменные отображаются как прямоугольные блоки с коннекторами входа и выхода (см. Рисунок 2.118).

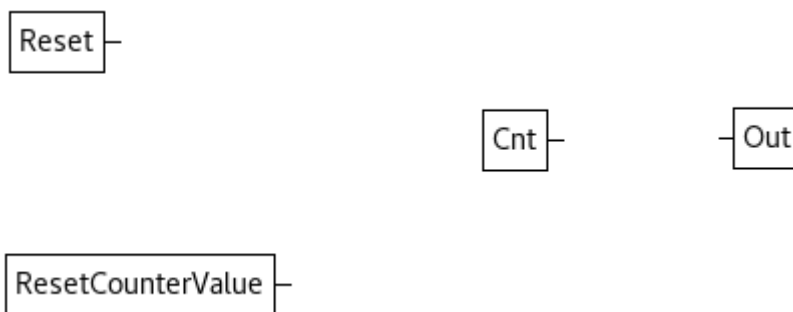


Рисунок 2.118: - Блоки переменных в поле редактирования

После переноса в поле редактирования всех переменных, добавьте числовой литерал со значением «1» при помощи кнопки «Создать новую переменную», в диалоговом окне создания переменной в поле «Выражение» напишите «1» (см. Рисунок 2.119). Таким способом задается шаг инкрементации счетчика.

Для того чтобы переменной «Cnt» можно было одновременно и присвоить значение и передать это значение переменной Out, задайте класс переменной «Вход/Выход». Сделать это можно щелчком правой кнопкой мыши по блоку переменной, во всплывающем меню следует выбрать «Вход/Выход» (см. Рисунок 2.120), или щелкнув по блоку двойным щелчком левой кнопки мыши, выбрав в выпадающем списке «Класс» вариант «Вход/Выход» (см. Рисунок 2.121).

Далее необходимо обратиться к редактору языка FBD. Для написания алгоритма и логики выполнения данной программы будут добавлены две функции: «ADD» и «SEL».

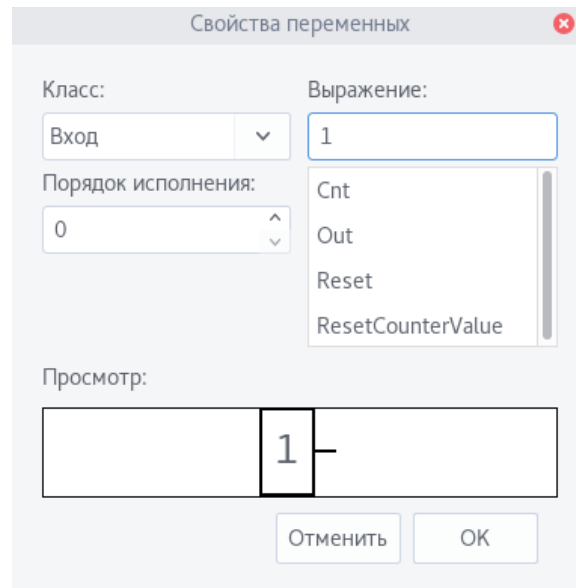


Рисунок 2.119: - Диалог создания переменной

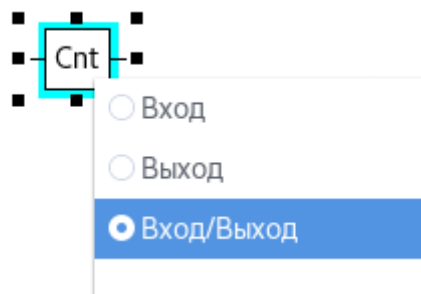


Рисунок 2.120: - Выбор коннектора для блока переменной

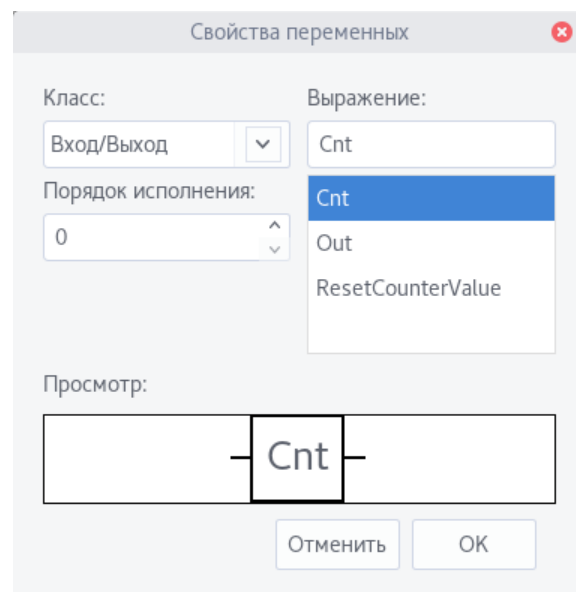


Рисунок 2.121: - Диалог редактирования свойств блока переменной

Функция «ADD» находится во вкладке «Математика» в Библиотеке функций и функциональных блоков, обозначает сложение от 2 до 20 входных значений (в нашем примере их 2) на входах «IN1» и «IN2», возвращает результат вычисления на выход «OUT».

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 функционального блока. Результатом вышеизложенных действий должна стать FBD диаграмма без соединений (см. [Рисунок 2.122](#)).

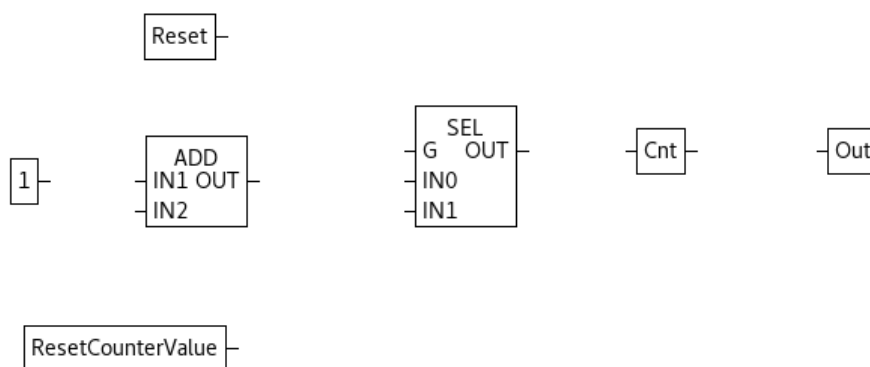


Рисунок 2.122: - FBD диаграмма без соединений

Следующим шагом станет соединение выходов переменных со входами функций. Соединим числовой литерал 1 с входом «IN1» функции ADD, а выход «OUT» функции ADD соединим с входом «IN0» функции SEL. В свою очередь, выход «OUT» функции SEL соединим с входным коннектором переменной Cnt, а выходной коннектор переменной Cnt соединим с входом переменной «Out». Соединение блоков осуществляется путем зажатия левой кнопки мыши на коннекторе блока, будет создана линия связи которую необходимо протянуть до коннектора присоединяемого блока (см. [Рисунок 2.123](#)).

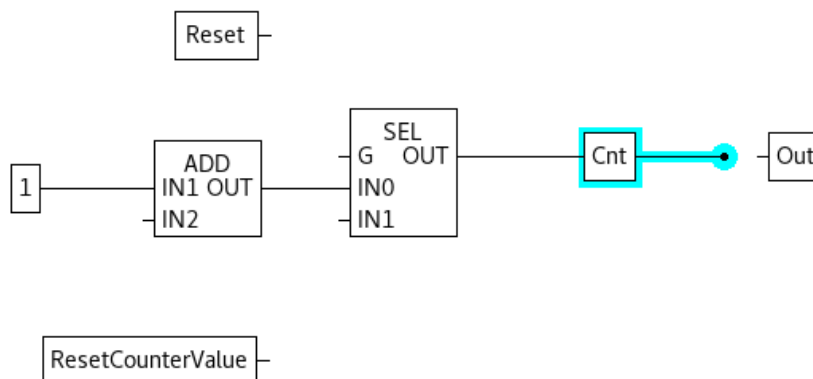


Рисунок 2.123: - Соединение блоков в FBD диаграмме

Далее присоединим переменную «Reset», управляющую сбросом счетчика, на вход «G» функции «SEL», а конфигурационную переменную «ResetCounterValue» на вход «IN1». Таким образом, меняя

значение переменной «Reset» мы управляем значением переменной «Cnt» через функцию выбора значения «SEL». Осталось добавить связь между переменной «Cnt» и входом «IN2» функции сложения ADD, тем самым обеспечив увеличение значения счетчика на 1 за один цикл ПЛК.

Полученная реализация алгоритма счетчика на языке FBD представлена на [Рисунок 2.124](#).

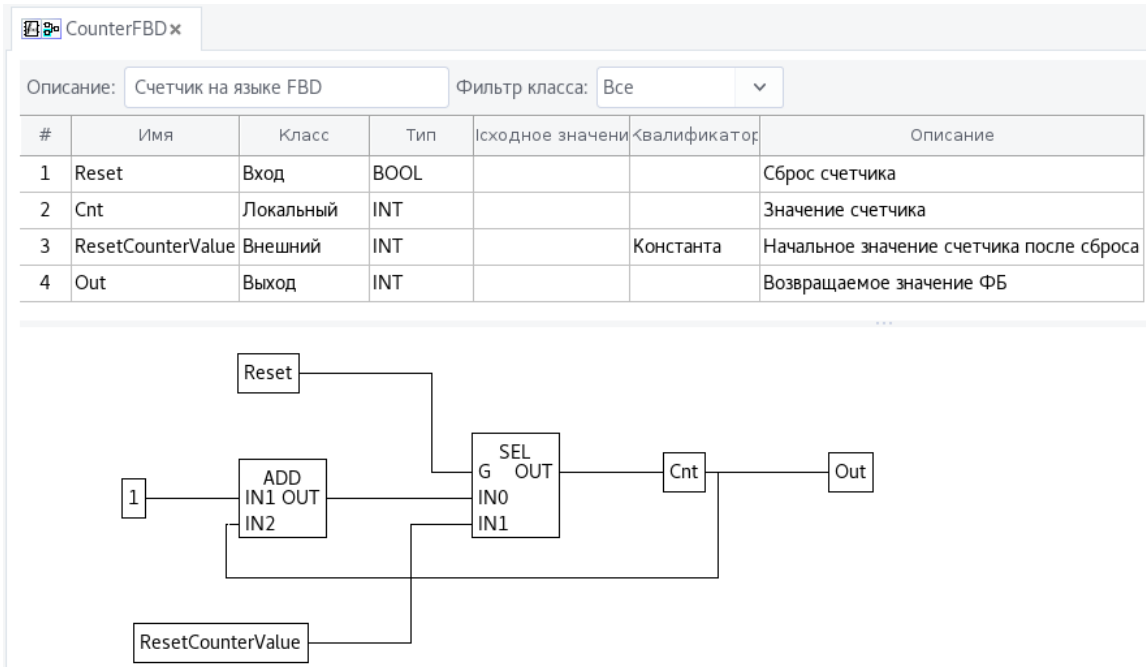


Рисунок 2.124: - Функциональный блок на языке FBD

Функциональный блок становится доступным в панели библиотеки функций и функциональных блоков и может использоваться в программных модулях типа «Программа» и «Функциональный блок». На [Рисунок 2.125](#) показано использование созданного функционального блока «CounterFBD» в основном программном модуле, написанном на языке FBD.

Функциональный блок на языке SFC

Создайте функциональный блок с именем «CounterSFC», в котором инструментами языка SFC будет реализован счетчик, принимающий на вход переменную «Reset» типа BOOL, и возвращающий значение счетчика «Out».

Добавим в панель переменных и констант возвращаемое значение «Out» типа INT и класса «Выход», локальную переменную «Cnt» типа INT, внешнюю конфигурационную переменную «ResetCounterValue» типа INT, и входную переменную «Reset» типа BOOL.

Для удобства редактирования SFC диаграмм в редакторе существует функция Drag&Drop, необходимые функциональные блоки и переменные можно добавить в поле редактирования из библиотеки функций и функциональных блоков и таблицы переменных путем перетаскивания в поле редактирования (см. [Рисунок 2.117](#)). необходимо левой клавишей мыши зажать столбец «#» для переменной в панели переменных и констант, далее перенести указатель на область редактирования SFC диаграммы и отпустить кнопку мыши (Drag&Drop).

Добавим начальный шаг диаграммы, нажав на кнопку «Создать исходный шаг», в диалоге изменим название шага по умолчанию на название «Start», коннектор потребуется только «Выход» (см. [Рисунок 2.126](#))

Следуя алгоритму, возможны два состояния – счетчик увеличивается и счетчик сброшен. Добавим

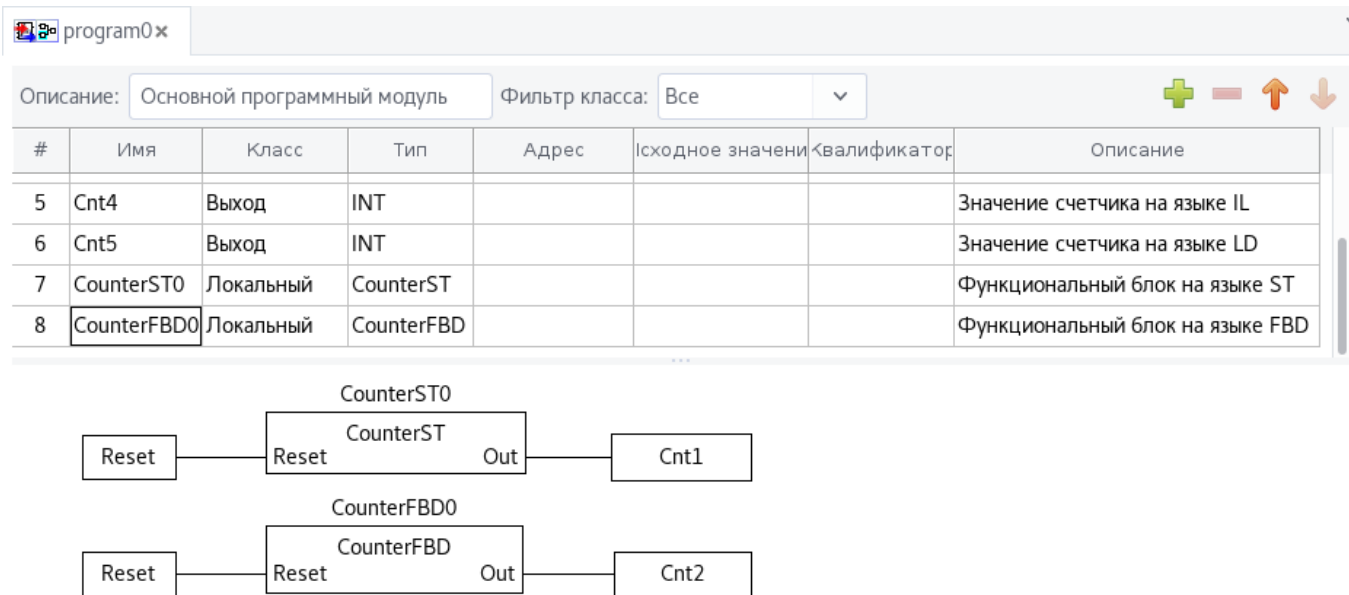


Рисунок 2.125: - Использование созданного функционального блока CounterFBD в основном программном модуле

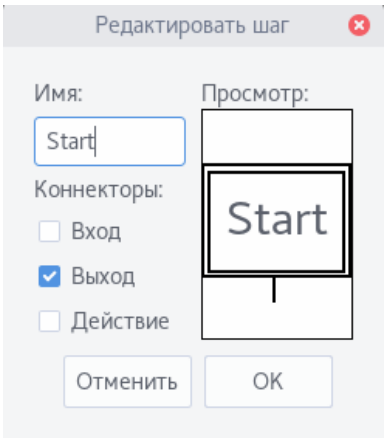


Рисунок 2.126: - Добавление начального шага

альтернативное ветвление с двумя ветвями. Согласно стандарту IEC 61131-3, каждая ветвь альтернативного ветвления должна оканчиваться переходом. Условиями переходов будет являться состояние переменной «Reset» : для первой ветви выражение «NOT Reset» , для второй ветви просто значения «Reset» (см.:numref:image173).

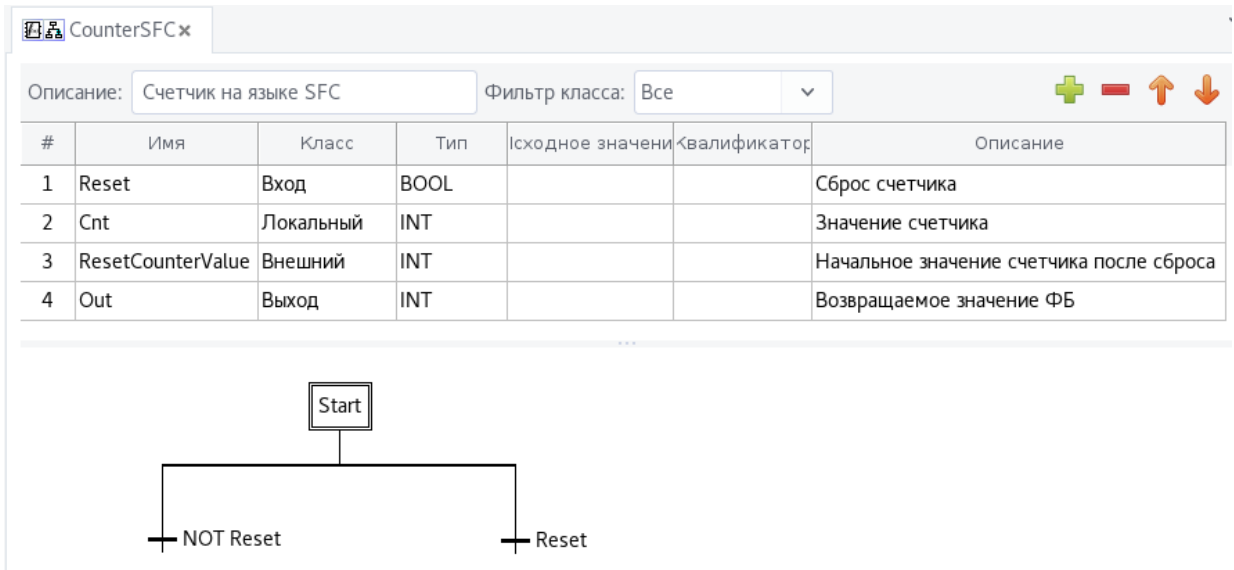


Рисунок 2.127: - Добавление альтернативного ветвления

В первом состоянии добавим шаг с действием «Count» (см. Рисунок 2.128), в действии на языке ST опишем увеличение счетчика на единицу, и присвоение значения переменной «Out» (см. Рисунок 2.129).

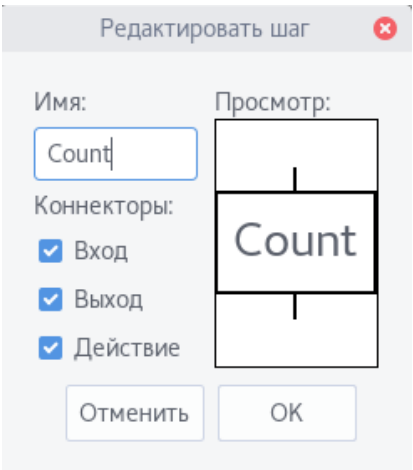


Рисунок 2.128: - Добавление шага с коннектором действия

Во второй ветви добавим шаг с действием «ResetCounter», в действии опишем присвоение переменной «Cnt» значение переменной «ResetCounterValue», и значению «Out» значение переменной «Cnt» (см.:numref:image176).

Первая ветвь отвечает за инкрементацию счетчика, вторая – за сброс (см. Рисунок 2.131).

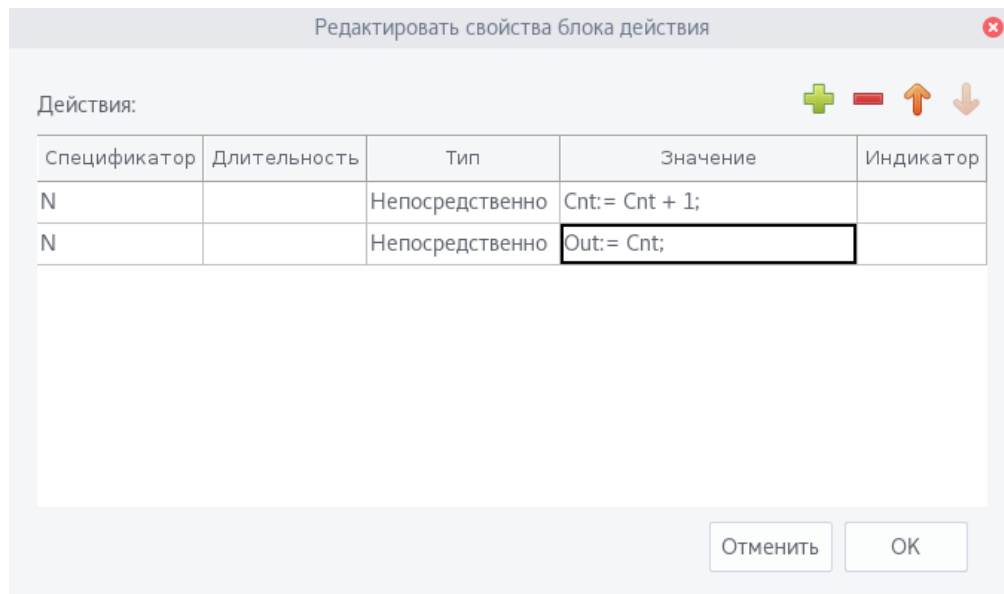


Рисунок 2.129: - Добавление действия инкрементации счетчика

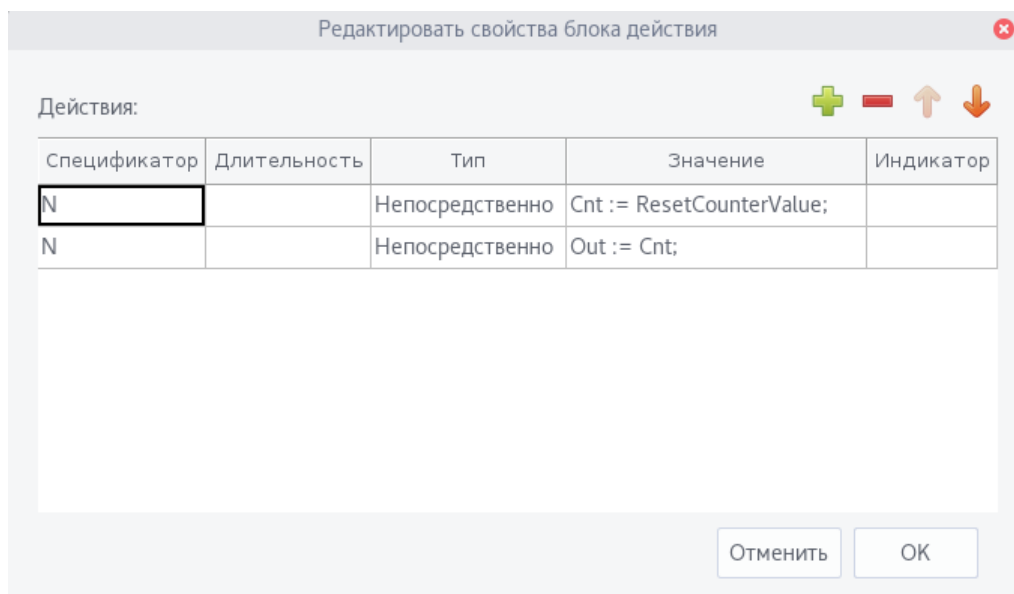


Рисунок 2.130: - Добавление действия сброса счетчика

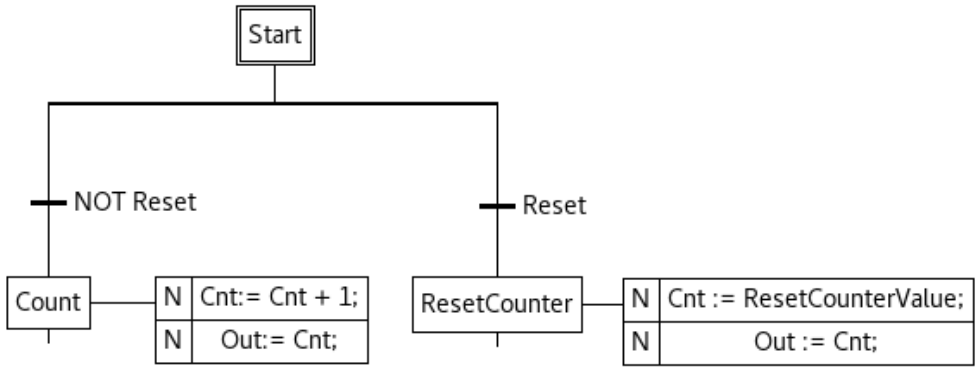


Рисунок 2.131: - Шаги с действиями

Для выходов из состояния добавим в первой ветви переход с условием «Reset», во второй ветви добавим переход с условием «NOT Reset» (см. Рисунок 2.132).

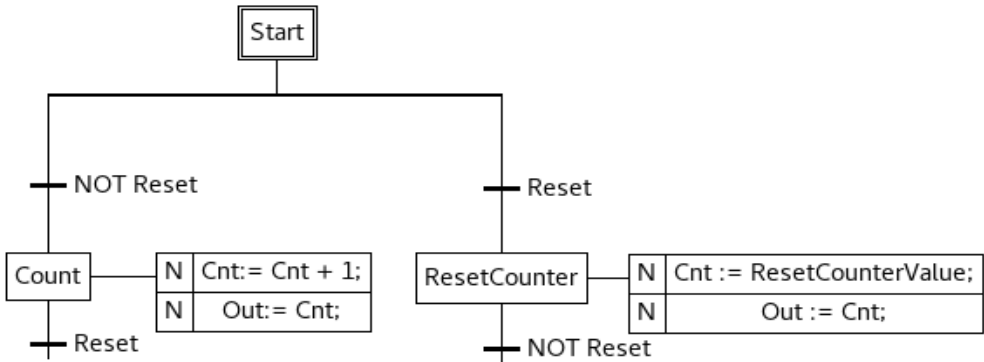


Рисунок 2.132: - Условные переходы для выхода из состояний

Далее, объединим ветви альтернативным объединением. Для того чтобы программа выполнялась циклически, после объединения добавим безусловный переход к начальному шагу «Start». Полученная реализация счетчика на языке SFC представлена на Рисунок 2.133 .

Функциональный блок становится доступным в панели библиотеки функций и функциональных блоков и может использоваться в программных модулях типа «Программа» и «Функциональный блок». На Рисунок 2.134 показано использование созданного функционального блока «CounterSFC» в основном программном модуле, написанном на языке FBD.

Функциональный блок на языке IL

Создайте функциональный блок с именем «CounterIL», в котором инструментами языка IL будет реализован счетчик , принимающий на вход переменную Reset типа BOOL, и возвращающий значение счетчика Out.

Добавим в панель переменных и констант возвращаемое значение «Out» типа INT и класса «Выход», локальную переменную «Cnt» типа INT, внешнюю конфигурационную переменную «ResetCounterValue» типа INT, и входную переменную «Reset» типа BOOL.

Для удобства редактирования кода в редакторе IL существует функция Drag&Drop , необходимые переменные можно добавить в поле редактирования из таблицы переменных путем перетаскивания в

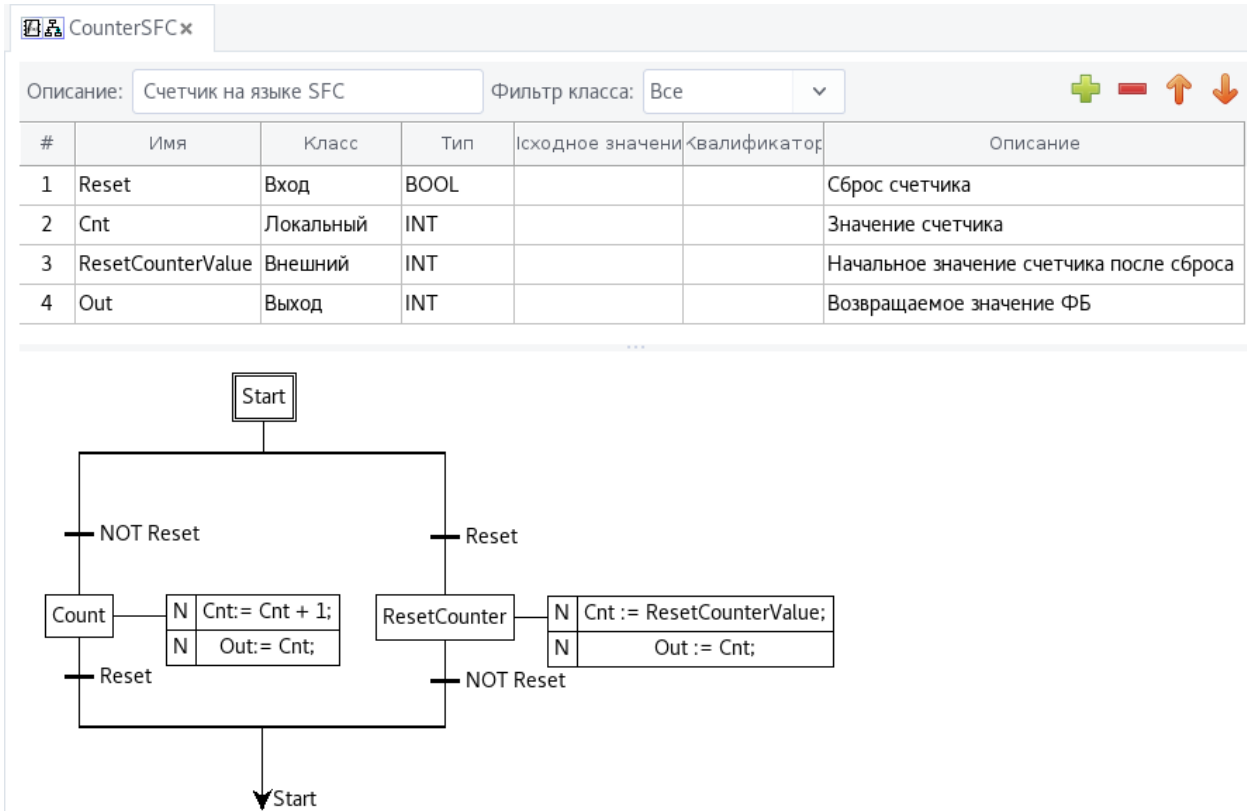


Рисунок 2.133: - Реализация счетчика на языке SFC

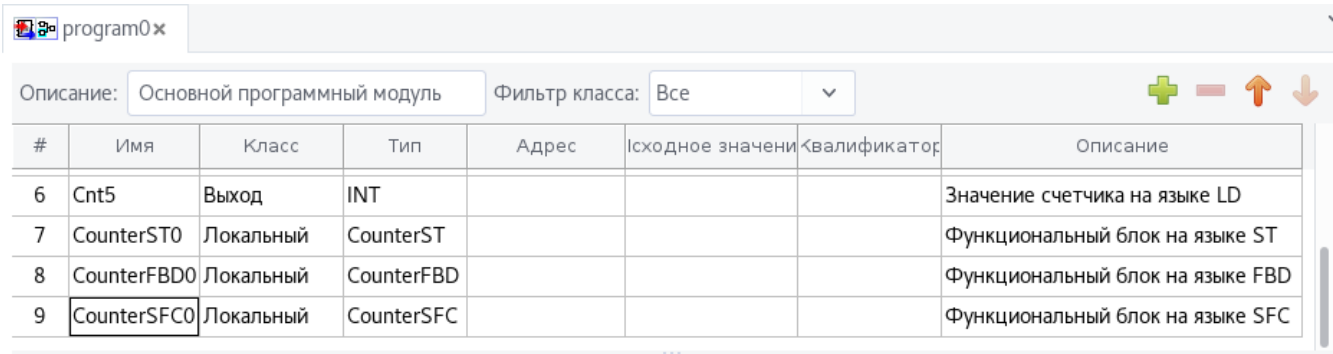


Рисунок 2.134: - Использование созданного функционального блока CounterSFC в основном программном модуле

поле редактирования (см. [Рисунок 2.116](#)). необходимо левой клавишей мыши зажать столбец «#» для переменной в панели переменных и констант, далее перенести указатель на область редактирования и отпустить кнопку мыши (Drag&Drop).

Напишем инструкции для сброса счетчика и сохранения результатов. Инструкцию для сброса счетчика назовем «ResetCnt», она загрузит операнд ResetCounterValue в аккумулятор:

```
ResetCnt:
(* reset counter *)
LD ResetCounterValue
```

Инструкцию для сохранения результатов назовем «QuitFb», она будет сохранять значения операндов «Cnt» и «Out»:

```
QuitFb:
(* save results *)
ST Cnt
ST Out
```

Загрузим в аккумулятор значение операнда «Reset». Если значение операнда «True», следует перейти к инструкции сброса счетчика ResetCnt, в случае значения «False» значение операнда должно увеличиться на единицу.

```
LD Reset
JMPC ResetCnt
(* increment counter *)
LD Cnt
ADD 1
JMP QuitFb
ResetCnt:
(* reset counter *)
LD ResetCounterValue
QuitFb:
(* save results *)
ST Cnt
ST Out
```

Полученная реализация счетчика на языке IL представлена на [Рисунок 2.135](#) .

Функциональный блок становится доступным в панели библиотеки функций и функциональных блоков и может использоваться в программных модулях типа «Программа» и «Функциональный блок». На [Рисунок 2.136](#) показано использование созданного функционального блока «CounterIL» в основном программном модуле, написанном на языке FBD.

Функциональный блок на языке LD

Создайте функциональный блок с именем «CounterLD», в котором инструментами языка LD будет реализован счетчик , принимающий на вход переменную «Reset» типа BOOL, и возвращающий значение счетчика «Out».

Добавим в панель переменных и констант возвращаемое значение «Out» типа INT и класса «Выход», локальную переменную «Cnt» типа INT, внешнюю конфигурационную переменную «ResetCounterValue» типа INT, и входную переменную «Reset» типа BOOL.

Для удобства редактирования LD диаграмм в редакторе существует функция Drag&Drop , необходимые функциональные блоки и переменные можно добавить в поле редактирования из библиотеки функций и функциональных блоков и таблицы переменных путем перетаскивания в поле редактирования (см. [Рисунок 2.117](#)). необходимо левой клавишей мыши зажать столбец «#» для переменной в

IL CounterILx

Описание: Счетчик на языке IL

Фильтр класса: Все

+

−

↑

↓

#	Имя	Класс	Тип	Исходное значение	Квалификатор	Описание
1	Reset	Вход	BOOL			Сброс счетчика
2	Cnt	Локальный	INT			Значение счетчика
3	ResetCounterValue	Внешний	INT		Константа	Начальное значение счетчика после сброса
4	Out	Выход	INT			Возвращаемое значение ФБ

```
1 LD Reset
2 JMPC ResetCnt
3
4 (* increment counter *)
5 LD Cnt
6 ADD 1
7 JMP QuitFb
8
9 ResetCnt:
10 (* reset counter *)
11 LD ResetCounterValue
12
13 QuitFb:
14 (* save results *)
15 ST Cnt
16 ST Out
```

Рисунок 2.135: - Реализация счетчика на языке IL

program0x

Описание: Основной программный модуль

Фильтр класса: Все

#	Имя	Класс	Тип	Адрес	Исходное значение	Квалификатор	Описание
3	CounterST0	Локальный	CounterST				Функциональный блок на языке ST
4	CounterFBD0	Локальный	CounterFBD				Функциональный блок на языке FBD
5	CounterSFC0	Локальный	CounterSFC				Функциональный блок на языке SFC
6	CounterIL0	Локальный	CounterIL				Функциональный блок на языке IL

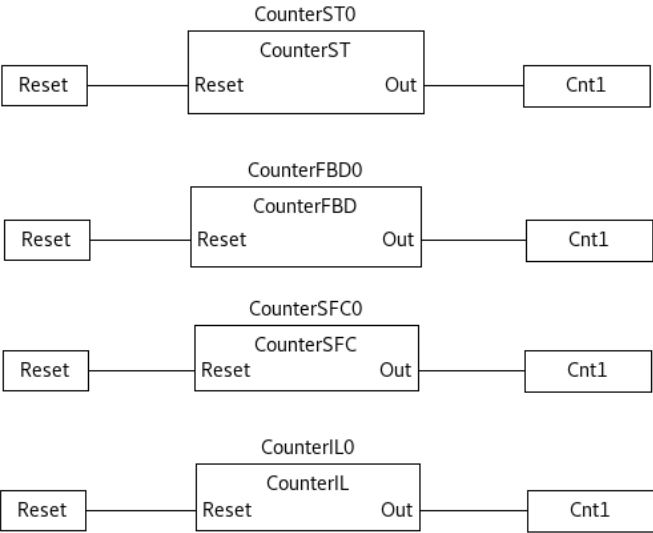


Рисунок 2.136: - Использование функционального блока CounterIL в основном программном модуле

панели переменных и констант, далее перенести указатель на область редактирования LD диаграммы и отпустить кнопку мыши (Drag&Drop).

Добавим шину питания, к ней присоединим контакт, связанный с переменной «Reset» (см. [Рисунок 2.137](#)).

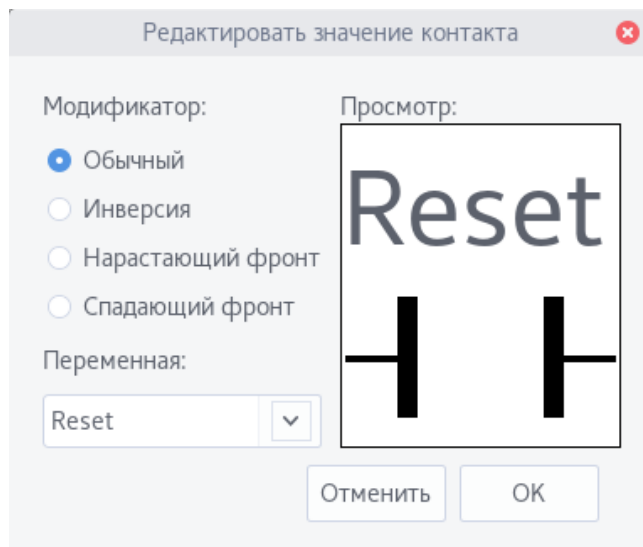


Рисунок 2.137: - Диалог добавления контакта

Полученная конструкция будет подавать сигнал на сброс счетчика при переходе значения переменной «Reset» в True (см. [Рисунок 2.138](#))

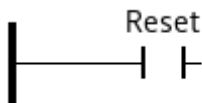


Рисунок 2.138: - Контакт ассоциированный с переменной Reset

Далее добавим числовой литерал со значением «1» при помощи кнопки «Создать новую переменную», в диалоговом окне создания переменной в поле «Выражение» напишите «1» (см. [Рисунок 2.139](#)). Таким способом задается шаг инкрементации счетчика.

Перенесенные на поле редактирования переменные отображаются как прямоугольные блоки с коннекторами входа и выхода (см. [Рисунок 2.140](#)).

Для того чтобы переменной «Cnt» можно было одновременно и присвоить значение и передать это значение переменной Out, задайте класс переменной «Вход/Выход». Сделать это можно щелчком правой кнопкой мыши по блоку переменной, во всплывающем меню следует выбрать «Вход/Выход» (см. [Рисунок 2.141](#)), или щелкнув по блоку двойным щелчком левой кнопки мыши, выбрав в выпадающем списке «Класс» вариант «Вход/Выход» (см. [Рисунок 2.142](#)).

Для написания алгоритма и логики выполнения данной программы будут добавлены две функции: «ADD» и «SEL».

Функция «ADD» находится во вкладке «Математика» в Библиотеке функций и функциональных блоков, и обозначает сложение от 2 до 20 входных значений (в нашем примере их 2) на входах «IN1» и «IN2», возвращает результат вычисления на выход «OUT».

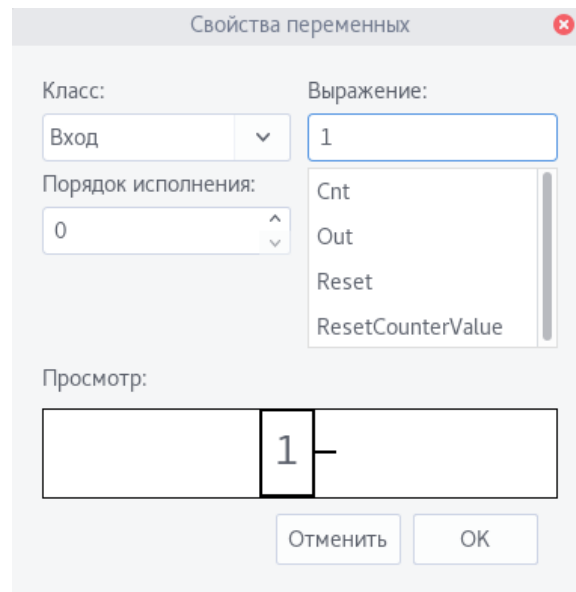


Рисунок 2.139: - Диалог создания переменной

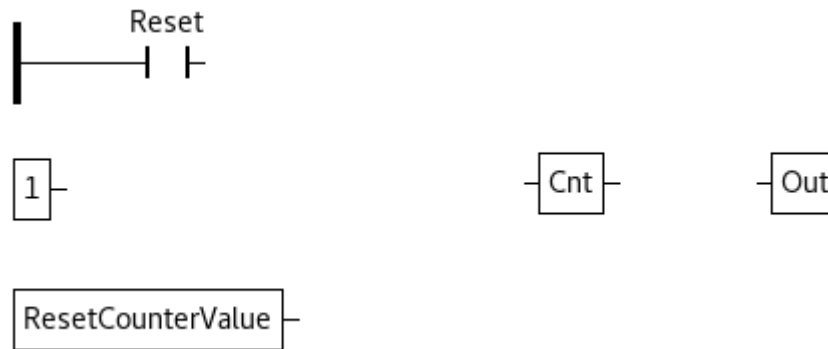


Рисунок 2.140: - Блоки переменных в поле редактирования

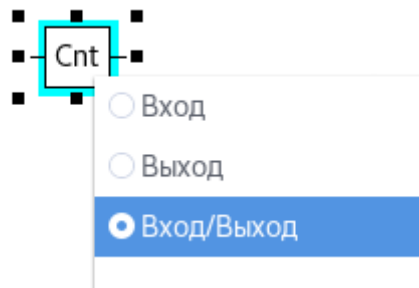


Рисунок 2.141: - Выбор коннектора для блока переменной

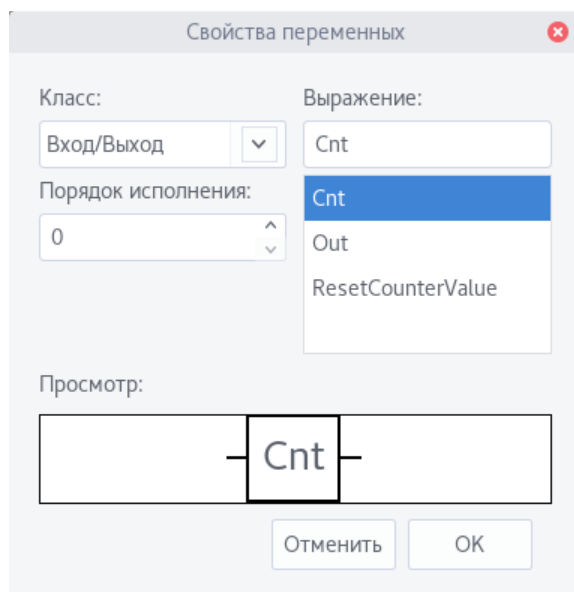


Рисунок 2.142: - Диалог редактирования свойств блока переменной

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 функционального блока. Результатом вышеизложенных действий должна стать LD диаграмма без соединений.

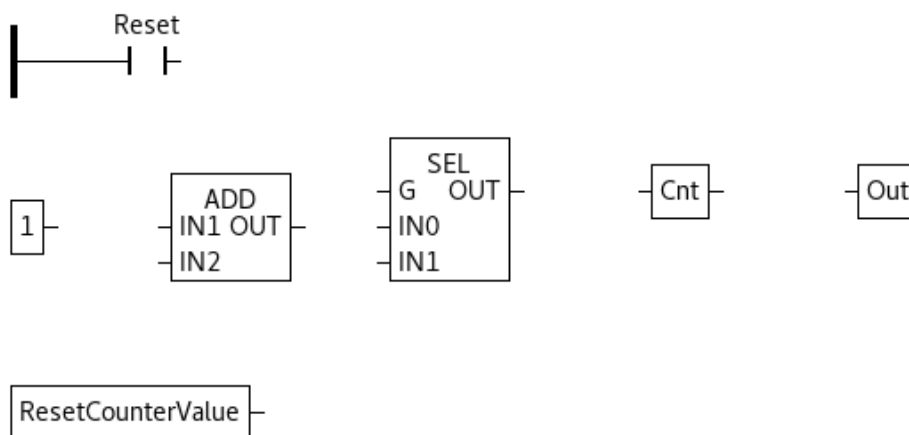


Рисунок 2.143: - LD диаграмма без соединений

Следующим шагом станет соединение выходов переменных со входами функций. Соединим числовой литерал 1 с входом «IN1» функции ADD, а выход «OUT» функции ADD соединим с входом «IN0» функции SEL. В свою очередь, выход «OUT» функции SEL соединим с входным коннектором переменной Cnt, а выходной коннектор переменной Cnt соединим с входом переменной «Out». Соединение блоков осуществляется путем зажатия левой кнопки мыши на коннекторе блока, будет создана линия

связи которую необходимо протянуть до коннектора присоединяемого блока .

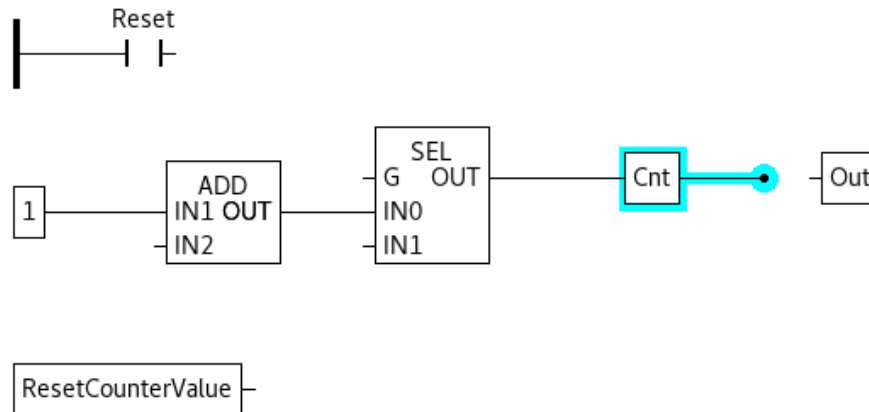


Рисунок 2.144: - Соединение блоков

Далее присоединим сигнал с контакта, ассоциированного с переменной «Reset», управляющей сбросом счетчика, на вход «G» функции «SEL», а конфигурационную переменную «ResetCounterValue» на вход «IN1». Таким образом, меняя значение переменной «Reset» мы управляем значением переменной «Cnt» через функцию выбора значения «SEL». Осталось добавить связь между переменной «Cnt» и входом «IN2» функции сложения ADD, тем самым обеспечив увеличение значения счетчика на 1 за один цикл ПЛК.

Полученная реализация алгоритма счетчика на языке LD представлена на [Рисунок 2.145](#) .

Функциональный блок становится доступным в панели библиотеки функций и функциональных блоков и может использоваться в программных модулях типа «Программа» и «Функциональный блок». На рис. 146 показано использование созданного функционального блока «CounterFBD» в основном программном модуле, написанном на языке FBD.

Функция

Добавление пользовательской функционального блока происходит путем нажатия на пункт «Функция» во всплывающем меню дерева проекта. В диалоговом окне задайте имя функции в поле «Имя POU», в поле «Тип POU» выберите «функция», в поле «Язык» выберите язык, на котором будет написан алгоритм работы функции.

Создадим функцию «AverageVal» на языке ST, которая будет вычислять среднее значение счетчиков в цикле. Поскольку счетчики никак не синхронизированы, среднее значение должно быть дробным. Выберем тип возвращаемого значения функции – для дробного значения это тип REAL.

В панели редактирования переменных и констант добавим пять переменных «Cnt1»..«Cnt5» типа INT, класса «Вход». К этим входам будут подключены выходные значения функциональных блоков, рассмотренных выше. Добавим переменную «InputsNumber» типа REAL, класса «Локальный».

Далее в редакторе языка ST пишется алгоритм и логика работы данной функции, как показано на рис. 149:

Для приведения типа INT к типу REAL воспользуемся функцией INT_TO_REAL из библиотеки функций и функциональных блоков, она преобразует значение типа INT на входе IN в значение типа REAL на выходе OUT (INT:IN) =>(REAL:OUT).

Добавим функцию в основной программный модуль. На панели библиотеки функций и функциональных блоков в разделе «Пользовательские POU» необходимо выбрать функцию «AverageVal» и с помо-

CounterLDx

Описание: Счетчик на языке LD

Фильтр класса: Все

+

-

↑

↓

#	Имя	Класс	Тип	Исходное значение	Квалификатор	Описание
1	Reset	Вход	BOOL			Сброс счетчика
2	Cnt	Локальный	INT			Значение счетчика
3	ResetCounterValue	Внешний	INT		Константа	Значение счетчика после сброса
4	Out	Выход	INT			Возвращаемое значение ФБ

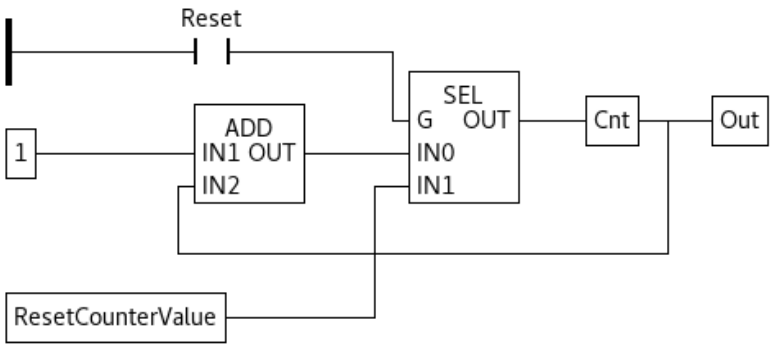


Рисунок 2.145: - Функциональный блок на языке LD

program0x

Описание: Основной программный модуль Фильтр класса: Все + - ↑ ↓

#	Имя	Класс	Тип	Адрес	Исходное значение	Квалификатор	Описание
8	CounterFBD0	Локальный	CounterFBD				Функциональный блок на языке FBD
9	CounterSFC0	Локальный	CounterSFC				Функциональный блок на языке SFC
10	CounterIL0	Локальный	CounterIL				Функциональный блок на языке IL
11	CounterLD0	Локальный	CounterLD				Функциональный блок на языке LD

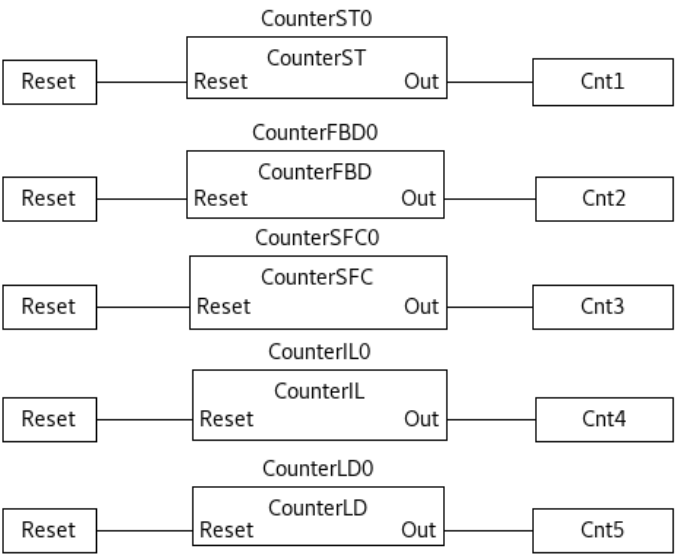


Рисунок 2.146: - Использование функционального блока на языке LD в основном программном модуле

Создать новый POU

Имя POU: AverageVal

Тип POU: функция

Язык: ST

Отменить OK

Рисунок 2.147: - Диалог создания функции

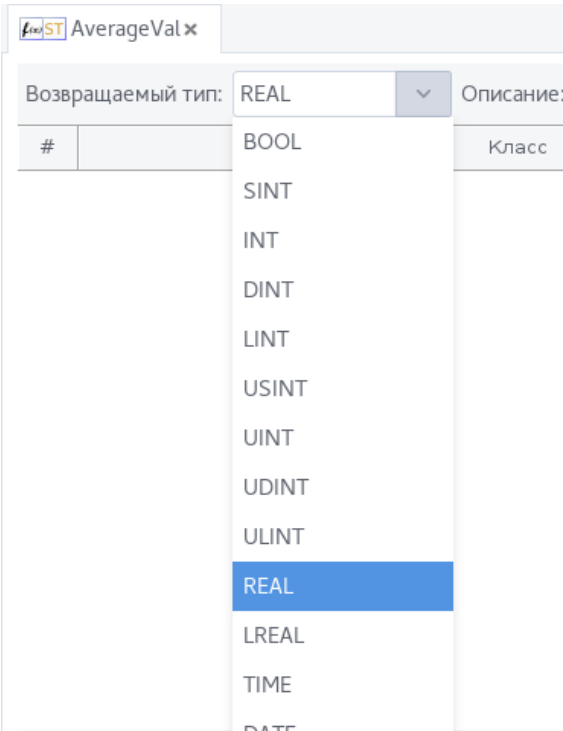


Рисунок 2.148: - Выбор типа возвращаемого значения функции

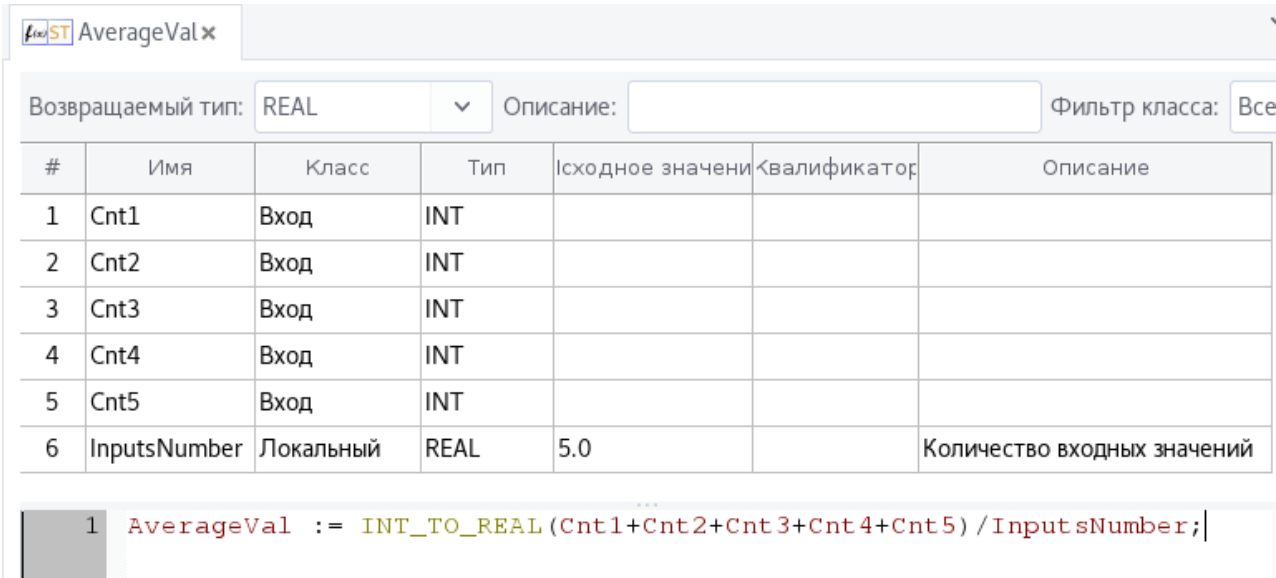


Рисунок 2.149: - Определение алгоритма и логики выполнения функции

щью указателя мыши (зажав левую кнопку мыши) перенести данную функцию (Drag&Drop) в область редактирования FBD диаграммы программного модуля «program0».



Рисунок 2.150: - Добавление на FBD диаграмму пользовательской функции

Подключим к входам функции значения пяти счетчиков. Для сохранения результата вычисления функции создадим переменную «AVCnt». На [Рисунок 2.151](#) показано использование созданной функции «AverageVal» в основном программном модуле, написанном на языке FBD.

Ресурс

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

Глобальные переменные ресурса

Глобальные переменные ресурса объявляются аналогично глобальным переменным проекта на панели переменных и констант выбранного ресурса с использованием кнопки «Добавить переменную», либо «Добавить переменные» (см. таблицу 3).

Использование данных глобальных переменных на уровне ресурса также аналогично использованию конфигурационных переменных проекта в программных модулях. Для использования в программном модуле глобальной переменной ресурса, добавьте в модуль переменную класса «Внешняя» с таким же именем, как у глобальной переменной, объявленные выше для ресурса.

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 кнопке «Добавить» на панели переменных и констант), необходимо задать её уникальное имя (поле «Имя») и выбрать тип выполнения задачи (поле «Запуск», см. [Рисунок 2.153](#)):

- «Циклический» – выполнение программного модуля типа «Программа» через заданный интервал времени, указанный в поле «Интервал»;

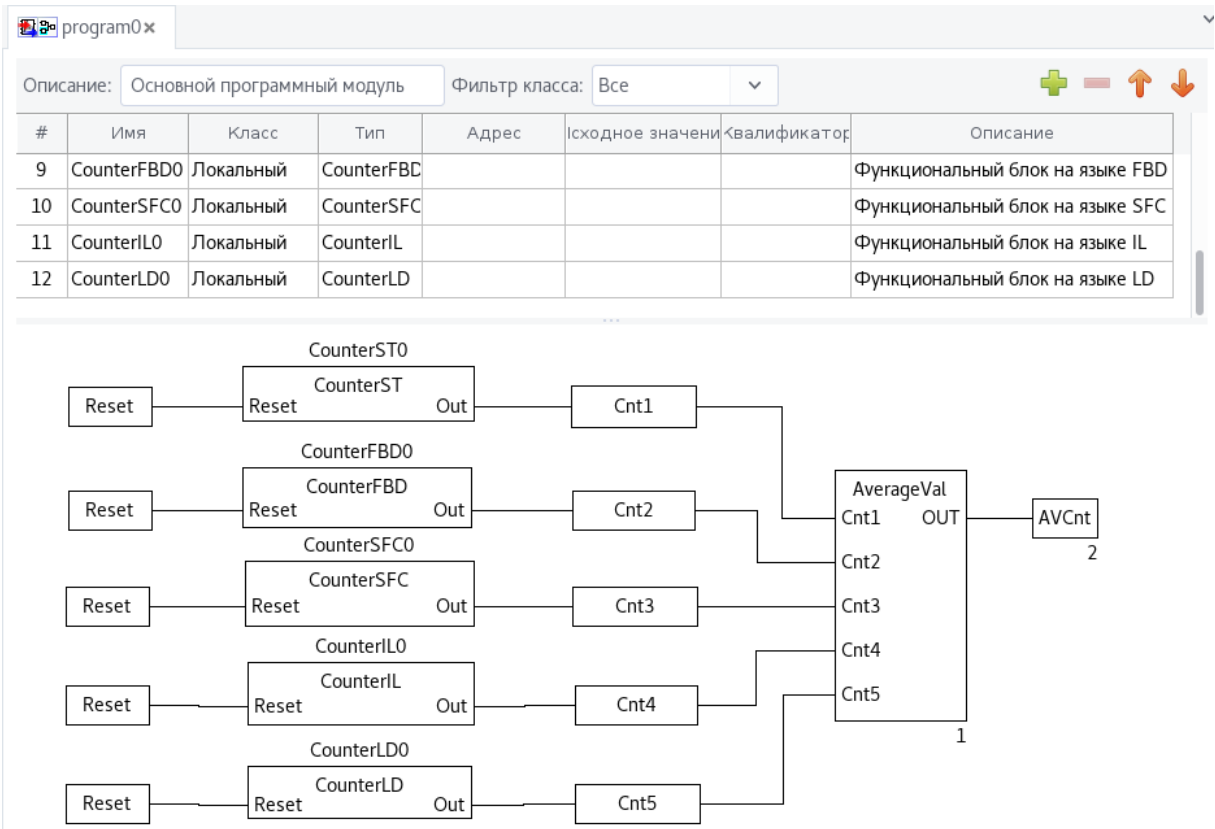


Рисунок 2.151: - Использование функции в основном программном модуле

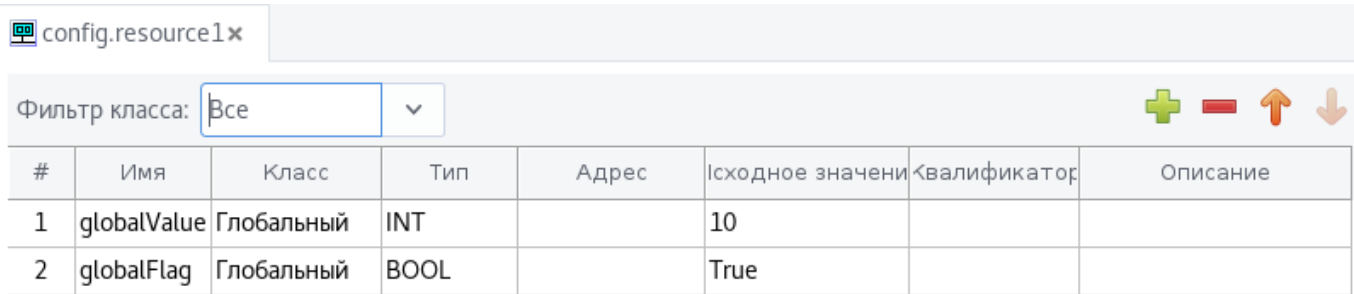


Рисунок 2.152: - Пример объявления в проекте глобальной переменной

- «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».

Задачи:

Имя	Запуск	Источник	Интервал	Приоритет
task0	Циклический			0

Прерывание
Циклический

Рисунок 2.153: - Выбор типа выполнения задачи

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

Задачи:

Имя	Запуск	Источник	Интервал	Приоритет
task0	Циклический		<input type="text"/> ...	0

Рисунок 2.154: - Добавление задачи с циклическим режимом выполнения

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

Редактировать длительность

Дни:	Часы:	Минуты:	Секунды:	Миллисекунды:	Микросекунды:
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="1"/>	<input type="text" value="100"/>	<input type="text" value="0"/>
				<input type="button" value="Отменить"/>	<input type="button" value="ОК"/>

Рисунок 2.155: - Диалог редактирования длительности задачи

Завершение ввода времени кнопкой «ОК» приводит к закрытию диалога и добавлению данного интервала времени в поле «Интервал» добавляемой задачи.

В случае выбора типа выполнения «Прерывание» в поле «Источник» необходимо указать переменную типа BOOL, определённую глобально либо на уровне проекта, либо на уровне ресурса. На [Рисунок 2.157](#) выбирается переменная «globalFlag», определённая в данном ресурсе.

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 зада-

Задачи:				
Имя	Запуск	Источник	Интервал	Приоритет
task0	Циклический		T#1s100ms ...	0

Рисунок 2.156: - Добавленный интервал выполнения

Задачи:				
Имя	Запуск	Источник	Интервал	Приоритет
task0	Прерывание	<div>▼</div>		0
		globalFlag		

Рисунок 2.157: - Выбор переменной типа BOOL как источника прерывания для начала выполнения задачи

чи имеют приоритет 0. Следует отметить, что в ресурсе должна быть определена как минимум одна задача с типом выполнения «Циклическое», в противном случае будет ошибка в компиляции в отладочной консоли. 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».

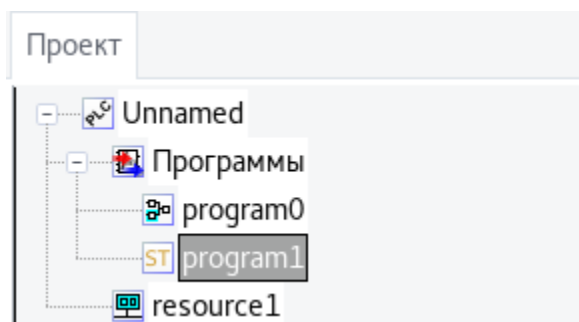


Рисунок 2.158: - Проект, содержащий два программных модуля типа “Программа”

Соответственно, при создании экземпляра в поле «Тип» оба этих программных модуля будут доступны (см. рис. 159).

Аналогичным образом выбирается задача из списка, в котором будут отображены определённые ранее задачи.

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

Задачи:

Имя	Запуск	Источник	Интервал	Приоритет
task0	Циклический		T#20ms	0
task1	Прерывание	globalFlag		0

Экземпляры:

Имя	Тип	Задача
instance0	<div>▼</div> <div>program0</div> <div>program1</div>	

Рисунок 2.159: - Выбор программного модуля типа “Программа” для экземпляра

Задачи:

Имя	Запуск	Источник	Интервал	Приоритет
task0	Циклический		T#20ms	0
task1	Прерывание	globalFlag		0

Экземпляры:

Имя	Тип	Задача
instance0	program1	<div>▼</div> <div>task0</div> <div>task1</div>

Рисунок 2.160: - Выбор задачи для экземпляра

Типы данных

Добавление типа данных происходит выбором пункта «Типа данных» в меню дерева проекта (см. [Рисунок 2.161](#)) в уже созданный проект, содержащий программный модуль типа «Программа» – «program0».

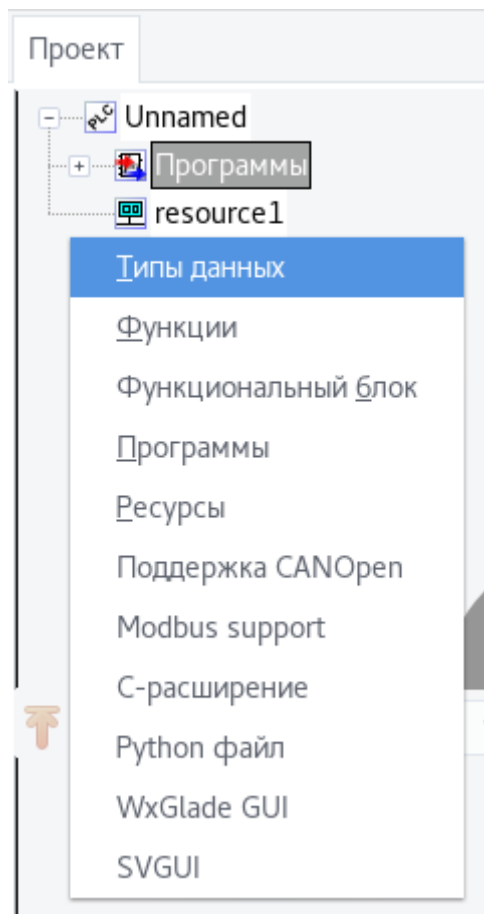


Рисунок 2.161: - Выбор пункта меню добавления пользовательского типа данных в дереве проекта

Будет создан массив типа INT размерностью 11 элементов. В дереве проекта появится панель редактирования добавленного типа данных с именем «datatype0». В поле «Механизм создания нового типа» необходимо выбрать «Массив» и указать тип INT, как показано на [Рисунок 2.162](#) :

С помощью кнопки «Добавить» (см. таблицу 10) создаётся поле для массива с указанием его размерности в соответствующем формате.

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

Сборка и передача на целевое устройство 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.

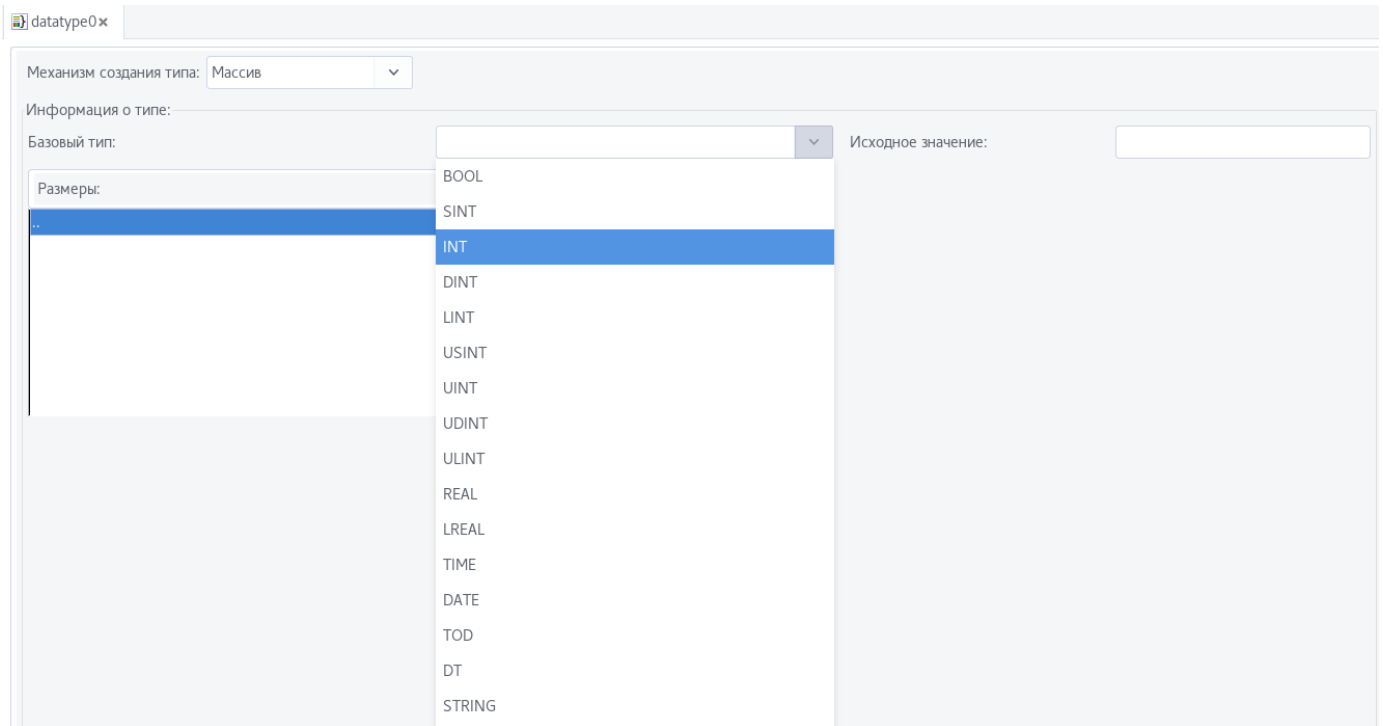


Рисунок 2.162: - Выбор базового типа для массива

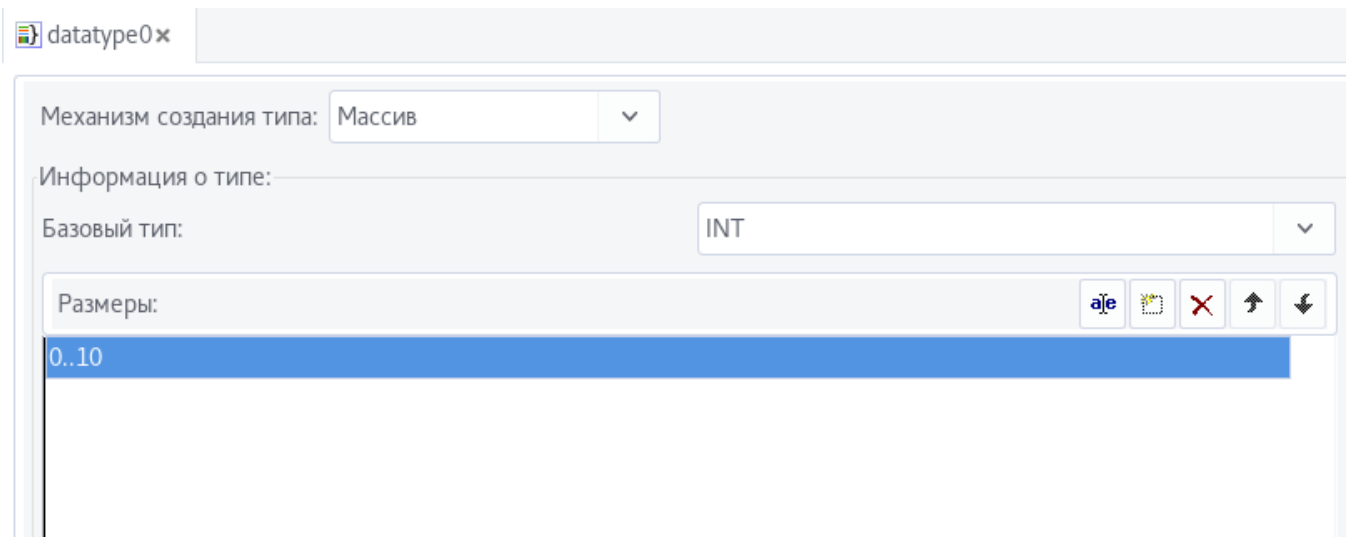


Рисунок 2.163: - Задание размерности для массива

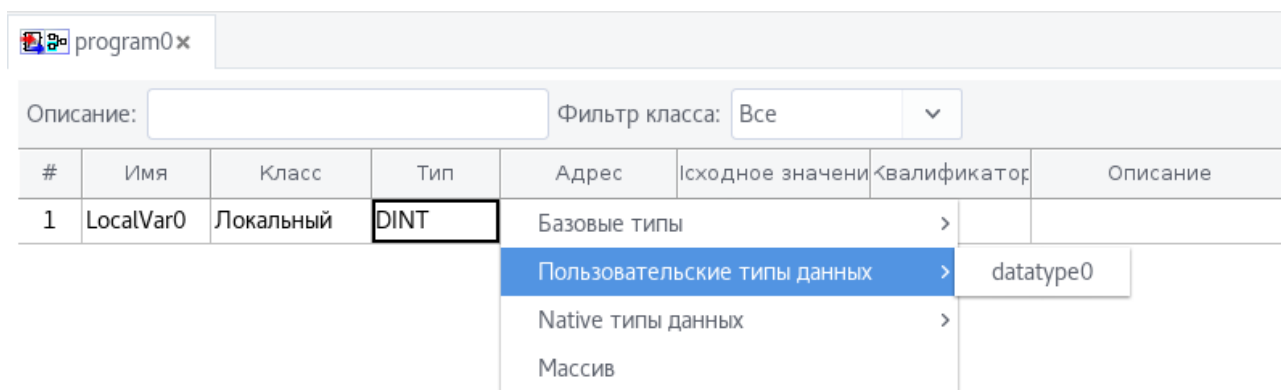


Рисунок 2.164: - Выбор добавленного типа данных в панели переменных и констант программного модуля

Сборка проекта осуществляется с помощью соответствующих кнопок, находящихся на панели инструментов. 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 запущенной на целевом устройстве серверной части среды разработки Beremiz.

Среда разработки Beremiz предоставляет следующие возможности отладки:

- Просмотр и изменение значения всех переменных проекта, используя панель отладки;
- Визуально отслеживание выполнения программ на графических языках и изменение значения различных графических элементов конкретного языка;
- Отображение значений переменных в виде графика.

Далее подробнее рассказывается про сборку прикладной программы, соединение с целевым устройством и передачу на него исполняемого файла и его отладке.

Сборка прикладной программы

Прежде чем проект будет передан ПЛК, его необходимо собрать. Настроить целевую платформу для сборки можно в окне настройки проекта, на вкладке Конфигурация.

Для сборки проекта нажмите кнопку «Сборка проекта в директории сборки» (см. табл. 2). Результаты сборки выводятся в консоль, расположенную в нижней части окна программы, ошибки сборки выделяются красным цветом. На примере проекта First_steps после сборки в консоль выведено сообщение о том, что сборка проведена успешно.

Пересборку проекта можно осуществить, очистив директорию сборки проекта нажатием на кнопку «Очистить директорию сборки проекта» (см. табл.2). Будет удален сгенерированный на языке ST код проекта и скомпилированный бинарный файл прошивки ПЛК. После этого нажмите кнопку «Сборка проекта в директории сборки», и проект будет собран заново.

```

Поиск  Консоль  Лог ПЛК
Сборка запущена в /home/sergey/PycharmProjects/Beremiz/Beremiz Projects/first_steps_with_func/build
Генерация МЭК-61131 ST/IL/SFC кода ПЛК...
Компиляция МЭК-программы в С-код...
Экспорт локальных переменных...
С-код успешно сгенерирован.
ПЛК:
  [CC]  plc_main.c -> plc_main.o
  [CC]  plc_debugger.c -> plc_debugger.o
  [CC]  config.c -> config.o
  [CC]  resource1.c -> resource1.o
Линковка:
  [CC]  plc_main.o plc_debugger.o config.o resource1.o -> first_steps_with_func.so
Сборка прошла успешно.

```

Рисунок 2.165: - Результаты сборки выведены в консоль

Запуск серверной части для отладки

Серверная часть среды разработки Beremiz, необходимая для передачи исполняемого файла на целевое устройство и его отладки, находится в сценарии на языке Python в файле Beremiz_service.py. Запуск данного файла осуществляется из командной строки, следующей командой с параметрами по умолчанию (см. таблицу 13):

```
$ python Beremiz_service.py
```

Также, при запуске могут быть указаны параметры, представленные в таблице 13.

Таблица 13 - Параметры командной строки запуска серверной части среды Beremiz

Параметр	Операция
-i	Указание IP адреса для обращения клиента, по умолчанию 127.0.0.1 (localhost)
-p	Номер порта, по умолчанию 3000
-h	Вывод в консоль справки по работе с данным сервером
-a	Автоматический запуск целевого устройства (0 – выключить, 1 – включить), по умолчанию 0.
-x	Включить/выключить иконку в панели задач (0 – выключить, 1 – включить), по умолчанию 0.
-t	Web-интерфейс на базе библиотеки Twisted (0 – выключить, 1 – включить), по умолчанию 0. Он позволяет отслеживать состояние выполнения программы через браузер. Адрес: <a href="http://<ip-адрес целевого устройства>:<8009>">http://<ip-адрес целевого устройства>:<8009> . 8009 – порт по умолчанию.

После указания всех параметров команды запуска серверной части Beremiz можно ввести адрес директории, в которой будут храниться файлы на файловой системе целевого устройства. По умолчанию этой директорией является временная папка, созданная для текущего запущенного экземпляра службы Beremiz_service.

Как правило, данный сценарий запускается в автоматическом режиме при включении целевого устройства, средствами операционной системы.

Соединение с целевым устройством и передача исполняемого файла

После того как скрипт серверной части среды разработки Beremiz запущен, можно производить соединение с целевым устройством. В панели настроек проекта необходимо указать URI-адрес целевого устройства:

PYRO://<IP-адрес>:<номер порт>

На [Рисунок 2.166](#) показан URI адрес целевого устройства LOCAL://, это адрес Soft PLC на локальной машине, и выделена красным цветом кнопка «Подключиться к целевому ПЛК» (см. таблицу 2), которая используется для соединения с целевым устройством.

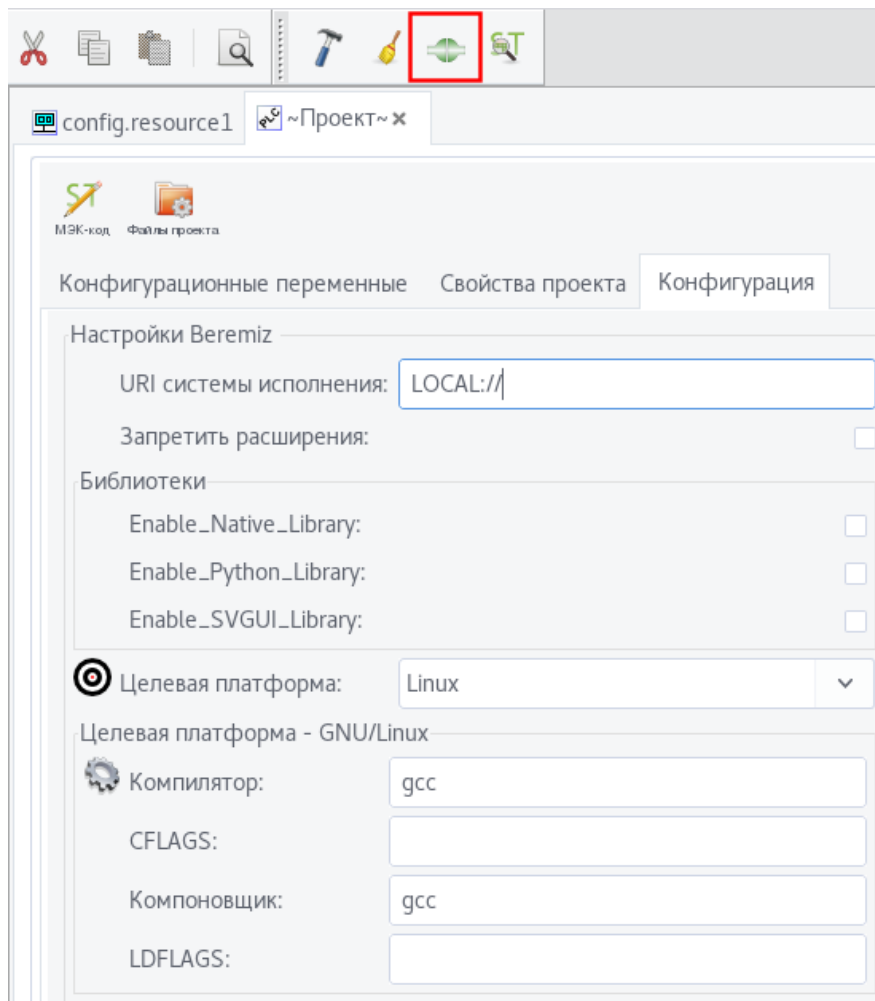


Рисунок 2.166: - Соединение с целевым устройством

В случае успешного соединения, в отладочной консоли будет выведено соответствующее сообщение и выведен статус прикладной программы.

Статусы прикладной программы могут быть следующие:

- «PLC Empty» – прикладная программа отсутствует;
- «PLC Started» – прикладная программа на целевом устройстве есть и она выполняется;
- «PLC Stopped» – прикладная программа на целевом устройстве есть, но остановлена.

Используя кнопки передачи, запуска и остановки прикладной программы на целевом устройстве (см. таблицу 2) можно передать исполняемый файл прикладной программы, запустить его и остановить. В отладочной консоли будут выведены соответствующие сообщения (после передачи, запуска и остановки прикладной программы), как показано на [Рисунок 2.168](#) :

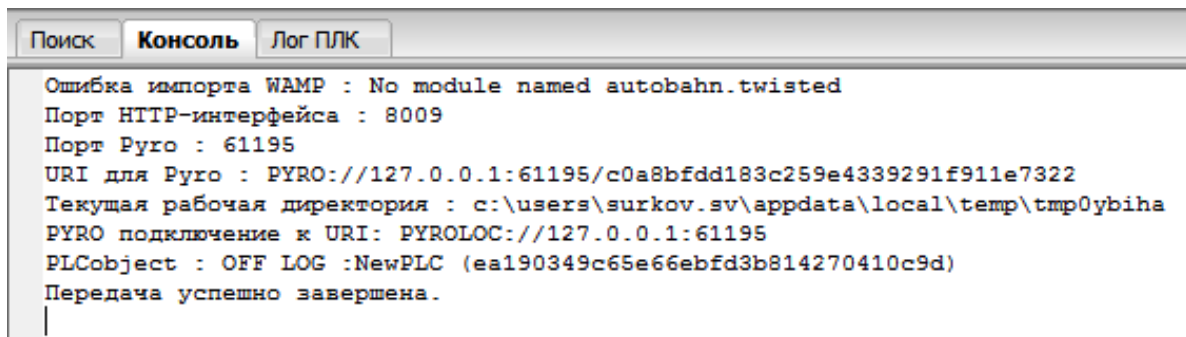


Рисунок 2.167: - Отладочная консоль после соединения с целевым устройством

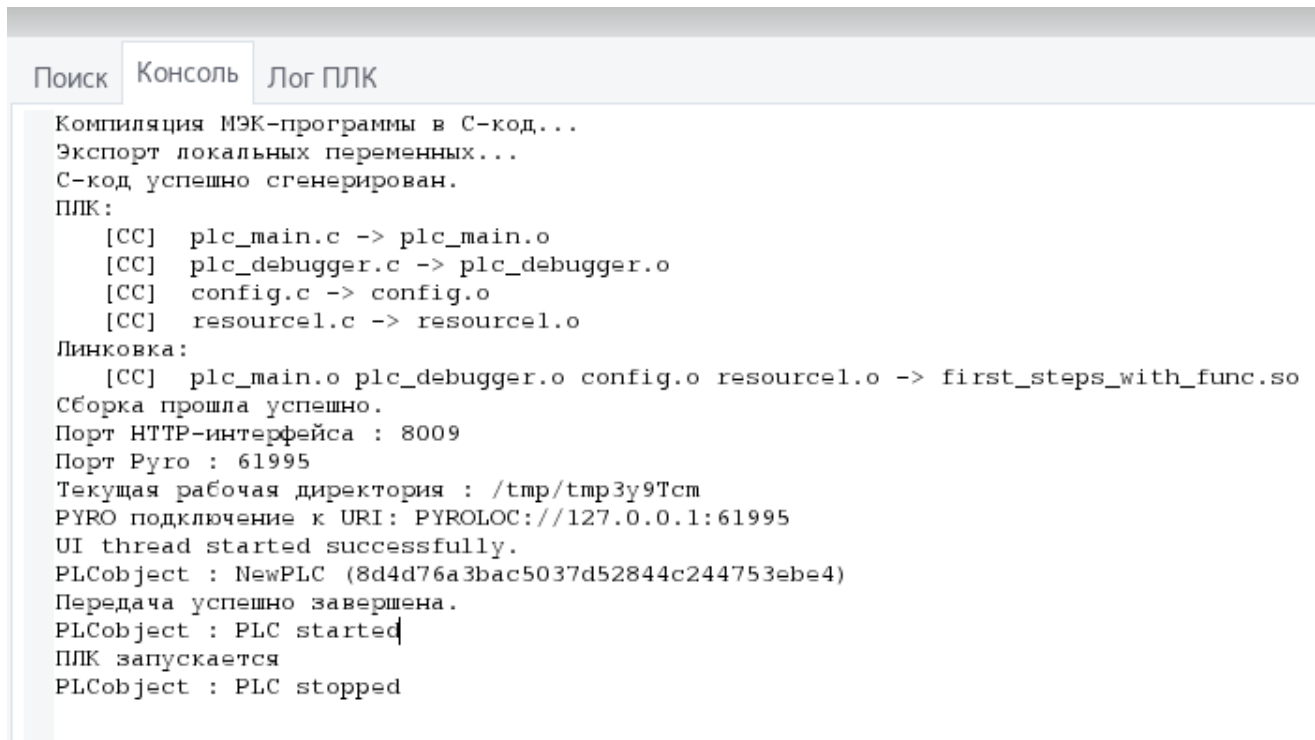


Рисунок 2.168: - Отладочная консоль после передачи прикладной программы, запуска и остановки

Отладка прикладной программы

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.

Retain переменные в прикладной программе

В SoftPLC реализована поддержка переменных, для которых в прикладной программе определено свойство ретанентности (Retain-переменные). При подключении к серверной части среды разработки Beremiz, создается временная папка для хранения прикладной программы, загружаемой в SoftPLC, Retain переменные хранятся в этой временной папке в виде бинарного файла, содержащего в себе хеш-сумму проекта, значения Retain переменных в бинарном формате, и контрольную сумму файла, вычисляемую по алгоритму CRC32.

Запуск отладчика

Осуществив передачу скомпилированной программы на SoftPLC, запустите отладку, нажав кнопку «Запустить ПЛК» (см. табл. 2). Переменные принимают исходные значения, затем начинается исполнение программы в ПЛК. Отладчик запущен.

Отладка текстовых языков

Работа с отладчиком подразумевает работу с экземпляром загруженной в ПЛК программы. Функциональные блоки, функции и переменные тоже имеют свои экземпляры, доступные для отладки только после того как программа будет запущена на исполнение. В левом нижнем углу основного окна среды разработки расположена панель экземпляров проекта. В адресной строке написан адрес ресурса, для которого ниже отображены экземпляры программ, глобальных переменных и функциональных блоков определенных в данном ресурсе.

Переход к родительскому экземпляру и его глобальным переменным осуществляется кнопкой «Родительский экземпляр» (см. табл. 11).

Кнопка «Отладка экземпляра» (см. табл. 11) напротив адресной строки в верхней части панели запускает отладку выбранного ресурса (программы или функционального блока). Для того чтобы включить отладку экземпляра блока или переменной, нажмите кнопку напротив этого элемента на панели. На панели отладки отобразятся текущие значения добавленных переменных.

После того как переменная выведена на панель отладки, для того чтобы установить значение нажмите кнопку «Форсировать значение».

В появившемся диалоге введите значение переменной (см. [Рисунок 2.172](#)). Для булевых переменных в диалоге присутствует кнопка «Переключить значение», которая меняет значение переменной на противоположное. После изменения значения переменной, она будет выделена синим цветом в таблице переменных и их значений во вкладке «Отладчик».

Форсируя значение переменной, вы устанавливаете неизменяемое значение, переопределение которого выполняемой программой будет невозможно. После установки значения, его можно освободить, дав возможность программе изменять значение переменной. Для освобождения переменной нажмите кнопку «Освободить значение».

Для корректного изменения, вводимое значение должно соответствовать типу переменной, иначе будет выведено сообщение об ошибке, как показано на [Рисунок 2.174](#) :

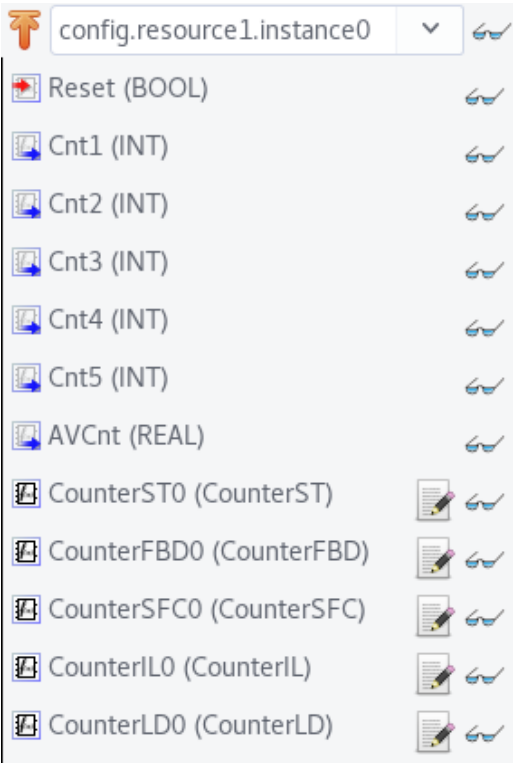


Рисунок 2.169: - Панель экземпляров проекта

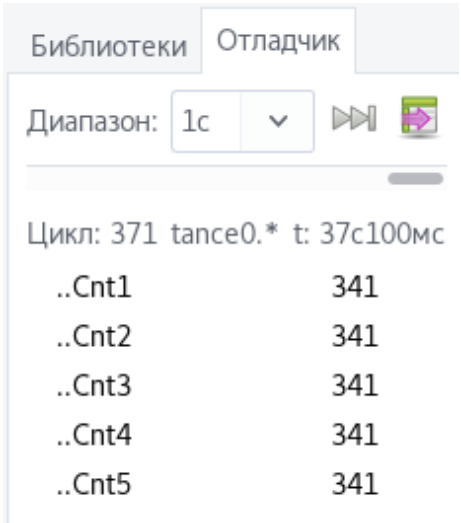


Рисунок 2.170: - Панель отладчика

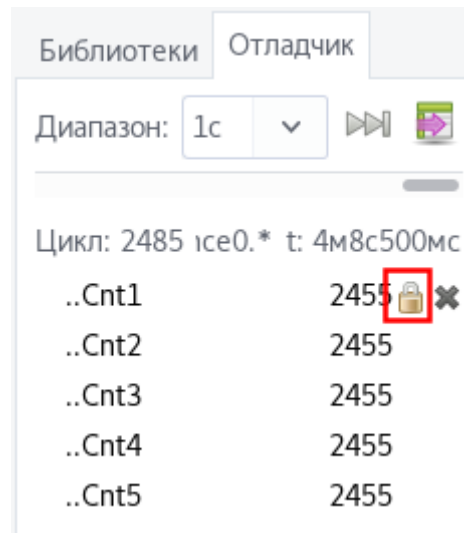


Рисунок 2.171: - Форсирование переменной в панели отладки

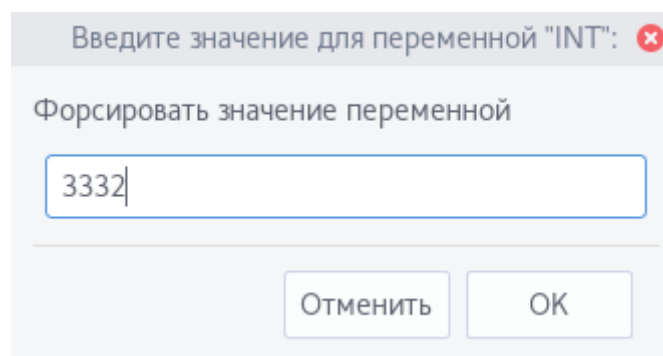


Рисунок 2.172: - Диалог установки значения

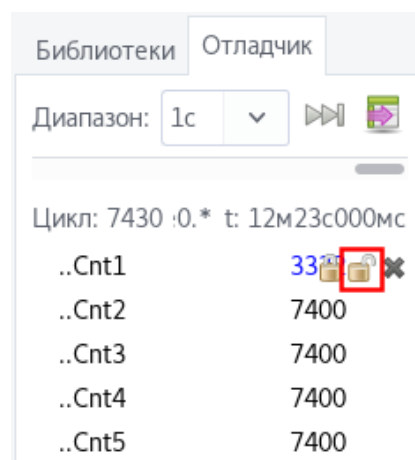


Рисунок 2.173: - Освобождение значения переменной

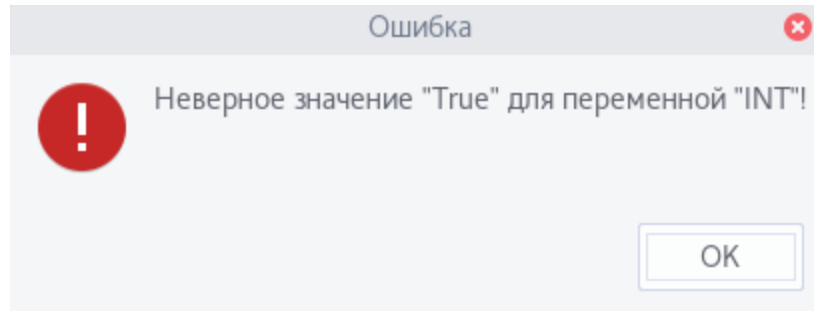


Рисунок 2.174: - Ошибка при вводе недопустимого значения изменяемой переменной в режиме отладки

Отладка графических языков

Во время отладки прикладной программы, в которой часть программных модулей написаны на графических языках, есть возможность видеть изменения всех значений на диаграмме и вносить необходимые изменения прямо на ней. Как уже упоминалось ранее, в случае нажатия кнопки запуска режима отладки (на [Рисунок 2.175](#) выделены красным цветом) для экземпляра программы, написанной на одном из графических языков, откроется вкладка с панелью диаграммы в режиме отладки.

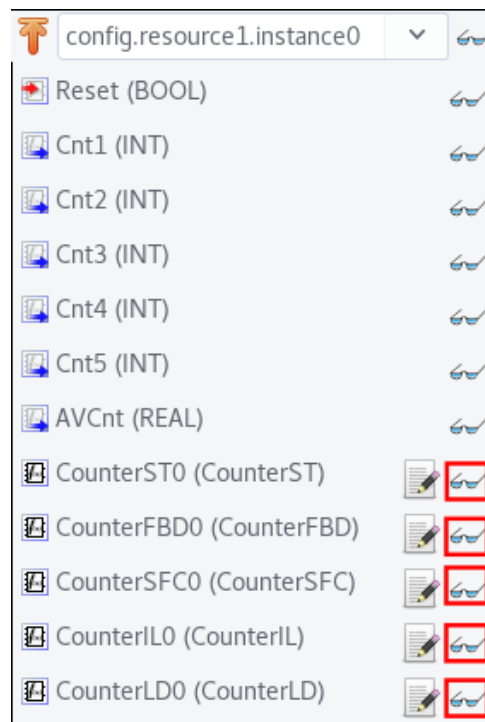


Рисунок 2.175: - Панель экземпляров проекта

Эти вкладки полностью повторяют те, в которых графические диаграммы программ или функциональных блоков редактируются, за исключением того что во вкладках отладчика невозможно внести какие-либо изменения, а связи между элементами выделяются разным цветом в зависимости от значения переменной, передаваемого по этой связи.

Линии, не выделенные цветом передают либо булево значение False, либо переменную не булевого типа (INT, DINT, WORD, REAL, TIME, и т.д.). Оранжевого цвета связи, передающие константное

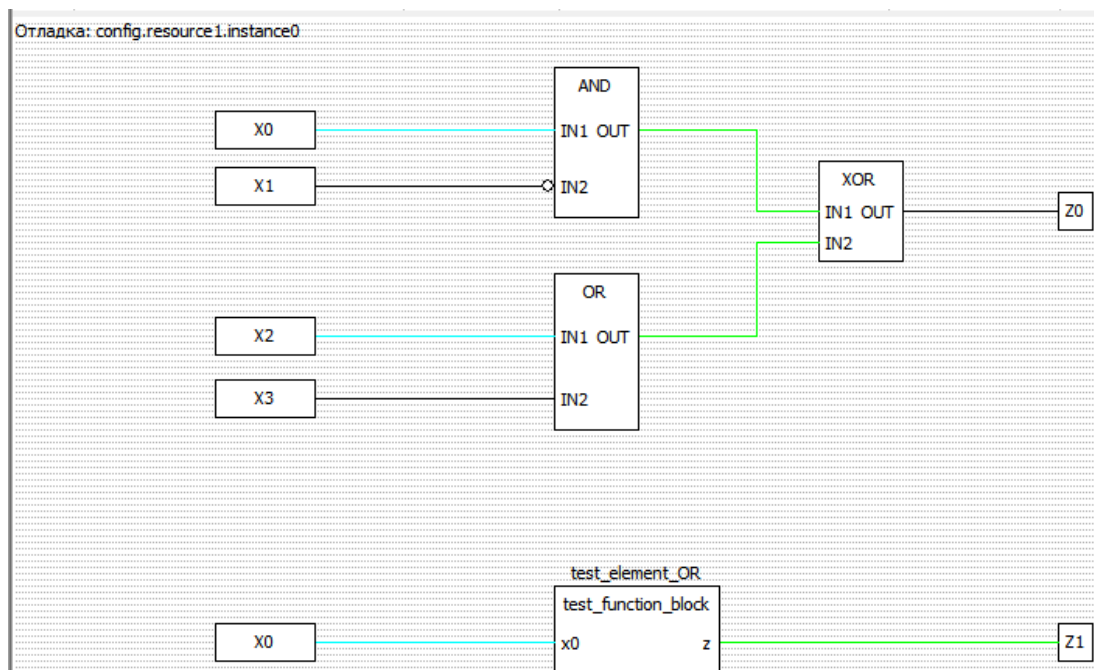


Рисунок 2.176: - Пример отлаживаемой FBD диаграммы

выражение. Светло-зелёным цветом выделены связи, которые передают булево значение True. Связи, выделенные светло-голубым и тёмно-синим цветом передают значения непосредственно установленные пользователем, значение True соответствует светло-голубому цвету, False - тёмно-синему.

Отладка FBD диаграммы

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

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

Отладка LD диаграммы

Отладка LD диаграммы осуществляется аналогично отладке FBD диаграммы. Для вызова всплывающего меню (см. рис. 196), в котором можно установить желаемое значение для контакта или катушки необходимо нажать правую клавишу мыши.

Появится диалог (см. [Рисунок 2.179](#)), в котором нужно ввести значение типа BOOL: TRUE – контакт «ON», FALSE – контакт «OFF».

Отладка SFC диаграммы

Отладка SFC диаграммы происходит аналогично отладке диаграмм FBD и LD. С помощью всплывающего меню (см. [Рисунок 2.180](#)), есть возможность устанавливать активность для шагов и переходов.

На [Рисунок 2.181](#) показано, как устанавливается значение (после выбора «Форсировать значение», появится диалог) TRUE для шага «ResetCounter».

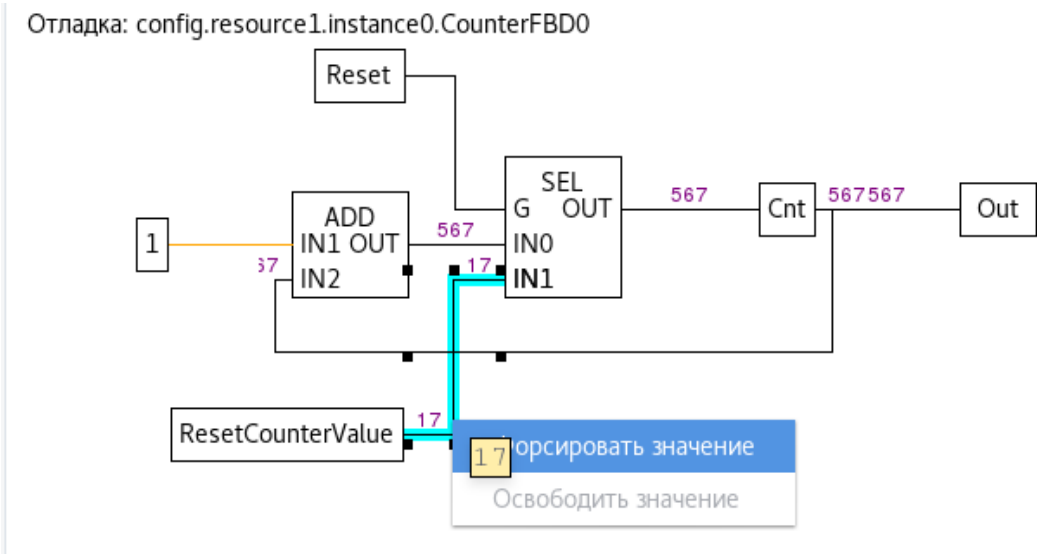


Рисунок 2.177: - Отладка FBD диаграммы

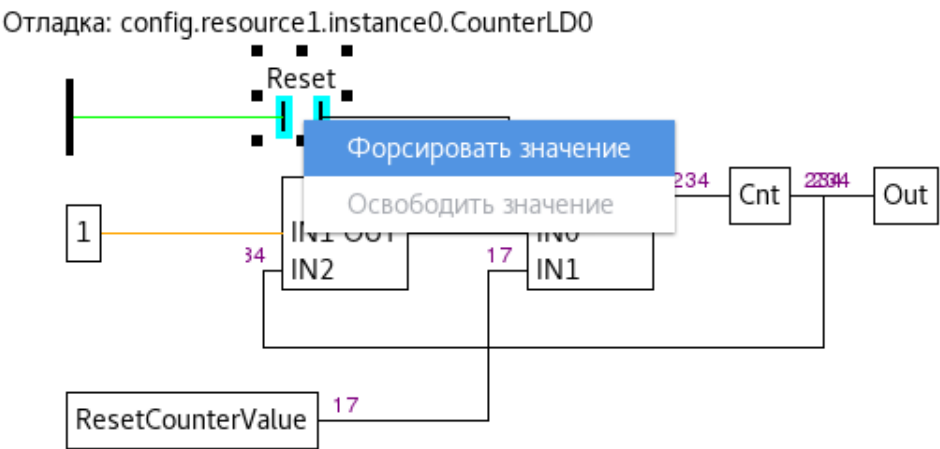


Рисунок 2.178: - Пример отладки LD диаграммы

Введите значение для переменной "BOOL": ❌

Форсировать значение переменной

False

Переключить значение

Отменить OK

Рисунок 2.179: - Диалог переключения состояния контакта

Отладка: config.resource1.instance0.CounterSFC0

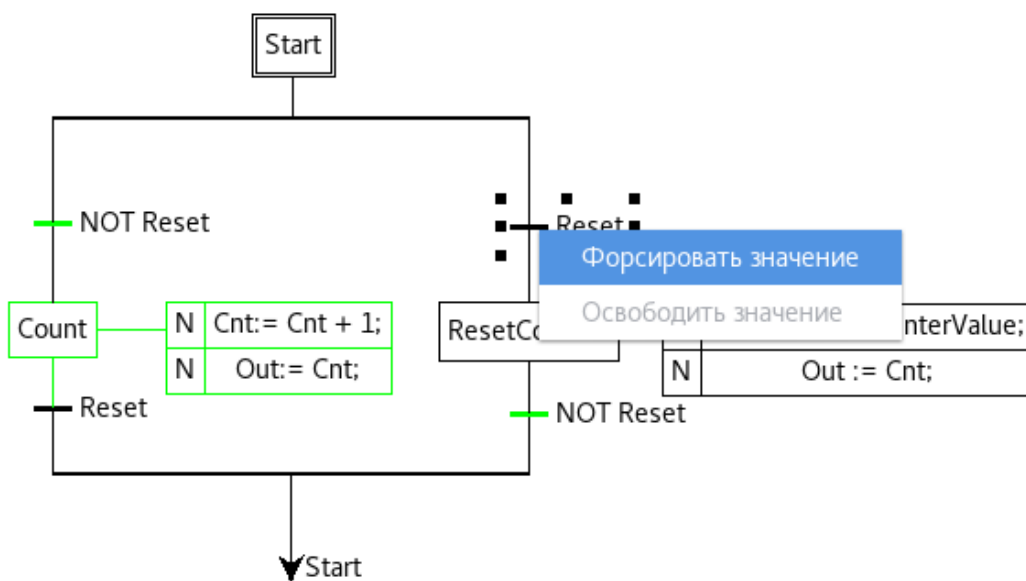


Рисунок 2.180: - Пример отладки SFC диаграммы

Отладка: config.resource1.instance0.CounterSFC0

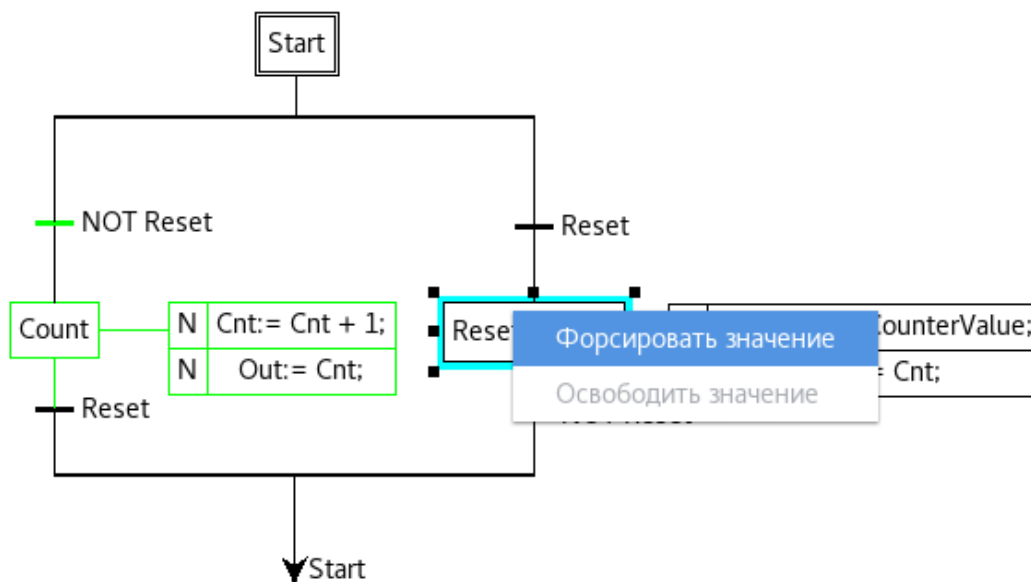


Рисунок 2.181: - Изменение состояния шага "ResetCounter"

После установки значения шага в TRUE с помощью режима отладки, шаг будет выделен голубым цветом. Как можно заметить по [Рисунок 2.182](#), т.к. шаг «ResetCounter» стал активным, блок действий, ассоциированный с ним, так же стал активным (выделен зелёным цветом), а действия внутри него, в данном случае присвоение переменной «Cnt» значения конфигурационной переменной «ResetCounterValue», а переменной «Out» значения переменной «Cnt»:

```
Cnt := ResetCounterValue;
```

```
Out := Cnt;
```

стало выполняться.

Отладка: config.resource1.instance0.CounterSFC0

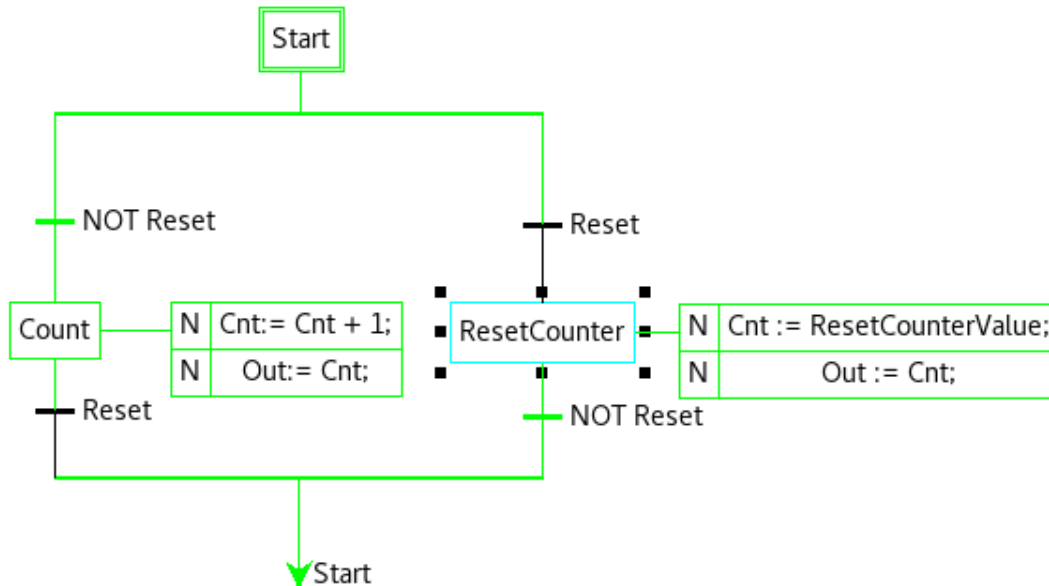


Рисунок 2.182: - Форсирование блока действия

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

График изменения значений переменной

Среда разработки Beremiz так же позволяет отображать в виде графика изменение значения переменной в режиме отладки. Для вывода панели с графиком, необходимо два раза кликнуть левой кнопкой мыши по кнопке «Отладка экземпляра» напротив переменной в панели экземпляров проекта.

Появившаяся панель графика изменения переменной позволяет отслеживать то, как значение определённой переменной изменяется в течение времени.

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

2.2.5 Описание библиотеки функций и функциональных блоков

Функции и функциональные блоки представляют собой предопределённые элементы, которые могут быть использованы при написании алгоритмов и логики программных модулей типа «Функциональный блок» и «Программа», 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.

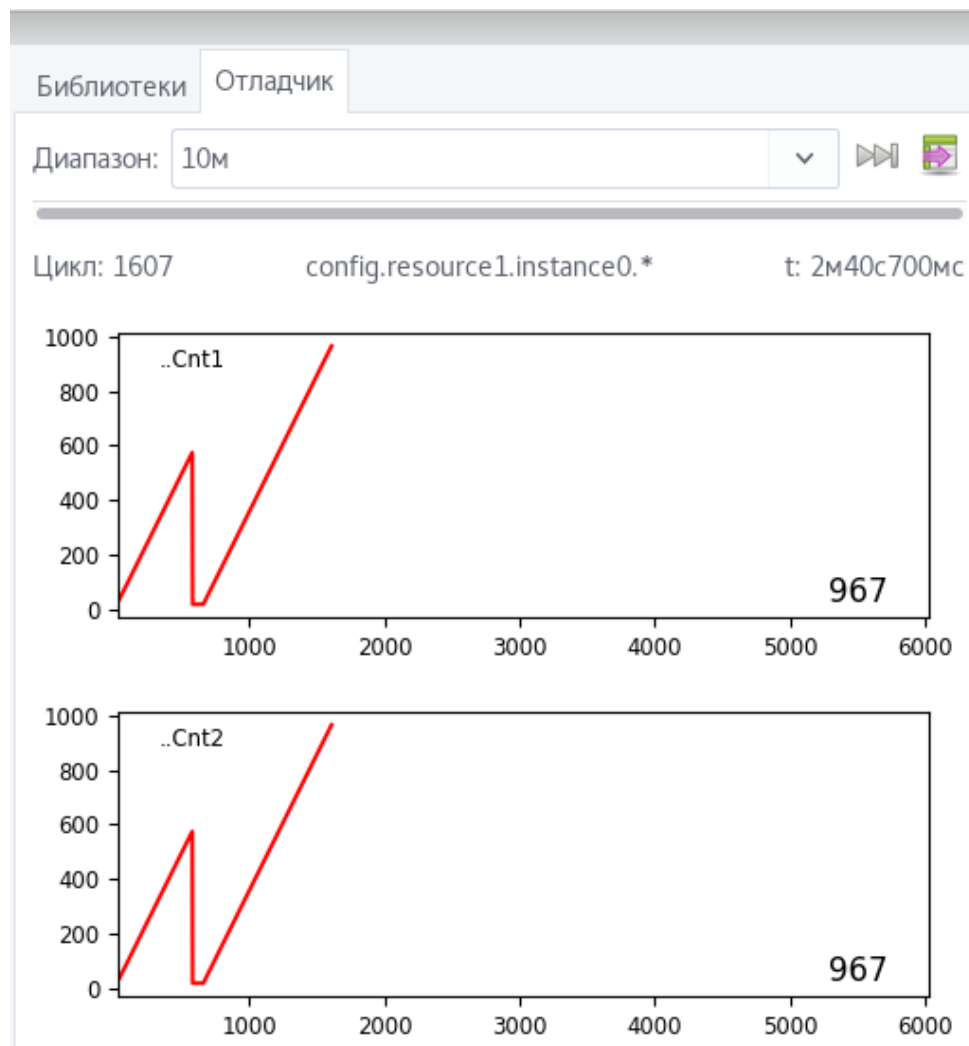


Рисунок 2.183: - График изменения переменных

Стандартные функциональные блоки

Бистабильный SR-триггер

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

Бистабильный RS-триггер

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

SEMA - Семафор

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

R_TRIG - Индикатор нарастания фронта

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

F_TRIG - Индикатор спада фронта

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

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

CTU - инкрементный счётчик

Данный функциональный блок представляет собой инкрементный счётчик. Сигнал “1” на входе R вызывает присваивание значения “0” выводу 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.

CTD - декрементный счётчик

Данный функциональный блок представляет собой декрементный счётчик. Сигнал “1” на входе LD вызывает присваивание значения на входе PV выводу 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.

CTUD - реверсивный счётчик

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

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

Когда $CV \geq PV$, 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.

TP - повторитель импульсов

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

TON - таймер с задержкой включения

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

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

TOF - таймер с задержкой отключения

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

Если IN становится “1”, Q становится “1”.

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

Если внутреннее время достигает значения PT, Q становится “0”.

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

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

Дополнительные функциональные блоки**RTC - часы реального времени**

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

Вход PDT (Preset DT) предназначен для установки времени. Часы начинают отсчет времени от значения PDT. Выход Q (BOOL) 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 входных параметров XIN1 и XIN2 (типа REAL с плавающей точкой).

Числовые операции

ABS - модуль числа

Данная функция возвращает в OUT модуль входного числа IN.

SQRT - квадратный корень

Данная функция возвращает в OUT квадратный корень входного числа IN.

LN - натуральный логарифм

Данная функция возвращает в OUT значение натурального логарифма от IN.

LOG - логарифм по основанию 10

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

EXP - возведение в степень экспоненты

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

SIN - синус

Данная функция возвращает в OUT значение синуса IN.

COS - косинус

Данная функция возвращает в OUT значение косинуса IN.

TAN - тангенс

Данная функция возвращает в OUT значение тангенса IN.

ASIN - арксинус

Данный функциональный блок возвращает в OUT значение арксинуса IN.

ACOS - арккосинус

Данная функция возвращает в OUT значение арккосинуса IN.

ATAN - арктангенс

Данная функция возвращает в OUT значение арктангенса IN.

Арифметические операции

ADD - сложение

Данная функция возвращает в OUT результат сложения IN1 и IN2.

MUL - умножение

Данная функция возвращает в OUT результат умножения IN1 и IN2.

SUB - вычитание

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

DIV - деление

Данная функция возвращает в OUT результат деления IN1 на IN2.

MOD - остаток от деления

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

EXPT - возведение в степень

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

MOVE - присвоение

Данная функция возвращает в OUT значение IN.

Временные операции

ADD_TIME - сложение переменных типа TIME

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

ADD_TOD_TIME - сложение времени дня TOD с интервалом времени TIME

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

ADD_DT_TIME - прибавление промежутка времени TIME к моменту времени DT

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

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

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

MULTIME - умножение времени TIME на число

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

SUB_TIME - разность двух значений типа TIME

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

SUB_DATE_DATE - разность двух значений типа DATE

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

SUB_TOD_TIME - вычитание из времени дня TOD интервала времени TIME

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

SUB_DT_TIME - вычитание из момента времени DT промежутка времени TIME

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

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

DIVTIME - деление времени TIME на число

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

Операции смещения бит

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.

Побитовые операции

AND - побитовое И

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

OR - побитовое ИЛИ

Данная функция представляет собой организацию «логического ИЛИ» для всех входных аргументов $IN_1...IN_n$.

XOR - побитовое исключающее ИЛИ

Данная функция представляет собой организацию «логического исключающего ИЛИ» для всех входных аргументов $IN_1...IN_n$.

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 IN_1 или IN_2 в зависимости от значения аргумента G . Если $G = 0$, то OUT равно X_1 , иначе - OUT равно X_2 .

MAX - максимум

Данная функция возвращает в OUT максимум из входных аргументов IN_1 и IN_2 .

MIN - минимум

Данная функция возвращает в OUT минимум из входных аргументов IN_1 и IN_2 .

LIMIT - ограничитель значения

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

MUX - Мультиплексор (выбор 1 из N)

Данная функция возвращает в OUT значение на входе $IN(K)$, в зависимости от входного K . Количество входов $IN:sub:(n)$ 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: $(IN_1 > IN_2) \& (IN_2 > IN_3) \& \dots (IN_{n-1} > IN_n)$, 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: $(IN_1 \geq IN_2) \& (IN_2 \geq IN_3) \& \dots (IN_{n-1} \geq IN_n)$, 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: $(IN_1 = IN_2) \& (IN_2 = IN_3) \& \dots (IN_{n-1} = IN_n)$, 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: $(IN_1 < IN_2) \& (IN_2 < IN_3) \& \dots (IN_{n-1} < IN_n)$, 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: $(IN1 \leq IN2) \& (IN2 \leq IN3) \& \dots (IN_{n-1} \leq IN_n)$, 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: $(IN1 <> IN2) \& (IN2 <> IN3) \& \dots (IN_{n-1} <> IN_n)$, 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.

2.3 Языки стандарта МЭК 61131-3

МЭК 61131-3 - раздел международного стандарта МЭК 61131, описывающий языки программирования для программируемых логических контроллеров.

2.3.1 Общие сведения о языке ST

Содержание

- *Общие сведения о языке ST*
 - *Типы данных*
 - *Конструкции языка*
 - *Арифметические операции*
 - * *Логические (битовые) операции*
 - * *Операции сравнения*
 - * *Присвоение*
 - * *Цикл FOR*
 - * *Цикл WHILE*
 - * *Цикл REPEAT UNTIL*
 - * *Конструкция CASE*

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

введены в символах верхнего регистра. Пробелы и метки табуляции не влияют на синтаксис, они могут использоваться везде.

Выражения в ST выглядят точно также, как и в языке Pascal:

```
[variable] := [value];
```

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

Типы данных

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

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

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

```
strA := strB;
```

Для работы со строками есть стандартный набор функций (см. приложение 2, раздел «Строковые операции с переменными типа STRING»).

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

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

Таблица 3.1 – Значения по умолчанию для типов данных IEC 61131-3

Тип(ы) данных	Значение
BOOL, SINT, INT, DINT, LINT	0
USINT, UINT, UDINT, ULINT	0
BYTE, WORD, DWORD, LWORD	0
REAL, LREAL	0.0
TIME	T#0S
DATE	D#0001-01-01
TIME_OF_DAY	TOD#00:00:00
DATE_AND_TIME	DT#0001-01-01-00:00:00
STRING	“ (пустая строка)

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

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

В определённых ситуациях при разработке программных модулей удобно использовать обобщения типов, т.е. общее именование группы типов данных. Данные обобщения приведены в таблице 3.2.

Таблица 3.2 – Обобщения типов данных IEC 61131-3

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

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 выполняется операция).

Логические (побитовые) операции

К данным операциям относятся:

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

Операции сравнения

Поддерживаются следующие операции сравнения:

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

В качестве результата сравнения всегда используется значение типа BOOL.

Присвоение

Для обозначения присвоения используется парный знак «:=». В правой и левой части выражения должны быть операнды одного типа (автоматического приведения типов не предусмотрено). В левой части выражения (принимаящая 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
Сравнения	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 описании 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. Строка 8 выполнится, если значение k будет равно 6 или будет принадлежать числовому отрезку $[9, 20]$.

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

При задании списка значений необходимо выполнять следующие условия:

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

В таблице 3.5 приведены примеры кода записи правильной и неправильной записи конструкции CASE.

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

Таблица 3.5 – Запись конструкции CASE

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

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

2.3.2 Общие сведения о языке IL

Содержание

- *Общие сведения о языке IL*
 - *Операторы языка*
 - *Пример программы на 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 загружаются 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 1 – Операторы языка IL

Оператор	Описание
LD	Загрузить значение операнда в аккумулятор
LDN	Загрузить обратное значение операнда в аккумулятор
ST	Присвоить значение аккумулятора операнду
STN	Присвоить обратное значение аккумулятора операнду
S	Если значение аккумулятора TRUE, установить логический операнд
R	Если значение аккумулятора FALSE, сбросить логический операнд
AND	Поразрядное И аккумулятора и операнда
ANDN	Поразрядное И аккумулятора и обратного операнда
OR	Поразрядное ИЛИ аккумулятора и операнда
ORN	Поразрядное ИЛИ аккумулятора и обратного операнда
XOR	Поразрядное разделительное ИЛИ аккумулятора и операнда
XORN	Поразрядное разделительное ИЛИ аккумулятора и обратного операнда
NOT	Поразрядная инверсия аккумулятора
ADD	Сложение аккумулятора и операнда, результат записывается в аккумулятор
SUB	Вычитание операнда из аккумулятора, результат записывается в аккумулятор
MUL	Умножение аккумулятора на операнд, результат записывается в аккумулятор
DIV	Деление аккумулятора на операнд, результат записывается в аккумулятор
GT	Значение аккумулятора сравнивается со значением операнда(>(greater than)). Значение (TRUE или FALSE)
GE	Значение аккумулятора сравнивается со значением операнда(>=greater than or equal). Значение (TRUE или FALSE)
EQ	Значение аккумулятора сравнивается со значением операнда(=(equal)). Значение (TRUE или FALSE)
NE	Значение аккумулятора сравнивается со значением операнда(<>(not equal)). Значение (TRUE или FALSE)
LE	Значение аккумулятора сравнивается со значением операнда(<=(less than or equal to)). Значение (TRUE или FALSE)
LT	Значение аккумулятора сравнивается со значением операнда(<(less than)). Значение (TRUE или FALSE)
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 программно или функционального блока
CALC	Вызов программно или функционального блока при условии, что значение аккумулятора TRUE
CALCN	Вызов программно 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 рис. 1 приведён пример программы на языке IL, которая эквивалентна следующему логическому выражению $C = A \text{ AND NOT } B$:

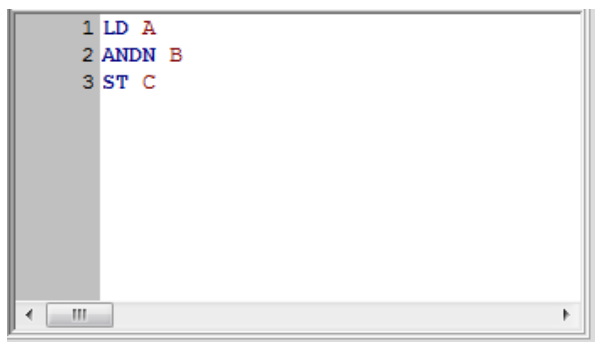


Рис. 1 – Пример программы на языке IL

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

2.3.3 Общие сведения о языке LD

Содержание

- *Общие сведения о языке LD*

LD (Ladder Diagram) – графический язык, основанный на принципах релейно-контактных схем (элементами релейно-контактной логики являются: контакты, обмотки реле, вертикальные и горизонтальные перемычки и др.) с возможностью использования большого количества различных функциональных блоков. Достоинствами языка LD являются: представление программы в виде электрического потока (близко специалистам по электротехнике), наличие простых правил, использование только булевых выражений. На рис. 6.1 приведён пример программы на языке LD (слева) и ее эквивалент в виде электрической цепи с реле и выключателями (справа).

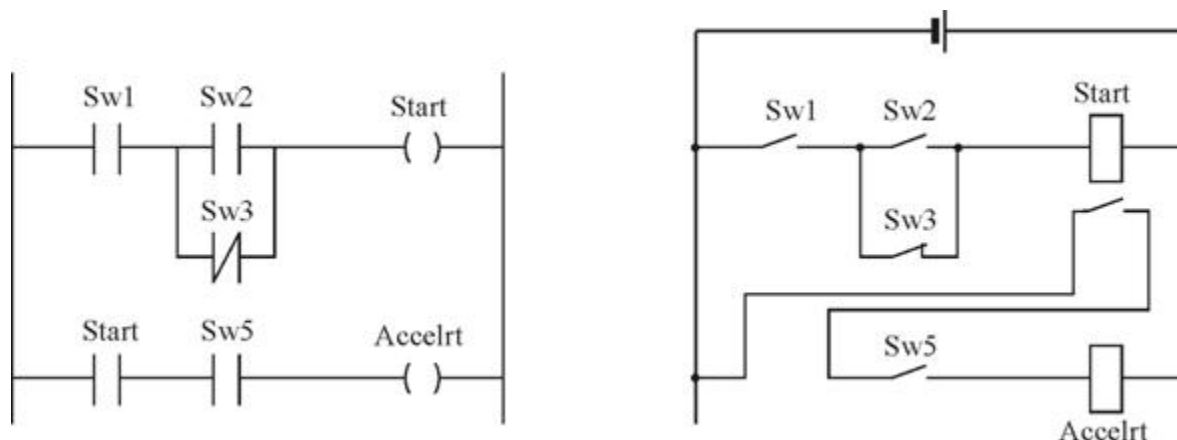


Рис. 6.1 – Программа на языке LD (слева) и ее эквивалент в виде электрической (справа)

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 передает «логическое И».

Контакт может быть инвертируемым. Такой контакт обозначается с помощью символа $|/|$ и передает состояние «ON», если значение переменной FALSE.

Язык LD позволяет:

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

Контакт

Контактом является LD-элемент, который передает 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 (см. рис. 6.2) состояние левой связи передается в правую связь, если состояние связанного логического фактического параметра TRUE. Иначе, состояние правой связи FALSE.

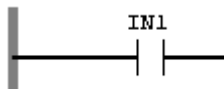


Рис. 6.2 – Нормальный контакт

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

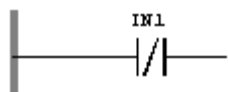


Рис. 6.3 – Инверсный контакт

В контактах для обнаружения нарастания фронта (см. рис 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.

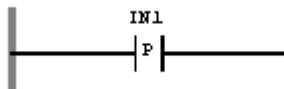


Рис. 6.4 – Контакт для обнаружения нарастания фронта

В контактах для обнаружения спада фронта (см. рис. 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.

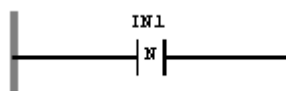


Рис. 6.5 – Контакт для обнаружения спада фронта

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

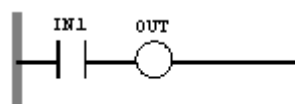


Рис. 6.6 – Нормальная катушка

В инвертирующей катушке (см. рис. 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.

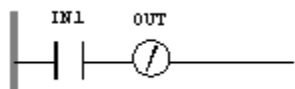


Рис. 6.7 – Инвертирующая катушка

В катушке установки (см. рис. 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.

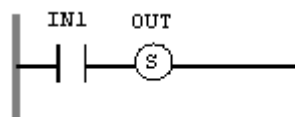


Рис. 6.8 – Катушка установки

В катушке сброса (см. рис. 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.

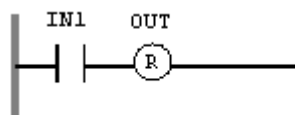


Рис. 6.9 – Катушка сброса

В катушке обнаружения нарастания фронта (см. рис. 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.

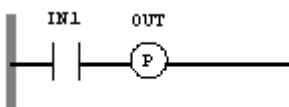


Рис. 6.10 – Катушка обнаружения нарастания фронта

В катушке обнаружения спада фронта (см. рис. 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.

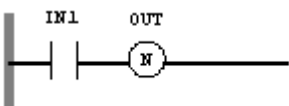


Рис. 6.11 – Катушка обнаружения спада фронта

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

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 A (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 (см. рис. 6.12), и результатом является программа, написанная на языке LD, реализующая логическое выражение:

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

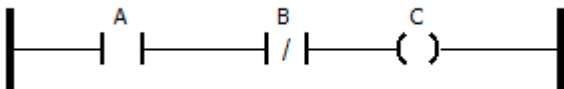


Рис. 6.12 – Пример LD диаграммы, реализующей логическое выражение $C = A \text{ AND NOT } B$

2.3.4 Общие сведения о языке SFC

Содержание

- *Общие сведения о языке SFC*
 - *Основные понятия языка SFC*
 - * Шаг
 - * Переход
 - * Блок действий
 - * «Прыжок» – переход на произвольный шаг
 - * Дивергенция и конвергенция
 - *Пример программы на 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 (см. рис. 7.1).

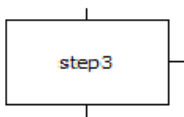


Рис. 7.1 – Графическое представление «Шага» языка SFC

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 (см. рис. 7.2).

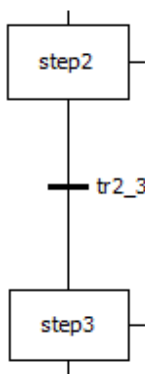


Рис. 7.2 – Шаги «step2» и «step3», соединённые переходом «tr2_3»

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:

$(i \leq 100) \text{ AND } b$

либо на любом другом языке.

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

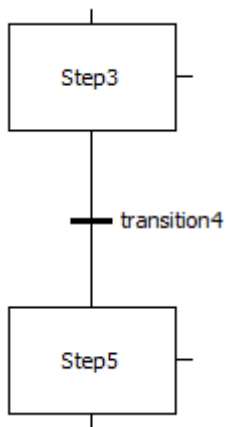


Рис. 7.3 – Переход между шагами «Step3» и «Step5» с предопределённым условием «transition4»

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

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

На рис. 7.4 представлен переход между шагами «Step6» и «Step7» в виде обычного условия:

level > 10

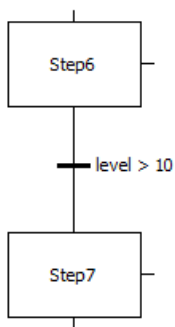


Рис. 7.4 – Переход между шагами «step6» и «step7» с предопределённым условием «transition4»

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

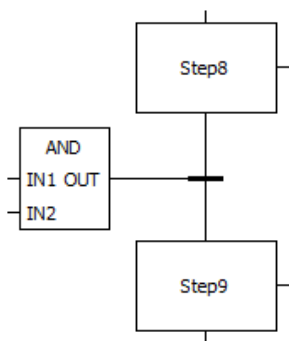


Рис. 7.5 – Переход между шагами «step8» и «step9», заданный «логическим И» на языке FBD

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

Блок действий

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

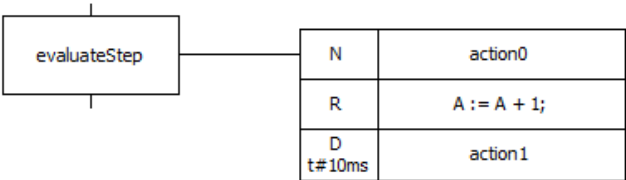


Рис. 7.6 – Шаг «evaluateStep» и связанный с ним блок действий, содержащий 3 действия

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 predetermined действий. Каждому predetermined действию присваивается имя (на рис. 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, описание которых приведено в таблице 6. 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 рис. 7.7 показана SFC диаграмма, содержащая два «прыжка».

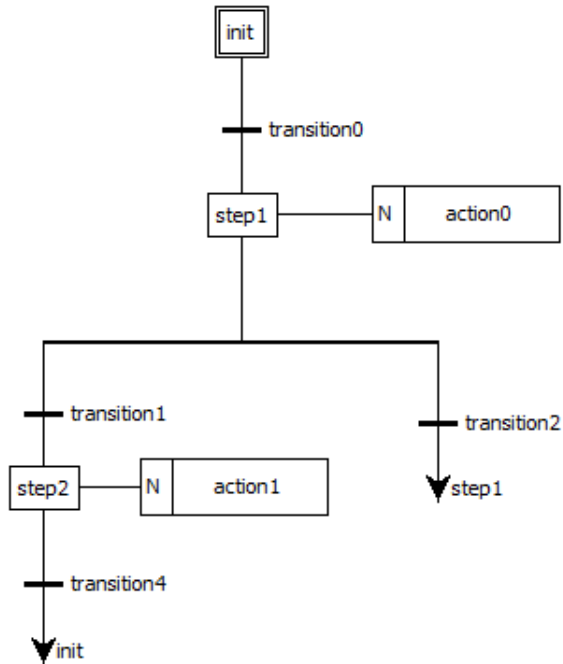


Рис. 7.7 – SFC диаграмма, содержащая «прыжки»

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 рис. 7.8 и выделен красным цветом:

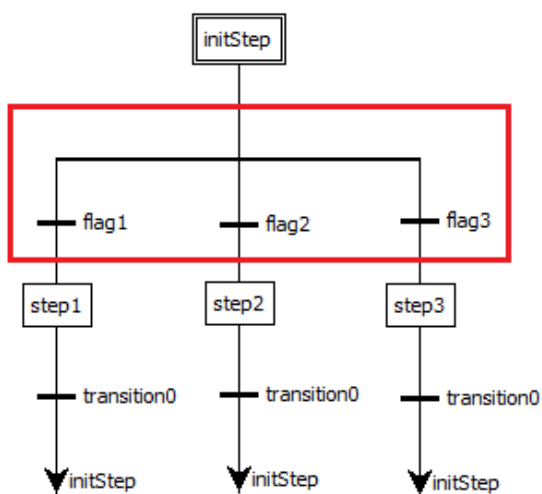


Рис. 7.8 – Дивергенция на SFC диаграмме

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 рис. 7.9 и выделен красным цветом:

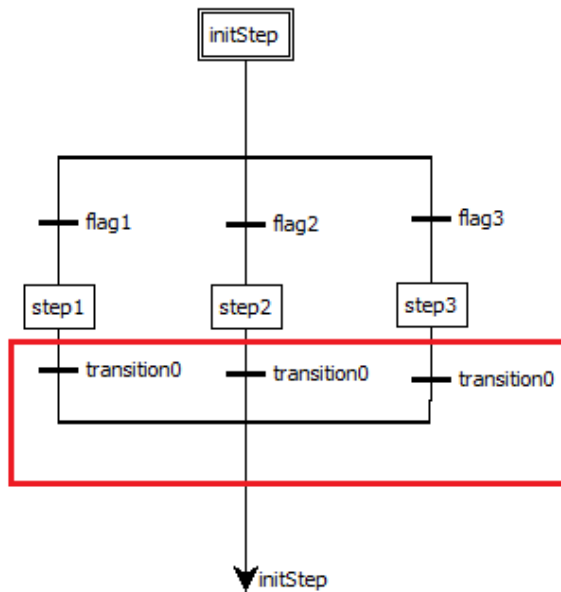


Рис. 7.9 – Конвергенция на SFC диаграмме

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 рис. 7.10 и выделен красным цветом:

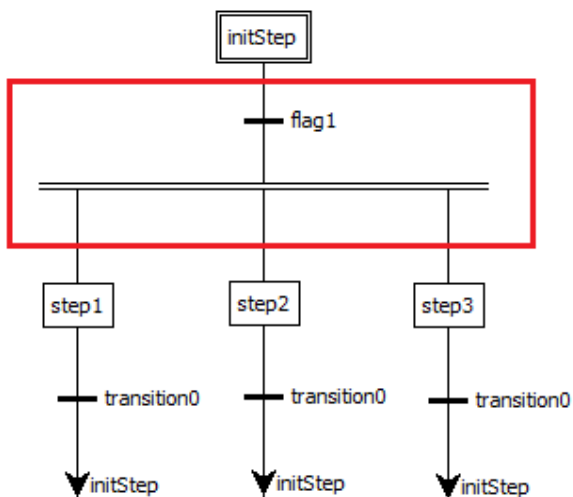


Рис. 7.10 – Параллельная дивергенция на SFC диаграмме

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 рис. 7.11 и выделен красным цветом:

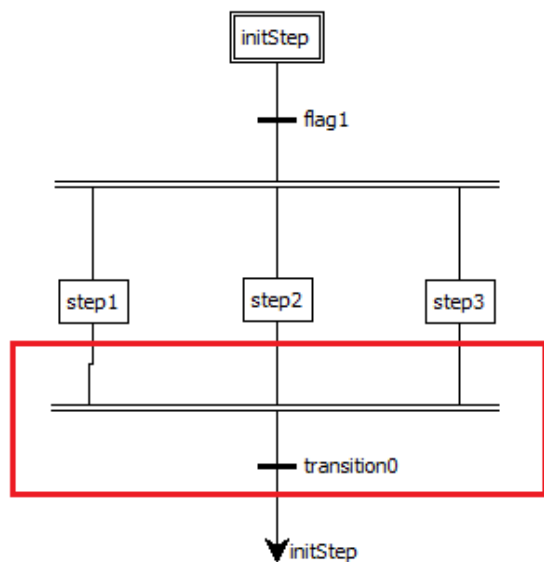


Рис. 7.11 – Параллельная конвергенция на SFC диаграмме

Пример программы на языке SFC

На рис. 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.

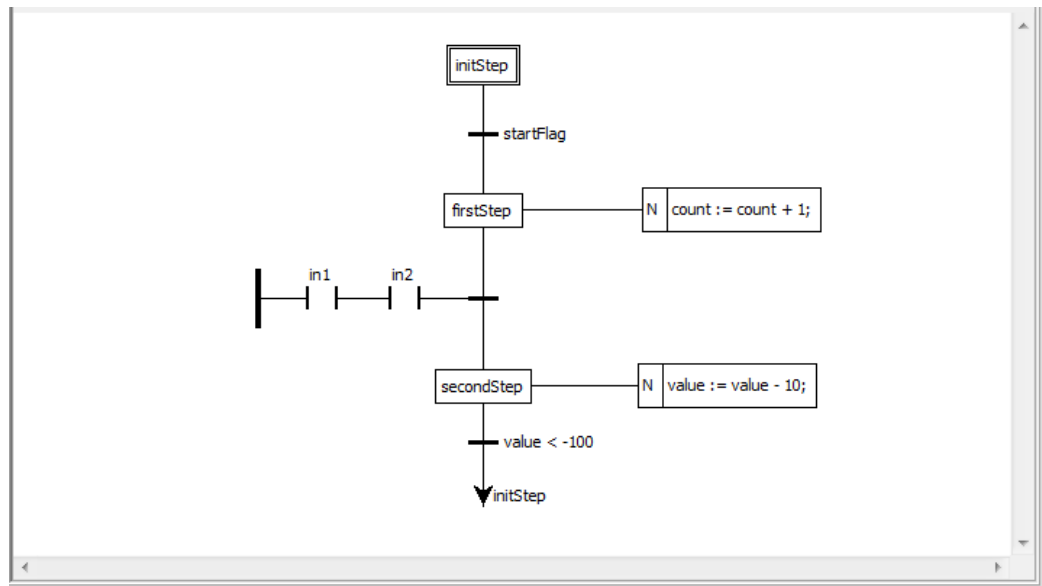


Рис. 7.12 – SFC диаграмма

2.3.5 Общие сведения о языке FBD

Содержание

- *Общие сведения о языке FBD*
 - *Основные понятия и конструкции языка*
 - *Пример программы на языке FBD*

FBD (Function Block Diagram) – это графический язык программирования высокого уровня, обеспечивающий управление потока данных всех типов. Позволяет использовать мощные алгоритмы простым вызовом функций и функциональных блоков. Удовлетворяет непрерывным динамическим процессам. Замечательно подходит для небольших приложений и удобен для реализации сложных вещей подобно ПИД регуляторам, массивам и т. д. Данный язык может использовать большую библиотеку блоков, описание которых приведено в приложении 2. FBD заимствует символику булевой алгебры и, так как булевы символы имеют входы и выходы, которые могут быть соединены между собой, FBD является более эффективным для представления структурной информации, чем язык релейно-контактных схем.

Основные понятия и конструкции языка

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

Переменные бывают входные, выходные и входные/выходные. На рис. 1 показаны: входная переменная – «in_var», выходная переменная – «out_var» и входная/выходная переменная – «in_out_var».

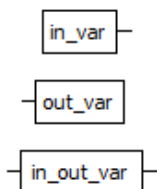


Рис. 1 – Изображение переменной в языке FBD

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

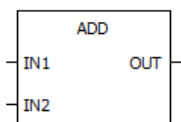


Рис. 2 – Изображение функции в языке FBD

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

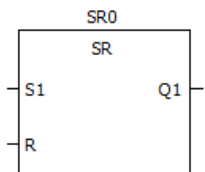


Рис. 3 – Изображение функционального блока в языке FBD

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.

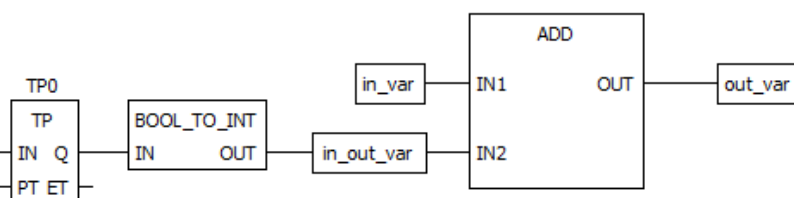


Рис. 4 – Пример соединения переменных, функций и функциональных блоков

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

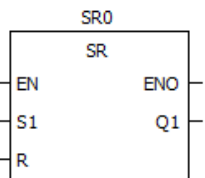


Рис. 5 – Изображение элементарного функционального блока с параметрами EN/ENO

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

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

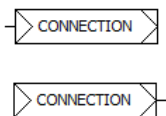


Рис. 6 – Изображение соединений в языке FBD

Они бывают двух видов: входное соединение и выходное. Основная задача соединений – передать значение из одного выхода на другой вход без прямого соединения выхода и входа. На рис. 5.7 показан пример, в котором выходное значение OUT функции BOOL_TO_INT передаётся на вход IN2 функции ADD:

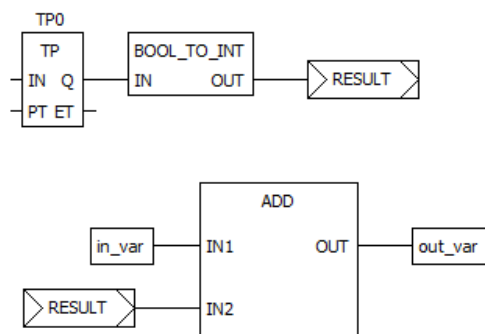


Рис. 7 – Пример использования соединения на FBD диаграмме

Пример программы на языке FBD

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

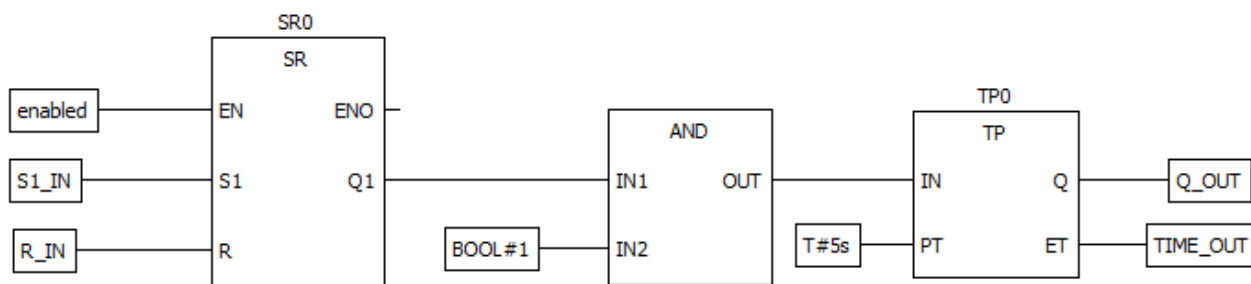


Рис. 8 – пример FBD диаграммы

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 (см. рис. 9).

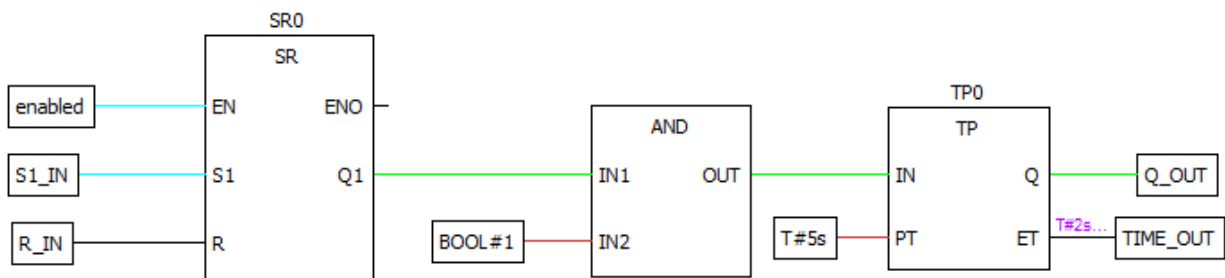


Рис. 9 – Выполнение FBD диаграммы

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 (см. рис. 10).

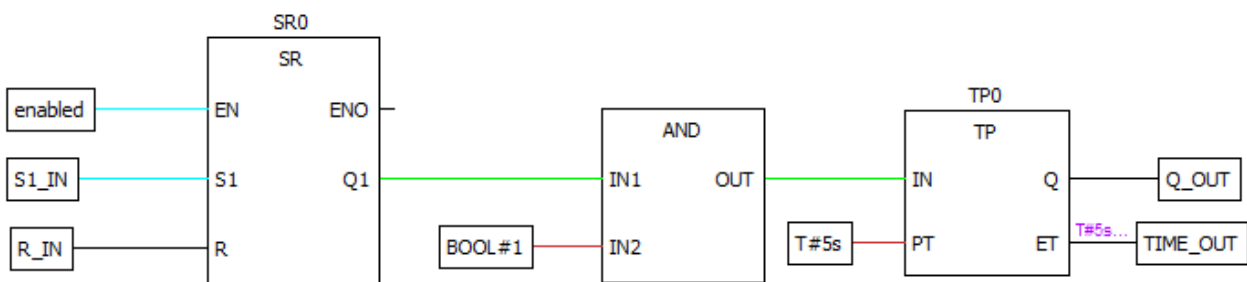


Рис. 10 – Выполнение FBD диаграммы

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

2.4 Beremiz: Руководство разработчика

2.4.1 Документация softPLC

Отладчик

Debugger code.

On “publish”, when buffer is free, debugger stores arbitrary variables content into, and mark this buffer as filled Buffer content is read asynchronously, (from non real time part), and then buffer marked free again.

Defines

`BUFFER_FREE`

`BUFFER_BUSY`

`__Unpack_case_t(TYPENAME)`

`__Unpack_case_p(TYPENAME)`

`__RegisterDebugVariable_case_t(TYPENAME)`


```

__RegisterDebugVariable_case_p(TYPENAME)
__ResetDebugVariablesIterator_case_t(TYPENAME)
__ResetDebugVariablesIterator_case_p(TYPENAME)

```

Typedefs

```
typedef void (*__for_each_variable_do_fp)(dbgvardsc_t *)
```

Functions

```

void __for_each_variable_do(__for_each_variable_do_fp fp)
void *UnpackVar(dbgvardsc_t *dsc, void **real_value_p, char *flags)
void Remind(unsigned int offset, unsigned int count, void *p)
void RemindIterator(dbgvardsc_t *dsc)
int CheckRetainBuffer(void)
void InitRetain(void)
void __init_debug(void)
void InitiateDebugTransfer(void)
void CleanupRetain(void)
void __cleanup_debug(void)
void __retrieve_debug(void)
void Retain(unsigned int offset, unsigned int count, void *p)
static void BufferIterator(dbgvardsc_t *dsc, int do_debug)
void DebugIterator(dbgvardsc_t *dsc)
void RetainIterator(dbgvardsc_t *dsc)
void PLC_GetTime(IEC_TIME *)
int TryEnterDebugSection(void)
long AtomicCompareExchange(long *, long, long)
long long AtomicCompareExchange64(long long *, long long, long long)
void LeaveDebugSection(void)
void ValidateRetainBuffer(void)
void InvalidateRetainBuffer(void)
void __publish_debug(void)
void RegisterDebugVariable(int idx, void *force)
void ResetDebugVariablesIterator(dbgvardsc_t *dsc)
void ResetDebugVariables(void)
void FreeDebugData(void)

```

```
int WaitDebugData(unsigned long *tick)
```

```
int GetDebugData(unsigned long *tick, unsigned long *size, void **buffer)
```

Variables

```
long buffer_state
```

```
char debug_buffer[BUFFER_SIZE]
```

```
char *buffer_cursor
```

```
unsigned int retain_offset
```

```
dbgvardsc_t dbgvardsc[]
```

```
unsigned long __tick
```

```
struct dbgvardsc_t
```

Public Members

```
void *ptr
```

```
__IEC_types_enum type
```

2.5 Участие в разработке документации

Для написании документации на Beremiz используется формат `restructredText`.

2.5.1 Быстрый старт

1. Установить Docker
2. Загрузить исходники Beremiz:

```
git clone https://github.com/jubnzv/beremiz beremiz
```

3. Для сборки документации:

```
cd beremiz/doc
make -f Makefile.docker
```

Сгенерированные документы в форматах *pdf* и *html* будут расположены в директории *beremiz/doc/build*.

2.5.2 Настройка окружения и сборка документации

По умолчанию: Использование Docker

Для упрощения установки на различные платформы предполагается использование `docker-контейнера`, содержащего необходимые для сборки пакеты.

1. Установить Docker

2. Загрузить исходники Beremiz:

```
git clone https://github.com/jubnzv/beremiz beremiz
```

3. Для сборки документации:

```
cd beremiz/doc
make -f Makefile.docker
```

Сгенерированные документы в форматах *pdf* и *html* будут расположены в директории *beremiz/doc/build*.

Альтернатива: Нативная установка (Debian GNU/Linux)

Для Debian Stretch потребуется установить следующие пакеты:

```
apt-get -y install make \
    python2.7 \
    python-sphinx \
    python-sphinx-rtd-theme \
    doxygen \
    graphviz \
    python-breathe \
    breathe-doc \
    texlive-base \
    texlive-latex-base \
    texlive-lang-cyrillic \
    texlive-fonts-recommended \
    texlive-generic-extra \
    texlive-latex-extra \
    texlive-latex-recommended
```

После этого сборка может быть запущена нативно:

```
cd beremiz/doc
make html latexpdf
```

2.5.3 TODO

- [] Текущий раздел:
 - [X] Quick start
 - [X] Linux installation notes
 - [] Краткое введение в rst и инструкции по настройке редакторов (emacs + pycharm)
 - [] Документирование кода (C-темплейты из *targets* и Python-код)
 - [] Работа с LaTeX и сборка pdf: внесение изменений в генерируемый .tex-код и модификация дефолтных опций форматирования
- [] IEC Guide
 - [] Актуализировать информацию для текущей версии Beremiz
- [] Usage Guide
 - [X] Обновить информацию; убрать функционал, относящийся к модулям ИНЭУМ

- [] Разбить документ на отдельные файлы по разделам
- [] Install guide
 - [] Обновить раздел с Windows 7
 - [] Установка на *nix
 - [] Инструкции для Windows 10

Symbols

__RegisterDebugVariable_case_p (макроподстановка C), 148
 __RegisterDebugVariable_case_t (макроподстановка C), 148
 __ResetDebugVariablesIterator_case_p (макроподстановка C), 149
 __ResetDebugVariablesIterator_case_t (макроподстановка C), 149
 __Unpack_case_p (макроподстановка C), 148
 __Unpack_case_t (макроподстановка C), 148
 __cleanup_debug (функция C++), 149
 __for_each_variable_do (функция C++), 149
 __for_each_variable_do_fp (тип C++), 149
 __init_debug (функция C++), 149
 __publish_debug (функция C++), 149
 __retrieve_debug (функция C++), 149
 __tick (поле C++), 150

A

AtomicCompareExchange (функция C++), 149
 AtomicCompareExchange64 (функция C++), 149

B

BUFFER_BUSY (макроподстановка C), 148
 buffer_cursor (поле C++), 150
 BUFFER_FREE (макроподстановка C), 148
 buffer_state (поле C++), 150
 BufferIterator (функция C++), 149

C

CheckRetainBuffer (функция C++), 149
 CleanupRetain (функция C++), 149

D

dbgvardsc (поле C++), 150
 dbgvardsc_t (класс C++), 150
 dbgvardsc_t::ptr (поле C++), 150
 dbgvardsc_t::type (поле C++), 150
 debug_buffer (поле C++), 150

DebugIterator (функция C++), 149

F

FreeDebugData (функция C++), 149

G

GetDebugData (функция C++), 150

I

InitiateDebugTransfer (функция C++), 149
 InitRetain (функция C++), 149
 InvalidateRetainBuffer (функция C++), 149

L

LeaveDebugSection (функция C++), 149

P

PLC_GetTime (функция C++), 149

R

RegisterDebugVariable (функция C++), 149
 Remind (функция C++), 149
 RemindIterator (функция C++), 149
 ResetDebugVariables (функция C++), 149
 ResetDebugVariablesIterator (функция C++), 149
 Retain (функция C++), 149
 retain_offset (поле C++), 150
 RetainIterator (функция C++), 149

T

TryEnterDebugSection (функция C++), 149

U

UnpackVar (функция C++), 149

V

ValidateRetainBuffer (функция C++), 149

W

WaitDebugData (функция C++), 149