Схемная эмуляция в основе системы графического программирования

Ковалев С.Э., г.Киев

Существующие в настоящее время системы графического программирования - это инструментальные средства, которые позволяют разрабатывать прикладное программное обеспечение для контроллеров систем автоматики, не прибегая к традиционному программированию. Именно в области АСУ ТП они достигли наибольшего своего развития, став неотъемлемой составной частью так называемых SCADA-систем (Supervisory Control And Data Acquisition) и SOFTLOGIC-систем.

При традиционном программировании алгоритм управления, разработанный программистом, реализуется путем написания им последовательности программных операторов. При этом подразумевается, что сам алгоритм программы предварительно был набросан программистом на листе бумаги или хотя бы воображен в голове. Графическое программирование состоит в том, что алгоритм рисуется не на бумаге, а на экране дисплея в среде графического редактора. Таким образом, разработка программ управления в таких системах заключается в "сборке" в среде графического редактора алгоритмов управления из готовых "кубиков" - функциональных блоков, а специальный ("графический") компилятор по созданному рисунку автоматически сгенерирует исходный код программы.

Главной целью при создании таких инструментальных систем преследовалась идея создания управляющих программ для систем автоматики без участия самих программистов.
Чем вызвана такая немилость?

Дело в том, что древнейшей мечтой проектировщиков АСУ ТП была мысль о единой спецификации проекта в цепочке "заказчик-технолог-программист". Ее решение позволило бы представителям различных профессиональных групп - Заказчику, Технологу и Программисту - однозначно понимать друг друга.

Как же обстоят дела сейчас?

Результатом совместной работы заказчика с исполнителем (пусть это будет технолог) является некий алгоритм, представленный в виде графического рисунка или словесного описания, что выступает основой документа, называемого техническим заданием, скрепляемого печатями организаций. Программисту необходимо проделать неформальную операцию перевода этого документа в язык машинных кодов - программного продукта, предоставляемого заказчику.

Вот тут-то и начинаются все беды. В большинстве случаев переход от алгоритмизации к программированию представляет определенную проблему. Программисту приходится додумывать, как представить ее вычислителю на понятном ему языке - языке программных кодов. По существу получается, что программа отображает то, как программист смог понять исходный алгоритм. В конечном счете появляются два алгоритма: один на бумаге, для отчетности и документирования проектных решений, а второй - в листинге программы.

Программирование, несмотря на свою массовость, остается искусством. На сегодняшний день все попытки полностью формализовать или автоматизировать процесс написания программ оказались несостоятельными. Другими словами, каждый программист программирует так, как умеет. Поэтому степень затрат времени и средств, выделяемых на проект, находятся в прямой зависимости от его способностей и амбиций.

К тому же, Он оказывается единственным держателем важнейшей логической информации и человеком, способным разобраться в своем творении и от которого, в конечном счете, зависит судьба всего проекта. Не-чего уже и говорить про случай, когда человек может просто уволиться в разгар проекта.

Кроме пресловутого "человеческого фактора" существует другой, не менее грозный, - это лавинообразный рост программного кода в программных продуктах, связанный с ростом функциональной сложности алгоритмов управления. Поэтому все больше усилий требуется от разработчика на написание и отладку кода, оставляя меньше сил и времени на проработку самой задачи и оптимизацию. Сам же процесс исправления ошибок все более напоминает латание дыр.

Согласно данным отчета Национального института по стандартам и технологии, объем экономических потерь из-за ошибочного ПО только в США достигает нескольких миллиардов долларов в год, что составляет около 1% национального валового внутреннего продукта.

У многих IT-специалистов уже сложился стереотип, что с появлением SCADA- и SoftLogic-систем и проблем вроде как уже никаких не существует, и все уже решено. На самом деле все далеко не так. Вроде такие системы позволяют запрограммировать контроллер путем всего лишь "рисования" на экране монитора некоторого алгоритма управления. На самом деле - это слишком утрированное высказывание.

Так сразу - не получится! Хотя каждая такая система может и не потребовать глубоких знаний аппаратного обеспечения контроллеров, все равно - это достаточно сложные системы, требующие немалых усилий и времени на освоение. Это тома документации и, опять же, языки программирования, пусть и другого толка (язык последовательных функциональных схем, релейной логики, функциональных блоковых диаграмм), да и от универсальных языков

(BASIC, C, Pascal) еще никто не отказывался. Более того, любой уважающий себя программист предпочтет разрабатывать серьезную АСУ ТП, используя именно универсальные языки, поскольку графическое проектирование пока еще остается больше игрушкой и красивой рекламной упаковкой для существующих систем.

Успешное внедрение таких систем практически невозможно без предварительной закупки у фирм-поставщиков опытных полигонов, интеграция которых намечается на объекте. В противном случае есть риск получить неработоспособную систему или, как минимум, неоптимизированную.

К функциональной сложности следует добавить еще и проблему ресурсов. Ведь для большинства известных систем требуются ресурсы PC, что в условиях промышленности автоматически предполагает использование дорогостоящих промышленных компьютеров, а не более дешевых контроллеров.

Не следует забывать также о проблеме мультипроцессорной обработки, когда всю "графику" приходится откладывать в сторону и вручную приступать к проектированию интерфейсов межпроцессорного обмена. А полазив по Сети, мы обнаружим многочисленные материалы, свидетельствующие в пользу того, что проблема эта в целом еще далека от окончательного разрешения даже на теоретическом уровне.

Для того чтобы полнее увидеть преимущества системы графического программирования, построенной на принципах схемной эмуляции, в сравнении с огромным числом уже известных на сегодня систем, необходимо разобраться - в чем состоит идеология работы последних. Проведенный анализ показывает, что любая Система Графического Программирования строится на трех китах: графическом редакторе (редакторе рисования схем), графическом компиляторе и системе исполнения.

Что касается редакторов рисования, то тут и проблем как бы нет: даже спроектировать такую "штуку" под силу практически любому программирующему студенту (который учится, а не стоит в очереди за дипломом). А вот графические компиляторы несут на себе гораздо большую функциональную нагрузку. В "обязанности" этой компоненты входит просмотр графического рисунка проекта и генерация исходного кода программы, который раньше программист набирал бы "ручками". Затем исходный код компилируется в некоторый промежуточный. Вся эта работа выполняется проектировщиком (программистом) на рабочей станции (ПК).

После этого файл промежуточного кода уже можно загружать во встраиваемый контроллер. Туда же загружается и специальный интерпретатор промежуточного кода - система исполнения.

В свете сказанного становится понятным, что принципиальной разницы между SCADA системами и обычными системами визуального программирования, такими как DELPHI, Visual C, Visual Basic (и др.), нет. Потому как в обоих случаях исходные тексты программ генерируются по визуальным компонентам, что были использованы в проекте.

Классическим примером сказанному может служить всенародно известная среда программирования ISaGRAF (ICS Triplex ISaGRAF Inc.). Здесь при компиляции файла графического проекта генерируется промежуточный, так называемый TIC-код, который затем загружается во встраиваемый контроллер. Туда же подгружается TIC-интерпретатор, который и выполняет функцию системы исполнения.

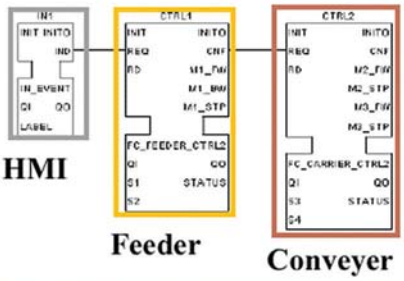
В не менее народной системе LabVIEW (National Instruments) генерируемые исходные тексты кодируются в так называемый промежуточный G-код, который затем интерпретируется (исполняется) непосредственно на PC из среды LabVIEW.

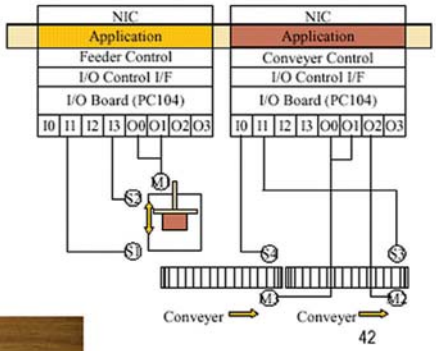
Можно выявить четкую аналогию, что TIC-интерпретатор для TIC-кода, это то же самое, что среда LabVIEW для G-кода.

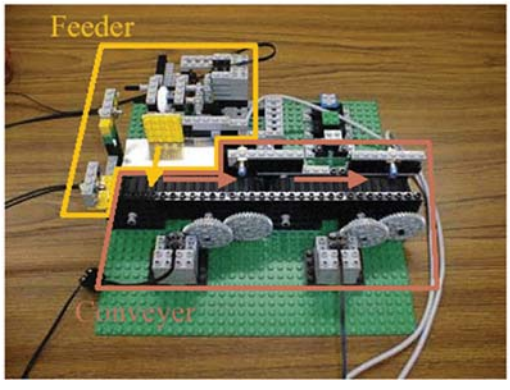
Параллельно рассмотренной выше идеологии существует несколько отличная, заключающаяся в том, что еще на этапе работы графического компилятора генерируется не промежуточный код, а сразу исполняемый, который и загружается в промышленный компьютер или встраиваемый PC-контроллер. Правда, говорить в этом случае о системе исполнения не приходится, поскольку в этом случае она подменяется так называемой средой исполнения ("железо"+операционная система), для чего, правда, уже требуются ресурсы компьютера.

Такая идеология, как правило, также предлагается известными системами графического программирования. В LabVIEW для этого имеется приложение Application Builder for LabVIEW, которое позволяет сгенерировать оптимизированный код, сравнимый с программным кодом C-компилятора. Такой исполняемый код принято называть RUN-TIME модулем.

Система ISaGRAF также позволяет проводить генерацию обычного C-исходного кода, откомпилировав который обычным C-компилятором мы получаем "классический" RUN-TIME модуль. Идея RUN-TIME модулей нашла применение и в других системах класса HMI и SCADA, например в LookOut (National Instruments). Аналогично, какую бы другую систему далее не рассматривали - увидим все то же самое.







ISaGRAF 5.0 - позволяет реализовывать проекты, отвечающие требованиям IEC-61499, нового стандарта для SoftLogic-систем

Не является исключением и программирование на языке лестничных диаграмм (LD) для всех известных линеек PLC-контроллеров - MODICON, Siemens, Allen-Bradley и др. А все по одной простой причине: ни интерпретацию, ни компиляцию в программировании пока еще никто не отменял.

То есть идеология всех современных систем основана на упорном стремлении от графики перейти к обычному программированию, чем автоматически вызывается к жизни весь тот геморрой, которым страдает программирование как отрасль. Но эта тема уже из области проблем, присущих как самим компиляторам, так и программированию в целом.

Присутствие языков программирования, а также компиляторов/интерпретаторов, в известных системах графического программирования делает такие системы достаточно сложными и дорогими продуктами. Ясно, что все они рассчитаны на автоматизацию заводов и фабрик, но никак не на уровень малых внедренческих фирм и, тем более, не на уровень частного предпринимателя. О стоимости "железа" в таких системах вообще лучше помолчать. Можно представить, сколько миллиардов долларов "крутится" по миру в сфере внедрения АСУ ТП. Это, конечно, не может не нравиться самим внедренческим фирмам. А куда деваться заказчикам? Да никуда! Ну, что-то немного дороже, что-то - дешевле, но в целом альтернативы-то все равно нет!

И, наконец, хочется отметить, что использование специализированных и универсальных языков программирования по-прежнему требует участия в проектах программиста. Так что пока ну никак нельзя сказать, что современные системы графического программирования в отечественных условиях уже позволяют представителям различных профессиональных групп - Заказчику, Технологу, Программисту - однозначно понимать друг друга.

Справедливости ради, следует отметить, что ситуация в стране, правда уж больно медленно, но меняется. Что такое LabVIEW уже хорошо знают студенты ВУЗов - подрастающее поколение автоматизаторов. А в новом году к ним добавятся еще и учащиеся колледжей и старшеклассники, а также их родители, бабушки и дедушки, учителя и просто увлекающихся современными технологиями люди всех возрастов. В Украину пришел Mindstorms NXT - необычный LEGO, нечто больше чем игрушка *(см., например www.mindstorms.com.ua)*.

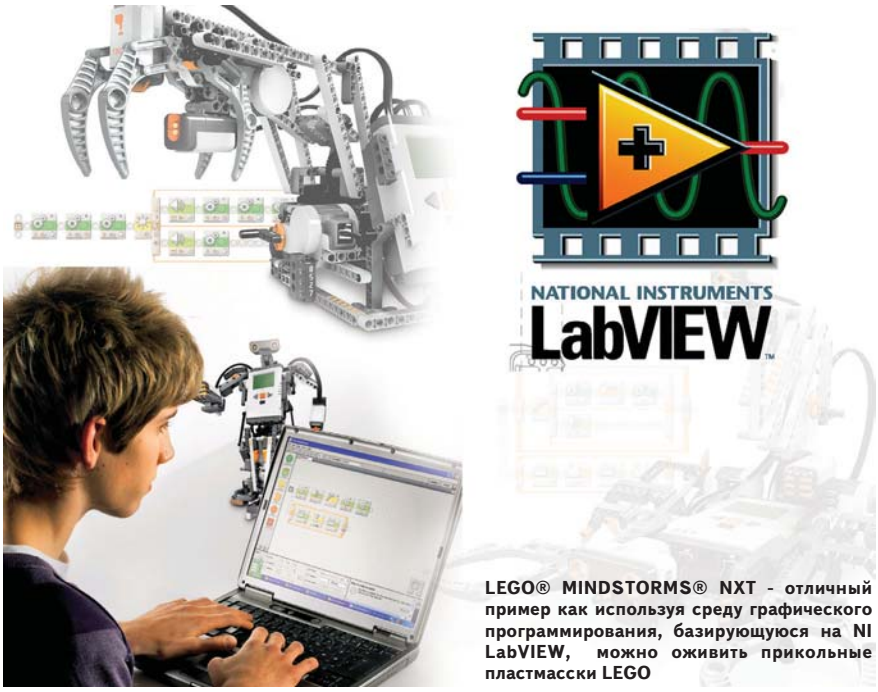
Но вернемся к насущным проблемам. Системы графического программирования, в основу которых будет заложен принцип схемной эмуляции, позволяют отказаться от исходных кодов, а значит и от компиляции/интерпретации также. Принцип схемной эмуляции лежит только в основе системы исполнения нового типа. Таким компонентам, как "графическому компилятору" (кит №2) и исполняемому интерпретатору (кит №3) в новой идеологии нет места. Что в свою очередь ведет к поразительному упрощению всей системы в функциональном плане, а значит и в ценовом. Такой системе не нужны мощности PC. Она проста в освоении даже для "чайников", коими есть, к примеру, те же технологи.

Что же такое схемная эмуляция?

В основе новой идеологии лежит "программа схемной эмуляции". Что следует понимать под этим словосочетанием? Ведь любой, даже мало-мальски образованный электронщик возразит, что нет в природе программ такого класса, и он будет абсолютно прав. Известны так называемые программы-симуляторы, такие как Micro-CAP, PSpice и др., то, что в советскую эпоху у нас принято было называть программами моделирования электронных устройств. Для того, чтобы пояснить, что же такое есть программа схемной эмуляции, рассмотрим как из обычного программного симулятора получить полноценную программу схемной эмуляции.

Уточним сразу, программы схемной симуляции традиционно делятся на две группы: моделирования аналоговых устройств и моделирования дискретных, проще говоря, цифровых устройств. Математический аппарат моделирования а налоговых устройств вряд ли применим к задачам, которые можно отнести к предмету данной статьи, поэтому в дальнейшем рассматриваться не будет. Пока же будем апеллировать исключительно к системам моделирования дискретных устройств. Алгоритмы, которые используются в основе программ моделирования цифровых устройств - тема самостоятельная и достаточно интересная. Про это написано достаточно большое количество учебных пособий и если кто-то все это в свое время пропустил, есть смысл все-таки обратиться к соответствующей литературе для устранения пробела в знаниях, хотя бы по причине более полного понимания сути освещаемой темы.

Исторически симуляторы появились в ответ на законное желание разработчиков цифровых устройств проверять работу спроектированной схемы до того, как она будет реализована с помощью паяльника. Ведь не очень приятно вновь и вновь за него браться, чтобы исправить ошибки, допущенные на этапе проектирования. А ведь речь шла об опытных промышленных образцах, содержащих



LEGO® MINDSTORMS® NXT - отличный пример как используя среду графического программирования, базирующуюся на NI LabVIEW, можно оживить прикольные пластмасски LEGO

до 70 и более корпусов интегральных микросхем на плате.

Более точно такие программы назывались "программами логического моделирования дискретных устройств". Были еще программы физического моделирования, проверяющие нагрузочную способность каждого вывода микросхемы, токов утечки, паразитных емкостей и т.д. Но программы такого класса не относятся к рассматриваемой теме по одной простой причине: алгоритмы, которые необходимо эмулировать, какими бы сложными они не были, всегда останутся программными моделями, а у программной модели не бывает ни токов утечки, ни паразитных емкостей.

Моделирование можно считать выполненным правильно в случае, если получены правильные временные соотношения между уровнями сигналов в абсолютно всех цепях устройства. Результатом моделирования в любом случае станет некоторый массив состояний уровней сигналов в цепях схемы, который можно затем вывести на принтер или дисплей в виде графиков.

Такие графики отображают всю динамику изменений сигналов в цепях модели, а не реальной схемы, поэтому могут рассматриваться как "мертвый" отпечаток событий в реальном устройстве, да к тому же сильно и сильно растянутым во времени. За единицу условной временной шкалы принимается задержка на том компоненте схемы, для которого согласно справочным данным она является наименьшей. Если же выразиться точнее - как правило, берется значение наименьшего кратного для значений задержек сигналов для всех типов компонент.

Уровни входных сигналов программа моделирования шаг за шагом "подает" на вход программной модели, выбирая их из так называемого массива входных воздействий. Разумеется, что само слово "массив" - это атрибут программистских "штучек" и не имеет ни малейшего отношения к реальным сигналам.

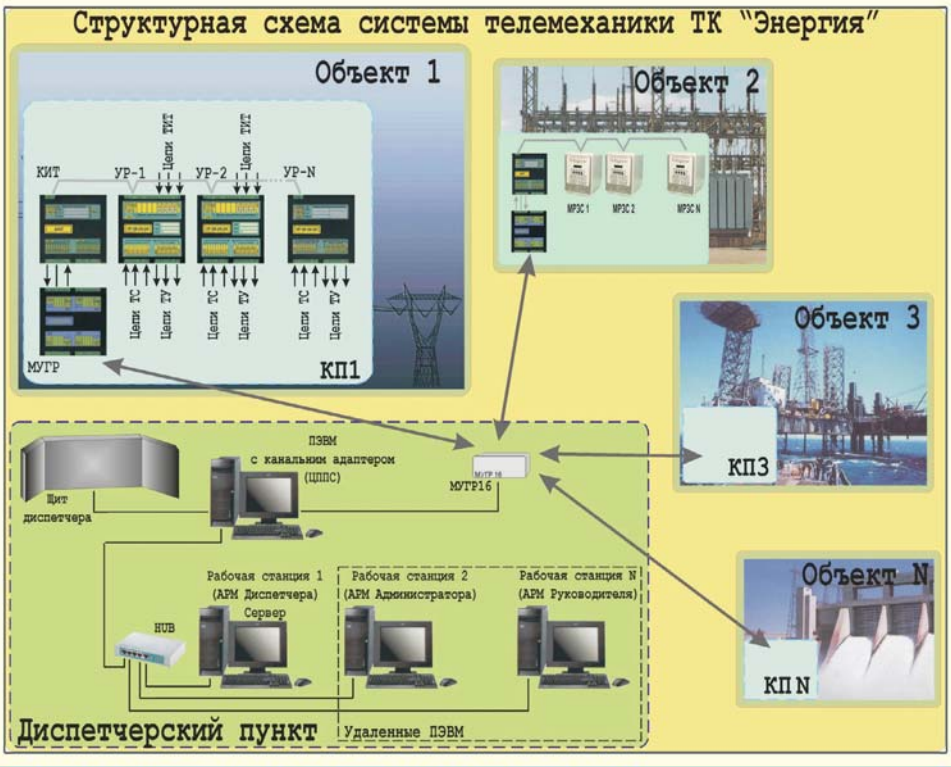
Это и есть моделирование! А что же тогда есть эмуляция?

Эмуляция - это имитация поведения реального устройства в режиме реального времени. Здесь уместна аналогия "черного ящика" - условно говоря, действительно ящика, на котором находится электрический разъем для подключения реальных входных и выходных сигналов от объекта управления. При этом внешний наблюдатель не сможет сказать (не путать с угадать), что в этом ящике находится: реальное электронное устройство или ПК, на котором в режиме реального времени имитируется его работа.

Стоит только уточнить, а что следует понимать под "режимом реального времени". В самом общем виде под этим следует понимать случай, когда скорость имитации (эмуляции) соизмерима со скоростью изменения процессов в реальном объекте управления. Другими словами, будет ли отклик (снимаемые сигналы) с программной модели алгоритма управления поспевать за меняющимися процессами в реальном объекте управления.

В нашем случае можно дать и более конкретное определение: если скорость эмуляции будет соизмерима (и даже выше) скорости выполнения программы управления, которую программист обычным способом написал бы в качестве управляющей для данного объекта - можно говорить о режиме реального времени. Под "обычным" способом подразумевается программная реализация цикла "считывание сигналов с датчиков - обработка входных сигналов по определенному алгоритму - формирова-

ТОВ "НВФ "Енергія" пропонує системи телемеханіки (АСДУ) на базі розпосереджених КП "РКП-5".



В якості периферійних пристроїв збору інформації та керування використовуються інтелектуальні пристрої:
-УР-08/08/08,
-tcon-ADA,
-МРЗС-05,
-лічильники енергоносіїв та ін.

Наші координати:
03057, Київ-57, а/с 95
08200, Ірпінь, пров. Нахімова, 2А
тел./факс: /044/ 592-76-54
тел./факс: /04497/ 62-347
E-mail: energy-avt@ukr.net

ние управляющих сигналов на исполнительные органы" на каком-либо языке программирования. Именно таким образом пишутся программы для систем автоматики.

Теперь попытаемся показать, какие условия надо выполнить, чтобы программу моделирования превратить в программу схемной эмуляции. Это поможет яснее представить, что же такое есть эмуляция. Прежде всего о входных сигналах. Уже подчеркивалось, что в программах схемной симуляции воздействия на программную модель цифрового устройства шаг за шагом выбираются из так называемого массива входных воздействий. Это означает, что "сигналы", которые подаются на программную модель какого-то электронного устройства, имеют не физическую, а программную природу.

Разработчикам этих самых симуляторов не составило бы большого труда дополнить свои программы модулями стыковки с "внешним миром". Для этого всего-навсего надо использовать так называемые порты компьютера. Задача эта абсолютно тривиальная, но это еще не превратит их программы в эмуляторы. Почему? Потому что их программа должна работать еще и в режиме реального времени.

Вот тут-то и начинаются проблемки. Ведь все известные на сегодняшний день схемные симуляторы работают исключительно в условной временной шкале, да иного от них пока и не требовалось. Это означает, что моделирование процессов, протекающих в реальном электронном устройстве за несколько микро- или миллисекунд, на PC проходит за минуты или даже часы.

Но представим, что и эту проблему все-таки побороли. Это хорошо! Но и этого мало! Почему? Потому, что современным симуляторам требуются еще и ресурсы PC. Да желательно еще и как можно более современного. Под ресурсами следует понимать большой объем дискового пространства и оперативной памяти. Ну и, конечно же, мощный процессор. Таким образом, говорить о возможности размещения таких программ во встраиваемых микропроцессорных платформах явно преждевременно.

И, наконец, всякая известная на сегодня программа цифрового моделирования работает с двумя дискретными значениями уровней логических сигналов в цепях схемы. В таких же приложениях как АСУ ТП мы имеем дело не только с дискретными сигналами, но и с аналоговыми. Поэтому система должна быть способной выполнять смешанное, аналого-цифровое, моделирование.

Если Вам удалось создать программу схемной симуляции, обладающую всей совокупностью только что рассмотренных необходимых свойств, то такую программу со всей справедливостью уже можно назвать эмулятором!

Идея! Применить принципы схемной эмуляции в основе систем графического программирования. Осталось ответить на последний вопрос: а какая вообще может быть связь между программой схемной эмуляции и системой графического программирования?

А все дело только в том, что понимать под словом "схема". Ведь это действительно может быть принципиальная схема некоторого электронного устройства, аппаратно реализующего некоторый алгоритм управления. Но это может быть и непосредственно функциональная схема алгоритма управления некоторой системы, к примеру, АСУ ТП, системы управления летательным аппаратом, алгоритма функционирования биологической или физической модели. Это может быть рисунок релейных или блоковых диаграмм. Это может быть нейронная сеть, становящаяся все более популярной сегодня для систем АСУ ТП верхнего уровня, или граф переходов цифрового автомата для случая автоматного программирования. Один знакомый предпочитает алгоритмы управления реализовывать исключительно на цифровых компонентах. Ему, как бывшему цифровику, ничего нет милее и понятнее родимых вентилей, счетчиков, триггеров и мультиплексоров. И к тому же фантазировать можно без ограничений. Платить-то за компоненты не нужно - они все виртуальные!

Эмуляция принципиальных схем - это, конечно, частный случай. Если кто-то владеет средствами языка функциональных блоковых диаграмм (FBD) или релейных диаграмм (LD) - его право пользоваться этими языками. Если кто-то предпочитает пользоваться Виртуальными Приборами: осциллографом, вольтметром, частотным фильтром и т.п. - пожалуйста. Поэтому можно сказать, что вид графического программирования, т.е. метод проектирования рисунка будет определяться только тем, какими компонентами из общей библиотеки компонентов системы Вы будете пользоваться. Главное, чтобы эти компоненты уже входили в так называемую библиотеку. Фирма, занимающаяся сопровождением такой системы, естественно, должна будет постоянно отслеживать вопросы расширения библиотек компонентов для различных областей, и предусмотреть возможность их загрузки с фирменного сайта. Ну а если кто-то решит разрабатывать свои, личные - в программной оболочке должен быть заложен соответствующий инструментарий.

К мысли об использовании алгоритма схемотехнического моделирования в основе системы исполнения подталкивает и внешнее сходство графических проектов. Те же "квадратики", "ромбики"... и множество линий (цепей), соединяющих между собой компоненты рисунка. Вот и родилась идея, зачем же рисовать алгоритм управления (к примеру), чтобы затем так над ним "надругаться"? - разрабатывать специальные графические компиляторы, чтобы вот так сразу взять и уничтожить всю графику, преобразовав ее в файлы каких-то (TIC,G и пр.) кодов. А потом еще разрабатывать компиляторы или интерпретаторы, которые начнут расшифровывать все закодированные инструкции, находящиеся в этих файлах, и шаг за шагом выполнять на целевой системе (контроллере).

Суть предложения состоит в том, что следует на рисунке и остановиться. Программы схемотехнического моделирования работают непосредственно по рисунку проекта, а как оказывается на поверку, их "математический" аппарат прекраснейшим образом подходит и к новому назначению. Ведь давней мечтой всех разработчиков программного обеспечения было то, чтобы программы работали также надежно, как электронные устройства. Моделирование как раз и позволяет рассматривать программу, представленную в графическом виде, как схему, и в то же время позволяет "не браться за паяльник", а просто эмулировать ее на микропроцессорной платформе или PC.

(продолжение следует)

КОНТАКТЫ:
тел: (044) 246-63-12
e-mail: simula@ukr.net