ISaGRAF - это очень просто !

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

Программный продукт, о котором пойдет речь, уже во многом известен в нашей стране и версия 3.4x неплохо освоена нашими инженерами и интеграторами, т.к. именно она поставляется в комплекте с наиболее популярными в нашей стране PAC контроллерами семейства i-7000/i-8000/WinCON/LinCON.

В настоящем цикле статей мы переходим на следующую ступеньку и начинаем освоение версии 5.0 этого замечательного продукта.

Предположим, Вам необходим инструмент для создания приложений для программируемых контроллеров автоматизации (Programmable Automation Controller, PAC), и Вы выбрали программный пакет ISaGRAF. Отлично, но каков будет следующий шаг, что делать дальше? Как достичь желаемого результата - получить прикладную программу, которая, быть может, будет управлять технологическим процессом на заводе или двигателем на борту атомной подводной лодки? Мы поставили перед собой цель помочь заинтересованным читателям в освоении этого многогранного продукта и постараемся осветить на страницах нашего издания в цикле статей "ISaGRAF - это очень просто!" различные аспекты его применения, начиная со знакомства со средой разработки, создания распределенных приложений и работы с различным аппаратным обеспечением.

Прежде всего, несколько слов о том, что же именно представляет собой сама рассматриваемая технология. ISaGRAF относится к классу программного обеспечение SoftPLC, которое позволяет превратить встраиваемый компьютер в полнофункциональный программируемый контроллер автоматизации (PAC), способный решать задачи PID-регулирования, обработки ввода/вывода и коммуникаций по различным каналам связи. Одно из требований, предъявляемых к подобным решениям - легкость программирования и переносимость программ. Легкость в данном случае значит абстрагирование от низкоуровневых подробностей разработки программы и концентрация исключительно на логике процесса. Ведь согласитесь, в конечном счете, Вы делаете программу, которая будет выполнять какой-то конкретный алгоритм, а не просто работает с портами ввода/вывода. Создав программу, работающую не с аппаратурой, а исключительно с переменными, Вы получите код с очень высокой степенью портируемости, который можно легко перенести на другую платформу, если такая необходимость возникнет. Достигнуть такой гибкости позволяет разделение технологии ISaGRAF на две основных составляющие: среду разработки и среду выполнения.

Среда выполнения представляет собой виртуальную машину, своего рода интерпретатор кода, которая служит прослойкой между аппаратной частью контроллера и программами пользователя. Когда звучит фраза - "этот контроллер поддерживает ISaGRAF", то это означает, что производитель уже установил среду выполнения на свое оборудование и оно готово обрабатывать пользовательские программы. В зависимости от используемого в PAC аппаратного и/или программного обеспечения, в контроллере можно запустить и больше одной виртуальной машины. В такой мультипрограммной конфигурации задачи пользователя будут выполняться в параллельном режиме, образуя как бы несколько дополнительных контроллеров. Вопрос обмена данными и синхронизации в случае необходимости решается ISaGRAF практически без участия программиста, а точнее сказать, разработчика программы, ибо классического текстового программирования с его бесконечными листингами здесь нет и в помине. Именно в этом смысле в дальнейшем изложении используется термин "программист". Единственное, что потребуется от разработчика программы, так это указать, какие переменные из удаленного проекта будут доступны в текущем проекте. Все остальное возьмет на себя ISaGRAF.

Программы же создаются в среде разработки, которая получила название ISaGRAF Workbench. Это интегрированное средство разработки, содержащее редактор кода, отладчик, средство контроля версий и еще множество других, не менее полезных для программиста инструментов. Именно в нем создаются программы управления, описывается вся логика технологического процесса и именно Workbench компилирует код для контроллера.

В качестве основы для изучения мы решили выбрать пятую версию ISaGRAF. Сделано это, прежде всего, потому, что она объединяет в себе все возможности более ранних версий, позволяя при этом строить приложения из функциональных блоков IEC 61499. Как нам известно, эта версия сейчас находится в стадии модификации и ожидаемая в скором времени обновленная пятая версия станет стандартным инструментом и заменит младшие версии.

Начнем наше изложение с терминологии. В программировании принято разбивать большое приложение на более мелкие составные

части. Функции, процедуры, подпрограммы. Все они служат для одного - правильно организовать основную программу, сделать ее более понятной для чтения и дальнейшего сопровождения. Подобные организационные единицы присутствуют и в ISaGRAF, для них даже есть соответствующий термин - Program Organization Unit (или более кратко - POU). Любой элемент кода, вынесенный в отдельную функцию или функциональный блок, в ISaGRAF является POU.

ISaGRAF-совместимый контроллер имеет по крайней мере одну предустановленную среду выполнения, так называемый target, хотя их может быть и несколько. Именно благодаря target-системе созданная инженером программа будет работать с оборудованием, проводить вычисления, общаться с другими устройствами - одним словом, выполнять полезную работу. Совокупность POU, индивидуальных настроек плат ввода/вывода и список переменных, специфических для одной среды выполнения, в терминах ISaGRAF называется Ресурс. Уместно подчеркнуть, что в пределах одного контроллера Ресурсов может быть несколько, и они будут выполняться в параллельном режиме, синхронизируясь по времени и обмениваясь данными в случае необходимости. Помимо Ресурса существует еще одно понятие - это конфигурация, описывающая целевую платформу или тот конкретный контроллер, на котором будет выполняться задача. Так как Конфигурация является наиболее приближенным к реальному миру понятием, то нужно будет указать, как именно соединены контроллеры в сеть, какие сетевые интерфейсы и с какими параметрами следует задействовать для того, чтобы Ресурсы одной Конфигурации могли взаимодействовать с Ресурсами из другой, быть может, удаленной от первой на многие километры.

На вопрос "Как изучать ISaGRAF?" нельзя ответить однозначно - можно лишь задать встречный вопрос: "А для каких целей?". Почему? Все дело в том, что технология ISaGRAF достаточно обширна и предназначена для двух классов разработчиков. Первые будут использовать ISaGRAF для создания систем управления, и именно на них ориентирован наш цикл статей. Разработчики из второй группы, менее многочисленной, занимаются разработкой ядра - это не есть вопрос нашего интереса в данном случае.

Каким образом учить ISaGRAF? Практикуясь! Создавая поначалу простые, элементарные программы, постепенно совершенствуя и оттачивая навыки - только так можно достичь успеха. Вот и в нашем цикле мы сделаем акцент на практику. У Вас может быть книга внушительных размеров, где очень точно расписаны все пункты меню, кнопки и закладки, подробнейшим образом изложен синтаксис языка и возможности отладчика, но после ее прочтения в голове ничего не остается. Книги подобного рода незаменимы в качестве справочника, куда заглядывают в случае необходимости. Нам же хочется сделать Ваше обучение более живым, поэтому ставка будет делаться на практическое применение ISaGRAF в работе.

В этом практикуме мы кратко рассмотрим графический интерфейс программы, не вдаваясь в излишние подробности относительно той или иной кнопки или закладки. Делается это намеренно, чтобы не засорять текст большим количество описаний, в настоящий момент просто ненужных. Заинтересованный читатель сможет сам ознакомиться со всеми этими возможностями, мы же будем возвращаться к ним по мере необходимости.

Наша прелюдия несколько затянулась. Ну что, запускаемся?

Первые ощущения после запуска программы и пяти минут работы, особенно у тех, кто уже видел и работал с предыдущими версиями - "жизнь стала проще, жить стало веселее". Инструменты вынесены на лицевую панель и снабжены понятными иконками, переработан механизм контекстных (вызываемых правой клавишей мыши) меню, редактор стал более привлекательным и удобным. Но исключительно внешние изменения хороши для программ класса "ширпотреб", ISaGRAF же представляет собой инструмент для профессиональных разработчиков в области автоматизации и контроля, так что если и искать внешние изменения, то рассматривать надо вопросы продуманности и законченности самого пользовательского интерфейса, его эргономичности и удобства.

ISaGRAF 5.0 выгодно отличается от своих более ранних версий наличием очень удобного навигационного меню. Построенное в виде дерева, оно позволяет быстро получить доступ ко всем элементам проекта - исходному тексту, объявлениям, конфигурации оборудования и сетевым протоколам. В стандартном режиме работы (по стандарту IEC 61131-3) Вы можете получить доступ к двум независимым режимам отображения проекта - ***Hardware Architecture*** и ***Link Architecture***. Вы можете переключаться между режимами в любой момент времени - например, в текущий момент Вы работаете с конфигурацией контроллера, выбираете, какое оборудование необходимо для реализации задачи, задаете параметры конфигурации сети, а в другой момент - устанавливаете связи между переменными, находящимися в разных Ресурсах или же Конфигурациях.

В режиме *Link Architecture* Вы увидите созданные Ресурсы и соединительные линии между ними. Эти линии обозначают, что между Ресурсами будет установлена связь, предназначенная для обмена данными. Ресурс, как одна из важнейших частей проекта, обладает большим количеством конфигурационных опций. Их можно вызвать, кликнув по нему правой кнопкой мыши и выбрав пункт меню Properties (мы вернемся к этому вопросу в дальнейшем).

Режим *Hardware Architecture* позволяет "объять необъятное". Каким образом получить работающую программу, часть которой функционирует на Linux, другая на QNX и третья что-то делает на Windows? Правильно, воспользоваться ISaGRAF! Конечно, Вам нет необходимости вникать в тонкости, что же за операционная система используется на контроллере или есть ли она там вообще, но сути это не меняет - *Hardware Architecture* отображает различные PAC контроллеры, в которые будет загружено приложение. Кроме самих PAC в этом разделе Вы можете настроить физические каналы обмена данными между всеми участниками. Например, один PAC может быть связан со вторым через Ethernet, а с третьим - обмениваться данными по CAN.

Узел ***Binding List*** служит для отображения связей между переменными в разных Ресурсах. Механизм связывания переменных в ISaGRAF является очень полезным, можно даже сказать необходимым, при реализации крупных проектов. Лучше всего это рассмотреть на простом примере - предположим, у Вас есть два контроллера, связанных через Ethernet. В соответствии с алгоритмом работы и на первом, и на втором PAC должен запус-

каться специфический процесс контроля и анализируется возникновение некоторого события (состояние "флага"). Решение же о состоянии "флага" может принять исключительно первый контроллер, но сам результат важен и для второго. Разрабатывая программу на стандартных языках программирования, Вам пришлось бы очень тесно познакомиться с протоколом TCP/IP или же углубиться в более высокоуровневые технологии обмена данными, например RPC. Это все, безусловно, очень интересно и стоит изучения, хотя бы для расширения собственного кругозора, но задача должна быть решена уже сейчас (как говорится, еще вчера), а для этого нужны уже иные инструменты. ISaGRAF позволяет, введя несколько параметров и проведя всего несколько соединительных линий, обеспечить связь переменных обоих контроллеров и быстро получить надежное решение для обмена данными между ними.

Перемещаясь ниже по дереву, Вы увидите те *Конфигурации* (аппаратные контроллеры) которые будут использоваться в проекте. В любой момент можно добавить новый, отредактировать или же удалить присутствующий PAC из проекта. Внутри каждой *Конфигурации* имеется список *Ресурсов*. Должен быть как минимум один *Ресурс* для Конфигурации, что же касается большего количества, то это уже зависит от поставщика оборудования - поддерживают ли их контроллеры параллельное выполнение задач или нет. Каждый *Ресурс* имеет свой собственный список используемого оборудования, переменных и программ. Также Вы можете управлять созданными *Функциями* и *Функциональными блоками*.

Любая создаваемая Вами программа будет оперировать данными и не суть важно, откуда они было получены - посредством измерения, вычисления или же через привязку к удаленной переменной в другом *Ресурсе* или *Конфигурации*. Все переменные должны быть определены до начала работы с ними.

Сделать это можно с помощью ***Словаря***. К стандартным типам относятся логические (BOOL), целочисленные (BYTE, DINT, INT, DWORD, LINT, LWORD, SINT, ULINT, UDINT, UINT, USINT, WORD) и вещественные (LREAL, REAL) переменные, а также типы дата (DATE), время (TIME) и строки (STRING). Типы имеют разную длину и поэтому предназначены для хранения переменных строго заданного размера. В таблице приведены диапазоны применимости того или иного типа:

Название	Тип данных	Диапазон
BOOL	Логический	Либо TRUE, либо FALSE
BYTE		8 бит
DATE	Дата	Диапазон дат от 01/01/1970 до 01/18/2038
DINT	Целочисленный	От -2147483648 до 2147483647
DWORD		32 бита
INT	Целочисленный	От -32768 до 32767
LINT	Целочисленный	От -9223372036854775808 до 9223372036854775807
LREAL	Вещественный	От -1.7E -308 до 1.7E +308
LWORD		64
REAL	Вещественный	32 бита
SINT	Целочисленный	От -128 до 127
STRING	Строка	256
TIME	Таймер	Временной интервал от 0 до 1193 часов 2 минут 47 секунд 294 мкс
UDINT	Целочисленный	От 0 до 4294967295
UINT	Целочисленный	От 0 до 65535
ULINT	Целочисленный	От 0 до 18446744073709551615
USINT	Целочисленный	От 0 до 255
WORD		16

Как видим, диапазон значений достаточно широк и этого набора вполне хватает для решения задач из реальной жизни. Однако человек привык работать с более абстрактными вещами, нежели компьютер, и инженеру удобнее разрабатывать программу с применением более высокоуровневых типов, описывающих процесс в целом (например, "Состояние Бака №1"), а не отдельных его компонент ("температура в зоне датчика 1").

Следующий элемент в палитре инструментов ISaGRAF, заслуживающий внимания на начальных этапах рассмотрения, это инструмент контроля версий. "Мне кажется, на прошлой неделе этот код работал правильно... Интересно, что же это я такое сделал, что... И где мне найти старый вариант после усиленного позавчерашнего штурма?". Думаю, каждый разработчик когда-нибудь сталкивался с подобным и сделал соответствующие выводы. Программное обеспечение, хранящее общую базу исходного кода и историю его модификации, не случайно становится обязательным инструментом профессионалов в области программирования. CVS, RCV, Microsoft SourceSafe... - продолжать можно и дальше. Конечно, проекты ISaGRAF не являются обыкновенными текстовыми файлами, с которыми, в большинстве своем, работают перечисленные выше системы, поэтому тут нужен иной подход. Компания ICS Triplex ISaGRAF предлагает свою, ориентированную специально под Workbench, систему контроля версий. Используя ее, Вы сможете вести запись истории модификаций проекта. Предположим, сегодня был удачно завершен один из этапов работы, все компоненты были подвергнуты тестированию, с которым успешно справились. После этого следует запустить *Check-In* и сохранить работу. Записанные изменения можно будет извлечь по прошествии любого периода времени. Еще одним полезным свойством *Check-In* является просмотр отличий. Сравните, что же отличало старую, работающую версию ПО от более современной, но ошибочной, и найдите решение даже по столь минимальным признакам как разница в одну строчку кода или одну переменную. *Check-In* позволяет работать как с *Конфигурацией*, так и с *Ресурсами* и POU.

Изюминкой технологии ISaGRAF является ее работа с устройствами ввода/вывода. Привяжите переменную к выбранному каналу в используемом УСО и ISaGRAF возьмет на себя все

тонкости низкоуровневого программирования, предоставив Вам заниматься непосредственно логикой программы. Безусловно, каждая *Конфигурация* имеет свой собственный список используемого оборудования, уникальных настроек и связанных переменных.

Следующими полезными кнопками на панели инструментов являются *Загрузка*, *Отладка*, *Симуляция* и *Изменения* в режиме реального времени.

Все эти инструменты выполняют схожие функции - либо загружают проект в PAC, либо же производят отладку этого проекта в виртуальном контроллере. Перед загрузкой приложение должно быть откомпилировано в переносимый байт-код (так называемый TIC-код). Процессу компиляции предшествует проверка на наличие ошибок в синтаксисе, объявлении переменных и выявлению прочих неточностей разработчика.

Выше мы очень кратко познакомились с инструментами, которые постоянно приходится использовать в повседневной работе. Давайте сейчас создадим маленький проект, который позволит сделать первый шаг в мир ISaGRAF.

Предположим следующую ситуацию - есть технологический процесс, который запускается только в случае включения двух рубильников. Другими словами - если включены оба рубильника, то происходит запуск, если включен только один - система никак не реагирует. Задача решается в терминах логической алгебры: ВЫХОД = РУБИЛЬНИК1 AND РУБИЛЬНИК2. После того, как был продуман алгоритм, следует выбрать тип переменных, наиболее соответствующий решаемой задаче. Здесь сделаем небольшое отступление на тему "типы переменных".

Как уже отмечалось выше, ISaGRAF предоставляется 18 базовых типов различной размерности и назначения. К выбору типа переменной, где Вы будете хранить свои данные, следует подойти достаточно ответственно. Для математических операций в нашем распоряжении есть целочисленные и вещественные типы разной длины. Целочисленные типы отлично подойдут в качестве счетчиков каких-либо событий или объектов, например, для учета количества прошедших по конвейеру упаковок. В переменных целого типа Вы сможете хранить коды ошибок или состояния программы, которые потом будут "извлекаться" и анализироваться SCADA-системами. Однако далеко не все в реальном мире имеет дискретную природу и хороший пример тому - измеряемые физические величины. Температура, давление, влажность, освещенность, скорость - все они и масса других величин имеют вещественный характер. Измерив, например, температуру и получив значение -273.12°C, его нужно будет сохранить в переменной типа REAL или DOUBLE, так как измерение имеет значимые цифры после запятой. Целые типы имеют разную длину, вещественные - длину и точность. Чем больше точность переменной или диапазон принимаемых ею значений, тем больше места она будет занимать в памяти. Конечно, в современных контроллерах теперь имеются достаточно большие объемы ОЗУ и может показаться, что применение LINT и LREAL во всех случаях может облегчить жизнь. Однако это не так. Проблема носит достаточно низкоуровневый характер, но меньше от этого не становится. Все дело в том, что в большинстве случаев разрядность центрального процессора контроллера составляет максимум 32 бита (а зачастую и того меньше). Загрузить целиком в один регистр CPU переменную размером 64 бита не представляется возможным и задачи по выполнению операций над ней (сложение, вычитание, деление, умножение) берет на себя программное обеспечение. Специальные библиотеки для работы с большими числами выполнят все арифметические действия над данными и на малоразрядных процессорах - проблем здесь нет никаких, эта технология существует уже давно и оптимизирована по максимуму. Однако на это будет тратиться дорогое процессорное время, которое можно использовать с большей пользой. Так что использовать повсеместно переменные LINT и LREAL не стоит. Кроме максимализма иногда встречаются поборники минимализма. Пытаясь воспользоваться переменными наименьшего размера, инженер может просмотреть или попросту не учесть некоторые ситуации, что потенциально может привести к сложно обнаружимым ошибкам времени выполнения. Вывод: разработчик должен знать, с какими данными предстоит работать его программе, чтобы на основании этого выбрать наиболее подходящие типы.

Вернемся к нашей задаче - в ней переменные принимают только два состояния: ВКЛЮЧЕНО или ВЫКЛЮЧЕНО. Это дает возможность воспользоваться для их описания и хранения типом BOOLEAN. В данном простом случае все оказалось и очевидно, и просто. Итак, мы определились с типом - давайте корректно объявим переменные. Следует помнить, что в ISaGRAF, помимо типа, переменной ставится в соответствие еще несколько дополнительных параметров. Их достаточно много, но для нашей задачи важно только такие свойства, как Атрибут и Направление.

По направлениям переменные делятся на три типа: входные, выходные и внутренние. Если переменная объявлена как входная, то это означает, что она может быть связана с входными каналами УСО. В начале каждого рабочего цикла ISaGRAF будет проводить "сканирование" оборудования и “заносить” значения измерений непосредственно в переменную. Программист же будет работать только с переменными, минуя прямое взаимодействие с аппаратным обеспечением. Выходная переменная является полной противоположностью входной. Изменение значения такой переменной в ходе выполнения программы будет изменять физическое значение на выходном канале платы УСО. Внутренняя переменная не подключается к оборудованию, а используется для работы. В них обычно хранятся результаты вычислений и прочие промежуточные данные.

Применив вышесказанное к нашему первому проекту можно заметить, что переменные РУБИЛЬНИК1 и РУБИЛЬНИК2 будут "входными", а переменная ВЫХОД - соответственно, иметь тип "выходной". Это сделано для того, чтобы связать их с каналами УСО и приблизить задачу к реальности.

Теперь переходим в режим *Словаря* . Перед Вами появится окно, содержащее несколько вкладок и дерево. Нас сейчас будет интересовать вкладка *Переменные* (открывается по умолчанию). Дерево переменных предназначено для группировки объектов внутри *Ресурсов* и *Конфигураций*. Давайте создадим три переменных типа BOOLEAN. Для этого перейдите в *Variables ->Resourse1 (Config1) -> Any groups -> All variables*. В таблице, которая появится справа, сделайте двойной щелчок мышкой по многоточию. Появится строка, приглашающая ввести описание новой переменной. Сначала введите имя переменной. Как правильно именовать перемен-

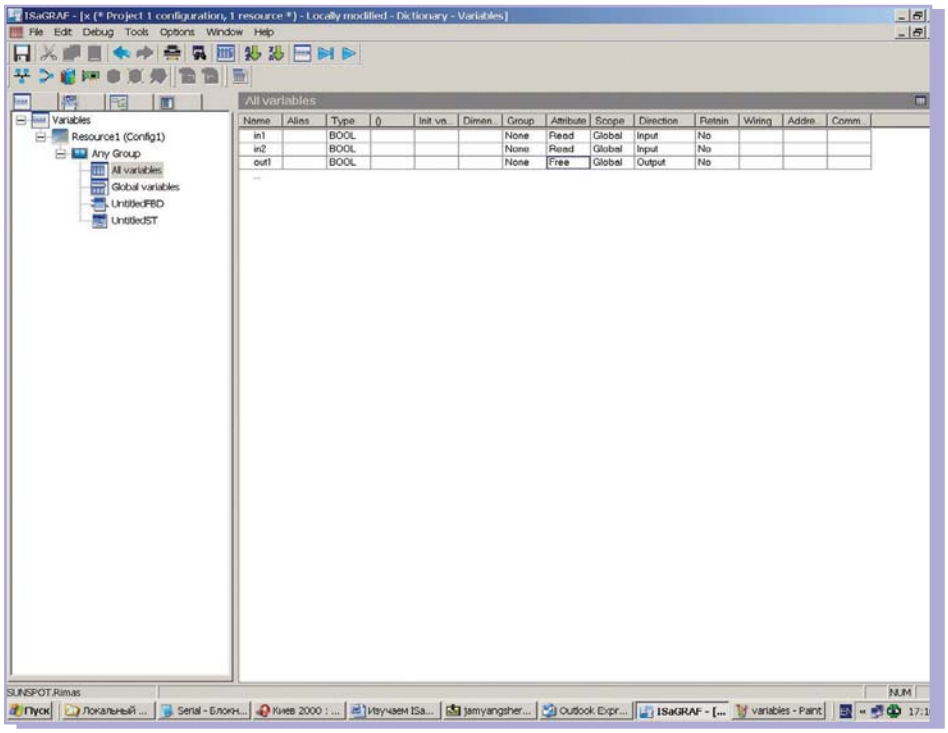
ные были написаны тома, в спорах было сломано сотни копий и разбиты тысячи клавиатур, но решения, удовлетворяющего всех, так и не смогли найти, хотя это и понятно. Старайтесь давать переменным осмысленные имена, которые позволили бы если не стороннему человеку, то хотя бы Вам самим разобраться в программе как минимум через неделю. Согласитесь, список из ста переменных типа a1, a2, a3...a100 выглядит не очень привлекательно, хотя некоторые способны оперировать и такими наборами.

Тип переменной можно задать в поле *Type (Тип)*. По умолчанию *Словарь* предложит Вам BOOL, что как раз и соответствует нашей задаче. Если же требуется нечто отличное от BOOL, то кликните по этому полю мышкой и выберите из выпадающего списка необходимый Вам иной тип.

Двойной щелчок по полю *Attribute (Атрибут)* выведет маленькое окошко, позволяющее задать *Direction (Направление)* переменной. Поле *Атрибут* в списке задает правило, может ли переменная быть прочитана или записана из основной программы. Это свойство полезно, потому что позволяет избежать случайной записи или чтения в переменные, не предназначенные для этого. Несмотря на то, что внимательный разработчик вполне может держать в голове такие нюансы, что и где будет записываться, пренебрегать этой функцией не стоит.

Задайте *Направление* как *Input* для переменных, соответствующим РУБИЛЬНИКА-ам и *Output* для переменной *РЕЗУЛЬТАТ*.

После того, как переменные были созданы, Вы получите приблизительно такую картину:



Давайте чуть глубже рассмотрим возможности *Словаря* по управлению переменными.

В поле *Alias* можно задать второе имя для переменной и затем вызывать ее дальше в программе под разными именами. Так как второе имя будет указывать на ту же область в памяти, которое хранит первое имя переменной, модификация по alias будет менять значение и оригинальной переменной.

Initial Value позволит установить значение по умолчанию для переменной. Полезная и удобная возможность, которая может определять ход выполнения программы. По умолчанию числовые переменные установлены в 0, логические в FALSE, TIME в 0, DATE в 1970-01-01.

Dimention задает размерность переменной. Например, если Вы работаете с переменной типа STRING, необходимо будет задать максимальное количество символов, которые будут храниться в ней.

Параметр *Group* позволяет логически разделять переменные по группам. Функциональной нагрузки он не несет, а сделано это для удобства разработчика. Вы можете создавать соответствующие задаче группы и помещать туда переменные, чтобы впоследствии получить к ним более удобный доступ. Для того, чтобы создать новую группу переменных, перейдите в режим *Link Architecture*, выберите целевой Ресурс, а потом в меню *Insert* выберите пункт *Add Variable Group*. Попробуйте создать две группы, разделяющие входные и выходные переменные.

Свойство *Scope* указывает на область видимости или принадлежность переменной. Аналогично с классическим программированием, переменные видны исключительно в строго заданных областях. Например, Вы можете создать локальные, существующие только для одной-единственной функции, переменные. При попытке обратиться к ним из других областей произойдет ошибка. Что же это дает? Например, Вы можете создавать переменную с одним и тем же именем, но в разных областях видимости. Наверное, Вы догадались, что области видимости в ISaGRAF соответствует POU. Вы можете создать ряд переменных с одинаковыми именами, но видимых лишь в пределах одной функции или функционального блока. Кроме локальных переменных существуют глобальные, область видимости которых составляют все функции и подпрограммы. Хорошим стилем является минимальное количество глобальных переменных, ведь это позволяет избежать появление трудно выявляемых ошибок, когда разные части программы модифицируют одни и те же данные.

Wiring показывает, к какому каналу оборудования привязана переменная. Поле *Address* служит для задания адресов переменной для последующего доступа из SCADA-систем. Поле *Комментарии* говорит само за себя, позволяя задать описание, абсолютно бесполезное для программы, но иногда так необходимое для разработчика.

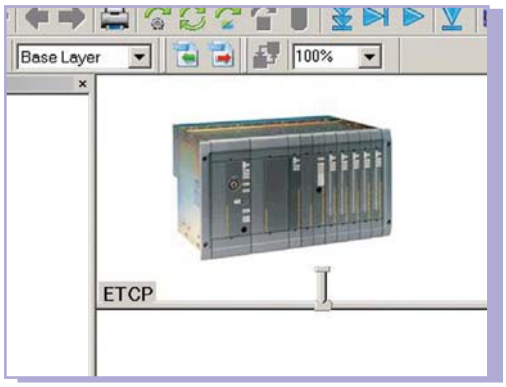
Помимо простых переменных, Словарь позволяет управлять сложными типами данных, такими как Массивы и Структуры, а так же Функции и Переопределения. Все эти возможности требуют детального рассмотрения, поэтому мы обязательно вернемся к ним в дальнейшем.

Отметим, что, к сожалению, в ISaGRAF 5 исчезла функция быстрого создания переменных, позволяющая задать шаблон имени и по нему в автоматическом режиме сгенерировать необходимый набор.

Итак, переменные созданы! Давайте теперь подготовим наш проект к работе. Каждый *Ресурс* должен выполняться на виртуальной машине ISaGRAF, установленной на контроллере. На начальной стадии работы с проектом от нас потребуется задать, какую же целевую функцию планируется использовать, и под какую платформу будет создаваться байт-код. Так как *Ресурсы* выполнятся в рамках Конфигурации, то целевую задачу можно задать как раз через вид *Hardware Architecture*. Перейдите туда и выберите *Конфигурацию Config1*. Вызовите правым щелчком мыши меню и выберите пункт *Properties*. Появится диалог, который позволит задать базовые настройки Конфигурации. Наша цель находится во вкладке *Hardware*. Перейдите туда и выберите из списка тип *Simulation*. Эта цель представляет собой виртуальный контроллер, который отлично подходит для целей предварительной отладки перед загрузкой на реальный контроллер.

Примечание: *при создании реального приложения для функционирования на PAC, Вам пришлось бы выбрать целевую функцию, соответствующую тому оборудованию, которое было приобретено, и которое будет использовано в проекте.*

Вкладка *General* позволяет задать описание *Конфигурации* - название, комментарии и изображение самого PAC. Последняя функция носит исключительно декоративный характер, но, согласитесь, гораздо приятнее работать в таком виде:



Правда, функция отображения картинок на *Конфигурации* по непонятной причине заблокирована. Снять блокировку можно в меню *Options*, выбрав пункт *Hide Bitmaps*. Сделав это, Вы сможете наслаждаться реальными изображениями контроллеров в проекте.

Итак, целевая задача была изменена. Наш контроллер, пусть и виртуальный, может иметь устройства ввода/вывода. Перейдите в *IO/Wiring* ,

т.е. в инструмент управления оборудованием. Интерфейс претерпел значительные изменения со времен версии 3.xx и привыкнуть к нему удается не сразу. Слева Вы увидите пока что пустое пространство, где в реальности будут отображаться платы УСО, справа же расположен список переменных, которые будут подключаться к оборудованию.

Давайте заполним эту пустоту. Нажмите кнопку *Add new device* или воспользуйтесь клавиатурным сокращением *Ctrl+A* и вызовите окно добавления платы. Появится диалог, отображающий специфические для данной целевой функции устройства. Так как мы используем тип *Simulation*, то и устройства будут виртуальными. Платы позволят вводить виртуальные сигналы, которые будут, тем не менее, оказывать воздействия на вполне реальный код программы. Это удобно, если у Вас нет под рукой оборудования, но необходимо опробовать алгоритм.

Диалог подключения оборудования не изобилует опциями, что тоже неплохо. Вы можете указать, в каком по счету слоту установлена плата ввода/вывода (параметр *Device Index*). Сказать по правде, выбор номера слота в ранних версиях ISaGRAF был более удобен и интуитивно понятен. Хотя не исключено, что со временем интерфейс вновь претерпит некоторые изменения, благо производители контроллеров имеют возможность вносить модификации по внешнему виду ISaGRAF Workbench. Кроме номера устройства можно задать количество отображаемых (или задействованных) каналов УСО. Возможность эта может быть задействована с пользой в приложениях, использующих виртуальные платы. Все они по умолчанию имеют 128 канала и отобразить в одном окне три-четыре таких устройства в развернутом состоянии просто невозможно.

Рекомендация: *сокращайте количество отображаемых каналов в проекте до минимально достаточного - это даст возможность более быстрой навигации по спискам.*

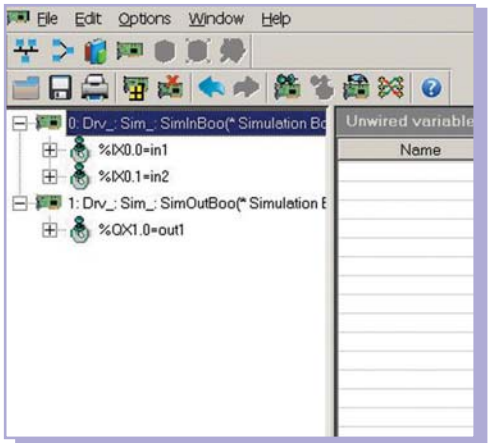
Выпадающий список отображает доступное для данной Конфигурации оборудование. Сейчас, когда мы выбрали в качестве целевой системы Simulation, у нас в распоряжении будут виртуальные платы, позволяющие принимать и выводить сигналы, соответствующие базовым переменным (BOOL, WORD, INT и т.д.). Вы можете заметить, как изменяется список доступного оборудования в зависимости от выбранной целевой функции. Проведите эксперимент, смените *Simulation* на любую доступную, например, *WIN32-TGTA-L* , и снова зайдите в I/O *Wiring*. Подключая новую плату в слот, Вы заметите, что список оборудования немного изменился - добавились новые позиции. Работая с реальными PAC контроллерами, Вы будете устанавливать дополнительные драйвера, надстройки над ISaGRAF Workbench, которые позволят программе создавать код для указанных типов контроллеров и плат УСО. Предположим, у Вас есть контролер LinCON (от ICP DAS) и пара модулей-плат В/В к нему. Установив предварительно драйвера и выбрав для Конфигурации целевую задачу для этого контроллера, затем при добавлении оборудования Вы увидите эти модули-платы среди доступных.

Если Вы последовали совету и проверили на опыте смену оборудования при смене целевой задачи, то верните задачу обратно на *Simulation*. Наша задача потребует двух плат: булевого ввода и булевого вывода (аналог плат дискретного ввода/вывода).

Выберем из списка плату *SimInBoo* и изменим количество задействованных каналов, уменьшив их до двух. Теперь добавьте еще одну плату, на этот раз булевого вывода *(SimOutBoo)* с одним каналом.

Выделим первую плату. Если Вы создали две входные переменные правильно, то они появятся в правой таблице (если же нет, то самое время это сделать). "Разверните" плату (нажмите на крестик перед ее названием) и выделите первый входной канал. Дважды щелкните по переменной, соответствующей состоянию первого рубильника. Переменная теперь привязана к первому каналу устройства. Аналогичный двойной щелчок по второй переменной и она тоже привязана. Теперь выделите вторую плату и привяжите к ней выходную переменную. Достаточно простые действия, не так ли? А ведь то, что мы только что проделали и является основной причиной популярности технологии ISaGRAF! В дальнейшем в своей программе Вы будете работать только с переменными, а ISaGRAF

сам будет инициализировать и опрашивать устройства, сохранять значения измерений и корректно завершать работу с оборудованием. Вся монотонная работа будет выполняться компьютером, для чего он собственно и создавался.



Переменные созданы, устройства добавлены и сконфигурированы, теперь можно перейти к основному предмету нашего разговора - как же пишутся программы под ISaGRAF.

ISaGRAF 5 является первым коммерческим программным продуктом, в котором реализована поддержка стандартов IEC 61131 и IEC 61499, что означает полную поддержку пяти стандартных языков программирования: SFC, FBD, LD, ST, IL. Эти технологические языки позволяют всесторонне "обрисовать" логику приложения на различных уровнях. Причем слово обрисовать употреблено не случайно - первые три языка являются графическими, т.е. все программирование на них сводится к правильному размещению графических элементов блоков с их последующим соединением. ST и IL представляют собой достаточно простые текстовые языки классического программирования типа Basic. Фирма ICS Triplex ISaGRAF дополнительно включила в свой продукт еще один язык, который получил наименование FC *(FlowChart)*. Он не имеет никакого отношения к стандарту, но очень прост в изучении и использовании, и наверняка понравится тем, кому привычны понятия кибернетика, алгоритмизация и блок-схема. Конечно, основным козырем ISaGRAF 5 является поддержка функциональных блоков IEC 61499. Скорее, это не язык, а технология разработки программного обеспечения, которая со временем станет стандартной для промышленных приложений.

В дальнейшем мы рассмотрим все упомянутые выше языки, изучим их более детально в разрезе решения практических задач. Сейчас же важно понять, что ISaGRAF - это просто, а именно простые в использовании технологии позволяют создавать сложные по своей внутренней сути приложения.

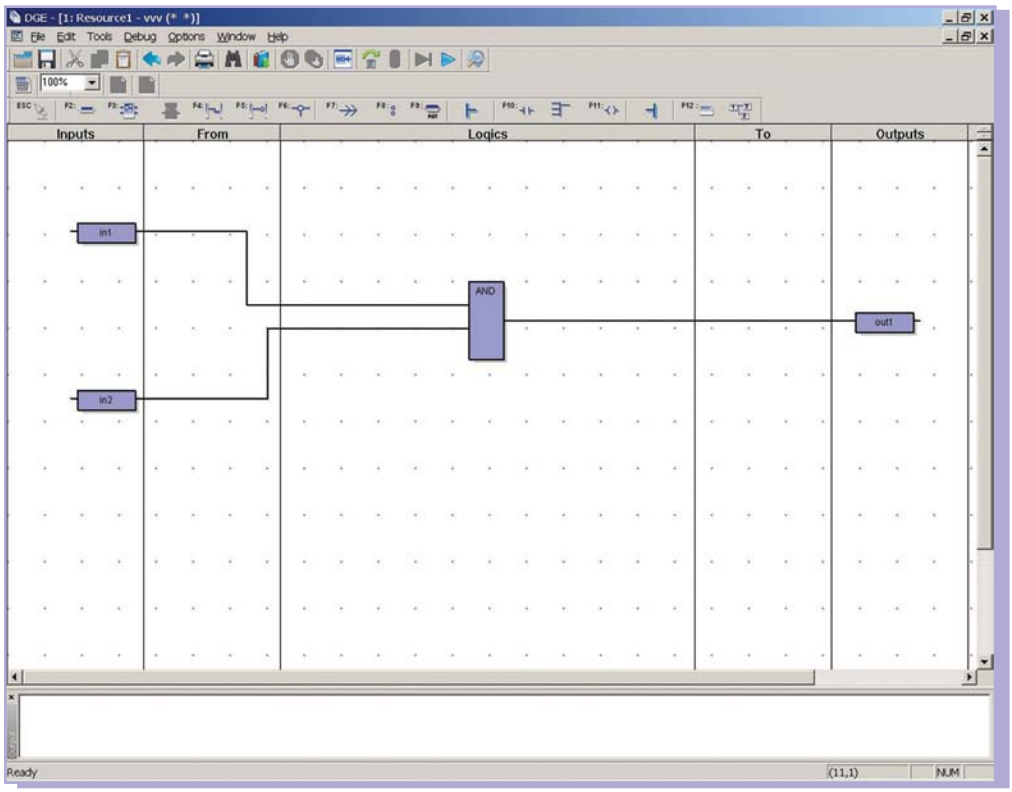
Давайте создадим логику для нашего простого приложения. Все основные программы, которые используются в проекте, ISaGRAF хранит в разделе *Programs Ресурса*. Перейдите в вид *Link Architecture* и выберите наш первый Ресурс. Выберите пункт *Programs* и щелкните по нему правой кнопкой мыши. В открывшемся меню выберите *Add Program -> FBD: Function Block Diagram*. ISaGRAF создаст новую запись в разделе *Programs*. Дайте программе имя, которое потом будет говорить само за себя. Помимо названия, окно *Ресурса* отображает комментарии к каждой программе, что вместе с "нормальными" именами облегчит Ваш труд.

Для нашего первого знакомства с ISaGRAF язык функционально-блочных диаграмм FBD был выбран чисто случайно. Это язык графического программирования, достаточно наглядный и простой, чтобы базовые операции были понятны даже новичку (хотя про любой другой язык ISaGRAF можно сказать то же самое, быть может, за исключением IL).

Двойной щелчок по нашей новой программе - и мы в редакторе! Работа с функциональными блоками сводится к операциям с "черными ящиками". У них есть входы и выходы, а также функция преобразования. Входы могут быть выходами других блоков или же переменными. Для того чтобы вызвать блоки, соответствующие переменным, нажмите на клавишу F2 или кликните на соответствующей кнопке. Курсор мышки поменяет свой вид, что говорит о том, что Вы находитесь в режиме добавления переменных. Кликните в любом месте экрана. Появится диалоговое окно, предлагающее выбрать, какая же именно переменная будет использована в программе. Выберите переменную, отвечающую за работу с первым рубильником, и нажмите OK. В результате на экране появится четырехугольник с именем переменной внутри. Точно так же добавьте переменную для второго рубильника и статуса. Наше приложение чрезвычайно простое и ограничивается только операцией "логического И". Нам нужно добавить блок, который выполняет эту функцию. ISaGRAF поставляется с достаточно обширной библиотекой готовых блоков, способных успешно решать стандартные задачи. Нажмите F3 или кнопку на панели инструментов. Появившееся окно содержит в себе список всех функциональных блоков, известных системе. Этот список разбит на подгруппы для более удобной навигации. Наша функция находится в разделе *Boolean Operations* и называется, как и следовало ожидать, AND. Выберите ее и нажмите OK. На экране появится функциональный блок с надписью, обозначающей выбранную функцию. Далеко не всегда удается разместить блоки на экране оптимальным образом и возникает необходимость их перемещения. Чтобы получить возможность расставить "кирпичики" своей программы на экране в том порядке, который Вам нужен, нажмите клавишу Esc, после чего можно захватывать любой блок и, удерживая кнопку мышки, двигать его.

Графическое программирование отличается от текстового тем, что нужно явно указать направление движения данных. У нас поток данных будет следующим: от двух рубильников до блока логического AND, откуда уже результат будет поступать на выходную переменную. Для того чтобы задать такую последовательность, воспользуемся инструментом *Draw Link* (или клавиша F4). Рисование соединительной линии происходит следующим образом: нажимаем кнопку мышки на выходе одного объекта и, не отпуская ее, перемещаем указатель к входу другого объекта, где ее отпускаем. Если все проделано верно, то появится соединительная линия. Подключите две входных переменных к передним контактам функционального блока, а переменную, в которую помещается результат - к выходу. В итоге должна получиться очень простая программа.

Программа написана - закройте окно редактора. ISaGRAF предложит сохранить результат проделанной работы, что, разумеется, и следует сделать. Перед Вами вновь главное окно программы. Теперь нужно скомпилировать приложение. Перейдем в меню *Project* и выберем пункт *Build Project/Library*. ISaGRAF выполнит синтаксическую проверку проекта и

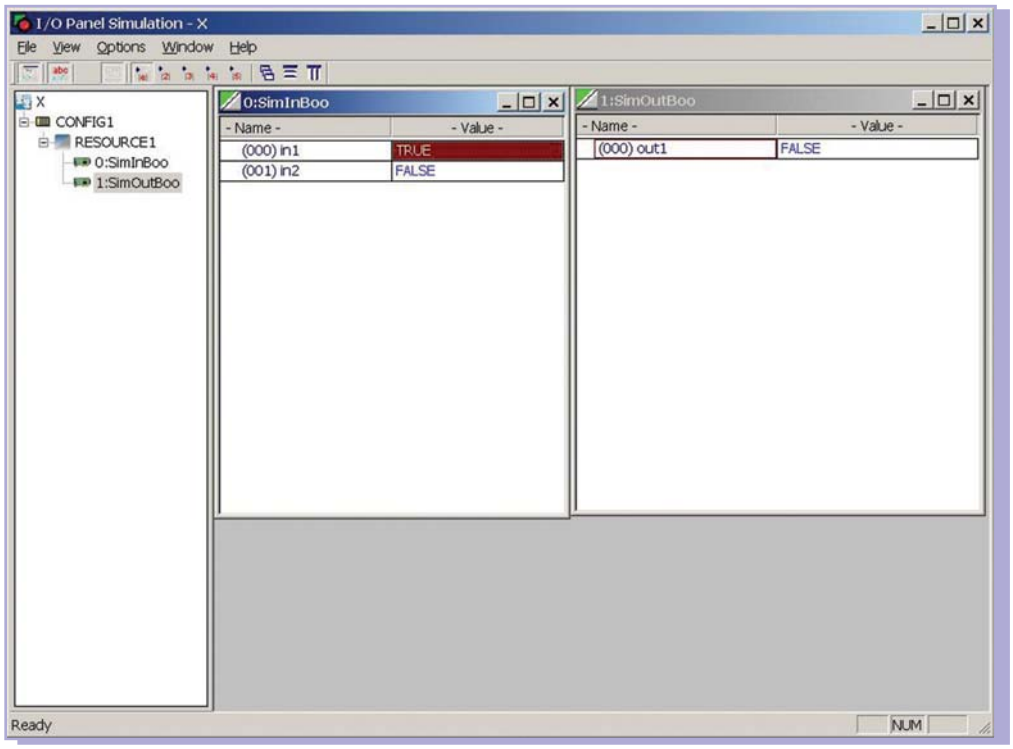


запустит процесс компиляции. Размер нашего проекта оставляет очень мало возможностей для возникновения ошибки, так что компиляция должна завершиться успешно.

Теперь давайте загрузим наше приложение в виртуальный контроллер. Нажмите на кнопку *Simulation* . Откроется три новых окна, но нас в настоящий момент будет интересовать только одно - *I/O Panel Simulation*.

В нем можно управлять своими виртуальными платами, задавая входные сигналы. Двойной щелчок по названию каждой платы будет открывать окно, содержащее список задействованных каналов и связанных переменных. Выберем плату *SimInBoo*, отвечающую за формирование входных сигналов. Как видим, два подключенных входа имеют значение FALSE, так как сигнал на них не поступает. Выберем плату *SimOutBoo* и посмотрим на состояние выходов. Теперь вернемся к входной плате и поменяем состояние одного входного канала. Это можно сделать двойным щелчком мышки на его значении в списке. Значение изменится на TRUE - так мы "замкнули" первый рубильник. Если посмотреть на выходную переменную, то заметим, что состояние ее не изменилось - так и задумано, ведь оно должно поменять свое состояние на TRUE только в случае, если включены оба рубильника одновременно. Включим второй рубильник - и выходная переменная тут же изменит своей состояние на TRUE. Выключим любой один из рубильников - переменная Результат вернется в FALSE. Время принимать поздравления - наша программа работает как положено! Нажимаем и останавливаем режим симуляции.

Вот и все. Мы написали первое, достаточно наивное, зато рабочее приложение под ISaGRAF. Несмотря на свою простоту, она наглядно демонстрирует основную концепцию разработки и преимущества применения ISaGRAF для решения практических задач.



Многое в этой статье осталось без внимания, большое количество интересных аспектов программы не были даже упомянуты, но мы обязательно вернемся к ним в последующих статьях, постепенно вводя читателя в интересный и многогранный мир программирования в ISaGRAF.

Как говорится – продолжение следует.

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