Краткий курс HDL.
Часть 1. Введение

Иосиф КАРШЕНБОЙМ
iosifk@narod.ru

Данный цикл статей посвящен описанию языка Verilog. Будут приведены примеры описаний типовых узлов схем, некоторые рекомендации по стилю выполнения описаний на этом языке и материалы по разработке тестбенчей.

Почему был написан этот материал?

Когда-то, в очень старые времена, в газетах журналисты применяли такую расхожую фразу — «письмо позвало в дорогу». Вот отрывки из письма Дмитрия, студента одного из российских университетов: «А книжку, которую Вы порекомендовали, в нашей библиотеке не нашел…», «…так как вижу, что спрос на литературу по ПЛИС (по крайней мере, здесь) очень велик… Спрос намного превышает предложение. Я считаю, что нужна книга с реализацией проекта от нуля до чего-то стоящего, и желательно для начинающих. В общем-то, Вы ближе всего подошли к решению данного вопроса».

Но для того чтобы автор смог решиться на такой шаг, как написание, пусть и небольшого, но все же учебного курса, понадобилось несколько лет общения с молодыми коллегами, студентами и преподавателями из различных университетов. Конечно, этот курс не заменит настоящего, полного изучения предмета, но все же автор надеется, что «Краткий курс HDL» (далее — КК) окажется полезным, в первую очередь тем, кто изучал цифровую схемотехнику по «безумно облегченным» университетским программам. Безусловно, есть хорошие книги и на русском языке, но, во-первых, большинство из них «про VHDL», а во-вторых, прочтите еще раз первую фразу из письма Дмитрия: эти книги почти недоступны студентам.

И еще надо добавить следующее. В отличие от многих изданий, выпущенных университетскими преподавателями, автор не ставил себе целью получить «учебник для изучения по программе ПЛИС за 4-й курс». Возможно, этот КК кому-то покажется сложным или избыточным. Может быть, кто-то из читателей захочет высказать свои замечания, предложения или прислать свои примеры. Пожалуйста, напишите об этом. Все ваши предложения будут учтены и размещены на сайте автора.

Несколько слов о студентах

Автор не берется судить обо всех предметах, изучаемых сегодня в вузах по различным направлениям «электронных специальностей». Но общение со студентами и преподавателями ведущих петербургских вузов и письма читателей, обучающихся в различных университетах страны, наводят на довольно грустные размышления. Подведем итог: студентов многих, даже приборостроительных специальностей, вообще не учат технологии проектирования с применением FPGA, либо учат только «картинкам» (схемное описание проекта), либо дают только некоторые и очень краткие теоретические сведения по отладке проектов, но без практики на реальных аппаратных платформах.

Автор надеется, что читатели все же имеют некоторый опыт работы с электронными приборами. По крайней мере умеют читать схемы и понимают то, как работают простейшие узлы цифровых схем: триггеры, счетчики, мультиплексоры и т. д. Поэтому данный КК строится так, чтобы облегчить переход от схемных описаний устройств к описаниям на HDL. Но для того чтобы объяснить некоторые базовые принципы, сделано довольно подробное введение в КК.

Как будет сформирован КК?

Обычно все авторы стараются сделать свои учебники всеобъемлющими. Они стараются как можно более полно изложить материал, предназначенный для изучения. Масса примеров и тонкостей, много сокращений и специальных терминов — это обычное дело для таких книг. Однако для реальной жизни нужно совсем другое. Когда-то, очень давно, в 1977 году, автору посчастливилось купить переводную книгу издательства «Мир». Она называлась «Справочник по нелинейным схемам». Стиль изложения материала в этой книге был выбран очень необычный и не походил на все то, что издавалось у нас в стране. Книга была разделена на три части. Первая часть, объем которой не превышал 15%, содержала небольшое введение и «готовые к употреблению» схемы логарифматоров и перемножителей. Нужен синусно-косинусный генератор — вот схема, перечень компонентов, небольшое описание к ней. Вторая часть книги, примерно 35% от ее объема, представляла подробные примеры расчета и синтеза тех же самых логарифматоров и перемножителей. Но в этом разделе книги можно было посмотреть и быстро пересчитать основные характеристики схемы. А вот третья часть, она занимала почти половину книги, представляла собой очень подробное описание все тех же самых логарифматоров и перемножителей. Но только теперь были рассмотрены все параметры, влияющие на точность работы такого узла: и частотные, и температурные.

Почему потребовалось столь подробное описание структуры книги?

Да потому, что и «Краткий курс HDL» тоже хочется построить таким же образом. Если стоит задача сделать простой проект и довольно быстро, то 500–600 страниц англоязычного текста оказываются довольно серьезной преградой. Мало того, множество подробностей и тонкостей, возникающих при изучении языка, требуют много времени и сил. Именно поэтому автор стремился преследовать только одну цель: упростить разработку проектов с применением HDL-кодирования. Облегчить переход от проектирования в «картинках» к проектированию на языках HDL. Изучение такого «Краткого курса HDL» будет первым шагом на пути освоения методики проектирования на языках HDL. Тот, кто посчитает для себя необходимым сделать дальнейшие шаги по изучению HDL-кодирования, всегда сможет найти для себя «толстый» учебник и развить свои способности. Еще раз хочется подчеркнуть, что изучение языка — это не самоцель, это только средство для успешного выполнения проектов. И в данном контексте хочется привести лишь один пример. При создании тест-бенча можно делать поведенческое описание тестирования, а можно сделать подробное описание на уровне триггеров и вентилей, такое же, как описание самого тестируемого объекта. Безусловно, поведенческое описание гораздо легче создать. Но это надо уметь делать! И если стоит задача выполнить это быстро, а на изучение того, как это должно быть сделано, у вас нет времени, то структурное описание окажется со-

здать легче, хотя и кода там будет больше, да и выглядит оно не так «красиво»…

И, наконец, еще одно замечание.

Первоначально «Краткий курс HDL» планировался только как очень краткое описание синтезируемых конструкций. Но в процессе работы над материалом автор пришел к твердому убеждению, что сегодня этого мало даже для краткого курса. Сейчас необходимо не только уметь быстро сформировать файл для микросхемы FPGA, содержащий пару счетчиков и дешифраторов. Разработчику необходимо уметь симулировать проект не на уровне отдельной микросхемы, а на уровне всей платы, содержащей FPGA, память и микроконтроллеры. А это подразумевает, что необходимо иметь очень четкое представление о том, как должна работать модель, как задать задержки и какие именно задержки в ней необходимы. Поэтому, сохранив первоначальное название, автор значительно расширил описание языковых конструкций и добавил большее количество примеров. Что же касается дополнительных разделов, то это, можно сказать, отдельные главы, и они более подробно описывают наиболее часто встречающиеся при проектировании проблемы.

Итак, вот что будет предложено читателям.

- Часть 1 — введение.
- Часть 2 — описание языка. Оно довольно подробное и занимает много места.
- Часть 3 — о написании кода. Что и как надо делать для того, чтобы проект легко читался и сопровождался.
- Часть 4 — шаблоны текстовых файлов. Где они находятся в программных инструментах.
- Часть 5 — написание кода, независимого от аппаратной платформы. Примеры кода для наиболее распространенных цифровых узлов схем — триггеров, регистров, счетчиков. Этот раздел можно воспринимать как справочный.
- Часть 6 — написание кода, зависимого от аппаратной платформы.
- Часть 7 — несколько слов об отладке проекта. Что и как надо делать для симуляции и отладки.
- Часть 8 — моделирование в ModelSim SE. Небольшой проект, представляющий собой пример кода, необходимый для тестирования устройства, содержащего FSM и программируемый таймер. Работа с файлами.
- Часть 9 — более подробно о параметризируемых модулях.
- Часть 10 — более подробно о сигнале «Сброс».
- Часть 11 — более подробно об асинхронных частотах и о пересечении клоковых доменов.

Немного философии

Герой известного кинофильма объяснял стратегию при помощи картошки. Ну, а мы, чтобы не отстать от классического шедевра, попробуем начать с яблок. Яблоки, какие они? Круглые, сочные, красные, «на снегу», «эх, яблочко» и так далее. Все это характеристики объекта, причем, ни одна из этих характеристик не описывает предмет целиком, а дает только свою часть описания. Точно таким же образом инженеры описывают разрабатываемое изделие. И на каждое изделие существует определенное стандартом количество документов, которые все вместе полностью описывают конкретное изделие. И называются они комплект конструкторской документации (КД). Каждый из входящих в него документов описывает какую-то одну сторону изделия. И при разработке узла какого-либо устройства нас, в первую очередь, будет интересовать то, какую функцию он выполняет, из чего состоит, как связан с другими узлами разрабатываемого изделия и по какому алгоритму работает.

И еще немного философии. При описании объектов люди часто пользуются различными абстракциями. Чаще всего в качестве абстрактного образа используется уличный светофор. Наверное, нет необходимости объяснять, почему… Точно так же и в схемотехнике. Кружочек, три черточки, одна поперечная линия и одна стрелка — это не описание транзистора, а только его условное обозначение на схеме. Но оно связано с описанием работы транзистора, возможно, оно связано и с его конструкцией. Таким образом, рассматривая условное графическое обозначение, можно представить то, как именно этот компонент будет работать. Более того, обычно мы рассматриваем компонент в схеме, то есть во взаимосвязи с другими компонентами. И опытный специалист, глядя на схему, в подавляющем большинстве случаев сразу скажет вам, как работает тот или иной ее узел.

Теперь давайте вспомним, что бывают такие схемы, которые не помещаются на одном листе. Или, например, какой-то фрагмент схемы повторяется многократно. Что делается в этом случае? Фрагмент выделяется в отдельный чертеж. При этом весь проект становится иерархическим. То есть на самом верхнем уровне чертеж состоит из прямоугольников, которые представляют собой части схемы, выделенные в отдельные чертежи. Те, кто программировал на языке высокого уровня, тут же вспомнят вызов функций. Действительно, принцип тот же самый. На более высоком уровне описания объекта присутствуют только «главные» характеристики объекта, а все то, что не соответствует этому уровню, находится в низших уровнях описания. То есть в описании функций или в тех самых отдельных чертежах, о которых шла речь ранее. Для примера можно представить себе механизм от заводной детской игрушки (наверное, каждый читатель имел в детстве опыт по «изучению внутренностей» заводных игрушек…). Что есть у такого механизма? Есть место, куда вставляется ключ, для того чтобы эту игрушку заводить, а из механизма выходит ось, которая вращает колеса игрушки. Все остальное сейчас для нас не так интересно: оно скрыто в корпусе механизма. Так что же с точки зрения схемотехники представляют эти два «внешних» для игрушечного механизма момента? Это две «точки подключения» механизма к остальному устройству. Точно так же и в схемотехнике: при создании иерархического проекта нас будет интересовать только то, как функционируют «квадратики», то есть то, что выделено в отдельные блоки, и как они подключаются ко всем остальным цепям верхнего проекта.

Теперь надо рассмотреть еще один момент. Если мы выделяем из схемы только один фрагмент и представляем его в виде «квадратика» на верхнем уровне, то тут все понятно. Что же происходит, когда надо компактно изобразить на чертеже верхнего уровня несколько одинаковых фрагментов? Решение здесь точно такое же. Выделяется унифицированный фрагмент схемы и из него делается чертеж нижнего уровня. А на верхнем уровне помещается столько «квадратиков», сколько нам нужно этих унифицированных узлов. Но как же мы узнаем, где и какой фрагмент находится и куда он подключается? По именам, которые мы даем при установке компонента в схему. Чтобы представить себе эту ситуацию более наглядно, вспомним, что комплект КД — это тоже только описание изделия. А вот серийные изделия на конвейере — это как раз полный аналог нашего случая. Так чем же отличаются новенькие автомобили, сходящие с конвейера? Они отличаются только своим «именем» — серийным номером.

Или другой пример: берем Си++, смотрим описание какого-нибудь класса. А также на то, как производится работа с экземпляром данного класса. Вот теперь мы подошли к объяснению терминов «instance» и «instantiate». Фрагмент схемы, выделенный в описание нижнего уровня, будем называть «instance» — компонент. Размещение на схеме верхнего уровня такого фрагмента в виде блока будем называть установкой компонента — «instantiate».

Итак, мы выяснили, что обсудим только часть описания изделия. Мы не будем обсуждать физическую сущность, конструкцию и прочее, все то, что не относится к схемотехнике. Мало того, «за кадром» останутся вопросы, связанные с алгоритмом функционирования. Они должны быть описаны в отдельном документе. Здесь же мы будем рассматривать только вопросы схемотехники. И так же, как при рассмотрении классических схем, можно сказать о том, как работает тот или иной узел, точно так же это суждение можно составить из HDL-описания блока. Но еще раз хочется напомнить, что классическое описание алгоритма работы устройства всегда должно быть сделано.

Языки программирования и языки описания аппаратуры

В чем же разница и в чем сходство таких языков? Мы уже выяснили, что любой предмет можно и нужно описывать следующим образом:

1. В какой предметной области данный предмет применяется?
2. Цели/задачи, для которых он предназначен?
3. Из чего состоит предмет?
4. Как эти части взаимодействуют?
5. Как предмет взаимодействует с внешней средой?

Начнем с языков программирования

Будем считать, что читатель уже имел дело, пусть хотя бы и поверхностно, с одним из таких языков. (Ну, а если вы еще не работали с языками программирования, то вспомните ваши действия по управлению телевизором или СВЧ-печкой…). Язык программирования задает последовательность действий в фиксированной среде. Представим, что мы программируем микроконтроллер. Сам микроконтроллер от наших действий не меняется. Меняется только порядок обработки данных, поступающих в микроконтроллер и выдаваемых из него. При этом программист знает, что структура микроконтроллера и описание его команд приводятся в его комплекте документации. Микроконтроллер под управлением потока команд обрабатывает данные последовательно. Да, конечно, существуют процессоры, которые за один такт «умеют» выполнять несколько команд, есть каналы DMA, которые пересылают данные на периферийные узлы микроконтроллера для дальнейшей автономной обработки, но все же, в целом, обработка данных идет последовательно. Представим, что мы хотим произвести инкремент 100 переменных. Тогда в тексте программы это будет выглядеть так:

```
X0++;
X1++;
….
X99++;
```

Можно сказать так: если представить, что каждая команда выполняется за один такт, то такой фрагмент программы будет выполнен микроконтроллером за 100 тактов.

Язык описания аппаратуры

Язык описания аппаратуры, в противоположность языку программирования, описывает связи в устройстве. А функционирование устройства будет происходить в соответствии с теми внешними воздействиями, которые будут к нему приложены. То есть разработчик аппаратуры, в соответствии с заданием для разработки, выбирает состав устройства и описывает этот состав и все связи устройства, пользуясь тем, что алгоритм его работы описан в комплекте документации или, если говорить более точно, в техническом задании, входящем в этот комплект. Поскольку составные части устройства могут работать независимо друг от друга, то для примера с инкрементированием 100 переменных на аппаратном уровне нужен будет всего лишь 1 такт. Конечно, скажет читатель, это же аппаратные «штучки», так можно сделать все, что разработчику угодно… Действительно, это так, но речь сейчас о другом. Дело в том, что в тексте описания аппаратуры, так же как и в тексте описания программы, будут присутствовать 100 строк примерно такого вида:

```
Xn = Xn +1;
```

Таким образом, внешне все очень похоже. Но только внешне. А по сути это будут 100 одновременных процессов. Необходимо еще раз подчеркнуть, что все процессы, изложенные на языке описания аппаратуры, идут одновременно. Конечно, языки описания аппаратуры кроме возможностей, предоставляемых структурным описанием, имеют набор возможностей и для поведенческого описания. Да, можно описать то, как должно работать устройство. Но здесь необходимо отметить, что поведенческое описание не всегда можно превратить в загружаемый в микросхему FPGA файл. Только самые элементарные поведенческие описания компилируются в синтезируемые конструкции. И из этого описания компилятор синтезирует опять только связи внутри устройства. А алгоритм работы на все устройство, как в целом, так и в этом случае, будет зависеть от связей и от входных воздействий на данное устройство.

Чем еще отличаются эти языки?

Текст программы на языке программирования пишется для того, чтобы он исполнялся на процессоре. Конечно, программа может быть написана на языке высокого уровня, который сам по себе не зависит от аппаратной платформы. Но компилятор с языка высокого уровня переводит весь текст программы в исполняемый код для определенного целевого процессора, на котором он и выполняется. И в текст программы могут быть добавлены фрагменты, которые позволяют на этапе ее отладки совершать какие-либо дополнительные действия. Можно применить условную компиляцию и вставить дополнительно фрагменты программы, позволяющие гибко перестраивать текст под нужды программиста. Текст описания аппаратуры на HDL-языке пишется и для того, чтобы провести симуляцию работы устройства, и для того, чтобы получить «прошивку», то есть загружаемый в FPGA файл. Поэтому в этих языках есть часть языковых конструкций, которые применяются только для симуляции и не могут быть синтезированы в аппаратные узлы. Разработчик обязан следить за тем, чтобы при описании того узла, который должен быть загружен в FPGA, применялись только синтезируемые конструкции языка или библиотечные компоненты, поставляемые производителем FPGA.

Что же общего у этих языков?

Общее — это то, что проекты и в том, и в другом языке строятся иерархически. Часть кода выделяется в классы, и экземпляры классов инстанциируются в проект. Используются библиотеки.

Далее необходимо отметить тенденцию по объединению этих языков. Дело в том, что сейчас появились версии языков программирования SystemC, System Verilog, позволяющие выполнять проекты на языке Си, с последующей автоматической компиляцией этих проектов на языки HDL.

VHDL и Verilog

Когда автора на одной из встреч в университете спросили студенты: «Что же нам надо изучать: VHDL или Verilog?», то ответ был такой: «А вы готовы пренебречь половиной рабочих мест только потому, что вы не захотели выучить оба эти языка?»

Нет нужды объяснять, что, как минимум, надо понимать оба этих языка. А на каком языке специализироваться? Есть такое негласное правило: фирма должна делать разработки на том языке программирования, на котором пишет ее лучший программист. Но, поскольку в нашей стране большинство книг написано университетскими преподавателями, а не разработчиками аппаратуры, то есть явный перекос в сторону VHDL. В то время как довольно большая часть инженеров с успехом пользуется языком Verilog. Поэтому автор уделит значительно больше внимания именно этому языку программирования. Практически, это и будет учебник по языку Verilog, а небольшие вкрапления примеров на VHDL помогут сравнить эти два языка и при необходимости быстрее перейти с одного языка на другой.

Переход от «картинок» к HDL

Схемное описание проекта, или «картинки» — это наиболее простое и доступное решение для небольших проектов. Оно, несомненно, наглядно и просто. И на этом все преимущества схемного описания проекта заканчиваются. Можно лишь добавить, что при использовании библиотек компонентов, поставляемых фирмой-изготовителем, удается применять параметризированные компоненты. Недостатки также очевидны. При уве-

личении сложности и объема проекта объем файлов тоже растет, и отладка такого проекта становится сложной. Более подробно об этом можно будет прочитать в тех разделах KK, где будут описаны методики отладки пользовательских проектов.

Несколько слов о терминологии

В любом языке, как в обычном разговорном, так и в техническом, есть свои устоявшиеся нормы и критерии. Есть свои слова для определения объектов и событий, а есть слова заимствованные. Когда заимствованные слова теряют свою новизну, то они воспринимаются уже как свои, «родные». Более сотни лет тому назад в литературных кругах шли дискуссии о «мокроступах» — так предлагали называть калоши некоторые отечественные литераторы, но в обиход вошло «импортное» слово. А когда СССР был ведущей научной державой, то все народы ввели в свои словари и начали применять слово «спутник». То же самое происходит и в других областях знаний. Те термины, которые первыми входят в оборот, получают наиболее широкое употребление. К сожалению, в той области знаний, к которой относится данная книга, наша страна сегодня не является лидером. Поэтому здесь нам приходится пользоваться терминологией, базирующейся на принятых во всем мире англоязычных терминах. Естественно, часть из этих терминов имеет понятный и привычный перевод. Но некоторые из них могут показаться читателю непривычными. Однако поскольку подавляющее большинство литературы в этой области написано на английском языке, то читателю, рано или поздно, придется к такой литературе обратиться. Поэтому автор не считает нужным вводить какие-либо «более литературные термины», для того чтобы читатель мог подготовить себя к чтению англоязычной литературы.

Теперь несколько слов о том, «как это работает». Если вы увидите такое выражение:

$$2+2 = 4,$$

то это, конечно, равенство. И тут все должно быть понятно.

Выражение Y = X в математике называют уравнением. А вот в языках HDL смысл такого выражения будет зависеть от того места, где оно приведено. Если выражение используется как математическое, то тут все понятно. А вот в том случае, когда такое выражение применяется для описания передачи сигналов от одних компонентов схемы к другим, его смысл меняется. При описании схемы используются термины **операция** и **операнд**. Операнды различаются на те, которые находятся справа и слева от знака операции.

Теперь необходимо познакомиться с теми терминами, которыми пользователь оперирует при описании проекта на языке Verilog.

Термин «**назначение на цепь**»: **assign** — назначить; **assignment** — назначение.

В терминах HDL приведенный пример можно трактовать так. Есть цепи Y — это приемник данных и X — это источник данных. И этим выражением цепи Y назначаются те данные, которые ей передает цепь X. При моделировании под выражениями Y и X понимаются и переменные.

Утверждение. Еще раз обратимся к нашему примеру. Вот такие выражения:

$$Y = X \text{ или } Y <= X.$$

Это не формула, не уравнение. Это нечто совсем другое. В англоязычной литературе для такого выражения применяется термин **statement** (**утверждение**).

Драйвер сигнала. Это тот компонент схемы, который управляет сигналом в какой-либо конкретной цепи. Одна цепь может иметь несколько драйверов.

Instance — установленный в проекте пользователя компонент, представляющий собой экземпляр того, что сделано в описании модуля. Аналогичен применяемому в C++ экземпляру класса.

Положительный фронт — переход сигнала из состояния низкого уровня в состояние высокого уровня.

Отрицательный фронт — переход сигнала из состояния высокого уровня в состояние низкого уровня.

Примечание. *В данном описании оставлены некоторые термины на английском языке, для того чтобы читатель смог в дальнейшем легко узнать их в англоязычной литературе.*

Предупреждение об ответственности

Все приведенные в книге примеры являются учебными и могут использоваться в том виде, в каком приведены. Автор не несет ответственности за применение этих примеров читателями в своих разработках. ■

Продолжение следует