Схемная эмуляция в основе системы графического программирования

(часть2)

Ковалев С.Э., г.Киев

Система графического программирования, построенная на принципах схемной эмуляции, состоит только из двух основных компонент: графического редактора и системы исполнения.

Идея использования графического редактора ровным счетом ничем не отличается от уже известных систем. Процесс проектирования какой-либо системы управления выглядит обычно так. Из библиотеки компонентов выбирается графический образ какой-либо компоненты и переносится на рабочую область рисунка графического редактора. Затем компоненты соединяются между собой необходимыми цепями - линиями связи.

Но графический компилятор в такой системе отсутствует, поскольку в генерации каких-либо кодов и в дальнейшей их компиляции необходимости нет. Вместо него используется программный модуль, который "просматривает" графический рисунок проекта и по нему формирует так называемый "файл описания проекта". Это обычный текстовый файл, содержащий всего лишь список всех линий связи графического проекта какой-либо системы управления и список компонент, которые технолог использовал в данном проекте. В принципе, все ровным счетом выполняется так же, как это делается в любом программном симуляторе.

А вот в качестве системы исполнения применяется программный модуль схемной эмуляции. Эмулирующее ядро - это не интерпретатор. Эмуляция (схемотехническое моделирование) по своей природе не имеет ничего общего с компиляцией или интерпретацией исходных текстов программ. В самом начале своей работы модуль эмуляции "просматривает" файл описания проекта и разворачивает в оперативной памяти контроллера (компьютера) программную модель системы управления, которая была разработана технологом в среде графического редактора.

Входные воздействия для такой модели снимаются с реальных датчиков, а отклик с программной модели подается на реальные исполнительные устройства. Поэтому всю библиотеку компонент, используемых в среде графического редактора на этапе проектирования рисунка проекта, можно условно разбить на два класса. К первому относятся те, которые отображают реальное оборудование: различные датчики и исполнительные органы, а ко второму - компоненты, не имеющие "железного" аналога. Это чисто функциональные элементы, такие как нормализаторы сигналов, частотные фильтры, сумматоры, компараторы, логика и т.п. Каждая компонента имеет графический образ в среде графического редактора и программный модуль в среде исполнения. Весь процесс разработки при таком подходе больше напоминает детский конструктор. На экране дисплея собирается схема из необходимых "кубиков" и надо только будет не забыть соответствующий аппаратный "кубик" подключить к общему интерфейсу контроллера.

Модуль эмуляции - это программа с особой архитектурой. Поэтому для обеспечения ее функционирования потребуется микропроцессорная платформа специальной конфигурации. Есть смысл разместить модуль эмуляции в постоянной памяти микропроцессора, закрыть его битом секретности и рассматривать уже законченное аппаратно-программное решение как универсальную эмулирующую платформу. Почему универсальную? Потому, что по своим ценовым и функциональным возможностям, а также возможностям масштабирования, она пригодна для использования в самом широком спектре областей - от лабораторного студенческого стенда или инженерного исследовательского центра до АСУ ТП масштаба предприятия, т.к. принцип эмуляции прекраснейшим образом поддерживает идеологию распределенного управления.

Реализация такой платформы на основе недорогого микропроцессора будет представлять собой недорогой коммерческий вариант, доступный самому широкому кругу пользователей. Не менее интересным представляется вариант "аппаратной" реализации алгоритма эмуляции на основе микросхем программируемой логики. Дело за фирмой, которая возьмется за "раскрутку" такого проекта.

Вряд ли стоило бы заниматься популяризацией идеи эмуляции, если бы отсутствие этапа генерации исходных /исполняемых кодов было бы единственным отличием рассматриваемого подхода к системе графического программирования от традиционных методов с компиляцией или интерпретацией. Именно идея отказа от всякого рода промежуточных и исполняемых кодов и служит источником большого числа принципиально новых возможностей. Давайте рассмотрим, какие преимущества дает применение схемотехнической эмуляции применительно к "программированию" систем АСУ ТП и вообще идеи эмуляции алгоритмов.

Ныне самым высокоуровневым языком графического программирования, вероятно, следует считать язык функциональных блоковых диаграмм (FBD). В связи с этим было бы логично назвать такой уровень описания проектов уровнем функций. Как корабль назовешь, так он и поплывет. Но все-таки было бы интересно разобраться в функциональных воз-

можностях этого языка. А для этого нелишним будет еще раз вспомнить, что в "обязанности" любого известного графического компилятора входит просмотр графического рисунка проекта и генерация исходных кодов программы. Но любая программа, будь-то сгенерированная автоматически или составленная программистом вручную, имеет еще один графический эквивалент - схему алгоритма. Рисунок алгоритма является самым подробным графическим отображением любой программы. Можно сказать, что в сравнении с FBD, LD и другими языками, представление проекта на уровне рисунка алгоритма программы можно назвать нижним уровнем графического описания проекта. Тогда можно совершенно справедливо заключить, что представление графических проектов управления на любом известном языке графического программирования высокого уровня, в конечном счете, может быть сведено к уровню схемы алгоритма, компонентами которой являются всем известные "кубики": последовательность операторов, разновидность циклов, переключатели, вызовы процедур и функций. Это позволяет сделать прозрачный вывод, что представление проектов на уровне современных языков графического программирования в функциональном плане соответствует всего-навсего уровню алгоритмов. И не выше!

Графическое программирование для систем эмуляции может быть выполнено на языке, обладающим гораздо большими функциональными возможностями. Такой язык можно было бы назвать языком структурных схем, который должен соответствовать уровню систем.

Зачем нужен новый язык? В психологии и других смежных науках давно доказано, что язык и мышление - вещи взаимосвязанные. Чем выше организован язык, тем развитее мышление, и наоборот. Поэтому естественно предположить, что среди папуасов экзотических стран вряд ли когда-нибудь появится Пушкин или Шекспир. По аналогии можно заключить, что использование языка структурных схем позволит создать проекты более высокого уровня сложности, чем это позволяют делать существующие языки графического программирования и языки программирования вообще.

АСУ ТП, включающие в себя системы прогнозирования и принятия решений, экспертные системы - безусловно, они соответствуют уровню

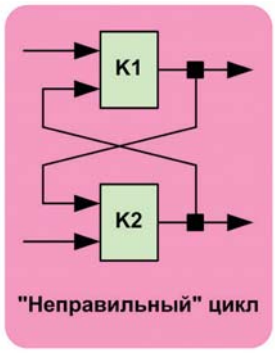
Простой алгоритм

"сложная система". В свою очередь, средой, в которой может исполняться язык структурных схем, может быть исключительно система схемной эмуляции, но никак не среды компилирования или интерпретации. Система - понятие более высокого уровня, чем схема алгоритмов. А приставка "сложная" добавлена для того, чтобы подчеркнуть, что современным SCADA и SoftLogic программам вообще не под силу компиляция и исполнение проектов, представленных на уровне систем.

Для уровня алгоритмов резонно предположить, что сложность любого проекта следует оценивать всего лишь количеством каких-либо "кубиков", которые технолог или программист использовали в графическом проекте системы управления, и сложностью самого алгоритма. Но такое определение сложности не вполне полное, поскольку количество определяет только объем программы. Сложность же алгоритма - понятие вообще субъективное. То, что для одного человека есть просто и понятно, для другого - выступает апофеозом сложности. А вот компилятору и процессору все равно, с каким алгоритмом работать, разница только во времени компиляции и исполнения/интерпретации. Правда, говорят, что процессору лучше вообще ничего не делать - меньше греться будет. ☺

В свою очередь, для уровня систем можно дать четкое определение, что сложность проекта следует определять наличием и количеством перекрестных обратных ветвей (цепей), примененных в проекте, сложными временны́ми соотношениями сигналов в разных ветвях рисунка проекта по отношению друг к другу, а также сложным видом и природой сигналов в каждой отдельной ветви.

Самый примитивный алгоритм можно изобразить просто цепочкой кубиков-компонент. Однако интересные вещи начинаются с включением в проект обратных связей. Но всякие ли обратные связи способны "переварить" современные компиляторы? Под словом "переварить" следует однозначно понимать ситуацию, когда, встретившись с такой "штукой", каждый из них будет в состоянии ее распознать и грамотно вставить сгенерированный блок операторов в тело программы. Оказывается, что современные компиляторы в состоянии распознать и закодировать только простые и простые вложенные связи. А все из-за того, что каждую обратную ветвь компилятор должен привести к циклу, а циклы в программировании могут быть только вложенными. И по одной простой причине - основной принцип программирования в архитектуре фон Неймана гласит, что прерывания для процессора могут быть только вложенными. Потому как внутри самих процессоров они реализуются стековыми регистровыми структурами (вот они - истоки "геморроя"!). Поэтому абсолютным нонсенсом с точки зрения программирования будет выглядеть, к примеру, представленный ниже проект. Потому как никакие компоненты в программировании не могут исполняться одновременно.



"Неправильный" цикл

Однако все это есть абсолютно нормальным явлением применительно к схемотехнике и системотехнике. Программные симуляторы, конечно же, не нарушают основополагающих принципов программирования, но они создают среду, которая позволяет имитировать такие нарушения. Любая программа, в том числе и схемной эмуляции, попав в среду современных процессоров, должна мириться с ее особенностями - регистровой организацией - идеологией, которая остается неизменной на протяжении всей истории их существования.

Поэтому скорость эмуляции в этом случае будет соизмерима со скоростью исполнения обычных программ, что годится для случаев управления относительно медленными технологическими процессами. Для обработки же быстрых процессов (эмуляции очень-очень больших структурных схем) модуль эмуляции необходимо поместить в "родную" среду. Поскольку алгоритм эмуляции основан на принципах параллельных потоков команд и данных, то наиболее оптимальным является вариант "аппаратной" реализации алгоритма эмуляции на ПЛИС-эмулирующей платформе.

Все проекты, представленные на каких-либо известных языках графического программирования непременно приводятся к простому списку программных операторов, которые могут быть исполнены процессором только последовательно. Другими словами, первейшей задачей любого графического компилятора есть определение порядка размещения программных модулей (Mn) в теле программы согласно всех связей, соединяющих графические образы соответствующих им компонент (Kn) на рисунке проекта.

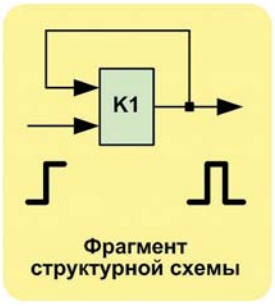
Как следует из эпюры, модуль M3 может быть расположен только в конце тела программы, поскольку оба входных его параметра (α, β) должны быть определены к началу его исполнения. В роли параметров, например, могут выступать идентификаторы. Приведенный пример позволяет наглядно показать, что если программист или технолог разработают алгоритм управления, состоящий из миллионов операторов, то все они одновременно будут "висеть" мертвым грузом в памяти компьютера и каждый будет ждать своей очереди на исполнение. В каждый отдельный момент времени процессор может исполнять только один оператор. Тогда о какой сложности временных соотношений в ветвях алгоритма может идти речь?

В свою очередь, если проектирование системы управления будет вестись на уровне структурной схемы (уровень системы), то в представленный проект может быть вложен совершенно другой смысл. Такой, например, что при изменении сигнала на входе компоненты K1 с уровня логического нуля до уровня логической единицы, на выходе K3 формируется импульс. Представленную схему можно рассматривать как фрагмент более сложной структурной схемы, в которой сформированный строб используется, например, для "сброса" каких-то узлов, счета перепадов и т.п.

Приведенный фрагмент наглядно показывает, что сложный (во времени) вид сигнала сформировался на выходе компоненты K3 в результате "гонки" сигналов в ветвях 1 и 2 схемы, что есть совершенно нормальным явлением для уровня структурных и принципиальных схем электронных систем, но представляется абсолютным нонсенсом для схем алгоритмов, а значит и программирования вообще.

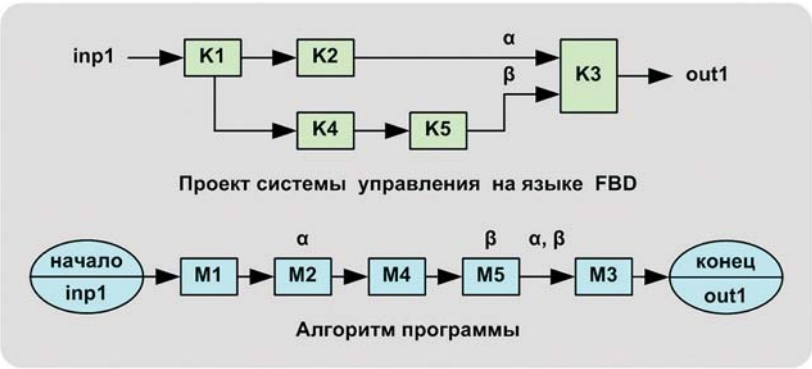
Говоря о проектировании схемы управления на уровне систем, подразумевается, что система эмуляции программных кодов не формирует, а "работает" исключительно по рисунку проекта. Поэтому в данном случае распространение сигналов по ветвям 1 и 2 происходит параллельно и говорить о возможности построении схемы алгоритма (единой цепочки модулей) совершенно некорректно.

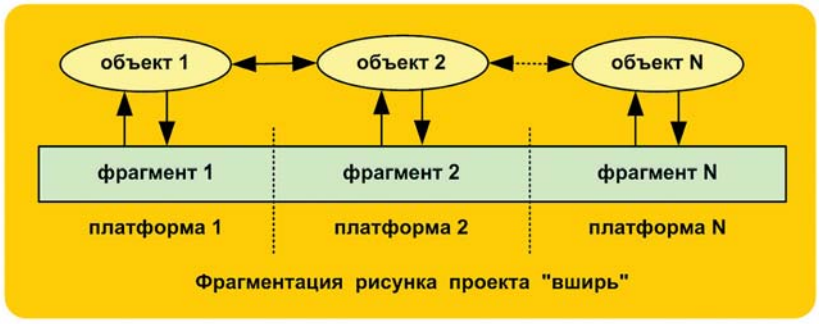
Сложный вид сигналов в системах можно получить не только путем организации гонок сигналов по разным цепям, но и применением обратных связей на одной компоненте.



В системах программирования каждой ветви алгоритма во взаимно однозначное соответствие может быть поставлен статический идентификатор! Системам эмуляции (даже самому простому симулятору) под силу так называемое многозначное кодирование. Это значит, что по ветвям структурной схемы могут одновременно передаваться сигналы разной природы - многозначной логики, аналоговый сигнал, сложный дискретный сигнал, значение импеданса, двунаправленные потоки данных... Все эти свойства являются необходимыми для проектирования действительно сложных систем.

Теперь о масштабировании. Под масштабированием следует понимать возможность наращивания суммарной мощности системы эмуляции путем простого добавления в нее требуемого количества однотипных





Фрагментация рисунка проекта "вширь"

плат - универсальных эмулирующих платформ.

Механизм масштабирования в системах эмуляции основан на идее разбиения рисунка проекта на фрагменты и размещения каждого такого фрагмента на отдельной платформе. Возможны три варианта разбиения.

Первый - "фрагментация вширь". Это тогда, когда необходимо автоматизировать, к примеру, конвейер, состоящий из отдельных станков, но которые должны синхронно взаимодействовать между собой в функциональном плане. Другими словами, подчиняться общей задаче. В этом случае каждая платформа управляет локально только тем станком, алгоритм управления которым соответствует фрагменту рисунка, в нее загруженному. И все!

Еще раз необходимо отметить, что система эмуляции, находящаяся в среде фон Неймана, только имитирует всякую параллельность. Тем не менее, это позволяет реализовать распараллеливание процессов, не прибегая к традиционным методам организации межпроцессорного обмена и синхронизации процессов, т.е. использования разного рода "флагов", "семафоров" и т.п.

Современным же системам программирования такое не под силу. В случае распараллеливания общего процесса всю "графику" необходимо отложить в сторону и вручную, средствами универсальными языков программирования, разрабатывать ПО межпроцессорного обмена и синхронизации процессов.

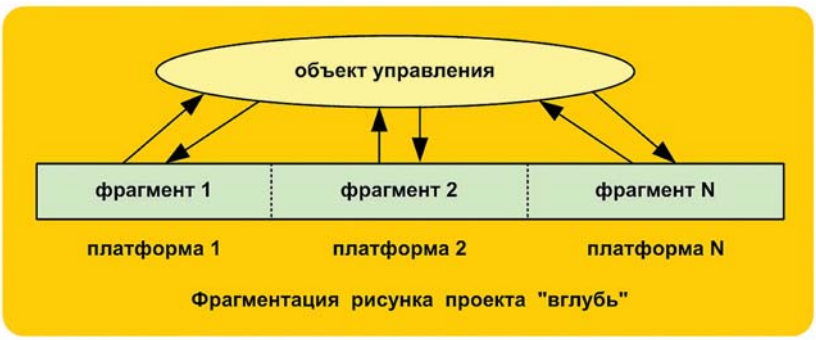
К тому же, не следует забывать, что даже на уровне теории задача эта далека от окончательного разрешения. Лучшим доказательством этому служит непрекращающийся поток публикаций научных статей по данной тематике и защит диссертаций.

Для организации "истинного" распараллеливания, например, для случая управления быстрыми процессами, необходимо использовать ПЛИС - эмулирующую платформу.

Вариант второй - "фрагментация вглубь". Это соответствует случаю, когда ресурсов одной платформы недостаточно для размещения в ней всего алгоритма управления одним объектом управления. В этом случае общий рисунок проекта, опять же, разбивается на фрагменты, каждый из которых загружается в отдельную платформу. Все платформы в этом случае находятся на одном объекте управления.

Вариант третий - смешанный, когда задействованы одновременно оба предыдущих варианта.

Можно назвать и два уровня распараллеливания. Первый - идея распараллеливания заложена в самом алгоритме схемной эмуляции, которая позволяет вести эмуляцию одновременно по всем N ветвям всего рисунка графического проекта или его фрагментов. Можно сказать, что проектирование систем управления на уровне структурных и функциональных схем приближает их разработку к уровню системотехники и проектирования электронных систем, где



Фрагментация рисунка проекта "вглубь"

изменение сигналов, в каждый отдельный момент времени, происходит одновременно в разных цепях схемы. К тому же, сигнал в каждой ветви может иметь сколь угодно сложный вид во времени. Поэтому применительно к этому случаю справедливо считать, что эмуляция проектов на уровне систем на все 100% поддерживает параллельность процессов в них происходящих.

В системах же программирования идея распараллеливания практически решена только для случаев вычислений однородных систем уравнений, к которым можно отнести, к примеру, задачу быстрого преобразования Фурье для систем цифровой обработки сигналов.

Что касается вопроса распараллеливания процессов в обычных программах, то задача эта еще далека от практического разрешения и пока находится больше на уровне теоретических исследований. По-видимому, наиболее существенным продвижением можно считать случай аппаратной реализации оператора IF в современных процессорах. Да и то путем решения проблемы "в лоб". Процессор ведет вычисления по обоим ветвям ("впрок"), и только тогда, когда становится известной истинность какой-либо ветви, ее данные используются для дальнейших вычислений. Результаты вычислений по ветви, получившей статус "ложь" в кэше процессора, просто уничтожаются.

Второй уровень - это идея разбиения всего рисунка проекта на фрагменты и размещение каждого фрагмента на отдельной платформе. Таким образом, фрагментация всего рисунка (вширь, в глубину и смешанная) по отдельным платформам добавляет процессу эмуляции еще одну степень распараллеливания.

Несомненно, эмуляция прекрасным образом способствует решению проблем "Движение за открытую проектную документацию" и стиля программирования, но не только. Объемы программ настолько выросли, что разобраться в любой из них, ковыряясь в мегатоннах исходных текстов, уже просто невозможно. И новая проблема растет экспоненциально с ростом объема кода. Это приводит к такой тривиальной ситуации, когда на место уволившегося программиста приходиться нанимать специалиста более высокой квалификации, который смог бы разобраться в творении предыдущего.

Каждому программисту хорошо знакомо чувство потерянности, которое он испытывает, впервые видя горы плохо документированных, или вообще не документированных, исходных кодов. В этом случае все проектные решения, задуманные автором проекта, просто растворяются в деталях кодирования. Выходит, что без исходных кодов плохо, но и с ними не очень хорошо. Чего не хватает для счастья? Проектной документации!

В многочисленных статьях, посвященной данной проблеме и путям ее разрешения, нередко встречается мысль - вот умеют же выпускать документацию к электронной аппаратуре!

С проблемой выпуска хорошей документации самым тесным образом переплетается другая, даже более ужасная - стиль и методика программирования, которая уже становится настоящим бичом программирования, и которая грозит программирование, как отрасль, поставить на грань кризиса.

А все дело в том, что программирование все еще остается искусством. И здесь подразумевается, к сожалению, не идея искусного написания программ, а принцип написания - как Бог на душу положит. Часто программирование сравнивают с процессом рисования художником картин, или создания литературного произведения писателем.

Каждый программист программирует так, как умеет. И попытки формализовать процесс написания программ пока оказываются несостоятельными. И опять-таки, в многочисленных статьях, посвященным путям решения этой проблемы, красной нитью проводится авторами мысль, что программы должны не писаться, а проектироваться. Так, как это обстоит с проектированием и производством электронной аппаратуры.

Вот и спрашивается, зачем тратить огромные усилия на поиск какой-то чудо-методики, которая якобы позволит, наконец, решить проблему проектирования программ и выпуска хорошей документации к ним? И которая, еще не известно, будет ли найдена?

Зачем изобретать велосипед, если он уже давно изобретен и с успехом используется? Это - методики проектирования электронных систем, на которые так любят ссылаться сами же теоретики от программирования. Просто нужно начать разрабатывать программы так, как разрабатываются электронные системы - на уровне функциональных и структурных схем. И тогда с программированием ситуация окажется даже более простой и приятной, чем с самой электроникой. Начинаться и заканчиваться разработка будет исключительно на рисунках проектов. В таком случае нет необходимости "браться за паяльник" и производство печатных плат (и прочих производственных атрибутов электроники), потому как универсальная плата схемной эмуляции оживит любой проект, реализованный только до уровня документации.

А что можно сказать об эмуляции систем с заранее неизвестным алгоритмом? Постановка вопроса справедлива, конечно, для сложных систем, алгоритм работы которых неизвестен, а может быть и вообще нельзя его описать при уровне современных знаний. Применительно к АСУ ТП это могут быть различного рода интеллектуальные системы - экспертные системы, системы принятия решений, нейронные сети и т.п.

К примеру, можно достаточно подробно изучить поведение одного нейрона, входящего в состав какой-либо структуры, и описание его поведения занести в библиотеку компонентов системы эмуляции. Затем достаточно только описать всю структуру, т.е. таблицу соединений, и начать ее эмуляцию. Таким образом, зная только поведение части целого, можно исследовать методом обучения все целое, тиражируя в дальнейшем обученную систему по копиям в промышленном масштабе.

На основании вышеприведенных примеров и рассуждений должно стать понятным, что для всех известных систем программирования, и для тех, которые еще когда-нибудь появятся, любой графический проект системы управления не может выглядеть сложнее, чем созданный на языке FBD.

Будет справедливо сказать, что внедрение в практику систем схемной эмуляции послужит толчком к развитию новых графических языков, соответствующих более высокому уровню описания, чем уровень описания алгоритмов и функций - языка структурных схем. К тому же, проектирование систем управления на таком уровне более информативно, наглядно, понятно всем, как заказчику, так и технологу, и позволяет быстрее провести разработку.

Как правило, АСУ ТП предприятия представляет собой двухуровневую систему управления. На нижнем уровне расположены контроллеры, обеспечивающие первичную обработку информации, поступающей непосредственно с объектов управления, и управление этими объектами. На верхнем уровне размещаются мощные компьютеры, выполняющие функции серверов баз данных для хранения и анализа поступившей информации, и рабочие станции для визуализации процессов управления.

В соответствии с идеологией эмуляции алгоритмов, в принципе, нет смысла что-либо эмулировать на верхнем уровне АСУ ТП. Все необходимое управление подразумевается на уровне контроллеров, а на станциях - только рабочие столы на дисплеях операторов, состоящих из красивых картинок, отображающих ход технологического процесса. Но отображение процессов - это одно назначение рабочих станций. Не менее важным следует считать возможность ввода оператором в систему нижнего уровня всякого рода установочных коэффициентов и параметров с целью изменения режима работы объекта управления.

При такой конфигурации АСУ ТП даже временный выход из строя компьютера оператора, или отключение его по какой-то другой причине не повлияет на ход технологических процессов. Это сделает некритичным использование в контуре управления недорогих компьютеров вместо дорогостоящих промышленных серверов.

Но есть, по крайней мере, два повода, видимых уже сегодня, которые сделают использование эмуляции желанным процессом и на верхнем уровне.

Повод первый. Дело в том, что в настоящее время все больше уделяется внимание автоматизации контроля и управления технологическими процессами в условиях аварий для потенциально опасных объектов. Уровень их сложности вырос настолько, что человек физически не в состоянии быстро и адекватно отреагировать на опасные ситуации, и тем более их спрогнозировать. Написание каких-то программ, автоматизирующих этот процесс, пока также не дало желаемого результата ввиду сложной логики принятия решений. Вот и пытаются эту проблему решить путем внедрения различного рода "экзотических" вещей, таких как нейронные сети или экспертные системы. Стоит только отметить, что системы эти мало того что сверхдорогие, но более относятся пока к области экспериментов и научных исследований.

Поэтому решение проблемы будет гораздо более простым и дешевым, если спроектировать модель объекта управления более высокого уровня, чем та, которая эмулируется в контроллерах, и синхронно эмулировать ее на компьютерах верхнего уровня. Тогда любые команды оператора сначала следует подавать на вход модели верхнего уровня. Это позволит в реальном масштабе времени спрогнозировать на ней возможные опасные отклонения технологического процесса. Если в результате воздействия команды оператора модель не перешла в аварийное состояние - сигнал управления пропускается в контроллеры нижнего уровня.

Что же касается вопроса автоматической выработки упреждающих сигналов управления - действительно, вряд ли можно обойтись без использования экспертных систем и систем принятия решений. И здесь идея эмуляции может стать очень полезной.

Повод второй. То, что выход станции со строя на какое-то время не повлияет на систему управления в целом - это приемлемо, например, для некоторых отраслей промышленности. Но вряд ли станет приемлемым для бортовых систем управления, к примеру, летательными и космическими аппаратами. Ведь даже применение сверхдорогих специализированных компьютеров до конца не способно решить вопрос "глюков" аппаратуры и, в особенности, программного обеспечения. В свете этого весьма интересным может выглядеть идея разработки уже не просто эмулирующей платы, а эмулирующего компьютера. Сердцем его будет не всем привычный универсальный процессор, к примеру, Pentium, а все та же эмулирующая платформа, выполненная на микросхемах программируемой логики.

Конечно, трудно представить, как можно было бы средствами графического программирования разработать систему подобную Windows-приложениям верхнего уровня. Тем не менее, задача такая представляется не столь фантастической, как могла бы показаться на первый взгляд. А как разрабатываются сверхбольшие интегральные микросхемы? Ведь в голове никакого самого гениального разработчика не запомнится расположение десятков миллионов транзисторов и никаким взглядом такое не охватишь! Здесь на помощь приходят языки очень высокого уровня, что-то вроде языка алгоритмов. Используя такой язык разработчик описывает алгоритм работы будущей системы. А сверхбольшой рисунок генерируется автоматически!

Поэтому разработка таких языков и компиляторов к ним может стать актуальным уже не только для систем проектирования СБИС, но и для PC-приложений. Ведь это - прямая дорога к созданию действительно надежных программ, свободных от ошибок программирования, связанного с невнимательностью или усталостью программиста, уровня профессионализма, большим объемом исходного кода, в котором легко "заблудиться" и т.д. И тогда программирование перестанет быть искусством! А специалисту, приступающему к изучению уже написанной ранее кем-то программы, не надо будет перемалывать мегатонны исходных текстов, чтобы понять алгоритм, ею реализуемый.

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