ГЛАВА 25

ПРОЕКТИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ОТКРЫТОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Как открыть магазин по продаже устройств на основе ПЛИС без крупных вложений

Уверен, что вам не часто приходилось сталкиваться с проектными компаниями, организованными по принципу «два парня в гараже», которые занимались бы проектированием заказных микросхем. И это не удивительно, так как стоимость средств проектирования, необходимых для разработки устройств этого класса, ужасающе высокая и составляет от 100 000$ и выше. Непреодолимым препятствием также является тот факт, что для организации производства микросхем объем необходимых инвестиций достигает миллионов долларов.

Хочу заметить, что моя задача не сводится к пропаганде менее поддерживаемых средств. Для бюджетных проектов предпочтительнее использовать недорогие средства проектирования от поставщиков ПЛИС. Мощные средства от крупных или специализированных поставщиков САПР будут предпочтительны для использования в больших по размеру и сложности устройствах. Однако если вы пытаетесь создать «мастерскую» по производству и продаже устройств на основе ПЛИС на дому без крупных капиталовложений (или вовсе без вложений), то в этом случае, я думаю, представленные здесь средства с открытым кодом, будут для вас весьма интересными.

В отличие от заказных микросхем, сочетание современной ПЛИС, а также последние достижения в области открытых САПР электронного проектирования и блоков интеллектуальной собственности сводит стоимость стартового набора для проектирования устройств на основе ПЛИС почти к нулю. Именно поэтому «магазин» по продаже устройств на основе ПЛИС могут открыть как выпускники колледжа, так и профессионалы.

Чтобы проектная компания была успешной, кроме наличия знаний в области проектирования цифровых логических устройств, требуется ряд других фундаментальных составляющих, в число которых входят:

- базовая платформа;
- среда верификации (тестирования);
- средства формальной верификации (необязательная составляющая);
- доступ к блокам интеллектуальной собственности общего назначения;
- средства синтеза и реализации;
- макетная плата (необязательная составляющая).

Базовая платформа: Linux

Созданная шведским инженером Линусом Торвальдсом (Linus Torvalds) в начале 1990-х годов, операционная система Linux очень быстро стала основной платформой для проектирования устройств на основе ПЛИС и заказных микросхем. Несмотря на то что большинс-

тво средств синтеза и реализации ПЛИС изначально работали под управлением операционной системы Microsoft Windows®, большинство из них переносятся или уже перенесены под Linux.

Linux и *GNU* (GNU — проект свободного распространения программного обеспечения. — *Прим. пер.*) поддерживают многочисленные средства аппаратной и программной разработки. Некоторые, наиболее распространённые из них (не в порядке по какому-то правилу, а так как расположил их автор) кратко рассмотрены ниже:

- *gcc*. Язык C остаётся наиболее быстрым языком моделирования и верификации устройств. Если ваше устройство является настолько большим, что средства моделирования HDL (Verilog или VHDL) с ним не справляются, то вы можете попробовать создать потактную модель на языке C и откомпилировать её с помощью компилятора с открытым исходным кодом *gcc* (аббревиатура от GNU C Compiler).
- *make*. Утилита *make* используется для автоматизации процесса построения. В контексте аппаратного обеспечения понятие «построение» может относиться к моделированию, созданию RTL-кода или логическому синтезу. Для того чтобы запустить утилиту *make* и начать процесс построения кода, вы должны в специальном файле, который называется *makefile*, описать все участвующие в проекте файлы, их зависимость от других файлов, а также взаимоотношения между собой.
- *gvim*. Самым распространённым текстовым редактором для операционной системы Unix, пожалуй, является *VI*, название которого образовано от слов *«visual interface»*, что значит *«визуальный интерфейс»*. Утилита под названием *vim* представляет собой расширенную версию *vi*, а её аналог с графическим интерфейсом называется *gvim*. В *gvim* встроены средства расширения редактора *vi*, которые позволяют графически выделять (подсвечивать) названия функций и макросов. Также в этот редактор встроена поддержка языков Verilog или VHDL, что делает его сверхбыстрым и мощным средством описания электронных систем.
- *EMACS* (от «Editing MACroS» — редактирование макросов) задумывался сообществом хакеров[1] как совершенный редактор, и представляет собой программируемый текстовый редактор со встроенным интерпретатором языка LISP. Отличается большей мощностью и сложностью, чем *VI*, а также содержит модули позволяющие использовать его для создания Verilog или VHDL-описаний.
- *cvs*. Система *CVS (Concurrent Version System — система согласования версий)* является основным средством контроля версий программного обеспечения с открытым исходным кодом (открытого программного обеспечения), которую могут использовать все желающие — от индивидуальных разработчиков до больших дистрибьюторских компаний. *CVS* поддерживает ветвление, многопользовательский режим и удалённое взаимодействие, а также сохраняет историю всех изменений, произведенных в

Официально аббревиатура LISP произошла от слов List Processor (язык обработки списков), хотя критики утверждают, что LISP следует расшифровывать как «Lots of Irritating, Superfluous Parentheses» (множество идиотских ненужных скобок).

---

[1] В программистском сообществе этот термин возник в конце 50-х годов прошлого века и обозначал людей, поддерживающих принципы открытого обмена программами и проектирования аппаратуры. В это сообщество входил и Ричард Столман (Richard Stallman), написавший редактор EMACS. Позже, после выхода в 1983 г. фильма «War Games», слово «хакер» стало ассоциироваться со словом «кракер» (cracker) и людьми, злонамеренно взламывающими программы и проникающими в чужие компьютеры. — *Прим. пер.*

подконтрольных ему папках (дереве каталогов) и файлах. Используя эту историю, *CVS* может восстанавливать прошлые состояния дерева каталогов и показывать, кто, когда и какие изменения в него вносил. Таким образом, если вы случайно испортили свой RTL-код или намерены пересинтезировать версию вашего устройства трёхмесячной давности, то *CVS* поможет легко решить эти вопросы.

- *PERL*. Языки сценариев очень часто используются для реализации одноразовых программных задач и для создания прототипов. В области разработки электронных устройств они также применяются для связывания отдельных операций в процессе проектирования друг с другом при помощи управления режимами работы отдельных средств проектирования и организации передачи данных между ними. Язык *PERL (Practical Extraction and Report Language — практический язык отчётов и извлечений)* исторически является одним из наиболее широко используемых языков сценариев. Разработанный Ларри Уоллом (Larry Wall) *PERL* в шутку называют «швейцарским армейским ножом» (универсальным языком) для программирования под ОС Unix (и Linux), и многие отдельные операции при проектировании аппаратуры склеиваются в единую технологическую цепочку с помощью сценариев, написанных на языке *PERL*.
- *Python*. Возможно, более мощный, чем *PERL*, язык *Python* является «всеобъемлющим» языком сценариев, который перерос во вполне законченный язык программирования. Язык *Python* разработал Гвидо ван Россум (Guido Van Rossum) в 1990 году и назвал его в честь Монти Пайтона (Monty Python), поклонником которого он являлся[1]. *Python* может использоваться для решения разнообразных задач, начиная от склейки между собой отдельных этапов проектирования, и до высокоуровневого моделирования, верификации и создания пользовательских средств САПР (*см.* также дополнительное обсуждение языка *Python* ниже в этой главе).
- *diff*. Относительно простая, но чрезвычайно полезная утилита *diff* используется для быстрого сравнения файлов исходного кода, обнаружения отличий между ними и формирования отчёта об этих отличиях.
- *grep*. Поддерживая *глобальный поиск по регулярным выражениям и отображая строки, которые им соответствуют*, утилита *grep* используется для быстрого поиска файла или группы файлов по запросу в виде отдельной текстовой строки или в виде шаблона.
- *OpenSSL*. Независимо от того, являетесь ли вы большой или малой компанией, вам следует обеспечить сохранность ваших блоков интеллектуальной собственности (IP). Вопросы сохранности возникают, когда вы собираетесь передавать IP через локальную сеть или через Интернет своим сотрудникам или покупателям. В этом случае предварительно необходимо зашифровать передаваемые данные, для чего можно воспользоваться открытым программным обеспечением под названием *OpenSSL*. Этот пакет осуществляет полноценную поддержку протоколов *SSL (Secure Sockets Layer — протокол безопасных соединений)* и *TLS (Transport Layer Security — защита транспортного уровня)*,

---

[1] Гвидо ван Россум был поклонником комедийного сериала BBC «Воздушный цирк Монти Пайтона», в честь которого он и назвал свой язык (а не в честь подсемейства змей семейства ложноногих — питонов). — *Прим. пер.*

1971 г. *Америка. Тед Хофф (Ted Hoff) разработал (а компания Intel реализовала) первый компьютер-на-кристалле, известный как микропроцессор 4004.*

1971 г. *Америка. Компания CTC создаёт компьютер Datapoint 2200.*

причём качество его работы не уступает специализированным промышленным криптографическим библиотекам.

- *OpenSSH.* Как быть, если ваша команда разработчиков разбросана по всему миру? В этом случае поможет утилита *ssh (Secure SHell — защищенная оболочка)*, которая представляет собой программу, предназначенную для регистрации на удалённой машине (компьютере) и выполнения на ней различных команд. При этом используются защищённое зашифрованное соединение между двумя хостами без установления доверительных отношений через открытую сеть. Утилита OpenSSH представляет собой открытое программное обеспечение, и позволяет шифровать весь трафик (включая пароли) для предотвращения перехвата данных и других сетевых атак. Также OpenSSH поддерживает различные аутентификационные методы и возможности установления защищённых туннелей.
- *tar, gzip, bzip2.* Эти утилиты используются для сжатия и архивирования информации.

Приобретение ОС Linux

До недавнего времени лидерами по распространению ОС Linux были компании Red Hat (www.redhat.com) и MandrakeSoft (www.mandrakesoft.com). Однако в наши дни большой популярностью пользуется дистрибутив компании Gentoo Linux™ (www.gentoo.org). В Gentoo встроена уникальная система распределения пакетов, которая автоматически скачивает, компилирует и устанавливает пакеты на Linux машины. Хотите установить Icarus Verilog? Тогда просто введите

```
$ emerge iverilog
```

и через несколько минут вы обнаружите Icarus в своей системе полностью готовым к работе!

Среда верификации

Вы можете придерживаться другой точки зрения, но многие инженеры считают средства верификации (тестирования) наиболее важной частью в проектировании электронных устройств. Можно усердно работать за клавиатурой и разрабатывать HDL-код, но именно тестирование предоставляет проектировщику обратную связь, с помощью которой он может довести устройство до правильной реализации.

Icarus Verilog

Наиболее широко распространённым открытым средством верификации является компилятор языка Verilog под названием Icarus (http://icarus.com/eda/verilog). В начальном виде Icarus компилирует исходный код устройсва, написанный на языке Verilog, в исполняемый код, который применяется в качестве модели устройства. Честно говоря, Icarus изначально использовался в качестве системы событийного моделирования, но он также поддерживает основы логического синтеза для ПЛИС компании Xilinx.

Несмотря на то что Verilog является сложным языком, Стивен Уильямс (автор пакета Icarus) выполнил блестящую работу по его реализации. Язык Icarus Verilog по распространению и производительности фактически превзошёл некоторые коммерческие системы моделирования.

Dinotrace и GTKWave

Icarus Verilog, рассмотренный выше, полностью является средством командной строки. (Средства командной строки широко распространены в ОС Unix и Linux, так как они легко взаимодействуют с помощью make-файла.)

Icarus не поддерживает графический пользовательский интерфейс (GUI) для отображения результатов моделирования. Вместо этого он может формировать файлы в промышленном формате VCD (value change dump — дамп изменения значений), которые впоследствии могут быть использованы отдельными приложениями для просмотра формы сигнала.

В отличие от Icarus, Dinotrace и GTKWave поддерживают графический пользовательский интерфейс (GUI) и могут использоваться для отображения результатов моделирования, представленных в формате VCD. Оба этих просмотрщика формы сигнала могут наблюдать за процессом моделирования, отслеживать отдельные элементы и осуществлять поиск по шаблону. Утилита Dinotrace (www.veripool.com/dinotrace) представляет собой законченную программу, но с ограниченной функциональностью. Для сравнения отметим, что GTKWave (www.cs.man.ac.uk/apt/tools/gtkwave) в начале своего развития представлял собой довольно сырой продукт, но в последнее время активно развивается.

1971 г. *Америка. Компания CTC создаёт компьютер Kenbak-1.*

Covered

При верификации устройства очень важно знать уровень кодового покрытия, чтобы быть уверенным, что ваши тестовые задающие воздействия (ваш тестовый вектор) охватывают все «тупиковые ситуации» в устройстве.

Утилита Covered (http://covered.sourceforge.net) осуществляет проверку кодового покрытия, написанного на языке Verilog, и применяется для оценки работы систем моделирования. Если быть более точным, то Covered для определения полноты функционального покрытия анализирует исходный код языка Verilog и файлы формата VCD, созданные системой моделирования Icarus Verilog.

В настоящее время Covered выполняет четыре типа измерений: анализ строк исходного кода, анализ переключения логических значений, комбинационный анализ и анализ конечных автоматов.

Verilator

В наши дни весьма актуальным стал вопрос о том, как управлять однокристальными системами, в которых аппаратная и программная составляющие встраиваются в один кристалл. Многие ПЛИС также содержат в своём составе аппаратные процессорные ядра или позволяют реализовывать программные микропроцессоры (*см.* также гл. 13).

Настоящие чудеса приходится творить при верификации аппаратных и программных частей однокристальных систем. Например, программа *Verilator* (www.veripool.com/verilator.html) преобразует код языка Verilog в потактную модель C/C++. Следует заметить, что возможность автоматически генерировать модели на языке C/C++ из RTL кода представляет собой довольно мощное средство отладки, которое поз-

Компания Tenison EDA разработала другой очень полезный инструмент под названием VTOC, который создаёт модели на языках C++ или SystemC из RTL-кода.

1971 г. *Первое прямое телефонное соединение Америки с Европой.*

воляет программной части системы интегрироваться с этой моделью для выполнения процедуры моделирования.

В дополнение к аппаратно-программной верификации системы, Verilator также используется для Verilog моделирования, так как моделирование с использованием потактных моделей языка C требует гораздо меньше времени, чем событийное HDL-моделирование. Всё, что вам для этого потребуется — это откомпилировать файл с C-кодом с помощью компилятора *gcc* (смотрите также раздел «Базовая платформа: Linux» выше в этой главе) и запустить его на выполнение[1].

Python

Python (www.python.org) является очень удобным высокоуровневым языком программирования и создания сценариев, также приобрел мировую известность благодаря своим возможностям быстрого достижения результатов. Разумеется, Python развивается как мощное средство для инженеров, проектирующих и тестирующих электронные устройства, в частности для решения задач системного моделирования, разработки тестовых устройств и общего управления устройством.

На практике многие проектные организации открыли для себя, что модели описания разрабатываемых ими устройств гораздо легче и быстрее создавать с помощью языка Python, чем использовать для этого языки Verilog или VHDL. Поэтому многие компании сначала разрабатывают модель устройства на Python'е, проверяют её с помощью системы моделирования и, затем, создают RTL-код, постоянно сравнивая его с этой «эталонной» моделью.

Для языка Python разработана специальная среда высокоуровневого системного моделирования под названием MyHDL (www.jandecaluwe.com/Tools/MyHDL/Overview.html), которая использует новые дополнения для этого языка (генераторы), предназначенные для эмуляции процесса параллельных вычислений. MyHDL также может соединяться с компилятором Icarus Verilog для создания смешанной среды моделирования Python/Verilog.

Формальная верификация

Как однажды сказал голландский математик и основатель многих компьютерных концепций Эдсгер Дейкстра (Edsger Wybe Dijkstra): «Тестирование программ может использоваться только для того, чтобы показать наличие ошибок в коде, но не для того, чтобы показать их отсутствие».

Хотя аппаратное моделирование по-прежнему остаётся основным способом системной проверки, убедиться в правильности работы устройства вы можете только с помощью средств формальной верификации (*см.* также гл. 19). В отличие от средств моделирования, формальная верификация математически доказывает, что реализация системы соответствует некоторой спецификации.

Существуют два основных типа *формальной верификации — автоматизированная формулировка логических выводов* и *верификация модели*. Верификация модели представляет собой метод анализа пространства состояния системы для проверки некоторых свойств, обычно называемых «утверждениями». Одной из разновидностей верификации моделей является проверка на эквивалентность, которая

---

[1] Заметим, что компилятор Icarus также поддерживает возможности генерирования кода на языке C.

сравнивает два представления системы (например, RTL и таблицу соединений вентилей). Для сравнения отметим, что автоматизированная формулировка логических выводов использует логику для доказательства того, что реализация соответствует исходной спецификации (подобно формальному математическому доказательству).

1971 г. *Никлаус Вирт (NiclausWirth) разработал язык программирования PASCAL (назван в честь Блеза Паскаля).*

Верификация модели

Наиболее широко распространённой открытой системой верификации моделей является программный пакет SPIN (http://spinroot.com), который почти 20 лет развивался Джерардом Холзманом (Dr. Gerard J. Holzmann) из лаборатории Bell Labs. Весьма знаменательно то, что SPIN недавно получил весьма почётную награду (Software and System Award) от *Ассоциации по вычислительной технике (*или *ACM — Association for Computing Machinery)*, которую до этого получали Unix, SmallTalk, TCP/IP и World Wide Web.

SPIN обрабатывает исходную спецификацию с интегрированной моделью системы, используя для этого язык PROMELA[1]. С помощью этого языка пользователи могут создавать сложные утверждения, определяющие события, которые никогда не должны происходить в системе. В представленной модели и спецификации SPIN осуществляет поиск пространства состояний ошибок.

Главный недостаток системы SPIN заключается в том, что она изначально предназначена для проверки асинхронного программного обеспечения, то есть использует методы *явной проверки*. Хотя явная проверка идеально подходит для тестирования программных протоколов, эти методы могут оказаться неэффективными для больших аппаратных устройств.

Для средних по размеру устройств лучше всего подходит так называемая *символическая система проверки модели*. В отличие от явной проверки, она использует *диаграммы двоичного выбора* и *алгоритмы высказываемой выполнимости* (или *SAT*)[2] для того, чтобы удержать проблемы и, по возможности, избежать расширения пространства состояний. Эти функции выполняет высококачественная система проверки с открытым исходным кодом под названием NuSMV (http://nusmv.irst.itc.it).

Автоматизированная формулировка логических выводов

Преимущество рассмотренного в предыдущем разделе метода верификации модели заключается в том, что этот процесс выполняется автоматически — нажимаем на кнопку и ждём результат. Однако вам, возможно, придётся довольно долго ждать окончания проверки.

Хотя символическое представление, используемое пакетом NuSMV, является более совершенным, чем явная проверка модели, для него всё же существует угроза расширения пространства состояний. Очень скоро наступит ситуация, когда рост размеров системы превысит практические ограничения, присущие системе верификации моделей. Другая проблема заключается в том, что некоторые сложные утверждения просто не могут быть описаны в среде верификации мо-

---

[1] Название PROMELA образовано от PROcess MEta LAnguage — язык метаобработки. — *Прим. пер.*

[2] Название SAT образовано от первых трёх букв слова «*satisfiability*», что означает *выполнимость*.

1972 г. *Америка. Компания Intel представила микропроцессор 8008.*

делей. Для решения этих проблем существуют средства *автоматизированной формулировки логических выводов*, или, как их ещё называют, средства *автоматизированного доказательства теорем*.

Для автоматизированной формулировки не свойственны ограничения, присущие верификации моделей. Например, размеры системы не играют столь большого значения, так как автоматизированная формулировка не производит поиск пространства состояний. Кроме того, она поддерживает более высокий уровень выражений для точного моделирования сложных спецификаций.

К сожалению, то, что достигнуто в одних областях, уже утеряно в других. Несмотря на название, автоматизированная формулировка не выполняется в автоматическом режиме. На практике, при проверке устройства разработчики проводят доказательства с помощью определенных средств. Более того, для эффективного использования этих средств инженерам необходимо хорошо владеть стратегией доказательств, математической логикой и другими приёмами. Всё это требует нестандартных знаний, но если вы желаете потратить время и силы, то автоматизированная формулировка логических выводов, вероятно, будет наиболее мощным средством верификации ваших систем.

В отличие от средств верификации моделей, где приложения с открытым исходным кодом соперничают с коммерческими приложениями, в области автоматизированных формулировок открытые средства являются наиболее проработанными. Самыми популярными продуктами этого класса являются пакеты

- HOL (http://hol.source-forge.net);
- TPS (http://gtps.math.cmu.edu/tps.html);
- MetaPRL (http://cvs.metaprl.org:12000/metaprl/default.html).

В чём заключается проблема?

Качества формальной верификации определяются тем, как инженеры её используют, что, впрочем, справедливо для всех средств. Фактически формальная верификация может ответить только на один вопрос — насколько моя реализация соответствует спецификации? Но появляется другой вопрос — насколько корректно составлена сама спецификация?

Оценка современных методов проектирования показывает, что большинство системных ошибок по существу не зависят от процесса реализации. Даже без использования средств формальной верификации, как правило, реализация устройства чаще выполняется корректно. Основная причина большинства ошибок обычно связана с теми требованиями, которые предъявляются к устройству или схеме.

Лучше всего обеспечить корректность составления спецификации можно в процессе открытого общения и сотрудничества, и во время написания этой книги единственным инструментом для решения этой проблемы была кора головного мозга.

Доступ к общим блокам интеллектуальной собственности

Полезное практическое правило для малых (и даже для больших) проектных организаций гласит, что не стоит изобретать велосипед. Со временем у каждой организации или компании собирается библиотека часто используемых компонентов, которые повышают скорость проектирования. Возможности проектной организации иногда оцениваются по её набору блоков интеллектуальной собственности (IP).

OpenCores

К счастью, у разработчиков есть доступ к большой библиотеке блоков интеллектуальной собственности в виде проекта OpenCores (www.opencores.org). OpenCores представляет собой основное хранилище промышленных аппаратных блоков интеллектуальной собственности (IP) с открытым исходным кодом, в котором собраны проекты ядер арифметических модулей, контроллеров связи, сопроцессоров, ЦСП, средств прямого исправления ошибок и встраиваемых микропроцессоров. Более того, OpenCores осуществляет поддержку стандарта Wishbone, который применяется в однокристальных устройствах в качестве протокола общей шины.

OVL

Разработчики электронных систем могут потратить до 70 процентов времени, выделенного на разработку устройства, на выполнение операций проверки. В связи с этим возникла необходимость доступа к библиотеке IP, предназначенных для тестирования устройств. Чтобы удовлетворить эти потребности, компания Accellera (www.accellera.org) начала развитие Библиотеки открытых средств проверки (или OVL — Open Verification Library)

Средства синтеза и реализации

Синтез (логический и физический) является одним из основных этапов в процессе проектирования устройств на основе ПЛИС, который нельзя реализовать в полном объёме только с помощью средств с открытым исходным кодом. К сожалению, эта ситуация вряд ли измениться в ближайшем будущем из-за сложности ПЛИС-синтеза.

В то время, когда я работал над этой книгой, компилятор Icarus (*см.* также раздел «Среда верификации» выше в этой главе) был единственным открытым средством, способным синтезировать прототипы ПЛИС из HDL кода. Недорогие средства синтеза предлагали только поставщики ПЛИС (как правило, их выбирали для недорогих решений).

Однако когда ваше устройство приближается к ёмкости самых мощных ПЛИС, то средства синтеза от поставщика ПЛИС не могут решить поставленную перед ними задачу. Это значит, что для проектирования больших и сложных устройств вам придётся раскошелиться и купить мощное средство синтеза.

Макетная плата

Если проектная организация решит заняться физической реализацией аппаратуры на основе ПЛИС, то обязательно потребуется макетная плата ПЛИС.

Проект OpenCores (*см.* также раздел «Доступ к общим блокам интеллектуальной собственности» выше в этой главе) предлагает несколько вариантов макетных плат ПЛИС, но большинство разработчиков предпочитают покупать профессиональные платы.

С другой стороны деньги, потраченные на плату, могут быть сэкономлены в других областях. Например, умные инженеры могут превратить небольшую макетную плату ПЛИС в логический анализатор с большими возможностями (да, это звучит как потенциальный проект OpenCores).

1973 г. *Америка. Компания Selbi Computer Consulting Company представила микрокомпьютерный набор для самостоятельной сборки под названием Selbi-8H.*

1973 г. *Америка. Разработан компьютер Xerox Alto.*

1973 г., *май, Франция. Разработан компьютер Micral на базе микропроцессора 8008.*

Другие необходимые средства и утилиты

В нижеследующем списке представлены Интернет-сайты, на которых можно найти заслуживающие внимание средства и утилиты:

- www.easics.be
  На странице этого сайта щёлкните на ссылке «WebTools» для того, чтобы найти утилиты создания контрольных сумм (CRC), которые позволят вам выбрать стандартную или полиномиальную проверку и сгенерировать соответствующие Verilog или VHDL-модули.
- www.linuxeda.com
  Здесь находятся средства САПР для ОС Linux.
- http://geda.weul.org
  На этом сайте можно найти коллекцию открытых САПР для проектирования электроники.
- www.veripool.com
  Здесь можно найти набор средств языка Verilog (домашняя страница пакетов Dinotrace и Verilator).
- http://ghdl.free.fr
  На этом сайте находится клиентская часть открытого языка VHDL для компилятора gcc.
- http://asics.ws
  На этом сайте находятся некоторые блоки интеллектуальной собственности с открытым кодом.

Во время путешествия по страницам Интернета можно встретить большое количество проектов с открытым исходным кодом, относящихся к ПЛИС или САПР электронных устройств. К сожалению, большинство из них не работают или заброшены на этапе, когда они ещё не достигли какого-либо полезного уровня функциональности. Поэтому, если вы встретите или сами создадите что-нибудь полезное, то, пожалуйста, свяжитесь со мной по адресу max@techbites.com, чтобы я, по-возможности, смог поместить этот материал в следующем издании этой книги.