ПРИКЛАДНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

УДК 004.31

А.А. Лялинский

ОСОБЕННОСТИ ПОСТРОЕНИЯ ПРИКЛАДНЫХ ПРОГРАММ С ВЕБ-ДОСТУПОМ

Рассматриваются вопросы организации веб-интерфейса к модулям, входящим в состав систем автоматизированного проектирования (САПР) в микроэлектронике. Исследуется влияние ряда факторов (доступной памяти компьютера, максимально разрешенного времени выполнения, степени однородности программного обеспечения) на структуру создаваемой системы. Описываются синхронный и асинхронный варианты взаимодействия управляющей и исполнительной частей. Даются ссылки на системы, позволяющие иметь доступ к прикладным программам в САПР микроэлектроники и построенные на принципах, изложенных в статье.

Введение

В микроэлектронике САПР активно используются уже десятки лет. Общепринятым для них является размещение на локальном компьютере пользователя или в локальной сети предприятия. Общение с такой САПР ведется через встроенную систему ввода-вывода данных (также известную как GUI – *Graphical User Interface*), тесно интегрированную с программным обеспечением (далее – специальное ПО), реализующим те задачи, на которые ориентирована данная САПР.

Оснащение систем проектирования веб-доступом, с одной стороны, обеспечивает более широкие возможности по их использованию, существенно расширяя круг лиц, которым эти системы становятся доступны. С другой стороны, с введением веб-доступа возникает ряд проблем юридического и технического характера. Не затрагивая вопросы соблюдения лицензионных соглашений и прав на использование такого ПО, рассмотрим только особенности построения данного специального ПО.

1. Структура связки «веб-сайт – специальное ПО»

Объект «веб-сайт – специальное ПО» (рис. 1) включает следующие части:

– ПО, отвечающее за входной веб-интерфейс и реализованное на предназначенных для этого языках (HTML, PHP, Javascript и т. п.);

– специальное ПО, обеспечивающее собственно вычислительные задачи, для решения которых и создана данная система;

– блок визуализации выходных данных.

Здесь может присутствовать также блок управления, но его роль обычно невелика, так как на нем лежит только задача подготовки пакетного запуска специального ПО.

Интерфейсная часть и блок визуализации размещены на сервере и, в случае использования языков Javascript или Java, частично на клиентской машине. Специальное ПО всегда размещено на сервере.

На ПО веб-интерфейса и блока визуализации лежит ответственность за прием и передачу всех данных между пользователем и специальным ПО. Блок входного интерфейса (или блок управления, если он есть) является управляющим по отношению к блоку специального ПО, т. е. вызовы всегда идут от интерфейсной части к специальной, а не наоборот. Последовательность действий такова:

1. Интерфейсная часть в режиме диалога обеспечивает прием данных от пользователя (по частям или единым пакетом).

2. При поступлении от пользователя сигнала об окончании формирования данных проверяется их реальная полнота и непротиворечивость отдельных частей входного пакета друг другу.

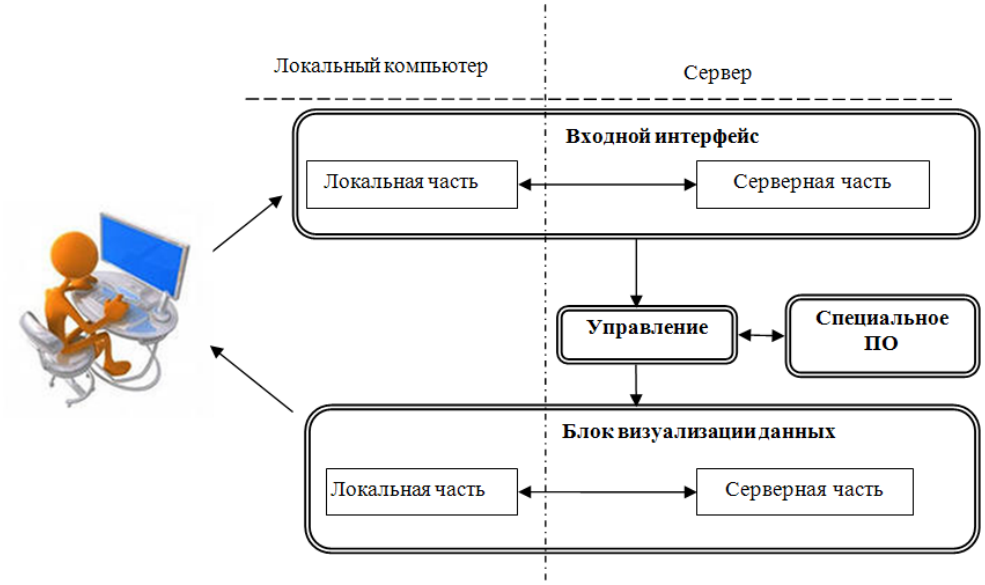

Рис. 1. Структурная схема связки «веб-сайт – специальное ПО»

3. Если получена положительная оценка полноты и правильности входных данных, то формируется внутренний блок данных, поступающий на вход блока специального ПО. Этот блок может быть размещен либо в оперативной памяти сервера, либо, если объем данных велик или непредсказуем, в виде файла на сервере. Если специальное ПО реализовано на скриптовом языке (например, на Матлабе [1, 2]), то при небольшом объеме входных данных можно отказаться от генерации специального файла входных данных, а вместо этого создавать модуль на скриптовом языке, в котором эти данные заданы путем инициализации переменных.

4. От интерфейсной части управление передается на формирование пакета вызова блока специального ПО.

5. Далее возможны два варианта взаимодействия управляющей и исполнительной частей системы. При *синхронном* варианте работа управляющей части приостанавливается до полного завершения работы исполнительной части. Это более простой вариант, можно сказать, «вариант по умолчанию». В *асинхронном* варианте управляющая часть после запуска части специального ПО начинает выполнение цикла, в котором через определенные промежутки времени происходит опрос флагов состояния блока специального ПО или наличия и/или содержимого определенных файлов на сервере. Выход из цикла возможен либо при получении сигнала об окончании выполнения задачи на стороне процесса специального ПО, либо при превышении заранее определенного лимита времени.

6. При завершении работы исполнительного блока извлекаются результаты его работы (скорее всего, это будет файл на сервере).

7. Если выходные данные блока специального ПО получены, они передаются на вход модуля визуализации данных с последующим его вызовом.

2. Особенности интерфейсной части

Та часть входного интерфейса блока, которая организует прием данных от пользователя, особого интереса в рассматриваемом случае не представляет, так как занимается обработкой информации, полученной из входных форм <input … >, которые заключены в блок (или блоки) <form> … </form>. Следует только обратить внимание на обязательное наличие проверки введенной информации как по линии защиты от хакерских атак, проводимых путем внедрения взламывающего кода в информацию, которая размещается в полях входных форм, так и по линии ее соответствия (как в целом, так и отдельных частей) требованиям, предъявляемым к входной информации данной конкретной задачи.

Рассмотрим организацию управляющего вызова из этого блока модуля специального ПО, который, собственно, и выполняет вычислительную задачу. Различного рода комбинации вызовов вытекают из ограничений, накладываемых на объем оперативной памяти, которая требуется для выполнения специальной части программы, и времени ее выполнения. Кроме того, важную роль играет вопрос об однородности ПО интерфейсной и специальных частей. Далее вкратце рассмотрим влияние этих трех факторов (памяти, времени выполнения, однородности ПО) на структуру системы.

2.1. Ограничение по оперативной памяти

В современных компьютерах требования к объему оперативной памяти, занимаемому выполняющей программой, отошли на второй план. Причина заключается в том, что при ее нехватке происходит сброс части памяти ОЗУ на диск с последующим ее восстановлением при обращении алгоритма к сохраненной на диске части программы (так называемая «виртуализация памяти»). Таким образом, вместо ограничения на память ОЗУ имеем ограничение на объем дискового пространства, отведенного под хранение виртуальных страниц, причем пользователь при необходимости может легко его увеличить. Естественная плата за использование виртуальной памяти – замедление работы программы из-за потерь на обмен страниц памяти между ОЗУ компьютера и диском. Подчеркнем, что главное здесь – отсутствие жесткого ограничения на объем используемой памяти.

К сожалению, при работе скриптового ПО, на котором написано большинство современных сайтов, имеет место реальное фиксированное ограничение на объем памяти, отведенной под динамически размещаемые блоки данных. Так, язык PHP в своем конфигурационном файле php.ini имеет параметр *memory_limit*, в котором задается ограничение на допустимый объем динамических данных (8 МБ для версий PHP до 5.2.0, 16 МБ для более старших версий), которые можно разместить при работе исполняемого скрипта. При нехватке памяти программист (создатель скрипта) может попытаться с помощью функции *ini_set* (memory_limit, запрашиваемый размер) расширить эту границу, что принципиально данную проблему не решает, так как запрашиваемый размер не может быть очень большим.

Отмеченное ограничение имеет значение, если известно, что модуль специального назначения сам по себе невелик и для простоты мог бы быть реализован на сервере непосредственно на одном из скриптовых языков, но во время своей работы он требует больших объемов динамической памяти (например, в задачах обработки образов). В этом случае приходится считаться с объемом доступной динамической памяти.

Итак, при реализации специальной части на скриптовых зыках следует помнить о существовании жесткого ограничения на доступную динамическую память.

2.2. Ограничение времени выполнения

При запуске задачи на локальном компьютере (или на любом компьютере, доступном в локальной сети) ограничения по времени выполнения задачи практически нет. Для больших задач реально оно определяется терпением пользователя или истощением дискового пространства выходными данными, что опять-таки не является жестким ограничением.

К сожалению, ограничение по времени имеет место при запуске задачи на веб-сервере. Причина заключается в том, что связь между клиентом, дающим команду со своего локального компьютера на запуск задачи, и веб-сервером, на котором размещена задача специального вида, осуществляется через канал, представляющий собой совокупность программных средств и физических линий связи. Так как пропускная способность канала не безгранична, существует конкретное ограничение по максимальному времени ожидания отклика от сервера (*timeout limit*). Это ограничение повышает пропускную способность канала, так как позволяет выводить из оборота «повисшие» запросы к серверу. Реально речь идет об отрезке времени в 30 с. Можно попытаться расширить этот временной диапазон изменением настроек веб-сервера, что принципиально проблему не решает.

Итак, второе ограничение на специальное ПО – малое время выполнения. Позже увидим, что это ограничение можно преодолеть.

2.3. Однородность интерфейсной и специальной частей

Интерфейсная часть, отвечающая за непосредственный контакт с пользователем, обычно представляет собой набор модулей, написанных на статических языках описания (HTML, XML), и поведенческих модулей, написанных на каких-либо скриптовых языках (PHP, JavaScript, ASP, Perl и т. п.). Скриптовые языки обычно исполняются в режиме интерпретации. Существует возможность написать интерфейсную часть и на компилируемом языках (C, C++, C#), но в данном случае это не представляет интереса из-за большей трудности написания (по сравнению со скриптовыми языками), малой распространенности и из-за того, что канал связи всегда работает медленнее компьютера и не позволяет воспользоваться ускорением, полученным при использовании компилируемых языков.

Специальное ПО обычно уже существует в каком-либо виде к тому моменту, когда возникнет потребность в подключении к нему веб-доступа. Вариантов программной реализации специального ПО великое множество, но в данном случае существенно только то, написано оно в виде скриптов или в виде компилируемых модулей. Если специальное ПО написано в виде скриптов, то проблем с разнородностью скриптов не будет, данные между управляющим и исполнительным скриптами могут передаваться через механизмы глобальных переменных $_POST, $_GET, $_SESSION, управление между скриптами осуществляется посредством http-вызовов. Если же специальное ПО написано на компилируемом языке (или на скриптовом, но выполняемом в режиме компиляции MATLAB), то такой гибкости уже нет. Данные в специальную часть и обратно передаются посредством файлов, управление обычно одностороннее – только в виде вызовов от интерфейсной части к специальной. Поэтому для небольших задач, написанных в исходном виде на компилируемом языке, имеет смысл рассмотреть вопрос о целесообразности их перевода на скриптовый язык.

3. Взаимодействие частей системы

Введем обобщенное определение управляющей и исполнительной частей системы. *Управляющая* часть включает в себя ПО, размещенное на сервере и, возможно, на локальном компьютере клиента и отвечающее как за организацию диалогов для приемки входных параметров, так и за обеспечение визуализации выходных данных. *Исполнительная* часть размещена только на сервере и отвечает за вычислительные аспекты решения задачи.

Как уже упоминалось ранее, возможны два варианта взаимодействия управляющей и исполнительной частей системы: *синхронный* и *асинхронный* (рис. 2).

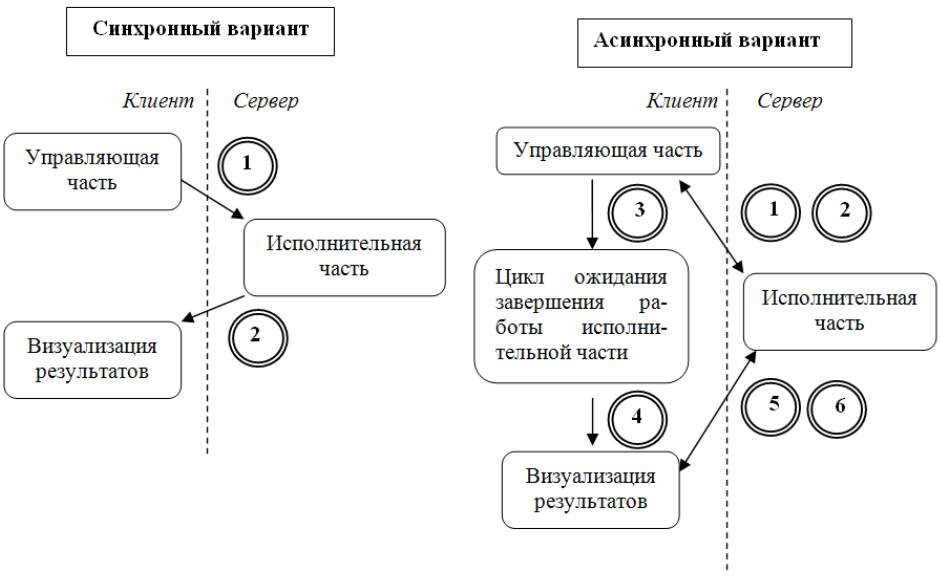

Рис. 2. Варианты взаимодействия управляющей и исполнительной частей

Синхронный вариант предполагает обычную последовательность работ: управляющая часть выдает задание исполнительной (шаг 1) и приостанавливает свою работу в ожидании сигнала завершения вычислительной задачи. По завершении работы исполнительной части происходит визуализация полученных результатов (шаг 2). Какие-либо средства синхронизации задач не применяются.

При выборе асинхронного варианта принципиальной является одновременная работа обеих частей системы и использование специальных методов для синхронизации их работы. Управляющая часть посылает запрос на исполнение (шаг 1) и сразу же возвращается к себе (шаг 2), запуская цикл ожидания завершения работы исполнительной части (шаг 3). При получении сигнала о завершении работы вызывается блок визуализации (шаг 4), который запрашивает (шаг 5) и получает (шаг 6) данные. Преимущество асинхронного подхода заключается в том, что он позволяет преодолеть ограничение на время работы исполнительной части программы (так как управляющая часть не держит канал занятым только для того, чтобы узнать об окончании работы исполнительной части – вместо этого проводится периодический опрос флагов состояния).

Использование синхронного варианта работы возможно (и предпочтительно), если известно, что время работы исполнительной части не превышает установленный максимальный таймаут. В этом случае можно ожидать, что соединение клиента с сервером не будет разорвано из-за превышения таймаута и система в целом обеспечит устойчивую работу. В противном случае необходимо использовать асинхронные методы.

В настоящее время на веб-серверах наиболее распространена организация асинхронной работы на основе технологии AJAX [3, 4]. Рассмотрим пример ее работы.

3.1. Схема работы с использованием AJAX-технологии

AJAX-технология предполагает написание на языке Javascript модуля, в котором на стороне клиента выполняются следующие действия:

– создание XMLHttpRequest-объекта;
– подготовка запроса;
– указание функции, отслеживающей состояние запроса;
– отправка запроса.

Создание XMLHttpRequest-объекта. Объект XMLHttpRequest дает возможность браузеру выполнять запросы к серверу без перезагрузки страницы. Кроссбраузерный текст создания XMLHttpRequest-объекта на языке javascript имеет вид

```
// code for IE7+, Firefox, Chrome, Opera, Safari
  var xmlHttp;
  if (window.XMLHttpRequest)
    xmlHttp = new XMLHttpRequest();
// code for IE6, IE5
  else
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

Подготовка запроса. Для подготовки запроса используется метод open объекта XMLHttpRequest. Имеется несколько вариантов списков параметров данного метода, наиболее простой для асинхронного вызова имеет вид

```
xmlHttp.open (тип запроса, url);
```

где тип запроса – 'GET' или 'POST'. В GET-запросе параметры передаются в строке вызова исполняемого скрипта, в POST-запросе – в теле, посылаемом через метод *send* (см. далее). Объем

параметров при посылке с использованием POST неограничен. Для POST необходим заголовок Content-Type, содержащий кодировку

```
xmlHttp.setRequestHeader ('Content-Type', 'application/x-www-form-
urlencoded');
```

url – адрес вызываемого скрипта.

Указание функции, отслеживающей состояние запроса. При асинхронном вызове в методе onreadystatechange необходимо указать функцию, которая отслеживает состояние объекта и вызывается каждый раз, когда XMLHttpRequest-объект изменяет состояние. Реально из пяти возможных состояний (0 – Uninitialized, 1 – Loading, 2 – Loaded, 3 – Interactive, 4 – Complete) гарантированно можно полагаться только на получение последнего (с кодом 4). Получение остальных зависит от браузера, использовать их не рекомендуется. Кроме проверки состояния XMLHttpRequest-объекта рекомендуется также проверка состояния запроса, возможные ответы: 200 – OK, 404 – Page not found. В результате минимально возможное описание функции имеет вид

```
xmlhttp.onreadystatechange = function() {
    if (xmlhttp.readyState == 4) {
        if(xmlhttp.status == 200) {
          alert(xmlhttp.responseText);
        }
     }
}
```

Отправка запроса. Для отправки запроса используется метод send объекта XMLHttpRequest. При GET-запросе в качестве параметра указывается null, при POST-запросе – список параметров, например

```
xmlhttp.send ("s=5&b=3");
```

После выполнения данной команды происходит непосредственный запуск асинхронно выполняемого скрипта.

3.2. Проблема индикации работы выполняемого блока

При выполнении достаточно длительных задач желательно показывать пользователю степень их выполнения. Учитывая итерационный характер решаемых в САПР микроэлектроники задач, выбрать для показа нужный индикатор нетрудно. К сожалению, в связке «веб-сайт – специальное ПО» возникает проблема технического характера, препятствующая такому показу.

При выполнении задачи в *синхронном* режиме управление полностью передается блоку специального ПО, которое не имеет возможности выводить какие-либо данные на экран компьютера пользователя, так как выполняется в пакетном режиме и вся работа с входными-выходными данными ведется через файлы.

В *асинхронном* режиме одновременно работают и интерфейсная часть, и блок специального ПО. Последний мог бы выводить информацию о своем состоянии в файл, но проблема в том, что в AJAX-технологии управляющая часть написана на javascript и, следовательно, размещена на компьютере клиента. По определению языка javascript в нем в целях безопасности не предусмотрены средства работы с файлами на сервере, поэтому даже при наличии файла состояния задачи нет возможности вывода данных из него на экран.

Таким образом, можно показывать только состояния запуска и окончания выполнения задачи.

4. Визуализация данных

Известно, что основные скриптовые языки, на которых создается большинство веб-сайтов, в своем составе не содержат графических средств, позволяющих проводить полноценную визуализацию данных (строить графики, диаграммы и т. п.). К счастью, в настоящее время существуют графические пакеты различного уровня, позволяющие выполнить такую работу. На сайте документации по PHP5 [5] (наиболее широко используемой версии PHP в настоящее время) доступна информация (есть полное описание, список функций и руководство по инсталляции) по трем графическим пакетам: Cairo [6], GD [7] и ImageMagick [8]. Эти графические пакеты позволяют проводить построение основных геометрических фигур, проводить их перемещение, раскраску, выполнять операции с альфа-каналом, читать и сохранять полученный образ в наиболее известных графических типах данных.

Работу с данными на более высоком уровне предоставляет графический пакет JpGraph [9], который дает возможность построения огромного количество типов графиков и диаграмм. Для бесплатной версии состав доступных функций ограничен, так же, как и возможность ее использования в коммерческих версиях.

Можно также упомянуть бесплатный пакет pChart [10], являющийся надстройкой над GD-библиотекой и предоставляющий средства для отображения различного рода статистической информации.

5. Примеры реализации прикладных программ с веб-доступом

На основе проведенного по данной тематике анализа в ИППМ РАН были созданы следующие системы [11–13]:

1. *Система расчета spice-макромодели интегральной спиральной индуктивности,* позволяющая рассчитывать сосредоточенные параметры индуктивности (собственно индуктивность, межвитковую емкость, сопротивление дорожек), формируемой на кристалле интегральной схемы по геометрическим размерам ее топологии и заданной технологии.

Исполнительная часть реализована в виде exe-модуля, исходный код которого написан на языке C.

2. *Система определения оптимального набора тестов логической функции,* позволяющая создавать оптимальный (с точки зрения последующего spice-моделирования) набор входных тестов для произвольной логической функции. Создаваемый набор тестов имеет минимально возможное количество входных воздействий.

Исполнительная часть написана полностью на PHP.

3. *Система автоматической генерации Verilog-моделей,* позволяющая создавать высокоуровневые модели (на языках Verilog или Verilog-AMS) цифровых комбинационных схем на основе заданного пользователем произвольного логического выражения. Кроме того, при указании технологической spice-библиотеки ячеек возможно построение моделей с временными параметрами. Это дает возможность упростить и автоматизировать процесс разработки высокоуровневых моделей цифровых ячеек, что, в свою очередь, ускоряет разработку крупных проектов в целом.

Исполнительная часть написана на PHP, кроме того, возможен вызов exe-модуля spice-симулятора.

4. *Система моделирования инерционной навигационной системы*, которая позволяет навигатору в течение некоторого времени после потери контакта со спутником продолжать ориентирование на местности исходя из показаний встроенных в него датчиков движения, а также оценивать точность работы инерционной навигационной системы и степень доверия к выдаваемым данным в зависимости от используемых сенсоров движения и параметров настройки ПО.

Исполнительная часть написана на языке Matlab.

В первых трех системах исполнительная часть вызывается синхронно, в последней – асинхронно.

Заключение

Оснащение систем автоматизированного проектирования веб-доступом значительно расширяет возможности по их применению, предоставляя к ним доступ из любой точки, где есть Интернет. Относительно вопроса о том, можно ли добавить веб-доступ к реально существующим САПР-системам, следует признать, что в настоящее время наиболее реально построение САПР с веб-доступом на основе специально написанных (или переработанных) блоков, учитывающих специфику требований, рассмотренных в данной статье.

Список литературы

1. MATLAB – пакет прикладных программ для решения задач технических вычислений и одноименный язык программирования [Электронный ресурс]. – Режим доступа : http://ru.wikipedia.org/wiki/MATLAB. – Дата доступа : 10.12.2012.

2. Дьяконов, В.П. MATLAB R2006/2007/2008 + Simulink 5/6/7. Основы применения / В.П. Дьяконов. – Изд-е 2-е, перераб. и доп. – М. : СОЛОН-Пресс, 2008. – С. 800. – (Библиотека профессионала).

3. AJAX (*Asynchronous Javascript and XML*) [Электронный ресурс]. – Режим доступа : http://ru.wikipedia.org/wiki/AJAX. – Дата доступа : 10.12.2012.

4. Крейн, Д. AJAX в действии: технология – Asynchronous JavaScript and XML = Ajax in Action / Д. Крейн, Э. Паскарелло, Д. Джеймс. – М. : Вильямс, 2006. – 640 с.

5. Руководство по PHP [Электронный ресурс]. – Режим доступа : http://php.net/manual/ru/index.php. – Дата доступа : 10.12.2012.

6. Cairo: 2D graphics library [Электронный ресурс]. – Режим доступа : http://cairographics.org/. – Дата доступа : 10.12.2012.

7. GD Graphics Library [Электронный ресурс]. – Режим доступа : http://ru.wikipedia.org/wiki/GD_Graphics_Library. – Дата доступа : 10.12.2012.

8. ImageMagick [Электронный ресурс]. – Режим доступа : http://www.imagemagick.org/script/index.php. – Дата доступа : 10.12.2012.

9. JpGraph: an Object-Oriented Graph creating library for PHP [Электронный ресурс]. – Режим доступа : http://jpgraph.net/. – Дата доступа : 10.12.2012.

10. pChart [Электронный ресурс]. – Режим доступа : http://www.pchart.net/. – Дата доступа : 10.12.2012.

11. Лялинский, А.А. Формирование высокоуровневых моделей цифровых ячеек с использованием веб-доступа // Проблемы разработки перспективных микро- и наноэлектронных систем – 2012 : сб. тр. / под общ. ред. акад. РАН А.Л. Стемпковского. – М. : ИППМ РАН, 2012. – С. 101–106.

12. Лялинский, А.А. Автоматизированное формирование тестов при характеризации цифровых ячеек с использованием веб-доступа / А.А. Лялинский // Проблемы разработки перспективных микро- и наноэлектронных систем – 2012 : сб. тр. / под общ. ред. акад. РАН А.Л. Стемпковского. – М. : ИППМ РАН, 2012. – С. 95–100.

13. Институт проблем проектирования в микроэлектронике РАН. Раздел Online-проектирования [Электронный ресурс]. – Режим доступа : http://www.ippm.ru/index.php?page=webcad. – Дата доступа : 10.12.2012.

Поступила 18.12.2012

Институт проблем проектирования
в микроэлектронике РАН,
Москва, Зеленоград,
ул. Советская, 3
e-mail: zelyal@inbox.ru

A.A. Lyalinsky

WEBSITE EXECUTION OF CAD MODULES

Web-based interface of modules that are part of the computer-aided design system in microelectronics is considered. The influence of several factors (available computer memory, maximum allowed run time, degree of homogeneity of the software) on the structure of the created system is investigated. Synchronous and asynchronous variants of interaction between control and executive parts are described. References on the systems that allow an access to applications in CAD microelectronics and are based on the principles discussed in this article are given.