ГЛАВА 5

ПРОГРАММИРОВАНИЕ ИЛИ КОНФИГУРИРОВАНИЕ ПЛИС

Ласковые слова

Перед тем как полностью погрузиться в эту тему, вероятно, имеет смысл в качестве небольшого вступления сказать несколько «*ласковых слов*». В этой связи на ум приходить поговорка: «Орел парит в небе, а ласка вряд ли попадёт в струю реактивного двигателя на высоте 10000 футов». (Рожденный ползать, летать не может. — *Прим. перев.*)

Дело в том, что у каждого производителя ПЛИС своя терминология, свои методы, средства работы. Чтобы жизнь не стояла на месте, механизмы программирования ПЛИС должны существенно меняться от семейства к семейству. Поэтому последующие рассуждения будут посвящены исключительно общим вопросам программирования микросхем.

Конфигурационные файлы и прочее

Вторая часть этой книги посвящена описанию множества инструментов и подходов, которые могут быть использованы для проектирования принципиальных схем и реализации ПЛИС-систем. С помощью этих средств разработчики получают *конфигурационный*, или *битовый, файл*, который содержит информацию, предназначенную для загрузки в микросхему, т. е. для её программирования с целью выполнения определенных функций.

Если микросхема использует для хранения своей конфигурации ячейки статического ОЗУ, конфигурационный файл содержит некоторую совокупность *конфигурационных данных*, или биты, используемые для определения состояния элементов программируемой логики и *конфигурационных команд*, т. е. инструкций, говорящих устройству, что ему необходимо делать с конфигурационными данными. При загрузке конфигурационного файла в устройство передаваемую информацию называют *конфигурационным двоичным потоком*.

Устройства на ячейках памяти ЭСППЗУ или Flash-памяти программируются аналогично своим родственникам на ячейках статического ОЗУ. В отличие от них в устройствах основанных на методе наращиваемых перемычек конфигурационный файл содержит только конфигурационные данные, которые будут использоваться для наращивания перемычек.

Конфигурационные ячейки

Основная концепция программирования ПЛИС относительно проста и представляет собой загрузку в устройство конфигурационного файла. Поскольку это довольно-таки сложный процесс, и чтобы он

1843 г. *Англия. Августа Ада Лавлейс (Augusta Ada Lovelace) опубликовала свои труды, посвященные рассмотрению концепции компьютера.*

был понятен, начнем с основ. Для начала рассмотрим элементарное устройство, состоящее из массива очень простых программируемых логических блоков, окруженных программируемыми внутренними соединениями (**Рис. 5.1**).

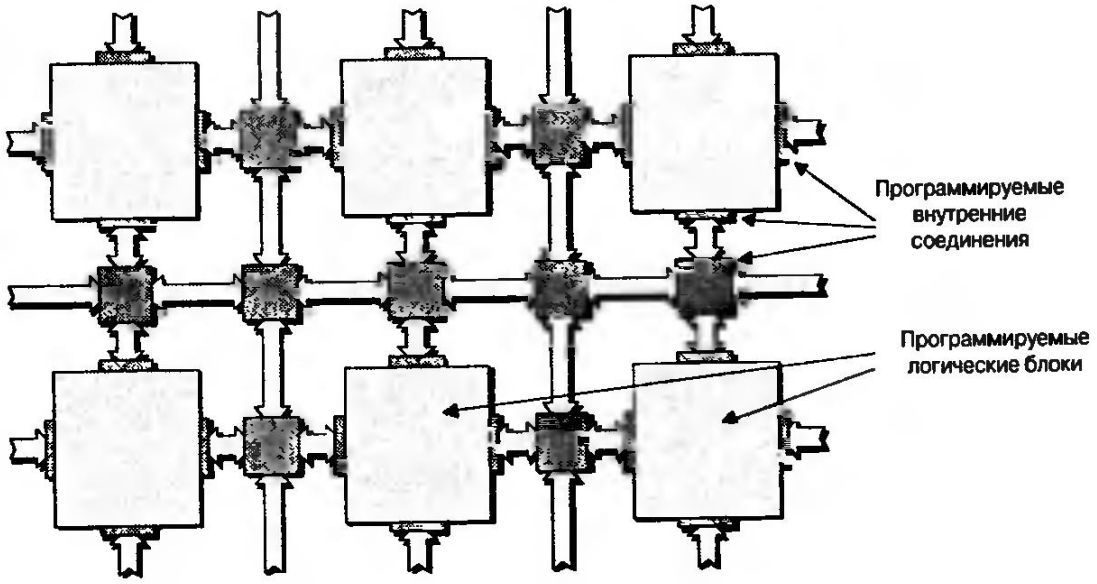

Рис. 5.1. Простая ПЛИС-архитектура (вид сверху)

Все возможности программируемых устройств реализуются с помощью специальных *конфигурационных ячеек*. Большинство ПЛИС используют ячейки статического ОЗУ, другие используют ЭСППЗУ или ячейки Flash-памяти, третьи основываются на наращиваемых перемычках.

Независимо от базовой технологии внутренние соединения устройства содержат большое количество связующих (соединительных) ячеек, которые можно использовать для его конфигурирования, чтобы соединить входы и выходы микросхемы с программируемыми логическими блоками, а также блоки между собой. Все блоки ввода/вывода, которые не показаны на **Рис. 5.1**, располагают несколькими соединительными ячейками, которые используются для их конфигурирования в соответствии с определенными стандартами интерфейсов ввода/вывода или другими параметрами.

Предположим, что каждый программируемый логический блок содержит всего лишь 4-входовую *таблицу соответствия*, мультиплексор и регистр (**Рис. 5.2**). Мультиплексор нуждается в соединительной конфигурационной ячейке, чтобы определить вход, сигнал с которого будет передаваться на его выход. Регистр нуждается в соединительных конфигурационных ячейках, чтобы определить будет ли он:

- выступать в роли триггера (как показано на **Рис. 5.2**) или в роли защелки;
- переключаться по фронту или по спаду синхроимпульса (при работе в режиме триггера) либо переключаться высоким или низким уровнем сигнала (в режиме защелки);
- инициализироваться логическим 0 или логической 1.

В свою очередь, 4-входовая таблица соответствия содержит 16 конфигурационных ячеек.

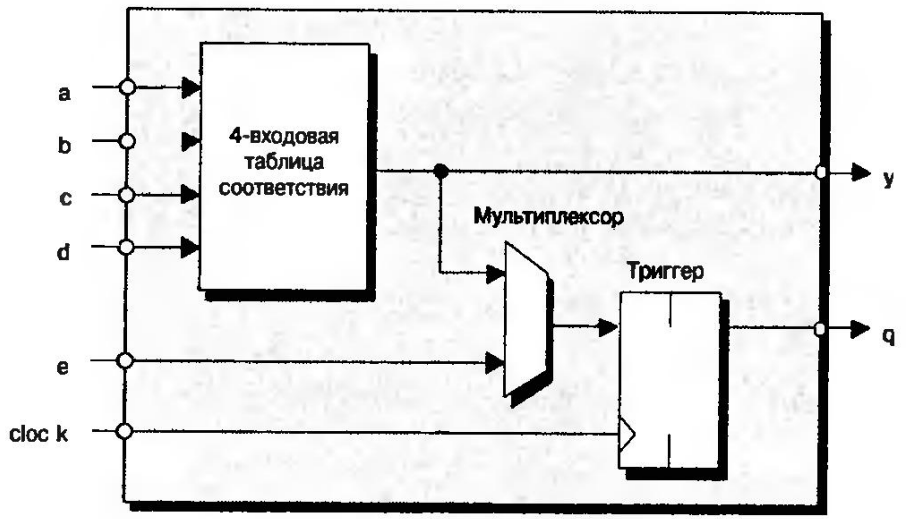

Рис. 5.2. Очень простой программируемый логический блок

ПЛИС на наращиваемых перемычках

В микросхемах на наращиваемых перемычках конфигурационные ячейки, как правило, распределены по всей поверхности устройства по определённым ключевым направлениям. Для программирования микросхема помещается в специальный программатор, в который из управляющего компьютера загружается конфигурационный или битовый файл. Этот файл используется программатором в качестве руководства по выдачи импульсов относительно высокого напряжения и большого тока на определенные выводы микросхемы, вследствие чего происходит поочерёдное наращивание перемычек.

Чтобы лучше понять процесс программирования, представим, что каждая наращиваемая перемычка имеет виртуальные *x-y* координаты на поверхности кристалла и эти *x-y* значения представляются в виде чисел. Далее вообразим, что одна группа контактов ввода/вывода микросхемы предназначена для представления значения *x*, а другая группа контактов предназначена для представления значения *y*. В реальной жизни всё выглядит намного сложнее, но этот пример позволяет понять процесс, не напрягая мозги.

После наращивания всех перемычек микросхема извлекается из программатора и помещается на печатную плату. Конечно же, устройства на наращиваемых перемычках являются *однократно программируемыми*, и с началом процесса программирования что-либо поменять в конфигурации устройства будет невозможно, как говорится, поезд ушел.

ПЛИС на ячейках статического ОЗУ

Рассмотрим ПЛИС-структуру на основе ячеек статического ОЗУ. Напомню, что эти устройства являются энергозависимыми. Это значит, что они программируются внутрисистемно, т. е. на печатной плате, причём программируются каждый раз после включения системы.

Все конфигурационные ячейки статического ОЗУ можно представить в виде одного длинного сдвигового регистра. Давайте рассмотрим простой рисунок поверхности кристалла, на котором изображены только контакты ввода/вывода и конфигурационные ячейки статического ОЗУ (**Рис. 5.3**).

Будем полагать, что начало и окончание, т. е. вход и выход, этой регистровой цепочки напрямую подключены к внешним контактам микросхемы. Однако при этом следует иметь в виду, что это возможно

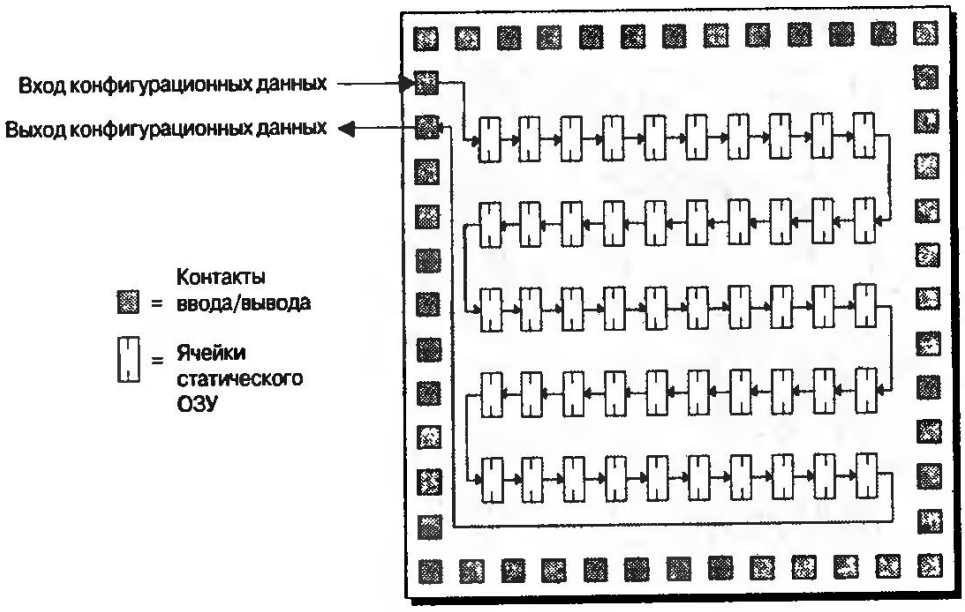

Рис. 5.3. Представление ячеек статического ОЗУ в форме большого сдвигового регистра

только в том случае, когда программирование осуществляется с помощью *конфигурационного порта* в режиме *последовательной загрузки*. Как будет показано, при последовательной загрузке ПЛИС может находиться в двух режимах: *ведущий (master)* и *ведомый (slave)*.

Также следует иметь в виду, что выходные контакты и выходные сигналы конфигурационных данных, показанные на **Рис. 5.3**, используются только в том случае, когда несколько ПЛИС сконфигурированы для работы в режиме каскадирования, т. е. последовательного опроса, или если по тем или иным причинам из устройства требуется прочитать конфигурационные данные.

Устройства на основе ячеек памяти Flash (и ЭСППЗУ) обычно программируются таким же способом, каким и микросхемы на основе статического ОЗУ, но при этом являются энергонезависимыми. Это свойство позволяет Flash-устройствам сохранять конфигурационную информацию при отключении питания системы. Другими словами, такие устройства не надо перепрограммировать при включении системы, хотя при необходимости такая процедура может быть выполнена. ПЛИС на основе Flash-элементов могут программироваться внутрисистемно, т. е. на печатной плате, или внесистемно с помощью отдельного программатора.

Ловкость рук и никакого мошенства

Вниманию читателя предлагается необязательный для чтения раздел. Если он торопится, может перейти к следующей теме. Но мне этот материал представляется очень интересным, и я надеюсь, что многим он придется по душе.

Как показано на **Рис. 5.3**, самый простой способ понять внутреннюю структуру программируемых ячеек статического ОЗУ заключается в представлении их в виде длинного сдвигового регистра. Так было бы в том случае, когда каждая ячейка была бы реализована в виде триггера, все триггеры были бы соединены в цепочку и управлялись общим синхросигналом.

Проблема заключается в том, что ПЛИС может содержать огромное количество конфигурационных ячеек. Уже к 2003-му году высокотехнологичные устройства могли запросто содержать 25 миллионов таких ячеек! Здесь необходимо заметить, что для реализации триггера

требуется восемь транзисторов, а для защелки всего лишь четыре. Поэтому конфигурационные ячейки в устройствах на статическом ОЗУ выполняются на защелках. Так, в устройстве с 25 миллионами конфигурационных ячеек такой подход позволяет сэкономить 100 миллионов транзисторов, что тоже немало.

Проблема кроется в том, что нельзя создать такой большой сдвиговый регистр из защелок. Ну, хорошо, хорошо. Ну, можно создать такой регистр, но, разумеется, не длиною в 25 миллионов ячеек. Поставщики ПЛИС обходят эту проблему с помощью группы триггеров, скажем из 1024-х, объединенных общим синхросигналом и конфигурируемых как классический сдвиговый регистр. Такую группу называют *фреймом*.

25 миллионов конфигурационных ячеек-защёлок в устройстве из нашего примера также разбивались на фреймы, при этом длина каждого равнялась длине специального триггерного фрейма. Внешне процесс представлял собой простую загрузку 25 миллионов бит конфигурационных данных в устройство. Однако внутри устройства, как только первые 1024 бита последовательно загружались в триггерный фрейм, специальная внутренняя схема автоматически параллельно копировала эти данные в первый фрейм защёлок. Затем следующие 1024 бита загружались в триггерный фрейм и автоматически параллельно копировались во второй фрейм защёлок и так далее до полного заполнения устройства. При чтении данных из устройства процесс протекал в обратном порядке.

Программирование ПЛИС может занимать значительное время. Если рассматривать высокотехнологичную микросхему, содержащую 25 миллионов ячеек статического ОЗУ, конфигурирование такого устройства в последовательном режиме с тактовой частотой 25 МГц займёт одну секунду. Это не очень плохой результат в случае, когда программирование производится при включении питания, но вряд ли пользователь захочет воспользоваться возможностью такой реконфигурации в процессе работы системы.

Программирование встроенных блоков ОЗУ, распределенного ОЗУ и других ОЗУ

Если ПЛИС содержит большие встроенные блоки ОЗУ, то ядра этих блоков выполняются из защелок на основе ячеек статической памяти. Каждая защелка, в свою очередь, является конфигурационной ячейкой, которая формирует часть воображаемой регистровой цепочки, которая рассматривалась в предыдущем разделе.

Интерес представляет тот факт, что каждая 4-входовая таблица соответствия (*см.* **Рис. 5.2**) может конфигурироваться для работы в качестве собственно таблицы соответствия либо в качестве небольшого блока (16×1) распределенного ОЗУ, либо в качестве 16-битного сдвигового регистра. Все эти конфигурации используют для своей работы группу из 16 защелок на основе статического ОЗУ, причем каждая защелка является конфигурационной ячейкой из состава рассмотренной нами регистровой цепочки.

«Ну и что следует из этого перевоплощения 16-битного сдвигового регистра? — спросит читатель. — Разве в этом случае не надо использовать настоящие триггеры?» А вот это уже хороший вопрос, и я рад, что он прозвучал. Вся хитрость схемы заключается в том, что в этом случае используется концепция ёмкостных защёлок, которые по многим параметрам превосходят классические защёлки. Очень похожим способом разработчики делали внешние триггеры из дискретных транзисторов, резисторов и конденсаторов в начале 60-х прошлого века.

1843 г. *Англия. Чарльз Уитстон (Sir Charles Wheatstone) совместно с Вильямом Куком (Sir William Fothergill Cooke) запатентовали 2-стрелочный электрический телеграф.*

1844 г. *Америка. Телеграф Морзе соединил города Вашингтон и Балтимор.*

Мультипрограммирование конфигурационных цепочек

На **Рис. 5.3** показаны конфигурационные ячейки в виде одной программируемой цепочки. Поскольку количество конфигурационных ячеек может достигать нескольких десятков миллионов, цепочка действительно может оказаться очень длинной. В некоторых микросхемах цепочка разбивается на части, и отдельные участки этой цепи подключаются к конфигурационному порту. Такой подход позволяет конфигурировать отдельные части устройства и упрощает реализацию различных концепций, таких как модульное и пошаговое проектирование.

Быстрая реинициализация устройства

Как уже отмечалось, регистры программируемых логических блоков соединены с конфигурационными ячейками, в которых содержится его исходное значение: логический 0 или логическая 1. Каждое семейство ПЛИС обычно поддерживает некоторый механизм, например, *вывод инициализации*, который при активации даёт команду регистрам вернуться к исходным значениям. Такой механизм не инициализирует встроенные блоки памяти или распределенное ОЗУ.

Конфигурационный порт

В своё время первые ПЛИС использовали инструмент, называемый *конфигурационным портом*. Даже сегодня, когда доступны более сложные методы, подобные JTAG-интерфейсу, конфигурационный порт все еще находит широкое применение, так как представляет собой довольно простой инструмент, и к тому же к нему с пониманием относятся его сторонки в сообществе ПЛИС.

Начнем рассмотрение с небольшой группы специальных контактов, или выводов, *режима конфигурации*, которые используются для установки режима конфигурации устройства. В первых ПЛИС для этих целей использовались только два вывода, которые позволяли задавать четыре режима конфигурации **(Табл. 5.1)**.

Таблица 5.1. **Четыре первичных режима конфигурации**

Выводы установки режима	Наименование режима
00	Последовательная загрузка, ПЛИС в режиме ведущий
01	Последовательная загрузка, ПЛИС в режиме ведомый
10	Параллельная загрузка, ПЛИС в режиме ведущий
11	Параллельная загрузка, ПЛИС в режиме ведомый

Следует иметь в виду, что названия режимов, а также соответствие кода на выводах установки режима и названием режима приведены только в качестве примера. Настоящие коды и названия режимов у каждого поставщика свои.

Выводы, или контакты, установки режима обычно подключаются к требуемым значениям логического 0 или логической 1 на печатной плате. Эти выводы могут подключаться к другим логическим устройствам, которые позволяют изменять режим программирования, но такая схема редко встречается на практике.

Кроме выводов установки режима, используется дополнительный вывод для информирования ПЛИС о начале процесса конфигурирования, и еще один вывод для выдачи сигнала, свидетельствующего об окончании процесса конфигурирования. Существуют также сигналы для определения ошибок, возникающих в ходе конфигурирования. Помимо конфигурирования ПЛИС при включении системы, при необходимости она может быть реинициализирована и возвращена к первоначальным конфигурационным данным.

Конфигурационный порт также использует дополнительные выводы микросхемы для управления загрузкой данных и для их ввода. Как будет показано, количество этих выводов зависит от выбранного режима конфигурации. Важное значение имеет тот факт, что при завершении конфигурирования большинство этих выводов могут использоваться в качестве входов/выходов общего назначения.

1845 г. *Англия. Майкл Фарадей (Michael Faraday) открыл явление вращения поляризации света при воздействии магнитного поля.*

Последовательная загрузка, ПЛИС в режиме ведущий

Вероятно, этот режим является самым простым из всех режимов программирования. Раньше для работы в этом режиме использовались внешние микросхемы ППЗУ. Впоследствии их заменили СППЗУ, затем ЭСППЗУ и теперь, как правило, используются устройства на ячейках flash-памяти. Эти компоненты специальной памяти содержат один вывод для выхода данных, который подсоединяется к *выводу ввода конфигурационных данных* ПЛИС (**Рис. 5.4**).

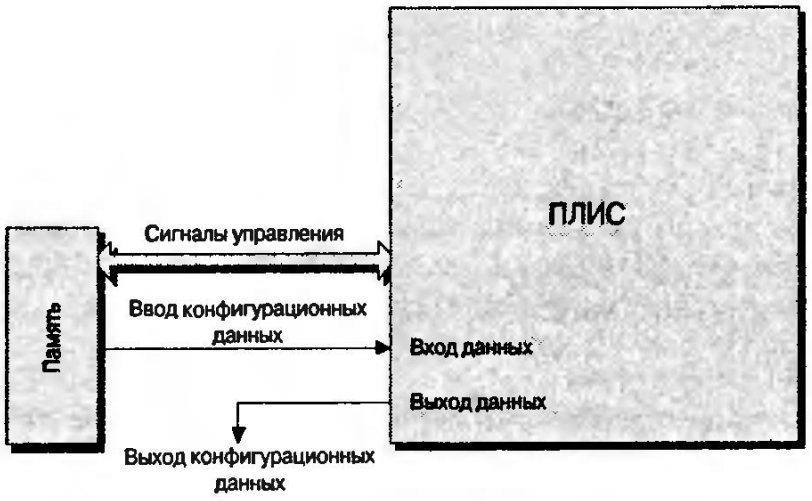

Рис. 5.4. **Последовательная загрузка, ПЛИС в режиме ведущий**

ПЛИС также использует несколько сигналов для управления внешней памятью. К ним относится сигнал сброса, который выдается ПЛИС при готовности ею начать чтение данных, а также синхросигнал, предназначенный для синхронизации конфигурационных данных.

В этом режиме ПЛИС не нуждается во внешней памяти с последовательной адресацией. При таком режиме используются простые импульсы сигнала сброса для передачи сигнала готовности начать чтение данных с начала конфигурационной последовательности, а затем применяется последовательность синхроимпульсов для синхронизации выдачи конфигурационных данных из памяти.

При необходимости выходной сигнал из ПЛИС может использоваться для чтения конфигурационных данных устройства. Такой подход имеет место при использовании нескольких микросхем ПЛИС на

1845 г. *Англия. Электрический телеграф впервые используется в качестве вспомогательного средства при раскрытии преступлений.*

одной печатной плате. В этом случае каждая микросхема может иметь собственное выделенное устройство внешней памяти и конфигурироваться по отдельности, как показано на **Рис. 5.4**. Согласно другому подходу, ПЛИС могут соединяться каскадом, т. е. по схеме последовательного опроса, и использовать совместно одно устройство внешней памяти (**Рис. 5.5**).

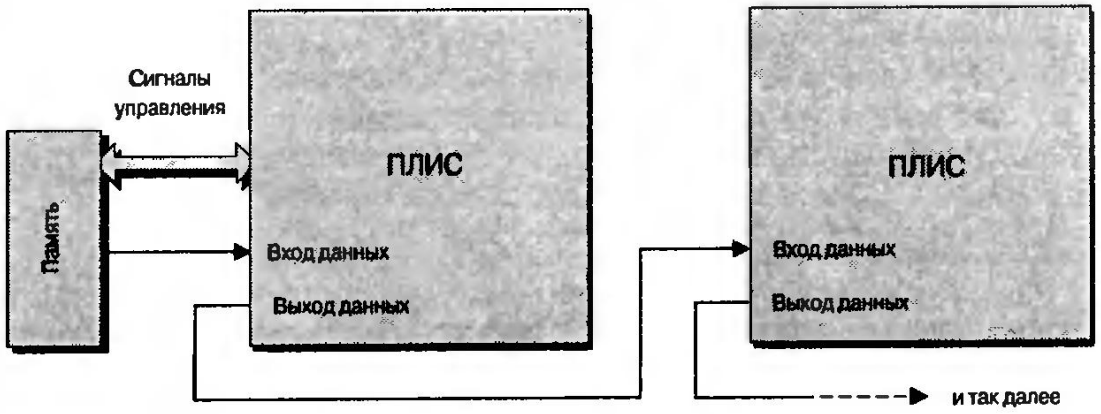

Рис. 5.5. Последовательное включение микросхем FPGA

В схеме, показанной на **Рис. 5.5**, первая ПЛИС в цепочке (она соединена напрямую с внешней памятью) будет конфигурироваться для работы в последовательном режиме как *ведущий* (*master*). Все последующие ПЛИС в этой цепочке должны быть сконфигурированы для работы в последовательном режиме как *ведомые* (*slave*).

Группы из четырёх битов в общем случае называются *ниблом* или *полубайтом*. Как это не удивительно, но два полубайта составляют один байт. Это ещё раз доказывает, что у инженеров с чувством юмора всё в порядке.

На заре компьютерной эры каждый желающий самостоятельно вводил в обращение свои собственные определения, и поначалу по разным версиям байт состоял из 5, 6, 7, 8 и даже 9 битов. Спустя некоторое время все заинтересованные лица пришли к консенсусу, и байты стали 8-битными. После этого все стали довольными и счастливыми. Правда и на этот раз не обошлось без недовольных, но на них никто не обращал внимания.

Параллельная загрузка, ПЛИС в режиме ведущий

Во многих отношениях этот режим подобен предыдущему, но при этом данные читаются 8-битными кусками из памяти, которая содержит также восемь выводов данных. Группы из восьми бит довольно широко распространены и называются *байтом*.

Кроме обеспечения управляющих сигналов, ПЛИС формирует для внешней памяти сигналы адреса, которые используются для указания байта конфигурационных данных, который будет загружаться на следующем такте (**Рис. 5.6**).

Подобная схема работает при условии, что ПЛИС содержит внутренний счетчик, который используется для генерации адреса для внешней памяти. Первые ПЛИС содержали 24-битные счетчики, которые позволяли им адресовать 16 миллионов байт данных. В начале конфигурационной последовательности счетчик сбрасывался в ноль. После того, как байта данных счетчика адреса будет считан, содержимое счетчика увеличивается для адресации следующего байта. Это процесс продолжался до тех пор, пока конфигурационные данные полностью не загрузятся в микросхему.

Казалось бы, что такой метод параллельной загрузки имеет преимущество в скорости по сравнению с рассмотренным ранее последовательным способом. Однако вначале это было не так. В первых микросхемах, как только байт данных считывался ПЛИС, он по-прежнему последовательно передавался во внутренний конфигурационный сдвиговый регистр. К счастью, эта ситуация была исправлена в более современных устройствах. Хотя восемь выводов микросхемы после загрузки конфигурационных данных могут использоваться как выводы

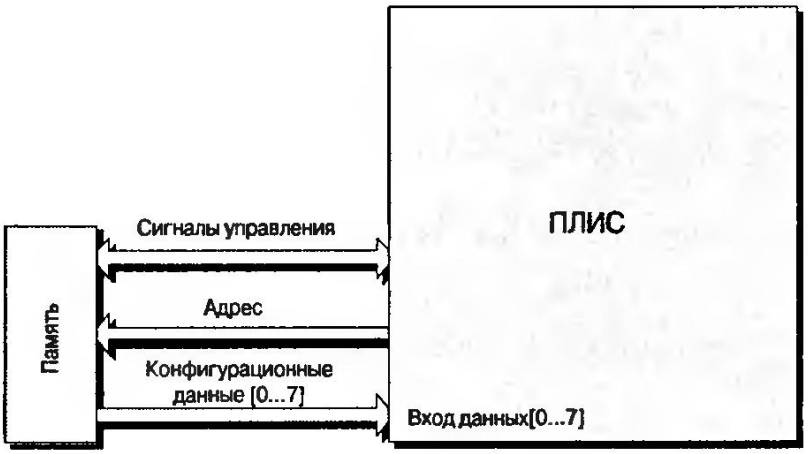

Рис. 5.6. Параллельная загрузка, ПЛИС в режиме ведущий (первый способ)

входа-выхода общего назначения, в реальных системах это свойство сопряжено с рядом трудностей. Вызваны они тем, что эти выводы жестко соединены с проводниками, соединяющими их с устройством внешней памяти, что может стать причиной проблем, связанных с интеграцией различных сигналов.

Действительная причина популярности этих микросхем заключалась в том, что специальные устройства памяти, используемые для последовательной загрузки конфигурационных данных в режиме *ведущий*, были довольно дорогими. По сравнению с ними параллельный метод позволял разработчикам использовать серийно выпускаемые устройства памяти, которые отличались низкой стоимостью.

Специальные устройства памяти, создаваемые для использования совместно с ПЛИС в настоящее время, являются относительно недорогими и изготавливаются по Flash-технологии, т. е. являются устройствами многократного использования. Современные ПЛИС используют новые варианты параллельной загрузки. В этих случаях внешняя память является специальным устройством, которое не требует внешней адресации, т. е. ПЛИС больше не нуждается во внутреннем счетчике для этих целей (**Рис. 5.7**).

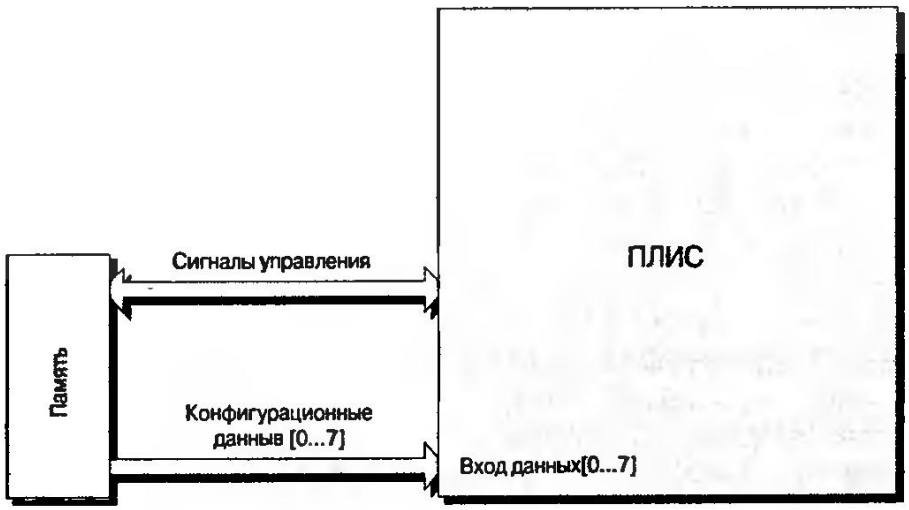

Рис. 5.7. Параллельная загрузка, ПЛИС в режиме ведущий (современная реализация)

1845 г. *Англия и Франция. Через Ла-Манш проложен первый телеграфный кабель.*

1846 г. *Германия. Густав Кирхгоф (Gustav Kirchhoff) сформулировал законы Кирхгофа для электрических цепей.*

В этом случае так же, как и в последовательном режиме, ПЛИС просто выдает сигнал сброса внешней памяти для обозначения того, что она готова начать чтение данных с начала конфигурационной последовательности, и затем выдает синхроимпульсы для синхронизации конфигурационных данных, поступающих из внешней памяти.

Параллельная загрузка, ПЛИС в режиме ведомый

Рассмотренные выше режимы конфигурирования, в которых ПЛИС выступала в роли *ведущей* (*master*) довольно привлекательны благодаря своей простоте и непритязательности, так как для работы требуется только ПЛИС и одна микросхема внешней памяти.

Однако на многих печатных платах содержатся микропроцессоры, которые обычно используются для выполнения разнообразных внутренних задач. В этом случае разработчики могут решить использовать микропроцессоры для загрузки конфигурационных данных в ПЛИС **(Рис. 5.8)**.

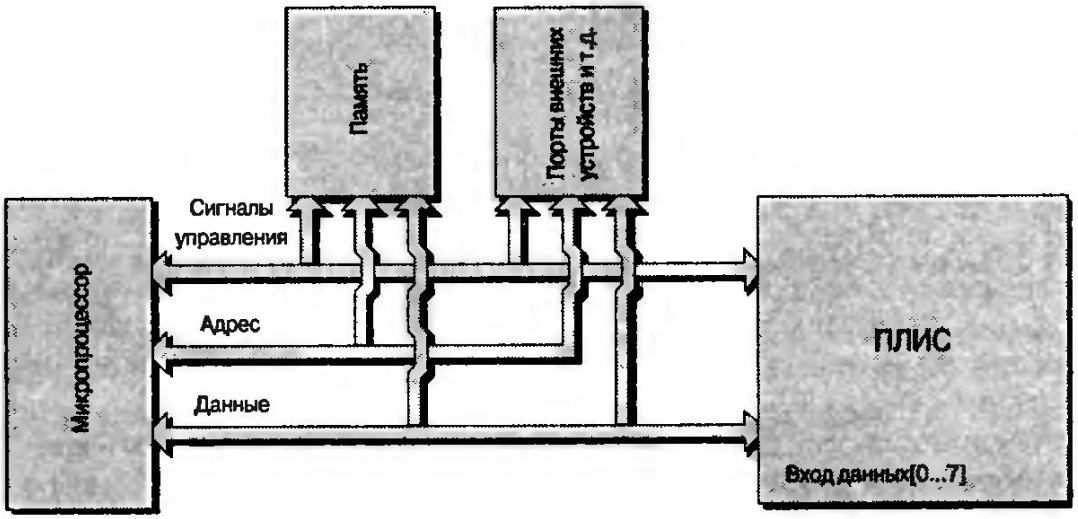

Рис. 5.8. Параллельная загрузка, ПЛИС в режиме ведомый

Идея заключается в том, что микропроцессор является управляющим устройством. В этом режиме микропроцессор сообщает ПЛИС, что он готов начать процесс конфигурирования, после этого производит чтение байта конфигурационных данных с определенного устройства памяти или с внешнего устройства, или ещё откуда-то, записывает эти данные в ПЛИС, считывает следующий байт данных из памяти, записывает его в ПЛИС и так до окончания процесса конфигурирования.

Такой подход обладает рядом преимуществ, не последним из которых является то, что микропроцессор может использоваться для запроса оборудования, принадлежащее соседней системе, и выбирать по определенному признаку конфигурационные данные для загрузки в ПЛИС.

Последовательная загрузка, ПЛИС в режиме ведомый

Этот режим аналогичен своему параллельному родственнику, но при этом для загрузки данных в ПЛИС используется только один бит. Микропроцессор по-прежнему производит чтение одного байта данных из устройства памяти и затем преобразует этот байт в последовательность битов для записи в ПЛИС.

Главным преимуществом такого подхода является то, что его реализация требует меньшее количество выводов, задействованных у ПЛИС. Другими словами, в процессе конфигурирования используется только один контакт ввода/вывода, который с помощью дополнительного проводника подключается к шине данных микропроцессора.

JTAG-порт

Многие современные устройства, в том числе и ПЛИС, оборудованы специальным интерфейсом, который называется *JTAG-порт*. Этот порт, поддерживаемый *Объединенной рабочей группой по автоматизации тестирования (Joint Test Automation Group, JTAG)* и известный как стандарт IEEE 1149.1, изначально разрабатывался для реализации *метода периферийного сканирования*, который применялся для тестирования печатных плат и цифровых микросхем.

Детальное описание порта JTAG и метода периферийного сканирования выходит за рамки этой книги. В данном контексте важно, что ПЛИС имеет некоторое количество выводов-контактов, которые используются в качестве JTAG-порта. При этом один из этих выводов используется для ввода данных, другой для вывода, а остальные для связи с последовательно соединенными JTAG-регистрами или триггерами (**Рис. 5.9**).

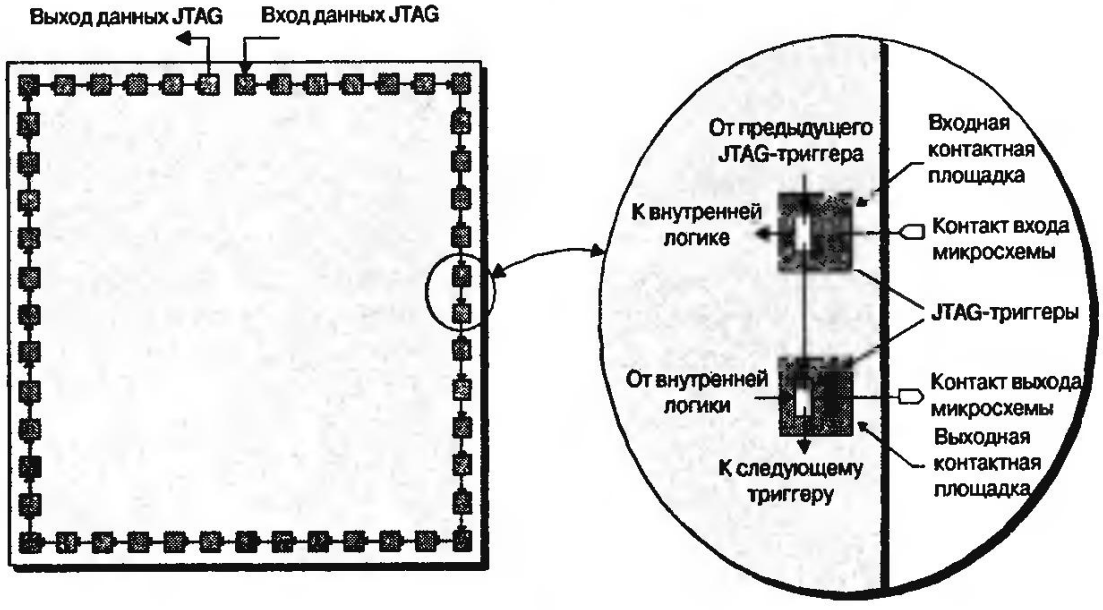

Рис. 5.9. JTAG-регистры последовательного сканирования

Идея последовательного сканирования состоит в следующем: через JTAG-порт можно последовательно передавать данные в JTAG-регистры, которые связаны с входными выводами микросхемы. В результате устройство, в данном случае ПЛИС, получает возможность работать с этими данными, сохранять результаты этой работы в JTAG-регистрах, связанных с выходными выводами микросхемы, и последовательно выдавать эти результаты обратно в JTAG-порт.

JTAG-устройства содержат различную дополнительную управляющую логику. В случае ПЛИС JTAG-порт, кроме операций последовательного сканирования, может выполнять и другие функции. Например, возможно выдавать специальные команды, которые загружаются в специальный командный JTAG-регистр (он не показан на **Рис. 5.9**) через вход JTAG-порта. Одна такая команда указывает ПЛИС подсоединить свой внутренний конфигурационный сдвиговый регистр к

1847 г. *Англия. Джордж Буль (George Boole) опубликовал первые идеи символической логики.*

сканирующей цепочке JTAG-порта. В этом случае JTAG-порт может использоваться для программирования ПЛИС. Таким образом, современные ПЛИС поддерживают пять различных режимов программирования, и, следовательно, для определения того, какой из них будет использоваться, потребуется уже три вывода, как показано на **Табл. 5.2** (в будущем также могут появиться и дополнительные режимы).

Таблица 5.2. **Пять современных конфигурационных режимов**

Выводы установки режима	Наименование режима
000	Последовательная загрузка, ПЛИС в режиме ведущий
001	Последовательная загрузка, ПЛИС в режиме ведомый
010	Параллельная загрузка, ПЛИС в режиме ведущий
011	Параллельная загрузка, ПЛИС в режиме ведомый
1xx	Используется только JTAG-порт

Заметим, что JTAG-порт всегда доступен к использованию, то есть устройство может быть сначала сконфигурировано через обычный конфигурационный порт, используя один из стандартных конфигурационных режимов, а затем, при необходимости, может быть реконфигурировано при помощи JTAG-порта. Также устройство может программироваться с помощью только JTAG-порта.

Встроенные процессоры

Но и это ещё не всё! В гл. 4 говорилось о том, что некоторые ПЛИС содержат встроенные процессорные ядра, причем каждое ядро имеет свою специфичную JTAG-цепочку последовательного сканирования. Рассмотрим ПЛИС, содержащую только один встроенный процессор (**Рис. 5.10**).

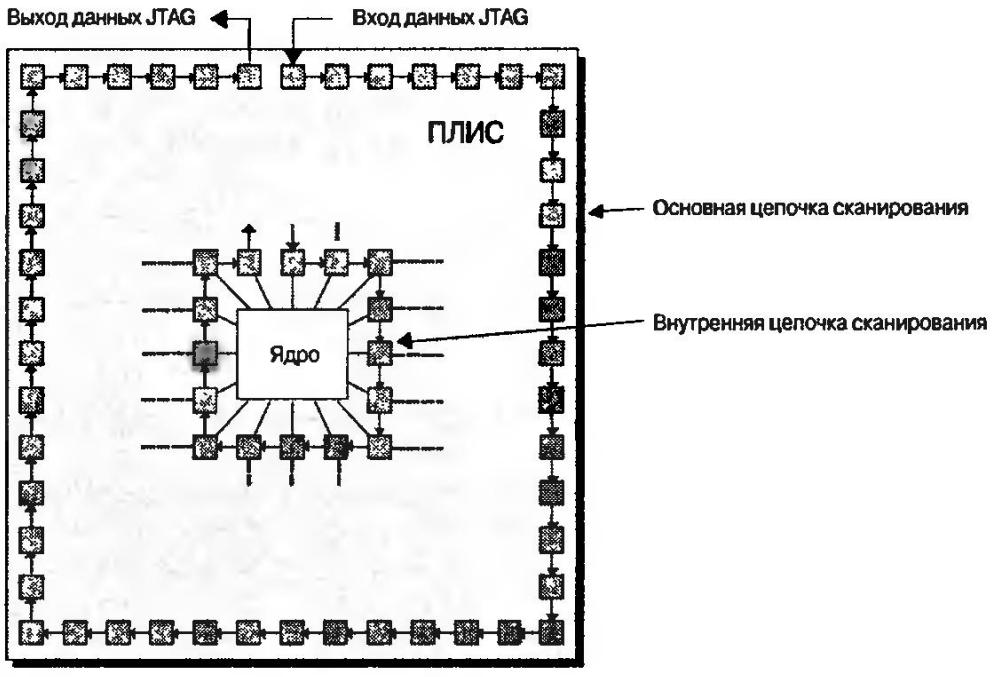

Рис. 5.10. Цепочка последовательного сканирования встроенного процессора

В этом случае ПЛИС будет содержать только один внешний JTAG-порт. При необходимости через этот порт можно загрузить команду, которая даст указание процессорной JTAG-цепочке связаться с главной JTAG-цепочкой устройства. Если, в зависимости от поставщика, обе цепочки могут быть соединены вместе по определению, то для их разделения будет использоваться добавочная команда.

Идея состоит в том, что JTAG-порт может использоваться для инициализации внутреннего микропроцессорного ядра и связанной с ним периферии до некоторого состояния, после которого процессор самостоятельно загружает основную часть конфигурационных данных. В некоторых случаях процессорное ядро может использоваться для выполнения запросов во внешнюю среду и выбора конфигурационных данных, предназначенных для загрузки в ПЛИС.

1850 г. *Англия. Френсис Гальтон (Francis Galton) изобрёл телетайп.*