Утвержден
ЛЯЮИ.00707-01 92 01-ЛУ

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

«ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ ПРОГРАММ НА ЯЗЫКАХ СТАНДАРТА МЭК 61131-3 ELPLC-LOGIC»

Приложение

Редактор кода САПР ELPLC-LOGIC

ЛЯЮИ.00707-01 92 01

Листов 81

Подпись и дата	
Инв.№ дубл.	
Взам.инв.№	
Подпись и дата	
Инв.№ подл.	

Перв. примен. Литера

2023

АННОТАЦИЯ

Данный документ содержит сведения по работе с редактором кода на языках стандарта МЭК61131-3 в системе автоматизированного проектирования «ELPLC-LOGIC». Рассказывается о приемах написания кода, описываются основные конструкции кода и правила его написания. Описываются интерфейсные решения.

СОДЕРЖАНИЕ

1. РЕДАКТОР КОДА ДЛЯ ЯЗЫКОВ СТАНДАРТА ИСО МЭК 61131-3

Редактор кода среды разработки «ELPLC-LOGIC» состоит из следующих компонент:

- текстовые редакторы языков ST и IL;
- графические редакторы языков FBD, SFC, LD;
- панель отладки;
- панель графика изменения значения переменной в режиме отладки.

Далее подробно рассказано про каждый компонент среды разработки «ELPLC-LOGIC» в отдельности.

2. ТЕКСТОВЫЙ РЕДАКТОР 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 (см. Рисунок 14).

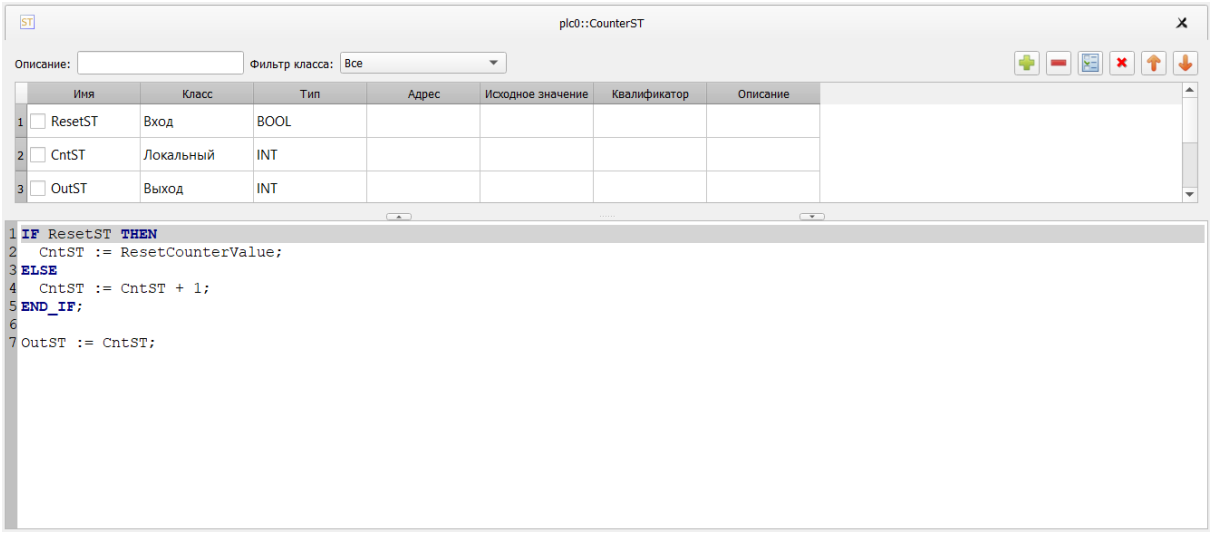

Рисунок 1 - Редактор языков ST и IL

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;

– увеличение или уменьшение размера шрифта (с помощью Ctrl + <колёсико мыши>).

3. ГРАФИЧЕСКИЕ РЕДАКТОРЫ ДИАГРАММ ЯЗЫКОВ FBD, LD, SFC

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

9.1. Редактор языка FBD

Основными элементами языка FBD являются: переменные, функции, функциональные блоки и подключения. При редактировании FBD диаграммы, в панели инструментов появляется следующая панель (см. Рисунок 15).

Рисунок 2 - Панель редактирования FBD диаграмм

С помощью данной панели можно добавить все элементы языка FBD (назначение каждой кнопки описано в таблице (см. Таблица 1).

Таблица 1 - 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 редакторе одного из графических языков
	Перемещение диаграммы	Перевод указателя мыши в состояние, при котором можно изменять размеры редактора одного из графических языков, с помощью его перемещения
	Выровнять по левому краю	Выбранные элементы диаграммы выравниваются вдоль по левому краю
	Выровнять по правому краю	Выбранные элементы диаграммы выравниваются вдоль по правому краю
	Выровнять по верхнему краю	Выбранные элементы диаграммы выравниваются вдоль по верхнему краю
	Выровнять по нижнему краю	Выбранные элементы диаграммы выравниваются вдоль по нижнему краю
	Сделать одинаковой высоты	Выбранные элементы диаграммы получают одинаковую высоту
	Сделать одинаковой ширины	Выбранные элементы диаграммы получают одинаковую ширину
	Создать новый комментарий	Вызов диалога редактирования комментария
	Создать новое подключение	Вызов диалога редактирования подключения

Внешний вид кнопки	Наименование кнопки	Функции кнопки
	Создать новую переменную	Вызов диалога редактирования переменной
	Создать новый блок	Вызов диалога редактирования функции или функционального блока

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

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

Далее рассмотрено добавление каждого элемента в отдельности.

9.1.1. Добавление функции или функционального блока

При добавлении функции или функционального блока одним из описанных выше способов, появится диалог «Свойства блока» (см. Рисунок 16).

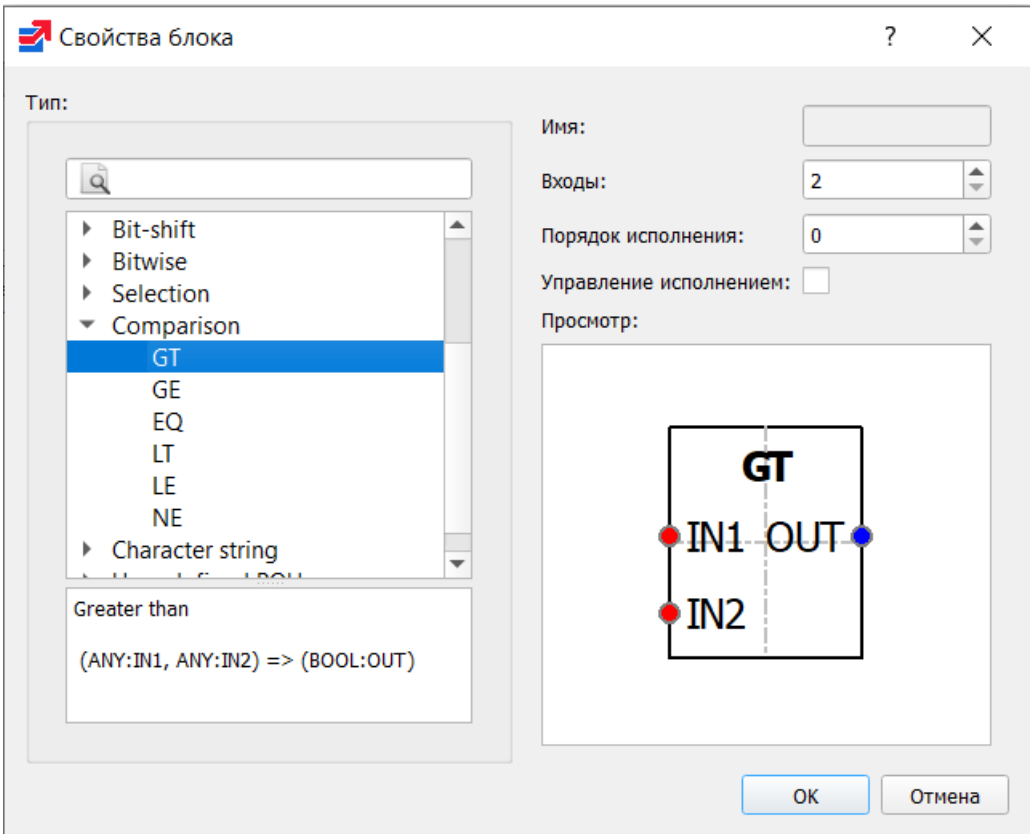

Рисунок 3 - Свойства функции или функционального блока

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

Опция «Управление выполнением» добавляет в функцию или функциональный блок дополнительные параметры EN/ENO. Для сохранения изменений необходимо нажать «OK». Добавление (путем копирования существующего блока), удаление и переименование функции или функционального блока осуществляется при помощи команд меню «Редактирование» в главном меню или с помощью всплывающего меню диаграммы.

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

9.1.2. Добавление переменной

Переменные добавляются из панели переменных и констант с помощью перетаскивания (Drag&Drop) левой клавишей мыши в область редактирования FBD диаграмм.

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

В данном диалоге можно задать порядок выполнения переменной и изменить её класс («Входная», «Выходная», «Входная/Выходная») (см. Рисунок 17).

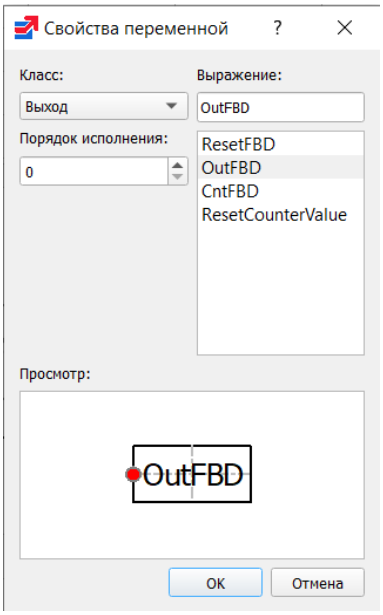

Рисунок 4 - Свойства переменной

9.1.3. Добавление подключения

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

После выбора добавления элемента «Подключение» появится диалог «Свойства подключения» (см. Рисунок 18).

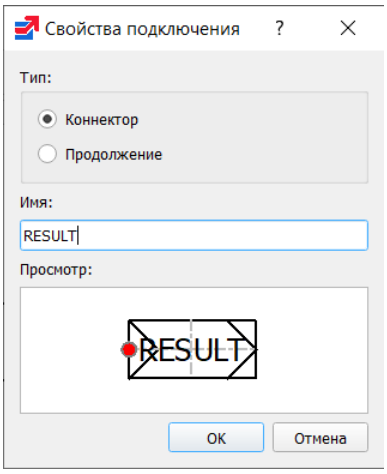

Рисунок 5 - Свойства подключения

В данном диалоге можно выбрать тип подключения: «Выходное подключение» – для выходного значения, «Входное подключение» – для входного значения, а также необходимо указать имя данного подключения.

На рисунке представлен пример использования подключений (см. Рисунок 19).

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

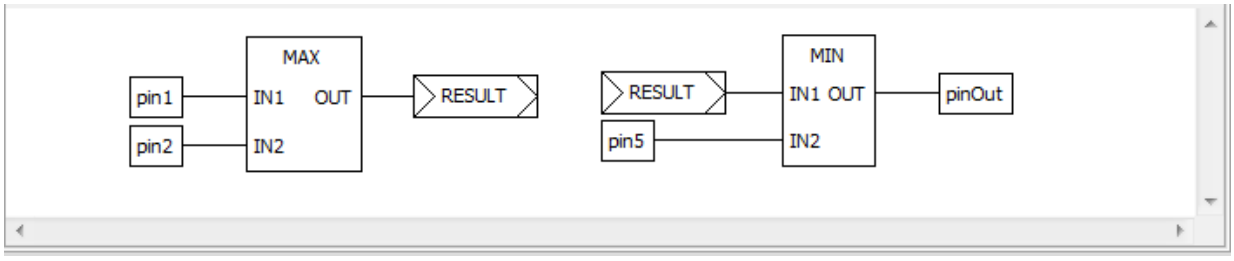

Рисунок 6 - Пример FBD диаграммы с использованием подключений

9.1.4. Добавление комментариев

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

После нажатия кнопки OK комментарий появится на диаграмме (см. Рисунок 7).

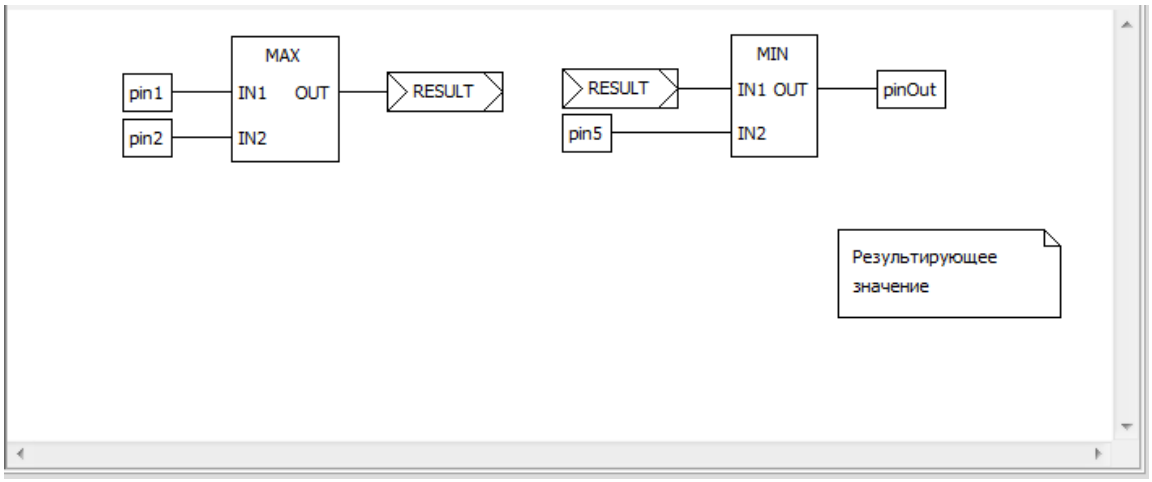

Рисунок 7 - Добавленный комментарий к FBD диаграмме

9.1.5. Порядок выполнения функций и функциональных блоков

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

Если обратиться к рисунку (см. Рисунок 8), то порядок выполнения функций будет следующим:

– 1 - SUB;
– 2 - MUL;
– 3 - ADD.

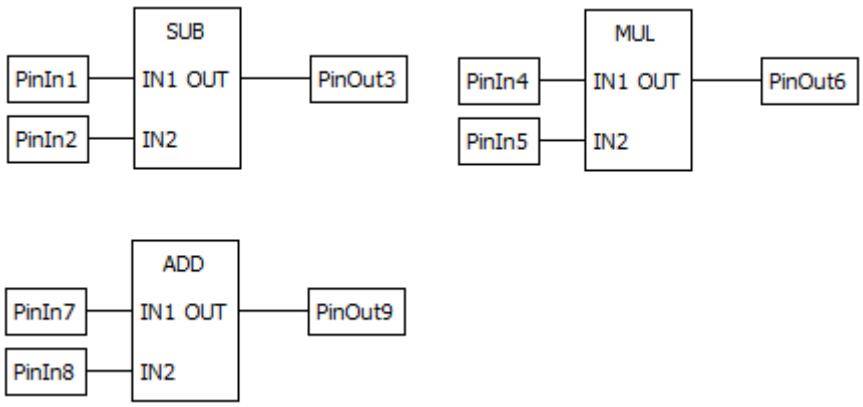

Рисунок 8 - Схема, содержащая функции с порядком выполнения (обсчёта) по расположению

Порядок выполнения может быть изменён вручную с помощью диалога свойств. Данная опция «Порядок выполнения» выделена красным цветом на рисунке (см. Рисунок 9).

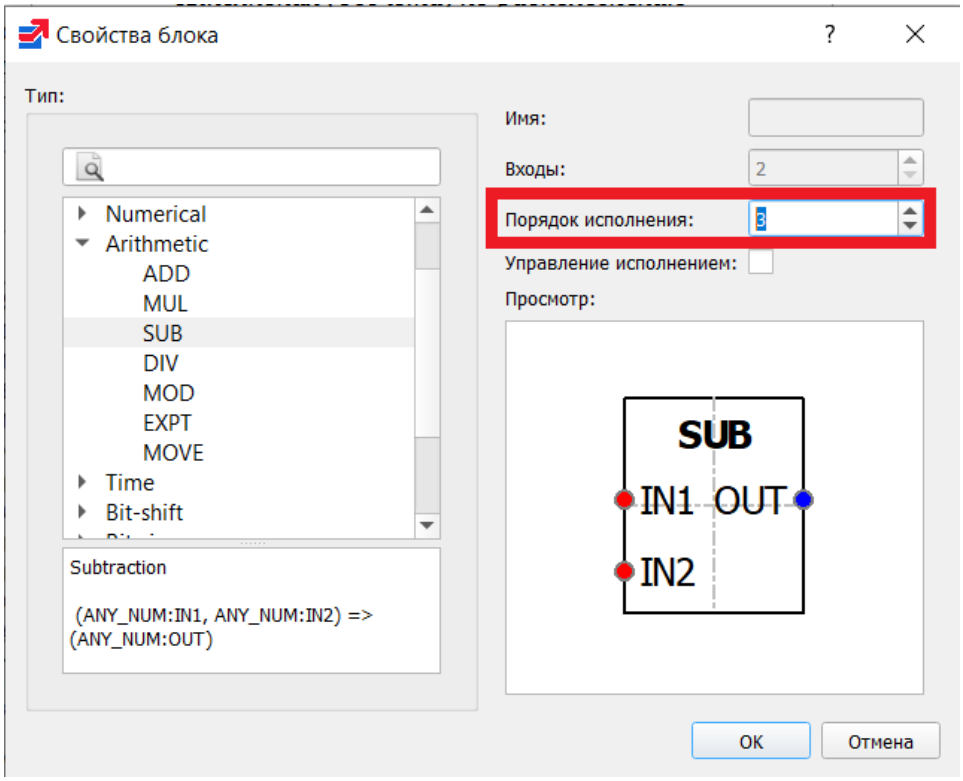

Рисунок 9 - Свойство порядок выполнения функции или функционального блока

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

9.2. Редактор языка LD

Язык LD или РКС (Релейно-Контактные Схемы) представляет собой графическую форму записи логических выражений в виде контактов и катушек реле. Основными элементами языка LD являются: шина питания, катушка, контакт. Добавить данные элементы, также как и элементы языка FBD, можно несколькими способами. Как только активной становится вкладка с редактированием LD диаграммы, в панели инструментов появляется панель (см. Рисунок 10) с элементами языка LD.

Рисунок 10 - Панель редактирования LD диаграмм

Аналогично редактору языка FBD с помощью данной панели можно добавить все элементы языка LD, а также и FBD, т.к. есть возможность комбинированного применения языков на одной диаграмме. В таблице (см. Таблица 2) приведено описание кнопок данной панели. Описание остальных кнопок, относящихся к языку FBD, находится в таблице (см. Таблица *2*).

Таблица 2 - Кнопки панели редактирования LD диаграммы

Внешний вид кнопки	Наименование кнопки	Функции кнопки
	Создать новую линию питания	Вызов диалога редактирования шины питания
	Создать новую цепь	Вызов диалога редактирования катушки или контакта

Во всплывающем меню для редактора LD диаграмм, также как и в панели инструментов помимо элементов LD языка, доступны элементы языка FBD.

9.2.1. Добавление шины питания

При добавлении шины питания, одним из описанных выше способов, появится диалог (см. Рисунок 11).

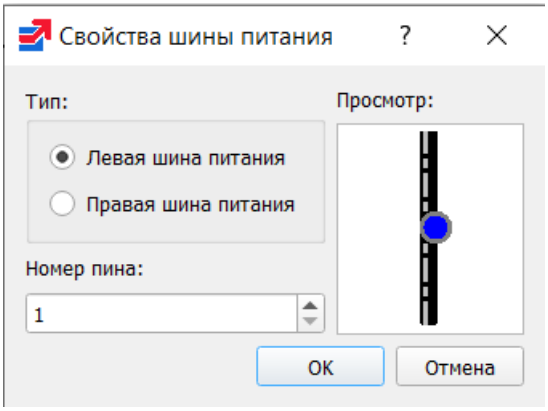

Рисунок 11 - Редактирование шины питания

В данном диалоге указываются следующие свойства:

– тип шины питания: шина питания слева или шина питания справа;

– количество контактов на добавляемой шине питания.

9.2.2. Добавление контакта

При добавлении контакта на LD диаграмму появится диалог «Свойства контакта / катушки» (см. Рисунок 12).

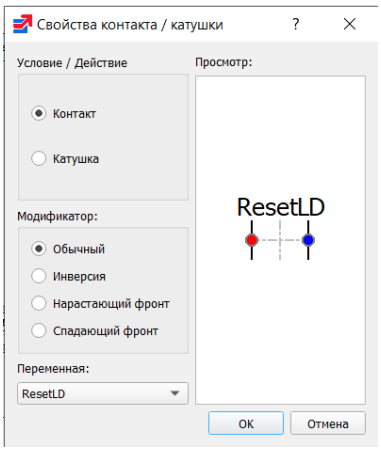

Рисунок 12 - Редактирование контакта

Данный диалог позволяет определить модификатор данного контакта:

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

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

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

9.2.3. Добавление катушки

При добавлении катушки на LD диаграмму появится диалог «Свойства контакта / катушки» (см. Рисунок 13).

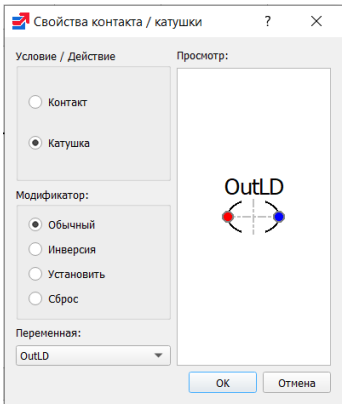

Рисунок 13 - Редактирование катушки

В данном диалоге можно определить модификатор данной катушки:

– «Обычный»;
– «Инверсия»;
– «Установить»;
– «Сброс».

Кроме того, диалог позволяет выбрать из выпадающего списка переменную, «связываемую» с данной катушкой. Эти переменные, как и для контактов, должны быть определены в панели переменных и констант для данного программного модуля типом BOOL.

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

9.3. Редактор языка SFC

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

Как только активной становится вкладка с редактированием SFC диаграммы, в панели инструментов появляется следующая панель (см. Рисунок 14).

Рисунок 14 - Панель редактирования SFC диаграмм

В таблице (см. Таблица 3) приведено описание кнопок данной панели. Описание остальных кнопок, относящихся к языку FBD и LD и так же находящихся на этой панели, приведены в таблицах (см. Таблица 1 и см. Таблица 2) соответственно.

Таблица 3 - Кнопки панели редактирования SFC диаграммы

Внешний вид кнопки	Наименование кнопки	Функции кнопки
	Создать новый шаг	Вызов диалога редактирования шага
	Создать новый переход	Вызов диалога редактирования перехода
	Создать новый блок действий	Вызов диалога редактирования блока действий
	Создать новое ветвление	Вызов диалога редактирования ветвления (дивергенции) или объединения (конвергенции)
	Создать новый безусловный переход («прыжок»)	Вызов диалога редактирования «прыжка»

Далее даётся описание добавления приведённых в (см. Таблица 3) элементов языка SFC.

9.3.1. Добавление шага инициализации и шага

Процедура добавления шага инициализации и обычного шага ничем не отличается. В обоих случаях вызывается диалог «Свойства шага» (см. Рисунок 15).

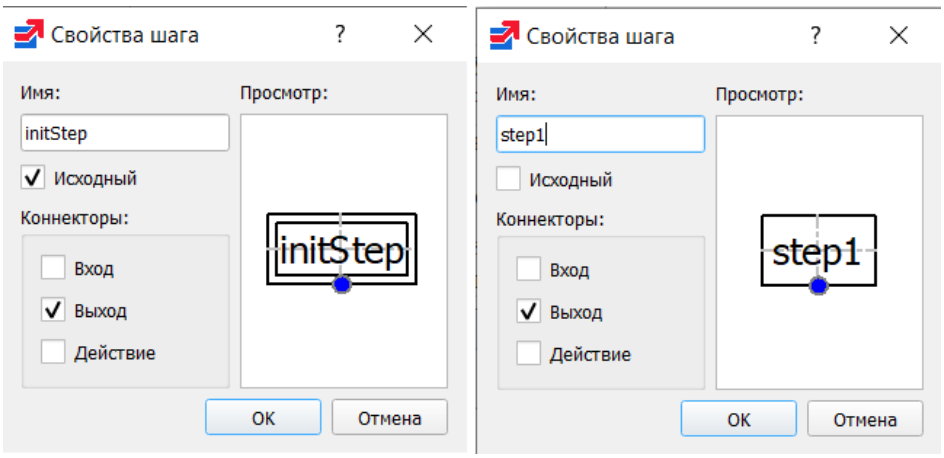

Рисунок 15 - Диалоги редактирования шага инициализации и обычного шага SFC диаграммы

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 (см. Рисунок 28).

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

– «Вход»;
– «Выход»;
– «Действие».

«Действие» добавляет соединитель для связывания данного шага с блоком действий. «Вход» и «Выход», как правило, соединены с переходом. Соответственно, после нажатия кнопки OK, на диаграмму будет добавлен шаг с указанными соединителями (см. Рисунок 16).

Рисунок 16 - Шаг инициализации языка SFC

9.3.2. Добавление перехода

При добавлении на SFC диаграмму перехода, появится диалог «Свойства перехода» (см. Рисунок 17).

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

– «Ссылка»;
– «Встроенный код»;
– «Соединение».

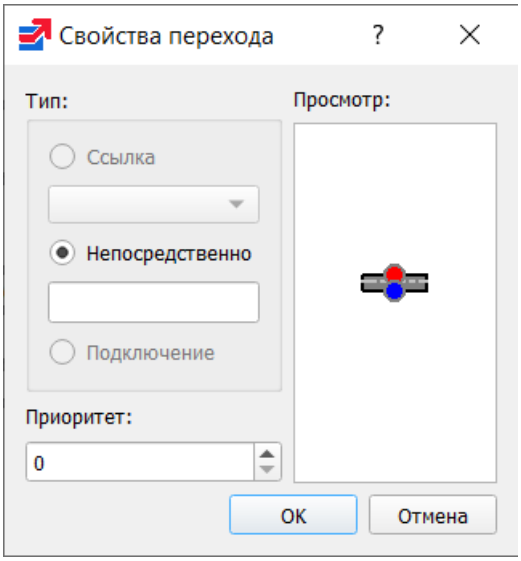

Рисунок 17 - Диалог редактирования перехода для SFC диаграммы

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 «Встроенный код», условие перехода можно написать в виде выражения на языке ST.

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

f3 = f4

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

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

При выборе типа перехода «Соединение», у добавленного перехода появится слева контакт, который необходимо соединить с выходным значением, например, функционального блока языка FBD или катушки LD диаграммы. Стоит отметить, что данное выходное значение должно быть типа BOOL. Ниже, на рисунке (см. Рисунок 18) красным цветом выделен пример перехода, условия которого задано с помощью языка LD – двух последовательно соединённых контактов языка LD.

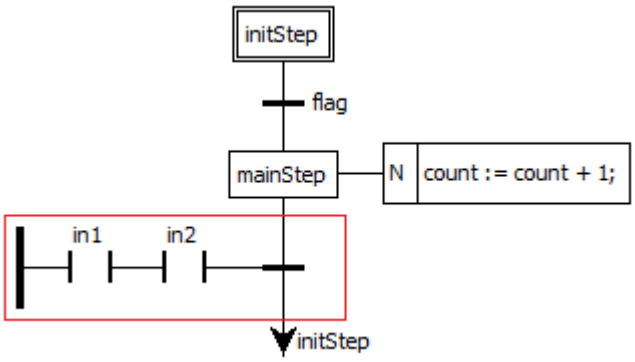

Рисунок 18 - Пример SFC диаграммы, в которой один из переходов задан с помощью языка LD

9.3.3. Добавление блока действий

При добавлении блока действий на диаграмму появится диалог «Свойства блока действий» (см. Рисунок 19).

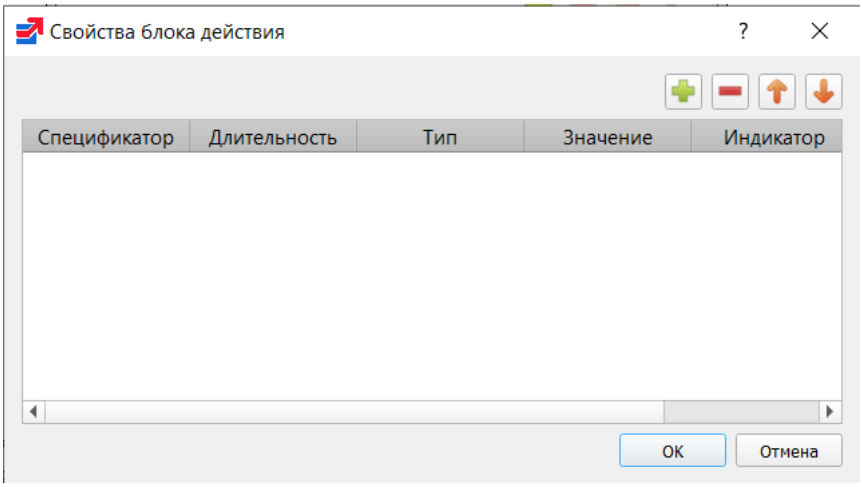

Рисунок 19 - Диалог редактирования свойств блока действий

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

– «Спецификатор»;
– «Длительность»;
– «Тип»: «Действие», «Переменная», «Встроенный код»;
– «Значение»;
– «Индикатор».

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

Подробное описание спецификаторов, которые выбираются из предлагаемого списка при добавлении действия приведено в таблице (см. Таблица 4).

Таблица 4 - Спецификаторы действий SFC диаграммы

Имя спецификатора	**Поведение блока действия**
D	Действие начинает выполняться через некоторое заданное время (если шаг ещё 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 остаётся 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 ещё активен
SL	Действие активно в течении некоторого, заданного интервала
SD	Действие начинается выполняться через некоторое время, даже в том случае если шаг уже не активен

Поле «Продолжительность» необходимо для установки интервала времени необходимого для некоторых квалификаторов, описанных выше в таблице (Таблица 4).

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

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

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

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

В случае выбора «Встроенный код», появляется возможность в поле «Значение» написать на языке ST код, который будет выполняться, когда действие становится активным (см. Рисунок 20).

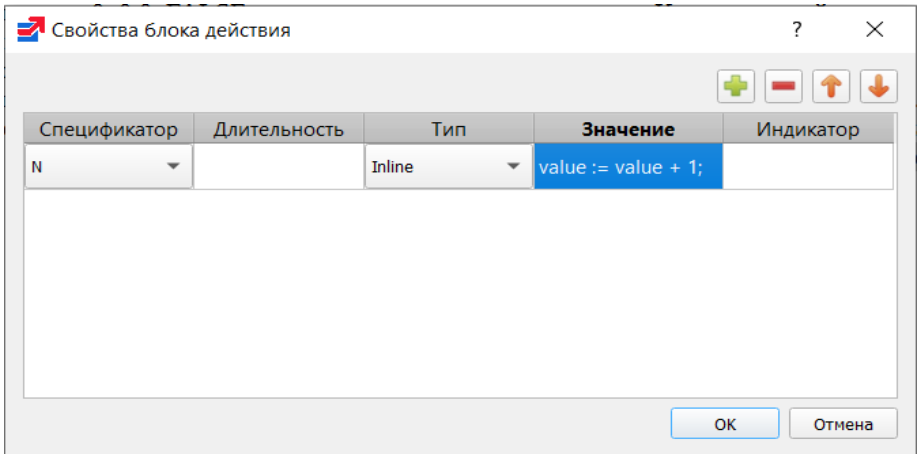

Рисунок 20 - Написание встроенного кода для действия

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

После добавления блока действия на диаграмму необходимо его ассоциировать с конкретным шагом. Данная операция выполняется обычным соединением правого контакта у шага и левого контакта у действия (см. Рисунок 21).

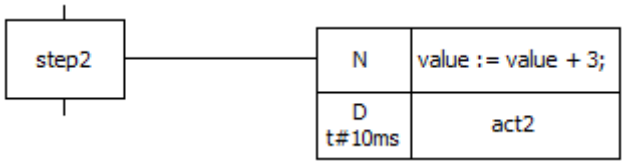

Рисунок 21 - Ассоциирование шага step2 блоком действия, содержащим два действия

9.3.4. Добавление ветвления / объединения

При добавлении дивергенции, появится диалог «Свойства ветвления/объединения» (см. Рисунок 22).

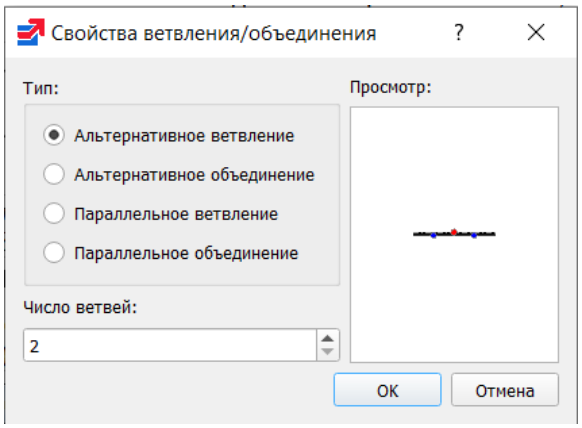

Рисунок 22 - Диалог редактирования ветвления

В первую очередь следует выбрать тип ветвления:

– «Альтернативное ветвление»;
– «Альтернативное объединение»;

– «Параллельное ветвление»;

– «Параллельное объединение».

Вторым параметром является количество разветвлений, которое определяет на сколько ветвей будет либо расходиться («Альтернативное / Параллельное ветвление») одна ветвь, либо сколько ветвей будет сходиться в одну ветвь («Альтернативное / Параллельное объединение»).

Пример дивергенции с двумя разветвлениями показан на рисунке (см. Рисунок 23) и выделен красным цветом.

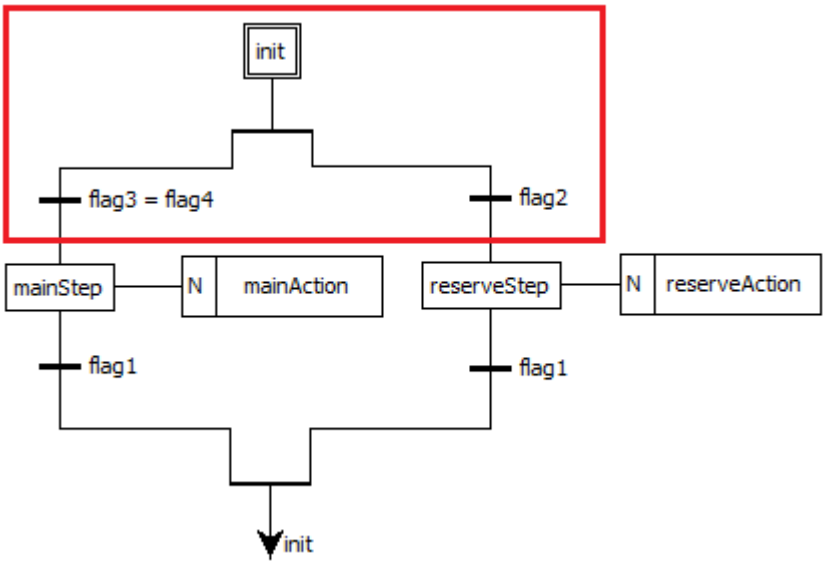

Рисунок 23 - Пример SFC диаграммы, содержащей дивергенцию

Пример конвергенции выделен красным цветом на рисунке (см. Рисунок 24).

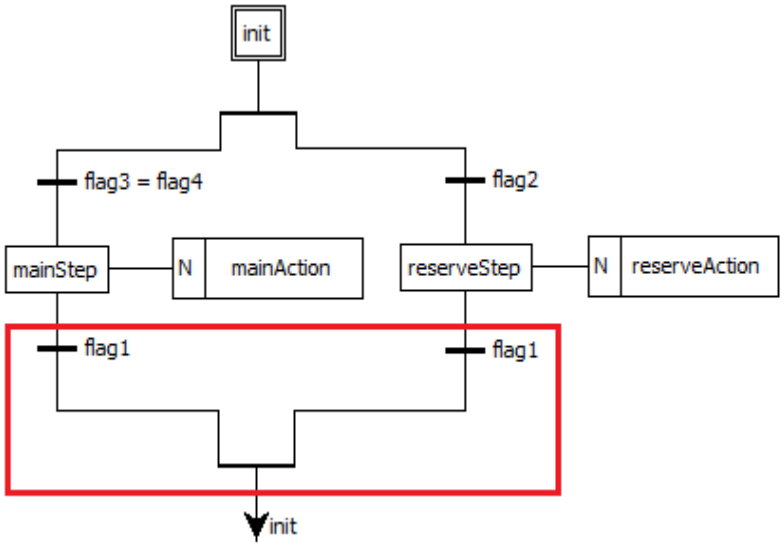

Рисунок 24 - Пример SFC диаграммы, содержащей конвергенцию

Пример параллельной дивергенции показан на рисунке (см. Рисунок 25) и выделен красным цветом.

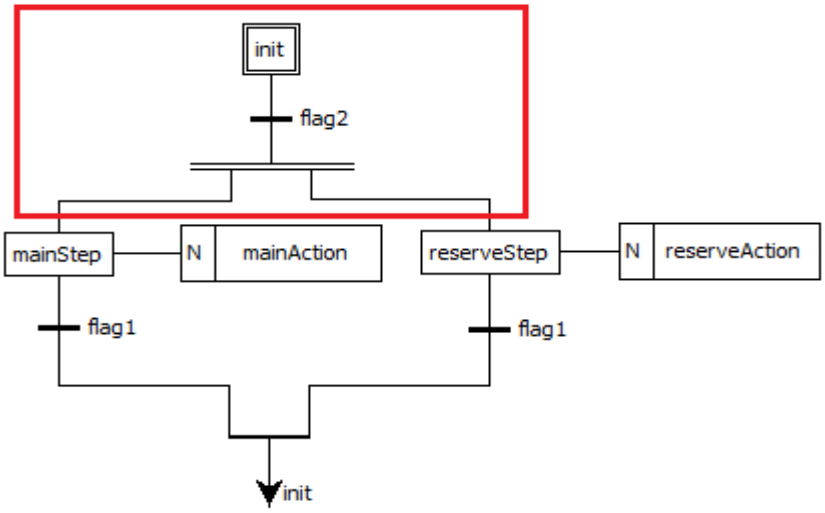

Рисунок 25 - Пример SFC диаграммы, содержащей параллельную дивергенцию

9.3.5. Добавление безусловного перехода («прыжка»)

Элемент «прыжок» на SFC диаграмме подобен выполнению оператора GOTO при переходе на определённую метку в коде в различных языках программирования. После выбора добавления прыжка на SFC диаграмму, появится диалог (см. Рисунок 26), в котором необходимо выбрать из списка шаг, к которому будет происходить «прыжок» – переход от одного шага SFC диаграммы к другому.

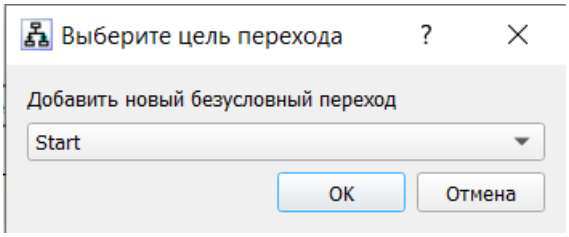

Рисунок 26 - Диалог добавления «прыжка»

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.

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

9.3.6. Предопределённые условия перехода и действия в дереве проекта

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

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

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

Добавление действия в структуру SFC диаграммы происходит аналогично добавлению перехода в данную структуру.

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

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

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

4. ПАНЕЛЬ РЕДАКТИРОВАНИЯ РЕСУРСА

Панель редактирования ресурса (см. Рисунок 27) содержит панель переменных и констант, которая позволяет определять глобальные переменные на уровне ресурса и панели, содержащие задачи и экземпляры.

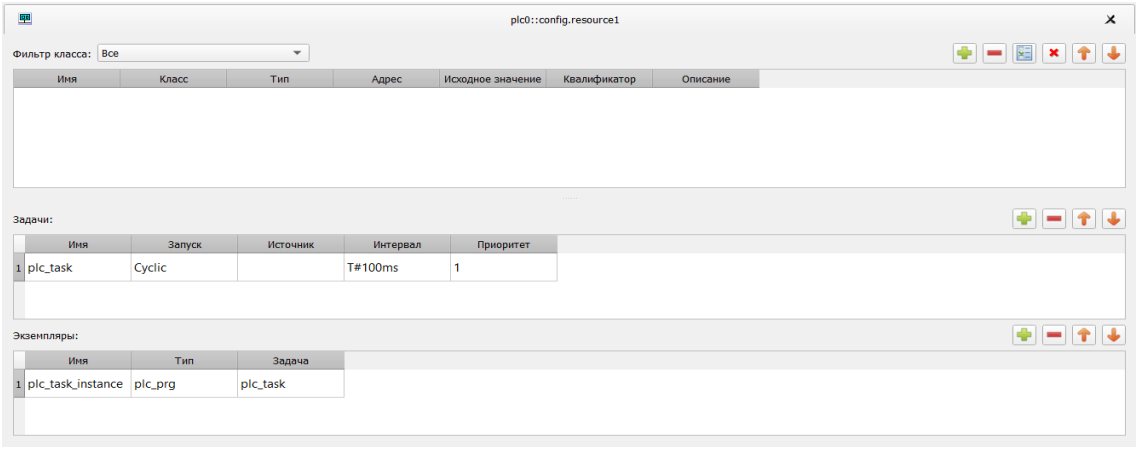

Рисунок 27 - Вкладка ресурс главной рабочей области

Добавление переменных в ресурс ничем не отличается от добавления переменных в программные модули, единственное исключение – переменные могут быть только класса «Глобальная». Основной задачей данной панели является возможность добавить экземпляр, указать для него программный модуль типа «Программа», из ранее определённых в проекте, для поля «Тип» и выбрать задачу из добавленных в список «Задачи».

5. ПАНЕЛЬ РЕДАКТИРОВАНИЯ ТИПА ДАННЫХ

Панель редактирования типа данных позволяет определить различные параметры создаваемого пользовательского типа данных.

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

– синоним;

– поддиапазон (выделение диапазона значений стандартного типа);

– перечисление;

– массив;

– структура, позволяющая определять тип, основанный на объединении несколько типов.

Далее рассмотрены подробнее параметры для каждого из вышеперечисленных типов.

11.1. Синоним

При выборе «Синоним» (см. Рисунок 28), из списка указывается базовый тип и его начальное значение.

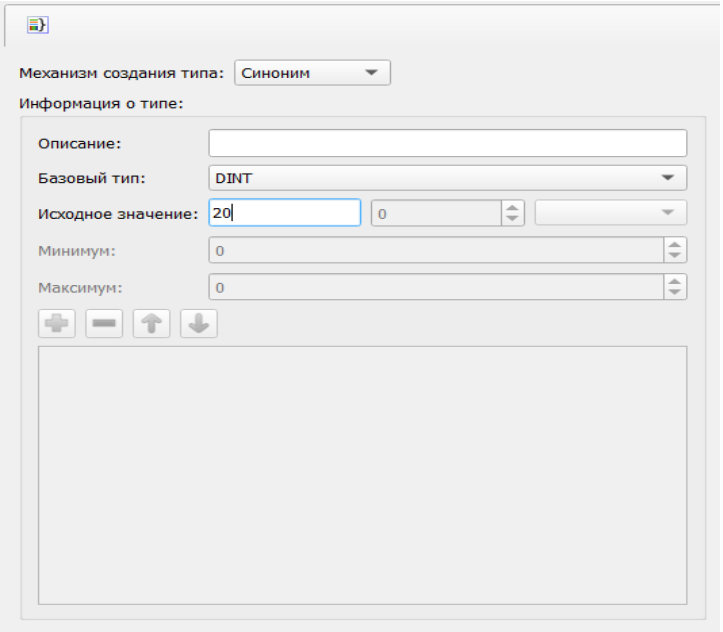

Рисунок 28 - Добавление синонима типа данных

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

11.2. Поддиапазон

В случае выбора механизма создания нового типа «Поддиапазон», помимо базового типа и начального значения производится установка параметров «Минимум» и «Максимум»

(см. Рисунок 29), т.е. соответственно минимального и максимального значения, которое может принимать создаваемый тип данных.

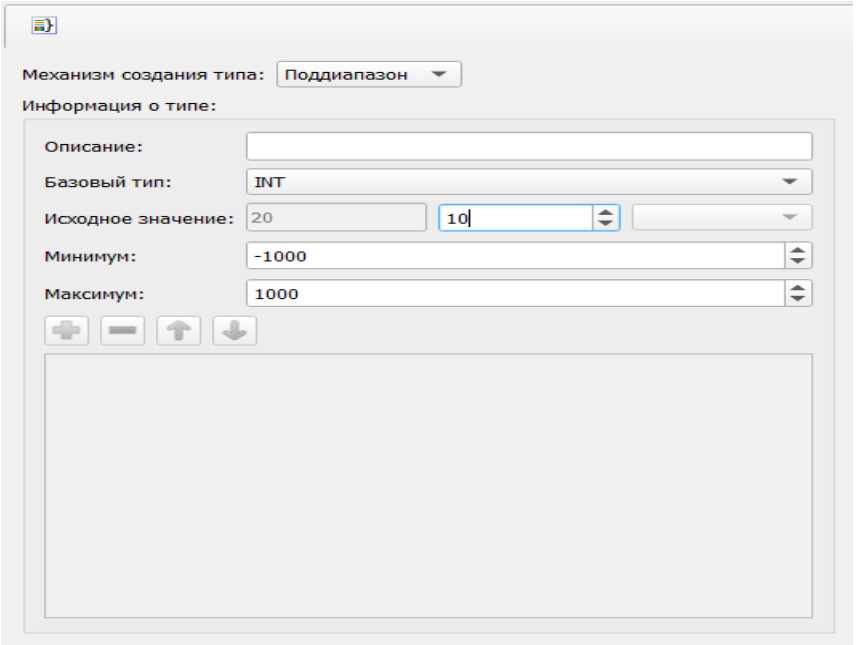

Рисунок 29 - Добавление нового типа данных, представляющего поддиапазон существующего типа

11.3. Перечисление

При выборе механизма создания нового типа «Перечисление» (см. Рисунок 30), появится панель, содержащая таблицу, в которой можно задать список возможных значений данного перечисляемого типа.

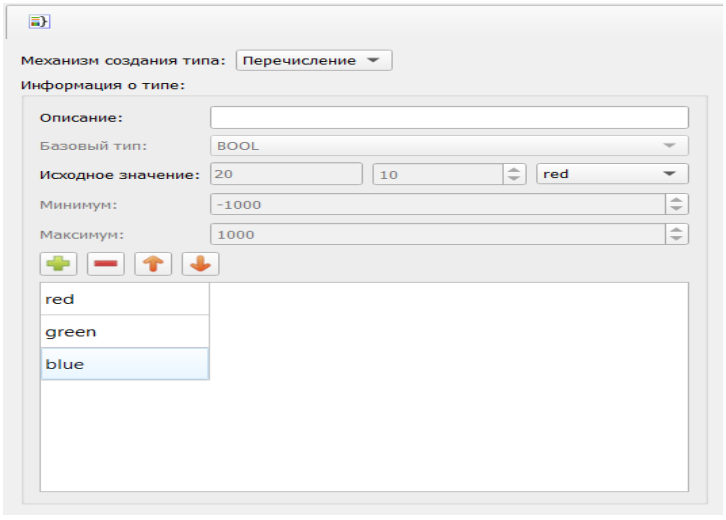

Рисунок 30 - Добавление перечисляемого типа данных

Добавление, редактирование, удаление, перемещение данных значений осуществляется с помощью кнопок, описание которых приведено в таблице (см. Таблица 5).

Таблица 5 - Кнопки редактирования значений перечисляемого типа

Внешний вид кнопки	Наименование кнопки	Функции кнопки
	Добавить	Добавить новое поле в таблицу
	Удалить	Удалить выделенное поле в таблице
	Переместить вверх	Переместить вверх выделенное поле в таблице
	Переместить вниз	Переместить вниз выделенное поле в таблице

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

11.4. Массив

При выборе механизма создания нового типа «Массив» (см. Рисунок *31*) появится панель, в которой необходимо указать базовый тип, начальное значение, а также размерность массива.

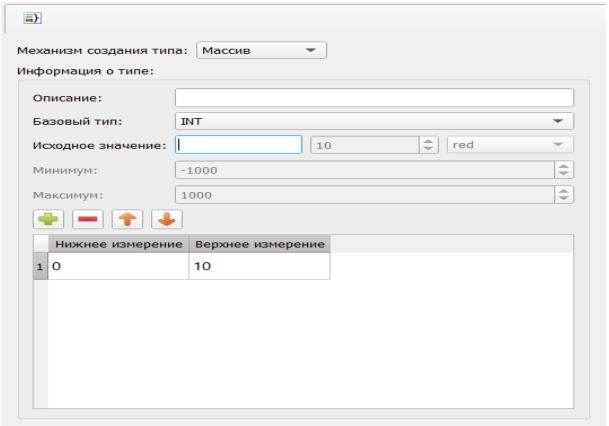

Рисунок 31 - Добавление типа данных – массива

11.5. Структура

При выборе механизма создания нового типа «Структура» (см. Рисунок 32), в появившейся таблице необходимо добавить необходимое количество полей структуры. Каждое поле имеет своё имя, тип и начальное значение.

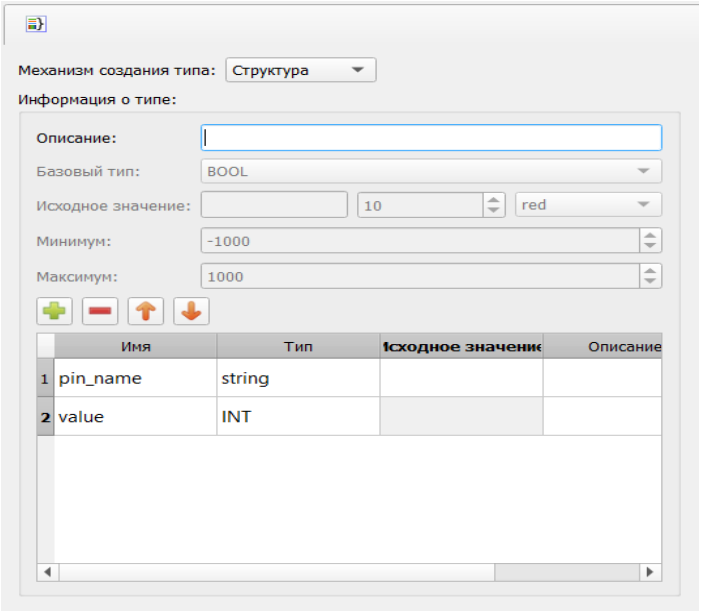

Рисунок 32 - Добавление типа данных – структуры

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

6. ПАНЕЛЬ ЭКЗЕМПЛЯРОВ ПРОЕКТА

Панель экземпляров проекта (см. Рисунок 33) обычно располагается слева в среде разработки «ELPLC-LOGIC» и отображаемые в ней экземпляры зависят от выбранного элемента в дереве проекта.

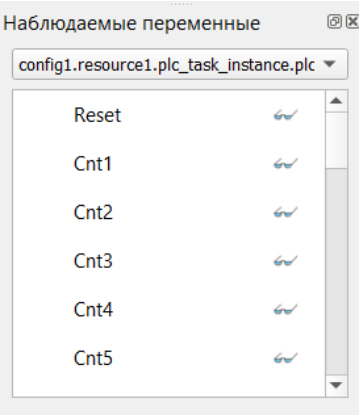

Рисунок 33 - Панель экземпляров проекта

При выборе в дереве проекта элемента, соответствующего ресурсу, в панели экземпляров проекта будут отображены экземпляры, определённые в данном ресурсе, а также глобальные переменные ресурса. При выборе в дереве проекта элемента, соответствующего программным модулям «Программа» и «Функциональный блок» в панели экземпляров будут отображены переменные, определённые в них. В случае выбора других элементов в дереве проекта, панель отладки будет пустой.

С правой стороны от каждого элемента в панели отладки располагается кнопка запуска режима отладки для экземпляра. В случае нажатия кнопки запуска режима отладки для экземпляра программы, написанной на одном из графических языков (FBD, LD или SFC), диаграмма будет отображена в режиме отладки. Если кнопка запуска режима отладки нажимается для элемента переменной, то переменная будет добавлена в панель отладки.

Описанные выше кнопки доступны только в режиме отладки прикладной программы.

7. ПАНЕЛЬ БИБЛИОТЕКИ ФУНКЦИЙ И ФУНКЦИОНАЛЬНЫХ БЛОКОВ

Панель библиотеки функций и функциональных блоков (см. Рисунок 34), как правило, располагается справа в среде разработки «ELPLC-LOGIC». Она содержит коллекцию стандартных функций и функциональных блоков, разделённых по разделам в соответствии с их назначением, которые доступны при написании алгоритмов и логики работы программных модулей.

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

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.

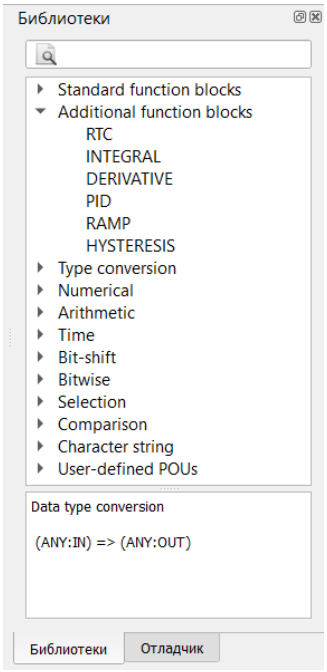

Рисунок 34 - Панель библиотеки функций и функциональных блоков

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

Имеется специальное поле поиска функционального блока по имени.

8. ПОИСК ЭЛЕМЕНТОВ В ПРОЕКТЕ

Для поиска интересующего элемента в проекте используется диалог «Поиск в проекте» (см. Рисунок 35). Его вызов происходит с помощью главного меню программы или панели инструментов.

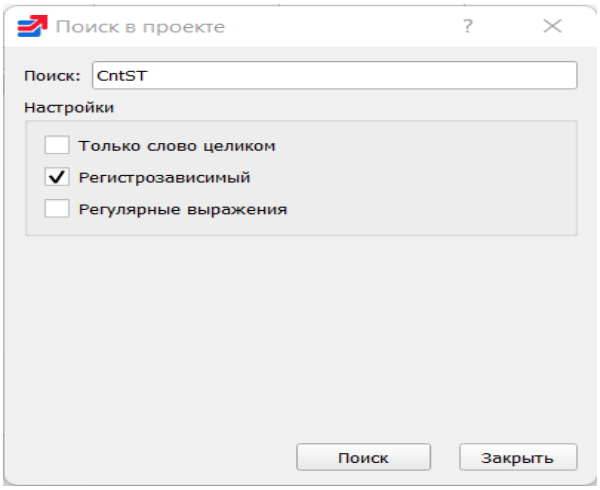

Рисунок 35 - Диалог поиска в проекте

В появившемся диалоге можно установить различные параметры поиска: шаблон поиска, область поиска, чувствительность к регистру при поиске, а также записать шаблон поиска в виде регулярного выражения. После того как все параметры установлены, необходимо нажать кнопку «Поиск» в этом диалоге. Ниже на рисунке (см. Рисунок 36) приведён пример поиска элемента с именем «CntST».

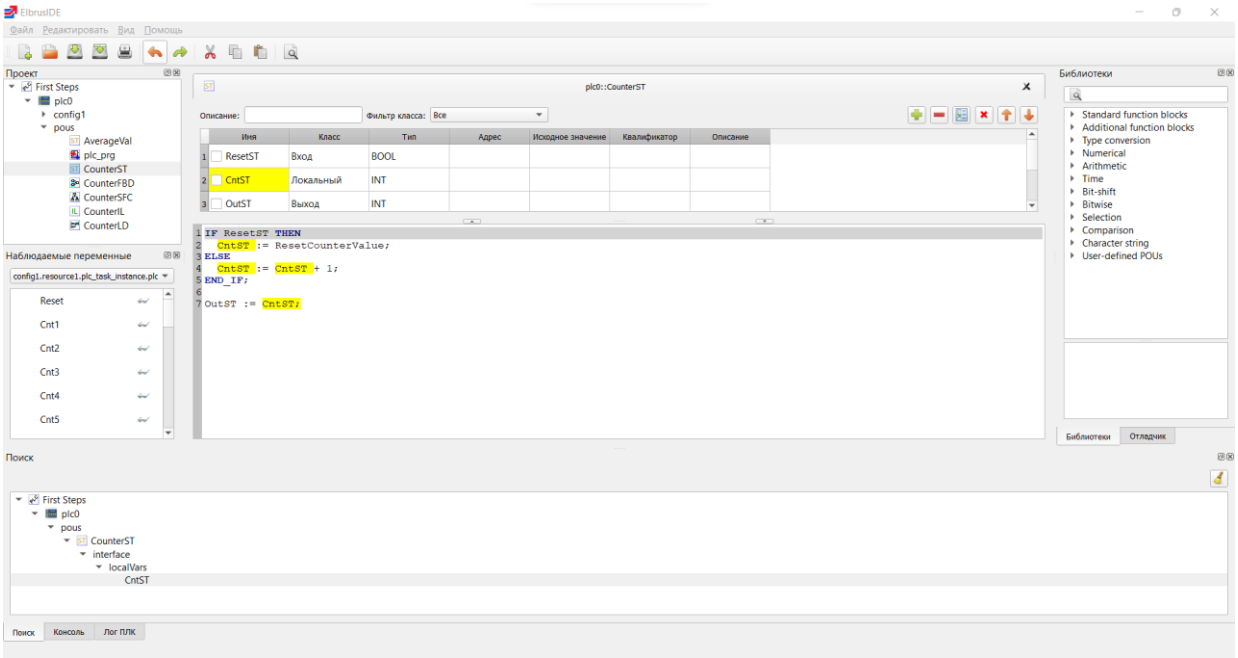

Рисунок 36 - Результат поиска элемента в проекте

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

9. ПАНЕЛЬ ОТЛАДКИ

Панель отладки располагается в правой части среды разработки «ELPLC-LOGIC» (см. Рисунок 37).

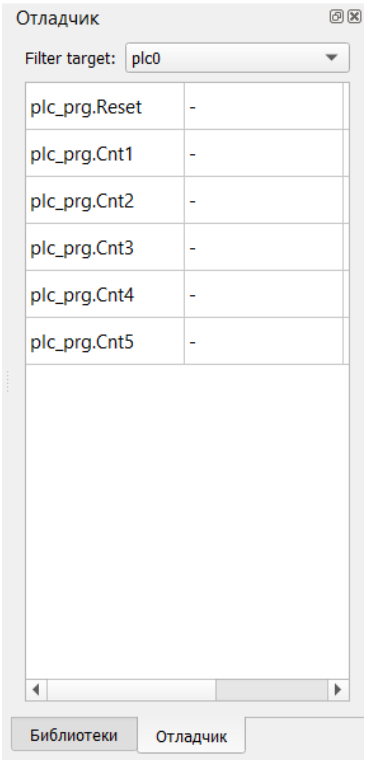

Рисунок 37 - Панель отладки

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

Изменение значений переменной во время отладки прикладной программы осуществляется нажатием специальной кнопки правее поля «Значение» интересующей переменной. Далее появится диалог ввода значения для выбранной переменной (см. Рисунок 38).

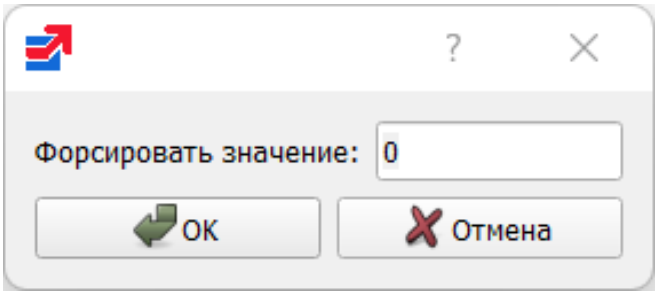

Рисунок 38 - Установка значения переменной во время отладки

10. ЯЗЫКИ СТАНДАРТА МЭК 61131-3

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

Стандарт IEC 61131-3 устанавливает пять языков программирования ПЛК (три графических и два текстовых) со следующими названиями:

- Structured Text (ST, структурированный текст);
- Instruction List (IL, список инструкций);
- Function Block Diagram (FDB, диаграммы функциональных блоков);
- Ladder Diagram (LD, релейно-контактные схемы, релейные диаграммы);
- Sequential Function Chart (SFC, последовательные функциональные схемы).

В данном разделе приведены общие сведения о каждом из языков.

18.1. Structured Text (ST)

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

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

[variable] := [value];

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

18.1.1. Типы данных

Согласно стандарту IEC 61131-3, язык ST поддерживает весь необходимый набор типов, аналогичный классическим языкам программирования. Целочисленные типы: SINT (char), USINT (unsigned char), INT (short int), UINT (unsigned int), DINT (long), UDINT (unsigned long), LINT (64 бит целое), ULINT (64 бит целое без знака). Действительные типы: REAL

(float), LREAL (double). Специальные типы BYTE, WORD, DWORD, LWORD представляют собой битовые строки длиной 8, 16, 32 и 64 бит соответственно. Битовых полей в ST нет. К битовым строкам можно непосредственно обращаться побитно. Например:

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

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

strA := strB;

Для работы со строками есть стандартный набор функций.

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

В таблице (см. Таблица 6) приведены значения по умолчанию, соответствующие описанным выше типам.

Таблица 6 - Значения по умолчанию для типов данных IEC 61131-3

Тип(ы) данных	**Значение**
BOOL, SINT, INT, DINT, LINT	0
USINT, UINT, UDINT, ULINT	0
BYTE, WORD, DWORD, LWORD	0
REAL, LREAL	0.0
TIME	T#0S
DATE	D#0001-01-01
TIME_OF_DAY	TOD#00:00:00
DATE_AND_TIME	DT#0001-01-01-00:00:00
STRING	‘’ (пустая строка)

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

str1: STRING := ‘Hello world’;

В определённых ситуациях при разработке программных модулей удобно использовать обобщения типов, т.е. общее именование группы типов данных. Данные обобщения приведены в таблице (см. Таблица 7).

Таблица 7 - Обобщения типов данных IEC 61131-3

NY	ANY_NUM	ANY_INT	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT
		ANY_REAL	REAL, LREAL
	ANY_BIT		BOOL, BYTE, WORD, DWORD, LWORD
	STRING		
	TIME		
	ANY_DATE		DATE, TIME_OF_DAY, DATE_AND_TIME

18.1.2. Конструкции языка

К конструкциям языка ST относятся:

- арифметические операции;
- логические (побитовые) операции;
- операции сравнения;
- операция присвоения;
- конструкция IF – ELSEIF – ELSE;
- цикл FOR;
- цикл WHILE;
- цикл REPAET UNTIL;
- конструкция CASE.

При записи арифметических выражений допустимо использование скобок для указания порядка вычислений. При записи выражений допустимо использовать переменные (локальные и глобальные) и константы.

18.1.3. Арифметические операции

К арифметическим операциям относятся:

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

Приоритет операций в выражениях указан в таблице (см. Таблица 8) (чем выше приоритет, тем раньше исполняется операция).

18.1.4. Логические (побитовые) операции

К данным операциям относятся:

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

18.1.5. 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.

18.1.6. Присвоение

Для обозначения присвоения используется парный знак «:=». В правой и левой части выражения должны быть операнды одного типа (автоматического приведения типов не предусмотрено). В левой части выражения (принимающая сторона) может быть использована только переменная. Правая часть может содержать выражение или константу.

В таблице (см. Таблица 8) приведены приоритеты при выполнении описанных выше операций.

Таблица 8 – Приоритеты операций

Операция	**Приоритет**
Сравнения	1
Сложение, вычитание	2
Умножение, деление	3
OR	4
AND, XOR	5
NOT	6
Унарный минус	7
Вызов функции	8

18.1.7. Конструкция IF – ELSEIF – ELSE

Для описания некоторых конструкций языка удобно использовать фигурные и квадратные скобки. Считается, что:

– выражение в фигурных скобках может использоваться ноль или больше раз подряд;
– выражение в квадратных скобках не обязательно к использованию.

Конструкция IF-ELSEIF-ELSE имеет следующий формат:

IF <boolean expression> THEN <statement list>
[ELSEIF <boolean expression> THEN <statement list>]

```
[ELSE <statement list>]
END_IF;
```

Например:

```
IF Var <> 0
THEN Var := 1
ELSEIF Var > 0
THEN Var := 0;
ELSE Var := 10;
END_IF;
```

Конструкция допускает вложенность, т.е. внутри одного IF может быть ещё один и т.д. Например:

```
IF Var > 10 THEN
IF Var < Var2 + 1
THEN Var := 10;
ELSE Var := 0;
END_IF;
END_IF;
```

18.1.8. Цикл FOR

Служит для задания цикла с фиксированным количеством итераций. Формат конструкций следующий:

```
FOR <Control Variable> := <expression1> TO <expression2>
[BY <expression3>] DO
<statement list>
END_FOR;
```

При задании условий цикла считается, что <Control Variable>, <expression1> … <expression3> имеют тип INT. Выход из цикла будет произведён в том случае, если значение переменной цикла превысит значение <expression2>. Например:

```
FOR i := 1 TO 10 BY 2 DO
k := k * 2;
END_FOR;
```

Оператор BY задает приращение переменной цикла (в данном случае i будет увеличиваться на 2 при каждом проходе по циклу). Если оператор BY не указан, то приращение равно 1. Например:

```
FOR i := 1 TO k / 2 DO
var := var + k;
k := k – 1;
END_FOR;
```

Внутри цикла могут использоваться другие циклы, операторы IF и CASE. Для выхода из цикла (любого типа) может использоваться оператор EXIT. Например:

```
FOR i := 1 TO 10 BY 2 DO
k := k * 2;
```

```
IF k > 20 THEN
EXIT;
END_IF;
END_FOR;
```

Примечания:

1. Выражения <expression1> … <expression3> вычисляются до входа в цикл, поэтому изменения значений переменных, входящих в любое из этих выражений, не приведёт к изменению числа итераций. Например:

```
01: k := 10;
02: FOR I := 1 TO k / 2 DO
03: k := 20;
04: END_FOR;
```

В строке 3 производится изменение переменной k, но цикл всё равно выполнится только пять раз.

2. Значение переменной цикла может изменяться внутри тела цикла, но в начале очередной итерации значение данной переменной будет выставлено в соответствие с условиями цикла. Например:

```
01: FOR I := 1 TO 5 DO
02: I := 55;
03: END_FOR;
```

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

18.1.9. Цикл WHILE

Служит для определения цикла с предусловием. Цикл будет исполняться до тех пор, пока выражение в предложении WHILE возвращает TRUE. Формат конструкции следующий:

```
WHILE <Boolean-Expression> DO
<Statement List>
END_WHILE;
```

Значение <Boolean-Expression> проверяется на каждой итерации. Завершение цикла произойдёт, если выражение <Boolean-Expression> вернёт FALSE. Например:

```
k := 10;
WHILE k > 0 DO
i := I + k;
k := k –1;
END_WHILE;
```

Внутри цикла могут использоваться другие циклы, операторы IF и CASE. Для досрочного завершения цикла используется оператор EXIT (см. пример в описание цикла FOR).

18.1.10. Цикл REPEAT UNTIL

Служит для определения цикла с постусловием. Завершение цикла произойдёт тогда, когда выражение в предложении UNTIL вернёт FALSE. Другими словами: цикл будет выполняться, пока условие в предложении UNTIL не выполнятся. Формат конструкции следующий:

```
REPEAT
<Statement List>
UNTIL <Boolean Expression>;
END_REPEAT;
```

Например:

```
k := 10;
REPEAT
i := i + k;
k := k – 1;
UNTIL k = 0;
END_REPEAT;
```

Внутри цикла могут использоваться другие циклы, операторы IF и CASE. Для досрочного завершения цикла используется оператор EXIT (см. пример в описании цикла FOR).

18.1.11. Конструкция CASE

Данная конструкция служит для организации выбора из диапазона значений. Формат конструкции следующий:

```
CASE <Expression> OF
CASE_ELEMENT {CASE_ELEMENT}
[ELSE <Statement List>]
END_CASE;
```

CASE_ELEMENT – это список значений, перечисленных через запятую. Элементом списка может быть целое число или диапазон целых чисел. Диапазон задаётся следующим образом BEGIN_VAL .. END_VAL.

Если текущее значение <Expression> не попало ни в один CASE_ELEMENT, то управление будет передано на предложение ELSE. Если предложение ELSE не указано, то никаких действий выполнено не будет.

Значение <Expression> может быть только целым. Например:

```
01: CASE k OF
02: 1:
03: k := k * 10;
04: 2..5:
05: k := k * 5;
```

```
06: i := 0;
07: 6, 9..20:
08: k := k – 1;
09: ELSE
10: k := 0;
11: i := 1;
12: END_CASE;
```

Строка 4 содержит диапазон значений. Если значение k принадлежит числовому отрезку [2, 5], то будут выполнены строки 05 и 06.

В строке 7 использован список значений. Строка 8 выполнится, если значение k будет равно 6 или будет принадлежать числовому отрезку [9, 20].

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

При задании списка значений необходимо выполнять следующие условия:

– наборы значений внутри одного CASE не должны пересекаться;

– при указании диапазона значений начало диапазона должно быть меньше его конца.

В таблице (см. Таблица 9) приведены примеры кода записи правильной и неправильной записи конструкции CASE.

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

Таблица 9 - Запись конструкции CASE

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

06: i := 0; 07: 6, 20..9: 08: k := k – 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE; В строке 7 диапазон значений задан неправильно.	06: i := 0; 07: 6, 9..20: 08: k := k – 1; 09: ELSE 10: k := 0; 11: i := 1; 12: END_CASE;

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

18.2. Instruction List (IL)

IL (Instruction List) представляет собой текстовый язык программирования низкого уровня, который очень похож на Assembler, но к конкретной архитектуре процессора не привязан. Он позволяет описывать функции, функциональные блоки и программы, а также шаги и переходы в языке SFC. Одним из ключевых преимуществ IL является его простота и возможность добиться оптимизированного кода для реализации критических секторов программ. Особенности IL делают его неудобным для описания сложных алгоритмов с большим количеством разветвлений.

18.2.1. Операторы языка

Основа языка программирования IL, как и в случае Assembler, это переходы по меткам и аккумулятор. В аккумулятор загружается значения переменной, а дальнейшее выполнение алгоритма представляет собой извлечение значения из аккумулятора и совершение над ним операций. Далее в таблице (см. Таблица 10) приведены операторы языка IL.

Таблица 10 - Операторы языка IL

Оператор	Описание
LD	Загрузить значение операнда в аккумулятор
LDN	Загрузить обратное значение операнда в аккумулятор
ST	Присвоить значение аккумулятора операнду
STN	Присвоить обратное значение аккумулятора операнду
S	Если значение аккумулятора TRUE, установить логический операнд
R	Если значение аккумулятора FALSE, сбросить логический операнд
AND	Поразрядное И аккумулятора и операнда

Оператор	Описание
ANDN	Поразрядное И аккумулятора и обратного операнда
OR	Поразрядное ИЛИ аккумулятора и операнда
ORN	Поразрядное ИЛИ аккумулятора и обратного операнда
XOR	Поразрядное разделительное ИЛИ аккумулятора и операнда
XORN	Поразрядное разделительное ИЛИ аккумулятора и обратного операнда
NOT	Поразрядная инверсия аккумулятора
ADD	Сложение аккумулятора и операнда, результат записывается в аккумулятор
SUB	Вычитание операнда из аккумулятора, результат записывается в аккумулятор
MUL	Умножение аккумулятора на операнд, результат записывается в аккумулятор
DIV	Деление аккумулятора на операнд, результат записывается в аккумулятор
GT	Значение аккумулятора сравнивается со значением операнда(>(greater than)). Значение (TRUE или FALSE) записывается в аккумулятор
GE	Значение аккумулятора сравнивается со значением операнда(>=greater than or equal)). Значение (TRUE или FALSE) записывается в аккумулятор
EQ	Значение аккумулятора сравнивается со значением операнда (=(equal)). Значение (TRUE или FALSE) записывается в аккумулятор
NE	Значение аккумулятора сравнивается со значением операнда (<>(not equal). Значение (TRUE или FALSE) записывается в аккумулятор
LE	Значение аккумулятора сравнивается со значением операнда (<=(less than or equal to)). Значение (TRUE или FALSE) записывается в аккумулятор
LT	Значение аккумулятора сравнивается со значением операнда (<(less than)). Значение (TRUE или FALSE) записывается в аккумулятор
JMP	Переход к метке
JMPC	Переход к метке при условии, что значение аккумулятора TRUE
JMPCN	Переход к метке при условии, что значение аккумулятора FALSE
CAL	Вызов программного или функционального блока
CALC	Вызов программного или функционального блока при условии, что значение аккумулятора TRUE
CALCN	Вызов программного или функционального блока при условии, что значение аккумулятора FALSE
RET	Выход из POU и возврат в вызывающую программу
RETC	Выход из POU и возврат в вызывающую программу при условии, что значение аккумулятора TRUE

Оператор	Описание
RETCN	Выход из POU и возврат в вызывающую программу при условии, что значение аккумулятора FALSE

18.3. Пример программы на языке IL

На рисунке (см. Рисунок 39) приведён пример программы на языке IL, которая эквивалентна следующему логическому выражению C = A AND NOT B:

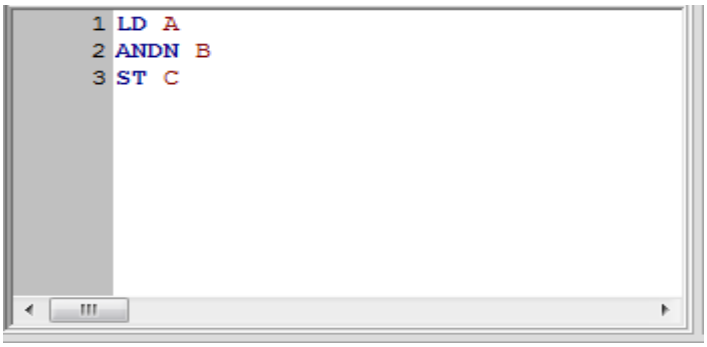

Рисунок 39 - Пример программы на языке IL

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

18.4. Function Block Diagram (FBD)

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

18.4.1. Основные понятия и конструкции языка

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

Переменные бывают входные, выходные и входные/выходные. На рисунке (см. Рисунок 40) показаны: входная переменная – «in_var», выходная переменная – «out_var» и входная/выходная переменная – «in_out_var».

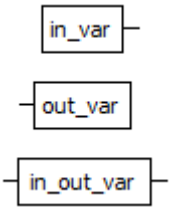

Рисунок 40 - Изображение переменной в языке FBD

Графическое изображение функции приведено на рисунке (см. Рисунок 41). С левой стороны располагаются входы (IN1 и IN2), с правой стороны выходы (OUT).

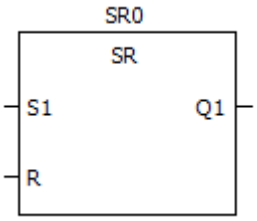

Рисунок 41 - Графическое изображение функции

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

Соответственно, переменные соединяются с входными и выходными параметрами функций и функциональных блоков. Входные переменные могут быть соединены только с входными параметрами функции или функционального блока, выходные переменные – только с выходными параметрами функции или функционального блока, входные/выходные переменные – как входами, так и с выходами функции или функционального блока. Также выходной параметр одной функции или функционального блока может быть напрямую соединён с входным параметром другого.

Все функциональные блоки могут быть вызваны с дополнительными (необязательными) формальными параметрами: EN (входом) и ENO (выходом). Пример такого функционального блока приведён на рисунке (см. Рисунок 42).

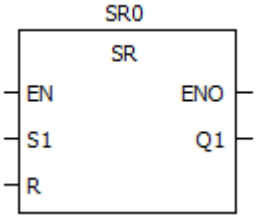

Рисунок 42 - Изображение элементарного функционального блока с параметрами EN/ENO

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

Для более компактного соединения входов и выходов различных функций и функциональных блоков используются элементы «Соединение», показанные на рисунке (см. Рисунок 43).

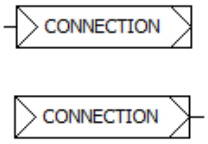

Рисунок 43 - Изображение соединений в языке FBD

Они бывают двух видов: входное соединение и выходное выходные соединение. Основная задача соединений – передать значение из одного выхода на другой вход без прямого соединения выхода и входа. На рисунке (см. Рисунок 61) показан пример, в котором выходное значение OUT функции BOOL_TO_INT передаётся на вход IN2 функции ADD:

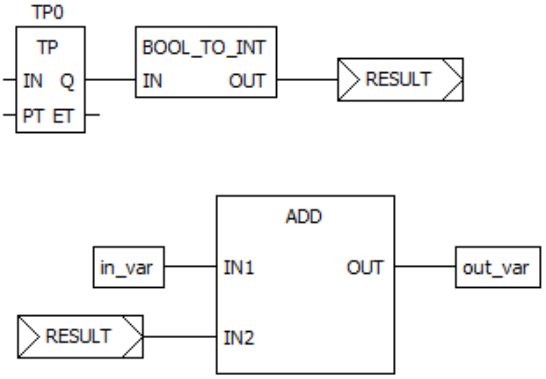

Рисунок 44 - Пример использования соединения на FBD диаграмме

18.5. Пример программы на языке FBD

На рисунке (см. Рисунок 45) приведена FBD диаграмма, состоящая из следующих функциональных блоков: SR0, AND, TP0.

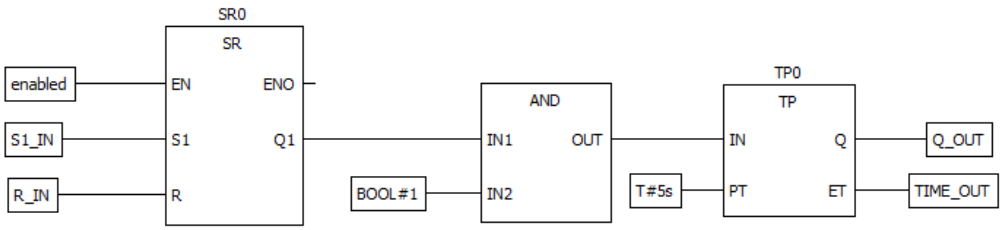

Рисунок 45 - Пример FBD диаграммы

Функциональный блок SR0 представляет собой бистабильный SR-триггер. У него имеются входы S1, R1 и выход Q1, а также дополнительный вход EN и выход ENO, позволяющие включать и выключать выполнение SR0. Выход Q1 соединён с входом IN1 блока AND, представляющий собой «Логическое И». Вход IN2 типа BOOL соединён с литералом «BOOL#1», который всегда положительный. Выход OUT блока AND соединён с входом IN функционального блока TP0, представляющий собой повторитель импульсов. Вход PT типа TIME, соединён с литералом «T#5s», который задаёт значение 5 секунд.

Если после запуска выполнения данного функционального блока enabled равно True и переменная S1_IN тоже True, функциональный блок SR0 начинает выполняться. На выходе OUT функционального блока AND будет значение True как только Q1 у SR0 будет равен True. Следовательно, как только OUT становится True вход IN функционального блока TP0 принимает тоже True и начинается отсчёт таймера ET.

Пока данный таймер не достигнет значения PT выход Q у функционального блока TP0 будет равен True. При достижении таймером ET значения PT, т.е. через 5 секунд выход Q становится False.

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

11. LADDER DIAGRAM (LD)

LD (Ladder Diagram) – графический язык, основанный на принципах релейно-контактных схем (элементами релейно-контактной логики являются: контакты, обмотки реле, вертикальные и горизонтальные перемычки и др.) с возможностью использования большого количества различных функциональных блоков. Достоинствами языка LD являются: представление программы в виде электрического потока (близко специалистам по электротехнике), наличие простых правил, использование только булевых выражений. На рисунке (см. Рисунок 46) приведён пример программы на языке LD (слева) и её эквивалент в виде электрической цепи с реле и выключателями (справа).

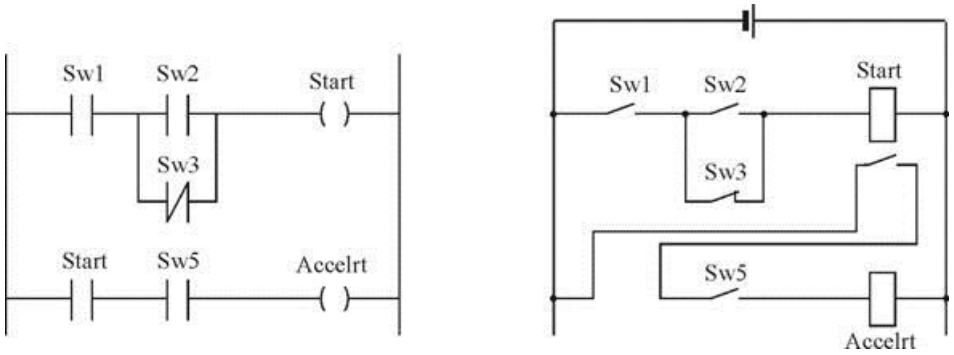

Рисунок 46 - Программа на языке LD (слева) и ее эквивалент в виде электрической (справа)

Схемы, реализованные на данном языке, называются многоступенчатыми. Они представляют собой набор горизонтальных цепей, напоминающих ступеньки лестницы, соединяющих вертикальные шины питания.

Объекты языка программирования LD обеспечивают средства для структурирования программного модуля в некоторое количество контактов, катушек. Эти объекты взаимосвязаны через фактические параметры или связи.

Порядок обработки индивидуальных объектов в LD-секции определяется потоком данных внутри секции. Ступени, подключенные к левой шине питания, обрабатываются сверху вниз (соединение к левой шине питания). Ступени внутри секции, которые не зависят друг от друга, обрабатываются в порядке размещения.

19.1. Основные конструкции языка

Слева и справа схема на языке LD ограничена вертикальными линиями – шинами питания. Между ними расположены цепи, образованные контактами и катушками реле, по аналогии с обычными электронными цепями. Слева любая цепь начинается набором контактов, которые посылают слева направо состояние «ON» или «OFF», соответствующие логическим значениям TRUE или FALSE. Каждому контакту соответствует логическая

переменная (типа BOOL). Если переменная имеет значение TRUE, то состояние передается через контакт. Иначе – правое соединение получает значение выключено ("OFF").

Контакты могут быть соединены параллельно, тогда соединение передаёт состояние «логическое ИЛИ». Если контакты соединены последовательно, то соединение передаёт «логическое И».

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

Язык LD позволяет:

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

19.1.1. Контакт

Контактом является LD-элемент, который передаёт состояние горизонтальной связи левой стороны горизонтальной связи на правой стороне. Это состояние – результат булевой AND-операции состояния горизонтальной связи с левой стороны с состоянием ассоциированной переменной или прямого адреса. Контакт не изменяет значения связанной переменой или прямого адреса.

Для нормальных контактов (см. Рисунок 47) состояние левой связи передается в правую связь, если состояние связанного логического фактического параметра TRUE. Иначе, состояние правой связи FALSE.

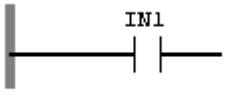

Рисунок 47 - Нормальный контакт

Для инверсных контактов состояние левой связи передаётся в правую связь, если состояние связанного логического фактического параметра FALSE. Иначе, состояние правой связи TRUE.

В контактах для обнаружения нарастания фронта правая связь устанавливается в состояние TRUE, если переход связанного фактического параметра происходит из FALSE в TRUE, и в то же время состояние левой связи TRUE. Иначе, состояние правой связи FALSE.

В контактах для обнаружения спада фронта (см. Рисунок 48) правая связь устанавливается в состояние TRUE, если переход связанного фактического параметра происходит из True в False, и состояние левой связи True в то же время. Иначе, состояние правой связи FALSE.

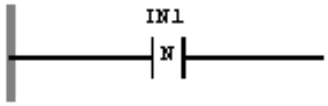

Рисунок 48 - Контакт для обнаружения спада фронта

19.1.2. Катушка

Катушка является LD-элементом, который передаёт состояние горизонтальной связи на левой стороне неизменяемым горизонтальной связи на правой стороне. В этом процессе состояние связанной переменной или прямого адреса будет сохранено.

В нормальных катушках (см. Рисунок 49) состояние левой связи передается в связанный логический фактический параметр и в правую связь.

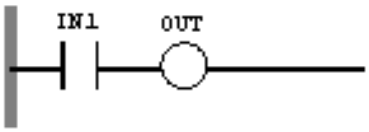

Рисунок 49 - Нормальная катушка

В инвертирующей катушке (см. Рисунок 50) состояние левой связи копируется в правую связь. Инвертированное состояние левой связи копируется в связанный логический фактический параметр. Если связь находится в состоянии FALSE, тогда правая связь тоже будет находиться в состоянии FALSE, и связанный логический фактический параметр будет находиться в состоянии TRUE.

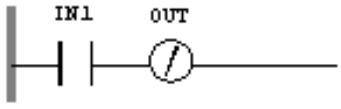

Рисунок 50 - Инвертирующая катушка

В катушке установки (см. Рисунок 51) состояние левой связи копируется в правую связь. Связанный логический фактический параметр устанавливается в состояние TRUE,

если левая связь имеет состояние TRUE, иначе он не изменяется. Связанный логический фактический параметр может сбрасываться только катушкой сброса.

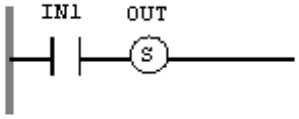

Рисунок 51 - Катушка установки

В катушке сброса (см. Рисунок 52) состояние левой связи копируется в правую связь. Связанный логический фактический параметр устанавливается в состояние FALSE, если левая связь имеет состояние TRUE, иначе он не изменяется. Связанный логический фактический параметр может устанавливаться только катушкой установки.

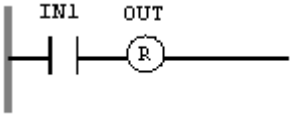

Рисунок 52 - Катушка сброса

В катушке обнаружения нарастания фронта (см. Рисунок 53) состояние левой связи копируется в правую связь. Связанный фактический параметр типа данных BOOL будет установлен в состояние TRUE для цикла программы, если произошел переход левой связи из FALSE в TRUE.

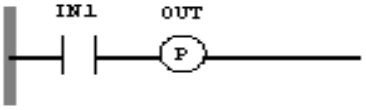

Рисунок 53 - Катушка обнаружения нарастания фронта

В катушке обнаружения спада фронта (см. Рисунок 54) состояние левой связи копируется в правую связь. Связанный фактический параметр типа данных BOOL будет установлен в состояние TRUE для цикла программы, если произошел переход левой связи из TRUE в FALSE.

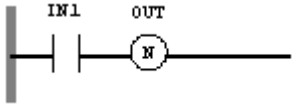

Рисунок 54 - Катушка обнаружения спада фронта

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

19.1.3. Шина питания

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

19.1.4. Пример программы на языке LD

Пример представляет собой реализацию логического выражения:

C = A AND NOT B

При создании LD диаграмм можно использовать только переменные типа BOOL. Добавим новый контакт и привяжем его к имени A (имени переменной). Далее добавляется шина питания слева, шина питания справа, нормальный контакт, инверсный контакт и нормальная катушка. Нормальный контакт ассоциируется с переменной A, инверсный контакт с переменой B, нормальная катушка с переменной C. Далее это всё последовательно соединяется (см. Рисунок 72) и результатом является программа, написанная на языке LD, реализующая логическое выражение:

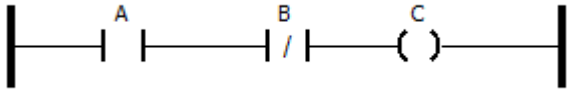

Рисунок 55 - Пример LD диаграммы, реализующей логическое выражение

C = A AND NOT B

12. SEQUENTIAL FUNCTION CHART (SFC)

SFC (Sequential Function Chart) расшифровывается как «Последовательность функциональных диаграмм», и является одним из языков стандарта IEC 61131-3. SFC позволяет легко описывать последовательность протекания процессов в системе.

SFC осуществляет последовательное управление процессом, базируясь на системе условий, передающих управления с одной операции на другую. Язык SFC состоит из конечного числа базовых элементов, которые используются как блоки для построения целостного алгоритма протекания программы.

20.1. Основные понятия языка SFC

Язык SFC использует следующие структурные элементы для создания программы: шаг (и начальный шаг), переход, блок действий, прыжок и связи типа дивергенция и конвергенция.

После вызова программного модуля, описанного языком SFC, первым выполняется начальный шаг. Шаг, выполняемый в данный момент, называется активным. Действия, связанные с активным шагом, выполняются один раз в каждом управляющем цикле. В режиме выполнения активные шаги выделяются цветом. Следующий за активным шагом шаг станет активным, только если в переходе между этими шагами условие будет истинно.

В каждом управляющем цикле будут выполнены действия, содержащиеся в активных шагах. Далее проверяются условия перехода, и, возможно, уже другие шаги становятся активными, но выполняться они будут уже в следующем цикле.

Далее описывается каждый элемент SFC диаграммы.

20.1.1. Шаг

Наиболее важным элементом языка SFC является шаг, который описывает одну операцию. Шаг изображается в виде прямоугольника с собственным именем внутри (см. Рисунок 56).

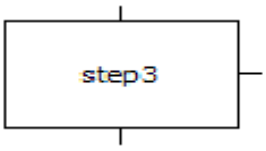

Рисунок 56 - Графическое представление «Шага» языка SFC

У каждого шага может быть 3 контакта. Сверху и снизу для соединения с переходом и справа для соединения с блоком действий. Шаг предваряется переходом, который определяет условие для активации данного шага в процессе выполнения программы и отображается в виде горизонтальной черты на ветви диаграммы процесса с указанием имени и условия. Два

шага никогда не могут быть соединены непосредственно, они должны всегда отделяться переходом (см. Рисунок 57).

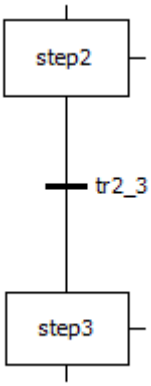

Рисунок 57 - Шаги «step2» и «step3», соединённые переходом «tr2_3»

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

20.1.2. Переход

Между шагами находятся так называемые переходы. Условием перехода может быть логическая переменная или константа, логический адрес или логическое выражение, описанное на любом языке. Условие может включать серию инструкций, образующих логический результат, в виде ST выражения, например:

(i<= 100) AND b

либо на любом другом языке.

На рисунке (см. Рисунок 58) приведён пример перехода между шагом «Step3» и «Step5» с именем «transition4».

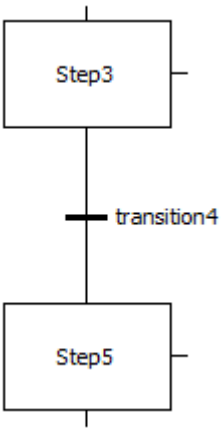

Рисунок 58 - Переход между шагами «Step3» и «Step5» с предопределённым условием «transition4»

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

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

На рисунке (см. Рисунок 59) представлен переход между шагами «Step6» и «Step7» в виде обычного условия: level > 10

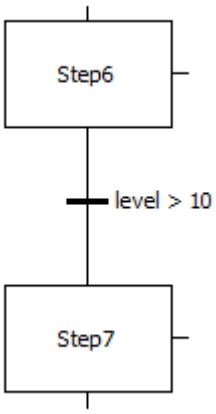

Рисунок 59 - Переход между шагами «step6» и «step7» с предопределённым условием «transition4»

На рисунке (см. Рисунок 60) представлен переход между шагами «Step8» и «Step9» в виде значения логического выражения «AND» на языке FBD:

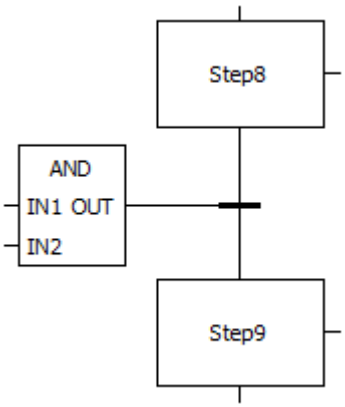

Рисунок 60 - Переход между шагами «step8» и «step9», заданный «логическим И» на языке FBD

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

20.1.3. Блок действий

Каждый шаг имеет нулевое или большее количеством действий, объединённых, как правило, на диаграмме, в блок действий. На рисунке (см. Рисунок 61) показан примера шага «evaluateStep» и связанный с ним блок действий.

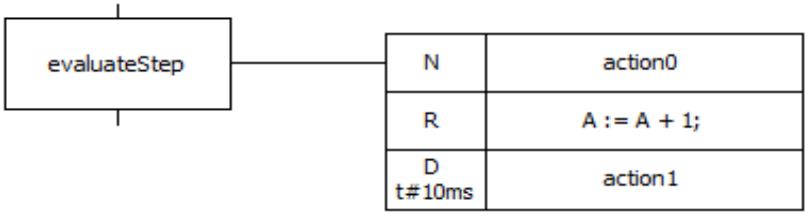

Рисунок 61 - Шаг «evaluateStep» и связанный с ним блок действий, содержащий 3 действия

Блок действий определяет операции, которые должны выполняться при активации (выполнении) шага. Шаги без связанного блока действий идентифицируются как ждущий шаг. Блок действий может состоять из предопределённых действий. Каждому предопределённому действию присваивается имя (это «action0» и «action1»). Одно действие может использоваться сразу в нескольких шагах. Действие может выполняться непрерывно, пока активен шаг, либо единожды. Это определяется специальными квалификаторами, описание которых приведено в таблице (см. Таблица 11). Квалификаторы также могут ограничивать время выполнения каждого действия в шаге.

Таблица 11 - Квалификаторы действий SFC диаграммы

Имя квалификатора	**Поведение блока действия**
D	Действие начинает выполняться через некоторое заданное время (если шаг ещё 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 остаётся 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 ещё 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

20.1.4. «Прыжок» – переход на произвольный шаг

Шаг может быть также заменён «прыжком». Последовательности шагов всегда ассоциируются с прыжком к другому шагу той же самой последовательности шагов. Это означает, что они выполняются циклически. Переход на произвольный шаг – это соединение на шаг, имя которого указано под знаком «прыжка». Такие переходы нужны для того, чтобы избежать пересекающихся и идущих вверх соединений. На рисунке (см. Рисунок 62) показана SFC диаграмма, содержащая два «прыжка».

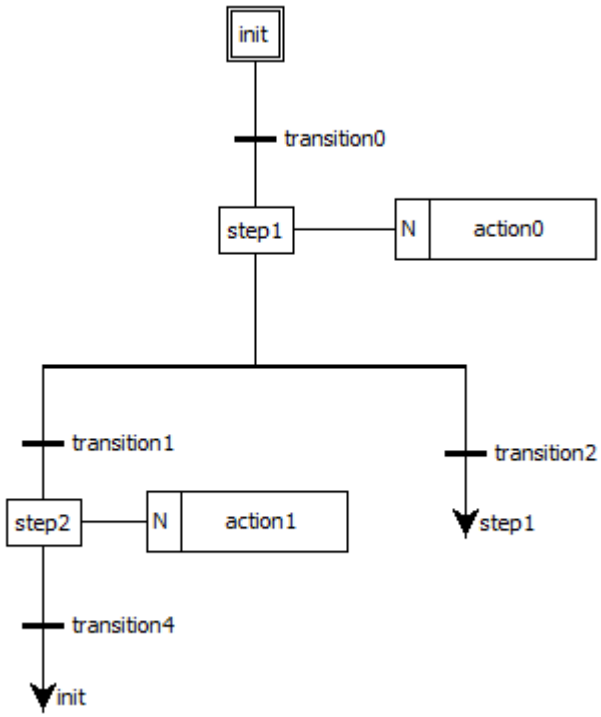

Рисунок 62 - SFC диаграмма, содержащая «прыжки»

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

20.1.5. Дивергенция и конвергенция

Дивергенция – это множественное соединение в направлении от одного шага к нескольким переходам. Активируется только одна из ветвей. Условия, связанные с различными переходами в начале дивергенции, не являются взаимоисключающими по умолчанию. Взаимоисключение должно быть явно задано в условиях переходов, чтобы гарантировать, что во время выполнения программы активируется одна конкретная ветвь. Пример дивергенции на SFC диаграмме приведён на рисунке (см. Рисунок 63) и выделен красным цветом:

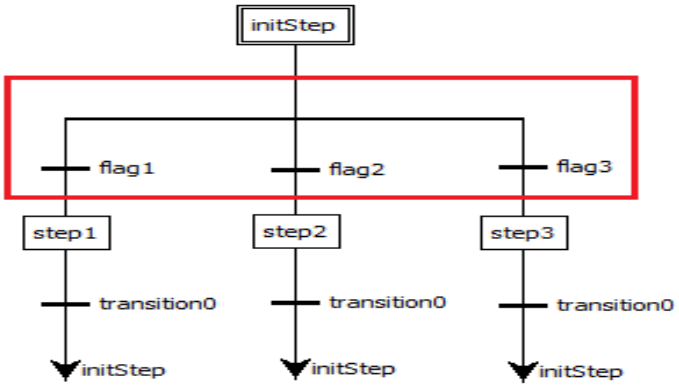

Рисунок 63 - Дивергенция на SFC диаграмме

Конвергенция – это множественное соединение, направленное от нескольких переходов к одному и тому же шагу. Она обычно используется для группировки ветвей SFC – программы, которые берут начало из одинарной дивергенции. Пример конвергенции на SFC диаграмме приведён на рисунке (см. Рисунок 64) и выделен красным цветом:

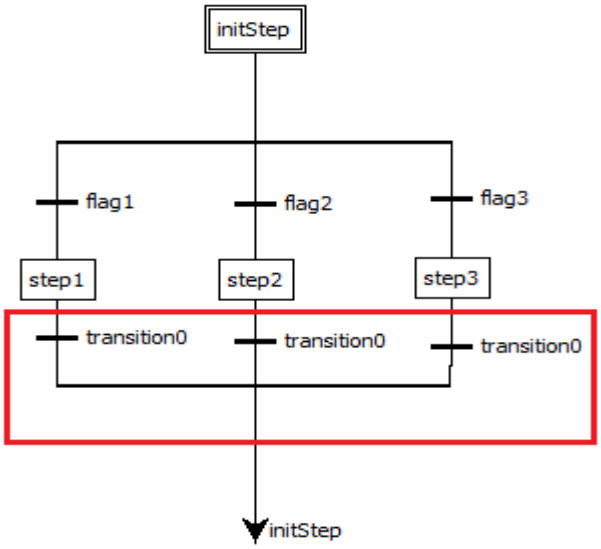

Рисунок 64 - Конвергенция на SFC диаграмме

Параллельная дивергенция – это множественное соединение, направленное от одного перехода к нескольким шагам. Она соответствует параллельному выполнению операций процесса. Пример параллельной дивергенции на SFC диаграмме приведён на рисунке (см. Рисунок 65) и выделен красным цветом:

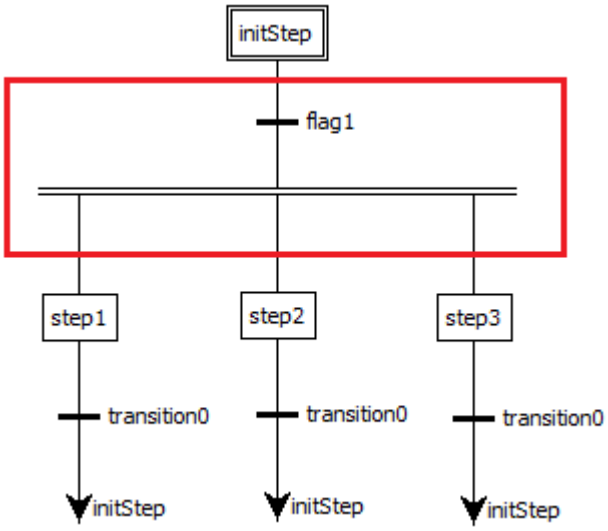

Рисунок 65 - Параллельная дивергенция на SFC диаграмме

Параллельная конвергенция – это соединение нескольких шагов к одному и тому же переходу. Обычно она используется для группирования ветвей, взявших начало дивергенции. Пример параллельной конвергенции на SFC диаграмме приведён рисунке (см. Рисунок 66) и выделен красным цветом:

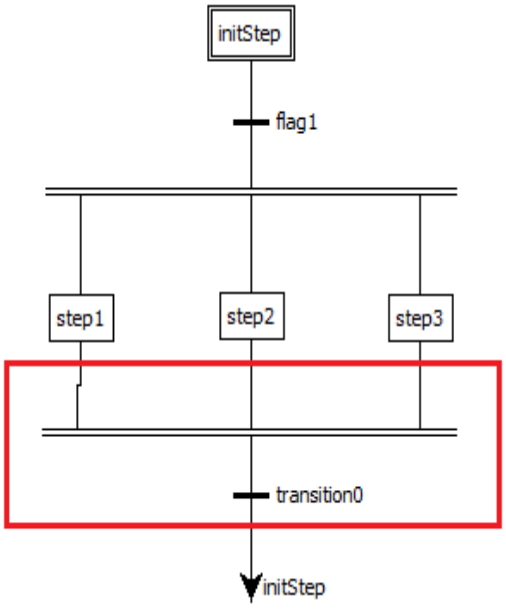

Рисунок 66 - Параллельная конвергенция на SFC диаграмме

20.2. Пример программы на языке SFC

На рисунке (см. Рисунок 67) приведён пример SFC диаграммы, состоящей из начального шага «initStep», шагов «firstStep» и «secondStep» и 3 перехода.

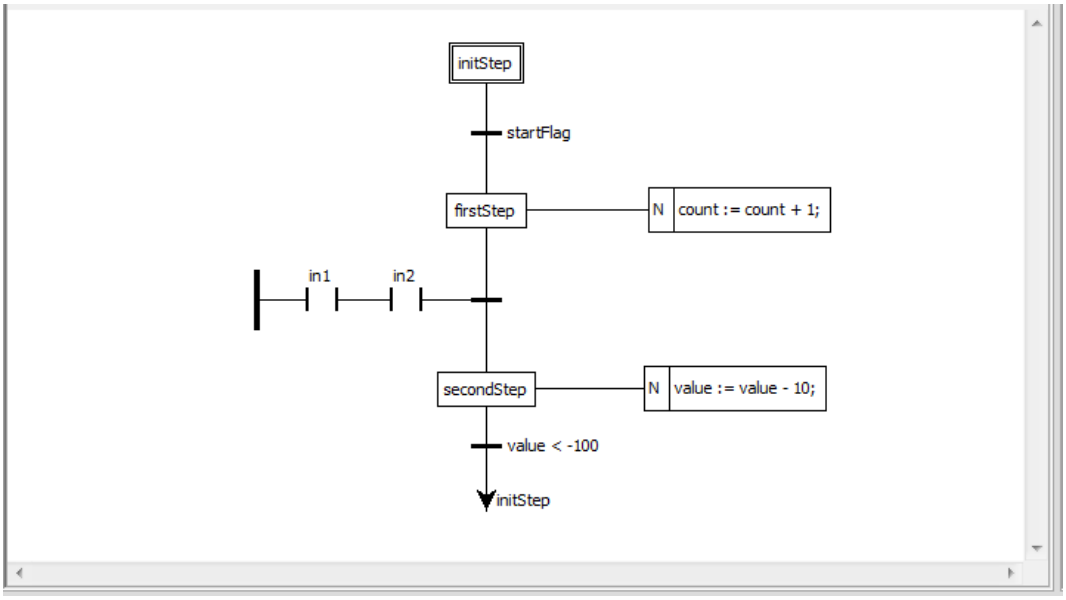

Рисунок 67 - SFC диаграмма

Переход «startFlag» представляет обычную переменную типа BOOL и полностью зависит от её значения. Переход между «firstStep» и «secondStep» зависит от LD диаграммы с двумя катушками, ассоциированными с переменными типа BOOL: «in1» и «in2». Переход активируется только в том случае, если «in1» и «in2» будут TRUE. Переход между «secondStep» и прыжком на initStep активирован, когда значение переменной «value» меньше -100.

Во время действия «firstStep» выполняется увеличение переменной count на 1. Во время действия «secondStep» из переменной «value» вычитается 10.

13. БИБЛИОТЕКА ФУНКЦИЙ И ФУНКЦИОНАЛЬНЫХ БЛОКОВ

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

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

21.1. Стандартные функциональные блоки

21.1.1. Бистабильный SR-триггер

Данный функциональный блок представляет собой бистабильный SR-триггер, с доминирующим входом S (Set) (см. Рисунок 85).

Выход Q1 становится "1", когда вход S1 становится "1". Это состояние сохраняется, даже если S1 возвращается обратно в "0".

Выход Q1 возвращается в "0", когда вход R становится "1".

Если входы S1 и R находятся в "1" одновременно, доминирующий вход S1 установит выход Q1 в "1".

Когда функциональный блок вызывается первый раз, начальное состояние Q1 это "0".

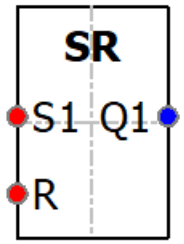

Рисунок 85 – Бистабильный SR – триггер

21.1.2. Бистабильный RS-триггер

Данный функциональный блок представляет собой бистабильный RS-триггер, с доминирующим входом R (Reset) (см. Рисунок 86).

Выход Q1 становится "1", когда вход S становится "1". Это состояние сохраняется, даже если S возвращается обратно в "0".

Выход Q1 возвращается в "0", когда вход R1 становится "1".

Если входы S и R1 находятся в "1" одновременно, доминирующий вход R1 установит выход Q1 в "0".

Когда функциональный блок вызывается первый раз, начальное состояние Q1 это "0".

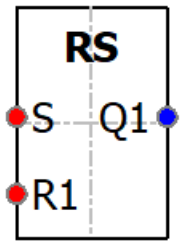

Рисунок 86 – Бистабильный RS – триггер

21.1.3. SEMA – Семафор

Данный функциональный блок представляет собой семафор, определяющий механизм, позволяющий элементам программы иметь взаимоисключающий доступ к определённым ресурсам (см. Рисунок 87).

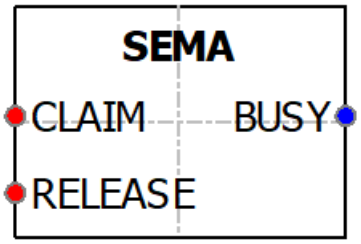

Рисунок 87 - Семафор

21.1.4. R_TRIG – Индикатор нарастания фронта

Данный функциональный блок представляет собой индикатор нарастания фронта, который генерирует на выходе одиночный импульс при нарастании фронта сигнала (см. Рисунок 88).

Выход Q становится "1", если происходит переход из "0" в "1" на входе CLK . Выход остаётся в состоянии "1" от одного выполнения блока до следующего (один цикл); затем выход возвращается в "0".

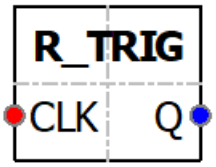

Рисунок 88 – Индикатор нарастания фронта

21.1.5. F_TRIG – Индикатор спада фронта

Данный функциональный блок представляет собой индикатор спада фронта, который генерирует на выходе одиночный импульс при спаде фронта сигнала (см. Рисунок 89).

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

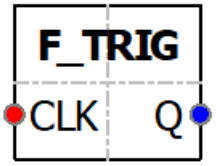

Рисунок 89 – Индикатор спада фронта

21.1.6. CTU – инкрементный счётчик

Данный функциональный блок представляет собой инкрементный счётчик (см. Рисунок 90).

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

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

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

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

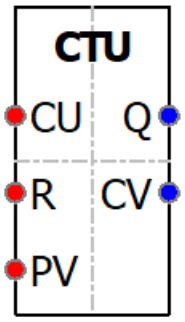

Рисунок 90 – Инкрементный счетчик

21.1.7. CTD – декрементный счётчик

Данный функциональный блок представляет собой декрементный счётчик (см. Рисунок 91).

Сигнал "1" на входе LD вызывает присваивание значения на входе PV выходу CV . При каждом переходе из "0" в "1" на входе CD значение CV уменьшается на 1. Когда CV <= 0, выход Q принимает значение "1".

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

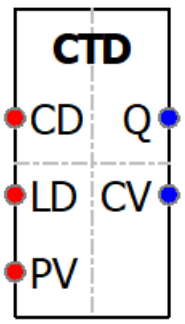

Рисунок 91 – Декрементный счетчик

21.1.8. CTUD – реверсивный счётчик

Данный функциональный блок представляет собой реверсивный счётчик (см. Рисунок 92).

Сигнал "1" на входе R вызывает присваивание значения "0" выходу CV . Сигнал "1" на входе LD вызывает присваивание значения на входе PV выходу CV . При каждом переходе из

"0" в "1" на входе CU значение CV увеличивается на 1. При каждом переходе из "0" в "1" на входе CD значение CV уменьшается на 1.

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

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

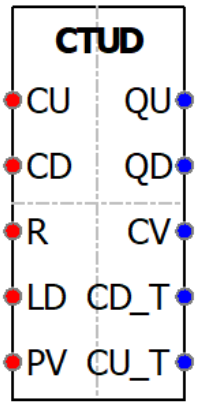

Рисунок 92 – Реверсивный счетчик

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

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

21.1.9. TP – повторитель импульсов

Данный функциональный блок представляет собой повторитель импульсов и используется для генерирования импульса с заданной продолжительностью (см. Рисунок 93).

Если IN становится "1", Q становится "1", и начинается отсчёт внутреннего времени (ET). Если внутреннее время достигает значения PT, Q становится "0" (независимо от IN). Отсчёт внутреннего времени останавливается/сбрасывается, если IN становится "0". Если внутреннее время не достигло значения PT, импульс IN не влияет на внутреннее время. Если внутреннее время достигло значения PT, и IN равен "0", отсчёт внутреннего времени останавливается/сбрасывается, и Q становится "0".

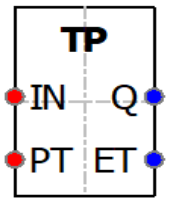

Рисунок 93 – Повторитель импульсов

21.1.10. TON – таймер с задержкой включения

Данный функциональный блок представляет собой таймер с задержкой включения. Он запускается, когда состояние сигнала на входе меняется от 0 к 1 и устанавливает на выходе 1 по истечении заданного времени (см. Рисунок 94).

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

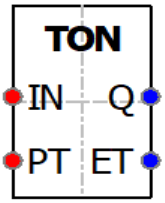

Рисунок 94 – Таймер с задержкой включения

21.1.11. TOF – таймер с задержкой отключения

Данный функциональный блок представляет собой таймер с задержкой отключения. Он запускается, когда состояние сигнала на входе меняется от 1 к 0 и устанавливает на выходе 0 по истечении заданного времени (см. Рисунок 95).

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

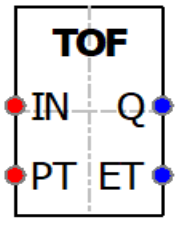

Рисунок 95 – Таймер с задержкой отключения

21.2. Дополнительные функциональные блоки

21.2.1. RTC – часы реального времени

Данный функциональный блок представляет собой часы реального времени и имеет много вариантов использования, включая добавление временных отметок, для установки даты и времени в формируемых отчетах, в аварийных сообщениях и т. Д (см. Рисунок 96).

Вход PDT (Preset DT) предназначен для установки времени. Часы начинают отсчёт времени от значения PDT.

Выход Q (BOOL) повторят значение EN.

Выход CDT (Current DT) дает текущее значение даты и времени.

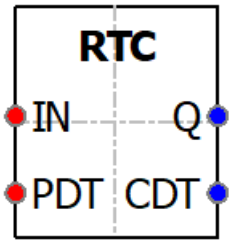

Рисунок 96 – Часы реального времени

21.2.2. INTEGRAL – Интеграл

Функциональный блок интеграл интегрирует входное значение XIN по времени (см. Рисунок 97).

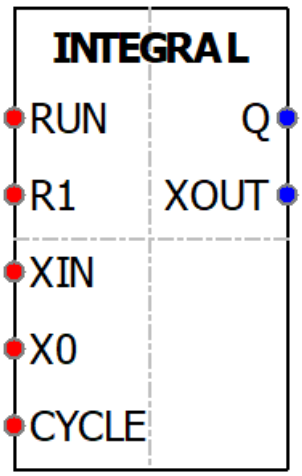

Рисунок 97 – Интеграл

21.2.3. DERIVATIVE – Производная

Функциональный блок производная выдаёт значение XOUT пропорционально скорости изменения входного параметра XIN (см. Рисунок 98).

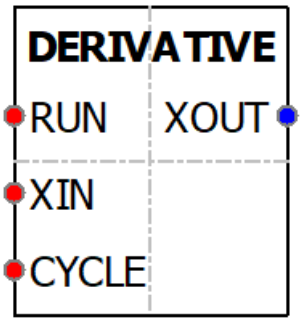

Рисунок 98 – Производная

21.2.4. PID – Пропорционально-интегрально-дифференциальный регулятор

Данный функциональный блок представляет собой устройство в цепи обратной связи, используемое в системах автоматического управления для формирования управляющего сигнала. ПИД-регулятор формирует управляющий сигнал, являющийся суммой трёх слагаемых, первое из которых пропорционально входному сигналу, второе – интеграл входного сигнала, третье – производная входного сигнала (см. Рисунок 99).

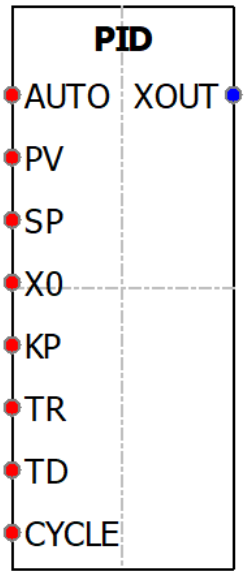

Рисунок 99 – Пропорционально-интегрально-дифференциальный регулятор

21.2.5. HYSTERESIS – гистерезис

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

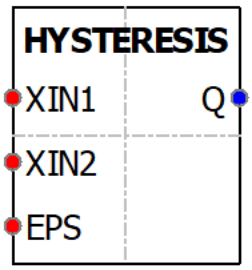

Рисунок 100 – Гистерезис

14. ПРЕОБРАЗОВАНИЯ ТИПОВ

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

Все операции преобразования типа имеют один вход (IN) и один выход (OUT).

15. ЧИСЛОВЫЕ ОПЕРАЦИИ

Все числовые операции имеют один вход (IN) и один выход (OUT).

ABS – модуль числа

Данная функция возвращает модуль входного числа IN.

SQRT – квадратный корень

Данная функция возвращает в OUT квадратный корень входного числа IN.

LN – натуральный логарифм

Данная функция возвращает значение натурального логарифма от IN.

LOG – логарифм по основанию 10

Данная функция возвращает значение логарифма по основанию 10 от IN.

EXP – возведение в степень экспоненты

Данная функция возвращает значение экспоненты, возведённой в степень IN.

SIN – синус

Данная функция возвращает значение синуса IN.

COS – косинус

Данная функция возвращает значение косинуса IN.

TAN – тангенс

Данная функция возвращает значение тангенса IN.

ASIN – арксинус

Данная функция возвращает значение арксинуса IN.

ACOS – арккосинус

Данная функция возвращает значение арккосинуса IN.

ATAN – арктангенс

Данная функция возвращает значение арктангенса IN.

16. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Все арифметические операции имеют как минимум два входа (IN1, IN2, […, INn]) и один выход (OUT). (Кроме MOVE, у которой один вход и один выход (IN), и один выход (OUT).)

ADD – сложение

Данная функция возвращает результат сложения IN1 и IN2.

MUL – умножение

Данная функция возвращает результат умножения IN1 и IN2.

SUB – вычитание

Данная функция возвращает результат вычитания из IN1 значения IN2.

DIV – деление

Данная функция возвращает результат деления IN1 на IN2.

MOD – остаток от деления

Данная функция возвращает остаток от деления IN1 на IN2.

EXPT – возведение в степень

Данная функция возвращает значение IN1 возведённое в степень IN2.

MOVE – присвоение

Данная функция возвращает значение IN.

17. ВРЕМЕННЫЕ ОПЕРАЦИИ

Все временные операции имеют два входа (IN1, IN2) и один выход (OUT).

ADD_TIME – сложение переменных типа TIME

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

ADD_TOD_TIME – сложение времени дня TOD с интервалом времени TIME

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

ADD_DT_TIME – прибавление промежутка времени TIME к моменту времени DT

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

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

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

MULTIME – умножение времени TIME на число

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

SUB_TIME – разность двух значений типа TIME

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

SUB_DATE_DATE – разность двух значений типа DATE

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

SUB_TOD_TIME – вычитание из времени дня TOD интервала времени TIME

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

SUB_DT_TIME – вычитание из момента времени DT промежутка времени TIME

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

соответствующим значением и бит двоичного результата (BR) слова состояния устанавливается в "0".

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

DIVTIME – деление времени TIME на число

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

18. ОПЕРАЦИИ СМЕЩЕНИЯ БИТ

Все операции смещения бит имеют два входа (IN, N) и один выход (OUT).

SHL – арифметический сдвиг влево

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

SHR – арифметический сдвиг вправо

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

ROR – циклический сдвиг направо

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

ROL – циклический сдвиг влево

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

19. ПОБИТОВЫЕ ОПЕРАЦИИ

Все побитовые операции имеют как минимум два входа (IN1, IN2, […, INn]) и один выход (OUT). (Кроме NOT, у которой один вход и один выход (IN), и один выход (OUT).)

AND – побитовое И

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

OR – побитовое ИЛИ

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

XOR – побитовое исключающее ИЛИ

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

NOT – побитовая инверсия

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

20. ОПЕРАЦИИ ВЫБОРА

SEL – выбор из двух значений

Данная функция возвращает один из двух аргументов IN1 или IN2 в зависимости от значения аргумента G. Если G = 0, то результат равен X1, иначе X2.

MAX – максимум

Данная функция возвращает максимум из входных аргументов IN1 и IN2.

MIN – минимум

Данная функция возвращает минимум из входных аргументов IN1 и IN2.

LIMIT – ограничитель значения

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

MUX – Мультиплексор (выбор 1 из N)

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

21. ОПЕРАЦИИ СРАВНЕНИЯ

Все операции сравнения имеют как минимум два входа (IN1, IN2, […, INn]) и один выход (OUT).

GT – больше чем

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

GE – больше чем или равно

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

EQ – равенство

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

LT – меньше чем

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

LE – меньше чем или равно

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

NE – не равно

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

22. СТРОКОВЫЕ ОПЕРАЦИИ С ПЕРЕМЕННЫМИ ТИПА STRING

LEN – длина строки

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

LEFT – левая часть строки

Данная функция возвращает в результат из строки IN первые L символов. Если L больше, чем текущая длина переменной типа STRING, то возвращается входное значение. При L = 0 и при пустой строке в качестве входного значения возвращается пустая строка. Если число L отрицательно, то выводится пустая строка. Параметру IN и возвращаемому значению можно
ставить в соответствие только символически определённую переменную.

RIGHT – правая часть строки

Данная функция возвращает в результат из строки IN последние L символов. Если L больше, чем текущая длина переменной STRING, то возвращается входное значение. При L = 0 и при пустой строке в качестве входного значения возвращается пустая строка. Если число L отрицательно, то выводится пустая строка. Параметру IN и возвращаемому значению можно
ставить в соответствие только символически определённую переменную.

MID – середина строки

Данная функция возвращает в результат из строки IN L-символов, начиная с позиции P. Если сумма L и (P–1) превосходит текущую длину переменной типа STRING, то возвращается строка символов, начиная с P-го символа входной строки до её конца. Во всех остальных случаях (P находится вне текущей длины, P и/или L равны нулю или отрицательны) выводится пустая строка. Параметру IN и возвращаемому значению можно ставить в
соответствие только символически определённую переменную.

CONCAT – объединение двух переменных STRING

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

CONCAT_DAT_TOD – объединение (конкатенация) времени

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

INSERT – вставка в переменной STRING

Данная функция возвращает в результат строку IN1, в которую вставлена строка IN2, начиная с позиции P. Если P равно нулю, то вторая строка символов вставляется перед первой строкой символов. Если P больше, чем текущая длина первой строки символов, то вторая строка символов присоединяется к первой. Если P отрицательно, то выводится пустая строка.

Входным параметрам IN1 и IN2 и выходному параметру можно ставить в соответствие только символически определённую переменную.

DELETE – удаление в переменной STRING

Данная функция возвращает в результат строку IN1, в которой удалено L символов, начиная с позиции P. Если L и/или P равны нулю или P больше, чем текущая длина входной строки, то возвращается входная строка. Если сумма L и P больше, чем входная строка символов, то строка символов удаляется до конца. Если L и/или P имеют отрицательное значение, то выводится пустая. Входному параметру IN и выходному параметру можно ставить в соответствие только символически определённую переменную.

REPLACE – замена в переменной STRING

Данная функция возвращает в результат строку IN1, в которой символы, начиная с позиции P, заменены L первыми символами строки IN2. Если L равно нулю, то возвращается первая строка символов. Если P равно нулю или единице, то замена происходит, начиная с 1-го символа (включительно). Если P лежит вне первой строки символов, то вторая строка присоединяется к первой строке. Если L и/или P отрицательны, то возвращается пустая строка. Входным параметрам IN1 и IN2 и выходному параметру можно ставить в соответствие только символически определённую переменную.

FIND – поиск в переменной STRING

Данная функция возвращает в результат номер позиции, в которой находится строка IN2 в строке IN1. Поиск начинается слева, сообщается о первом появлении строки символов. Если вторая строка символов не содержится в первой, то возвращается нуль. Входным параметрам IN1 и IN2 можно ставить в соответствие только символически определённую переменную.

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в документе	Номер документа	Входящий номер сопрово-дительного документа и дата	Подпись	Дата
Изм.	изменен-ных	заменен-ных	новых	аннули-рованных					