ISaGRAF -это очень просто!

(часть VI, разработка приложений)

Бубович Д.С.,
НТУУ "Киевский политехнический институт", г.Киев

В предыдущих публикациях, посвященных ISaGRAF, достаточно подробно были рассмотрели языки программирования ПЛК в соответствии со стандартом IEC 61131-3, а именно:

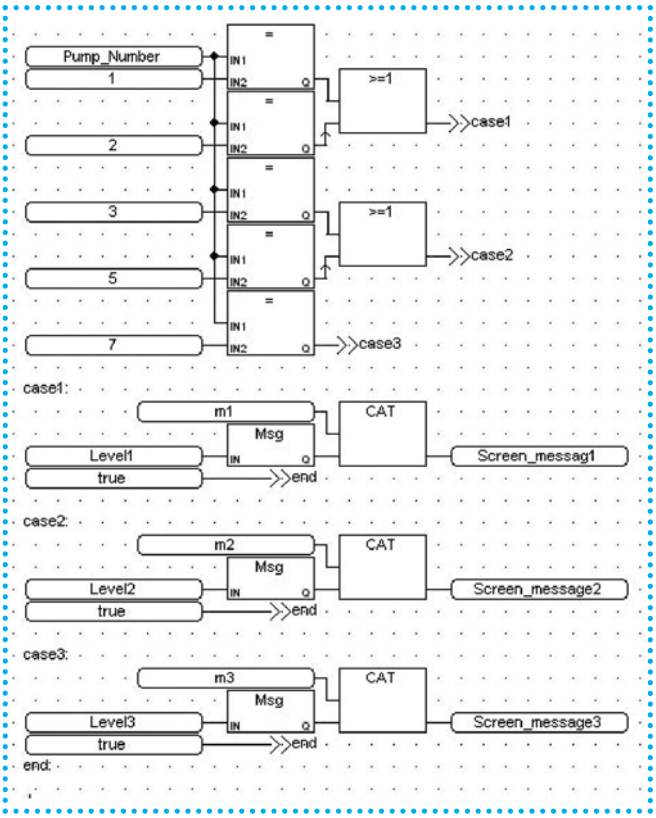
- **LD** (релейно-контактные диаграммы);
- **FBD** (функциональные блочные схемы);
- **ST** (структурный текст);
- **FC** (потоковые диаграммы);
- **SFC** (последовательные функциональные схемы);
- **IL** (язык инструкций).

Вполне естественно, что возникает вопрос, какой же язык лучше выбрать для написания программ? Давайте проиллюстрируем особенности использования различных языков стандарта при разработке приложений на примере проекта с использованием контроллера Visicon PRO и ISaGRAF v3.

Разнообразие подходов, предусмотренное в ISaGRAF, призвано обеспечить максимум удобств инженеру, предоставляя ему возможность решать ту или иную задачу с помощью более понятного для него языка. Например, если работу Вашей программы можно представить эквивалентной релейно-контактной схемой, Вы можете использовать релейные диаграммы LD. Язык LD позволяет описывать работу с булевыми данными, помещая графические символы в схему программы. Графические символы LD организованы внутри схемы программы так же, как электрическая схема. Конечно, помимо работы с булевыми данными, в программе может понадобиться обрабатывать символы и строки, писать функции преобразования, создавать журналы выполнения программы. Такую сложную логику легче описывать с помощью инструкций ST, текстового языка высокого уровня, который по синтаксису напоминает Pascal. По большому счету, языки IEC во многих случаях могут быть взаимозаменяемы, т.е. нет разницы в том, чтобы использовать LD, FBD или ST. С другой стороны, графическое программирование может быть эффективным, когда нужно описать целостную структуру задачи. Используя его, можно получить понятную и хорошо читаемую программу, что снижает вероятность возникновения ошибок во время разработки. Но у графического программирования есть и свои недостатки. Сложные логические выражения выглядят громоздко на FBD. Для примера рассмотрим конструкцию **case** на ST и попробуем реализовать то же самое на FBD.

```
CASE  Pump_Number OF
1,2:  Screen_message1:=m1+ Msg(Level1);
3,5: Screen_message2:=m2  + Msg(Level2);
7: Screen_message3:=m3  + Msg(Level3);
END_CASE;
```

В фрагменте нашей программы будет выполняться один из трех списков инструкций ST, в зависимости от значения целочисленной переменной **Pump_Number**. Каждая строка кода преобразует целочисленную величину **Level** в текстовую с помощью стандартной функции **Msg()**, складывая ее с другой строковой переменной **m**. Эквивалентная программа на FBD будет выглядеть приблизительно так:



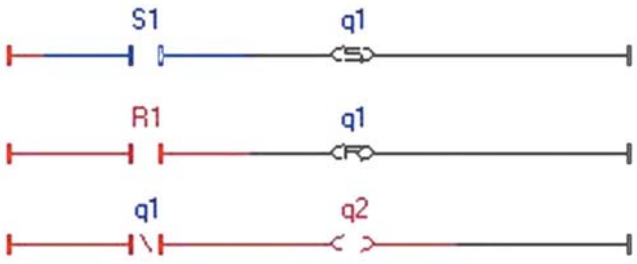
Согласитесь, что в данном случае в редакторе ST пришлось затратить гораздо меньше усилий, чтобы получить нужный результат, да и код получился намного читабельнее. Такие логические выражения лучше описывать в отдельных функциональных блоках или функциях IEC, используя текстовый язык. Потом созданные блоки можно включать в основную FBD программу, чтобы разделить основной алгоритм и промежуточные преобразования.

В некоторых случаях графические языки могут быть более удобочитаемы по сравнению с текстовыми инструкциями ST. Давайте попробуем создать RS-триггер на LD. Конечно, такая необходимость вряд ли возникнет у разработчиков, так как стандартная библиотека содержит

готовый функциональный блок, поэтому пример будет чисто иллюстративным.

Ко входу триггера будут подключены две переменные **S1** (вход set), **R1** (вход reset), а к выходу привязаны **Q1**, **Q2**, инверсная переменная **Q1**. Программа на LD будет выглядеть как показано ниже.

Первый виток содержит символьное обозначение **S**. Это значит, что виток имеет тип **SET**. Переменная **Q1**



принимает значение **TRUE**, когда состояние левой линии связи становится равным **TRUE**. Переменная **Q1** удерживает это состояние до тех пор, пока она не будет инвертирована витком **RESET**, что соответствует работе RS-триггера. Последний контакт в LD схеме является инверсным и используется для формирования значения второго выхода триггера **Q2**. Программа на ST выглядит довольно лаконично, хотя, возможно, менее наглядно, чем на LD.

```
q1 := NOT (s1 or q2);
q2 := NOT (r1 or q1);
```

Так что относительно LD можно сделать такой вывод: язык может быть удобен при описании работы с булевыми данными, релейно-контактных схем, а также в определении переднего или заднего фронта импульсов.

Теперь давайте обратим внимание на особенности работы программ, написанных на языках IEC. Известно, что программы, созданные с помощью LD, FBD, ST или IL выполняются в цикле целевой системой ISaGRAF. Такое поведение программ вполне устраивает, когда постоянно опрашиваются все входные переменные, обрабатываются полученные значения по нужному алгоритму и обновляются выходные каналы УСО. Но как быть в случае, когда программа должна реагировать на некоторые события, к примеру, производится изменение переменных процесса,которые вводятся оператором с помощью HMI, принимаются по GSM модему или с WEB-портала?

Конечно, все можно решить используя тот же принцип поведения программ ISaGRAF. Написать на ST программу, которая будет выполняться циклически и, к примеру, реализовать выбор инструкций, обрабатывающих нужное событие с помощью структуры **case**, или сделать аналогичную задачу на FBD, что будет выглядеть достаточно неудобочитаемо, что и было успешно проиллюстрировано выше.

К счастью, на помощь разработчику приходит мощное средство программирования - язык SFC который позволит разделить работу программы на несколько этапов с помощью основных элементов этого языка - шагов и переходов, а также просто создавать параллельно выполняющиеся программы. Шаг представляет собой программу, которая выполняется за один цикл ISaGRAF. По правилам SFC все шаги должны быть разделены переходами. К переходам можно привязать логические выражения, которые будут переключать работу с одного шага на другой по нужному событию. С помощью ветвей SFC можно также создавать параллельно работающие программы. Например, одна часть программы решает основный алгоритм управления процессом, вторая - занимается выводом данных на дисплей, третья - обменом данными с другими контроллерами, серверной станцией или устройствами сбора данных.

В качестве примера рассмотрим программу, которая могла бы проиллюстрировать маленькую частичку возможностей ISaGRAF в сфере решении сложных задач управления технологическим процессами. Задача заключается в том, чтобы включать аварийную сигнализацию при возникновении критических ситуаций, отображать на дисплее состояний дискретных входов и отчетов о возникших авариях, принимать и передавать данные о событиях главному серверу, вести архив аварийных сообщений. Система должна реагировать на 6 аварийных сигналов, которые поступают на дискретные каналы, а также на 4 аварийных сигнала от сервера, поступающих по каналу интерфейса RS-485. Тревога по первому каналу имеет самый высокий приоритет вывода на дисплей. Также нужно обеспечить возможность просмотра состояния остальных каналов (тревоги от сервера) и архивных данных. При возникновении тревоги должна включаться звуковая исветовая сигнализация, которая управляется двумя каналами дискретного вывода. Сообщения о тревогах нужно передавать на сервер.

Начнем с выбора аппаратной части системы. Нужен программируемый логический контроллер, платы дискретного В/В, дисплей, клавиатура, канал последовательного интерфейса для соединения с сервером. Таким требованиям вполне удовлетворяет контроллер VisiCON PRO, который укомплектован всем необходимым оборудованием, а самое главное - имеет встроенную поддержку ISaGRAF!

Библиотека ISaGRAF содержит следующий набор функций для работы с дисплеем и клавиатурой:

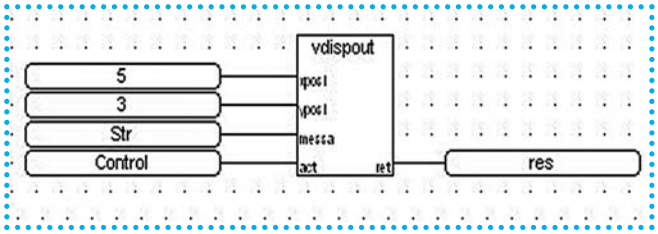
- функциональный блок **VRKEY {ReadKey}** предназначен для опроса клавиатуры. Первый выход блока возвращает переменную типа **boolean**, которая принимает значение;
- **TRUE**, если была нажата какая-либо клавиша. Второй выход возвращает скан-код нажатой клавиши;
- **VDISPOUT {WriteMes}** - Функциональный блок для вывода текстового сообщения на 2-х или 4-х строчный дисплей. С помощью входов **xposition**, **yposition** задается позиция начала строки;
- **VWR_INT {WriteInt}** - Функциональный блок для вывода целочисленного значения на дисплей;
- **VWRFLOAT {WriteRea}** - Функциональный блок отображает значение **float**. Блок позволяет задать количество цифр после запятой;
- **VDISPCHAR {WriteChr}** - вывод символа на дисплей. Задается ASCII код символа;
- **VREADINT {ReadInt}** - Поле ввода целого числа с

клавиатуры. С помощью входов **xposition**, **yposition** задаются координаты поля ввода на дисплее. Набранное численное значение передается привязанной переменной к выходу блока после нажатия клавиши Enter. При этом поле автоматически очищается. При нажатии Esc поле также очищается, ввод значения не происходит;

- **VRFLOAT {ReadRea}** аналогично VREADINT, функциональный блок создает поле ввода числа с плавающей запятой;
- **VCLRDISP {ClrDisp}** - Функциональный блок для очистки дисплея.

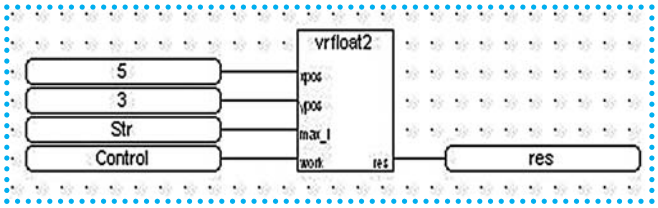
Чтобы перечисленные функциональные блоки работали, в проект необходимо добавить блок **vlcdinit**, который представляет собой драйвер дисплея и клавиатуры.

Проиллюстрируем работу функционального блока для вывода строки на дисплей. Входные параметры функционального блока **xpos** и **ypos** задают позицию строки на дисплее. Сообщение будет расположено в 3-й строке



начиная с 5 ячейки дисплея. Переменная **STR** содержит само текстовое сообщение. Вход **act** разрешает отображение сообщения. Текстовое сообщение будет выводиться на дисплей, если изменились координаты, либо текст сообщения и значение **control** равно **true**. Также сообщение выводится при переходе **control** из **false** в **true**.

Для ввода действительных значений с клавиатуры можно использовать функциональный блок **VRFLOAT**.



Чтобы реализовать ввод числа, нет необходимости включать в программу дополнительный функциональный блок опроса клавиатуры, так как **VRFLOAT** выполняет считывание скан-кода нажатых клавиш автоматически. Входные параметры блока задают координаты поля ввода, в котором отображается набранное число. Вход **act** активизирует работу блока. Данные поступают на выход **res** по нажатию клавиши **enter**, при этом область поля ввода на дисплее очищается. Если нажать **esc** - очистится область ввода на дисплее и буфер **VRFLOAT**. Таким образом, чтобы организовать в программе ввод параметра с клавиатуры с выводом набранного числа на дисплей, достаточно добавить блок **VRFLOAT**, который включает все необходимые функции для работы с клавиатурой и дисплеем.

Для опроса клавиатуры в наборе библиотеки есть функциональный блок **VRKEY**, который возвращает признак нажатия любой клавиши и ее скан-код. К примеру, этот блок можно использовать, если нужно проверить нажатия управляющих кнопок **F1** или **ENTER**. Давайте напишем простенькую программку ввода числа с клавиатуры. Программа будет реализовывать следующий простейший алгоритм: при нажатии **F1** должно появиться поле ввода на дисплее. Затем программа ожидает, пока пользователь введет число и нажмет **ENTER**. После нажатия **ENTER** введенное значение присваивается переменной **value1**, а поле ввода отключается.

Константа **F1_CODE** содержит значение скан кода **F1**, а константа **Enter_Code** - клавиши **ENTER**. Когда условие проверки нажатия **F1** истинно, выход **SR** триггера переводится в **TRUE** и активизируется работа функционального блока ввода числа. Если проверка нажатия **ENTER** возвращает **TRUE**, сохраняем значение введенного числа **temp** в переменной **value1**. Переменная **Off** принимает значение **TRUE** и сбрасывает триггер, тем самым отключая поле ввода.

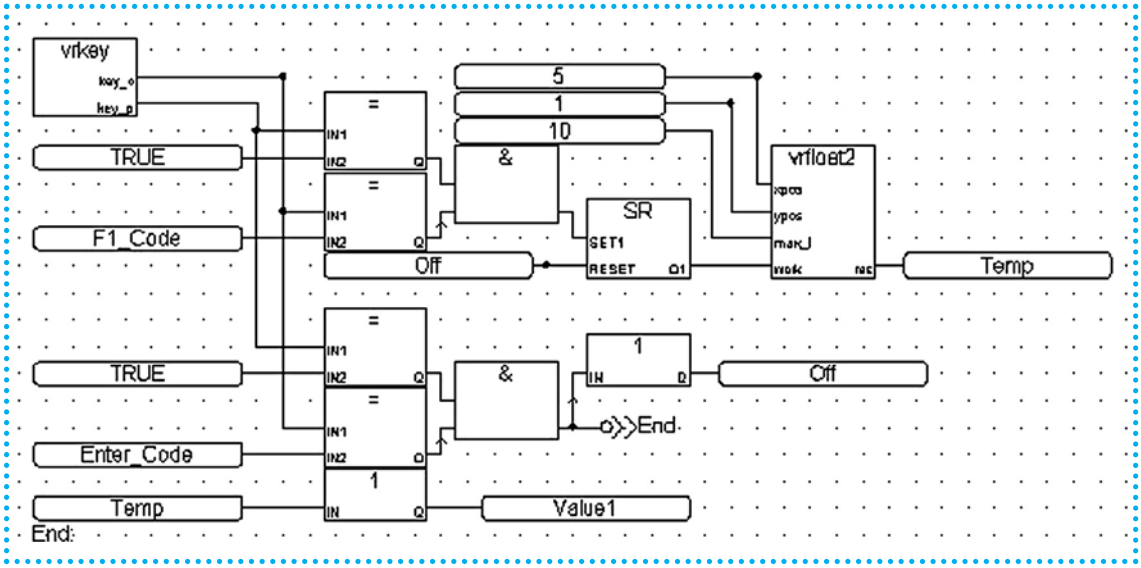
Пример такой же по сути программы, но на ST:

```
ReadKey();
IF  ReadKey.key_press AND ReadKey.key_code=F1_CODE THEN
 WORK:=TRUE;
END_IF;

IF  ReadKey.key_press AND ReadKey.key_code=ENTER_CODE THEN
 WORK:=FALSE;
 value1:=readvalue.res;
END_IF;

readvalue(1,5,10,WORK);
```

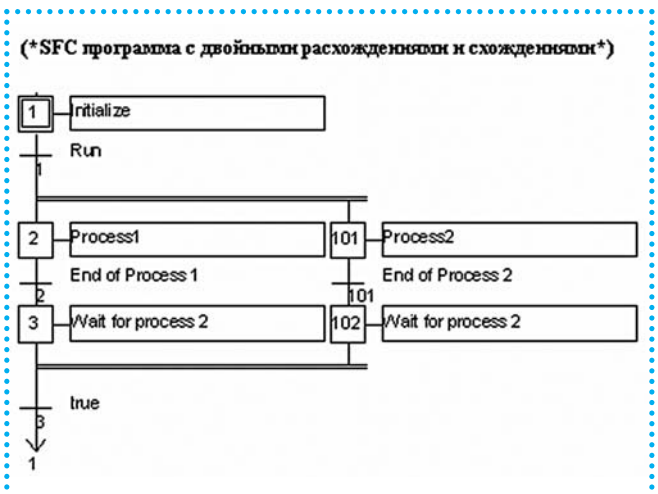
Библиотека ISaGRAF содержит также набор драйверов для различных устройств сбора данных - встроенных в контроллер каналов дискретного В/В с гальванической развязкой и модулей серий i-70xx/i-870xx (ICP_DAS, Тайвань), а также функциональные блоки приема и передачи данных по ModBus.



Нельзя считать систему законченной, если она не предоставляет функции по работе с файлами. И VisiCON PRO обладает достаточно широким набором возможностей по работе с файловой системой:

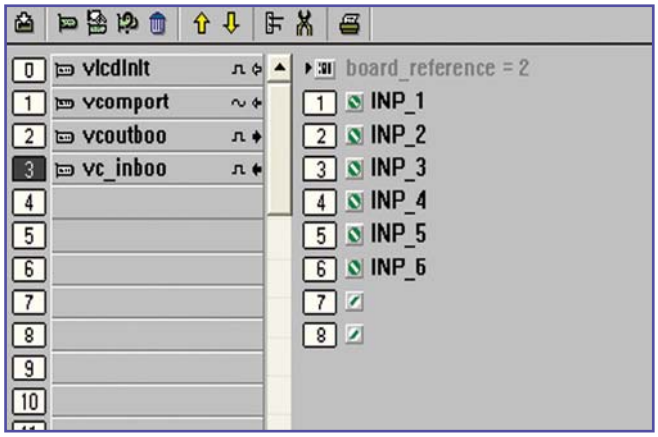
- **Analog V_FOPEN(Message F_name, Message F_param**) - создает или открывает уже существующий файл. Первый параметр **F_name** задает имя файла. Параметр **F_param** определяет способ доступа. Чтобы открыть файл для записи текстовых сообщений, необходимо задать параметр **'at'**. Для чтения необходимо указать параметр **'r'**. Функция возвращает дескриптор, который используется остальными функциями для работы с этим файлом;
- **Analog V_FCLOSE(Analog F_ID)** - После завершения операций записи или чтения, файл должен быть закрыт для корректного сохранения данных;
- **Analog V_F_LIST(Analog F_ID, Message Str)** - записывает сообщения в текстовый файл, созданный или открытый функцией **v_fopen(F_name, F_param)**, при этом добавляет время и дату записи. Если запись выполнена без ошибок, функция возвращает значение 1;
- **Analog GET_L_N(Analog F_ID)** - возвращает количество строк, содержащихся в файле с дескриптором **F_ID**. Предварительно файл должен быть открыт для чтения;
- **Analog SET_L_P(Analog F_ID, Analog Line_Num)** - выбирает номер строки Line_Num для чтения. Функция возвращает 1 в случае успешного завершения. Нумерация строк начинается с 0;
- **Message GET_LINE(Analog F_ID)** - последовательно читает строки текстового файла. Файл должен быть открыт для чтения функцией **V_FOPEN** с параметром доступа **'r'**. Предварительно, номер первой строки для чтения выбирается функцией **SET_L_P**.

Теперь давайте представим задачу в целом и очертим общую структуру программы. Программа должна опрашивать дискретные каналы контроллера, и, если на вход поступил аварийный сигнал, выводить на дисплей сообщение об аварии и отправлять аварийные сообщение на сервер. Также следует отрабатывать нажатие комбинации клавиш и выводить статус выбранных каналов на дисплей. Логика алгоритма обработки тревог и выдача решающего воздействия на дискретные выходы проста. К сожалению, создание пользовательского интерфейса - довольно трудоемкая и неприятная задача. ISaGRAF более предназначен для разработки систем управления, а не программирования пользовательского интерфейса. Для создания HMI с многооконным графическим пользовательским интерфейсом, анимацией техпроцессов, трендами, видео и т.п. приходится писать сложные программы, которые, к тому же, требуют значительных аппаратных ресурсов. Как известно, считается хорошим тоном разделять задачи управления и задачи отображения и ввода данных. Для создания HMI применяется отдельный класс устройств (операторские панели), которые самостоятельно управляют отображением графической информацией и вводом данных, и, таким образом, снимают лишнюю нагрузку с контроллера. Но сделать простенькое меню, вывести несколько строк на монохромный текстовый дисплей не представляет особого труда. К тому же, библиотека ISaGRAF для Visicon PRO содержит набор готовых функций для работы с клавиатурой и дисплеем. ISaGRAF имеет преимущество над программированием в C, Pascal, Assembler под DOS. Реализация в ISaGRAF несколько параллельно выполняющихся процессов тривиальна и потребует добавления элемента двойного расхождения в SFC-программе. Второй способ - создать несколько SFC или FC программ, которые также будут работать параллельно.



Таким образом, можно разделить основной алгоритм управления, программу вывода данных на дисплей, программу для связи с сервером на параллельно выполняющиеся программы.

Приступим, наконец, к реализации намеченного плана. Для опроса входных каналов и обновления выходных данных в окне соединений B/B добавим платы дискретного B/B и привяжем логические переменные. Опрос каналов готов, ISaGRAF избавил нас от низкоуровневого программирования, обращения к адресному пространству дискретных каналов и различных преобразований величин.



Также мы добавили платы **vlcdinit** для управления дисплеем и клавиатурой. Плата **vcomport** конфигурирует последовательный порт, который используется для обмена данными с сервером.

Теперь нужно заставить программу реагировать на нажатие кнопок. Библиотека Visicon PRO содержит функциональный блок **vrkey** для опроса клавиатуры. Создадим FBD-программу **ReadKey** секции **begin** и добавим в нее **vrkey**. К выходам блока привяжем булевую и целочисленную переменные. Значение булевой переменной **TRUE** будет свидетельствовать о том, что нажата клавиша, вторая переменная будет содержать ее скан-код. Эти переменные будут использованы в основной программе. Так как **ReadKey** находится в секции **Begin**, она будет выполняться в начале каждого цикла ISaGRAF, т.е. опрос

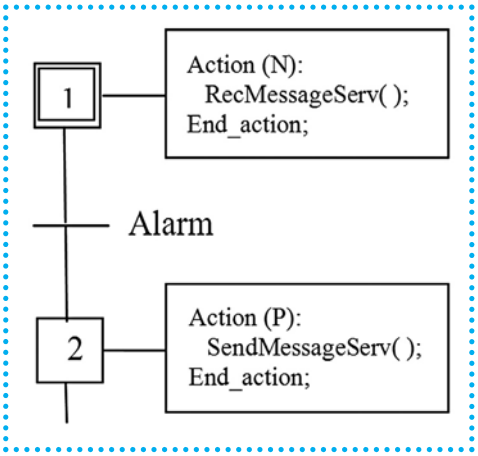
клавиатуры будет происходить каждые 50..200 мс, в зависимости от быстродействия основной программы.

Основной алгоритм программы, предполагает написание довольно сложных логических конструкций управления дисплеем, архивирования событий, отслеживания тревог и прочих требований, описанных выше. Поэтому для его реализации лучше всего применить SFC в связке с ST. Задачу можно разбить на отдельные участки с помощью SFC шагов. К переходам привяжем события нажатия клавиш - для просмотра каналов или архива. Один шаг будет отслеживать тревоги, формировать текстовые сообщения для дисплея, вызывать функции архивирования. Остальные шаги будут активизироваться при нажатии клавиш. Программы этих шагов будут выводить состояния каналов либо архив событий на дисплей. Разбив всю программу на некоторое количество SFC шагов, можно повысить скорость опроса входных каналов или клавиатуры. Так как шаг SFC выполняется за один цикл, то целевая система не будет прокручивать всю задачу целиком, а лишь отдельную ее часть, которая должна выполняться в данный момент.

Теперь рассмотрим подробнее программу для общения с сервером. Для передачи сообщений о тревогах был разработан тривиальный протокол записи в ascii-формате, который описывает номер канала и его состояние. Протокол реализован на C в виде отдельного функционального блока. Блок самостоятельно формирует команды, управляет приемом и передачей, проверяет правильность принятых данных, наличие связи с сервером, отправляет сообщения подтверждения приема данных от сервера. Протокол обмена с сервером можно сделать на ISaGRAF с применением функции работы с COM-портом, но блок на C будет работать быстрее и скрывать в себе низкоуровневые операции приема передачи данных. Если возник аварийный сигнал, на вход блока подается отличное от нуля значение, содержащее код тревоги, который затем отправляется на сервер. В случае, если авария устранена, на сервер также отправляется соответствующее сообщение. Выход блока возвращает коды сообщений, принятых от сервера. Это номера тревог, код подтверждения принятых сервером данных от контроллера, код отсутствия связи с сервером. Используя готовый функциональный блок, создадим SFC-программу **SERVERCNT** для управления приемом и передачей аварийных сообщений серверу. **SERVERCNT** реализована в виде самостоятельной программы, так как не имеет ничего общего с основным алгоритмом, кроме того, что обновляет глобальные переменные, содержащие аварийные сообщения от сервера. Эта программа будет выполняться параллельно с основной. Вторая причина разделения - применение временных задержек при ожидании ответа от сервера, что несколько усложняет синхронизацию с основной программы. Итак, на первом шаге **SERVERCNT** проверяет сообщения от сервера и обновляет соответствующие глобальные переменные. Если пришел сигнал аварии по дискретным входам, то переходим на следующий шаг и отправляем сообщение серверу. Ждем некоторое время, например, 100 мс, чтобы получить подтверждение успешной передачи. Если подтверждение не пришло, то повторяем передачу. Можно задавать способ поведения шага. Если шаг объявлен с модификатором **N**, то он будет выполняться целевой системой циклически, пока не

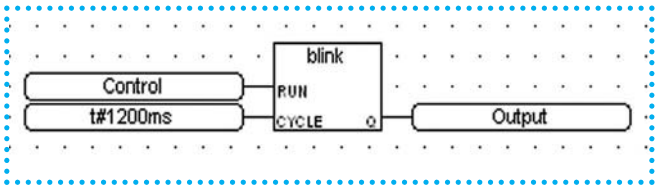
откроется переход. Если задать модификатор **P**, шаг выполнится один раз и маркер будет ожидать перехода на следующий шаг.

Первый шаг программы можно объявить с модификатором **N**, так как проверка сообщений от сервера выполняется гораздо чаще, чем отправка сообщения о тревогах. Можно сказать, что проверка будет выполняться циклически до тех пор, пока не придет аварийное сообщение по дискретному каналу. Переходы SFC также имеют очень полезное свойство - в них можно проверять время активности шага. Поэтому с помощью SFC удобно решать задачи, где алгоритм работы разбит на заданные временные интервалы.



Вернемся к основной программе и рассмотрим возможности создания архива аварийных сообщений. Самый простой способ - писать сообщения в текстовый файл, который потом можно прочитать с помощью функций контроллера или переписать с диска. Кроме стандартных функций **FW_OPEN**, **FA_WRITE**, **F_CLOSE**, которые работают с бинарными файлами, целевая система ISaGRAF для VISICON PRO поддерживает новые функции для работы с текстовыми файлами. Созданный файл можно переписать с помощью терминала для связи с контроллером в компьютер. Вы можете записывать в файл сообщения о возникновении различных событий, тревог, сохранять расчетные данные в процессе работы целевой системы. При этом функции автоматически будут добавлять время записи сообщения в файл. Последующие данные записываются в новой строке. Функции библиотеки также позволяют просмотреть количество записанных строк в файл и прочитать нужную строку.

При возникновении тревоги необходимо формировать сигнал типа меандр на дискретном выходе, к которому подключается звуковая сигнализация, а также включать световую сигнализацию. Стандартная библиотека содержит функциональный блок **blink**, который инвертирует логическую переменную с определенным периодом. Программа, выводящая сигнал управления звуком, может выглядеть таким образом:



Переменная **Output** объявлена как выходная и привязана к дискретному каналу **vcoutboo**.

Итак, имеем модуль В/В для опроса дискретных каналов и FBD-программу для опроса клавиатуры, которая выполняется целевой системой в начале каждого SFC основной программы. Программа **MAIN** выполняет основной алгоритм обработки тревог и управляет отображением информации на дисплее. **Servernt** также написана с использованием SFC и ST, и обеспечивает обмен данными с сервером, работая параллельно с **MAIN**.

Вернемся еще раз к общей архитектуре проекта и особенностям выполнения программ. В терминах ISaGRAF каждая программа - это логическая программируемая единица, которая описывает операции с переменными процесса. Программы могут описывать как последовательные, так и циклические операции. Циклические программы выполняются на каждом цикле целевой системы. К ним относятся программы, написанные на ST, FBD, LD или IL. Исполнение последовательных программ на SFC или FC определяется динамическими правилами языка SFC. Программы могут быть разделены на четыре основных секции: **begin**, **sequential**, **end** и **функции**. К секции **функции** относятся подпрограммы, созданные на языках IEC, кроме SFC или FC, и могут быть вызваны другой программой. Основные программы секции **begin** выполняются в начале каждого цикла, а секции **end** - в его конце. SFC-программы состоят из набора шагов и переходов. Шаг представляет программу, выполняющуюся за один цикл. Выполнению программы шага предшествует выполнение всех основных программ секции **begin**, а после шага выполняются все программы секции **end**.

Программирование в ISaGRAF является достаточно простым и, что немаловажно, интересным занятием. Рутинные действия упрощаются и их значение нивелируется, оставляя разработчику свободное время для творческого поиска. Далеко не все задачи имеют уровень "автоматизация атомной электростанции" или "управление самонаводящейся ракетой воздух-воздух", в большинстве случаев приходится решать что-то более приземленное и тут важно правильно выбрать инструмент. Контроллер VisiCON PRO, "начиненный" целевой функцией ISaGRAF, как раз и предназначен для решения повседневных задач. Да, он не самый "шустрый". Хотя разве можно назвать малопроизводительным процессор в 80МГц? VisiCON PRO успешно используется в задачах мониторинга и управления технологическими процессами, сбора данных и даже в переносных лабораториях. Клавиатура, дисплей, встроенное УСО, возможность подключать внешние модули сбора данных, предустановленная целевая система ISaGRAF - получается замечательный джентльменский набор, который заслуживает серьезного к нему внимания!

P.S. Как стало известно от разработчиков контроллера VisiCON-PRO, уже ведутся работы по замене его процессорного ядра, и в ближайшей перспективе, благодаря применению новейшей процессорной платформы Vortex 86SX 300МГц совместно с DRAM DDR2 128МБ, его производительность существенно возрастет.

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