УДК 004.01
ББК 32.972
Р69

Коллектив авторов
Антонов А. А., Барабанов А. В., Данчек Ч. Т., Жельнио С. Л.,
Иванец С. А., Кудрявцев И. А., Панчул Ю. В., Романов А. Ю.,
Романова И. И., Телятников А. А., Шуплецов М. С.

Р69 Цифровой синтез: практический курс / под общ. ред. А. Ю. Романова, Ю. В. Панчула. – М.: ДМК Пресс, 2020. – 556 с.

ISBN 978-5-97060-850-0

Книга представляет собой расширенный практический курс, ориентированный на язык Verilog и обеспечивающий возможность выполнения практических задач на дешевых отладочных платах. Этот практикум дополняет и объединяет теоретические курсы по цифровой логике, языкам описания аппаратуры, компьютерной архитектуре и микроархитектуре, а также подготавливает студентов к работе с промышленными процессорными ядрами, к созданию специализированных вычислителей (например, ускорителей нейросетей) и курсов VLSI по проектированию массовых микросхем ASIC.

Материал каждой главы можно изучать автономно. В конце глав приводятся вопросы и упражнения, позволяющие преподавателям встраивать данный материал в любой учебный курс, а читателям книги – закрепить новые знания, самостоятельно выполнив предлагаемые задания.

Издание предназначено для студентов технических вузов, разработчиков аппаратно-программных систем, а также специалистов в области прикладной математики, интересующихся алгоритмами САПР.

УДК 004.01
ББК 32.972

ISBN 978-5-97060-850-0

Оглавление

Юрий Панчул, Александр Романов

Цифровой синтез: практический курс

Введение

Verilog – не просто один из редких языков, а обязательный инструмент современного разработчика электроники

До сих пор встречаются люди, которые считают, что **Verilog** – просто один из редких языков программирования, а **ПЛИС** – устройство для очень специальных применений вроде обработки сигнала с радиотелескопа. В действительности же **Verilog** и **ПЛИС** – вход во всю современную цифровую электронику. Это так, поскольку подавляющее большинство цифровых микросхем, разработанных за последние 25 лет, использует технологию компиляции (синтеза) схем из языков описания аппаратуры, главный из которых – **Verilog**. Огромное число инженеров, которые сейчас разрабатывают микросхемы в Apple, Intel и других электронных компаниях, во время учебы в таких университетах, как Беркли и MIT, прошли через лабораторные работы с использованием учебных отладочных плат на **ПЛИС**. Такого рода практические занятия позволяют наработать опыт в технологии проектирования на уровне регистровых передач (**Register Transfer Level**, **RTL**), которая используется для создания массовых микросхем внутри популярных цифровых устройств вроде Apple iPhone.

Данный учебник – важный шаг на пути построения экосистемы разработки современной электроники в России и в других странах бывшего СССР. России предстоит пройти тот же путь, который прошли Япония, Южная Корея, Тайвань и который сейчас проходит Китайская Народная Республика. На этом пути необходимо создать большое количество групп разработчиков разной специализации, готовых слаженно работать вместе. Таких разработчиков необходимо выращивать из сегодняшних студентов:

- Некоторые из студентов после окончания университета будут специализироваться в разработке микроархитектуры процессоров, сетевых устройств и других логически сложных блоков. Они будут или сами использовать **Verilog**, или создавать модели устройств, основываясь на понимании того, как работает технология **RTL**.
- Другие студенты будут специализироваться на физическом уровне проектирования. Им придется решать проблемы физического уровня, возникающие при превращении логического графа схемы в план расположения дорожек и транзисторов на пластине кремния при ее фабричном производстве. Хотя эти инженеры не будут писать на Verilog сами, им нужно понимать основы того, как работает в пространстве и **времени** граф, который они раскладывают.
- Третья группа студентов будет специализироваться на создании программ автоматизированного проектирования (**САПР**), которые помогают работать разработчикам аппаратуры. Компании, разрабатывающие такие программы, образуют целую небольшую индустрию автоматизации проектирования (**Electronic Design Automation**, **EDA**). В этой индустрии востребованы математически мыслящие инженеры, умеющие решать алгоритмически сложные задачи, которые возникают в программах синтеза, размещения и трассировки схем, автоматического доказательства их свойств и проверки их эквивалентности высокоуровневым моделям. Этим инженерам также необходимо пони-

мать основы цифрового синтеза и программирования на языках проектирования аппаратуры (**HDL**).

- Знать основы логического проектирования электроники необходимо и создателям аппаратно-программных систем. В компьютерах и встраиваемых системах XX века аппаратура и программы были довольно сильно разделены. В XXI веке, когда повышение скорости процессоров за счет простого уменьшения размера транзисторов зашло в тупик, началось быстрое развитие специализированных вычислителей. Сначала появились графические процессоры для вычислений трехмерной графики, сейчас бурно развиваются ускорители нейронных сетей, чипы для машинного зрения и даже специализированные вычислители криптовалют. Создателям всех этих устройств необходимо понимать и программную, и аппаратную стороны вычислений.

Обоснования для создания этой книги

Данная книга – «Цифровой синтез: практический курс» – создана совместными усилиями преподавателей и инженеров из нескольких университетов и компаний не только из России, но и из Украины и США. Этот практикум – один из нескольких образовательных проектов, нацеленных на подъем электроники в странах постсоветского пространства, которые все вместе можно уже рассматривать как осознанную стратегию.

К таким проектам относятся, например:

- Симулятор **MIPT-MIPS**[1], созданный базовой кафедрой Intel в МФТИ.
- Совместный курс компьютерной архитектуры и **ПЛИС**, созданный ВМК МГУ в партнерстве с европейскими университетами.
- Курс по интернету вещей, созданный в российском отделении Samsung в партнерстве с российскими университетами.
- Учебное софт-процессорное **MIPS** ядро **schoolMIPS**[2], разработанное Станиславом Жельнио из IVA Technologies.

Предтечами создания практикума стали три проекта:

- Перевод вводного учебника Дэвида Харриса и Сары Харрис «Цифровая схемотехника и архитектура компьютера». Этот перевод сделала в 2015 году группа из сорока с лишним преподавателей российских и украинских университетов, русских сотрудников компаний в Silicon Valley (включая MIPS, AMD, Synopsys, Apple и NVidia) и российских компаний (включая НИИСИ, МЦСТ, Модуль). Начинание поддержали британская компания Imagination Technologies, образовательное отделение РОСНАНО, а также российское издательство «ДМК-Пресс». Эта книга закрыла брешь в теоретической части преподавания языков описания аппаратуры и микроархитектуры, связала их с основами цифровой логики и программированием. Данный практикум можно рассматривать как расши-

[1] https://mipt-ilab.github.io/mipt-mips/.
[2] https://github.com/MIPSfpga/schoolMIPS.

ренное продолжение и дополнение к этому курсу, ориентированное на практическое применение.

- Семинары **MIPSfpga**, организованные в 2015–2017 годах Imagination Technologies в партнерстве с российскими, украинскими и казахскими университетами (МГУ, НИУ ВШЭ (МИЭМ), МИФИ, МФТИ, МИЭТ, ИТМО, Самарский университет, Томский ТГУ, украинские КПИ и КНУ, казахский AlmaU). **MIPSfpga** – это базовая конфигурация разработанного на **Verilog** промышленного процессора **MIPS interAptiv UP**, различные варианты которого используют такие компании, как Microchip Technology, Broadcom и Байкал Электроникс. Проекты, где студенты соединяют с **MIPSfpga** свои собственные блоки и синтезируют для **ПЛИС** простые системы на кристалле, позволяют им работать с тем же кодом, с которым работают инженеры в промышленности. К сожалению, **MIPSfpga** слишком сложен для начальной демонстрации основных принципов микроархитектуры. Данный практикум содержит целую главу, посвященную проекту **schoolMIPS**, который значительно проще, чем **MIPSfpga**. При этом **schoolMIPS** позволяет студенту понять базовое устройство процессоров, работу процессорного конвейера и прерываний.
- Цикл популярных семинаров Nanometer ASIC, организованный РОСНАНО, МИСиС, КПИ и Imagination Technologies, состоялся в 2016 году. Автор этих материалов – Чарльз Данчек, преподаватель Университета Калифорнии Санта-Круз и бывший инженер Intel. Мистер Данчек написал приложение к данному практикуму. В нем рассказано, как студент, выполнивший упражнения на **ПЛИС**, сможет в будущем перейти к разработке заказных микросхем (**ASIC**, **Application Specific Integrated Circuits**). К **ASIC** относятся практически все микросхемы, которые применяются в массовых электронных устройствах.

Юрий Панчул,
разработчик процессорных ядер MIPS и ускорителя нейросетей Wave,
Саннивейл, Калифорния

История создания данной книги

История создания книги «Цифровой синтез: практический курс» традиционно для таких проектов не проста. После успеха переводной версии учебника «Цифровая схемотехника и архитектура компьютера» на русском языке стало ясно, что необходимо его продолжение в виде расширенного практического курса, ориентированного на **Verilog** и обеспечивающего возможность выполнения практических задач на дешевых отладочных платах. Ясно было также и то, что одному человеку (и даже коллективу университетской кафедры) создать такой курс – непосильная задача, при том что издание книги требовало финансирования. У истоков идеи написания данной книги стоял Юрий Панчул, сумевший объединить для ее воплощения множество преподавателей и инженеров из России, Украины и США. Второй фактор, который помог появиться книге, – это знакомство Юрия Панчула (в рамках семинаров Nanometer ASIC) со мной, Александром Романовым, руководителем Учебной лаборатории Систем автоматизированного проектирования Московского института электроники и математики Национального университета «Высшая школа экономики» (УЛ САПР НИУ ВШЭ). Несмотря на экономический уклон НИУ ВШЭ, МИЭМ в прошлом был отдельным техническим вузом, готовившим студентов в области электроники и математики. Сейчас же, опираясь на необходимые технические и людские ресурсы, МИЭМ является одним из ведущих центров компетенций, в том числе и по цифровому синтезу. Идея создания учебника получила поддержку на уровне руководства в лице Евгения Аврамовича Крука, после чего началась работа по написанию книги, сбору и редактированию глав от разных авторов, рецензированию и разработке дополнительных материалов.

Чем эта книга отличается от других?

Особенность данной книги в том, что во всех главах каждый пример кода сопровождается листингом и тестбенчем, которые находятся в дополнительных материалах к книге (https://github.com/RomeoMe5/DDLM). Это позволяет читателю легко использовать уже готовые исходные коды, проводить моделирование, прототипирование на учебной плате с ПЛИС и модифицировать работающие примеры программ.

Немного про отладочные платы: поскольку команда, разрабатывавшая книгу, находилась территориально в разных местах, одна из целей курса состояла в том, чтобы сделать его максимально доступным для людей разного достатка. Изначально книга ориентирована на плату **De10-Lite** от компании Terasic на основе **ПЛИС MAX 10K** производства **Intel FPGA** (в прошлом – компания Altera). Выбор платы обусловлен ее относительно небольшой стоимостью (около $55 по академической цене) и доступностью, а также тем, что она обладает достаточной периферией, функционалом, емкостью ресурсов и даже может быть интегрирована с платформой Arduino. Платы на основе **ПЛИС Intel FPGA** (**Altera**) популярны в России и ближнем зарубежье и имеются во многих академических организациях. При этом исходные коды примеров могут быть легко перенесены и на другие платы. Поскольку данный проект ориентирован на широкое сообщество, в ближайшее время планируется миграция примеров кодов на другие попу-

лярные отладочные платы на основе **ПЛИС**: **Марсоход**, **Terasic De1-SoC**, **Terasic De10-Standard**, **Digilent Nexys 3/4** и др. Для выполнения работ не требуется какого-либо платного программного обеспечения, так как **Quartus Prime Lite Edition** + **Modelsim** / **GTKWave** распространяются свободно. Таким образом, чтобы приступить к изучению цифрового синтеза, достаточно иметь только эту книгу; при желании увидеть результаты не только моделирования, но и прототипирования понадобится какая-либо отладочная плата на основе **ПЛИС**.

Содержание книги

Еще одна особенность книги – то, что авторы представили ее главы в виде отдельных независимых разделов. То есть можно изучать тот раздел, который необходим сейчас, и обращаться к другим главам по необходимости. Каждую главу в основном писали один-два автора, после чего редактировали и рецензировали другие люди. Вот почему каждая глава имеет свой неповторимый оригинальный стиль, приведенный, впрочем, к единому оформлению и терминологии. Таким образом, в случае если одна глава воспринимается читателем тяжело, то, возможно, другая будет читаться им намного легче.

Поскольку книга задумана как практикум, ее подразделы сопровождаются заданиями для самостоятельной проработки. В конце каждой главы приводятся вопросы и упражнения, позволяющие преподавателям встраивать данный материал в любой учебный курс, а читателям книги – закрепить новые знания, самостоятельно выполнив предлагаемые задания.

Рассмотрим кратко по главам содержание книги:

Глава 1. Основы комбинационной логики. Маршрут разработки цифровых схем
Глава знакомит читателя с типичным циклом разработки цифровой системы на примере проектирования простой комбинационной схемы, которая содержит всего несколько логических вентилей. Сначала демонстрируется, как описать цифровую схему с помощью графического редактора. Далее та же схема проектируется с использованием языка описания аппаратуры. После этого демонстрируются этапы моделирования и прототипирования.

Глава 2. Основы последовательностной логики. Управление энергопотреблением цифровой схемы
Данная глава посвящена разработке последовательностных устройств. Рассматриваются защелки и триггеры на примерах их различных реализаций на языке **Verilog**. Глава позволяет понять, чем комбинационная логика отличается от последовательностной и как можно управлять энергопотреблением цифровых устройств на этапе их проектирования.

Глава 3. Шифраторы и дешифраторы. Скорость работы комбинационных блоков
В этой главе приводятся примеры реализаций таких важных комбинационных блоков, как шифраторы и дешифраторы, а также дается методика оценки временных характеристик цифровых блоков и их оптимизации.

Глава 4. Мультиплексор, демультиплексор и селектор. Построение иерархических модулей
В главе на примере разработки различных вариантов реализаций таких комбинационных блоков, как мультиплексор, демультиплексор и селектор, демонстрируется иерархический подход к проектированию цифровых устройств. Также вводятся понятие параметризации модулей и конструкция **generate**. В конце главы демонстрируются некоторые приемы использования мультиплексоров на практических примерах.

Глава 5. Сумматор, компаратор, устройство сдвига и АЛУ. Повышение скорости арифметических операций
В главе рассматриваются примеры всевозможных реализаций комбинационной арифметики (сумматоров, компараторов, устройств сдвига и АЛУ на их основе). Отдельный раздел посвящен повышению скорости арифметических блоков на этапе их проектирования.

Глава 6. Последовательностная логика. Счетчики и сдвиговые регистры
В данной главе изложение возвращается к последовательностной логике и особенностям ее разработки на **Verilog** (блокирующие/неблокирующие присвоения, понятие защелок и т. д.) на примере разработки счетчиков и сдвиговых регистров. В конце главы даны примеры организации взаимодействия цифровых систем с простыми периферийными модулями.

Глава 7. Память: регистровый файл и стек
Эта глава посвящена различным вариантам реализации памяти: регистровая память, однопортовая/многопортовая память, стек, очередь и т. д. В конце главы приводится небольшой пример проектирования на **HDL** памяти с привязкой к библиотекам фабрик-производителей **ASIC**.

Глава 8. Конечные автоматы: основы
В главе приводятся основные понятия и приемы для проектирования конечных автоматов. Иллюстрируются особенности проектирования конечных автоматов Мили и Мура и рассматриваются наиболее оптимальные случаи их использования. Также демонстрируется использование специальных инструментов для проектирования и анализа конечных автоматов.

Глава 9. Использование конечных автоматов для связи с периферийными устройствами
Эта глава расширяет тему проектирования конечных автоматов путем формализации академического подхода к проектированию автоматов; также демонстрируются и другие автоматы, например на основе счетчика. Глава обосновывает применение конечных автоматов в проектировании цифровых устройств как ключевого блока управления ими.

Глава 10. Конвейерная обработка данных
Глава посвящена описанию конвейерного подхода к обработке данных и особенностям разработки на **Verilog**. Приводится сравнение комбинационного, многотактного и конвейерного подходов на примере разработки арифметического

блока; описываются дополнительные приемы повышения эффективности конвейерных схем.

Глава 11. Софт-процессор: основы микроархитектуры
Данная глава объединяет в себе необходимые знания из предыдущих разделов этой книги при проектировании **HDL**-реализации относительно простого, но при этом реально функционирующего, однотактного софт-процессорного ядра с архитектурой **MIPS**. В главе даются понятия микроархитектуры, тракта данных, устройства управления и т. д.; демонстрируется процесс добавления новых инструкций и расширения процессорного ядра и его блоков.

Приложение А. Путь вперед: от FPGA-устройств к массовому рынку ASIC для популярных гаджетов
Приложение дает общее представление об этапах проектирования чипов **ASIC**, начиная с идеи специализированного чипа и заканчивая его конечной реализацией на кристалле. Это первый шаг в подготовке инженеров, прошедших обучение проектированию на **ПЛИС**, к применению имеющихся у них навыков в разработке **ASIC** для конкретных приложений.

Приложение Б. История успеха победы российской команды на международном конкурсе InnovateFPGA от Intel
Это приложение – небольшое интервью с участником российской команды, занявшей второе место на одном из основных международных конкурсов по проектированию на **ПЛИС** – **InnovateFPGA** от Intel.

Таким образом, данный практикум по **Verilog** и **ПЛИС** дополняет и объединяет теоретические курсы по цифровой логике, языкам описания аппаратуры, компьютерной архитектуре и микроархитектуре. Практикум также подготавливает студентов к работе с промышленными процессорными ядрами, к созданию специализированных вычислителей (например, ускорителей нейросетей) и курсов **VLSI** по проектированию массовых микросхем **ASIC**. Он будет полезен разработчикам аппаратно-программных систем, а также прикладным математикам, интересующимся алгоритмами **САПР**.

Выражаем надежду на то, что практикум станет такой же надежной основой курсов по цифровой электронике для большого количества университетов России и стран СНГ, какой уже стал переводной учебник «Цифровая схемотехника и архитектура компьютера» авторов Дэвида и Сары Харрис. В результате этого в России появится новое поколение инженеров, способных объединять лучшие мировые практики с изобретательностью, присущей народам наших стран; так Россия вместе со своими соседями займет достойное место в мировой промышленной электронике.

Александр Юрьевич Романов,
к. т. н., доцент МИЭМ НИУ ВШЭ,
преподаватель курсов «Проектирование систем на кристалле»
и «Системное проектирование цифровых устройств»,
г. Москва, Россия

Данный учебник позволяет первично познакомиться с цифровым дизайном через изучение языка Verilog и правильных design-style и best practices

Прочтение этой книги вызвало у меня воспоминания об относительно далеком прошлом. В те времена, будучи студентом факультета технической информатики Мангеймского университета в Германии, я имел первый опыт программирования на языке **Verilog** и загрузки программного кода в **ПЛИС**. Самой сложной задачей была реализация программы для рисования, и самые продвинутые студенты рисовали круги. Помнится, что студенты одной из групп смогли разработать элементарный процессор и запрограммировать отрисовку кругов на ассемблере для этого процессора. Эти студенты стали теперь профессорами в университетах Германии.

Купив в 1990 году компанию Gateway Design Automation, в которой работал изобретатель языка **Verilog** Фил Мурби, компания Cadence Design Systems стала правообладателем **Verilog**. Этот язык изначально разрабатывался для описания микросхем для цифрового симулятора. Со временем **Verilog** также стал использоваться для синтеза, то есть превращения описания микросхемы на более абстрактном уровне **Register-Transfer-Logic RTL** в менее абстрактный **Gate-Level**, где описываются вентили, которые предоставляют конкретные изготовители микросхем (такие как: TSMC, UMC – Тайвань, GlobalFoundries – США). Этих изготовителей принято называть foundries; российские foundries – Ангстрем и Микрон в Зеленограде.

Verilog настолько успешен, что существуют несколько языков, которые имеют с **Verilog** общее название, но используются для разных целей. Например, **Verilog-A** – для моделирования аналоговых микросхем, **Verilog-AMS** – для моделирования цифроаналоговых микросхем, **SystemVerilog** – для верификации больших цифровых микросхем.

Конкретно этот учебник позволяет первично ознакомиться с цифровым дизайном, но охватывает при этом более широкие аспекты – обучает не только самому языку, но правильному **design-style** и **best practices**, что чрезвычайно важно для недопущения ошибок при подготовке к синтезу. Даже самое умное ПО не может синтезировать оптимальный **PPA** (**power**, **performance**, **area** – три самых важных показателя качества микросхемы), если исходный код (даже будучи грамматически правильным) не соблюдал определенные **design-styles**. Таким образом, данный учебник обладает двумя весомыми преимуществами: во-первых, он включает в себя базовую информацию, поиск которой требует, как правило, больших затрат времени и сил (проблемы выбора подходящего и, по возможности, бесплатного ПО и совместимой **ПЛИС**, проблемы их установки/настройки), и во-вторых, может использоваться для самообучения как студентами, так и научными сотрудниками, поскольку снабжен всеми необходимыми материалами от подготовленных исходных кодов программ для моделирования и синтеза до презентаций к каждой главе.

Компания Cadence (сотрудником которой я являюсь уже в течение 16 лет) выпускает программное обеспечение, применяемое на самых различных стадиях

проектирования и разработки комплексных микросхем, а также целых систем (приборы с печатными платами, **RF**-компонентами и несколькими системами на кристалле (**SoC**) в одной упаковке). Хотя ПО компании Cadence существенно облегчает процесс проектировки систем на кристалле с миллиардами элементов, именно талантливые и хорошо обученные инженеры являются главным капиталом компаний – разработчиков микросхем. Поэтому Cadence в высокой степени заинтересована в обучении следующего поколения инженеров-микроэлектронщиков, так как видит в них и будущих разработчиков, и будущих клиентов. В 2007 году была создана Cadence Academic Network (группа внутри компании Cadence), поддерживающая связи с ведущими университетами и предоставляющая им академические лицензии для доступа к ПО. Следует отметить, что данное программное обеспечение используется разработчиками самых передовых компаний мира, и Cadence Academic Network распространяет учебные материалы для обучения этим комплексным программам.

Компания Cadence приветствует появление такой нужной книги на российском рынке и будет рада контактам с читателями, усвоившими азы разработки цифровых микросхем и готовыми к дальнейшему обучению. Хотелось бы пожелать всем читателям этого учебника успехов и новых познаний.

Антон Клотц,
University Program Manager
Cadence Design Systems

Точно так же, как до этого в России стала массовой профессия программиста, данный курс поможет сделать массовой профессию электронного инженера – разработчика цифровой аппаратуры любой сложности

История развития мировой вычислительной техники насчитывает уже более 70 лет и является прямым отражением той жестокой конкурентной борьбы сверхдержав и их союзников, которую они ведут за контроль над мировой экономикой, финансовыми и товарными потоками с целью получения долгосрочных преимуществ перед остальными странами. Кроме того, разработка быстродействующих компьютеров является неотъемлемой частью технологического соперничества нынешних сверхдержав – США и Китая. Галопирующее развитие и широкое внедрение массовых коммуникаций и встроенных микрокомпьютеров в контексте интернета вещей приводит к тотальной цифровизации экономики.

Технологическая зависимость в области электронной компонентной базы может вызвать катастрофические последствия в случае разного рода санкций и ограничений на поставки со стороны зарубежных партнеров.

Несмотря на то что цифровая электронная инженерия была невостребованной в российской промышленности и устойчиво деградировала с 1990-х гг., в настоящее время наличие собственных инженерных компетенций в данной области на массовом уровне становится условием выживания любой страны, претендующей даже на частичный технологический суверенитет.

Учебное пособие «Цифровой синтез: практический курс» как комплекс практических работ является очередным этапом в создании массовой школы цифровой электронной инженерии на всем пространстве СНГ. Этому предшествовало издание профильных учебников и проведение серий семинаров, нацеленных на создание практических работ по различным разделам проектирования цифровых схем и микропроцессоров, о чем подробно написано Юрием Панчулом в предисловии.

Различные лабораторные работы по проектированию и синтезу цифровых схем на языках регистровых передач для описания аппаратуры стали необходимой частью подготовки электронных инженеров – проектировщиков микросхем для массовых электронных изделий. Через подобный практикум проходят будущие создатели смартфонов, разработчики автомобильной электроники и процессоров для различных встроенных применений, включая электронику для космических зондов. В США известным примером такого практического курса является 6.111 из MIT[1], а также различные варианты лабораторных заданий с **Verilog**/**VHDL** и **ПЛИС**/**FPGA** предусмотрены учебными программами практически для всех, кому преподают электронику, включая студентов местных университетов в небольших штатах.

Авторы надеются, что данный курс поможет сделать массовой профессию электронного инженера – разработчика цифровой аппаратуры любой сложности. Точно так же, как до этого в России стала массовой профессия программиста или

[1] http://web.mit.edu/6.111/volume2/www/f2018/index.html.

программного инженера – разработчика систем и приложений. Основами языков регистровых передач или **Register Transfer Level** (**RTL**) нужно владеть не только самим разработчикам, но и представителям смежных профессий – верификаторам, специалистам по физическому проектированию, а также алгоритмистам, которые пишут инструментальные программы для разработчиков электроники. Программисты встроенных систем и программ искусственного интеллекта тоже получат пользу от понимания того, как работают классические процессоры, GPU и нейроускорители.

Данное пособие создано усилиями международного авторского коллектива специалистов из ведущих университетов и ИТ-компаний. Помимо традиционных основ проектирования цифровой техники, авторы ввели в курс дополнительные проекты, которых не хватало в имеющихся курсах. Они включают развернутое объяснение принципов конвейерных вычислений, введение в микроархитектуру процессоров, а также детальные разъяснения о том, как студент может использовать свой опыт, полученный от лабораторных работ на ПЛИС, в своей дальнейшей карьере. Полученные практические навыки обеспечивают статус разработчика массовых изделий на специализированных полузаказных микросхемах **ASIC** (**Application Specific Integration Circuits**) и крупных систем на кристалле (СнК), которых так остро не хватает в отечественной индустрии.

Выражаю уверенность в том, что это пособие имеет очень хороший потенциал для его последующего перевода на английский язык и массового использования в кооперации с компаниями, разрабатывающими инструментарий электронной инженерии. Такими партнерами могут стать крупные компании Cadence, Synopsys и Mentor Graphics.

С пожеланиями успехов авторам и пользователям данного учебного пособия,

Тимур Турсунович Палташев,
доктор технических наук, профессор,
руководитель академических проектов,
Radeon Technology Group,
Корпорация Advanced Micro Devices.
17 марта 2020 года

Книга «Цифровой синтез: практический курс» является, по сути, продолжением классического учебника по проектированию микроэлектроники «Цифровая схемотехника и архитектура компьютера» Дэвида Харриса и Сары Харрис, новая редакция которого (на русском языке) вышла всего несколько лет назад

Рецензируемая книга представляет собой расширенный практический курс, ориентированный на **Verilog** и обеспечивающий возможность выполнения практических задач на широкодоступных отладочных платах **FPGA**. Именно такой подход позволяет эффективно организовать подготовку квалифицированных разработчиков для отечественной микроэлектроники. Следует отметить, что данное издание, подготовленное международным авторским коллективом русскоязычных специалистов из ведущих университетов и ИТ-компаний, обладает всем необходимым потенциалом для последующего перевода на английский язык и дальнейшего использования в университетской среде.

История развития систем автоматизированного проектирования берет свое начало в 70–80-х годах прошлого века. К тому времени сложность систем, возраставшая по мере увеличения количества транзисторов на микросхеме по закону Мура, стала такой, что ручное проектирование схемотехники микроэлектронных устройств становилось практически невозможным. Проектирование аппаратуры повторило историю развития языков программирования: если первые вычислительные системы программировались на уровне кодов, то уже в 50-е годы приходилось приступать к разработке систем автоматизации программирования и, соответственно, языков программирования высокого уровня. По тому же пути (несколько позднее) пошли и разработчики аппаратуры – для проектирования новых вычислительных систем использовалось программное обеспечение, разработанное для уже существующих компьютеров. Но если для проектирования печатных плат устройств уже в середине 80-х годов существовали специализированные САПР, то к промышленному проектированию схемотехники микроэлектронных изделий удалось приступить только в начале 90-х (при том что сам **Verilog** был разработан в середине 80-х). Однако первоначально он был ориентирован на описание и моделирование логических схем; применение его для синтеза на уровне логических элементов и гейтов было реализовано лишь с ростом популярности основанных на нем средств моделирования и отладки.

Знание **Verilog** и использование его обширного инструментария с целью проектирования цифровых систем сегодня является абсолютно необходимым навыком для любого инженера-электронщика (кроме разве что специалистов в аналоговой и силовой электронике, где количество элементов весьма ограничено, зато чрезвычайно важны их физические характеристики). В микроэлектронике ситуация прямо противоположная – все цифровые системы строятся из огромного (как правило, исчисляемого миллионами, а иногда десятками и сотнями миллионов) количества физически одинаковых элементов. Поэтому для инженера-схемотехника необходимы инструменты иерархической абстракции, позволяющие манипулировать логикой крупных модулей и строить из них системы, не опускаясь на уровень отдельных транзисторов и имея при этом возможность анализировать их поведение во времени. Именно такую возможность и предоставляет **Verilog**

с его инструментарием, зачастую поддерживающий и другие языки описания аппаратуры (в частности, **VHDL**).

В течение достаточно длительного времени одной из основных проблем отладки и тестирования микроэлектронной аппаратуры являлись технологические процессы производства микросхем. Ситуация коренным образом изменилась с появлением в середине 80-х и широким распространением в 90-е годы прошлого века технологии **FPGA** (**ПЛИС** в русскоязычной литературе) – программируемых логических матриц, состоящих из сотен тысяч и миллионов одинаковых индивидуально программируемых гейтов. В то же время именно **Verilog** и другие языки описания аппаратуры сделали эффективным проектирование и использование систем на **FPGA**. И если в 90-е годы они использовались в основном в телекоммуникационном и сетевом оборудовании, то сегодня трудно представить большую цифровую систему, не использующую **FPGA** в качестве акселераторов тех или иных процессов. Эта технология оказалась чрезвычайно удачным компромиссом между эффективностью, сравнимой с чисто аппаратными решениями (хотя и уступающей им), и гибкостью, простотой использования, характерной для программного обеспечения.

Таким образом, **FPGA** стали активно применяться в тестировании блоков схемотехники или целых цифровых систем, ориентированных на дальнейшую реализацию в микроэлектронном исполнении (**ASIC**). Хотя до сих пор не существует полностью автоматизированного процесса переноса **Verilog**-программы из **FPGA**-прототипа на технологический процесс конкретного изготовителя, этот процесс существенно повышает эффективность и снижает стоимость отладки аппаратных решений микроэлектроники. Для упрощения процесса прототипирования на **FPGA** все основные производители стали выпускать готовые отладочные платы, включающие (кроме самой микросхемы **FPGA**) все необходимое окружение, в том числе стандартные интерфейсы с компьютером; отпала нужда в специализированных программаторах, логических анализаторах и другом инженерном оборудовании стадии отладки.

В то же время с массовым распространением микросхем **FPGA**, отладочных плат на их основе и радикальным падением стоимости за счет эффекта масштаба стало возможным использовать их для учебных задач. Современные инженерные курсы в области микроэлектроники, как правило, используют **FPGA** в качестве основного инструмента проверки и тестирования программ на **Verilog** даже при ориентации на создание в дальнейшем **ASIC** и **SoC** (систем на кристалле).

Предлагаемый в книге «Цифровой синтез: практический курс» практикум по инструментам и технологиям цифрового синтеза схемотехники полностью охватывает все основные разделы цифровой схемотехники, а также используемый при проектировании бесплатный инструментарий. Покрывается практически весь материал, представленный в учебнике «Цифровая схемотехника и архитектура компьютера» Дэвида Харриса и Сары Харрис. Важной особенностью рецензируемой книги является то, что во всех главах каждый пример кода сопровождается исходным кодом и тестбенчем, которые находятся в дополнительных материалах к книге.

В заключение хотелось бы отметить, что необходимость в подготовке квалифицированных кадров в области проектирования микроэлектроники является вполне объективной – востребованность таких кадров будет только расти в связи с массовым распространением устройств интернета вещей и переводом все большего объема функционала на системы на кристалле. Если в последние десятилетия основным фокусом при обучении специалистов по цифровым системам была подготовка программистов и (позднее) аналитиков данных, то в ближайшие годы будут все более востребованы инженеры со специализацией на стыке программирования и аппаратуры, специалисты по киберфизическим системам и электронике; в нашей стране их готовят в ограниченном количестве учебных заведений, при этом ощущается существенный дефицит опыта и литературы. Представляется важным, чтобы МИЭМ НИУ ВШЭ стал одним из лидеров этого направления в российской академической среде; таким образом, издание рецензируемой книги в «ДМК Пресс» – ведущем издательстве, специализирующемся на выпуске компьютерной и радиотехнической литературы, – является важным шагом в этом направлении.

Игорь Рубенович Агамирзян,
вице-президент НИУ ВШЭ,
профессор факультета компьютерных наук НИУ ВШЭ,
канд. физ.-мат. наук

Сергей Иванец, Александр Романов

Цифровой синтез: практический курс

Глава 1. Основы комбинационной логики. Маршрут разработки цифровых схем

Содержание

*Глава посвящена основам цифрового дизайна и знакомит с логическими вентилями – основными элементами цифровых систем. Вначале описывается проектирование простой схемы, содержащей всего несколько логических вентилей, с помощью графического редактора. Далее спроектирована та же схема, но с использованием языка описания аппаратуры (**Hardware Description Language**, **HDL**). Спроектированная схема проверяется с помощью симулятора – специальной программы для тестирования цифровых схем. Для того чтобы увидеть, как работает схема «в железе», программируется микросхема **ПЛИС** (**Программируемая логическая интегральная схема, Field-Programmable Gate Array, FPGA**). Выполнив все описанные шаги, вы познакомитесь с типичным циклом разработки цифровой системы.*

Требования к аппаратным и программным средствам

Для выполнения практических работ понадобится следующее программное обеспечение:

- *персональный компьютер с установленной операционной системой Windows (виртуальная машина с ОС Windows не подойдет), x64, 8GB RAM, USB port;*
- *пакет **Quartus Prime Lite Edition 17.0**[1];*
- *пакет **ModelSim Altera Edition**;*
- *программы **Icarus Verilog** и **GTKWave**[2].*

*Программы Quartus и ModelSim являются платными, но они имеют и студенческие бесплатные версии, которые могут быть свободно скачаны с сайта производителя **ПЛИС Altera** (**Intel FPGA**).*

*Также в данном практикуме используется отладочная плата компании **Terasic DE10Lite**[3]. Она содержит микросхему **ПЛИС** компании **Intel FPGA MAX10**[4] (**10M50DAF484C7G**). В папке doc дополнительных материалов к настоящей главе* (https://github.com/RomeoMe5/DDLM, lab_01/doc) *размещены инструкция к данной отладочной плате и ее электрическая схема (эти же документы могут быть бесплатно загружены с сайта компании **Terasic**).*

*Хотя в этом практикуме используется отладочная плата с микросхемой **MAX10** от компании **Intel FPGA**, концепции и методологии, которые вы узнаете при выполнении работ, могут быть использованы и при работе с **ПЛИС** от других производителей, например **Xilinx**. Однако следует учитывать то, что инструменты для проектирования и микросхемы быстро развиваются, и последние версии **САПР** компании **Xilinx** (**Vivado Design Suite**)[5] больше не поддерживают схематический редактор, а только разработку на основе языков описания аппаратуры.*

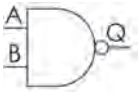
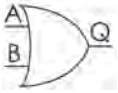
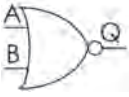
---

1 http://dl.altera.com/?edition=lite.
2 Инсталлятор можно найти в папке **pkg** материалов к данной главе.
3 http://de10-lite.terasic.com/.
4 https://www.altera.com/products/fpga/max-series/max-10/overview.html.
5 https://www.xilinx.com/support/answers/53764.htm.

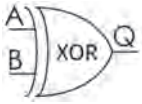
1.1. Краткие теоретические сведения

Рассмотрим цикл разработки **комбинационного устройства**, схема которого будет содержать логические вентили – **И**, **ИЛИ**, **НЕ**, а также **исключающее ИЛИ**. Особенностью комбинационных схем является то, что они выполняют только заданную логическую функцию над входными сигналами, но не сохраняют их значения. В следующей главе также рассмотрены **последовательностные** устройства, которые содержат элементы для хранения значений, и их состояние поэтому может зависеть не только от текущего набора входных сигналов, но и от предыстории. Логические вентили являются теми основными «кирпичиками», с помощью которых строятся все остальные элементы цифровых систем – от простых элементов, таких как дешифратор или триггер, до самых сложных – процессоров и систем на кристалле (**system-on-chip**, **SoC**).

Для построения устройства на основе логических элементов необходимо определить логические функции, которые описывают требуемые логические операции. В таблице, приводимой ниже, показаны основные логические элементы, их обозначения, уравнения и таблицы истинности.

Вентиль	Символ	Уравнение	Таблица истинности
НЕ (NOT)		$Q = \overline{A}$	A | Q 0 | 1 1 | 0
И (AND)		$Q = A \cdot B = A \& B$	A | B | Q 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1
И-НЕ (NAND)		$Q = \overline{A \cdot B} = \overline{A \& B}$	A | B | Q 0 | 0 | 1 0 | 1 | 1 1 | 0 | 1 1 | 1 | 0
ИЛИ (OR)		$Q = A + B$	A | B | Q 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1
ИЛИ-НЕ (NOR)		$Q = \overline{A + B}$	A | B | Q 0 | 0 | 1 0 | 1 | 0 1 | 0 | 0 1 | 1 | 0

Исключающее ИЛИ (XOR)



$Q = A \oplus B$

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

На начальном этапе для разработки принципиальной схемы будет использован схемотехнический редактор. Этот метод создания цифровых систем использовался в начале 1980-х годов и позднее был вытеснен языками описания аппаратуры. Описание схемы на языке описания аппаратуры с помощью компилятора синтезируется в принципиальную схему, которая в дальнейшем реализуется либо на микросхеме **ПЛИС**, либо с помощью специализированной микросхемы (**Application-Specific Integrated Circuit**, **ASIC**). Такой подход позволяет значительно увеличить скорость создания цифровых устройств и обеспечивает существование наиболее передовых современных разработок в виде систем на кристалле, которые используются в мобильных телефонах, планшетах и других устройствах. Данный практический курс в первую очередь ориентирован на создание цифровых устройств с помощью языков описания аппаратуры.

Еще одним преимуществом использования языков описания аппаратуры является увеличение скорости отладки проектов. Это происходит потому, что исправлять схему гораздо дольше, чем код, ее описывающий. Кроме того, HDL содержат в своем составе конструкции, предназначенные для написания отладочных тестов (**test bench**), которые используются для симуляции цифровых устройств, что еще больше ускоряет отладку и верификацию проектов.

На рис. 1.1 приведены основные операции, необходимые для создания устройства с использованием **ПЛИС**. Рассмотрим их более подробно. Сначала необходимо создать спецификацию или техническое задание – документ, содержащий полный список требований к устройству. Затем выполняется разработка с помощью языков описания аппаратуры. На этом этапе можно использовать библиотеки, в которых могут быть описаны блоки различной сложности. Далее выполняется верификация проекта в симуляторе и синтез списка связей (**netlist**). Все описанные выше этапы относят к **front end** разработке микросхем или проектов на **ПЛИС**. **Back end** процесса разработки, часто называемый **place and route**, включает в себя размещение элементов на кристалле и разводку межсоединений. База данных разводки (например, **GDSII**-файл), полученная после этого этапа, может быть использована производителем чипов для изготовления специализированной микросхемы.

Поскольку изготовление микросхем является весьма дорогостоящей операцией, в данном практическом курсе предлагается более дешевый метод – с использованием **ПЛИС**. **ПЛИС** – это специализированная микросхема, содержащая матрицу ячеек с реконфигурируемой логической функцией. Ячейка может быть сконфигурирована для выполнения различных логических функций: одна для выполнения операции **И**, другая – операции **ИЛИ** и т. д. Функции ячеек и их соединение между собой могут быть изменены и записаны в специальной конфигурационной памяти **ПЛИС**.

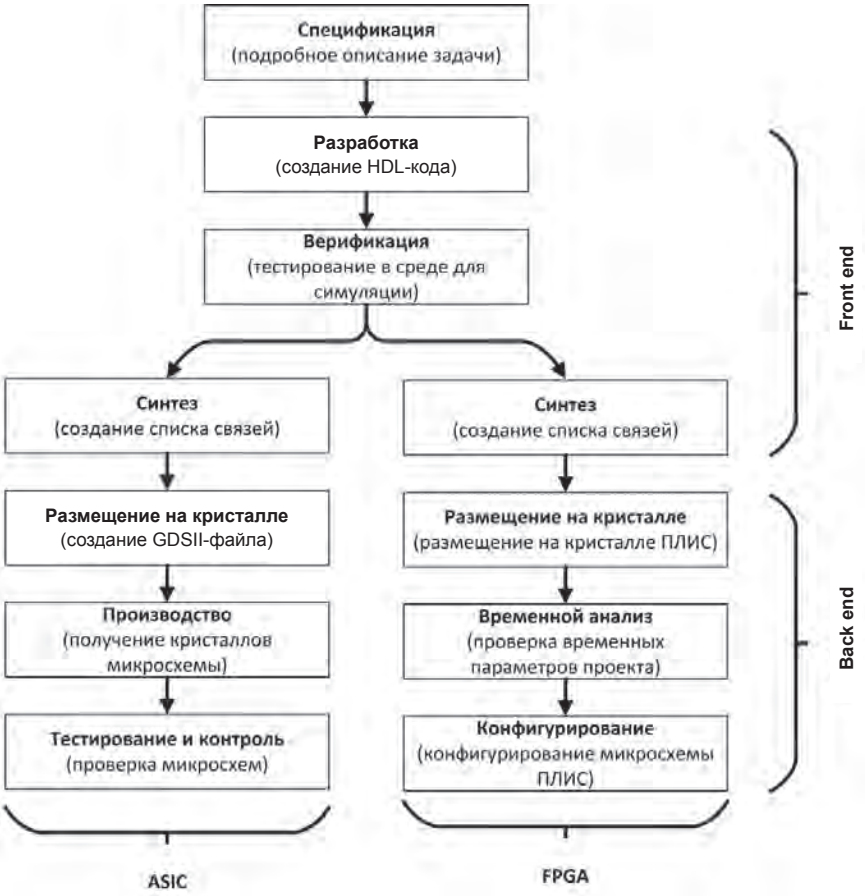


Рис. 1.1 Порядок разработки проектов для ASIC и FPGA

Ключевой концепцией данного практического курса является анализ и учет на этапе проектирования временных параметров устройства. Хотя сигнал через логический вентиль проходит очень быстро (данный параметр исчисляется единицами и десятыми долями наносекунды), задержка сигнала внутри вентиля не является нулевой. Таким образом, сигналу необходимо какое-то время для того, чтобы пройти со входа на выход. На рис. 1.2 показана задержка распространения сигнала через буфер, то есть повторитель сигнала. Данный параметр для одинаковых микросхем может варьироваться в зависимости от различных факторов: от максимального значения **tpd** (**propagation delay**) до минимального **tcd** (**contamination delay**). Для последовательно соединенных нескольких вентилей результирующая задержка представляет собой сумму задержек отдельных вентилей, и ее значение может превышать максимально допустимое. Далее при выполнении работ будет показано, что инструменты синтеза учитывают разброс значений задержки, суммируют задержку на нескольких вентилях и т. д. Задача разработчика – определить реалистичные временные параметры проекта. Если тактирование схемы не выполняется, то разработчик должен разделить проект

на несколько небольших частей, с которыми он будет работать во время анализа временных параметров. Далее в последующих главах будет введено понятие тактового, или синхронизирующего, сигнала, который используется для упрощения синхронизации сложных последовательностных схем путем размещения фрагментов логики между регистрами.

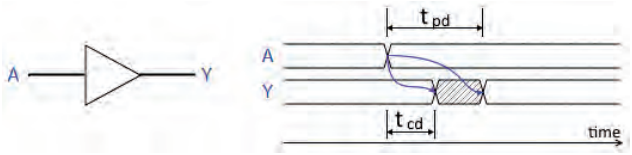


Рис. 1.2 Минимальная и максимальная задержки

1.2 Использование схемотехнического редактора

Quartus Prime – это интегрированная среда разработки, которая используется для проектирования на основе **ПЛИС** от компании **Intel FPGA**. Проекты могут отличаться по своей сложности – от простого управления светодиодом до сложной системы на кристалле, содержащей одно или несколько процессорных ядер. Quartus содержит инструменты для разработки проекта, его синтеза, размещения на кристалле и разводки межсоединений (данная операция носит название **place-and-route** или **fitting** для проектов на **ПЛИС**), генерации конфигурационного файла и загрузки этого файла в микросхему на плате.

Конфигурирование **ПЛИС** – это процесс записи определенной последовательности бит в конфигурационную память микросхемы, то есть программирования ее на выполнение определенной логической функции. Поэтому загрузку конфигурации в микросхему **ПЛИС** часто называют программированием. Важно отличать это от понятия программирования как процесса разработки программ.

1.2.1 Установка пакета Quartus Prime

Существует несколько версий пакета Quartus Prime:

- **Quartus Prime Lite Edition**[1] – полностью бесплатная версия пакета **Quartus Prime**;
- **Quartus Prime Pro Edition** или **Quartus Prime Standard Edition** – коммерческая версия пакета.

Quartus Prime Lite Edition поддерживает все функции, необходимые для разработки проекта на **ПЛИС**, и поэтому в данном практическом курсе будет использоваться именно эта версия программы. Необходимую версию пакета **Quartus Prime Lite Edition** можно скачать с сайта компании **Intel FPGA**. В настоящем практическом курсе будет использована версия 17.0 данного пакета. Различие между версиями пакета описано в документации на **Quartus Prime**[2]. При скачивании не-

[1] https://fpgasoftware.intel.com/?edition=lite.
[2] https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/po/ss-quartus-comparison.pdf.

обходимо позаботиться о том, чтобы скачать не только сам пакет **Quartus Prime Lite Edition**, но и поддержку микросхемы **MAX 10**, а также пакет ModelSim – **Intel FPGA Edition**.

В процессе установки **Quartus Prime Lite Edition** необходимо обратить внимание на несколько параметров:

- папку, в которую будет устанавливаться пакет (желательно, чтобы ее имя было на латинице и без пробелов);
- поддерживаемые семейства **ПЛИС**. Поддержка различных семейств и определенных микросхем зависит от версии пакета, поэтому перед его установкой следует убедиться в том, что необходимая микросхема поддерживается выбранной версией пакета. Иногда необходимо устанавливать более старые версии Quartus для работы с семействами микросхем, выпускавшимися ранее;
- устанавливаемые компоненты. Настоятельно рекомендуется выбирать пункт **Modelsim – Intel FPGA Starter Edition** при инсталляции. Это позволит установить бесплатную версию пакета Modelsim – мощного симулятора цифровых систем, используемого в промышленности.

Основные этапы установки приведены на рис. 1.3, 1.4, 1.5, 1.6, 1.7.

Рис. 1.3 Начало установки Quartus Prime Lite Edition

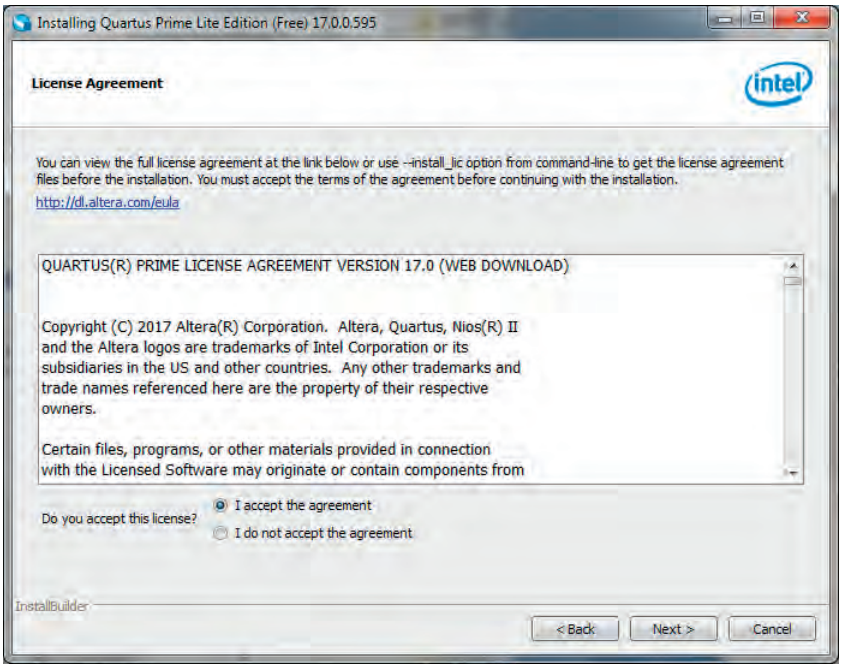


Рис. 1.4 Соглашение о лицензии в Quartus Prime Lite Edition

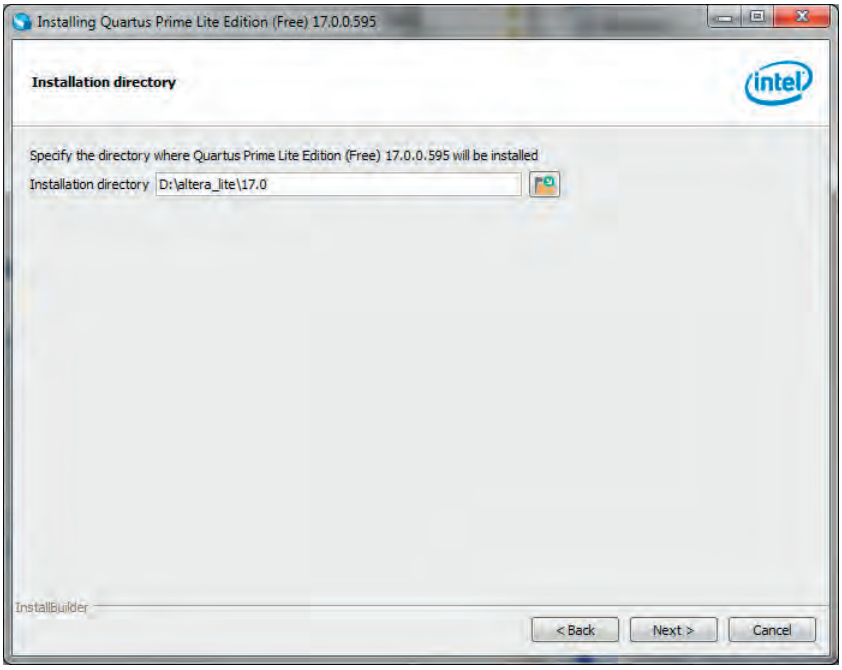


Рис. 1.5 Назначение папки, в которую будет установлен Quartus Prime Lite Edition

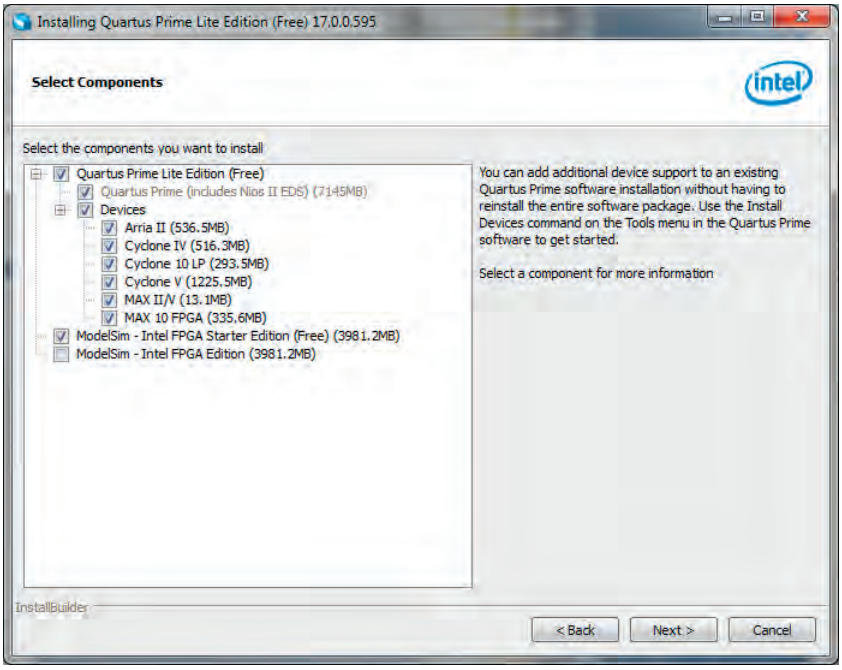


Рис. 1.6 Выбор компонентов Quartus Prime Lite Edition и поддерживаемых семейств микросхем

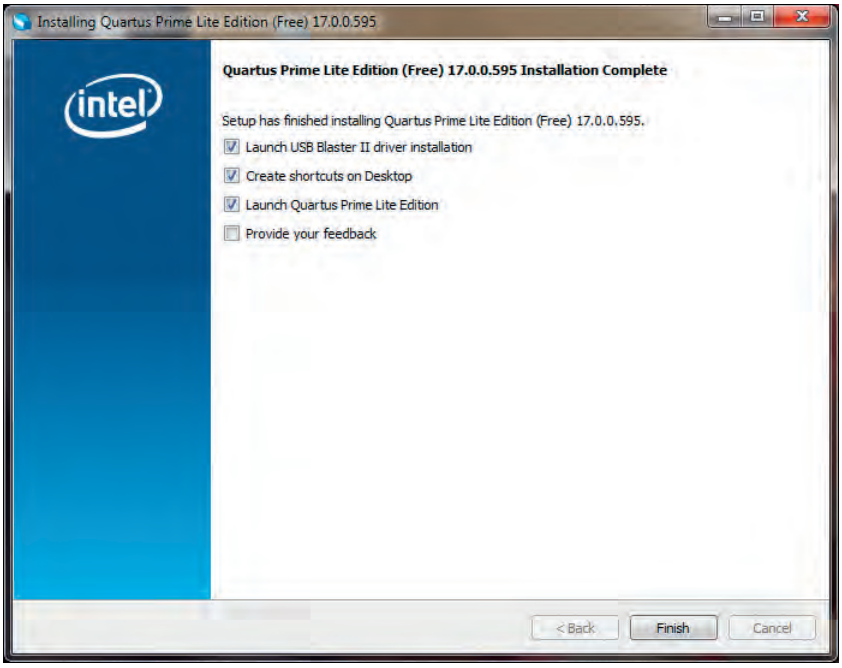


Рис. 1.7 Окончание установки Quartus Prime Lite Edition

1.2.2 Создание проекта в схемотехническом редакторе

Quartus Prime рассматривает любую схему независимо от ее размера (будь то несколько вентилей или система на кристалле) как проект (**project**). Каждый проект соответствует папке, в которой он находится. Все файлы проекта располагаются в папке проекта независимо от того, созданы ли они разработчиком или являются результатом работы компилятора, поэтому рекомендуется использовать разные папки для разных проектов. **Quartus Prime** может работать одновременно только с одним проектом.

Перед созданием нового проекта необходимо определить некоторые условия:

- папку, в которой будет располагаться проект на диске;
- имя проекта. Недопустимо использование русских букв в имени проекта или пути к нему, а также символов ^ & ? * < >;
- имя объекта верхнего уровня, то есть имя модуля, который вы разрабатываете либо в виде схемы, либо описываете на языке описания аппаратуры;
- целевое семейство микросхем (в нашем случае это **MAX10**);
- дополнительные файлы библиотек;
- другие необходимые файлы, например файлы тестбенчей.

Для создания нового проекта выберите пункт меню **File → New Project Wizard...** (рис. 1.8).

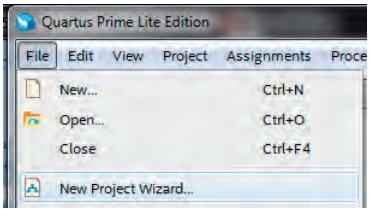


Рис. 1.8 Запуск мастера нового проекта New Project Wizard

Примечание: пункт меню **File → New...** (или кнопка 🗋) создает новый файл, а не новый проект.

Запускается помощник, который помогает создать проект и ввести всю необходимую информацию. Рассмотрим все этапы создания нового проекта по шагам.

Первый шаг при создании проекта – это определение путей для размещения проекта и его имени (рис. 1.9).

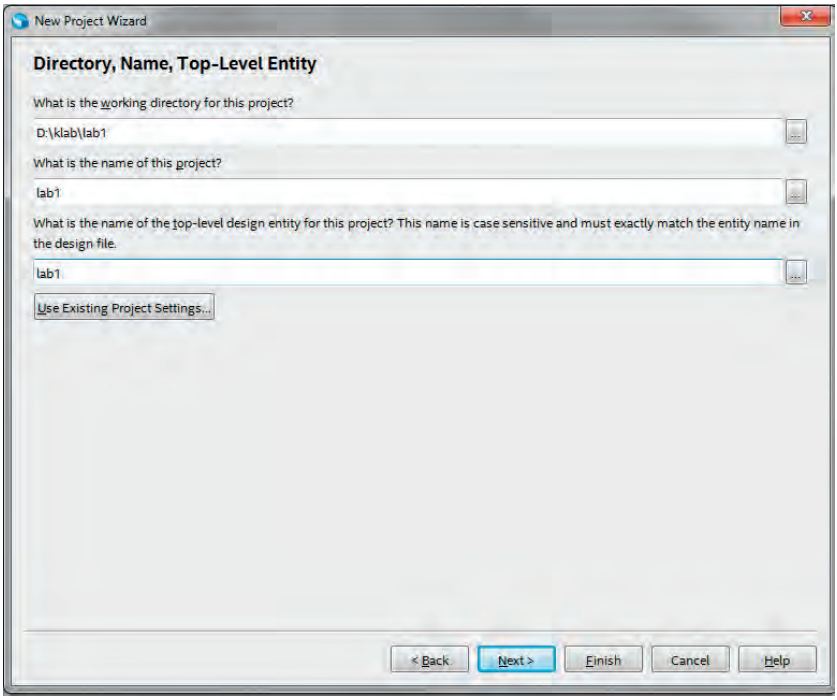


Рис. 1.9 New Project Wizard. Шаг 1

What is the working directory for this project? – Определите имя папки, в которой будет размещаться ваш проект. **Quartus Prime** создает большое количество файлов во время работы, поэтому строго рекомендуется хранить разные проекты в разных папках.

What is the name of this project? – Определите имя проекта. Часто разработчики устанавливают имя проекта таким же, как и имя файла верхнего уровня. Это является хорошим тоном.

What is the name of the top-level design entity for this project? – Определите имя файла верхнего уровня, который может быть как схемой, так и файлом на языке описания аппаратуры. Следует обращать внимание на то, что имя чувствительно к регистру.

Use Existing Project Settings... – Использовать параметры уже существующего проекта или нет.

На втором шаге (рис. 1.10) необходимо выбрать **Empty project**, если для создания проекта не используется шаблон.

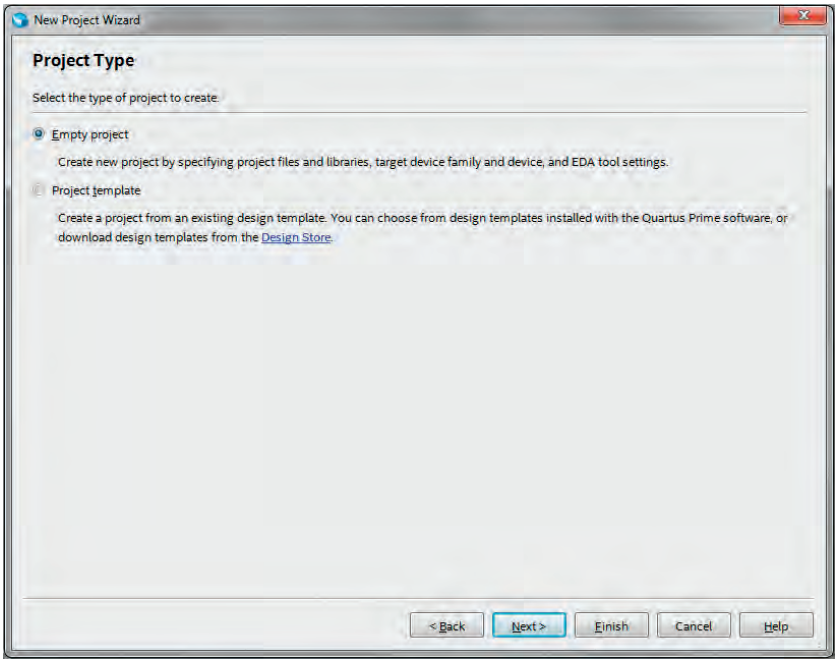


Рис. 1.10 New Project Wizard. Шаг 2

Диалог, приведенный на рис. 1.11, определяет дополнительные библиотеки или файлы пользователя, которые могут быть подключены к проекту. К проекту могут быть добавлены файлы таких типов: графические (**.bdf**, **.gdf**), файлы с описанием на языках описания аппаратуры (**VHDL**, **Verilog**, **SystemVerilog**), файлы **EDIF**-формата. При этом нет необходимости добавлять файлы, находящиеся в рабочей папке проекта. Также можно подключить к проекту библиотеки пользователя с помощью кнопки **User Libraries...**.

В этой практической работе не нужны дополнительные файлы, поэтому данный шаг можно пропустить.

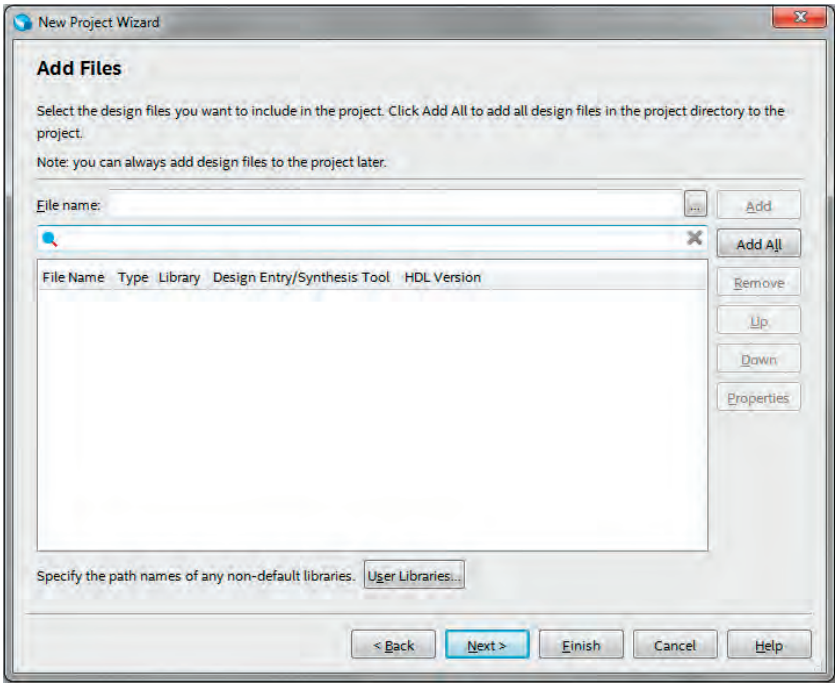


Рис. 1.11 New Project Wizard. Шаг 3

В следующем окне следует задать нужное семейство микросхем и конкретную микросхему из этого семейства (рис. 1.12). Так как работа ориентирована на плату **Terasic DE10-Lite FPGA**, нужно выбирать микросхему **10M50DAF484C7G**, как это показано на рисунке. Выбор микросхемы конкретно с такими параметрами очень важен, так как в обозначении микросхемы задаются количество вентилей, тип корпуса (**Package**), количество выводов (**Pin count**) и градация скорости (**Core speed grade**), поэтому другая микросхема из этого же семейства будет отличаться от нужной, что может привести к совершенно иным результатам компиляции. Кроме того, если выбрать не ту микросхему, это не позволит сконфигурировать микросхему на отладочной плате. Временные параметры микросхемы, используемые при компиляции проекта, определяются параметром **speed grade**. Этот параметр не описывает задержку сигнала в микросхеме, а служит для обозначения более быстрых или более медленных микросхем. Так, микросхема со **speed grade 4** быстрее микросхемы со **speed grade 5**.

Замечание: используйте фильтр при поиске (**Show in 'Availble devices' list**), чтобы упростить поиск микросхемы в списке.

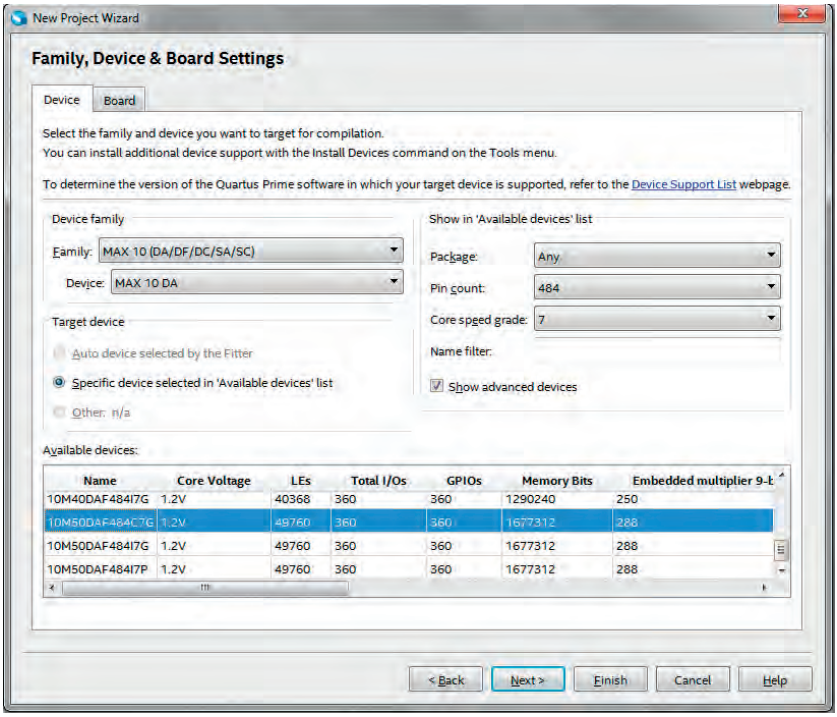


Рис. 1.12 New Project Wizard. Шаг 4

Следующий диалог определяет нестандартные средства для отладки, верификации и синтеза (рис. 1.13). Здесь ничего менять не нужно.

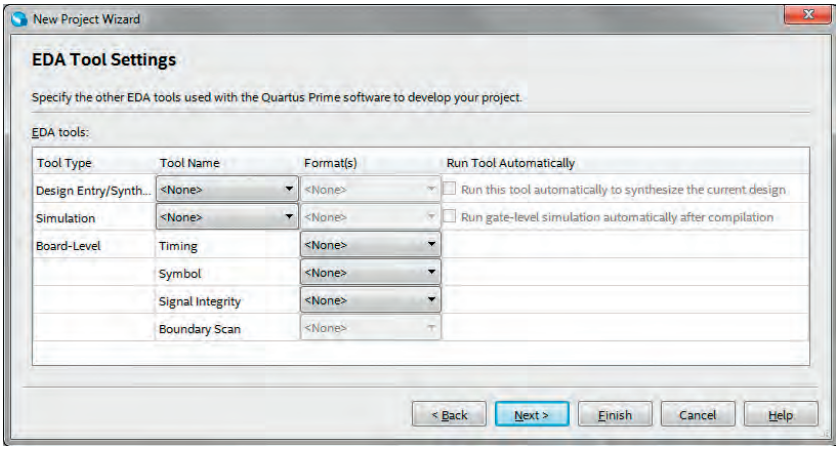


Рис. 1.13 New Project Wizard. Шаг 5

Последний диалог выводит суммарную информацию о выбранной конфигурации нового проекта (рис. 1.14). Это окно не требует никаких действий и служит для проверки корректности установок, заданных на предыдущих шагах. Если все сде-

Илья Кудрявцев, Ирина Романова,
Александр Романов, Юрий Панчул

Цифровой синтез: практический курс

Глава 2. Основы последовательностной логики. Управление энергопотреблением цифровой схемы

Содержание

Александр Барабанов, Александр Романов

Цифровой синтез: практический курс

Глава 3. Шифраторы и дешифраторы. Скорость работы комбинационных блоков

Содержание

Александр Романов, Юрий Панчул

Цифровой синтез: практический курс

Глава 4. Мультиплексор, демультиплексор и селектор. Построение иерархических модулей

Содержание

*Глава содержит базовые сведения о различных способах реализации мультиплексоров, демультиплексоров и селекторов, об особенностях грамотного «стиля разработки кода» (**coding style**) на **Verilog**, а также приводятся примеры возможных ошибок, которые разработчики могут допускать, и пути их исправления. Вводится понятие модульности, приведен пример использования директив компилятора, а также показано, как создавать параметризированные модули с помощью параметров.*

Требования к аппаратным и программным средствам

Для выполнения практических работ вам понадобится следующее программное и аппаратное обеспечение:

- *персональный компьютер с установленной операционной системой Windows (виртуальная машина с ОС Windows не подойдет), x64, 8GB RAM, USB port;*
- *пакет **Quartus Prime** (есть студенческая версия);*
- *пакет **ModelSim Altera Edition** или программы **Icarus Verilog** и **GTKWave**;*
- *отладочная плата компании **Terasic DE10Lite** или другая отладочная плата на основе **ПЛИС Intel FPGA** или **Xilinx** (может потребоваться миграция проектов, если она еще не сделана в дополнительных материалах[1] к данной книге).*

4.1 Проектирование мультиплексоров

Мультиплексором (**MUX**) называют комбинационное логическое устройство, предназначенное для управления передачей данных от нескольких источников на один выходной канал. В соответствии с определением мультиплексор должен иметь один выход и два типа входов (информационный и адресный). Код, поступающий на адресный вход, определяет, какой из информационных входов в данный момент подключен к выходу. Если количество адресных входов мультиплексора равно **n**, то максимально возможное количество его информационных входов будет $\mathbf{2^n}$. Такой мультиплексор называют **полным**, а если информационных входов меньше – мультиплексор **неполный**.

Следует хорошо понимать, что мультиплексор является комбинационным устройством, а не последовательностным. Это значит, что изменение сигналов на входе мультиплексора непосредственно влечет изменение на выходе (через время, определяемое задержкой распространения сигнала).

4.1.1 Однобитный мультиплексор 2в1

Рассмотрим различные способы создания мультиплексоров на примере двухвходового однобитного мультиплексора. Мультиплексор имеет два однобитных информационных входа **d0** и **d1**, один адресный вход **sel** и однобитный выход, на который коммутируются **d0** или **d1** в зависимости от **sel**.

Поскольку устройство комбинационное, то его всегда можно описать в виде комбинационной функции в базисе «**И**», «**ИЛИ**», «**НЕ**». В рассматриваемом примере функция имеет следующий вид:

[1] https://github.com/RomeoMe5/DDLM.

$$y = (sel \ \& \ d1) \mid ((\sim sel) \ \& \ d0).$$

Пример простейшей реализации мультиплексора на языке **Verilog** приведен ниже:

```
module b1_mux_2_1_comb
(
   input d0,
   input d1,
   input sel,
   output y
);
   assign y = (sel & d1) | ((~sel) & d0);
endmodule
```

Листинг 4.1 Комбинационный мультиплексор 2в1

Если количество входов мультиплексора и их разрядность больше, чем в приведенном примере, то синтез комбинационной функции, описывающей мультиплексор, становится нетривиальной задачей. Поэтому мультиплексоры обычно реализуют другими способами.

Одно из возможных решений – использование тернарного оператора (оператора выбора). Оно продемонстрированно в следующем примере:

```
module b1_mux_2_1_sel
(
   input d0,
   input d1,
   input sel,
   output y
);
   assign y = sel ? d1 : d0;
endmodule
```

Листинг 4.2 Мультиплексор 2в1 на основе тернарного оператора

Этот же мультиплексор можно реализовать и с помощью условного оператора (**if**):

```
module b1_mux_2_1_if
(
   input d0,
   input d1,
   input sel,
   output reg y
);
   always@(*)
      begin
         if(sel)
            y = d1;
         else
```

```
            y = d0;
      end
endmodule
```

Листинг 4.3 Мультиплексор 2в1 на основе условного оператора

Наиболее предпочтительной реализацией является использование оператора множественного выбора (**case**), поскольку он позволяет проще всего описывать сложные мультиплексоры для большого количества входов:

```
module b1_mux_2_1_case
(
   input d0,
   input d1,
   input sel,
   output reg y
);
   always@(*)
      begin
         case (sel)
            0: y = d0;
            1: y = d1;
         endcase
      end
endmodule
```

Листинг 4.4 Мультиплексор 2в1 на основе оператора множественного выбора

Предлагаемые реализации мультиплексоров выполняют одну и ту же функцию, в чем можно убедиться, проведя моделирование:

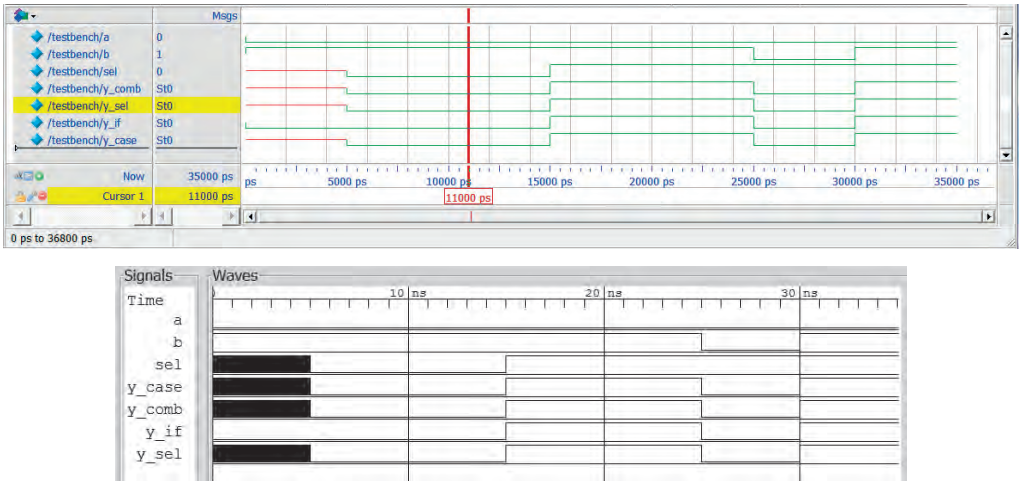


Рис. 4.1 Результаты моделирования различных реализаций однобитных мультиплексоров 2в1

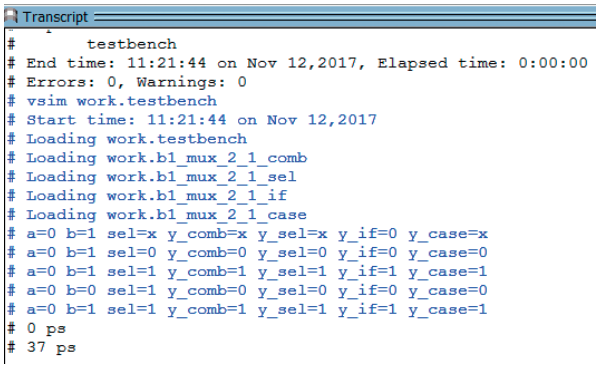


Рис. 4.1 (Продолжение)

Содержимое тестбенча (тестового окружения) приведено ниже:

```
`timescale 1 ns / 100 ps
// testbench is a module which only task is to test another module
// testbench is for simulation only, not for synthesis
module testbench;
   // input and output test signals
   reg a;
   reg b;
   reg sel;
   wire y_comb;
   wire y_sel;
   wire y_if;
   wire y_case;

   // creating the instance of the module we want to test
   b1_mux_2_1_comb b1_mux_2_1_comb (a, b, sel, y_comb);
   b1_mux_2_1_sel b1_mux_2_1_sel (a, b, sel, y_sel);
   b1_mux_2_1_if b1_mux_2_1_if (a, b, sel, y_if);
   b1_mux_2_1_case b1_mux_2_1_case (a, b, sel, y_case);

   initial
      begin
         a = 1'b0;
         b = 1'b1;
         #5;
         sel = 1'b0; // sel change to 0; a -> y
         #10;
         sel = 1'b1; // sel change to 1; b -> y
         #10
         b = 1'b0; // b change; y changes too. sel == 1'b1
         #5
         b = 1'b1;
         #5; // pause
```

```
      end
   // do at the beginning of the simulation
   // print signal values on every change
   initial
      $monitor("a=%b b=%b sel=%b y_comb=%b y_sel=%b y_if=%b y_case=%b",
           a, b, sel, y_comb, y_sel, y_if, y_case);
   // do at the beginning of the simulation
   initial
      $dumpvars; //iverilog dump init
endmodule
```

Листинг 4.5 Тестбенч для моделирования различных реализаций однобитных мультиплексоров 2в1

Различия заключаются лишь в том, что мультиплексор на основе оператора **if** интерпретирует неопределенный сигнал **sel (1'bx)** как **0** и соответственно коммутирует **0**-й вход на выход, когда в остальных случаях на выходе – неопределенное состояние.

Несмотря на то что мультиплексоры работают одинаково, их **RTL**-реализация (схемотехническое представление после конвертации в **RTL**-код) в **Quartus Prime** отличается:

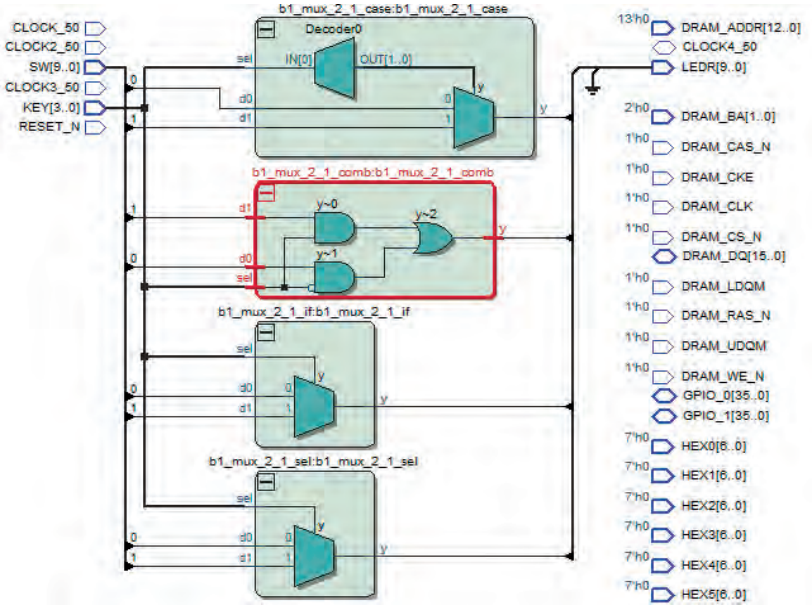


Рис. 4.2 RTL-представление различных реализаций однобитных мультиплексоров 2в1

Комбинационный мультиплексор представляется в виде соединения логических элементов, а **case** – в виде мультиплексора, на вход которого поступает декодированный сигнал **sel**. **IF** и **SELECT** реализации мультиплексоров представляются одинаково в виде мультиплексора.

Тем не менее логика работы всех реализаций мультиплексоров одинакова, а при синтезе любой из мультиплексоров будет реализован одинаково как часть комбинационного блока.

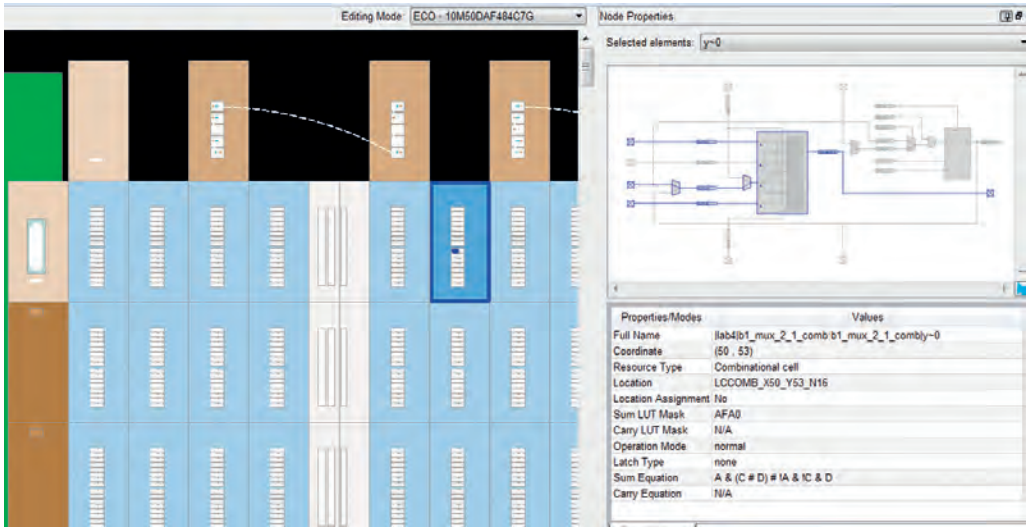


Рис. 4.3 Мультиплексор 2в1 в Chip Planner Quartus Prime

Дополнительное задание для самостоятельной работы

Исследуйте работу и проведите моделирование еще одной версии однобитного мультиплексора, реализованного с помощью операции конкатенации:

```
module b1_mux_2_1_concate
(
   input d0,
   input d1,
   input sel,
   output y
);
   wire [1:0] dataIn;
   assign dataIn = {d1,d0};
   assign y = dataIn[sel];
endmodule
```

Листинг 4.6 Мультиплексор 2в1, реализованный с помощью операции конкатенации

4.1.2 Двухбитный мультиплексор 2в1

В рассматриваемых выше примерах входы мультиплексора являлись однобитными, если же их размерность выше (например, 2), следует учитывать, что такие входы являются шинами.

В случае реализации мультиплексоров, использующих операторы **if** или **case**, сложностей не возникает, поскольку размеры входов и выходов просто увеличиваются:

```
module b2_mux_2_1_sel
(
   input [1:0] d0,
   input [1:0] d1,
   input sel,
   output [1:0] y
);
   assign y = sel ? d1 : d0;
endmodule

module b2_mux_2_1_if
(
   input [1:0] d0,
   input [1:0] d1,
   input sel,
   output reg [1:0] y
);
   always@(*)
      begin
         if(sel)
            y = d1;
         else
            y = d0;
      end
endmodule

module b2_mux_2_1_case
(
   input [1:0] d0,
   input [1:0] d1,
   input sel,
   output reg [1:0] y
);
   always@(*)
      begin
         case (sel)
            0: y = d0;
            1: y = d1;
         endcase
      end
endmodule
```

Листинг 4.7 Двухбитный мультиплексор 2в1, реализованный с помощью различных операций

С комбинационным мультиплексором существует опасность сделать грубую ошибку, которая сделает работу мультиплексора некорректной.

```
module b2_mux_2_1_comb_incorrect
(
   input [1:0] d0,
   input [1:0] d1,
   input sel,
   output [1:0] y
);
   assign y = (sel & d1) | ((~sel) & d0);
endmodule
```

Листинг 4.8 Некорректный комбинационный двухбитный мультиплексор 2в1

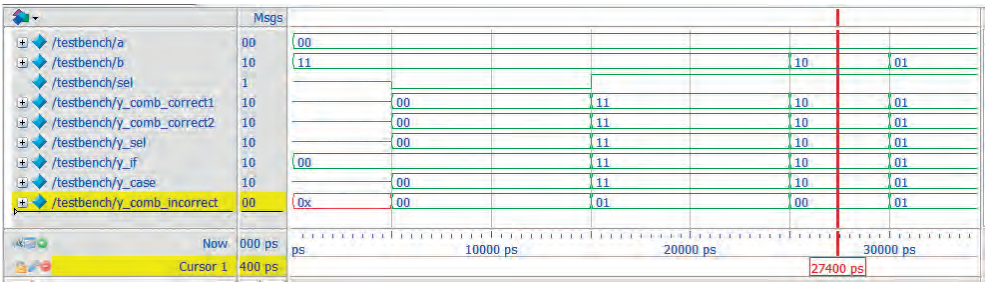


Рис. 4.4 Результаты моделирования двухбитного мультиплексора 2в1

Следует отметить, что в данном примере используются побитовые операции, в связи с чем надо следить за размерностью участвующих в них операндов, чтобы не получить неожиданный результат.

Примеры правильной реализации комбинационного мультиплексора приведены ниже:

```
module b2_mux_2_1_comb_correct1
(
   input [1:0] d0,
   input [1:0] d1,
   input sel,
   output [1:0] y
);
   assign y[0] = (sel & d1[0]) | ((~sel) & d0[0]);
   assign y[1] = (sel & d1[1]) | ((~sel) & d0[1]);
endmodule

module b2_mux_2_1_comb_correct2
(
   input [1:0] d0,
   input [1:0] d1,
   input sel,
```

```
   output [1:0] y
);
   wire [1:0] select;
   assign select = {2{sel}};
   assign y = (select & d1) | (~select & d0);
endmodule
```

Листинг 4.9 Правильный комбинационный двухбитный мультиплексор 2в1

Дополнительное задание для самостоятельной работы

Синтезируйте вышеупомянутые реализации мультиплексоров в **Quartus Prime**. Рассмотрите их представление в **RTL Viewer**, объясните, в чем состоит ошибка в модуле **b2_mux_2_1_comb_incorrect** (листинг 4.8).

4.1.3 Двухбитный мультиплексор 4в1

Когда количество входов увеличивается, увеличивается также и ширина входного сигнала **select**. Размер такого сигнала зависит от количества информационных входов и может определяться уравнением: $N_{sel} = [\log_2 N]$.

Такие мультиплексоры легче всего реализуются с помощью оператора **case**:

```
module b2_mux_4_1_case
(
   input [1:0] d0, d1, d2, d3,
   input [1:0] sel,
   output reg [1:0] y
);
   always @(*)
      case (sel)
         2'b00: y = d0;
         2'b01: y = d1;
         2'b10: y = d2;
         2'b11: y = d3;
      endcase
endmodule
```

Листинг 4.10 Двухбитный мультиплексор 4в1, реализованный с помощью оператора case

Ниже представлена еще одна версия двухбитного мультиплексора 4в1, реализованного на основе тернарного оператора:

```
module b2_mux_4_1_sel
(
   input [1:0] d0, d1, d2, d3,
   input [1:0] sel,
   output [1:0] y
);
```

```
   assign y = sel [1] ? (sel [0] ? d3 : d2)
        : (sel [0] ? d1 : d0);
endmodule
```

Листинг 4.11 Двухбитный мультиплексор 4в1, реализованный с помощью условного оператора

4.1.4 Иерархический подход при проектировании цифровых систем

Использование тернарного оператора с увеличенным количеством входов приводит к трудночитаемому коду. Улучшить читаемость кода можно путем использования иерархического подхода, в котором более сложные модули построены на более простых модулях. Этот метод является общим и применяется не только для мультиплексоров.

Рассмотрим следующую реализацию двухбитного мультиплексора 4в1, построенного на основе двухбитного мультиплексора 2в1 (листинг 4.2), разработанного в разделе 4.1.1:

```
module b2_mux_4_1_block
(
   input [1:0] d0, d1, d2, d3,
   input [1:0] sel,
   output [1:0] y
);
   wire [1:0] w01, w23;
   b2_mux_2_1_sel mux0(.d0(d0), .d1(d1), .sel(sel[0]), .y(w01));
   b2_mux_2_1_sel mux1(.d0(d2), .d1(d3), .sel(sel[0]), .y(w23));
   b2_mux_2_1_sel mux2(.d0(w01), .d1(w23), .sel(sel[1]), .y(y));
endmodule
```

Листинг 4.12 Двухбитный модульный мультиплексор 4в1

Результаты моделирования всех представленных вариантов мультиплексоров совпадают:

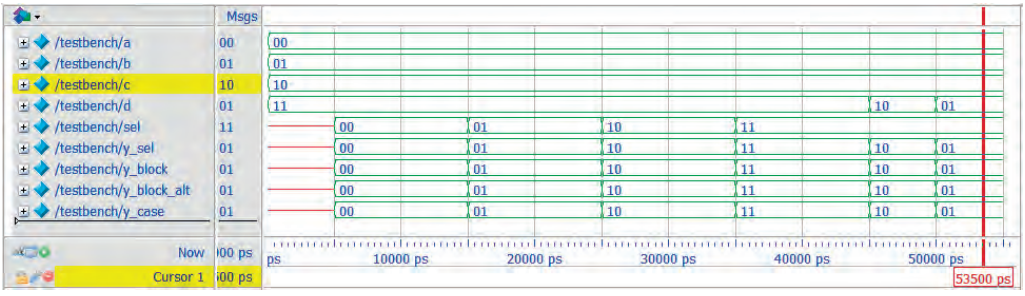


Рис. 4.5 Результаты моделирования различных реализаций мультиплексоров 4в1

Результаты синтеза различных реализаций мультиплексоров существенно различаются:

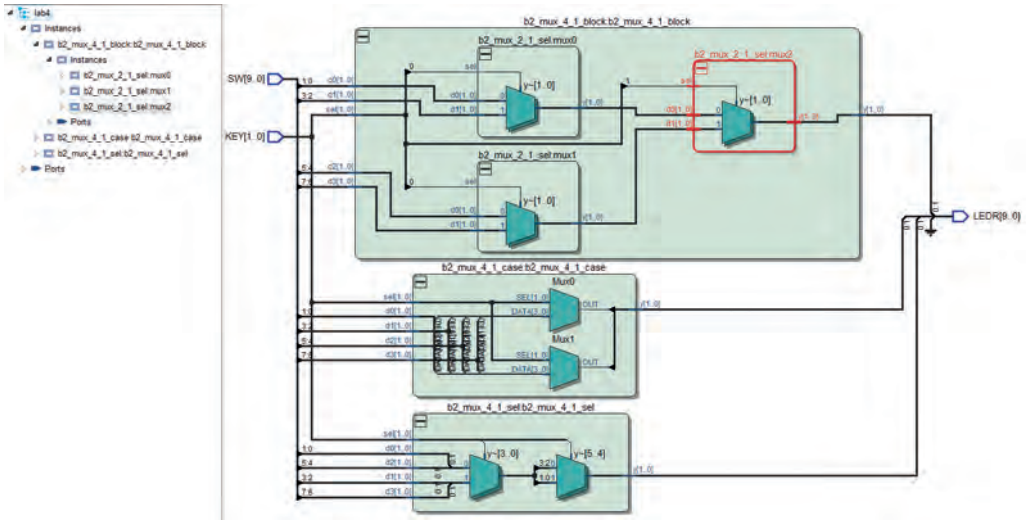


Рис. 4.6 RTL-схема различных реализаций мультиплексоров 4в1

Еще один способ улучшения кода, специфичный для мультиплексоров, – это их реализация с использованием оператора case. Можно построить еще одну альтернативную реализацию модульного двухбитного мультиплексора 4в1 на основе однобитного мультиплексора 4в1:

```
module b1_mux_4_1_case
(
   input d0, d1, d2, d3,
   input [1:0] sel,
   output reg y
);
   always @(*)
      case (sel)
         2'b00: y = d0;
         2'b01: y = d1;
         2'b10: y = d2;
         2'b11: y = d3;
      endcase
endmodule

module b2_mux_4_1_block_alt
(
   input [1:0] d0, d1, d2, d3,
   input [1:0] sel,
   output [1:0] y
);
```