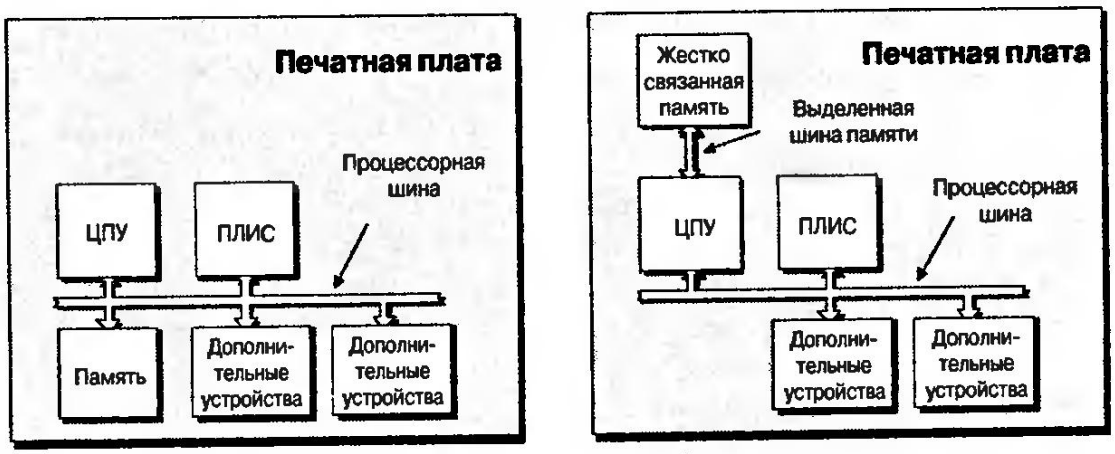
ГЛАВА 13

ПРОЕКТИРОВАНИЕ УСТРОЙСТВ СО ВСТРОЕННЫМИ МИКРОПРОЦЕССОРАМИ

Введение

В этой книге рассматриваются только электронные системы, которые на ***печатной плате*** содержат одну или несколько ПЛИС. Огромное количество таких систем используют также микропроцессоры общего назначения (МП) для формирования различных приложений управления и обработки данных[1]. Их часто называют ***центральными процессорами (ЦП)*** или ***микропроцессорными устройствами (МПУ)***.

До недавнего времени микропроцессоры и периферийные устройства обычно реализовывались в виде отдельных микросхем на печатной плате. Существует почти бесконечное множество возможных вариантов соединения этих элементов, но в двух основных из них микропроцессор соединён с памятью (**Рис. 13.1**).



а) Память соединена с процесором через общую шину

б) Жестко связанная память соединена с процессором через выделенную шину

Рис. 13.1. Два варианта расположения элементов на уровне печатной платы

В вариантах, показанных на **Рис. 13.1**, микропроцессор соединён с ПЛИС и другими устройствами через шину общего назначения. Под «дополнительными устройствами» преимущественно подразумеваются периферийные устройства, такие как таймеры, контроллеры прерываний, устройства связи и другие.

В некоторых случаях блок памяти будет соединён также с процессором с помощью главной процессорной шины, как показано на **Рис. 13.1, а.** На самом деле это соединение будет выполняться через специальный периферийный блок, называемый контроллером памяти, но он для простоты изложения не показан на этом рисунке. Кроме того, память может соединяться с процессором напрямую с помощью выделенной шины памяти, как показано на **Рис. 13.1, б.**

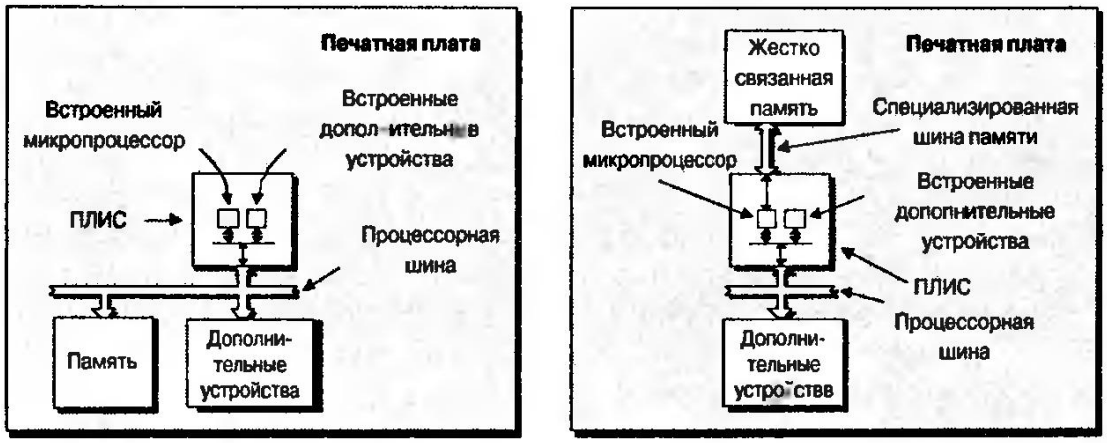
---

[1] Также можно использовать микроконтроллеры, которые сочетают в себе микропроцессорное ядро с некоторой периферией и специализированными входами и выходами.

1910 г. *Разработана первая электрическая стиральная машина.*

Суть вопроса состоит в том, что представление микропроцессора и его различных периферийных устройств в виде выделенной микросхем на печатной плате требует финансовых затрат и места на печатной плате. Кроме того, это влияет на надежность платы, поскольку каждое паяное соединение, т. е. точка подсоединения, является потенциальным сбойным механизмом.

Альтернативное решение заключается во встраивании микропроцессора вместе с некоторой его периферией внутрь ПЛИС[1] (**Рис. 13.2**).



а) Память соединена с процесором через общую шину

б) Память с жесткой связью соединена с процессором через специализированную шину

Рис. 13.2. Два варианта расположения элементов на уровне ПЛИС

Довольно часто, если микропроцессор использует сравнительно небольшое количество памяти, и она тоже включается в состав ПЛИС. Во время написания этой книги очень редко вся память микропроцессора встраивалась внутрь ПЛИС.

Создание ПЛИС такого типа сразу породило огромное множество новых проблем. Во-первых, разработчики системы должны были решить, какие функции должны быть реализованы программно, т. е. в виде инструкций для выполнения микропроцессором, а какие подлежат аппаратной реализации (используя главную структуру ПЛИС). Во-вторых, среда разработки должна была поддерживать концепцию совместной проверки, при которой аппаратная и встроенная программная части устройства должны подвергаться совместной верификации, чтобы убедиться, что всё работает как следует.

Кроме микропроцессоров, каждый поставщик ПЛИС поддерживает соответствующую процессорную шину. Например, компании Altera и Quicklogic поддерживают шину *AMBA* компании ARM (открытая спецификация, которая может быть бесплатно загружена с сайта www.arm.com для последующих модификаций). Xilinx для своих ядер использует шину *CoreConnect* компании IBM. CoreConnect существует в двух вариантах. Главный из них представляет собой 64-битную шину под названием *PLB (processor local bus — локальная процессорная шина)*. Она может использоваться вместе с одной или несколькими 32-битными шинами *OPB (on-chip peripheral busses — внутрикристальная периферийная шина)*.

---

[1] Ещё одно возможное решение может заключаться во встраивании микропроцессорного ядра в заказную микросхему ASIC, но это уже другая песня!

1910 г. *Франция. Джордж Клод (George Claude) изобрёл неоновую лампу.*

Аппаратные и программные ядра

Аппаратные ядра

Аппаратные микропроцессорные ядра представляют собой специализированные, предопределённые, т. е. аппаратно-реализованные, блоки, которые представляют собой ядра, доступные только в определённых семействах устройств. Все ведущие поставщики предпочитают реализовывать в своих устройствах некоторые определённые типы процессоров. Например, Altera предлагает встроенные ARM процессоры, QuickLogic предпочитает решения на основе MIPS-процессоров, а Xilinx встраивает ядра PowerPC.

Конечно, каждый поставщик с превеликим удовольствием пустится в пространные рассуждения о том, почему именно эта реализация превосходит любые другие. Проблема принятия решения, какая из реализаций будет действительно лучше других, довольно многогранна, и осложняется только тем, что разные процессоры лучше подходят для решения разных задач.

Как обсуждалось в гл. 4, существует два главных подхода к интеграции таких ядер внутри ПЛИС. Один из них предполагает расположение их в виде отдельной полосы вдоль стороны главной ПЛИС-структуры (**Рис. 13.3**). При таком подходе все компоненты обычно формируются на одном кремниевом кристалле, хотя могут быть реализованы и на двух кристаллах и упакованы в виде *многокристального модуля (МКМ)*.

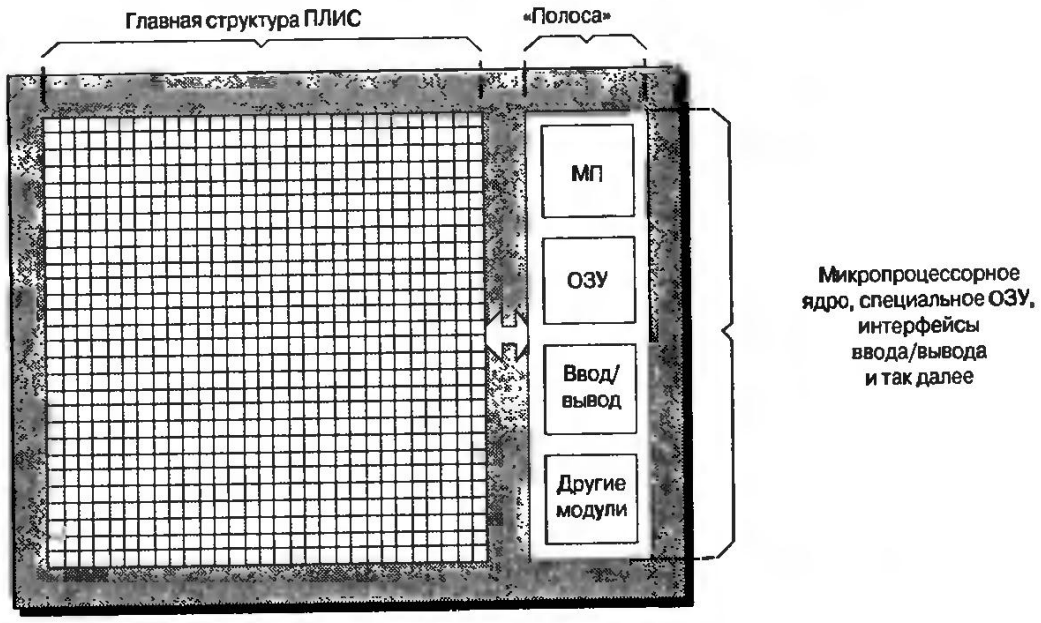
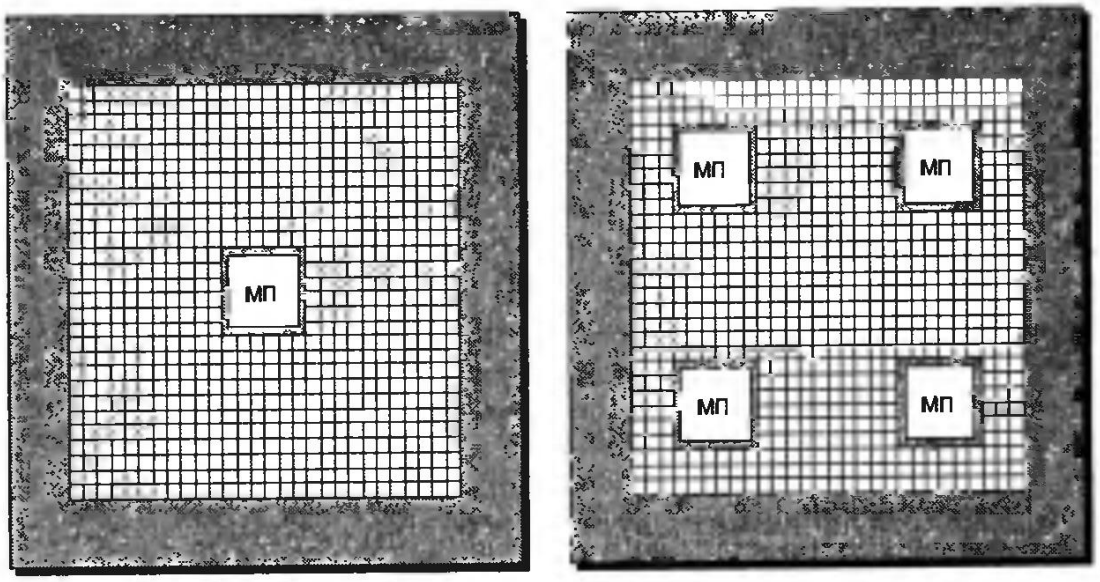


Рис. 13.3. Вид на кристалл со встроенным микропроцессорным ядром за пределами главной структуры

Одно из преимуществ такой реализации заключается в том, что главная структура ПЛИС получается идентичной для устройств как со встроенным микропроцессорным ядром, так и без него, что может упростить средства проектирования, используемые разработчиками. Другое преимущество заключается в том, что поставщики ПЛИС могут связать все дополнительные функции, такие как память, устройства ввода/вывода и другие, в одну полосу для дополнения микропроцессорного ядра[1)].

[1)] Этот метод используется такими поставщиками, как Altera (www.altera.com) и QuickLogic (www.quicklogic.com).

Альтернативным решением является встраивание одного или нескольких микропроцессорных ядер прямо в главную структуру ПЛИС. Во время написания этих строк существовали микросхемы с одним, двумя и даже четырьмя ядрами (**Рис. 13.4**).



а) Одно встроенное ядро *б)* Четыре встроенных ядра

Рис. 13.4. Вид на кристаллы с микропроцессорными ядрами встроенными внутрь главной структуры

В этом случае средства проектирования должны учитывать присутствие микропроцессоров в структуре микросхемы. Память, используемая ядром, формируется из встроенных блоков ОЗУ, и любые функции сопряжения реализуются с помощью групп программируемых логических блоков общего назначения. Сторонники этой схемы утверждают, что размещение микропроцессорного ядра в непосредственной близости к главной структуре ПЛИС дает ей преимущество в скорости[1].

Программные микропроцессорные ядра

Кроме физического встраивания микропроцессора в структуру кристалла, можно сконфигурировать группу программируемых логических блоков для работы в качестве микропроцессора. Такую группу блоков обычно называют *программным ядром*, но более точно они могут быть классифицированы как *программные (soft)* и *микропрограммные (firm)* в зависимости от способа реализации функциональности микропроцессора с помощью логических блоков. Например, если ядро реализовано в форме таблицы соединений RTL, которая будет синтезирована вместе с другой логикой, это будет действительно программная реализация. Если же ядро реализовано в виде размещенных и разведенных конфигурируемых логических блоков (КЛБ) или таблиц соответствия, такая реализация обычно считается микропрограммной.

В обоих случаях все периферийные устройства, такие как таймеры, контроллеры прерываний, контроллеры памяти, функции связи и другие, также реализуются в виде программных или микропрограммных

---

[1] Этот метод использует компания Xilinx (www.xilinx.com), которая также реализует множественные функции сопряжения в виде программных блоков интеллектуальной собственности.

ядер. Причем поставщики ПЛИС обычно предлагают большие библиотеки таких ядер.

Программные ядра медленнее и проще, чем их аппаратные аналоги, хотя, они невероятно быстрые в человеческом понимании. Кроме невысокой цены у них есть одно преимущество, которое заключается в том, что при необходимости можно реализовывать ядро или ядра до тех пор, пока не будут исчерпаны все ресурсы в виде программируемых логических блоков.

Ядра Nios основаны на архитектуре SPARC, использующей концепцию регистровых окон, а MicroBlaze построены на классической RISC-архитектуре.

Как и прежде, все ведущие поставщики ПЛИС предпочитают реализовывать определённые типы процессоров в своих программных ядрах. Например, Altera предлагает ядра Nios, а Xilinx предпочитает MicroBlaze. Ядра Nios существуют в 16-битных и 32-битных вариантах архитектуры, которые соответственно работают с 16-битными и 32-битными блоками данных. Оба варианта используют одну и ту же систему 16-битных команд. В отличие от них, MicroBlaze является полностью 32-битным процессором, т. е. он поддерживает 32-битные команды и работает с 32-битными блоками данных. Снова повторюсь, что каждый поставщик с большой радостью поведает о том, почему его программное ядро является лучшим и почему конкурирующие предложения обречены на провал. Сожалею, но делать выбор вам, читатель, придется самостоятельно.

Хочу вас обрадовать: *интегрированная среда разработки (IDE — integrated development environment)* компании Xilinx одинаково воспринимает аппаратные ядра PowerPC и программные ядра MicroBlaze. Эта среда включает в себя оба процессора, которые основаны на одной и той же процессорной шине CoreConnect, и использует для сопряжения общие программные ядра интеллектуальной собственности. Это позволяет относительно легко перейти от одного микропроцессора к другому.

Интерес представляет и тот факт, что Xilinx предлагает небольшие 8-битные программные ядра, называемые PicoBlaze, которые могут быть реализованы с помощью всего лишь (примерно) 150 логических ячеек. В отличие от них MicroBlaze использует примерно 1000 логических ячеек. Это[1] вполне приемлемо для реализации 32-битного процессора, особенно когда кое-кто использует ПЛИС, содержащие 70000[2] или более таких ячеек.

Особый интерес представляет программное ядро LisaTek компании CoWare Inc. (www.coware.com). С помощью специального языка в этом ядре можно определить требуемый набор команд и микроархитектуру (ресурсы, конвейеры, тактовые частоты). Затем LisaTek по этим определениям генерирует требуемый RTL-код программного ядра вместе с соответствующими программными средствами, такими как компилятор языка C, ассемблер, компоновщик и система моделирования.

Компания QuickLogic предлагает 9-битные программные микроконтроллеры, которые обозначаются как Q90C1xx (9-битные слова данных могут быть полезны для некоторых функций в телекоммуникации).

---

[1] В рамках рассматриваемого материала предположим, что логическая ячейка содержит 4-входовую таблицу соответствия, регистр и другие блоки, такие как мультиплексоры и цепи быстрого переноса.

[2] Эти 70000 соответствовали истине сегодня утром, когда я завтракал, но это значение, несомненно, увеличится к тому времени, когда вы будете читать эту книгу.

Разделение устройства на аппаратные и программные компоненты

Как уже упоминалось в гл. 4, почти все части электронного устройства могут быть реализованы аппаратно (с использованием логических вентилей, регистров и прочим) или программно (в виде инструкций микропроцессора)[1]. Одним из главных частных критериев выбора между аппаратной и программной реализацией функции является время, за которое эта функция должна выполнять такие задачи, как:

- *Пикосекундная и наносекундная логика* — эта логика должна работать безумно быстро и реализовываться аппаратно в структурах ПЛИС.
- *Микросекундная логика* — умеренно быстрая логика. Может выполняться как аппаратно, так и программно. Тип логики, при котором основная масса времени уходит на определение того, каким путём пойти.
- *Миллисекундная логика* — эта логика используется при реализации интерфейсов, таких как опрос состояния переключателей или зажигание светодиодов. Основные усилия будут направлены на замедление аппаратной части при реализации таких функций; например, используя громаднейшие счетчики для генерации задержек. Часто такие задачи лучше реализовывать в микропроцессорном коде, поскольку процессор позволяет получить низкую скорость (в сравнении с аппаратной реализацией), но следует учитывать тот факт, что задача может оказаться очень сложной.

Вся хитрость состоит в том, чтобы решить все проблемы наиболее экономичным способом. Определённые функции должны реализовываться аппаратно, другие должны быть выполнены программно, а некоторые их них могут быть реализованы и тем, и другим способом в зависимости от предположений, как лучше использовать доступные ресурсы (ресурсы на кристалле и работу инженеров-разработчиков и программистов).

Давайте рассмотрим «идеальную» среду *разработки системного уровня*, в которой архитекторы системы вначале описывают устройство с помощью графического интерфейса в виде совокупности соединённых вместе функциональных блоков. Каждый из этих блоков может быть описан на системном (алгоритмическом) уровне, например, с помощью SystemC. После этого все устройство целиком может быть проверено на функциональность, прежде чем будет принято решение, какую часть следует реализовывать аппаратным, а какую программным путём.

Когда дойдёт очередь до разделения устройства, можем представить себе, что существует возможность выделять каждый логический блок мышкой и выбирать для него опцию аппаратной или программной реализации. После того как эта операция будет проведена со всеми блоками, можно было бы нажать кнопку **Пуск**, и среда разработки позаботилась бы о синтезе аппаратного обеспечения, компиляции программного кода и произвела бы сборку всей конструкции.

---

[1] Некоторые циники могут сказать, что части устройства, которые изначально всем понятны и не будут подвержены изменению, как правило, реализуются аппаратно, ну, а те части, в отношении которых на момент начала проектирования системы есть какие-то сомнения, наверняка, будут реализованы программно, так как программные модули могут быть «настроены в последнюю минуту».

А теперь с оглушительным треском возвращаемся в реальный мир. На самом деле ряд перспективных средств разработки подают некоторые надежды, а новые инструменты и технологии появляются почти ежедневно. Однако во время написания этой книги системные инженеры обычно разделяли устройство на аппаратные и программные части вручную, затем передавали эти высокоуровневые функции соответствующим инженерам и надеялись на лучшее.

Что касается программной части устройства, здесь всё выглядит немного проще, примерно как конечный автомат, используемый для управления интерфейсом оператора (чтение состояния переключателей и управление устройством отображения). Хотя конечный автомат сам по себе может быть довольно сложным, но подобный уровень программного обеспечения далёк от ракетной техники. Вместе с тем, эта часть устройства может иметь невероятно сложные программные требования и должна содержать:

Системы реального времени работают по принципу не только **что** выполнить, но и **когда** это должно быть выполнено.

- подпрограммы инициализации системы и уровень аппаратной абстракции (HAL — hardware abstraction layer);
- набор тестов аппаратной диагностики;
- *операционную систему реального времени (ОС РВ)*;
- драйверы устройств ОС РВ;
- другие встроенные приложения.

Этот код обычно описывается с помощью языка C/C++ и затем компилируется в машинный код, который будет выполняться микропроцессорным ядром. В экстремальных ситуациях можно попытаться выжать последнюю каплю производительности устройства, написав вручную точные подпрограммы на языке Ассемблера.

А в это время разработчики аппаратуры описывают свою часть устройства обычно на RTL-уровне абстракции, используя язык VHDL или Verilog, или SystemVerilog.

Современное проектирование систем является настолько сложным, что их аппаратные и программные части следует проверять совместно. К сожалению, выбор среди множества альтернативных средств совместной проверки и их сложность могут расстроить и вывести из себя даже взрослого, в том числе и меня.

Аппаратное и программное мировоззрение

Одна из самых больших проблем, с которой приходится сталкиваться при совместной проверке аппаратных и программных частей устройства, заключается в совершенно различном мировоззрении их разработчиков.

Разработчики аппаратуры обычно представляют свои части устройства в виде RTL-блоков, которые состоят из таких компонентов, как регистры, логические функции и проводники, соединяющих всё вместе. При отладке своих частей разработчики аппаратуры мыслят категориями редактора исходного кода, системы моделирования, и графического дисплея, отображающего форму сигналов и изменение переменных с привязкой ко времени. В типичной среде разработки аппаратного обеспечения щелчок мышкой при определённом событии на графическом дисплее автоматически укажет соответствующую строку RTL-кода, которая вызвала это событие.

В отличие от разработчиков аппаратуры программисты оперируют понятиями исходного кода языка C/C++, регистров процессора (и внешних устройств) и содержимым различных участков памяти. При отладке программы они часто предпочитают проходить через код в пошаговом режиме и отслеживать при этом изменения в различных

регистрах. Они, возможно, также захотят установить одну или несколько точек останова, которые представляют собой метки на специфичных участках кода, запустить программу и ждать её выполнения до тех пор, пока она не достигнет этих точек, и затем в режиме паузы посмотреть текущие результаты моделирования. Кроме того, программисты могут задать определённые состояния, например, когда регистр будет содержать определённое значение, затем запустить программу и ожидать этого состояния, и затем в режиме паузы просмотреть что произошло во время выполнения.

Когда программисты пишут код приложений, например компьютерной игры, они определённо уверены в том, что аппаратная часть, скажем домашний компьютер, исправна и устойчива к ошибкам. Однако всё обстоит по-иному, когда речь идет об инженерах-программистах, создающих встраиваемые приложения, которые предназначенные для работы на аппаратном обеспечении, разрабатываемом в это же время. Когда обнаруживается какая-то проблема, определить, в чьей зоне ответственности, аппаратной или программной, находится ошибка, может оказаться чрезвычайно сложно. Классической шуткой стал разговор между двумя представителями:

Инженер-программист: «Я думаю, что обнаружил аппаратную проблему во время работы моего встроенного приложения».

Разработчик аппаратуры: «В какой момент произошла ошибка? Можете ли вы дать мне контрольный пример, который поможет локализовать проблему?»

Инженер-программист: «Ошибка произошла сегодня утром в 9:30, контрольным примером может служить моё приложение».

В современных средствах совместной проверки аппаратный и программный миры тесно связаны. Это значит, что если программисты обнаруживают потенциальный аппаратный сбой, они определяют выполняемую строку кода, которая непосредственно укажет разработчикам аппаратуры на соответствующий моделированию интервал времени на графическом дисплее. Аналогично, если разработчики аппаратной части обнаруживают потенциальную программную ошибку (например, код, запрашивающий недопустимую аппаратную транзакцию), они могут использовать свой интерфейс для указания программистам соответствующей строки исходного кода. К сожалению, подобные средства разработки могут оказаться весьма дорогими, поэтому иногда приходится выбирать менее сложные решения.

ПЛИС как среда проектирования

Возможно, проще всего начать разговор на эту тему с описания областей, в которых ПЛИС применяются в качестве средства разработки. Идея заключается в том, что ПЛИС на основе ячеек статического ОЗУ со встроенным процессором (аппаратным или программным) располагается на макетной плате, которая соединена с компьютером пользователем. Кроме ПЛИС, на этой плате также установлена память, в которой будут храниться программы, выполняемые встроенным микропроцессором (**Рис. 13.5**).

После того как системные инженеры (архитекторы) определили, какие части устройства должны быть реализованы аппаратно, а какие программно, разработчик аппаратуры начинают описывать свои RTL-блоки и функции и синтезировать из них таблицу соединений КЛБ/таблиц соответствия. Тем временем программисты начинают создавать на языке C/C++ программы и подпрограммы и компилируют их в машинный код. В итоге таблица соединений загружается в ПЛИС

1911 г. *Датский физик Хейке Камерлинг-Оннс (Heike Kamerlingh Onnes) открыл явление сверхпроводимости.*

1912 г. *Америка. Сидней Рассел (Dr. Sidney Russell) изобрёл одеяло с электрообогревом.*

1912 г. *Радиоприёмники начали оснащать обратными связями и строить по гетеродинной схеме.*

1912 г. *«Титаник» во время первого плавания послал сигнал бедствия после столкновения с айсбергом.*

Рис. 13.5. Использование ПЛИС в качестве среды проектирования

через конфигурационный файл, а исполняемый модуль машинного кода загружается в память, и система готова к «запуску» в свободное плавание (**Рис. 13.6**).

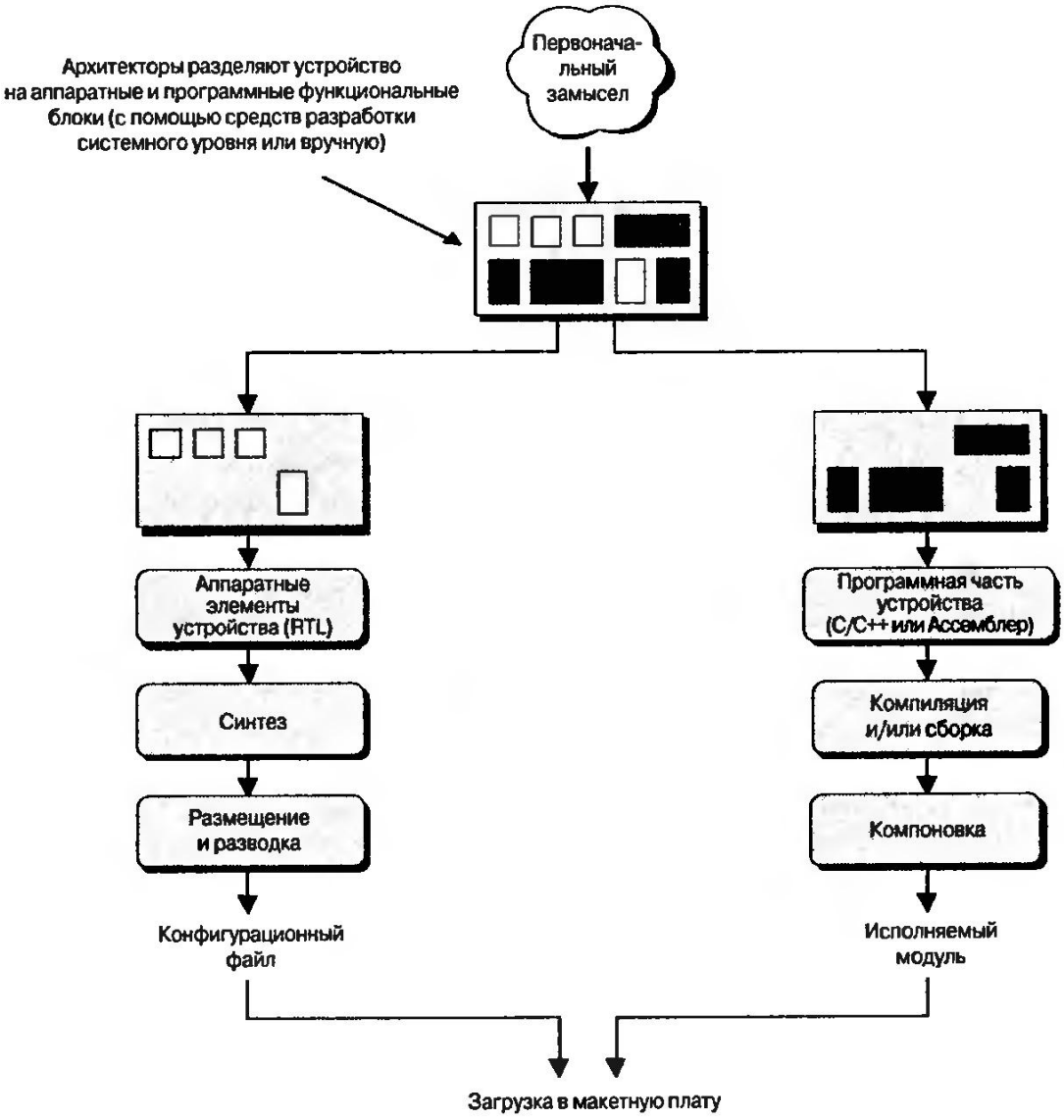


Рис. 13.6. Очень простой метод проектирования

Через конфигурационный файл во встроенные в ПЛИС блоки ОЗУ может быть также загружен любой машинный код.

Контроль состояния устройства

Главная проблема в сценарии, описанном в прошлом разделе, заключается в плохой наглядности того, что происходит в аппаратной части устройства. Один из способов решения этой проблемы заключа-

ется в использовании виртуального логического анализатора, который позволяет наблюдать за событиями внутри аппаратной части конструкции (этот способ более детально рассматривается в гл. 16).

Ситуация осложняется при попытке определить, что происходит в программной части. Но при этом следует иметь в виду, что встроенное микропроцессорное ядро содержит свою собственную специализированную JTAG цепочку последовательного сканирования, как было показано в гл. 5 (**Рис. 13.7**).

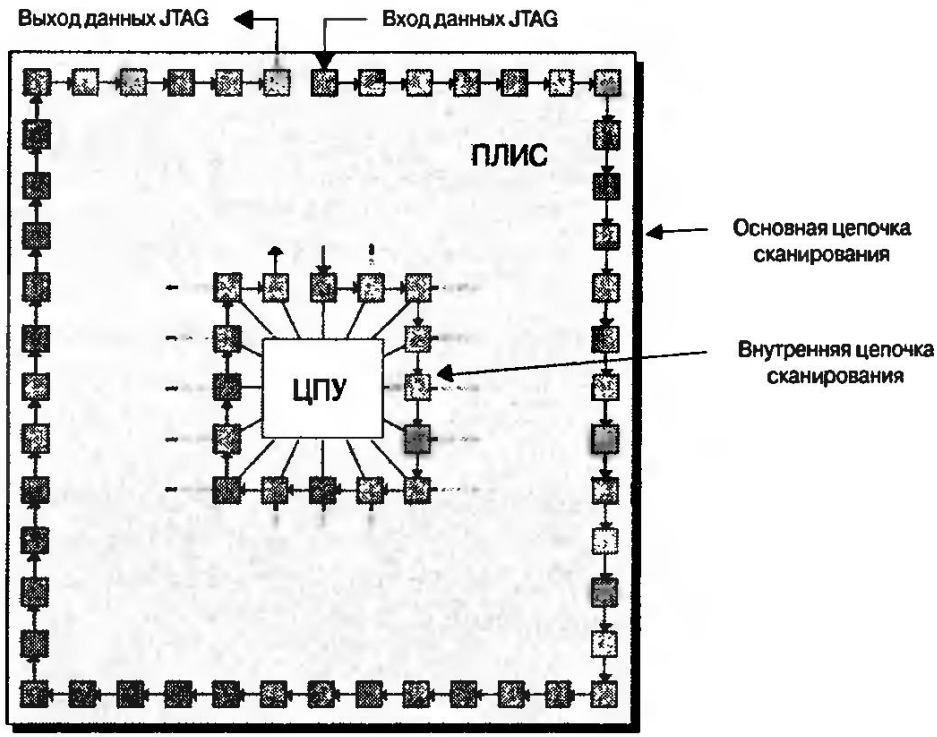


Рис. 13.7. Цепочка последовательного сканирования ПЛИС со встроенным процессором

Это справедливо как для аппаратных ядер, так и для большинства сложных программных ядер. В этом случае средства совместной проверки могут использовать цепочки сканирования для наблюдения за активностью шин и сигналов управления, соединяющих микропроцессор с остальной частью системы. Через порт JTAG также могут быть доступны внутренние регистры микропроцессора, что позволяет внешнему отладчику управлять устройством, реализовывать пошаговый режим, устанавливать точки останова и так далее.

Некоторые альтернативные методы совместной проверки

Один из способов получить представление о том, что же в действительности происходит в аппаратной части устройства — использовать логическое моделирование. В этом случае большинство систем должны быть сконструированы и промоделированы средствами языка VHDL, Verilog и SystemVerilog на RTL-уровне абстракции. Однако микропроцессорное ядро может быть представлено несколькими способами (**Рис. 13.8**).

Независимо от типа модели, используемой для представления микропроцессора, встроенные программные части устройства (машинный код) будут загружены в какую-либо память — во встроенную память внутри ПЛИС или во внешнюю память системы — после чего микропроцессор сможет выполнить эти машинные команды.

1913 г. *Вильям Кулидж (William D. Coolidge) изобрёл рентгеновскую трубку с термокатодом из тонкой вольфрамовой спирали. Трубка Кулиджа стала стандартным генератором рентгеновских лучей для медицины.*

1914 г. *Америка. В Кливленде, шт. Огайо, впервые установлен светофор.*

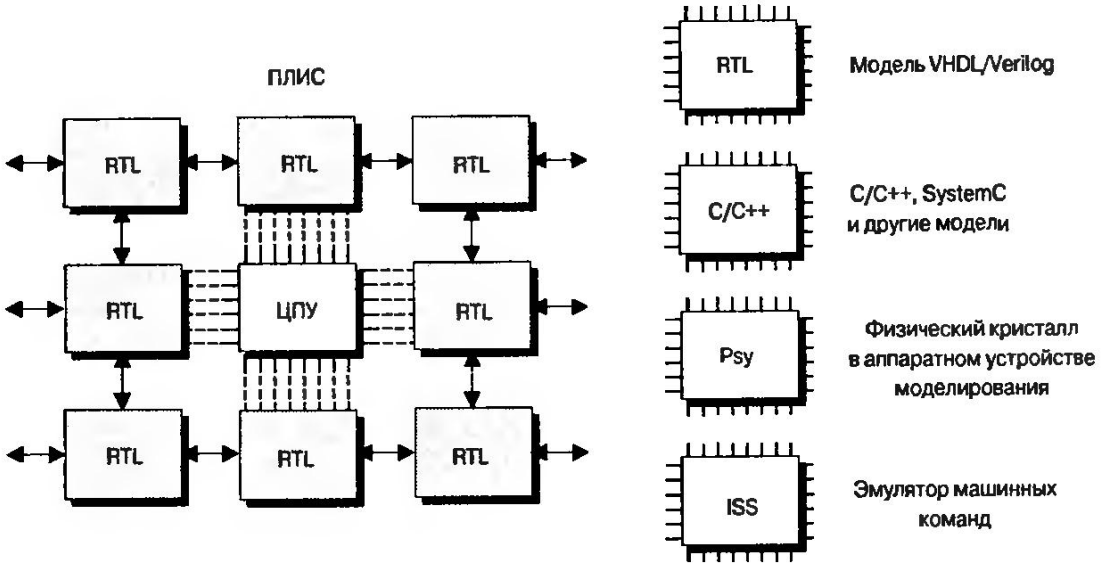


Рис. 13.8. Альтернативные представления микропроцессора

На **Рис. 13.8** показано только высокоуровневое представление содержимого ПЛИС. Если машинный код сохранён во внешней памяти устройства, это устройство также должно быть частью системы моделирования. На практике, если программное обеспечение взаимодействует с каким-то объектом, то этот объект должен быть частью системы совместной проверки.

RTL-описания: VHDL или Verilog

Возможно, что наличие RTL-описания микропроцессора обеспечит самый простой вариант проверки, так как в этом случае все действия выполняются в системе логического моделирования. Одним из недостатков этого метода является то, что для реализации простейших задач микропроцессор выполняет огромное количество внутренних операций. Другими словами, работа системы моделирования будет чрезвычайно медленной. Читатель, вы будете безмерно счастливы, если удастся смоделировать 10...20 системных тактов за секунду реального времени.

Другой недостаток заключается в отсутствии наглядности, т. е.: что же там выполняет программное обеспечение на уровне исходного кода. Всё, что можно сделать в этом случае — проанализировать изменения логических значений на проводниках и внутри регистров.

К тому же, следует учесть, что независимо от того, кто продал вам микропроцессор, он не будет заинтересован в том, чтобы пользователь знал, как работает его устройство. Это связано с тем, возможно, что поставщик использует запатентованные приёмы и желает сохранить в тайне свою интеллектуальную собственность. В этом случае будет довольно сложно понять работу RTL-модели микропроцессора.

C/C++, SystemC и т. д.

В качестве противовеса RTL-модели в обычной практике используются некоторые C/C++ модели микропроцессоров. (Сторонники языка SystemC мечтают о моделях микропроцессоров и основных периферийных устройств, описанных на этом языке, как о стандарте при работе в своей среде проектирования.)

Откомпилированная версия этой модели микропроцессора может быть связана с системой моделирования через *интерфейс языка про-*

граммирования (*PLI — Programming Language Interface*), в случае Verilog-моделирования, либо через *интерфейс незнакомых слов* (*FLI — Foreign language interface*) или его эквивалент, в случае VHDL-моделирования.

Данные модели имеют ряд преимуществ: они могут выполняться намного быстрее, чем их RTL-аналоги; могут поставляться в откомпилированном виде, тем самым, скрывая различные секреты, представляющие собой интеллектуальную собственность поставщика; и, по крайней мере, для ПЛИС, обычно являются бесплатными (поставщики ПЛИС продают кристаллы, но не модели).

Недостаток этого подхода состоит в том, что модели C/C++ не обеспечивают на 100% точное потактовое описание работы реального микропроцессора, что может стать причиной различных проблем при неаккуратной работе. Но, вновь повторюсь: всё же главный недостаток таких моделей заключается в том, что они представляют собой программу в машинном коде, т. е. пользователь не может просмотреть работу программного обеспечения на уровне исходного кода. Всё, что можно сделать — это проследить изменение логических значений на проводниках и внутри регистров.

1914 г. *Для изготовления радиоприёмников начали использоваться лампы-триоды.*

Когда-то давно компания Logic Modeling Corporation (LMC), которая впоследствии была приобретена компанией Synopsys, определила интерфейс связи между поведенческими моделями аппаратных блоков с системами логического моделирования, который назывался SWIFT. Модели, такие как микропроцессоры, которые поддерживали эту спецификацию, назывались SWIFT-моделями.

Реальные микросхемы в аппаратных устройствах моделирования

Однако существует и другой способ описать для моделирования аппаратное микропроцессорное ядро. Например, при использовании ядра PowerPC в ПЛИС фирмы Xilinx можете легко взять настоящий микропроцессор PowerPC. Этот микропроцессор можно установить в так называемое аппаратное устройство моделирования, которое может быть включено в систему логического моделирования.

Преимущество этого метода заключается в том, что физическая модель с максимальной эффективностью стремится функционально соответствовать аппаратному ядру. К недостаткам метода можно отнести некоторые трудности, связанные с большой стоимостью и сложностью таких аппаратных устройств моделирования.

Большинство аппаратных решений, основанных на устройствах моделирования, не поддерживают работу средств отладки совместно с исходным кодом. Другими словами, невозможно проследить работу программного обеспечения на уровне исходного кода[1]. Всё, что можно сделать — это проследить изменение логических значений на проводниках и внутри регистров.

Эмулятор машинных команд

Как уже отмечалось, в некоторых случаях роль программной части ПЛИС может быть отчасти ограничена. Например, программное обеспечение может быть использовано в качестве конечного автомата для

[1] На самом деле некоторые устройства моделирования предоставляют некоторые возможности отладки на уровне исходного кода, например, интересные решения предлагает компания Simpod Inc. (www.simpod.com).

1914 г. *Осуществлён первый трансконтинентальный телефонный звонок.*

управления некоторым интерфейсом. Программы могут также применяться для инициализации некоторых средств аппаратной части устройства, а после этого переходить в ждущий режим и контролировать работу аппаратуры. Такие C/C++ модели или физические модели, вероятно, будут достаточно полными, по крайней мере в том объеме, как полагают разработчики.

В некоторых случаях аппаратная часть устройства может в основном выполнять функции сопряжения с внешним миром. Например, аппаратура может считывать из внешнего источника блок данных и сохранять его в памяти ПЛИС. После этого микропроцессор может выполнить огромное количество сложных преобразований с этими данными. В этом и в других подобных случаях программистам необходимо иметь возможность проводить отладку своих программ на уровне исходного кода. Это, в свою очередь, потребует использования эмулятора машинных команд (ЭМК), который обеспечивает виртуальное представление микропроцессора.

Хотя ЭМК почти всегда создаётся с помощью языка C/C++, тем не менее, он будет существенно сильно отличаться от C/C++ моделей микропроцессоров, которые были рассмотрены в этой главе. Причина такого отличия заключается в очень высоком уровне абстракции, на котором описывается ЭМК. Кроме того, он оперирует терминами транзакций, например «дайте мне слово данных, расположенное по адресу *x* в памяти» и не заботится о мелких деталях, например, как сигналы будут вести себя в реальном мире. Наиболее просто объяснить принцип действия такого метода можно с помощью **Рис. 13.9**.

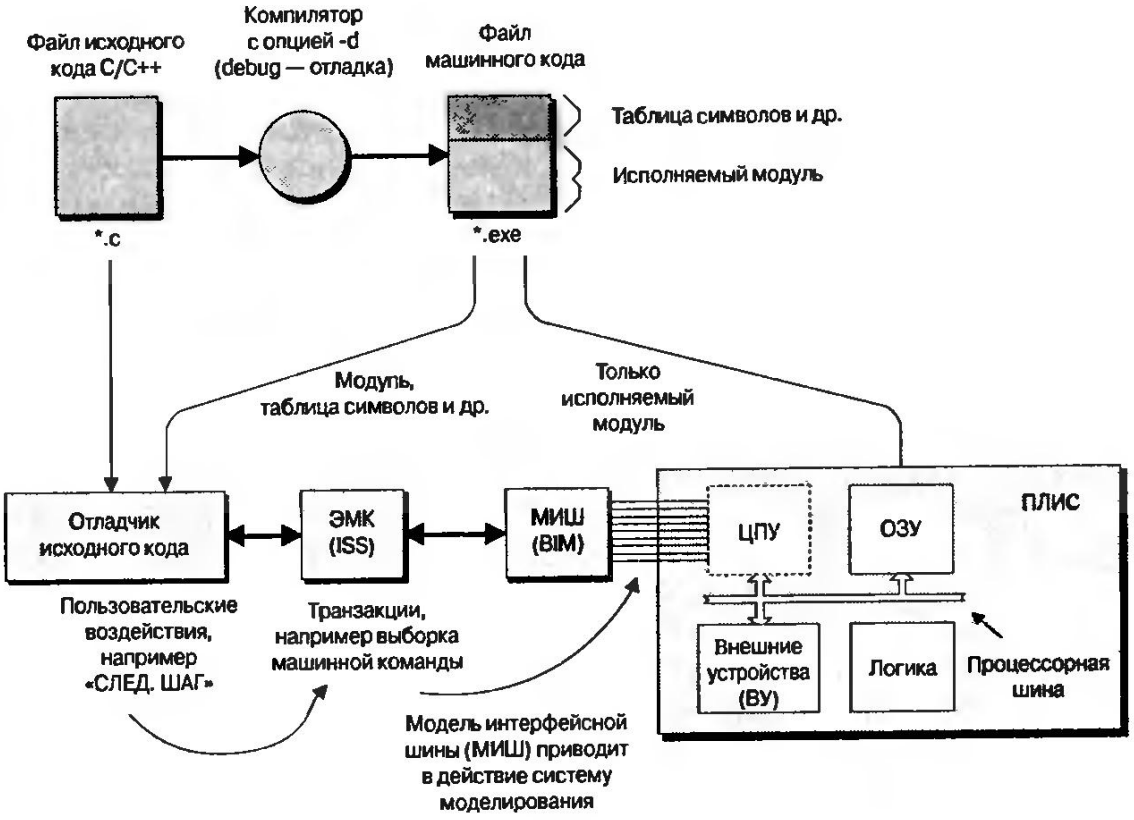


Рис. 13.9. Применение ЭМК

Во-первых, программисты описывают свои программы на языке C/C++. Затем производят компиляцию исходного кода, используя опцию *-d* компилятора (debug — режим отладки), вследствие чего совместно с исполняемым кодом генерируется таблица символов и другая отладочная информация.

1914 г. *Осуществлена передача радиосообщения с земли на аэроплан.*

Когда дело доходит до совместной проверки устройства, то процесс разделяется на несколько ветвей. С одной стороны находится отладчик исходного кода, чей интерфейс используется программистами для взаимодействия с оборудованием, а с другой стороны расположена система логического моделирования, которая работает с описаниями устройств памяти, внешних периферийных устройств, логики общего назначения и так далее (для простоты рисунка предположим, что вся память программ находится внутри ПЛИС).

При использовании микропроцессоров система логического моделирования очень тщательно анализирует место расположения этой функции. Если быть более точным, система моделирования обрабатывает набор входов и выходов, соответствующих микропроцессору. Эти входы и выходы, объединённые в одно целое, называются *моделью интерфейсной шины или МИШ (BIM — bus interface model)*, которая действует как транслятор между системой моделирования и ЭМК.

Далее в отладчик исходного кода загружается исходный код и исполняемый двоичный модуль вместе с таблицей символов и другой отладочной информацией. В это же время модуль двоичного исполняемого кода загружается в память ПЛИС. Когда пользователь указывает отладчику выполнить в пошаговом режиме строку кода, он выдаёт команду ЭМК. В свою очередь, эмулятор начнёт выполнять высокоуровневую транзакцию, например извлечение команды или чтение/запись памяти, или команду ввода/вывода. Эта транзакция проходит в блок МИШ, который с помощью соответствующих выводов ПЛИС приводит в действие систему моделирования.

Аналогично, когда какой-нибудь из блоков, подсоединённых к процессорной шине внутри ПЛИС, попытается взаимодействовать с процессором, этот процесс «расшевелит» систему МИШ, которая переведёт низкоуровневые действия в высокоуровневые транзакции, передаст их в ЭМК, который, в свою очередь, проинформирует отладчик исходного кода о происходящих событиях. После этого отладчик отобразит состояние программных переменных, регистров микропроцессора и другую подобную информацию.

На рынке существуют различные, невероятно сложные, часто отпугивающие дороговизной системы проектирования данного типа[1]. Каждая из них обладает своими хитрыми приёмами и возможностями. Некоторые в большей степени подходят для заказных микросхем, чем для ПЛИС, другие наоборот. Как обычно, все стремительно меняется и развивается, поэтому необходимо тщательно проанализировать каждое предложение, перед тем как выложить свои кровно заработанные деньги за тот или иной продукт.

Очень интересная среда проектирования

Насколько возможно (в разумной степени), в этой книге не акцентируется внимание на отдельных компаниях и их продукции. Но из всякого правила есть исключения, в том числе и из этого. В данном случае исключением является компания Altium Ltd. (www.altium.com), предложившая весьма интересную среду проектирования ПЛИС под названием Nexar, которая заслуживает, чтобы о ней упомянули в этой главе.

---

[1] Например, среда разработки Seamless компании Mentor (www.mentor.com), среда разработки Incisive компании Cadence (www.cadence.com) и XoC компании Axis Systems (www.axissystems.com).

1915 г. *Осуществлён первый трансатлантический радиотелефонный разговор.*

Я даже не знаю с чего начать. Ну, давайте начнём с того, что поговорим о полном наборе средств аппаратного и программного проектирования и тестирования для ПЛИС стоимостью примерно 7995 долларов США[1]. Эта среда проектирования предназначается инженерам, разрабатывающим небольшие системы, например простые контроллеры для бытовой техники, скажем, для стиральных машин, и основным их достоинством является доступность. Это значит, что пользователь может купить микросхему ПЛИС, содержащую более 1 миллиона системных логических элементов, примерно за 20 долларов США[2].

Среда разработки Nexar включает макетную плату, которая подключается к персональному компьютеру. Эта макетная плата оснащена двумя дочерними картами: на одной из них установлена ПЛИС фирмы Xilinx, а другая оснащена устройством компании Altera. Среда Nexar характеризуется также набором программных микропроцессорных ядер, имитирующих функциональность стандартных промышленных 8-битных устройств, таких как 8051, Z80, и PIC-микроконтроллеров (в будущем планируется поддержка 16-битных и 32-битных процессорных и ЦСП (DSP)-ядер). В состав пакета также входит библиотека периферийных устройств и библиотека из примерно 1500 компонентов, от простых вентилей до более сложных функций, таких как счётчики, а также операционная система реального времени (ОС РВ).

С помощью интерфейса графического описания схемы пользователь размещает блоки, представляющие собой процессоры, периферию, различные логические функции, и связывает их проводниками между собой. Все поддерживаемые средой Nexar блоки являются бесплатными, и не требуют выплаты авторского гонорара. Эти блоки прошли предварительную процедуру синтеза, поэтому на соответствующем этапе проектирования они могут быть загружены в ПЛИС на макетной плате. При необходимости можно создать собственные блоки и описать их содержимое на уровне регистровых передач. Впоследствии эти блоки будут обработаны средством синтеза из состава пакета Nexar.

Чтобы ввести исходный код программы на языке C/C++, которая должна выполняться этим процессором, надо щелкнуть мышью на блоке процессора. После ввода код будет откомпилирован одним из компиляторов, входящих в состав пакета Nexar.

Идея такого подхода заключается в том, что все части разрабатываемого устройства — и аппаратные, и программные — будут загружены в ПЛИС на макетной плате. Чтобы увидеть, что же происходит внутри аппаратной части устройства, можно включить в схему различные виртуальные блоки инструментов, в том числе логический анализатор, генераторы частот и прочее. Для контроля программной части в состав Nexar входит отладчик исходного кода, позволяющий формировать все стандартные задачи отладки, например установку точек останова, определение просматриваемых выражений, пошаговый режим и другие.

Что могу сказать лично я? Я действительно видел своими глазами эту штуковину в действии и был поражён. Мне очень понравилось, что в данном случае предлагается решение «под ключ», т. е. законченное решение, размеры которого не превышают размеров коробки из-под обуви, не считая, разумеется, дорогих достоинств. Что касается уровня разработки, я думаю, что такую блестящую работу как Nexar в ближайшее время вряд ли кто-то в силах повторить.

---

[1] Эта цена была действительна в ноябре 2003 года.

[2] Опять же, это количество элементов и цена были действительны в ноябре 2003 года.