ISaGRAF - это очень просто!

(часть III, FBD)

Гулько С.В., ХОЛИТ Дэйта Системс, Киев

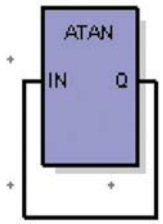
Предположим, перед Вами стоит задача создать язык программирования, в основе которого лежат функции, но при этом сам язык является графическим. Если попробовать представить функцию в графическом виде, то проще всего это будет сделать с использованием черных ящиков. Ящик имеет: 1) входы, к которым подключается управляющее воздействие или входные данные, 2) само тело, реализующее алгоритм обработки, который может быть сколь угодно сложным и, наконец, 3) выходы, с которых снимается обработанный сигнал или новое управляющее воздействие. Сейчас трудно сказать, кто изначально был автором этой идеи, но можно отметить, что одно из первых упоминаний о функциональных блоках было сделано в 1975 году в книге "Структурное программирование", авторами которой являлись такие видные идеологи программирования, как Дейкстра, Дал и Хоор.

Нужно признать, что идея оказалась более чем удачной. Функциональные блоки стали стандартом de-facto в системах промышленной автоматизации. Многие SCADA-системы используют его в качестве вспомогательного языка, расширяющего базовую функциональность, а SoftPLC решения вообще немыслимы без FBD. Кстати, еще одним примером ПО, использующего функциональные блоки, является известный пакет графического программирования LabVIEW.

В данной статье речь пойдет о языке FBD, который является частью стандарта IEC-61131 и положен в основу концепции объектно-ориентированных блоков нового стандарта IEC-61499.

Язык программирования FBD описывает связи между входными и выходными переменными. Под входными переменными следует понимать любые переменные ISaGRAF - внутренние или же входные, которые "вводят" информацию в функциональный блок. Аналогично, выходными будут такие переменные, которые "выводят" информацию из блока. Блок представляет собой функцию, которая сама по себе может быть сформирована из других таких же блоков. Входные и выходные переменные подключаются к блокам с помощью соединительных линий, обозначающих потоки распространения данных. Не всегда удобно иметь дополнительные переменные для хранения промежуточных вычислений, поэтому выходы одного блока можно напрямую подключать к входам другого.

Функциональный блок обладает строго заданным количеством выходов. Что же касается входов, то их число может задаваться пользователем, конечно же, при условии, что это разрешено разработчиком самих блоков. Для всех входов и выходов обязательно должно быть установлено подключение, т.е. все они должны быть задействованы и не могут "болтаться в воздухе". ISaGRAF позволяет подключать выходы функционального блока к его входам, создавая этим обратные связи. В процессе работы над статьей у автора возник вопрос, что будет, если замкнуть выход элемента **Atan**, отвечающего за вычисление арктангенса, на его вход. Безусловно, такая конструкция не имеет смысла, но этот нехитрый тест позволяет определить зрелость технологии и качество компилятора. Попытка собрать приложение-тест прошла успешно, но ISaGRAF предупредил, что сделанная мною программа смысла не имеет.



Графически блоки отображаются в виде прямоугольников и представляют собой единичную операцию над входными переменными. Название этой операции (или функции) расположено в прямоугольнике. Важно помнить, что входные и выходные параметры блоков имеют строго заданные типы (FLOAT, DINT и т.д.), и пользователь не может без предварительного преобразования подать на вход, объявленный как целочисленный, строковую переменную. Кроме того, соблюдаются правила, установленные архитектурой ISaGRAF - на вход блоков могут подаваться только внутренние и входные переменные, а также сигналы с выходов блоков. Допускается также использование констант в качестве входных параметров.

Выходные контакты блока могут подключаться к выходным и внутренним переменным. Кроме того, можно передавать выходной сигнал на вход другому функциональному блоку или же функции.

Переменные и входы/выходы блоков соединяются с помощью линий связи. Линия указывает направление распространения сигнала, при этом соблюдается правило, что сигнал идет слева направо. Правый и левый концы линий должны связывать данные одного типа. Кроме связи типа "переменная-переменная" существует множественная связь, называемая дивергенцией (расхождением). Используя ее, можно распространять данные от одного источника на большое количество принимающих компонентов. Естественно, должно соблюдаться правило о соответствии типов данных.

Как видим, правила использования FBD достаточно просты и логичны. Ознакомившись с ними, давайте перейдем к изучению редактора и его возможностей.

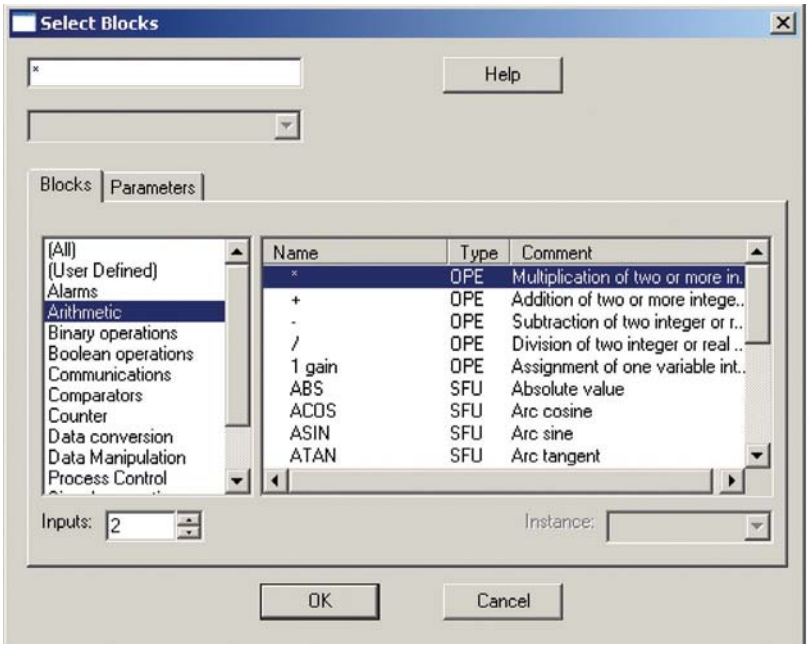
Создадим программу. Для этого нажмем правой кнопкой мышки на пункте **Programs** и в меню **Add Pro-**

gram выберем **FBD**: **Function Block Diagram**. Появившееся окно не ошеломляет количеством возможностей и изобилием различных кнопок и меню. И это радует. Включено только самое необходимое, имеющее отношение к разработке - базовые элементы, словарь, компиляция-проверка, симуляция. Давайте рассмотрим элементы более подробно.

Select - основной элемент управления и редактирования. С его помощью выполняются все операции по перемещению, изменению размеров и направлений. Можно двигать как функциональные блоки, так и линии связи, делая, таким образом, диаграмму более читабельной.

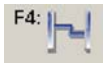
Variable - функция вставки переменной или константы. При нажатии появится диалоговое окно, содержащее все переменные проекта, сгруппированные по категориям. Диалог позволяет легко осуществить выбор среди входных или выходных переменных, найти переменную нужного типа, например, BOOL.

Function Block - функция добавления нового функционального блока в проект. Появившееся окно будет содержать в себе список всех доступных функциональных блоков, сгруппированным по категориям. Количество блоков и их типы могут изменяться в зависимости от поставщиков программного или аппаратного обеспечения. Следует учитывать, что потенциально каждый блок обладает набором внутренних данных, которые появляются в результате вычислений. Например, если используется блок сложения двух величин, то как таковых промежуточных данных не будет - система просто передаст результат сложения с входа на выход. А вот использование блока "ПИД-регулятор" уже подразумевает индивидуальный набор данных, особенно если таких блоков будет два или больше. Звучит все это немного устрашающе, однако все сложности на себя берет ISaGRAF. Работа с функциональными блоками в ISaGRAF организована аналогично динамически загружаемым библиотекам (например, DLL). Каждый раз, когда пользовательская программа вызывает функциональный блок, будет выполняться один и тот же участок кода. Примечательно то, что наряду с общим кодом блоки имеют собственные наборы переменных и внутренних данных.

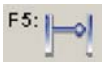


Некоторые блоки имеют произвольное количество входных параметров. Например, к блоку AND может быть подключено две и более переменных, результатом же будет логическая функция над всеми входами. Задать такое количество входов можно с использованием элемента **Inputs**, однако он доступен не для всех функций.

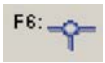
Вкладка **Parameters** дает возможность получить доступ к скрытым элементам функциональных блоков. Их наличие зависит от того, предусмотрел это разработчик функциональных блоков или нет. Предположим, блок имеет ряд входных значений, задаваемых как константы. Конечно, можно потребовать от пользователя каждый раз создавать переменные требуемого типа и подключать их на диаграмме, однако гораздо более гуманно будет воспользоваться константами. Если блок содержит эти константы, их значение можно задать самостоятельно.



Draw Link служит для создания (рисования) связей между компонентами диаграммы. Нажав на иконку или клавишу горячего доступа, обратите внимание, как изменил курсор мышки свою форму. Для того чтобы связать объекты, щелкните мышкой по выходу первого объекта. После этого, не отпуская клавишу, переместите линию на вход второго объекта. Если все прошло успешно, будет построена соединительная линия. Инструмент обладает некоторым интеллектом и пользователю не удастся связать входы и выходы с разными типами данных.

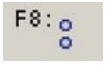


Negation connection links предназначена для создания связей с отрицанием между компонентами. Вы можете связать только логические входы/выходы. Во всем остальном инструмент аналогичен **Draw Link.**



Corner - инструмент для задания правил следования сигналов. Используя его, можно задавать собственные правила распространения сигналов. Для этого выберите инструмент и разместите Corner в любой точке диаграммы. После этого добавьте ссылки, ведущие к элементу, а затем ссылки от него.

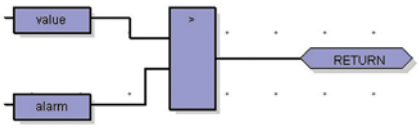
Jump служит для создания условных переходов внутри диаграммы. Элемент имеет логический вход и поданное на него значение TRUE приведет к переходу на указанную метку. При нажатии на кнопку появится окно со списком доступных меток. Можно выбрать одну из них или создать новую, введя ее имя в текстовое поле, после чего следует нажать Add.



Label - предназначен для вставки меток в тело диаграммы. Метка служит для изменения хода выполнения программы. Метку можно выбрать в появившемся диалоговом окне. В случае необходимости, в том же окне ее можно и создать, введя имя в текстовое поле. Метки могут быть расположены в любом месте диаграммы, и это делает их одновременно мощным и опасным инструментом. Написать

программу, входящую в бесконечный цикл, используя метки и переходы, труда не составит. Дать каких-либо разумных рекомендаций по тому, как избежать зацикливания, тоже не получится. Можно лишь посоветовать размещать метки в левой части диаграммы, в отдельной колонке, чтобы сделать их более читаемыми.

Return служит для создания элемента завершения диаграммы. При достижении этого элемента ни одна последующая команда выполняться не будет. Элемент принимает на вход логический уровень и при значении TRUE завершает работу текущей программы. В качестве примера можно рассмотреть такой образец кода



Comments позволит вставить комментарии в текст диаграммы. Это свободный элемент, его можно размещать где угодно. Если размер блока не устраивает, то его всегда можно изменить.

Show or hide execution order - полезный инструмент, позволяющий отобразить последовательность выполнения программы. При текстовом программировании порядок операторов достаточно очевиден. Графический подход, при ряде неоспоримых преимуществ, из-за пространственного представления имеет все же в большей степени "хаотично выглядящую" программу. При включении рассматриваемого режима, операторы будут пронумерованы согласно порядку их выполнения.

Как можно было заметить, на панели присутствуют дополнительные элементы. Они относятся к языку программирования **LD**, который можно комбинировать с **FBD**.

Язык программирования достаточно прост, в нем нет сложных правил и грамматик. Создание программ сводится, по сути, к настройке потоков данных между блоками. Такой упрощенный подход к разработке приводит к тому, что необходимо знать, из каких блоков формировать программу, где использование одного компонента предпочтительнее, чем другого. Библиотека готовых функциональных блоков, с которыми поставляется ISaGRAF, достаточно обширна. Но иногда требуется использовать уникальный алгоритм обработки или протокол передачи данных. В этом случае можно, и даже нужно, воспользоваться средствами ISaGRAF по созданию собственных функциональных блоков.

Создание функциональных блоков

Грамотное программирование подразумевает под собой вынесение повторно использующегося кода в функции или процедуры. Так поступают из нескольких соображений. Во-первых, модульную программу легче читать, а значит и поддерживать. Одним взглядом можно охватить весь алгоритм, тонкости которого, тем не менее, "скрыты за фасадами" функций. Но на первоначальном этапе это не столь важно, нюансы можно уяснить и потом - важно видеть общую картину.

Часто одна и та же последовательность действий может вызываться из разных мест программы. Вот Вам и еще один кандидат на вынесение в процедуры. При программировании целесообразно разбивать код на связанные по смыслу компоненты и последовательности, создавая на их основе свои библиотеки. Такое видение приходит не сразу, обычно его получают, поработав над десятком разноплановых проектов. Однако даже это правило не всегда помогает, и разработчики начинают переделывать уже готовую программу, пытаясь получить в большей степени структурированный код. Подобный процесс назван красивым словом рефакторинг и его адепты рекомендуют почаще возвращаться к написанному коду для его улучшения. Это является хорошей практикой, помогающей не совершать схожие ошибки в будущем. Идеальные программисты с задатками стратегов всегда создают программы с первого захода. Все же, приходится признать, что в большинстве случаев требуется модификация уже созданного кода. Учиться на своих ошибках удается не всегда, но программы начинают работать лучше.

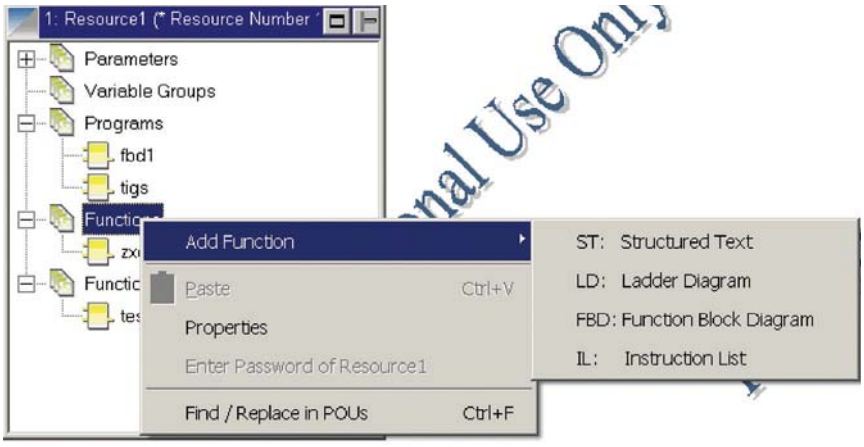
Еще одной важной причиной, по которой нужно группировать код в функциях, является поиск и устранение ошибок. Изначально проще протестировать небольшую последовательность команд с помощью элементарного граничного теста, чем выловить ошибку в большом приложении. Строить программу из таких вот проверенных кирпичиков всегда лучше, ведь надежность системы определяется надежностью ее компонентов.

Давайте посмотрим, что нам может предложить ISaGRAF для "правильного" программирования. В окне **Ресурсов** можно обнаружить две ветки - **Функции(Functions)** и **Функциональные блоки(Function blocks)**. Рассмотрим их детальнее.

Функции

Функция представляет собой последовательность кода на языках **ST**, **IL**, **FBD** или **LD**. Основная особенность функции - у нее нет возможности создания экземпляров. Это значит, что вызов одной и той же функции из другого места "затрет" все внутренние переменные, заменив новыми.

Для того чтобы создать новую функцию, следует нажать правой клавишей мышки на разделе **Functions** и выбрать один из языков, на котором будет вестись разработка. ISaGRAF создаст новый **POU**. Старайтесь давать осмысленные имена новым функциям. Функции могут иметь как входные, так и выходные параметры. Нажмем правой кнопкой мышки на имени только что созданной функции.



В появившемся меню выберем **Parameters**. В результате будет открыт уже знакомый нам Словарь. Используя его, можно создавать и редактировать входные параметры функций. Для того чтобы добавить новый аргумент функции, выберем в меню **Edit->Add Row**. При этом откроется диалоговое окно, позволяющее задать основные параметры, такие как имя и тип. Поле **Short Name** служит для задания псевдонима, который будет потом отображаться в редакторе **FBD**. Не следует игнорировать его. Обратите внимание, что поле **Direction** является неактивным, и по умолчанию установлено в **Input**. Функции имеют обязательный выходной параметр, тип которого можно изменить, и свободное количество входных переменных. После того, как переменные были созданы, можно приступать к редактированию тела самой функции. Для этого выполним мышкой двойной щелчок по имени только что созданной функции. Откроется стандартный редактор **FBD** диаграмм, где, используя уже существующий набор функций и блоков, можно создать что-то свое.

Работа с функциями подразумевает получение данных из входных переменных и запись результата в выходную. Обратиться к входным параметрам можно точно так же, как и ко всем переменным среды ISaGRAF - для этого следует нажать на кнопку **Variable** и в открывшемся диалоговом окне выбрать их из списка. Следует помнить, что входные переменные допускают только чтение данных, а выходные - только запись.

После того, как функция создана, к ней можно обращаться по имени из любого языка, поддерживаемого ISaGRAF. Например, в **FBD** это делается через стандартную панель вставки блоков . Созданные таким образом функции будут расположены в разделе **User Defined**.

Функциональные блоки

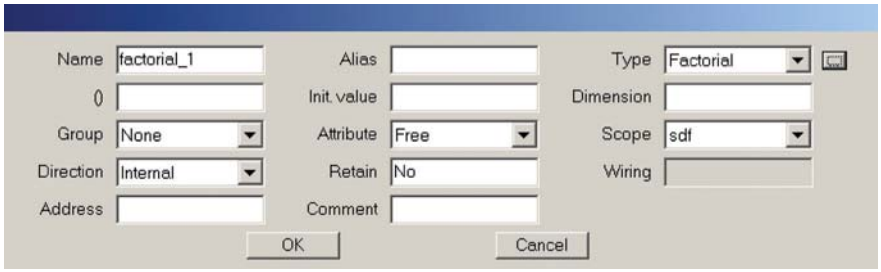
Создание функциональных блоков практически ничем не отличается от создания функций. Выберем раздел **Function blocks** в меню редактирования Ресурса и нажмем на нем правой кнопкой мышки. Так как рассматривается язык **FBD**, будем его и использовать для создания нового блока.

start_command	BOOL	None
alarm	BOOL	None
command	BOOL	None

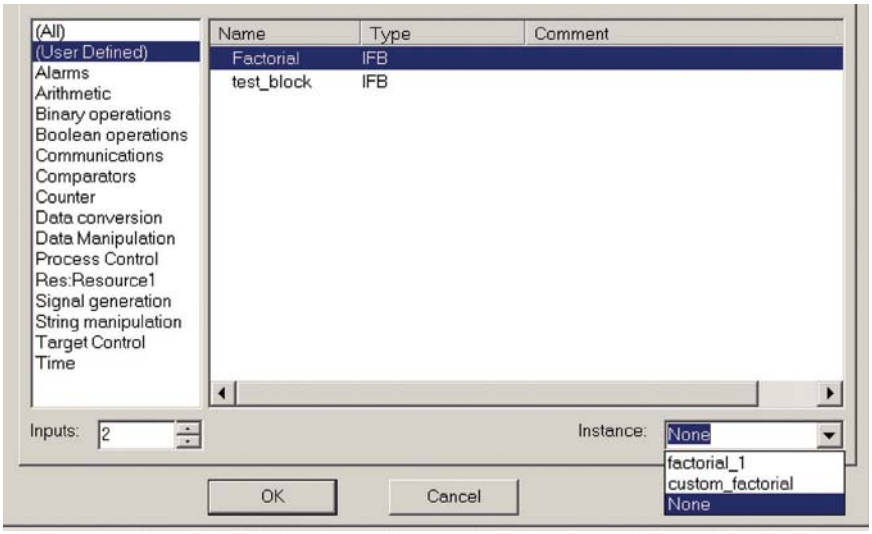
Важное отличие блоков от функций состоит в том, что первые позволяют задавать несколько выходных значений. Убедиться в этом можно, перейдя в режим настройки параметров (меню **Parameters**). При вставке каждой новой переменной будет доступно поле **Direction**, где можно указать, что именно создается - вход, выход или локальная переменная.

В отличие от функций, блоки представляют собой объекты с устойчивым состоянием. ISaGRAF устроен таким образом, что блоки одного типа используют один и тот же код, но при этом каждый блок работает со своим собственным набором внутренних переменных. Рассмотрим этот вопрос более детально. Предположим, у нас есть функциональный блок **Factorial**, который служит для вычисления факториала. Мы успешно создали тело блока, описали входные и выходные переменные и теперь хотим использовать его в своей программе. Чтобы сделать это, перейдем в Словарь и добавим новую переменную. Сделать это можно как через меню **Edit->Add Row**, так и нажатием на многоточие под последней переменной.

Задаем имя переменной, например, **factorial_1**, и в качестве типа данных выбираем **Factorial**. Сделать это можно, нажав на вызов диалога со списком всех функций (кнопка "..." возле поля **Type**). Все созданные пользователем функциональные блоки можно найти в секции **User Defined**. В результате будет создан экземпляр функционального блока **Factorial**.



В дальнейшем, когда потребуется использование своего блока в программе (имеется в виду **FBD** диаграмма), нужно будет указать, какой экземпляр следует использовать



Библиотека функциональных блоков

Как уже отмечалось выше, для того, чтобы писать программы на **FBD**, требуется знать набор доступных функций по обработке данных. Давайте более детально изучим базовую библиотеку готовых компонентов ISaGRAF. Библиотека достаточно обширна, поэтому в рамках этой статьи мы лишь сделаем первые шаги.

Арифметические функции

"*" умножение:

Эта функция предназначена для перемножения целых или вещественных величин между собой. Входных параметров может больше двух. Изменить это значение можно в окне вставки нового функционального блока (или редактировании уже добавленного - просто сделайте двойной щелчок мышкой на блоке). Входные переменные должны быть все одного типа. Это значит, что невозможно умножить FLOAT и DINT без предварительного приведения типов.

"+" сложение:

Функция аналогична умножению, за исключением того, что в качестве входных параметров могут выступать переменные типа STRING и TIME. Выполнение функции сложения над строковыми переменными приведет к их конкатенации, над временными - логическим сложением временнЫх интервалов.

"-" вычитание:

Функция вычитает из первого числа второе. Допустимые входные типы данных - все числовые и TIME. Вычитать строки друг из друга нельзя.

"/" деление:

Функция делит первое число на второе и записывает результат в выходную переменную. Входными переменными могут быть численные данные всех типов.

ABS – функция модуля:

Функция вычисления модуля числа от входного значения. Входной величиной может быть любое вещественное число.

ACOS – арккосинус:

Входные параметры - вещественное число в диапазоне от -1.00 до 1.00 включительно. Выходное значение возвращается в радианах и может принимать значения от 0, что означает некорректный ввод, и до π .

ASIN – арксинус:

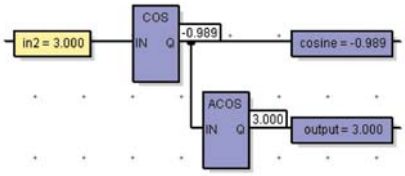
Входной диапазон значений: от -1.00 до 1.00. Выходное значение: 0 - некорректный ввод, - $\pi/2$ до + $\pi/2$.

ATAN – арктангенс:

Входное значение - вещественная переменная в диапазоне от -1.00 до 1.00. Выходное значение: 0 - некорректный ввод, - $\pi/2$ до + $\pi/2$.

COS – косинус:

Входное значение: любое вещественное число. Выходное значение: вещественное число в диапазоне от -1.00 до 1.00.



EXPT – экспонента или возведение в степень:

Блок возведения в степень. Имеет два входных параметра - вещественное основание (in) и целочисленную степень(expt). Результатом операции будет вещественное число, вычисленное по формуле in^{expt}

"=" – сравнение:

Блок производит сравнение двух переменных одного типа. Переменные могут быть целыми, вещественными, логическими или строчными. Выходное значение - логическая

переменная, равная TRUE, если входы равны, или FALSE в противном случае.

LOG – логарифм по основанию 10:

Входное значение представляет собой вещественное число больше 0, выходное значение также будет вещественным числом.

MOD – остаток после деления:

Функция позволяет найти остаток после деления двух целочисленных величин. Блок принимает два входных значения - делимое и базу, причем база должна быть больше 0. Выходное значение будет представлять собой целочисленное неотрицательное число.

SQRT – вычисление корня квадратного:

Входное значение - неотрицательное вещественное число.

RAND – генератор псевдослучайных чисел:

На вход блок принимает целочисленное число, которое определяет диапазон выходных значений. Выборка будет осуществляться среди множества целых чисел из [0, (N-1)], где - N - начальное значение, и результат будет передаваться на выход блока.

TRUNC – отбрасывание дробной части:

Своего рода округление вещественных значений. Если входное значение больше 0, то будет возвращено наибольшее целое, меньшее или равное входному значению. Если входное значение меньше 0, то будет возвращено наименьшее целое, большее или равное входному значению.

SIN – синус числа:

Входное значение представляет собой вещественную переменную, выходное - вещественное число в диапазоне [-1.00;1.00]

TAN – тангенс угла:

Входное значение не может быть равно $\pi/2$. В случае некорректного ввода на выход будет выдано значение 1E+38.

Битовые операции

AND_MASK – побитовая операция AND:

Блок принимает на вход два целочисленных числа, над которыми поразрядно выполняется операция "И". Результат передается на выход.

OR_MASK – побитовая операция OR:

В качестве входных параметров выступают два целочисленных значения. Результат ИЛИ между ними передается на выход.

XOR_MASK – исключающее ИЛИ:

Функция выполняет операцию XOR между двумя целочисленными значениями, результат в виде целого числа передается на выход.

NOT_MASK – побитное инвертирование:

На вход функции подается 32-х разрядное число, для каждого бита которого выполняется инверсия.

ROL – побитовый циклический сдвиг влево с переносом:

Функция принимает два значения - число, над которым будет производиться операция, и количество сдвигов - шагов по перемещению. При каждом шаге производится сдвиг влево одного бита числа. Если указать число 0 или отрицательное число, то сдвиг производиться не будет.

ROR – побитовый циклический сдвиг вправо с переносом:

На вход функции подаются два числа - база и количество сдвигов. Число 0 или отрицательные значения не допускаются, так как не имеют

смысла. Количество сдвигов с переносом может быть от 1 до 31. При значении 1 циклически переместится вправо один бит, при значении 31 - пройдет полный циклический сдвиг всех битов числа и возврат к исходному числу.

SHL – побитовый циклический сдвиг вправо:

Функция аналогична ROL за одним небольшим, но существенным исключением - биты при сдвиге не переносятся. Позиции сдвинутых битов заполняются нулями.

SHR – побитовый циклический сдвиг вправо:

Функция аналогична ROR с таким же исключением, что и в предыдущем случае - биты при сдвиге не переносятся, а позиции сдвинутых битов заполняются нулями.

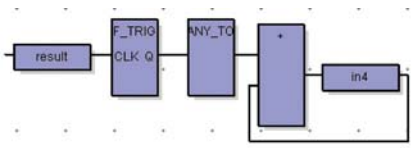
Булевы операции

AND – логическое И:

Функциональный блок, который выполняет операцию "И" над входными логическими значениями. Количество входных каналов может быть более двух.

F_TRIG – триггер по спаду:

Блок имеет один вход, к которому подключается переменная типа Boolean. Выход блока будет равен TRUE, если входное значение изменилось с TRUE на FALSE. Для всех остальных случаев на выходе блока будет формироваться FALSE.



FlipFlop – триггер:

Блок представляет собой обыкновенный триггер. На вход блока подаются два булевых значения - SET и RESET. В зависимости от состояния входов формируется выходное значение.

ODD – функция четности:

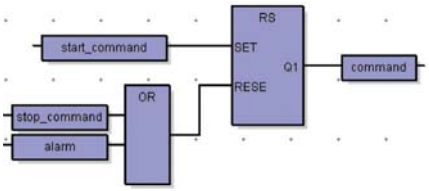
На вход блока подается любое целочисленное число. На выходе будет сформирован логический сигнал TRUE в случае, если входное число четное, и FALSE, если нечетное.

R_TRIG – триггер по переднему фронту:

Входным значением блока является переменная типа Boolean. Выход блока будет равен TRUE, если входное значение изменилось с FALSE на TRUE. Для всех остальных случаев на выходе блока будет формироваться FALSE.

RS – сброс элемента с двумя устойчивыми состояниями:

Блок имеет два логических входа - SET и RESET и выход Q. Выходная переменная связана с входами по следующему закону: если SET = TRUE, то Q устанавливается в TRUE. Если RESET = TRUE, то выход Q устанавливается равным FALSE(доминанта).



SR – установка элемента с двумя устойчивыми состояниями:

Блок имеет два логических входа - SET и RESET и выход Q. Выходная переменная связана с входами по следующему закону: если SET = TRUE, то Q устанавливается в TRUE(доминанта). Если RESET = TRUE, то выход Q устанавливается равным FALSE.

Функции сравнения

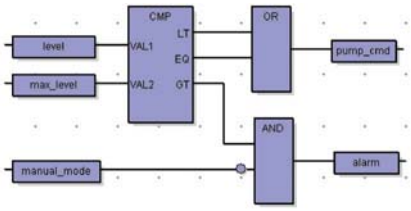
Существует блок функций, предназначенных для проведения сравнения входных переменных. Количество входов для таких блоков всегда равно двум, а выход представляет собой логическое значение. К входам можно подключить переменные типа SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, или STRING.

К числу наиболее простых функций относятся '<'(меньше), '<='(меньше или равно), '>'(больше), '>='(больше или равно), '<>'(не равно), '='(равно). Не будем вдаваться в описание достаточно очевидных операций. Следует отметить, что хотя все функции позволяют сравнивать переменные типа TIME, делать это не рекомендуется. Для этого есть ряд специализированных операций, таких как TON, TP, TOF, BLINK.

Отдельного внимания заслуживают два блока с более "продвинутыми" возможностями.

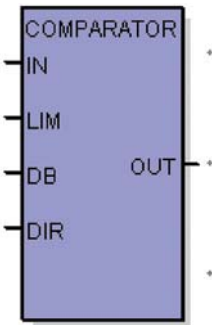
CMP – комплексное сравнение:

Функция предназначена для сравнения двух целочисленных значений и выдачи информации на три логических выхода: "меньше чем", "равно", "больше чем". На одном из выходов всегда будет присутствовать значение TRUE, а два других будут установлены в FALSE. Аналогичного результата можно было бы добиться и комбинацией элементарных операций, но код при этом получится не столь компактным. Можно порекомендовать, в порядке упражнения, попробовать сделать аналогичную программу на основе '<', '=','>'



Comparator – определитель уровня:

В простейшем случае блок сравнивает входное значение с установленным пределом (уставкой), и в случае его превышения выдает сигнал TRUE на выход. Имеется четыре входа: IN - входной сигнал, LIM - верхняя (нижняя) допустимая граница, DB - зона нечувствительности, DIR - логическая переменная, позволяющая задавать один из двух алгоритмов работы.



Вход DIR(DirectActing) отвечает за выбор типа операции. Например, если DIR=TRUE, выход блока будет принимать значение TRUE при IN >= LIM-DB, а при DIR=FASLE блок отрабатывает обнаружение нижней границы и устанавливает выход в TRUE, если IN<=LIM+DB

Мы рассмотрели далеко не полный набор доступных функций, однако даже с его помощью можно написать реальную "боевую" программу. А как для начала, то ничего другого и не надо. В следующий раз мы рассмотрим более "продвинутые" функции, такие как функции обмена данными с другими параллельно работающими целевыми системами, т.е. контроллерами, функции работы с портами ввода/вывода и другим "железным" оборудованием.

КОНТАКТЫ:
т. (044) 492-31-08, 492-31-09
e-mail: info@isagraf.com.ua