TOPAZ ALGORITHM CREATOR

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Москва 2021

СОДЕРЖАНИЕ

Основные термины и определения

DAS - Data Access Server - Программно-технический комплекс, включающий в себя промышленный контроллер под управлением ОС Линукс и специализированное ПО для выполнения сбора и обработки телемеханической информации.

База текущих значений DAS (база данных DAS) - Область памяти содержащая текущие значения телемеханических параметров с меткой времени последнего изменения и дополнительными флагами. Состоит из трех таблиц, по одной для каждого типа параметров:

- **дискреты** - целочисленные данные, значения от 0 до 8191 (13 бит);
- **аналоги** - данные с плавающей запятой;
- **счетчики** - целочисленные данные, значения от 0 до 4294967295 (32 бита);

Размер каждой таблицы - 65535 записей.
Каждый параметр имеет так же флаги: *недостоверность*, *динамика*, *блокировка*, *замещение*, *актуальность*. Аналоги так же имеют флаг *переполнения.*
Флаги *актуальности* и *переполнения* не могут быть изменены из среды разработки.

Библиотека - Организованный набор пользовательских программ, имеющий уникальное имя в пределах проекта TOPAZ TMBuilder. Библиотека - минимальная единица, загружаемая для исполнения в контроллер.

Загрузчик 61131(iec61131loader) – программный модуль, загружаемый в контроллер. Является подгружаемой библиотекой основного процесса iec-controls.

Исполняемый модуль 61131 – программный модуль, загружаемый в контроллер. Является подгружаемой библиотекой основного процесса iec61131loader.so. Представляет собой исполняемый файл с пользовательскими программами, написанным по стандарту МЭК 61131.

Пользовательская программа (сценарий, прикладная задача) - законченная часть программы, имеющая уникальное (в пределах проекта имя) и один или более входных параметров.

1. О программе

Программный комплекс **TOPAZ ALGORITHM CREATOR** является составной частью программного комплекса **TOPAZ TMBuilder** и предназначен для реализации возможности дополнительной обработки телемеханических сигналов путем исполнения алгоритмов пользователей (пользовательских программ). Программы разрабатываются на языке функциональных блоков (FBD), структурированного текста (ST) и линейных диаграмм (LD) стандарта МЭК 61131.

1.1 Основные функции программного компонента:

- Реализация программного доступа к данным базы текущих значений IEC DAS
- Создание пользовательских алгоритмов для обработки значений телемеханических параметров
- Создание пользовательских блоков для дальнейшего использования в алгоритмах
- Интеграция с ПО **TOPAZ TMBuilder** для возможности загрузки и выполнения пользовательских функций в контроллерах под управлением ПО **TOPAZ DAS**

1.2 Особенности структуры и функционирования программного компонента:

Программный комплекс **TOPAZ ALGORITHM CREATOR** состоит из нескольких подпрограмм:

1. **Конфигуратор 61131** – подпрограмма TOPAZ TMBuilder для создания, описания и структурированного хранения пользовательских библиотек
2. **Дизайнер 61131** - подпрограмма Конфигуратора IEC61131 для создания, описания, структурированного хранения и компиляции пользовательских программ внутри библиотек.
3. **Просмотрщик 61131** - подпрограмма Конфигуратора IEC61131 для просмотра состояния выходов и выходов функциональных блоков в режиме реального времени.

Программный компонент на уровне исполнительного ПО состоит из ***iec61131loader.so*** и непосредственно скомпилированной пользовательской библиотеки(исполняемого модуля 61131) ***user_lib_name.so***.

Программный компонент является событийным (работает по изменению входных переменных), что обеспечивает низкую нагрузку на ЦП. При запуске компонента выполняется первичный прогон всех схем и далее подписка на события. Промежуточным звеном между экземпляром библиотеки и базой DAS является *dbproxy*. *dbproxy* подписывается на события (изменение состояния и\или метки времени) сигнала в базе DAS и, если изменение произошло, делаем вызов библиотеки. Так же в случае использования таймеров внутри пользовательских алгоритмов расчет времени таймеров вынесен в *dbproxy*. В остальное время экземпляр алгоритма не обрабатывается ЦП.

1.3 Установка на персональный компьютер

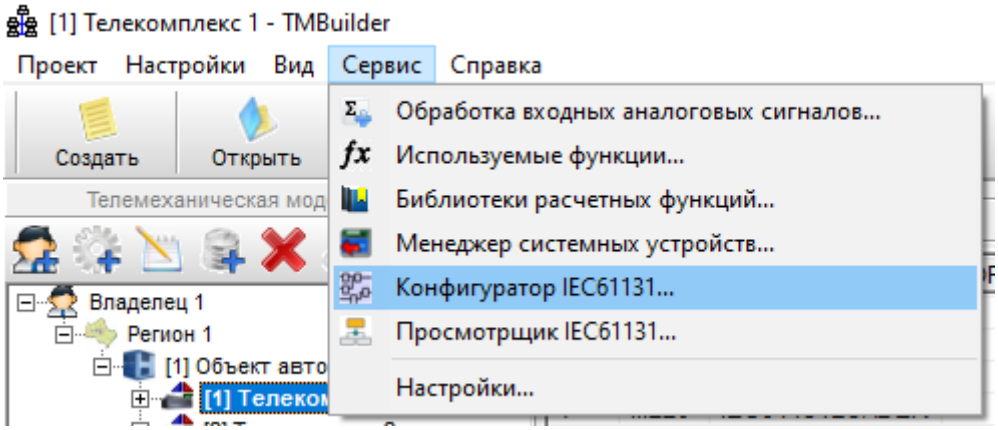
Для установки **TOPAZ ALGORITHM CREATOR** на персональный компьютер необходимо запустить инсталляционные файлы

TOPAZ Algorithm Creator vX. X. X. XXXX.exe
TOPAZ Algorithm Creator X. X. X. XXXX Addon vX.X.X.XXXX.exe
где **X** - текущая версия дистрибутива.

2. Пользовательский интерфейс

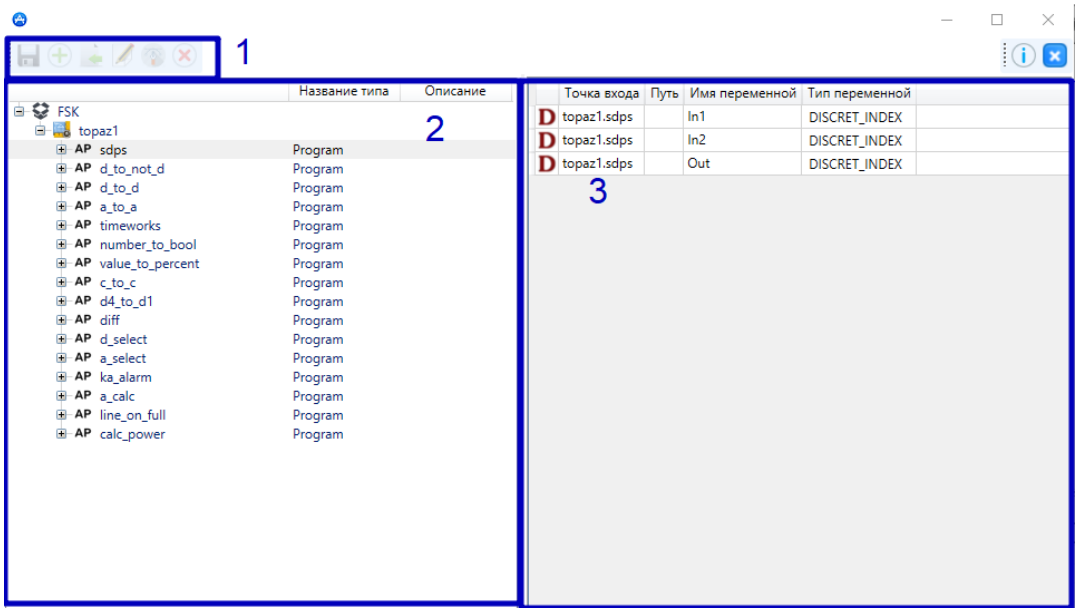
2.1 Начало работы

Конфигуратор IEC61131 вызывается из **TOPAZ TMBuilder** (меню «Сервис»)



2.2 Элементы главного окна

Пример главного окна показан на рисунке ниже



1. *Панель инструментов* - служит для управления проектом и доступа ко всем функциям. Содержит следующие элементы:

Кнопка панели	Описание
	Сохранение текущего проекта
	Создание нового проекта
	Добавление уже имеющегося проекта
	Редактирование проекта
	Открывает проект в Дизайнере

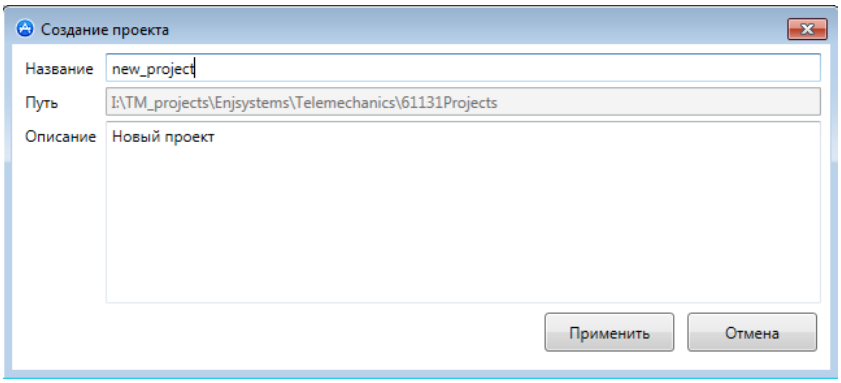
	Удаление проекта

2. *Структура проекта* – отображает структуру текущего проекта в виде дерева: Проект, программа, переменные
3. *Таблица сигналов* – табличное представление переменных с указанием их типа.

2.3 Создание библиотеки

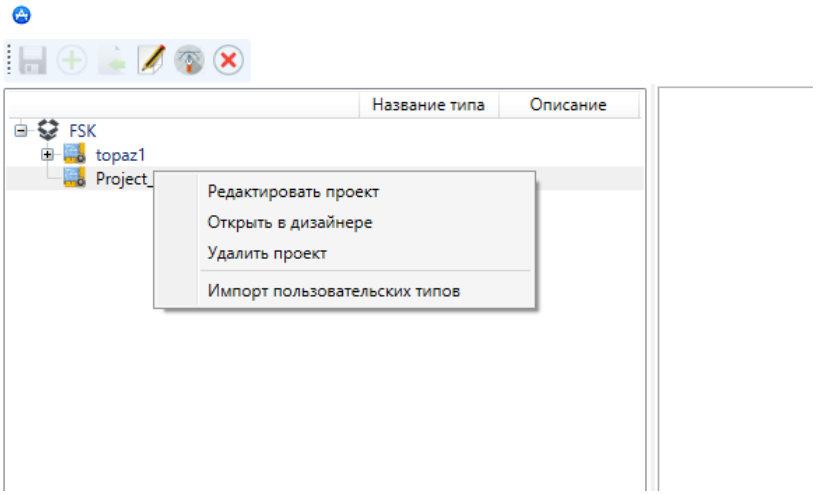
Создание библиотеки происходит в **Конфигураторе IEC61131**.

Для создания новой библиотеки надо нажать кнопку на панели инструментов. Откроется диалоговое окно, в котором необходимо ввести название и (необязательно) описание. Имя и описание можно изменить в любой момент.



В названии библиотеки допускается использование латинских букв, знаков подчеркивания и цифр. В описании можно использовать кириллицу.

После этого в дереве появится новый проект, Редактирование имени и описания доступно из контекстного меню по правому клику мыши:



Если проект уже был создан, его можно добавить, нажав на кнопку на панели инструментов. Импортировать стоит файл с именем ***plc.xml***, который располагается в директории:

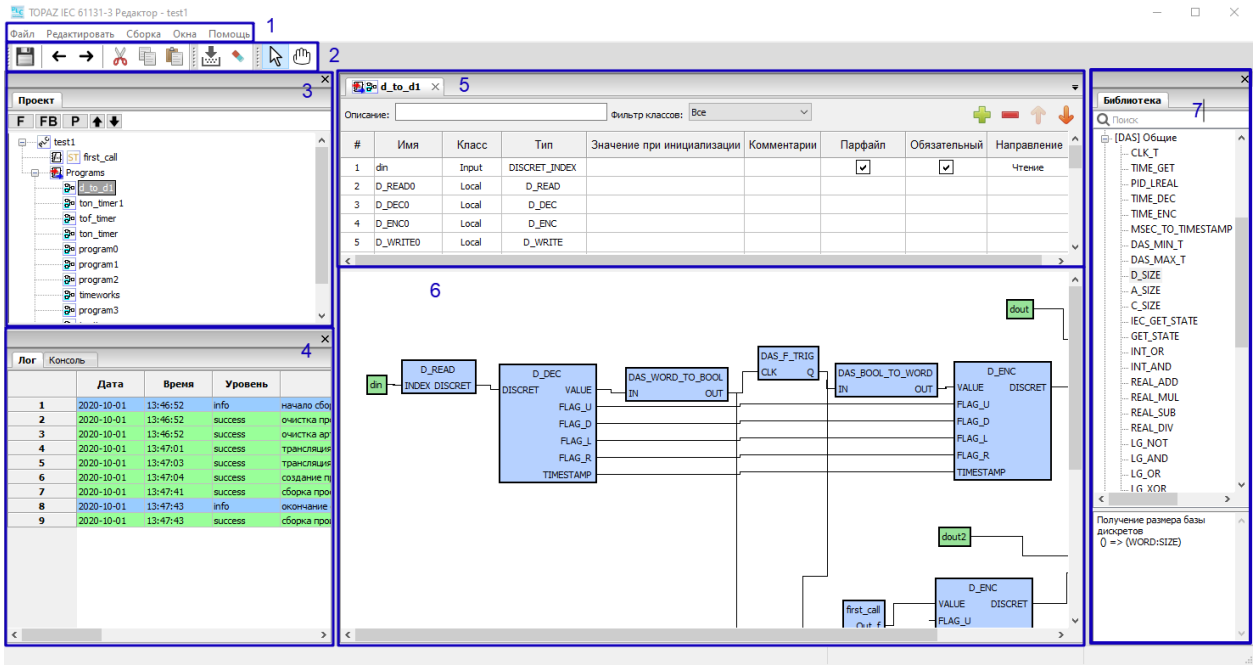
Папка_проекта\Telemechanics\61131Projects\new_project

Во время работы можно создавать или открывать несколько проектов одновременно.

3. Создание и загрузка проекта

3.1 Интерфейс Дизайнера

Подпрограмма Дизайнер открывается нажатием соответствующей кнопки  на панели инструментов.



Окно **Дизайнера** состоит из нескольких областей.

1. *Меню*
2. *Панель инструментов*
3. *Дерево проекта*
4. *Лог и Консоль*
5. *Список переменных и блоков*
6. *Рабочая область*
7. *Библиотека блоков*

Панель инструментов имеет следующие кнопки:

Кнопка панели	Описание
	Сохранение проекта
	Отменить/вернуть последнее действие
	Вырезать элемент
	Копировать элемент

	Вставить элемент
	Построение проекта(компиляция)
	Очистить проект
	Режим выделения объектов
	Режим перемещения рабочей области

Дерево проекта представляет текущий проект в структурированном виде. Оно содержит несколько подразделов:

1. Функция (function)
2. Функциональный блок (function block)
3. Программа (POU)

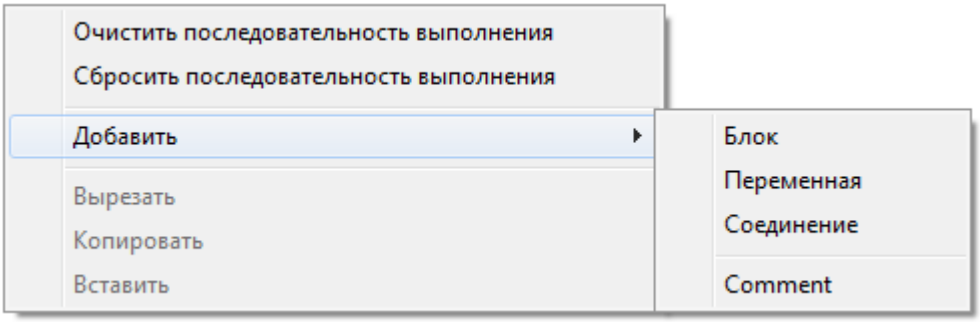
Список переменных и блоков содержит перечень всех используемых переменных и блоков, при этом для переменных существует несколько настраиваемых параметров:

	Описание
#	Номер переменной
Имя	Уникальный в пределах функции\блока\программы идентификатор переменной
Class	Класс переменной - локальная(Local), входная(Input), выходная(output), входная-выходная(InOut). Для всех сигналов, которые читаются/пишутся в базу DAS должен быть Input
Тип	Для переменных DAS – тип(дискреты, аналоги, счетчики и тд), для числовых переменных указывается типа данных, для блоков автоматически проставляется его название
Значение при инициализации	Значение, которое принимает переменная при первом вызове
Комментарий	Пользовательское описание переменной
Парфайл	отмечается чекбоксом, если мы задаем в ПАРФАЙЛе ретрансляцию в/из алгоритма 61131
Обязательный	Отмечается, если данная входная переменная является обязательной. В случае, если данный чекбокс отмечен и отсутствует привязка к данной переменной - TMBuilder будет сообщать об этом.
Направление	Указывается переменная записывается в базу DAS или читается из нее. По сути представляет собой флаг подписки на события(изменение) этой переменной.

Добавить/удалить переменную или блок можно с помощью кнопок  справа от таблицы. Упорядочить взаимное положение - с помощью кнопок  .

Программа составляется непосредственно в *рабочей области* проекта.

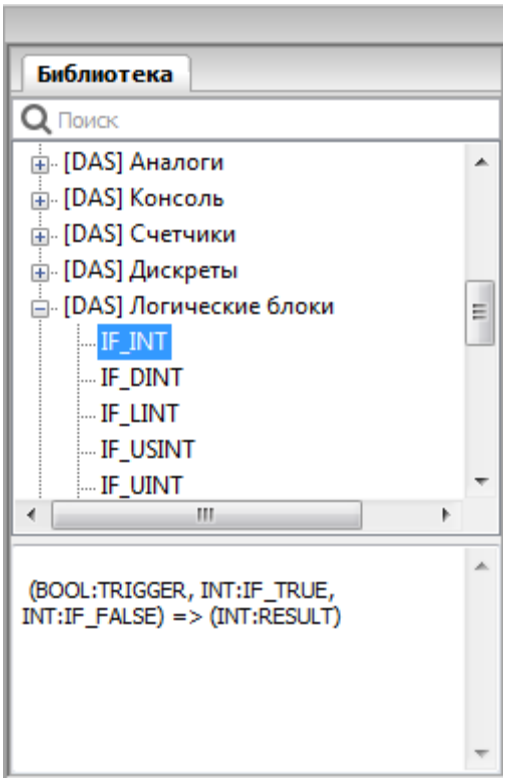
Необходимый блок или переменную можно добавить либо из Библиотеки, либо вызвав контекстное меню правой клавишей мыши:



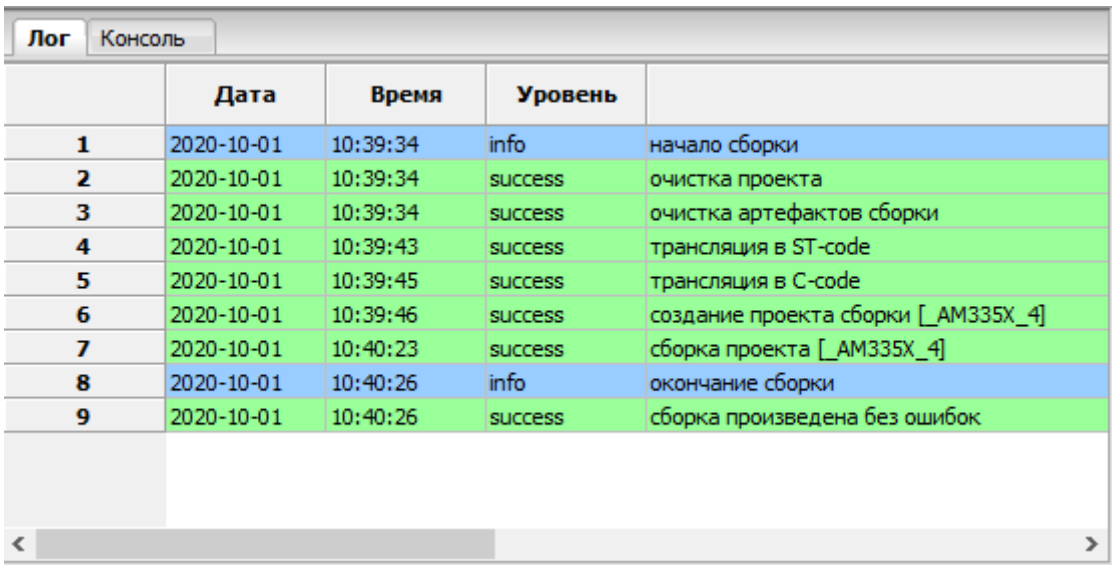
Блоки в *Библиотеке* упорядочены по разделам. Блоки с префиксом DAS имеют каноническое исполнение и работают согласно стандарту МЭК 61131, кроме блоков необходимых для работы с базой DAS.

В случае, если какой-то функциональный блок есть с префиксом DAS и без него, рекомендуется использовать первый вариант.

В нижнем поле *Библиотеки* показываются входные и выходные переменные блока, а так же формат данных.



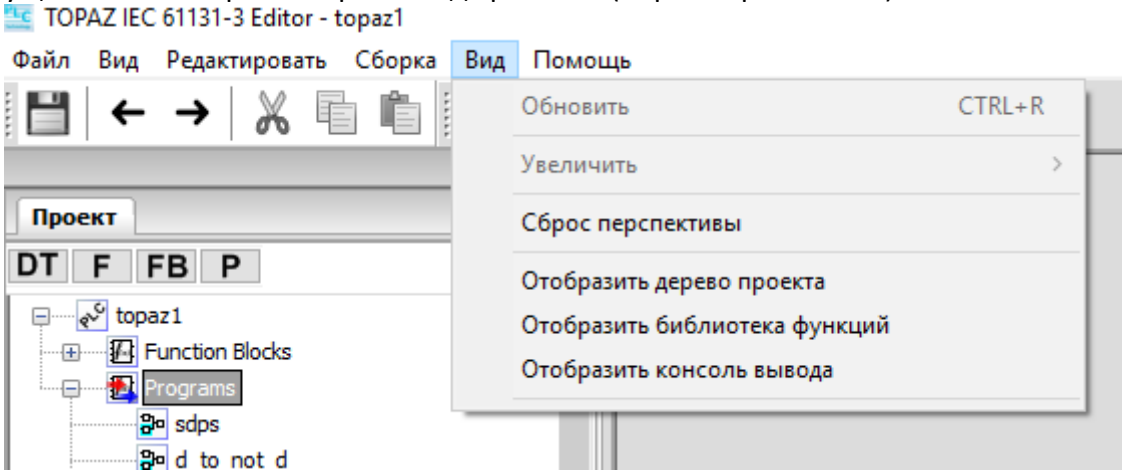
Консоль и лог необходимы при компиляции



	Дата	Время	Уровень	
1	2020-10-01	10:39:34	info	начало сборки
2	2020-10-01	10:39:34	success	очистка проекта
3	2020-10-01	10:39:34	success	очистка артефактов сборки
4	2020-10-01	10:39:43	success	трансляция в ST-code
5	2020-10-01	10:39:45	success	трансляция в C-code
6	2020-10-01	10:39:46	success	создание проекта сборки [_AM335X_4]
7	2020-10-01	10:40:23	success	сборка проекта [_AM335X_4]
8	2020-10-01	10:40:26	info	окончание сборки
9	2020-10-01	10:40:26	success	сборка произведена без ошибок

Все области дизайнера можно перемещать, изменять ширину или скрывать (пиктограмма крестика на соответствующем подокне).

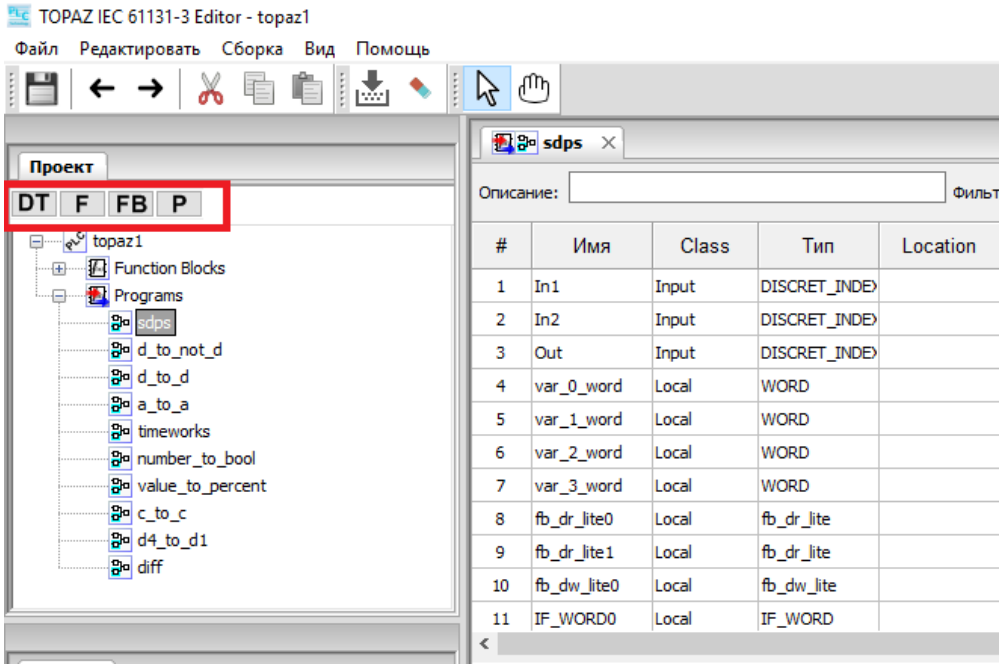
Скрытые элементы интерфейса можно вернуть через контекстное меню «ВИД», там же осуществляется сброс настроек на дефолтные (Сброс перспективы).



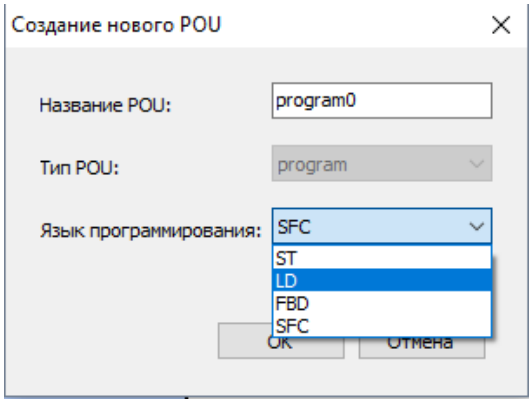
3.2 Работа в Дизайнере

Для создания новой или редактирования имеющейся пользовательской программы на языках стандарта МЭК 61131 необходимо встать на выбранный проект и открыть Дизайнер открывается нажатием соответствующей кнопки  на панели инструментов.

Чтобы создать программу (функцию, функциональный блок) нужно нажать на соответствующую кнопку в области проекта:



В появившемся диалогов окне выбрать интересующий нас язык программирования и тип POU(тип данных, функция, функциональный блок, программа).



Создание программы стоит начинать с объявления переменных в таблице.

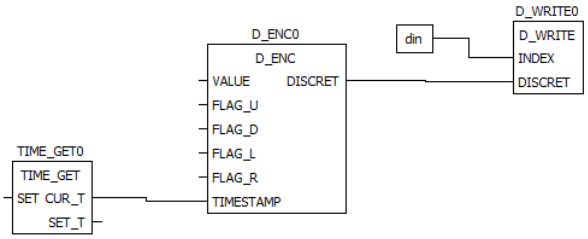
В случае, если эта переменная читается или записывается в базу DAS(или берется константой), необходимо указать тип данных указать ****_INDEX (где *** - тип сигнала: DISCRETE, ANALOG или COUNTER), Класс Input, отметить чекбоксом Парфайл, указать обязательность\необязательность задания данной переменной, а так же указать Направление. Если переменная И читается, И пишется в базу DAS, то необходимо указать Чтение.

#	Имя	Класс	Тип	Значение при инициализации	Комментарии	Парфайл	Обязательный	Направление
1	din	Input	DISCRET_INDEX			☑	☑	Чтение
2	D_READ0	Local	D_READ					
3	D_DEC0	Local	D_DEC					
4	D_ENC0	Local	D_ENC					
5	D_WRITE0	Local	D_WRITE					

Считываемый сигнал из базы DAS перед использованием должен быть декодирован с помощью соответствующего его типу блока DEC.

Соответственно перед записью сигнал должен быть кодирован с помощью блока ENC.

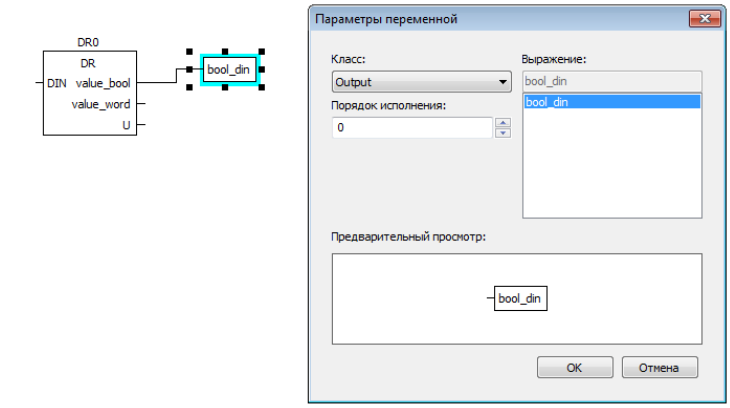
Так же необходимо при записи проставить метку времени, так как в случае ее отсутствия может быть некорректно интерпретирован со стороны контроллера. Метка времени ставится с помощью блока TIME_GET.

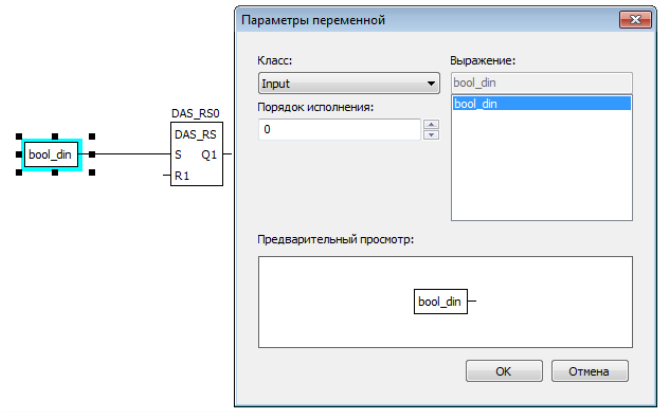


Стандарт МЭК 61131 позволяет создавать свои собственные блоки для часто повторяющихся операций и использовать их внутри пользовательских программ.

Это позволяет делать программы более читабельными и облегчает понимание их структуры.

Для облегчения понимания структуры так же можно использовать локальные переменные. Для этого в области переменных необходимо создать переменную класса local, указав нужный тип данных. В месте программы, где необходимо «забрать» значение указывается класс «Output», где вставить – класс «Input». В данном случае класс меняется вызовом окна настройки переменной по двойному клику мыши.

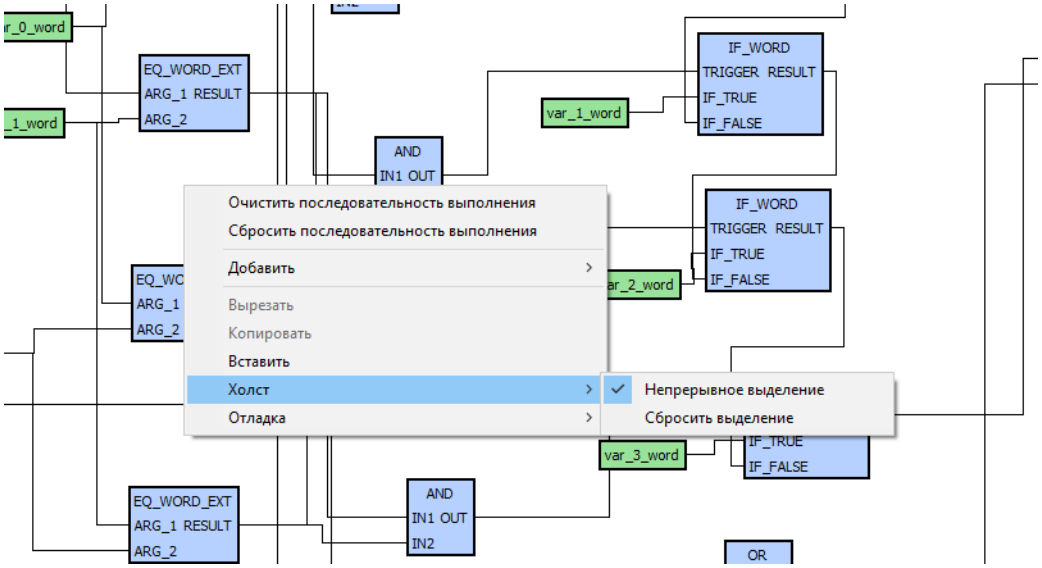




3.3 Отладочные механизмы в Дизайнере

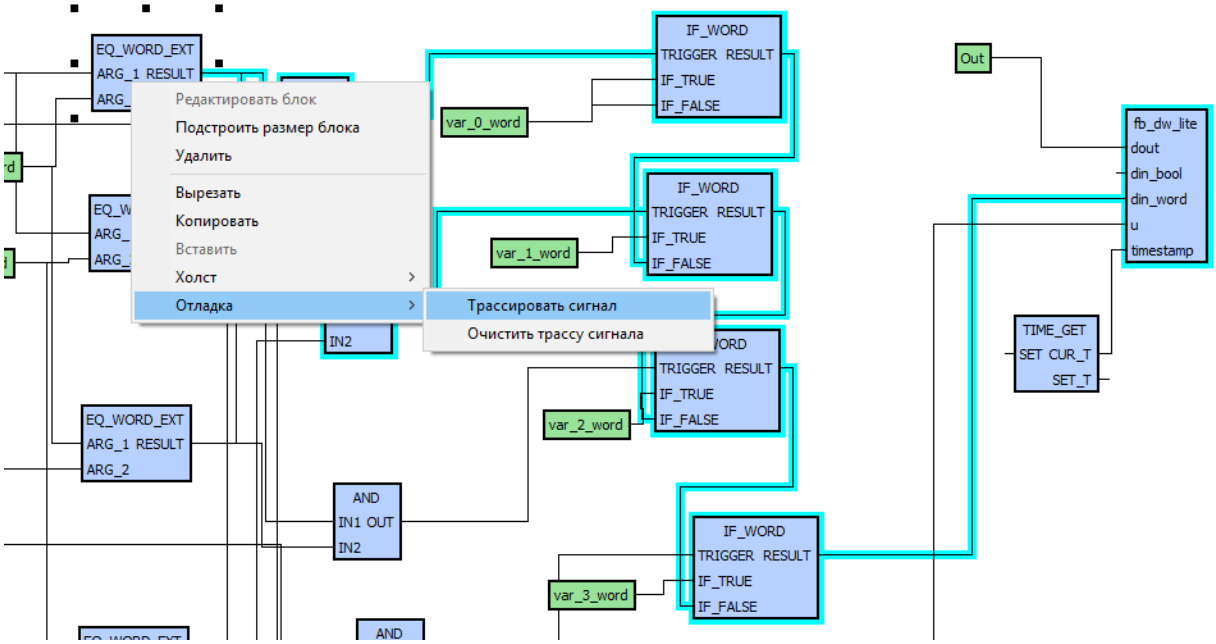
При наведении на линию связи FDB выполняется ее подсветка голубым цветом.

Есть 2 режима выделения: при наведении и непрерывное. Выбор осуществляется из контекстного меню по правому клику мыши.



Сброс непрерывного выделения осуществляется либо через контекстное меню, либо по двойному клику по пустому полю.

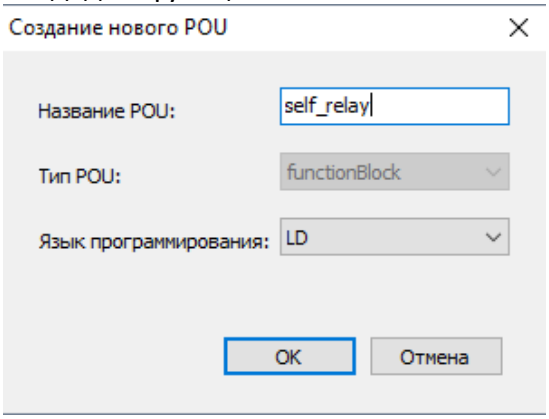
Для трассировки сигналов необходимо выделить блок и по правому клику мыши из контекстному меню выбрать Отладка-Трассировать сигнал. Сигнал протрассируется через все блоки до выходного. Сброс осуществляется так же через контекстное меню или по двойному левому клику мыши по пустому полю.



3.4 Использование LD при создании блоков

Рассмотрим как использовать блоки, созданные на языке LD на примере блока «самоводхвата реле».

Создадим функциональный блок с именем self_relay и языком программирования LD.



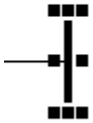
В верхней области увидим дополнительную панель инструментов LD:

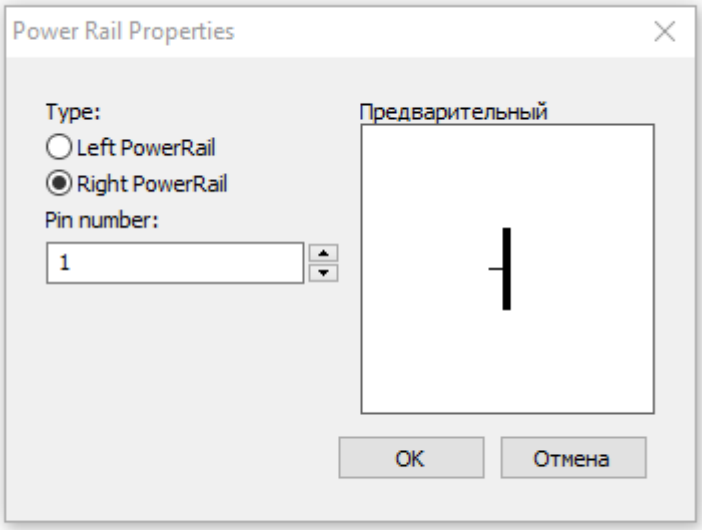
Назначение кнопок-пиктограмм

1. Выделение
2. Перемещение рабочей области
3. Добавить силовую линию
4. Добавить Реле
5. Добавить контакт реле
6. Добавить переменную
7. Добавить функциональный блок
8. Добавить коннектор

Добавим три переменных: in, out и reset типа BOOL.

Разработку алгоритма следует начинать с добавления силовых линий. Количество присоединений и левое\правое расположение меняются из контекстного меню

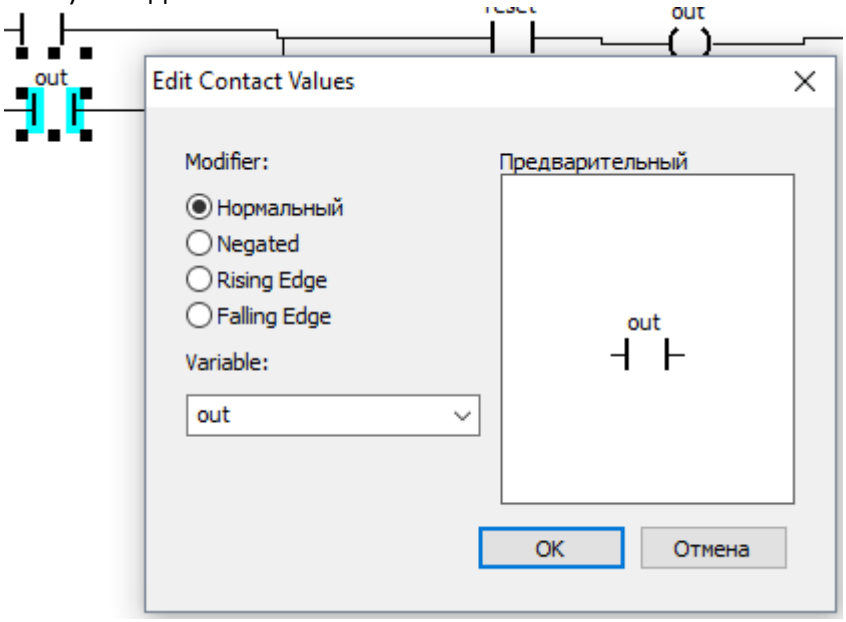




Добавлять переменные в рабочую область можно либо перетягиванием из таблицы переменных, либо по клику на панель инструментов LD.

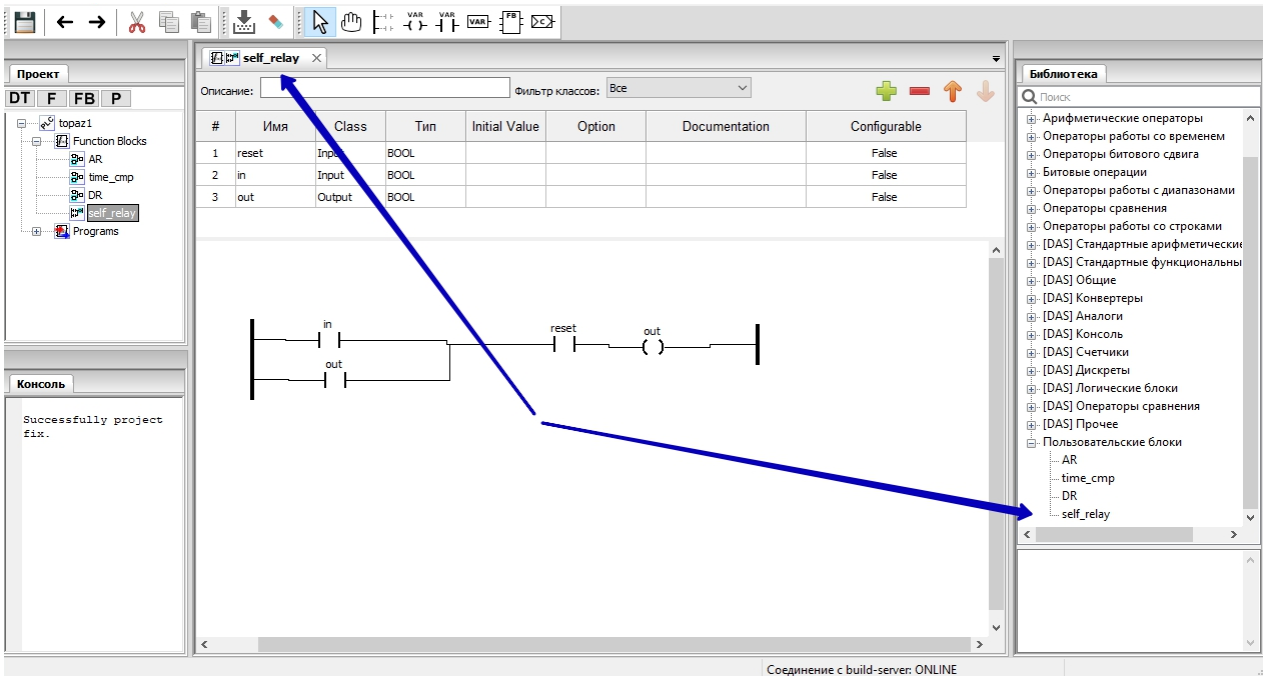
Далее необходимо произвести настройку контактов и реле. По двойному клику левой кнопкой мыши открывается настроечное окно, в котором можно выбрать характеристики для контакта:

1) Нормально открытый
2) Нормально закрытый
3) Фронт
4) Спад



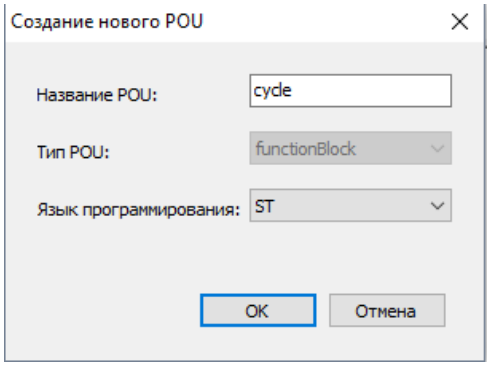
После добавления всех реле и сухих контактов – соединяем перетягиванием.

После составления алгоритма – он появится в пользовательских блоках и может быть использован в других программах.



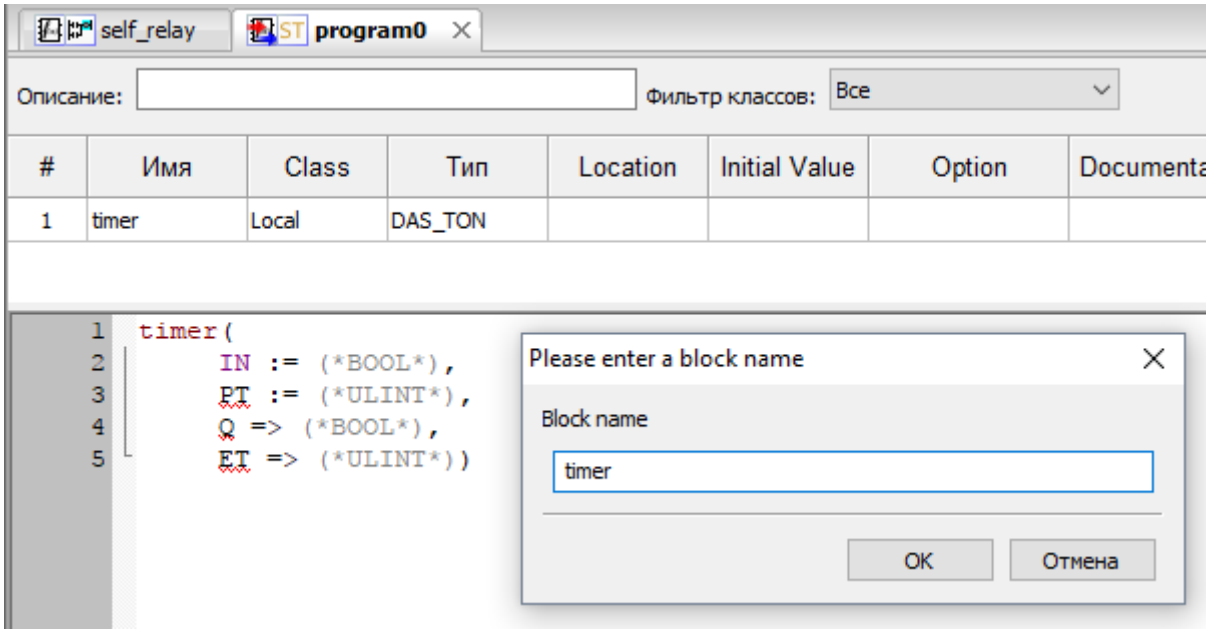
3.5 Использование ST при создании блоков и создание алгоритмов с помощью ST.

Для создания функционального блока, функции или программы на ST необходимо создать необходимый POU и выбрать язык программирования ST



Все переменные так же объявляются в таблице переменных.

Если планируется использование блоков из библиотеки – их объявлять необязательно! После перетягивания из библиотеки в рабочую область система сама предложит добавить имя блоку и сама укажет его тип в таблице переменных.



Следует помнить, что при чтении\записи переменных из\в базу DAS – необходимо выпонять декодирование\кодирование сигналов с помощью блоков (*_READ, *_DEC, *_WRITE, *_ENC).

В случае же использования блоков на ST в качестве алгоритмических вставок внутрь других программ – данные блоки использовать не нужно.

При использовании блоков из библиотеки нет необходимости изучать их внутреннюю структуру.

Наиболее часто применяемые конструкции на ST – инструкции.

Инструкция IF

Используя инструкцию IF, можно проверить условие, и в зависимости от этого условия выполнить какие-либо действия.

Синтаксис:

IF <Условие1> THEN
<IF_Инструкции>
{ELSIF <Условие2> THEN
<ELSIF_Инструкции1>
.ELSIF <Условие n> THEN
<ELSIF_Инструкции n-1>
ELSE
<ELSE_Инструкции>}
END_IF;

Часть конструкции фигурных скобках не обязательна.

Если <Условие1> возвращает истину, тогда <IF_Инструкции> выполняется.

В противном случае будут выполняться остальные логические выражения одно за другим, пока одно из них не возвратит истину. Тогда выполняются инструкции, стоящие

после этого логического выражения до следующего ELSIF или ELSE.

Если все логические выражения ложны, то выполняются инструкции, стоящие после ELSE.

Пример:

IF temp < 12
THEN heating_on: = TRUE;
ELSE heating_on: = FALSE;
END_IF

В этом примере нагревание (heating) включается, когда температура опустится ниже 12 . градусов, иначе оно останется выключенным.

Инструкция CASE

С помощью инструкции CASE можно нескольким различным значениям целочисленной переменной сопоставить различные инструкции.

Синтаксис:

CASE <Var1> OF
<Value1>: <Инструкция 1>
<Value2>: <Инструкция 2>
<Value3, Value4, Value5>: <Инструкция 3>
<Value6 .. Value10>: <Инструкция 4>
…
<Value n>: <Инструкция n>
ELSE <ELSE Инструкция>
END_CASE;

Инструкция CASE выполняется согласно следующим правилам:

Если переменная <Var1> имеет значение <Value i>, то выполняется инструкция <Инструкция i>

Если <Var1> не принимает ни одного из указанных значений, то выполняется <ELSE Инструкция>.

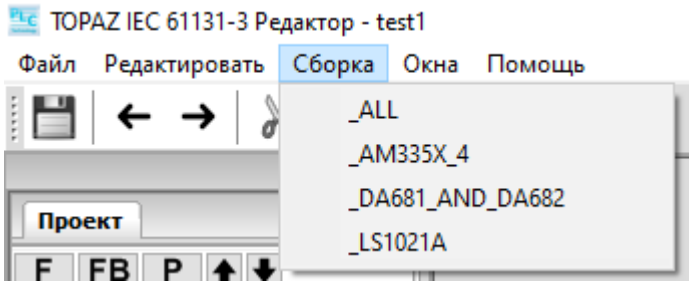
Чтобы одна и та же инструкция выполнялась при различных значениях переменной <Var1>, необходимо перечислить эти значения через запятую.

Чтобы одна и та же инструкция выполнялась для целого диапазона значений, необходимо указать начальное и конечное значения, разделенные двумя точками.

3.6 Компиляция проекта

Перед компиляцией необходимо выполнить «Очистку проекта», кликнув на пиктограмму ластика на панели инструментов 

Компиляция осуществляется через выбор соответствующей платформы в меню «Сборка».



Нажатие кнопки  в панели инструментов приведет к сборке для всех доступных платформ, что значительно увеличит время компиляции.

Если проект собран корректно, то в консоль будет выведена информация об успешном завершении компиляции.

Лог | Консоль

	Дата	Время	Уровень	
1	2020-10-01	10:39:34	info	начало сборки
2	2020-10-01	10:39:34	success	очистка проекта
3	2020-10-01	10:39:34	success	очистка артефактов сборки
4	2020-10-01	10:39:43	success	трансляция в ST-code
5	2020-10-01	10:39:45	success	трансляция в C-code
6	2020-10-01	10:39:46	success	создание проекта сборки [_AM335X_4]
7	2020-10-01	10:40:23	success	сборка проекта [_AM335X_4]
8	2020-10-01	10:40:26	info	окончание сборки
9	2020-10-01	10:40:26	success	сборка произведена без ошибок

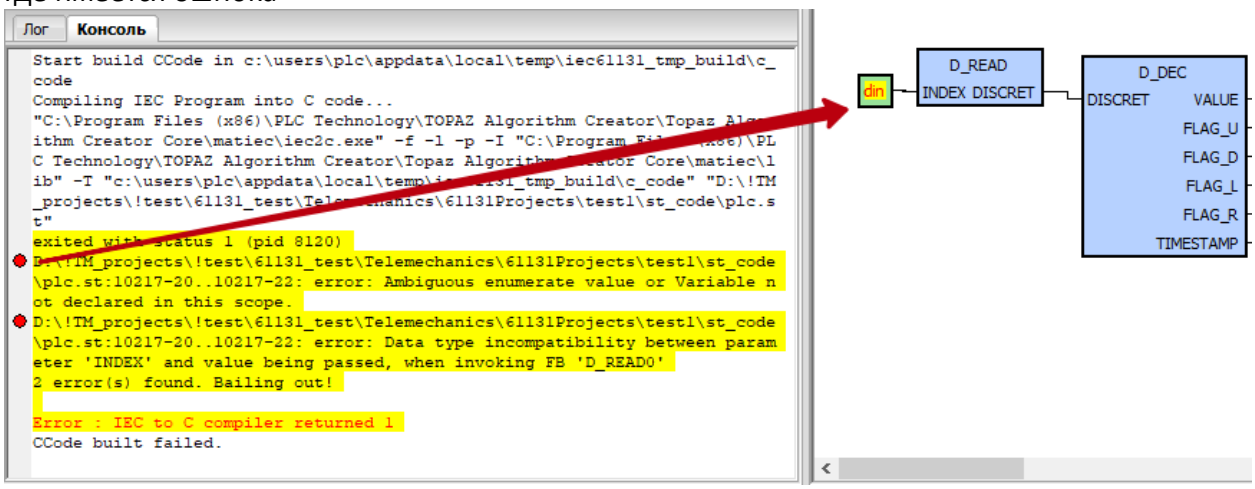
В случае наличия ошибок в консоли появится следующая запись:

Лог | Консоль

	Дата	Время	Уровень	
1	2020-10-01	10:42:43	info	начало сборки
2	2020-10-01	10:42:43	success	очистка проекта
3	2020-10-01	10:42:43	success	очистка артефактов сборки
4	2020-10-01	10:42:52	success	трансляция в ST-code
5	2020-10-01	10:42:53	fail	трансляция в C-code
6	2020-10-01	10:42:53	info	окончание сборки
7	2020-10-01	10:42:53	error	сборка произведена с ошибками

Для получения более подробной информации необходимо перейти на вкладку

«Консоль» и кликнуть на красный индикатор – то покажется конкретное место в программе, где имеется ошибка

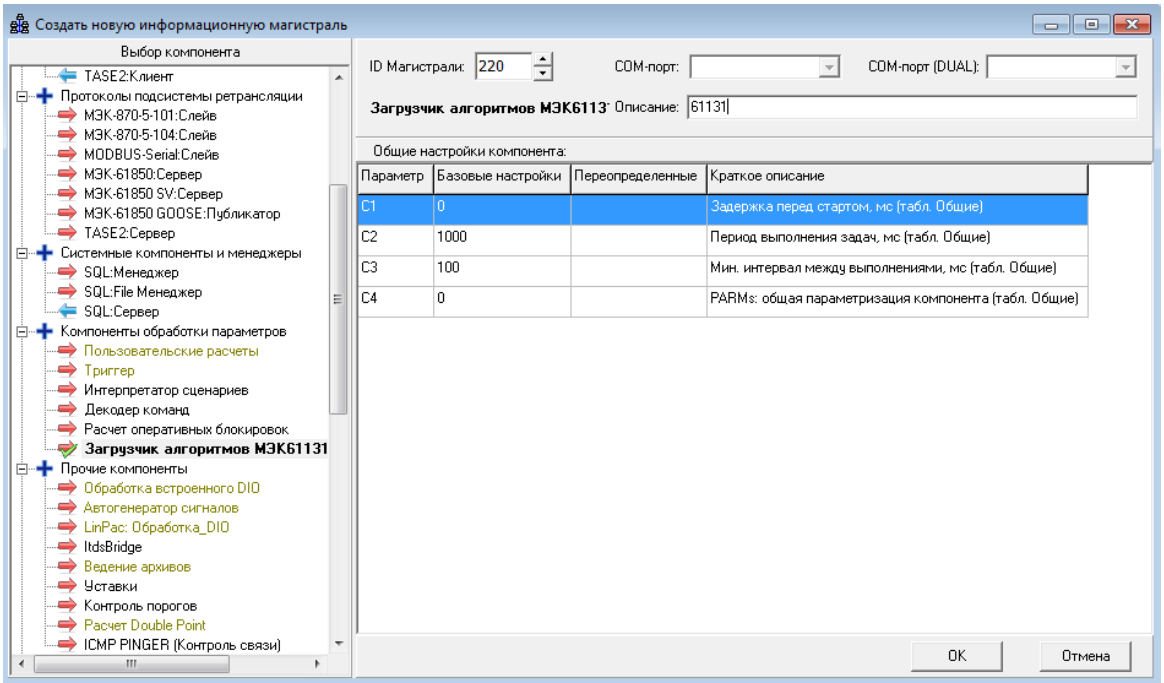


Наиболее частые ошибки:

1) Переменная отсутствует в таблице переменных
2) Несоответствие типа данных
3) Несоответствие формата данных «Значение при инициализации» данной переменной
4) Для выходных сигналов отсутствует привязка к входу TIMESTAMP
5) В проекте присутствуют блоки, не имеющие ни выходных, ни выходных сигналов
6) В названиях блоков присутствуют кириллические символы.

3.7 Конфигурирование проекта в TMBuilder

Для использования пользовательских программ, созданных в Конфигураторе IEC61131, в проекте создается магистраль «Загрузчик алгоритмов МЭК61131»



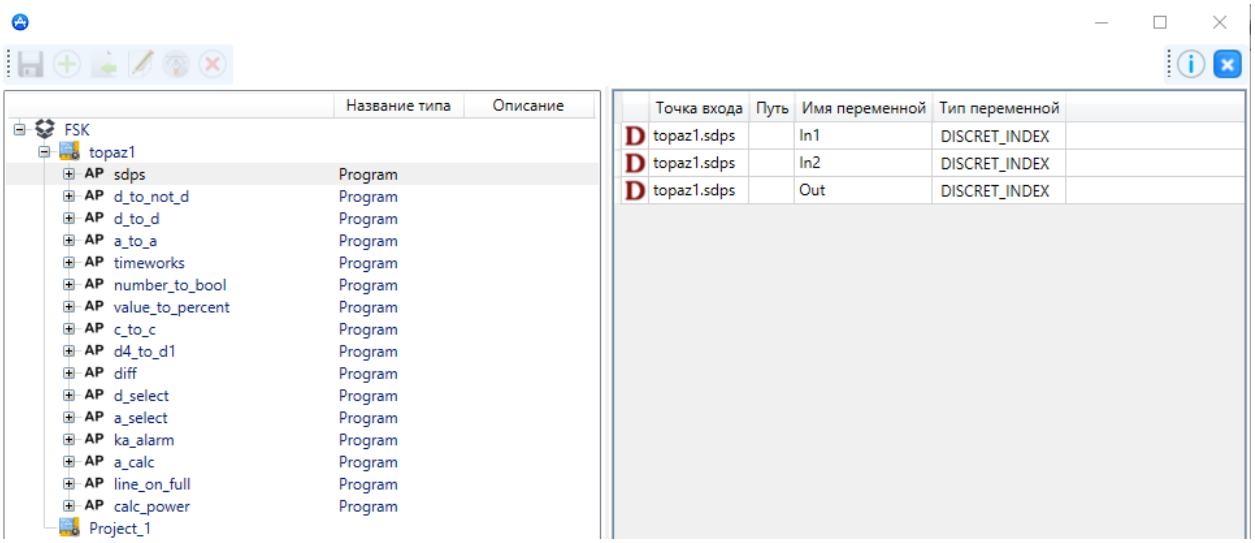
Магистраль имеет следующие настройки:

C1: Задержка перед стартом
C2: Период выполнения задач, мс (Tmax)
C3: Мин интервал между выполнениями, мс (Tmin)
C4: PARMS (15йбит=16384 – вывод отладочной информации в консоль)

Если
dt > Tmax, используем задержку Tmin;
Если dt < Tmax И (Tmax - dt) < Tmin, используем задержку Tmin ;
Если dt < Tmax И (Tmax - dt) > Tmin, используем задержку (Tmax - dt),
Где dT – время выполнения программы

Важно учитывать, что параметры C2 и C3 нужно подбирать таким образом, чтобы соотношение быстродействие/нагрузка на процессор соответствовало решаемой задаче. Частый вызов программы приводит к повышению нагрузки.

Входные и выходные переменные интересующей нас программы показываются в конфигураторе

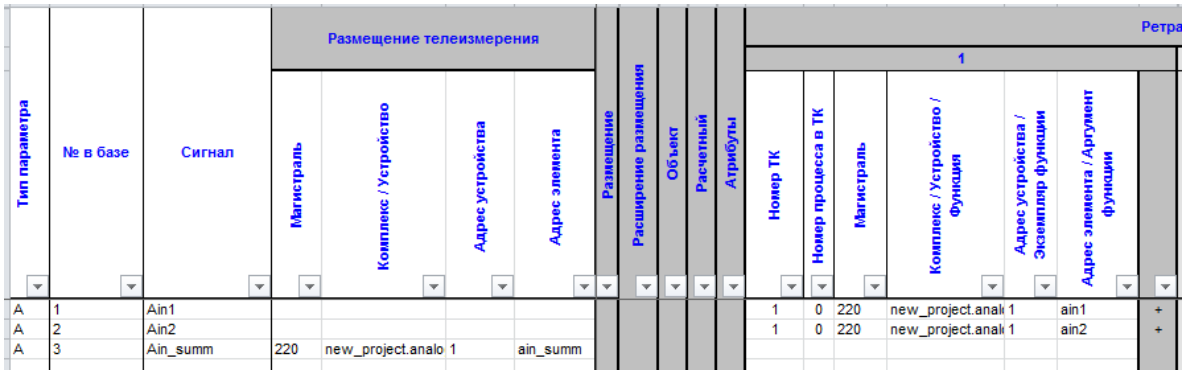


В Парфайле выходные переменные мы можем записывать в PLACE, а входные – в LOADTO.

В графе «Комплекс/Устройство» указывается пользовательская программа в следующем формате: «Project_name.Program_name», что соответствует записи «Точка входа» в **Конфигураторе IEC61131**.

В «Адресе устройства» = экземпляр данной программы.

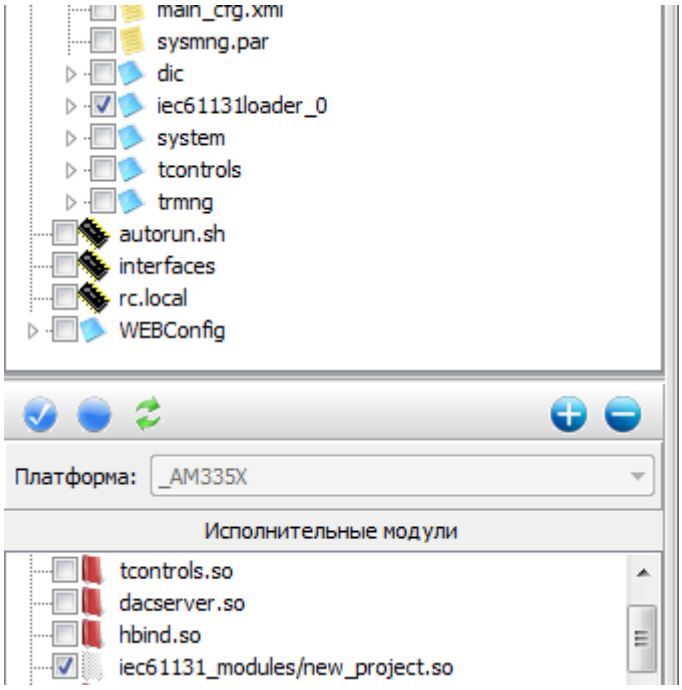
В «Адресе элемента» = имя переменной.



Тип параметра	№ в базе	Сигнал	Размещение телеизмерения				Размещение	Расширение размещения	Объект	Расчетный	Атрибуты	Ретра						
			Магистраль	Комплекс / Устройство	Адрес устройства	Адрес элемента						1						
												Номер ТК	Номер процесса в ТК	Магистраль	Комплекс / Устройство / Функция	Адрес устройства / Экземпляр функции	Адрес элемента / Аргумент функции	
A	1	Ain1										1	0	220	new_project.anal	1	ain1	+
A	2	Ain2										1	0	220	new_project.anal	1	ain2	+
A	3	Ain_summ	220	new_project.analo	1	ain_summ												

3.8 Загрузка проекта

В **DAS Загрузчике** все данные, касающиеся программ на 61131, представлены в виде исполняемого модуля и конфигурационных файлов (*.ini).

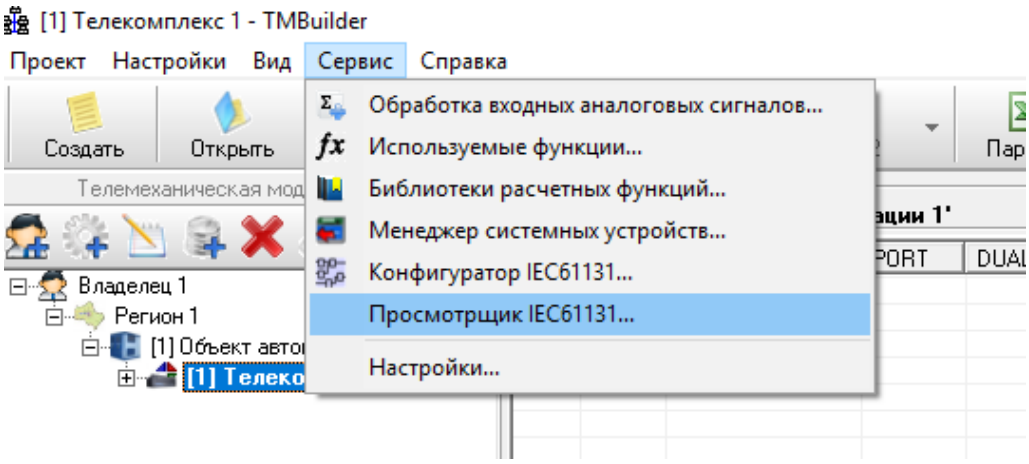


При этом файл ***iec61131_modules/имя_проекта.so*** представляет собой бинарный файл с программой, а в разделе CFG представлена информация о магистрали и входных-выходных переменных.

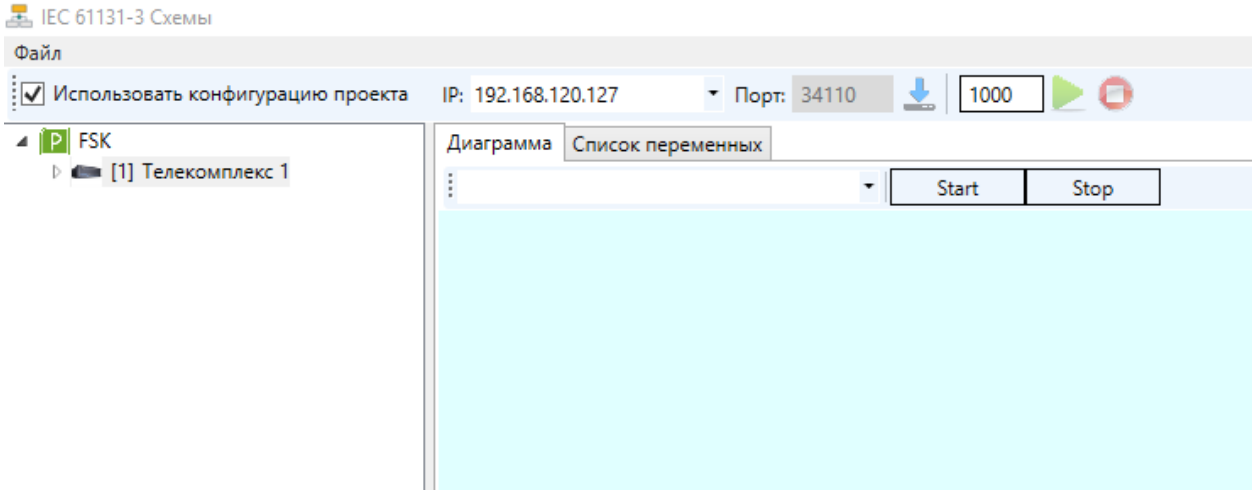
Исполняемый модуль вызывается iec-controls.

4. Отладка программы с помощью Просмотрщика.

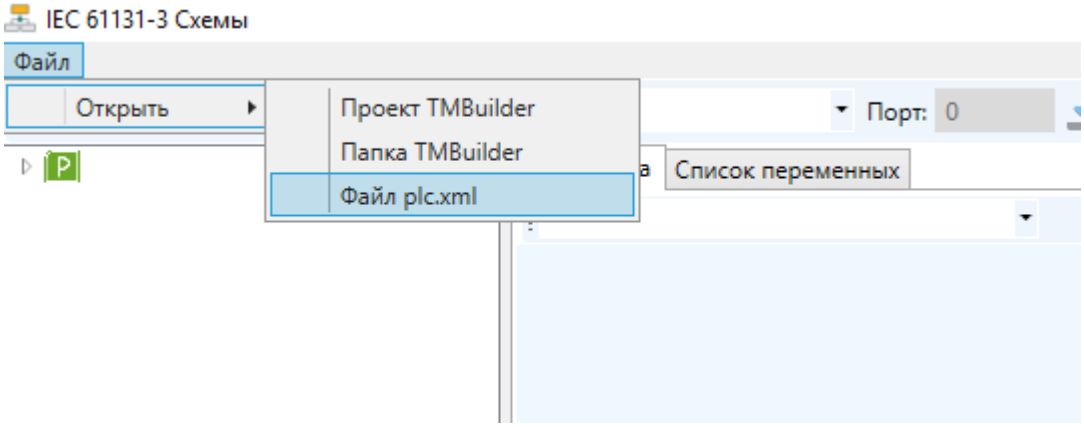
После успешной заливки конфигурации и модулей в контроллер и успешного запуска iec-controls для отладки можно использовать Просмотрщиком 61131.
Просмотрщик вызывается из меню Сервис TMBuilder’а



В Просмотрщик 61131 можно «Использовать конфигурацию проекта»

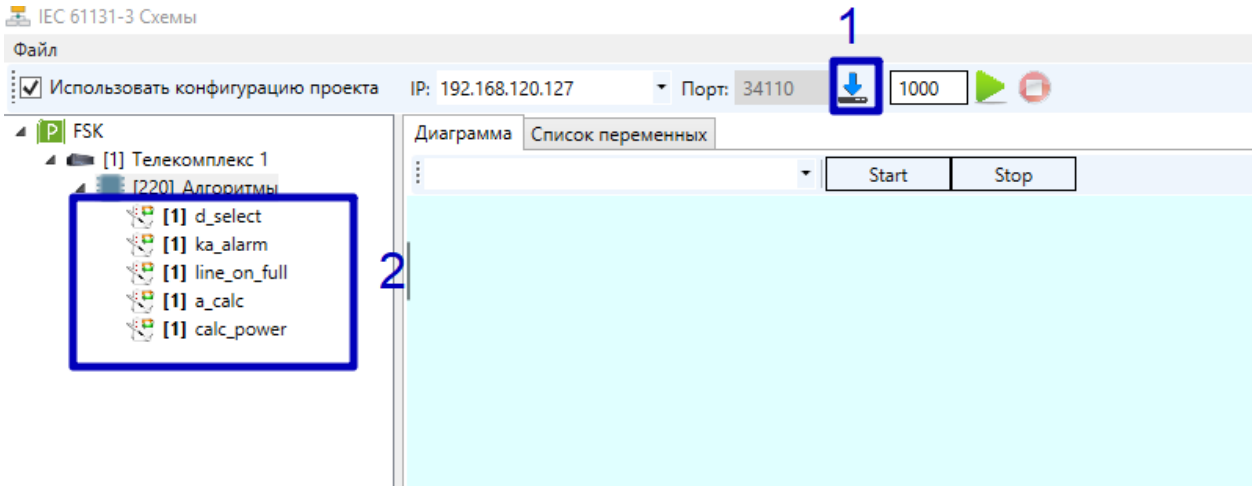


Чтобы загрузить свою конфигурацию, указываем путь до файла Имя_проекта_TMBuilder\Telemechanics\61131Projects\имя_проекта_61131\plc.xml

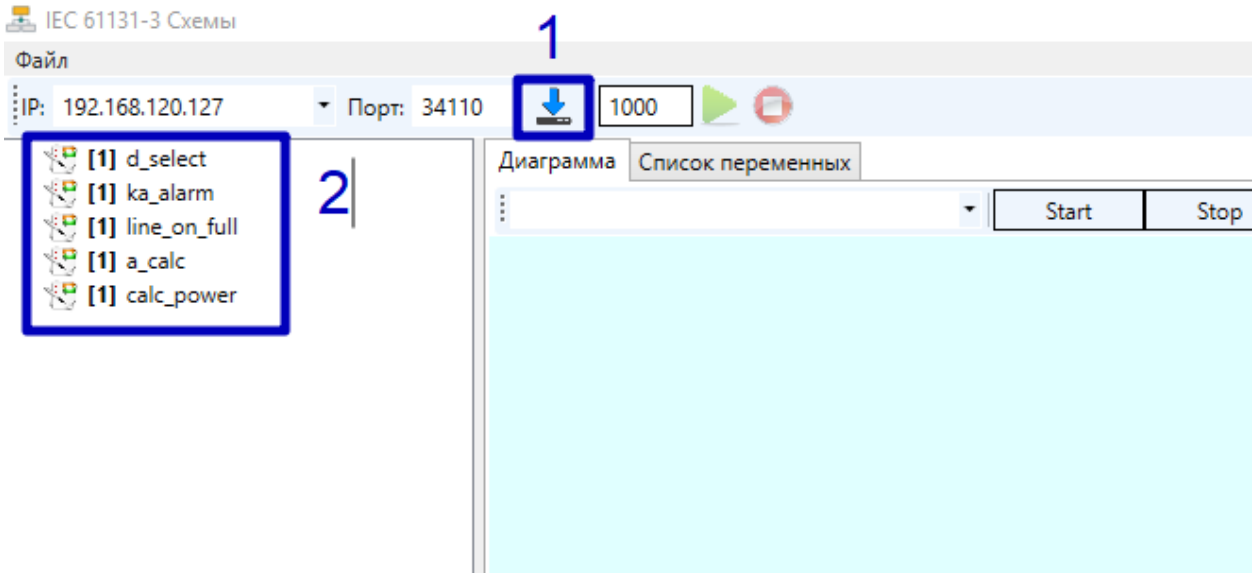


В поле IP вбиваем IP-адрес нашего контроллера. Порт по умолчанию 34110. Время обновления указывается в миллисекундах и подбирается самостоятельно. Рекомендуется ставить 100.

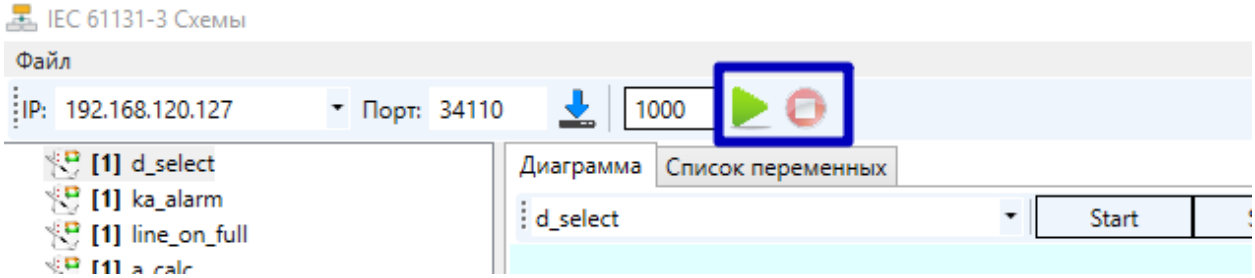
При использовании конфигурации проекта необходимо встать на узел с [номер магистрали]имя магистрали и нажать на пиктограмму «Скачать». После этого появится список загруженных программ(2). В квадратных скобках указан номер экземпляра.



При загрузке своей конфигурации поступаем аналогично, с тем лишь различием, что в структуре не будет присутствовать телекомплекс и магистраль



Выбираем нужный нам экземпляр программы и нажимаем пиктограмму зеленой стрелки.



Просмотрщик будет обновлять данные на входах и выходах блоков с указанной периодичностью.

5. Функциональные блоки Конфигуратора IEC61131

Все стандартные функциональные блоки разбиты по разделам.
Так же условно их можно разделить на 2 большие группы: блоки DAS и остальные.

В блоках DAS представлены блоки, оптимизированные для работы с оборудованием TOPAZ.

1) Операторы конвертирования типов

В данном разделе представлены функциональные блоки для конвертирования одного типа представления данных в другой. Набор охватывает все типы представления данных, используемых в Конфигураторе.

2) Численные операторы

В данном разделе представлены блоки, необходимые для вычисления значений стандартных функций, таких как: корень(SQRT), логарифм(LOG), экспонента(EXP), синус(SIN) и тд.

Все названия - в соответствии со стандартными общепринятыми сокращениями на латинице.

3) Арифметические операторы

В данном разделе представлены блоки, необходимые для выполнения арифметических вычислений.

Все названия - в соответствии со стандартными общепринятыми сокращениями на латинице.

4) Операторы работы со временем

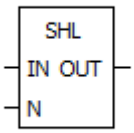
В данном разделе представлены блоки, необходимые для выполнения арифметических вычислений со временем(формат данных TIME)

Все названия - в соответствии со стандартными общепринятыми сокращениями на латинице.

5) Операторы битового сдвига

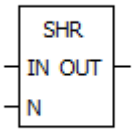
В данном разделе представлены блоки, необходимые для выполнения операций битового сдвига, заполнения или замещения

SHL



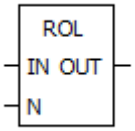
Побитный сдвиг операнда **in** влево на **n** бит с дополнением нулями справа.
Входные переменные и результат должны быть типа **BYTE**, **WORD** или **DWORD**.

SHR



Побитный сдвиг операнда **in** вправо на **n** бит с дополнением нулями слева.
Входные переменные и результат должны быть типа **BYTE**, **WORD** или **DWORD**.

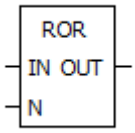
ROL



Циклический сдвиг операнда **in** влево на **n** бит, младшие биты последовательно заполняются старшими.

Входные переменные и результат должны быть типа **BYTE**, **WORD** или **DWORD**.

ROR



Циклический сдвиг операнда **in** вправо на **n** бит, младшие биты последовательно заменяют старшие.

Входные переменные и результат должны быть типа **BYTE**, **WORD** или **DWORD**.

6) Битовые операции

В данном разделе представлены блоки, необходимые для выполнения логических операций на основе булевой логики

Все названия - в соответствии со стандартными общепринятыми сокращениями на латинице.

7) Операторы работы с диапазонами

8) Операторы сравнения

В данном разделе представлены блоки, необходимые для выполнения операций сравнения значений различных типов данных.

Все названия - в соответствии со стандартными общепринятыми сокращениями на латинице.

9) Операторы работы со строками

10) [DAS] Стандартные арифметические блоки

В данном разделе представлены блоки, необходимые для выполнения арифметических вычислений.

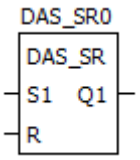
Все названия - в соответствии со стандартными общепринятыми сокращениями на латинице.

11) [DAS] Стандартные функциональные блоки

В данном разделе представлены стандартные блоки языка FBD, оптимизированные для работы с контроллерами TOPAZ.

Все названия - в соответствии со стандартными общепринятыми сокращениями на латинице.

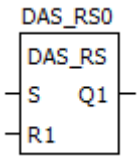
DAS SR



Переключатель с доминантой включения:
Q1 = SR (SET1, RESET) означает:
Q1 = (NOT RESET AND Q1) OR SET1
Входные переменные **SET1** и **RESET** - как и выходная переменная **Q1** типа **BOOL**.
Доминанта **S** означает, что при единовременном наличии сигналов **TRUE** на входах **S1** и **R** на выходе **Q1** будет сигнал **TRUE**.

```
SRInst(SET1:= VarBOOL1 , RESET:=VarBOOL2 );
VarBOOL3 := SRInst.Q1 ;
```

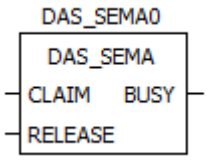
DAS RS



RS Переключатель с доминантой выключения:
Q1 = RS (SET, RESET1) означает:
Q1 = NOT RESET1 AND (Q1 OR SET)
Входные переменные **SET** и **RESET1** - как и выходная переменная **Q1** типа **BOOL**.

```
RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
VarBOOL3 := RSInst.Q1 ;
```

DAS_SEMA



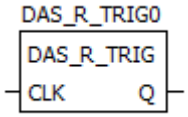
Программный семафор.
BUSY = SEMA(CLAIM, RELEASE) означает:

```
BUSY := X;
IF CLAIM THEN X:=TRUE;
ELSE IF RELEASE THEN BUSY := FALSE;
 X:= FALSE;
END_IF
```

X - это внутренняя **BOOL** переменная, изначально имеющая значение **FALSE**.
Входные переменные **CLAIM** и **RELEASE** - как и выходная переменная **BUSY** типа **BOOL**. (**CLAIM** – запрос захвата, **RELEASE** - освобождение).
Семафор предназначен для организации асинхронного доступа к одному

аппаратному ресурсу. Если при вызове семафора с **CLAIM** = **TRUE** возвращаемое значение **BUSY** = **FALSE**, то ресурс свободен (запрашивается впервые или уже освобожден вызовом **RELEASE** = **TRUE**). Возвращаемое значение **BUSY** = **FALSE**, это означает, что ресурс занят

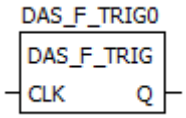
DAS_R_TRIG



Функциональный блок **DAS_R_TRIG** генерирует импульс по переднему фронту входного сигнала.

Выход **Q** равен **FALSE** до тех пор, пока вход **CLK** равен **FALSE**. Как только **CLK** получает значение **TRUE**, **Q** устанавливается в **TRUE**. При следующем вызове функционального блока выход сбрасывается в **FALSE**. Таким образом, блок выдает единичный импульс при каждом переходе **CLK** из **FALSE** в **TRUE**.

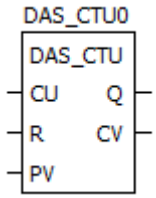
DAS_F_TRIG



Функциональный блок **DAS_F_TRIG** генерирует импульс по заднему фронту входного сигнала.

Выход **Q** равен **FALSE** до тех пор, пока вход **CLK** равен **TRUE**. Как только **CLK** получает значение **FALSE**, **Q** устанавливается в **TRUE**. При следующем вызове функционального блока выход сбрасывается в **FALSE**. Таким образом, блок выдает единичный импульс при каждом переходе **CLK** из **TRUE** в **FALSE**.

DAS_CTU



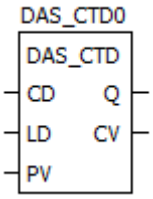
Функциональный блок ‘инкрементный счетчик’.

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

По каждому фронту на входе **CU** (переход из **FALSE** в **TRUE**) выход **CV** увеличивается на 1. Выход **Q** устанавливается в **TRUE**, когда счетчик достигнет значения заданного **PV**.

Счетчик CV сбрасывается в 0 по входу **RESET** = **TRUE**.

DAS_CTD



Функциональный блок ‘декрементный счетчик’.

Входы **CD**, **LOAD** и выход **Q** типа **BOOL**, вход **PV** и выход **CV** типа **WORD**.

По каждому фронту на входе **CD** (переход из **FALSE** в **TRUE**) выход **CV** уменьшается на 1. Когда счетчик достигнет 0, счет останавливается, выход **Q** переключается в **TRUE**.

Счетчик **CV** загружается начальным значением, равным **PV** по входу **LOAD** = **TRUE**.

DAS_CTUD

****

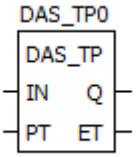
Функциональный блок ‘инкрементный / декрементный счетчик’.

Входы **CU**, **CD**, **RESET**, **LOAD** и выходы **QU** и **QD** типа **BOOL**, **PV** и **CV** типа **WORD**.

По входу **RESET** счетчик **CV** сбрасывается в **0**, по входу LOAD загружается значением PV. По фронту на входе **CU** счетчик увеличивается на 1.

По фронту на входе **CD** счетчик уменьшается на **1** (до **0**). **QU** устанавливается в **TRUE**, когда **CV** больше или равен **PV**. **QD** устанавливается в **TRUE**, когда **CV** равен **0**.

DAS_TP



Функциональный блок ‘таймер’.

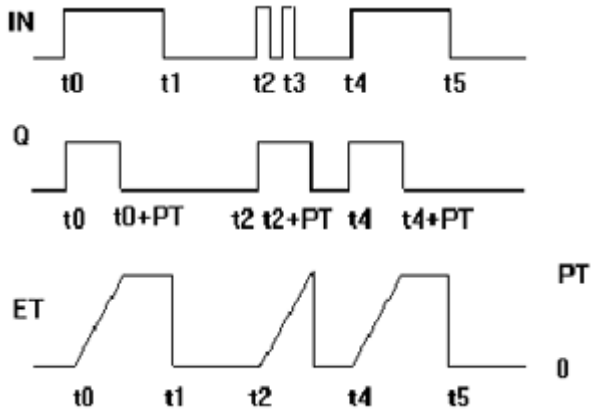
TP(IN, PT, Q, ET)

Входы **IN** и **PT** типов **BOOL** и **TIME** соответственно.

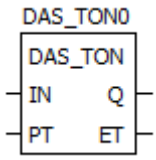
Выходы **Q** и **ET** аналогично типов **BOOL** и **TIME**.

Пока **IN** равен **FALSE**, выход **Q** = **FALSE**, выход **ET** = **0**. При переходе **IN** в **TRUE** выход **Q** устанавливается в **TRUE** и таймер начинает отсчет времени (в миллисекундах) на выходе **ET** до достижения длительности, заданной **PT**. Далее счетчик не увеличивается. Таким образом, выход **Q** генерирует им- пульс длительностью **PT** по фронту входа **IN**.

Временная диаграмма работы **TP**:



DAS_TON



Функциональный блок ‘таймер с задержкой включения’.

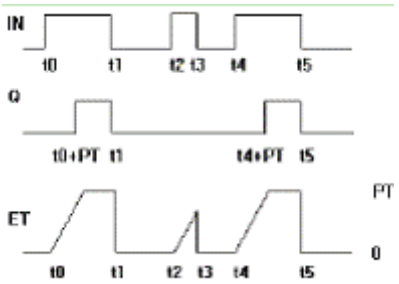
TON(IN, PT, Q, ET)

Входы **IN** и **PT** типов **BOOL** и **TIME** соответственно.

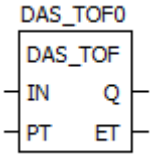
Выходы **Q** и **ET** аналогично типов **BOOL** и **TIME**.

Пока **IN** равен **FALSE**, выход **Q** = **FALSE**, выход **ET** = 0. Как только **IN** становится **TRUE**, начинается отсчет времени (в миллисекундах) на выходе ET до значения, равного **PT**. Далее счетчик не увеличивается. **Q** равен **TRUE**, когда **IN** равен **TRUE** и **ET** равен **PT**, иначе **FALSE**. Таким образом, выход **Q** устанавливается с задержкой **PT** от фронта входа **IN**.

Временная диаграмма работы TON:



DAS_TOF



Функциональный блок ‘таймер с задержкой выключения’.

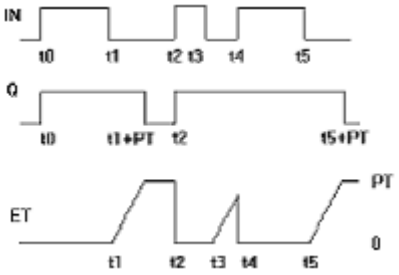
TOF(IN, PT, Q, ET)

Входы **IN** и **PT** типов **BOOL** и **TIME** соответственно.

Выходы **Q** и **ET** аналогично типов **BOOL** и **TIME**.

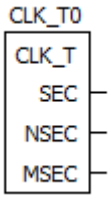
Если **IN** равен **TRUE**, то выход **Q** = **TRUE** и выход **ET** = **0**. Как только **IN** переходит в **FALSE**, начинается отсчет времени (в миллисекундах) на выходе **ET**. При достижении заданной длительности отсчет останавливается. Выход **Q** равен **FALSE**, если **IN** равен **FALSE** и **ET** равен **PT**, иначе - **TRUE**. Таким образом, выход **Q** сбрасывается с задержкой **PT** от спада входа **IN**.

Временная диаграмма работы **TOF**:



12) [DAS] Общие

CLK_T

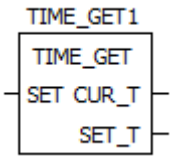


Счетчик времени между вызовами.

Выходы **SEC**, **NSEC**, **MSEC** типа **ULINT**.

Показывает время в выбранных единицах между вызовами программы. Блок необходим для отслеживания времени исполнения программы.

TIME_GET



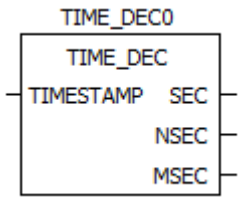
Получение временной метки.

Вход **SET** типа **BOOL**.

Выходы **CUR_T** и **SET_T** типа **TIMESPEC**.

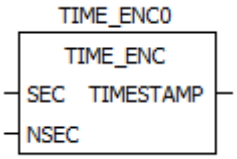
Выход **CUR_T** всегда имеет текущую временную метку(в момент обращения). Выход **SET_T** замирает на текущем значении по фронту на входе **SET**.

TIME_DEC



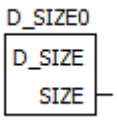
Декодирование временной метки
Вход **TIMESTAMP** типа **TIMESPEC**
Выходы **SEC**, **NSEC**, **MSEC** типа **ULINT**.

TIME_ENC



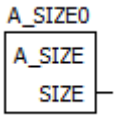
Кодирование временной метки
Входы **SEC**, **NSEC** типа **ULINT**.
Выход **TIMESTAMP** типа **TIMESPEC**

D_SIZE



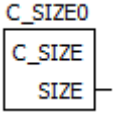
Получение размера базы дискретов.
Выход **SIZE** типа **WORD**.

A_SIZE



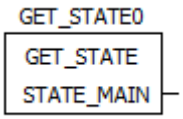
Получение размера базы аналогов.
Выход **SIZE** типа **WORD**.

C_SIZE



Получение размера базы счетчиков.
Выход **SIZE** типа **WORD**.

GET_STATE



Получение текущего состояния контроллера (MAIN|STANDBY)
Выход **STATE_MAIN** типа **INT**

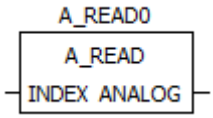
13) [DAS] Конверторы

Блоки конвертирования типа данных

14) [DAS] Аналоги

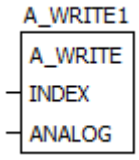
Данный раздел предназначен для работы с аналоговыми сигналами.

A_READ



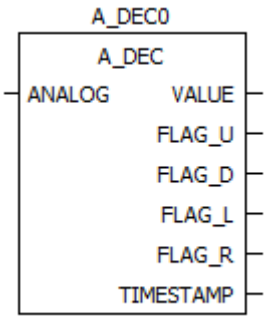
Чтение аналога из базы DAS
Вход: **INDEX** типа **ANALOG_INDEX**
Выход **ANALOG**
На вход подается непосредственно значение из базы DAS, с выхода снимается сигнал для декодирования.

A_WRITE



Запись аналога в базу DAS
Входы: **INDEX** типа **ANALOG_INDEX, ANALOG**
На вход **ANALOG** подается закодированный сигнал, на вход **INDEX** – выходная переменная для записи в базу DAS.

A_DEC



Декодирование параметров аналога.

Блок служит для декодирования параметров аналогового сигнала для дальнейшего их использования. Используется после блока **A_READ**.

Вход: **ANALOG**

Выходы:

VALUE (значение) типа **REAL**

FLAG_U (недостоверность) типа **BOOL**

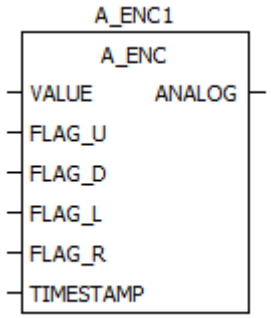
FLAG_D (динамика) типа **BOOL**

FLAG_L (блокировка) типа **BOOL**

FLAG_R (подмена) типа **BOOL**

TIMESTAMP (метка времени) типа **TIMESTAMP**

A_ENC



Кодирование параметров аналога

Блок служит для кодирования параметров аналогового сигнала перед записью в базу DAS. Используется перед блоком **A_WRITE**.

Входы

VALUE (значение) типа **REAL**

FLAG_U (недостоверность) типа **BOOL**

FLAG_D (динамика) типа **BOOL**

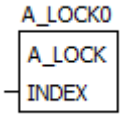
FLAG_L (блокировка) типа **BOOL**

FLAG_R (подмена) типа **BOOL**

TIMESTAMP (метка времени) типа **TIMESTAMP**

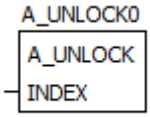
Выход: **ANALOG**

A_LOCK



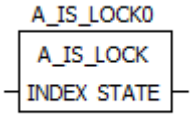
Блокирование аналога.
Вход: **INDEX** типа **ANALOG_INDEX**

A_UNLOCK



Деблокирование аналога
Вход: **INDEX** типа **ANALOG_INDEX**

A_IS_LOCK

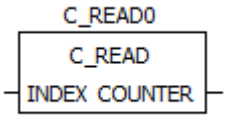


Проверка состояния блокировки аналога
Вход: **INDEX** типа **ANALOG_INDEX**
Выход: **STATE** типа **BOOL**

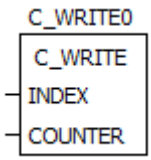
15) [DAS] Счетчики

Данный раздел предназначен для работы с счетчиками.

C_READ



Чтение счетчика из базы DAS
Вход: **INDEX** типа **COUNTER_INDEX**
Выход **COUNTER**
На вход подается непосредственно значение из базы DAS, с выхода снимается сигнал для декодирования.
C_WRITE

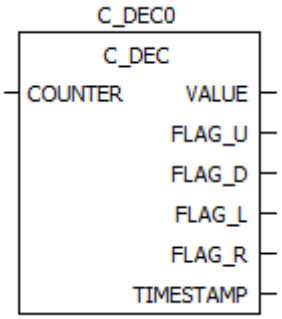


Запись счетчика в базу DAS

Входы: **INDEX** типа **COUNTER_INDEX, COUNTER**

На вход **COUNTER** подается закодированный сигнал, на вход **INDEX** – выходная переменная для записи в базу DAS.

C_DEC



Декодирование параметров счетчика.

Блок служит для декодирования параметров счетчика для дальнейшего их использования. Используется после блока **C_READ**.

Вход: **COUNTER**

Выходы:

VALUE (значение) типа **DWORD**

FLAG_U (недостоверность) типа **BOOL**

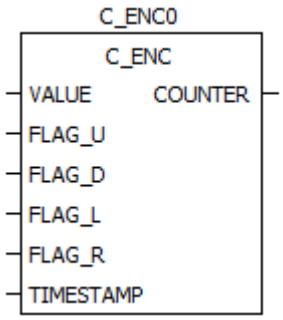
FLAG_D (динамика) типа **BOOL**

FLAG_L (блокировка) типа **BOOL**

FLAG_R (подмена) типа **BOOL**

TIMESTAMP (метка времени) типа **TIMESTAMP**

C_ENC



Кодирование параметров счетчика

Блок служит для кодирования параметров счетчика ереД записью в базу DAS. Используется перед блоком **C_WRITE**.

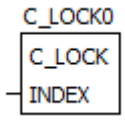
Входы

VALUE (значение) типа **DWORD**

FLAG_U (недостоверность) типа **BOOL**
FLAG_D (динамика) типа **BOOL**
FLAG_L (блокировка) типа **BOOL**
FLAG_R (подмена) типа **BOOL**
TIMESTAMP (метка времени) типа **TIMESTAMP**

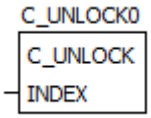
Выход: **COUNTER**

C_LOCK



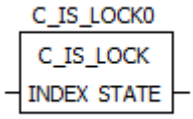
Блокирование счетчика.
Вход: **INDEX** типа **COUNTER_INDEX**

C_UNLOCK



Деблокирование счетчика
Вход: **INDEX** типа **COUNTER_INDEX**

C_IS_LOCK

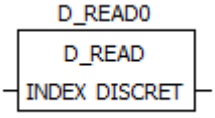


Проверка состояния блокировки счетчика
Вход: **INDEX** типа **COUNTER_INDEX**
Выход: **STATE** типа **BOOL**

16) [DAS] Дискреты

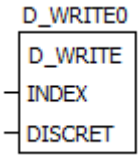
Данный раздел предназначен для работы с дискретными сигналами.

D_READ



Чтение дискрета из базы DAS
На вход подается непосредственно значение из базы DAS, с выхода снимается сигнал для декодирования.
Входы: **INDEX** типа **DISCRET_INDEX, DISCRET.**

D_WRITE

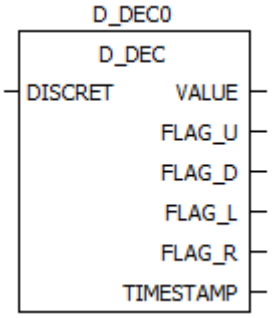


Запись дискрета в базу DAS

Входы: **INDEX** типа **DISCRET_INDEX, DISCRET**

На вход **DISCRET** подается закодированный сигнал, на вход **INDEX** – выходная переменная для записи в базу DAS.

D_DEC



Декодирование параметров дискрета.

Блок служит для декодирования параметров дискрета для дальнейшего их использования. Используется после блока **D_READ**.

Вход: **DISCRET**

Выходы:

VALUE (значение) типа **WORD**

FLAG_U (недостоверность) типа **BOOL**

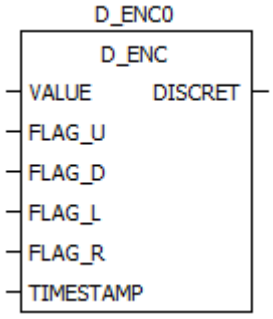
FLAG_D (динамика) типа **BOOL**

FLAG_L (блокировка) типа **BOOL**

FLAG_R (подмена) типа **BOOL**

TIMESTAMP (метка времени) типа **TIMESTAMP**

D_ENC



Кодирование параметров дискрета

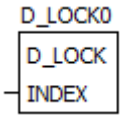
Блок служит для кодирования параметров дискрета перед записью в базу DAS.

Используется перед блоком **D_WRITE**.

Входы
VALUE (значение) типа **WORD**
FLAG_U (недостоверность) типа **BOOL**
FLAG_D (динамика) типа **BOOL**
FLAG_L (блокировка) типа **BOOL**
FLAG_R (подмена) типа **BOOL**
TIMESTAMP (метка времени) типа **TIMESTAMP**

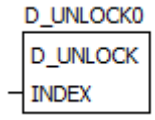
Выход: **DISCRET**

D_LOCK



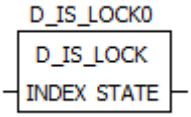
Блокирование дискрета.
Вход: **INDEX** типа **DISCRET_INDEX**

D_UNLOCK



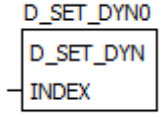
Деблокирование дискрета
Вход: **INDEX** типа **DISCRET_INDEX**

D_IS_LOCK



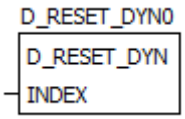
Проверка состояния блокировки дискрета
Вход: **INDEX** типа **DISCRET_INDEX**
Выход: **STATE** типа **BOOL**

D_SET_DYN



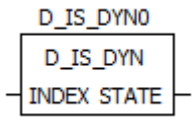
Установка бита динамики у дискрета
Вход: **INDEX** типа **DISCRET_INDEX**

D_RESET_DYN



Сброс бита динамики у дискрета
INDEX типа **DISCRET_INDEX**

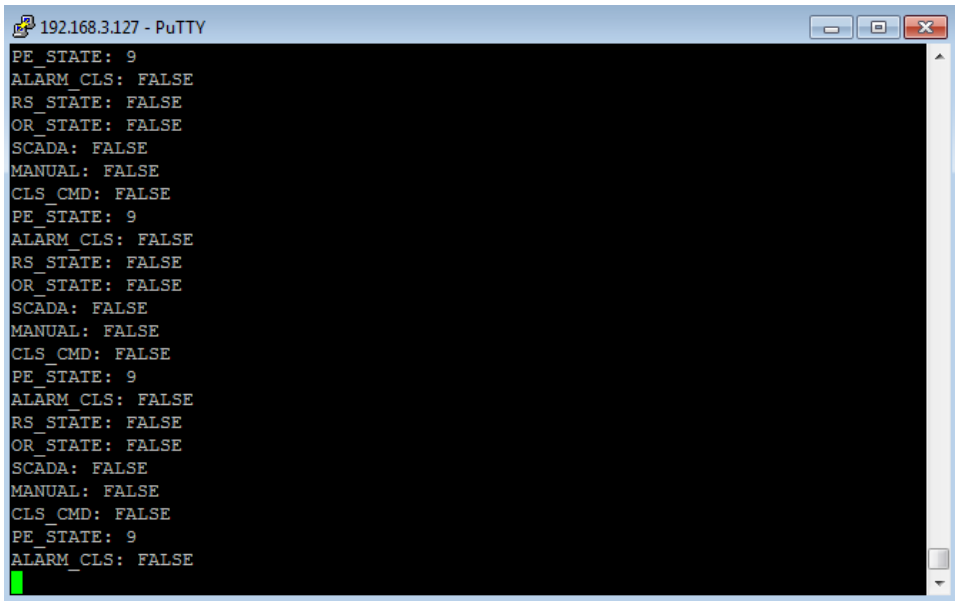
D_IS_DYN



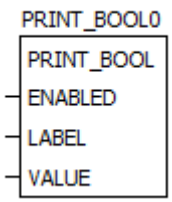
Проверка бита динамики у дискрета
Вход: **INDEX** типа **DISCRET_INDEX**
Выход: **STATE** типа **BOOL**

17) [DAS] Консоль

Данный раздел предназначен для вывода отладочной информации в консоль.



PRINT_BOOL



ENABLED – разрешение вывода отладочной информации типа **BOOL**

LABEL – метка для вывода отладочной информации. Объявляется в таблице переменных. Тип **STRING**, класс **Input**. В **Initial Value** вводится текст, которым метка обозначена в консоли:

VALUE – снимаемое значение для вывода в консоль. Тип соответствует типу блока, для **PRINT_BOOL** тип **BOOL**, для **PRINT_REAL** тип **REAL** и тд

Функциональный блок **PRINT_** представлен для всех используемых в FBD типов данных.

При выводе в консоль метки буду повторяться циклически