



ISaGRAF -

это очень просто !

(2-я часть)

С. В. Гулько, ООО "ХОЛИТ Дэйта Систем", г. Киев

Вопрос выбора языковых средств всегда остро стоял в среде программистов. Очень часто он служил причиной раздоров и ссор, размахивания руками и достаточно нелицеприятных высказываний. Так было и так будет, таковы люди. Конечно, не все придерживаются максималистских принципов, и большая часть людей согласится с высказыванием - "каждому языку - свои задачи". Пишите драйвера или сервисы на С, создавайте пользовательские интерфейсы на Java или Tk, занимайтесь обработкой данных с помощью MathScript или LabVIEW. Да, языков очень много и у каждого из них своя специфика, отражающая и облегчающая решения задач, для решения которых он, собственно, и был предназначен.

От сего дня мы будем говорить о языках, применяемых для написания систем управления. Прежде всего, давайте определим, что характеризует данный класс задач и выделим его специфические стороны. В центре находится контроллер промышленной автоматизации (PLC или PAC), обладающий даже в базовом варианте устройствами ввода/вывода сигналов. Со стороны объекта автоматизации через датчики в PAC поступает информация о его (объекта) состоянии, устройства вывода дают возможность оказывать воздействие на объект. Согласитесь, схема достаточно простая, но ее в любой момент можно расширить взаимодействием типа "контроллер-контроллер", "контроллер-SCADA", добавить различные варианты резервирования и горячей замены. И вот здесь уже будет над чем подумать классическому программисту на С.

Системам управления также присущ целый ряд специфических параметров построения алгоритма работы программы - это событийность, цикличность, логический параллелизм, механизмы абстрагирования от аппаратного обеспечения.

PAC работает с реальными данными, которые могут поступать из внешнего мира в совершенно непредсказуемом порядке. В задачу системы управления входит обработка входящих данных и принятие решения на их основе, таким образом, ход течения программы будет меняться в зависимости от состояния объекта.

Подавляющее большинство систем в промышленной автоматизации работают по замкнутому циклу - считали данные с устройства ввода, обработали, приняли решение, обновили устройства вывода сигналов.

Алгоритмы управления могут быть произвольной сложности, начиная от мониторинга одного входного канала и заканчивая приложениями с десятками тысяч внешних переменных и параллельно выполняющихся частей программы. Наличие параллелизма может быть обусловлено как событийностью, так и требованием к самой программе, когда некоторые участки кода нужно выполнять одновременно. Конечно, можно было построить алгоритм на основе единого блока с выполнением по условию, однако это может сделать программы громоздкими и трудно читаемыми.

Требования, учитывающие все перечисленные выше требования, вылились в создание целого ряда специальных языков, нашедших применение в виде самостоятельных продуктов, или же входящих в интегрированные среды разработки.

Стандарт IEC 61131 объединил в себе несколько языков, в той или иной степени соответствующих перечисленным требованиям. Три графических (SFC, FBD, LD) и два текстовых (ST и IL) языка, входящие в стандарт, теоретически должны позволить инженеру-программисту - и даже скорее инженеру, чем программисту, решить любые задачи в области создания систем управления. Некоторые считают, что

часть языков там присутствует в виде наследия, такого себе "привета из прошлого", однако наш опыт показывает, что все они в большей или меньшей степени нашли применение на практике.

На страницах "ПиКАД" мы познакоим Вас с основными возможностями всех языков IEC 61131, давая основные знания, а вот что использовать в реальной деятельности предстоит уже решать Вам.

SFC

Начнем мы с SFC (Sequential Function Chart) - графического языка, в котором алгоритмы задаются в виде последовательного набора диаграмм. Почему именно с SFC? В настоящий момент он является основным, и единственным, языком для создания объектных расширений стандарта IEC 61499.

Но это преимущество, не столь очевидное для пользователей. Среди очевидных: легкость и наглядность написанной программы, очевидное отображение параллельно выполняемых операций, событийность.

SFC вобрал в себя все лучшие черты сетей Петри, унаследовав и некоторые не совсем удобные его качества, такие как излишнее абстрагирование и структурирование. Язык позволяет описать алгоритм на очень высоком уровне, не вдаваясь в детали, однако это не всегда применимо, ведь детали задачи или одного и того же алгоритма могут меняться. Решается эта проблема достаточно просто - SFC позволяет делать вставки на других языках программирования, например, на ST или LD, расширяя базовые возможности практически до бесконечности.

Язык SFC предназначен для описания последовательных процессов. Сам процесс разделяется на несколько

ко логических под-шагов, соединенных переходами. Такой подход позволяет не только разбить задачу на более мелкие составные части, но очень удобно обрабатывать возникающие события. Основная блок-схема, составленная из компонентов SFC, называется программой первого уровня. Если Вы сделаете двойной щелчок по шагу, то получите доступ к программам второго уровня. Именно на нем выполняется основная работа - обработка сигналов и вычисления. По сути, SFC играет больше декоративную роль. Средств, которыми он располагает, явно недостаточно для комфортной работы, но никто и не требует использовать только один язык. ISaGRAF тем и отличается, что позволяет комбинировать сильные стороны различных инструментов. Что же касается декоративных функций SFC - даже декорации должны быть удобными и не затруднять, а наоборот, облегчать работу программиста.

Редактор SFC

Давайте поближе познакомимся с SFC, его возможностями и функциями редактора в ISaGRAF. Для начала создайте новый или откройте существующий проект. В окне Ресурсов найдите пункт Programs и добавьте в него новую программу на языке SFC. Сделать это можно, кликнув правой кнопкой мыши по Programs и выбрав в меню Add program -> SFC. К аналогичному результату можно прийти через основное меню: Insert -> Add program -> SFC. Задайте осмысленное имя своей программе, далее двойной щелчок - и Вы внутри редактора.

Любая программа на SFC должна содержать хотя бы один начальный шаг.



Step - рабочая лошадка программы на SFC - именно в нем задаются действия, которые будут выполняться при достижении тех или иных условий. Каждому шагу присваивается уникальный номер, который впоследствии может быть изменен. Этот номер будет использоваться оператором Jump для организации переходов.



Transition служит для связывания между собой двух шагов и задания условий о вхождении в последующий шаг.



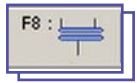
OR divergence - одиночное расхождение, элемент языка, служащий для указания, какой участок кода из возможных вариантов следует выполнить. При достижении начала одиночного расхождения программа продолжит свое выполнение в зависимости от условий на входах.



OR convergence или одиночное схождение, служит для объединения нескольких ветвей программы, вышедших из одиночного расхождения.



AND divergence, двойное расхождение, являет собой аналог параллельного выполнения программы. Процессы будут выполняться во всех ветках, связанных в двойном расхождении.



AND convergence, двойное схождение, служит для объединения параллельных ветвей программы, вышедших из AND divergence.

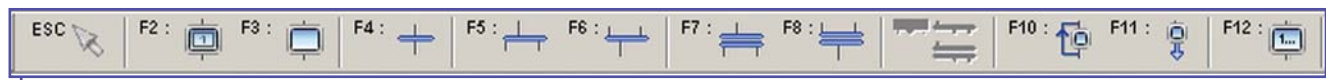
смыслу к Link, так как выполняет аналогичные функции, но без рисования линий связи между шагами. При создании этого элемента Вам нужно будет указать, на какой из шагов будет совершаться переход.

Редактировать программу можно следующим образом - выбрав из палитры инструмент, поднесите его к последнему элементу на уже созданной диаграмме и оставьте его там. Редактор сам дорисует соединительные линии. Выберите инструмент *Select(Esc)*, выделите только что нарисованный элемент и попробуйте его переместить. Редактор автоматически будет перестраивать и перерисовывать линии связи.

Конечно, присутствует и стандартный минимум для работы - *Copy(Ctrl+C)*, *Paste(Ctrl+V)*, *Cut(Ctrl+X)*, *Delete(Del)*. Вы можете работать как с отдельными элементами, так и с группами заранее выделенных блоков. Получить доступ к этим функциям можно как с помощью клавиатурных сокращений, так и через контекстное меню, доступное через щелчок правой кнопкой мыши на элементе. Также в этом меню будет присутствовать достаточно полезная функция *Rename(переименование)*. С ее помощью Вы сможете задавать осмысленные названия для шагов(вместо S1, S2...).

Первые шаги

Как Вы заметили, язык SFC описывает действия. Какова должна быть реакция на определенные события, что нужно запустить, а что остановить



Палитра инструментов SFC на первый взгляд не впечатляет, но не стоит смущаться - все возможности языка станут видны чуть позже. Все элементы снабжены надписями, описывающими соответствующие им клавиши быстрого доступа. Запомнив их один раз, Вы будете экономить свое время в работе.



Инструмент **Select** служит для того, чтобы выделять и перемещать объекты в плоскости рисования.



Initial Step представляет собой первый шаг программы, именно с него будет начинать жизнь Ваш алгоритм при включении контроллера.



New branch - инструмент, позволяющий добавить еще одну ветвь схождения или расхождения. По умолчанию расхождение создается с двумя возможными потоками данных. Конечно, в реальных приложениях этого не всегда хватает, число условий может достигать сотен. Если Вы видите, что два пути - это откровенно мало, воспользуйтесь **New branch**.

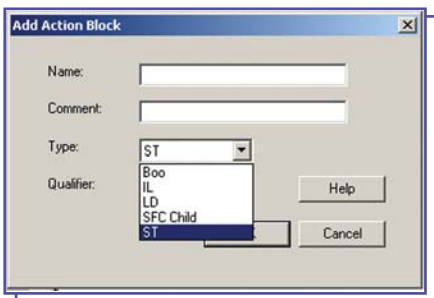


Link - позволит Вам переходить с одного шага на другой в пределах одной ветви. Делается это перетягиванием соединительной линии к целевому шагу.



Jump - очень близок по

- все это идеально подходит для решения задач промышленной автоматизации. Кроме этого, в SFC есть возможность задавать и выполнять действия в период активности шага. Что это значит? Предположим, у Вас есть последовательность, описывающая дверь и систему освещения в комнате. Используя возможности SFC, можно указать, что при наступлении события "Дверь открыта" выходной переменной "Включить освещение", связанной с платой ввода/вывода, будет присвоено значение true(истина), что, в конечном итоге, приведет к включению света. Конечно, это немного наивный пример, возможности SFC гораздо шире!



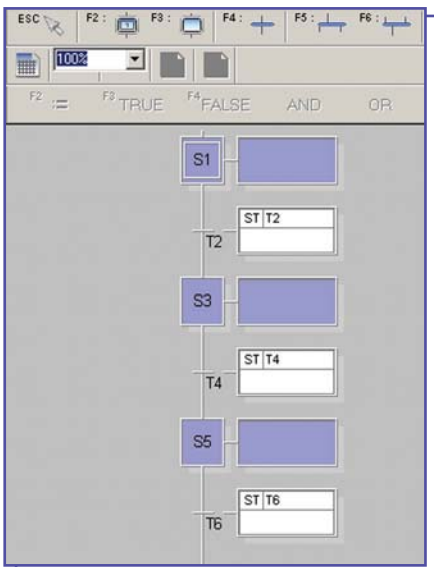
ISaGRAF позволяет создавать действия (step) ST, IL, LD и SFC. Помимо стандартных языков, Вы можете воспользоваться и набором логических действий.


Булевы действия

SFC позволяет менять значения логических переменных при входе(активизации) и выходе из шага. Переменные могут быть как внутренними, так и выходными. Редактор ISaGRAF даст возможность быстро задать последовательность действий над булевыми переменными. Всего существует три типа таких действий: (S)et, (R)eset и (N)on stored.

Если требуется установить значение переменной (активен - TRUE, неактивен - FALSE), воспользуйтесь квалификатором N. В то же время S будет возвращать TRUE при входе в шаг, а R выдаст FALSE при тех же условиях.

Давайте посмотрим, как это можно реализовать в ISaGRAF. Проект уже создан и редактор SFC открыт. Сделайте в нем небольшую программу, как показано на рисунке.



 **Небольшой совет** - измените цвет фона рабочего окружения на любой комфортный для Вас цвет, кроме белого. Сделать это можно через меню Options >> Customize >> SFC Editor, по-

меняв Background color. Делается это потому, что некоторые комментарии в редакторе SFC выводятся светлым шрифтом, что приводит к их затруднительному прочтению на белом фоне - это как увидеть черную кошку в темной комнате.

Создайте две переменные типа boolean. После этого сделайте двойной щелчок на шаге. Появившееся окно показывает созданные действия.

В настоящий момент список пуст, что мы исправим, вызвав контекстное меню через нажатие правой клавиши мыши. Единственный доступный пункт - Add action block не оставляет особого выбора. А вот появившийся после нажатия диалог имеет достаточно много вариантов выбора. В настоящий момент нас интересует Type: boo. Редактор заботливо выведет список всех доступных логических переменных, над которыми мы можем производить некоторые действия. Выбираем одну переменную и задаем квалификатор N. Аналогичные действия продельваем для второго шага нашей программы. Сохраняем наш проект, компилируем его и запускаем на Симуляцию.

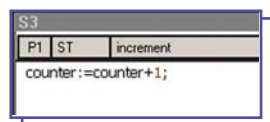
После этого откройте редактор SFC нашей тестовой программой и добавьте несколько переменных в список отслеживания. Его рабочая зона находится в самом низу экрана. Как видите, программа выполняется, маркер активности шага перемещается от одного шага к другому, при этом меняются значения переменных. Обратите внимание, что их значения соответствуют текущей активности шага.

Его рабочая зона находится в самом низу экрана. Как видите, программа выполняется, маркер активности шага перемещается от одного шага к другому, при этом меняются значения переменных. Обратите внимание, что их значения соответствуют текущей активности шага.

| Name | Resource | Value | Physical | Scope | Alias |
|--------|------------|-------|----------|-------|-------|
| fstep1 | Resourc... | FALSE | FALSE | first | |
| fstep2 | Resourc... | TRUE | TRUE | first | |

Действия на ST

Кроме создания логических действий, ISaGRAF позволяет применять и другие языки, например, ST. Последовательность создания полностью аналогична работе с логическими переменными. Добавился, или правильнее будет сказать, расширился квалификатор P - можно выполнить последовательность действий сразу после активации шага(P1) или же сразу после деактивации(P0). В качестве небольшого теста можно написать счетчик шагов. Для этого создадим переменную типа DINT с именем count и воспользуемся уже существующей программой, созданной ранее. Выберите первый блок действий и добавьте в него импульсную последовательность на ST, нажав на Add Action Block



В раскрывшемся окне введем код:

`count:=count+1;`

Сохраним файл, откомпилируем программу и запустим ее в Симуляторе. Если после этого Вы перейдете в окно редактора SFC и найдете в списке переменную count, Вы заметите, что ее значение изменяется всякий раз, когда шаг становится активным.

Условия

Все шаги в SFC соединяются с помощью переходов. Переход предназначен для задания некоторого условия, истинность которого сделает активным следующий шаг. В качестве этого условия может выступать логическая переменная, выражение на ST, IL или LD или результат, возвращаемый функцией.

Самый простой вариант задачи условий - логическая переменная. Ход программы будет определяться состоянием переменной(TRUE или FALSE). Если TRUE - программа продолжает свое выполнение, FALSE - "заморозит" алгоритм до появления истинны.

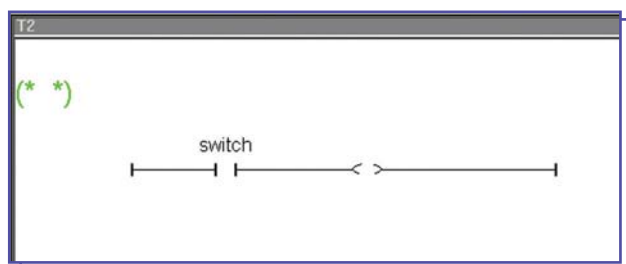
Воспользуемся уже готовым шаблоном и на его основе создадим несколько условий. Создайте логическую переменную с названием switch. Сохраните ее в Словаре и вернитесь в редактор SFC. Сделайте двойной щелчок по переходу. В результате появится окно для ввода текста, которое мы будем использовать для задания условий переходов. В редакторе введем имя нашей пере-

менной и завершим условие точкой с запятой: **switch;**

Запущенная в Симуляторе программа наглядно продемонстрирует работу переключателей, остановив маркер и ничего не делая. Но стоит лишь поменять значение на TRUE (это можно сделать, перейдя в Словарь), как жизнь возобновится.

Логические условия можно задавать не только с помощью булевых переменных. Допустимы любые выражения, результатом которых является TRUE или FALSE. Сравнение двух чисел или выражений - хороший тому пример. Допустима даже конструкция $2 < 1$;

Если по тем или иным причинам Вас не устраивает ST, можете задать условия для переходов на языке LD



Значение витка будет являться значением перехода. Используя LD, Вы сможете построить достаточно сложную логическую цепочку.

Однако намного более широкие возможности для определения переходов предлагают функции и функциональные блоки. Внутри может быть сколь угодно сложное условие или вычисления, ведь функциональные блоки можно написать и на С, что практически безгранично расширяет возможности программиста. Да, это требует определенных затрат, но и результат окупается сторицей.

Давайте для простоты сделаем функцию, которая будет постоянно возвращать значение TRUE. Мы немножко забежим вперед, так как тема создания функций будет рассмотрена в последующих уроках, но обойтись без этого никак нельзя. Возвратимся в окно редактирования Ресурса и пункт *Functions* и добавим нашу новую функцию *ruler*. Правый щелчок откроет обширное меню, где нас будет интересовать пункт *Parameters*. Раскрывшееся окно позволит задать аргументы функции, а также тип возвращаемого значения. По умолчанию тип будет установлен в DINT, но нам нужен BOOL. Внесите соответствующие изменения. Теперь перейдите в окно редактирования исходного текста функции. Здесь нам потребуется ввести всего две строки кода: первая присво-

| Name | Short n... | Type | Direction | Comment | () | Dimensi... |
|-------|------------|------|-----------|---------|----|------------|
| ruler | | BOOL | Output | | | |

ит значение выходной переменной, вторая выполнит выход из функции

ruler=true;

return;

Заметьте, что выходная переменная имеет точно такое же имя, как и функция.

Теперь дело за малым - перейти к нашему "испытательному стенду" в редакторе SFC и изменить любой из переходов, указав в качестве выражения созданную только что функцию, скомпилировать программу заново и загрузить ее в Симулятор. Конечно,

результат ничего особо интересного не покажет, ведь переход всегда будет открыт (наша функция постоянно

возвращает **true**). Вы можете сами расширить возможности функции **ruler**, введя более сложные условия.

Родители и дети

Изучая возможности редактора SFC по созданию действий в шагах, Вы, скорее всего, обратили внимание на пункт SFC Child. Очень важной способностью SFC является запуск дочерних программ, которые будут выполняться в параллельном режиме. Таким образом, все SFC программы делятся на два типа - основные, или родительские, программы и потомки. Родители могут управлять ходом выполнения потомков: запускать, перезапускать, останавливать и удалять подчиненные программы.

Программы на SFC, которые не имеют родителей, называются основными. Они выполняются ядром ISaGRAF на момент запуска приложения. Программы же с атрибутом "child" самостоятельно не запускаются, их можно лишь вызвать из любой основной программы.


Присутствует ряд ограничений, вызванных, прежде всего, вопросами синхронизации и обеспечения целостности данных. Ограничения достаточно логичны по своей сути, например, родитель может быть только один, потомком может управлять только один родитель, а вот управ-

лять потомками потомка родитель не может.

Давайте попробуем на практике работу с подчиненными программами.

Для начала создадим потомка. Перейдите в окно Ресурсов, выберите готовую SFC программу и сделайте на ней щелчок правой клавишей мыши, после чего выберите Add Child SFC. В редакторе создадим программу подобную той, что была создана ранее для инкрементации счетчика. Создайте переменную типа DINT (назовем ее, для примера, count1), которая будет выполнять роль счетчика, затем модифицируйте код редактируемой программы так, чтобы инкрементировалась именно переменная count1. После того, как дочерняя программа создана, вернитесь к редактированию родительской программы, поместив ее вызов в одно из действий SFC. Далее - компиляция, запуск - и мы можем наблюдать, как в параллельном режиме выполняются две программы.

Конечно же, это замечательная возможность - запускать несколько функций одновременно, но гораздо важнее иметь возможность управлять запущенными потоками. Для этого существует набор функций *gstart*, *gkill*, *gfreeze*, *grst*, *gstatus*.

 Эти функции не являются частью языка SFC или стандарта IEC 61131, они присутствуют лишь в ISaGRAF, так что если в требованиях к будущей программе стоит обязательное соответствие требованиям стандарта, то Вам следует воздержаться от их использования.

Вот, пожалуй, и все описание языка SFC. Как видите, здесь нет ничего сложного.

Для полноты картины нужно рассматривать несколько языков одновременно. В следующий раз мы остановимся на языке ST, наиболее подходящем в качестве дополнения к SFC. Это небольшое описание было лишь вводным уроком в мир увлекательного программирования в ISaGRAF и впереди нас ждут еще много интересных тем.



КОНТАКТЫ:

тел. (044) 492-31-08(09)

e-mail: info@isagraf.com.ua