ISaGRAF - это очень просто!

(часть III, FBD, продолжение)

Гулько С.В., ХОЛИТ Дэйта Системс, Киев

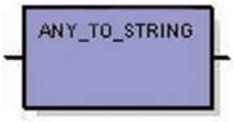
Работа со строками

Попробуйте ответить на вопрос - а зачем нужна работа со строками в системе контроля? Ведь там нет непосредственного обмена данными с пользователем, так как это берет на себя SCADA. Тогда зачем же создавать отдельный тип данных и вводить целый блок функций, предназначенных для работы с текстовой информацией? Ответов может быть несколько. Не секрет, что по мере роста проекта и увеличения его значимости, а также ценности обрабатываемой информации, растет потребность и в протоколировании выполненных системой действий, дабы впоследствии на основе полученного журнала отслеживать состояние и искать какие-либо проблемы. Предположим, на объекте произошла какая-нибудь внештатная ситуация. Причин может быть масса. Конечно, среди них может быть и случайные, практически "непобедимые". Например, человеческий фактор. Но даже и с ним можно достаточно успешно бороться. Однако, зачастую причину следует искать в различного рода неполадках оборудования или огрехах программного обеспечения. Если у Вас есть журнал, найти истину можно, проанализировав значения в различных контрольных точках. Даже если журнал не полный, то есть не протоколируется "все и вся", Вы уже получите отправную точку в поисках первопричины и методов ее устранения. В печальном и достаточно распространенном случае, когда журнала нет вообще, понять, что же произошло, достаточно сложно. Восстановить картину происходящего могут только люди с богатым воображением.

Кроме протоколирования, работа со строками может пригодиться в задачах взаимодействия с оборудованием. Например, с интеллектуальными датчиками или контроллерами сторонних производителей, работающих по своим протоколам. В качестве примера можно было бы привести контроллеры visiCON PRO, где данные на индикаторную панель поступают как раз в виде текстовых строк. Эти же контроллеры имеют собственную систему протоколирования, где также используются строковые операции.

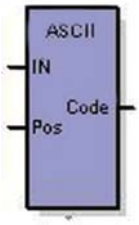
Итак, давайте перейдем непосредственно к предмету нашего разговора - строкам.

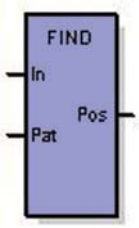
Any_to_string. Как Вам должно быть известно, одной из важнейших функций по работе со строками является преобразователь any_to_string, позволяющий получить текстовое (или строковое) представление любой переменной. Данные, с которыми работает компьютер, достаточно сильно отличаются от тех, которые мы видим на экране или в файле на жестком диске. Предположим, Вы хотите записывать результаты измерений по 2-м каналам в файл с определенной периодичностью. Сделать это можно и простой побайтной записью в файл самих значений переменных. Однако такой метод подразумевает написание парсера (программы, которая в дальнейшем будет преобразовывать бинарные файлы в форму, пригодную для прочтения человеком). Рассматриваемая функция ISaGRAF предоставляет необходимый сервис по преобразованию, так что в файлы будут попадать только текстовые данные, что даст возможность открыть их, не прилагая особых усилий, в любой программе, например в Excel.



Функция имеет один входной и один выходной параметр. Входная переменная может быть любого типа, например, Boolean, float или integer, выходная же представляет собой строку, содержащую результат преобразования. Рассмотрим простейший пример. Подключим к any_to_string логическую переменную, находящуюся в положении "ИСТИНА". В этом случае на выходе функции будет сформировано текстовая строка, содержащая слово "TRUE". Аналогично для "ЛОЖЬ" будет сформировано "FALSE". Что же касается числовых данных, то они будут преобразовываться без изменений - число 1.24 будет трансформировано в строку "1.24"

Ascll. Хотя все и говорят о том, что Unicode это панацея от всех проблем, связанных с кодировками, но как продолжали использовать коды **ascll**, так и продолжают это делать. Связано это с огромным количеством программ, доставшихся по наследству, переделать которые не представляется возможным, да и смысла в этом нет. Например, тех же американцев вполне устраивает кодовая таблица на 127 символов и им нет особо дела до того, что английский (или американский) для кого-то родным языком не является. Это все не очень хорошо, но уж коль подавляющее количество ПО идет из-за рубежа, значит будем принимать чужие правила игры. Функция **ascll** служит для получения ASCII кода любого символа в строке. Как видите, на вход поступает две переменных - первая содержит саму строку, вторая - индекс символа, код для которого нужно вернуть. Следует отметить, что нумерация символов начинается с 1, а не с 0. ASCII код символа возвращается в переменной "code".





Find. Это простейшая функция поиска по шаблону. Разбор

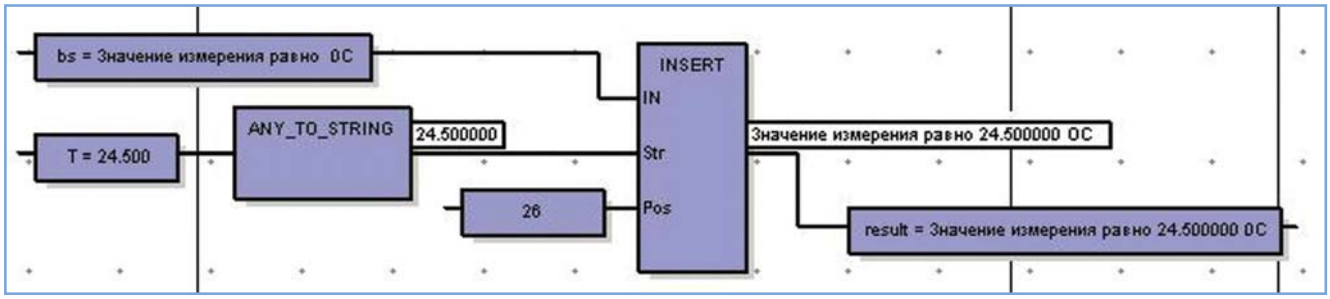
текстовых строк в последнее время становится все более и более популярной темой, так как большое количество внешних сервисов начали возвращать результаты обработки не в виде бинарных массивов, а как текст. Кроме того, часть устройств выдает свои статусные данные все в тех же форматированных строках. Таким образом, наличие хотя бы самых элементарных функций для текстового поиска является сейчас обязательным.

Алгоритмы поиска по строкам совершенствуются с каждым днем, становясь все более эффективными. Есть такое понятие - "регулярные выражения" или "regular expression" (regexp), которое как раз и характеризует сам подход. Идея состоит в следующем - программист задает шаблон на основе определенных правил, далее программный код библиотеки или языка программирования выполняет этот шаблон и возвращает результат. Преимущество такого подхода состоит в том, что нужно иметь только одну универсальную точку входа (или же одну функцию работы со строками) - do_regexp (название взято для примера). Одна и та же функция может быть использована для поиска, замены, поиска с подстановкой, сравнения и подобных им операций. Шаблоны задаются на основе определенного языка, который является более-менее стандартным и общедоступным. Более подробно с идей регулярных выражений можно познакомиться на http://www.regular-expressions.info

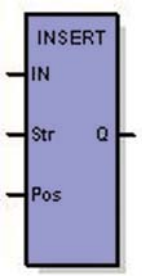
Следует признать, что сервисные возможности, которые предоставляет ISaGRAF, на настоящий момент для текстового поиска являются не слишком полными. Однако, если рассматривать этот вопрос с другой стороны - а нужны ли они вообще большому количеству пользователей? Вопрос к читателям журнала "ПиКАД" - существует ли у Вас необходимость в более расширенных возможностях по обработке строк? Ядро ISaGRAF тем и хорошо, что позволяет расширять его, добавляя новые функциональные возможности. Если у Вас есть в этом потребность - пожалуйста, напишите на info@isagraf.com.ua.

В ISaGRAF функции поиска по шаблону выполняет блок find, который представляет собой поиск по простому текстовому шаблону. Как видите, блок имеет два входа, один из которых служит для задания строки, где будет производится поиск, второй же содержит сам шаблон. Результат выполнения будет помещен в выходную переменную и представляет собой число. Если значение этого числа равно 0, то шаблон в строке найден не был. Значение, отличное от 0, указывает на номер символа, с которого начинается первое включение шаблона в строку. Как видите, сама функция не самая удачная, так как ищется лишь первое совпадение, второе же обнаружено не будет, так что следует быть аккуратным. К чести данной функции следует сказать, что выполняется она достаточно быстро и различает строчные и заглавные буквы.

Рассмотрим небольшой пример. К входам функционального блока find подключено две строки. "Try the best SoftPLC solution in the world - ISaGRAF!!!" и "SoftPLC". Значение, выставленное на выходе будет равно 13, что соответствует началу подстроки "SoftPLC".

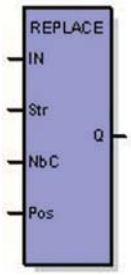


Insert. Кроме поиска подстрок, ISaGRAF предоставляет ряд функций для модификации текстовых данных. Функциональный блок insert предназначен для вставки подстроки в строку с указанной позиции. Результатом операции будет новая строка, содержащая вставку. Помните, что символы начинают нумероваться с 1, а не с 0.



Сегодня многие разработчики ПО идут по пути добавления дополнительных проверок в предлагаемые ими функции. Делается это дабы избежать ошибок случайных или же намеренных со стороны пользователей библиотек. Например, если Вы укажите в качестве начальной позиции для врезки число равное или меньшее 0, то получите пустую строку, что, согласитесь, правильно. Если же Вы попытаетесь произвести insert в позицию, которая больше первоначальной строки (что тоже не вполне корректно), то ISaGRAF выполнит объединение двух строк, что будет аналогично операции объединения.

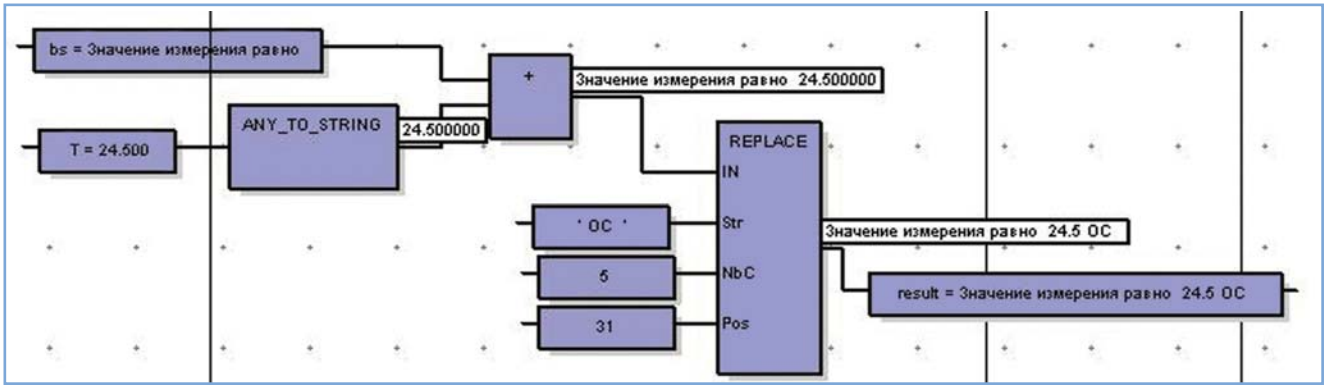
Пример. Предположим, у Вас есть строка-шаблон, содержащая "Значение измерения равно 0C". Необходимо вставить некоторое число перед лексемой "\0C", но после основной надписи. Значение возьмем из вещественной переменной temperature. Для начала преобразуем вещественную переменную в строку с помощью функции any_to_string. Вставка должна производится с позиции 26. Укажем это число в качестве одного из параметров insert. Ваш шаблон подает на первый вход функционального блока, а преобразованное из числа значение - на второй. В результате будет сформирована вполне "читабельная" строка, которую можно выводить на индикаторную панель.



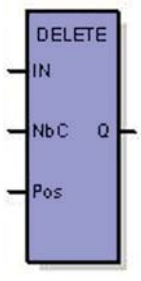
Replace. Там где есть вставка, всегда будет и замена. Данная функция позволит Вам заменить часть строки другой подстрой. На вход функция принимает следующие параметры:

- IN(string) - переменная, над которой будут выполняться действия замены;
- Str(string) - переменная, содержащая подстроку, которая должна быть вставлена;
- NbC(dint) - количество символов, которые должны быть удалены;
- Pos(dint) - позиция первого изменяемого символа.

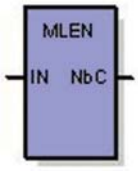
У функции есть всего один выходной параметр и по его состоянию можно судить о результатах выполнения replace.



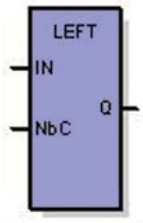
Перед тем, как начать использовать эту функцию в своей работе, следует привыкнуть к несколько необычной логике ее работы. На первом шаге курсор устанавливается в позицию, заданную Pos, после чего затирается NbC символов и на последнем шаге происходит непосредственно сама вставка. Необычность состоит в некоторых ограничениях, которые накладываются на значения Str и Pos. Например, задание Pos=0 к ошибке не приведет, но и ожидаемого результата не даст - при Pos=0 replace потенциально должен был работать аналогично insert, чего нет на самом деле, так как функция возвращает строку IN для Pos<=0. В то же время при указании Pos больше, чем длинна модифицируемой строки, данная функция сработает как конкатенация и Вы получите на выходе результирующую строку, равную слиянию IN и Str. Пример наглядно разъясняет вышесказанное.

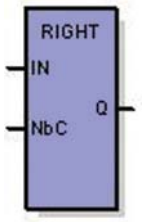


Delete. Функциональный блок, предназначенный для удаления подстрок из начальной строки. Применение достаточно простое, нужно лишь указать строку, начальную позицию и количество символов, которое необходимо удалить и забрать результат с выхода. Защитные ограничения функции такие же, как и у всех - в качестве стартовой позиции для удаления можно использовать 1 или более высокие номера, и удалить больше, чем вся строка, Вам тоже не удастся.



Mlen. Функция вычисления длины строки. Полезный сервисный функциональный блок, если учесть, что размер строки в ISaGRAF ограничен 256 символами. Лучше лишний раз проверить, каким же будет размер строки после выполнения некоторой операции, чем "словить" ошибку времени исполнения. Использовать функцию чрезвычайно просто - на вход подается строка, на выходе получается число символов.





Left, Right. Назначение данных функций практически одинаково, различаются только направления. Входные параметры для обеих функций:

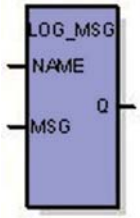
- IN(string) - строка, откуда будут извлекаться данные
- NbC(dint) - число символов для копирования.

В зависимости от типа функции, в результирующую переменную будут скопированы первые NbC символов, начиная от крайней позиции либо слева(для функции left), либо же справа(для функции right). Результат копирования будет размещен в итоговой переменной Q.

Рассмотрим пример, который иллюстрирует применение этих двух функций. Предположим, у нас есть начальная строка "12345678". Подадим ее одновременно на входы блоков left и right, кроме того укажем, что копирование следует производить с 4-ой позиции. Результаты работы функций объединим с помощью блока +. Если Вы запустите этот пример в отладчике, то найдете, что в итоговую переменную было записано значение "56781234"

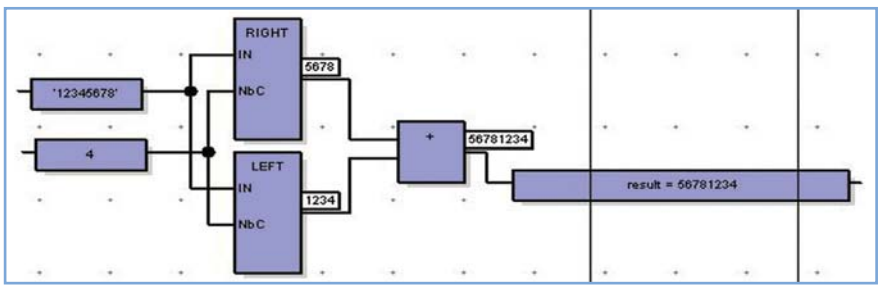
Протоколирование

Вопрос протоколирования происходящих событий поднимался в этой статье уже не один раз и, если рассматривать средства самого ISaGRAF, был бы неполон без упоминания о встроенных средствах самой целевой системы и функции доступа к ним. Не вдаваясь в подробности, так как это тема отдельной, пусть и небольшой, но статьи, можно сказать следующее - целевые системы ISaGRAF имеют собственный сервис для ведения журналов. Сервис обладает гибкими функциями настройки и позволяет выводить сообщения на консоль или файл, в удаленную базу данных или другую систему, обеспечивающую соответствующий прикладной интерфейс. При всей своей гибкости и особенностях настройки, со стороны API сервис представлен всего одной функцией - log_msg, которая позволяет вести протоколирование из тела программы, не тратя время на создание собственной системы логирования.



Log_msg. Как видите, функция явно не перегружена настройками и возможностями. Существует всего два входных параметра:

- name(string) - имя файла(или интерфейса) регистрации, куда будут отправляться сообщения. Эти сообщения поступают на обработку процессу PrintLog, который представляет из себя сервис протоколирования. По умолчанию созда-

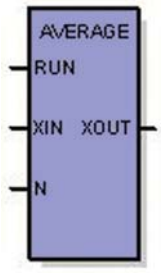


ется файл протокола, который связывается с файловым дескриптором STDERR, что приводит к простому выводу всех сообщений на консоль (точнее на STDERR). Вы можете самостоятельно настроить PrintLog, указав, куда и как должны отправляться данные;

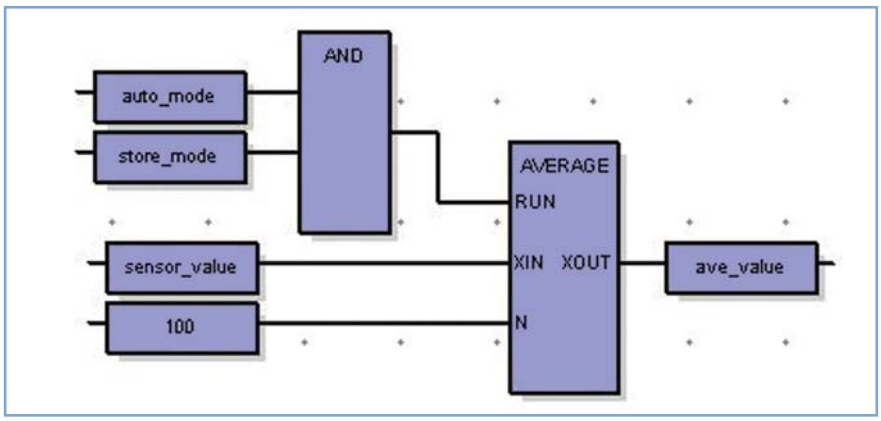
- msg(string) - тело сообщения. Размер сообщения не должен быть больше 94 символов. Кажется, что не так и много, ведь хороший лог должен сопровождаться временными метками, а это тоже место плюс небольшая ручная работа по получению текущего времени и его преобразованию в строку. Однако не все так плохо, например, последнюю операцию PrintLog выполняет автоматически, без участия человека, то есть можно просто отправлять текстовые сообщения и система будет их снабжать временными метками. Что же касается ограничения в 94 байта..."Достаточно длинное тестовое сообщение, иллюстрирующие ограничения по размерам" содержит 77 символов. Конечно, протоколы такого типа в природе встречаются достаточно редко, на практике же рекомендуется применять что-то более лаконичное, например, CSV, где элементами могут выступать результаты измерений.

Функция возвращает статусное значение, позволяющее отследить, было ли записано сообщение в протокол или нет. В принципе, ошибка записи в лог-файл потенциально может произойти, например, если в качестве цели используется сетевая база данных, так что нужно быть готовым и к такому развитию событий, и в особо критических местах делать простой вывод на STDERR. Если же и он не работает - значит наступили действительно темные времена и компьютер или контроллер нуждается в немедленной перезагрузке.

Обработка данных



Average. Функция нахождения среднего значения. Достаточно полезная функция, которая может пригодиться, например, при динамическом формировании уставок для процесса. Данный блок позволяет накапливать значения и на их основе формировать среднее значение поступающей величины. Рассмотрим входы блока более подробно:



Run(Boolean) - управляющий вход всего блока. Значение "TRUE" сигнализирует о том, что необходимо производить накопление значений и расчет среднего на их основе. Установка "FALSE" означает, что необходимо произвести сброс счетчика и быть готовым к началу следующего накопления;

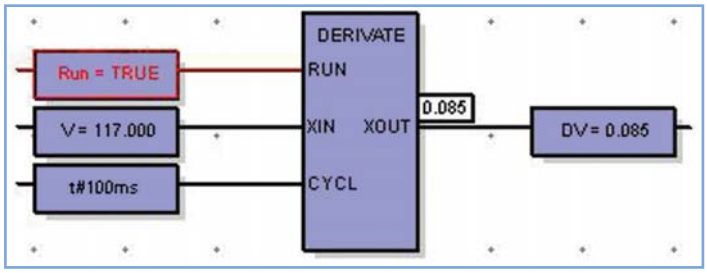
XIN(real) - любая аналоговая переменная;

N(dint) - количество циклов наполнения величины XIN. Максимально допустимое значение N составляет 128. По большому счету, никто не запрещает установить большее количество N, однако следует быть готовым к тому, что по достижению рубежа в 128 точек установившееся среднее будет сброшено и на его место начнет поступать новое среднее, базирующиеся на других значениях.

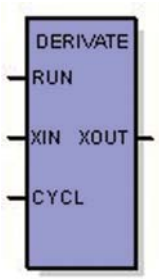
Среднее значение можно снять с выхода XOUT.

Вы можете динамически перенастраивать счетчик, задавая число измерений. Для этого необходимо предварительно сбросить блок, установив "FALSE" на входе Run, после чего задать необходимый N и снова активизировать блок сигналом "TRUE".

Давайте рассмотрим пример {average_example}. Процедуру сброса/запуска блока можно организовать с использованием блока AND, который позволит переключить режимы "на лету". Количество отсчетов в примере задано как константа, однако это вполне может быть и переменное значение, зависящее от алгоритма.



Derivate. Модуль дифференцирования, который производит одноименную операцию над массивом значений на указанном временном интервале. Назначение входов данного модуля:



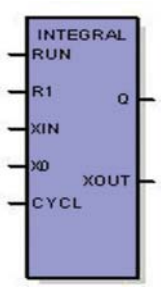
RUN(boolean)- переключатель режимов работы, по сути своей аналогичный одноименному входу модуля average, описанному выше. "TRUE" указывает на то, что следует продолжать дифференцирование, "FALSE" - остановить операцию, сбросить все значения и быть готовым к установке новых настроек;

XIN(float) - любая аналоговая величина, которая будет накапливаться данным блоком для использования в операции дифференцирования;

CYCLE(time) - период дифференцирования. Желательно изначально задавать время больше, нежели время работы одного цикла ISaGRAF. В противном же случае система принудительно будет использовать продолжительность цикла в качестве величины для расчета.

Значение после дифференцирования можно получить с выхода XOUT.

А теперь пример использования данного блока. Как видите, мы установили время дифференцирования заведомо большее времени одного рабочего цикла, плюс управление блоком (вход RUN) выполнено на основе переменной, значением которой можно управлять из другого места программы.



Integral. Функция, как видно из ее названия, предназначена для интегрирования входного сигнала за заданный временной интервал. Как известно, существует 1000+1 метод расчета интеграла, в которых, в зависимости от алгоритма, можно получить либо высокую точность, либо минимальное время вычисления. К сожалению, нельзя сказать, какой именно алгоритм используется для вычисления площади фигуры и какова погрешность данного блока, так что остается уповать на разработчиков, выбравших золотую середину между точностью и производительностью. Входные значения блока следующие:

Run(boolean) - вход управления блоком. Если установлен в "TRUE", то производится интегрирование, в противном случае на выходе просто удерживается одно и то же значение;

R1(Boolean) - основной сброс, управляющий вход, позволяющий управлять блоком на уровне произведения "перезагрузок". То есть если на R1 поступит импульсный сигнал, блок произведет сброс значений и интегрирование начнется заново;

XIN(real) - любая аналоговая величина;

X0(real) - начальная точка интегрирования;

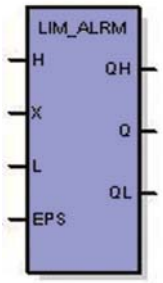
Cycle(time) - период, за который следует произвести интегрирование. Следует отметить, что если данный параметр меньше, чем время цикла выполнения для данного Ресурса целевой системы, то период интегрирования будет равен времени цикла.

Выходные параметры блока:

Q(Boolean) - значение, обратное R1;

XOUT(Boolean) - рассчитанное значение интеграла, равное площади фигуры от X0 до последней задействованной точки.

Достаточно часто в задачи автоматизации входит отслеживание уровня сигнала и проверка его текущего значения на пересечение нижней или верхней границы. Безусловно, сделать самостоятельно блок, включающий несколько условий, особых усилий не составит, однако зачем тратить на это свое время, если это все уже сделано командой ISaGRAF?



Lim_alrm. Блок слежения за текущим значением на основе заданных границ. Не вдаваясь в излишние рассуждения, давайте рассмотрим входные и выходные параметры данного блока и тогда все станет совершенно ясно. К входным параметрам относятся следующие:

H(real) - значение верхнего предела;

X(real) - текущее значение, поступающее на блок;

L(real) - значение нижнего предела;

EPS(real) - величина гистерезиса. Одно из условий - это значение должно быть больше 0.

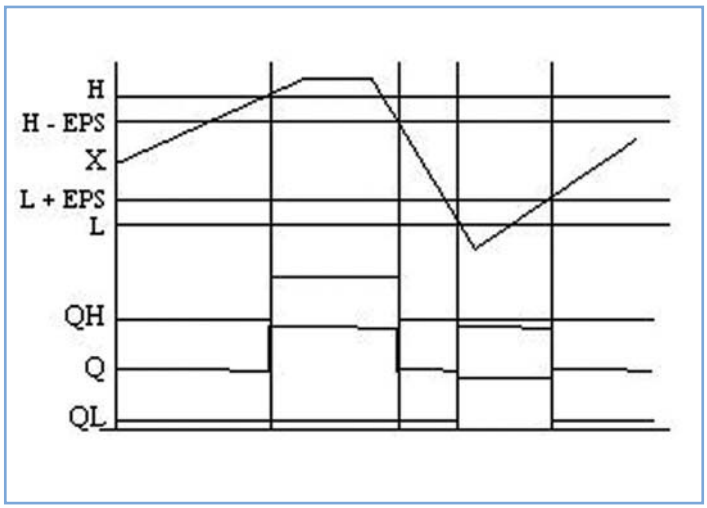
Как видите, названия говорят сами за себя. При настройке блока просто укажите нижнюю и верхнюю границы, при превышении которых должна срабатывать тревога, задайте уровень гистерезиса и получайте результаты с выходов блока:

QH(Boolean) - устанавливается в "TRUE", когда X пересекает верхний предел;

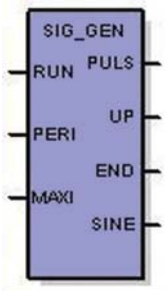
Q(Boolean) - устанавливается в "TRUE", если значение находится в пределах границ;

OL(Boolean) - устанавливается в "TRUE", если текущее значение меньше нижнего предела.

Дельта гистерезиса, использующаяся как для нижнего предела, так и для верхнего, равна половине параметра EPS. Для более глубокого понимания работы данной функции приведем временную диаграмму:



Иногда возникают задачи, требующие генерации сигналов стандартной формы. Трудно назвать реальные приложения, где подобные генераторы могли бы найти применение, можно лишь сказать, что очень часто подобные блоки используются для написания разного рода тестов и демонстраций. Что ж, тоже неплохое подспорье, ускоряющее процесс разработки. Вот почему следующий блок следует взять на заметку.



Sig_gen. Генератор, который позволит Вам создавать синусоиды, пилы и булевые мигающие сигналы. По большому счету, данный генератор можно использовать для построения сигналов гораздо более сложной формы, однако это потребует нетривильаных усилий. Блок имеет три управляющих входа:

RUN(Boolean) - управляющий, который переводит блок в режим генерации при поступлении сигнала "TRUE" или же производит сброс выходов при получении "FALSE";

Period(time) - продолжительность генерации одного образа;

Maximum(dint) - максимальное значение счетчика;

Выходные параметры блока:

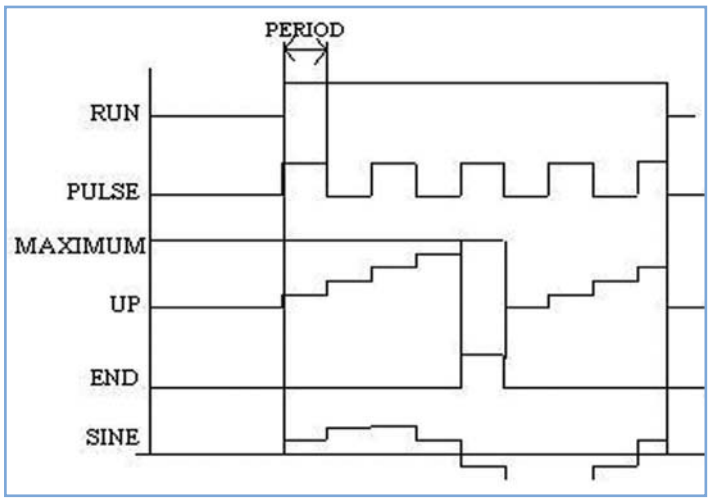
Pulse(Boolean) - инвертируется после каждого образа;

Up(dint) - нарастающий счетчик, увеличивающийся при каждом новом образе;

End(Boolean) - принимает значение "TRUE", когда нарастание сигнала закончилось;

Sine(real) - сигнал синусоидальной формы, где полупериод равен времени счета.

Когда счетчик достигает максимума, он перезапускается с нуля. Таким образом, End удерживает значение "TRUE" в течение только одного значения Period. Для лучшего понимания работы данной функции обратите внимание на временную диаграмму, которая иллюстрирует работу генератора в нескольких режимах:



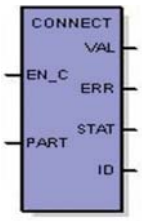
Анализируя тенденции развития информационных технологий, можно заметить, что все мы все дальше движемся в сторону распределенных вычислений. Не стала исключением и отрасль автоматизации. При всей своей консервативности и приверженности проверенным временем технологиям, даже эта отрасль стала постепенно вбирать некоторые ранее не используемые идеи. Это происходит не так быстро, да и не та эта сфера, где стоит торопиться, ошибка здесь не решается перезагрузкой, и зачастую стоит больших денег, а то и человеческой жизни.

Что такое распределенные вычисления в рамках ISaGRAF? Напомним, что задачи выполняются в как называемых Ресурсах (процессах), которые существуют в рамках Конфигураций (физических контроллеров). Ресурсы выполняются по умолчанию в изолированном окружении. В рамках одного проекта Ресурсы равнозначны и находятся как бы в едином пространстве. Группировка Ресурсов на конкретных контроллерах происходит в Конфигурации, однако и это не нарушает картины единого пространства. Сразу возникает вопрос - если Ресурсы изолированы, то как же они могут общаться с друг другом? Очень просто - существует по крайней мере два пути. Первый подразумевает использования механизмов binding-а, когда настраиваются связи между переменными из разных конфигураций. Это наиболее удобный метод, однако он требует дополнительных затрат на приобретение ПО. Многое из того, что делается автоматически, можно сделать вручную. Да, это будет не столь удобно, но это будет работать, что, согласитесь, иногда бывает важнее удобства.

ISaGRAF предоставляет несколько функций, позволяющих передавать данные между Ресурсами.

Коммуникации между Ресурсами

Connect. Перед тем, как начать передавать данные между Ресурсами, необходимо установить связь. Сделать это можно с использованием функции connect, указав в качестве входных параметров имя Ресурса, к которому Вы хотите подключиться и/или дав разрешение на внешнее подключение.



Имя Ресурса, к которому Вы хотим подключиться, передается в параметре Partner. Для этого поля задан специальный формат подачи имен - 'НомерРесурса@Адрес'. Например, '10@10.0.2.10', где 10 - номер Ресурса, а 10.0.2.10 - Адрес. Ес-

ли же Ресурс выполняется на том же контроллере что и инициатор соединения, Адрес можно не указывать, достаточно лишь номера.

Выходные параметры данного блока содержат статусную информацию, которая позволит узнать, было ли установлено соединение или же возникла ошибка, и в чем ее причина:

Valid(Boolean) - если выход принял значение "TRUE", значит ID соединения правильный и можно начинать/продолжать работу;

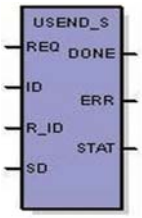
Error(Boolean)- если установлено в "TRUE", во время работы произошла ошибка, информацию о которой можно получить из поля Status;

Status(dint) - поле, содержащие более расширенную информацию о возникшей ошибке. Ознакомимся с этими кодами.

0 - подключение завершено успешно,
1 - ожидание ответа,
2 - слишком много соединений CFB,
3 - нет готовности для нового подк лючения,
4 - нарушение подключения,
5 - плохой партнер .

Если подключение потерпело неудачу, то новое подключение автоматически не создается, должен быть обнаружен передний фронт параметра EN_C

После того, как соединение было создано, можно приступать к передаче данных. Посылка данных осуществляется с использованием функции usend_s.



Usend_s. Функция служит для передачи строки на удаленный или локальный Ресурс. Рассмотрим входные параметры блока:

Req(Boolean) - отправлять запрос по переднему фронту или нет;

ID - идентификатор канала подключения, возвращенного функцией connect;

R_ID(string) - идентификатор удаленного Ресурса внутри канала;

SD(string) - посылаемая строка.

Следует упомянуть, что перед вызовом usend_s обязательно надлежит установить соединение с использованием connect, причем сделать это нужно в том же цикле целевой системы.

Блок имеет следующие выходные параметры:

Done(Boolean) - если принял значение "TRUE", значит выполнение прошло успешно;

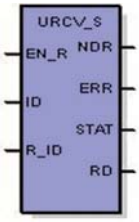
Error(Boolean) - если "TRUE", значит возникла определенная ошибка, более детальную информацию можно получить в поле Status;

Status(dint) - статусная информация о выполненном последнем действии.

Принимает следующие значения:

0 - посылка завершена успешно;
1 - процесс посылки;
2 - неправильный идентификатор;.
3 - неготовность к посылке;
6 - нарушение диалога;
7 - нарушение посылки.

Если посылка не произведена, новая посылка автоматически не делается, ожидается передний фронт параметра Req.



Urcv_s. Функциональный блок, отвечающий за примем данных. Его входные и выходные параметры практически идентичны usend_s:

EN_R(Boolean) - разрешение на получение данных;

ID - идентификатор канала подключения, возвращенного функцией connect;

R_ID(string) - идентификатор удаленного Ресурса внутри канала;

RD(string) - принятая строка;

NDR(Boolean) - если принял значение "TRUE", значит получена новая строка;

Error(Boolean) - если "TRUE", значит возникла определенная ошибка, более детальную информацию можно получить в поле Status;

Status(dint) - статусная информация о выполненном последнем действии.

Принимает следующие значения:

0 - прием завершен успешно;
1 - ожидание сообщения;
2 - неправильный идентификатор;
3 - неготовность к приему;
6 - ожидание сообщения;
7 - нарушение диалога.

Рассмотрим два маленьких примера по отправке и получению сообщения между двумя Ресурсами. В первом случае Resource1 посылает строку, во втором - Resource2 принимает строку.

В этой статье рассмотрена лишь малая часть возможностей ISaGRAF. Целью было не научить чему-то конкретному, а, скорее, показать, какие возможности потенциально существуют и доступны простым разработчикам.

Пробуйте, экспериментируйте, двигайтесь вперед, не бойтесь задавать вопросы - спрашивайте, учитесь и учите сами!

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