SystemePLC Studio

Программное обеспечение SystemePLC Studio

для интеллектуальных реле SystemePLC SR и ПЛК SystemePLC S172

Быстрый старт. Часть 3.

Программирование на языке LD

systeme.ru

Système electric

Содержание

Программирование на языке лестничных диаграмм (LD)

LD (*Ladder Diagram* — «лестничные диаграммы») — один из графических языков программирования для программируемых логических контроллеров (ПЛК), стандартизированных в **МЭК 61131-3**.
Язык был разработан на базе представления релейно-контактных электрических схем (контакты и катушки реле), широко применявшихся в автоматизации до эпохи ПЛК.

Программа на LD в SystemePLC Studio представляется одним или несколькими сегментами программы – **Network**

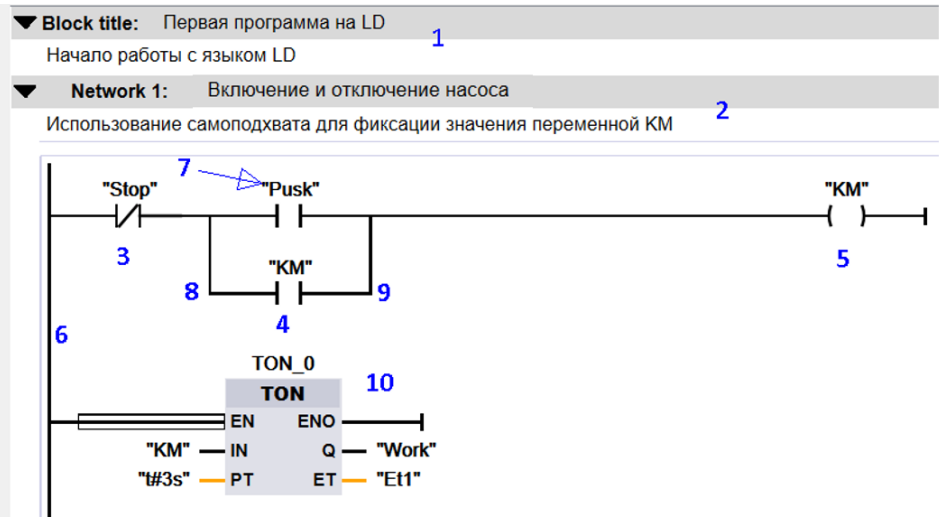
Каждый сегмент программы (**Network**) может включать в себя:

- **Шину питания**, располагающуюся слева вертикально. Все логические цепочки должны начинаться от шины питания
- **Горизонтальные цепи** — логические цепочки из контактов (входных элементов) и катушек (выходных элементов)
- **Контакты,** которые соответствуют входным булевским сигналам или внутренним булевским переменным, чьё значение нужно **прочитать**
- **Катушки**, которые соответствуют выходным или внутренним булевским переменным, чьё значение нужно **записать**

Логика работы считается слева направо и сверху вниз: если цепочка контактов «замкнута», соответствующая катушка активируется.

Сложные функции представляются **функциональными блоками**.

Элементы программы



1. Название и комментарий программного блока
2. Название и комментарий сегмента (Network)
3. Нормально закрытый (НЗ) контакт
4. Нормально открытый (НО) контакт
5. Катушка
6. Шина питания
7. Переменная, связанная с контактом
8. Нисходящая ветвь (ответвление)
9. Восходящая ветвь (ответвление)
10. Функциональный блок

Шина питания
Каждый сегмент LD-программы (**Network**) содержит шину питания, от которой начинается как минимум одна логическая цепочка.
Ответвления от шины питания могут быть использованы для создания параллельных структур внутри сегмента

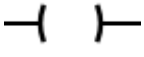
Контакт
Контакты используются для чтения значения входных булевских сигналов или внутренних булевских переменных. Подобно контактам в релейно-контактных схемах, слева направо через контакт течёт «ток», представляя собой результат логического выражения.

В LD-программе в настоящее время доступны следующие типы контактов:

-  Нормально разомкнутый (NO) контакт: передает на выход "1" , если значение связанной с ним переменной равно "1". Если значение связанной переменной равно "0", то и на выходе будет тоже "0"
-  Нормально замкнутый (NC) контакт: передает на выход "1" , если значение связанной с ним переменной равно "0". Если значение связанной переменной равно "0", то на выходе будет "1"

Катушка
Катушки используются для записи значений в выходные или внутренние булевские переменные. Они могут устанавливать или сбрасывать значения булевских переменных в зависимости от результатов логического выражения.

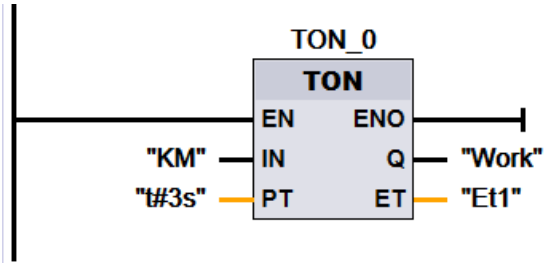
В LD-программе в настоящее время доступны следующие типы катушек:

-  Нормальная катушка: записывает в значение связанной с ней переменной "1" , если результат логического выражения перед катушкой равен "1". Если результат логического выражения перед катушкой равен "0", то и значение связанной с ней переменной тоже будет равно "0".

- —(/)— Инверсная катушка: записывает в значение связанной с ней переменной "1" , если результат логического выражения перед катушкой равен "0". Если результат логического выражения перед катушкой равен "0", то значение связанной с ней переменной будет равно "1".

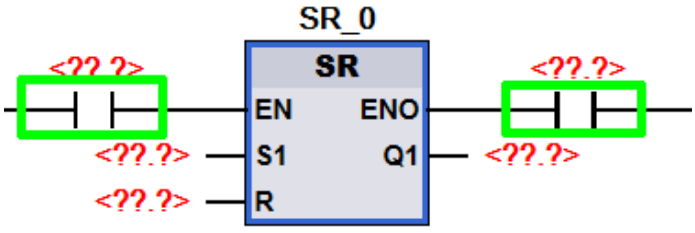
Функциональный блок

Функциональные блоки — это элементы LD-программы, которые реализуют более сложные функции: арифметические, таймеры и счётчики, работа с битами в словах и другие.
В LD-программах могут использоваться функциональные блоки с механизмом EN/ENO: логика функционального блока выполняется только тогда, когда значение сигнала на разрешающем входе "EN" равно "1". Если логика функционального блока выполнена без ошибок, на выходе "ENO" устанавливается "1"; если во время обработки возникает ошибка, выход "ENO" сбрасывается в "0". Так можно выстраивать логические цепочки из ФБ, соединяя, как в FBD, "ENO" с "EN".



Черные выводы ФБ соответствуют типу BOOL, а оранжевые - типам, отличным от BOOL.
Для выводов ФБ типа BOOL можно либо ввести имя переменной, либо выполнить соединение через логическую цепочку.
Для выводов, не относящихся к типу BOOL, можно ввести **только имена** переменных.

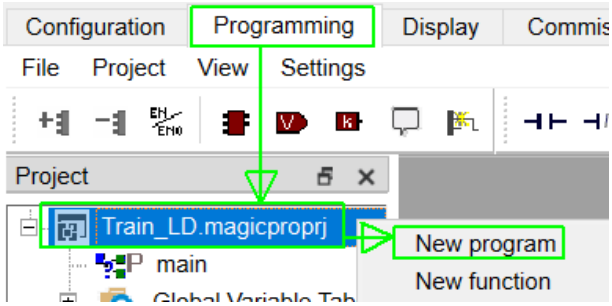
Обратите внимание! Из выводов функционального блока **только первый** булевский поддерживает подключение через логическую цепочку («проводное подключение»). Для остальных выводов нужно использовать имена переменных.



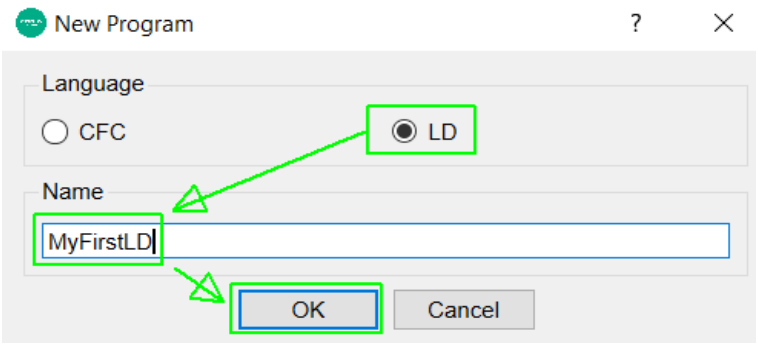
Создание программ на LD

Языком программирования по умолчанию для новых проектов в SystemePLC Studio является FBD. Для программирования с использованием необходимо создать отдельные LD программы. Давайте создадим первую программу на LD – управление контактором с самоподхватом (насос, клапан, задвижка)
Для этого:

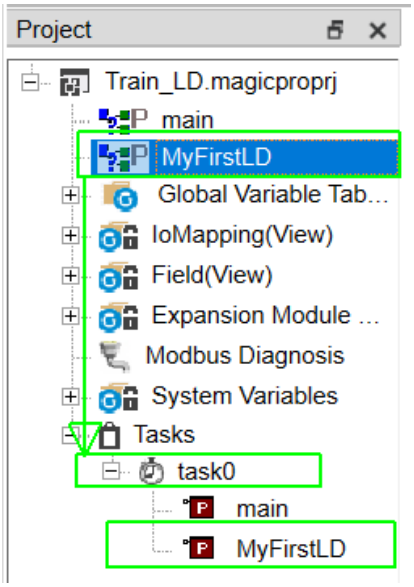
1. Откройте созданный проект. В меню **Programming** щелкните правой кнопкой мыши (далее - ПКМ) по названию проекта и создайте новую программу (**New program**)



2. Выберите язык (Language) - **LD** и дайте имя программе **MyFirstLD** -> OK



3. Перетащите (drag-n-drop) программу **MyFirstLD** в существующую задачу: **Tasks** -> **task0** для того, чтобы она выполнялась.



4. Создайте несколько булевых переменных, для чего в области декларации переменных щёлкните ПКМ -> Insert и введите имена в соответствие с рисунком

	Name	Type	Address	Array
0	Pusk	BOOL	AUTO	No
1	Stop	BOOL	AUTO	No
2	KM	BOOL	AUTO	No
3	Work	BOOL	AUTO	No

Описание панели инструментов

	Insert contact	Вставить НО контакт
	Insert close contact	Вставить НЗ контакт
	Insert coil	Вставить катушку
	Insert block	Вставить блок
	Insert branch	Вставить нисходяшую ветвь
	Insert up branch	Вставить восходящую ветвь
	Insert network	Вставить сегмент (Network)
	Delete network	Удалить сегмент (Network)
	Open all network	Открыть все сегменты
	Close all network	Закрыть все сегменты

Добавление комментариев

В программную секцию и каждый сегмент (Network) можно добавить название и описание (в т.ч. и на русском языке). Использование комментариев упрощает понимание программы и свидетельствует о хорошем стиле программирования

5. Щёлкните справа от надписи **Block title** и добавьте описание программной секции. Щёлкните справа от надписи **Network 1** и добавьте название сегмента. Ниже добавьте комментарий (см. рис.):

Block title: Первая программа на LD

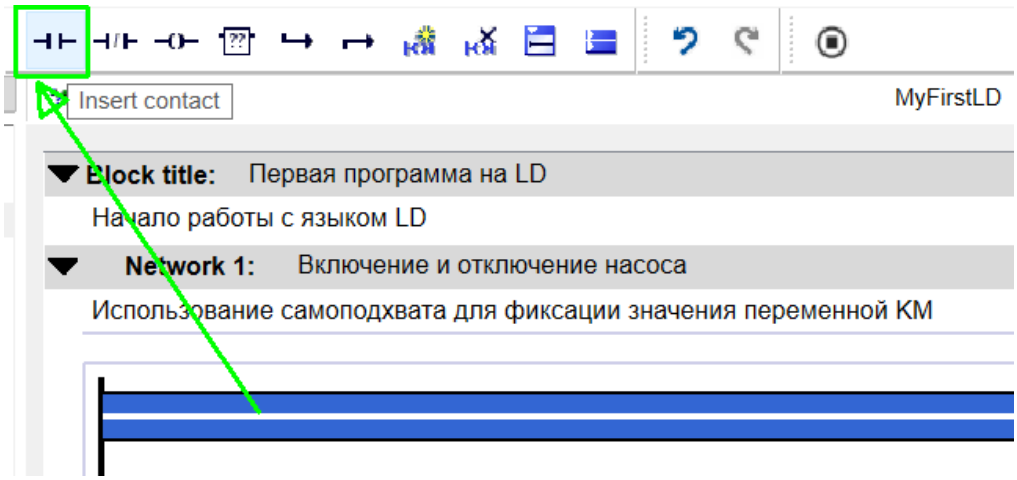
Начало работы с языком LD

Network 1: Включение и отключение насоса

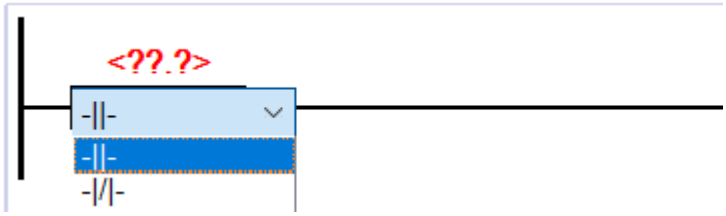
Использование самоподхвата для фиксации значения переменной KM

Добавление и изменение контактов

6. Выберите щелчком мыши позицию в сегменте (Network), куда необходимо вставить контакт – она выделится цветом
7. Щелкните в панели инструментов LD по значку НО контакта

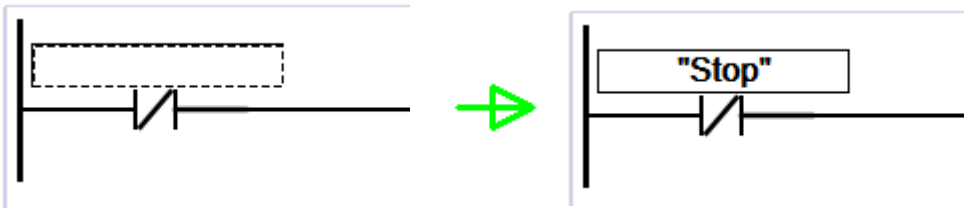


8. Дважды щёлкните по контакту и выберите из выпадающего списка НЗ контакт

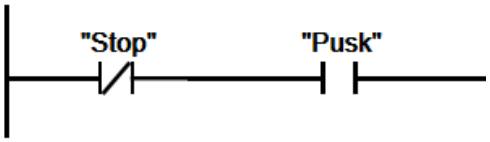


Примечание. Можно сразу выбрать НЗ контакт в п.7. В примере показан пример изменения типа уже существующего контакта

9. Дважды щёлкните над контактом по обозначению **<??.?>**, введите имя переменной чтобы привязать переменную к контакту.
Обратите внимание! Переменная должна быть создана заранее – вы это сделали в п.4

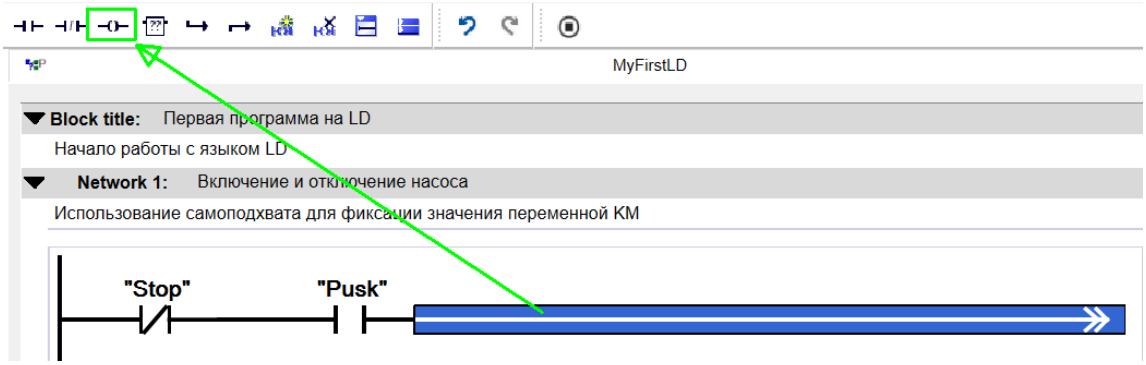


10. Добавьте слева от НЗ контакта НО контакт и привяжите его к переменную Pusk:



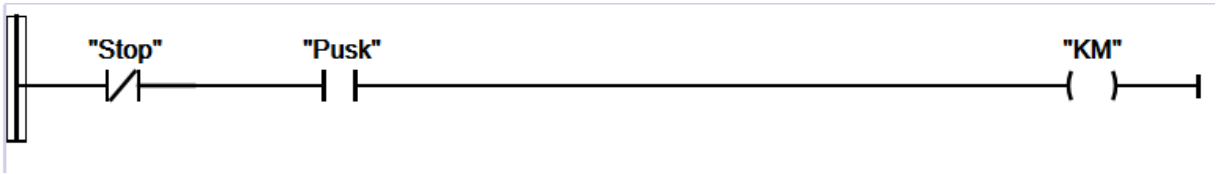
Добавление и изменение катушек

11. Выберите щелчком мыши позицию в сегменте (она должна быть в конце логической цепочки) – она выделится цветом.
12. Щёлкните по символу катушки на панели инструментов:

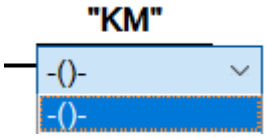


13. Дважды щёлкните над катушкой по обозначению **<??.?>**, введите имя переменной чтобы привязать переменную к катушке:

Результат:



Обратите внимание!
Для изменения типа катушки, нужно щёлкнуть дважды по катушке и в выпадающем списке выбрать нужную (сейчас доступны только нормальная и инверсная):



Ветвления (Branches)

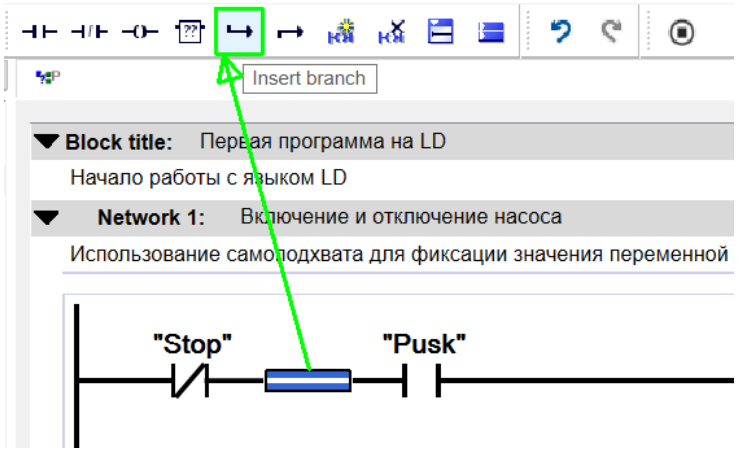
В программировании на языке LD используются два основных логических метода:

- Последовательное выполнение логики
- Параллельное выполнение логики (ветвление)

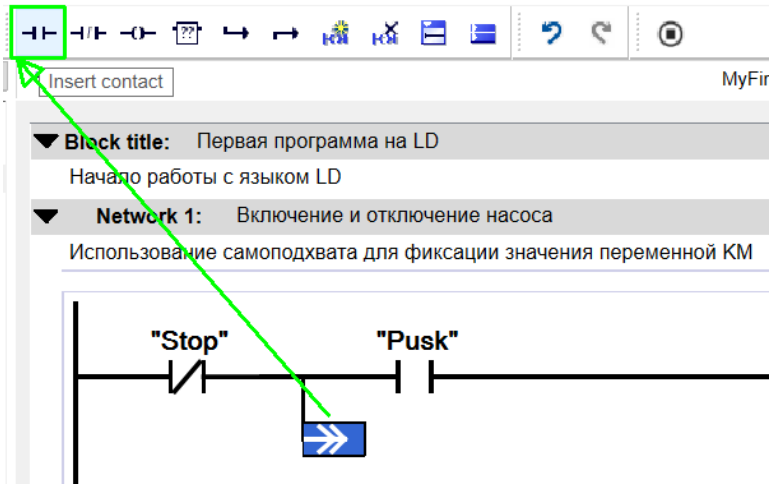
При последовательном выполнении элементы в логической цепочке подключаются один за другим последовательно. «Ток» может проходить только тогда, когда все контакты включены (на выходе контакта «1»). Последовательное подключение реализует логическую функцию «И».
Параллельное выполнение: несколько контактов/катушек подключены параллельно. «Ток» проходит одновременно через все элементы, подключённые параллельно.
Ветвление реализует логическую функцию «ИЛИ».

Параллельная ветвь может быть вставлена только в том случае, если основная логическая цепочка содержит **хотя бы один** элемент LD.
Параллельная (нисходящая) ветвь может быть добавлена вниз от элемента LD или непосредственно подсоединена к шине питания.
Ответвление должно содержать по крайней мере **один элемент** LD, который необходимо перетащить в ответвление, и завершить конструкцию восходящей ветвью.
Чтобы удалить параллельное ответвление, необходимо удалить **все** элементы LD в ответвлении: ПКМ -> **Delete**. Когда последний элемент LD удаляется из ветви, **оставшаяся** часть ветви также должна быть удалена: ПКМ -> **Delete**.

14. Добавьте параллельно НО контакту **Pusk** НО контакт **KM**. Для этого выделите место, от которого должна начинаться параллельная ветвь, после чего этот участок цепи будет выделен цветом.
15. Щёлкните по изображению нисходящей ветви (I**nsert branch**) на панели инструментов:

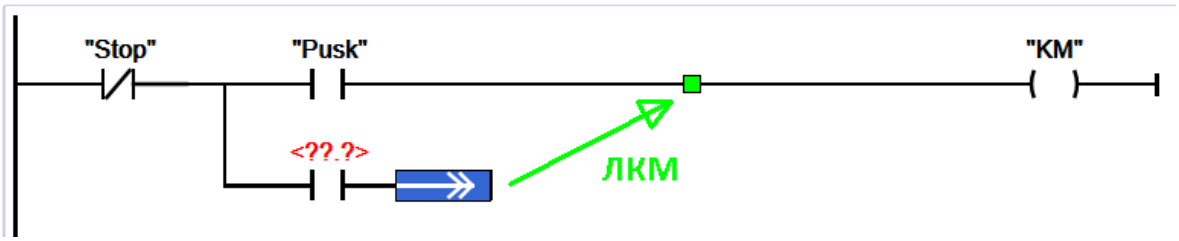


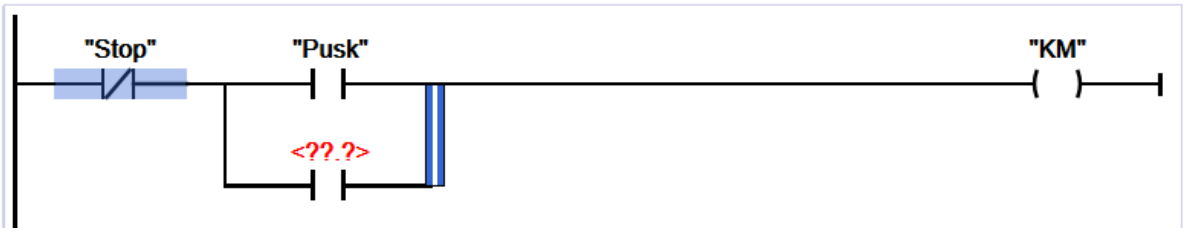
16. Щёлкните по стрелке добавленной нисходящей ветви, чтобы она была выделена цветом. После чего на панели инструментов щёлкните по значку НО контакта – для добавления его в ответвление:



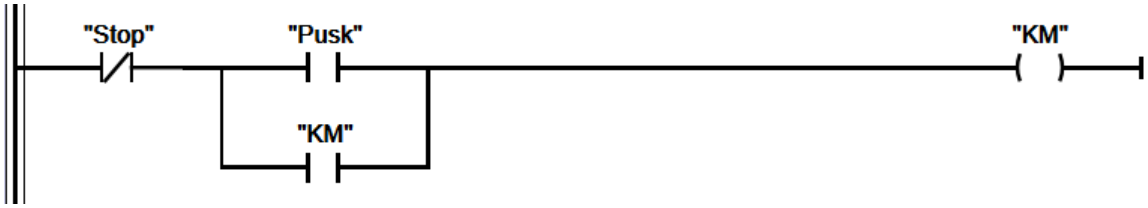
Точки пересечения

17. После добавления контакта в ответвление щёлкните по стрелке добавленной нисходящей ветви, чтобы она была выделена цветом. Нажав левой кнопкой мыши (далее – ЛКМ) по выделенной стрелке, удерживайте её нажатой и перемещайте курсор к **точке пересечения**, которая отобразится белым квадратом. Как только точка пересечения будет достигнута, цвет квадрата изменится с белого на зелёный. После этого левую кнопку мыши можно отпустить – параллельное соединение создано.



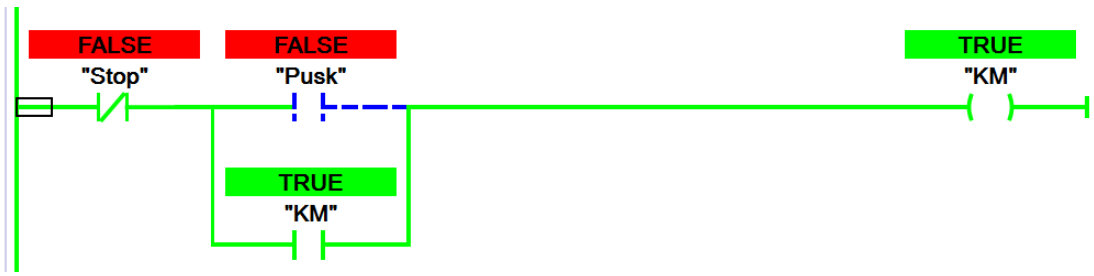


18. Дважды щёлкните над добавленным контактом по обозначению **<??.?>**, введите имя переменной **KM** чтобы привязать переменную к контакту:



Скомпилируйте программу, исправьте ошибки при их наличии, загрузите программу в симулятор и проверьте правильность работы:

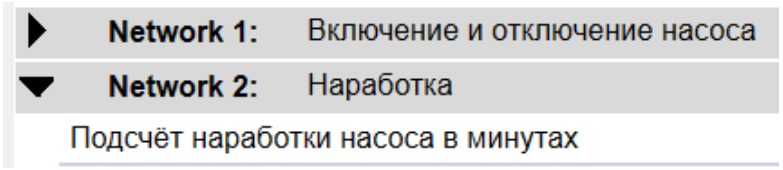
- При установке (**Pusk** = 1) и сбросе (**Pusk** = 0) переменной **Pusk** значение переменной **KM** устанавливается и фиксируется в TRUE (“1”)
- При установке (**Stop** = 1) и сбросе (**Stop** = 0) переменной **Stop** значение переменной **KM** сбрасывается и фиксируется в FALSE (“0”)



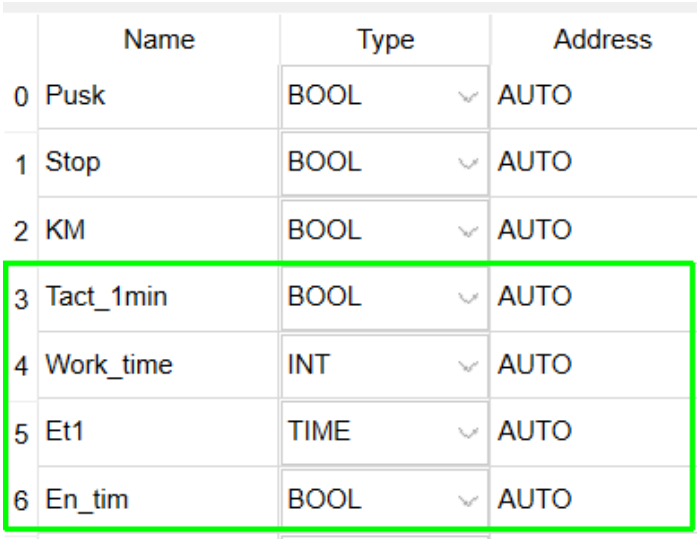
Добавление функций и функциональных блоков

19. Создайте сегмент (**Network**) для подсчёта наработки нашего насоса – времени, когда **KM** будет в **TRUE** (“1”).
Для этого в панели инструментов нажмите кнопку добавления нового сегмента (**Insert network**):

20. Добавьте название и комментарий для **Network 2**:



21. Добавьте новые переменные в область декларации программной секции (выделены рамкой):

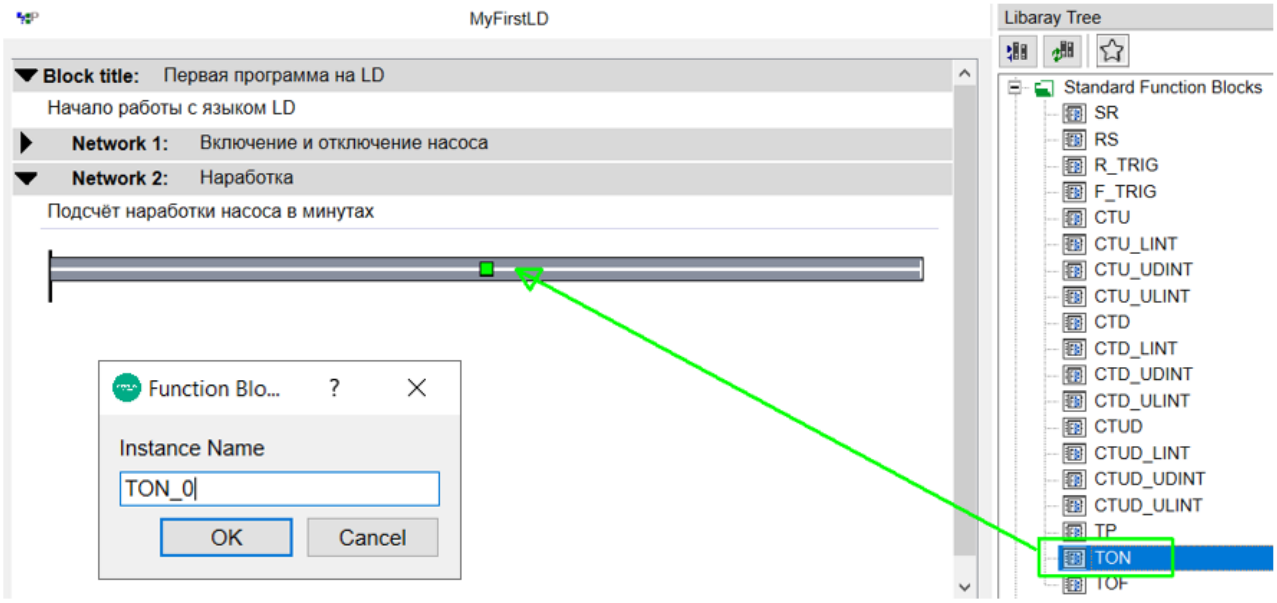


	Name	Type	Address
0	Pusk	BOOL	AUTO
1	Stop	BOOL	AUTO
2	KM	BOOL	AUTO
3	Tact_1min	BOOL	AUTO
4	Work_time	INT	AUTO
5	Et1	TIME	AUTO
6	En_tim	BOOL	AUTO

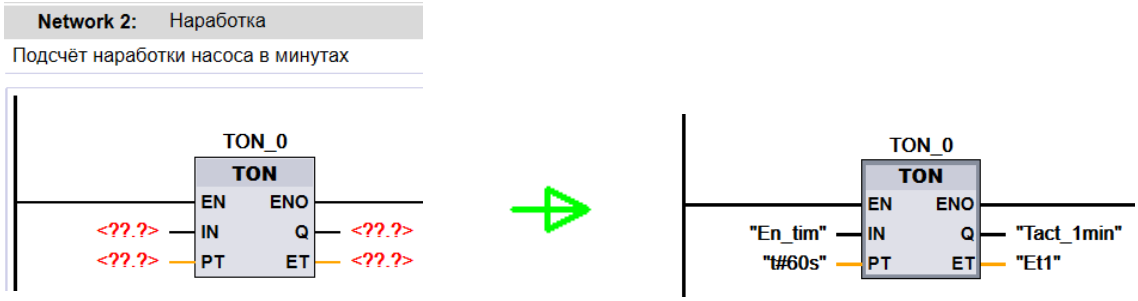
22. Добавьте **таймер TON** в сегмент (**Network 2**).
Для этого нужно открыть библиотеку функций (**Library tree**) - > **Standard Function Blocks** - > и найти **TON**.

Если на экране не отображается библиотека функций, включите отображение, зайдя в **Programming** -> **View** -> **Library Tree** и установите чекбокс («галочку»)

Щёлкните по логической связи в сегменте (**Network 2**), чтобы она стала выделенной цветом. Нажав левой кнопкой мыши (далее – ЛКМ) по таймеру TON в библиотеке (Library tree), удерживайте её нажатой и перемещайте курсор на линию связи до появления зелёного квадрата. После этого левую кнопку мыши можно отпустить.
При этом появится окно создания имени экземпляра – система автоматически использует название функционального блока и порядковый номер, например, **TON_0**. Нажмите **OK**

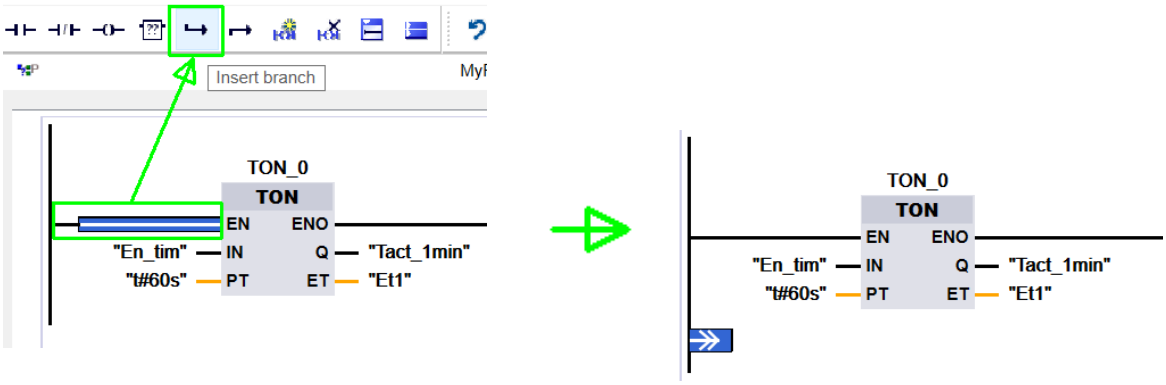


После добавления таймера введите вместо символов **<??.?>** имена переменных из п.21:

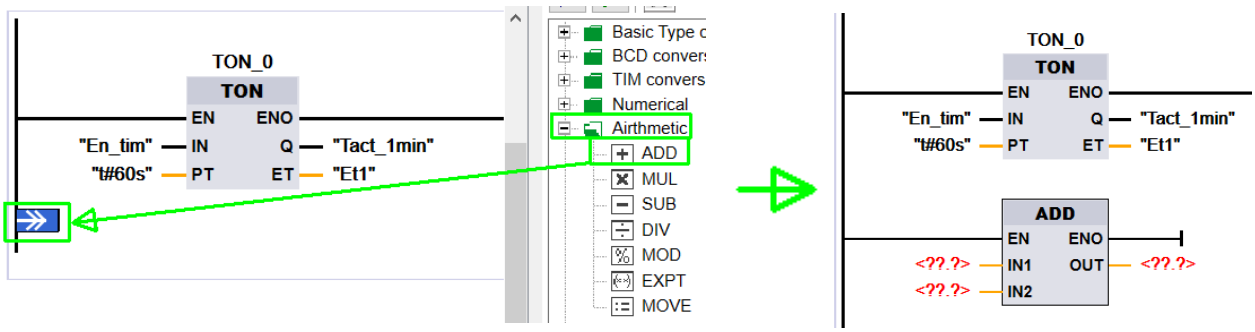


Обратите внимание!
В качестве уставки таймера на входе **PT (Preset Time**) используется константа, заданная в формате **TIME**: **t#60s**

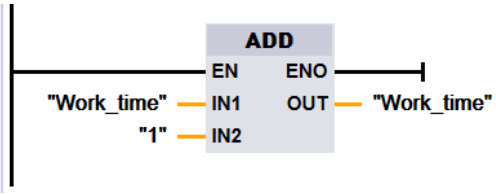
23. Добавьте в сегмент блок сложения **ADD**, который находится в группе **Arithmetic** библиотеки функций.
Для этого щёлкните по части логической цепочки, которая начинается от шины питания, чтобы выделить её цветом. После чего добавьте нисходящую ветвь (**Insert branch**):



Выделите добавленное начало логической цепочки нажатием на стрелку.
Нажав ЛКМ по функции **ADD** в библиотеке (Library tree), удерживайте её нажатой и перемещайте курсор на стрелку до появления зелёного квадрата. После этого левую кнопку мыши можно отпустить.

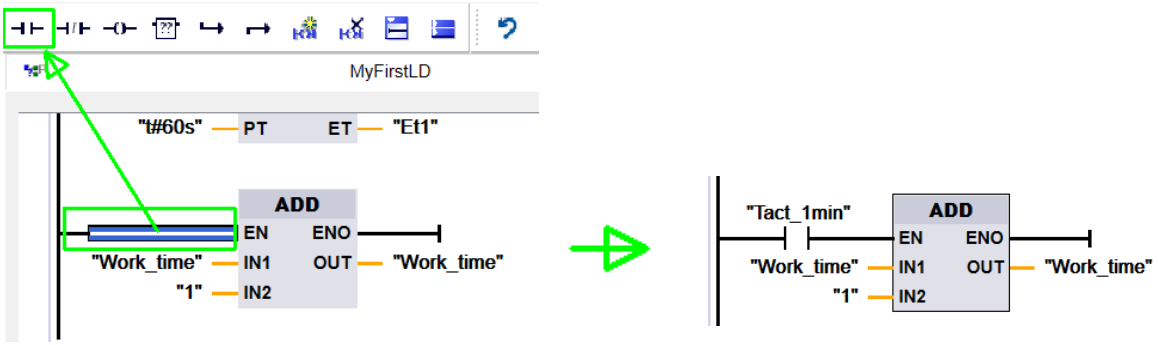


После добавления функции **ADD** введите вместо символов **<??.?>** имена переменных из п.21:

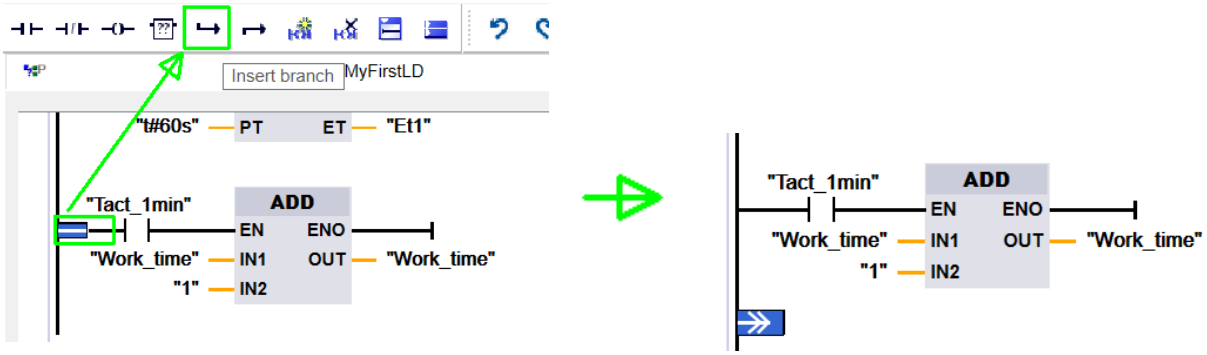


Обратите внимание!
На вход **IN2** блока **ADD** добавлена константа “1”, для чего нужно дважды щёлкнуть по символам **<??.?>,** ввести число 1 и завершить ввод нажатием **Enter**.

24. Добавьте в программу логику, которая будет вызывать блок сложения ADD не каждый скан ПЛК, а только по срабатыванию таймера TON_0.
Для этого выделите начало логической цепочки, идущее ко входу **EN** функции **ADD**, после чего добавьте в это место НО контакт и привяжите к нему переменную **Tact_1min**:

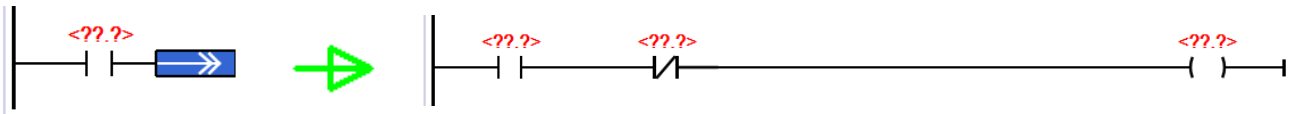


25. Сделайте перезапуск таймера.
Для этого добавьте ещё одну логическую цепочку в сегмент (**Network 2**): выделите начало логической цепочки до НО контакта **Tact_1min** - > добавьте ответвление (**Insert branch**):

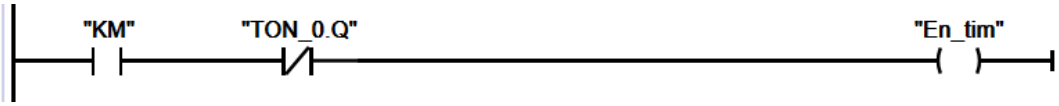


Добавьте в цепочку последовательно НО контакт, НЗ контакт и катушку.

<u>Обратите внимание!</u>
Для последовательного добавления элементов нужно сперва выделить стрелку на конце, а затем добавить элемент:



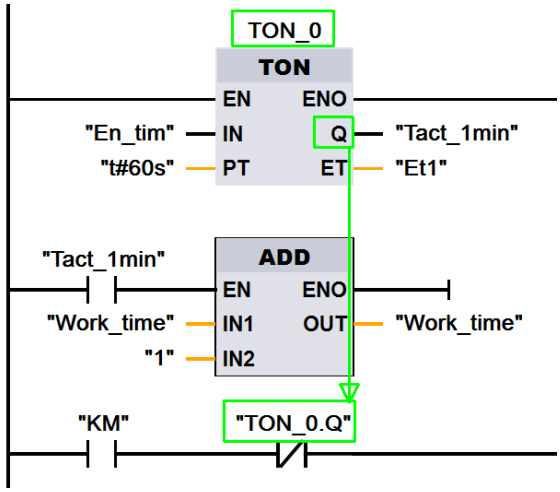
Свяжите элементы с переменными (см.рис.):



Обратите внимание!

При использовании функциональных блоков в программе можно обратиться непосредственно к входам/выходам ФБ, используя синтаксис:

Имя_экземпляра_ФБ . имя_параметра



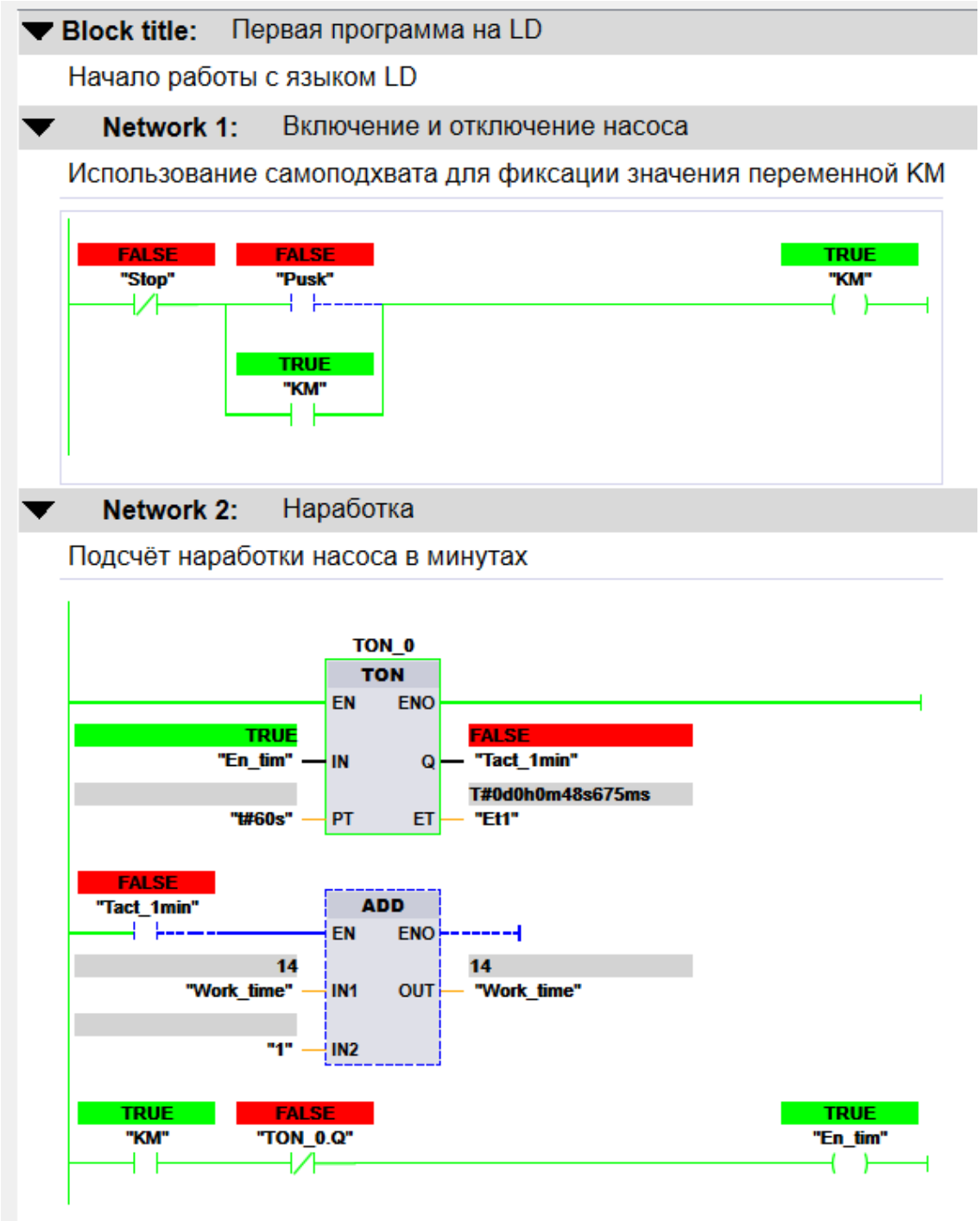
Так, для перезапуска таймера используется обращение к выходу **Q** таймера **TON_0** с помощью синтаксиса **TON_0.Q**

Пояснения к логике перезапуска таймера и наработки:

- Когда контактор (насос) отключён (KM = 0) – команды на запуск таймера нет
- Когда контактор (насос) включён (KM = 1) – подаётся команда на запуск таймера
- При достижении уставки (ET = PT = 60 секунд) появляется сигнал на выходе таймера (TON_0.Q = 1)
- Этот сигнал устанавливает переменную Tact_1min: (Tact_1min = 1), которая запускает добавление единицы к времени наработки Work_time: (Work_time = Work_time +1)
- И этот же сигнал через НЗ контакт сбрасывает переменную En_tim (En_tim = 0), что приводит к сбросу таймера
- При сбросе таймера выход TON_0.Q тоже сбрасывается (TON_0.Q = 0)
- Что при включённом контакторе (KM = 1) устанавливает бит En_tim: (En_tim = 1) и запускает таймер

Сохраните, скомпилируйте проект, исправьте ошибки при наличии, загрузите в симулятор и проверьте работу программы:

- При запуске контактора кнопкой **Pusk** каждую минуту должно увеличиваться значение переменной **Work_time**
- При остановке контактора кнопкой **Stop** значение переменной **Work_time** должно фиксироваться до следующего пуска контактора



Удаление контактов, катушек, функций и функциональных блоков

ПКМ на элементе и выбрать "Delete" для удаления.

Используемые сокращения и термины

Сокращения

- PC – персональный компьютер (ПК)
- PLC – программируемый логический контроллер (ПЛК)
- LMB (Left Mouse Button) –левая кнопка мыши (ЛКМ)
- RMB (Right Mouse Button) –правая кнопка мыши (ПКМ)
- IO (Inputs/Outputs) – входы/выходы (в/в) ПЛК и модулей расширения
- POU (Program Operation Unit) – программа пользователя
- FB (Function Block) – функциональный блок
- Function Block Diagram (FBD) – язык функциональных блоков
- Ladder Diagram (LD) – язык лестничных диаграмм
- Continuous Function Chart (CFC) – язык непрерывных функциональных схем
- ВОС – внутренняя операционная система ПЛК [прошивка ПЛК]

Термины частей интерфейса

- Toolbar – панель инструментов
- Tabs – вкладки меню: содержит основные разделы: Configuration (Конфигурация), Programming (Программирование), Display (Отображение), Commissioning (Ввод в эксплуатацию).
- Menu – пункты меню
- Project Tree – дерево проекта
- Library Tree – библиотека элементов
- Properties Window – свойства
- Output – поле вывода
- Status bar – строка состояния
- Programming Toolbar – панель инструментов программирования
- Variable Declaration Area – область объявления переменных
- Program editing area – область написания программного кода
- Display Toolbar – панель инструментов Дисплей
- Page editing area – область создания пользовательских экранов
- Run/Stop – режим работы ПЛК (пуск/стоп)
- Simulation Mode – оффлайн симулятор ПЛК
- Live Debug Mode – онлайн мониторинг ПЛК
- Watch Window – окно наблюдения за переменными
- ЛКМ – щелчок левой кнопкой мыши
- ПКМ – щелчок правой кнопкой мыши

Социальные сети

systemeelectric_official

youtube.com/c/SystemeElectric

vk.com/Systemeelectric

Systeme Electric

www.systeme.ru

Наши бренды

Systême electric

Dēkraft

Механотроника

Systême soft

Ver 1.0

02/2026