ISaGRAF - это очень просто!

(часть V, Язык программирования FC и LD)

Гулько С.В., ХОЛИТ Дэйта Системс, г.Киев

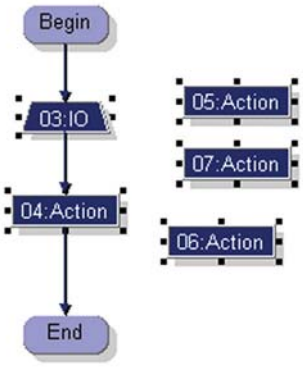
Если языки программирования появляются, значит это кому-нибудь нужно. Неважно, насколько велико число заинтересованных лиц - миллионы или 2-3 человека. Главное, чтобы были ресурсы и веские основания изобрести что-то новое.

Статьи данной серии описывают языки программирования, доступные в программном пакете ISaGRAF. Мы рассматриваем синтаксис, приводим небольшие примеры, однако не так часто акцентируем внимание на областях применения, оставляя за читателем право решать, что и как использовать. Вы знаете свои проекты лучше, чем мы, а выбор основного языка предопределяется именно тем, что нужно сделать. Если программа мала - нет смысла разворачивать SFC схемы, проще написать пару строк на ST или разместить FBD блоки. А вот масштабные проекты, содержащие большое количество ветвлений и переходов, лучше оформлять на графических языках, таких как SFC или FC. Цель тут только одна - получить достаточно наглядное представление об алгоритме. Видя перед собой основные узлы программы, значительно проще искать ошибки.

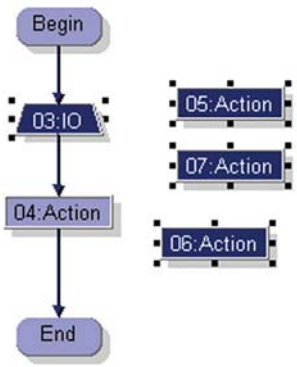
В стандарте IEC 61131-3 в качестве такого языка-декоратора выступает SFC. Он хорошо справляется с возложенными на него обязанностями, однако не все пользователи его научились(точнее, захотели научиться) нормально читать и воспринимать. Что же тогда делать? Писать все на FBD? Не совсем так. Выходом из ситуации может стать язык FC в силу своей привычности. Многие инженеры изучают основы построения алгоритмов с использованием блочных диаграмм. Что ж, эти навыки будут носить теперь не только теоретический характер. Итак, давайте знакомиться - язык программирования FC!

Редактор

Для написания программ на FC используется ставший уже нам знакомым редактор. Если Вы создаете новую программу, ISaGRAF откроет окно, содержащее всего два базовых элемента - ***Begin*** и ***End***. Графическая панель предоставляет доступ ко всем элементам управления, которые можно использовать в своей программе. Рассмотрим их подробнее.



Select - позволяет выбирать элемент или группу элементов



программы. Для того, чтобы выбрать группу элементов, можно, удерживая клавишу *Shift*, нажимать на каждый элемент в отдельности. Так Вы сможете более точно создать группу, предназначенную, например, для удаления. Кроме того, можно создать область выделения, удерживая нажатой левую клавишу мышки и перемещая указатель по экрану. Все элементы, попавшие в окно, считаются выделенными. А вот для того, чтобы снять выделение с элемента, который расположен в центре, нажмите *Shift* и кликните по элементу, подлежащему исключению.

Элементы

F2: ***Action*** - действие. Блок предназначен для ввода кода, который необходимо выполнить. Код может быть введен на одном из трех языков ISaGRAF: ST, LD или IL. Вставить секцию кода в тело основной программы можно простым перетаскиванием элемента. После того, как блок ***Action*** будет вставлен, ISaGRAF заботливо откроет для Вас окно редактирования кода(редактор второго уровня). Оно будет расположено справа. В нем можно сменить название самого блока действия, выбрать язык и закрыть редактор второго уровня. Для того, чтобы выбрать язык программирования, нажмите на кнопке ***ST***. В появившемся списке выберите один из доступных языков. В зависимости от Вашего выбора панель инструментов будет ме-

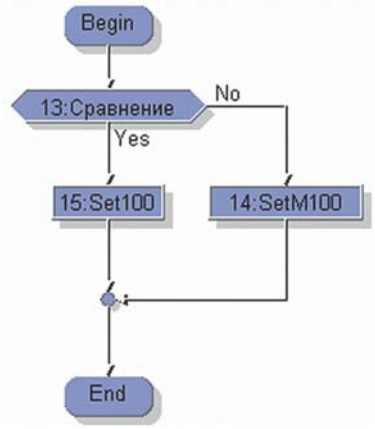
нять свой внешний вид, предоставляя наборы элементов для программирования на ST(используется по умолчанию), LD и IL.

F3: ***I/O*** - блок вызова драйверов. Данный блок по своим свойствам ничем не отличается от ***Action*** и служит исключительно для выделения на графическом уровне участков, где используются нестандартные библиотеки, поставляемые производителем оборудования. К нестандартным можно смело отнести все драйвера

устройств ввода/вывода. Такой подход позволяет разделить стандартные и нестандартные компоненты, что значительно облегчит переход на другое оборудование в будущем(если это будет иметь место).

F4: ***Test*** - условный переход. Модуль принятия решений и изменения направления выполнения программы. Данный блок аналогичен программной конструкции *if-then-else*. Условие, которое задается в редакторе, может быть как статической переменной, так и вычисляемым значением или возвращаемой из функции величиной. Допускается два типа переменных - булевые(логические) и целочисленные(допускаются два значения: 0 и 1). Рассмотрим небольшой пример управления ходом выполнения. Условие: если переменной switch будет присвоено значение ИСТИНА, в переменную value будет занесено 100. Если *switch* принимает значение *FALSE*, то *value* будет равно -100. Для того, чтобы воплотить нашу программу в коде, необходимо с помощью словаря создать две переменные. Как Вы уже догадались, называться они будут ***switch*** и ***value***(тип *bool* и *sint* соответственно). После того, как они созданы, перейдем к написанию(точнее, к рисованию) логики.

В результате должна получиться приблизительно такая схема:



В блоке "Сравнение" мы должны указать следующий код:

switch=true

Этим самым мы задаем условие перехода по двум из доступных веток. Если ***switch*** равен ***TRUE***, выполнение пойдет по ветке ***Yes***, в противном случае - по ***No***. Отредактируем блок ***Action*** за веткой ***Yes*** следующим образом:

value:=100;

Ветка ***No*** приведет нас к коду

value:=-100;

Компилируем проект и запускаем его на симуляцию. Проверив переменные, убедимся в правильности выполнения кода.

F5: ***Flow*** - линия связи между элементами. Показывает направление распространения сигнала.

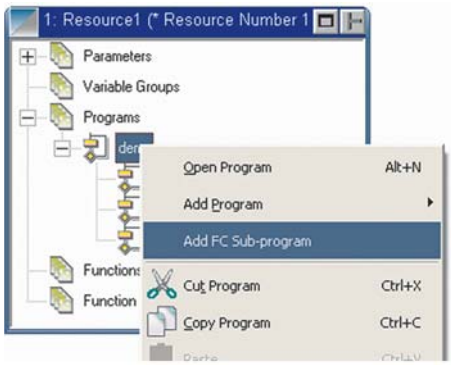
F6: ***Connector*** - точка объединения сигналов. Данный элемент используется, например, для сведения нескольких сигналов от разных источников в одно место. Наша демонстрационная программа для блока ***Test*** также использует точку объединения сигналов от ***Action*** блоков.

F7: ***If-then-else*** - комбинированный блок, представляющий собой объединение блоков ***Test*** и ***Action***. Вы можете создать аналогичный блок самостоятельно, однако, если заранее известно, что в данном месте программы будет использоваться условие, то можно слегка сэкономить время и разместить на диаграмме ***If-then-else*** конструкцию.

F8: ***Do-while*** - выполнение какого-либо участка кода, пока условие истинно. Как видите, этот блок опять же представляет собой композицию ***Test*** и ***Action***. Данная конструкция аналогична циклу ***until***, в котором код выполнится хотя бы один раз вне зависимости от того, истинно ли условие или же нет.

F9: ***While-do*** - классический *while*, облаченный в графическое представление.

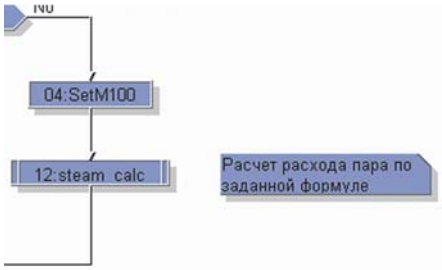
F10: ***Subprogram*** - вызов подпрограмм в языке ***FC***. После размещения данного элемента на диаграмме появится диалоговое окно, в котором можно будет осуществлять выбор из доступных процедур. Подпрограммы для языка ***FC*** пишутся также на языке ***FC***. Для того, чтобы добавить новую подпрограмму в проект, необходимо нажать правой кнопкой мышки на имени родительской программы и в контекстном меню выбрать пункт *Add FC Sub-program*. На подпрограммы



распространяются общие для ***FC*** правила редактирования.

F11: ***Renumber*** - обновить нумерацию. Достаточно часто нумерация следования блоков сбивается. Причиной может быть удаление блоков в процессе редактирования или же их перемещение и т.п. При этом может возникнуть ситуация, когда за блоком 2 идет 10, а уже потом 8. Это никак не сказывается на порядке выполнения самой программы, однако может вносить путаницу в сопроводительную спецификацию финальной версии программы. Поэтому бывает полезно перенумеровать блоки заново. Для этого достаточно нажать по кнопку *Renumber*, после чего ISaGRAF самостоятельно изменит порядковые номера блоков.

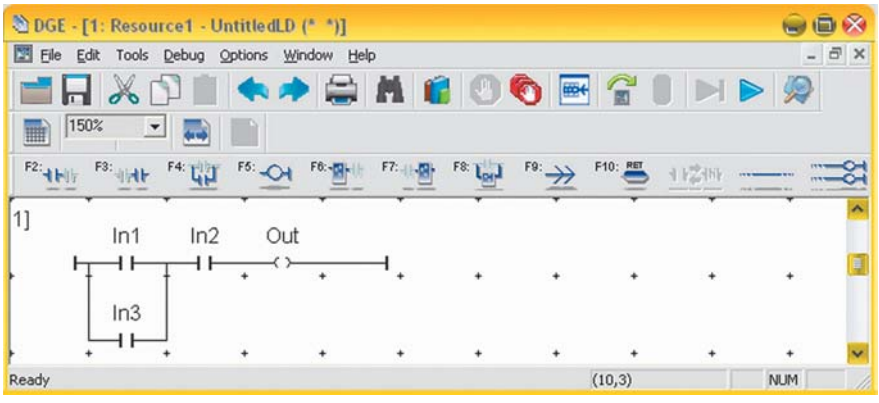
F12: ***Comment*** - вставить блок комментариев. О пользе комментирования кода мы уже говорили не единожды, поэтому не будет заострять наше внимание на этом. Просто покажем на небольшом примере использование данного блока.



Вот, пожалуй, и все, что можно сказать про ***FC***. Как видите, язык простой и наглядный, хорошо приспособленный для описания комплексных алгоритмов. По большому счету, именно для этого он и вводился в ISaGRAF. Он практически не используется для описания задачи в целом, а вот для облегчения понимания отдельно взятых участков является идеальным кандидатом.

Ladder Diagram - это язык так называемой релейной (лестничной) логики. Он широко используется для замены логических схем, выполненных на релейной технике. Наиболее подходит для использования инженерами по автоматизации, работающими на промышленных предприятиях и не имеющих предварительного опыта программирования промышленных контроллеров. Окно разработки программ на языке ***LD*** в ISaGRAF 5.0 представлено ниже.

Вся концепция работы LD программы построена на взаимной работе



логических элементов цепи, которыми являются ключи и витки. Ключ представляет собой контакт, который управляется определенной булевой переменной. Виток же наоборот - изменяет состояние присвоенной ему булевой переменной. Если проводить сравнение с электрической цепью, то ключ на ней - это выключатель, а виток - лампочка. Вся выстроенная цепь

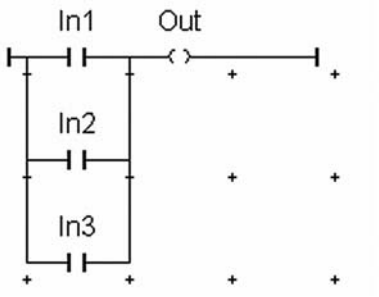
получает напряжение от левой и правой вертикальных шин питания, причём "ток" течёт по схеме слева направо, то есть левая шина "**+**" а правая "**–**".

Каждая линия схемы (провод) имеет логическое состояние *FALSE* или *TRUE*. Линии, соединенные вместе, имеют одно и то же логическое состояние. Любая горизонтальная линия, соединенная с левой вертикальной шиной питания, имеет состояние *TRUE*, то есть "подключена к сети".

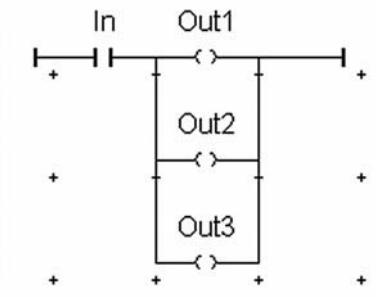
Элементы диаграммы могут соединяться последовательно и параллельно.

Параллельное соединение.

Представляет собой набор элементов, соединенных параллельно, то есть все элементы присоединены слева к одной линии и справа ко второй линии. Пример параллельного соединения показан на рисунке.

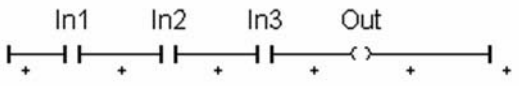


При данном соединении виток ***Out*** примет значение *TRUE*, если хотя бы один из контактов In1, In2, In3 будет замкнут (то есть, если хотя бы один из выключателей включен, лампа зажжется). Это пример логического элемента *ИЛИ*. Так же можно одним контактом присвоить значения нескольким переменным (одним выключателем зажечь несколько ламп). Этот вариант показан на втором рисунке.



Последовательное соединение.

Представляет собой набор элементов, соединенных последовательно один за другим. Пример такого соединения показан на рисунке. При таком соединении переменная Out примет значение TRUE только если все контакты слева будут иметь значения TRUE(то есть лампа зажжется только в том случае, если включить все 3 выключателя).

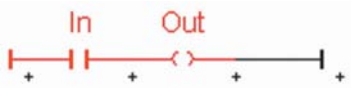


Теперь следует ознакомить читателя со всеми разновидностями контактов и витков.

Контакт - это элемент диаграммы, который меняет состояние связей между другими элементами диаграммы.

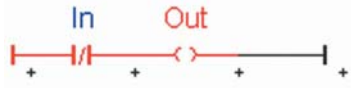
Прямой контакт.

Это обычный "выключатель". Если переменная контакта имеет значение TRUE, то контакт замкнут, если FALSE - разомкнут. Очевидно, что сигнал пройдёт на виток Out только когда контакт In "замкнут", как показано на рисунке.



Обратный (инвертированный) контакт.

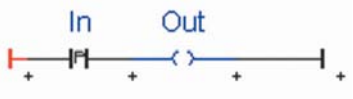
Контакт работает противоположным способом от прямого контакта(что ясно из названия). Он пропускает сигнал через себя при "выключенном" состоянии *(In = FALSE)*, то есть сигнал пойдёт на виток ***Out*** только когда контакт ***In*** "разомкнут", как поазано на рисунке.



Контакт с определением переднего фронта.

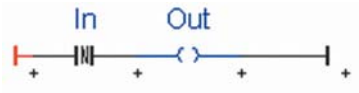
Импульсный контакт, на диаграмме обозначен буквой ***P*** *(P - англ. Positive)*. Пропускает сигнал через себя только в момент, когда переменная

контакта переходит из состояния *FALSE* в состояние *TRUE*, то есть в момент положительного фронта сигнала переменной ***In***. Во всех остальных случаях контакт "закрыт". В итоге, "лампа" ***Out*** "мигнёт" только если "включить" контакт ***In***. Чтобы она мигнула ещё раз, нужно контакт "выключить" и затем снова "включить" (то есть перевести переменную ***In*** из *TRUE* в *FALSE*, а затем обратно в *TRUE*).



Контакт с определением заднего фронта.

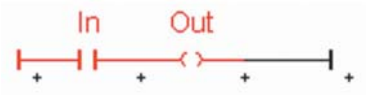
Импульсный контакт, на диаграмме обозначен буквой ***N*** *(N - англ. Negative)* Работает аналогично предыдущему, но с той лишь разницей, что сигнал проходит через контакт в момент, когда переменная контакта переходит из состояния *TRUE* в состояние *FALSE*, то есть в момент отрицательного(негативного) фронта сигнала переменной ***In***.



Виток - это элемент диаграммы, который присваивает значения булевской переменной, привязанной к нему.

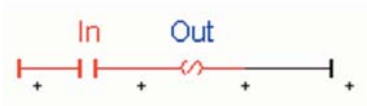
Прямой виток.

Присваивает привязанной к нему переменной значение *TRUE*, если есть сигнал слева от витка, или значение *FALSE*, если сигнал отсутствует. В данном случае переменная ***Out*** будет иметь значение *TRUE* только в случае, если контакт In замкнут *(**In** = TRUE)*. То есть это аналог лампочки - есть напряжение, лампа горит, нет напряжения - потушена.



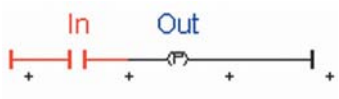
Обратный виток.

Действует по обратной аналогии к прямому витку. То есть переменная витка будет иметь значение *TRUE*, если сигнала слева нет, или значение *FALSE*, если сигнал слева есть.



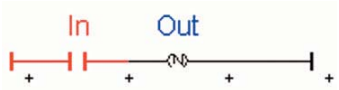
Виток с определением переднего фронта.

Импульсный виток, на диаграмме обозначен буквой ***P*** *(P - англ. Positive)*. Присваивает привязанной переменной значение TRUE только тогда, когда сигнал слева переходит из состояния FALSE в состояние TRUE, то есть в момент положительного фронта. Во всех остальных случаях переменной присваивается значение FALSE.



Виток с определением заднего фронта.

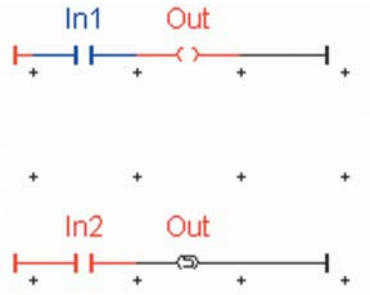
Импульсный виток, на диаграмме обозначен буквой ***N*** *(N - англ. Negative)*. Присваивает привязанной переменной значение TRUE только тогда, когда сигнал слева переходит из состояния TRUE в состояние FALSE, то есть в момент отрицательного фронта. Во всех остальных случаях пере-



менной присваивается значение FALSE.

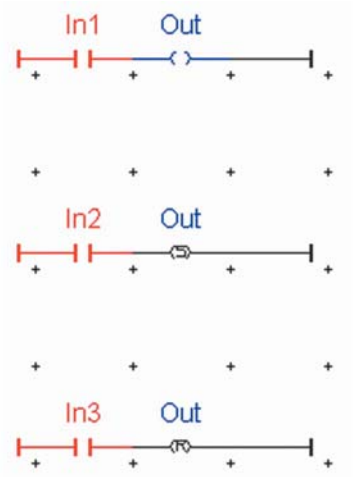
SET виток (виток установки).

Позиционный виток, на диаграмме обозначен буквой ***S*** *(S - англ. Set)*. Устанавливает привязанную к нему переменную в значение *TRUE*, когда слева на виток подается сигнал *TRUE*. Если сигнал слева изменяется на *FALSE*, переменная витка остается в состоянии *TRUE*, если только в другой части диаграммы этой переменной не присваивается значение *FALSE*. Установка *SET* витка является более приоритетной, чем установки других витков (кроме *RESET* витка), то есть если в диаграмме прямым витком переменной ***Out*** присвоено значение *FALSE*, а *SET* витком присвоено значение *TRUE*, переменная будет иметь значение *TRUE* до тех пор, пока не будет "выключен" *SET* виток.



RESET виток (виток сброса).

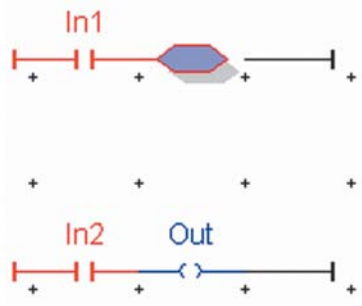
Позиционный виток, на диаграмме обозначен буквой ***R*** *(R - англ. Reset)*. Устанавливает привязанную к нему переменную в значение *FALSE*, когда слева на виток подается сигнал *TRUE*. Установка *RESET* витка является более приоритетной операцией, чем любые другие установки, то есть даже если на диаграмме переменной витка присваивается значение *TRUE*, она все равно будет иметь значение *FALSE*.

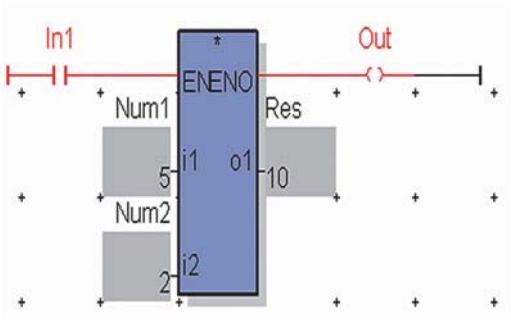


Метка - некое предложение на диаграмме, на которое можно "перескочить" с помощью оператора перехода.

Оператор RETURN.

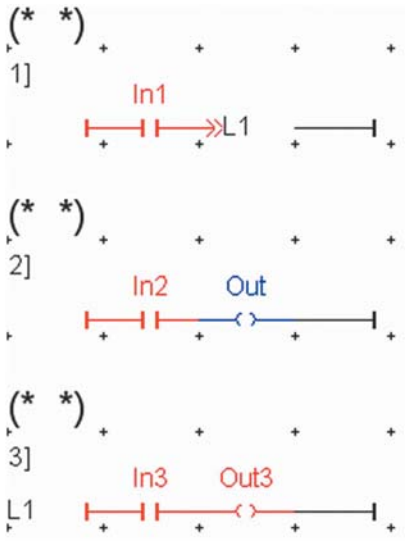
Специальный оператор, который завершает выполнение программы на том месте, где он установлен, то есть все строки диаграммы, расположенные ниже оператора *RETURN* не будут выполняться, если оператор активен (на него слева подан сигнал *TRUE*).





Переходы и метки.

В языке ***LD*** есть механизм перехода от одной строки программы к другой "не по порядку". Для этого используются метки и операторы перехода. Если на оператор перехода слева подан сигнал *TRUE*, программа продолжает выполняться с того места, где установлена соответствующая метка. Если переход осуществляется через несколько строк вниз, то пропущенные строки не выполняются.



Функциональные блоки в LD.

В диаграмме можно использовать функциональные блоки так же, как и в ***FBD*** программе, только с некоторыми особенностями. Если у блока первый вход не булевского типа, то в блок добавляется дополнительный булевский вход *EN*, который стоит на первом месте и определяет рабочее состояние блока. Если на вход *EN* подаётся сигнал *TRUE* - блок работает, если *FALSE* - блок выключен.

Если первый выход блока не булевского типа, то в блок добавляется дополнительный выход *ENO*, который имеет такое же значение, как и первый булевский вход блока.

Как видите, язык достаточно несложный, хотя и специфический. В основном на нем создаются вставки или дополнения к диаграммам ***SFC*** и очень редко ***LD*** встречается в качестве самостоятельного языка проекта.

На этом заканчивается обзорный цикл статей, посвященных языкам программирования ISaGRAF. "За бортом" остался ***IL***, своего рода язык ассемблера, однако его в настоящий момент уже практически не применяют. В самом ISaGRAF он присутствует исключительно для совместимости со стандартом IEC 61131-3, встретить его в реальных проектах в настоящий момент почти невозможно.

В следующих публикациях мы планируем перейти к ISaGRAF программированию с более практической точки зрения, ведь теоретическая база уже есть - настало время сделать что-то настоящее!

КОНТАКТЫ:
тел: (044) 492-31-08, 492-31-09
e-mail: s.gulko@isagraf.com.ua