Посвящаю моей жене

Предисловие ко второму изданию

Прошло уже достаточно много времени с момента первого издания данной книги и, по словам издателя, читательский интерес к книге не ослабевает. Целью первого издания было привлечение внимания отечественных разработчиков цифровой аппаратуры к языку Verilog. Дело в том, что на тот момент (2014 год) было издано достаточно много литературы, посвященной языку VHDL, и ни одной книги по языку Verilog. И это несмотря на то что в мире язык Verilog не менее (если не более) популярен, чем язык VHDL.

Первое издание книги успешно справилось с поставленной задачей. С этой целью был специально занижен объем книги, чтобы сделать ее дешевой и доступной большинству читателей. В первом издании кратко описываются основы языка Verilog, достаточные для первоначального знакомства с языком. Однако знание конструкций языка Verilog является необходимым, но не достаточным условием для успешного проектирования цифровых устройств с использованием языка Verilog. Поэтому во второе издание было добавлено пять глав, касающихся вопросов проектирования цифровых устройств на языке Verilog.

Современные цифровые системы в большинстве случаев представляют собой совокупность комбинационных схем, конечных автоматов и блоков памяти. Их проектированию на FPGA (ПЛИС) посвящены добавленные главы 19–21.

Еще одной особенностью языка Verilog является то, что он позволяет применять различные методики для проектирования цифровых устройств. В главе 22 рассматривается традиционная методика проектирования цифровых устройств, когда проект представляется в виде операционного устройства (операционного автомата) и устройства управления (управляющего автомата).

В главе 23 предлагается новая методика, названная ASMD-FSMD, проектирования цифровых устройств и систем, которая позволяет не только уменьшить стоимость реализации и повысить быстродействие цифровых устройств, но также значительно (в несколько раз!) сократить время проектирования и увеличить надежность проектов. Методика ASMD-FSMD раскрывает новые возможности языка

Verilog, делая разработку цифровых устройств очень похожей на разработку программного обеспечения. Последнее позволяет привлечь к проектированию цифровых устройств многочисленную армию программистов, не искушенных в хитростях проектирования цифровой аппаратуры.

Автор выражает искреннюю благодарность читателям, которые прислали отзывы и замечания на первое издание книги, что позволило улучшить рукопись второго издания.

Научные исследования, результаты которых вошли в данную книгу, частично финансировались грантом WZ/WI-IIT/4/2020 Белостокского технологического университета (Польша).

Введение

Начало нашего века знаменуется бурным развитием и внедрением электронных изделий в повседневную жизнь, а также во все сферы человеческой деятельности (представьте ребенка без электронной игрушки, подростка без смартфона и самолет без бортового компьютера). Однако все эти интересные вещи, прежде чем изготовить, необходимо спроектировать. Поэтому огромная армия инженеров трудится над созданием новых и более совершенных электронных устройств. Канули в лету времена, когда принципиальные схемы сложных устройств чертились вручную или с помощью графических редакторов. На смену им пришли языки проектирования.

Сейчас разработка какого-либо электронного проекта во многом напоминает разработку программ: написание кода программы (написание кода проекта), компиляция программы (компиляция проекта), отладка программы на эталонных примерах (моделирование проекта с помощью специальной программы, называемой *симулятором*), создание исполняемого кода программы (реализация проекта в микросхеме), отладка программы на реальных примерах (физическое моделирование проекта на макетных платах), передача программы заказчику (реализация проекта в виде электронного изделия), сопровождение программы (сопровождение проекта).

В настоящее время наблюдается бурное развитие языков описания аппаратуры (Hardware Description Language — HDL) высокого уровня. Одним из таких языков является язык Verilog. Язык Verilog родился в среде разработчиков цифровых систем, поэтому быстро завоевал популярность у инженеров-практиков. Вскоре язык Verilog стал главным конкурентом языка VHDL. Язык VHDL был разработан по заказу Министерства обороны США, которое всячески способствовало его широкому распространению. На сегодняшний день некоторые программные средства проектирования работают следующим образом: проекты, написанные на языке VHDL, транслируют в язык Verilog, а затем используют компилятор языка Verilog, поскольку язык Verilog более близок к аппаратуре, чем язык VHDL. Отсюда возникает вопрос, не лучше ли сразу создавать проекты на языке Verilog, поскольку любая трансляция требует дополнительных накладных расходов с точки зрения быстродействия, потребляемой мощности и площади на кристалле, занимаемой проектом.

Однако широкому распространению языка Verilog среди русскоязычных проектировщиков в значительной степени препятствует отсутствие книг по языку Verilog. Те немногочисленные статьи, которые иногда появляются в периодических журналах [1–3] и книга Полякова [4] не могут закрыть указанную проблему. Изучение же языка по стандартам [5–7] весьма затруднительно, поскольку стандарты языка Verilog, прежде всего, предназначены не для пользователей языка, а для разработчиков компиляторов языка. В то же время в мире издано большое количество книг, посвященных языку Verilog, как для начинающих пользователей [8, 9], так и для опытных специалистов [10, 11]. Отметим также, что язык Verilog является объектом серьезных научных исследований [12–21].

В данной книге достаточно полно описываются основные синтаксические элементы и конструкции языка Verilog с точки зрения их практического использования. Каждая конструкция сопровождается примером. Поскольку язык Verilog предназначен как для описания проекта, так и для моделирования проекта, то не все конструкции языка Verilog могут быть реализованы аппаратно, т. е. *синтезированы*. Если конструкция синтезируется, то в книге приводится ее реализация на уровне регистровых передач (Register Transfer Level — RTL). Кроме того, непосредственно в тексте книги предлагается большое количество заданий, выполнение которых способствует более глубокому изучению языка. Некоторые задания посвящены изучению особенностей используемого читателем компилятора. Выполнение таких заданий позволяет приобрести практические знания о возможностях конкретного компилятора. В книге также много замечаний. Они заостряют внимание читателя на отдельных свойствах языка Verilog, его реализации в компиляторах и др. Все замечания следует внимательно прочитать и переосмыслить.

Изложение материала данной книги не привязывается к определенной элементной базе или конкретному программному средству проектирования. Другими словами, материал книги может использоваться при разработке проектов как на заказных СБИС и БМК, так и на ПЛИС. Для демонстрации результатов реализации примеров используется пакет Quartus II версии 12.1 фирмы Altera, однако читатель может использовать любой другой пакет.

Книга имеет следующую структуру. В самом начале, в главе 1, дается быстрое введение в язык Verilog, где на простом примере показан процесс разработки проектов на основе языка Verilog, от описания проекта до его моделирования, а также показаны возможности языка Verilog при разработке сложных иерархических проектов. Затем приводятся базовые элементы языка. Таким образом, прочтение только

первой главы уже позволяет читателям разрабатывать простые проекты.

Главными конструкторскими единицами языка Verilog являются модули, им посвящена глава 2. Чтобы в очередной раз «не изобретать велосипед», можно воспользоваться уже готовыми решениями, такими, как примитивы и библиотечные модули, которые описываются в главе 3. Каждый язык имеет свои типы данных, с которыми он оперирует. Типы данных языка Verilog имеют свои специфические особенности и рассматриваются в главе 4. Использование языка проектирования невозможно без знаний операций, описываемых в главе 5. Особую роль в языке Verilog играет оператор непрерывного назначения, рассматриваемый в главе 6. Функционирование проекта в языке Verilog описывается в виде совокупности взаимодействующих между собой процессов, которые представляют процедурные блоки, рассматриваемые в главе 7. При моделировании важнейшую роль играет время функционирования отдельных частей проекта и всего проекта в целом. Операторы управления процедурным временем представлены в главе 8.

И вот все готово для описания алгоритма функционирования проекта во времени. Для этого служат операторы процедурного назначения, рассматриваемые в главе 9, и операторы процедурного программирования, описываемые в главе 10. Важную роль в языке Verilog также играют атрибуты, представленные в главе 11. С помощью атрибутов пользователь может передать указания (и параметры) компилятору как выполнять компиляцию отдельных фрагментов кода. Блоки генерации, рассматриваемые в главе 12, позволяют сократить описание повторяющихся или незначительно отличающихся фрагментов кода. Задачи и функции языка Verilog, описываемые в главе 13, во многом подобны подпрограммам и функциям языков программирования. Блок спецификаций, рассматриваемый в главе 14, позволяет определять значения временных параметров, которые могут использоваться при работе симулятора и временно́го анализатора. Системные задачи и функции приводятся в главе 15, а директивы компилятора — в главе 16. При разработке сложных проектов различными проектировщиками для согласования местоположений отдельных частей проекта служат конфигурации и конфигурационные блоки, рассматриваемые в главе 17. Синтезируемые конструкции языка Verilog приводятся в главе 18.

Настоящая книга в первую очередь предназначена разработчикам цифровых устройств и систем для самостоятельного изучения языка Verilog. Поскольку изложение основных элементов и конструкций языка достаточно полное, книга может также использоваться в

качестве справочника языка Verilog. Кроме того, материал книги может быть использован преподавателями для чтения лекций и проведения практических занятий, а также, безусловно, книга предназначена для студентов соответствующих специальностей в качестве учебного пособия при подготовке к практическим занятиям, экзаменам, при написании курсовых и дипломных работ.

Автор выражает искреннюю благодарность сотрудникам фирмы Атомтех (г. Минск, Республика Беларусь), которые прослушали курс лекций на основе рукописи данной книги, за многочисленные вопросы и замечания, способствовавшие улучшению материала книги.

1 Предварительное знакомство с языком Verilog

1.1. История языка Verilog

Язык Verilog разработали Phil Moorby и Prabhu Goel зимой 1983/1984 года для фирмы Automated Integrated Design System (в 1985 г. была переименована в Gateway Design Automation). Первоначально язык Verilog предназначался исключительно для моделирования цифровых устройств. В 1984 г. фирма Gateway начала продажу программы моделирования (симулятора) под названием Verilog XL. С течением времени симулятор Verilog XL становится все более популярным среди разработчиков цифровых систем.

В 1987 г. фирма Synopsys начала использовать язык Verilog в качестве языка для спецификации при синтезе проектов цифровых систем. С этого момента язык Verilog стал использоваться для задач логического синтеза.

В 1989 г. фирму Gateway купила фирма Cadence — крупный производитель систем автоматизации проектирования (САПР) электронной техники. В соответствии со своей политикой фирма Cadence разделила язык Verilog и симулятор Verilog XL на два независимых продукта. В результате несколько фирм купили права на использование языка Verilog.

В 1990 г. фирма Cadence дала разрешение на публичное использование языка Verilog. С этого момента любая фирма могла использовать язык Verilog без специального лицензионного соглашения. Главной целью такого шага было повышение конкурентоспособности языка Verilog с языком VHDL. В этом же году была создана группа под названием Open Verilog International (OVI) для контроля спецификаций языка Verilog, поскольку каждый разработчик компилятора языка Verilog мог по-своему интерпретировать язык Verilog. Группа OVI выполнила несколько улучшений языка Verilog и в 1993 г. объявила о новой спецификации языка.

В 1995 г. был принят первый стандарт языка Verilog — IEEE 1364-1995. С этого момента язык Verilog, подобно языку VHDL, стал полноправным открытым языком проектирования.

В 2001 г. появился очередной стандарт языка Verilog — IEEE 1364-2001, который определил ряд существенных улучшений языка Verilog и был назван Verilog-2001.

В 2000 г. была начата работа под руководством группы Accelera (консорциум фирм, занимающихся проектированием цифровых систем и разработкой САПР) над языком System Verilog.

В 2005 г. появилось сразу два стандарта, касающихся языка Verilog, Verilog-2005 (IEEE 1364-2005) и System Verilog-2005 (IEEE 1800-2005). В стандарте Verilog-2005 получили отражения небольшие изменения стандарта Verilog-2001.

В последующем идеи языка Verilog получили свое развитие в языке System Verilog, для которого были разработаны стандарты IEEE 1800-2009 и IEEE 1800-2012.

В настоящее время большинство производителей программного обеспечения для проектирования и моделирования цифровой аппаратуры поддерживают стандарт языка Verilog-2001 (Aldec, Altera, Axiom Design Automation, Cadence Design Systems, Dolphin Integration, Fintronic, Frontline, Huada Empyrean Software, Mentor Graphics, Simucad Design Automation, Sugawara Systems, SynaptiCAD, Synopsys, Tachyon Design Automation, WinterLogic, Xilinx и др.). Поэтому в данной книге рассматривается версия языка Verilog-2001 стандарта IEEE 1364-2001. Для демонстрации примеров описания цифровых устройств используется пакет Quartus II версии 12.1 фирмы Altera.

1.2. Первый проект на языке Verilog

1.2.1. Описание проекта

Таблица 1.1
Таблица истинности однобитового сумматора

Входы			Выходы	
cin	a	b	cout	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Традиционно, при изучении языков программирования первая программа выводит на экран символьную строку «Hello, world!». Безусловно, язык Verilog также позволяет выводить строку символов на экран, но все же Verilog это язык проектирования, а не программирования. Поэтому в качестве первого проекта на языке Verilog рассмотрим описание однобитового сумматора. Функционирование однобитового сумматора можно представить в виде *таблицы истинности*, приведенной в табл. 1.1, где a и b — единичные биты слов A и B соответственно; cin — перенос из предыдущего разряда; s — бит суммы; cout — перенос в следующий разряд.

На основании табл. 1.1 можно записать следующие логические уравнения выходных функций:

$$\begin{aligned} s &= \overline{cin} \cdot \overline{a} \cdot b + \overline{cin} \cdot a \cdot \overline{b} + cin \cdot \overline{a} \cdot \overline{b} + cin \cdot a \cdot b; \\ cout &= \overline{cin} \cdot a \cdot b + cin \cdot \overline{a} \cdot b + cin \cdot a \cdot \overline{b} + cin \cdot a \cdot b, \end{aligned} \tag{1}$$

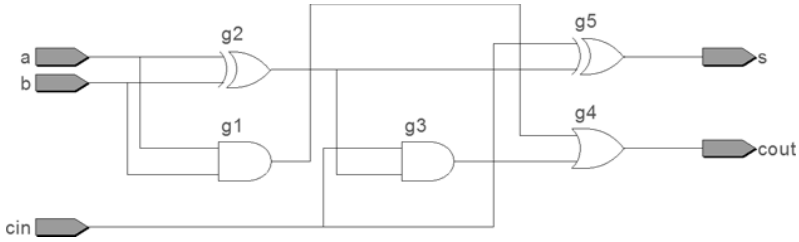

Рис. 1.1. Схема однобитового сумматора на вентильном уровне

которые после минимизации и эквивалентных логических преобразований можно представить в следующем виде (сомневающиеся могут обратиться к любому учебнику по основам электроники и вычислительной техники):

$$\begin{aligned} s &= a \oplus b \oplus cin; \\ cout &= a \cdot b + (a \oplus b) \cdot cin, \end{aligned} \tag{2}$$

где знак $\oplus$ означает логическую операцию Исключающее ИЛИ (XOR). Преимуществом уравнений (2) по сравнению с уравнениями (1) является то, что они могут быть реализованы на вентилях с двумя входами (рис. 1.1).

Описание нашего первого примера однобитового сумматора на языке Verilog представлено в листинге 1.1.

Листинг 1.1. *Структурное описание* однобитового сумматора на рис. 1.1.

```
/*************************************************************
код проекта однобитового сумматора, выполненный в стиле описания
структурной модели на вентильном уровне
*************************************************************/
module add_1_1
  (input cin, a, b,                  // описание входных портов
  output s, cout);                   // описание выходных портов
    /* объявления */
  wire g1_o, g2_o, g3_o;             // промежуточные переменные
    /* операторы */
  and g1 (g1_o, a, b);               // экземпляр первого вентиля AND
  xor g2 (g2_o, a, b);               // экземпляр первого вентиля XOR
  and g3 (g3_o, g2_o, cin);          // экземпляр второго вентиля AND
  or g4 (cout, g1_o, g3_o);          // экземпляр вентиля OR
  xor g5 (s, g2_o, cin);             // экземпляр второго вентиля XOR
endmodule
```

Основной единицей языка Verilog является *модуль*, который начинается ключевым словом **module** и заканчивается словом **endmodule**. За ключевым словом **module** следует название проекта *add_1_1*, за которым в круглых скобках следует список портов модуля: входных (**input**) и выходных (**output**). Описание портов модуля в языке Verilog всегда заканчивается точкой с запятой («;»).

Если для описания проекта требуются дополнительные переменные, они должны быть объявлены перед их первым использованием. В нашем примере это *цепи* (**wire**), соответствующие выходам вентилей: *g1_o, g2_o* и *g3_o*. Собственно описание нашего проекта состоит из описаний экземпляров каждого вентиля на рис. 1.1. Поскольку логические вентили относятся к примитивам языка Verilog, они могут использоваться без предварительного описания или объявления.

Описание каждого отдельного вентиля состоит из названия типа вентиля (**and**, **or**, **xor**), названия экземпляра (*g1, g2, g3, g4, g5*), а также списка портов конкретного экземпляра. Список портов представляет собой список сигналов, назначаемых портам соответствующего экземпляра. Для вентилей характерным является то, что выход вентиля в списке портов всегда является первым элементом, за которым следуют входы вентиля. Такая организация портов вентилей вполне логична, поскольку вентили могут иметь произвольное число входов и только один выход. Для передачи сигналов между вентилями используются объявленные ранее дополнительные переменные *g1_o, g2_o* и *g3_o*.

Замечание. Скалярные (однобитовые) внутренние соединения в языке Verilog можно не объявлять, компилятор их создает автоматически.

Согласно приведенному замечанию строку в листинге 1.1 с объявлением переменных *g1_o, g2_o* и *g3_o* можно опустить.

Задания.

1. Выполните на языке Verilog описание однобитового сумматора по уравнениям (1).

2. Выполните минимизацию уравнений (1) (например, с помощью *карт Карно*) и по полученным уравнениям опишите однобитовый сумматор на языке Verilog.

Листинг 1.1 представляет собой структурное описание проекта на вентильном уровне. Подобным образом можно описать любой проект, причем не обязательно на вентильном уровне. Фактически структурное описание полностью соответствует графическому представлению проекта и здесь не видно преимуществ использования языка проектирования, по сравнению с графическим редактором. Поэтому подоб-

ный стиль описания проектов в языках проектирования используется редко.

Этот же проект можно описать с помощью логических уравнений (2).

Листинг 1.2. Описание однобитового сумматора с помощью параллельных операторов назначения

```
/*************************************************************
код проекта однобитового сумматора, выполненный в стиле описания
логических уравнений
*************************************************************/
module add_1_2
  (input cin, a, b,                          // описание входных портов
  output s, cout);                           // описание выходных портов
  assign s = a ∧ b ∧ cin;                    // описание функции суммы
  assign cout = a & b | (a ∧ b) & cin;       // описание функции переноса
endmodule
```

В листинге 1.2 для описания проекта однобитового сумматора использованы операторы непрерывного назначения **assign**. Данные операторы всегда выполняются параллельно независимо от их места в коде проекта. Здесь справа от знака равенства описаны выражения, использующие логические операции, обозначаемые следующими знаками: «|» — ИЛИ (OR), «&» — И (AND), «∧» — Исключающее ИЛИ (XOR). Отметим также, что отрицание обозначается знаком «∼». Реализация модуля *add_1_2* показана на рис. 1.2. Отметим, что схемы на рис. 1.1 и 1.2 полностью совпали, хотя были представлены различными стилями описания.

Замечание. Стиль описания проекта, приведенный в листинге 1.2, наиболее часто используется для описания комбинационных схем на основании *логических (булевых) уравнений.*

Язык Verilog также позволяет описывать проекты на уровне поведения, т. е. алгоритма функционирования, с помощью *процедурных операторов*. Чтобы описать наш проект однобитового сумма-

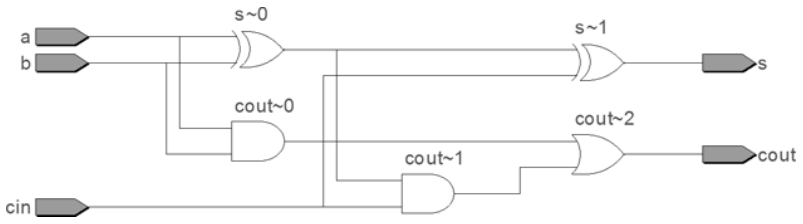

Рис. 1.2. Реализация модуля *add_1_2* из листинга 1.2: пример описания однобитового сумматора с помощью логических уравнений

тора на уровне поведения, выявим некоторые закономерности в его функционировании. Например, функция переноса *cout* будет равна единице, если на входах сумматора одновременно появляется не менее двух единиц. В этом случае описание однобитового сумматора можно представить следующим образом:

Листинг 1.3. Описание однобитового сумматора с помощью процедурных операторов

```
/************************************************************
код проекта однобитового сумматора, выполненный в стиле описания
поведения
************************************************************/
module add_1_3
  (input cin, a, b,                    // описание входных портов
  output reg s, cout);                 // описание выходных портов
  always @(cin, a, b) begin
    if (a & b | cin & a | cin & b) cout=1; // описание функции суммы
    else cout=0;
    s = a ∧ b ∧ cin;                   // описание функции переноса
  end
endmodule
```

В листинге 1.3 использован процедурный оператор **if-else**, а также процедурный оператор блокирующего назначения «=». Кроме того, для выходных функций *s* и *cout* был использован тип переменных **reg.** Дело в том, что согласно правилам языка Verolog переменные, записываемые слева от знака равенства в процедурных операторах назначения, должны иметь тип **reg**. Реализация модуля *add_1_3* показана на рис. 1.3.

После того как некоторый компонент проекта описан, он может быть использован в проекте многократно. Например, однобитовый

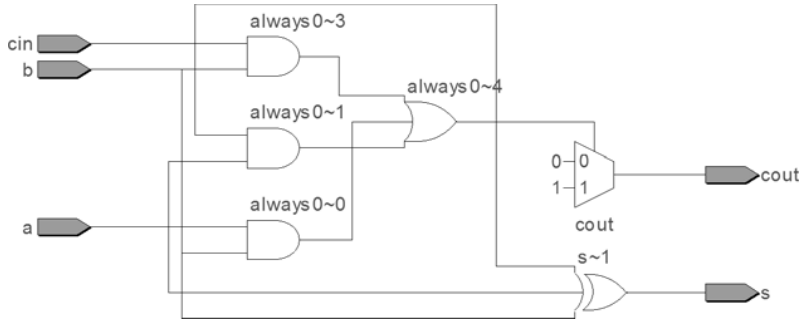

Рис. 1.3. Реализация модуля *add_1_3* из листинга 1.3: пример описания однобитового сумматора с помощью процедурных операторов

сумматор может использоваться для построения 4-битового сумматора.

Листинг 1.4. Описание 4-битового сумматора

```
/*************************************************************
код проекта 4-битового последовательного сумматора, выполненный
путем создания четырех экземпляров однобитового сумматора
*************************************************************/
module add_4
  (input CIN, input wire [3:0] A, B,      // описание входов
  output wire [3:0] S, output COUT);      // описание выходов
  wire [2:0] C;                           // описание промежуточных
                                          // переменных
    /* описание четырех экземпляров однобитовых сумматоров */
  add_1_1 sum0(CIN,A[0],B[0],S[0],C[0]);
  add_1_1 sum1(C[0],A[1],B[1],S[1],C[1]);
  add_1_1 sum2(C[1],A[2],B[2],S[2],C[2]);
  add_1_1 sum3(C[2],A[3],B[3],S[3],COUT);
endmodule
```

В листинге 1.4 использовано четыре экземпляра модуля однобитового сумматора *add_1_1* с именами *sum0, sum1, sum2* и *sum3*. Для передачи значения сигнала переноса между разрядами здесь использованы промежуточные переменные *C[0], C[1], C[2]* типа **wire**. Реализация модуля *add_1_4* показана на рис. 1.4.

Подобным образом на языке Verilog описываются достаточно сложные иерархические проекты.

В общем случае язык Verilog позволяет выполнять описание проектов на следующих уровнях:

- транзисторов;
- вентилей;
- логических уравнений;
- регистровых передач (register transfer level — RTL);
- поведенческом (behavioral);
- структурном (системном).

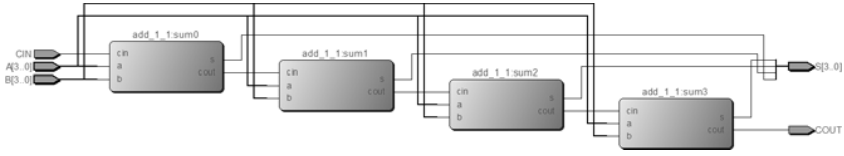

Рис. 1.4. Реализация модуля *add_4* из листинга 1.4: иерархическая структура последовательного сумматора на 4 бита, построенная из четырех экземпляров однобитовых сумматоров

Описание проектов на уровне транзисторов используется редко. Например, такой стиль описания может быть использован при проектировании новых библиотечных элементов для заказных и полузаказных СБИС. Пример описания проекта на уровне вентилей приведен а листинге 1.1. Для описания проекта на уровне логических уравнений достаточно представить проект в виде системы логических уравнений. При этом могут использоваться промежуточные переменные. Пример описания проекта на уровне логических уравнений приведен в листинге 1.2. Описание на уровне регистровых передач (RTL) отличается от описания проектов на уровне вентилей только тем, что вместо вентилей используются функциональные элементы уровня регистровых передач: регистры, счетчики, сумматоры, дешифраторы, мультиплексоры и др. Для описания проектов на поведенческом уровне используются процедурные операторы языка Verilog. Пример описания однобитового сумматора на поведенческом уровне приведен в листинге 1.3. Описание проекта на системном уровне подобно описанию на уровне регистровых передач, только вместо функциональных элементов уровня регистровых передач используются функциональные блоки системного уровня (процессоры, памяти, шины, устройства управления, устройства ввода-вывода и др.).

1.2.2. Моделирование проекта

После того как проект описан, его следует проверить (верифицировать), т. е. выполнить моделирование проекта. Ранее, до появления языка Verilog, для моделирования использовались графические редакторы, специальные языки, отличные от языков описания аппаратуры, и др. Однако, поскольку язык Verilog изначально был предназначен для моделирования, то он же используется и для моделирования проекта.

Модуль для моделирования однобитового сумматора представлен в листинге 1.5.

Листинг 1.5. Модуль для моделирования однобитового сумматора

```
‘timescale 1ns/1ps                  // определение единицы времени
                                    // моделирования
module test_add_1();                // начало модуля
  reg tcin, ta, tb;                 // объявление переменных
                                    // для входных векторов
  wire ts, tcout;                   // объявление переменных
                                    // для выходных функций
  add_1_1 sum(tcin,ta,tb,ts,tcout); // создание экземпляра модуля
  initial begin: test               // процедурный оператор
```

```
                                    // с именем test
    integer i;                      // объявление локальной
                                    // переменной
    $display("Результаты моделирования однобитового сумматора:");
    $timeformat(-9,1,"ns",8);       // определение формата времени
    $monitor($time,": cin=%b a=%b b=%b s=%b cout=%b",
    tcin,ta,tb,ts,tcout);
    for (i=0; i<8; i=i+1) begin     // начало оператора цикла
      #10;                          // оператор задержки
                                    // на 10 единиц времени
      {tcin,ta,tb} = i;             // формирование входного вектора
    end                             // окончание оператора цикла
  end                               // окончание процедурного
                                    // оператора test
endmodule                           // окончание модуля
```

Отметим некоторые особенности приведенного описания. В первой линии кода с помощью системной директивы **‘timescale** определяется значение временных единиц 1ns (одна наносекунда) и точность измерения времени 1ps (одна пикосекунда). Поскольку модуль для моделирования с именем *test_add_1* извне не получает никаких сигналов, а также не формирует сигналы во внешнюю среду, то он портов не имеет. Затем объявляются переменные для формирования входных векторов *tcin, ta, tb* и получения результатов суммирования *ts, tcout*. В следующем пункте создается модель проекта в виде экземпляра модуля *add_1_1* с именем *sum*.

Эмуляция модели осуществляется с помощью процедурного блока **initial** с именем *test*, который выполняется только один раз. Здесь объявляется локальная переменная i целого типа и вызываются системные задачи: **$display** для вывода строки символов; **$timeformat** для определения формата представления времени и **$monitor** для наблюдения за указанными переменными и вывода их значений в случае изменения хотя бы одной из переменных (кроме значения времени). Затем следует процедурный оператор типа **for**, в теле которого выполняется задержка на 10 единиц времени и формируется значение входных векторов.

Поскольку в языке Verilog процедурные блоки и экземпляры модулей выполняются параллельно, то сформированное в блоке **initial** значение входного вектора (переменные *tcin, ta* и *tb*) автоматически будет передано на вход экземпляра *sum*. Время работы экземпляра *sum* не определено, поэтому принимается равным 0.0. Таким образом, значения выходных переменных *ts* и *tcont* будут сформированы без задержки, а системная задача **$monitor** выведет их значения и значения входных переменных на экран.

```
# Результаты моделирования однобитового сумматора:
# 0:   cin=x  a=x  b=x  s=x  cout=x
# 10:  cin=0  a=0  b=0  s=0  cout=0
# 20:  cin=0  a=0  b=1  s=1  cout=0
# 30:  cin=0  a=1  b=0  s=1  cout=0
# 40:  cin=0  a=1  b=1  s=0  cout=1
# 50:  cin=1  a=0  b=0  s=1  cout=0
# 60:  cin=1  a=0  b=1  s=0  cout=1
# 70:  cin=1  a=1  b=0  s=0  cout=1
# 80:  cin=1  a=1  b=1  s=1  cout=1
```

Рис. 1.5. Результаты моделирования однобитового сумматора

Задание. Сравните листинги с описанием однобитового сумматора с листингом 1.5. Как вы считаете, что занимает у проектировщика больше времени описание проекта или моделирование проекта?

Результаты моделирования однобитового сумматора приведены на рис. 1.5.

Анализируя результаты моделирования, разработчик может сделать выводы о правильности функционирования проекта и в случае необходимости внести в описание проекта соответствующие изменения.

Замечание. Анализ результатов моделирования также может выполняться с помощью языка Verilog, а на экран при этом будут выводиться только результаты такого анализа.

1.3. Базовые элементы языка Verilog

1.3.1. Ключевые слова

Предварительное знакомство с ключевыми словами является хорошей практикой при изучении языков программирования и проектирования. Знание ключевых слов необходимо хотя бы потому, что их нельзя использовать в качестве идентификаторов. В табл. 1.2 и 1.3 приводятся ключевые слова стандартов Verilog-1995 и Verilog-2001, реализованные в пакете Quartus II фирмы Altera. Если вы используете другую интегрированную среду для разработки своих проектов, то вам необходимо ознакомиться с ключевыми словами реализованной версии языка Verilog. Заметим, что внимательное прочтение ключевых слов нового языка дает некоторое представление о языке, а также вызывает определенный интерес и вопросы, ответы на которые можно найти при последующем изучении языка.

Задание. Ознакомьтесь с ключевыми словами компилятора языка Verilog используемой вами интегрированной среды проектирования.