# Руководство по программированию ПЛК BRIC в среде разработки Beremiz/OpenPLC

**СНЭМА-СЕРВИС**

**авг. 29, 2022**

## Содержание



Интегрированная среда разработки OpenPLC предназначена для создания и отладки прикладных программ на языках стандарта IEC 61131-3 для программируемых логических контроллеров. 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).

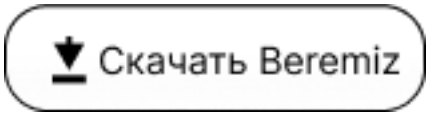
# 1 Руководство по программированию ПЛК BRIC в среде разработки OpenPLC

## 1.1 Введение

Интегрированная среда разработки OpenPLC предназначена для создания и отладки прикладных программ на языках стандарта IEC 61131-3 для программируемых логических контроллеров. 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).

В данном руководстве представлено описание порядка программирования ПЛК BRIC в среде разработки BEREMIZ. Документ содержит информацию о назначении программы, условиях выполнения, элементах пользовательского интерфейса, порядке разработки прикладных программ. Рассмотрены основные её компоненты и их назначение. Описан процесс работы с редакторами языков стандарта IEC 61131-3. В документе приведены тексты сообщений, выдаваемых в ходе выполнения программы и описание их содержания.

## 1.2 Инструкции по установке Beremiz

Инсталлятор в формате «.exe» можно скачать кликнув по 


При переходе по ссылке открывается страница загрузки файла «Beremiz_BRIC_0_12_0_4_Setup.exe».


Beremiz_BRIC_0_12_0_4_Setup.exe

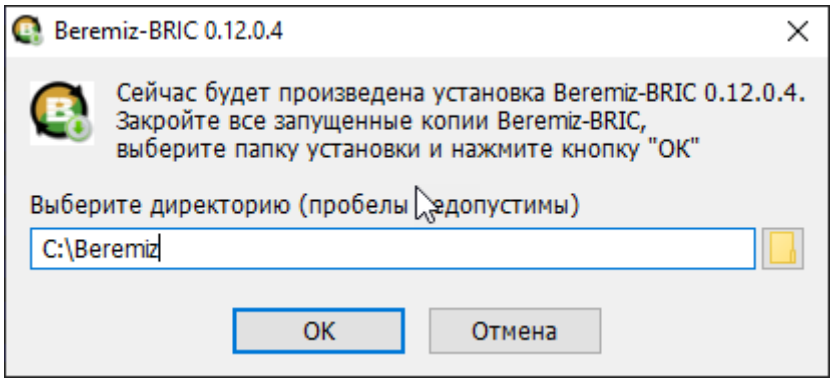
После скачивания необходимо запустить файл, дважды кликнув левой клавишей мыши. Открывается окно установки, в котором предлагается выбрать директорию.

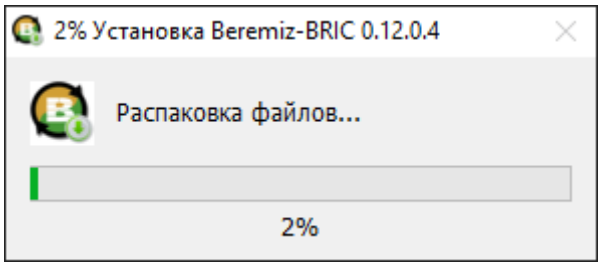
После выбора папки необходимо нажать кнопку «OK», с момента которого начинается процесс распаковки файлов.

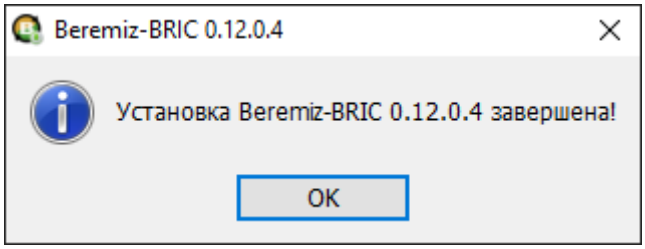
При успешной установке Beremiz выводится окно уведомления.

В результате установки в выбранной директории появится папка «Beremiz», в котором содержатся файлы и папки, представленные на рисунке ниже. Также появится ярлык на рабочем столе.

Запуск Beremiz осуществляется двойным нажатием левой клавишей мыши по файлу «Beremiz.exe», либо по ярлыку на рабочем столе.




| Имя | Дата изменения | Тип | Размер |
|---|---|---|---|
| beremiz | 28.03.2022 9:55 | Папка с файлами | |
| bric_examples | 28.03.2022 9:55 | Папка с файлами | |
| build | 28.03.2022 9:54 | Папка с файлами | |
| cmake-3-13-1-win32-x86 | 28.03.2022 9:54 | Папка с файлами | |
| gcc_6_3_1 | 28.03.2022 9:54 | Папка с файлами | |
| matiec | 28.03.2022 9:55 | Папка с файлами | |
| mingw | 28.03.2022 9:55 | Папка с файлами | |
| PortableGit | 28.03.2022 9:55 | Папка с файлами | |
| python2 | 28.03.2022 9:55 | Папка с файлами | |
| python3 | 28.03.2022 9:56 | Папка с файлами | |
| sofi | 28.03.2022 9:54 | Папка с файлами | |
| Uninstall | 28.03.2022 9:55 | Папка с файлами | |
| Beremiz.exe | 17.02.2022 14:50 | Приложение | 389 КБ |
| license.txt | 24.01.2017 5:12 | Файл "TXT" | 18 КБ |

## 1.3 Запуск интегрированной среды разработки OpenPLC и обзор главной страницы

Для создания нового проекта для ПЛК BRIC необходимо зайти в OpenPLC, обозначенную значком, указанным на рисунке.

Рис. 1: Иконка OpenPLC

ИСР OpenPLC имеет несколько областей на своем главном окне:

- Панель инструментов
- Дерево проекта
- Панель переменных и констант
- Панель библиотеки функций и функциональных блоков (ФБ)
- Панель экземпляров проекта
- Отладочная панель (подобнее указано в Приложении А).

Основные термины и определения OpenPLC представлены в Приложении Б.

## 1.4 Создание и загрузка пользовательской программы

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.

**Внимание:** Название проекта не должно содержать пробел и недопустимые символы

### ШАГ 1. СОЗДАНИЕ НОВОГО ПРОЕКТА

Новый проект создаётся с помощью главного меню «File» – «New», либо с помощью кнопки «New» на панели управления.

Далее появится диалог, в котором необходимо выбрать папку, где будет храниться данный проект.

В появившемся диалоге вам будет предложено настроить основной программный модуль проекта. В данном диалоге три поля:

- «POU Name»;
- «POU Type»;
- «Language».

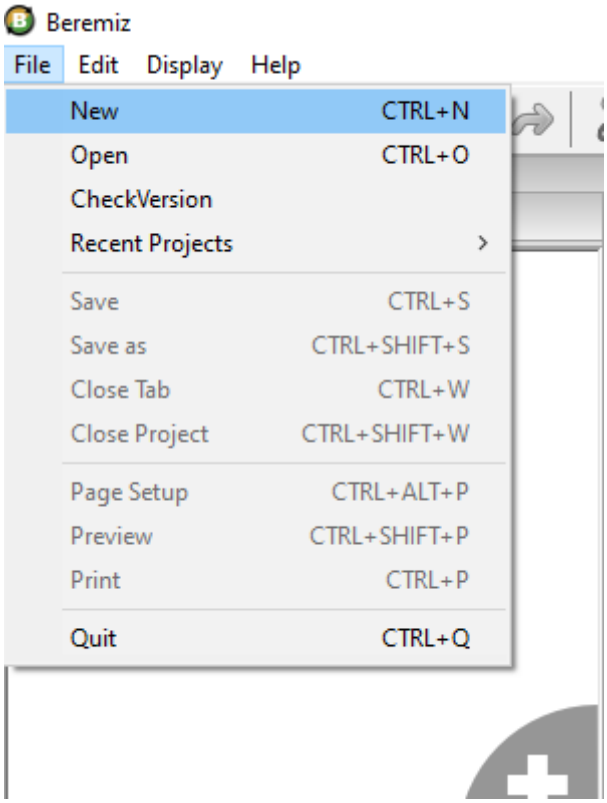



Рис. 2: Создание нового проекта с помощью главного меню

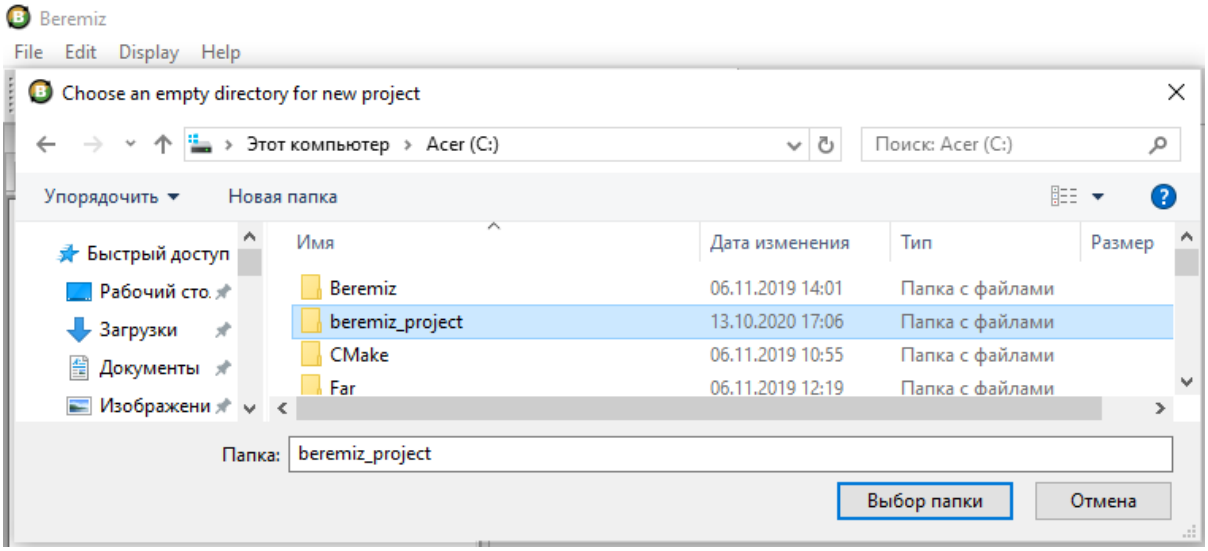



Рис. 3: Диалог выбора папки для нового проекта

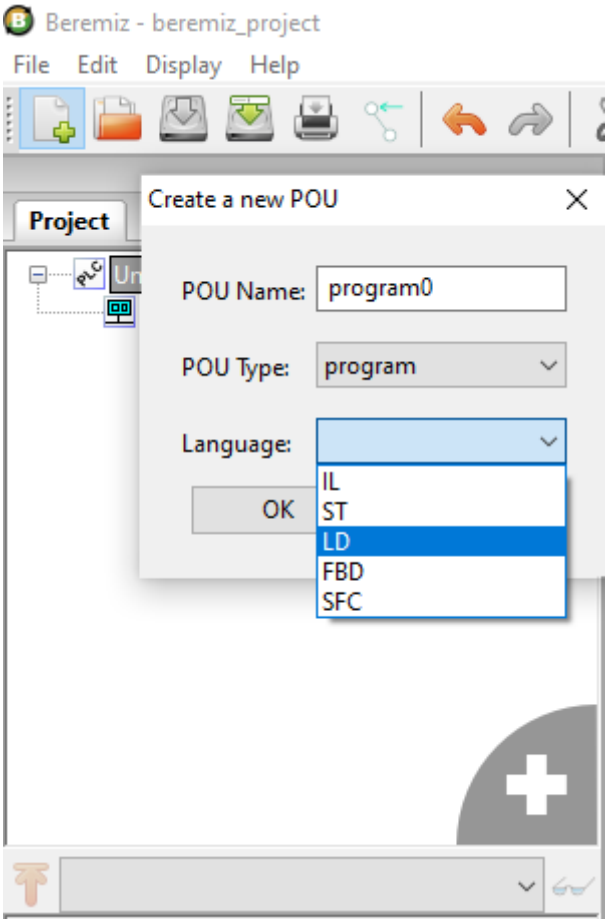



Рис. 4: Диалог добавления основного программного модуля

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 «Cancel» будет создан пустой проект без каких-либо настроек.

Для подробного описания процесса создания разработаем новую программу на языке FBD. Например, пусть программа является счетчиком, увеличивающий значение выхода на единицу до тех пор, пока на входе «RES» 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.

Также для подробного изучения создания проекта добавим функциональные блоки счетчиков, написанных на языках FBD, LD и ST.

**Конфигурационные переменные проекта**

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

В панели переменных и констант добавим конфигурационную константу «ZERO» типа INT с начальным значением 0, с помощью кнопки «Добавить переменную». Таким же образом добавим остальные переменные. На рисунке ниже предоставлен результат объявления конфигурационных переменных.

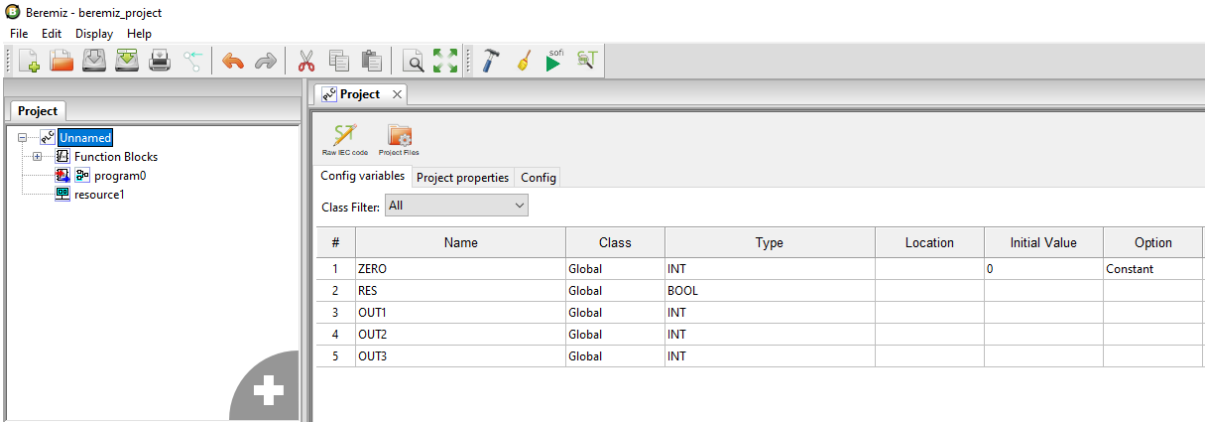



Рис. 5: Объявление конфигурационных переменных

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» (External).

**Настройки сборки проекта и соединения с ПЛК BRIC**

Для использования написанной прикладной программы необходимо её собрать (скомпилировать и скомпоновать), т.е. получить исполняемый файл и передать на целевое устройство (ПЛК BRIC) 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.

---

**Примечание:** Адресом последовательно порта ПЛК BRIC по умолчанию является *192.168.1.232*.

---

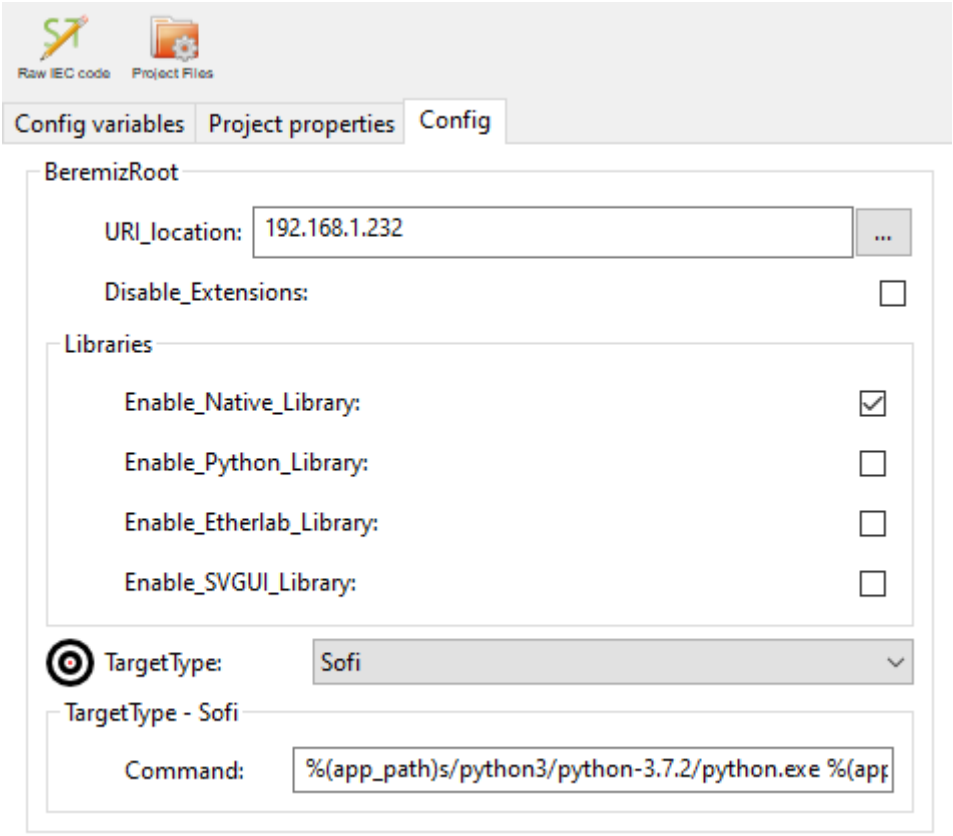



Рис. 6: Конфигурация проекта

**Внимание:** ПЛК BRIC и модули расширения имеют архитектуру целевой платформы «Sofi», поэтому во вкладке «Config» в разделе TargetType требуется установить целевую платформу «Sofi».

**Программа**

Ниже будет приведён пример добавления в проект программы, написанной на языке FBD. Логика и алгоритм работы данного программного модуля следующие: определена переменная RES типа BOOL, отвечающая за включение/выключение каждого из трёх счетчиков, определены три переменные OUT1..OUT3 типа INT, в них хранится значение каждого из трёх счетчиков, и добавлены три функциональные блоки, представляющих собой инкрементирующий счетчик на языках FBD, LD и ST. При запуске программы начальное значение переменной RES устанавливается по умолчанию True. Значения счетчиков начнут увеличиваться, начиная с 0, когда переменная RES примет значение False. Для обнуления счетчиков переменную RES необходимо форсировать значением True. Переменным OUT1..OUT3 будет присвоено начальное значение конфигурационной константы ZERO, таким образом значения счетчиков обнулятся.

Путём нажатия на кнопку «+» на правом верхнем углу добавим в панели переменных и констант переменную RES типа BOOL, отвечающую за вкл/выкл каждого из трёх счетчиков, а так же три переменные OUT1..OUT3 типа INT. Классы переменных назначим как Внешняя (External). Далее необходимо обратиться к редактору языка FBD. Для написания алгоритма и логики выполнения данной программы нам понадобятся функциональные блоки счетчиков.

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. 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).

Наведя мышь на переменную OUT1 в редакторе FBD и нажимая правую кнопку можно поменять тип пере-

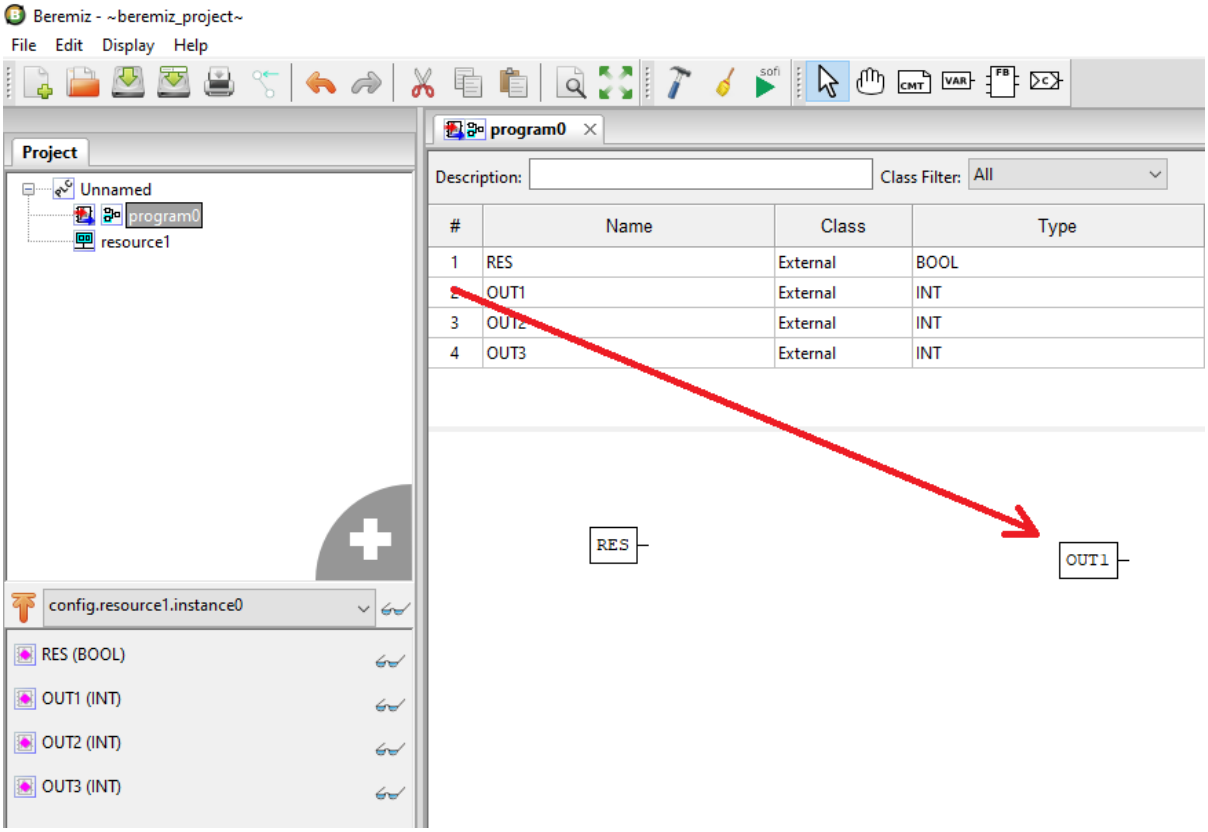



Рис. 7: Перенос переменных в поле редактирования

менной с Input на Output.

**Функциональный блок**

Добавление пользовательского функционального блока происходит путем нажатия на пункт «Функциональный блок» во всплывающем меню дерева проекта . В диалоговом окне задаем имя функционального блока в поле «POU Name», в поле «POU Type» выбираем «functionBlock», в поле «Language» выбираем язык, на котором будет написан алгоритм работы блока.

**Функциональный блок на языке FBD**

Создаём функциональный блок с именем «FBD», в котором инструментами языка FBD будет реализован счетчик , принимающий на вход переменную RES типа BOOL, и возвращающий значение счетчика OUT. На рисунке ниже показана реализация данного функционального блока.

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

Функция «ADD» находится во вкладке «Arithmetics» 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 «Selection». 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.

Соединение блоков осуществляется путем зажатия левой кнопки мыши на коннекторе блока, будет создана линия связи которую необходимо протянуть до коннектора присоединяемого блока.

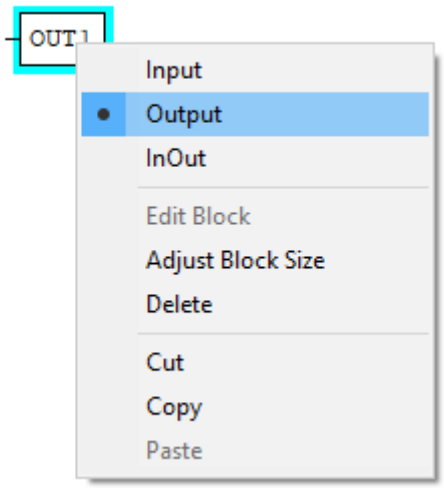



Рис. 8: Выбор коннектора для блока переменной

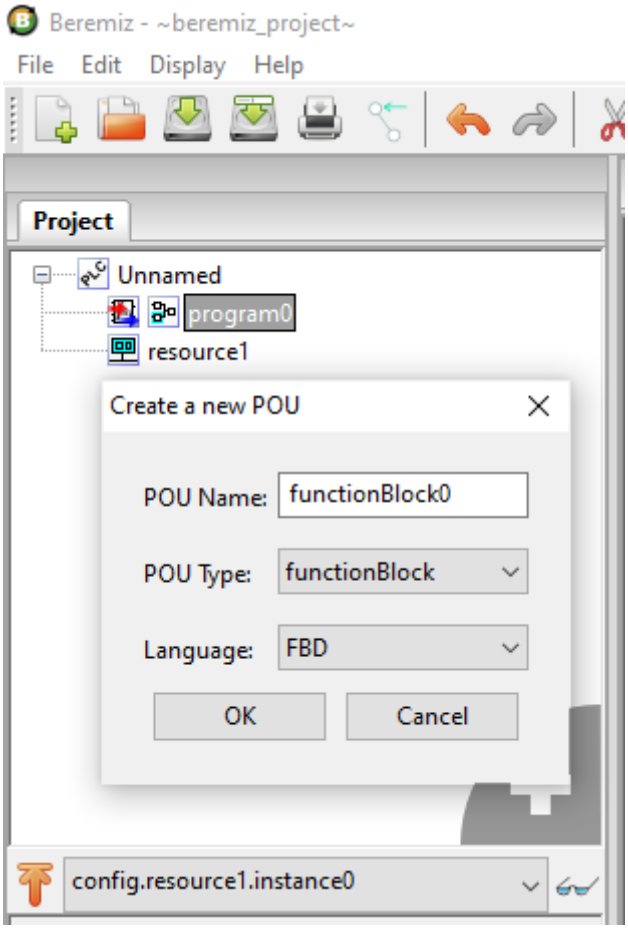



Рис. 9: Диалог создания нового функционального блока

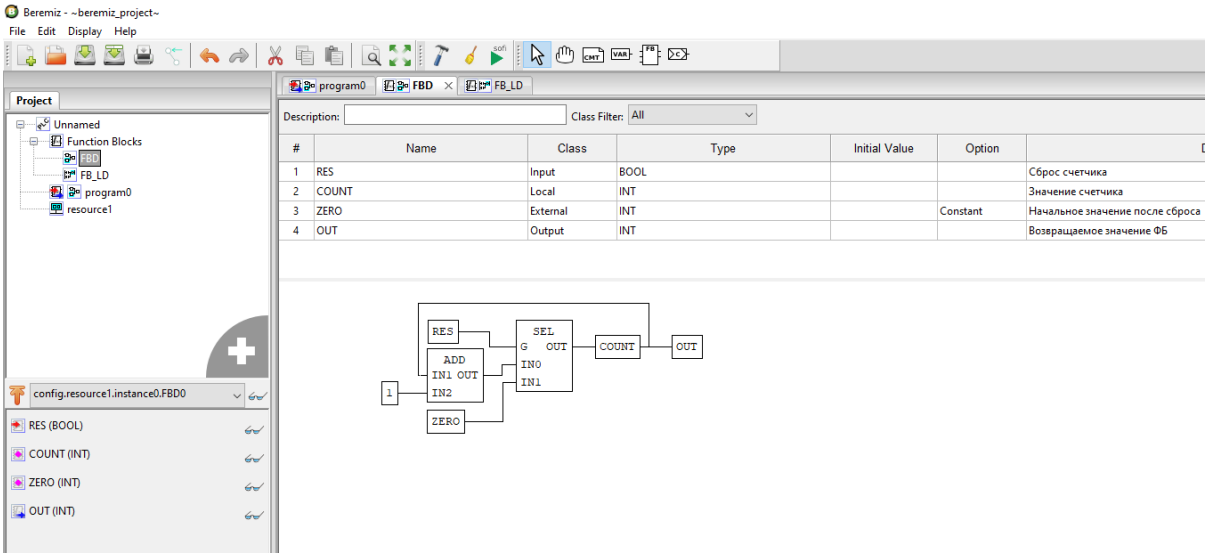



Рис. 10: Описание пользовательского функционального блока на языке FBD

Функциональный блок становится доступным в панели библиотеки функций и функциональных блоков «User-defined POUs» и может использоваться в программных модулях типа «Программа» и «Функциональный блок».

**Внимание:** Наименование функционального блока не должно совпадать с существующими в библиотеке

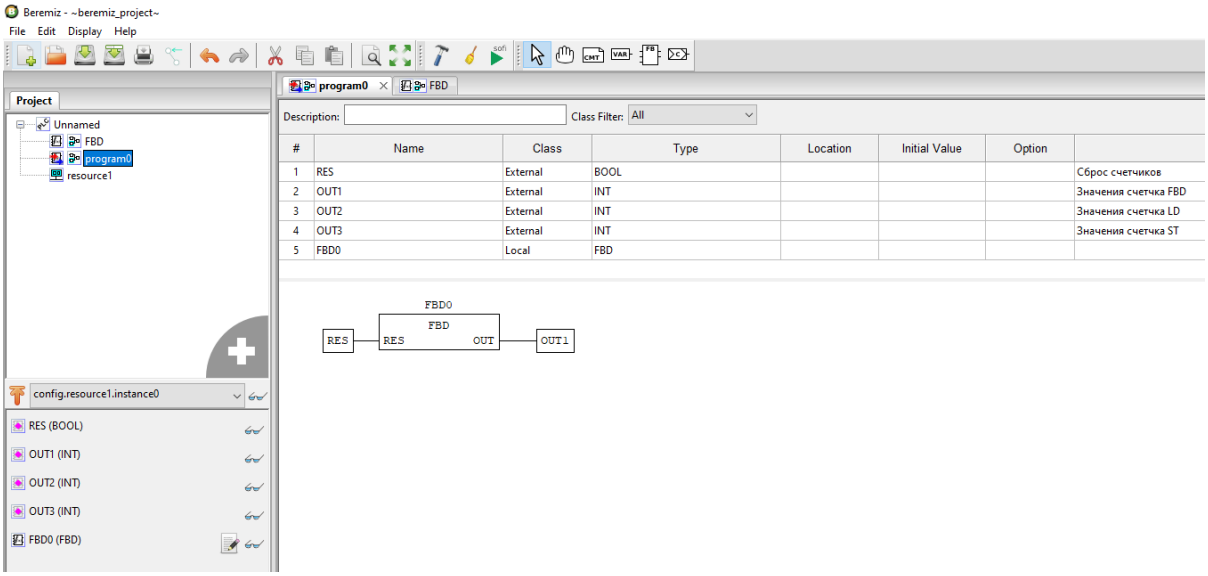



Рис. 11: Использование созданного функционального блока FBD в основном программном модуле

### Функциональный блок на языке LD

Создаём функциональный блок с именем «FBD», в котором инструментами языка FBD будет реализован счетчик , принимающий на вход переменную RES типа BOOL, и возвращающий значение счетчика OUT.

Добавим в панель переменных и констант возвращаемое значение «OUT» типа INT и класса «Выход», локальную переменную «COUNT» типа INT, внешнюю конфигурационную переменную «ZERO» типа INT, и входную переменную «RES» 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 из 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 LD диаграммы и отпустить кнопку мыши (Drag&Drop).

Добавим шину питания, к ней присоединим контакт, связанный с переменной «RES».

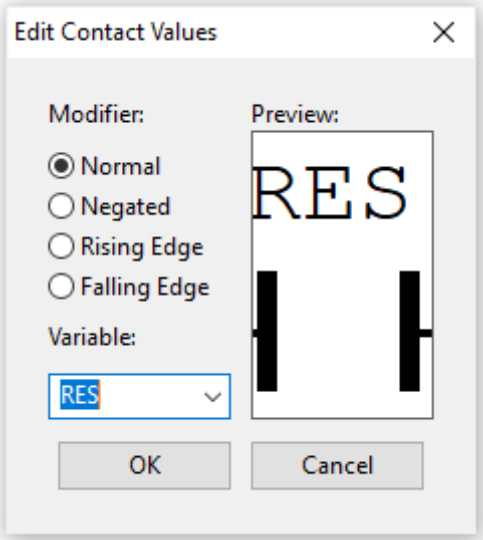



Рис. 12: Диалог добавления контакта

Полученная конструкция будет подавать сигнал на сброс счетчика при переходе значения переменной «RES» в True.

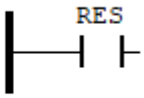



Рис. 13: *Контакт ассоциированный с переменной RES*

Далее добавим числовой литерал со значением «1» при помощи кнопки «Создать новую переменную», в диалоговом окне создания переменной в поле «Expression» напишем «1». Таким способом задается шаг инкрементации счетчика. Для написания алгоритма и логики выполнения данной программы будут добавлены две функции: «ADD» и «SEL». Добавление данных функций удобнее осуществить переносом соответствующей функции с помощью мыши (Drag&Drop) из панели Библиотеки функций и функциональных блоков в область редактирования LD диаграммы функционального блока.

На рисунке ниже показана реализация всего функционального блока на языке LD.

Функциональный блок становится доступным в панели библиотеки функций и функциональных блоков «User-defined POUs» и может использоваться в программных модулях типа «Программа» и «Функциональный блок». На рисунке ниже показано использование созданного функционального блока «FB_LD» в основном программном модуле, написанном на языке FBD.

**Внимание:** Наименование функционального блока не должно совпадать с существующими в библиотеке

**Функциональный блок на языке ST**

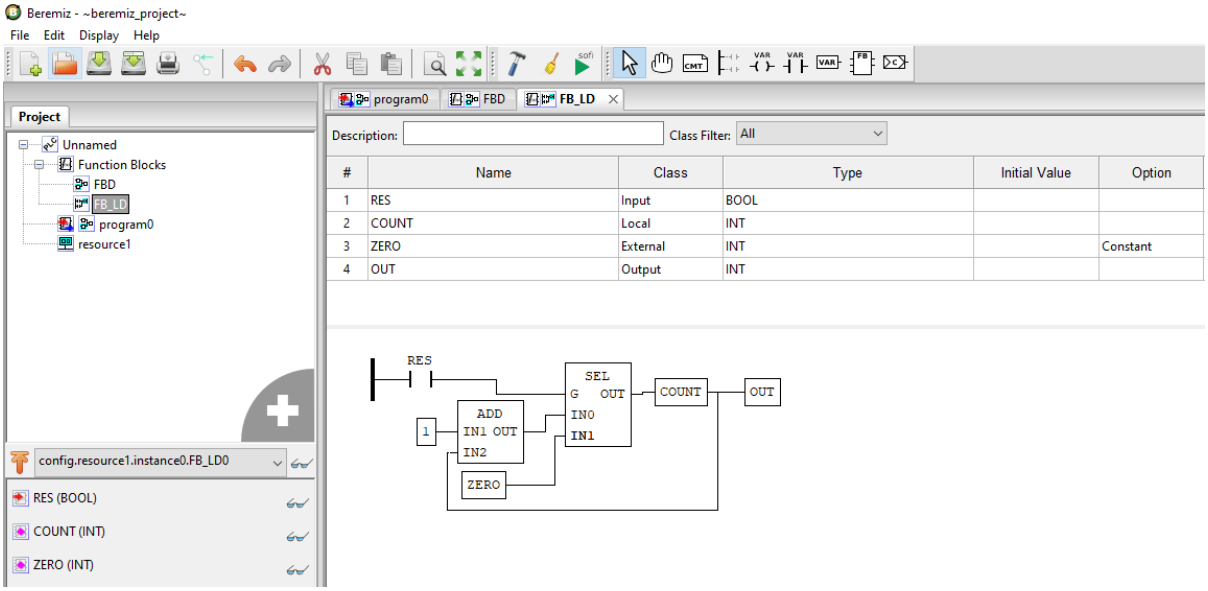



Рис. 14: *Функциональный блок на языке LD*

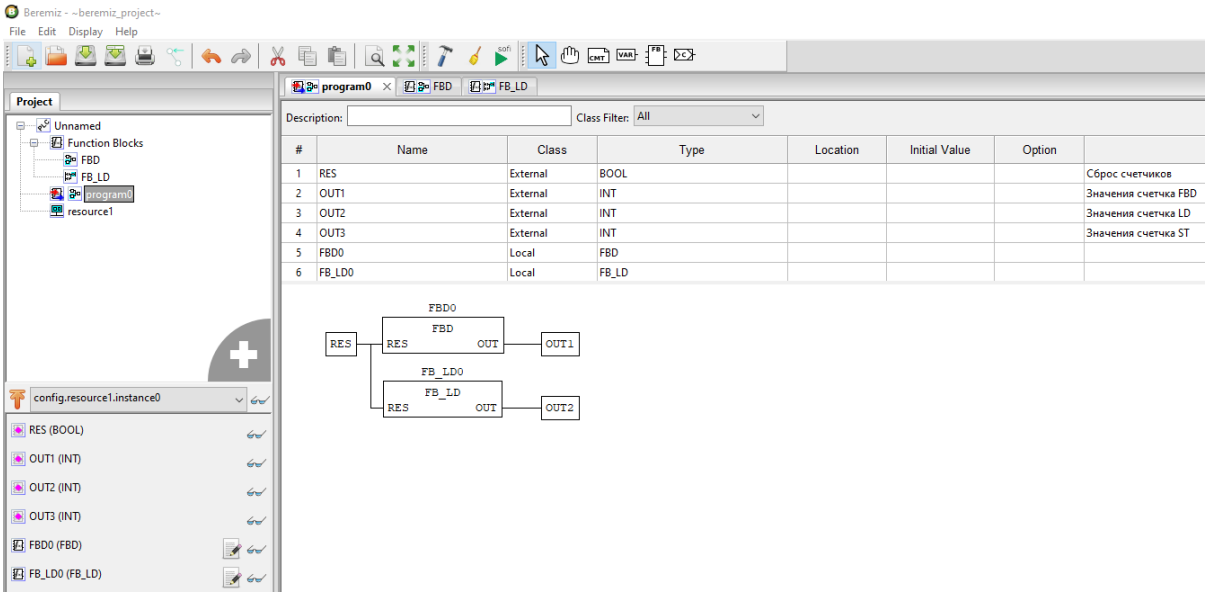



Рис. 15: *Использование функционального блока на языке LD в основном программном модуле*

Создаём функциональный блок с именем «FB_ST», в котором инструментами языка ST будет реализован счетчик , принимающий на вход переменную RES типа BOOL, и возвращающий значение счетчика OUT.

На рисунке ниже показана реализация данного функционального блока на языке ST.

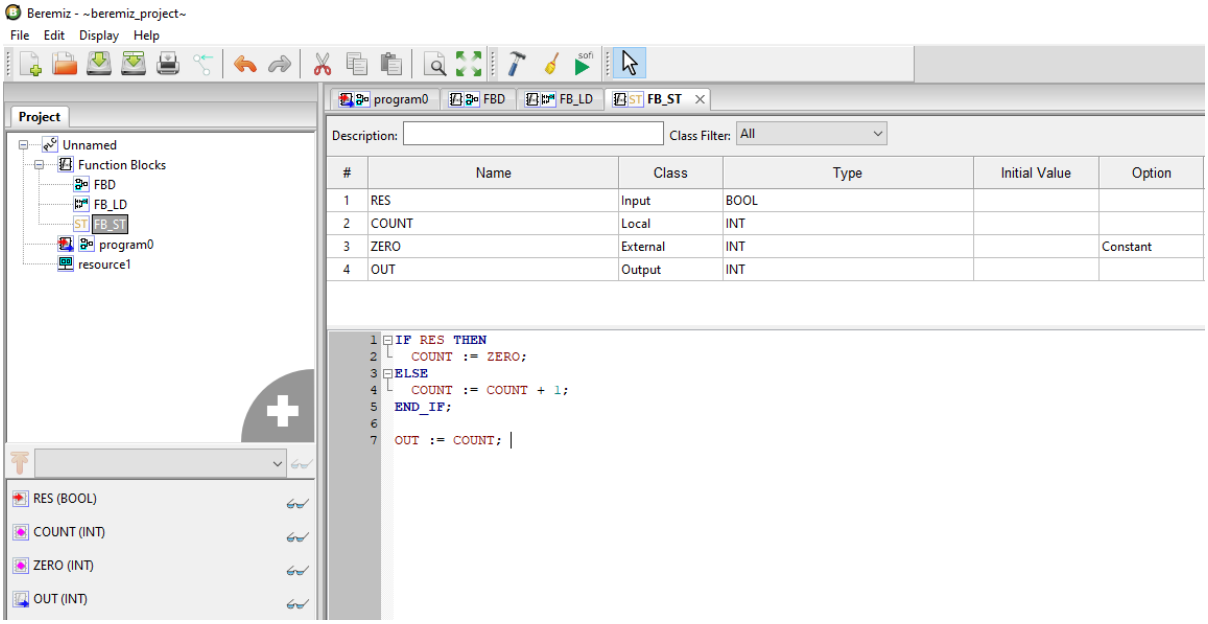



Рис. 16: Описание пользовательского функционального блока на языке ST

Функциональный блок становится доступным в панели библиотеки функций и функциональных блоков «User-defined POUs» и может использоваться в программных модулях типа «Программа» и «Функциональный блок».

**Внимание:** Наименование функционального блока не должно совпадать с существующими в библиотеке

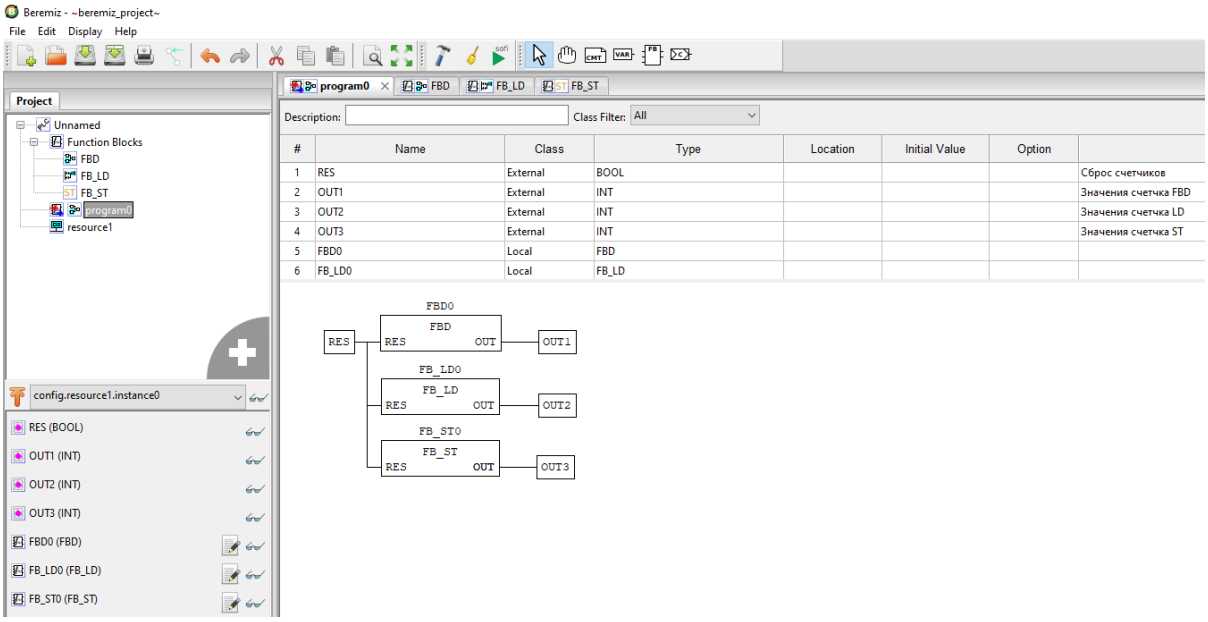



Рис. 17: Использование функционального блока на языке ST в основном программном модуле

**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 объявляются аналогично глобальным переменным проекта на панели переменных и констан выбранного ресурса с использованием кнопки «Добавить переменную», либо «Добавить переменные»

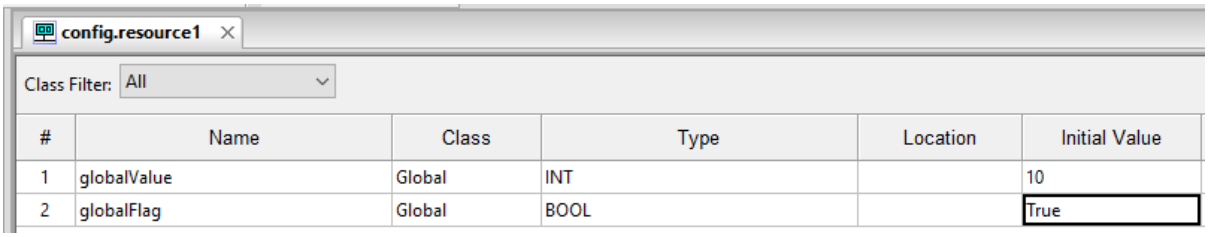



Рис. 18: Пример объявления в проекте глобальной переменной

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» (External) с таким же именем, как у глобальной переменной, объявленные выше для ресурса.

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

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

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

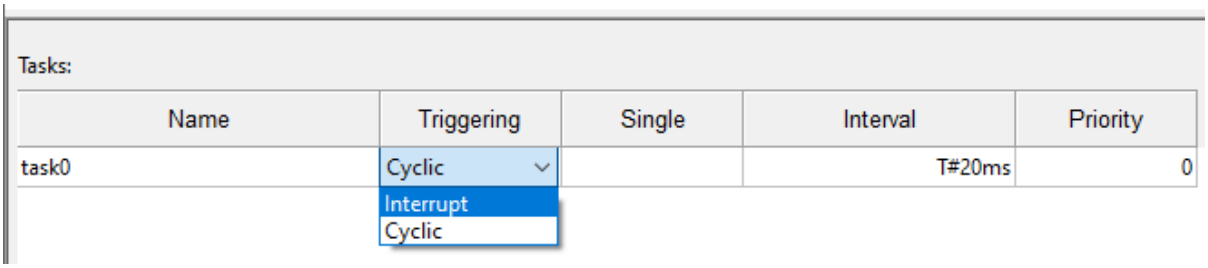



Рис. 19: Выбор типа выполнения задачи

В случае выбора типа выполнения «Cycle», в поле «Interval» необходимо указать интервал, с которым будет выполняться данная задача. Двойной щелчок левой кнопкой мыши по полю «Interval» приводит к появлению кнопки «…». Нажатие данной кнопки вызывает диалог «Edit Duration» в котором можно указать время, используя микросекунды, миллисекунды, секунды, минуты, часы и дни.

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

В случае выбора типа выполнения «Interrupt» в поле «Источник» необходимо указать переменную типа BOOL, определённую глобально либо на уровне проекта, либо на уровне ресурса. На рисунке ниже выбирается переменная «globalFlag», определённая в данном ресурсе.

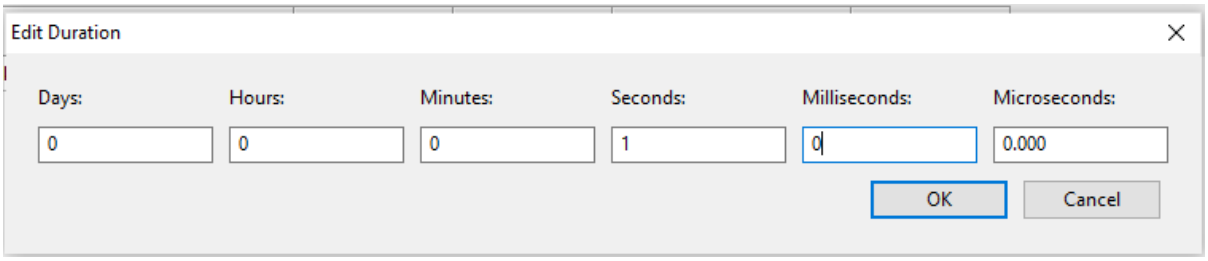



Рис. 20: Диалог редактирования длительности задачи


Рис. 21: Выбор переменной типа BOOL как источника прерывания для начала выполнения задачи

Задача будет выполнена один раз, как только значение переменной, определённой в этом поле, будет TRUE. Поле «Priority» позволяет указать приоритет выполнения задачи, по умолчанию все задачи имеют приоритет 0. Следует отметить, что в ресурсе должна быть определена как минимум одна задача с типом выполнения «Cycle», 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 «Type» и одну из задач в поле «Task».

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

### ШАГ 2. 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.

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).

Для сборки проекта нажмите кнопку «Сборка проекта в директории сборки».

Рис. 22: Кнопка сборки проекта

Результаты сборки выводятся в консоль, расположенную в нижней части окна программы, ошибки сборки выделяются красным цветом. На примере нашего проекта после сборки в консоль выведено сообщение о том, что сборка проведена успешно (подчеркнуто красным цветом).

Пересборку проекта можно осуществить, очистив директорию сборки проекта нажатием на кнопку «Очистить директорию сборки проекта». 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.

Итоговый бинарный файл, который будет загружен в ПЛК, находится в папке проекта с названием *sofi_task_crc.bin*. На рисунке ниже показан путь в папке проекта файла и название файла.

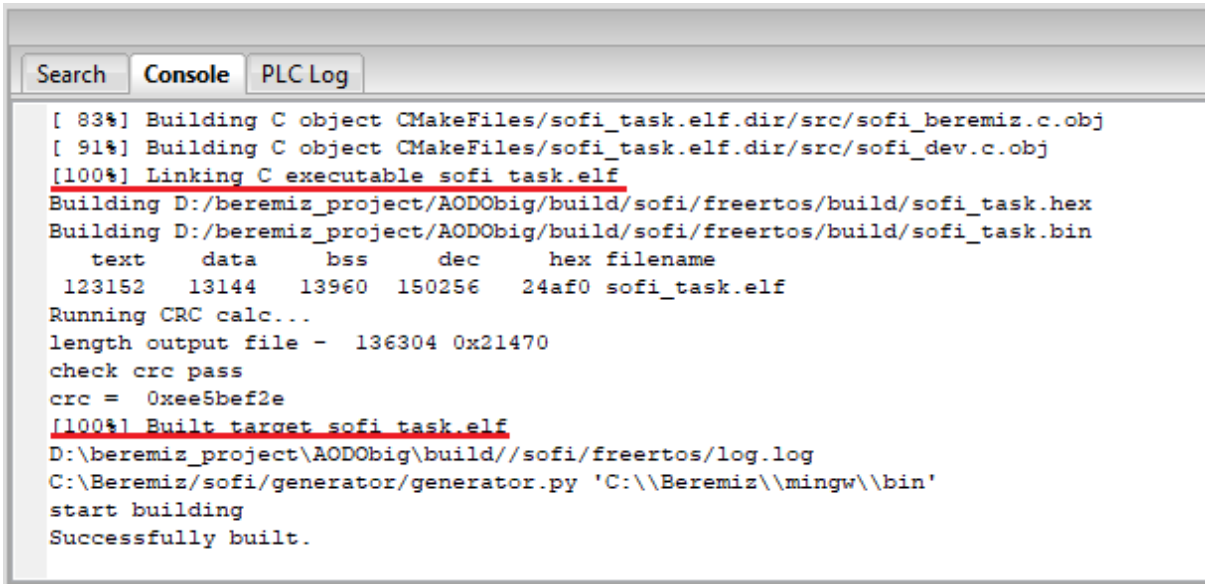



Рис. 23: Результаты сборки выведены в консоль

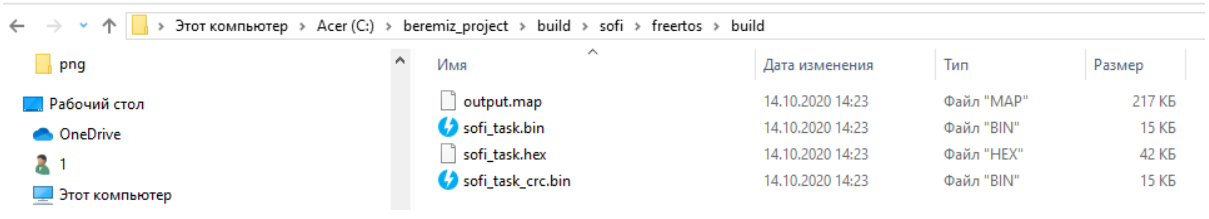



Рис. 24: Расположение итогового файла проекта

Для загрузки созданного проекта необходимо нажать на кнопку «Загрузить проект»


Рис. 25: Кнопка загрузки проекта

Результат загрузки выводится в консоль.

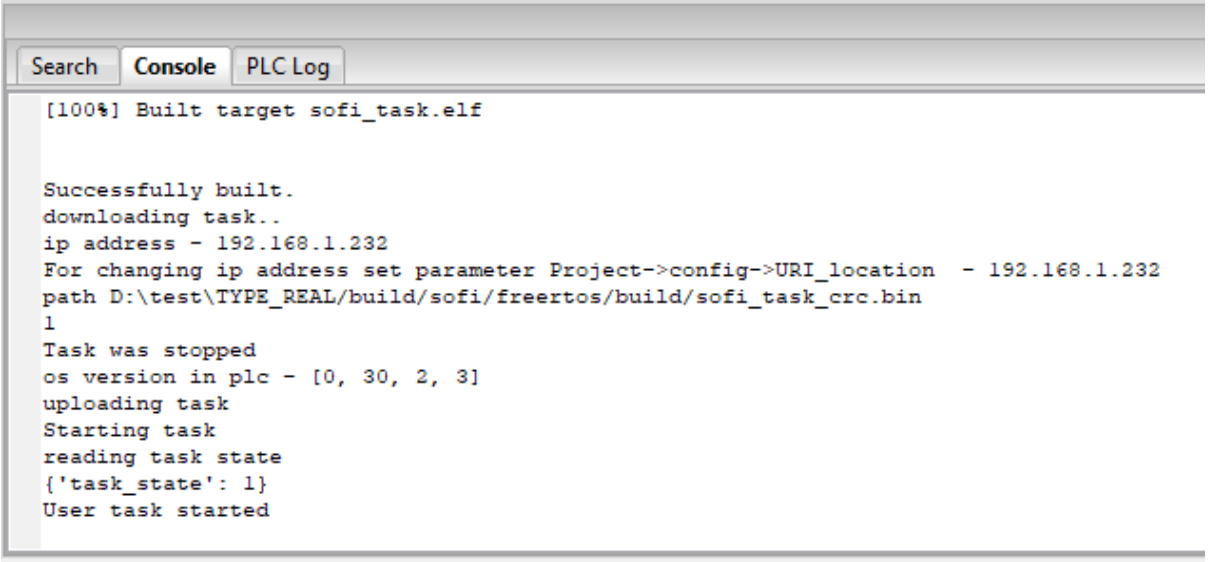



Рис. 26: Результат загрузки проекта в ПЛК

Так же имеется второй способ загрузки *user_task* – через WEB-интерфейс контроллера.

Для загрузки созданного бинарного файла в ПЛК BRIC заходим в браузер по адресу указанного порта, по умолчанию: 192.168.1.232

Далее нажимаем на кнопку «Enter Password» и вводим пароль (пароль по умолчанию «bric»), после чего нажимаем на кнопку «Download task» и выбираем запрашиваемый файл sofi_task_crc.bin. После нажатия кнопки «Download» дожидаемся окончания загрузки. После появления надписи «File Upload Done!» нажимаем на кнопку «Apply changes». Переход на главную страницу произойдет автоматически через 10 секунд.

Нажимая кнопку «Hide self regs» и устанавливая минимальный «Update period» мы можем в режиме реального времени наблюдать за изменениями показаний счетчиков при «RES» в состоянии False.

При изменении состояния переменной «RES» в True, путем нажатия кнопки «Change value», мы можем увидеть сброс всех счетчиков.

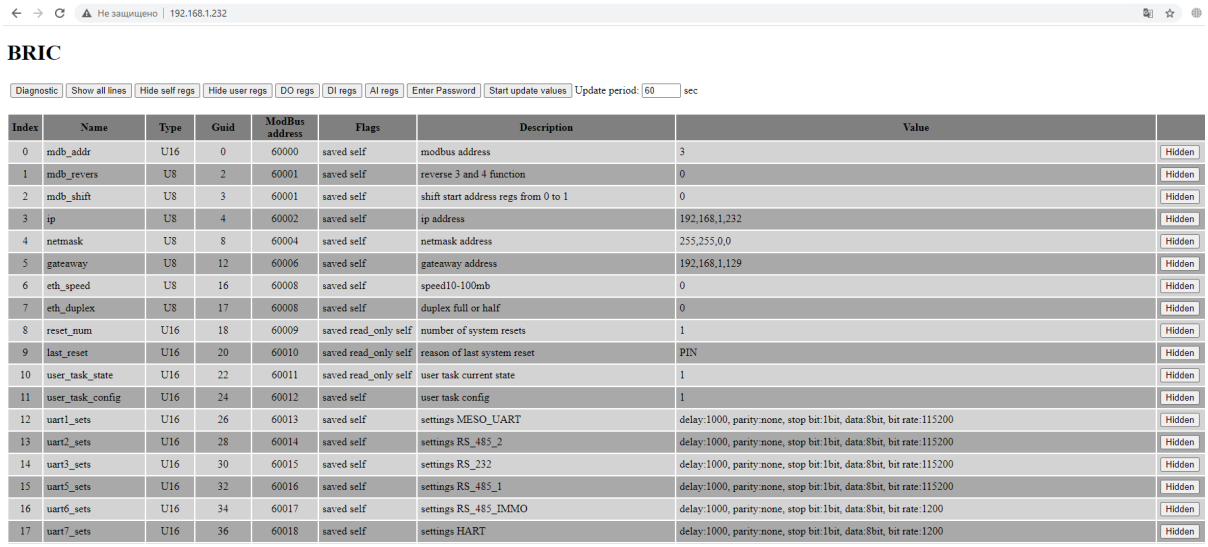
## 1.5 Стандартные средства конфигурирования (Modbus)

Для ПЛК BRIC имеется возможность опроса данных по протоколу Modbus, ретрансляции и расширения адресного пространства. Для этого необходимо заложить в структуру проекта элемент «Поддержка Modbus» и элемент для выполнения необходимого функционала:

- ModbusRTUMaster (фоновый опрос устройств для чтения и записи данных по протоколу Modbus);
- ModbusRoute (ретрансляция пакетов из одного канала в другой);
- MemoryArea (расширение адресного пространства).

По умолчанию, Modbus адресное пространство имеет зарезервированную область адресов:

0x30000–0x39999 – Область адресов, выделенных под чтение и запись массивов пользовательской программы;



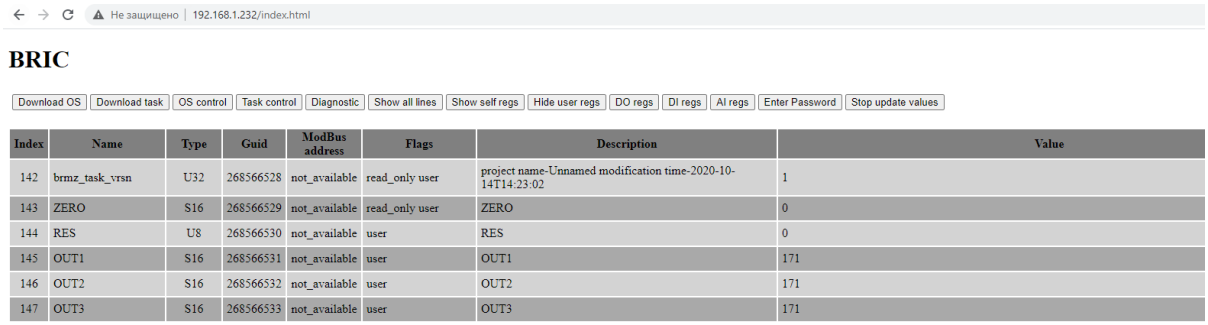
Не защищено | 192.168.1.232

**BRIC**

Diagnostic | Show all lines | Hide self regs | Hide user regs | DO regs | DI regs | AI regs | Enter Password | Start update values | Update period: 60 sec

| Index | Name | Type | Guid | ModBus address | Flags | Description | Value | |
|---|---|---|---|---|---|---|---|---|
| 0 | mdb_addr | U16 | 0 | 60000 | saved self | modbus address | 3 | Hidden |
| 1 | mdb_revers | U8 | 2 | 60001 | saved self | reverse 3 and 4 function | 0 | Hidden |
| 2 | mdb_shift | U8 | 3 | 60001 | saved self | shift start address regs from 0 to 1 | 0 | Hidden |
| 3 | ip | U8 | 4 | 60002 | saved self | ip address | 192,168,1,232 | Hidden |
| 4 | netmask | U8 | 8 | 60004 | saved self | netmask address | 255,255,0,0 | Hidden |
| 5 | gateaway | U8 | 12 | 60006 | saved self | gateaway address | 192,168,1,129 | Hidden |
| 6 | eth_speed | U8 | 16 | 60008 | saved self | speed10-100mb | 0 | Hidden |
| 7 | eth_duplex | U8 | 17 | 60008 | saved self | duplex full or half | 0 | Hidden |
| 8 | reset_num | U16 | 18 | 60009 | saved read_only self | number of system resets | 1 | Hidden |
| 9 | last_reset | U16 | 20 | 60010 | saved read_only self | reason of last system reset | PIN | Hidden |
| 10 | user_task_state | U16 | 22 | 60011 | saved read_only self | user task current state | 1 | Hidden |
| 11 | user_task_config | U16 | 24 | 60012 | saved self | user task config | 1 | Hidden |
| 12 | uart1_sets | U16 | 26 | 60013 | saved self | settings MESO_UART | delay:1000, parity:none, stop bit:1bit, data:8bit, bit rate:115200 | Hidden |
| 13 | uart2_sets | U16 | 28 | 60014 | saved self | settings RS_485_2 | delay:1000, parity:none, stop bit:1bit, data:8bit, bit rate:115200 | Hidden |
| 14 | uart3_sets | U16 | 30 | 60015 | saved self | settings RS_232 | delay:1000, parity:none, stop bit:1bit, data:8bit, bit rate:115200 | Hidden |
| 15 | uart5_sets | U16 | 32 | 60016 | saved self | settings RS_485_1 | delay:1000, parity:none, stop bit:1bit, data:8bit, bit rate:115200 | Hidden |
| 16 | uart6_sets | U16 | 34 | 60017 | saved self | settings RS_485_IMMO | delay:1000, parity:none, stop bit:1bit, data:8bit, bit rate:1200 | Hidden |
| 17 | uart7_sets | U16 | 36 | 60018 | saved self | settings HART | delay:1000, parity:none, stop bit:1bit, data:8bit, bit rate:1200 | Hidden |

Рис. 27: WEB-страница контроллера



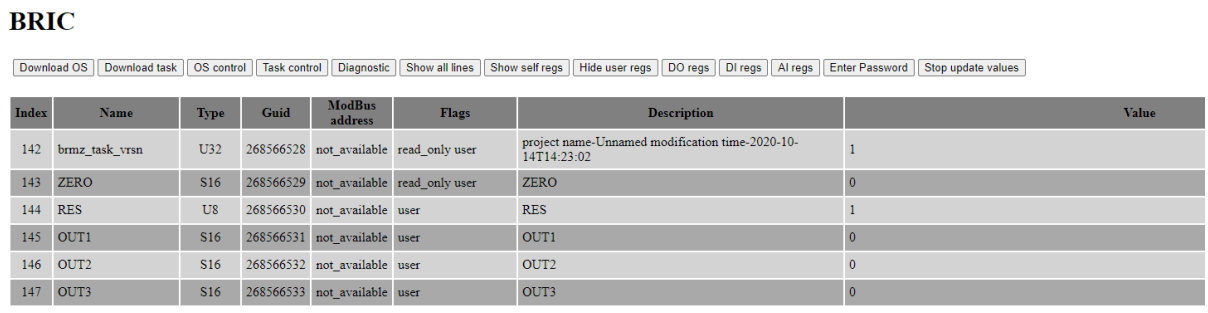
Не защищено | 192.168.1.232/index.html

**BRIC**

Download OS | Download task | OS control | Task control | Diagnostic | Show all lines | Show self regs | Hide user regs | DO regs | DI regs | AI regs | Enter Password | Stop update values

| Index | Name | Type | Guid | ModBus address | Flags | Description | Value |
|---|---|---|---|---|---|---|---|
| 142 | brmz_task_vrsn | U32 | 268566528 | not_available | read_only user | project name-Unnamed modification time-2020-10-14T14:23:02 | 1 |
| 143 | ZERO | S16 | 268566529 | not_available | read_only user | ZERO | 0 |
| 144 | RES | U8 | 268566530 | not_available | user | RES | 0 |
| 145 | OUT1 | S16 | 268566531 | not_available | user | OUT1 | 171 |
| 146 | OUT2 | S16 | 268566532 | not_available | user | OUT2 | 171 |
| 147 | OUT3 | S16 | 268566533 | not_available | user | OUT3 | 171 |

Рис. 28: WEB-страница контроллера с переменными при RES в состоянии False



**BRIC**

Download OS | Download task | OS control | Task control | Diagnostic | Show all lines | Show self regs | Hide user regs | DO regs | DI regs | AI regs | Enter Password | Stop update values

| Index | Name | Type | Guid | ModBus address | Flags | Description | Value |
|---|---|---|---|---|---|---|---|
| 142 | brmz_task_vrsn | U32 | 268566528 | not_available | read_only user | project name-Unnamed modification time-2020-10-14T14:23:02 | 1 |
| 143 | ZERO | S16 | 268566529 | not_available | read_only user | ZERO | 0 |
| 144 | RES | U8 | 268566530 | not_available | user | RES | 1 |
| 145 | OUT1 | S16 | 268566531 | not_available | user | OUT1 | 0 |
| 146 | OUT2 | S16 | 268566532 | not_available | user | OUT2 | 0 |
| 147 | OUT3 | S16 | 268566533 | not_available | user | OUT3 | 0 |

Рис. 29: WEB-страница контроллера с переменными при RES в состоянии True

0x40000–0x49999 – Область адресов, выделенных под чтение и запись регистров пользовательской программы;

0x60000–0x69999 – Область адресов, выделенных под системные регистры ПЛК.

Настройка интерфейсов и протоколов обмена в OpenPLC имеет следующую последовательность действий:


### Создание подмодуля ModbusRTUMaster

Обмен данными со сторонними устройствами по Modbus осуществляет элемент «ModbusRTUMaster». Для подключения необходимо добавить его в элемент «Поддержка Modbus», щелкнув правой клавишей мыши и выбрав пункт «Добавить ModbusRTUMaster». Затем в окне конфигурации настроить канал опроса. Для каждого физического канала может быть не более одного элемента «ModbusRTUMaster».

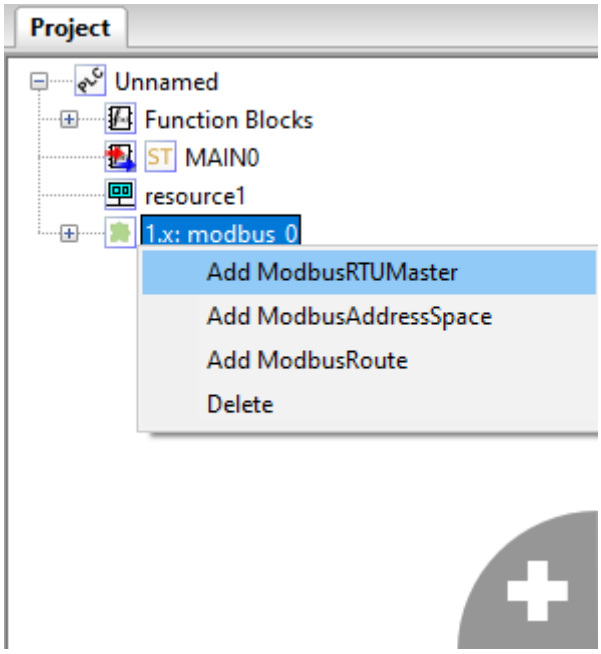



Рис. 30: Добавление в структуру проекта ModbusRTUMaster

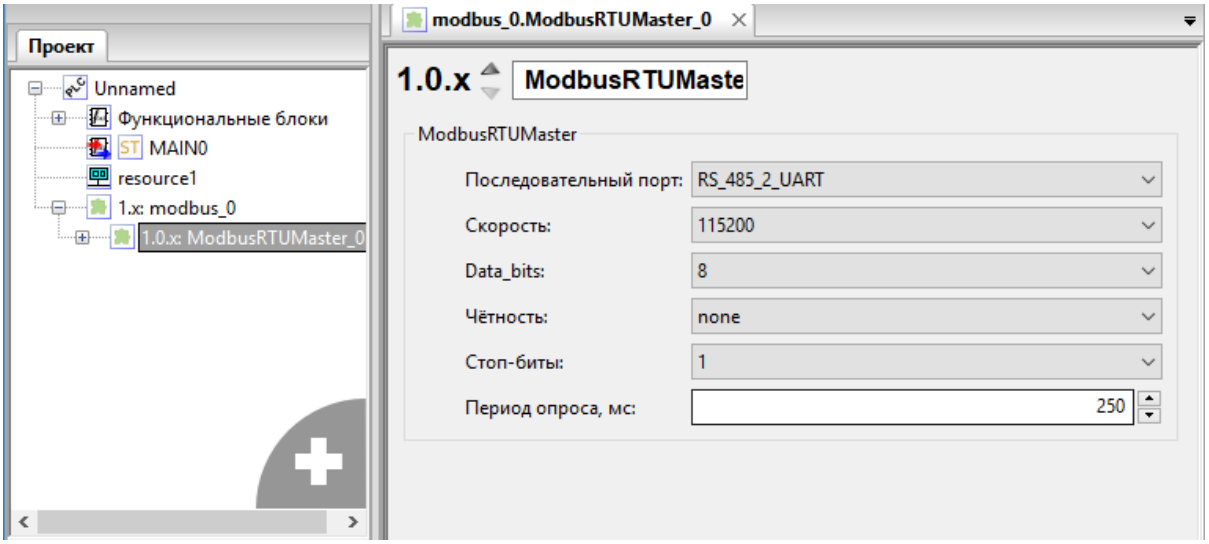



Рис. 31: Выбор параметров ModbusRTUMaster

Установка параметров ModbusRTUMaster включает следующие позиции:

- Последовательный порт (выбрать порт из перечня: RS_232_UART, RS_485_1_UART, RS_485_2_UART)
- Скорость передачи данных (выбор производится из установленного ряда: 1200, 2400, 4800, 9600,14400, 19200, 28800, 38400, 56000 57600, 76800, 115200 бит/с.)
- Количество информационных битов
- Тип чётности байта при её наличии (выбор производится из установленного ряда: even(чётный), none(без чётности), odd(нёчетный))
- Количество Стоп–битов
- Период опроса.

Далее необходимо добавить элементы «ModbusRequest» и настроить их. Разрешается добавлять несколько элементов с разными параметрами, тогда они будут встроены в порядок опроса последовательно.

Установка параметров ModbusRequest включает следующие позиции:

- Выбор команды (описание команд указано в Приложении Г)
- Адрес slave–устройства с которым производится обмен данными по Modbus протоколу
- Число Reg/Coil в одном пакете (для Reg до 120)
- Адрес первого Reg/Coil в пакете согласно адресного пространства slave–устройства
- Таймаут в мс. (не должен превышать период опроса).

После установления необходимых параметров «ModbusRequest» необходимо обозначить глобальные переменные, которые используются при передаче данных по протоколу Modbus. Для этого в панели переменных и констант необходимо для переменной в ячейке «Location» записать ссылку на адрес. Подробное описание конфигурирования см. в разделе Привязка глобальным переменным Modbus адреса.

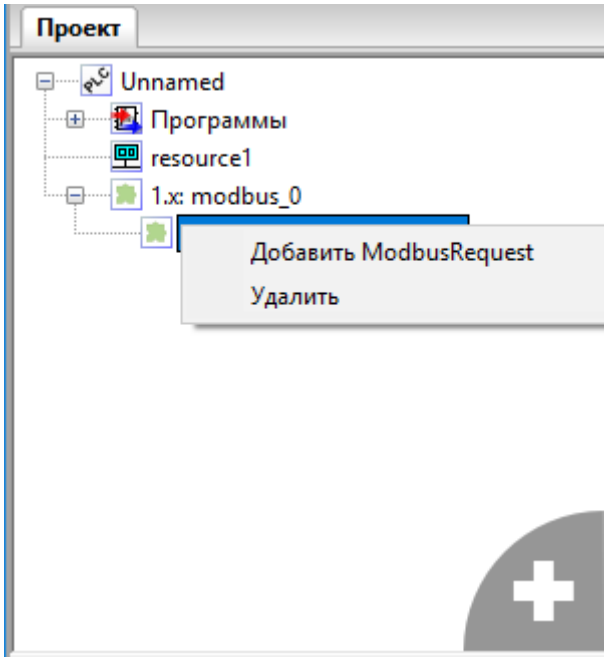



Рис. 32: Добавление в структуру проекта ModbusRequest

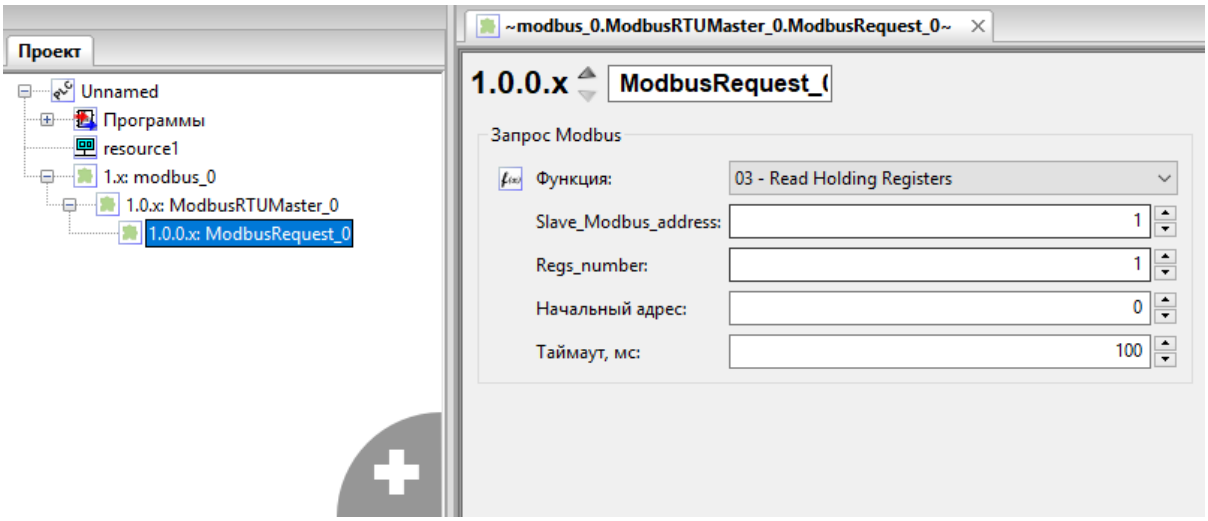
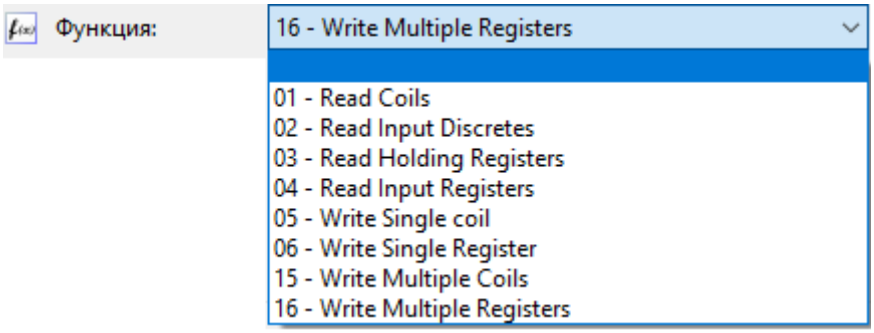



Рис. 33: Выбор параметров ModbusRequest



### Создание ModbusRoute

ПЛК BRIC имеет возможность ретранслировать пакеты из одного канала в другой. Также есть возможность приёма-передачи пакетов Modbus TCP в Modbus RTU и Modbus RTU – Modbus RTU. Для подключения подмодуля «ModbusRoute» необходимо подвести курсор к созданной ветке «Поддержка Modbus», щелкнуть правой клавишей мыши и выбрать пункт «Добавить ModbusRoute».

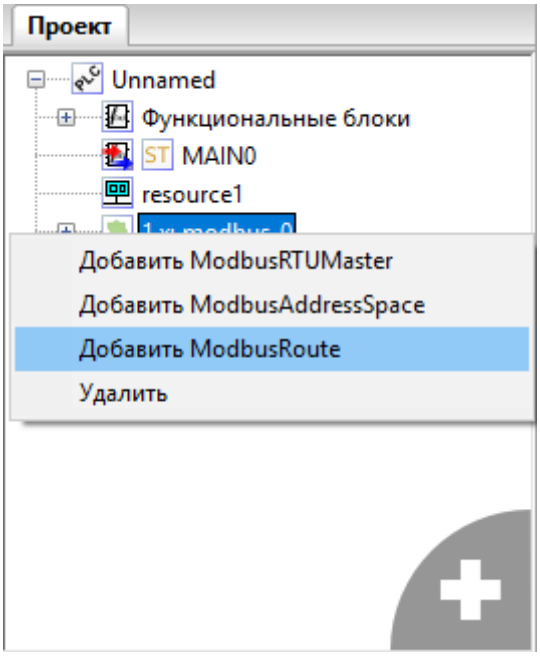



Рис. 34: Добавление в структуру проекта ModbusRoute

Окно конфигурирования ModbusRoute представлено на рисунке ниже:

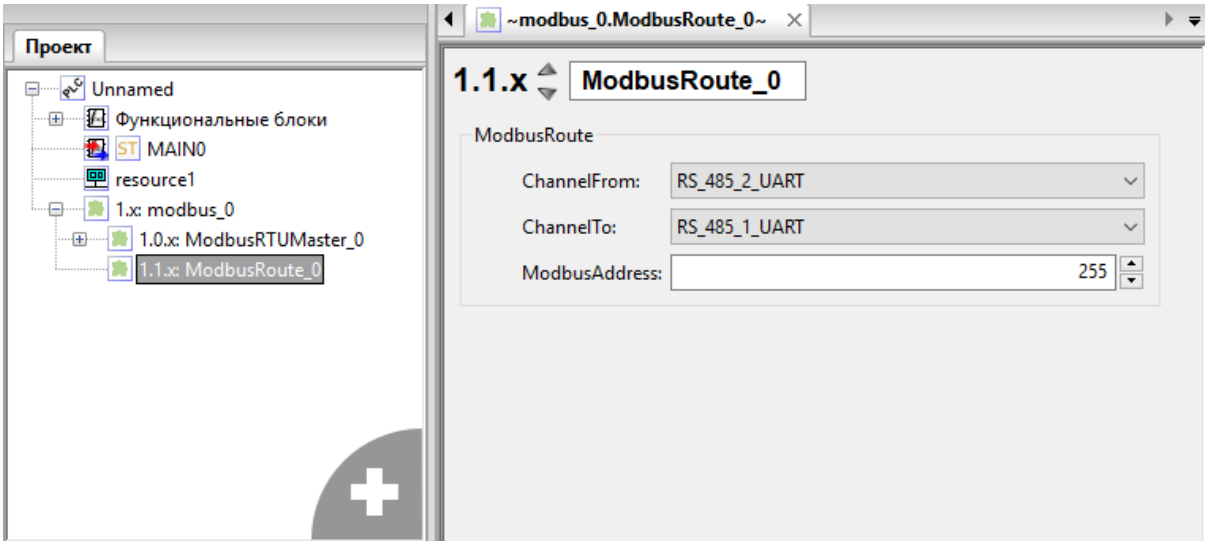
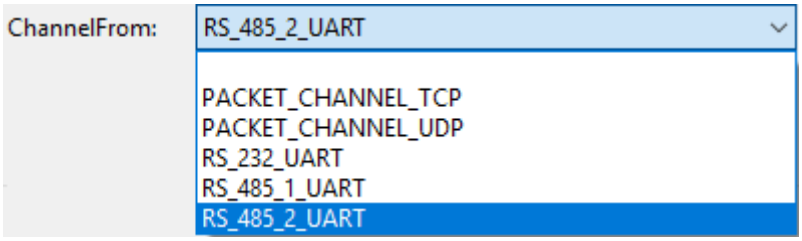



Рис. 35: Выбор параметров ModbusRoute

Установка параметров ModbusRoute включает следующие позиции:

- Задействованные каналы (выбрать порт из перечня: PACKET_CHANNEL_TCP, RS_232_UART, RS_485_1_UART, RS_485_2_UART, PACKET_CHANNEL_UDP)[1] ;

[1] PACKET_CHANNEL_TCP, PACKET_CHANNEL_UDP данные передаваемые протоколами (TCP, UDP) через канал связи Ethernet.




- ModbusAddress (Modbus адрес устройства, для которого производится ретрансляция из одного канала в другой)[2] .

### Создание MemoryArea

ПЛК BRIC имеет возможность увеличить адресное пространство для каждого типа регистров (Coils, Input Discrete, Input Registers, Holding Registers) при помощи подключения подмодуля «MemoryArea». Для подключения «MemoryArea» необходимо подвести курсор к созданной ветке «Поддержка Modbus», щелкнуть правой клавишей мыши и выбрать пункт «Добавить MemoryArea».

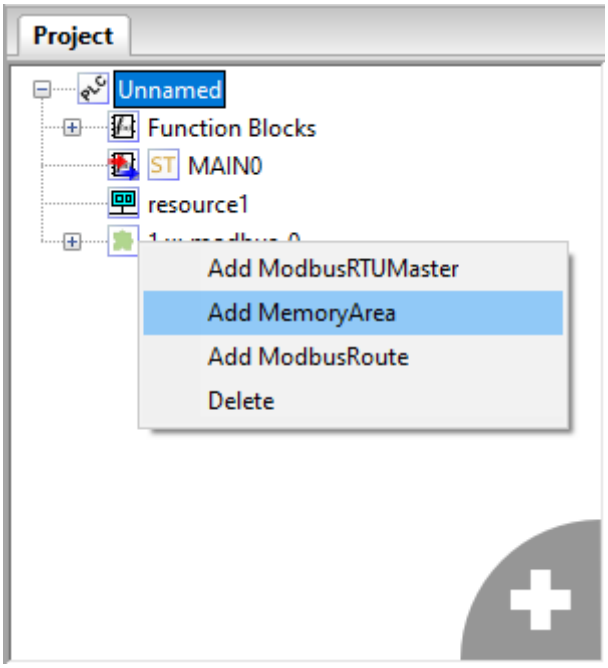



Рис. 36: Добавление в структуру проекта MemoryArea

Окно конфигурирования MemoryArea показано на рисунке ниже:

Установка параметров MemoryArea включает следующие позиции:

- Выбор типа регистра адресного пространства (Coils, Input Discrete, Input Registers, Holding Registers)
- Regs_number количество регистров в адресном пространстве[3]
- Start_Address первый Modbus-адрес адресного пространства.

Расположение разных типов регистров независимо, поэтому номера регистров разных типов могут иметь одинаковое значение. Ограничением по количеству регистров является:

- Размер итогового файла сборки проекта, указываемого в отладочной панели под элементом «dec» строки (text data bss dec hex filename), составляющий не более «197524»

[2] При указании адреса 255 ретранслирует все пакеты полученные с канала «извлечения» в канал «записи».
[3] Modbus-адреса, задаваемые в разных адресных пространствах для одинаковых типов регистров, не должны повторяться

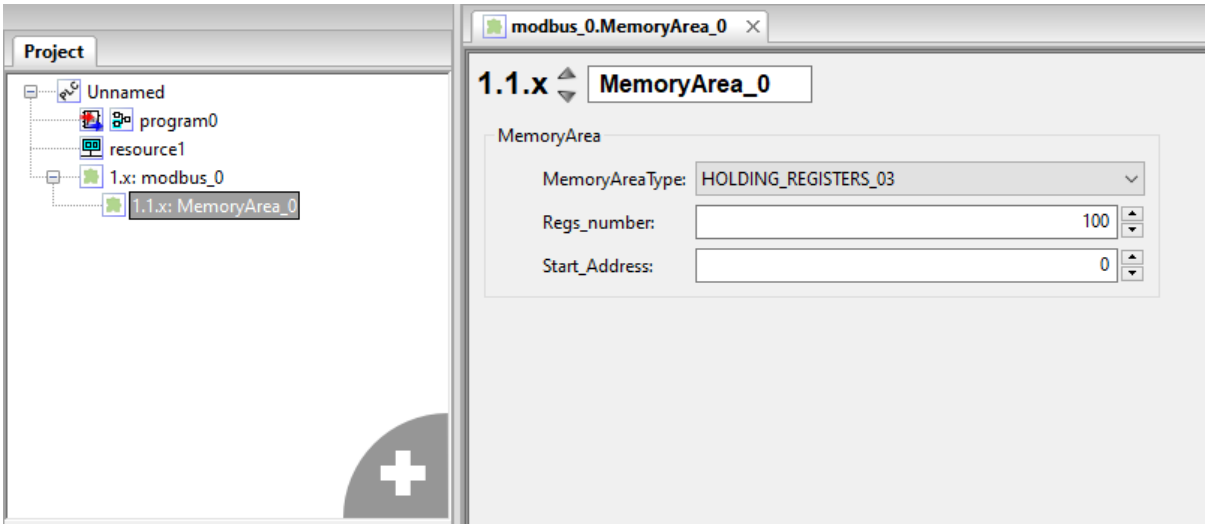
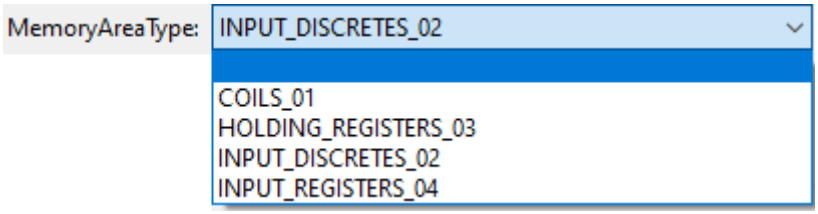



Рис. 37: Выбор параметров MemoryArea




- Количество регистров в адресном пространстве (65530).

## 1.6 Привязка глобальным переменным Modbus адреса

Для запроса с ПЛК информации хранящейся в пользовательских регистрах, необходимо интересуемым переменным присвоить Modbus адреса. Для этого в панели переменных и констант для необходимой переменной в ячейке «Location» требуется записать Modbus адрес[1] .

| # | Name | Class | Type | Location | Initial Value |
|---|---|---|---|---|---|
| 1 | REG1 | Global | UINT | %MW0.3 | |
| 2 | REG2 | Global | UINT | %MW1.1.0 | |
| 3 | REG4 | Global | UINT | %IW1.0.0.0 | |
| 4 | REG5 | Global | UINT | %QW1.0.1.0 | |
| 5 | REG3 | Global | ARRAY [0..4] OF UINT | | [0, 4, 98, 45, 9] |

Рис. 38: Пример создания переменных с Modbus адресами

Структура записи адреса приведена ниже:

**%[Форма] [Размер] [Идентификатор] . [Номер]**

[1] Для устройств, опрашивающих ПЛК BRIC имеется возможность как использовать команды чтения, так и записи регистров.

Таблица 1: **Форма регистра**

| Тип формы | Описание |
|---|---|
| Q | Глобальная переменная используется для записи Reg/Coil slave–устройства (WriteSingleCoil, WriteSingleRegister, WriteMultipleCoils, WriteMultipleRegisters) |
| I | Глобальная переменная используется для чтения Reg/Coil slave–устройства (ReadCoils, ReadInputDiscretes, ReadHoldingRegisters, ReadInputRegisters) |
| M | Глобальная переменная используется для записи чтения |

Таблица 2: **Размер переменной**

| Размер | Количество байтов | Тип данных |
|---|---|---|
| D | 4 | DINT, REAL, UDINT, DWORD |
| L | 8 | LINT, ULINT, LREAL, LWORD |
| B | 1 | BYTE, USINT, SINT |
| X | 1 | BOOL |
| W | 2 | WORD, INT, UINT |

Таблица 3: **Идентификатор элемента**

| Структура | Предназначение |
|---|---|
| X.X.X | Для ModbusRequest |
| X.X | Для MemoryArea |
| X | Для остальных, не входящих в модуль расширения |

**Номер регистра** выставляется согласно номеру в выборке ModbusRequest, при этом номер первого регистра равен 0.

## 1.7 Добавление модулей в ПЛК BRIC

В ПЛК BRIC реализовано подключение нескольких устройств линейки BRIC по межмодульной шине. Для этого необходимо добавить «Поддержка modules» в структуру проекта, созданного в OpenPLC. Для добавления нового устройства необходимо подвести курсор к созданной ветке «Поддержка modules», щелкнуть левой клавишей мыши и выбрать пункт «Добавить AO/DO/DI/AI/BRIC», как показано на рисунке ниже:

В окне добавленного модуля автоматически присваивается номер модуля расширения/ПЛК в межмодульной шине, пользователь имеет возможность изменить номер (верхний левый угол окна).

Область регистров модулей расширения/ПЛК делятся на 2 типа:

- PDO – регистры, автоматически опрашиваемые ведущим ПЛК;
- SDO – регистры, опрашиваемые ведущим ПЛК при указании пользователем.

**Внимание:** Для работы с переменными SDO необходимо в окне модуля расширения/ПЛК в ячейке «Polling» указать действие, которое необходимо совершить с переменной (**«read» – чтение, «write» – запись**). После этого их можно использовать в функциях, ФБ и программах.

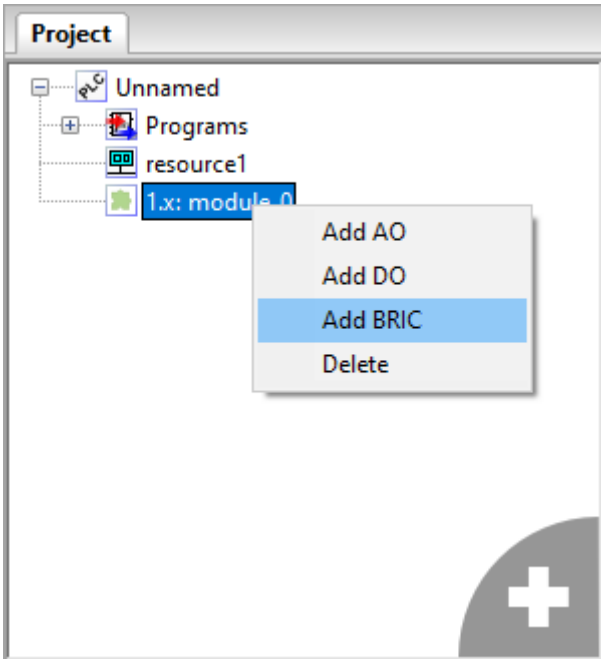



Рис. 39: Добавление в структуру проекта модуля расширения/ПЛК по межмодульной шине

~module_0.AO_1~

1.1.x AO_1

| # | Name | Type | Polling | Initial | Options | Address | Description |
|---|---|---|---|---|---|---|---|
| 1 | AO_1_mdb_addr | UINT | 0 | 0 | sdo | 0x20000010 | modbus address |
| 2 | AO_1_module_number | UINT | 0 | 0 | sdo | 0x20010010 | module ao number 0 - 1 |
| 3 | AO_1_reset_num | UINT | 0 | 0 | sdo | 0x20020010 | number of system reset |
| 4 | AO_1_last_reset | UINT | 0 | 0 | sdo | 0x20030010 | reason of last system res |
| 5 | AO_1_uart1_sets | UINT | 0 | 0 | sdo | 0x20040010 | settings MESO_UART |
| 6 | AO_1_uart3_sets | UINT | 0 | 0 | sdo | 0x20050010 | settings immodule uart |
| 7 | AO_1_internal_temp | REAL | 0 | 0 | sdo | 0x20060020 | temperature internal ser |
| 8 | AO_1_v_pwr | REAL | 0 | 0 | sdo | 0x20070020 | PWR voltage |
| 9 | AO_1_v_bat | REAL | 0 | 0 | sdo | 0x20080020 | 3V battery voltage |
| 10 | AO_1_ao_val_0 | UINT | 1 | 0 | pdor_0x200 | 0x20090010 | AO DAC value |
| 11 | AO_1_ao_val_1 | UINT | 1 | 0 | pdor_0x200 | 0x200a0010 | AO DAC value |
| 12 | AO_1_ao_val_2 | UINT | 1 | 0 | pdor_0x200 | 0x200b0010 | AO DAC value |
| 13 | AO_1_ao_val_3 | UINT | 1 | 0 | pdor_0x200 | 0x200c0010 | AO DAC value |
| 14 | AO_1_ao_state_0 | USINT | 0 | 0 | sdo | 0x200d0008 | AO state |
| 15 | AO_1_ao_state_1 | USINT | 0 | 0 | sdo | 0x200e0008 | AO state |
| 16 | AO_1_ao_state_2 | USINT | 0 | 0 | sdo | 0x200f0008 | AO state |
| 17 | AO_1_ao_state_3 | USINT | 0 | 0 | sdo | 0x20100008 | AO state |
| 18 | AO_1_ao_config_0 | USINT | 0 | 0 | sdo | 0x20110008 | AO config |
| 19 | AO_1_ao_config_1 | USINT | 0 | 0 | sdo | 0x20120008 | AO config |
| 20 | AO_1_ao_config_2 | USINT | 0 | 0 | sdo | 0x20130008 | AO config |
| 21 | AO_1_ao_config_3 | USINT | 0 | 0 | sdo | 0x20140008 | AO config |
| 22 | AO_1_sys_tick_counter | ULINT | 0 | 0 | sdo | 0x20150040 | tick in ms |

Рис. 40: Окно модуля расширения/ПЛК по межмодульной шине

## 1.8 Описание работы с языками МЭК 61131-3

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

Стандарт устанавливает пять языков программирования со следующими названиями:

- Структурированный текст (ST – Structured Text);
- Последовательные функциональные схемы (SFC – «Sequential Function Chart»);
- Диаграммы функциональных блоков (FBD – Function Block Diagram);
- Релейно-контактные схемы, или релейные диаграммы (LD – Ladder Diagram);
- Список инструкций (IL – Instruction List).

Графическими языками являются SFC, FBD, LD. Языки IL и ST являются текстовыми.

### Описание работы с текстовыми редакторами языков ST и IL

Текстовый редактор языков ST и IL позволяет создавать и редактировать алгоритмы и логику выполнения программных модулей на языках ST и IL.

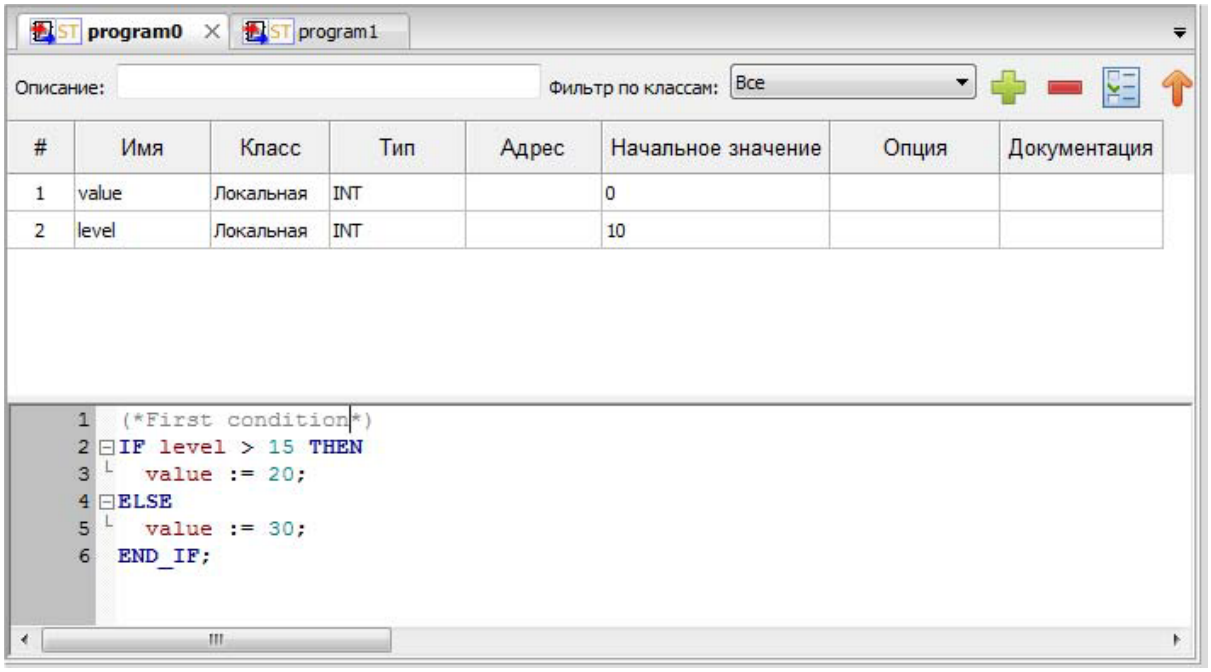



Рис. 41: Редактор языков ST и IL

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 шрифт выполняется с помощью Ctrl + <колёсико мыши>.

### Конструкции языка ST

К конструкциям языка ST относятся:

1. Арифметические операции
2. Логические (побитовые) операции
3. Операции сравнения
4. Операция присвоения
5. Конструкция IF – ELSE
6. Цикл FOR
7. Цикл WHILE
8. Конструкция CASE.

**1. Арифметические операции**

К арифметическим операциям относятся:

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

**2. Логические (побитовые) операции**

К данным операциям относятся:

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

**3. 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.

**4. Операция присвоения**

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 (принимающая сторона) может быть использована только переменная. Правая часть может содержать выражение или константу.

**5. Конструкция IF – ELSE**

Конструкция IF–ELSE имеет следующий формат:

```
IF<логическое выражение>THEN<выполняемое выражение>

[ELSE<выполняемое выражение>]

END_IF;
```

Например:

```
IF Vary<> 0

THEN Var4 := 1

ELSE Var4 := 10;

END_IF;
```

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

Например:

```
IF Var5 > 10 THEN

IF Var5 < Var2 + 1

THEN Var5 := 10;

ELSE Var5 := 0;

END_IF;

END_IF;
```

**6. Цикл FOR**

Служит для задания цикла с фиксированным количеством итераций.

Формат конструкции следующий:

```
FOR<переменная управления> := <выражение1> TO <выражение2>

[BY< выражение3>] DO

<выполняемое выражение>

END_FOR;
```

---

**Примечание:** Выражения <выражение1> … <выражение1> вычисляются до входа в цикл, поэтому изменения значений переменных, входящих в любое из этих выражений, не приведет к изменению числа итераций.

---

При задании условий цикла считается, что <переменная управления>, <выражение1> …<выражение3> имеют тип INT. Выход из цикла будет произведен в том случае, если значение переменной цикла превысит значение <выражение3>.

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

var3 := var3 + 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;
```

Другой пример:

```
k := 10;

FOR i := 1 TO k / 2 DO

k := 20;

END_FOR;
```

На строках между DO и END_FOR производится изменение переменной k, при этом эти строки выполнятся пять раз.

Например:

```
FOR i := 1 TO 5 DO

i := 55;

END_FOR;
```

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

**7. 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<выполняемое выражение>

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.

**8. Конструкция CASE**

Данная конструкция служит для организации выбора из диапазона значений.

Формат конструкции следующий:

```
CASE<выражение>OF

CASE_ELEMENT {CASE_ELEMENT}

[ELSE<выполняемое выражение>]

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 <выражение> 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 <выражение> может быть только целым.

Например:

```
CASE k OF

1:

k := k * 10;

2..5:

k := k * 5;

i := 0;

6, 9..20:

k := k – 1;

ELSE

k := 0;

i := 1;

END_CASE;
```

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.

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

### Конструкции языка IL

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

Таблица 4: Операторы языка IL

| Опера-тор | 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 |
| ADD | Сложение аккумулятора и операнда, результат записывается в аккумулятор |
| SUB | Вычитание операнда из аккумулятора, результат записывается в аккумулятор |
| MUL | Умножение аккумулятора на операнд, результат записывается в аккумулятор |
| DIV | Деление аккумулятора на операнд, результат записывается в аккумулятор |
| GT | Значение аккумулятора сравнивается со значением операнда(>(greater than)). Значение (TRUE или FALSE) записывается в аккумулятор |
| GE | Значение аккумулятора сравнивается со значением операнда (>=greater than or equal). Значение (TRUE или FALSE) записывается в аккумулятор |
| EQ | Значение аккумулятора сравнивается со значением операнда (=(equal)). Значение (TRUE или FALSE) записывается в аккумулятор |
| NE | Значение аккумулятора сравнивается со значением операнда <>(not equal). Значение (TRUE или FALSE) записывается в аккумулятор |
| LE | Значение аккумулятора сравнивается со значением операнда (<=(less than or equal to)). Значение (TRUE или FALSE) записывается в аккумулятор |
| LT | Значение аккумулятора сравнивается со значением операнда (<(less than)). Значение (TRUE или FALSE) записывается в аккумулятор |
| JMP | Переход к метке |
| JMPC | Переход к метке при условии, что значение аккумулятора TRUE |
| JMPCN | Переход к метке при условии, что значение аккумулятора FALSE |

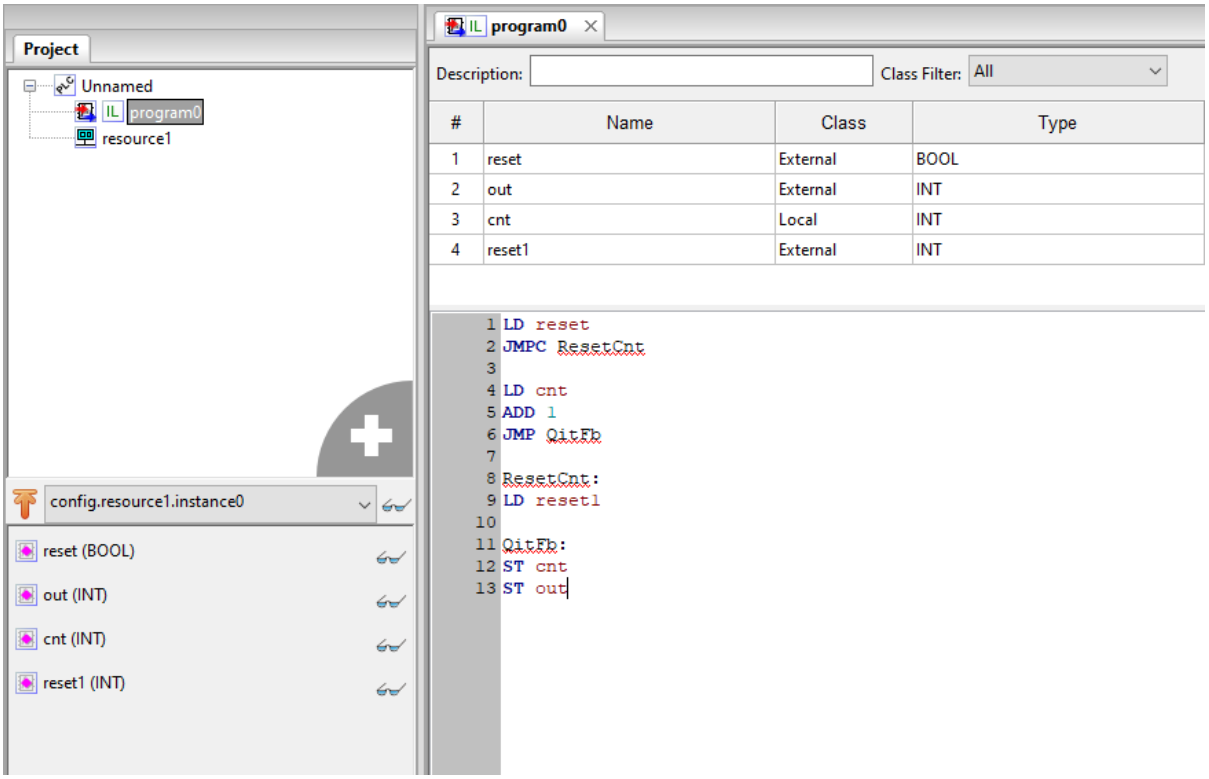



Рис. 42: Пример программы на языке IL

## Описание работы с графическими редакторами языков FBD, LD SFC

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

## Конструкции языка FBD

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


Рис. 43: Панель редактирования FBD диаграмм

С помощью данной панели можно добавить все элементы языка FBD. Назначение каждой кнопки описано в таблице ниже.

Таблица 5: Кнопки панели редактирования FBD диаграммы

| Внешний вид кнопки | Наименование кнопки | Функции кнопки |
|---|---|---|
|  | Выделение объектов на диаграмме | Перевод указателя мыши в состояние, при котором можно осуществлять выделение объектов в редакторе |
|  | Перемещение диаграммы | Перевод указателя мыши в состояние, при котором можно изменять размеры редактора |
|  | Создать новый комментарий | Вызов диалога создания комментария |
|  | Добавить переменную | Вызов диалога добавления переменной |
|  | Добавить ФБ | Вызов диалога добавления функционального блока |
|  | Добавить соединение | 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 при помощи нажатия правой клавишей мыши и выбора пункта «Add» (Добавить), в котором будет: «Block» (Блок), «Variable» (Переменная), «Connection» (Соединение), «Comment» (Комментарий).

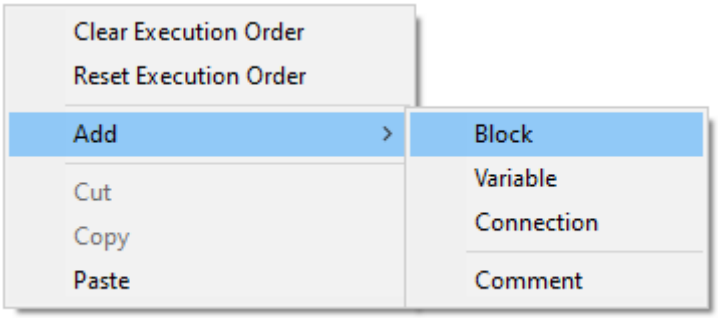



Рис. 44: Всплывающее меню редактора языка FBD

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

В данном диалоге приведено краткое описание функционального блока в нижнем левом окне и предоставлена возможность задать некоторые свойства (имя, количество входов, порядок выполнения).

Добавление блока происходит путем перетаскивания необходимой функции из панели библиотеки функций и функциональных блоков, через окно «Block Properties» или путем копирования существующего блока.

Переменные добавляются из панели переменных и констант с помощью перетаскивания левой клавишей мыши за область (#), указанную на рисунке ниже в область редактирования FBD диаграмм или через диалог «Variable Properties» (Свойства переменной), вызванный через всплывающего меню редактора языка FBD.

В данном диалоге можно задать порядок выполнения переменной и изменить её класс («Input» (Входная), «Output» (Выходная), «InOut» (Входная/Выходная)).

Когда необходимо передать выходное значение одного функционального блока на один из входов другого для удобства можно использовать элемент «Connection» (Соединение). На схемах с большим количеством функциональных блоков элемент «Connection» позволяет избежать пересечения прямых соединений, которые приводит к тому, что схема становится менее понятной.

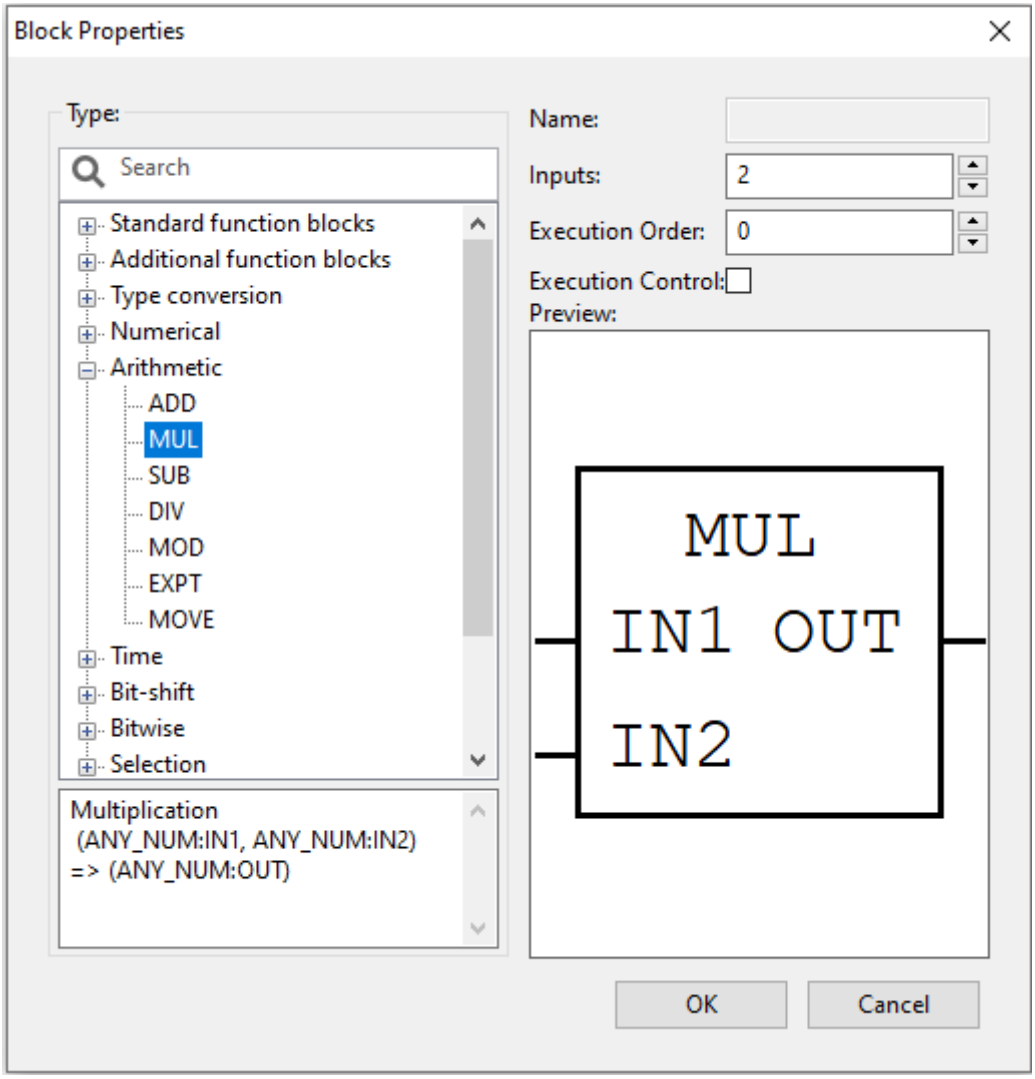



Рис. 45: Свойства функционального блока

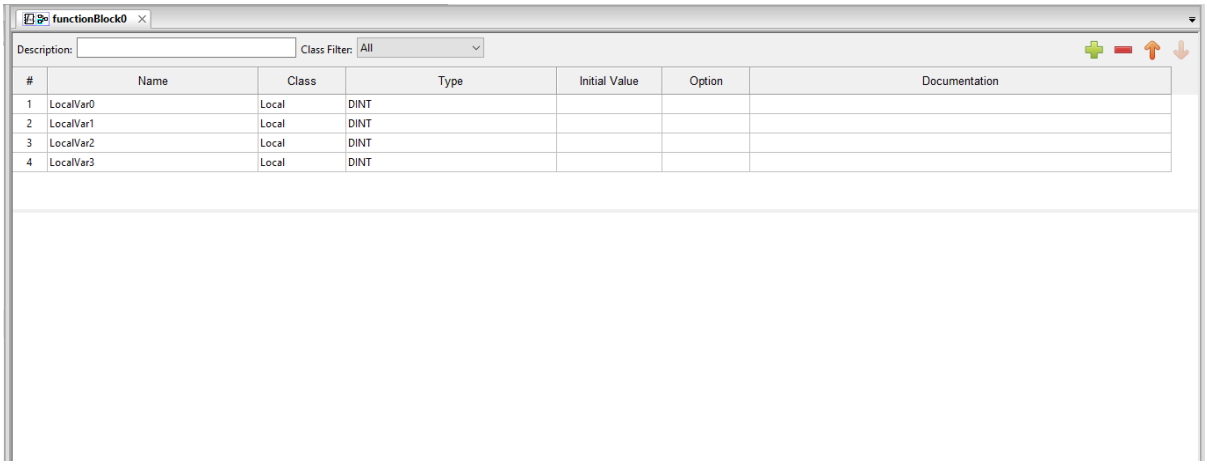



Рис. 46: Добавление переменной из панели переменных и констант

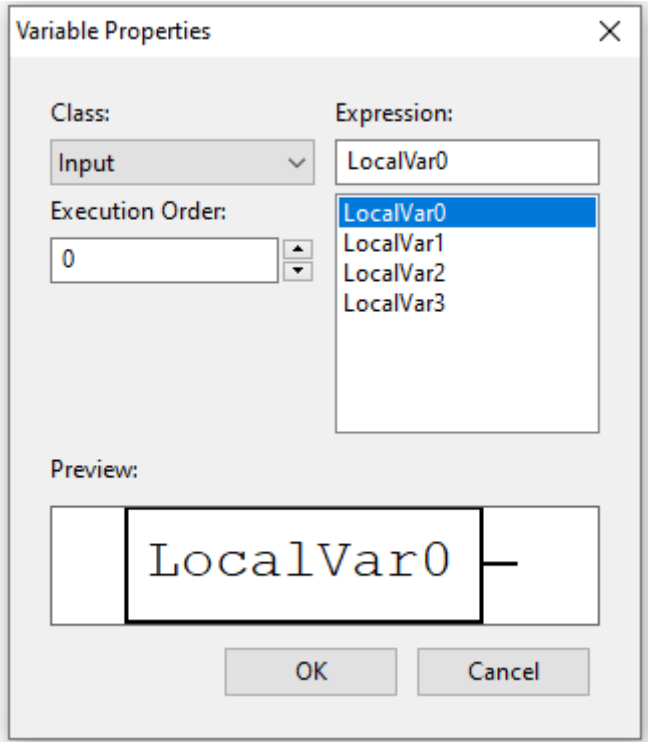



Рис. 47: Свойства переменной

После выбора добавления элемента «Connection» появится диалог «Connection Properties» (Свойства соединения).

В данном диалоге можно выбрать тип соединения: «Connector» (Выходное соединение) – для выходного значения, «Continuation» (Входное соединение) – для входного значения, а так же необходимо указать имя данного соединения. Ниже представлен пример использования соединений.

Редактор FBD диаграмм позволяют добавлять комментарии на диаграмму. После выбора на панели редактирования комментария и добавления его в область редактирования появится диалог для ввода текста комментария.

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 ФБ прямоугольника, тем раньше данная функция или функциональный будет выполнен. Порядок выполнения может быть изменён вручную с помощью диалога свойств опцией «Execution Order» (Порядок выполнения).

### Конструкции языка LD

Язык LD представляет собой графическую форму записи логических выражений в виде контактов и катушек реле.

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

С помощью данной панели можно добавить все элементы языка LD. В таблице ниже приведено описание кнопок данной панели.

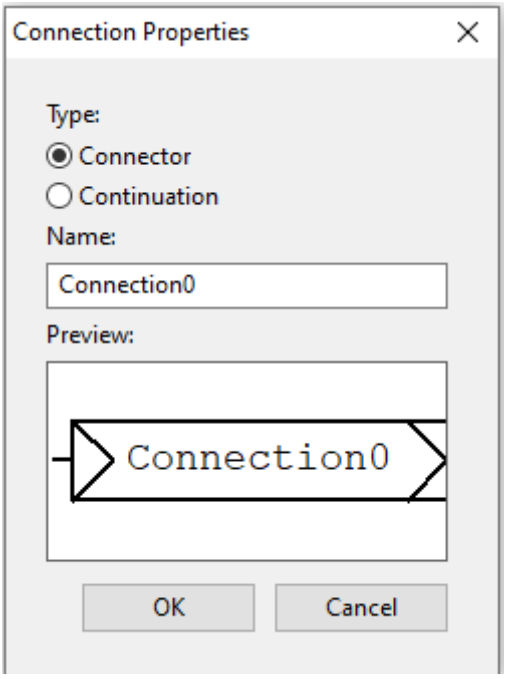



Рис. 48: Диалог добавления соединения для FBD

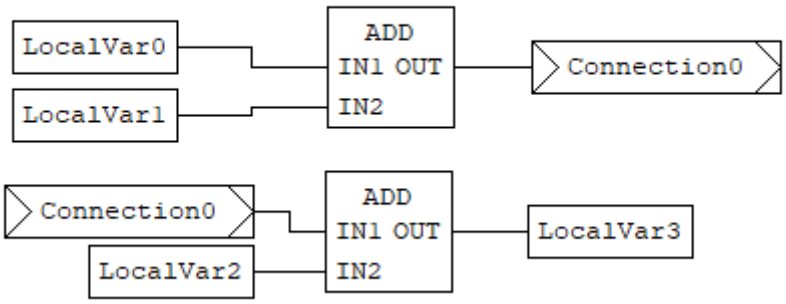



Рис. 49: Пример FBD диаграммы с использованием соединений

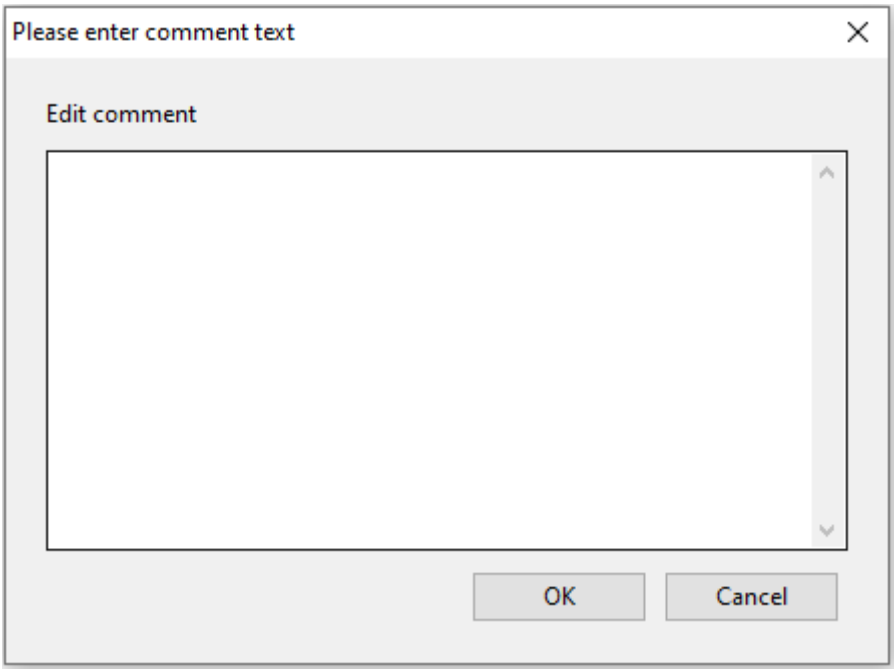



Рис. 50: Диалог добавления комментария


Рис. 51: Панель редактирования LD диаграмм

Таблица 6: Кнопки панели редактирования LD диаграммы :header: «Внешний вид кнопки»,»Наименование кнопки», «Функции кнопки» :widths: 10, 40, 60

| | | | |
|---|---|---|---|
| | Выделение объектов на диаграмме | Перевод указателя мыши в состояние, при котором можно осуществлять выделение объектов в редакторе | |
| | Перемещение диаграммы | Перевод указателя мыши в состояние, при котором можно изменять размеры редактора | |
| | Создать новую шину питания | Вызов диалога создания новой шины питания | |
| VAR | Создать новую катушку | Вызов диалога создания новой катушки[6] | |
| VAR | Создать новый контакт | Вызов диалога создания нового контакта[7] | |
| CMT | Создать новый комментарий | Вызов диалога создания комментария | |
| VAR | Добавить переменную | Вызов диалога добавления переменной | |
| FB | Добавить ФБ | Вызов диалога добавления функционального блока | |
| | Добавить соединение | Вызов диалога добавления соединения | |
| | | | |

При добавлении шины питания появится диалог «Power Rail Properties» (Свойства шины питания).

В данном диалоге указываются следующие свойства:

- «Type» (тип шины питания) («Left PowerRail» (шина питания слева), «Right PowerRail» (шина питания справа));
- «Pin number»( количество контактов на добавляемой шине питания).

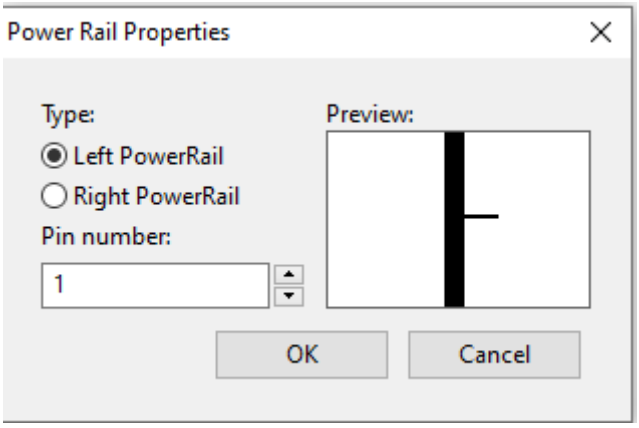



Рис. 52: Свойство шины питания

При добавлении контакта на LD диаграмму появится диалог «Edit Contact Values» (Редактирование значения контакта).

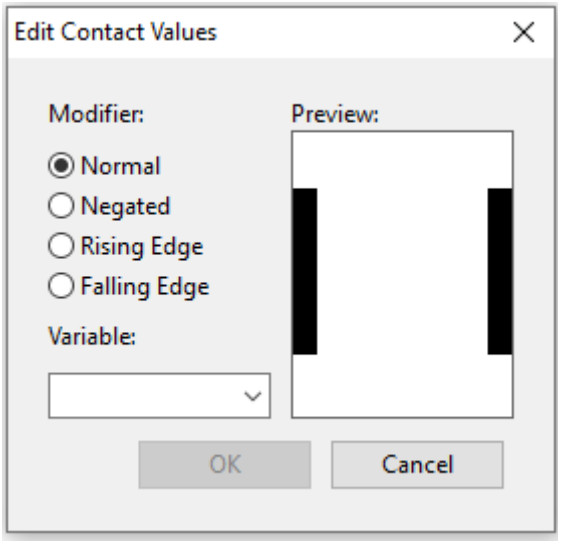



Рис. 53: Редактирование контакта

Данный диалог позволяет определить модификатор данного контакта:

- «Normal» (Нормальный);
- «Negated» (Инверсный);
- «Rising Edge» (Нарастание фронта);
- «Falling Edge» (Спад фронта).

Диалог позволяет выбрать из списка переменную, к которой он связан. Переменные должны быть определены в панели переменных и констант и иметь тип переменной BOOL.

[6] Устанавливает соответствующий битовый объект в значение
[7] Контакт замкнут

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

При добавлении катушки на LD диаграмму появится диалог «Edit Coil Values» (Редактирование значения катушки).

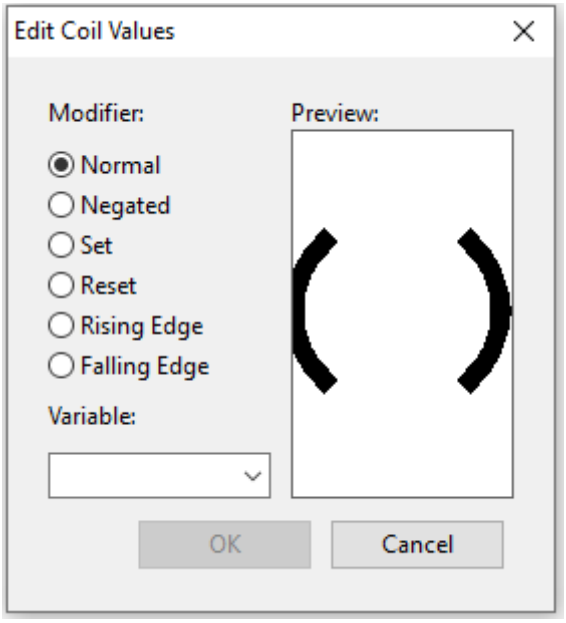



Рис. 54: Редактирование катушки

В данном диалоге можно определить модификатор данного контакта:

- «Normal» (Нормальный);
- «Negated» (Инверсный);
- «Set» (Установка);
- «Reset» (Сброс);
- «Rising Edge» (Нарастание фронта);
- «Falling Edge» (Спад фронта).

### Конструкции 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.

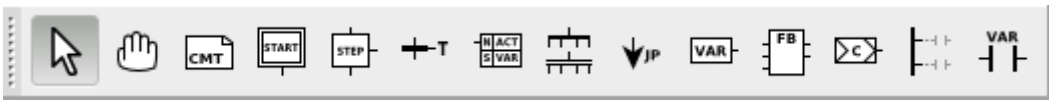



Рис. 55: Панель редактирования SFC диаграмм

В таблице ниже приведено описание кнопок данной панели.

Таблица 7: Кнопки панели редактирования SFC диаграммы

| Внешний вид кнопки | Наименование кнопки | Функции кнопки |
|---|---|---|
|  | Выделение объектов на диаграмме | Перевод указателя мыши в состояние, при котором можно осуществлять выделение объектов в редакторе |
|  | Перемещение диаграммы | Перевод указателя мыши в состояние, при котором можно изменять размеры редактора |
|  | Создать новый комментарий | Вызов диалога создания комментария |
|  | Создать новый начальный шаг | Вызов диалога редактирования шага |
|  | Создать новый шаг | Вызов диалога редактирования шага |
|  | Создать новый переход | Создать новый переход |
|  | Создать новый блок действий | Вызов диалога редактирования блока действий |
|  | Создать новую дивергенцию | Вызов диалога создания новой дивергенции и конвергенции |
|  | Создать новый «прыжок» | Вызов диалога создания«прыжка» |
|  | Добавить переменную | Вызов диалога добавления переменной |
|  | Добавить ФБ | Вызов диалога добавления функционального блока |
|  | Добавить соединение | Вызов диалога добавления соединения |
|  | Создать новую шину питания | Вызов диалога создания новой шины питания |
|  | Создать новый контакт | Вызов диалога создания нового контакта |

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

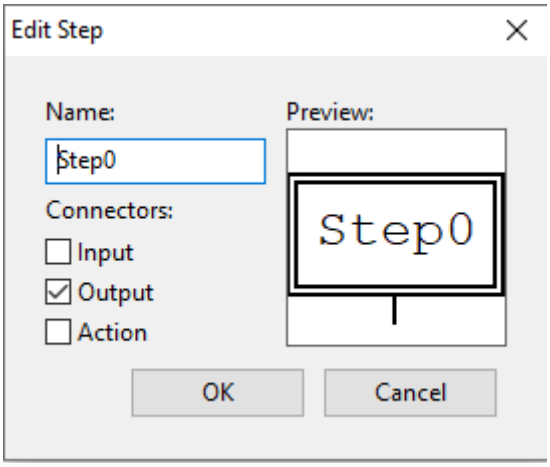



Рис. 56: Диалоги редактирования шага инициализации и обычного шага SFC диаграммы

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.

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

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

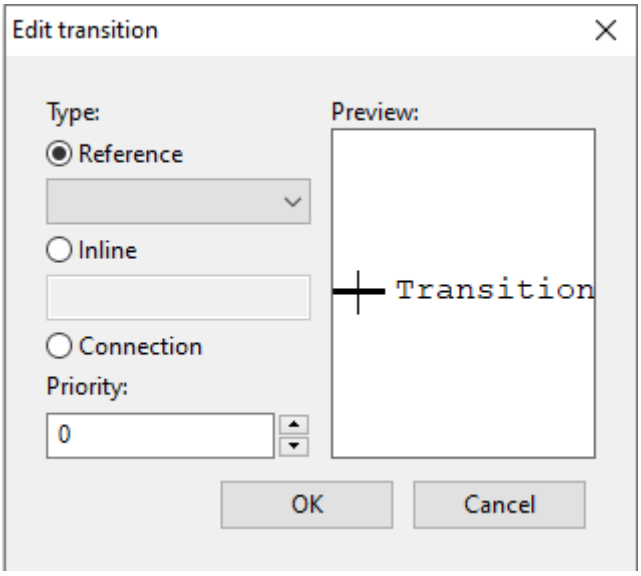



Рис. 57: Диалог редактирования перехода для SFC диаграммы

- «Reference» (Ссылка);
- «Inline» (Встроенный код);
- «Connection» (Соединение).

При выборе типа перехода «Ссылка» в открывающемся списке 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.

При выборе типа перехода «Inline» условие перехода можно написать в виде выражения на языке ST.

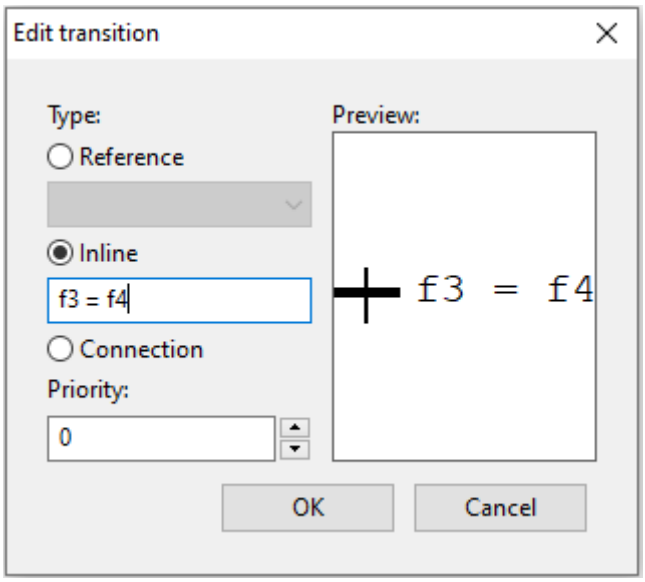



Рис. 58: Условие перехода в виде встроенного кода, написанного на языке ST

Реализация перехода таким способом удобна в случае, когда необходимо короткое условие, например: переменные «f3» и «f4» типа INT равны. Встроенный код для такого условия выглядит следующим образом:

```
f3 = f4
```

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 «Connection», в качестве условия перехода можно использовать выходные значения элементов языка FBD или LD.

При выборе типа перехода «Connection», 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.

При добавлении блока действий на диаграмму появится диалог «Edit action block properties» (Редактировать свойство блока действий).

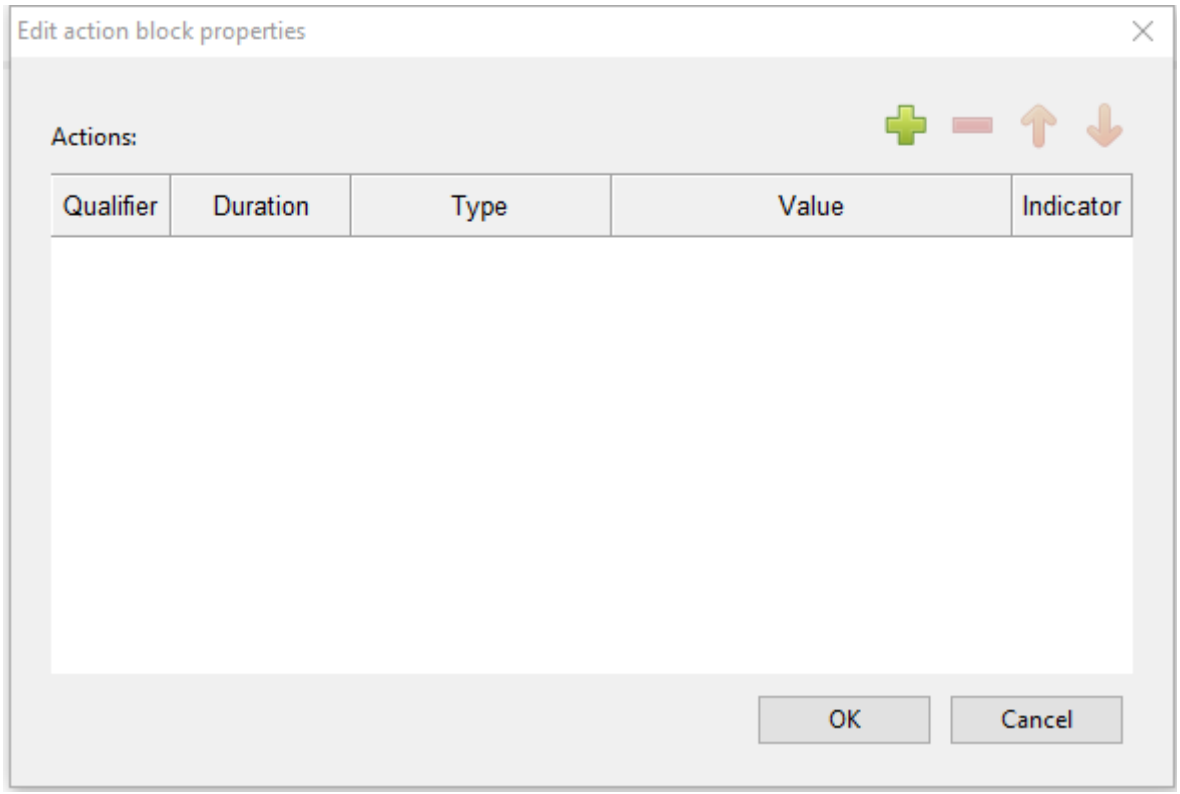



Рис. 59: Диалог «Редактировать свойство блока действий»

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

- «Qualifier» (Квалификатор);
- «Duration» (Продолжительность);
- «Type» (Тип): «Action» (Действие), «Variable» (Переменная), «Inline» (Встроенный код);
- «Value» (Значение);
- «Indicator» (Индикатор).

Поле «Qualifier» определяет момент времени, когда действие начинается, сколько времени продолжается и когда заканчивается. Выбрать квалификатор можно из списка.

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

Таблица 8: Квалификаторы действий SFC диаграммы

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

Поле «Duration» необходимо для установки интервала времени необходимого для некоторых квалификаторов, описанных выше.

«Type» 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.

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

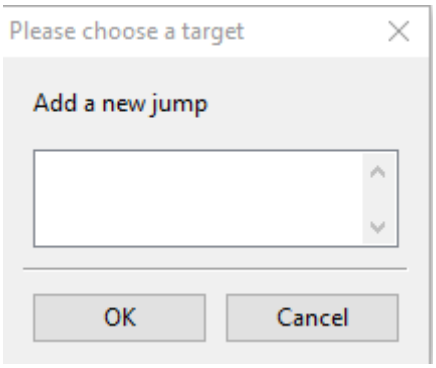



Рис. 60: Диалог добавления «прыжка»

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

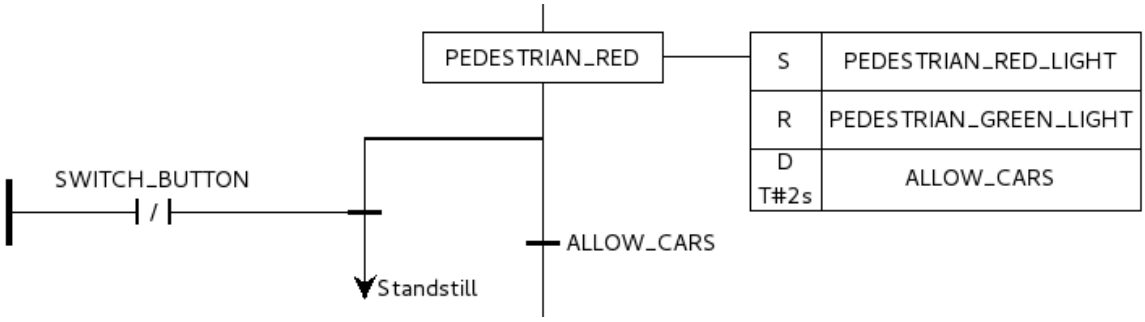



Рис. 61: Пример алгоритма, напианного на языке SFC

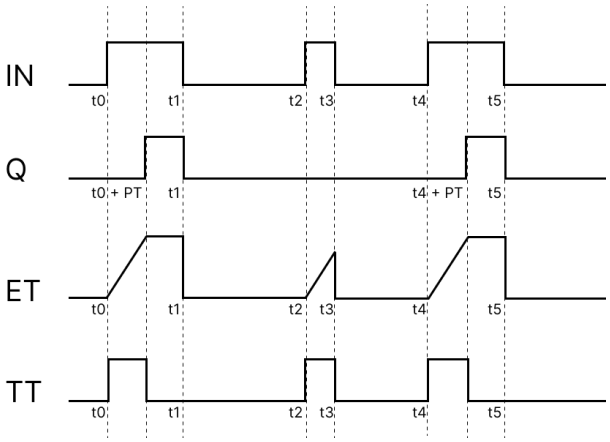
## 1.9 Описание функциональных блоков библиотеки ИСР BEREMIZ для ПЛК BRIC

**Внимание:** Для использования стандартных и дополнительных функциональных блоков в языке ST необходимо перетащить необходимый из библиотеки в зону перечисления переменных и дать наименование.

### BRIC Library

### Блок UTON

Данный ФБ представляет собой таймер с задержкой включения.




| Вход | Тип | Описание |
|---|---|---|
| IN | BOOL | Подача импульса |
| PT | ULINT | Время задержки в формате ULINT |
| **Выход** | **Тип** | **Описание** |
| Q | BOOL | Если ET = PV и IN = 1, то Q = 1, иначе Q = 0 |
| ET | ULINT | Счетчик времени считает пока ET < PV и IN = 1 |
| TT | BOOL | TRUE, пока ET < PV |

Форма записи на языке ST:

```
FB(
 IN := (*BOOL*),
 PT := (*ULINT*),
```

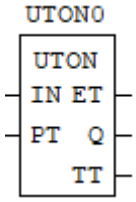
(continues on next page)

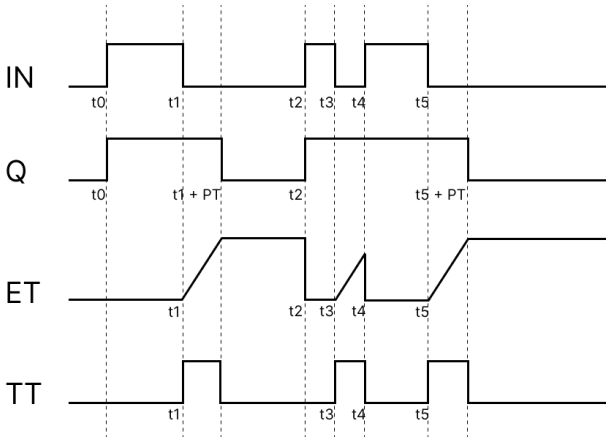
(продолжение с предыдущей страницы)

```
 ET => (*ULINT*),
 Q => (*BOOL*),
 TT => (*BOOL*));
```

### Блок UTOF

Данный ФБ представляет таймер, реализующий задержку выключения. Когда вход изменяется с TRUE на FALSE (задний фронт) проходит определенное время, пока выход не станет FALSE.



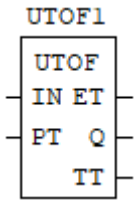

| Вход | Тип | Описание |
|---|---|---|
| IN | BOOL | Подача импульса |
| PT | ULINT | Время задержки в формате ULINT |
| **Выход** | **Тип** | **Описание** |
| Q | BOOL | Если IN = TRUE => Q = TRUE. При переходе IN в FALSE, если ET = PV, то Q = 0, иначе Q = 1 |
| ET | ULINT | Счетчик времени считает пока ET < PV и был переход IN из TRUE в FALSE |
| TT | BOOL | TRUE, пока ET < PV |

Форма записи на языке ST:

```
FB(
 IN := (*BOOL*),
 PT := (*ULINT*),
 ET => (*ULINT*),
 Q => (*BOOL*),
 TT => (*BOOL*));
```

### Блок WORD_ON_BOOL

ФБ WORD_ON_BOOL выводит значение входной переменной WORD в битах.



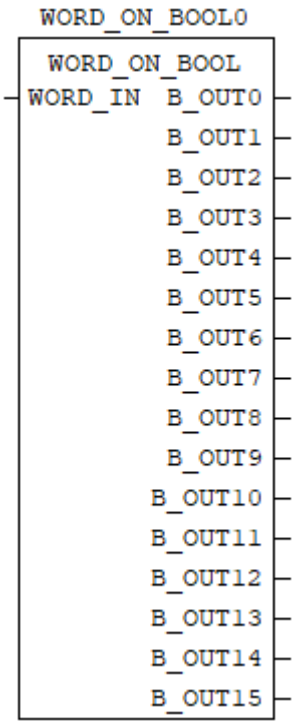

| Вход | Тип | Описание |
|---|---|---|
| WORD_IN | WORD | Входная переменная |
| **Выход** | **Тип** | **Описание** |
| B_OUT0 | BOOL | Вывод значения нулевого бита |
| B_OUT1 | BOOL | Вывод значения первого бита |
| B_OUT2 | BOOL | Вывод значения второго бита |
| B_OUT3 | BOOL | Вывод значения третьего бита |
| B_OUT4 | BOOL | Вывод значения четвертого бита |
| B_OUT5 | BOOL | Вывод значения пятого бита |
| B_OUT6 | BOOL | Вывод значения шестого бита |
| B_OUT7 | BOOL | Вывод значения седьмого бита |
| B_OUT8 | BOOL | Вывод значения восьмого бита |
| B_OUT9 | BOOL | Вывод значения девятого бита |
| B_OUT10 | BOOL | Вывод значения десятого бита |
| B_OUT11 | BOOL | Вывод значения одиннацатого бита |
| B_OUT12 | BOOL | Вывод значения двенадцатого бита |
| B_OUT13 | BOOL | Вывод значения тринадцатого бита |
| B_OUT14 | BOOL | Вывод значения четырнадцатого бита |
| B_OUT15 | BOOL | Вывод значения пятнадцатого бита |

Форма записи на языке ST:

```
FB(
 WORD_IN := (*WORD*),
 B_OUT0 => (*BOOL*),
 B_OUT1 => (*BOOL*),
 B_OUT2 => (*BOOL*),
 B_OUT3 => (*BOOL*),
 B_OUT4 => (*BOOL*),
 B_OUT5 => (*BOOL*),
 B_OUT6 => (*BOOL*),
 B_OUT7 => (*BOOL*),
 B_OUT8 => (*BOOL*),
 B_OUT9 => (*BOOL*),
 B_OUT10 => (*BOOL*),
 B_OUT11 => (*BOOL*),
 B_OUT12 => (*BOOL*),
 B_OUT13 => (*BOOL*),
 B_OUT14 => (*BOOL*),
 B_OUT15 => (*BOOL*));
```

### Блок BOOL_ON_WORD

ФБ BOOL_ON_WORD выводит значение в формате WORD входных булевых переменных.




| Вход | Тип | Описание |
|---|---|---|
| B_IN0 | BOOL | Ввод значения нулевого бита |
| B_IN1 | BOOL | Ввод значения первого бита |
| B_IN2 | BOOL | Ввод значения второго бита |
| B_IN3 | BOOL | Ввод значения третьего бита |
| B_IN4 | BOOL | Ввод значения четвертого бита |
| B_IN5 | BOOL | Ввод значения пятого бита |
| B_IN6 | BOOL | Ввод значения шестого бита |
| B_IN7 | BOOL | Ввод значения седьмого бита |
| B_IN8 | BOOL | Ввод значения восьмого бита |
| B_IN9 | BOOL | Ввод значения девятого бита |
| B_IN10 | BOOL | Ввод значения десятого бита |
| B_IN11 | BOOL | Ввод значения одиннацатого бита |
| B_IN12 | BOOL | Ввод значения двенадцатого бита |
| B_IN13 | BOOL | Ввод значения тринадцатого бита |
| B_IN14 | BOOL | Ввод значения четырнадцатого бита |
| B_IN15 | BOOL | Ввод значения пятнадцатого бита |
| **Выход** | **Тип** | **Описание** |
| WORD_OUT | WORD | Выходная переменная |

Форма записи на языке ST:

```
FB(
 B_IN0 := (*BOOL*),
 B_IN1 := (*BOOL*),
 B_IN2 := (*BOOL*),
 B_IN3 := (*BOOL*),
 B_IN4 := (*BOOL*),
 B_IN5 := (*BOOL*),
```

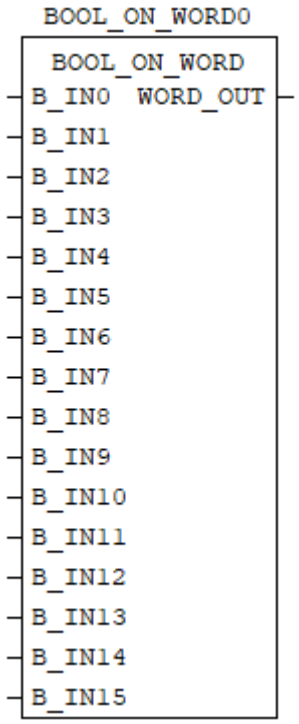
(continues on next page)

(продолжение с предыдущей страницы)

```
 B_IN6 := (*BOOL*),
 B_IN7 := (*BOOL*),
 B_IN8 := (*BOOL*),
 B_IN9 := (*BOOL*),
 B_IN10 := (*BOOL*),
 B_IN11 := (*BOOL*),
 B_IN12 := (*BOOL*),
 B_IN13 := (*BOOL*),
 B_IN14 := (*BOOL*),
 B_IN15 := (*BOOL*),
 WORD_OUT => (*WORD*));
```

### Блок SETBIT

Данный ФБ записывает значение BIT_VALUE в любом из битов в входной переменной IN (DWORD).



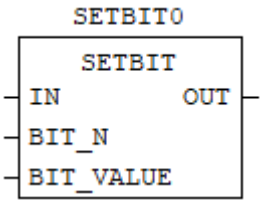

| Вход | Тип | Описание |
|---|---|---|
| IN | DWORD | Входная переменная |
| BIT_N | USINT | Выбор необходимого номера бита |
| BIT_VALUE | BOOL | Значение бита |
| **Выход** | **Тип** | **Описание** |
| OUT | DWORD | Выходная переменная |

Форма записи на языке ST:

```
FB(
 IN := (*DWORD*),
 BIT_N := (*USINT*),
 BIT_VALUE := (*BOOL*),
 OUT => (*DWORD*));
```

### Блок GETBIT

Данный ФБ прочитывает значение булевой переменной в определенном из битов IN (DWORD).




| Вход | Тип | Описание |
|---|---|---|
| IN | DWORD | Входная переменная |
| BIT_N | USINT | Выбор необходимого номера бита |
| **Выход** | **Тип** | **Описание** |
| OUT | BOOL | Выходное значение бита |

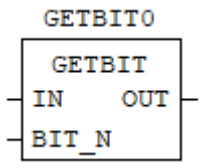
Форма записи на языке ST:

```
FB(
 IN := (*DWORD*),
 BIT_N := (*USINT*),
 OUT => (*BOOL*));
```

### DO Functional blocks

### Блок READ_DO_SC

ФБ READ_DO_SC предоставляет информацию о состоянии короткого замыкания на дискретных выходах.




| Выход | Тип | Описание |
|---|---|---|
| DO_SC_FLAG | USINT | Флаг обнаруживания короткого замыкания по каналу. Если КЗ обнаружено, то управление каналом блокируется |
| DO_SC_EN | USINT | Флаг выставленной программной защиты от короткого замыкания |

Форма записи на языке ST:

```
FB(
 DO_SC_FLAG => (*USINT*),
 DO_SC_EN => (*USINT*));
```

---

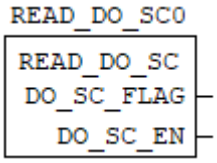
**Примечание:** Логическое состояние КЗ на дискретных каналах определяется как значение младшего полубайта

| Канал DO | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Значение переменной | 1 | 2 | 4 | 8 |

---

### Блок READ_DO

ФБ READ_DO предоставляет информацию о состоянии дискретных выходов.




| Вы-ход | Тип | Описание |
|---|---|---|
| DO_OUT | USINT | Чтение значения состояния дискретных выходов |

Форма записи на языке ST:

```
FB(
 DO_OUT => (*USINT*));
```

---

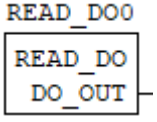
**Примечание:** Логическое состояние каналов определяется как значение младшего полубайта

| Канал DO | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Значение переменной | 1 | 2 | 4 | 8 |

---

### Блок WRITE_DO

ФБ WRITE_DO согласно маске DO_MASK и подаче напряжений на соответствующие дискретные выхода DO_VALUE (выходы прописываются побитово).




| Вход | Тип | Описание |
|---|---|---|
| DO_VALUE | USINT | Запись значения состояния дискретных выходов |
| DO_MASK | USINT | Запись маски, разрешающей изменять состояние дискретных выходов |

Форма записи на языке ST:

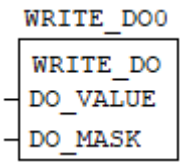
```
FB(
 DO_VALUE := (*UINT*),
 DO_MASK := (*UINT*));
```

**Примечание:** Логическое состояние каналов определяется как значение младшего полубайта

| Канал DO | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Значение переменной | 1 | 2 | 4 | 8 |

### Блок WRITE_DO_SC

ФБ WRITE_DO_SC устанавливает номера дискретных выходов, на которых включить программную защиту от КЗ и дискретные выхода на которых сработала программная защита от КЗ.




| Вход | Тип | Описание |
|---|---|---|
| DO_SC_FLAG | UINT | Запись перечисления дискретных выходов, на которых сработала аппаратная защита от короткого замыкания (использовать только возможность сброса) |
| DO_SC_EN | UINT | Запись перечисления дискретных выходов, на которых сработала программная защита от короткого замыкания (использовать только возможность сброса) |

Форма записи на языке ST:

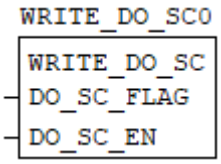
```
FB(
 DO_SC_FLAG := (*UINT*),
 DO_SC_EN := (*UINT*));
```

**Примечание:** Логическое состояние КЗ на дискретных каналах определяется как значение младшего полубайта

| Канал DO | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Значение переменной | 1 | 2 | 4 | 8 |

### Блок WRITE_DO_PWM_FREQ

ФБ WRITE_DO_PWM устанавливает ШИМ дискретных выходов в диапазоне от 0 до 10000 Гц.




| Вход | Тип | Описание |
|---|---|---|
| DO_PWM_FREQ | UINT | Частота ШИМ подаваемая на каналы дискретных выходов, Гц |

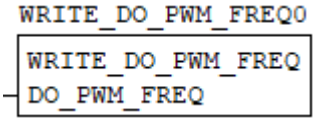
Форма записи на языке ST:

```
FB(
 DO_PWM_FREQ := (*UINT*));
```

**Примечание:** Значение PWM Frequency задается одно для всех каналов

### Блок WRITE_DO_PWM_CTRL

ФБ WRITE_DO_PWM_CTRL устанавливает для выбранного дискретного выхода скважность сигнала. Указывается диапазон от 10 до 90%.




| Вход | Тип | Описание |
|---|---|---|
| DO_NUMBER | USINT | Номер канала дискретного выхода ПЛК BRIC |
| DO_PWM_DUTY | UINT | Указание скважности канала |
| DO_PWM_RUN | BOOL | Включение ШИМ |

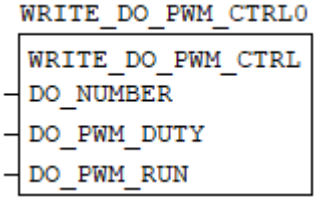
Форма записи на языке ST:

```
FB(
 DO_NUMBER := (*USINT*),
 DO_PWM_DUTY := (*UINT*),
 DO_PWM_RUN := (*BOOL*));
```

## SOFI Function blocks

### Блок DWORD_TO_FLOAT

Данный ФБ преобразовывает значение типа данных DWORD в тип данных с плавающей точкой (float).



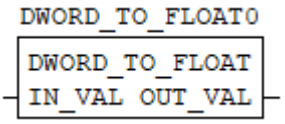
| Вход | Тип | Описание |
|---|---|---|
| IN_VAL | DWORD | Входная переменная |
| **Выход** | **Тип** | **Описание** |
| OUT_VAL | REAL | Выходная переменная |

Форма записи на языке ST:

```
FB(
 IN_VAL := (*DWORD*),
 OUT_VAL => (*REAL*));
```

### Блок WORDS_TO_FLOAT

Данный ФБ преобразовывает значение двух типов данных WORD в тип данных с плавающей точкой (float).



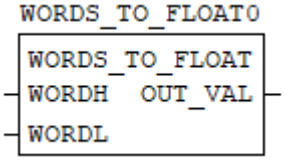

| Вход | Тип | Описание |
|---|---|---|
| WORDH | WORD | Входная переменная (старший байт) |
| WORDL | WORD | Входная переменная (младший байт) |
| **Выход** | **Тип** | **Описание** |
| OUT_VAL | REAL | Выходная переменная |

Форма записи на языке ST:

```
FB(
 WORDH := (*WORD*),
 WORDL := (*WORD*),
 OUT_VAL => (*REAL*));
```

### Блок PARSING_UINT

ФБ PARSING_UINT выводит значение входной переменной UINT в битах.



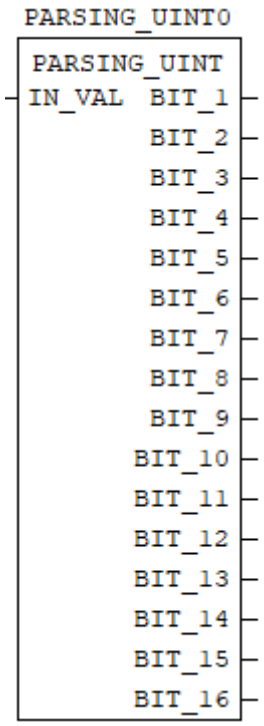
| Вход | Тип | Описание |
|---|---|---|
| IN_VAL | UINT | Обрабатываемая переменная |
| **Выход** | **Тип** | **Описание** |
| BIT_1 | BOOL | Вывод значения первого бита |
| BIT_2 | BOOL | Вывод значения второго бита |
| BIT_3 | BOOL | Вывод значения третьего бита |
| BIT_4 | BOOL | Вывод значения четвертого бита |
| BIT_5 | BOOL | Вывод значения пятого бита |
| BIT_6 | BOOL | Вывод значения шестого бита |
| BIT_7 | BOOL | Вывод значения седьмого бита |
| BIT_8 | BOOL | Вывод значения восьмого бита |
| BIT_9 | BOOL | Вывод значения девятого бита |
| BIT_10 | BOOL | Вывод значения десятого бита |
| BIT_11 | BOOL | Вывод значения одиннацатого бита |
| BIT_12 | BOOL | Вывод значения двенадцатого бита |
| BIT_13 | BOOL | Вывод значения тринадцатого бита |
| BIT_14 | BOOL | Вывод значения четырнадцатого бита |
| BIT_15 | BOOL | Вывод значения пятнадцатого бита |
| BIT_16 | BOOL | Вывод значения шестнадцатого бита |

Форма записи на языке ST:

```
FB(
 IN_VAL := (*UINT*),
 BIT_1 => (*BOOL*),
 BIT_2 => (*BOOL*),
 BIT_3 => (*BOOL*),
 BIT_4 => (*BOOL*),
 BIT_5 => (*BOOL*),
 BIT_6 => (*BOOL*),
 BIT_7 => (*BOOL*),
 BIT_8 => (*BOOL*),
 BIT_9 => (*BOOL*),
 BIT_10 => (*BOOL*),
 BIT_11 => (*BOOL*),
 BIT_12 => (*BOOL*),
 BIT_13 => (*BOOL*),
 BIT_14 => (*BOOL*),
 BIT_15 => (*BOOL*),
 BIT_16 => (*BOOL*));
```

### Блок PARSING_USINT

ФБ PARSING_USINT выводит значение входной переменной USINT в битах.



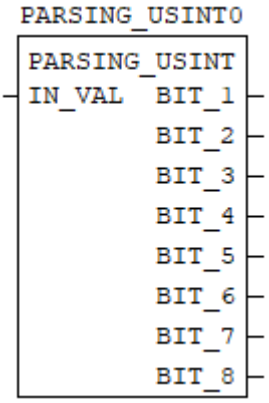

| Вход | Тип | Описание |
|---|---|---|
| IN_VAL | USINT | Обрабатываемая переменная |
| **Вы-ход** | **Тип** | **Описание** |
| BIT_1 | BOOL | Вывод значения первого бита |
| BIT_2 | BOOL | Вывод значения второго бита |
| BIT_3 | BOOL | Вывод значения третьего бита |
| BIT_4 | BOOL | Вывод значения четвертого бита |
| BIT_5 | BOOL | Вывод значения пятого бита |
| BIT_6 | BOOL | Вывод значения шестого бита |
| BIT_7 | BOOL | Вывод значения седьмого бита |
| BIT_8 | BOOL | Вывод значения восьмого бита |

Форма записи на языке ST:

```
FB(
 IN_VAL := (*USINT*),
 BIT_1 => (*BOOL*),
 BIT_2 => (*BOOL*),
 BIT_3 => (*BOOL*),
 BIT_4 => (*BOOL*),
 BIT_5 => (*BOOL*),
 BIT_6 => (*BOOL*),
 BIT_7 => (*BOOL*),
 BIT_8 => (*BOOL*));
```

### Блок RES_TEMP_SENSOR

Данный ФБ выдает значение сопротивления и температуры датчиков ТСП.



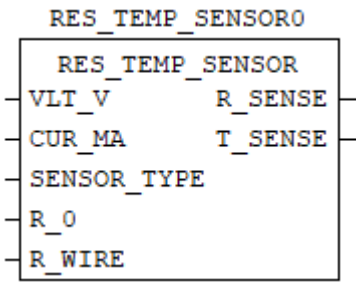

| Вход | Тип | Описание |
|---|---|---|
| VLT_V | REAL | Входное напряжение, В |
| CUR_MA | REAL | Входной ток, mA |
| SENSOR_TYPE | UINT | Тип сенсора, 0 - Pt, alpha = 0.00385; 1 - Pt, alpha = 0.00391; 2 - Cu, alpha = 0.00426 |
| R_0 | REAL | Номинальное сопротивление при 0°C, ОМ |
| R_WIRE | REAL | Сопротивление проводов, Ом |
| **Выход** | **Тип** | **Описание** |
| R_SENSE | REAL | Значение сопротивления датчика, Ом |
| T_SENSE | REAL | Значение сопротивления датчика, °C |

Форма записи на языке ST:

```
FB(
 VLT_V := (*REAL*),
 CUR_MA := (*REAL*),
 SENSOR_TYPE := (*UINT*),
 R_0 := (*REAL*),
 R_WIRE := (*REAL*),
 R_SENSE => (*REAL*),
 T_SENSE => (*REAL*));
```

### Блок ROUND_REAL

ФБ ROUND_REAL округляет значения входной переменной IN_REAL до DECIMAL_POINT знаков после запятой.



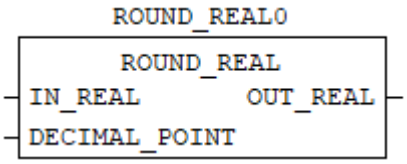
| Вход | Тип | Описание |
|---|---|---|
| IN_REAL | REAL | Входное значение |
| DECIMAL_POINT | USINT | Количество знаков после запятой (0-4). По умолчанию - 0 |
| **Выход** | **Тип** | **Описание** |
| OUT_REAL | REAL | Выходное значение |

Форма записи на языке ST:

```
FB(
 IN_REAL := (*REAL*),
 DECIMAL_POINT := (*USINT*),
 OUT_REAL => (*REAL*));
```

### Блок READ_PWR

ФБ READ_PWR предоставляет информацию о входном напряжении питания и напряжение батареи ПЛК.



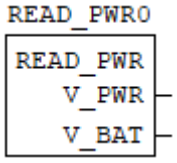

| Выход | Тип | Описание |
|---|---|---|
| V_PWR | REAL | Входное напряжение ПЛК |
| V_BAT | REAL | Напряжение батареи ПЛК |

Форма записи на языке ST:

```
FB(
 V_PWR => (*REAL*),
 V_BAT => (*REAL*));
```

### Блок READ_INTERNAL_TEMP

ФБ READ_INTERNAL_TEMP предоставляет информацию температуре микропроцессора ПЛК BRIC.



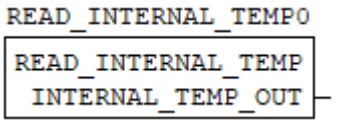

| Выход | Тип | Описание |
|---|---|---|
| INTERNAL_TEMP_OUT | REAL | Показание температуры микропроцессора в ПЛК |

Форма записи на языке ST:

```
FB(
 INTERNAL_TEMP_OUT => (*REAL*));
```

### Блок READ_RESET

ФБ READ_RESET предоставляет информацию о количестве перезагрузок с момента обновления ОС ПЛК и причине последней перезагрузки.



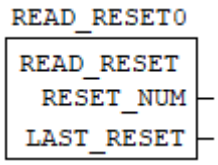

| Вы-ход | Тип | Описание |
|---|---|---|
| RESET_NUM | UINT | Количество перезагрузок с момента обновления ОС ПЛК |
| LAST_RESET | UINT | Код причины последней перезагрузки |

Форма записи на языке ST:

```
FB(
 RESET_NUM => (*UINT*),
 LAST_RESET => (*UINT*));
```

### Блок READ_PARAM_UDINT/UINT/USINT

ФБ READ_PARAM_UDINT/UINT/USINT предоставляет информацию о регистре в зависимости от типа данных.



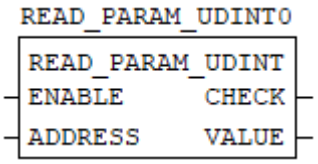

| Вход | Тип | Описание |
|---|---|---|
| ENABLE | BOOL | Включение функционального блока |
| ADRESS | UDINT/UINT/USINT | Адрес необходимого регистра Guid в формате UDINT/UINT/USINT (U32/U16/U8) |
| **Вы-ход** | **Тип** | **Описание** |
| CHECK | BOOL | Статус выхода |
| VALUE | UDINT/UINT/USINT | Значение регистра |

Форма записи на языке ST:

```
FB(
 ENABLE := (*BOOL*),
 ADDRESS := (*UDINT*),
 CHECK => (*BOOL*),
 VALUE => (*UDINT*));
```

### Блок READ_SYS_TICK_COUNTER

ФБ READ_SYS_TICK_COUNTER предоставляет информацию о времени работы ПЛК с момента последней перезагрузки указанной в мс.



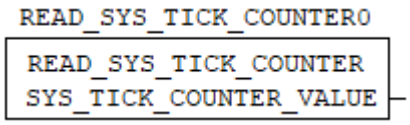

| Вход | Тип | Описание |
|---|---|---|
| SYS_TICK_COUNTER_VALUE | ULINT | Счетчик времени выполнения с момента последнего сброса в мс. |

Форма записи на языке ST:

```
FB(
 SYS_TICK_COUNTER_VALUE => (*ULINT*));
```

### Блок READ_AREA_LOCATION

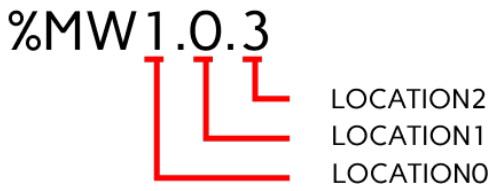
ФБ READ_AREA_LOCATION предоставляет информацию о регистре, определенным Modbus MemoryArea.




| Вы-ход | Тип | Описание |
|---|---|---|
| LOCATION0 | UINT | Номер нулевого элемента выборки MemoryArea |
| LOCATION1 | UINT | Номер первого элемента выборки MemoryArea |
| LOCATION2 | UINT | Номер второго элемента выборки MemoryArea |
| **Вы-ход** | **Тип** | **Описание** |
| VALUE | WORD | Вывод регистра |
| LOCTION_READED | BOOL | Контрольный бит-статус |

---

**Примечание:** Номер элемента определяется данным образом:




---

Форма записи на языке ST:

```
FB(
 LOCATION0 := (*UINT*),
 LOCATION1 := (*UINT*),
 LOCATION2 := (*UINT*),
```

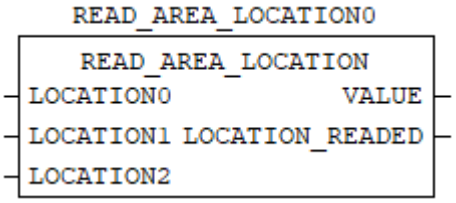
(continues on next page)

(продолжение с предыдущей страницы)

```
 VALUE => (*WORD*),
 LOCATION_READED => (*BOOL*));
```

### Блок READ_REQUEST_LOCATION

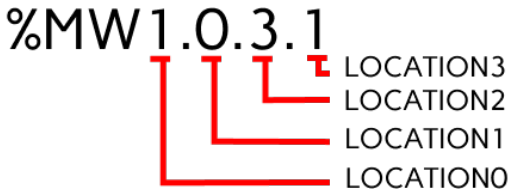
ФБ READ_REQUEST_LOCATION предоставляет информацию о регистре, определенным ModbusRequest.




| Вы-ход | Тип | Описание |
|---|---|---|
| LOCATION0 | UINT | Номер нулевого элемента выборки ModbusRequest |
| LOCATION1 | UINT | Номер первого элемента выборки ModbusRequest |
| LOCATION2 | UINT | Номер второго элемента выборки ModbusRequest |
| LOCATION3 | UINT | Номер третьего элемента выборки ModbusRequest |
| **Вы-ход** | **Тип** | **Описание** |
| VALUE | WORD | Вывод регистра |
| LOCTION_READED | BOOL | Контрольный бит-статус |

---

**Примечание:** Номер элемента определяется данным образом:



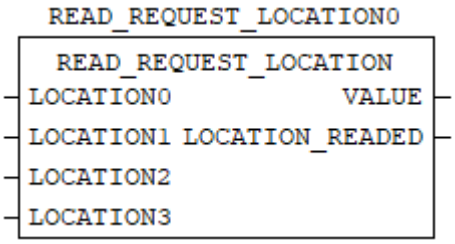

---

Форма записи на языке ST:

```
FB(
 LOCATION0 := (*UINT*),
 LOCATION1 := (*UINT*),
 LOCATION2 := (*UINT*),
 LOCATION3 := (*UINT*),
 VALUE => (*WORD*),
 LOCATION_READED => (*BOOL*));
```

### Блок SET_PARAM_UDINT/UINT/USINT

ФБ SET_PARAM_UDINT/UINT/USINT записывает данные в регистр в зависимости от типа данных.



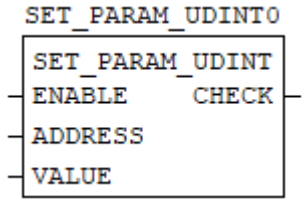

| Вход | Тип | Описание |
|---|---|---|
| ENABLE | BOOL | Включение функционального блока |
| ADRESS | UDINT/UINT/USINT | Адрес необходимого регистра Guid в формате UDINT/UINT/USINT (U32/U16/U8) |
| VALUE | UDINT/UINT/USINT | Значение регистра |
| **Выход** | **Тип** | **Описание** |
| CHECK | BOOL | Статус выхода |

Форма записи на языке ST:

```
FB(
 ENABLE := (*BOOL*),
 ADDRESS := (*UDINT*),
 VALUE := (*UDINT*),
 CHECK => (*BOOL*));
```

### Блок STRUCT_REAL_TIME

ФБ STRUCT_REAL_TIME позволяет считать время с ПЛК.



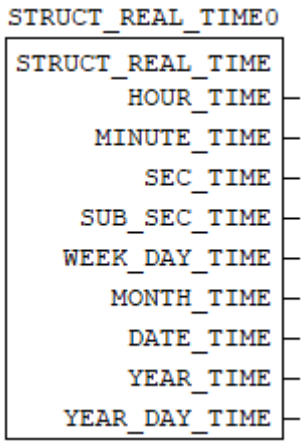

| Выход | Тип | Описание |
|---|---|---|
| HOUR_TIME | UINT | Текущий час реального времени внутреннего определения |
| MINUTE_TIME | USINT | Текущая минута |
| SEC_TIME | USINT | Текущая секунда |
| SUB_SEC_TIME | UINT | Текущая миллисекунда |
| WEEK_DAY_TIME | USINT | Текущий день недели |
| MONTH_TIME | USINT | Текущий месяц |
| DATE_TIME | USINT | Текущий день месяца |
| YEAR_TIME | UINT | Текущий год |
| YEAR_DAY_TIME | UINT | Текущий день года |

Форма записи на языке ST:

```
FB(
 HOUR_TIME => (*USINT*),
 MINUTE_TIME => (*USINT*),
 SEC_TIME => (*USINT*),
 SUB_SEC_TIME => (*USINT*),
 WEEK_DAY_TIME => (*USINT*),
 MONTH_TIME => (*USINT*),
 DATE_TIME => (*USINT*),
 YEAR_TIME => (*USINT*),
 YEAR_DAY_TIME => (*UINT*));
```

### Блок WRITE_MDB_ADRESS

ФБ WRITE_MDB_ADRESS позволяет установить Modbus адрес ПЛК (от 1 до 247).



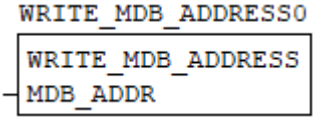

| Вход | Тип | Описание |
|---|---|---|
| MDB_ADDR | UINT | Установление адреса по протоколу Modbus в ПЛК |

Форма записи на языке ST:

```
FB(
 MDB_ADDR := (*UINT*));
```

### Блок WRITE_UART_SETS

ФБ WRITE_UART_SETS предоставлятn возможность изменять параметры UART каналов записанных в формате U16.




| Вход | Тип | Описание |
|---|---|---|
| MESO_UART | UINT | Запись параметров настройки интерфейса на канале Mesonin (доступ к порту только при вскрытии крышки ПЛК) |
| SET_RS485_2 | UINT | Запись параметров настройки интерфейса на канале RS–485–1 |
| SET_RS232 | UINT | Запись параметров настройки интерфейса на канале RS–232 |
| SET_RS485_1 | UINT | Запись параметров настройки интерфейса на канале RS–485–1 |
| SET_RS485_IMMO | UINT | Запись параметров настройки интерфейса на межмодульной шине |
| SET_HART | UINT | Запись параметров HART интерфейса на каналах AI |

Форма записи на языке ST:

```
FB(
 MESO_UART := (*UINT*),
 SET_RS_485_2 := (*UINT*),
 SET_RS_232 := (*UINT*),
 SET_RS_485_1 := (*UINT*),
 SET_RS_485_IMMO := (*UINT*),
 SET_HART := (*UINT*));
```

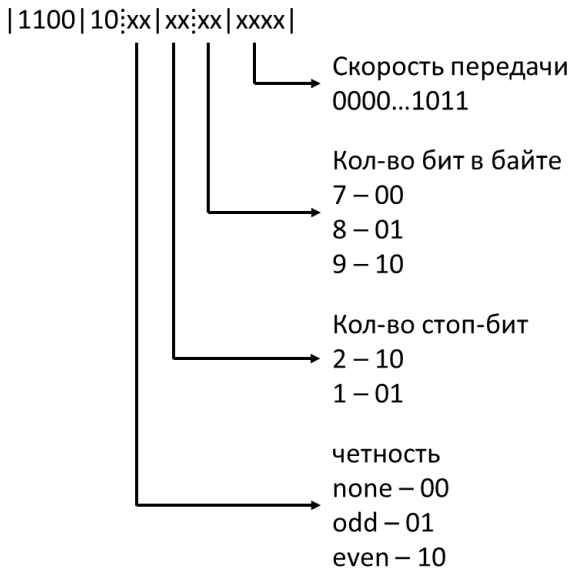
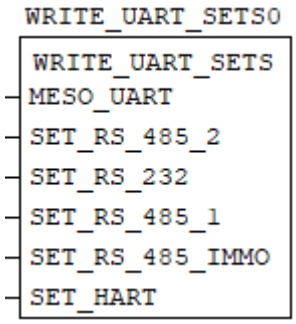



Рис. 62: Структура изменения параметров UART каналов

### Блок WRITE_CH_TIMEOUT

ФБ WRITE_CH_TIMEOUT изменяет время задержки UART каналов.




| Вход | Тип | Описание |
|---|---|---|
| CH_NUMBER | USINT | Номер канала |
| CHANNEL_TIMEOUT | UDINT | Задержка |

Форма записи на языке ST:

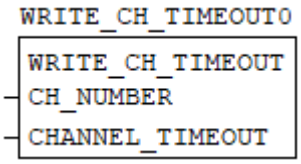
```
FB(
 CH_NUMBER := (*USINT*),
 CHANNEL_TIMEOUT := (*UDINT*));
```

**Примечание:**

| Номер канала | Наименование UART канала |
|---|---|
| 0 | MESO_UART_UART |
| 1 | RS_485_2_UART |
| 2 | RS_232_UART |
| 4 | RS_485_1_UART |
| 5 | RS_485_IMMO_UART |
| 6 | HART_UART |

### Блок WRITE_STRUCT_TIME

ФБ WRITE_STRUCT_TIME устанавливает время на ПЛК, выхода функционального блока используются для проверки записанного времени.



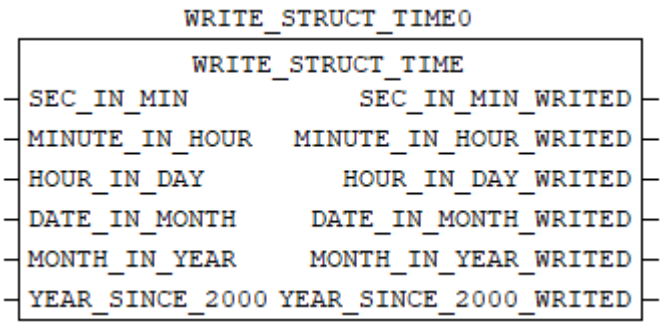

| Вход | Тип | Описание |
|---|---|---|
| SEC_IN_MIN | USINT | Ввод секунд |
| MINUTE_IN_HOUR | USINT | Ввод минут |
| HOUR_IN_DAY | USINT | Ввод часов |
| DATE_IN_MONTH | USINT | Ввод дня месяца |
| MONTH_IN_YEAR | USINT | Ввод месяца года |
| YEAR_SINCE_2000 | USINT | Ввод года |
| **Выход** | **Тип** | **Описание** |
| SEC_IN_MIN_WRITED | USINT | Вывод присвоенного значения |
| MINUTE_IN_HOUR_WRITED | USINT | Вывод присвоенного значения |
| HOUR_IN_DAY_WRITED | USINT | Вывод присвоенного значения |
| DATE_IN_MONTH_WRITED | USINT | Вывод присвоенного значения |
| MONTH_IN_YEAR_WRITED | USINT | Вывод присвоенного значения |
| YEAR_SINCE_2000_WRITED | USINT | Вывод присвоенного значения |

Форма записи на языке ST:

```
FB(
 SEC_IN_MIN := (*USINT*),
 MINUTE_IN_HOUR := (*USINT*),
 HOUR_IN_DAY := (*USINT*),
 DATE_IN_MONTH := (*USINT*),
 MONTH_IN_YEAR := (*USINT*),
 YEAR_SINCE_2000 := (*USINT*),
 SEC_IN_MIN_WRITED => (*USINT*),
 MINUTE_IN_HOUR_WRITED => (*USINT*),
 HOUR_IN_DAY_WRITED => (*USINT*),
 DATE_IN_MONTH_WRITED => (*USINT*),
 MONTH_IN_YEAR_WRITED => (*USINT*),
 YEAR_SINCE_2000_WRITED => (*USINT*));
```

### Блок WRITE_AREA_LOCATION

ФБ WRITE_AREA_LOCATION изменяет значения регистров, определенных Modbus MemoryArea.




| Вы-ход | Тип | Описание |
|---|---|---|
| VALUE | WORD | Значение регистра |
| LOCATION0 | UINT | Номер нулевого элемента выборки MemoryArea |
| LOCATION1 | UINT | Номер первого элемента выборки MemoryArea |
| LOCATION2 | UINT | Номер второго элемента выборки MemoryArea |
| **Вы-ход** | **Тип** | **Описание** |
| LOCATION_WRITED | BOOL | Контрольный бит-статус |

---

**Примечание:** Номер элемента определяется данным образом:

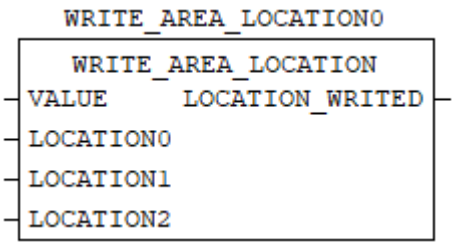

---

Форма записи на языке ST:

```
FB(
 VALUE := (*WORD*),
 LOCATION0 := (*UINT*),
 LOCATION1 := (*UINT*),
 LOCATION2 := (*UINT*),
 LOCATION_WRITED => (*BOOL*));
```

### Блок WRITE_REQUEST_LOCATION

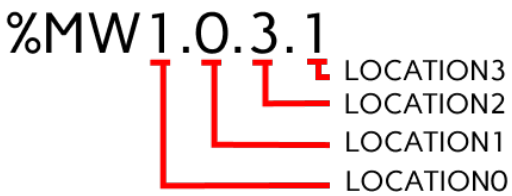
ФБ WRITE_REQUEST_LOCATION изменяет значения регистров, определенных ModbusRequest.




| Вы-ход | Тип | Описание |
|---|---|---|
| VALUE | WORD | Значение регистра |
| LOCATION0 | UINT | Номер нулевого элемента выборки ModbusRequest |
| LOCATION1 | UINT | Номер первого элемента выборки ModbusRequest |
| LOCATION2 | UINT | Номер второго элемента выборки ModbusRequest |
| LOCATION3 | UINT | Номер третьего элемента выборки ModbusRequest |
| **Вы-ход** | **Тип** | **Описание** |
| LOCATION_WRITED | BOOL | Контрольный бит-статус |

---

**Примечание:** Номер элемента определяется данным образом:



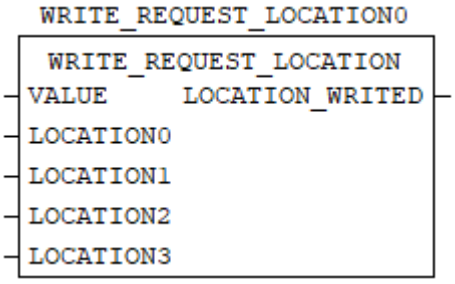

---

Форма записи на языке ST:

```
FB(
 VALUE := (*WORD*),
 LOCATION0 := (*UINT*),
 LOCATION1 := (*UINT*),
 LOCATION2 := (*UINT*),
 LOCATION3 := (*UINT*),
 LOCATION_WRITED => (*BOOL*));
```

### Блок UNIX_TIME

ФБ UNIX_TIME устанавливает время на ПЛК (время указывается в мс.).



| Вход | Тип | Описание |
|---|---|---|
| UNIX_TIME_WRITE | UDINT | Запись Unix time на ПЛК |
| **Выход** | **Тип** | **Описание** |
| UNIX_TIME_READ | UDINT | Unix time |
| UNIX_TIME_WRITED | UDINT | Последние записанное значени UNIX_TIME_WRITE |

---

**Примечание:** Время задается в контроллер только по изменению UNIX_TIME_WRITE, начальное значение для сравнения 0. ФБ можно использовать для чтения UNIX time в нескольких местах программы, для этого UNIX_TIME_WRITE оставляем нулевым.

---

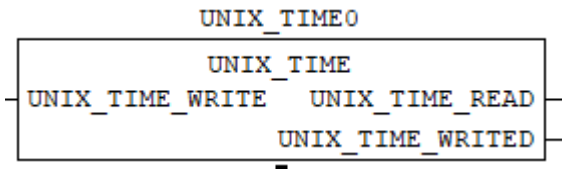
Форма записи на языке ST:

```
FB(
 UNIX_TIME_WRITE := (*UDINT*),
 UNIX_TIME_READ => (*UDINT*),
 UNIX_TIME_WRITED => (*UDINT*));
```

### DI Functional blocks

### Блок READ_DI

ФБ READ_DI предоставляет данные с дискретных входных каналов ПЛК записанного в одну переменную типа (UDINT) при этом 0 бит соответствует DI_0, а 15 бит DI_15.



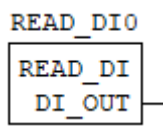

| Выход | Тип | Описание |
|---|---|---|
| DI_OUT | UDINT | Состояние дискретных входов ПЛК BRIC |

Форма записи на языке ST:

```
FB(
 DI_OUT => (*UDINT*));
```

### Блок READ_DI_CNT

ФБ READ_DI_CNT предоставляет значение счетчика дискретного входного канала ПЛК номер, которого прописан на входе блока.



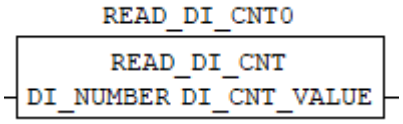
| Вход | Тип | Описание |
|---|---|---|
| DI_NUMBER | UINT | Номер исследуемого дискретного входа |
| **Выход** | **Тип** | **Описание** |
| DI_CNT_VALUE | ULINT | Значение счетчика дискретного входа |

Форма записи на языке ST:

```
FB(
 DI_NUMBER := (*UINT*),
 DI_CNT_VALUE => (*ULINT*));
```

### Блок READ_DI_FREQ

ФБ READ_DI_FREQ предоставляет значение частоты дискретного входного канала ПЛК, номер которого прописан на входе блока.



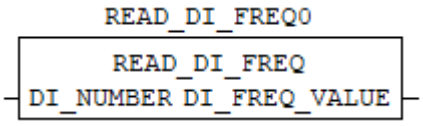

| Вход | Тип | Описание |
|---|---|---|
| DI_NUMBER | UINT | Номер исследуемого дискретного входа |
| **Выход** | **Тип** | **Описание** |
| DI_FREQ_VALUE | UINT | Значение частоты переключения дискретного входа |

Форма записи на языке ST:

```
FB(
 DI_NUMBER := (*UINT*),
 DI_FREQ_VALUE => (*REAL*));
```

### Блок WRITE_DI_NOISE_FLTR_10US

ФБ WRITE_DI_NOISE_FLTR_10US для указанного дискретного входа задается период нечувствительности импульса в диапазоне от 0 до 65512, при этом считается в десятках мс.



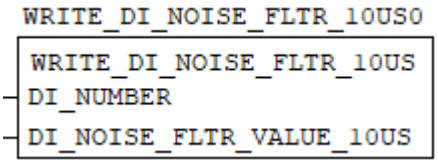

| Вход | Тип | Описание |
|---|---|---|
| DI_NUMBER | UINT | Номер канала дискретного входа ПЛК BRIC |
| DI_NOISE_FLTR_VALUE_10US | UINT | Период, за который счетчик обнаруживает не более 1 импульса |

Форма записи на языке ST:

```
FB(
 DI_NUMBER := (*UINT*),
 DI_NOISE_FLTR_VALUE_10US := (*UINT*));
```

### Блок WRITE_DI_PULSELESS

ФБ WRITE_DI_PULSELESS для указанного дискретного входа задает период, за который ведется подсчет импульсов для расчета частоты



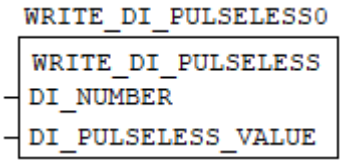

| Вход | Тип | Описание |
|---|---|---|
| DI_NUMBER | UINT | Номер канала дискретного входа ПЛК BRIC |
| DI_PULSELESS_VALUE | UDINT | Период, относительно которого рассчитывается частота канала |

Форма записи на языке ST:

```
FB(
 DI_NUMBER := (*UINT*),
 DI_PULSELESS_VALUE := (*UDINT*));
```

### Блок WRITE_DI_MODE

ФБ WRITE_DI_MODE для указанного дискретного входа обозначает подключенные опции (0– не подключены, 1– подключен счетчик импульсов, 2– подключен расчет частоты дискретного входа, 3– подключен счетчик импульсов и расчет частоты дискретного входа).




| Вход | Тип | Описание |
|---|---|---|
| DI_NUMBER | UINT | Номер канала дискретного входа ПЛК BRIC |
| DI_MODE_VALUE | UINT | Код подключенных функций данного канала |

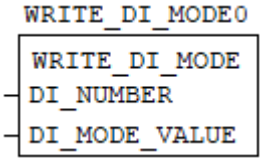
Форма записи на языке ST:

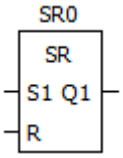
```
FB(
 DI_NUMBER := (*UINT*),
 DI_MODE_VALUE := (*UINT*));
```

## Standart function blocks

### Блок SR

Данный ФБ представляет собой бистабильный SR–триггер, с доминирующим входом S (Set).





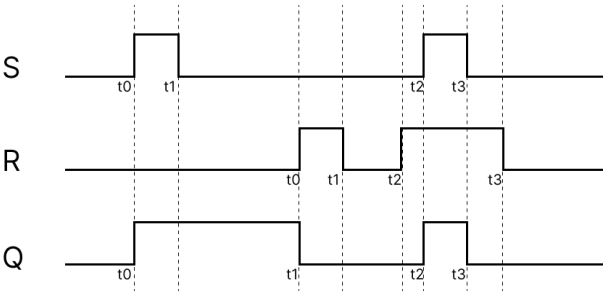

| Вход | Тип | Описание |
|---|---|---|
| S1 | BOOL | Вход (доминирующий) |
| R | BOOL | Сброс |
| **Выход** | **Тип** | **Описание** |
| Q1 | BOOL | Выход становится «1», когда на вход S1 приходит «1». При переходе S1 в «0» сохраняется состояние. Выход Q1 возвращается в «0», когда вход R становится «1» |

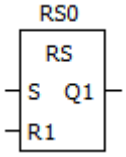
Форма записи на языке ST:

```
FB(
 S1 := (*BOOL*),
 R := (*BOOL*),
 Q1 => (*BOOL*));
```

## Блок RS

Данный ФБ представляет собой бистабильный RS–триггер, с доминирующим входом R (Reset).



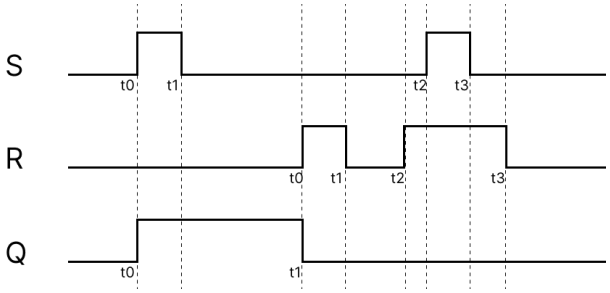

| Вход | Тип | Описание |
|---|---|---|
| S | BOOL | Вход |
| R1 | BOOL | Сброс (доминирующий) |
| **Выход** | **Тип** | **Описание** |
| Q1 | BOOL | Выход становится «1», когда вход R1 становится «0». При переходе R1 в «0» сохраняется состояние. Выход Q1 возвращается в «1», когда вход S становится «1» |

Форма записи на языке ST:

```
FB(
 S := (*BOOL*),
 R1 := (*BOOL*),
 Q1 => (*BOOL*));
```

### Блок SEMA

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

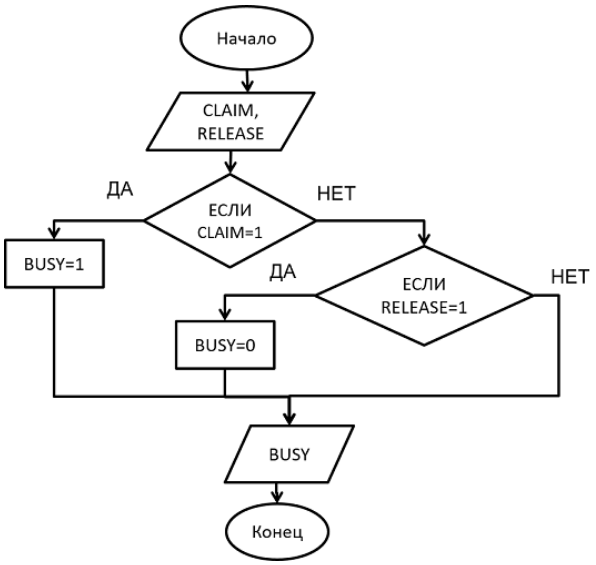



| Вход | Тип | Описание |
|---|---|---|
| CLAIM | BOOL | Вход (доминирующий) |
| RELEASE | BOOL | Сброс |
| **Выход** | **Тип** | **Описание** |
| BUSY | BOOL | Выход активируетс при CLAIM = 1 и деактивируется при RELEASE = 1 |

Форма записи на языке ST:

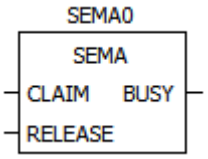
```
FB(
 CLAIM := (*BOOL*),
 RELEASE := (*BOOL*),
 BUSY => (*BOOL*));
```

Блок-схема SEMA



## Блок R–TRIG

Данный ФБ представляет собой индикатор нарастания фронта, который генерирует на выходе одиночный импульс при нарастании фронта сигнала.

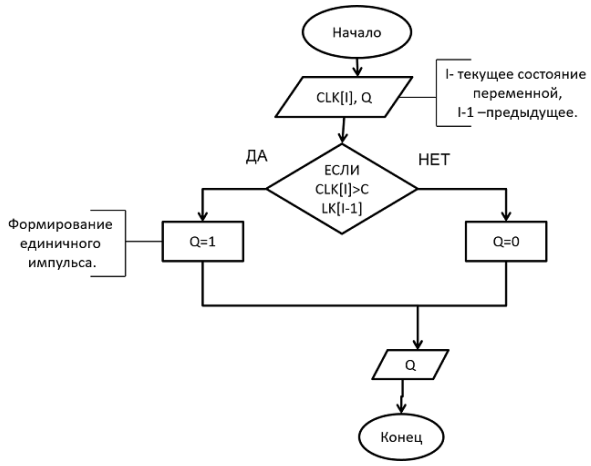



| Вход | Тип | Описание |
|---|---|---|
| CLK | BOOL | Вход |
| **Выход** | **Тип** | **Описание** |
| Q | BOOL | Выход генерирует единичный импульс, если на входе передний фронт |

Форма записи на языке ST:

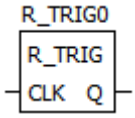
```
FB(
 CLK := (*BOOL*),
 Q => (*BOOL*));
```

Блок-схема R-TRIG




## Блок F–TRIG

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

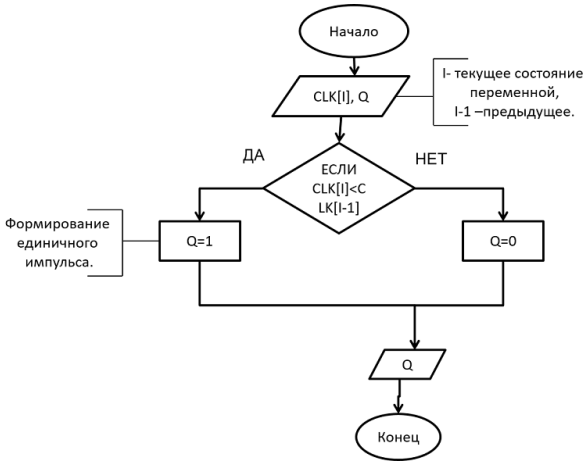



| Вход | Тип | Описание |
|---|---|---|
| CLK | BOOL | Вход |
| **Выход** | **Тип** | **Описание** |
| Q | BOOL | Выход генерирует единичный импульс, если на входе задний фронт |

Форма записи на языке ST:

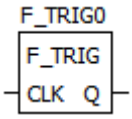
```
FB(
 CLK := (*BOOL*),
 Q => (*BOOL*));
```

Блок-схема F-TRIG




## Блок CTU/CTU_DINT...

Данный ФБ представляет собой инкрементный счётчик.




| Вход | Тип | Описание |
|---|---|---|
| CU | BOOL | Подача импульса |
| R | BOOL | Сброс |
| PV | ANY_INT | Предел счета |
| **Вы-ход** | **Тип** | **Описание** |
| Q | BOOL | Принимает значение „TRUE“ когда CV >= PV |
| CV | ANY_INT | Считает количество импульсов (CV = CV + 1) пока Q = 0 |

---

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

---

Форма записи на языке ST:

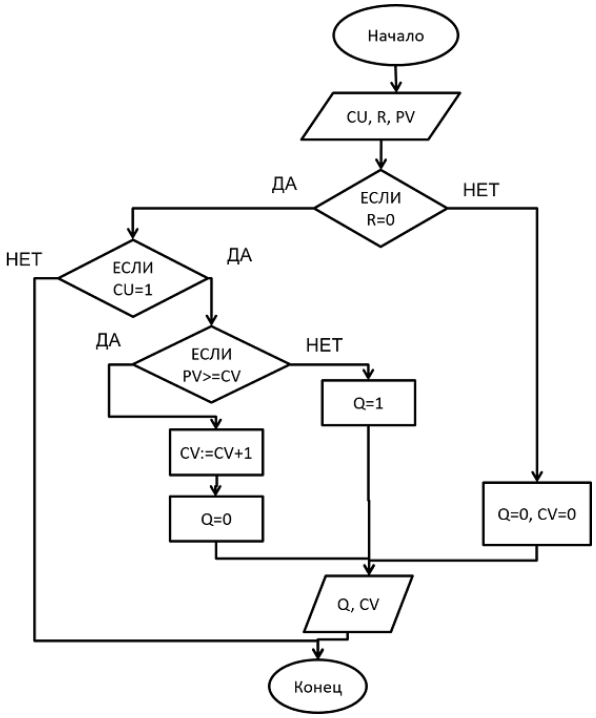
```
FB(
 CU := (*BOOL*),
 R := (*BOOL*),
 PV := (*DINT*),
 Q => (*BOOL*),
 CV => (*DINT*));
```

Блок-схема CTU

---

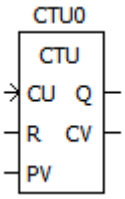
**Примечание:** PV не включает тип данных USINT, SINT.

---




### Блок CTD/CTD_DINT...

Данный ФБ представляет собой декрементный счётчик.




| Вход | Тип | Описание |
|---|---|---|
| CD | BOOL | Подача импульса |
| LD | BOOL | Сброс |
| PV | ANY_INT | Предел импульсов |
| **Вы-ход** | **Тип** | **Описание** |
| Q | BOOL | Принимает значение „TRUE“ когда CV = 0 |
| CV | ANY_INT | Считает количество импульсов (CV = CV – 1) пока Q = 0 |

---

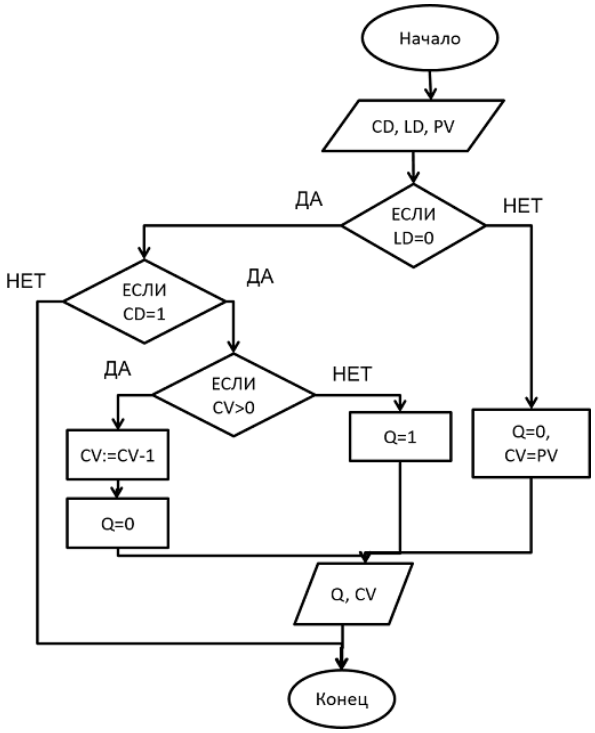
**Примечание:** Счетчик работает только до достижения минимального значения используемого типа данных. Переполнения не происходит.

---

Форма записи на языке ST:

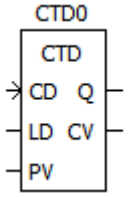
```
FB(
 CD := (*BOOL*),
 LD := (*BOOL*),
 PV := (*INT*),
 Q => (*BOOL*),
 CV => (*INT*));
```

Блок-схема CTD




### Блок CTUD/CTUD_DINT...

Данный ФБ представляет собой реверсивный счётчик.




| Вход | Тип | Описание |
|---|---|---|
| CU | BOOL | Подача импульса |
| CD | BOOL | Подача импульса |
| R | BOOL | Сброс до 0 |
| LD | BOOL | Сброс до PV |
| PV | ANY_INT | Верхний предел импульсов |
| **Выход** | **Тип** | **Описание** |
| QU | BOOL | Принимает значение „TRUE“ когда CV >= PV |
| QD | BOOL | Принимает значение „TRUE“ когда CV = 0 |
| CD_T | R_TRIG | (в существующей версии не применяется) |
| CU_T | R_TRIG | (в существующей версии не применяется) |

---

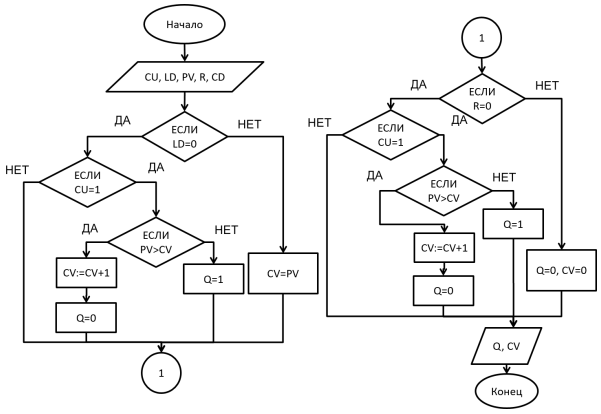
**Примечание:** Вычитающий счетчик работает только до достижения минимального значения используемого типа данных, суммирующий счетчик работает только до достижения максимального значения используемого типа данных. Переполнения не происходит.

---

Форма записи на языке ST:

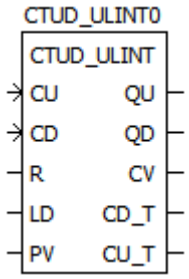
```
FB(
 CU := (*BOOL*),
 CD := (*BOOL*),
 R := (*BOOL*),
 LD := (*BOOL*),
 PV := (*INT*),
 QU => (*BOOL*),
 QD => (*BOOL*),
 CV => (*INT*),
 CD_T => (*R_TRIG*),
 CU_T => (*R_TRIG*));
```

Блок-схема CTUD




## Блок TP

Данный ФБ представляет собой повторитель импульсов и используется для генерирования импульса с заданной продолжительностью.

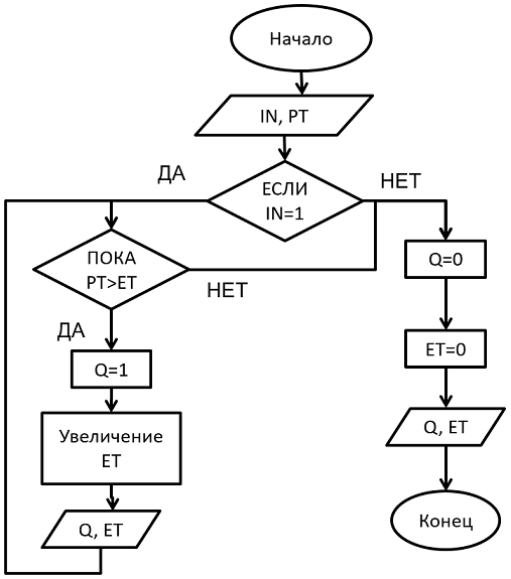



| Вход | Тип | Описание |
|---|---|---|
| IN | BOOL | Подача импульса |
| PT | TIME | Время одного импульса |
| **Выход** | **Тип** | **Описание** |
| Q | BOOL | Выход (пока ET < PT, Q = „TRUE“) |
| ET | TIME | Пока IN = 1 и ET < PT идет счет времени ET |

Форма записи на языке ST:

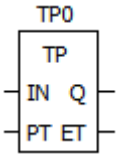
```
FB(
 IN := (*BOOL*),
 PT := (*TIME*),
 Q => (*BOOL*),
 ET => (*TIME*));
```

Блок-схема TP




## Блок TON

Данный ФБ представляет собой таймер с задержкой включения.

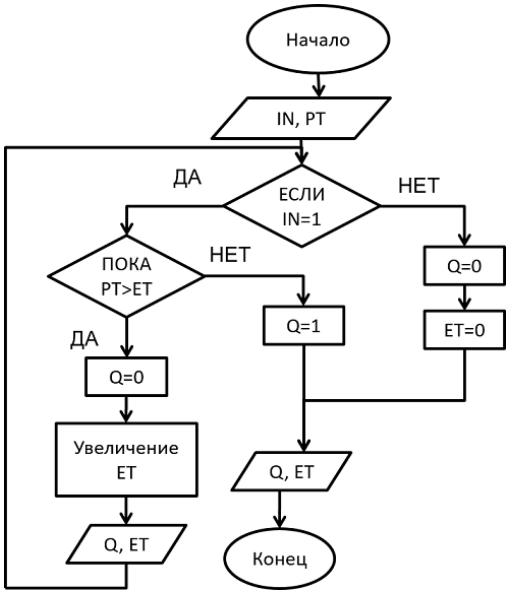



| Вход | Тип | Описание |
|---|---|---|
| IN | BOOL | Подача импульса |
| PT | TIME | Время задержки |
| **Вы-ход** | **Тип** | **Описание** |
| Q | BOOL | Если ET = PV и IN = 1, то Q = 1, иначе Q = 0 |
| ET | TIME | Счетчик времени считает пока ET < PV и IN = 1 |

Форма записи на языке ST:

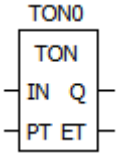
```
FB(
 IN := (*BOOL*),
 PT := (*TIME*),
 Q => (*BOOL*),
 ET => (*TIME*));
```

Блок-схема TON




## Блок TOF

Данный ФБ представляет собой таймер с задержкой отключения.

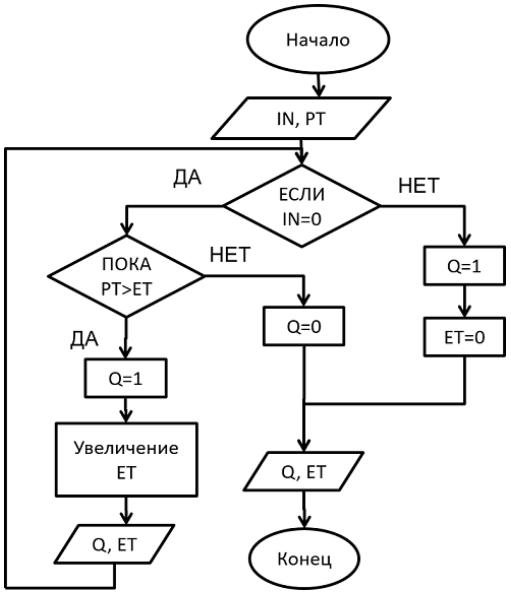



| Вход | Тип | Описание |
|---|---|---|
| IN | BOOL | Подача импульса |
| PT | TIME | Время задержки |
| **Вы-ход** | **Тип** | **Описание** |
| Q | BOOL | Если ET = PV и IN = 1, то Q = 1, иначе Q = 0 |
| ET | TIME | Счетчик времени считает пока ET < PV и IN = 1 |

Форма записи на языке ST:

```
FB(
 IN := (*BOOL*),
 PT := (*TIME*),
 Q => (*BOOL*),
 ET => (*TIME*));
```

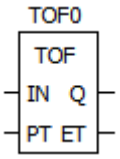
Блок-схема TOF




## Additional Functional blocks

## Блок RTC

Данный ФБ 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.

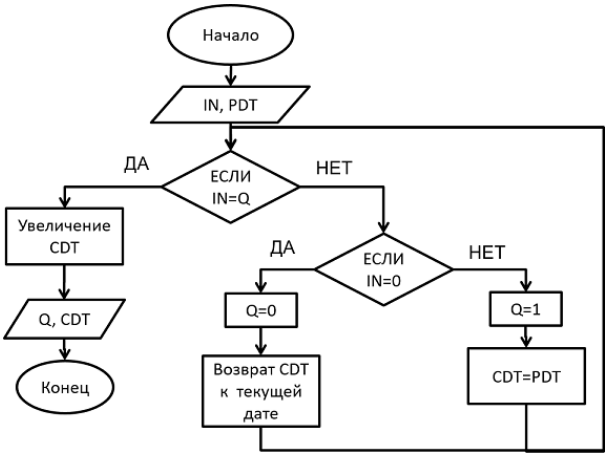



| Вход | Тип | Описание |
|---|---|---|
| IN | BOOL | Переключение режима |
| PDT | TIME | Время начала работы |
| **Вы-ход** | **Тип** | **Описание** |
| Q | BOOL | Индикация режима |
| CDT | DT | Вывод времени |

Форма записи на языке ST:

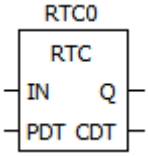
```
FB(
 IN := (*BOOL*),
 PDT := (*DT*),
 Q => (*BOOL*),
 CDT => (*DT*));
```

Блок-схема RTC




## Блок INTEGRAL

ФБ интегрирует входное значение XIN по времени.

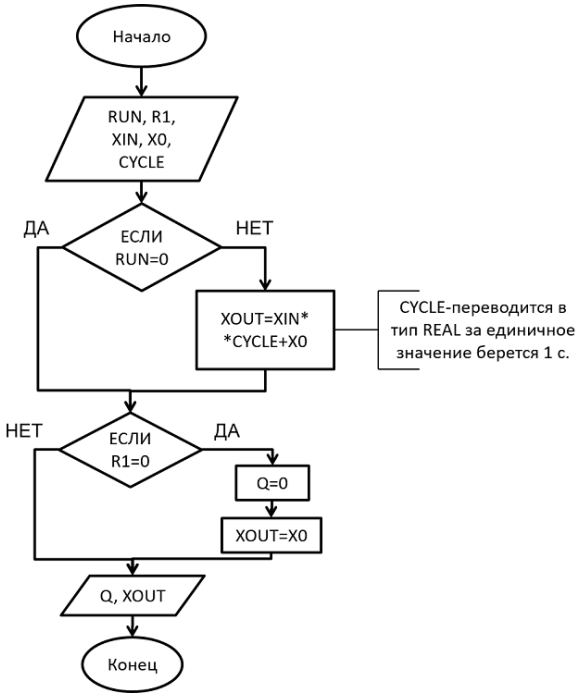



| Вход | Тип | Описание |
|---|---|---|
| RUN | BOOL | Включение блока |
| R1 | BOOL | Сброс (XOUT = X0) |
| XIN | REAL | Интегрируемое значение |
| X0 | REAL | Начальное значение XOUT |
| CYCLE | TIME | Время интегрирования |
| **Выход** | **Тип** | **Описание** |
| Q | BOOL | Q = 1, если R1 = 0 |
| XOUT | REAL | Выход (интегрирует входное значение XIN во времени) (XOUT = X0 * CYCLE (количество тактов при RUN = 1)) |

Форма записи на языке ST:

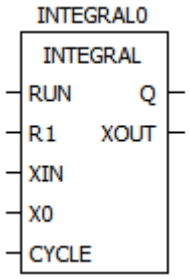
```
FB(
 RUN := (*BOOL*),
 R1 := (*BOOL*),
 XIN := (*REAL*),
 X0 := (*REAL*),
 CYCLE := (*TIME*),
 Q => (*BOOL*),
 XOUT => (*REAL*));
```

Блок-схема INTEGRAL




## Блок DERIVATIVE

ФБ выдаёт значение XOUT пропорционально скорости изменения входного параметра XIN.

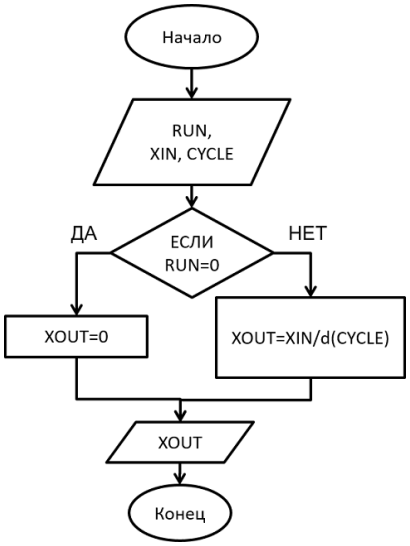



| Вход | Тип | Описание |
|---|---|---|
| RUN | BOOL | Включение блока |
| XIN | REAL | Вход |
| CYCLE | TIME | Время дифференцирования |
| **Выход** | **Тип** | **Описание** |
| XOUT | REAL | Формирование сигнала пропорционально частоте изменения входа XIN |

Форма записи на языке ST:

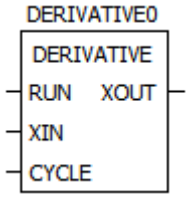
```
FB(
 RUN := (*BOOL*),
 XIN := (*REAL*),
 CYCLE := (*TIME*),
 XOUT => (*REAL*));
```

Блок-схема DERIVATIVE




### Блок PID

Данный ФБ представляет собой устройство в цепи обратной связи, используемое в системах автоматического управления для формирования управляющего сигнала




| Вход | Тип | Описание |
|---|---|---|
| AUTO | BOOL | Включение режима ПИД-регулятора (0 - ручное регулирование, 1 - автоматическое) |
| DIR | BOOL | Направление регулирования ( 0 - обратное регулирование, 1 - прямое. По умолчанию стоит прямое регулирование) |
| PV | REAL | Входной сигнал (автоматический режим) |
| SP | REAL | Заданное значение (уставка в автоматическом режиме) |
| X0 | REAL | Заданное значение (уставка в ручном режиме) |
| KP | REAL | Пропорциональный коэффициент |
| TR | REAL | Интегральный коэффициент |
| TD | REAL | Дифференциальный коэффициент |
| CYCLE | TIME | Время цикла |
| **Выход** | **Тип** | **Описание** |
| XOUT | REAL | Выход ПИД-регулятора |

Форма записи на языке ST:

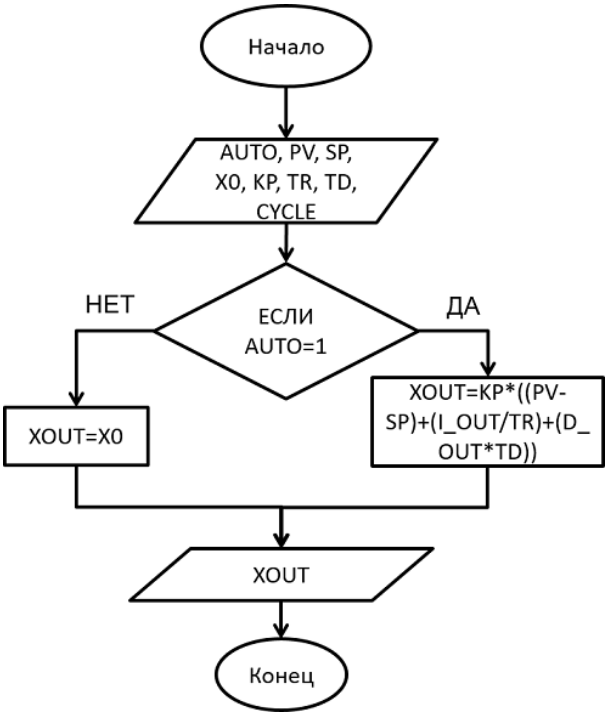
```
FB(
 AUTO := (*BOOL*),
 DIR := (*BOOL*),
 PV := (*REAL*),
 SP := (*REAL*),
 X0 := (*REAL*),
 KP := (*REAL*),
 TR := (*REAL*),
 TD := (*REAL*),
```

(continues on next page)

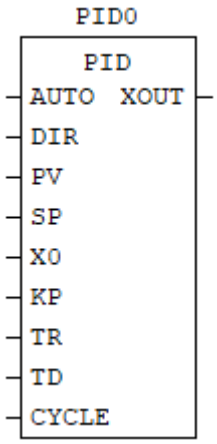
(продолжение с предыдущей страницы)

```
CYCLE := (*TIME*),
XOUT => (*REAL*));
```

Блок-схема PID




### Блок RAMP

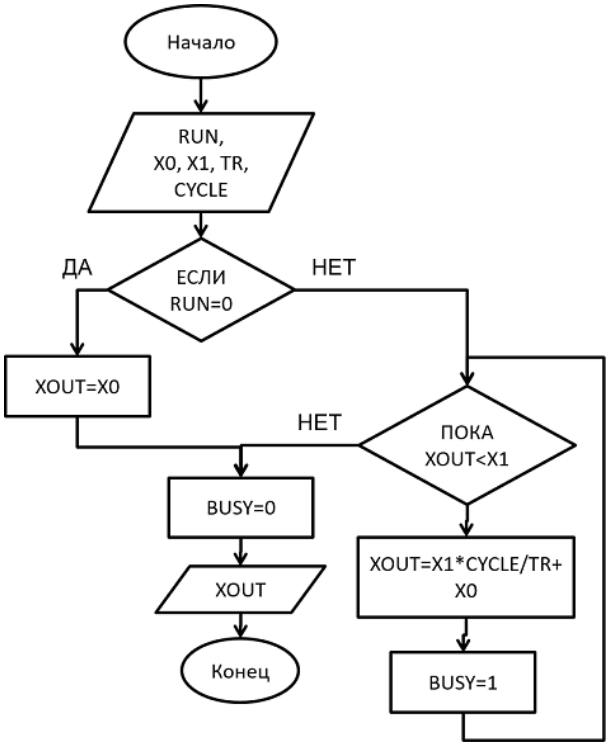


| Вход | Тип | Описание |
|---|---|---|
| RUN | BOOL | Включение блока |
| X0 | REAL | Начало отсчета |
| X1 | REAL | Конечное значение |
| TR | TIME | Время перехода |
| CYCLE | TIME | Время интегрирования |
| **Вы-ход** | **Тип** | **Описание** |
| BUSY | BOOL | BUSY = 1, если значения XOUT меняются |
| XOUT | REAL | Если RUN = 1, то пока XOUT < X1 XOUT = X1*CYCLE*t/TR + X0 иначе XOUT = X0 |

Форма записи на языке ST:

```
FB(
 RUN := (*BOOL*),
 X0 := (*REAL*),
 X1 := (*REAL*),
 TR := (*TIME*),
 CYCLE := (*TIME*),
 BUSY => (*BOOL*),
 XOUT => (*REAL*));
```

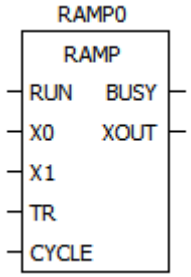
Блок-схема RAMP



## AI Functional blocks

### Блок READ_AI_STATE

ФБ READ_AI_STATE показывает информацию о состоянии аналогового входа.



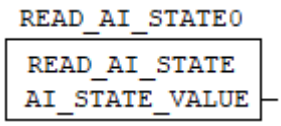

| Вы-ход | Тип | Описание |
|---|---|---|
| AI_STATE_VALUE | UINT | Состояние канала. Логическая 1 – измеренное значение тока лежит в диапазоне 4 - 20 мА, логический 0 – измеренное значение ниже 4 мА либо выше 20 мА. |

Форма записи на языке ST:

```
FB(
 AI_STATE_VALUE => (*UINT*));
```

### Блок READ_AI

ФБ READ_AI выводит значение аналогового входного канала ПЛК, номер которого прописан на входе блока.



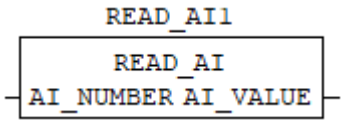

| Вход | Тип | Описание |
|---|---|---|
| AI_NUMBER | UINT | Номер канала |
| **Вы-ход** | **Тип** | **Описание** |
| AI_VALUE | UINT | Показание аналогового канала по шкале 0 – 16383, которая соответствует шкале принимаемого унифицированного сигнала |

Форма записи на языке ST:

```
FB(
 AI_NUMBER := (*UINT*),
 AI_VALUE => (*UINT*));
```

### Блок READ_AI_REAL

ФБ READ_AI_REAL выводит значение аналогового входного канала ПЛК в мА, номер которого прописан на входе блока.




| Вход | Тип | Описание |
|---|---|---|
| AI_NUMBER | UINT | Номер канала |
| **Выход** | **Тип** | **Описание** |
| AI_VALUE | REAL | Показание аналогового канала в мА |

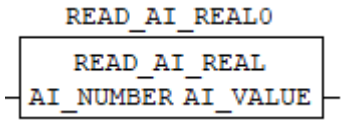
Форма записи на языке ST:

```
FB(
 AI_NUMBER := (*UINT*),
 AI_VALUE => (*REAL*));
```

### Type conversion

### Блок *_TO_*

Данный ФБ предназначен для всех возможных и корректных, согласно стандарту IEC 61131–3, преобразований между типами данных.




| Вход | Тип | Описание |
|---|---|---|
| IN | … | Преобразуемая информация |
| **Выход** | **Тип** | **Описание** |
| OUT | … | Преобразованная информация |

Форма записи на языке ST:

```
ANY_TO_ANY(
 IN := (*ANY*),
 OUT => (*ANY*));
```

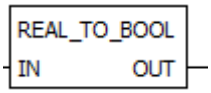
Таблица 9: Перечисление блоков преобразования типов

| Тип | Описание |
|---|---|
| REAL_TO_* | Преобразование REAL в остальные типы данных |
| SINT_TO_* | Преобразование SINT в остальные типы данных |
| LINT_TO_* | Преобразование LINT в остальные типы данных |
| DINT_TO_* | Преобразование DINT в остальные типы данных |
| DATE_TO_* | Преобразование DATE в остальные типы данных |
| DWORD_TO_* | Преобразование DWORD в остальные типы данных |
| DT_TO_* | Преобразование DT в остальные типы данных |
| TOD_TO_* | Преобразование TOD в остальные типы данных |
| UDINT_TO_* | Преобразование UDINT в остальные типы данных |
| WORD_TO_* | Преобразование WORD в остальные типы данных |
| STRING_TO_* | Преобразование STRING в остальные типы данных |
| LWORD_TO_* | Преобразование LWORD в остальные типы данных |
| UINT_TO_* | Преобразование UINT в остальные типы данных |
| LREAL_TO_* | Преобразование LREAL в остальные типы данных |
| BYTE_TO_* | Преобразование BYTE в остальные типы данных |
| USINT_TO_* | Преобразование USINT в остальные типы данных |
| ULINT_TO_* | Преобразование ULINT в остальные типы данных |
| BOOL_TO_* | Преобразование BOOL в остальные типы данных |
| TIME_TO_* | Преобразование TIME в остальные типы данных |
| INT_TO_* | Преобразование INT в остальные типы данных |

## Numerical

### Блок ABS

Данный ФБ возвращает в OUT модуль входного числа IN.



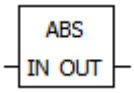

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_NUM | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_NUM | Выход (модуль числа) |

Форма записи на языке ST:

```
ABS(
 IN := (*ANY_NUM*),
 OUT => (*ANY_NUM*));
```

### Блок SQRT

Данный ФБ возвращает в OUT квадратный корень входного числа IN.



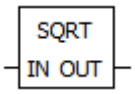

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (корень числа) |

Форма записи на языке ST:

```
SQRT(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок LN

Данный ФБ возвращает в OUT значение натурального логарифма от IN.



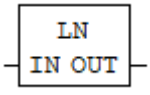

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (натуральный логарифм числа) |

Форма записи на языке ST:

```
LN(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок LOG

Данный ФБ возвращает в OUT значение логарифма по основанию 10 от IN.



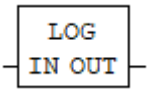

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (логарифм числа) |

Форма записи на языке ST:

```
LOG(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок EXP

Данный ФБ возвращает в OUT значение экспоненты, возведённой в степень IN.



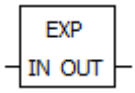

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (экспонента числа) |

Форма записи на языке ST:

```
EXP(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок SIN

Данный ФБ возвращает в OUT значение синуса IN.



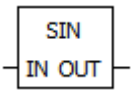

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (синус числа) |

Форма записи на языке ST:

```
SIN(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок COS

Данный ФБ возвращает в OUT значение косинуса IN.



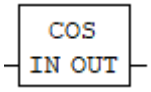
| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (косинус числа) |

Форма записи на языке ST:

```
COS(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок TAN

Данный ФБ возвращает в OUT значение тангенса IN.



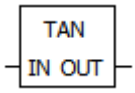

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (тангенс числа) |

Форма записи на языке ST:

```
TAN(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок ASIN

Данный ФБ возвращает в OUT значение арксинуса IN.



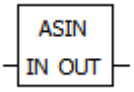

| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (арксинус числа) |

Форма записи на языке ST:

```
ASIN(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Блок ACOS

Данный ФБ возвращает в OUT значение арккосинуса IN.




| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (арккосинус числа) |

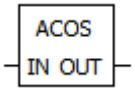
Форма записи на языке ST:

```
ACOS(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

*Блок-схема ACOS*

### Блок ATAN

Данный ФБ возвращает в OUT значение арктангенса IN.




| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_REAL | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (арктангенс числа) |

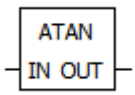
Форма записи на языке ST:

```
ATAN(
 IN := (*ANY_REAL*),
 OUT => (*ANY_REAL*));
```

### Arithmetic

### Блок ADD

Данный ФБ возвращает в OUT результат сложения IN1 и IN2.



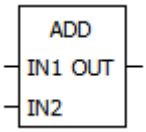
| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_NUM | Вход |
| IN2 | ANY_NUM | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_NUM | Выход (сложение) |

Форма записи на языке ST:

```
ADD(
 IN1 := (*ANY_NUM*),
 IN2 := (*ANY_NUM*),
 OUT => (*ANY_NUM*));
```

### Блок MUL

Данный ФБ возвращает в OUT результат умножения IN1 и IN2.



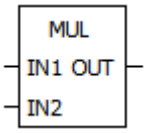

| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_NUM | Вход |
| IN2 | ANY_NUM | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_NUM | Выход (умножение) |

Форма записи на языке ST:

```
MUL(
 IN1 := (*ANY_NUM*),
 IN2 := (*ANY_NUM*),
 OUT => (*ANY_NUM*));
```

### Блок SUB

Данный ФБ возвращает в OUT результат вычитания из IN1 значения IN2.



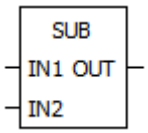

| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_NUM | Вход |
| IN2 | ANY_NUM | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_NUM | Выход (разность) |

Форма записи на языке ST:

```
SUB(
 IN1 := (*ANY_NUM*),
 IN2 := (*ANY_NUM*),
 OUT => (*ANY_NUM*));
```

### Блок DIV

Данный ФБ возвращает в OUT результат деления IN1 на IN2.



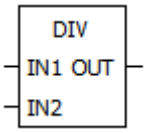

| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_NUM | Вход (делитель) |
| IN2 | ANY_NUM | Вход (делитель) |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_NUM | Выход (деление) |

Форма записи на языке ST:

```
DIV(
 IN1 := (*ANY_NUM*),
 IN2 := (*ANY_NUM*),
 OUT => (*ANY_NUM*));
```

### Блок MOD

Данный ФБ возвращает в OUT остаток от деления IN1 на IN2.



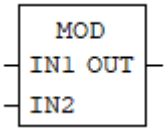

| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_INT | Вход (делимое) |
| IN2 | ANY_INT | Вход (делитель) |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_INT | Выход (остаток от деления) |

Форма записи на языке ST:

```
MOD(
 IN1 := (*ANY_INT*),
 IN2 := (*ANY_INT*),
 OUT => (*ANY_INT*));
```

### Блок EXPT

Данный ФБ возвращает в OUT значение IN1 возведённое в степень IN2.



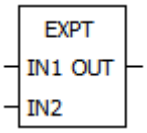

| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_REAL | Вход |
| IN2 | ANY_NUM | Вход (степень) |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_REAL | Выход (число в степени) |

Форма записи на языке ST:

```
EXPT(
 IN1 := (*ANY_REAL*),
 IN2 := (*ANY_NUM*),
 OUT => (*ANY_REAL*));
```

### Блок MOVE

Данный ФБ возвращает в OUT значение IN.




| Вход | Тип | Описание |
|---|---|---|
| IN | ANY | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY | Выход (присваивание) |

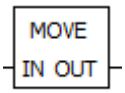
Форма записи на языке ST:

```
MOVE(
 IN := (*ANY*),
 OUT => (*ANY*));
```

### Bit-shift

### Блок SHL

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

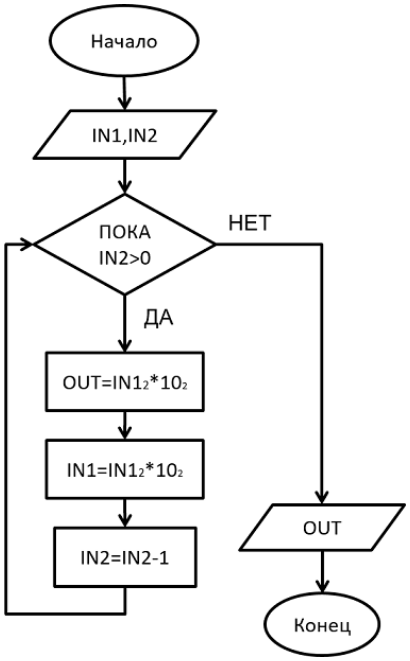


| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_BIT | Вход |
| IN2 | ANY_INT | Количество сдвигов влево |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход |

Форма записи на языке ST:

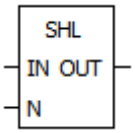
```
SHL(
 IN := (*ANY_BIT*),
 N := (*ANY_INT*),
 OUT => (*ANY_BIT*));
```

*Блок-схема SHL*[3]




## Блок SHR

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



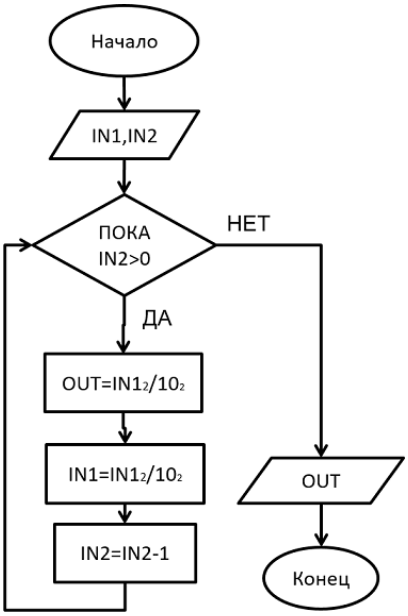

| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_BIT | Вход |
| IN2 | ANY_INT | Количество сдвигов вправо |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход |

[3] IN1 Тип переменной выхода должен соответствовать типу переменной входа. У OUT нет возможности использования переменных типа BOOL.

Форма записи на языке ST:

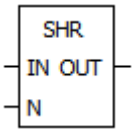
```
SHR(
 IN := (*ANY_BIT*),
 N := (*ANY_INT*),
 OUT => (*ANY_BIT*));
```

*Блок-схема SHR*[4]




## Блок ROR

Данный ФБ возвращает в OUT циклический сдвиг аргумента IN на N бит влево.




| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_BIT | Вход |
| IN2 | ANY_INT | Количество сдвигов вправо |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход |

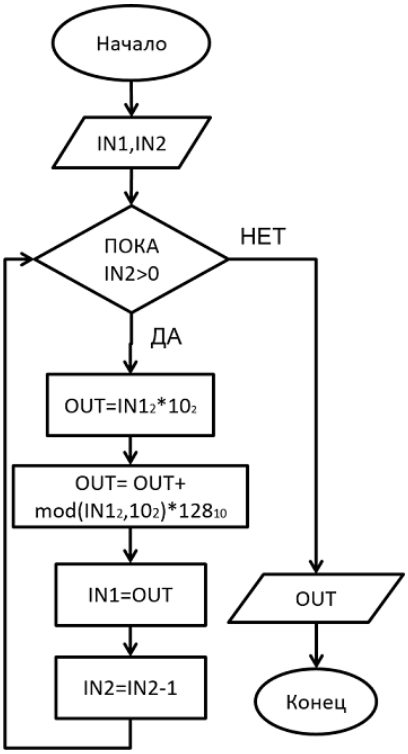
Форма записи на языке ST:

```
ROR(
 IN := (*ANY_NBIT*),
 N := (*ANY_INT*),
 OUT => (*ANY_NBIT*));
```

*Блок-схема ROR*[5]

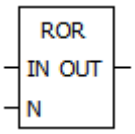
[4] IN1 Тип переменной выхода должен соответствовать типу переменной входа. У OUT нет возможности использования переменных типа BOOL.

[5] IN1 Тип переменной выхода должен соответствовать типу переменной входа. У OUT нет возможности использования переменных типа BOOL.




## Блок ROL

Данный ФБ возвращает в OUT циклический сдвиг аргумента IN на N бит вправо.



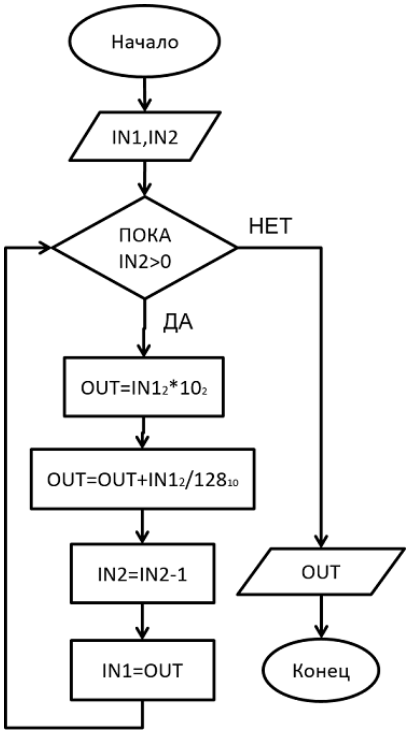

| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_BIT | Вход |
| IN2 | ANY_INT | Количество сдвигов влево |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход |

Форма записи на языке ST:

```
ROL(
 IN := (*ANY_NBIT*),
 N := (*ANY_INT*),
 OUT => (*ANY_NBIT*));
```

*Блок-схема ROL*[6]

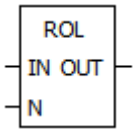
[6] IN1 Тип переменной выхода должен соответствовать типу переменной входа. У OUT нет возможности использования переменных типа BOOL.




## Bitwise

## Блок AND

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

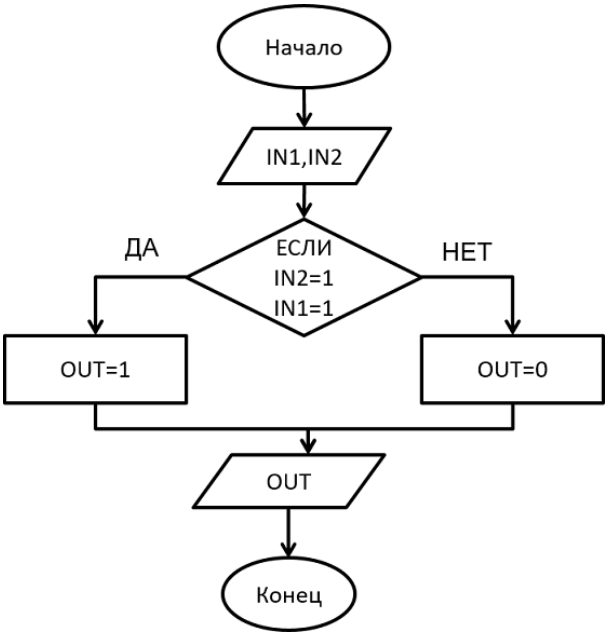



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_BIT | Вход |
| IN2 | ANY_BIT | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход |

Форма записи на языке ST:

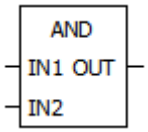
```
AND(
 IN1 := (*ANY_BIT*),
 IN2 := (*ANY_BIT*),
 OUT => (*ANY_BIT*));
```

*Блок-схема AND*




### Блок OR

Данный ФБ представляет собой организацию «логического ИЛИ» для всех входных аргументов IN1…INn.

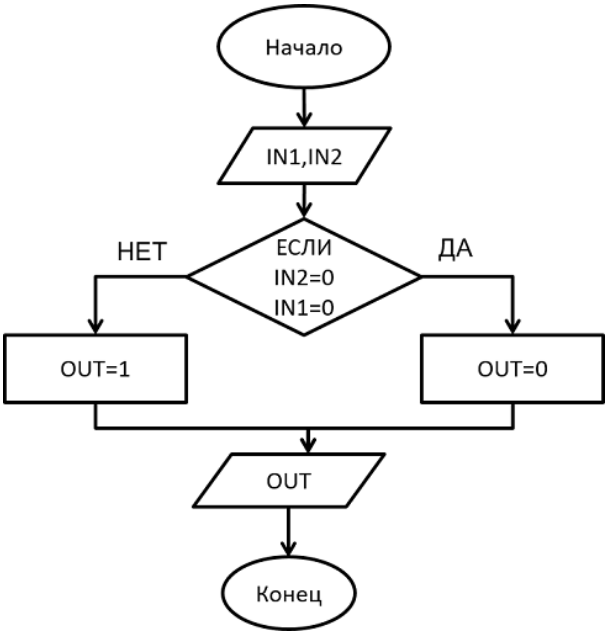



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_BIT | Вход |
| IN2 | ANY_BIT | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход |

Форма записи на языке ST:

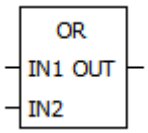
```
OR(
 IN1 := (*ANY_BIT*),
 IN2 := (*ANY_BIT*),
 OUT => (*ANY_BIT*));
```

*Блок-схема OR*




### Блок XOR

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

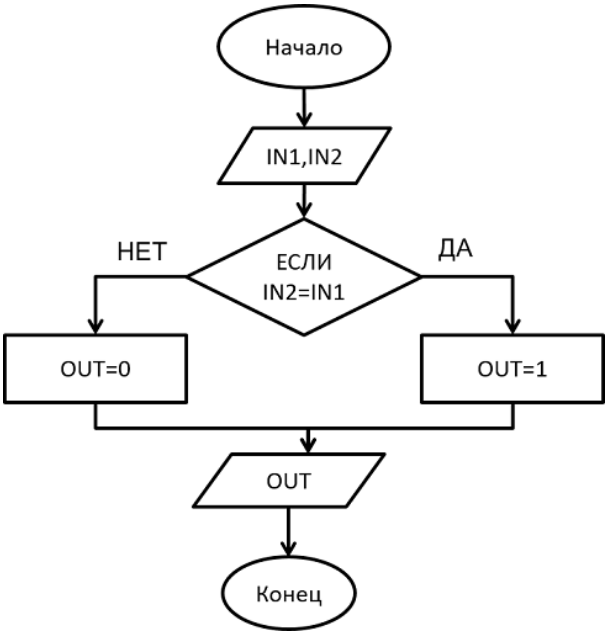



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY_BIT | Вход |
| IN2 | ANY_BIT | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход |

Форма записи на языке ST:

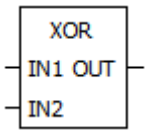
```
XOR(
 IN1 := (*ANY_BIT*),
 IN2 := (*ANY_BIT*),
 OUT => (*ANY_BIT*));
```

*Блок-схема XOR*




### Блок NOT

Данный ФБ представляет собой организацию «логической инверсии» для входного аргумента IN.




| Вход | Тип | Описание |
|---|---|---|
| IN | ANY_BIT | Вход |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY_BIT | Выход (инверсия) |

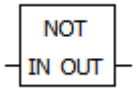
Форма записи на языке ST:

```
NOT(
 IN := (*ANY_BIT*),
 OUT => (*ANY_BIT*));
```

### Selection

### Блок SEL

Данный ФБ возвращает в OUT один из двух аргументов IN1 или IN2 в зависимости от значения аргумента G.

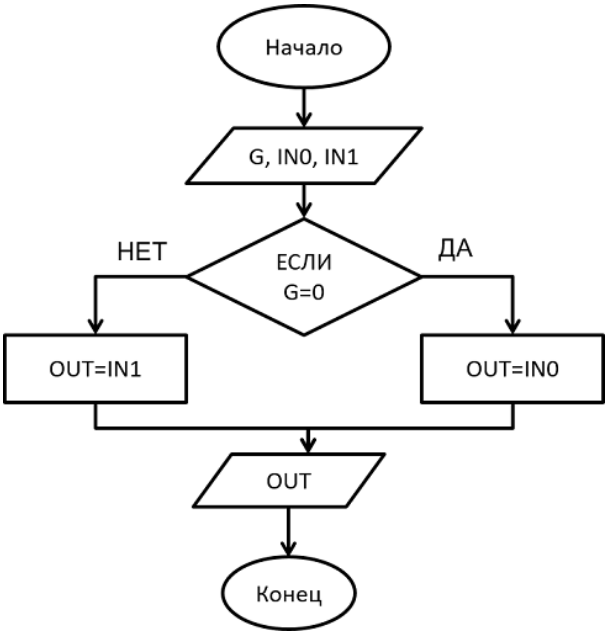


| Вход | Тип | Описание |
|---|---|---|
| G | BOOL | Номер входа |
| IN0 | ANY | Вход 1 |
| IN1 | ANY | Вход 2 |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY | Выход |

Форма записи на языке ST:

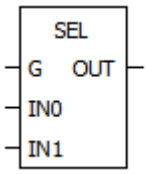
```
SEL(
 G := (*BOOL*),
 IN0 := (*ANY*),
 IN1 := (*ANY*),
 OUT => (*ANY*));
```

*Блок-схема SEL*




## Блок MAX

Данный ФБ возвращает в OUT максимум из входных аргументов IN1 и IN2.

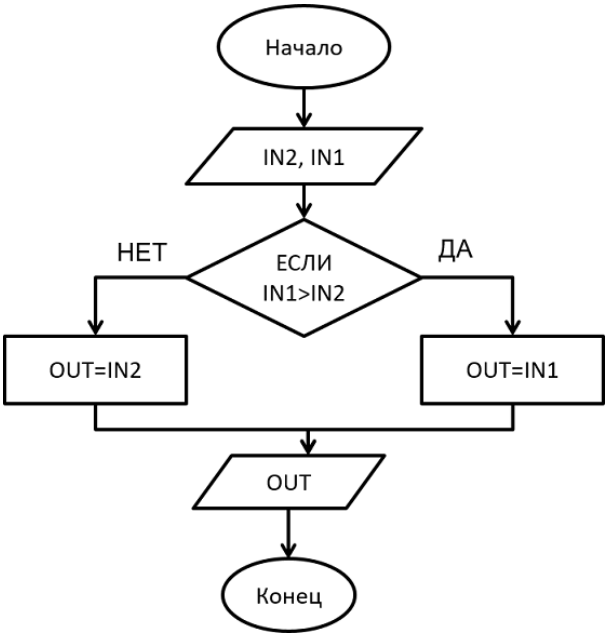



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | … | Вход … |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY | Выход |

Форма записи на языке ST:

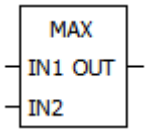
```
MAX(
 IN1 := (*ANY*),
 IN2 := (*ANY*),
 OUT => (*ANY*));
```

*Блок-схема MAX*




## Блок MIN

Данный ФБ возвращает в OUT минимум из входных аргументов IN1 и IN2.

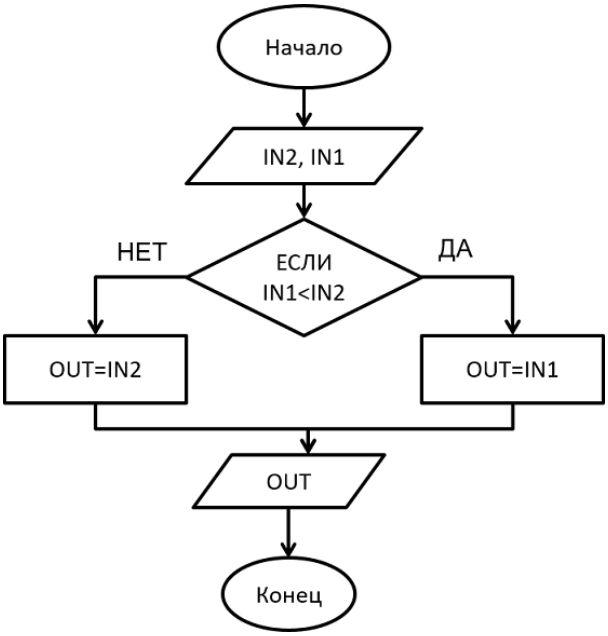



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | … | Вход … |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY | Выход |

Форма записи на языке ST:

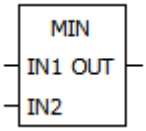
```
MIN(
 IN1 := (*ANY*),
 IN2 := (*ANY*),
 OUT => (*ANY*));
```

*Блок-схема MIN*




### Блок LIMIT

Данный ФБ 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.

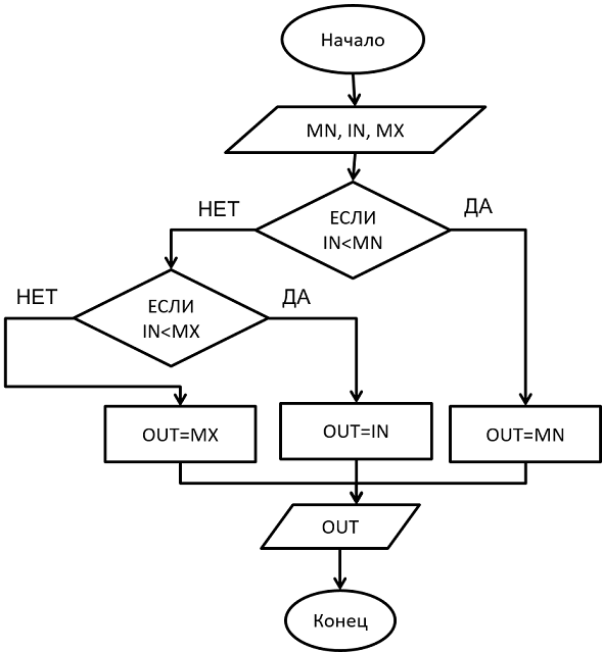



| Вход | Тип | Описание |
|---|---|---|
| MN | ANY | Вход (нижний предел) |
| IN | ANY | Вход |
| MX | ANY | Вход (верхний предел) |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY | Выход |

Форма записи на языке ST:

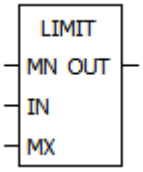
```
LIMIT(
 MN := (*ANY*),
 IN := (*ANY*),
 MX := (*ANY*),
 OUT => (*ANY*));
```

*Блок-схема LIMIT*




## Блок MUX

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

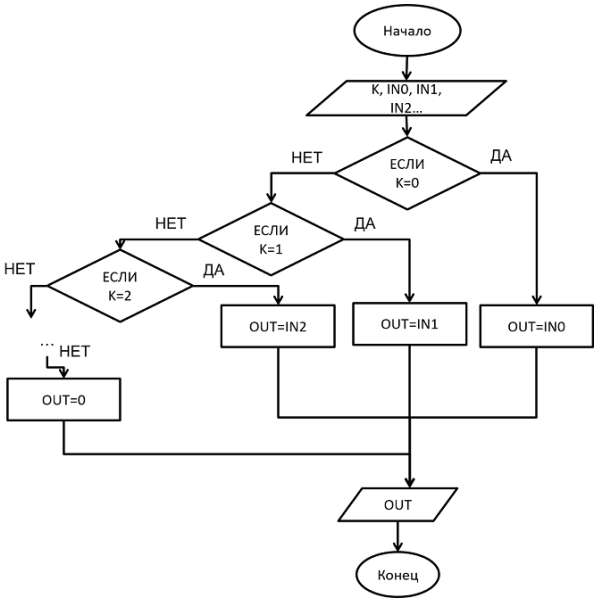



| Вход | Тип | Описание |
|---|---|---|
| K | ANY_INT | Переключатель |
| IN0 | ANY | Вход 1 |
| IN1 | ANY | Вход 2 |
| … | ANY | Вход … |
| **Выход** | **Тип** | **Описание** |
| OUT | ANY | Выход |

Форма записи на языке ST:

```
MUX(
 K := (*ANY_INT*),
 IN0 := (*ANY*),
 IN1 := (*ANY*),
 OUT => (*ANY*));
```

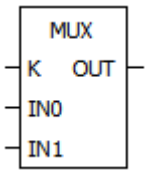
*Блок-схема MUX*




## Comparsion

## Блок GT

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

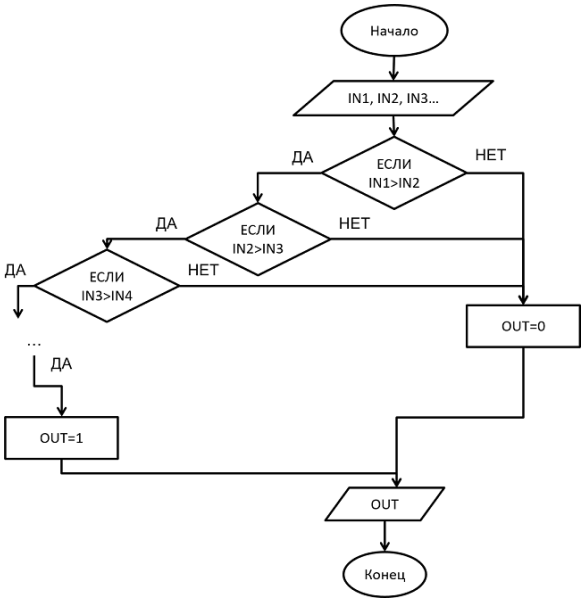



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | ANY | Вход … |
| **Вы-ход** | **Тип** | **Описание** |
| OUT | BOOL | Выход |

Форма записи на языке ST:

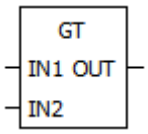
```
GT(
 IN1 := (*ANY*),
 IN2 := (*ANY*),
 OUT => (*BOOL*));
```

*Блок-схема GT*




## Блок GE

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

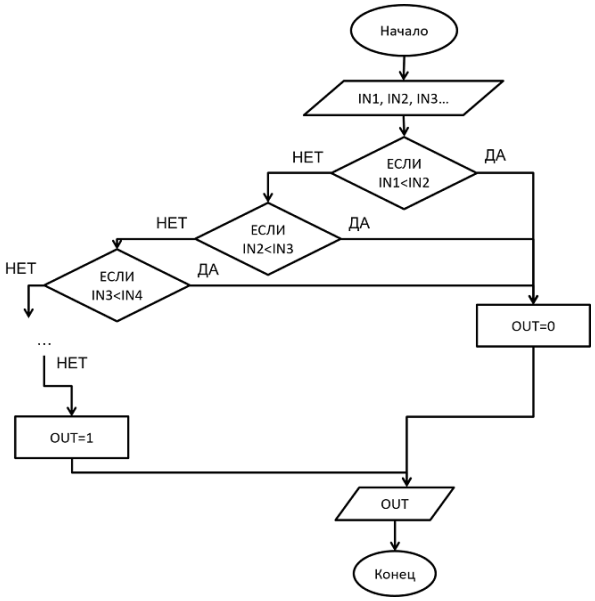



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | ANY | Вход … |
| **Выход** | **Тип** | **Описание** |
| OUT | BOOL | Выход |

Форма записи на языке ST:

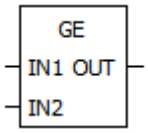
```
GE(
 IN1 := (*ANY*),
 IN2 := (*ANY*),
 OUT => (*BOOL*));
```

*Блок-схема GE*




## Блок EQ

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

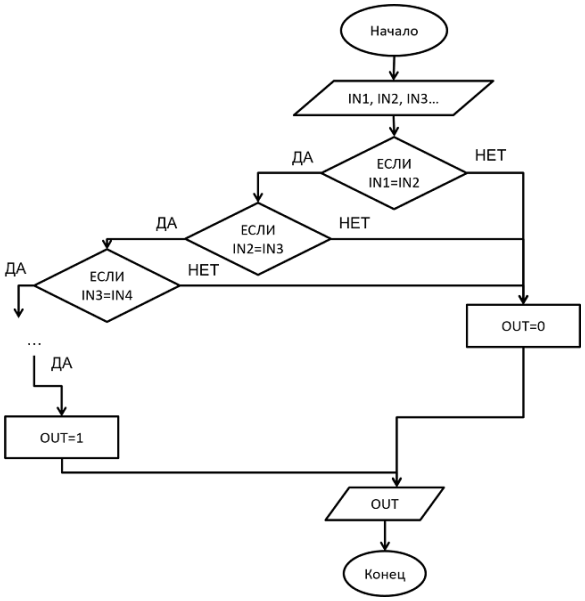



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | ANY | Вход … |
| **Выход** | **Тип** | **Описание** |
| OUT | BOOL | Выход |

Форма записи на языке ST:

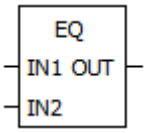
```
EQ(
 IN1 := (*ANY*),
 IN2 := (*ANY*),
 OUT => (*BOOL*));
```

*Блок-схема EQ*




## Блок LT

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

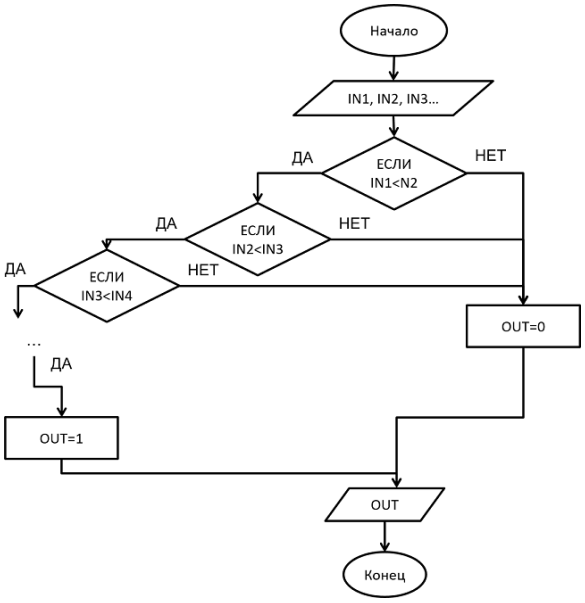



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | ANY | Вход … |
| **Вы-ход** | **Тип** | **Описание** |
| OUT | BOOL | Выход |

Форма записи на языке ST:

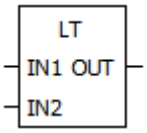
```
LT(
 IN1 := (*ANY*),
 IN2 := (*ANY*),
 OUT => (*BOOL*));
```

*Блок-схема LT*




## Блок LE

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

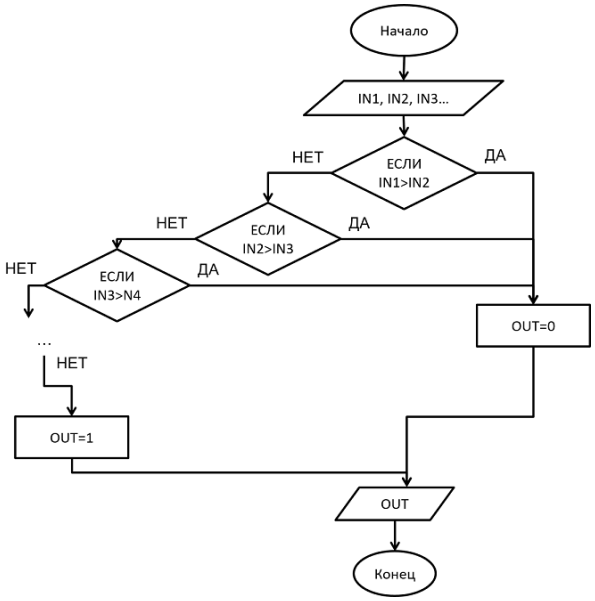



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | ANY | Вход … |
| **Вы-ход** | **Тип** | **Описание** |
| OUT | BOOL | Выход |

Форма записи на языке ST:

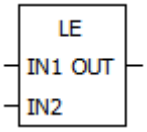
```
LE(
 IN1 := (*ANY*),
 IN2 := (*ANY*),
 OUT => (*BOOL*));
```

*Блок-схема LE*




## Блок NE

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

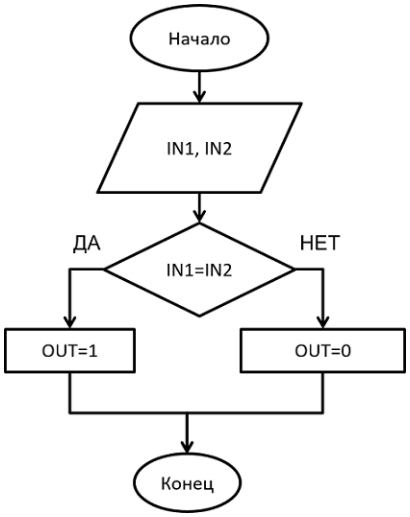



| Вход | Тип | Описание |
|---|---|---|
| IN1 | ANY | Вход 1 |
| IN2 | ANY | Вход 2 |
| … | ANY | Вход … |
| **Вы-ход** | **Тип** | **Описание** |
| OUT | BOOL | Выход |

Форма записи на языке ST:

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

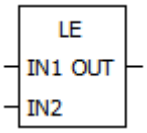
*Блок-схема NE*




### BRIC FB

#### Функция TASK_PERIOD

Данная фунцкия выводит информацию о цикле программы в миллисекундах.




| Вход | Тип | Описание |
|---|---|---|
| IN | STRING | Выбор цикла из ресурсов |
| **Выход** | **Тип** | **Описание** |
| OUT | UDINT | Вывод цикла в мс |

---

**Примечание:** Наименование для IN должно совпадать с наименованием цикла в ресурсах, а также должен быть объявлен локально.

---

Форма записи на языке ST:

```
TASK_PERIOD(
 IN := (*STRING*),
 OUT => (*UDINT*));
```

## 1.10 Архивирование данных

В OpenPLC имеется возможность архивирования данных пользовательского проекта. Весь процесс создания архивов можно описать в двух шагах:

- Добавление в структуру проекта «Поддержки Архивов»;
- Написание программы для архивирования данных.

### Добавление в структуру проекта «Поддержки Архивов»

Для добавления в структуру проекта необходимо щелкнуть правой кнопкой мыши в область дерева проекта и нажать «Поддержка Архивов». Далее необходимо подвести курсор к созданной ветке «Поддержка Архивов», щелкнуть левой клавишей мыши и выбрать пункт «Добавить Arch», как показано на рисунке ниже:

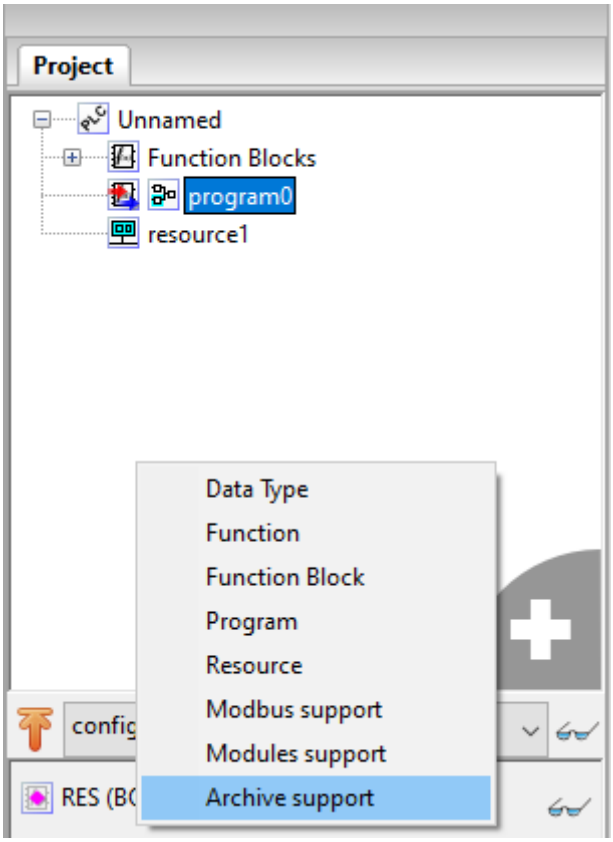



Рис. 63: Добавление «Поддержки Архивов»

Появляется окно добавленного архива с определенными настройками по умолчанию. Область адресов, выделенных регистров для базовых настроек архивирования начинается с 50000. Данные, начинающиеся с адреса 50013 по 50025 (по умолчанию), являются заголовком архива. Туда входят идентификатор, длина заголовка в битах, длина суммы пользовательских данных и заголовка в битах, время, дата в представлении UNIX, флаг и уникальная контрольная сумма архива и его порядковый номер. В опциях они обозначены как *arc_header_data*.

---

**Примечание:** Переменная *Arch_save_arc* является триггером для записи данных в архив: при изменении из 0 в 1 данные записываются.

---

Имеется возможность поменять адреса данных архива. Данные настройки находятся во вкладке «Config»:

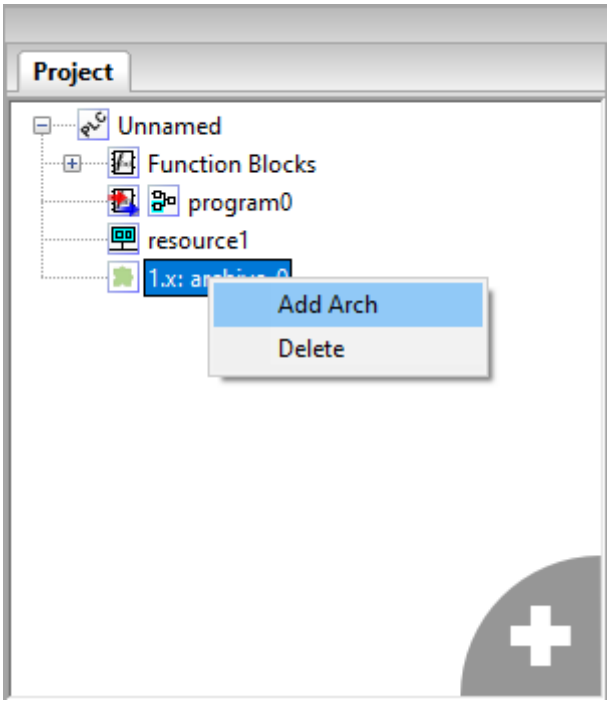



Рис. 64: Добавление Arch

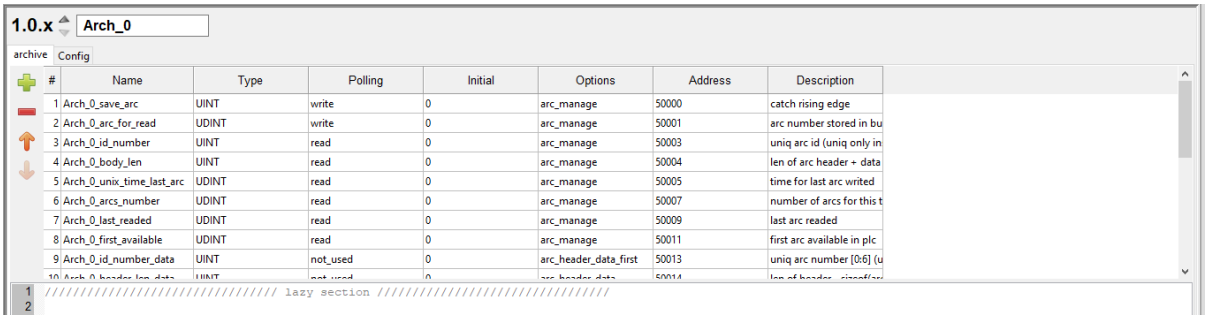



Рис. 65: Окно данных archive

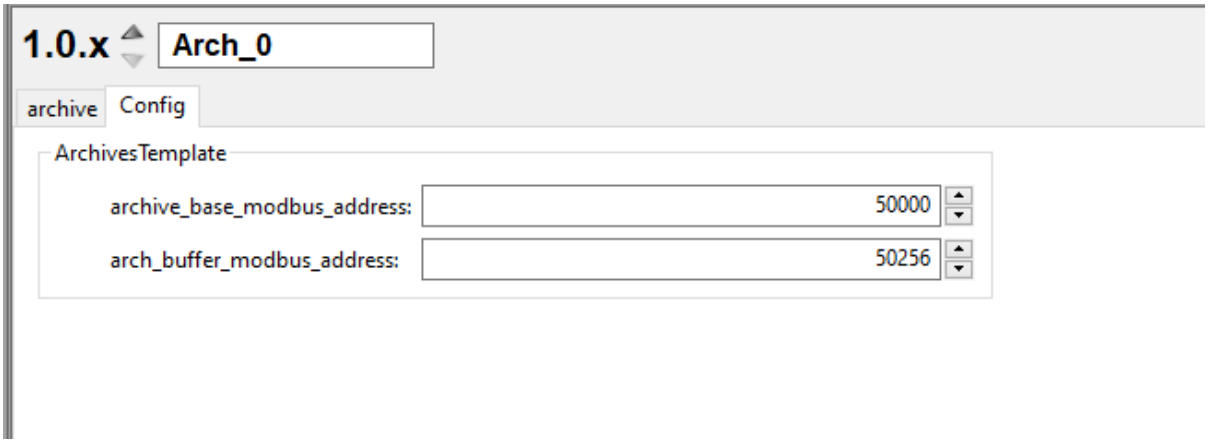
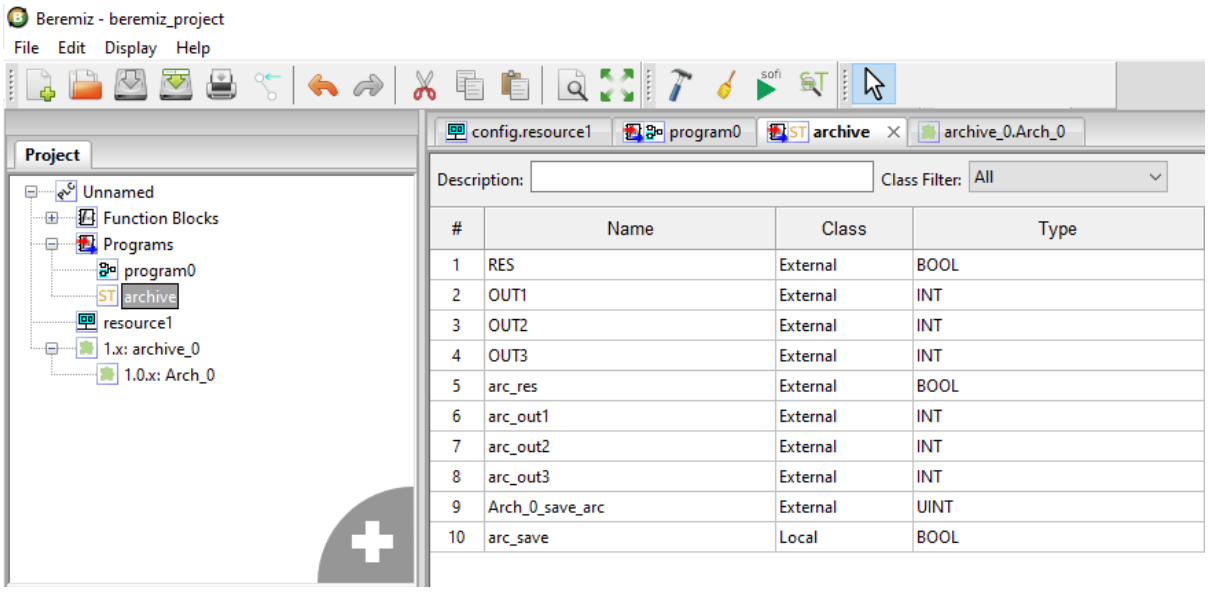



Рис. 66: Окно конфигурирования адресов archive

### Программа для архивирования данных

Для записи пользовательских данных в архив необходимо написать определенную программу. Для примера возьмем ранее реализованную программу *beremiz_project*. Программу напишем на языке ST. На рисунке ниже представлены данные для программы.




| # | Name | Class | Type |
|---|---|---|---|
| 1 | RES | External | BOOL |
| 2 | OUT1 | External | INT |
| 3 | OUT2 | External | INT |
| 4 | OUT3 | External | INT |
| 5 | arc_res | External | BOOL |
| 6 | arc_out1 | External | INT |
| 7 | arc_out2 | External | INT |
| 8 | arc_out3 | External | INT |
| 9 | Arch_0_save_arc | External | UINT |
| 10 | arc_save | Local | BOOL |

Рис. 67: Переменные программы archive

В глобальные переменные *arc_out1…arc_out3* записываются данные переменных *OUT1…OUT3*, в *arc_res* – переменная RES при изменении *Arch_0_save_arc* с 0 на 1. Локальная булевая переменная *arc_save* с каждым циклом будет менять свое значение, тем самым меняя значение *Arch_0_save_arc*. Реализация программы представлена ниже:

Далее необходимо добавить переменные *arc_res arc_out1…arc_out3* в окно данных архива. Обратите внимание, в «Polling» автоматически присваивается значение *write*, а так же присваивается адрес переменной:

После написания программы, необходимо создать задачу для данной программы, указать время цикла. Пользовательские данные будут записываться с циклом в 1 минуту, реализация представлена ниже:

После загрузки проекта в ПЛК на WEB-интерфейсе появятся данные об архиве.

В WEB-интерфейсе ПЛК BRIC помимо пользовательских переменных предоставлены данные архива. В таблице ниже описана структура архива:

Таблица 10: Описание структуры архива

| Имя | Тип данных | Описание |
|---|---|---|
| archive_read_buffer_0 | U8 | Данные архива пользовательских переменных и заголовка, представленных в формате U8 |
| CONFIG__ARCH_0_SAVE_ARC | U16 | Запись данных в архив |
| CONFIG__ARCH_0_ARC_FOR_READ | U32 | Данные архива из буфера, представленные для чтения |
| CONFIG__ARCH_0_ID_NUMBER | U16 | Уникальный идентификатор архива данных |
| CONFIG__ARCH_0_BODY_LEN | U16 | Размер архива с учетом заголовка и пользовательских данных |
| CONFIG__ARCH_0_UNIX_TIME_LAST_ARC | U32 | Время записи последних данных в архив в формате UNIX |
| CONFIG__ARCH_0_ARCS_NUMBER | U32 | Количество архивов, доступных для чтения |
| CONFIG__ARCH_0_LAST_READ | U32 | Номер последнего прочитанного архива |
| CONFIG__ARCH_0_FIRST_AVAILABLE | U32 | Номер первого архива, доступного для чтения |

В ПЛК BRIC с версией OS 0.30.2.5 и выше предоставлена возможность преобразования архивов в нужные

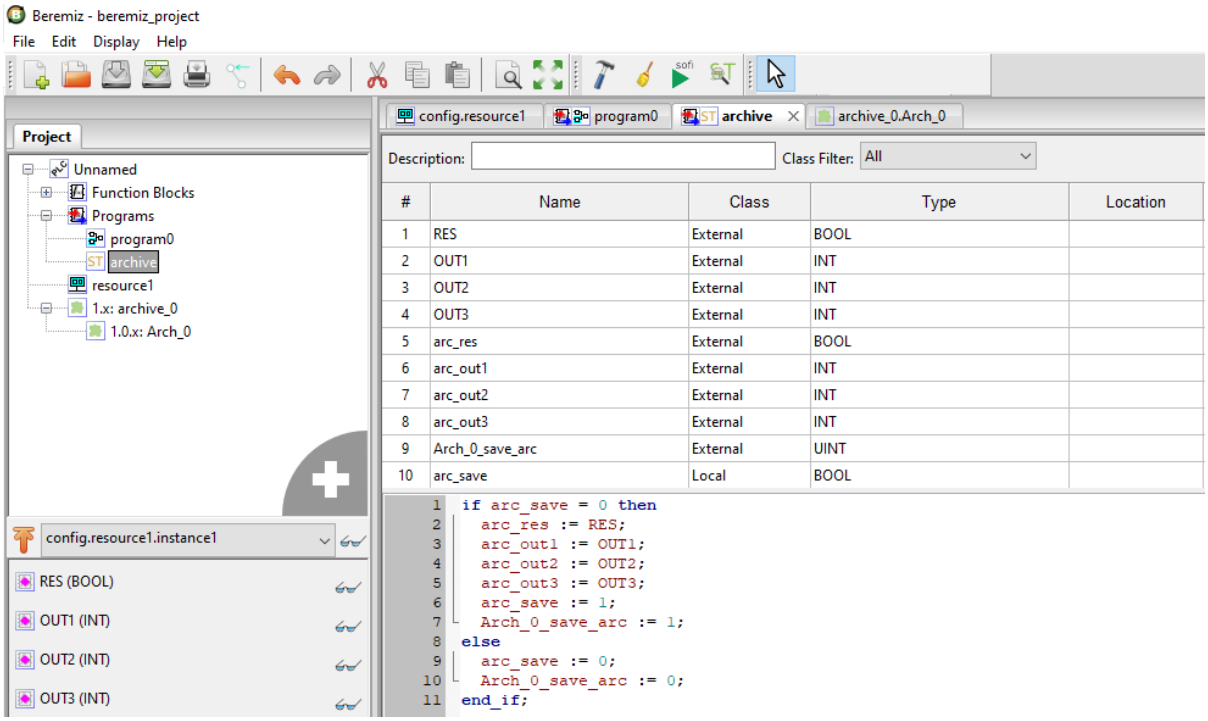



Рис. 68: Программа archive

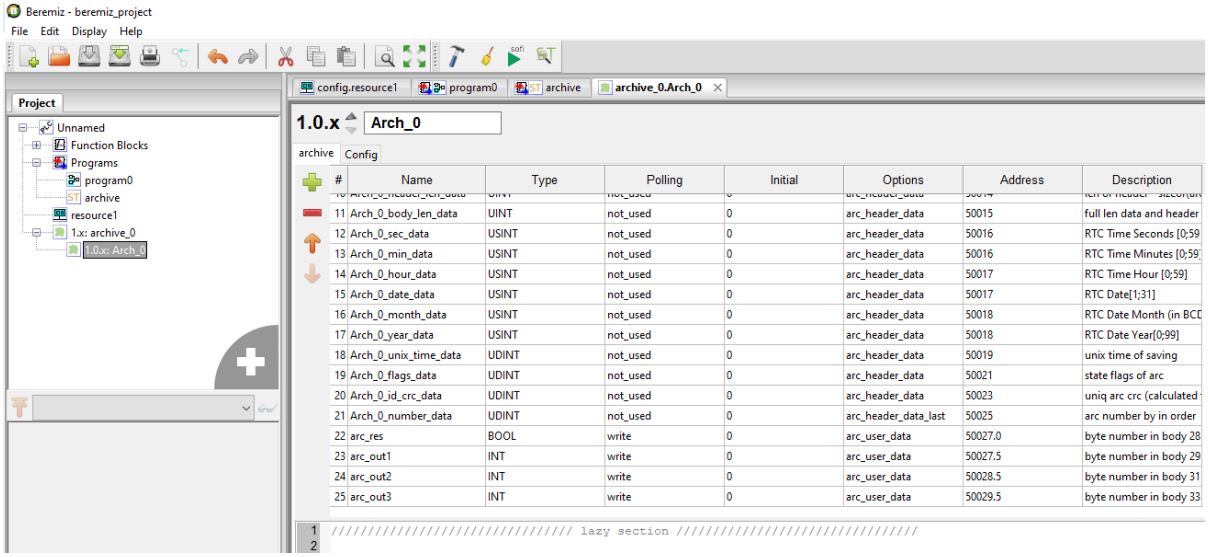



Рис. 69: Добаленные переменные

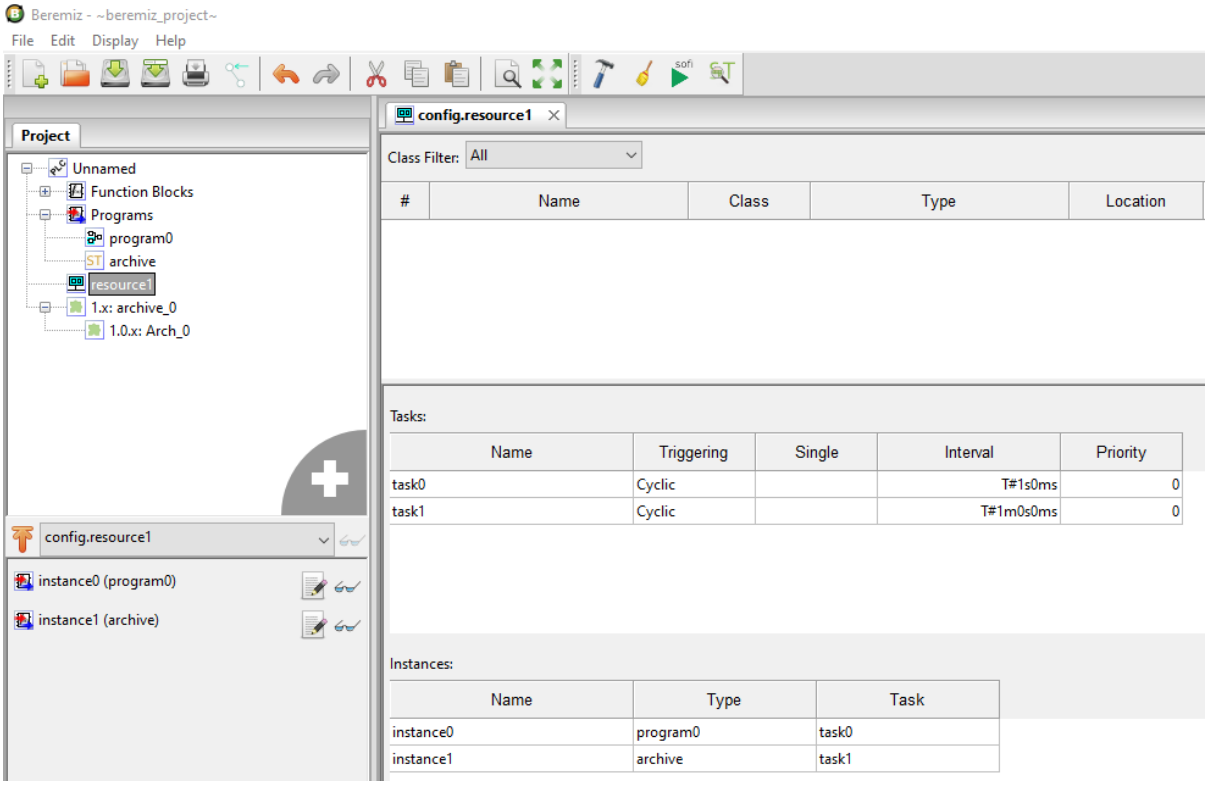



Рис. 70: Настройка ресурса программы

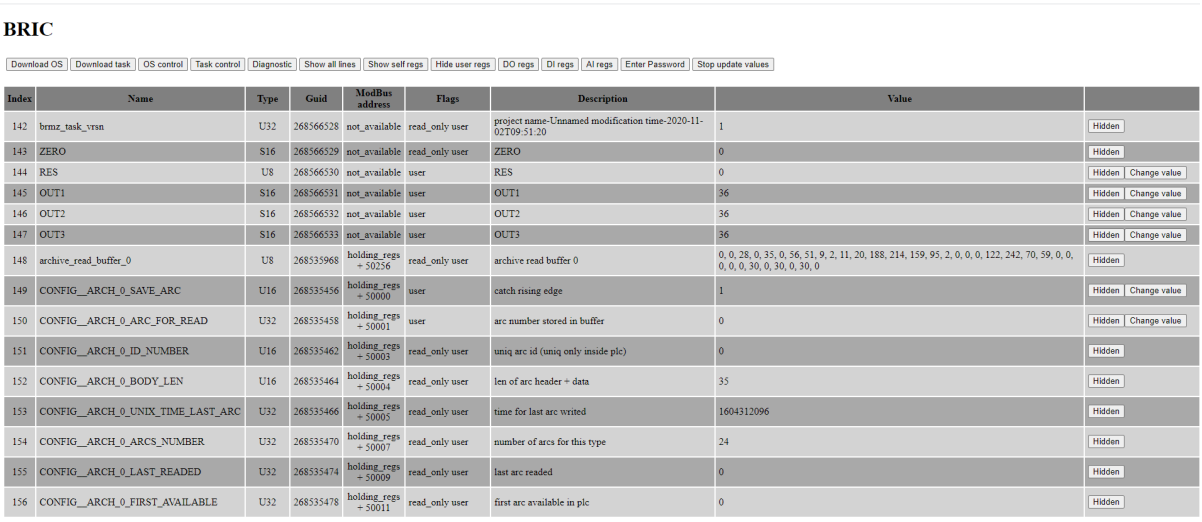



Рис. 71: WEB-интерфейс проекта

типы данных, а также сохранения их в формате *txt* для дальнейших манипуляций. Для этого необходимо зайти в WEB-интерфейс архива, введя **IP-адрес/arc.html**.

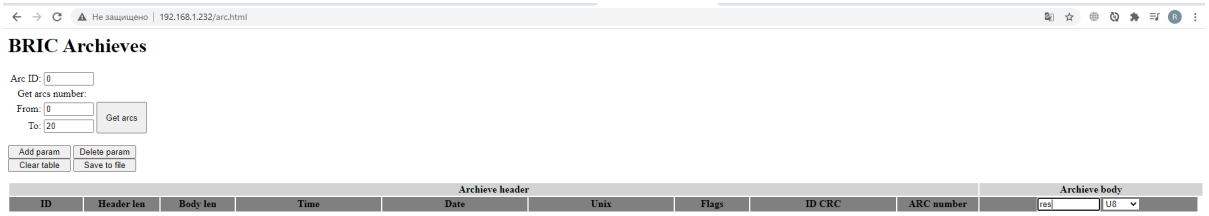



Рис. 72: WEB-интерфейс архива

В данном окне прописана информация о заголовке архива, а также первый параметр пользовательских данных – в нашем примере это переменная *RES*. Добавим и остальные переменные с помощью кнопки *Add param* и преименуя их в *OUT1…OUT3*. Далее устанавливается тип данных переменной.

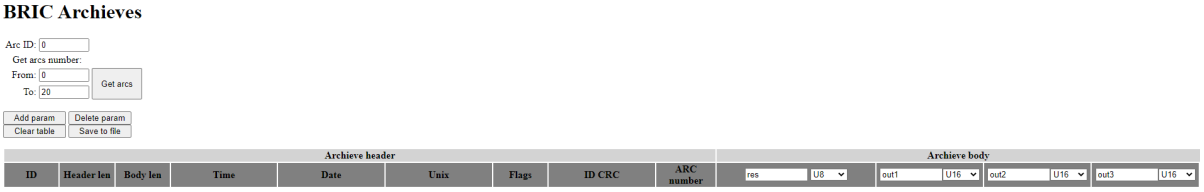



Рис. 73: WEB-интерфейс архива с пользовательскими данными

Следующим шагом является выбор номера и диапазона количества архива. Выберем от 0 до 20 архивов, после чего нажимаем кнопку *Get arcs*. Результат представлен на рисунке ниже:

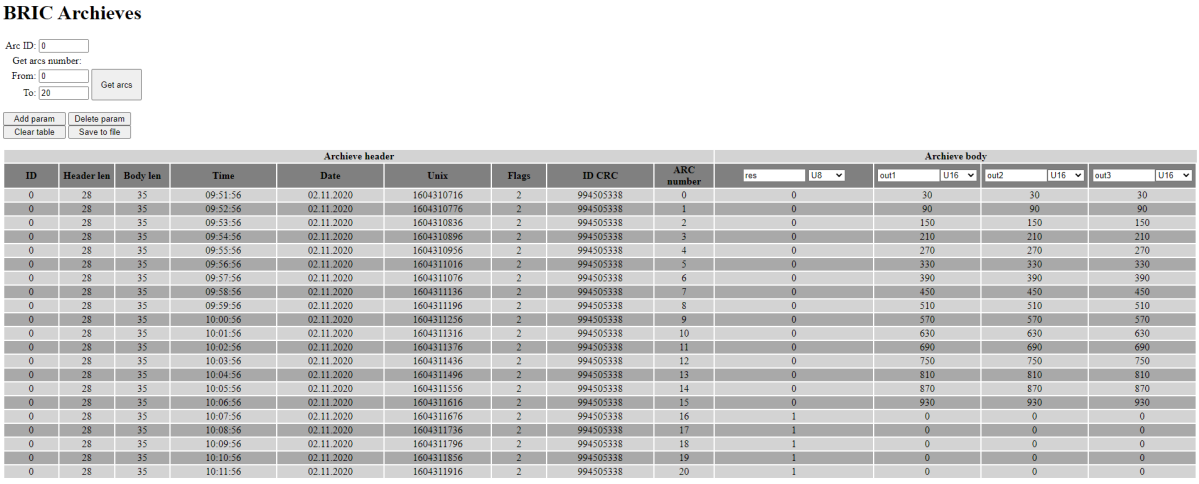



Рис. 74: Результат вывода пользовательских архивных данных

По таблице данных видно, что в момент изменения параметра *RES* с 0 на 1 все счетчики были сброшены в 0. Имеется возможность сохранить данный архив с помощью кнопки *Save to file*. Архив будет сохранен в фомате *txt* по указанному пути.

## 1.11 ПРИЛОЖЕНИЕ А. ДЕТАЛЬНОЕ ОПИСАНИЕ ПАНЕЛЕЙ ГЛАВНОГО ОКНА

Рис. 75: Панель инструментов

Комбинации быстрого вызова команд представлена в Приложении В.

Таблица 11: Кнопки панели инструментов

| Внешний вид кнопки | Название кнопки | Функция кнопки |
|---|---|---|
|  | Новый проект | Создать новый проект |
|  | Открыть проект | Открыть существующий проект |
|  | Сохранить проект | Сохранить текущий проект |
|  | Сохранить проект как | Сохранить текущий проект в определённую папку |
|  | Печать | Печать на принтере текущей программы |
|  | Проверка версии | Проверка версии |
|  | Отменить | Отмена последнего действия в среде разработки |
|  | Повторить | Повтор отменённого действия в среде разработки |
|  | Вырезать | Удалить в буфер обмена выделенные элементы в редакторе |
|  | Копировать | Копировать в буфер обмена выделенные элементы в редакторе |
|  | Вставить | Вставить из буфера обмена находящиеся там элементы в редактор |
|  | Поиск в проекте | Вызов диалога поиска данных в проекте |
|  | Развернуть/свернуть окно | Развернуть/свернуть окно на полный экран |

Дерево проекта имеет следующие опции:

- Переходить в структуру элемента проекта;
- Добавлять и удалять элементы такие как:
  - Функция;
  - Функциональный блок;
  - Программа;
  - Модуль ввода/вывода;
  - Modbus поддержка;
  - Конфигурация проекта.

Панель переменных и констант отображает с помощью таблицы все глобальные переменные и константы проекта, указанные пользователем.

Каждая переменная имеет следующие параметры:

- Имя переменной, представляющее собой уникальный идентификатор переменной в пределах её области видимости и действия;

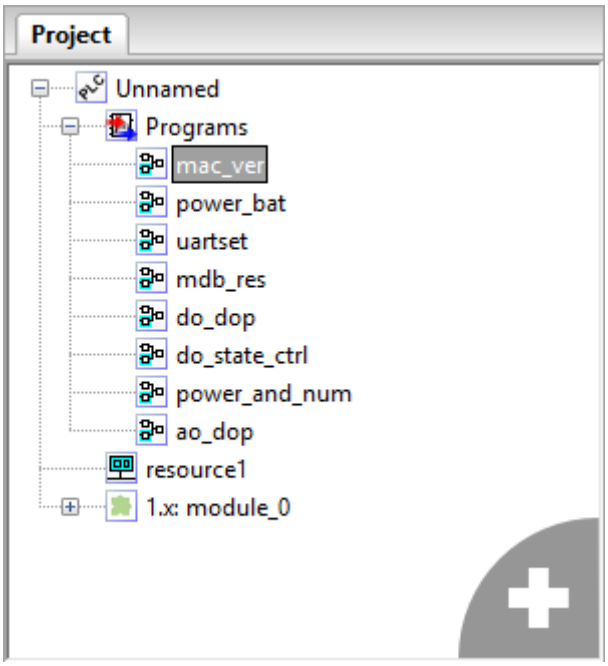



Рис. 76: Дерево проекта

mac_ver | Project

Raw IEC code | Project Files

Config variables | Project properties | Config

Class Filter: All

| # | Name | Class | Type | Location | Initial Value |
|---|---|---|---|---|---|
| 1 | DO_1_module_number0 | Global | UINT | | |
| 2 | DO_2_module_number0 | Global | UINT | | |
| 3 | DO_3_module_number0 | Global | UINT | | |
| 4 | DO_4_module_number0 | Global | UINT | | |
| 5 | DO_5_module_number0 | Global | UINT | | |
| 6 | DO_6_module_number0 | Global | UINT | | |
| 7 | DO_7_module_number0 | Global | UINT | | |
| 8 | DO_8_module_number0 | Global | UINT | | |
| 9 | AO_1_module_number0 | Global | UINT | | |
| 10 | AO_2_module_number0 | Global | UINT | | |
| 11 | DO_1_v_pwr0 | Global | REAL | | |
| 12 | DO_2_v_pwr0 | Global | REAL | | |
| 13 | DO_3_v_pwr0 | Global | REAL | | |
| 14 | DO_4_v_pwr0 | Global | REAL | | |
| 15 | DO_5_v_pwr0 | Global | REAL | | |
| 16 | DO_6_v_pwr0 | Global | REAL | | |
| 17 | DO_7_v_pwr0 | Global | REAL | | |
| 18 | DO_8_v_pwr0 | Global | REAL | | |
| 19 | AO_1_v_pwr0 | Global | REAL | | |
| 20 | AO_2_v_pwr0 | Global | REAL | | |
| 21 | DO_1_do_state0 | Global | UINT | | |

Рис. 77: Панель переменных и констант

- Класс переменной, указывающий на ее роль в структуре проекта:
- «Глобальная» (только этот тип, если указывается в главном окне проекта);
- «Входная» (указывает, что данная переменная зависит от значения переменной подаваемой на вход данного ФБ, функции);
- «Выходная» (указывает, что от данной переменной зависит значение переменной выходящей из выхода данного ФБ, функции);
- «Входная/Выходная», «Локальная» (используется только в данном ФБ, функции и удаляется по окончании работы ФБ, функции);
- «Внешняя» (возможно использовать любой программой/ФБ/функцией проекта);
- «Временная».
- Тип, определяющий тип переменной и может принадлежать базовому типу (в соответствии со стандартом IEC 61131–3: BOOL, SINT, INT, LINT, DINT, USINT, UINT, ULINT, UDINT, REAL, LREAL, BYTE, STRING, WORD, LLWORD, DWORD, TIME, DAT, TOD, DT (последние 4 могут используются только в качестве внутренних переменных)), пользовательскому типу (ФБ, массиву);
- Адрес – идентификатор, необходимый для связывания данной переменной с Modbus–переменной;
- Начальное значение – инициализация переменной некоторым начальным значением;
- Опция – задание константности, реманентности (сохранение её значения в энергонезависимой памяти) и неремanентности переменной;
- Документация – комментарий к назначению данной переменной или константы.

Первый символ имени переменной или константы должен быть буквой, или символом (_), далее могут следовать цифры, буквы латинского алфавита и символы подчеркивания.

При выборе типа переменной «Array» (Массив) появится окно «Edit array type properties» (изменение свойств массива).

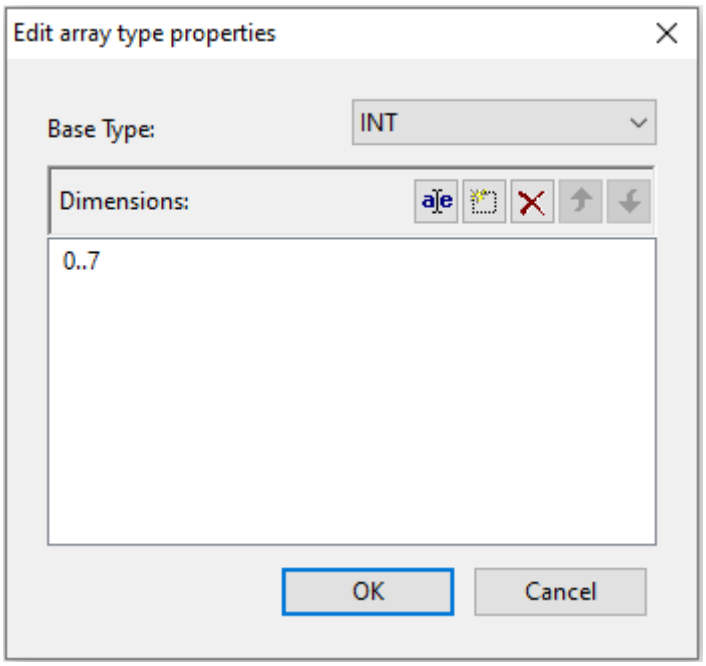



Рис. 78: Редактирование свойств массива

Свойство «Base Type» определяет какому типу будут принадлежать элементы массива. Номера элементов массива при помощи кнопки «Edit item». Пример удачного создания массива приведен на рисунке ниже

При выборе в дереве проекта элемента, соответствующего ресурсу, в панели экземпляров проекта будут отображены экземпляры, определённые в данном ресурсе, а также глобальные переменные ресурса.

| # | Name | Class | Type |
|---|---|---|---|
| 1 | LocalVar0 | Global | ARRAY [0..7] OF INT |

Рис. 79: Созданный массив на панели переменных и констант

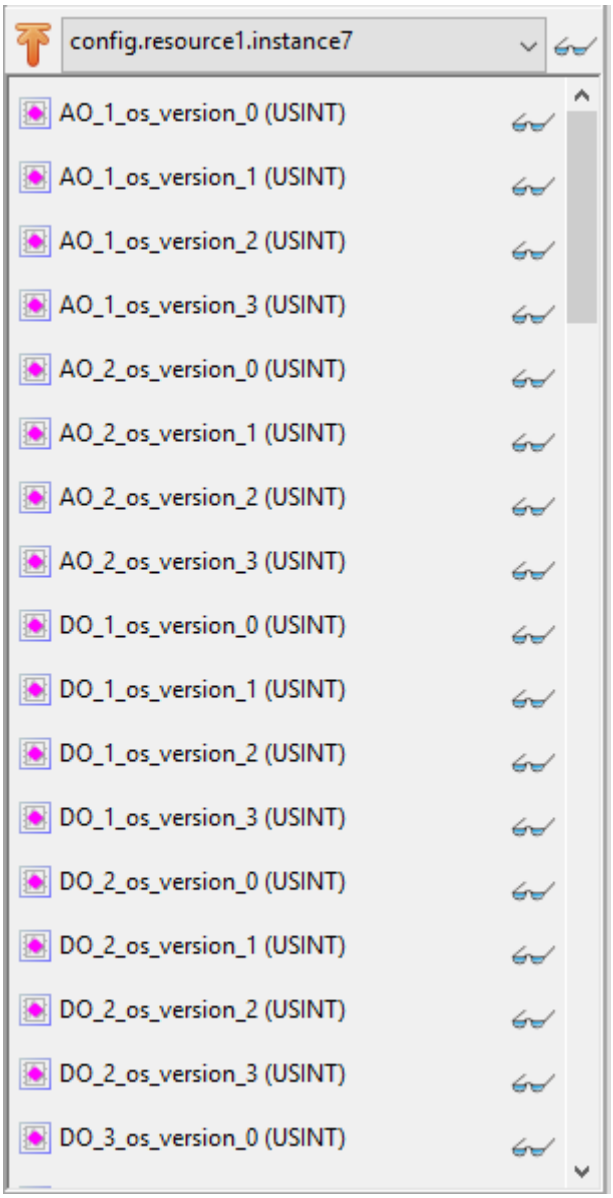



Рис. 80: Панель экземпляров проекта

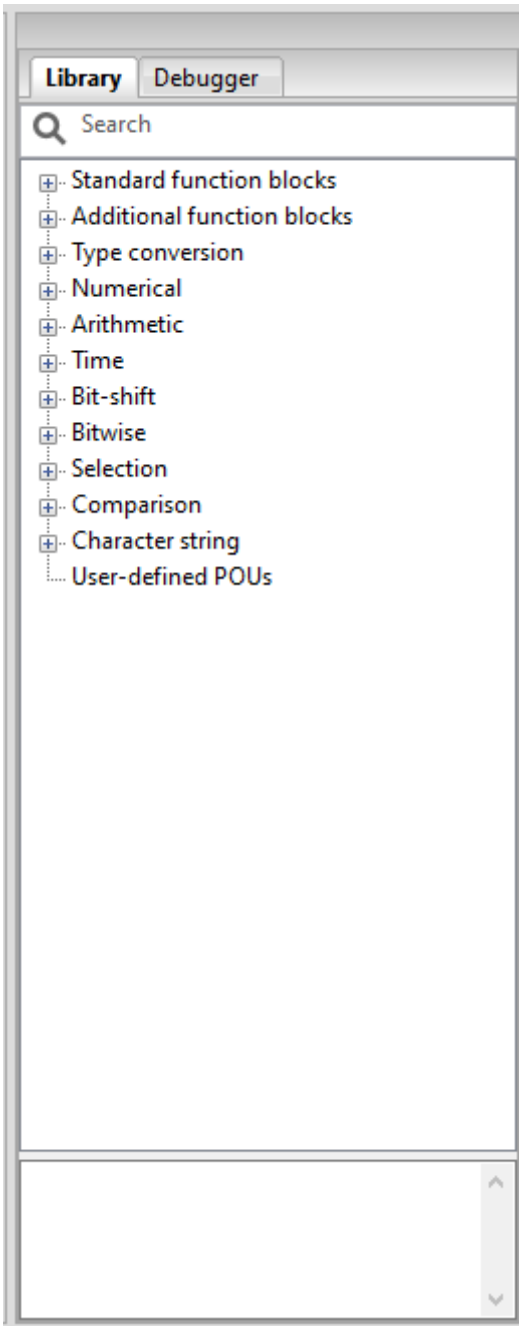



Рис. 81: Панель библиотеки функций и функциональных блоков

Панель библиотеки функций и функциональных блоков 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. 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 в область редактирования: либо текстовый редактор (ST), либо графический редактор (FBD). Имеется специальное поле поиска функционального блока по имени.

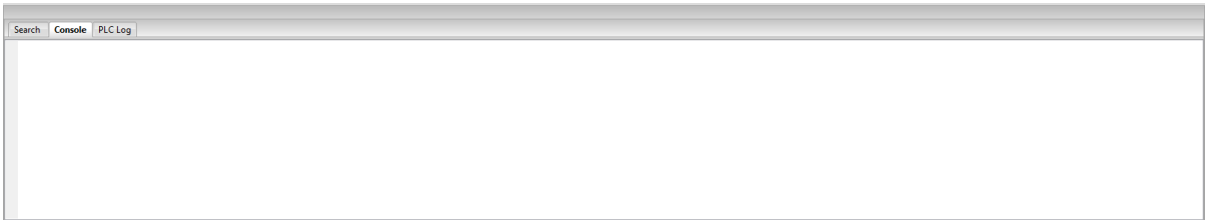



Рис. 82: Отладочная панель

Отладочная панель служит для отображения в виде текстовых сообщений:

— Результатов генерации ST и C кода;

— Результатов компиляции и компоновки прикладной программы;

— Процесса соединения и передачи прикладной программы на целевое устройство;

— Различных промежуточных манипуляций в процессы создания прикладной программы.

В случае, если необходимо вывести предупреждения ИСР Beremiz или ошибки компиляторов (MatIEC или C кода) во время их работы цвет вывода текстовых сообщений становится красным (*исключением составляет: CMake Warning: Manually–specified variables were not used by the project: CMAKE_SH*). Критические ошибки также выделяется красным цветом, но при этом еще желтым фоном (см. Приложение Д).

## 1.12 ПРИЛОЖЕНИЕ Б. ОСНОВНЫЕ ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ OPENPLC

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

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

**Плагин для модуля УСО –** интерфейс, состоящий из специальных драйверов и элементов пользовательского интерфейса для OpenPLC, 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, ФБ 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 ФБ 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 ФБ.

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

**Класс переменной –** тип использования переменной:

- Локальная (появляется при работе ФБ где фигурирует);
- Вход (локальная переменная, требующая подключения внешней переменной на вход ФБ/функции, где она фигурирует);
- Выход (локальная переменная, требующая подключения внешней переменной на выход из ФБ/функции, где она фигурирует);
- Вход/Выход (локальная переменная, требующая подключения внешней переменной на выход и вход ФБ/функции, где она фигурирует),
- Внешняя (сохраняется в адресном пространстве ПЛК)[1] .

**Исходное значение –** значение переменной на момент запуска программы на ПЛК.

**Настройка переменной –** возможность изменение переменной:

- constant (неизменная);
- retain (сохранение значения при перезагрузке ПЛК);
- non-retain (сброс значения при перезагрузке).

## 1.13 ПРИЛОЖЕНИЕ В. КОМБИНАЦИИ БЫСТРОГО ВЫЗОВА КОМАНД ИСР BEREMIZ

Часть операций, выполняемых с помощью выбора определённого пункта меню мышью, могут быть исполнены с помощью «горячей клавиши». Далее будет подробно описаны «горячие клавиши».

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

- «Новый» – создание нового проекта (CTRL + N);
- «Открыть» – открытие существующего проекта (CTRL + O);
- «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 для открытого проекта (CTRL + W);
- «Закрыть проект» – закрыть открытый проект (CTRL + SHIFT + W);

[1] Не поддерживаются типы переменных TIME, DATE, TOD, DT, STRING

- «Параметры страницы» – настройка параметров страницы для печати на принтере активной программы, представленной в виде диаграммы (CTRL + ALT + P);
- «Просмотр» – предварительный просмотр перед печатью на принтере активной программы (CTRL + SHIFT + P);
- «Печать» – печать на принтере активной программы (CTRL + P);
- «Выход» – закрытие текущего проекта и выход из программы OpenPLC (CTRL+ Q);
- «Отменить» – отмена последнего действия в редакторе (CTRL + Z);
- «Повторить» повтор отменённого действия в редакторе (CTRL + Y);
- «Вырезать» – удалить в буфер обмена выделенные элементы в редакторе (CTRL + X);
- «Копировать» – копировать в буфер обмена выделенные элементы в редакторе (CTRL + C);
- «Вставить» – вставить из буфера обмена находящиеся там элементы в редактор (CTRL + V);
- «Поиск в проекте» – вызов диалога поиска данных в проекте (CTRL + SHIFT + F);
- «Выделить всё» – выделение всех элементов в активной вкладке редактора (CTRL +A);
- «Обновить» – обновление данных и снятие выделения в редакторе (CTRL + R);
- «Очистить ошибки» – очистка указателей ошибок в редакторе (CTRL + K).

## 1.14 ПРИЛОЖЕНИЕ Г. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПО РАБОТЕ С MODBUS

**Последовательный порт** — наименование канала, по которому осуществляется связь по протоколу.

**Скорость** — скорость передачи данных по каналу.

**Чётность** — выбор режима (проверка на четность/проверка на нечетность/без проверки).

**Стоп–биты** — количество битов оповещающих об окончании сообщения.

**Период опроса** — время между повторным опросом.

**Адрес устройства** — указание идентифицируемого номера устройства с кем осуществляется обмен данными

**Число каналов** — сколько регистров/реле используются в записи/чтении (для функций 05, 06 всегда равно 1).

**Начальный адрес** — номер первого регистра/реле используемого в записи/чтении.

**Таймаут** — время необходимое для ответа slave–устройству.

а) Функция 01 (ReadCoils) — Команда используется для получения состояний определенного количества реле, начиная с указанного в запросе. Состояние одного реле при этом передается одним битом. Если бит установлен в 1 – реле включено, если 0 – реле отключено.

б) Функция 02 (ReadInputDiscretes) — Команда используется для получения состояний определенного количества дискретных входов, начиная с указанного в запросе. Состояние одного входа при этом передается одним битом. Если бит установлен в 1 – вход замкнут, если 0 – вход разомкнут.

в) Функция 03 (ReadHoldingRegisters) — Команда используется для чтения указанного количества 2–Байтных регистров.

г) Функция 04 (ReadInputRegisters) — Команда используется для получения состояний определенного количества 2–Байтных регистров, хранящих состояние дискретных входов, начиная с указанного в запросе. Значение одного регистра передается двумя байтами.

д) Функция 05 (WriteSingleCoil) — Команда используется для включения/отключения одного реле. Требуемое состояние реле передается двумя байтами.

е) Функция 06 (WriteSingleRegister) — Команда выполняет запись нового значения в указанный регистр.

ж) Функция 15 (WriteMultipleCoils) — Команда используется для групповой установки состояний определенного количества реле, начиная с указанного. Состояние одного реле при этом передается одним битом. Если бит установлен в 0 – реле отключено, если 1 – реле включено.

з) Функция 16 (WriteMultipleRegisters) — Команда выполняет запись новых значений в указанные регистры.

## 1.15 ПРИЛОЖЕНИЕ Д. ЧАСТО ВСТРЕЧАЕМЫЕ ОШИБКИ ПРИ НАПИСАНИИ ПРОЕКТА В OPENPLC

Причиной возникновения ошибки, указанной на рисунке ниже является использование типа переменной (TIME, DATE, TOD, DT, STRING) не поддерживаемой в классе «внешняя» для платформы «Sofi». Для исправления ошибки необходимо перевести их в локальный класс переменных и при необходимости изменения/считывания их использовать преобразователи типов переменных.

```
C:\Beremiz/sofi/generator/generator.py "C:\\Beremiz\\mingw\\bin"
Traceback (most recent call last):
  File "C:\Beremiz/sofi/generator/generator.py", line 330, in <module>
    main()
  File "C:\Beremiz/sofi/generator/generator.py", line 76, in main
    make(args.path_beremiz, args.path_lib, args.path_gcc, args.path_cmake)
  File "C:\Beremiz/sofi/generator/generator.py", line 120, in make
    generate(base_object.FREERTOS_OS)
  File "C:\Beremiz/sofi/generator/generator.py", line 175, in generate
    plc_xml.find_vars_description()
  File "C:\Beremiz\sofi\generator\xml_analysis.py", line 91, in find_vars_description
    var_dict["byte_size"] = self.iec_type_size(var_dict["type"])
  File "C:\Beremiz\sofi\generator\base_object.py", line 255, in iec_type_size
    return self.iec_type[iec_type]
KeyError: 'TIME'
C:\Beremiz/python3/python-3.7.2/python.exe C:\Beremiz/sofi/generator/generator.py -path_b D:\test\test_req\test_time\build/ -path_l C:\Beremiz/sofi/generator/template/freertos/lib/ -path_gcc C:\Beremiz/gcc_6_3_1/bin/ -path_
cmake C:\Beremiz/cmake-3-13-1-win32-x86/bin/ -command none
exited with status 1 (pid 9900)
C compilation failed.
```

Причиной возникновения ошибки, указанной на рисунке ниже является использование в программном модуле переменных внешнего класса не указанных на главной странице.

```
Start build in D:\test\test_req\test_time\build
Generating SoftPLC IEC-61131 ST/IL/SFC code...
Compiling IEC Program into C code...
"C:\Beremiz\matiec\iec2c.exe"  -f -l -p -I "C:\Beremiz\matiec\lib" -T "D:\test\test_req\test_time\build" "D:\test\test_req\test_time\build\plc.st"
exited with status 1 (pid 17440)
D:\test\test_req\test_time\build\plc.st:15-10..15-13: error: Declaration error. The external variable does not match with any global variable.
In section: PROGRAM program0
0015:     L6 : REAL;
1 error(s) found. Bailing out!

Error : IEC to C compiler returned 1
PLC code generation failed !
```

Причиной возникновения ошибки, указанной на рисунке ниже является использование функциональных блоков, поддерживаемых только платформой «Sofi». В разделе Config необходимо поменять платформу на «Sofi».

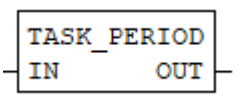
```
Start build in D:\test\test_req\test_time\build
Generating SoftPLC IEC-61131 ST/IL/SFC code...
Compiling IEC Program into C code...
Extracting Located Variables...
C code generated successfully.
PLC :
   [CC]  plc_main.c -> plc_main.o
   [CC]  plc_debugger.c -> plc_debugger.o
In file included from D:\test\test_req\test_time\build\plc_debugger.c:24:0:
D:\test\test_req\test_time\build\/POUS.h:25:3: error: unknown type name 'READ_DI'
D:\test\test_req\test_time\build\plc_debugger.c:103:33: error: request for member 'EN' in something not a structure or union
D:\test\test_req\test_time\build\plc_debugger.c:104:33: error: request for member 'ENO' in something not a structure or union
D:\test\test_req\test_time\build\plc_debugger.c:105:33: error: request for member 'DI_OUT' in something not a structure or union
"gcc" -c "D:\test\test_req\test_time\build\plc_debugger.c" -o "D:\test\test_req\test_time\build\plc_debugger.o" -O2  "-IC:\Beremiz\matiec\lib\C" -Wno-unused-function
exited with status 1 (pid 10288)
C compilation of plc_debugger.c failed.
C Build failed.
```

Причиной возникновения ошибки, указанной на рисунке ниже является отсутствие подключенной переменной на входе в функциональный блок.

Причиной возникновения ошибки, указанной на рисунке ниже является неверный адрес на несуществующий путь в поддержке Modbus или несоответствие типа указанного в адресе переменной.

Причиной возникновения ошибки, указанной на рисунке ниже является неверное название папки проекта (наличие пробела).

Причиной возникновения ошибки, указанной на рисунке ниже является отсутствие task в resource и не все программы проинициализированы в Instances.



```
Start build in D:\test\test_req\test_time\build
Generating SoftPLC IEC-61131 ST/IL/SFC code...
Compiling IEC Program into C code...
"C:\Beremiz\matiec\iec2c.exe"  -f -l -p -I "C:\Beremiz\matiec\lib" -T "D:\test\test_req\test_time\build" "D:\test\test_req\test_time\build\plc.st"
exited with status 1 (pid 1524)
D:\test\test_req\test_time\build\plc.st:32-36..32-37: error: no parameter defined in function invocation of ST expression.
In section: PROGRAM program0
0032:   TIME_TO_REAL4_OUT := TIME_TO_REAL();

1 error(s) found. Bailing out!

Error : IEC to C compiler returned 1
PLC code generation failed !
```

```
Start build in D:\test\test_req\test_time\build
Generating SoftPLC IEC-61131 ST/IL/SFC code...
Compiling IEC Program into C code...
"C:\Beremiz\matiec\iec2c.exe"  -f -l -p -I "C:\Beremiz\matiec\lib" -T "D:\test\test_req\test_time\build" "D:\test\test_req\test_time\build\plc.st"
exited with status 1 (pid 7808)

Internal compiler error in file generate_location_list.cc at line 150.

Error : IEC to C compiler returned 1
PLC code generation failed !
```

Search | Console | PLC Log

```
-- Build files have been written to: D:/test/Modbus_grupp/Modbus_area_100_reg_overlay ranges/build/sofi/freertos/cmake_arm

Scanning dependencies of target sofi_task.elf
[  8%] Building C object CMakeFiles/sofi_task.elf.dir/src/beremiz_regs_description.c.obj
[ 16%] Building C object CMakeFiles/sofi_task.elf.dir/src/beremiz_task.c.obj
[ 25%] Building C object CMakeFiles/sofi_task.elf.dir/src/config_task.c.obj
[ 33%] Building C object CMakeFiles/sofi_task.elf.dir/src/modbus_master.c.obj
[ 41%] Building C object CMakeFiles/sofi_task.elf.dir/src/os_service.c.obj
[ 50%] Building C object CMakeFiles/sofi_task.elf.dir/src/plc_debugger.c.obj
[ 58%] Building C object CMakeFiles/sofi_task.elf.dir/src/plc_main.c.obj
[ 66%] Building C object CMakeFiles/sofi_task.elf.dir/src/POUS.c.obj
[ 75%] Building C object CMakeFiles/sofi_task.elf.dir/src/resource1.c.obj
[ 83%] Building C object CMakeFiles/sofi_task.elf.dir/src/sofi_beremiz.c.obj
[ 91%] Building C object CMakeFiles/sofi_task.elf.dir/src/sofi_dev.c.obj
[100%] Linking C executable sofi_task.elf
arm-none-eabi-gcc.exe: error: ranges/build/sofi/freertos/build/output.map: No such file or directory
arm-none-eabi-gcc.exe: error: ranges/build/sofi/freertos/ldscript/task_internal_flash.ld: No such file or directory
make.exe[2]: *** [sofi_task.elf] Error 1
CMakeFiles\sofi_task.elf.dir\build.make:232: recipe for target 'sofi_task.elf' failed
CMakeFiles\Makefile2:71: recipe for target 'CMakeFiles/sofi_task.elf.dir/all' failed
make.exe[1]: *** [CMakeFiles/sofi_task.elf.dir/all] Error 2
Makefile:82: recipe for target 'all' failed
make.exe: *** [all] Error 2
D:\test\Modbus_grupp\Modbus_area_100_reg_overlay ranges\build//sofi/freertos/log.log
C:\Beremiz/sofi/generator/generator.py 'C:\\Beremiz\\mingw\\bin'
routes 0
start building
Successfully built.
```

Search | Console | PLC Log

```
Start build in D:\test\LD\build
Generating SoftPLC IEC-61131 ST/IL/SFC code...
Compiling IEC Program into C code...
"C:\Beremiz\matiec\iec2c.exe"  -f -l -p -I "C:\Beremiz\matiec\lib" -T "D:\test\LD\build" "D:\test\LD\build\plc.st"
exited with status 1 (pid 13808)
D:\test\LD\build\plc.st:32-10..33-14: error: unknown error in resource declaration.
In section: CONFIGURATION config
0032:   RESOURCE resource1 ON PLC
0033:   END_RESOURCE

1 error(s) found. Bailing out!

Error : IEC to C compiler returned 1
PLC code generation failed !
```