МЕТОДОЛОГИЯ ПРОЕКТИРОВАНИЯ СИСТЕМ НА КРИСТАЛЛЕ

ОСНОВНЫЕ ПРИНЦИПЫ, МЕТОДЫ, ПРОГРАММНЫЕ СРЕДСТВА

Н.Евтушенко,
В.Немудров,
И.Сырцов

Статья открывает цикл публикаций по проблеме проектирования "систем на кристалле". Изложены характерные особенности СБИС типа "система на кристалле", сформулированы основные принципы новой методологии проектирования, представлена модель маршрута проектирования. Рассмотрен системный уровень проектирования: назначение, задачи, маршрут, программные средства.

ВВЕДЕНИЕ: ОТ ASIC К SoC

Интенсивный прогресс в области микроэлектроники за последние 10 лет привел к появлению принципиально нового класса СБИС – "систем на кристалле" (SoC – System-on-Chip). Это обусловлено, в первую очередь, существенным прогрессом технологии производства интегральных схем – степень интеграции достигает нескольких десятков миллионов вентилей на кристалле [1], а минимальные топологические размеры в ближайшее время составят 0,09 мкм. Очевидно, что для проектирования и тестирования таких СБИС необходимы принципиально новые подходы и средства проектирования.

Сегодняшние реалии таковы, что:

- в условиях рынка прибыль в значительной степени зависит от временных затрат на проектирование;
- такие технические параметры СБИС, как производительность, площадь кристалла и потребляемая мощность, являются ключевыми элементами в продвижении товара на рынок;
- увеличение степени интеграции делает задачу верификации качественно более сложной;
- из-за особенностей технологии глубокого субмикрона (DSM – Deep Submicron) все труднее удовлетворять всем требованиям по временным ограничениям (timing);
- команды разработчиков высокоинтегрированных СБИС обладают различными знаниями и опытом в области проектирования, и часто при выполнении проектов СБИС расположены в различных частях мира.

С другой стороны, при традиционном подходе к проектированию СБИС хороший дизайнер может работать со средней скоростью порядка 100 вентилей в день или 30 строк RTL-кода [2] (эти цифры остаются постоянными на протяжении последних пяти лет). В этом случае, чтобы спроектировать СБИС типа ASIC (Application Specific Integrated Circuit) сложностью 100 тыс. вентилей, потребуется 1000 человеко-дней, т.е. команда из пяти человек сможет разработать такую СБИС за год. Следовательно, для разработки сложной СБИС, содержащей порядка 10 млн. вентилей, в течение одного года потребуется команда из 500 человек, что, безусловно, неприемлемо с точки зрения себестоимости разработки.

По более точным аналитическим прогнозам (рис.1), при переходе на технологии 0,18 мкм и менее применение существующей методологии проектирования влечет увеличение трудоемкости проектов до 250 человеко-лет, что совершенно неприемлемо. Кроме того, в последнее время постоянно растет доля затрат на разработку программного обеспечения (ПО) РЭА (рис.2). Если вести разработку ПО и СБИС раздельно, то увеличивается вероятность выявления ошибок на этапах тестирования или эксплуатации всего комплекса аппаратуры, что приводит к дополнительным трудо- и времязатратам.

Выход из создавшейся ситуации очевиден – необходимо изменить методологию проектирования СБИС, и наиболее перспектив-

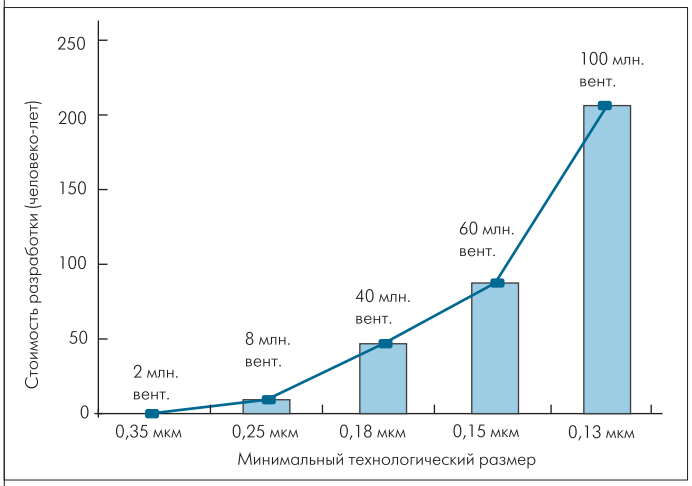

Рис.1. Зависимость стоимости разработки СБИС от технологии. По данным компании Cadence Design Systems

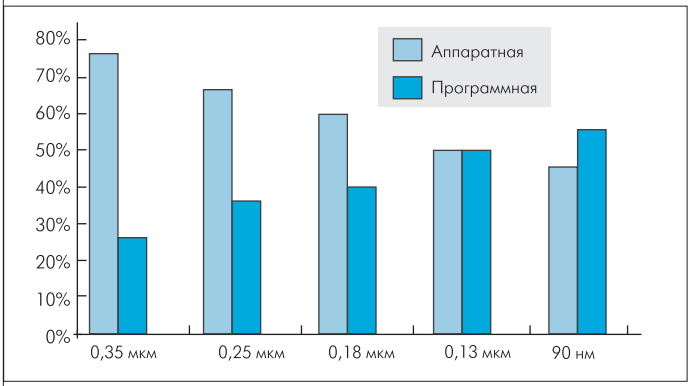

Рис.2. Отношение затрат на разработку программной и аппаратной части. По данным компании Cadence Design Systems

ным направлением представляется методология проектирования СБИС типа "система на кристалле".

ОСНОВНЫЕ ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ SoC

Сегодня нет четко детерминированного определения СБИС типа "система на кристалле", однако практика показывает, что SoC должна изготавливаться по технологии не ниже 0,35 мкм и содержать не менее 1 млн. вентилей. В самом общем виде, в состав SoC могут входить такие компоненты, как (рис.3):

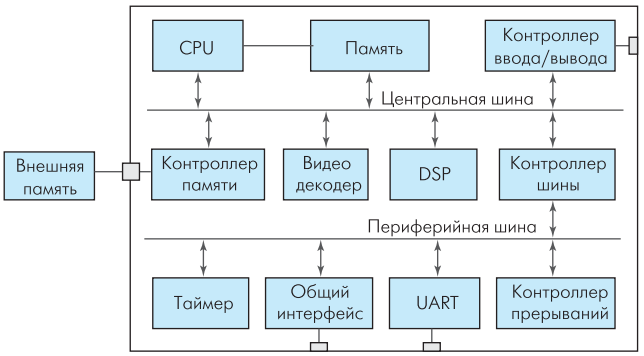

Рис.3. Пример структуры SoC

- микропроцессор (или микропроцессоры) и подсистема памяти (статической и/или динамической). Тип процессора может варьироваться от простейшего 8-разрядного до высокоскоростного 64-разрядного RISC-процессора;
- шины – центральная (высокоскоростная) и периферийная – для обмена данными между блоками;
- контроллер внешней памяти (например, DRAM, SRAM или Flash);
- контроллер ввода/вывода информации: PCI, Ethernet, USB и т.п.;
- видеодекодер, например MPEG2, AVI, ASF;
- таймер и контроллер прерываний;
- общий интерфейс ввода/вывода (например, для вывода на светодиодный индикатор информации о наличии питания);
- интерфейс UART (universal asynchronous receiver/transmitter) и т.п.

Столь сложная и разнообразная структура диктует особые требования к принципам проектирования SoC.

В основе методологии проектирования SoC лежит принцип повторного использования Intellectual Property блоков (IP-блоков), разрабатываемых целенаправленно или в рамках какого-либо проекта. По аналогии с системой на плате, где в качестве компонентов выступают готовые микросхемы, система на кристалле конструируется из повторно используемых блоков. IP-блоки могут быть двух типов: soft IP, описанные на RTL-уровне, и hard IP – на топологическом уровне. Иногда выделяют firm IP, в состав которых входят разные типы представлений – от RTL до списка цепей с планировкой субблоков.

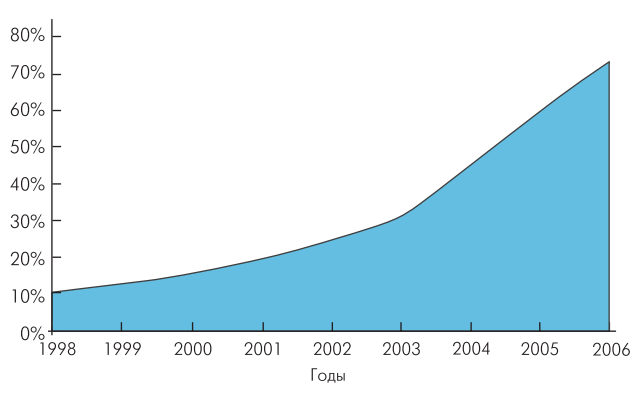

Рис.4. Рост доли цифроаналоговых систем в общем объеме SoC. По данным компании Cadence Design Systems

Отметим, что сегодня для обозначения повторно используемых (reuse) блоков наиболее часто используют термин IP-блок, т.е. блок, представляющий собой объект интеллектуальной собственности. Однако в ходу и другие термины: СФ-блок (в основном в пределах РФ), macro и core (обычно обозначает блоки типа центральный процессор CPU или процессор цифровой обработки сигнала DSP). Некоммерческая международная организация VSIA (Virtual Socket Interface Alliance) (www.vsi.org), основная задача которой – разработка нормативной документации по проблемам проектирования IP-блоков и SoC на их основе, использует собственное обозначение – VC (Virtual Component).

Одна из проблем проектирования SoC – создание IP-блоков. Практика показывает, что стоимость reuse-блока в среднем в 10 раз превышает стоимость аналогичного однократно используемого блока, а для процессоров эта величина на порядок выше. По этой причине reuse-блоки предназначены, как правило, для решения общих, логически формализуемых задач, например, MPEG-кодер, CPU, DSP, USB- и PCI -интерфейсы и т.п. При выборе IP-блоков учитывается стоимость готового блока и оцениваются затраты на разработку собственно блока.

Другая принципиальная особенность SoC – это наличие программируемых блоков (процессоров). Поэтому SoC – не просто интегральная схема (ИС), а комплекс, в состав которого входят как аппаратная часть (собственно кристалл), так и программная – встраиваемое программное обеспечение (ПО). Следовательно, маршрут проектирования SoC должен содержать операции по совместной верификации и отладке программной и аппаратной частей.

Еще одна особенность SoC – устойчивый рост доли смешанных цифроаналоговых систем в общем объеме SoC (рис. 4), поэтому в маршрут проектирования должны быть включены этапы по совместной разработке и верификации цифровой и аналоговой частей SoC.

МАРШРУТЫ ПРОЕКТИРОВАНИЯ ASIC И SoC

Традиционный маршрут проектирования ASIC (рис.5) начинается с разработки спецификации на проектируемую ASIC. Для сложных СБИС (например, устройства обработки графической информации), в состав спецификации включается алгоритм обработки информации, который затем используется разработчиками для написания RTL-кода.

После функциональной верификации происходит синтез СБИС на вентильном уровне в виде списка цепей. Здесь же выполняется верификация временных требований. Как только временные требования удовлетворены, список цепей передается на физический синтез: размещение элементов и трассировка цепей. В конце создается и тестируется физический прототип СБИС, на основе которого впоследствии выполняется системная интеграция и тестируется ПО.

Используя такой маршрут, можно проектировать схемы сложностью не более 100 тыс. вентилей по технологии не ниже 0,5 мкм. Это связано с тем, что проект выполняется по нисходящему принципу и не происходит возврат на предыдущие фазы. Например, RTL-дизайнер не может прийти к системному разработчику и сказать, что его алгоритм нереализуем, или группа логического синтеза не может попросить изменить RTL-код, чтобы добиться необходимых временных параметров. Для схем, изготавливаемых по тех-

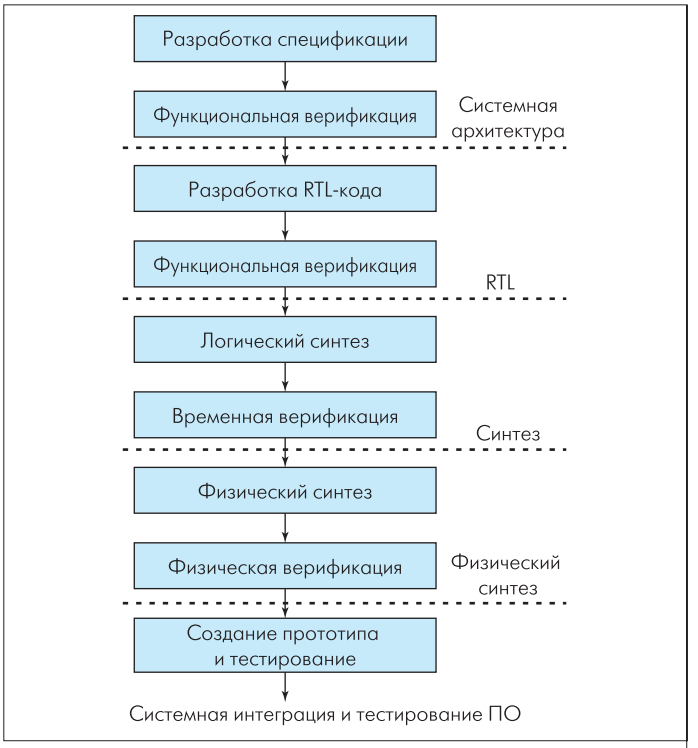

Рис.5. Маршрут проектирования ASIC

нологиям глубокого субмикрона, такой маршрут вообще не приемлем, поскольку особенности физической реализации должны учитываться уже на логическом уровне проектирования.

Многие зарубежные фирмы переходят от традиционной нисходящей модели проектирования к новой спиралевидной модели (рис.6) [2]. Здесь проектирование выполняется одновременно по четырем направлениям: разработка ПО, разработка RTL-кода, логический и физический синтез. При этом в процессе работы группы разработчиков обмениваются результатами проектирования. Новая модель характеризуется такими полезными свойствами, как параллельная разработка аппаратного обеспечения (АО) и ПО, параллельная верификация и логический синтез блоков, планировка, размещение и трассировка включены в процесс синтеза, разрешен возврат на предыдущие фазы проектирования и корректировка результатов.

СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ SoC

Фактически весь процесс разработки SoC делится на четыре этапа: разработка архитектуры SoC на системном уровне, выбор имеющихся IP-блоков из базы данных (внутри фирмы, других фирм или поставщиков IP-блоков), проектирование оставшихся блоков и интеграция всех блоков на кристалле. Остановимся на системном уровне.

Начальный этап проектирования заключается в рекурсивной разработке, верификации и уточнении набора спецификаций до такой степени детализации, чтобы на их основе можно было начать создавать RTL-код. Быстрая разработка четких, полных и последовательных спецификаций – сложная, трудоемкая и ответственная задача. Если четко знать, что надо построить, то ошибки при дальнейшей реализации могут быть быстро обнаружены и устранены. В противном случае можно не выявить ошибку на про-

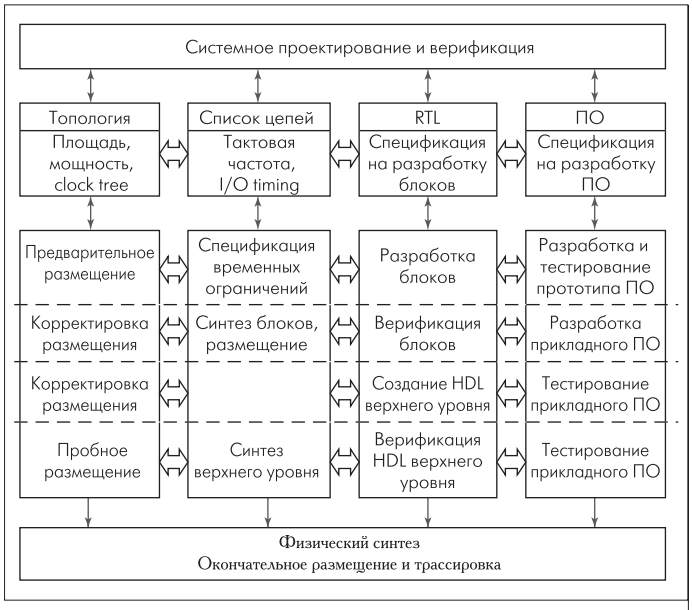

Рис.6. Спиралевидная модель процесса проектирования

тяжении всего цикла проектирования вплоть до изготовления чипа.

Спецификации описывают поведение системы [2], точнее – как управлять системой, чтобы добиться от нее нужного поведения. В этом смысле понятие спецификации в значительной мере связано с понятием интерфейса. Функциональная спецификация описывает интерфейс системы или блока так, как его видит внешний пользователь. Она содержит информацию о контактах, шинах, регистрах и о том, как с ними обращаться. Архитектурная спецификация описывает взаимодействия между частями блока и поведение на системном уровне.

Спецификации разрабатываются как для аппаратной, так и для программной части проекта. Для аппаратной части спецификации должны включать список выполняемых функций, внешний интерфейс к другим блокам (контакты, шины, протоколы), интерфейс с ПО (регистры), временные параметры, быстродействие, особенности физического уровня (площадь кристалла, потребляемая мощность). Для программной части в спецификации необходимо описать выполняемые функции, временные параметры, быстродействие, интерфейс к аппаратной части, структуру и ядро.

Традиционно спецификации пишутся на естественных языках (русский, английский...), что вносит в них неопределенность и ошибки. Чтобы избавиться от этих проблем, многие компании начинают использовать исполняемые спецификации, описанные на языках программирования высокого уровня C, C++ или вариации C++, например, SystemC (www.systemc.org) [3]. Для описания аппаратуры обычно используют языки VHDL или Verilog. Разработка исполняемых моделей позволяет верифицировать основные выполняемые функции и интерфейсы на ранних стадиях проектирования, задолго до детализации проекта.

Процесс проектирования SoC на системном уровне (рис.7) начинается с формулирования целей и задач, выполняемых SoC. На начальном этапе следует определить основные эксплуатационно-технические свойства: требуемое быстродействие, допустимую потребляемую мощность и т.п. На основании этих свойств создается системная спецификация, которая может выступать частью технического задания на разработку системы. Как правило, этот этап не требует средств САПР.

Затем создается высокоуровневая поведенческая модель всей разрабатываемой системы. Она, как правило, строится в виде блок-схемы. Для верификации разработанной поведенческой модели создается тестовое окружение системы (Testbench), которое включает в себя генераторы входных сигналов, тестовые последовательности и блоки отображения выходной информации. Тестовое окружение должно максимально полно проверять работу системы. Впоследствии на основе этого тестового окружения будут разрабатываться тестовые векторы для верификации проекта на нижних уровнях проектирования и для тестирования опытных образцов СБИС.

Поведенческая модель верифицируется путем компьютерного моделирования. Если в процессе верификации обнаруживаются какие-либо отклонения от требований системной спецификации, то модель корректируется и моделирование повторяется. Кроме верификации, на данном шаге можно выбрать оптимальные параметры алгоритма системы. Например, разработчик может найти компромисс между вычислительной сложностью и точностью.

Поскольку SoC включает одно или несколько программируемых процессорных ядер, на следующем этапе разработчик должен принять решение о том, какие части поведенческой модели будут впоследствии реализованы на аппаратном уровне, а какие – на программном в виде встроенного в СБИС программного обеспечения. Также необходимо определить, каким образом будут взаимодействовать программная и аппаратная части, т.е. следует разработать интерфейс между АО и ПО. Здесь же определяется общая архитектура SoC: тип процессора, тип памяти и ее объем, аппаратные блоки, интерфейс АО-ПО, тип используемой шины, описание программной части и т.п. В итоге формируется набор спецификаций на разработку программного обеспечения и на разработку каждого аппаратно реализуемого блока.

В методологии проектирования ASIC программно-аппаратная верификация выполняется только после изготовления опытного образца. Возникающие при этом ошибки можно исправить лишь соответствующими изменениями ПО, не переделывая сам кристалл. Такой метод неприемлем для SoC, поскольку из-за большой сложности схемы исправить ошибки АО путем корректировки ПО очень трудно, а зачастую просто невозможно. Поэтому в маршрут

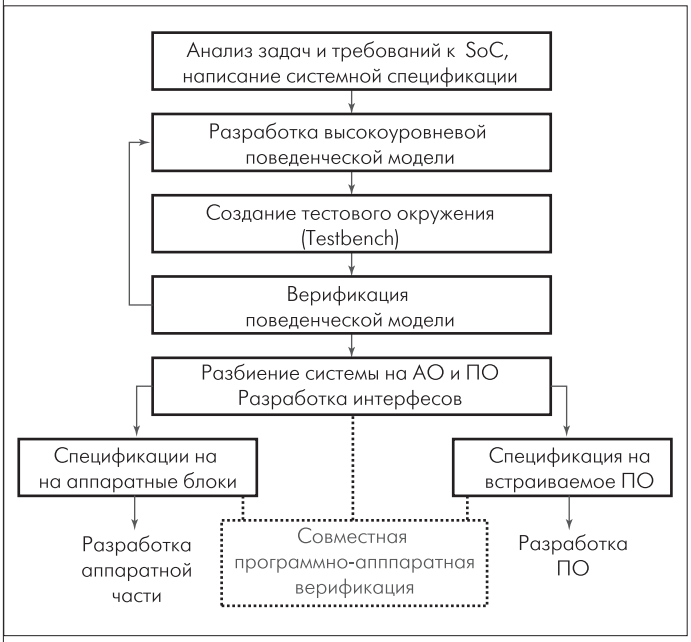

Рис.7. Маршрут системного проектирования SoC

проектирования SoC на разных уровнях вводится операция программно-аппаратной верификации. Наиболее ответственны в этом смысле этапы функционального и логического проектирования.

Программно-аппаратная верификация системного уровня сегодня не является обязательной операцией. Тем не менее, многие разработчики включают ее в маршрут проектирования SoC. В качестве аппаратной части здесь выступают исполняемые спецификации аппаратно реализуемых блоков, в качестве программной – прототип ПО. В результате можно удостовериться, что аппаратная часть, разрабатываемая в соответствии с имеющимися спецификациями, будет корректно работать под управлением встраиваемого ПО в режиме реального времени.

ПРОГРАММНЫЕ СРЕДСТВА САПР ДЛЯ СИСТЕМНОГО УРОВНЯ

До недавнего времени задачи системного уровня – разработка спецификаций – в большинстве случаев решались без специальных программных средств. При переходе к методологии проектирования SoC стала очевидной необходимость автоматизации процесса системного проектирования.

Во-первых, невозможно проверить работоспособность столь сложной системы, как SoC, используя только аналитические методы расчета – необходимо специальное прикладное ПО. Во-вторых, исполняемая спецификация должна быть представлена в определенном формате на языках C, C++, SystemC, Verilog или VHDL. Получить такое описание невозможно без соответствующих программных средств.

Кроме этого, при переходе к спиралевидной модели проектирования (см. рис.6) в процессе работы будут возникать постоянные возвраты с нижних уровней проектирования к верхним. Часто системный уровень сливается с функциональным уровнем проектирования (разработка RTL-кода), образуя системно-функциональный уровень. В этом случае удобно пользоваться едиными программно-техническими средствами. В мире создано множество программных средств, применимых для автоматизации системного проектирования. Эти средства можно классифицировать на пять групп.

Первая группа – средства разработки и отладки прикладного программного обеспечения. Достаточно популярным и удобным для представления спецификаций оказался язык программирования C и его модификация C++. Поэтому разработчики системного уровня активно используют такие средства разработки ПО, как Microsoft Visual Studio (www.microsoft.com); Inprise Borland C++ Builder (www.borland.com), а в ОС Linux – Borland Kylix C++ (www.borland.com). Иногда используют и специальные средства для анализа и автоматизации разработки ПО, например Rational Rose и язык UML (www.uml.ru). Основные достоинства этих систем – низкая стоимость, простота в освоении и использовании. Главный недостаток связан с отсутствием специализированных библиотек системного уровня, поэтому поведенческую модель разработчику приходится создавать практически "с нуля".

Вторая группа – средства математического моделирования. Наиболее типичный представитель этой группы – программный пакет MATLAB/Simulink (www.mathworks.com). Основные достоинства средств этой группы – низкая стоимость и простота в использовании. Кроме того, есть библиотеки моделей. Но объем специализированных библиотек, как правило, недостаточен и большинство моделей приходится создавать вручную, во внутренних форматах MATLAB (M-файлы, MEX-файлы). Последний недостаток, по всей видимости, скоро будет устранен, так как фирма The MathWorks планирует выпустить полноценный транслятор из формата M-файлов в формат C/C++.

В третью группу можно объединить средства моделирования общего назначения, например MLDesigner (www.mldesigner.com) или SES/Workbench (www.hyperformix.com). Их отличительная особенность в том, что они не привязаны к какому-то конкретному объекту проектирования. С их помощью можно моделировать и архитектуру СБИС, и, например, систему спутниковой связи или навигации. При сравнительно низкой цене они охватывают широкий спектр областей применения. Основной недостаток – отсутствие связи с функциональным и логическим уровнями проектирования.

Четвертая группа – самая многочисленная. К ней относятся узкоспециализированные программные средства, каждое из которых предназначено для решения какого-либо определенного круга проектных задач системного уровня. При этом общий спектр решаемых задач велик: от разработки программно-аппаратной архитектуры до интеграции процессорных ядер и разработки встраиваемого программного обеспечения. В качестве примера производителей данного ПО можно упомянуть такие фирмы, как Co-ware (www.coware.com), Mentor Graphics (www.mentor.com), Elanix (www.elanix.com), Summit Design (www.sd.com) и др. Специализированное ПО достаточно привлекательно с точки зрения экономии средств.

Пятая группа – это мощные интегрированные программные пакеты, при помощи которых разработчик способен выполнять весь цикл системного и функционального проектирования, а также весь цикл разработки СБИС, вплоть до физической реализации. На сегодняшний день в эту группу входит ПО только двух фирм: Synopsys (www.synopsys.com) – CoCentric System Studio, Design Ware, VCS, VCSi, Scirocco, SystemC HDL Co-Sim, CoCentric SystemC Compiler) [4] и Cadence Design Systems (www.cadence.com) – Incisive-SPW, Incisive unified simulator, Incisive-XLD, Incisive-AMS, NC-SystemC, NC-Verilog, NC-VHDL [5]. Главный недостаток обоих пакетов – их большая стоимость, что весьма существенно в условиях российского рынка.

При окончательном выборе программных средств и разработке на их основе маршрута проектирования необходимо учитывать целый ряд дополнительных факторов, например специфику разрабатываемых устройств, общий объем работ (число одновременно выполняемых проектов), имеющееся на предприятии ПО для функционального и логического уровней проектирования и т.п.

Продолжение следует.

ЛИТЕРАТУРА

1. **Henry Chang, Larry Cooke, Merril Hunt, Grant Martin, Andrew McNelly, Lee Todd.** Surviving the SOC Revolution// Kluwer Academic Publishers, Boston/Dordrech/London, 1999.
2. **Michael Keating, Pierre Bricaud.** Reuse Methodology Manual, Third Edition// Kluwer Academic Publishers, Boston/Dordrech/London, 2002.
3. **Thorsten Grottker, Stan Liao, Grant Martin, Stuart Swan.** System Design with SystemC// Kluwer Academic Publishers, Boston/Dordrech/London, 2002.
4. **Иванов А.** САПР фирмы Cadence. Общий обзор. – ЭЛЕКТРОНИКА: НТБ, 2003, №5.
5. **Кравченко В., Радченко Д.** САПР компании Synopsys. Основные средства и возможности. – ЭЛЕКТРОНИКА: НТБ, 2003, №5.