ГЛАВА 12

ПРОЕКТИРОВАНИЕ СРЕДСТВ ЦИФРОВОЙ ОБРАБОТКИ СИГНАЛОВ

Введение в ЦОС

Понятие *цифровая обработка сигналов,* или *ЦОС*, относится к области электроники, предметом которой является представление и обработка сигналов в цифровой форме. Такая обработка включает в себя сжатие, распаковку, модуляцию, коррекцию ошибок, фильтрацию и другие действия с аудио- (голос, музыка и так далее) и видеосигналами, изображениями и другими данными для связи, локации и обработки изображений, включая рентгеновские изображения.

Во многих случаях до начала обработки сигналы находятся в реальной, или аналоговой, форме. Аналоговый сигнал периодически дискретизируется и каждый отсчёт преобразуется в цифровую форму путем *аналогово-цифрового преобразователя (АЦП),* **Рис. 12.1**.

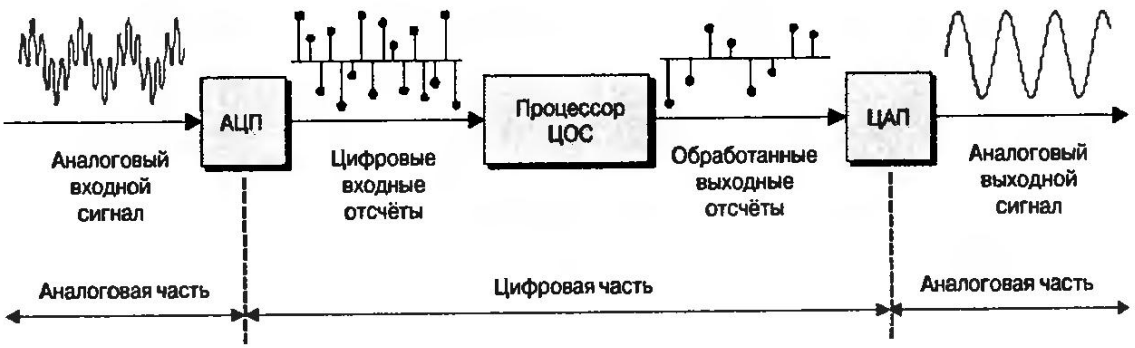

Рис. 12.1. Что такое цифровая обработка сигналов

Затем эти отсчёты обрабатываются в цифровой части системы. Во многих случаях обработанные цифровые данные вновь преобразуются в аналоговую форму с помощью *цифро-аналогового преобразователя (ЦАП).*

Цифровая обработка сигналов применяется во всех сотовых телефонах и телефонных системах, в CD, DVD и MP3-плеерах, настольных кабельных системах, в средствах беспроводной связи, медицинском оборудовании, электронных системах видеонаблюдения... Этот список можно продолжить. Другими словами, рынок применения ЦОС просто огромен, и по некоторым оценкам его оборот в 2003 году превысил 10 миллиардов долларов США!

Альтернативы реализации ЦОС

Выбрать любое устройство, но мне его не показывать

И опять не всё так просто, как мечтается, поскольку задачи цифровой обработки могут быть решены разными способами и с помощью различных средств:

- *Микропроцессоры общего назначения (МП)* — центральные процессоры (ЦП) или микропроцессорные устройства (МПУ), ко-

торые могут производить цифровую обработку сигнала с помощью соответствующего алгоритма ЦОС.

- *Цифровой сигнальный процессор, ЦСП (DSP — Digital Signal processor)* — микропроцессорный кристалл, или ядро, специально разработанный для более быстрого и эффективного выполнения задач цифровой обработки сигналов, чем при использовании микропроцессоров общего назначения.
- *Специализированные системы на основе заказных микросхем* — в рамках этого раздела будем полагать, что это название относится к изготовленному по заказу устройству на заказных микросхемах, которое выполняет задачи цифровой обработки сигналов. Однако необходимо иметь в виду, что задачи цифровой обработки могут быть реализованы программно путем включения в состав заказной микросхемы микропроцессорного ядра или ядра ЦСП.
- *Специализированные системы на основе ПЛИС* — в рамках этого раздела будем полагать, что это название относится к изготовленному по заказу устройству на основе ПЛИС, которое выполняет задачи цифровой обработки сигналов. Здесь также необходимо иметь в виду, что задачи цифровой обработки могут быть реализованы программно посредством встраивания микропроцессорного ядра в ПЛИС. Во время написания этой книги среди ПЛИС не существовало заказных аппаратных ядер цифровой обработки сигналов.

Оценки системного уровня и алгоритмическая верификация

Независимо от технологии реализации, т. е. использования микропроцессора, ЦСП (DSP), заказной микросхемы или ПЛИС, при создании системы с реализацией алгоритмов цифровой обработки в общем случае на первом этапе проектирования с помощью соответствующих программных средств выполняется оценка устройства на системном уровне и его алгоритмическая проверка или *верификация*.

В этой книге я пытался по мере сил и возможностей не акцентировать внимания на определённых компаниях или продуктах, но было бы весьма некорректно не упомянуть, что во время ее написания дефакто промышленным стандартом алгоритмической проверки систем ЦОС являлся программный комплекс MATLAB®[1] компании The MathWorks (www.mathworks.com)[2].

Поэтому в рамках данной главы я при необходимости буду ссылаться на MATLAB. При этом замечу, что инженеры располагают некоторыми другими очень мощными средствами. К числу таких средств относятся, например, Simulink от компании The MathWorks, которое очень популярно, особенно в области телекоммуникации, программа Signal Processing Worksystem (SPW) компании CoWare[3] (www.coware.com), а также программные продукты компании Elanix (www.elanix.com).

---

[1] MATLAB и Simulink являются зарегистрированными торговыми марками компании The MathWorks Inc.

[2] Следует заметить, что MATLAB и Simulink могут быть использованы для широкого круга задач, включая разработку и анализ систем управления, финансовое моделирование и так далее.

[3] Область САПР электронных систем довольно быстро меняется. Например, программа SPW в то время, когда я начинал писать эту главу, находилась под покровительством компании Cadence, а когда я написал полглавы, она перешла к компании Coware!

1906 г. *Америка. Начала вещание радиопрограмма, передающая голос и музыку.*

Работа программного обеспечения на ядре ЦСП

Допустим, что алгоритм цифровой обработки сигналов должен быть реализован с помощью микропроцессора или цифрового сигнального процессора (ЦСП). В этом случае методика реализации может выглядеть так, как показано на **Рис. 12.2.**

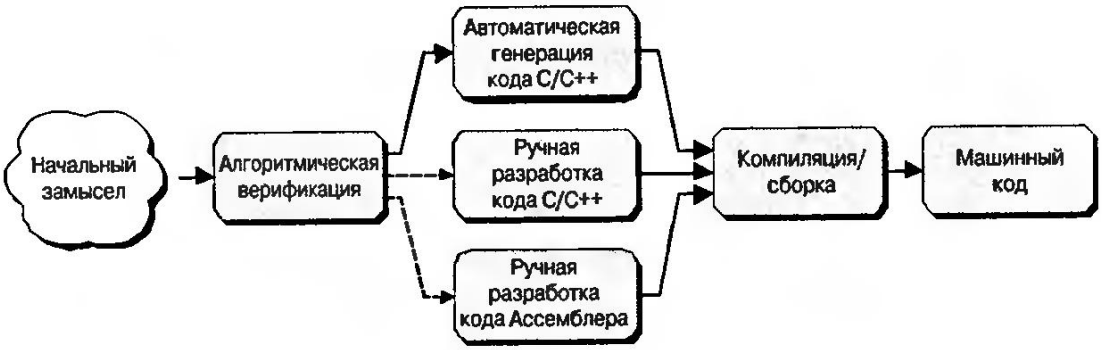

Рис. 12.2. Простой метод проектирования программного ЦСП

Процесс начинается с того, что у кого-то возникает идея нового алгоритма или совокупности алгоритмов. Эта новая концепция обычно подвергается проверке с помощью специальных средств, таких как MATLAB. В некоторых случаях можно перейти непосредственно от концепции к ручной разработке кода на языке C/C++ или на языке Ассемблера.

После проверки алгоритмы должны быть переведены в код языка C/C++ или языка Ассемблера. MATLAB может автоматически генерировать код C/C++ специально для ядер цифровой обработки сигналов. Однако в некоторых случаях разработчики предпочитают выполнять эту операцию пошагово и вручную, поскольку полагают, что могут достичь более оптимального представления. Существуют и другие методы создания кода. Например, можно сначала автоматически сгенерировать C/C ++ код из окружения алгоритмической проверки, проанализировать и спрофилировать его для определения узких мест в производительности устройства и затем вручную перекодировать наиболее критичные узлы. Это хороший пример старого правила 80:20, которое гласит, что 80% рабочего времени тратится на наиболее критичные 20% устройства.

После создания представления устройства в виде кода на языке C/C++ или языке Ассемблера этот код должен быть скомпилирован, или транслирован, в машинный код, который будет выполняться микропроцессором и ядром ЦСП.

Этот подход является очень гибким, так как позволяет относительно просто и быстро производить необходимые изменения в устройстве путем простой модификации и перекомпилирования исходного кода. Однако подобный подход приводит к медленной работе алгоритма ЦОС, так как микропроцессор и кристалл ЦСП попадают под определение машины Тьюринга (Turing machines). Это значит, что их первичная роль заключается в выполнении инструкций, поэтому оба этих устройства работают по следующей схеме:

- Выборка команды.
- Декодирование команды.
- Выборка блока данных.
- Выполнение операции с данными.
- Сохранение результатов.
- Выборка другой инструкции и повтор всех операций.

Конечно, алгоритм цифровой обработки действительно реализуется на аппаратном обеспечении, выполненном в виде микропроцессора или ЦСП. Но здесь он рассматривается в качестве программной реализации, поскольку действительное, или физическое, воплощение алгоритма представляет собой программу, которая выполняется на кристалле.

В 1937 г. *Еще будучи студентом, эксцентричный английский гений Алан Тюринг написал работу «Теория логических вычисляющих машин». Не имея доступа к реальным компьютерам, поскольку они в то время просто не существовали, он разработал собственный метод «вычислений на бумаге». Позже эту теоретическую модель назвали машиной Тюринга.*

Специализированное аппаратное обеспечение ЦОС

Существует множество способов реализации алгоритмов цифровой обработки на основе заказной микросхемы и ПЛИС. Данная глава посвящена самым последним достижениям в этой области. Но перед тем как окунуться в эту сложную тему, давайте сначала рассмотрим, как различные архитектуры могут влиять на скорость и размеры, с точки зрения места на кристалле, разрабатываемого устройства.

Алгоритмы цифровой обработки обычно требуют выполнения огромного количества операций умножения и сложения. В качестве очень простого примера возьмем алгоритм цифровой обработки, который содержит выражение вида Y = (A * B) + (C * D) + (E * F) + (G * H).

По традиции этот синтаксис не соответствует ни одному конкретному HDL-языку и используется только в наших рассуждениях. Естественно, это мизерный элемент чрезвычайно сложного алгоритма. Но, в конце-то концов, алгоритмы цифровой обработки обычно содержат огромное количество выражений подобного типа.

Для читателей нетехнического круга следует заметить, что переменные A, B, C и D в вышеприведённом выражении используются для представления шины (группы) двоичных сигналов. При умножении двух двоичных чисел одинаковой ширины, ширина результата будёт в два раза шире, чем у аргументов (так если A и B занимают по 16 бит, то результат получится шириной 32 бита).

Суть заключается в том, что мы можем использовать параллелизм, свойственный аппаратным решениям, для более быстрого выполнения функций ЦОС, чем можно было бы достичь с помощью программного подхода. Допустим, например, что все операции умножения были выполнены параллельно, одновременно, а затем последовали две стадии сложения (**Рис. 12.3**).

Несмотря на то что сумматоры и особенно умножители являются относительно большими и сложными модулями, всё же можно предположить, что эта реализация будет очень быстрой, но в то же время будет потреблять большое количество ресурсов.

Вместе с тем, можно использовать *распределение ресурсов* (распределяя некоторые умножители и сумматоры между операциями умножения) и выбрать решение, которое будет являться смесью параллельного и последовательного соединения (**Рис. 12.4**).

Это решение потребует дополнительно четырёх мультиплексоров вида 2:1 и одного регистра, причем каждый из них должен иметь ту же разрядность, что и используемые сигналы. Однако мультиплексоры и регистр занимают намного меньше места на кристалле, чем два умножителя и сумматор, которые, между прочим, уже не требуются в текущей реализации.

Этот метод медленнее предыдущего, поскольку сначала надо: произвести умножения (A * B) и (C * D), сложить результаты вместе, сложить получившуюся сумму с текущим содержимым регистра, который в начале будет содержать значение 0, и записать результат в регистр.

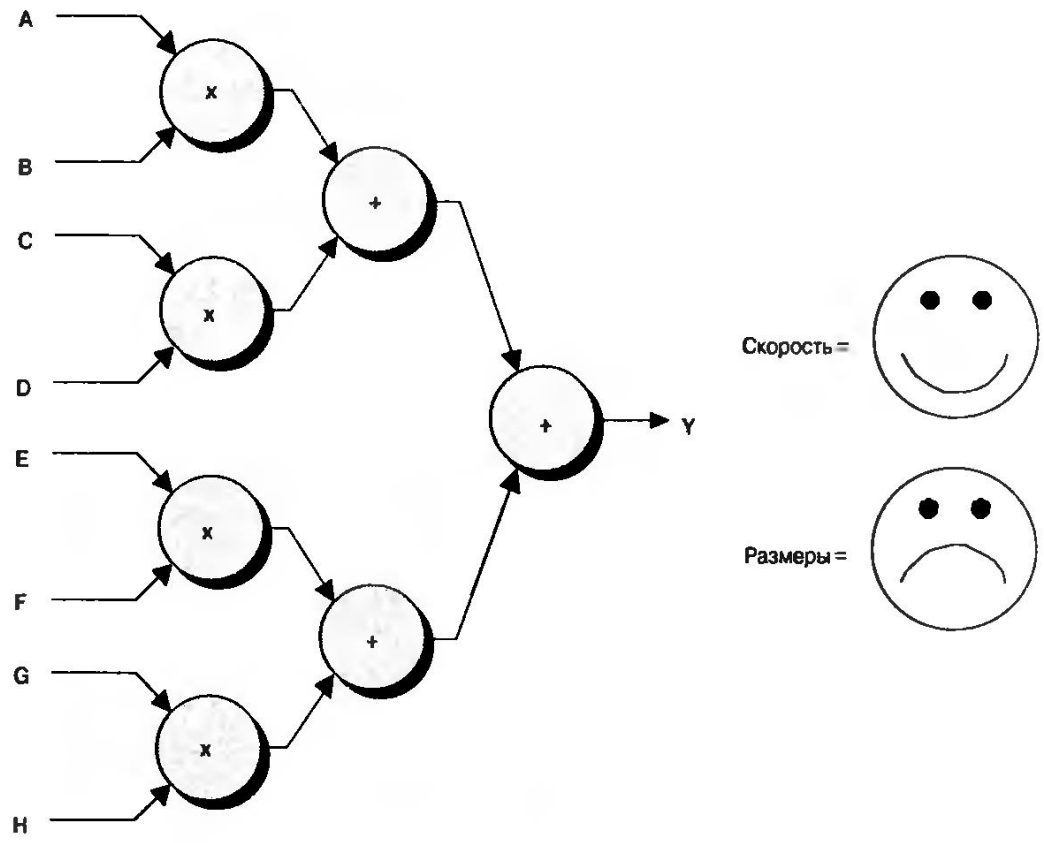

Рис. 12.3. Параллельная реализация функции

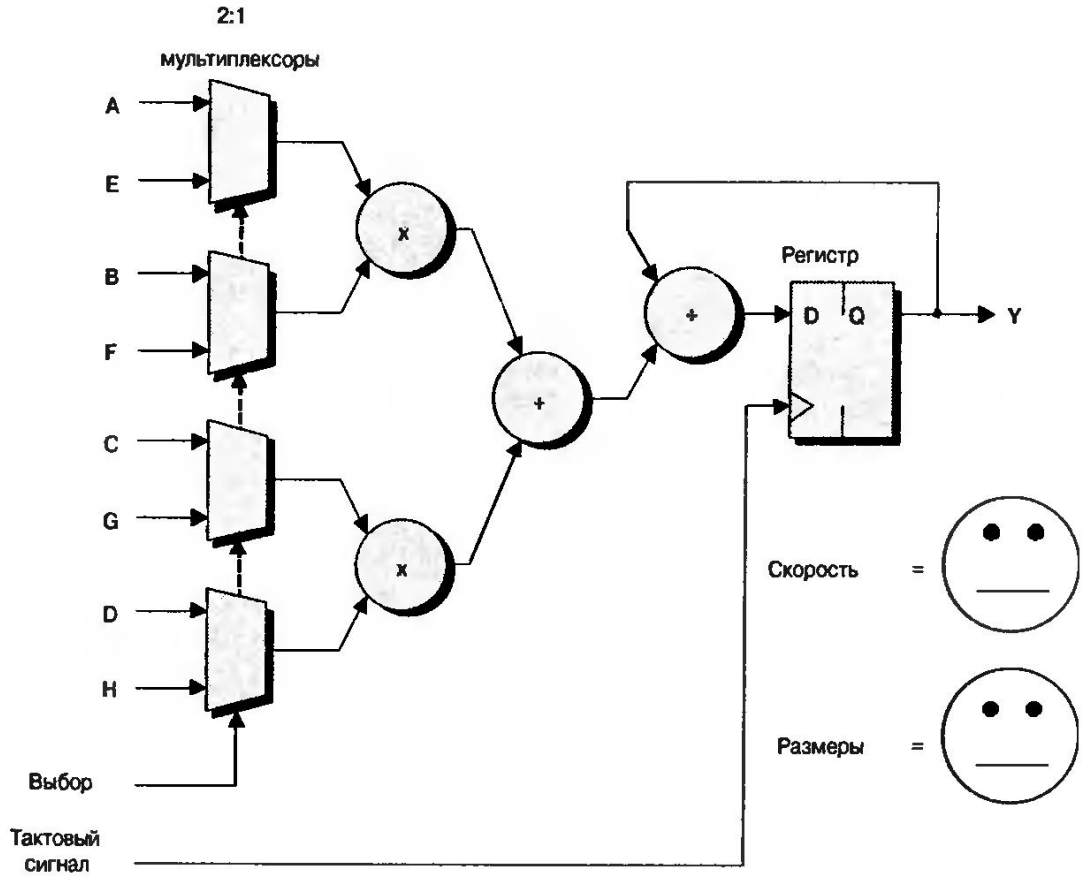

Рис. 12.4. Компромиссная реализация функции

Затем выполнить умножения (E * F) и (G * H), просуммировать результаты вместе, сложить полученную сумму с текущим содержимым регистра, который теперь содержит результаты первой части умножений и сложений, и сохранить результат в регистр.

1906 г. *Разработана первая лампа накаливания с вольфрамовой нитью.*

А можно поступить иначе, т. е. последовательное решение (**Рис. 12.5**).

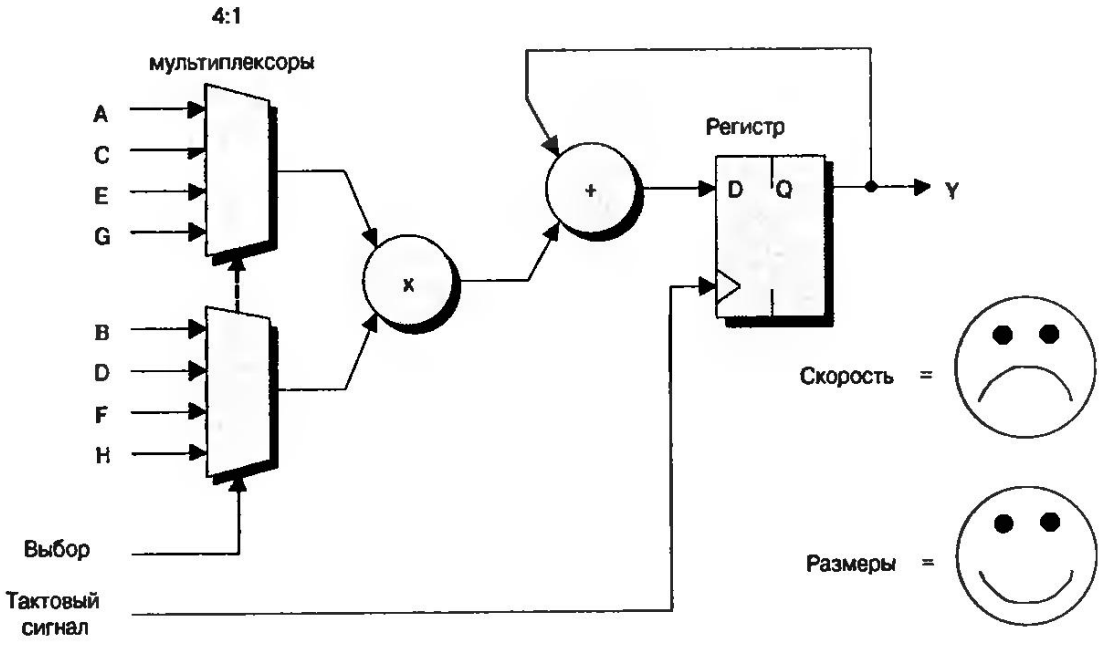

Рис. 12.5. Последовательная реализация функции

Последняя реализация очень эффективна с точки зрения занимаемого на кристалле места, так как для этого потребуется только один умножитель и один сумматор. Однако это довольно медленная реализация, поскольку сначала надо выполнить умножение (A * B), сложить результаты с текущим значением регистра, который в начальный момент времени будет содержать значение 0, и записать результат в регистр. Затем надо выполнить умножение (C * D), сложить результат с содержимым регистра и записать новую сумму в регистр. И так далее со всеми оставшимися операциями умножения. Замечу, что когда мы говорим «это медленная реализация», мы говорим это по сравнению с другими аппаратными решениями, но самое медленное аппаратное решение может оказаться намного быстрее программной реализации на микропроцессоре или ЦСП.

Другие встраиваемые в ПЛИС ресурсы ЦОС

Как уже обсуждалось в гл. 4, некоторые функции, например, умножители, по своей сути являются довольно медленными, если они реализованы с помощью большого количества соединённых вместе программируемых логических блоков. Поскольку эти функции востребованы большим количеством приложений, многие устройства ПЛИС содержат встроенные аппаратные умножители. Эти блоки обычно располагаются в непосредственной близости к встроенным блоками ОЗУ, так как эти функции очень часто используются вместе.

Некоторые ПЛИС содержат специализированные блоки сумматоров. В приложениях цифровой обработки сигналов довольно часто используется операция, называемая *умножение с накоплением*. Как следует из названия, эта операция сводится к умножению двух чисел и сложению результата с текущим значением аккумулятора, или регистра. Для обозначения этой операции используется аббревиатура MAC (multiply-and-accumulate), которая подразумевает умножение, сложение и накопление (**Рис. 12.6**).

1907 г. *Америка. Ли де Форест (Lee de Forest) разработал усилитель на лампе с тремя электродами (триод).*

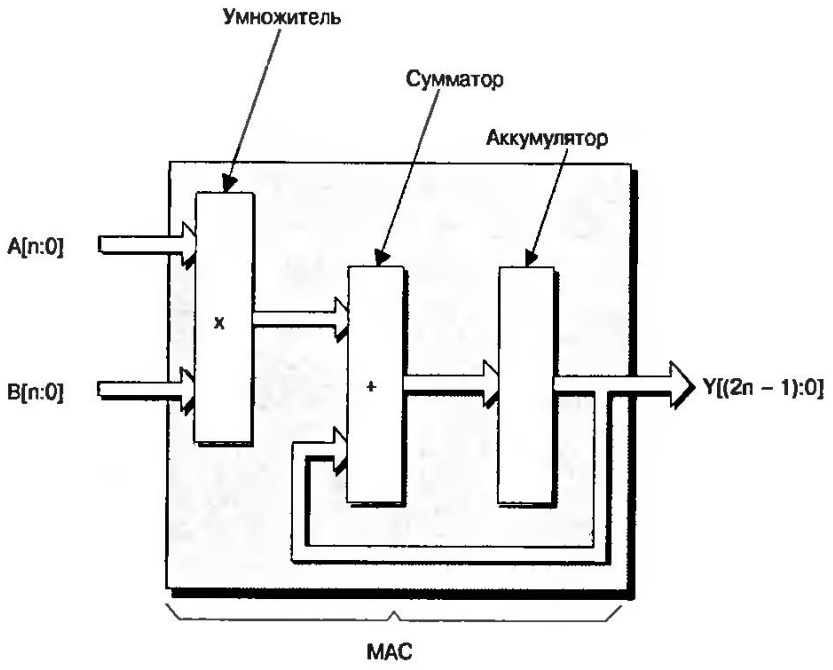

Рис. 12.6. Формирование функции MAC

Последовательно расположенные умножитель, сумматор и регистр, как показано на **Рис. 12.5**, представляют собой классический пример функции умножения с накоплением. Если ПЛИС содержит только встроенные умножители, эту функцию необходимо реализовывать с помощью комбинации умножителя с сумматором, сформированным из нескольких логических блоков, при этом результат будет сохраняться в блоке ОЗУ или в нескольких ячейках распределённого ОЗУ. Реализовать такую функцию будет намного проще, если ПЛИС содержит встроенные сумматоры, а некоторые из этих устройств предоставляют полностью готовые встроенные функции умножения с накоплением.

Разработка ПЛИС для ЦОС

Меня просто бросало в дрожь, когда я писал эти слова. Это могло быть от предвкушения, а может быть от предчувствия. Дело в том, что в то время, когда писалась эта книга, идея использовать ПЛИС для цифровой обработки сигналов была сравнительно новой. Это означало отсутствие устоявшихся технологий проектирования устройств на основе ПЛИС. Такая ситуация способствовала тому, что каждый стремился разработать собственный подход, и поэтому, какой бы подход не выбрал пользователь, он мог тут же обнаружить еще один.

Специализированные языки

Размеры и сложность электронных систем увеличиваются с каждым днём. Чтобы отслеживать эту проблему и поддерживать или, в большинстве случаев, увеличивать производительность, необходимо сохранить тенденцию повышения уровня абстракции для описания функциональности устройства и его проверки.

По этой причине схемотехнические методы проектирования, описанные в гл. 8, были вытеснены RTL-описаниями, выполненными на языках VHDL или Verilog (гл. 9). Аналогично методы проектирования

на основе языка C, которые рассматривались в гл. 11, больше подходят для быстрого и простого описания устройств, а также упрощают архитектурный анализ и исследование возможных вариантов устройств.

Специалисты из области цифровой обработки сигналов, системные инженеры и инженеры-разработчики могут значительно повысить производительность своей работы с помощью *специализированных языков*, которые обеспечивают более быстрые способы представления специфических задач, чем языки общего назначения, такие как C/C++ или SystemC.

Примером такого языка может служить MATLAB, который позволяет инженерам цифровой обработки сигналов описывать функции преобразования сигнала, такие как быстрое преобразование Фурье (БПФ), которые потенциально могут занимать всю микросхему ПЛИС с помощью одной строки кода[1] вида

```
y = ffx( x );
```

Входные воздействия для системы моделирования MATLAB могут поступать из одной или нескольких математических функций, таких как генератор синусоиды, или формироваться из реальных данных в виде аудио- или видеофайла.

Название MATLAB относится как к языку, так и к средствам моделирования алгоритмического уровня. Чтобы избежать путаницы, в общем случае различают *M-код*, что значит MATLAB-код и *M-файл*[2], т. е. файл, содержащий код MATLAB. Некоторые разработчики иногда употребляют сокращенно — M-язык, но этот термин не поддерживается сотрудниками компании The MathWorks.

Кроме сложных операторов преобразования, таких как БПФ, также существуют менее сложные, например сумматоры, блоки вычитания, умножители, логические операторы, матричные вычисления и другие. При необходимости из этих простых операторов могут формироваться более сложные операторы преобразования сигналов, например БПФ. Результат работы каждой функции преобразования сигнала может быть использован как аргумент, т. е. входное значение, для одной или нескольких последующих функций преобразования и так далее до тех пор, пока вся система не будет представлена на высшем уровне абстракции.

Важное значение имеет тот факт, что такие представления на системном уровне изначально не предполагают способа реализации устройства. Так, например, при использовании ядра цифрового сигнального процессора (ЦСП) это может значить, что вся функция реализуется программно. Разработчики системы могут разделить устройство и таким образом, что одни функции будут реализоваться программно, а другие, критичные по производительности, аппаратно, используя заказные микросхемы или ПЛИС структуры. В этом случае инженерам необходим доступ к системам проектирования как аппаратуры, так и программного обеспечения (*см.* гл. 13). Но в рамках этой главы мы будем рассматривать только аппаратные реализации.

---

[1] В этом примере точка с запятой не является обязательной. Она используется при необходимости подавить вывод на дисплей.

[2] M-файлы могут содержать сценарии либо преобразования, либо и то и другое. Кроме того, одни M-файлы могут иерархически включать другие M-файлы. Первичный (наивысшего уровня) M-файл обычно содержит сценарий, который определяет процесс моделирования. Этот сценарий может дать пользователю только общую информацию об используемых коэффициентах фильтрации, именах файлов с входными воздействиями и так далее, а затем вызывать другие M-файлы для определения других необходимых параметров.

Системы проектирования и моделирования на системном уровне

Средства проектирования и моделирования системного уровня концептуально находятся на более высоком уровне, чем специализированные языки. Прекрасным примером среды проектирования и моделирования такого рода может служить пакет Simulink компании The MathWorks. Порой может казаться, что Simulink является просто графическим пользовательским интерфейсом к пакету MATLAB. На самом деле Simulink является независимым приложением динамического моделирования, которое *работает с* программой MATLAB.

При использовании Simulink процесс разработки начинается обычно с создания графической структурной схемы системы, на которой схематически изображаются функциональные блоки и соединения между ними. При этом каждый блок может быть определён пользователем или может находиться в одной из библиотек, поставляемых вместе с Simulink. Эти библиотеки включают наборы для цифровой обработки сигналов, телекоммуникации, управления. При работе с определяемыми пользователем блоками, можно щёлкнуть мышью на этом блоке и отобразить его содержимое в виде новой структурной схемы. Можно создавать блоки, содержащие функции MATLAB, M-код, C/C++, FORTRAN... Этот список может быть продолжен.

Разработанный в 1962 году язык Фортран (Fortran), что в переводе означает транслятор формул, был одним из первых высокоуровневых языков программирования.

После описания принципа действия устройства для моделирования и проверки функциональности можно использовать средства Simulink. Так же как и при использовании пакета MATLAB, задающее воздействие для модели в Simulink может генерироваться одной или несколькими математическими функциями, например генераторами синусоиды, или могут использоваться реальные данные в виде аудио или видеофайлов. Во многих случаях используются оба вида воздействий, например реальные данные могут дополняться псевдослучайными шумами, генерируемыми стандартным блоком из библиотеки Simulink.

Необходимо иметь в виду, что на этом этапе не существует правила «все сразу и побыстрее». Одни инженеры предпочитают использовать MATLAB в качестве средства начальной разработки, другие выбирают Simulink, что бывает крайне редко. Кое-кто считает, что это предпочтение зависит от опыта пользователя, т. е. занимался ли он разработкой программных средств ЦОС или разработкой заказных микросхем и ПЛИС. Однако есть и такие, которые считают, что все это чушь. На самом деле опыт в данном случае не причем. Но даже если и причем, причины, по которым пользователи делают свой выбор, просто ничто по сравнению с тем, что их ожидает.

Модели с фиксированной и плавающей точкой

Независимо от того, какое из средств будет выбрано, Simulink или MATLAB, или подобный продукт другого производителя, на начальной стадии проектирования, первая версия модели системы почти всегда использует *числа с плавающей точкой*. В десятичной системе счисления это соответствует числам вида 1.235×10^3 (другими словами дробное число, умноженное на 10 в некоторой степени). В таких приложениях, как MATLAB, эквивалентное для этой дроби двоичное значение представляется внутри компьютера в виде стандарта IEEE для чисел с плавающей точкой двойной точности.

Этот тип чисел с плавающей точкой обладает определенными преимуществами, так как обеспечивает высокую точность в огромном динамическом диапазоне. Однако реализация вычислений с плавающей точкой этого типа данных в специализированных устройствах на базе ПЛИС или заказных микросхем требует огромного количества ресурсов и по меркам аппаратного обеспечения является чрезвычайно медленной. Поэтому на некоторой стадии проектирования устройство должно быть переведено на использование чисел с *фиксированной точкой*, т. е. числа с фиксированным количеством битов для представления целой и дробной частей. Этот процесс обычно называется *квантованием*.

1907 г. *Ли де Форест (Lee de Forest) начал постоянное вещание музыкальной радиостанции.*

Процесс квантования целиком и полностью зависит от типа системы и алгоритма, и может потребоваться немалое количество экспериментов для определения оптимального баланса между использованием наименьшего количества битов для представления чисел тем самым, уменьшая количество требуемых ресурсов и увеличивая скорость вычисления, и сохранением точности вычислений. Это может натолкнуть на мысль о компромиссной величине погрешности, которую разработчик допускает для данного количества битов. В некоторых случаях конструкторы могут провести несколько дней, решая: «Следует ли использовать 14, 15 или 16 бит для представления этих данных?». И специально для поднятия настроения можно попробовать использовать различное количество битов для представления данных на разных участках алгоритма или системы.

Ещё больше «удовольствия» можно получить при переходе от чисел с плавающей точкой к данным с фиксированной точкой на более высоком системном или алгоритмическом уровне проектирования, или на более низком уровне кода C/C++.

При работе в окружении MATLAB эти преобразования могут быть выполнены путем прогона сигналов с плавающей точкой через специальные преобразовательные функции, называемые *дискретизаторами*. Аналогично при работе в окружении Simulink преобразования могут быть выполнены путем прогона сигналов с плавающей точкой через специальные блоки с фиксированной точкой.

Перевод из системного/алгоритмического уровня к RTL. Ручной способ

Во времена, когда писалась эта книга, многие инженеры, занятые в сфере проектирования устройств цифровой обработки сигналов, начинали работу в среде MATLAB или аналогичной с проверки алгоритмов и оценки на системном уровне параметров устройства, используя числа с плавающей точкой. Часто описание устройства включало и промежуточные этапы, на которых создавались модели на языке C/C++ с фиксированной точкой для использования в средствах быстрого моделирования или проверки. На этом этапе многие конструкторы переходили напрямую к созданию вручную кода RTL с фиксированной точкой, используя при этом языки VHDL или Verilog (**Рис. 12.7, а**). Однако разработчики могли сначала перейти от плавающей точки к числам с фиксированной точкой на системном (алгоритмическом) уровне и лишь только после этого переходить к написанию кода на VHDL или Verilog (**Рис. 12.7, б**).

Конечно, при такой последовательности возникает ряд проблем, так как налицо существенное концептуальное и представительское различие между работой инженеров-системщиков на системном (ал-

1908 г. *Чарльз Фредрик Кросс (Charles Fredrick Cross) изобрёл целофан.*

1909 г. *Дженерал Электрик представила миру первый тостер.*

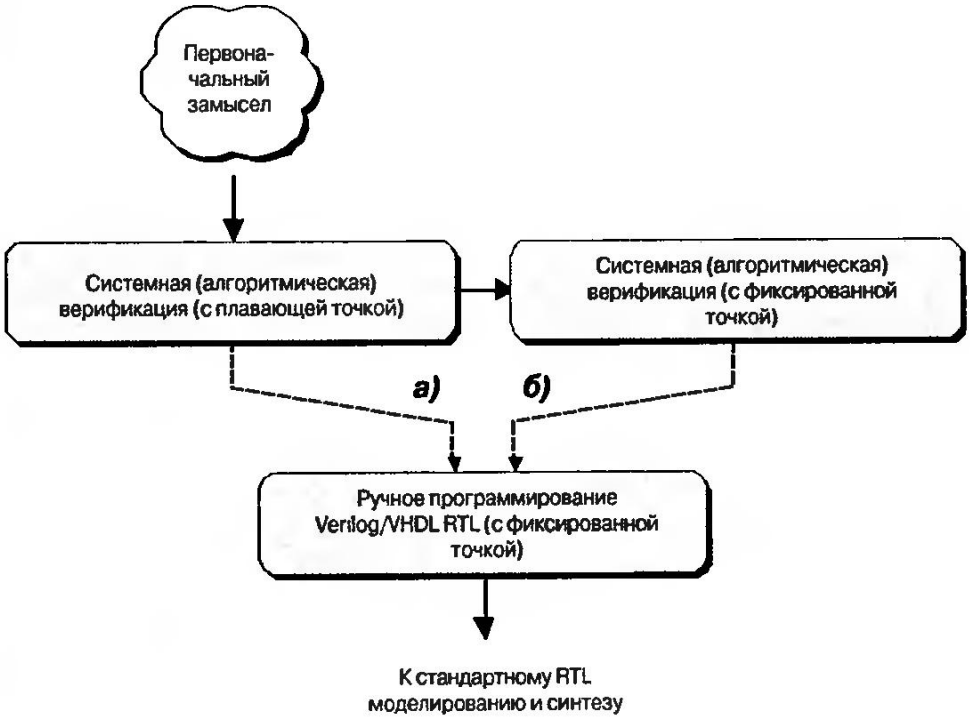

Рис. 12.7. Ручное создание RTL-кода

горитмическом) уровне и работой инженеров-проектировщиков на уровне регистровых передач в среде VHDL или Verilog.

Поскольку системный (алгоритмический) уровень и уровень регистровых передач (RTL) существенно отличаются, ручной перенос кода из одной области в другую занимает массу времени и имеет такие нежелательные последствия, как наличие ошибок. Итоговое RTL-представление устройства зависит также от конкретной реализации, поскольку создание оптимального устройства на ПЛИС требует иного стиля программирования, чем при использовании заказной микросхемы.

Что же касается модификации вручную и перепроверки RTL-кода для формирования ряда оценок возможных вариантов микроархитектурных реализаций, то они требует чрезвычайно много времени. Кстати, такие оценки могут включать формирование определённых операций в параллельном и последовательном режимах: с применением и без применения канала передачи данных, с распределёнными ресурсами, например две операции используют один умножитель, и со специализированными средствами и т. д.

Аналогично, если в ходе проектирования в начальный замысел были внесены некоторые изменения, их можно довольно просто реализовать и оценить на системном (алгоритмическом) уровне представления, но последующая ручная свёртка этих изменений в RTL-код может оказаться трудоёмкой и потребовать много времени.

После создания RTL-представления устройства можно воспользоваться низкоуровневыми средствами логического синтеза, которые были рассмотрены в гл. 9.

Перевод из системного/алгоритмического уровня к RTL. Автоматический способ

Как отмечалось в прошлом подразделе, ручная трансляция из системного (алгоритмического) уровня в RTL код требует много времени и способствует возникновению ошибок. Существуют альтернативные методы, в частности некоторые средства проектирования системного

(алгоритмического) уровня позволяют производить непосредственную трансляцию в код VHDL или Verilog (**Рис. 12.8**).

Как обычно, проектирование на системном (алгоритмическом) уровне начинается с описания устройства с применением чисел с плавающей точкой. Согласно одному из подходов к проектированию, система (среда) проектирования используется также для перевода этих представлений в их эквиваленты с фиксированной точкой, а затем выполняется автоматическая генерация эквивалентного RTL кода на языке VHDL или Verilog[1] (**Рис. 12.8, а**).

1909 г. *Лео Бакеланд (Leo Baekeland) запатентовал искусственный пластик и назвал его Бакелит.*

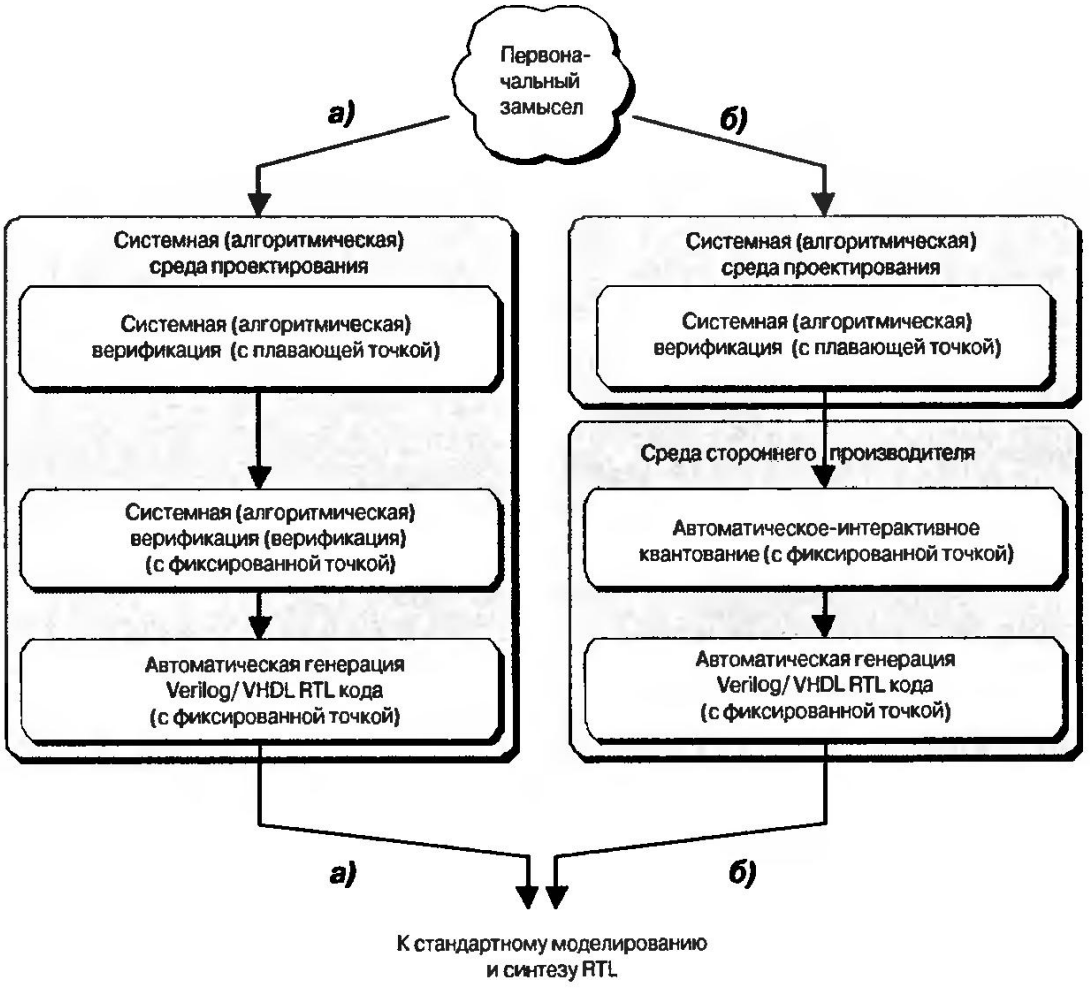

Рис. 12.8. Прямой переход к RTL

В процессе проектирования можно также использовать среду разработки сторонних поставщиков для представления устройства на системном уровне с использованием чисел с плавающей точкой, автоматического перевода их в числа с фиксированной точкой и автоматического создания эквивалентного RTL кода на языке VHDL или Verilog (**Рис. 12.8, б**)[2].

Как и раньше, после создания RTL-представления устройства можно воспользоваться низкоуровневыми средствами логического синтеза, которые были рассмотрены в гл. 9.

---

[1] Хороший пример такой среды конструирования предлагает компания Elanix Inc. (www.elanix.com).

[2] Хороший пример такой среды проектирования предлагает компания AccelChip Inc. (www.accelchop.com). Программный пакет этой компании принимает M-файлы с плавающей точкой среды MATLAB, формирует их эквиваленты с фиксированной точкой для верификации, и затем использует эти файлы для автоматического создания RTL-кода.

Перевод из системного/алгоритмического уровня в код C/C++ и другие

В последнее время наметилась тенденция использовать для решения проблем, связанных с анализом устройства на уровне регистровых передач (RTL), так называемый *метод ступенек*. Этот метод подразумевает преобразование из системного (алгоритмического) представления устройства в некоторое представление кода C/C++, которое затем переводится в код RTL. Одна из причин такого двойного перехода заключается в том, что большинство разработчиков, проектирующих системы цифровой обработки сигналов, используют представления на языке C/C++ в качестве *золотой*, т. е. эталонной, модели, причём такие модели являются бесплатными, в чём весьма заинтересованы разработчики, работающие с RTL.

Разумеется, сначала необходимо принять решение, где и когда в процессе проектирования перейти от моделей с плавающей точкой к описаниям с фиксированной точкой (**Рис. 12.9**).

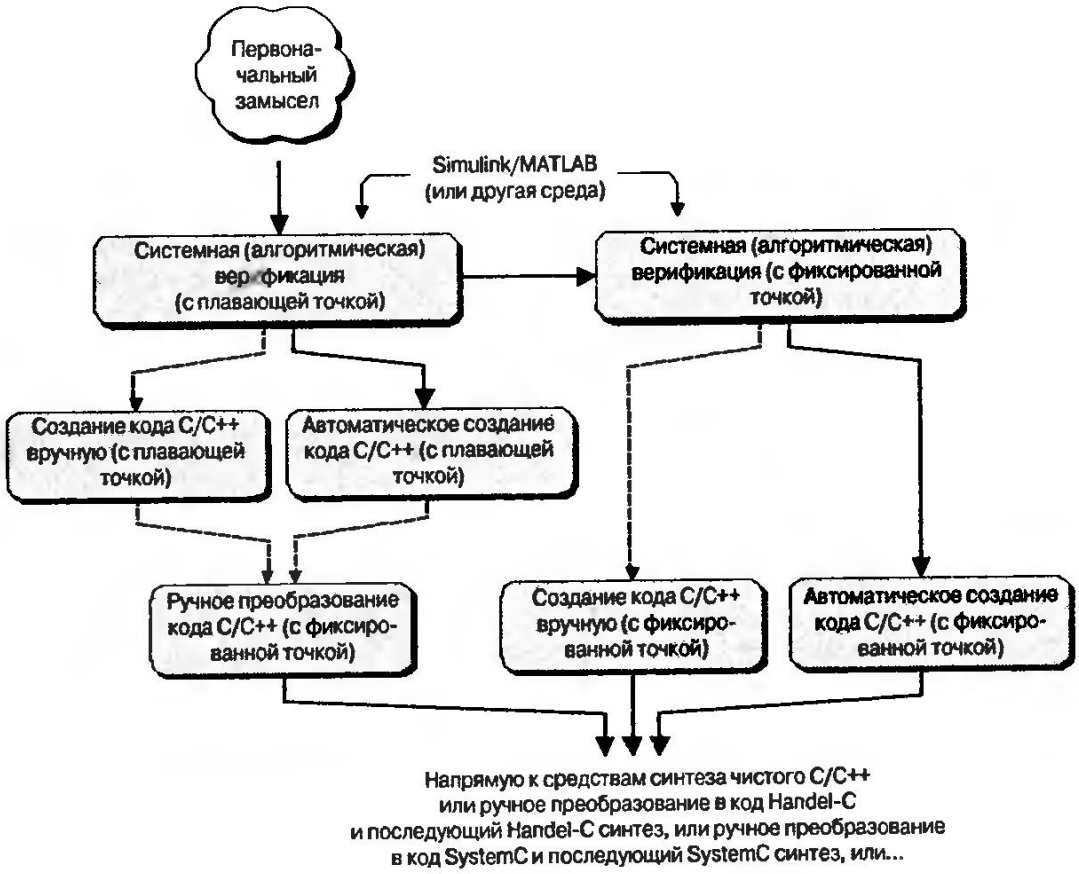

Рис. 12.9. Переход от представления с плавающей точкой к представлению с фиксированной точкой

Такой переход выглядит устрашающе и это притом, что на **Рис. 12.9** представлена только часть возможных вариантов. Например, при кодировании вручную вместо первого этапа разработки кода на C/C++ и постепенного его перевода в Handel-C или SystemC можно сразу приступать к разработке кода на этих языках. Но при этом следует помнить, что после того, как будет получено описание устройства с фиксированной точкой в одной из версий языка C/C++, можно использовать один из низкоуровневых методов C/C++ проектирования, которые были описаны в гл. 11. Особый интерес вызывает версия чистого, без учета временных параметров, C/C++, используемого пакетом Precision C компании Mentor.

Иногда довольно трудно классифицировать другие методы преобразования кода. На практике может встречаться один из следующих подходов:

а) Можно довольно легко вручную перевести код из MATLAB в C/C++, потратив на это от нескольких часов до нескольких дней. Автоматическая трансляция, как правило, используется только для систем моделирования или кода ЦОС в зависимости от того, насколько критичны требования к производительности.

б) Довольно просто можно осуществить ручное исследование процесса квантования, особенно для инженеров-системщиков (автоматическое квантование применяется не так часто). Многие разработчики рассчитывают также на результаты анализа шумов, который может помочь в управлении процессом квантования.

в) Ручное преобразование из кода MATLAB или C/C++ в код RTL представляет собой довольно сложную задачу, которая может затянуться на недели или месяцы. Автоматизация в этой области играет огромную роль при условии достижения качественного результата.

г) Средства проектирования, основанные на пакетах MATLAB/Simulink, обычно плохо справляются с задачей применения ядер-блоков интеллектуальной собственности (ядер IP) при необходимости реализовать действительно оригинальную концепцию.

Блоки интеллектуальной собственности

Не всё так просто устроено в этом мире, так как во всяком деле найдётся ещё один или несколько вариантов его реализации. Например, можно создать библиотеку функциональных блоков ЦОС на системном (алгоритмическом) уровне абстракции совместно с такой же эквивалентной библиотекой RTL для языков VHDL или Verilog.

Идея заключается в том, что можно описать и проверить устройство, используя иерархию функциональных блоков на системном (алгоритмическом) уровне абстракции. Насладившись вдоволь полученным устройством, можно создать таблицу соединений RTL-блоков и использовать её для передачи средствам низкоуровневого моделирования и синтеза. Эти блоки должны быть параметризованы на всех уровнях абстракции, в первую очередь для того, чтобы позволить точно определить такие параметры, как, например, ширину шины и другие.

Возможно и другое решение, которое заключается в том, что крупные поставщики ПЛИС обычно предлагают генераторы ядер интеллектуальной собственности (IP). В данном контексте понятие *ядро* обозначает блоки, которые выполняют специфичные логические функции; этот термин не относится к микропроцессорным или ЦСП ядрам. В некоторых случаях эти генераторы ядер могут быть интегрированы в среду системного (алгоритмического) уровня проектирования. Это значит, что можно создать устройство, состоящее из этих блоков в среде системного (алгоритмического) уровня проектирования, определить все параметры этих блоков и произвести системную (алгоритмическую) проверку (верификацию).

Позже, когда уже можно будет танцевать рок-н-ролл, генератор ядер автоматически сгенерирует аппаратные модели каждого блока[1]. Причем модели системного (алгоритмического) уровня и аппаратные модели,

---

[1] Хорошим примером реализации такого подхода может служить интеграция в Simulink утилиты System Generator компании Xilinx (www.xilinx.com).

1909 г. *Маркони получил Нобелевскую премию по физике за вклад в развитие телеграфии.*

сформированные генераторами ядер, абсолютно идентичны по всем параметрам. В некоторых случаях аппаратные блоки будут генерироваться как синтезируемые на RTL в формате VHDL или Verilog. Они могут быть представлены также как ядра на уровне таблиц соответствия/КЛБ, тем самым максимально используя ресурсы выбранной ПЛИС.

Существенный недостаток, присущий этому методу, заключается в том, что по своей сути блоки интеллектуальной собственности основаны на аппаратной микроархитектуре. Это значит, что возможность создания устройства, предназначенного для решения узкоспецифичных задач, будет до некоторой степени ограничена. Поэтому, разрабатывая что-либо с использованием блоков интеллектуальной собственности, можно достичь стадии реализации быстрее и с меньшим риском, но, в то же время результат может оказаться не вполне оптимальным с точки зрения занимаемой площади, производительности и потребляемой мощности по сравнению с заказным вариантом.

Не забудьте о средствах тестирования!

При продаже средств проектирования систем цифровой обработки сигналов продавцы часто забывают упомянуть о средствах тестирования. Например, допустим, что метод проектирования предполагает ручной перевод их системных (алгоритмических) описаний устройства в код RTL. В этом случае этот же подход используется и для средств тестирования. Во многих случаях это может оказаться нетривиальной задачей, реализация которой может занять несколько дней или недель.

Или, скажем, метод проектирования предполагает ручной перевод системного (алгоритмического) описания устройтства с плавающей точкой в код C/C++ также с плавающей точкой, после чего при желании может быть выполнена проверка этого нового описания устройства. Затем можно взять этот C/C++ код с плавающей точкой и вручную перевести его в C/C++ код с фиксированной точкой и в этом месте вновь произвести тестирование устройства. После этого можно взять полученный C/C++ код с фиксированной точкой и (будем надеяться) автоматически синтезировать эквивалентное RTL представление, и в этом месте... ну... смысл ясен[1].

Проблема заключается в том, что на каждом этапе будут производиться одни и те же действия с наборами тестов[2] (если только не делается что-то хитроумное, как описано в следующем (и последнем — ура!) разделе.

Смешанные системы проектирования: ЦОС и VHDL/Verilog

В предыдущей главе мы упомянули о том, что ряд компаний, специализирующихся на САПР электронных систем, могут предоставлять системы проектирования и проверки устройств смешанного уровня, которые могут поддерживать совместное моделирование модулей,

---

[1] Не надо смеяться, потому что я знаю одну очень крупную компанию, которая поступает именно таким образом!

[2] Что касается процесса перевода кода C/C++ на уровень RTL, даже если вы располагаете алгоритмом синтеза RTL из C/C++, ваш набор тестов, как правило, будет содержать языковые конструкции, неподдающиеся синтезу, т. е. придётся вернуться назад и выполнить эти операции вручную.

описанных на разных уровнях абстракции. Например, можно начать проектирование в графическом блочном редакторе, который отображает основные функциональные модули устройства, где содержимое каждого блока может быть описано с помощью языков:

- VHDL;
- Verilog;
- SystemVerilog;
- SystemC;
- Handel-C;
- Чистый C/C++.

В этом случае на верхнем уровне устройство может быть описано с помощью традиционного HDL, который вызывает субмодули, описанные в различных версиях языка HDL и в одной или нескольких разновидностях языка C/C++. На верхнем уровне устройство может быть также описано одной из разновидностей языка C/C++, который вызывает субмодули, написанные на других языках.

Недавно появились системы проектирования, объединяющие системный (алгоритмический) уровень и уровень реализации. Принцип работы таких систем зависит от того, кто на них работает и чего он хочет добиться. Например, разработчики, проектируя на системном (алгоритмическом) уровне, т. е. в MATLAB, могут решить заменить один или несколько блоков эквивалентными реализациями, выполненными в VHDL или Verilog на уровне регистровых передач. Или наоборот, инженеры, работая в VHDL или Verilog на уровне абстракции RTL, могут решить вызвать один или несколько блоков, описанных на системном (алгоритмическом) уровне.

Оба этих случая требуют совместного моделирования в среде разработки как системного уровня, так и VHDL/Verilog. Главное отличие заключается в том, кто кого вызывает. Конечно, это звучит просто, особенно если произнести всё быстро, но существует огромное количество факторов, которые необходимо учитывать. Например, синхронизация времени между двумя представлениями или определение, как различные типы сигналов переходят из одного представления в другое и обратно.

Это именно тот случай, когда следует рассматривать любые заранее подготовленные демонстрации с определенной долей недоверия. Если вы планируете реализовать подобные задачи, необходимо сесть вместе с инженером компании-поставщика и проработать свой собственный пример от корки до корки, т. е. от начала до конца. Можете назвать меня старым циником, но я бы посоветовал вам позволить инженеру компании поучить вас, при этом вы должны твёрдо держать клавиатуру и мышку в своих руках. Уверяю вас, вы будете потрясены тем, как много может произойти за несколько секунд, стоит вам только отвернуться от монитора в ответ на старую уловку: «Какое несчастье! Вы видели, что случилось за окном?»

1909 г. *Радиосигнал бедствия (SOS) спас 1900 жизней после крушения двух кораблей.*

1910 г. *Америка. На почтовых линиях связи между Нью-Йорком и Бостоном установлены телетайпы.*