А. В. Строгонов

РЕАЛИЗАЦИЯ ЦИФРОВЫХ УСТРОЙСТВ В БАЗИСЕ ПРОГРАММИРУЕМЫХ ЛОГИЧЕСКИХ ИНТЕГРАЛЬНЫХ СХЕМ

Учебное пособие

Под редакцией доктора физико-математических наук,
профессора С. И. Рембезы

Ай Пи Эр Медиа
Саратов • 2019

УДК 621.3
ББК 32.97
С86

Автор:
Строгонов А. В. — д-р техн. наук, профессор кафедры
полупроводниковой электроники и наноэлектроники ВГТУ

Рецензенты:
Бормонтов Е. Н. — д-р физ.-мат. наук, профессор,
зав. кафедрой физики полупроводников и микроэлектроники ВГУ;
Ромащенко М. А. — д-р техн. наук, доцент

Строгонов, А. В.

С86 Реализация цифровых устройств в базисе программируемых логических интегральных схем [Электронный ресурс] : учебное пособие / А. В. Строгонов; под ред. С. И. Рембезы. — Электрон. дан. и прогр. (12 Мб). — Саратов : Ай Пи Эр Медиа, 2019. — 151 с.

ISBN 978-5-4486-0541-3

В учебном пособии рассматривается использование программных инструментов с открытым исходным кодом для реализации цифровых устройств, представленных Verilog-кодом в базис академических ПЛИС. Даются практические примеры проектирования цифровых устройств с использованием синтезатора Odin-II САПР VTR, синтезатора Yosys в маршрутах проектирования с использованием индустриальных ПЛИС и заказных БИС Qflow. Затрагиваются вопросы программирования ПЛИС серии 5578 с использованием среды разработки конфигурационных данных АО «КТЦ «ЭЛЕКТРОНИКА».

Издание соответствует требованиям Федерального государственного образовательного стандарта высшего образования по направлению подготовки 11.04.04 «Электроника и наноэлектроника» (магистерская программа «Приборы и устройства в микро- и наноэлектронике») и предназначено для изучения дисциплины «Программируемые логические интегральные схемы».

Учебное электронное издание

Редактор *К.С. Досмухамбетова*
Технический редактор, компьютерная верстка *Ю.Ю. Желтова*
Корректор *А.В. Мисенжников*
Обложка *С.С. Сизиумовой*

Для создания электронного издания использовано:
Приложение pdf2swf из ПО Swftools, ПО IPRbooks Reader,
разработанное на основе Adobe Air

Подписано к использованию 14.01.2019. Объем данных 12 Мб.

Издание представлено в электронно-библиотечных системах
IPR BOOKS (www.iprbookshop.ru),
Библиокомплектатор (www.bibliocomplectator.ru)

Бесплатный звонок по России: **8-800-555-22-35**
Тел.: 8 (8452) 24-77-97, 8 (8452) 24-77-96

Отдел продаж и внедрения ЭБС:
доб. 206, 213, 144, 145
E-mail: sales@iprmedia.ru

Отдел комплектования ЭБС:
доб. 224, 227, 208
E-mail: mail@iprbookshop.ru

По вопросам приобретения издания обращаться:
доб. 208, 201, 222, 224
E-mail: izdat@iprmedia.ru, author@iprmedia.ru

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Большинство современных проектов цифровых схем создается с использованием языков описания аппаратуры (hardware description language — HDL), таких как Verilog или VHDL. Для преобразования HDL-кода в цифровые схемы применяют программные инструменты синтеза с открытым программным кодом, такие как Yosys и Odin-II.

Академический САПР VTR (Verilog to Routing) с открытым исходным кодом является совместной разработкой трех университетов: Торонто (Канада, Торонто), Нью-Брансвик (Канада, UNB), Калифорнийский университет в Беркли (США, University of California, Berkeley) и предназначен для реализации цифровых устройств в базисе академических ПЛИС (программируемые логические интегральные схемы) типа FPGA с одноуровневой структурой трассировочных ресурсов. Академический САПР VTR предполагает использование следующих программных ин-струментов: ODIN-II, ABC и VPR.

В первой главе рассматриваются архитектура академических ПЛИС, используемая в САПР VTR. САПР VTR позволяет задавать и исследовать сегментацию межсоединений и топологию маршрутизаторов в трассировочных каналах, внутрикластерную коммутацию, структуру логических элементов, коммутацию внутри логического элемента, размеры LUT, различные связи, например, между блоками памяти и умножителями в ПЛИС, строить планировку кристалла и многое другое. VTR позволяет создавать маршрутизаторы с полной коммутацией и с шинным мультиплексированием. Ядром VTR является VPR (Versatile Place and Route). Упаковщик AAPack реализует кластеризацию взаимосвязанных логических элементов в структурно-зависимые блоки ПЛИС, основан на «жадном» эвристическом алгоритме упаковки с использованием затравочного механизма роста кластера. Размещение кластеров на кристалле ПЛИС осуществляется с помощью алгоритма «имита-

ции отжига», а процесс трассировки использует модификацию алгоритма PathFinder.

Во второй главе рассматривается применение синтезатора Yosys. Yosys, преобразующий Verilog-проекты в формат BLIF, позволяет их отображать в базисе ПЛИС Xilinx серий Spartan-6 и Virtex-7. Для логической оптимизации и отображения в базисе ПЛИС в Yosys интегрирован программный инструмент ABC. Программный инструмент Yosys позволяет также реализовывать Verilog-проекты в двух сериях ПЛИС: Altera Cyclone IV и MAX II посредством vqm-файла. В отличие от Odin II может синтезировать крупные Verilog-проекты с сайта Open Cores (например, микропроцессорные ядра or1200, MSP430, K68 и др.). Yosys позволяет отображать Verilog-проекты в базис библиотечных ячеек заказных БИС (в формате Liberty) и используется в маршруте проектирования заказных БИС Qflow с возможностью последующего изготовления на кремниевых фабриках по единым правилам проектирования масштабируемой КМОП-технологии (MOSIS Scalable CMOS design rules), предоставляемые в открытом доступе. Области использования Yosys — это поведенческий синтез, синтез на уровне регистровых передач и логический синтез.

В главе 3 затрагиваются вопросы программирования ПЛИС серии 5578 с использованием среды разработки конфигурационных данных (АО «КТЦ «ЭЛЕКТРОНИКА») и отладочной платы ОП5578 (АО «Воронежский завод полупроводниковых приборов — сборка»). Процесс проектирования ПЛИС серии 5578 осуществляется в САПР Altera Quartus II версии 9.0 до 13.0 включительно с поддержкой ПЛИС серии Cyclone II и основывается на использовании VQM-файлов (Verilog Quartus Mapping). VQM-файл является ограниченным подмножеством формата Verilog и используется для технологического отображения проекта с помощью сетевых примитивов в уникальный базис ПЛИС.

Целью данного пособия является обеспечение основ проектирования высокоинтегрированных ПЛИС по КМОП-

технологии. Изучение пособия должно способствовать формированию у магистрантов навыков разработки основных функциональных узлов ПЛИС: логических и соединительных блоков; коммутаторов в локальных и глобальных трассировочных ресурсах.

Для достижения цели ставятся задачи: изучение архитектур индустриальных и академических ПЛИС; схемотехнических решений в трассировочных ресурсах; изучение маршрутов проектирования академических ПЛИС с использование программных инструментов с открытым программным кодом: ODIN-II, ABC, VPR, Yosys; получение практических навыков работы с САПР VTR и Qflow; совершенствование навыков использования высокоуровневого языка Verilog.

В результате освоения материала учебного пособия магистр по направлению подготовки 11.04.04 «Электроника и наноэлектроника» получит следующие дополнительные профессиональные компетенции:

1) способность владеть современными методами расчета и проектирования микроэлектронных приборов и устройств твердотельной электроники, способность к восприятию, разработке и критической оценке новых способов их проектирования;

2) способность аргументировано идентифицировать новые области исследований, новые проблемы в сфере проектирования и применения микроэлектронных приборов и устройств;

3) способность самостоятельно разрабатывать новые приборы и устройства микро- и наноэлектроники, работающие на новых физических принципах.

1. АРХИТЕКТУРА АКАДЕМИЧЕСКИХ ПЛИС, ИСПОЛЬЗУЕМАЯ В САПР VTR

Развитие архитектур академических ПЛИС повторяло, а в некоторых случаях даже повлияло на инженерные решения в области развития архитектур индустриальных ПЛИС, в частности, с помощью САПР VTR было установлено ухудшение производительности современных гетерогенных ПЛИС по 3D технологиям при использовании кремниевых интерпозеров для соединения кристаллов ИС с логическими ресурсами.

Современные ПЛИС по своей сути представляют гетерогенную систему в корпусе (3D SIP). Например, ПЛИС Altera серии Stratix 10 по технологии Intel EMIB, когда логическое ядро ПЛИС и высокоскоростные приемопередатчики расположены на разных кристаллах, выполненных по разным технологическим процессам, объединенные между собой по технологии Intel EMIB (Embedded Multi-die Interconnect Bridge), без использования кремниевой коммутационной платы. Для сохранения однокристальности (для сравнения: фирма Xilinx в ранних 3D ПЛИС FPGA серии Virtex-7 и более поздних сериях Virtex UltraScale+ ™ на 16 нм технологическом процессе FinFet+ TSMC проводит разбиение кристалла ПЛИС на секции для последующей реализации на отдельных кристаллах и их соединении с помощью кремниевой коммутационной платы) логических ресурсов ПЛИС была разработана новая технология соединительных ресурсов HyperFlex.

Академический САПР VTR является совместной разработкой трех университетов: Торонто (Канада, Торонто), Нью-Брансвик (Канада, UNB), Калифорнийский университет в Беркли (США, University of California, Berkeley).

В простейшем случае ПЛИС типа программируемые пользователем вентильные матрицы (ППВМ) можно представить в виде матрицы конфигурируемых логических блоков (КЛБ), окруженных со всех сторон трассировочными каналами, сегментируемых маршрутизаторами (рис. 1.1). КЛБ под-

ключаются к трассировочным каналам с помощью соединительных блоков.

Академических ПЛИС целесообразно рассматривать в сравнении с более ранними ПЛИС компании Altera, с сериями FLEX и Stratix II, III, IV и ПЛИС компании Xilinx серий XC3000, XC4000, Virtex-6.

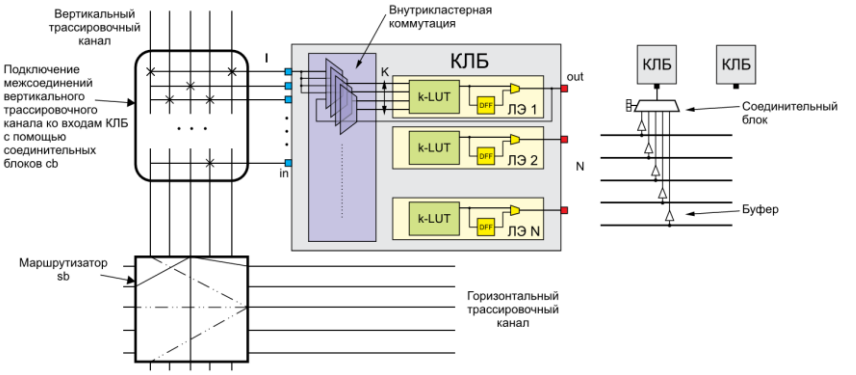


Рис. 1.1. Архитектура академической ПЛИС, принятая в VTR

Коммутаторы («кросс-бары») обеспечивают бесконфликтную параллельную передачу информации с множества Y входов на множество Z выходов, но имеют большую аппаратную «избыточность», и применение их ограничено созданием коммутационных систем небольшой размерности. Полный коммутатор является наиболее гибким, однако, его недостатком является то, что он требует большое количество конфигурационной памяти, что приводит к потреблению избыточной площади кристалла ПЛИС. В более поздних ПЛИС предпочтение стали отдавать использованию разряженных коммутаторов. На рис. 1.2 показано использование разряженных и полных коммутаторов в индустриальной ПЛИС Altera серии FLEX 8K.

Подключение кластера (КЛБ или LAB, как принято в ПЛИС Altera) из 8 логических элементов (ЛЭ) с 4 входовыми

LUT с числом входов 24 к горизонтальному трассировочному каналу из 168 межсоединений (FastTrack — длинное непрерывное межсоединение в трассировочном канале) осуществляется с помощью разряженного коммутатора 1/12 (168 входов × 24 выхода).

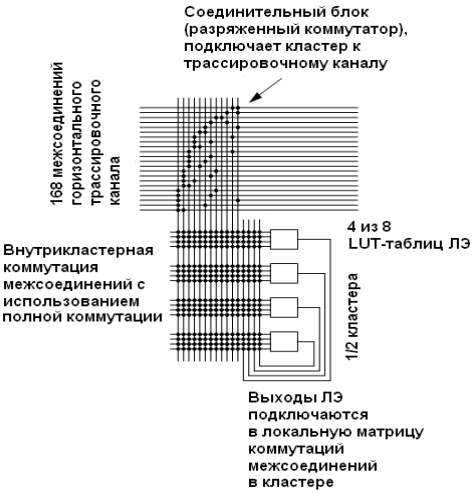


Рис. 1.2. Разряженный и полный коммутатор
в индустриальной ПЛИС Altera серии FLEX 8K

Каждая строка из канала может быть скоммутирована на входы КЛБ. Внутри КЛБ коммутация межсоединений осуществляется с помощью полной коммутации. Полная коммутация позволяет любой вход подключить к любому выходу. Например, в ПЛИС EPF10K10 внутрикластерный коммутатор имеет 22 входа и 30 выходов, а в ПЛИС EPF10K100 — 26 входов и 34 выхода. Соединительный блок в индустриальных ПЛИС иногда называют глобальный коммутатор, а внутрикластерный коммутатор — локальный.

На рис. 1.3 показана стандартная схема коммутации в академических ПЛИС, принятая в САПР VTR. Используется двухуровневая схема коммутации. L1 уровень использует раз-

ряженный коммутатор, а L2 уровень — полный коммутатор. Для сравнения: СнК ПЛИС серии SmartFusion2 фирмы Microsemi имеют трехуровневую внутрикластерную схему коммутации.

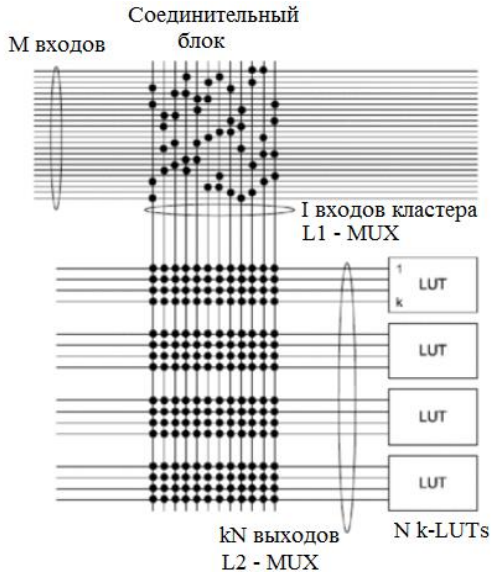


Рис. 1.3. Двухуровневая коммутация,
используемая в академическом САПР VTR
для соответствия индустриальным ПЛИС Altera

Соединительный блок в простейшем случае можно рассматривать как мультиплексор. Объединяя несколько соединительных блоков с заданным коэффициентом объединения по входу Fc_in в группу, можно сконструировать разряженный коммутатор. Например, fc_in = 0,15 означает, что на каждый вход кластера с помощью соединительного блока подключается до 15% доступных межсоединений в трассировочном канале шириной W.

В архитектурном файле САПР VTR используется тег <pinlocations pattern="spread"/>, который позволяет связывать входы/выходы (контакты) КЛБ с трассировочными каналами.

Предусмотрены специальные шаблоны для указания, с какой стороны располагаются входы/выходы КЛБ. Шаблон «spread» соответствует равномерному распределению входов/выходов со всех сторон КЛБ круговым способом. Так, если у КЛБ 33 контакта (входов I = 22, выходов O = 10), то 8 контактов с fc_in = 0,15 назначаются по всем сторонам, кроме одной, для которой назначены 9 контактов (один из них для подключения синхросигнала).

Все входы и выходы кластера логически эквивалентны (equivalent="true"). Логическая эквивалентность входов означает, что подключения с этими входами могут быть заменены без изменения их функциональности.

В современных индустриальных ПЛИС с адаптивными логическими модулями (АЛМ), например, Stratix IV, предпочтение отдается разряженным коммутаторам. Поэтому академические ПЛИС были доработаны и для этого случая. На рис. 1.4 показано подключение кластера из логических элементов (BLE, такое обозначение принято в академических ПЛИС) к горизонтальным и вертикальным трассировочным каналам ($W_X = W_Y$) в ПЛИС с помощью разряженных коммутаторов. Внутрикластерная коммутация осуществляется с использованием разряженных коммутаторов. Однако такая схема внутрикластерной коммутации не является стандартной для САПР VTR и требует доработки архитектурного файла.

Подключение входов/выходов кластера к трассировочным каналам осуществляется с четырех сторон. Трассировочные каналы сегментируются маршрутизаторами типа Disjoint с коэффициентом разветвления F_S = 3. Для сравнения: в ПЛИС Stratix III используется трехсторонняя, а в ПЛИС Virtex — двухсторонняя схема подключения кластера к трассировочным каналам.

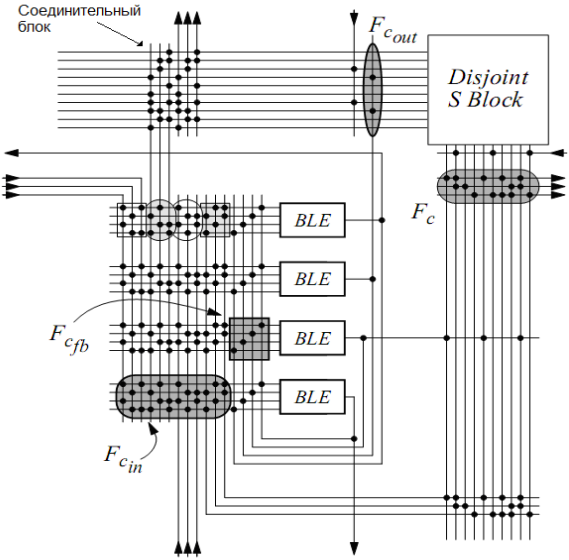


Рис. 1.4. Использование разряженных коммутаторов для подключения КЛБ к трассировочным каналам и для внутрикластерной коммутации в академической ПЛИС

В маршрутизаторах типа Disjoint используются n-МОП шеститранзисторные ключи, которые могут быть реализованы с использованием мультиплексоров и буферов с третьим состоянием. VTR позволяет использовать 3 типа маршрутизаторов в трассировочных каналах: Disjoint, Wilton и Universal.

Маршрутизатор, построенный с использованием двунаправленных межсоединений и двунаправленных ключей, получил название multi-driver, а с использованием однонаправленных межсоединений и мультиплексорных структур — single-driver switch block.

Методология single-driver широко используется в сериях ПЛИС серий Stratix и Virtex. В ПЛИС серий Stratix используется патентованная технология DirectDrive™, которая гарантирует идентичные соединительные ресурсы для любой реализуемой булевой функции, независимо от ее месторасполо-

жения на кристалле, а в ПЛИС Virtex-5 для реализации логических функций и локальных межсоединений используется технология ExpressFabric™.

Методология single-driver также распространяется на соединительные блоки. В случае использования маршрутизаторов Wilton осуществляется непосредственное подключение выходов КЛБ в мультиплексоры ближайших маршрутизаторов, что позволяет отказаться от использования выходных демультиплексоров в соединительных блоках.

На рис. 1.5 показано подключение КЛБ и ЦОС-блоков к маршрутизаторам трассировочных каналов в индустриальных ПЛИС Xilinx. Главное отличие логических плиток (tile) Xilinx от плиток академических ПЛИС заключается в том, что внутрикластерные коммутаторы вынесены в глобальные трассировочные структуры (в маршрутизаторы). В ПЛИС Xilinx логическими плитками могут быть как КЛБ, умножители, так и блоки памяти.

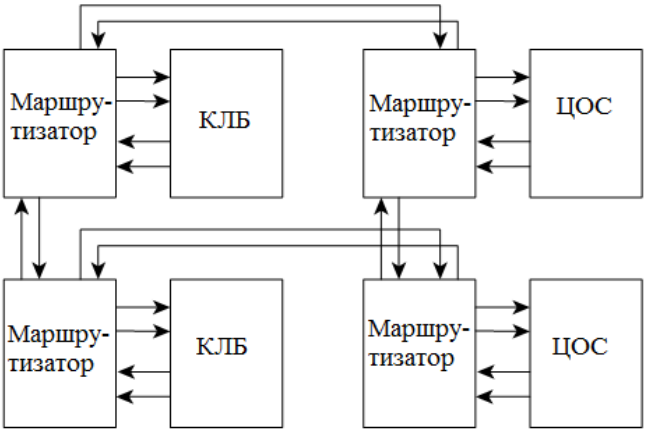


Рис. 1.5. Подключение КЛБ и ЦОС-блоков к маршрутизаторам трассировочных каналов в индустриальных ПЛИС Xilinx

Для ПЛИС с числом входов LUT, равным 4 (K = 4), и числом ЛЭ, равным 8, число входов КЛБ определяется по формуле I = K/2 (N + 1). Тогда полный коммутатор имеет 26 (I = 18 входов и N = 8 выходов обратной связи) входов и 32 выхода (K*N). Полный коммутатор реализуется из двух мультиплексоров по двухступенчатой схеме с разделением управляющих сигналов (рис. 1.6).

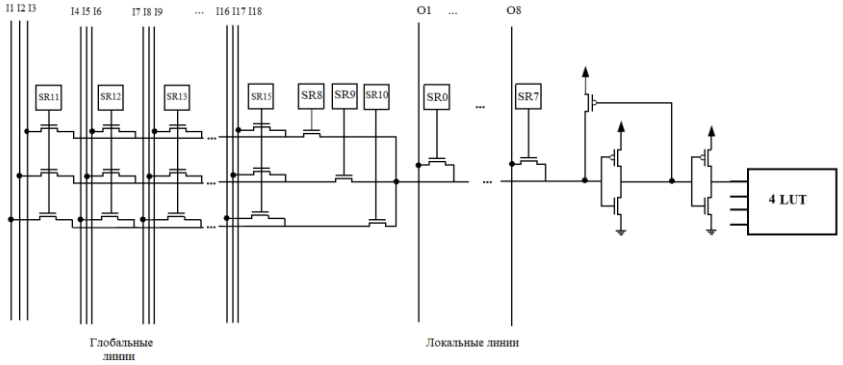


Рис. 1.6. Структурная схема внутрикластерной коммутации на транзисторном уровне

На первой ступени с помощью мультиплексора MUX 18:1 подключаются входы КЛБ к LUT ЛЭ, а с помощью мультиплексора MUX 8:1 второй ступени выходы КЛБ подключаются ко входам LUT (рис. 1.7). Полный коммутатор 26:32 требует 32*16 управляющих сигналов (SR0 – SR15), т. е. 512 конфигурационных ячеек памяти (SR). На рис. 1.8 показана внутрикластерная коммутация с использованием полных коммутаторов в КЛБ академической ПЛИС.

На рис. 1.9 показана внутрикластерная коммутация в КЛБ академической ПЛИС с использованием коммутатора 60×60 (полная коммутация, 60 входных линий и 60 выходных, 40 внешних входов и 20 линий обратной связи), используемая в VTR 8.0. Для сравнения: локальный коммутатор АЛМ ПЛИС Stratix IV (72×88) использует 52 входные линии и

20 линий обратной связи, имеет 88 выходных линий, является разряженным на 50%, т. е. коммутируется не 60 линий на каждый вход ЛЭ, как в академических ПЛИС, а 36 линий в одну.

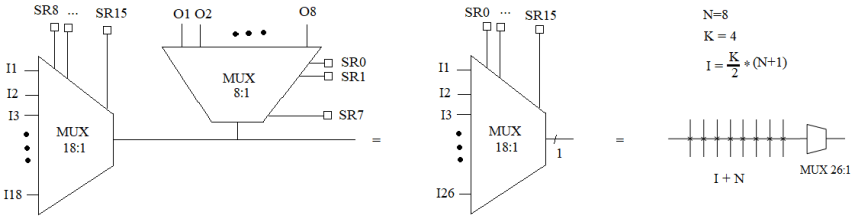


Рис. 1.7. Структурная схема коммутатора 26 в 1
и его условное обозначение

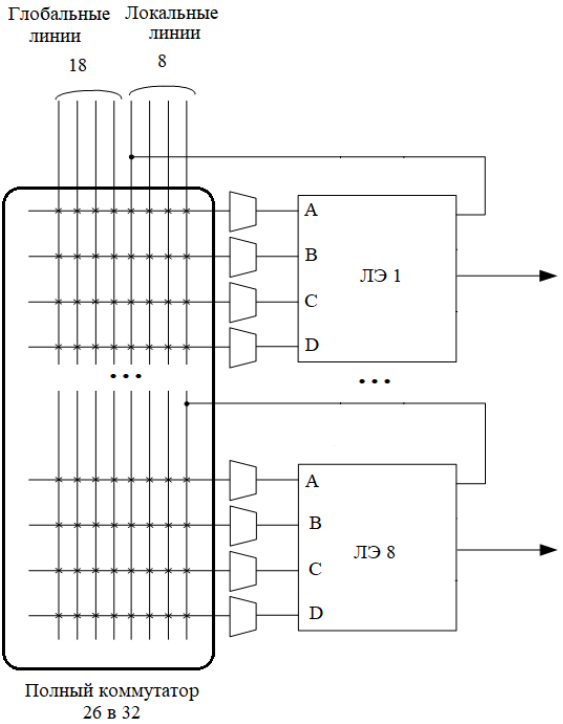


Рис. 1.8. Внутрикластерная коммутация с использованием
полных коммутаторов в КЛБ академической ПЛИС

КЛБ состоит из 10 базовых логических элементов ble (N = 10) (рис. 1.9). ЛЭ имеет 6-входовую перестраиваемую таблицу перекодировок (fracturable 6-входовая LUTs или адаптивный LUT по аналогии с ПЛИС Altera). LUT может быть сконфигурирован как чисто 6-входовой LUT или как два 5-входовых с пятью общими входами. ЛЭ с перестраиваемыми таблицами перекодировок в САПР VTR 8.0 обозначаются как fle, который можно рассматривать как адаптивный логический элемент современных индустриальных ПЛИС.

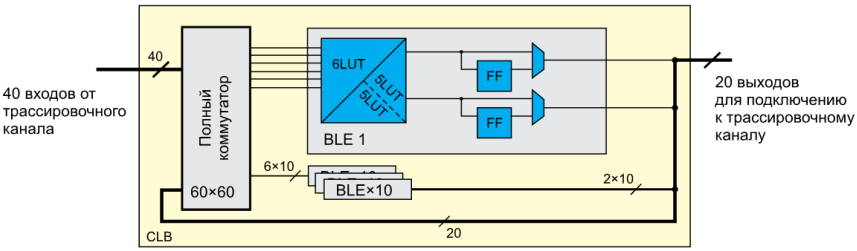


Рис. 1.9. КЛБ в САПР VTR 8.0.
Внутрикластерная коммутация
реализуется с использованием полных коммутаторов

Архитектурный файл, позволяющий создавать различные ПЛИС в САПР VTR 8.0, в большей степени ориентирован на индустриальные ПЛИС Altera. Но в то же время в САПР VTR 8.0 возможно реализовать модели индустриальных ПЛИС Xilinx (рис. 1.10). Для реализации в базисе индустриальных ПЛИС Xilinx базовый КЛБ, используемый в САПР VTR 8.0, необходимо изменить, так как в нем используется полный коммутатор, а не разряженный.

Однако реализация ПЛИС серии Virtex-6 в VTR сопряжена с большими трудностями, так как некоторые особенности ЛЭ Virtex-6 не поддерживаются синтезом VTR (алгоритмом упаковки (кластеризации) оптимизированного blif-файла в технологически-зависимый базис ПЛИС, состоящий из LUT, триггеров, умножителей и блоков памяти), например, мульти-

плексоры F7/F8, LUTRAM (LUT в режиме ОЗУ) и SRL (LUT как сдвиговый регистр) в секции SLICEM. Например, двухтактные триггера FF в ПЛИС Virtex 6 могут тактироваться нарастающим или спадающим фронтом синхросигнала, но все FF в секции должны тактироваться одинаковыми фронтами. Алгоритм упаковки AAPack в САПР VTR 8.0 не содержит встроенных средств обеспечения этого правила.

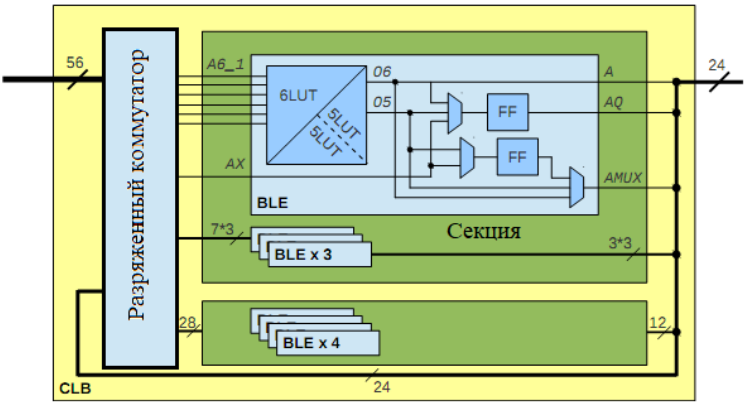


Рис. 1.10. КЛБ ПЛИС Virtex-6
с разряженным коммутатором
для реализации в САПР VTR 8.0

В более поздних экспериментах с архитектурами ПЛИС внутрикластерный коммутатор был вынесен из КЛБ (рис. 1.11). Также были использованы аппаратные сумматоры на выходах LUT (XADDER) для организации вертикальных цепей переноса для каскадирования сумматоров, а также доработана модель трассировочных ресурсов, с использованием однонаправленных и двунаправленных межсоединений (bi-dir) с различной длиной сегментации, проходящие непрерывно через L = 1, 2, 4 и 16 кластеров в горизонтальном, вертикальном, диагональном и изогнутом направлениях.

В заключение отметим, что гипотетические академические архитектуры ПЛИС унаследовали все ключевые решения

архитектур индустриальных ПЛИС, включая адаптивные логические элементы, внутрикластерные коммутаторы, соединительные блоки, маршрутизаторы, гетерогенные блоки, такие как умножители и блоки памяти.

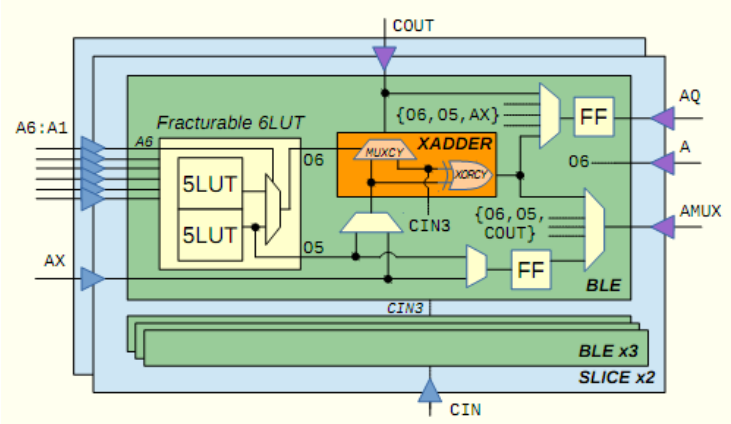


Рис. 1.11. Доработанный КЛБ,
используемый в архитектурном
файле VTR для соответствия ПЛИС серии Virtex-6

САПР VTR 8.0 позволяет задавать и исследовать сегментацию межсоединений и топологию маршрутизаторов в трассировочных каналах, внутрикластерную коммутацию, структуру логических элементов, коммутацию внутри логического элемента, размеры LUT, различные связи, например, между блоками памяти и умножителями в ПЛИС, строить планировку кристалла и многое другое. VTR позволяет создавать кросс-бары с полной коммутацией и с шинным мультиплексированием. Ядром VTR является VPR (Versatile Place and Route). Упаковщик AAPack реализует кластеризацию взаимосвязанных логических элементов в структурно-зависимые блоки ПЛИС, основан на «жадном» эвристическом алгоритме упаковки с использованием затравочного механизма роста кластера. Размещение кластеров на кристалле ПЛИС осуществляется с помощью алгоритма «имитации отжига», а

процесс трассировки использует модификацию алгоритма PathFinder.

Стандартный VTR не может генерировать битовый поток. VTR не предоставляет никаких средств для создания RTL-моделей архитектур академических ПЛИС, что не позволяет по структурному Verilog-коду синтезировать физическую топологию кристалла для последующего его изготовления.

1.1. Программные средства с открытым исходным кодом для проектирования цифровых устройств в базис ПЛИС

Предлагается рассмотреть программный инструмент для Verilog-синтеза Odin-II с открытым программным кодом и его использование для проектирования цифровых устройств в базисе ПЛИС.

Академический САПР VTR предполагает использование следующих программных инструментов: ODIN-II (UNB), ABC (UC Berkeley) и VPR (University of Toronto). ODIN-II и ABC являются открытыми программными продуктами. ODIN_II конвертирует схемное описание некоторого сложнофункционального устройства на языке Verilog в специальный файл в blif (Berkeley Logic Interchange Format)-формате, в котором выделяет логические вентили для описания логики устройства и «черные ящики» для гетерогенных блоков, таких как умножители, блоки памяти и др. Фактически blif-формат представляет собой технологически независимый нетлист. Программный инструмент ODIN-II осуществляет переход с RTL-уровня на вентильный.

Синтаксический анализатор ODIN-II разработан на основе программных средств bison (лексический анализатор) и flex (синтаксический анализатор). По Verilog-коду синтаксический анализатор ODIN-II строит абстрактное синтаксическое дерево и перерабатывает его в плоский список (нетлист) (рис. 1.12).

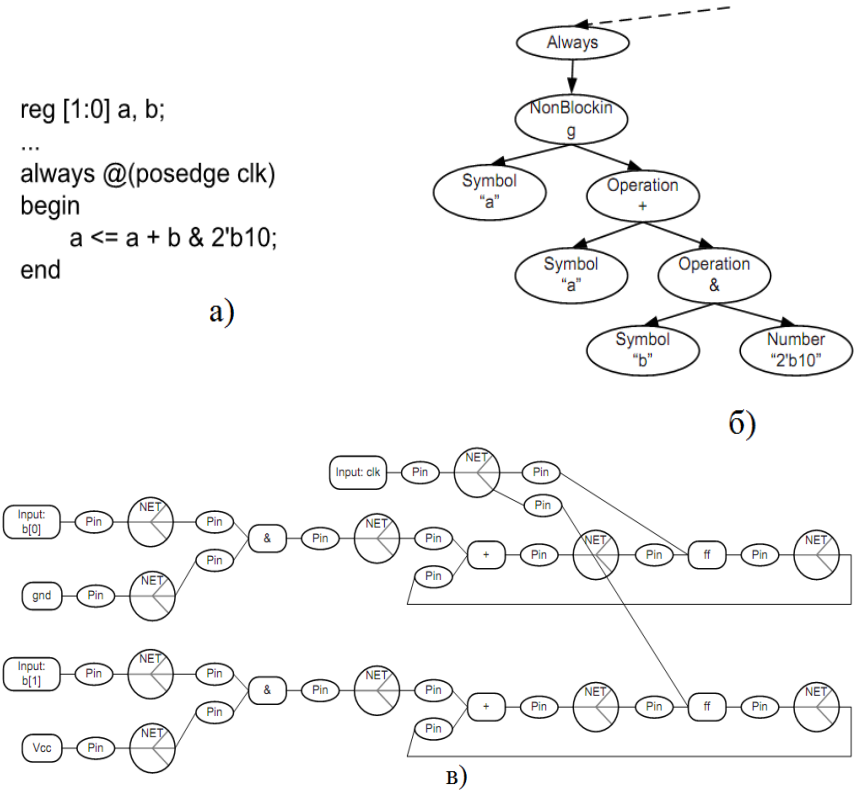


Рис. 1.12. Представление структурных данных в ODIN II:
а — фрагмент Verilog-кода; *б* — AST-дерево;
в — плоский нетлист

Далее с использованием программного инструмента ABC проводится логическая оптимизация схемы и ее размещение в логические блоки академической ПЛИС. Такая операция называется технологический мэппинг (привязка к библиотеке) или технологическое отображение, перенос описания логической схемы на множество LUT элементов уникального базиса ПЛИС (рис. 1.13.). Вентиль с меткой × покрывается обоими LUT. Также показано возможное начальное отображение схемы в базис 4-LUT и отображенная схема в базис 4-LUT. Выходным файлом ABC также является файл в .blif

формате, в котором выделяются LUT, D-триггеры логических блоков и гетерогенные блоки (технологически зависимый нетлист).

Более 200 ведущих мировых дизайн-центров по проектированию БИС и более 1 000 учебных образовательных центров широко используют программный инструмент VPR (Versatile Place and Route) как для проектирования, так и для исследования новых архитектур ПЛИС, например, мемристорные ПЛИС.

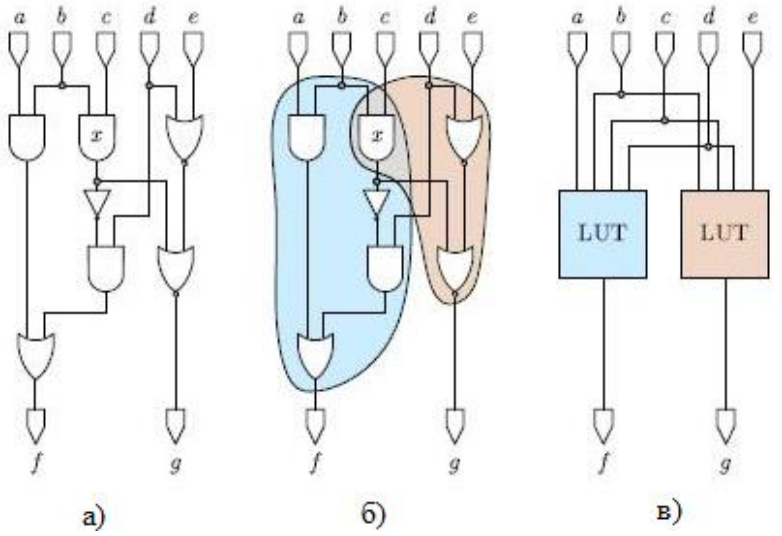


Рис. 1.13. Технологическое отображение в базис 4-входовых LUT:
а — логическая схема на вентильном уровне;
б — возможное начальное отображение схемы в базис 4-LUT;
в — схема, отображенная в базис 4-LUT

VPR (является ядром VTR) размещает кластеры логических блоков и гетерогенные блоки (умножители, блоки ОЗУ и др.) по кристаллу ПЛИС и организует глобальные и локальные трассировочные ресурсы для меж- и внутрикластерой связи логических блоков наиболее оптимальным образом, с учетом требований, например, к минимальной ширине трассиро-

вочного канала, быстродействию, экономии площади кристалла (рис. 1.14).

Для размещения функциональных блоков на кристалле ПЛИС применяется алгоритм «имитации отжига». Процесс «имитации отжига» может быть представлен на основе четырех ключевых компонентов: представления состояния текущего решения, набора перемещений из одного состояния в другое, целевой функции стоимости для оценки каждого состояния и «схемы охлаждения», определяющей, как можно перейти от начального поиска к локальной оптимизации. Разводка электрических связей между кластерами осуществляется на основе модификаций алгоритма PathFinder с учетом задержек распространения сигналов в трассировочных ресурсах ПЛИС.

Флагманский архитектурный файл САПР VTR7 позволяет исследовать академические ПЛИС, близкие по техническим характеристикам (задержки в КЛБ, в маршрутизаторах трассировочных каналах, в умножителях и блоках памяти) к ПЛИС Altera серии Stratix IV GX (EP4SGX230DF29C2X, 40 нм).

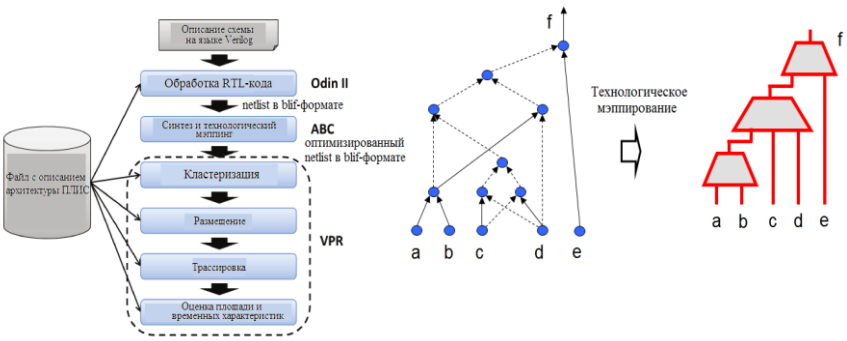


Рис. 1.14. Маршрут реализации Verilog-проектов
в базис академических ПЛИС
с применением САПР VTR 7

На рис. 1.15 показан маршрут реализации больших HDL-проектов проектирования академических ПЛИС (проект Титан) с использованием VPR и VQM-файлов (Verilog Quartus Mapping) САПР Quartus II. VQM-файл можно извлечь как на этапе анализа и синтеза, так и после полной компиляции с учетом размещения и трассировки. VQM-файл является ограниченным подмножеством формата Verilog и представляет собой технологическое мэппирование (отображение) проекта с помощью сетевых примитивов в уникальный базис ПЛИС.

В качестве примера приведем наиболее важные сетевые примитивы ПЛИС серии Cyclone II: cycloneii_lcell_comb — Lut в различных режимах; cycloneii_lcell_ff — регистр ЛБ; cycloneii_io — элементы ввода/вывода; Cyclone II_mac_mult — умножитель; Cyclone II_mac_out — аккумулятор (в ПЛИС серии Cyclone II — регистр на выходе умножителя); cycloneii_ram_block — блок ОЗУ. Для конвертации VQM в BLIF используется специально разработанная утилита VQM2BLIF, которая «один в один» преобразует сетевые примитивы технологического базиса ПЛИС серии Stratix IV в структуру blif-файла: .subckt; .names; .latch.

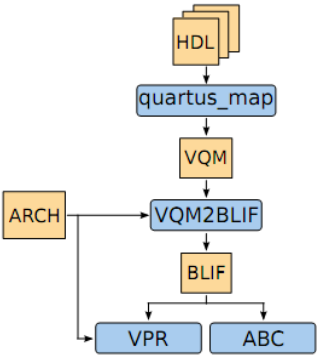


Рис. 1.15. Маршрут реализации HDL-проектов
в базис академических ПЛИС (проект Титан)
с использованием VPR

Существуют и индустриальные САПР, например, синтезатор Synplify Pro от Synopsys позволяет по VHDL/Verilog-проектам, созданным в САПР Quartus II на этапе анализа и синтеза, создавать «свои» VQM-файлы в технологическом базисе ПЛИС Altera. VQM-файлы, созданные с помощью Synplify Pro, отличаются по формату от VQM-файлов, созданных Quartus II (синтезатор QIS, Quartus II Integrated Synthesis от Altera), но САПР Quartus II допускает их повторное использование в проектах.

Идея метода показана на рис. 1.16. Использование VQM-файла, созданного с помощью Synplify Pro, позволяет не только исключить этап синтеза из маршрута проектирования САПР Quartus II, но и сократить число используемых логических ресурсов ПЛИС за счет более развитых средств синтеза логики.

На рис. 1.17 показано представление в Synplify Pro мегафункций LPM_MULT и ALTACCUMULATE проекта КИХ-фильтра с использованием сетевых примитивов ПЛИС Cyclone II в виде черных ящиков.

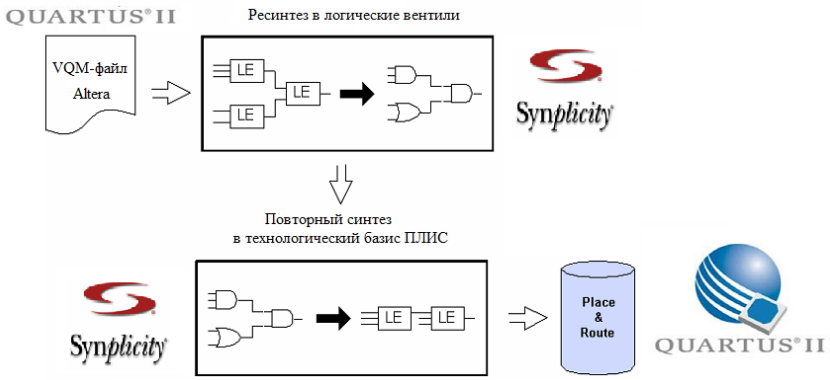


Рис. 1.16. С помощью VQM-файла проекта, созданного САПР Quartus II, осуществляется ресинтез технологического мэппинга в логические вентили и последующий синтез логики в технологический базис ПЛИС синтезатором Synplify Pro с получением VQM-файла

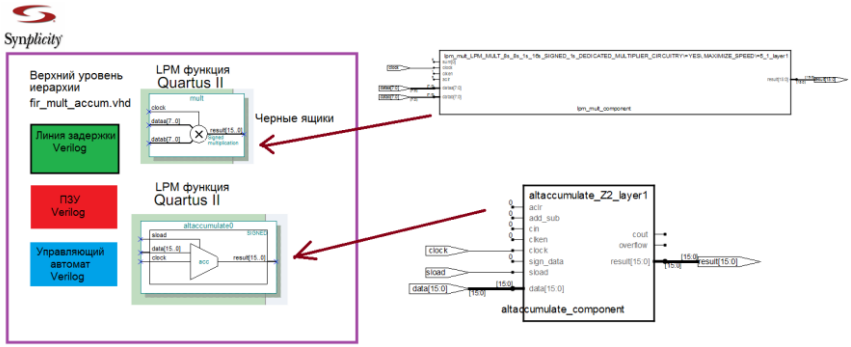


Рис. 1.17. Представление мегафункций LPM_MULT и ALTACCUMULATE САПР Quartus II в виде черных ящиков (сетевых примитивов ПЛИС Cyclone II)

В академических ПЛИС для обеспечения программируемой коммутации существует две технологии соединений: multi-driver и single-driver, которые распространяются как на соединительные блоки, так и на коммутаторы-маршрутизаторы. Маршрутизатор с использованием двунаправленных межсоединений и двунаправленных ключей, реализованных на буферах с третьим состоянием, получил название multi-driver, при этом также возможно использование n-МОП ключей, а с использованием однонаправленных межсоединений и мультиплексорных структур — single-driver switch block.

В настоящее время считается, что использование однонаправленных межсоединений в совокупности с мультиплексорными структурами в маршрутизаторах наиболее перспективно, так как позволяет получать существенный выигрыш по сравнению с технологией multi-driver по быстродействию и по площади кристалла.

Индустриальные ПЛИС типа ППВМ серии Stratix фирмы Altera и Virtex фирмы Xilinx имеют сегментируемую трассировочную структуру с использованием однонаправленных межсоединений (unidirectional). Данная структура хорошо ис-

следована с использованием программного инструмента VPR. В архитектуре ПЛИС семейства Stratix фирмы Altera соединения между кластерами, TriMatrix памятью, DSP-блоками и элементами ввода/вывода (ЭВВ) осуществляются с помощью сети многоканальных межсоединений MultiTrack с использованием технологии DirectDrive™. Детерминированная технология маршрутизации DirectDrive гарантирует идентичные соединительные ресурсы для любой реализуемой булевой функции, независимо от ее месторасположения на кристалле ПЛИС, что обеспечивается использованием однонаправленных межсоединений и мультиплексорных структур типа single-driver. В ПЛИС Altera Stratix 10 появилась новая трассировочная структура HyperFlex («регистры повсюду») во внутри- и межкластерной коммутации. Применение регистров в локальных и глобальных трассировочных ресурсах позволяет выравнивать задержки распространения сигналов за счет их перераспределения и конвейеризации.

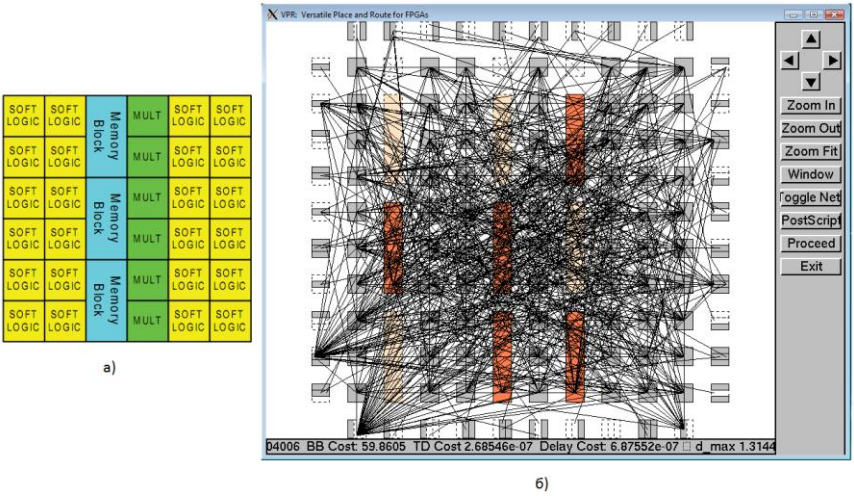


Рис. 1.18. Гетерогенная архитектура академической ПЛИС (*а*) с одноуровневой структурой межсоединений (4 входовая LUT, размер кластера 10 логических блоков) со связями между функциональными блоками в САПР VPR 5.0 (*б*)

На рис. 1.18 показана гетерогенная архитектура ПЛИС типа ППВМ с одноуровневой структурой межсоединений размером 10×10 кластеров и встроенными блоками умножителей 36×36 (9 шт.) в САПР VPR 5.0. Каждый кластер состоит из 10 ЛБ, каждый ЛБ состоит из 4 входовой LUT-таблицы и триггера. По периферии кристалла располагаются блоки ввода/вывода. В базис ПЛИС размещена тестовая схема БИХ-фильтра. Задействованные блоки умножителей для реализации БИХ-фильтра отображены оранжевым цветом.

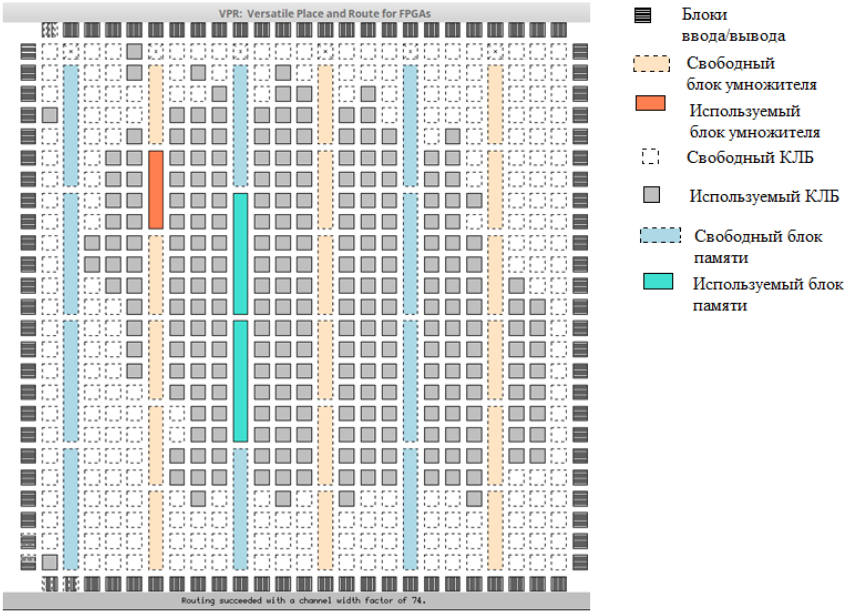


Рис. 1.19. После этапа размещения и трассировки микропроцессорного ядра or1200 в САПР VTR 7.0

На рис. 1.19. демонстрируется реализация микропроцессорного ядра or1200 в базис академической ПЛИС с использованием САПР VTR 7.0. Микропроцессор or1200 основан на 32-битной архитектуре набора команд (ISA) ORBIS32. Центральный процессор содержит MAC-блок, чтобы лучше под-

держивать ЦОС-приложения. Для реализации проекта была сгенерирована архитектура ПЛИС с 18 умножителями с размерностью операндов 36×36, из них задействован только 1, так как процессорное ядро or1200 содержит 1 MAC-блок, 12 блоков двухпортовой памяти, из которых задействовано только 2, а также 475 конфигурируемых ЛБ (КЛБ), из которых задействовано 257. Для реализации комбинационной логики требуется 1 267 6-входовых LUT и 691 триггер. В трассировочных ресурсах используется маршрутизатор типа Wilton с однонаправленными межсоединениями. Оптимальная ширина трассировочного канала — 74 межсоединения. Максимальная тактовая частота работы процессорного ядра в базисе академической ПЛИС составляет f_max: 74,9708 МГц.

VPR также использовался для разработки ПЛИС NATURE (hybrid CMOS/NanoTUbe REconfigurable architecture, NATURE) типа ППВМ с одноуровневой структурой межсоединений. Для хранения конфигурационных данных используется энергонезависимая оперативная память NRAM (Nanotube-based Random Access Memory) на основе углеродных нанотрубок. Такая память сочетает в себе лучшие качества запоминающих устройств — дешевизну (DRAM) и энергонезависимость (флэш-память), а также будет обладать высокой стойкостью к воздействию температуры и магнитных полей.

Есть успешные примеры применения VPR для создания САПР гибридных КМОП/наномолекулярных устройств (CMOL FPGA, CMOS + MOLecular electronics) и САПР mrFPGA с использованием мемристоров в программируемых соединительных блоках и маршрутизаторах и КМОП-совместимого технологического процесса. Мемристор сформирован пересечением электродов из платиновой нанопроволоки, разделенных пленкой диоксида титана.

1.2. САПР VTR 7 для проектирования академических ПЛИС

В данном разделе предлагается рассмотреть программные инструменты САПР VTR (Verilog to Routing) с открытым кодом, разработанным в университете Торонто (Канада, Торонто, http://www.eecg.utoronto.ca/vpr) для проектирования академических ПЛИС типа FPGA с одноуровневой структурой трассировочных ресурсов, когда кластеры из конфигурируемых логических блоков (КЛБ) окружены с четырех сторон межсоединениями горизонтальных и вертикальных трассировочных каналов, равномерно распределенных по всей площади кристалла.

Академический САПР ПЛИС VTR 7 (архив vtr-verilog-to-routing-master.zip) может быть скачан по адресу (https://github.com/verilog-to-routing/vtr-verilog-to-routing). В настоящее время доступна официальная версия 7.07 (рис. 1.20). САПР полностью общественный, может быть загружен и изменен любым человеком.

Предлагается использовать официальную версию 7. По ссылке https://github.com/verilog-to-routing/vtr-verilog-to-routing/wiki будет осуществлен переход на сайт университета Торонто http://www.eecg.utoronto.ca/vtr/terms.html.

Далее необходимо установить ОС Linux Mint 17.3 «Rosa» — Cinnamon (64-bit) KDE Edition по ссылке http://www.linuxmint.com/start/rosa/ (рис. 1.21). Возможно использовать и другие разновидности ОС linux, например, Ubuntu (64 bit). Для запуска в ОС Windows ОС Linux Mint 17.3 «Rosa» необходимо скачать образ LinuxMint-17.3-mate-64bit.iso и настроить виртуальную машину. В качестве виртуальной машины будем использовать Oracle VM VirtualBox. Ссылка для скачивания виртуальной машины: download. virtualbox.org/virtualbox/5.0.14/VirtualBox-5.0.14-105127-Win.exe.

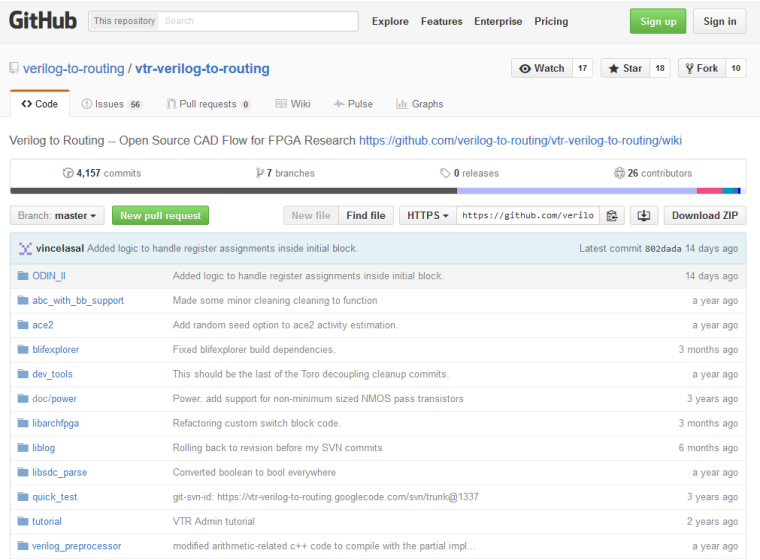


Рис. 1.20. Страница на сайте github,
посвященная САПР ПЛИС VTR7

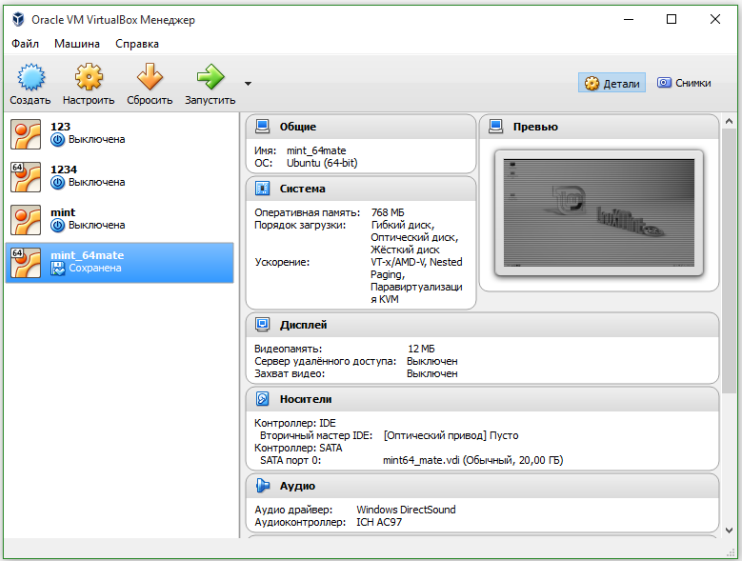


Рис. 1.21. Виртуальная машина Oracle VM VirtualBox версия 5.04
для запуска ОС Linux Mint 17.3 «Rosa» — Cinnamon (64-bit)

Для запуска VTR 7 в среде Windows можно также использовать свободно распространяемую UNIX-подобную среду Cygwin. Cygwin обеспечивает тесную интеграцию Windows приложений, данных и ресурсов с приложениями, данными и ресурсами UNIX-подобной среды. Процесс инсталляции необходимо начать на сайте http://www.cygwin.com и загрузить программу setup-x86_64.exe (64-bit installation).

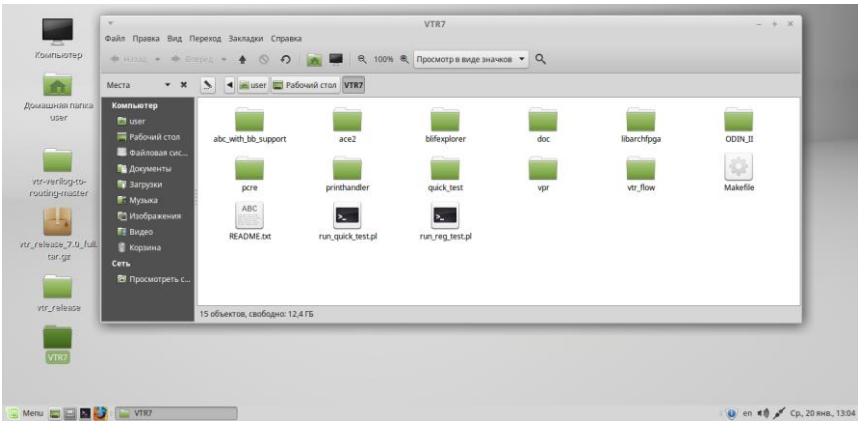


Рис. 1.22. Структура каталогов САПР VTR7

Заходим в каталог VTR 7 (рис. 1.22), в который установлен САПР VTR 7, и командой «Make» получаем необходимые exe-файлы. В папках abc_with_bb_support, ODIN_II, VPR должны появиться скомпилированные exe-файлы. Если по каким-то причинам не установлены следующие пакеты: g++, ctags, Libx11-dev, Libxft-dev, bison, flex, то необходимо с помощью команд:

```
sudo apt-get install g++
sudo apt-get install ctags
sudo apt-get Libx11-dev
sudo apt-get Libxft-dev
sudo apt-get bison
sudo apt-get flex
sudo apt-get install g++
```

либо обновить, либо доустановить требуемые пакеты. Для того чтобы подключить графику к программе VPR, необходимо отредактировать make-файл. В строке ENABLE_GRAPHICS = false необходимо переменную false заменить на true.

Типовой маршрут проектирования гетерогенных академических ПЛИС, предполагает использование следующих программных инструментов: ODIN-II, ABC и VPR. ODIN-II (под лицензией MIT) и ABC (University of California, Berkeley) являются открытыми программными продуктами. ODIN_II конвертирует схемное описание некоторого сложнофункционального устройства («бенч марк», тестовая схема на языке Verilog HDL) в специальный файл в blif (Berkeley Logic Interchange Format)-формате, в котором выделяет логические вентили для описания логики устройства и «черные ящики» для гетерогенных блоков, таких как умножители, блоки памяти и др. Синтаксический анализатор ODIN-II разработан на основе программных средств bison (лексический анализатор) и flex (синтаксический анализатор). По Verilog-коду синтаксический анализатор ODIN-II строит абстрактное синтаксическое дерево и перерабатывает его в плоский список.

С использованием программного инструмента ABC проводится логическая оптимизация схемы с применением специального стиля описания, независимого от технологии проектирования БИС, и ее размещение в логические блоки академической ПЛИС (привязка к библиотеке или технологический мэппинг). Выходным также является файл в blif-формате, в котором выделяются LUT-таблицы, D-триггеры логических блоков и гетерогенные блоки.

Для работы программ Odin_II и VPR требуется конфигурационный файл для описания архитектуры ПЛИС. Будем использовать архитектурный файл (пример 1) для описания ПЛИС (файл k6_frac_N10_mem32K_40nm.xml находится в папке vtr_flow/arch/timing). Предполагается, что ПЛИС будет изготовлена по 40 нм технологическому КМОП-процессу с типовыми задержками в межсоединениях, вентилях и ключах.

Флагманский архитектурный файл позволяет получать академические ПЛИС, близкие по техническим характеристикам (задержки в КЛБ, в маршрутизаторах трассировочных каналах, в умножителях и блоках памяти) к ПЛИС Altera серии Stratix IV GX (EP4SGX230DF29C2X).

Конфигурационный логический блок (кластер) состоит из 10 адаптивных логических элементов (N = 10) (рис. 1.23). Логический элемент (ЛЭ) имеет 6-входовую таблицу перекодировок (fracturable 6-входовая LUTs или перестраиваемый, адаптивный LUT). Адаптивный LUT может быть сконфигурирован как чисто 6-входовой LUT или как два 5-входовых с пятью общими входами (рис. 1.24). LUT КЛБ академической ПЛИС по аналогии с ПЛИС Altera — это адаптивный логический модуль (АЛМ), но только режимов конфигурации два, а не семь, как у Stratix IV. Какие-либо цепи переноса на выходах LUT в КЛБ не используются.

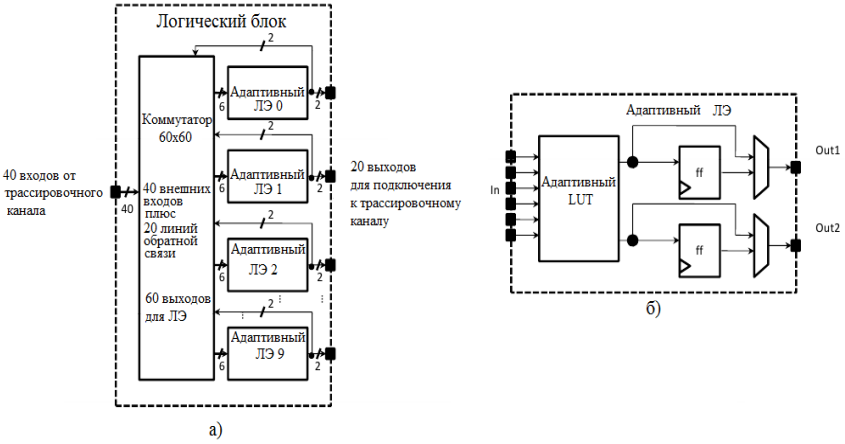


Рис. 1.23. КЛБ академической ПЛИС,
состоящий из десяти адаптивных логических элементов,
с локальными трассировочными ресурсами (*а*),
и структура адаптивного логического элемента (*б*)

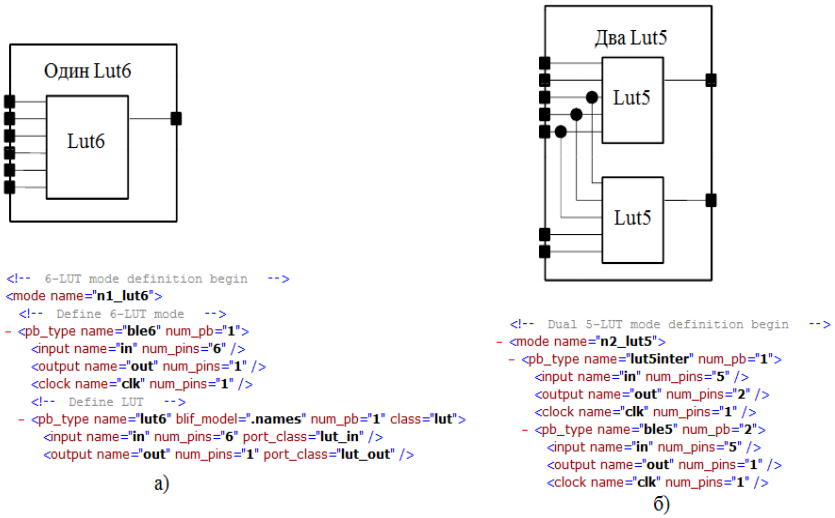


Рис. 1.24. Режимы кофигурации LUT
с фрагментами кода архитектурного файла:
а — 6-входовой LUT; *б* — два 5-входовых LUT
с тремя общими входами

Для сравнения: на рис. 1.25 показан АЛМ в самых современных сериях ПЛИС Altera Stratix IV и Stratix 10. Для Stratix 10 АЛМ состоит из 8-входовой адаптивной LUT, имеющей семь режимов работы, пары сумматоров и четырех триггеров.

На рис. 1.23 показана внутрикластерная коммутация с использованием коммутатора 60×60 (полная коммутация, 60 входных линий и 60 выходных, 40 внешних входов и 20 линий обратной связи). Полная коммутация позволяет любой вход любого ЛЭ в КЛБ подсоединить к любому произвольному входу ЛЭ КЛБ или выходу. Все входы и все выходы в таком КЛБ являются логически эквивалентными и функционально идентичными. При полной коммутации достигаются лучшие показатели: площадь — быстродействие, при этом достигается максимальная гибкость при коммутации связей внутри КЛБ.

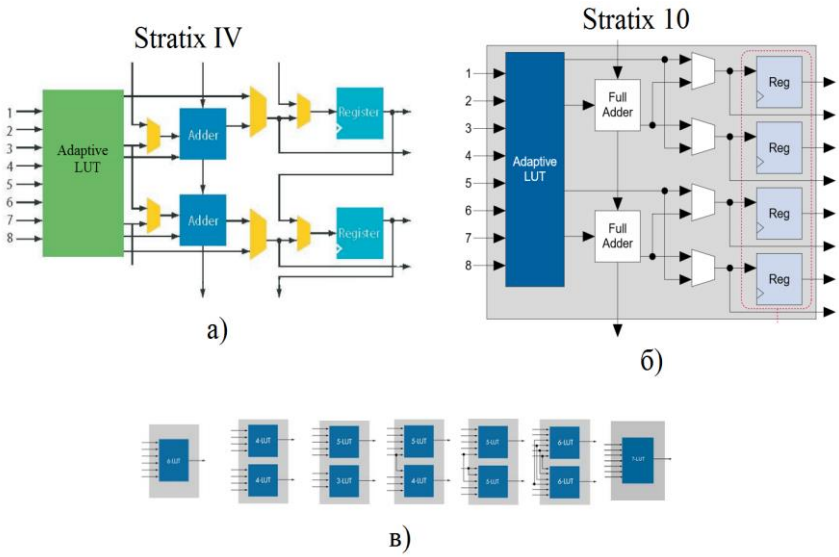


Рис. 1.25. Инновации в структуре АЛМ ПЛИС Altera Stratix IV *(а)* Stratix 10 (*б*) и режимы конфигурации LUT Stratix IV (*в*)

Коммутатор размерностью 60×60 можно представить как 60 мультиплексоров 60 в 1 с использованием ключей на n-МОП-транзисторах. На каждый вход ЛЭ может быть скоммутировано до 60 входных линий. Для сравнения: коммутатор АЛМ ПЛИС Sratix IV (72×88) использует 52 входные линии и 20 линий обратной связи, имеет 88 выходных линий, является разряженным на 50%, т. е. коммутируется не 60 линий на каждый вход ЛЭ, как в академических ПЛИС, а 36 линий в одну.

Считается, что коммутатор 72×88 по занимаемой площади на кристалле и задержкам распространения сигналов больше, чем коммутатор 60×60, но с учетом разреженности можно добиться примерно равных результатов. Оба коммутатора имеют примерно равное число точек коммутации — 3 168 и 3 600.

В ПЛИС Sratix IV задержки по входам LUT имеют разные значения, а в САПР VTR принято, что задержки по входам LUT одинаковые.

Блоки памяти объемом по 32 Кбит. Ширина шины данных изменяется от 1 до 64 разрядов. По протяженности модуль памяти занимает 6 КЛБ. Умножители с размером операндов 36×36, два 18×18, каждый из которых может быть сконфигурирован как два 9×9. По протяженности умножитель занимает 4 КЛБ.

Коэффициент объединения по входу/выходу для соединительных блоков fc_in = 0,15, Fc_out = 0,1. Длина сегментации межсоединений в трассировочных каналах L = 4, т. е. межсоединение проходит непрерывно 4 КЛБ.

Максимальная ширина трассировочного канала, заложенная в академическую ПЛИС, составляет 300 межсоединений, что хорошо согласуется с современными индустриальными ПЛИС и обеспечивает высокую маршрутизируемость.

Конфигурационный файл состоит из двух частей, непосредственно для программы Odin_II (помечен <!-- ODIN II specific config--> и <!-- ODIN II specific config ends -->) и для программы VPR, в которой содержится информация о требуемых размерах кристалла ПЛИС; о сопротивлении и минимальных геометрических размерах n- и p-МОП-ключей; об емкостях входных буферов мультиплексорных структур соединительных блоков, задержках сигналов через эти буферы и мультиплексоры; о типах маршрутизаторах; о соединительных блоках; о сегментации межсоединений в каналах, о типе межсоединений (двунаправленные или однонаправленные) и их сопротивлениях и емкостях; о ширине трассировочного канала ядра и периферийного канала между ядром и блоками ввода/вывода и др.

```
<!-- ODIN II specific config-->
<models><model name="multiply">
<input_ports>
<port name="a"/>
<port name="b"/>
</input_ports>
<output_ports>
<port name="out"/>
```

```
</output_ports>
</model>
<model name="single_port_ram">
<input_ports>
<port name="we"/>   <!-- control -->
<port name="addr"/> <!-- address lines -->
<port name="data"/>
 <!-- data lines can be broken down into smaller bit widths minimum size 1 -->
<port name="clk" is_clock="1"/> <!-- memories are often clocked -->
</input_ports>
<output_ports>
<port name="out"/>
 <!-- output can be broken down into smaller bit widths minimum size 1 -->
</output_ports>
</model>
<model name="dual_port_ram">
<input_ports>
<port name="we1"/>   <!-- write enable -->
<port name="we2"/>   <!-- write enable -->
<port name="addr1"/> <!-- address lines -->
<port name="addr2"/> <!-- address lines -->
<port name="data1"/>
<!-- data lines can be broken down into smaller bit widths minimum size 1 -->
<port name="data2"/>
<!-- data lines can be broken down into smaller bit widths minimum size 1 -->
<port name="clk" is_clock="1"/>
 <!-- memories are often clocked -->
</input_ports>
<output_ports>
<port name="out1"/>
<!-- output can be broken down into smaller bit widths minimum size 1 -->
<port name="out2"/>
<!-- output can be broken down into smaller bit widths minimum size 1 -->
</output_ports>
</model>
</models>
 <!-- ODIN II specific config ends -->
<!-- Physical descriptions begin -->
<layout auto="1.0"/>
<device>
<sizing R_minW_nmos="8926" R_minW_pmos="16067"
ipin_mux_trans_size="1.222260"/>
<timing C_ipin_cblock="1.47e-15" T_ipin_cblock="7.247000e-11"/>
<area grid_logic_tile_area="0"/>
<chan_width_distr>
<io width="1.000000"/>
<x distr="uniform" peak="1.000000"/>
```

```
<y distr="uniform" peak="1.000000"/>
</chan_width_distr>
<switch_block type="wilton" fs="3"/>
</device>
<switchlist>
<segment freq="1.000000" length="4" type="unidir" Rmetal="101" Cmetal="22.5e-15">
<mux name="0"/>
<sb type="pattern">1 1 1 1 1</sb>
<cb type="pattern">1 1 1 1</cb>
</segment>
</segmentlist>
<complexblocklist>
```

Пример 1. Фрагмент конфигурационного файла k6_frac_N10_mem32K_40nm.xml

В качестве примера рассмотрим проект синтезированного микропроцессорного ядра or1200 на языке verilog. Получить дополнительную информацию можно по ссылке: http://ru.knowledgr.com/06057739/OpenRISC1200. А сам проект можно скачать по ссылке: https://github.com/openrisc/or1200. По указанной ссылке представлен иерархический проект (or1200-master.zip). Файл верхнего уровня иерархии or1200_cpu.v. Официальное внедрение проекта сохраняется за разработчиками в OpenCores.org.

Микропроцессор OR1200 основан на 32-битной архитектуре набора команд (ISA) ORBIS32. Центральный процессор также содержит MAC-блок, чтобы лучше поддерживать ЦОС-приложения.

Для запуска САПР VTR воспользуемся скрипт-файлом run_vtr_flow.pl, который находится в папке vtr_flow/scripts. Предположим, что нам необходимо разместить в базис ПЛИС с архитектурным файлом k6_n10.xml синтезируемое процессорное ядро OR1200, представленное в виде единственного verilog-файла.

Требуемый файл or1200.v («так называемый плоский файл», не иерархический, проект представлен единственным файлом) находится в папке vtr_flow/benchmarks/Verilog (VPR Benchmark). Является адаптацией оригинального проекта

or1200_cpu.v для задач размещения и трассировки в САПР VTR. Для реализации оригинального проекта or1200_cpu.v без какой-либо переделки в САПР Quartus II потребуется значительно большее число вводов/выводов ПЛИС. Предварительно два файла or1200.v и k6_n10.xml необходимо поместить в папку scripts. Для запуска скрипта в командной строке набираем следующее:

./ run_vtr_flow.pl or1200.v k6_n10.xml.

В папке scripts будет автоматически создана папка temp со следующими файлами odin.out (пример 2), abc.out (пример 3), or1200.pre-vpr.blif (пример 4), vpr.out (пример 5), vpr_stdout.log (пример 6) и др. Ниже приведены фрагменты этих файлов. Для работы программы логического синтеза ODIN II требуется конфигурационный файл odin_config.xml, который создается в автоматическом режиме из архитектурного файла k6_n10.xml. Программа ODIN II определила, что для реализации проекта дополнительно требуется два блока двухпортовой памяти и один умножитель с размерностью операндов 36×36.

Выходным файлом программы ODIN II является файл or1200.odin.blif. Непосредственно он недоступен. Далее с помощью программы ABC будет проведена оптимизация. Выходным файлом является файл с именем or1200.abc.blif, который также недоступен. В процессе работы скрипта будет создан файл or1200.pre-vpr.blif (пример 4), необходимый программе VPR. Выходные файлы работы программы VPR — это vpr.out (пример 5) и vpr_stdout.log (пример 6).

Анализируя полученные файлы vpr.out и vpr_stdout.log, можно сделать выводы, что для реализации проекта была сгенерирована архитектура ПЛИС с 18 умножителями с размерностью операндов 36×36, из них задействован только один, так как процессорное ядро or1200 содержит 1 MAC-блок, 12 блоков двухпортовой памяти, из которых задействовано только два, а также 475 КЛБ, из которых задействовано 257. Для реализации комбинационной логики требуется 1 267 6-входовых LUT и 691 триггер. В трассировочных ресурсах

используется маршрутизатор типа Wilton с однонаправленными межсоединениями. Оптимальная ширина трассировочного канала — 74 межсоединения.

Максимальная тактовая частота работы процессорного ядра в базисе академической ПЛИС составляет f_max: 74,9708 МГц.

```
/home/tricut/work/VTR7/vtr_flow/../ODIN_II/odin_II.exe -c odin_config.xml
Welcome to ODIN II version 0.1 - the better High level synthesis tools++ targetting FPGAs (mainly VPR)
Email: jamieson.peter@gmail.com and ken@unb.ca for support issues
Reading Configuration file
Reading FPGA Architecture file
--------------------------------------------------------------------
High-level synthesis Begin
Parser starting - we'll create an abstract syntax tree. Note this tree can be viewed using GraphViz (see documentation)
Optimizing module by AST based optimizations
Converting AST into a Netlist. Note this netlist can be viewed using GraphViz (see documentation)
Performing Optimizations of the Netlist
Hard Logical Memory Distribution
============================
DPRAM: 32 width 5 depth
DPRAM: 32 width 5 depth
Total Logical Memory Blocks = 2
Total Logical Memory bits = 2048
Max Memory Width = 32
Max Memory Depth = 5
Performing Partial Map to target device
Outputting the netlist to the specified output format
Successful High-level synthesis by Odin in 830.6ms
--------------------------------------------------------------------
Hard Multiplier Distribution
============================
32 X 32 => 1
Total # of multipliers = 1
```

Пример 2. Фрагмент файла odin.out

```
/home/tricut/work/VTR7/vtr_flow/../abc_with_bb_support/abc -c read or1200.odin.blif; time; resyn; resyn2; if -K 6; time; scleanup; time; scleanup; time; scleanup; time; scleanup; time; scleanup; time; scleanup; time; scleanup; time; scleanup; time; write_hie or1200.odin.blif or1200.abc.blif; print_stats
```

```
The 4-input library started with 43906 nodes and 24772 subgraphs. Time =  0.12 sec
Warning: The network contains hierarchy.
Hierarchy reader converted 65 instances of blackboxes.
elapse: 0.26 seconds, total: 0.26 seconds
or1200_flat : i/o = 588/ 509 lat = 691 nd = 3136 net = 14565 aig = 16663 lev = 27
```

Пример 3. Фрагмент файла abc.out

```
# Benchmark "or1200_flat" written by ABC on Tue Jan 19 14:50:15 2016
.model or1200_flat
.inputs top^clk top^rst top^icpu_dat_i~0 top^icpu_dat_i~1 top^icpu_dat_i~2 \
 top^icpu_dat_i~3 top^icpu_dat_i~4 top^icpu_dat_i~5 top^icpu_dat_i~6 \
 top^icpu_dat_i~7 top^icpu_dat_i~8 top^icpu_dat_i~9 top^icpu_dat_i~10 \
 top^icpu_dat_i~11 top^icpu_dat_i~12 top^icpu_dat_i~13 top^icpu_dat_i~14 \
 top^icpu_dat_i~15 top^icpu_dat_i~16 top^icpu_dat_i~17 top^icpu_dat_i~18 \
 top^icpu_dat_i~19 top^icpu_dat_i~20 top^icpu_dat_i~21 top^icpu_dat_i~22 \
 top^icpu_dat_i~23 top^icpu_dat_i~24 top^icpu_dat_i~25 top^icpu_dat_i~26 \
 top^icpu_dat_i~27 top^icpu_dat_i~28 top^icpu_dat_i~29 top^icpu_dat_i~30 \
…
.model multiply
.inputs a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9] a[10] a[11] a[12] \
 a[13] a[14] a[15] a[16] a[17] a[18] a[19] a[20] a[21] a[22] a[23] a[24] \
 a[25] a[26] a[27] a[28] a[29] a[30] a[31] a[32] a[33] a[34] a[35] b[0] \
 b[1] b[2] b[3] b[4] b[5] b[6] b[7] b[8] b[9] b[10] b[11] b[12] b[13] b[14] \
 b[15] b[16] b[17] b[18] b[19] b[20] b[21] b[22] b[23] b[24] b[25] b[26] \
 b[27] b[28] b[29] b[30] b[31] b[32] b[33] b[34] b[35]
.outputs out[0] out[1] out[2] out[3] out[4] out[5] out[6] out[7] out[8] \
 out[9] out[10] out[11] out[12] out[13] out[14] out[15] out[16] out[17] \
 out[18] out[19] out[20] out[21] out[22] out[23] out[24] out[25] out[26] \
 out[27] out[28] out[29] out[30] out[31] out[32] out[33] out[34] out[35] \
 out[36] out[37] out[38] out[39] out[40] out[41] out[42] out[43] out[44] \
 out[45] out[46] out[47] out[48] out[49] out[50] out[51] out[52] out[53] \
 out[54] out[55] out[56] out[57] out[58] out[59] out[60] out[61] out[62] \
 out[63] out[64] out[65] out[66] out[67] out[68] out[69] out[70] out[71]
.blackbox
.end

.model dual_port_ram
.inputs clk data2 data1 addr2[0] addr2[1] addr2[2] addr2[3] addr2[4] \
 addr2[5] addr2[6] addr2[7] addr2[8] addr2[9] addr2[10] addr2[11] addr2[12] \
 addr2[13] addr2[14] addr1[0] addr1[1] addr1[2] addr1[3] addr1[4] addr1[5] \
 addr1[6] addr1[7] addr1[8] addr1[9] addr1[10] addr1[11] addr1[12] \
 addr1[13] addr1[14] we2 we1
.outputs out2 out1
.blackbox
.end
```

Пример 4. Фрагмент blif–файла or1200.pre-vpr.blif

/home/tricut/work/VTR7/vtr_flow/../vpr/vpr or1200 or1200 --blif_file or1200.pre-vpr.blif --timing_analysis on --timing_driven_clustering on --cluster_seed_type timing --sdc_file --seed 1 --nodisp

VPR FPGA Placement and Routing.
Version: Version 7.0
Compiled: Jan 19 2016.
University of Toronto
vpr@eecg.utoronto.ca
This is free open source code under MIT license.
BLIF circuit stats:
13 LUTs of size 0
0 LUTs of size 1
157 LUTs of size 2
556 LUTs of size 3
405 LUTs of size 4
656 LUTs of size 5
1267 LUTs of size 6
385 of type input
394 of type output
691 of type latch
3054 of type names
64 of type dual_port_ram
0 of type single_port_ram
1 of type multiply
Timing analysis: ON
Circuit netlist file: or1200.net
Circuit placement file: or1200.place
Circuit routing file: or1200.route
Circuit SDC file:
Operation: RUN_FLOW
Packer: ENABLED
Placer: ENABLED
Router: ENABLED
PackerOpts.allow_early_exit: FALSE
PackerOpts.allow_unrelated_clustering: TRUE
PackerOpts.alpha_clustering: 0.750000
PackerOpts.aspect: 1.000000
PackerOpts.beta_clustering: 0.900000
PackerOpts.block_delay: 0.000000
PackerOpts.cluster_seed_type: TIMING
PackerOpts.connection_driven: TRUE
PackerOpts.global_clocks: TRUE
PackerOpts.hill_climbing_flag: FALSE
PackerOpts.inter_cluster_net_delay: 1.000000
PackerOpts.intra_cluster_net_delay: 0.000000
PackerOpts.recompute_timing_after: 32767

PackerOpts.sweep_hanging_nets_and_inputs: TRUE
PackerOpts.timing_driven: TRUE
PlacerOpts.place_freq: PLACE_ONCE
PlacerOpts.place_algorithm: PATH_TIMING_DRIVEN_PLACE
PlacerOpts.pad_loc_type: FREE
PlacerOpts.place_cost_exp: 1.000000
PlacerOpts.inner_loop_recompute_divider: 0
PlacerOpts.recompute_crit_iter: 1
PlacerOpts.timing_tradeoff: 0.500000
PlacerOpts.td_place_exp_first: 1.000000
PlacerOpts.td_place_exp_last: 8.000000
PlaceOpts.seed: 1
AnnealSched.type: AUTO_SCHED
AnnealSched.inner_num: 1.000000
RouterOpts.route_type: DETAILED
RouterOpts.router_algorithm: TIMING_DRIVEN
RouterOpts.base_cost_type: DELAY_NORMALIZED
RouterOpts.fixed_channel_width: NO_FIXED_CHANNEL_WIDTH
RouterOpts.acc_fac: 1.000000
RouterOpts.bb_factor: 3
RouterOpts.bend_cost: 0.000000
RouterOpts.first_iter_pres_fac: 0.500000
RouterOpts.initial_pres_fac: 0.500000
RouterOpts.pres_fac_mult: 1.300000
RouterOpts.max_router_iterations: 50
RouterOpts.astar_fac: 1.200000
RouterOpts.criticality_exp: 1.000000
RouterOpts.max_criticality: 0.990000
RoutingArch.directionality: UNI_DIRECTIONAL
RoutingArch.switch_block_type: WILTON
RoutingArch.Fs: 3
Nets on critical path: 9 normal, 0 global.
Total logic delay: 1.0119e-08 (s), total net delay: 3.21953e-09 (s)
Final critical path: 13.3385 ns
f_max: 74.9708 MHz

Пример 5. Фрагмент файла vpr.out

```
The circuit will be mapped into a 25 x 25 array of clbs.
Resource usage...
        Netlist    0 blocks of type: <EMPTY>
        Architecture 10    blocks of type: <EMPTY>
        Netlist    779     blocks of type: io
        Architecture 800   blocks of type: io
        Netlist    257     blocks of type: clb
        Architecture 475   blocks of type: clb
        Netlist    1 blocks of type: mult_36
        Architecture 18    blocks of type: mult_36
        Netlist    2 blocks of type: memory
        Architecture 12    blocks of type: memory
```

Пример 6. Фрагмент файла vpr_stdout.log

На рис. 1.26 показано первичное размещение микропроцессорного ядра or1200 в базис академической ПЛИС (до оптимизации), а на рис. 1.27 после этапа размещения и трассировки. На рис. 1.28 показан фрагмент ПЛИС с трассировочными ресурсами. Подключение входов/выходов КЛБ к трассировочным каналам осуществляется с четырех сторон. Трассировочные каналы сегментируются маршрутизаторами (матрица переключателей) типа Wilton с коэффициентом разветвления Fs = 3. Сегментированная трассировочная структура реализуется с использованием однонаправленных межсоединений (unidirectional). В нашем случае имеем 74 межсоединения, равномерно распределенных по всей площади кристалла. На рис. 1.29 показана детализация трассировочных ресурсов. Показаны маршрутизаторы типа Wilton в трассировочных каналах, соединительные блоки для подключения КЛБ к каналам. Используются только мультиплексорные структуры на n-МОП-ключах.

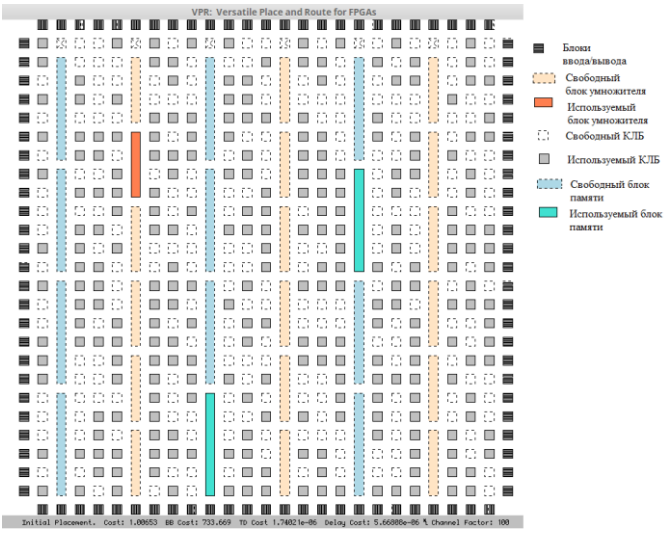


Рис. 1.26. Первичное размещение микропроцессорного ядра or1200 в ПЛИС (до оптимизации)

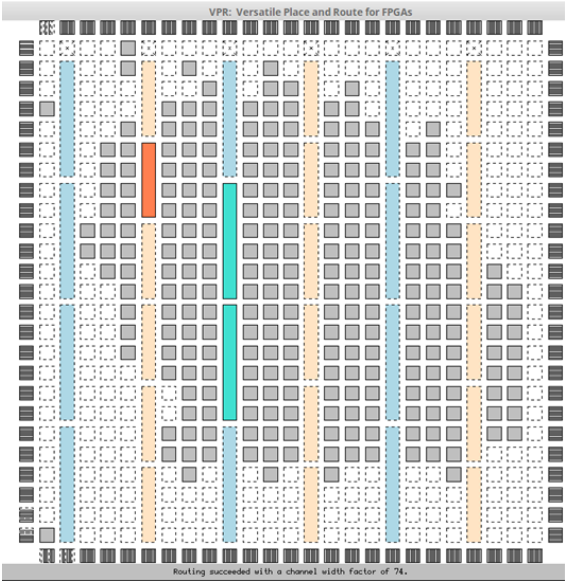


Рис. 1.27. После этапа размещения и трассировки микропроцессорного ядра or1200

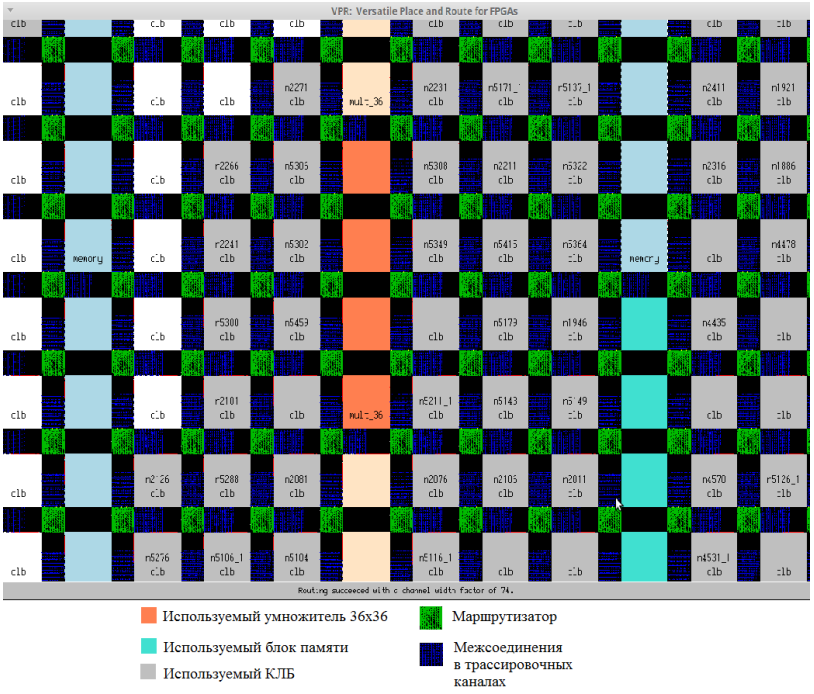


Рис. 1.28. Фрагмент ПЛИС с трассировочными ресурсами

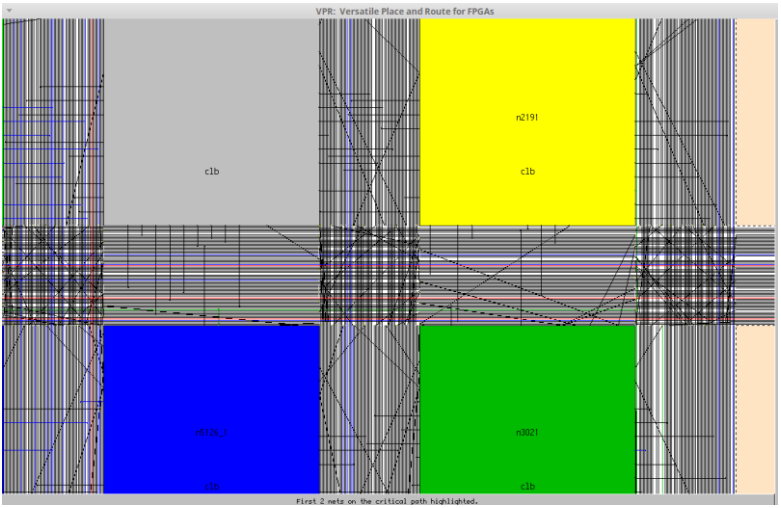


Рис. 1.29. Детализация трассировочных ресурсов ПЛИС

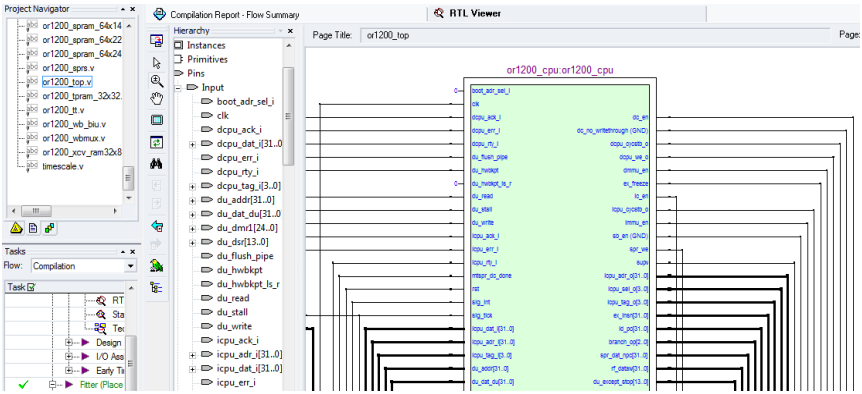


Рис. 1.30. Представление проекта микропроцессорного ядра or1200 на RTL-уровне в САПР Quartus II v.9.0 в составе более сложного проекта or1200_top

Рис. 1.31. Размещение проекта микропроцессорного ядра or1200 в ПЛИС Altera Stratix III EP3SL340F1760C2

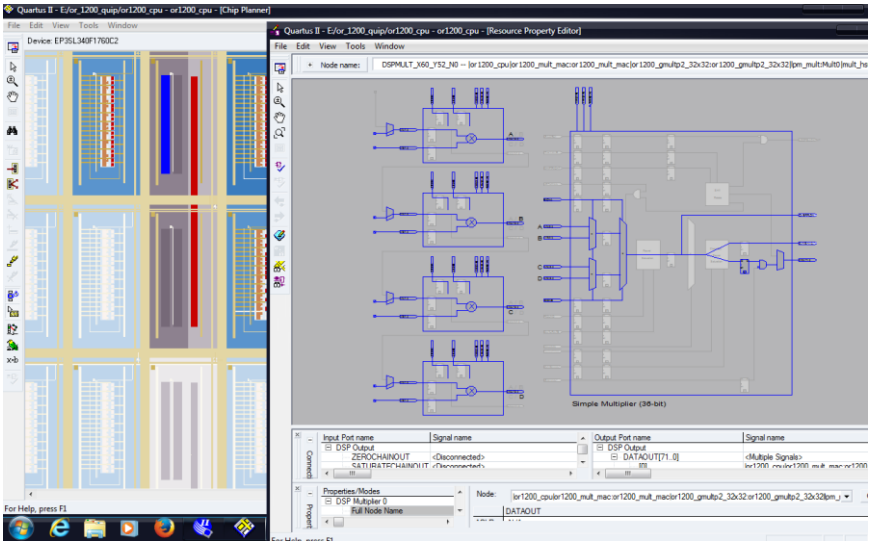


Рис. 1.32. Четыре умножителя с размерностью операндов 18×18 и дерево сумматоров для реализации умножителя 36×36

На рис. 1.30 показано представление проекта микропроцессорного ядра or1200 на RTL-уровне в САПР Quartus II v.9.0 в составе более сложного проекта or1200_top. На рис. 1.31 показано размещение проекта микропроцессорного ядра or1200 (or1200_cpu.v) в ПЛИС Altera Stratix III EP3SL340F1760C2 с использованием САПР Quartus II v.9.0. ПЛИС EP3SL340F1760C2 выбрана из соображений, что для реализации проекта or1200_cpu.v требуется 1026 пользовательских входов/выводов. Для ПЛИС же Stratix IV EP4SGX230 доступно максимально 888 пользовательских входов/выходов.

На рис. 1.32 показано, как с помощью четырех умножителей с размерностью операндов 18×18 реализуется умножитель с размерностью 36×36. Анализ задействованных ресурсов ПЛИС при реализации проекта микропроцессорного ядра or1200 при использовании САПР Quartus II v.9.0 и САПР VTR показан в табл. 1.1.

Таблица 1.1

Анализ задействованных ресурсов ПЛИС при реализации проекта микропроцессорного ядра or1200 при использовании САПР Quartus II v.9.0 и САПР VTR 7

Ресурсы ПЛИС	**ПЛИС Altera Stratix III EP3SL340F1760C2 (or1200_cpu.v)**	**Академическая ПЛИС (or1200.v)**
Технология, проектные нормы	КМОП, 65 нм	КМОП, 40 нм
Конфигурируемых блоков, шт.	170 АЛМ	257 КЛБ
Адаптивных LUT для реализации комбинационных функций, шт.	2 300 (из них 462 6- LUT)	1 267 6- LUT
Триггеров для реализации последовательностной логики, шт.	841	691
Аппаратные ЦОС-блоки, встроенные в базис ПЛИС, шт.	1 умножитель 36×36 (4 умножителя 18×18)	1 умножитель 36×36
Блоки памяти, бит	2 блока типа M9K в режиме Simple Dual Port (2 048)	2 (2 048)
Входов/выходов	1 026 (банков ввода/вывода 24)	779
Максимальная частота в наихудшем случае, Fmax, МГц	143,86 (временная модель Slow 1 200 мВ 85°C)	74,97

1.3. САПР VTR 8 для проектирования академических ПЛИС

В данном разделе предлагается рассмотреть некоторые особенности архитектурного файла САПР VTR 8.0 (Verilog to Routing), используемого для проектирования академических ПЛИС типа FPGA с одноуровневой структурой трассировочных ресурсов, когда конфигурируемые логические блоки (КЛБ) окружены с четырех сторон межсоединениями горизонтальных и вертикальных трассировочных каналов, равномерно распределенных по всей площади кристалла.

В САПР VTR 8.0 имеются заранее заготовленные различные архитектурные файлы, например, файл k6_frac_N10_mem32K_40nm.xml позволяет разработать архитектуру академической ПЛИС, близкую к ПЛИС компании Altera (Intel) серии Stratix IV с умножителями и блоками памяти, а в самой инструкции приводится фрагмент архитектурного файла, описывающий секцию КЛБ ПЛИС компании Xilinx серии Virtex-6.

Будем использовать архитектурный файл ПЛИС, без умножителей и блоков памяти, близкий к ПЛИС серии Stratix III. Предполагается, что ПЛИС в дальнейшем будет изготовлена по 65 нм технологическому КМОП-процессу кремниевой фабрики TSMC с использованием метода стандартных ячеек, применяемого для проектирования заказных цифровых специализируемых БИС.

КЛБ (кластер) академической ПЛИС, поддерживаемый САПР VTR 8.0, состоит, например, из 10 логических элементов (ЛЭ или BLE) с числом N = 10 (рис. 1.33). Логический элемент состоит из 6-входовой LUT и триггера.

Коммутаторы или маршрутизаторы обеспечивают бесконфликтную параллельную передачу информации с множества Y входов на множество Z выходов. В зарубежной литературе их называют «кросс-бары», которые бывают полной или частичной (разряженные) коммутации. В данном примере ис-

пользуется маршрутизатор Wilton на мультиплексорных структурах для однонаправленных межсоединений.

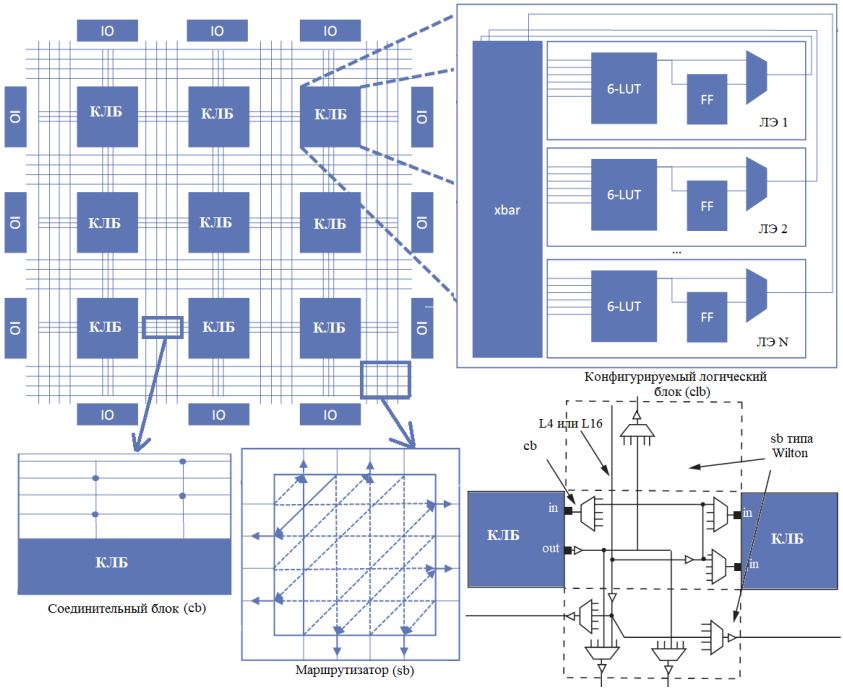


Рис. 1.33. Архитектура академической ПЛИС с одноуровневой структурой межсоединений, используемая в САПР VTR 8.0

Например, в индустриальной ПЛИС серии FLEX 8K компании Altera подключение кластера из 8 ЛЭ с 4 входовыми LUT-таблицами с числом входов 24 к горизонтальному трассировочному каналу из 168 межсоединений осуществляется с помощью мультиплексора частичной коммутации 168 входов × 24 выхода (разреженный коммутатор на 1/12). Каждая строка (FastTrack, «быстрое межсоединение») из канала может быть скоммутирована на входы кластера.

Внутри кластера коммутация межсоединений осуществляется с помощью мультиплексоров полной коммутации.

Мультиплексор с полной коммутацией является наиболее гибким, однако, его недостатком является большое количество ключей коммутации с соответствующим числом конфигурационной памяти.

Адаптивный логический элемент (fle) имеет 8-входовую таблицу перекодировок (8-входовая LUTs или адаптивный LUT) и два триггера. Адаптивный LUT может быть сконфигурирован как чисто 6-входовой LUT (ble6) и/или один триггер или как два 5-входовых с двумя общими входами (lut5inter) и двумя триггерами (рис. 1.34). LUT КЛБ академической ПЛИС по аналогии с ПЛИС Altera — это LUT адаптивного логического модуля (АЛМ), но только режимов конфигурации LUT два. Например, у ПЛИС Stratix 10 АЛМ состоит из 8-входовой адаптивной LUT, имеющей семь режимов работы, пары сумматоров и четырех триггеров.

Рассмотрим некоторые особенности формирования архитектурного файла ПЛИС. В теге device задается следующая спецификация. Сопротивления n-МОП и p-МОП транзисторов с минимальной шириной канала транзисторов, размер транзисторов в мультиплексах соединительных блоков (ipin). Распределение межсоединений в горизонтальном и вертикальном трассировочных каналах, их относительную ширину и тип маршрутизатора в канале. Межсоединения в каналах могут быть как однонаправленные, так и двунаправленные. Однонаправленные межсоединения объединяются в пары.

ipin_mux_trans_size указывается в условных единицах площади, привязанных к минимальной ширине (канала) транзистора для получения минимальной площади. Соединительный блок реализуется как мультиплексор (рис. 1.35). C_ipin_cblock — входная емкость буфера, изолирующая межсоединение в трассировочном канале от соединительных блоков (мультиплексоров), которые выбирают сигнал для его подключения ко входу КЛБ.

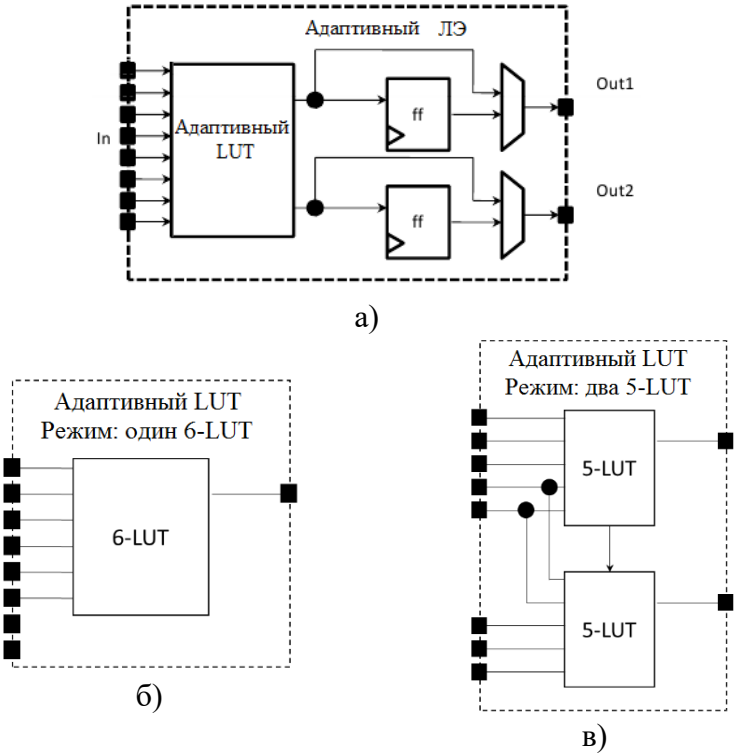


Рис. 1.34. Адаптивный логический элемент
и режимы кофигурации LUT:
а — структура адаптивного логического элемента (fle);
б — 6-входовой LUT (ble6); *в* — два 5-входовых LUT
для организации 8-входового LUT (lut5inter)

В данном примере задается одинаковая ширина трассировочных x- и y-каналов ядра кристалла ПЛИС и его периферии и маршрутизатор типа Wilton на мультиплексорных структурах для однонаправленных межсоединений.

```
<device>
  <sizing R_minW_nmos="8926" R_minW_pmos="16067"
ipin_mux_trans_size="1.222260"/>
  <timing C_ipin_cblock="1.47e-15" T_ipin_cblock="7.247000e-11"/>
<area grid_logic_tile_area="0"/>
```

```
<chan_width_distr>
 <x distr="uniform" peak="1.000000"/>
 <y distr="uniform" peak="1.000000"/>
</chan_width_distr>
<switch_block type="wilton" fs="3"/>
</device>
```

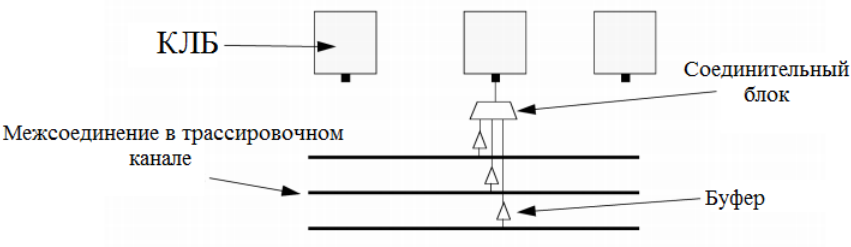


Рис. 1.35. Описание соединительных блоков ПЛИС, подключающие межсоединения из трассировочного канала ко входам кластера

Коэффициент объединения по входу fc_in 0,15 означает, что на каждый вход кластера с помощью соединительного блока подключается до 15% доступных межсоединений в трассировочном канале шириной W.

Тег <pinlocations pattern="spread"/> используется для связывания контактов КЛБ с трассировочным каналом. Используются специальные шаблоны для указания, с какой стороны располагаются входы/выходы логического блока. Шаблон "spread" соответствует равномерному распределению контактов со всех сторон КЛБ круговым способом.

Если у КЛБ 33 контакта (входов I = 22, выходов O = 10), то 8 контактов назначаются по всем сторонам, кроме одной, для которой назначены 9 контактов (один из них для подключения синхросигнала).

При использовании двунаправленных межсоединений все маршрутизаторы имеют коэффициент разветвления *Fs* = 3, т.е., когда горизонтальный и вертикальные трассировочные каналы пересекаются, каждое межсоединение может коммутироваться с тремя другими межсоединениями. Топология

маршрутизаторов может быть трех вариантов. Например, планарный или доменный маршрутизатор, используемый в ранних ПЛИС Xilinx серии 4000, в котором двунаправленное межсоединение с порядковым номером 0 может подключаться только к другим межсоединениям с номерами 0 (типа disjoint) и т. д. В маршрутизаторах типа disjoint используются n-МОП шеститранзисторные ключи, которые могут быть реализованы с использованием мультиплексоров и буферов с третьим состоянием. Для коммутации однонаправленных межсоединений используются только мультиплексорные структуры (маршрутизатор Wilton).

В настоящее время индустриальные ПЛИС имеют сегментируемую трассировочную структуру с использованием однонаправленных межсоединений (unidirectional).

В общем случае маршрутизатор Wilton является лучшим из этих трех топологий. В настоящее время считается, что использование однонаправленных межсоединений в совокупности с мультиплексорными структурами в маршрутизаторах наиболее перспективно, так как позволяет получать существенный выигрыш по быстродействию и по площади кристалла, а сам маршрутизатор обладает наилучшей разводимостью.

Содержимое в теге <complexblocklist> описывает сложные логические блоки, найденные внутри ПЛИС. Каждый тип сложного логического блока определяется тегом <pb_type name = "string" height = "int"> внутри тега <complexblocklist>. Атрибут name — это имя для сложного блока. Атрибут height указывает, сколько слоев сетки занимает блок.

Внутри сложного блока используется иерархия тегов <pb_type>. Тег верхнего уровня <pb_type> задает сложный блок. Дети <pb_type> — это либо кластеры (которые содержат другие теги <pb_type>), либо примитивы (листья, которые не содержат другие теги <pb_type>).

Кластеры могут содержать другие кластеры и примитивы, поэтому нет ограничений на иерархию. Все теги <pb_type>

содержат атрибут num_pb = "int", который описывает количество экземпляров этого конкретного типа кластера или листового блока в этом разделе иерархии. Все теги <pb_type> должны иметь атрибут name = "string", где имя должно быть уникальным в отношении любого родительского, родственного или дочернего тега <pb_type>.

Лист <pb_type> тега может иметь атрибут blif_model = "string". Этот атрибут описывает тип блока в blif-файле, что этот конкретный лист может реализовать. Например, лист, который реализует LUT, должен иметь blif_model = ".names". Аналогично лист, который реализует .subckt user_block_A, должен иметь атрибут blif_model ="subckt user_block_A". Входные, выходные и/или синхросигналы, порты для этих листов должны соответствовать указанным портам в разделе <models> файла архитектуры.

Описание межсоединений в ПЛИС (тег <interconnect>). Тег mode содержит теги <pb_type> и тег <interconnect>. Ниже описываются теги, используемые в теге <interconnect>.

<complete name = "string" input = "string" output = "string" />

Обязательные атрибуты:

• name — идентификатор для межсоединения;

• вход — контакты, которые являются входами для этого межсоединения;

• выход — контакты, которые являются выходами этого межсоединения.

На рис. 1.36 показано использование кросс-бара с полной коммутацией в кластере из ЛЭ (BLE) ранних академических ПЛИС. Наиболее распространенный кластер состоит из N = 10 логических элементов, число входов кластера I = 22, число выходов O = 10, число входов LUT K = 4. Для достижения 98% логического использования кластера требуется число входов, равное 2N + 2 (I = 2N + 2). Все входы и выходы кластера логически эквивалентны (equivalent="true"). Логическая эквивалентность входов означает, что подключения с этими входами могут быть заменены без изменения их функцио-

нальности. Входы в BLE могут поступать либо со входов КЛБ, либо с других входов BLE в логическом блоке через кросс-бар с полной коммутацией.

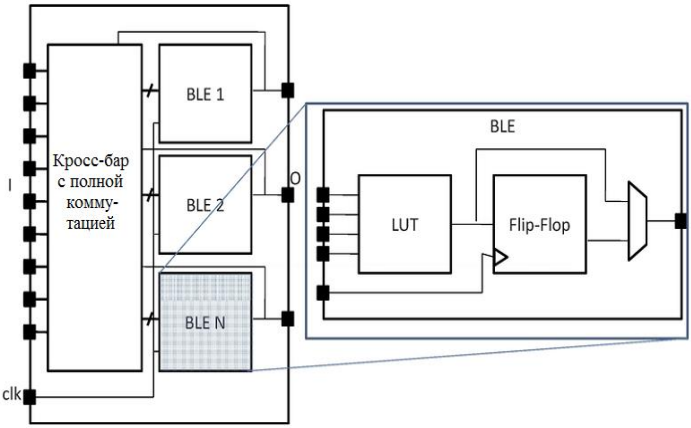


Рис. 1.36. Кросс-бар с полной коммутацией в кластере из N логических элементов академической ПЛИС

Кросс-бары в VTR различают с полной коммутацией (complete), с прямой коммутацией (direct) и с шинным мультиплексированием (bus-based multiplexer).

Кросс-бар с полной коммутацией (любой вход коммутатора может быть подключен к любому его выходу) описывается следующей конструкцией (рис. 1.37):

<complete input="Top.in" output="Child.in"/>

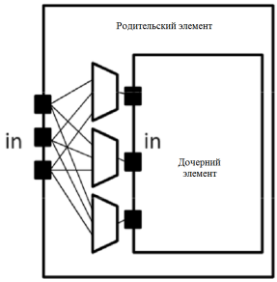


Рис. 1.37. Кросс-бар с полной коммутацией

Прямое подключение входов коммутатора к его выходам по принципу «один в один» (рис. 1.38):

<direct input="Top.in[2:1]" output="Child[1].in"/>

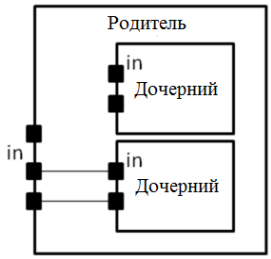


Рис. 1.38. Кросс-бар с прямой коммутацией

Кросс-бар с шинным мультиплексированием (рис. 1.39):

<mux input="Top.A Top.B" output="Child.in"/>

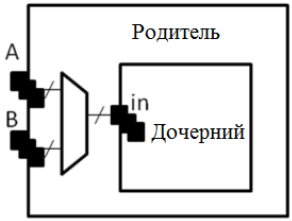


Рис. 1.39. Кросс-бар с шинным мультиплексированием

Кластер (clb) описывается с помощью тега:

```
<pb_type name="clb">
<input name="I" num_pins="22" equivalent="true"/>
<output name="O" num_pins="10" equivalent="true"/>
<clock name="clk" equivalent="false"/>
```

Логический элемент (ble) описывается с помощью тега:

```
<pb_type name="ble" num_pb="10">
<input name="in" num_pins="4"/>
<output name="out" num_pins="1"/>
<clock name="clk"/>
```

В простейшем случае ЛЭ состоит из 4-LUT и триггера тактируемого фронтом синхросигнала (FF). LUT описывается blif-моделью с именем .names, а триггер — blif-моделью с именем .latch.

```
 <pb_type name="lut_4" blif_model=".names" num_pb="1" class="lut">
<input name="in" num_pins="4" port_class="lut_in"/>
<output name="out" num_pins="1" port_class="lut_out"/>
</pb_type>
<pb_type name="ff" blif_model=".latch" num_pb="1" class="flipflop">
<input name="D" num_pins="1" port_class="D"/>
<output name="Q" num_pins="1" port_class="Q"/>
<clock name="clk" port_class="clock"/>
</pb_type>
```

Ниже представлена коммутация межсоединений ЛЭ с помощью тега <interconnect> (рис. 1.40.):

```
<interconnect>
<direct input="lut_4.out" output="ff.D"/>
<direct input="ble.in" output="lut_4.in"/>
<mux input="ff.Q lut_4.out" output="ble.out"/>
<direct input="ble.clk" output="ff.clk"/>
</interconnect>
</pb_type>
```

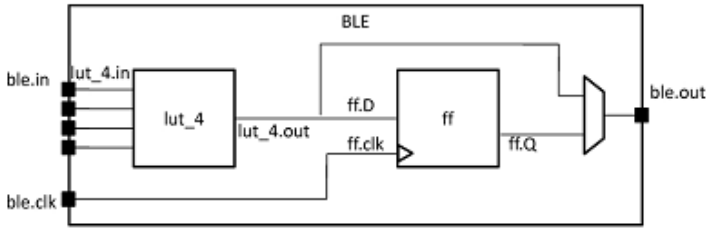


Рис. 1.40. Имена сигналов, используемые для обозначения межсоединений внутри ЛЭ

Локальная внутрикластерная коммутация, осуществляемая с помощью кросс-баров, также описывается с помощью тега <interconnect>. С помощью мультиплексоров полной коммутации (complete) можно: любой вход кластера CLB

(clb.I) подключить к любому из входов 10 BLE (BLE [9: 0] .in); любой из 10 выходов ЛЭ ble[9:0].out подключить на любой вход кластера clb ble[9:0].in. Синхросигнал clk со входа кластера подключается к тактовым входам триггеров ЛЭ также через мультиплексор полной коммутации:

```
<interconnect>
<complete input="{clb.I ble[9:0].out}" output="ble[9:0].in"/>
<complete input="clb.clk" output="ble[9:0].clk"/>
<direct input="ble[9:0].out" output="clb.O"/>
</interconnect>
```

Тег <complete>, <direct> или <mux> может принимать дополнительный, необязательный тег, называемый <pack_pattern> (шаблон упаковки), который используется для описания молекул. Шаблон упаковки представляет собой мощную пользовательскую функцию, указывающую, что инструмент САПР должен группировать определенные атомы netlist (например, LUT, FF, цепи переноса) вместе во время потока САПР.

Это позволяет САПР распознать структуры, которые имеют ограниченную гибкость, так что атомы списка соединений, которые соответствуют этим структурам, сохраняются вместе как единое целое. Этот тег влияет только на работу САПР и не оказывает никакого архитектурного влияния на определения молекул.

Пример шаблона упаковки:

```
<pack_pattern name="string" in_port="string" out_port="string"/>
```

Обязательные атрибуты:

- name — имя шаблона;
- in_port — входные контакты шаблона;
- out_port — выходные контакты шаблона.

Этот тег дает подсказку инструментам САПР, что некоторые архитектурные структуры должны оставаться вместе во время упаковки. Метка связывает края с именем шаблона. Все примитивы, связанные одним и тем же именем пачки пакетов,

становятся единым упаковочным шаблоном. Любая группа атомов в пользовательском списке соединений, соответствующая упаковочному шаблону, будет сгруппирована VPR для образования молекулы. Молекулы хранятся вместе как единое целое в VPR. Это позволяет проектировщику архитектуры ПЛИС помочь инструменту САПР присваивать атомы сложным логическим блокам, которые имеют межсоединения с очень ограниченной гибкостью. Примеры архитектурных структур, для которых можно применять упаковочные шаблоны, включают: необязательные зарегистрированные входы/выходы, цепи переноса, блоки с умножением и накопления и т. д.

Пример. Рассмотрим классический ЛЭ (BLE), который состоит из LUT и триггера. Если LUT связан с триггером в списке соединений, то проектировщик архитектуры ПЛИС хотел бы, чтобы триггер был упакован с LUT в одном и том же ЛЭ. Чтобы дать VPR указание на то, что эти блоки должны быть соединены вместе, необходимо обозначить межсоединение, соединяющее LUT и триггер в пару, с помощью шаблона pack_pattern (рис. 1.41):

<pack_pattern name="ble" in_port="lut.out" out_port="ff.D"/>

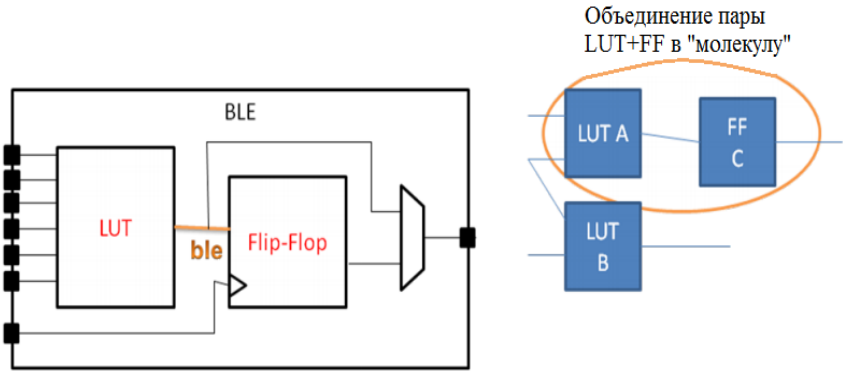


Рис. 1.41. Соединение пары LUT и триггера FF в молекулу с помощью шаблона упаковки с именем ble

Рассмотрим тег <segmentlist>, описывающий сегментацию межсоединений в трассировочном канале ПЛИС. Содержимое в теге <segmentlist> состоит из группы тегов . Тег и его содержимое описано ниже.
<segment name = "unique_name" length = "int" type = "{bidir | unidir}" freq = "float" Rmetal = "float".

Обязательные атрибуты:

• name — уникальное буквенно-цифровое имя для идентификации этого типа сегмента;

• length — количество логических блоков (элементов), охватываемых каждым сегментом, или длина сегмента. Ярлык означает, что сегменты этого типа охватывают весь массив FPGA;

• freq — частота появлений межсоединений в трассировочном канале в зависимости от его типа (однонаправленные или двунаправленные межсоединения). VPR автоматически определяет процент межсоединений для каждого типа в зависимости от суммы всех частот. В одном канале может быть несколько межсоединений различной длины, например, L4 имеет частоту 0,87, а L16 — 0,13;

• Rmetal — сопротивление межсоединения на единицу длины (с точки зрения логических блоков), в Омах. Например, сегмент длины 5 (охватывает 5 блоков) с Rmetal = 10 Ом/логический блок имеет сквозное сопротивление 50 Ом.

Cmetal — емкость межсоединения на единицу длины (с точки зрения логических блоков), в Фарадах. Например, сегмент длины 5 с Cmetal = 2e-14 Ф/логический блок будет иметь общую емкость 10e-13Ф.

Ниже показаны примеры депопуляции (удаление) маршрутизаторов (Switch Box) и соединительных блоков (Connection Box) в трассировочном канале с четырьмя линиями (рис. 1.42).

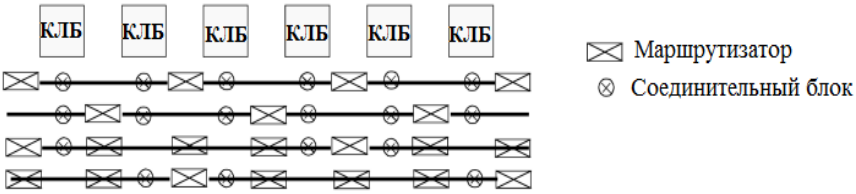


Рис. 1.42. Схемы депопуляции коммутаторов,
используемых для реализации маршрутизаторов
и соединительных блоков в линиях трассировочного канала

Например, для длины канала в 6 кластеров (L = 6) для первой линии существует схема депопуляции маршрутизаторов: sb 1 0 1 0 1 0 1. Вторая линия имеет схему депопуляции: 0 1 0 1 0 1 0. Единица указывает на существование маршрутизатора в трассировочном канале, а 0 указывает, что в этом месте его нет. В списке целых чисел для L = 6 имеется 7 записей. Для значения L должно быть L + 1 записей, разделенных пробелами. Для рассмотренных примеров депопуляции маршрутизаторов первая и вторая линии проходят непрерывно через 2 КЛБ.

Соединительный блок описывается следующим тегом:

<cb type = "pattern"> int list </ cb>

Этот тег описывает депопуляцию соединительных блоков (показано кружками) для конкретного сегмента межсоединения. Например, для первой линии при L = 6 имеем следующую схему (шаблон) подключения входов/выходов кластера к трассировочному каналу: sb 1 1 1 1 1 1.

Третья линия в трассировочном канале имеет следующий шаблон подключений: 1 0 0 1 1 0. 1 указывает на существование соединительного блока, и 0 указывает, что в этой точке нет соединительного блока. В списке целых чисел при L = 6 должно быть 6 записей, разделенных пробелами.

Архитектурный файл академической ПЛИС с архитектурой, близкой к индустриальным ПЛИС серии Stratix III, демонстрирует пример 1. При разработке трассировочной струк-

туры ПЛИС используются межсоединения L4 и L16, схожие с многоканальными соединениями, (оптимизированный набор шин различной длины и скорости) применяемыми в архитектуре ПЛИС семейства Altera Stratix II. Например, горизонтальный канал R20 ПЛИС Altera Stratix III является самым быстрым каналом, распространяющимся через 20 кластеров, а R4 обладает наивысшей трассировочной способностью. Также существуют и вертикальные трассировочные каналы C4 и C12.

```
<architecture>
<models>
</models>
<layout>
  <auto_layout aspect_ratio="1.0">
    <!--Perimeter of 'io' blocks with 'EMPTY' blocks at corners-->
    <perimeter type="io" priority="100"/>
    <corners type="EMPTY" priority="101"/>
    <!--Fill with 'clb'-->
    <fill type="clb" priority="1"/>
    </auto_layout>
</layout>
<device>
  <sizing R_minW_nmos="8926" R_minW_pmos="16067"
ipin_mux_trans_size="1.222260"/>
  <timing C_ipin_cblock="1.47e-15" T_ipin_cblock="7.247000e-11"/>
                          <area grid_logic_tile_area="0"/>
  <chan_width_distr>
    <x distr="uniform" peak="1.000000"/>
    <y distr="uniform" peak="1.000000"/>
  </chan_width_distr>
  <switch_block type="wilton" fs="3"/>
 </device>
<switchlist>
  <switch type="mux" name="0" R="551" Cin=".77e-15" Cout="4e-15" Tdel="58e-12"
mux_trans_size="2.630740" buf_size="27.645901"/>
</switchlist>
<segmentlist>
<segment freq="0.87" length="4" type="unidir" Rmetal="101" Cmetal="22.5e-15">
<mux name="0"/>
<sb type="pattern">1 1 1 1 1 </sb>
<cb type="pattern">1 1 1 1 </cb>
</segment>
<segment freq="0.13" length="16" type="unidir" Rmetal="101" Cmetal="22.5e-15">
<mux name="0"/>
```

```
<sb type="pattern">1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 </sb>
<cb type="pattern">1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1</cb>
</segment>
</segmentlist>
<complexblocklist>
<pb_type name="io" capacity="8" area="0">
  <input name="outpad" num_pins="1"/>
  <output name="inpad" num_pins="1"/>
  <clock name="clock" num_pins="1"/>
  <mode name="inpad">
   <pb_type name="inpad" blif_model=".input" num_pb="1">
    <output name="inpad" num_pins="1"/>
   </pb_type>
   <interconnect>
    <direct name="inpad" input="inpad.inpad" output="io.inpad">
     <delay_constant max="4.243e-11" in_port="inpad.inpad" out_port="io.inpad"/>
    </direct>
   </interconnect>
  </mode>
  <mode name="outpad">
   <pb_type name="outpad" blif_model=".output" num_pb="1">
    <input name="outpad" num_pins="1"/>
   </pb_type>
   <interconnect>
    <direct name="outpad" input="io.outpad" output="outpad.outpad">
     <delay_constant max="1.394e-11" in_port="io.outpad" out_port="outpad.outpad"/>
    </direct>
   </interconnect>
  </mode>
  <fc default_in_type="frac" default_in_val="0.15" default_out_type="frac"
default_out_val="0.10"/>
  <pinlocations pattern="custom">
   <loc side="left">io.outpad io.inpad io.clock</loc>
   <loc side="top">io.outpad io.inpad io.clock</loc>
   <loc side="right">io.outpad io.inpad io.clock</loc>
   <loc side="bottom">io.outpad io.inpad io.clock</loc>
  </pinlocations>
  <!-- Place I/Os on the sides of the FPGA -->
  <power method="ignore"/>
 </pb_type>
 <!-- Define I/O pads ends -->

<pb_type name="clb" area="53894">
<input name="I1" num_pins="13" equivalent="true"/>
<input name="I2" num_pins="13" equivalent="true"/>
<input name="I3" num_pins="13" equivalent="true"/>
<input name="I4" num_pins="13" equivalent="true"/>
```

```
<output name="O" num_pins="20" equivalent="true"/>
<clock name="clk" num_pins="1"/>

<pb_type name="fle" num_pb="10">
<input name="in" num_pins="8"/>
<output name="out" num_pins="2"/>
<clock name="clk" num_pins="1"/>

<!-- Dual 5-LUT mode definition begin -->
<mode name="n2_lut5">
<pb_type name="lut5inter" num_pb="1">
<input name="in" num_pins="8"/>
<output name="out" num_pins="2"/>
<clock name="clk" num_pins="1"/>

<pb_type name="ble5" num_pb="2">
<input name="in" num_pins="5"/>
<output name="out" num_pins="1"/>
<clock name="clk" num_pins="1"/>

<mode name="blut5">
<pb_type name="flut5" num_pb="1">
<input name="in" num_pins="5"/>
<output name="out" num_pins="1"/>
<clock name="clk" num_pins="1"/>
<pb_type name="lut5" blif_model=".names" num_pb="1" class="lut">
<input name="in" num_pins="5" port_class="lut_in"/>
<output name="out" num_pins="1" port_class="lut_out"/>
<!-- LUT timing using delay matrix -->
 <delay_matrix type="max" in_port="lut5.in" out_port="lut5.out">
       261e-12
       261e-12
       261e-12
       261e-12
       261e-12
      </delay_matrix>
</pb_type>
 <pb_type name="ff" blif_model=".latch" num_pb="1" class="flipflop">
 <input name="D" num_pins="1" port_class="D"/>
 <output name="Q" num_pins="1" port_class="Q"/>
 <clock name="clk" num_pins="1" port_class="clock"/>
 <T_setup value="66e-12" port="ff.D" clock="clk"/>
 <T_clock_to_Q max="124e-12" port="ff.Q" clock="clk"/>
 </pb_type>
<interconnect>
 <direct name="direct1" input="flut5.in" output="lut5.in"/>
 <direct name="direct2" input="lut5.out" output="ff.D">
```

```
 <pack_pattern name="ble5" in_port="lut5.out" out_port="ff.D"/>
 </direct>
<direct name="direct3" input="flut5.clk" output="ff.clk"/>
<mux name="mux1" input="ff.Q lut5.out" output="flut5.out">
<delay_constant max="25e-12" in_port="lut5.out" out_port="flut5.out"/>
<delay_constant max="45e-12" in_port="ff.Q" out_port="flut5.out"/>
</mux>
 </interconnect>
</pb_type>
<interconnect>
 <direct name="direct1" input="ble5.in" output="flut5.in"/>
 <direct name="direct2" input="ble5.clk" output="flut5.clk"/>
 <direct name="direct3" input="flut5.out" output="ble5.out"/>
</interconnect>
</mode>
</pb_type>
<interconnect>
<direct name="direct1" input="lut5inter.in[6]" output="ble5[0:0].in[4]"/>
<direct name="direct2" input="lut5inter.in[3:0]" output="ble5[0:0].in[3:0]"/>
<direct name="direct4" input="lut5inter.in[7]" output="ble5[1:1].in[4]"/>
<direct name="direct5" input="lut5inter.in[5:4]" output="ble5[1:1].in[3:2]"/>
<direct name="direct6" input="lut5inter.in[1:0]" output="ble5[1:1].in[1:0]"/>
<direct name="direct3" input="ble5[1:0].out" output="lut5inter.out"/>
<complete name="complete1" input="lut5inter.clk" output="ble5[1:0].clk"/>
</interconnect>
</pb_type>
<interconnect>
 <direct name="direct1" input="fle.in" output="lut5inter.in"/>
 <direct name="direct2" input="lut5inter.out" output="fle.out"/>
 <direct name="direct3" input="fle.clk" output="lut5inter.clk"/>
</interconnect>
</mode>

<!-- 6-LUT mode definition begin -->
<mode name="n1_lut6">
<!-- Define 6-LUT mode -->

<pb_type name="ble6" num_pb="1">
<input name="in" num_pins="6"/>
<output name="out" num_pins="1"/>
<clock name="clk" num_pins="1"/>
<pb_type name="lut6" blif_model=".names" num_pb="1" class="lut">
<input name="in" num_pins="6" port_class="lut_in"/>
<output name="out" num_pins="1" port_class="lut_out"/>
<!-- LUT timing using delay matrix -->
<delay_matrix type="max" in_port="lut6.in" out_port="lut6.out">
      261e-12
```

```
        261e-12
        261e-12
        261e-12
        261e-12
        261e-12
       </delay_matrix>
</pb_type>
 <pb_type name="ff" blif_model=".latch" num_pb="1" class="flipflop">
 <input name="D" num_pins="1" port_class="D"/>
 <output name="Q" num_pins="1" port_class="Q"/>
 <clock name="clk" num_pins="1" port_class="clock"/>
 <T_setup value="66e-12" port="ff.D" clock="clk"/>
 <T_clock_to_Q max="124e-12" port="ff.Q" clock="clk"/>
 </pb_type>

<interconnect>
 <direct name="direct1" input="ble6.in" output="lut6[0:0].in"/>
 <direct name="direct2" input="lut6.out" output="ff.D">
 <pack_pattern name="ble6" in_port="lut6.out" out_port="ff.D"/>
 </direct>
<direct name="direct3" input="ble6.clk" output="ff.clk"/>
<mux name="mux1" input="ff.Q lut6.out" output="ble6.out">
<delay_constant max="25e-12" in_port="lut6.out" out_port="ble6.out"/>
<delay_constant max="45e-12" in_port="ff.Q" out_port="ble6.out"/>
</mux>
 </interconnect>
</pb_type>
<interconnect>
 <direct name="direct11" input="fle.in[3:0]" output="ble6.in[3:0]"/>
 <direct name="direct12" input="fle.in[7:6]" output="ble6.in[5:4]"/>
<direct name="direct2" input="ble6.out" output="fle.out[0:0]"/>
 <direct name="direct3" input="fle.clk" output="ble6.clk"/>
</interconnect>
</mode>
</pb_type>
<interconnect>
<complete name="lutA" input="clb.I4 clb.I3 fle[1:0].out fle[3:2].out fle[8:8].out"
output="fle[9:0].in[0:0]">
<!-- delay matrix -->
</complete>
<complete name="lutB" input="clb.I3 clb.I2 fle[3:2].out fle[5:4].out fle[9:9].out"
output="fle[9:0].in[1:1]">
<!-- delay matrix -->
</complete>
<complete name="lutC" input="clb.I2 clb.I1 fle[5:4].out fle[7:6].out fle[8:8].out"
output="fle[9:0].in[2:2]">
<!-- delay matrix -->
```

```
</complete>
<complete name="lutD" input="clb.I4 clb.I2 fle[1:0].out fle[5:4].out fle[9:9].out"
output="fle[9:0].in[3:3]">
<!-- delay matrix -->
</complete>
<complete name="lutE" input="clb.I3 clb.I1 fle[3:2].out fle[7:6].out fle[8:8].out"
output="fle[9:0].in[4:4]">
<!-- delay matrix -->
</complete>
<complete name="lutF" input="clb.I4 clb.I1 fle[1:0].out fle[7:6].out fle[9:9].out"
output="fle[9:0].in[5:5]">
<!-- delay matrix -->
</complete>
<complete name="lutG" input="clb.I4 clb.I3 fle[1:0].out fle[3:2].out fle[8:8].out"
output="fle[9:0].in[6:6]">
<!-- delay matrix -->
</complete>
<complete name="lutH" input="clb.I3 clb.I2 fle[3:2].out fle[5:4].out fle[9:9].out"
output="fle[9:0].in[7:7]">
<!-- delay matrix -->
</complete>
<complete name="clks" input="clb.clk" output="fle[9:0].clk">
</complete>
<direct name="clbouts1" input="fle[9:0].out[0:0]" output="clb.O[9:0]"/>
<direct name="clbouts2" input="fle[9:0].out[1:1]" output="clb.O[19:10]"/>
</interconnect>
<fc default_in_type="frac" default_in_val="0.055"
default_out_type="frac" default_out_val="0.10"/>
<pinlocations pattern="spread"/>
</pb_type>
</complexblocklist>
</architecture>
```

Пример 1. Архитектурный файл академической ПЛИС с архитектурой, близкой к индустриальным ПЛИС серии Stratix III

Рассмотрим двухразрядный сумматор, реализованный в академической ПЛИС, по своей архитектуре схожей с архитектурой ПЛИС серии Straix III. Предварительно blif-файл сумматора подвергнут оптимизации программным инструментом ABC. Из рис. 1.43 видно, что межсоединения L4 и L16 не используются из-за простоты проекта. На рис. 1.44 показан

двухразрядный сумматор, реализованный в ПЛИС. Задействованные lut5 подсвечены серым цветом.

Используется только периферийный трассировочный канал для подключения элементов ввода/вывода ко входам КЛБ. Проект занимает всего 1 clb. Для реализации сумматора требуются два адаптивных логических элемента fle типа lut5inter в режиме n2_lut5 (рис. 1.44) или три lut5. В одном lut5inter задействуются два lut5, а в другом — один lut5 (задействованные lut5 подсвечены серым цветом).

Из тестов производительности САПР VTR 7.0 (Verilog-проекты) возьмем файл raygentop.v. Предварительно подвергнем оптимизации с помощью ABC. Для реализации проекта требуется 1029 КЛБ (рис. 1.45). Ширина трассировочного канала — 150 однонаправленных межсоединений. В более сложном проекте уже используются адаптивные логические элементы fle двух типов: ble6 и lut5inter (рис. 1.46). Детальное раскрытие структуры fle двух типов: ble6 и lut5inter демонстрирует рис. 1.47. На рис. 1.48 подсвечены синим цветом межсоединения L4 (по вертикали) и L16 (по горизонтали), проходящие непрерывно через 4 КЛБ и 16 КЛБ.

Рис. 1.49 демонстрирует детальное раскрытие структуры маршрутизатора Wilton в трассировочном канале. Треугольники (серые и черные) по периферии маршрутизатора показывают направление передачи сигналов по межсоединениям, которые проходят через маршрутизатор непрерывно. На концах однонаправленных межсоединений L4 (направление распространения сверху вниз), которые сегментируются маршрутизатором, находятся мультиплексоры 12 в 1 (подсвечены желтым). На рис. 1.50 показаны маршрутизаторы Wilton в трассировочном канале (фрагменты). Синим цветом подсвечены соединительные блоки, красным — выходы КЛБ, которые заводятся непосредственно в маршрутизаторы (выделены зеленым цветом). Синим цветом также подсвечены входы, а красным — выходы КЛБ.

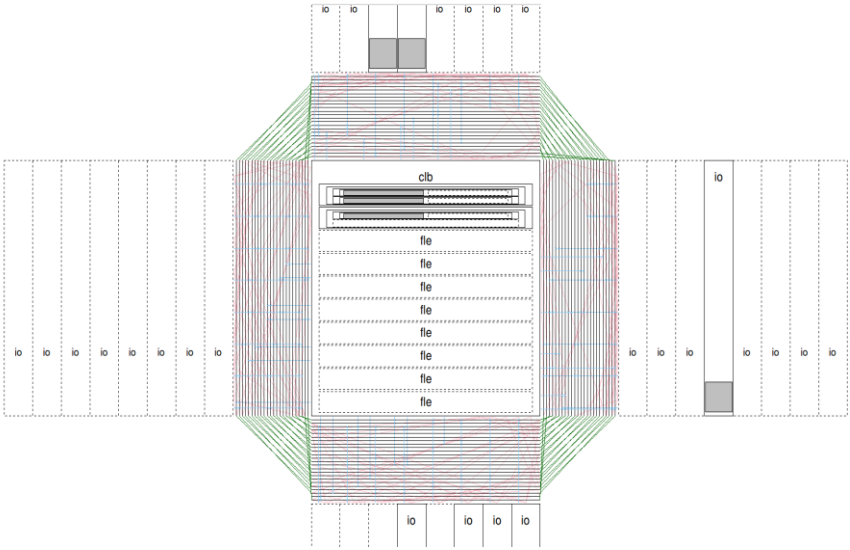


Рис. 1.43. Двухразрядный сумматор в ПЛИС

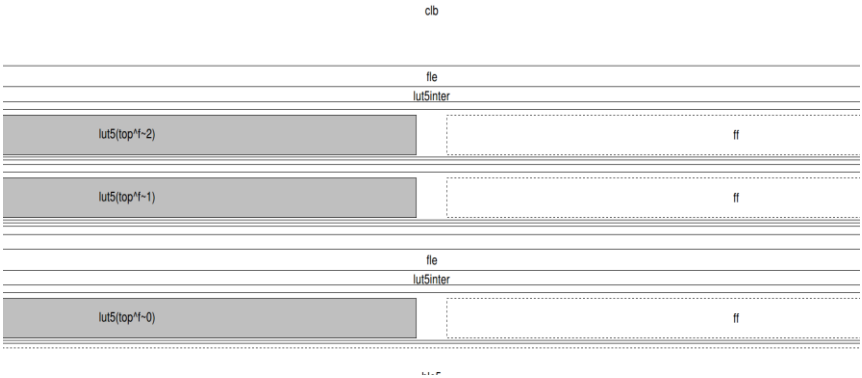


Рис. 1.44. Для реализации сумматора требуются
два блока lut5inter в режиме n2_lut5

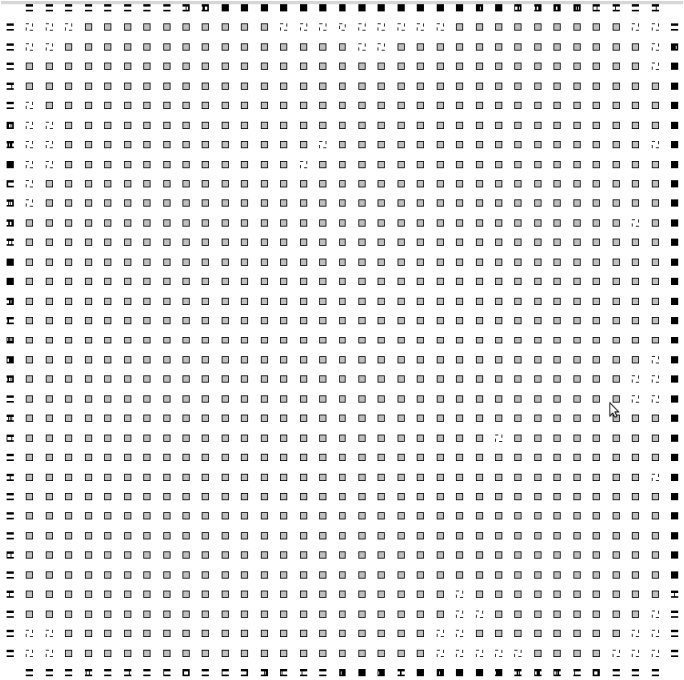
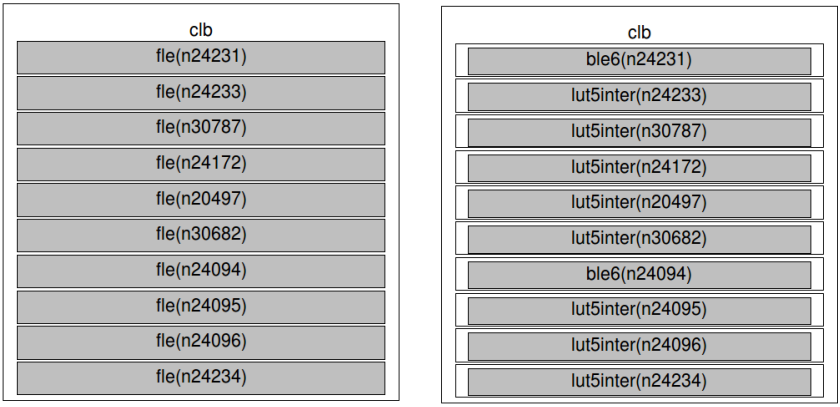


Рис. 1.45. Реализация проекта raygentop.v в академическую ПЛИС



а) б)

Рис. 1.46. Кластер (clb) из 10 адаптивных fle (*а*)
и использование адаптивных логических элементов fle
двух типов: ble6 и lut5inter (*б*)

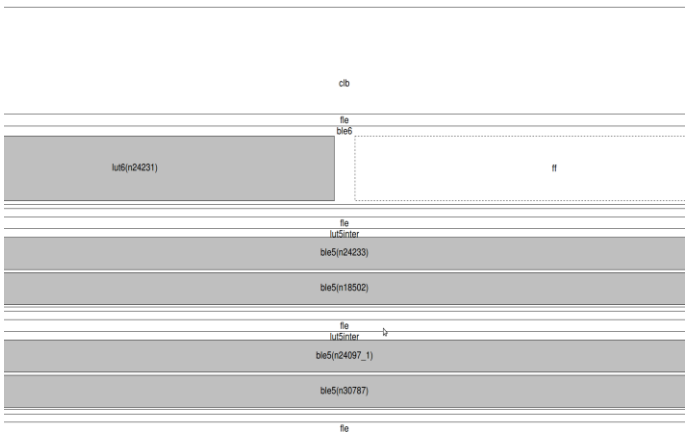
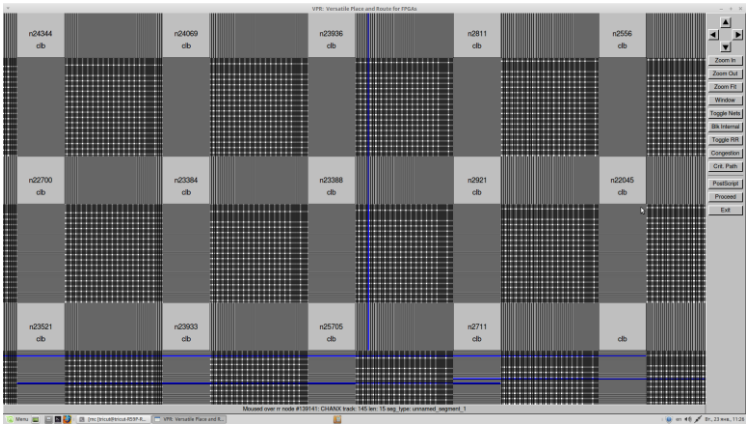
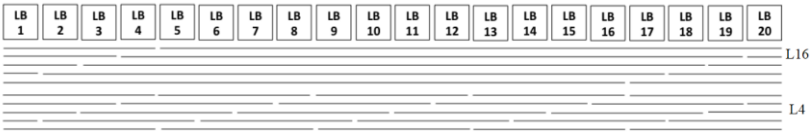


Рис. 1.47. Детальное раскрытие структуры fle двух типов: ble6 и lut5inter



а)



б)

Рис. 1.48. Подсвечены синим цветом межсоединения L4 (по вертикали) и L16 (по горизонтали), проходящие непрерывно через 4 КЛБ и 16 КЛБ (*а*) и упрощенное представление межсоединений L16 и L4 (*б*)

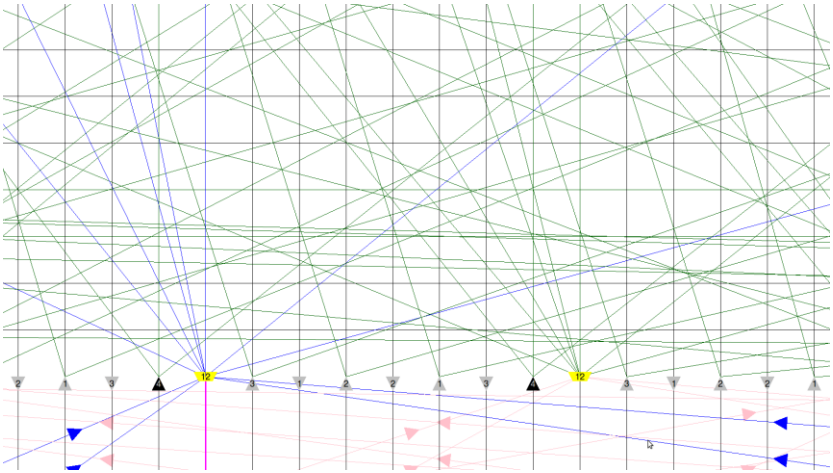


Рис. 1.49. Детальное раскрытие структуры маршрутизатора Wilton в трассировочном канале.
Мультиплексоры 12 в 1 подсвечены желтым цветом

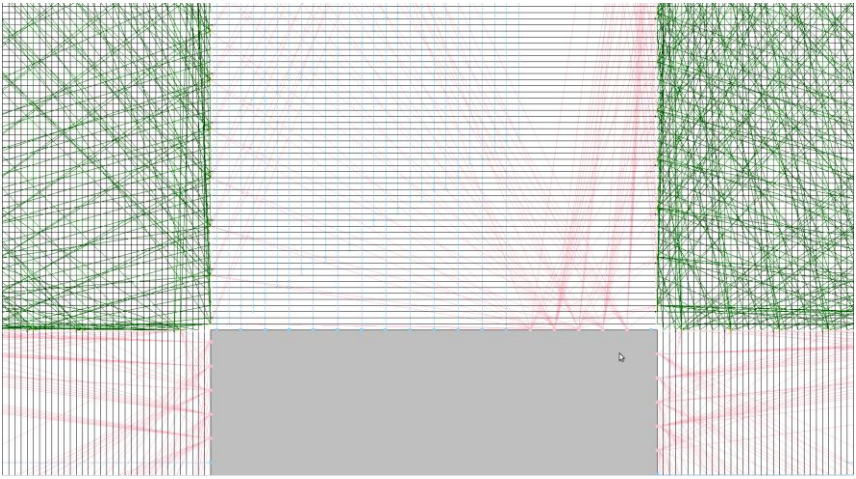


Рис. 1.50. Маршрутизаторы Wilton в трассировочном канале (фрагменты). Синим цветом подсвечены соединительные блоки, красным — выходы КЛБ, которые заводятся непосредственно в маршрутизаторы (выделены зеленым цветом)

1.4. Реализация Verilog-проектов в САПР VTR 8

В разделе предлагается рассмотреть некоторые особенности новой версии САПР VTR 8.0 (Verilog to Routing) с открытым кодом, разработанной в университете Торонто для исследования академических ПЛИС со встроенными умножителями и блоками памяти по нанопроектным нормам КМОП-технологии. САПР ПЛИС VTR8.0 (архив vtr-verilog-to-routing-master.zip) может быть скачан по адресу: (https://github.com/verilog-to-routing/vtr-verilog-to-routing).

САПР VTR 8.0 предполагает использование следующих программных инструментов: синтезатор Verilog-проектов ODIN-II, программа логической оптимизации и технологического отображения в k-LUT ABC, размещение и трассировка в базис ПЛИС VPR (Versatile Place and Route). ODIN-II (Нью-Брансвик, Канада, UNB) и ABC (Калифорнийский университет в Беркли, США, University of California, Berkeley), VPR (Канада, Торонто, http://www.eecg.utoronto.ca/vpr) являются открытыми программными продуктами.

Кроме синтезатора Odin_II, широко распространен программный инструмент с открытым кодом Yosys (Open SYnthesis Suite, разработчик Clifford Wolf). Yosys версии 0.7+194 способен отображать Verilog-проекты в базис индустриальных ПЛИС Intel FPGA (Altera) серии Cyclone IV, MAX 10, ПЛИС Xilinx Spartan-6, Virtex-7, а также Lattice. Для логической оптимизации и отображения в базис ПЛИС в Yosys интегрирован программный инструмент ABC. Связь между синтезатором Yosys и синтезатором САПР Quartus Prime осуществляется посредством VQM-файлов, а для связи с Xilinx ISE используется edif-файл.

Рассмотрим типовой маршрут проектирования академических ПЛИС, который предполагает использование следующих программных инструментов: ODIN-II, ABC и VPR (рис. 1.51). ODIN_II конвертирует схемное описание некоторого сложно-функционального устройства на языке Verilog

HDL в специальный файл в blif (Berkeley Logic Interchange Format)-формате, в котором выделяет логические вентили для описания логики устройства и «черные ящики» для гетерогенных блоков, таких как умножители, блоки памяти, сумматоры и др. Синтаксический анализатор ODIN-II разработан на основе программных средств bison (лексический анализатор) и flex (синтаксический анализатор). По Verilog-коду синтаксический анализатор ODIN-II строит абстрактное синтаксическое дерево и перерабатывает его в плоский список цепей.

С использованием программного инструмента ABC проводится логическая оптимизация схемы и ее размещение в логические блоки академической ПЛИС (технологическое отображение). Выходным также является файл в blif-формате, в котором выделяются LUT-таблицы, D-триггеры логических блоков и гетерогенные блоки.

Blif-формат поддерживают не только синтезаторы verilog-кода, такие как SIS (система логического синтеза), Odin-II, Yosys, ABC, VPR, но и индустриальный САПР ПЛИС Altera Quartus с помощью программного модуля QUIP (University Interface Program).

VPR (является ядром VTR) размещает кластеры из конфигурационных логических блоков (КЛБ) и гетерогенные блоки (умножители, блоки ОЗУ и др.) на кристалле ПЛИС и организует глобальные и локальные трассировочные ресурсы для меж- и внутрикластерой связи логических блоков наиболее оптимальным образом, с учетом требований, например, к минимальной ширине трассировочного канала, быстродействию, экономии площади кристалла и др. Для размещения КЛБ на кристалле ПЛИС применяется алгоритм «имитации отжига». Процесс «имитации отжига» может быть представлен на основе четырех ключевых компонентов: представления состояния текущего решения, набора перемещений из одного состояния в другое, целевой функции стоимости для оценки каждого состояния и «схемы охлаждения», определяющей, как можно перейти от начального поиска к локальной опти-

мизации. Разводка электрических связей между кластерами осуществляется на основе модификаций алгоритма PathFinder с учетом задержек распространения сигналов в трассировочных ресурсах ПЛИС.

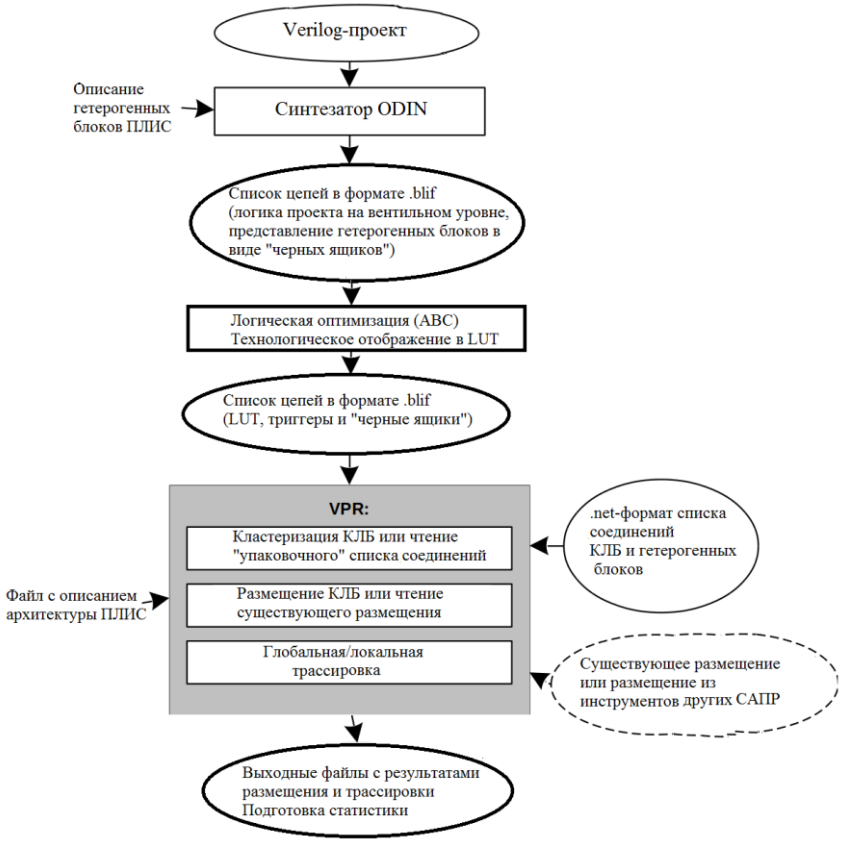


Рис. 1.51. Маршрут проектирования ПЛИС в VTR8.0

Для работы программ Odin_II и VPR требуется конфигурационный файл для описания архитектуры ПЛИС. Будем использовать архитектурный файл (определяет заданный базис ПЛИС) для описания ПЛИС (файл

k6_frac_N10_mem32K_40nm.xml находится в папке vtr_flow/arch/timing). Предполагается, что ПЛИС будет изготовлена по 40 нм технологическому КМОП-процессу с типовыми задержками в межсоединениях, вентилях и ключах. Флагманский архитектурный файл позволяет получать академические ПЛИС, близкие по техническим характеристикам (задержки в КЛБ, в маршрутизаторах трассировочных каналов, в умножителях и блоках памяти) к ПЛИС Altera серии Stratix IV GX (EP4SGX230DF29C2X).

КЛБ академической ПЛИС состоит из 10 адаптивных логических элементов (N = 10). Логический элемент (ЛЭ) имеет 6-входовую таблицу перекодировок (fracturable, 6-входовая LUTs или адаптивный LUT). Адаптивный LUT может быть сконфигурирован как чисто 6-входовой LUT или как два 5-входовых с пятью общими входами. LUT КЛБ академической ПЛИС по аналогии с ПЛИС Altera — это адаптивный логический модуль, но только режимов конфигурации два, а не семь, как у Stratix IV. Какие-либо цепи переноса на выходах LUT в КЛБ не используются. Объем встроенного блока двухпортовой памяти — 32 K, ЦОС-блок может быть сконфигурирован как умножитель с размером операндов 36×36 или как два 18×18, каждый из которых может быть сконфигурирован как два 9×9.

Конфигурационный файл состоит из двух частей, непосредственно для программы Odin_II (помечен <!-- ODIN II specific config--> и <!-- ODIN II specific config ends -->) и для программы VPR.

Для проверки правильности установки VTR 8.0 необходимо запустить скрипт:

```
$VTR_ROOT/vtr_flow/scripts/run_vtr_task.pl basic_flow,
```

результатом работы которого является следующая запись в командной строке:

```
k6_N10_memSize16384_memData64_40nm_timing/ch_intrinsics...OK.
```

Рассмотрим запуск VTR 8.0 в автоматическом режиме с помощью скрипта run_vtr_flow.pl. Пример запуска скрипта в консоли показан ниже.

```
$VTR_ROOT/vtr_flow/scripts/run_vtr_flow.pl <circuit_file> <architecture_file>
```

Для запуска скрипта нужно указать два аргумента:

• <имя Verilog-проекта>, например, выбирается любой Verilog-файл из тестов производительности VTR7.0;

• <имя архитектурного файла ПЛИС>.

Предварительно два файла — sha.v и k6_frac_N10_mem32K_40nm.xml — необходимо поместить в папку scripts. Для запуска скрипта в командной строке набираем следующее (рис. 1.52):

```
tricut@tricut-R59P-R60P-R61P ~/vtr8_xxx/vtr_flow/scripts $
tricut@tricut-R59P-R60P-R61P ~/vtr8_xxx/vtr_flow/scripts $ ./run_vtr_flow.pl sha.v k6_frac_N10_mem32K_40nm.xml
k6_frac_N10_mem32K_40nm/sha...OK
```

Рис. 1.52. Пример запуска скрипта run_vtr_flow.pl

Рассмотрим запуск Verilog-проектов в ручном режиме, без использования встроенных умножителей и блоков памяти (Verilog-проекты реализуются в логические ресурсы ПЛИС). Для запуска САПР VTR8.0 воспользуемся скрипт-файлом run_vtr_flow.pl, который находится в папке vtr_flow/scripts. Из тестов производительности VTR 7.0 выберем проект sha.v (алгоритм шифрования SHA-160). Синтезатор Odin II преобразует Verilog-файл проекта с именем sha.v в файл в blif-формате с именем sha_odin.blif. В папке ODIN_II с помощью текстового редактора редактируем скрипт go.sh. Содержание скрипта:

```
#!/bin/bash
./odin_II -V sha.v -o sha_odin.blif
```

Отчет о работе программы Verilog-синтеза Odin II для проекта sha.v показан на рис. 1.53. С помощью программного инструмента ABC создается файл sha_abc.blif (рис. 1.54).

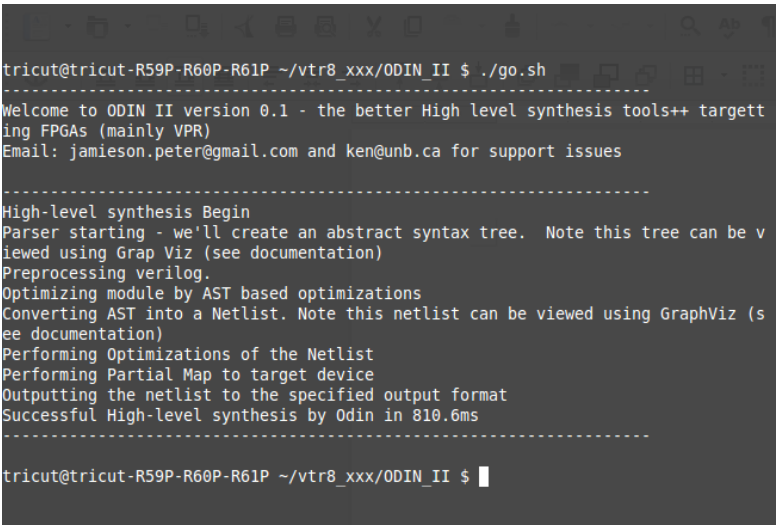


Рис. 1.53. Отчет о работе программы Verilog-синтеза Odin II для проекта sha.v

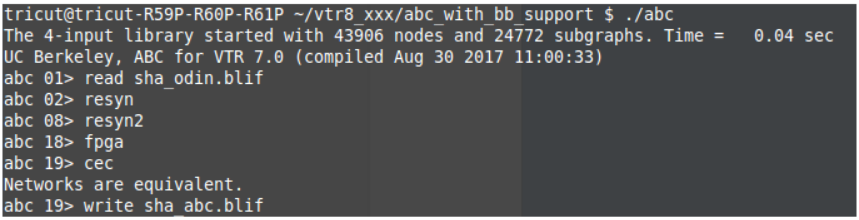


Рис. 1.54. Набор команд для программы логической оптимизации abc для проекта sha.v

Для запуска VPR создаем файл go.sh:

```
#!/bin/sh
./vpr k6_frac_N10_mem32K_40nm.xml sha_abc.blif
```

На рис. 1.55 представлен отчет работы программы VPR. Проект реализуется в логические ресурсы ПЛИС. Умножители и блоки памяти не используются. Всего 266 КЛБ, из них задействуются 244, 6 входовых LUT: 3 110, триггеров 911. На

рис. 1.56 показано первичное размещение проекта в логические ресурсы ПЛИС с архитектурным файлом k6_frac_N10_mem32K_40nm.xml. Задействованные КЛБ и элементы ввода/вывода подсвечены серым цветом. Первоначальная ширина трассировочного канала составляет 100 межсоединений. На рис. 1.57 показано расположение КЛБ на кристалле после размещения и трассировки в ПЛИС. Определена оптимальная ширина трассировочного канала в 52 межсоединения.

```
Packing took 7.83701 seconds
Begin loading packed FPGA netlist file.
Netlist generated from file 'sha_abc.net'.
n4589 is a constant generator.
Finished loading packed FPGA netlist file (took 0.407691 seconds).

Netlist num_nets: 1587
Netlist num_blocks: 318
Netlist EMPTY blocks: 0.
Netlist clb blocks: 244.
Netlist mult_36 blocks: 0.
Netlist memory blocks: 0.
Netlist inputs pins: 38
Netlist output pins: 36

FPGA sized to 21 x 21 (auto)

Resource usage...
        Netlist      0  blocks of type: EMPTY
        Architecture 0  blocks of type: EMPTY
        Netlist      74 blocks of type: io
        Architecture 608        blocks of type: io
        Netlist      244        blocks of type: clb
        Architecture 266        blocks of type: clb
        Netlist      0  blocks of type: mult_36
        Architecture 8  blocks of type: mult_36
        Netlist      0  blocks of type: memory
        Architecture 9  blocks of type: memory
```

Рис. 1.55. Отчет работы программы VPR для проекта sha.v

На рис. 1.58 показан фрагмент ПЛИС с трассировочными ресурсами. Подключение входов/выходов КЛБ к трассировочным каналам осуществляется с четырех сторон.

Трассировочные каналы сегментируются маршрутизаторами типа Wilton с коэффициентом разветвления Fs = 3, коэффициенты разветвления по входам и выходам соединительных блоков Fc_in = 0,15 и Fc_in = 0,1.

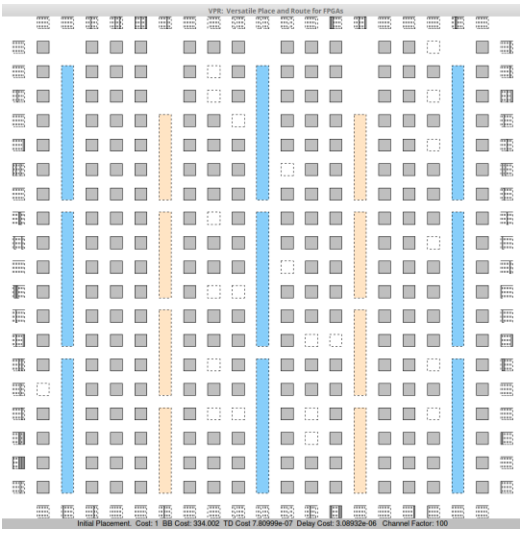


Рис. 1.56. Первичное размещение проекта sha.v
в логические ресурсы ПЛИС с архитектурным файлом
k6_frac_N10_mem32K_40nm.xml

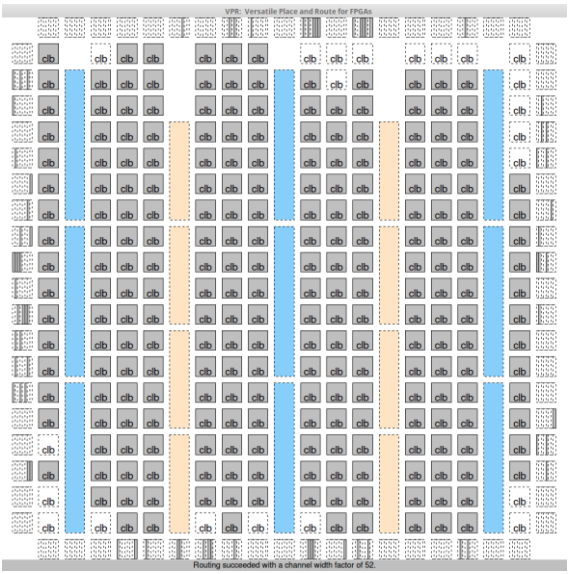


Рис. 1.57. После размещения и трассировки проекта sha.v в ПЛИС.
Определена ширина трассировочного канала — 52 межсоединения

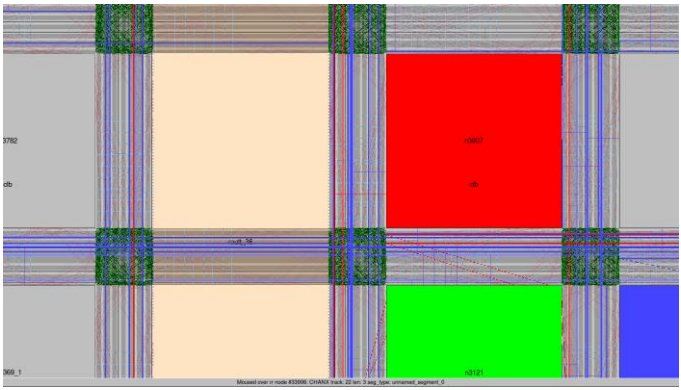


Рис. 1.58. Трассировочные ресурсы ПЛИС.
Задействованный КЛБ с номером n3121 подсвечен зеленым цветом.
Соседи, связанные с ним, обозначены синим и зеленым

Сегментированная трассировочная структура реализуется с использованием однонаправленных межсоединений (unidirectional). В нашем случае имеем 52 межсоединения, равномерно распределенных по всей площади кристалла. Длина сегментации межсоединений в трассировочных каналах L = 4, т. е. межсоединение проходит непрерывно 4 КЛБ. Также показаны соединительные блоки для подключения входов/выходов КЛБ к каналам. Маршрутизаторы и соединительные блоки реализуются на мультиплексорных структурах.

Рассмотрим запуск Verilog-проектов в ручном режиме, с использованием встроенных умножителей и блоков памяти. В папку Odin II необходимо поместить файл проекта умножителя двух 16-разрядных чисел mult.v, конфигурационный файл odin_config.xml и архитектурный файл k6_frac_N10_mem32K_40nm.xml.

В конфигурационном файле необходимо указать файл проекта, выходной файл в формате blif (mult.odin.blif) и имя архитектурного файла.

Код умножителя показывает пример 1. Пример 2 демонстрирует файл конфигурации odin_config.xml для синтезатора Odin_II.

```
module mult (a,b,y);
input [15:0] a;
input [15:0] b;
output [32:0] y;
assign y = a*b;
endmodule
```

Пример 1. Verilog-код умножителя
с размерностью операндов 16×16

```
<config>
<verilog_files>
        <!-- Way of specifying multiple files in a project -->
        <verilog_file>mult.v</verilog_file>
        </verilog_files>
        <output>
        <!-- These are the output flags for the project -->
        <output_type>blif</output_type>
<output_path_and_name>mult.odin.blif</output_path_and_name>
        <target>
        <!-- This is the target device the output is being built for -->
        <arch_file>k6_frac_N10_mem32K_40nm.xml</arch_file>
        </target>
        </output>
        <optimizations>
        <multiply size="3" fixed="1" fracture="0" padding="-1"/>
        <memory split_memory_width="1" split_memory_depth=""/>
        <adder size="0" threshold_size="1"/>
        </optimizations>
        <debug_outputs>
        <!-- Various debug options -->
        <debug_output_path>.</debug_output_path>
        <output_ast_graphs>1</output_ast_graphs>
        <output_netlist_graphs>1</output_netlist_graphs>
        </debug_outputs>
</config>
```

Пример 2. Файл конфигурации odin_config.xml
для синтезатора Odin_II

В папке ODIN_II с помощью текстового редактора редактируем скрипт go.sh. Содержание скрипта:

```
#!/bin/bash
./odin_II -c odin_config.xml -V mult.v -o mult_odin.blif
```

Из отчета видно (рис. 1.59), что проект реализуется не на логических ресурсах ПЛИС, а на аппаратном умножителе. Выделяется один аппаратный умножитель с размерностью операндов 16×16. На рис. 1.60 показан набор команд для технологического отображения в базис 6-входовых LUT (команда if –K 6). С предупреждением, что будет создан 1 черный ящик. В результате работы программы логической оптимизации создается файл mult.abc.blif. На рис. 1.61 представлен отчет работы программы VPR, а на рис. 1.62 реализация проекта в базис ПЛИС с использованием аппаратного умножителя.

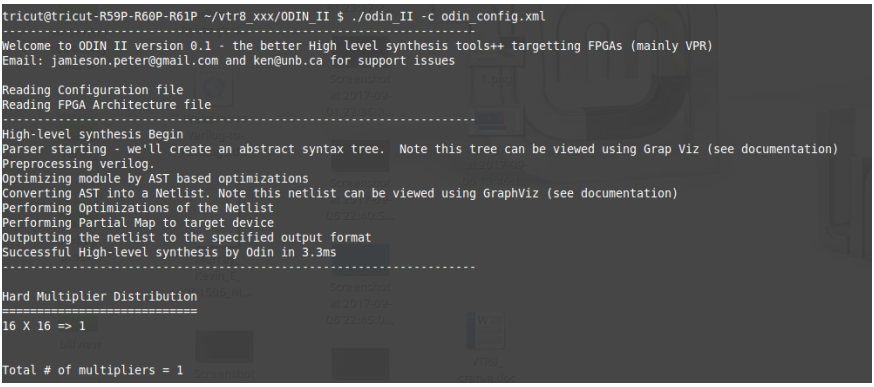


Рис. 1.59. Отчет о работе программы Verilog-синтеза Odin II для проекта mult.v

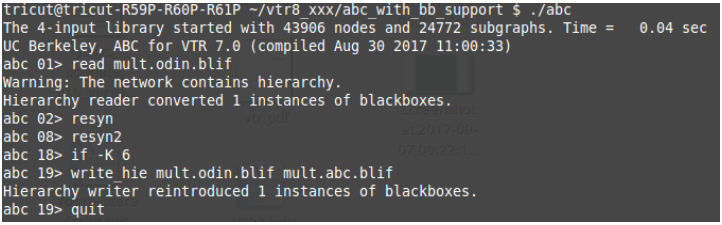


Рис. 1.60. Набор команд для логической оптимизации и технологического отображения в базис 6-входовых LUT для проекта mult.v

Пример 3 показывает фрагмент файла mult.abc.blif, созданного программой ABC.

```
.inputs top^a~0 top^a~1 top^a~2 top^a~3 top^a~4 top^a~5 top^a~6 top^a~7 \
 top^a~8 top^a~9 top^a~10 top^a~11 top^a~12 top^a~13 top^a~14 top^a~15 \
 top^b~0 top^b~1 top^b~2 top^b~3 top^b~4 top^b~5 top^b~6 top^b~7 top^b~8 \
 top^b~9 top^b~10 top^b~11 top^b~12 top^b~13 top^b~14 top^b~15
.outputs top^y~0 top^y~1 top^y~2 top^y~3 top^y~4 top^y~5 top^y~6 top^y~7 \
 top^y~8 top^y~9 top^y~10 top^y~11 top^y~12 top^y~13 top^y~14 top^y~15 \
 top^y~16 top^y~17 top^y~18 top^y~19 top^y~20 top^y~21 top^y~22 top^y~23 \
 top^y~24 top^y~25 top^y~26 top^y~27 top^y~28 top^y~29 top^y~30 top^y~31 \
 top^y~32
.subckt multiply a[0]=top^a~0 a[1]=top^a~1 a[2]=top^a~2 a[3]=top^a~3 a[4]=top^a~4 \
a[5]=top^a~5 a[6]=top^a~6 a[7]=top^a~7 a[8]=top^a~8 a[9]=top^a~9 a[10]=top^a~10 \
a[11]=top^a~11 a[12]=top^a~12 a[13]=top^a~13 a[14]=top^a~14 a[15]=top^a~15
a[16]=unconn \
….
```

Пример 3. Фрагмент файла mult.abc.blif

```
Netlist num_nets: 65
Netlist num_blocks: 67
Netlist EMPTY blocks: 0.
Netlist clb blocks: 1.
Netlist mult_36 blocks: 1.
Netlist memory blocks: 0.
Netlist inputs pins: 32
Netlist output pins: 33

FPGA sized to 8 x 8 (auto)

Resource usage...
        Netlist      0  blocks of type: EMPTY
        Architecture 0  blocks of type: EMPTY
        Netlist      65 blocks of type: io
        Architecture 192       blocks of type: io
        Netlist      1  blocks of type: clb
        Architecture 24 blocks of type: clb
        Netlist      1  blocks of type: mult_36
        Architecture 1  blocks of type: mult_36
        Netlist      0  blocks of type: memory
        Architecture 1  blocks of type: memory
```

Рис. 1.61. Отчет работы программы VPR для проекта mult.v

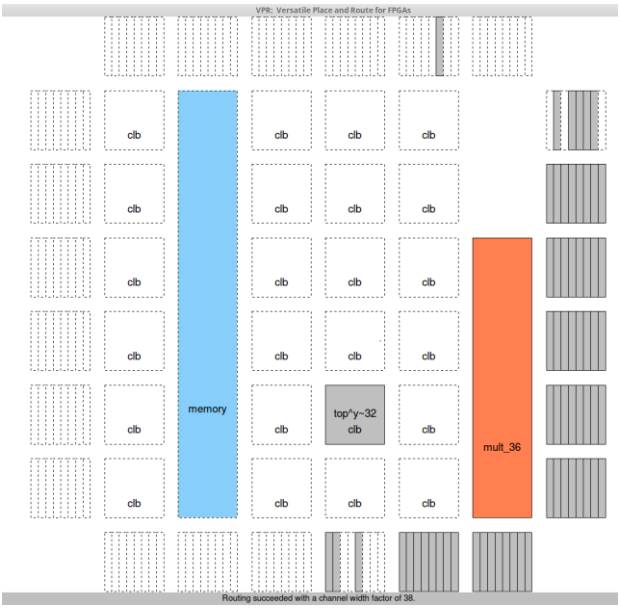


Рис. 1.62. ПЛИС с архитектурным файлом k6_frac_N10_mem32K_40nm.xml. Для реализации проекта mult.v задействуются один КЛБ и один аппаратный умножитель (подсвечен оранжевым цветом)

Рассмотрим реализацию Verilog-проектов в базисе ПЛИС в многозадачном режиме. В папке vtr_flow/tasks/timing/confing/ находится файл config.txt с заданием для эксперимента. Выбираем произвольно следующие Verilog-проекты: diffeq1.v, raygentop.v, sha.v, stereovision0.v и реализуем их в базис ПЛИС с архитектурой, определяемой файлом k6_frac_N10_mem32K_40nm.xml.

Неиспользуемые проекты из тестов производительности (бенч марк САПР VTR 7.0, некоторые из Verilog-проектов задействуют свыше 10 тыс. 6-входовых LUT) должны быть закомментированы символом # (рис. 1.63). Заходим в папку vtr_flow/scripts и запускаем в командной строке скрипт run_vtr_task.pl с указанием эксперимента (timing):

```
./run_vtr_task.pl timing
```

В консоли выводится статус реализации проектов в базис ПЛИС (рис. 1.64.), а в папке task/timing/runxxx/k6_frac_N10_mem32K_40nm.xml создаются подпапки diffeq1.v, raygentop.v, sha.v, stereovision.v со статистической информацией о работе программы VTR (рис. 1.65.).

Для получения отчета в виде таблицы необходимо в консоли запустить скрипт:

```
./parse_vtr_task.pl timing -create_golden
```

В результате его работы создается файл golden_result.txt, который можно найти в папке timing/config. В частности, в нем приводятся сведения по ширине трассировочного канала ПЛИС (min_chan_width), критической задержке в наихудшем пути, значения времени, затраченного на упаковку и размещение проекта в базис ПЛИС, и другая важная информация (рис. 1.66).

```
# Path to directory of circuits to use
circuits_dir=benchmarks/verilog

# Path to directory of architectures to use
archs_dir=arch/timing

# Add circuits to list to sweep
#circuit_list_add=arm_core.v
#circuit_list_add=bgm.v
#circuit_list_add=blob_merge.v
#circuit_list_add=boundtop.v
#circuit_list_add=ch_intrinsics.v
circuit_list_add=diffeq1.v
#circuit_list_add=diffeq2.v
#circuit_list_add=LU8PEEng.v
#circuit_list_add=LU32PEEng.v
#circuit_list_add=LU64PEEng.v
#circuit_list_add=mcml.v
#circuit_list_add=mkDelayWorker32B.v
#circuit_list_add=mkPktMerge.v
#circuit_list_add=mkSMAdapter4B.v
#circuit_list_add=or1200.v
circuit_list_add=raygentop.v
circuit_list_add=sha.v
circuit_list_add=stereovision0.v
#circuit_list_add=stereovision1.v
#circuit_list_add=stereovision2.v
#circuit_list_add=stereovision3.v

# Add architectures to list to sweep
arch_list_add=k6_frac_N10_mem32K_40nm.xml
```

Рис. 1.63. Тесты производительности VTR 7.0 (Verilog-проекты)

```
tricut@tricut-R59P-R60P-R61P ~/vtr8_xxx/vtr_flow/scripts $ ./run_vtr_task.pl timing

timing
----------------------------------------
Current time: сент.-06 01:36 .  Expected runtime of next benchmark: Unknown
k6_frac_N10_mem32K_40nm/diffeq1...OK
Current time: сент.-06 01:36 .  Expected runtime of next benchmark: Unknown
k6_frac_N10_mem32K_40nm/raygentop...
k6_frac_N10_mem32K_40nm/stereovision0...OK

tricut@tricut-R59P-R60P-R61P ~/vtr8_xxx/vtr_flow/scripts $
```

Рис. 1.64. Сообщения в консоли об успешной реализации проектов diffeq1, raygentop и stereovision0 в заданный базис ПЛИС

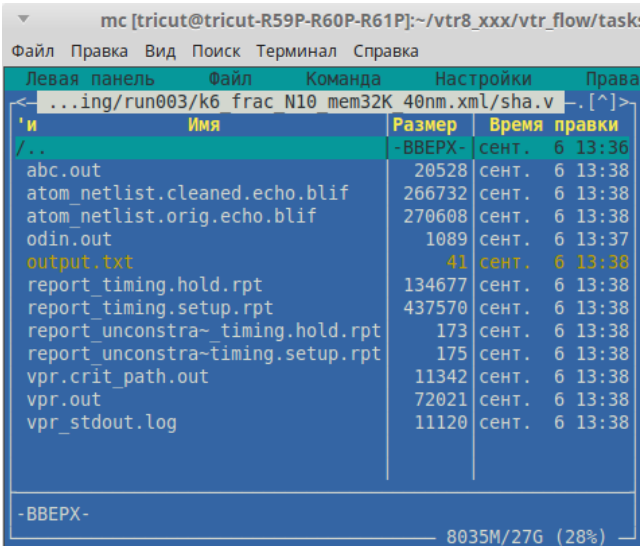


Рис. 1.65. Отчеты о работе трех программ: Odin, ABC и VPR для проекта sha.v

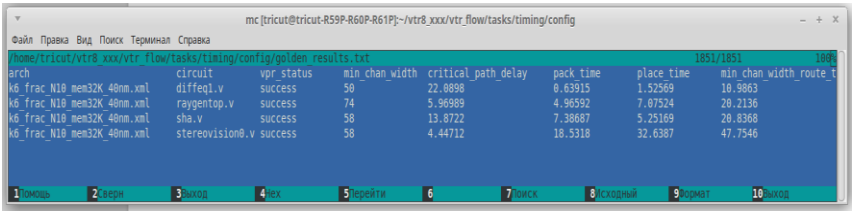


Рис. 1.66. Содержимое файла golden_result.txt с оценочными значениями

В данном разделе показано, что САПР VTR 8.0 позволяет обрабатывать крупные Verilog-проекты с умножителями и блоками памяти, создавать и исследовать новые архитектуры академических ПЛИС, по своим техническим характеристикам близкие к индустриальным ПЛИС Intel (Altera) и Xilinx с 6 входовыми LUT с адаптивными логическими блоками.

1.5. Синтез Verilog-проектов с помощью Altera QIS и их реализация в базис академических ПЛИС с помощью САПР VTR 7

Рассматриваются Verilog-проекты, синтезированные в базис ПЛИС с помощью индустриального САПР Altera Quartus II, и их последующая реализация в базисе академической ПЛИС в САПР VTR 7.0, по своей архитектуре схожей с ПЛИС Altera серии Stratix IV.

Реализуем простейший проект (сумматор, пример 1) в ПЛИС Stratix IV GX (EP4SGX230DF29C2X) с помощью САПР Quartus II. Программный инструмент синтеза HDL-проектов (Quartus Integrated Synthesis, QIS) Altera САПР Quartus II с помощью программного модуля Quartus II University Interface Program (QUIP) поддерживает выгрузку VHDL, Verilog-проектов в формате blif (Berkeley Logic Interchange Format) для связи с академическими программными инструментами с открытым исходным кодом. Blif-формат поддерживают не только синтезаторы verilog-кода, такие как SIS (система логического синтеза), Odin-II, Yosys, программа логической оптимизации, например, ABC, но и программы для кластеризации булевых функций после технологического отображения, размещения и трассировки в базис ПЛИС, например, программный инструмент VPR академического САПР VTR 7.0 (рис. 1.67).

В blif-формате выделяются логические вентили для описания логики устройства и «черные ящики» для гетерогенных блоков, таких как умножители, блоки памяти и др. Фактиче-

ски blif-формат представляет собой технологически независимый нетлист на вентильном уровне. Blif-формат с помощью QUIP может быть извлечен до технологически независимой многоуровневой логической оптимизации (multi-level logic optimization, MLS), так и после полной оптимизации и технологического отображения в k-LUT. В простейшем случае, без принятия специальных мер, QUIP не выгружает в blif-формат, т. е. не преобразует в вентили (комбинационные логические элементы) черные ящики (сумматоры, блоки умножителей, блоки ОЗУ), арифметические цепи ускоренного переноса сумматоров, игнорирует асинхронные сигналы, не подсоединенные входы LUT подключаются к «нулю».

```
module adder (a,b,f);
input [1:0] a;
input [1:0] b;
output [2:0] f;
assign f=a+b;
endmodule
```

Пример 1. Verilog-код 2-разрядного сумматора

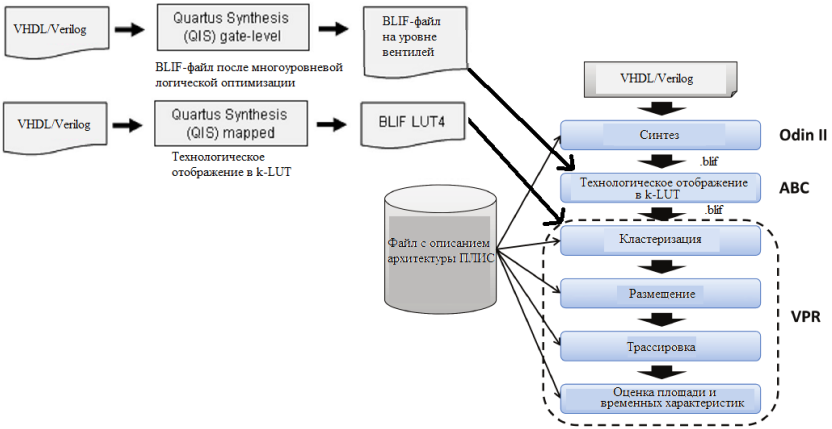


Рис. 1.67. Маршрут проектирования академических ПЛИС с использованием САПР VTR 7.0

Для выгрузки blif-файлов до технологически независимой многоуровневой логической оптимизации необходимо в файл настроек проекта (qsf-файл) с помощью TCL-команд добавить следующие команды (пример 2):

```
set_global_assignment -name INI_VARS
"no_add_ops=on;opt_dont_use_mac=on;dump_blif_before_optimize=on;abort_after_du
mping_blif = on"
set_global_assignment -name AUTO_SHIFT_REGISTER_RECOGNITION OFF
set_global_assignment -name DSP_BLOCK_BALANCING "LOGIC ELEMENTS"
set_global_assignment -name IGNORE_CARRY_BUFFERS ON
```

Пример 2. Настройки в qsf-файле проекта для извлечения blif-файла до технологически независимой многоуровневой логической оптимизации

Например, установка «no_add_ops» преобразует сумматоры в списке соединений в вентили. Установка «opt_dont_use_mac» запрещает использовать ЦОС-блоки (MAC-блоки ПЛИС). Для выгрузки blif-файлов после полной оптимизации необходимо в файл настроек проекта (qsf-файл) добавить следующие команды (пример 3):

```
set_global_assignment -name INI_VARS
"no_add_ops=on;opt_dont_use_mac=on;dump_blif_after_optimize=on;abort_after_dum
ping_blif = on"
set_global_assignment -name AUTO_SHIFT_REGISTER_RECOGNITION OFF
set_global_assignment -name DSP_BLOCK_BALANCING "LOGIC ELEMENTS"
set_global_assignment -name IGNORE_CARRY_BUFFERS ON
```

Пример 3. Настройки в qsf-файле проекта для извлечения blif-файла после полной оптимизации

Примеры 4 и 5 демонстрируют blif-файлы сумматора до технологически независимой многоуровневой логической оптимизации и после. Примеры показывают, что blif-файлы получились разные.

```
.model adder
.inputs a[0] b[0] a[1] b[1]
.outputs f[0] f[1] f[2]
#f[0] = Add0~synth
.names a[0] b[0] f[0]
10 1
01 1
#g2 = Add0~synth
.names a[0] b[0] g2
11 1
#g2 = Add0~synth
#f[1] = Add0~synth
.names g2 a[1] b[1] f[1]
100 1
010 1
001 1
111 1
#g2 = Add0~synth
#g4 = Add0~synth
.names g2 a[1] g4
11 1
#g2 = Add0~synth
#g5 = Add0~synth
.names g2 b[1] g5
11 1
#g6 = Add0~synth
.names a[1] b[1] g6
11 1
#g4 = Add0~synth
#g5 = Add0~synth
#g6 = Add0~synth
#f[2] = Add0~synth
.names g4 g5 g6 f[2]
000 0
.end
```

Пример 4. Blif-файл сумматора до технологически независимой многоуровневой логической оптимизации

```
.model adder
.inputs a[0] b[0] b[1] a[1]
.outputs f[0] f[1] f[2]
#f[0] = Add0~synth
.names a[0] b[0] f[0]
10 1
01 1
#g2 = Add0~synth
.names a[0] b[0] g2
11 1
#g3 = Add0~synth
.names a[1] b[1] g3
11 1
#g4 = Add0~synth
.names a[1] b[1] g4
00 0
#g2 = Add0~synth
#g4 = Add0~synth
#g5 = Add0~synth
.names g2 g4 g5
11 1
#g3 = Add0~synth
#g5 = Add0~synth
#f[2] = Add0~synth
.names g3 g5 f[2]
00 0
#g7 = Add0~synth
.names a[1] b[1] g7
10 1
01 1
#g2 = Add0~synth
#g7 = Add0~synth
#f[1] = Add0~synth
.names g2 g7 f[1]
10 1
01 1
.end
```

Пример 5. Blif-файл сумматора после полной логической оптимизации

Для выгрузки blif-файлов после технологического отображения в k-LUT ПЛИС необходимо в файл настроек проекта (qsf-файл) добавить следующие команды (пример 6):

```
set_global_assignment -name INI_VARS
"no_add_ops=on;opt_dont_use_mac=on;dump_blif_after_lut_map=on;abort_after_dump
ing_blif = on"
set_global_assignment -name AUTO_SHIFT_REGISTER_RECOGNITION OFF
set_global_assignment -name DSP_BLOCK_BALANCING "LOGIC ELEMENTS"
set_global_assignment -name IGNORE_CARRY_BUFFERS ON
```

Пример 6. Настройки в qsf-файле проекта для извлечения blif-файла после технологического отображения в k-LUT

В зависимости от используемой версии САПР Quartus II (рис. 1.68, 1.69) выдается или нет информация о создании blif-файла. Работа синтезатора приостанавливается согласно команде abort_after_dumping_blif = on. Для разных серий ПЛИС blif-файл после технологического отображения в k-LUT формируется одинаковый. Пример 7 демонстрирует blif-файл сумматора после технологического отображения в базис 4-входовых LUT.

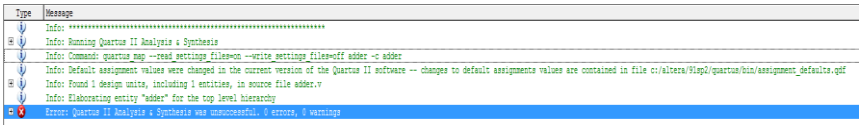


Рис. 1.68. После того как будет создан blif-файл в САПР Quartus II ver 9.1, выдается сообщение об ошибке на этапе анализа и синтеза, работа синтезатора приостанавливается

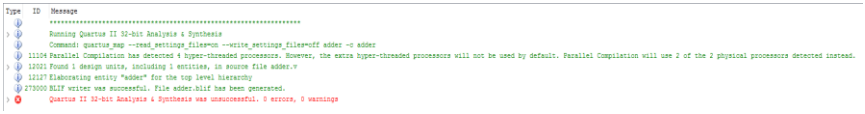


Рис. 1.69. После того как будет создан blif-файл в САПР Quartus II ver 13.1, выдаются сообщения о том, что файл adder.blif создан и об ошибке на этапе анализа и синтеза, работа синтезатора приостанавливается

Синтезатор QIS САПР Quartus II для некоторых серий ПЛИС позволяет выгружать RTL-проекты на «атомном» уровне с использованием сетевых примитивов и мегафункций (рис. 1.70). Пример демонстрирует VQM-файл сумматора при его реализации в базис ПЛИС серии Cyclone II. Рис. 1.70 показывает, что для реализации сумматора требуется три сетевых примитива cycloneii_lcell_comb (Lut в различных режимах). Данный формат может быть переведен на вентильный уровень (ресинтез) для последующей оптимизации с помощью индустриальных средств САПР, таких как, например, синтезатор Synplify Pro от Synopsys.

```
.model adder
.inputs a[0] b[0] a[1] b[1]
.outputs f[0] f[1] f[2]
#f[0] = Add0~synth
.names a[0] b[0] f[0]
10 1
01 1
#f[1] = Add0~synth
.names a[0] b[0] a[1] b[1] f[1]
1100 1
0010 1
1010 1
0110 1
0001 1
1001 1
0101 1
1111 1
#f[2] = Add0~synth
.names a[0] b[0] a[1] b[1] f[2]
1110 1
1101 1
0011 1
1011 1
0111 1
1111 1
.end
```

Пример 7. Blif-файл сумматора после технологического отображения в ПЛИС (в базис 4-входовых LUT)

```
module        adder (a,b,f);
input         [1:0] a;
input         [1:0] b;
output        [2:0] f;
wire \Add0~0 ;
wire \Add0~1 ;
wire \Add0~2 ;
wire gnd;
wire vcc;
assign gnd = 1'b0;
assign vcc = 1'b1;
cycloneii_lcell_comb \Add0~0_I (
.datac(a[0]),
.datad(b[0]),
.combout(\Add0~0 ));
defparam \Add0~0_I .sum_lutc_input = "datac";
defparam \Add0~0_I .lut_mask = "0FF0";
cycloneii_lcell_comb \Add0~1_I (
    .dataa(a[0]),
    .datab(b[0]),
    .datac(a[1]),
    .datad(b[1]),
    .combout(\Add0~1 ));
defparam \Add0~1_I .sum_lutc_input = "datac";
defparam \Add0~1_I .lut_mask = "8778";
cycloneii_lcell_comb \Add0~2_I (
    .dataa(a[1]),
    .datab(b[1]),
    .datac(a[0]),
    .datad(b[0]),
    .combout(\Add0~2 ));
defparam \Add0~2_I .sum_lutc_input = "datac";
defparam \Add0~2_I .lut_mask = "E888";
assign f[0] = \Add0~0 ;
assign f[1] = \Add0~1 ;
assign f[2] = \Add0~2 ;
endmodule
```

Пример 8. VQM-файл сумматора в базис ПЛИС серии Cyclone II

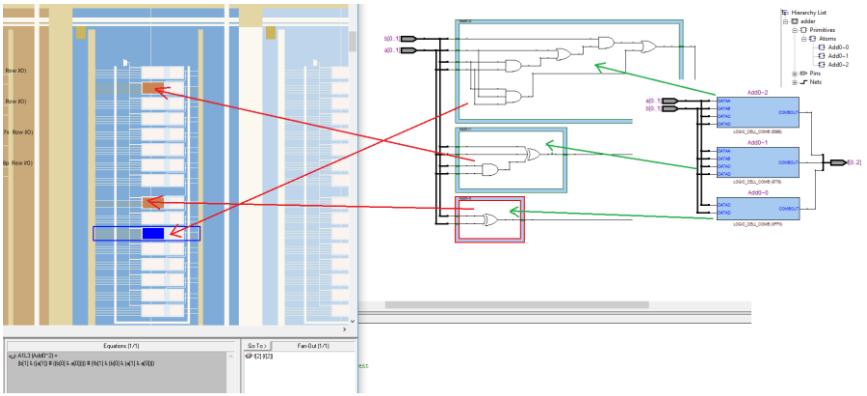


Рис. 1.70. Отображение в базис ПЛИС серии Cyclone II
с использованием сетевых примитивов cycloneii_lcell_comb
в САПР Quartus II ver 9.1

Извлечем blif-файл сумматора после технологического отображения в k-LUT с помощью QIS и реализуем проект в САПР VTR 7.0 (связка трех программных инструментов Odin II, ABC и VPR) на академической ПЛИС с архитектурным файлом k6_frac_N10_mem32K_40nm.xml (пример 9). Для работы программ Odin_II и VPR требуется конфигурационный файл для описания архитектуры ПЛИС.

На рис. 1.71 показано первичное размещение сумматора в базис академической ПЛИС с учетом кластеризации. Потребовался 1 КЛБ. На рис. 1.72 показан размещенный и разтрассированный проект сумматора в базис академической ПЛИС. Задержка в критическом пути составила 0,942472 нс, а максимальная частота, определенная по критическому пути, — 1 061,04 МГц. На рис. 1.73 показано раскрытие структуры КЛБ. Также виден периферийный трассировочный канал, элементы ввода/вывода. Для комбинационной схемы TimeQuest анализ не дает оценки рабочей частоты проекта в наихудшем случае (Fmax), а дает сведения по минимальной задержке распространения сигнала в пути входной порт — выходной (например, a[0]-f[0]) для определенных параметров.

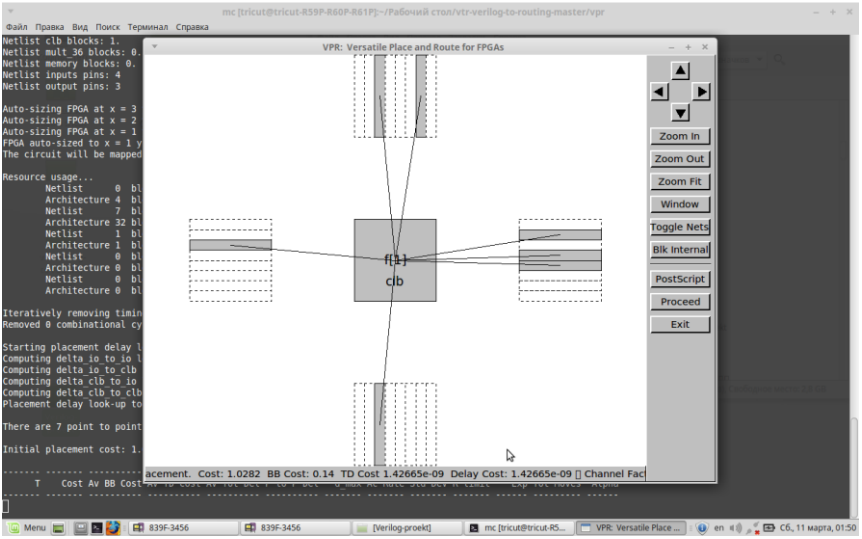


Рис. 1.71. Первичное размещение сумматора (blif-файла) в базис академической ПЛИС с архитектурным файлом k6_frac_N10_mem32K_40nm.xml

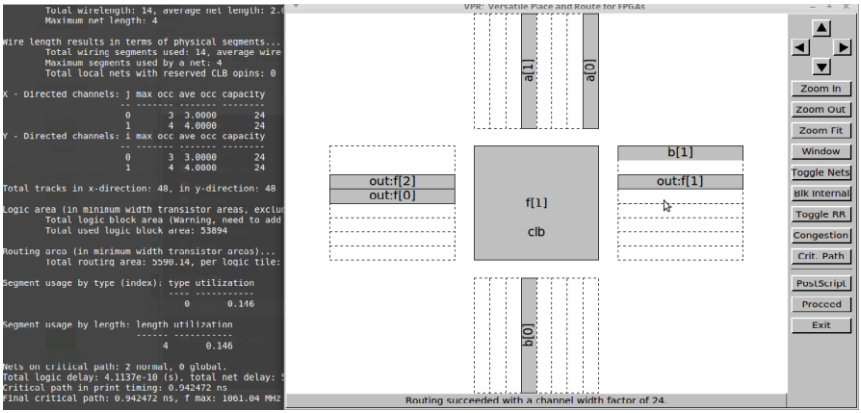


Рис. 1.72. Размещенный и разтрассированный проект сумматора в базис академической ПЛИС. Задержка в критическом пути составила 0,942472 нс, максимальная частота — 1061,04 МГц

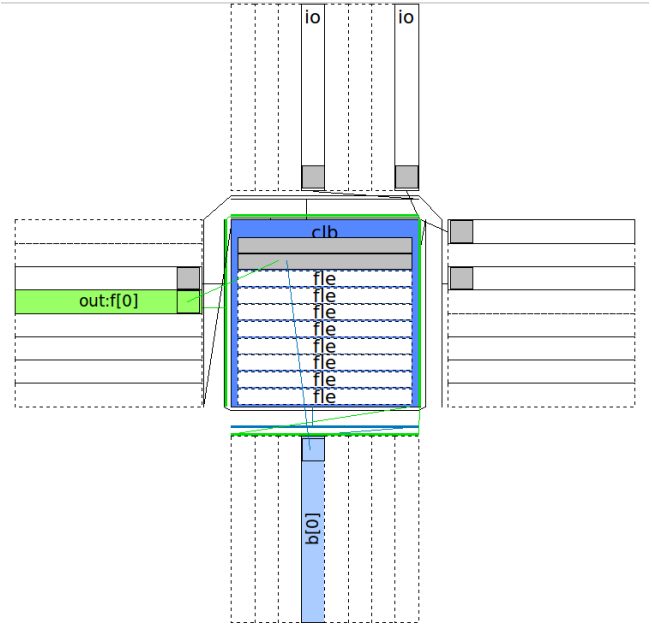


Рис. 1.73. Раскрытие структуры КЛБ.
Цветом выделены два критических пути

Реализуем Verilog-проект barrel16a (сдвиговый регистр на мультиплексорах) из тестового набора quip. На рис. 1.74 и рис. 1.75 показано первичное размещение проекта в базис ПЛИС и после полного размещения и трассировки.

Задержка в критическом пути 3,10578 нс, максимальная частота — 321,98 МГц. Для ПЛИС EP4SGX230DF29C2X рабочая частота в наихудшем случае Fmax для временной модели TimeQuest анализа Slow 990mV 85°C Model (напряжение питания ядра 900 мВ, температура 85°C) составит 584,11 МГц.

Рассмотрим синтез Verilog-проектов. Извлечем blif-файл из Verilog-кода сумматора с помощью синтезатора Odin II, проведем технологическое отображение с помощью ABC и реализуем проект в базисе академической ПЛИС с архитектурным файлом k6_frac_N10_mem32K_40nm.xml с применением VPR. Ниже показан скрипт для запуска синтезатора Odin II.

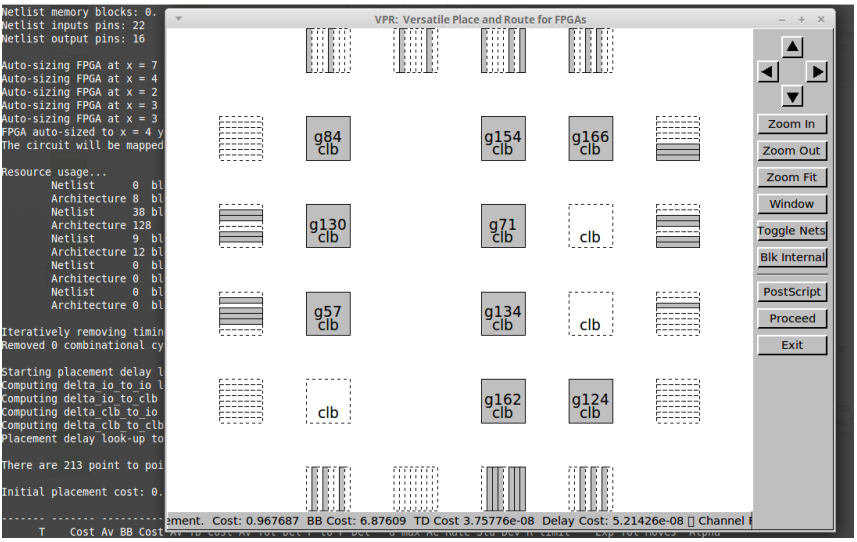


Рис. 1.74. Первичное размещение (blif-файл barrel16a)
в базис академической ПЛИС

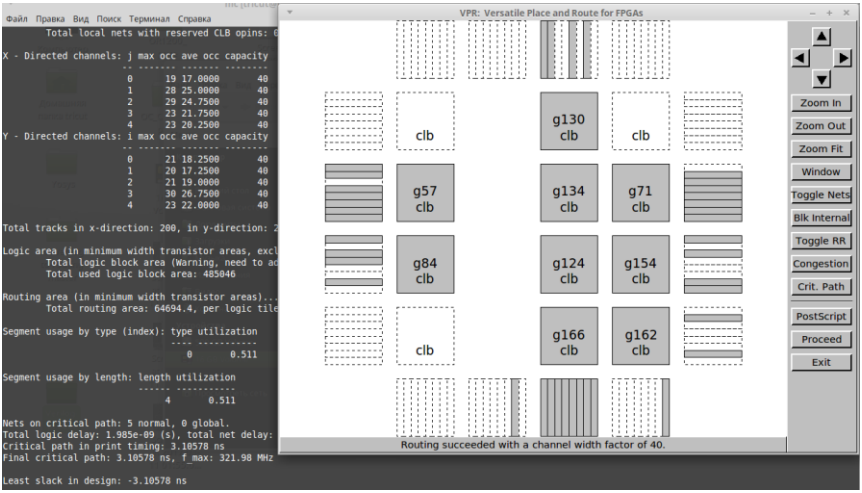


Рис. 1.75. После размещения и трассировки
в базис академической ПЛИС. Потребовалось 9 КЛБ

На рис. 1.76 демонстрируются этапы обработки Verilog-кода синтезатором: построение абстрактного синтаксического дерева (AST); препроцессинг; оптимизация AST; конвертация AST в «плоский» нетлист и его дальнейшая оптимизация; частичное отображение в базис ПЛИС. Рис. 1.77 показывает набор команд для технологического отображения в базис 4-входовых LUT. Программа читает файл adder_odin.blif (пример 9) и записывает результаты оптимизации в файл adder_adc.blif (пример 10).

Предъявим файл adder_adc.blif VPR для размещения в базис академической ПЛИС с тем же архитектурным файлом (рис. 1.78). Видим, что задержка в критическом пути увеличилась с 0,942472 нс (максимальная частота 1061,04 МГц) до 1,62359 нс (максимальная частота 615,918 МГц). Рис. 1.78 демонстрирует уже иное расположение элементов ввода/вывода по периферии кристалла, чем на рис. 1.72.

```
./odin_II.exe -V adder.v -o adder_odin.blif
```

```
Welcome to ODIN II version 0.1 - the better High level synthesis tools++ targetting FPGAs (mainly VPR)
Email: jamieson.peter@gmail.com and ken@unb.ca for support issues

------------------------------------------------------------
High-level synthesis Begin
Parser starting - we'll create an abstract syntax tree.  Note this tree can be viewed using GraphViz (see documentation)
Preprocessing verilog.
Optimizing module by AST based optimizations
Converting AST into a Netlist. Note this netlist can be viewed using GraphViz (see documentation)
Performing Optimizations of the Netlist
Performing Partial Map to target device
Outputting the netlist to the specified output format
Successful High-level synthesis by Odin in 3.1ms
------------------------------------------------------------
```

Рис. 1.76. Основные этапы работы синтезатора Odin II

```
The 4-input library started with 43906 nodes and 24772 subgraphs. Time =    0.12 sec
UC Berkeley, ABC for VTR 7.0 (compiled Mar 10 2017 04:32:12)
abc 01> ready adder_odin.blif
** cmd error: unknown command 'ready'
abc 01> read adder_odin.blif
abc 02> resyn
abc 08> resyn2
abc 18> fpga
abc 19> cec
Networks are equivalent.
abc 19> write adder_abc.blif
```

Рис. 1.77. Основные этапы работы программы оптимизации логии ABC

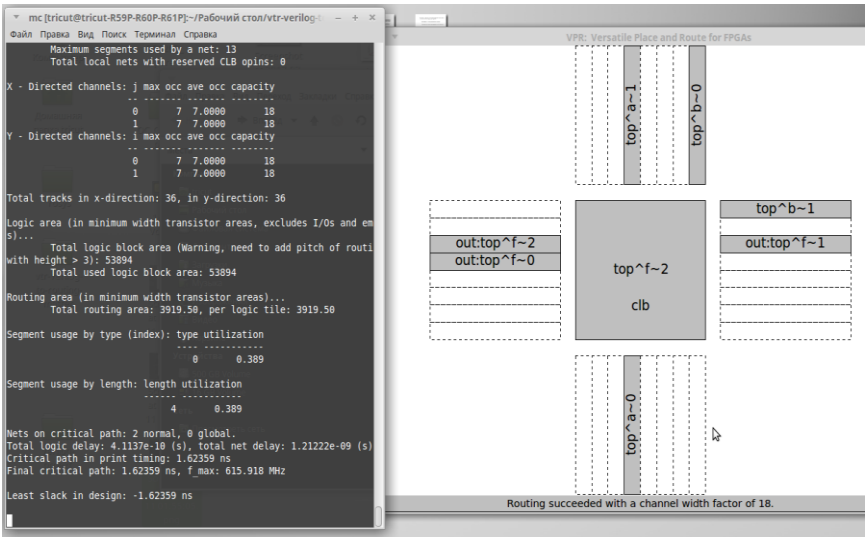


Рис. 1.78. После размещения и трассировки в базис академической ПЛИС (blif-файл получен совместной работой Odin II и ABC). Задержка в критическом пути 1,62359 нс, максимальная частота 615,918 МГц

```
.model adder
.inputs top^a~0 top^a~1 top^b~0 top^b~1
.outputs top^f~0 top^f~1 top^f~2
.names gnd
.names unconn
.names vcc
1
.names gnd top^a~0 top^b~0
top^ADD~0^ADDER_FUNC~4
001 1
010 1
100 1
111 1
.names gnd top^a~0 top^b~0
top^ADD~0^CARRY_FUNC~5
011 1
101 1
110 1
111 1
.names top^ADD~0^CARRY_FUNC~5 top^a~1
top^b~1 top^ADD~0^ADDER_FUNC~6
```

```
# Benchmark "adder" written by
ABC on Sat Mar 11 02:51:24 2017
.model adder
.inputs top^a~0 top^a~1 top^b~0
top^b~1
.outputs top^f~0 top^f~1 top^f~2
.names top^a~0 top^b~0 top^f~0
01 1
10 1
.names top^a~0 top^a~1 top^b~0
top^b~1 top^f~2
111- 1
1-11 1
-1-1 1
.names top^a~0 top^a~1 top^b~0
top^b~1 top^f~1
00-1 1
01-0 1
1010 1
1111 1
-001 1
```

```
001 1
010 1
100 1
111 1
.names top^ADD~0^CARRY_FUNC~5 top^a~1
top^b~1 top^ADD~0^CARRY_FUNC~7
011 1
101 1
110 1
111 1
.names top^ADD~0^CARRY_FUNC~7 gnd gnd
top^ADD~0^ADDER_FUNC~8
001 1
010 1
100 1
111 1
.names top^ADD~0^ADDER_FUNC~4 top^f~0
1 1
.names top^ADD~0^ADDER_FUNC~6 top^f~1
1 1
.names top^ADD~0^ADDER_FUNC~8 top^f~2
1 1
.end
```

Пример 9. Blif-файл сумматора, синтезированный Odin II

```
-100 1
.end
```

Пример 10. Blif-файл сумматора после технологического отображения в базис 4-входовых LUT с помощью ABC

Рассмотрим особенность реализации умножителей (пример 11) в виде модели «черного ящика». Для этого необходимо в qsf-файл добавить следующие команды (пример 12): opt_dont_use_mac=off; dump_blif_with_blackboxes=on. На рис. 1.79 показаны настройки по реализации аппаратных умножителей (опция DSP Block Balancing) не на логических элементах, а на ЦОС-блоках ПЛИС. Выбирается умножитель с размерностью операндов 18×18. Пример 13 показывает blif-файл умножителя после полной логической оптимизации, представленный в виде модели «черного ящика». Однако такой файл САПР VTR 7.0 «прочитать» не может, т. к. две подсхемы blackbox_g1 и blackbox_f[0] построены на мегафункции lpm_mult, которая в VPR не определена, однако может фрагменты Verilog-кода синтезировать в аппаратные умножители и блоки памяти академической ПЛИС. В качестве примера возьмем проект умножителя с

размерностью операндов 16×16 и сделаем прогон Odin II – ABC – VPR. Результат показан на рис. 1.80.

```
module mult (a,b,f);
input [3:0] a;
input [3:0] b;
output [7:0] f;
assign f=a*b;
endmodule
```

Пример 11. Умножитель двух четырехразрядных сигналов

```
set_global_assignment -name INI_VARS
"no_add_ops=on;opt_dont_use_mac=off;dump_blif_with_blackboxes=on;
dump_blif_before_optimize=on;abort_after_dumping_blif = on"
set_global_assignment -name AUTO_SHIFT_REGISTER_RECOGNITION OFF
set_global_assignment -name DSP_BLOCK_BALANCING "SIMPLE 18-BIT
MULTIPLIERS"
set_global_assignment -name IGNORE_CARRY_BUFFERS ON
```

Пример 12. Настройки в qsf-файле проекта для извлечения blif-файла после технологического отображения в черные ящики

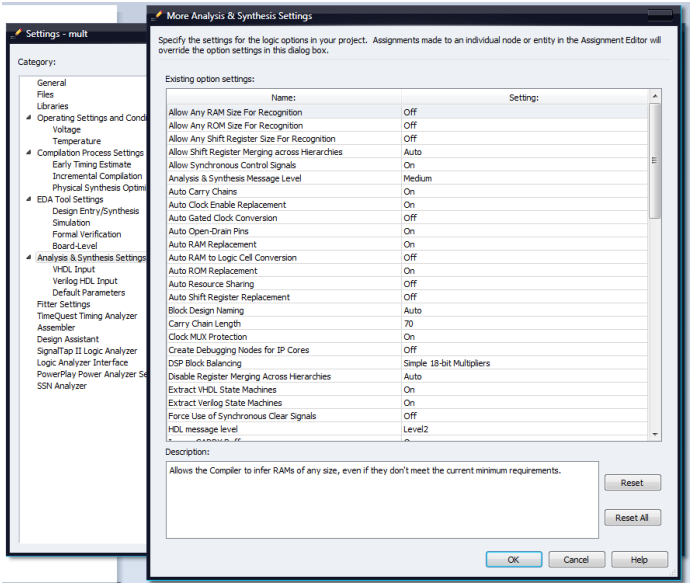


Рис. 1.79. Настройки по реализации аппаратных умножителей (DSP Block Balancing)

```
.model mult
.inputs a[3] a[2] a[1] a[0] b[3] b[2] b[1] b[0]
.outputs f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7]
.subckt blackbox_g1 a[0]=a[0] a[1]=a[1] a[2]=a[2] a[3]=a[3] b[0]=b[0] b[1]=b[1] b[2]=b[2]
b[3]=b[3]
g1=g1 g2=g2 g3=g3 g4=g4 g5=g5 g6=g6 g7=g7 g8=g8
.subckt blackbox_f[0] g1=g1 g2=g2 g3=g3 g4=g4 g5=g5 g6=g6 g7=g7 g8=g8 f[0]=f[0]
f[1]=f[1] f[2]=f[2]
f[3]=f[3] f[4]=f[4] f[5]=f[5] f[6]=f[6] f[7]=f[7]
.end
# blackbox_g1 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1
#g1 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1
#g2 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT1
#g3 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT2
#g4 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT3
#g5 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT4
#g6 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT5
#g7 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT6
#g8 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT7
.model blackbox_g1
.inputs a[0] a[1] a[2] a[3] b[0] b[1] b[2] b[3]
.outputs g1 g2 g3 g4 g5 g6 g7 g8
.blackbox
.end
# blackbox_f[0] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2
#g1 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1
#g2 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT1
#g3 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT2
#g4 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT3
#g5 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT4
#g6 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT5
#g7 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT6
#g8 = lpm_mult:Mult0|mult_71v:auto_generated|mac_mult1~DATAOUT7
#f[0] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2
#f[1] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2~DATAOUT1
#f[2] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2~DATAOUT2
#f[3] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2~DATAOUT3
#f[4] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2~DATAOUT4
#f[5] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2~DATAOUT5
#f[6] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2~DATAOUT6
#f[7] = lpm_mult:Mult0|mult_71v:auto_generated|mac_out2~DATAOUT7
.model blackbox_f[0]
.inputs g1 g2 g3 g4 g5 g6 g7 g8
.outputs f[0] f[1] f[2] f[3] f[4] f[5] f[6] f[7]
.blackbox
.end
```

Пример 13. Blif-файл умножителя после полной логической оптимизации, представленный в виде модели «черного ящика»

```
// DEFINES
`define BITS 16     // Bit width of the operands
`define B2TS 32     // Bit width of the operands
module    bm_base_multiply(clock, a_in, b_in,out0);
// SIGNAL DECLARATIONS
input       clock;
input [`BITS-1:0] a_in;
input [`BITS-1:0] b_in;
output [`B2TS-1:0] out0;
reg [`B2TS-1:0]  out0;
always @(posedge clock)
begin
            out0 <= a_in * b_in;
end
endmodule
```

Пример 14. Verilog-код умножителя
с размерностью операндов 16×16

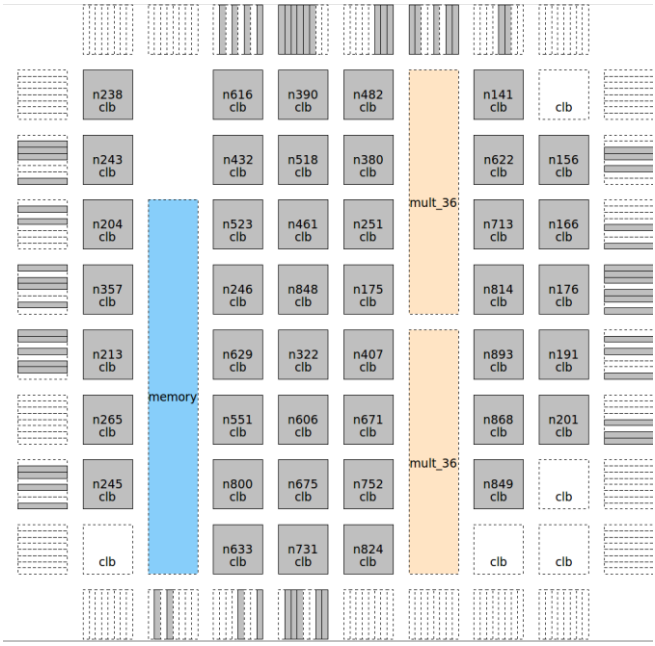


Рис. 1.80. Реализация Verilog-кода умножителя
с размерностью операндов 16×16 в логические ресурсы (clb),
в два аппаратных умножителя (mult_36) и блок памяти (memory)
академической ПЛИС

Таким образом, использование blif-файлов, созданных синтезатором QIS и выгружаемых из САПР Quartus II с помощью Quip, позволяет осуществить связь с академическими программными инструментами с открытым исходным кодом.

Максимальная частота проекта (сумматора), у которого использовался blif-файл, полученный после технологического отображения в базис 4-входовых LUT с помощью Altera QIS, снизилась с 1061,04 МГц до 615,918 МГц при использовании академического синтезатора Odin II совместно с ABC при реализации в базисе академической ПЛИС с архитектурным файлом k6_frac_N10_mem32K_40nm.xml, близкой по техническим характеристикам к ПЛИС Altera серии Stratix IV GX.

2. ПРОГРАММНЫЙ ИНСТРУМЕНТ YOSYS ДЛЯ VERILOG-СИНТЕЗА В БАЗИС БИС И ПЛИС

2.1. Программный инструмент Yosys для Verilog-синтеза в базис заказных БИС

Кроме Odin II, широко используется программный инструмент Yosys (разработчик Clifford Wolf) с открытым программным кодом для Verilog-синтеза (Yosys Open SYnthesis Suite (Yosys)). Yosys обрабатывает практически любой синтезируемый проект на языке Verilog-2005. В отличие от Odin II может синтезировать крупные Verilog-проекты с сайта Open Cores (например, микропроцессорные ядра or1200, msp430, k68 и др.).

Преобразует Verilog в форматы BLIF, EDIF, BTOR, SMT-LIB, упрощенный RTL Verilog. Содержит встроенные формальные методы проверки свойств и эквивалентности. Позволяет отображать Verilog-проекты в базис библиотечных ячеек заказных БИС (в формате Liberty) и в базис ПЛИС Xilinx седьмой серии и Lattice. Для логической оптимизации и отображения в базис заказных БИС и ПЛИС используется программный инструмент ABC. На рис. 2.1. показаны уровни проектирования цифровых БИС и ПЛИС и область использования Yosys. Выделяются следующие уровни: системный; алгоритмический уровень; поведенческий; уровень регистровых передач; уровень логических вентилей; физический (библиотеки логических элементов заказных БИС и LUT ПЛИС, включая триггеры); транзисторный (схемотехнический).

Ниже приводятся примеры скриптов для синтеза и последующего отображения в базис библиотеки заказных БИС и в базис ПЛИС Xilinx серии Virtex 7 на примере Verilog-проекта счетчика (рис. 2.2). По команде read_verilog counter осуществляется построение по Verilog-коду счетчика абстрактного синтаксического дерева с последующим RTL-представлением. По команде techmap крупнозернистые RTL-

ячейки (например, сумматоры или умножители) преобразуются в мелкозернистые ячейки (например, в вентили 2И-НЕ, 2ИЛИ-НЕ, триггеры и др.).

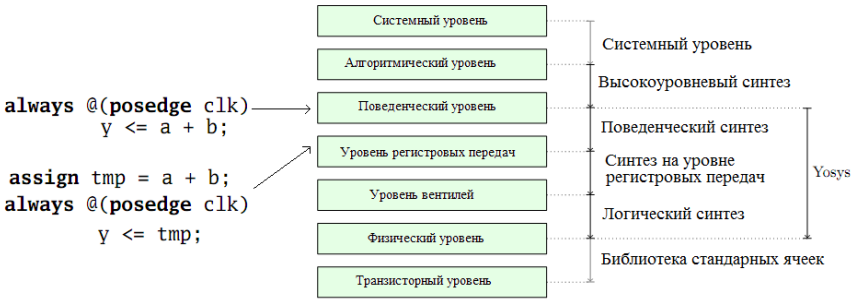


Рис. 2.1. Уровни абстракции и синтеза при проектировании цифровых БИС, а также область использования программного инструмента Yosys (три уровня синтеза)

```
# read design
read_verilog counter.v
# high-level synthesis
hierarchy -check -top counter
proc; opt; fsm; opt; techmap; opt
# mapping registers to ASIC cells
dfflibmap -liberty asic_cells.lib
# mapping logic to ASIC cells using Berkeley ABC
abc -liberty asic_cells.lib; opt
# write netlist
write_verilog asic_synth.v
```

а)

```
# read design
read_verilog counter.v
# high-level synthesis
hierarchy -check -top counter
proc; opt; fsm; opt; techmap; opt
# mapping logic to LUTs using Berkeley ABC
abc -lut 4; opt
# map internal cells to FPGA cells
techmap -map fpga_cells.v; opt
# write netlist
write_verilog fpga_synth.v
```

б)

Рис. 2.2. Примеры скриптов для синтеза и последующего отображения в базис библиотеки заказных БИС (*а*) и в базис ПЛИС Xilinx серии Virtex 7 (*б*) на примере Verilog-проекта 3-разрядного суммирующего счетчика

Далее мелкозернистые ячейки (их можно назвать еще и как софт-ячейки) в два приема отображаются в аппаратные комбинационные элементы и триггеры, соответствующие какой-либо библиотеке (например, asic_cells). В библиотеке asic_cells определены как вентили, так и триггеры. Для заказ-

ных БИС триггеры отображаются с помощью команды dfflibmap в формате liberty с использованием библиотеки asic_cells. Логика проекта отображается с помощью ABC и библиотеки asic_cells (рис. 2.2, а). Для ПЛИС логика отображается непосредственно в 4-входовые LUT (рис. 2.2, б).

На рис. 2.3.–2.6 показаны этапы преобразования Verilog-кода 3-разрядного суммирующего счетчика в мультиплексоры и триггеры с последующим представлением в базис комбинационных логических элементов и регистров библиотеки asic_cells.

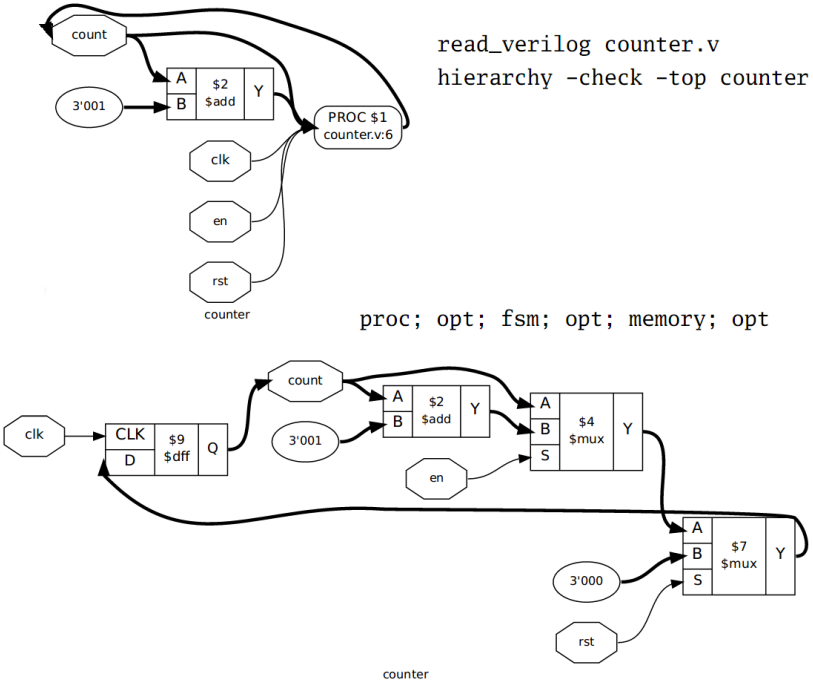


Рис. 2.3. *а* — RTL-представление проекта с помощью команды read_verilog hierarchy с последующим анализом (hierarchy); *б* — поведенческое представление проекта с использованием крупнозернистых RTL-ячеек с помощью команд proc и opt

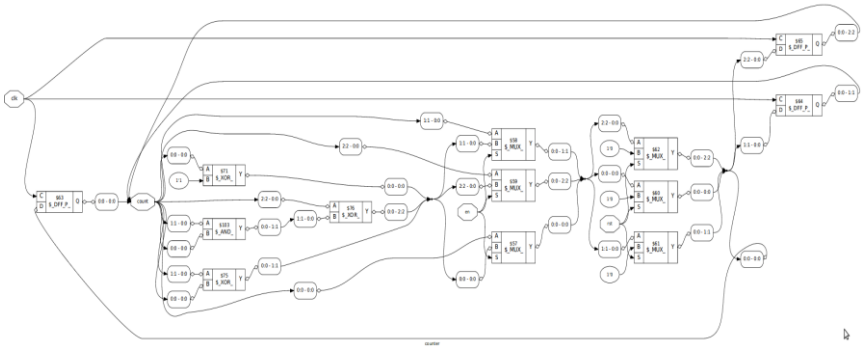


Рис. 2.4. Отображение крупнозернистых RTL-ячеек
в мелкозернистые с помощью команд techmap и opt

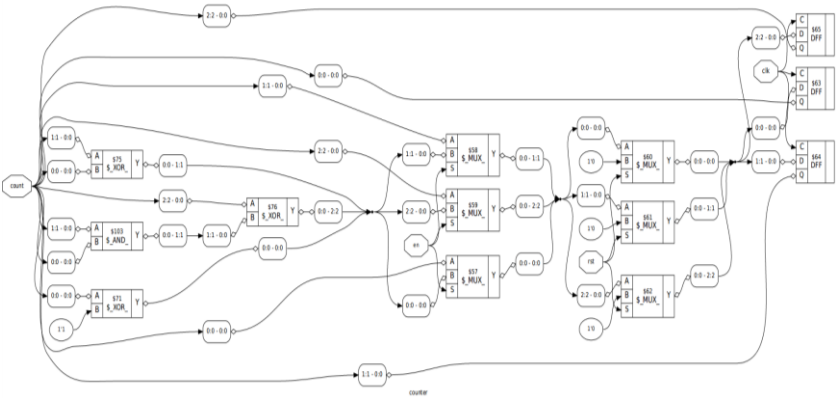


Рис. 2.5. Отображение триггеров с помощью команды
dfflibmap –liberty в базис библиотеки asic_cells

Qflow представляет собой полный набор инструментов для синтеза цифровых схем, начиная с Verilog-кода и заканчивая физической топологией для конкретного технологического процесса изготовления. В настоящее время компания efabless (efabless.com) при поддержки кремниевой фабрики Xfab осуществляет сотрудничество в области проектирования БИС с использованием программных продуктов с открытым кодом.

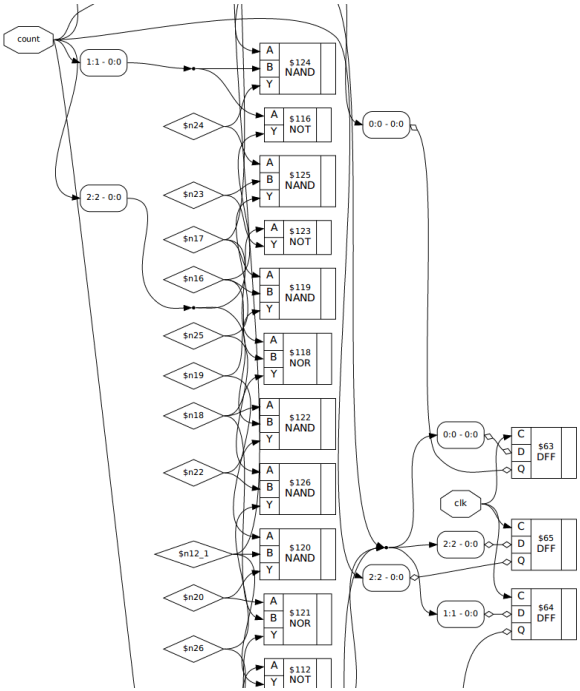


Рис. 2.6. Отображение логических элементов с помощью команды abc –liberty в базис библиотеки asic_cells

Рассмотрим использование маршрута проектирования заказных БИС Qflow, который предполагает использование программных инструментов с открытым кодом Yosys/ABC (синтез и оптимизация), Graywolf (глобальное размещение), Qrouter (детальная трассировка), Magic (топологический редактор) (рис. 2.7). Qflow поддерживает основные промышленные форматы САПР БИС — LEF (Library Exchange Format, формат библиотечного обмена), DEF (Design Exchange Format, формат конструктивного обмена), GDSII (для топологии) и др. LEF-файл условно делится на библиотечную и технологическую части. Библиотечная часть содержит описание внешней геометрии ячеек (границы ячеек, позиции), информацию о расположении входов и выходов и другие обструкции (препятствия) для трассировки. Данная абстракция используется

на этапах размещения и разводки элементов на кристалле, а информация о внутренней структуре ячеек для этих стадий не важна. Технологическая часть содержит разнообразные правила проектирования, необходимые для размещения и разводки. Включает требования к размерам металла, к направлениям, к спейсингам (расстояния между соседними блоками, межслойными переходами (via), металлами, ячейками и т. д.) и множество другой технологической информации. Эта информация используется для правильного размещения и разводки в соответствии с технологическими нормами.

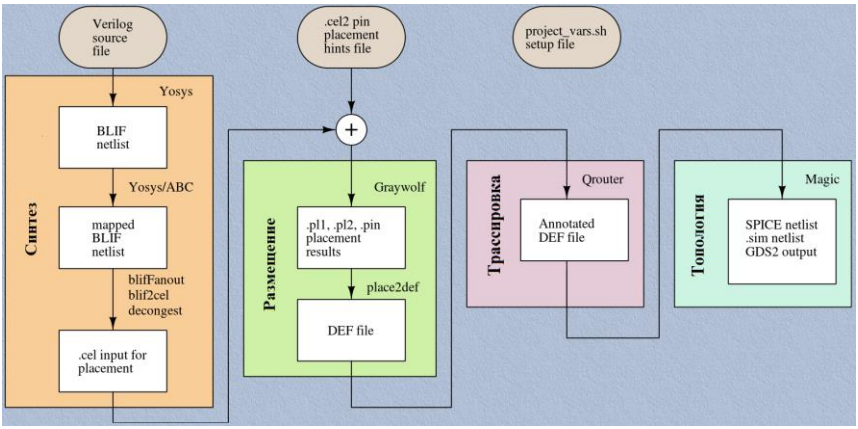


Рис. 2.7. Маршрут проектирования заказных цифровых БИС Qflow с использованием программных инструментов с открытым кодом Yosys/ABC, GrayWolf, Qrouter, Magic

Применение маршрута проектирования заказных БИС Qflow с открытым кодом подобно проектированию БИС с использованием индустриальных (коммерческих) САПР БИС, например, Cadence (размещение и трассировка средствами Cadence Encounter), Synopsys, Tanner.

По умолчанию основным средством Verilog-синтеза выбран Yosys/ABC, а Odin-II/ABC могут быть указаны в качестве альтернативного интерфейса.

После стадии отображения логических элементов на библиотеку стандартных ячеек осуществляется их размещение и трассировка на кристалле. Этап размещения определяет грубую оценку маршрутизации. Все ячейки (логические элементы) размещаются в блоке таким способом, чтобы минимизировать суммарную длину проводников, соединяющих все контакты ячеек вместе. В настоящее время разработан инструмент размещения, известный как GrayWolf (под названием «TimberWolf») в Йельском университете (Yale University, частный исследовательский университет США), и распространялся с открытым исходным кодом в течение длительного времени, пока не был принят как коммерческим. Последняя версия с открытым исходным кодом не выполняет детальную трассировку, но является инструментом размещения профессионального уровня.

Для детальной трассировки размещенных ячеек на кристалле БИС используется программный инструмент Qrouter (поддерживается http://opencircuitdesign.com/qrouter/).

Qrouter представляет собой инструмент для физического соединения топологических ячеек логических элементов внутри блока согласно списку соединений с помощью сгенерированных металлических слоев и межслойных контактов. Базируется на использовании модифицированного волнового алгоритма трассировки Ли. Представляет собой двухстадийный, многослойный трассировщик, прокладывает трассы поверх топологических ячеек (по принципу бесканальное «море вентилей»). Читает форматы LEF и DEF как входные и формирует аннотируемый DEF-файл как выходной.

Библиотеки стандартных ячеек цифровых БИС являются основным компонентом маршрута проектирования. Инструменты высокоуровневого синтеза коммерческих САПР БИС используют патентованные библиотеки, предоставляемые различными кремниевыми фабриками. Проблема с патентованными библиотеками заключается в том, что они не могут быть использованы в качестве примеров проектирования

цифровых устройств и размещены на общедоступных вебсайтах. Однако все же существует несколько комплектов библиотек стандартных ячеек с открытым исходным кодом для популярных технологических процессов. Некоторые из них основаны на единых правилах проектирования масштабируемой КМОП-технологии (MOSIS Scalable CMOS design rules) от MOSIS (www.mosis.com, www.isi.edu) предоставляемых в открытом доступе. Набор правил проектирования от MOSIS не является столь специфичным, чем правила кремниевых фабрик.

Многие кремниевые фабрики, такие как TSMC, IBM, AMI, Orbit выпускают по правилам проектирования MOSIS Scalable CMOS design rules субмикронные БИС. Единые правила проектирования, поддерживаемые разработчиками САПР БИС, позволяют дизайн-центрам, не имеющим своих производств, решить задачу выхода на рынок своих изделий и обеспечения переносимости проектов среди ведущих в мире изготовителей.

В качестве основной единицы измерения расстояния выбрана масштабная величина λ (лямбда). В масштабируемой технологии КМОП (SCMOS) топология схемы рисуется в соответствии с лямбда методологией. Единица измерения λ масштабируется в соответствии с изменением технологии в сторону уменьшения размеров, что позволяет избежать повторного проектирования топологии кристалла.

Топологический редактор Magic в маршруте Qflow использует библиотеки стандартных ячеек цифровых БИС, реализованные по масштабируемой КМОП-технологии, совместимые с топологическими библиотеками коммерческих кремниевых фабрик, главным образом TSMC и AMI 0.18, 0.35 и 0.5 мкм. По умолчанию используется библиотека с открытым исходным кодом OSU 0.35 (разработана в государственном университете Оклахомы, США) с топологическими проектными нормами 0.35 мкм, не требующая лицензионного соглашения, как, например, использование библиотек кремние-

вых фабрик (MOSIS AMI, HP и др.). Набор библиотек для проектирования по методу стандартных ячеек доступен для скачивания с сайта vlsiarch.ecen.okstate.edu. Библиотека OSU 0.35 содержит набор файлов, включая файлы в формате LEF и GDSII.

На рис. 2.8 показан программный инструмент Graywolf в процессе размещения логических элементов 3-разрядного суммирующего счетчика на кристалле. В состав Graywolf входит дополнительная утилита place2def, которая преобразует выходные результаты в формат DEF, который содержит информацию о списках цепей и размещенных элементах. Записывает дополнительную информацию о количестве используемых слоев металлизации в cfg-файл, необходимую для маршрутизации. На рис. 2.9 представлена абстрактная топология счетчика в топологическом редакторе Magic с учетом размещения и детальной трассировки. На рис. 2.10 показана топология счетчика по масштабируемой КМОП-технологии OSU 0.35 (модификация технологического процесса кремниевой фабрики TSMC, SCN4M_SUB.20, масштабируемая КМОП-технология по правилам MOSIS, N-карман, 4 слоя металлизации, 1 слой поликремния, субмикронные проектные нормы, лямбда 0.2, https://www.mosis.com/vendors/view/tsmc/035) в топологическом редакторе Magic.

Для построения физической абстракции нужно загрузить lef-файл в консоле редактора Magic с помощью команды lef read /usr/local/share/qflow/tech/osu035/osu035_stdcells.lef.

Для перехода от абстрактного описания к полному описанию топологии (gds), определенной технологическим процессом OSU 0.35, необходимо загрузить технологический файл в формате gds2 в консоле редактора Magic с помощью команды gds read osu035_stdcells.gds2.

Таким образом, программные средства для Verilog-синтеза с открытым программным кодом позволяют реализовывать цифровые устройства как непосредственно в базис индустриальных ПЛИС, так и в базис заказных БИС с возможностью последующего изготовления на кремниевых фабриках.

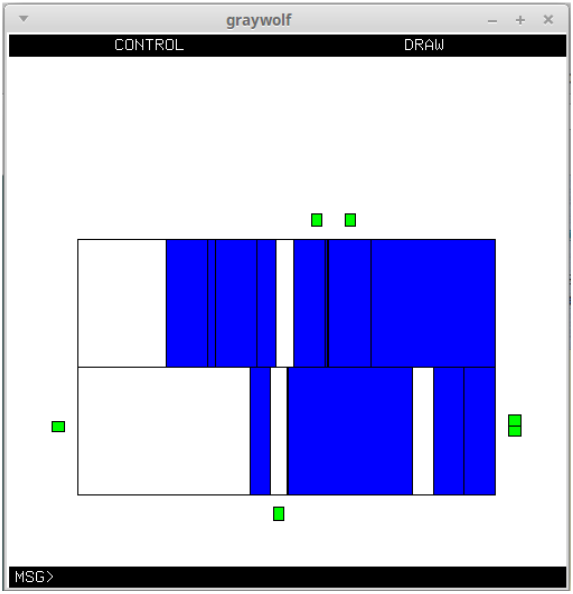


Рис. 2.8. Программный инструмент Graywolf
в процессе размещения логических элементов
3-разрядного суммирующего счетчика на кристалле

Рис. 2.9. Абстрактная топология 3-разрядного суммирующего
счетчика в топологическом редакторе Magic

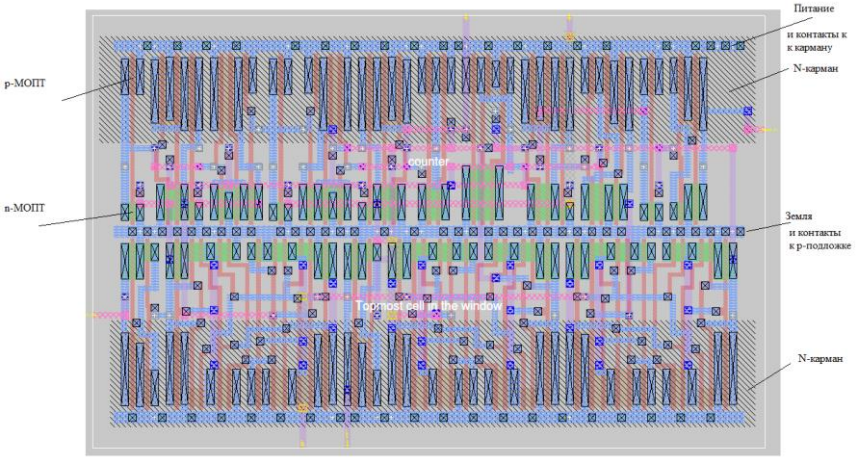


Рис. 2.10. Топология 3-разрядного суммирующего счетчика
по масштабируемой КМОП-технологии OSU 0.35
в топологическом редакторе Magic

2.2. Реализация Verilog-проектов в базис индустриальных ПЛИС Xilinx с применением синтезатора Yosys

В предыдущей главе рассматривались Verilog-проекты, синтезированные в базис ПЛИС с помощью индустриального САПР Altera Quartus II, и их последующая реализация в базисе академической ПЛИС в САПР VTR 7.0. Цель данного раздела — показать возможности программного инструмента с открытым кодом для Verilog-синтеза Yosys (разработчик Clifford Wolf, версия 0.7) в базис индустриальных ПЛИС Xilinx с применением САПР ISE ver 14.2 (рис. 2.11).

Для начала реализуем Verilog-проект «бегущие огоньки» (пример 1) с сайта github.com/cliffordwolf/yosys в базис ПЛИС xc6slx9-2-tqg144 (рис. 2.12) и посмотрим RTL-представление (рис. 2.13) в САПР ISE. Получить дополнительную информацию можно по приведенным ниже ссылкам. Также интересно

посмотреть RTL-представление в САПР Quartus II ver.13.1 (рис. 2.14). Verilog-код проекта (делитель импульсов) представляет собой 32-разрядный счетчик на базе сумматора и регистра с обратной связью, к разрядам которого (24–31) подключены 8 светодиодов led_0 – led_7. Для непрерывного счета необходимо на вход ctrl сумматора подавать сигнал логической единицы.

Воспользуемся тем, что Yosys преобразует Verilog-проекты в формат BLIF и позволяет их отображать в базис ПЛИС Xilinx серий Spartan-6 и Virtex-7. Для логической оптимизации и отображения в базис ПЛИС в Yosys интегрирован программный инструмент ABC. Реализуем проект в ПЛИС серии Spartan-6 xc6slx9-2-tqg144.

```
module top(clk, ctrl, led_7, led_6, led_5, led_4, led_3, led_2, led_1, led_0);
input clk, ctrl;
output led_7, led_6, led_5, led_4;
output led_3, led_2, led_1, led_0;
reg [31:0] counter;
always @(posedge clk)
        counter <= counter + (ctrl ? 4 : 1);
assign {led_7, led_6, led_5, led_4, led_3, led_2, led_1, led_0} = counter >> 24;
endmodule
```

Пример 1. Verilog-код счетчика (файл example.v)

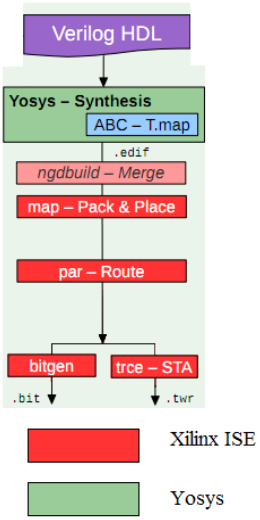


Рис. 2.11. Маршрут проектирования цифровых устройств, представленных Verilog-кодом с использованием синтезатора Yosys и САПР Xilinx ISE в базисе ПЛИС

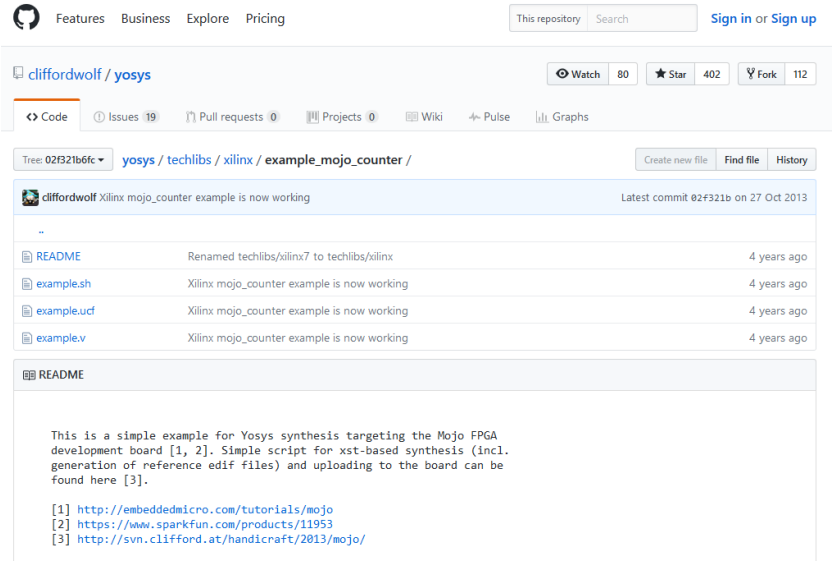


Рис. 2.12. Проект «бегущие огоньки»

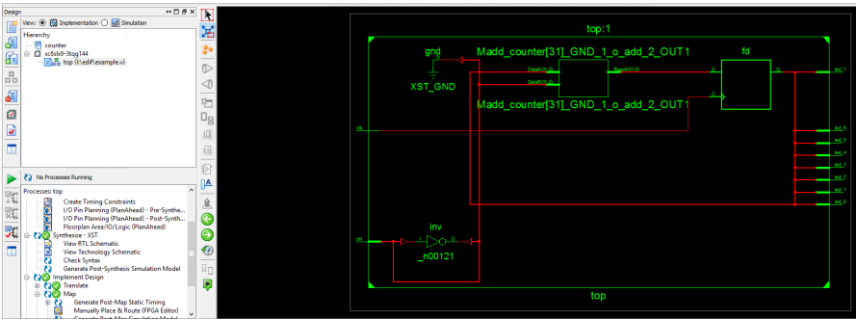


Рис. 2.13. RTL-представление проекта в САПР ISE версии 14.2

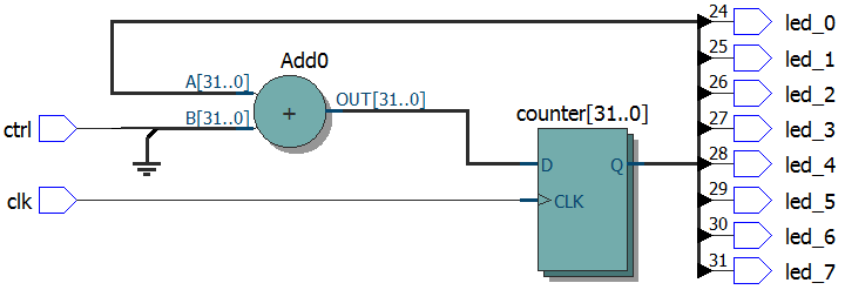


Рис. 2.14. RTL-представление проекта
в САПР Quartus II версии 13.1

Ниже приводится скрипт (пример 2) для синтеза и отображения в базис ПЛИС Xilinx серии Spartan 6 на примере рассматриваемого Verilog-проекта. Также показан файл временных ограничений (ucf-файл) для рассматриваемого проекта. Кратко рассмотрим основные используемые команды. Более подробную информацию о командах для синтеза в базис ПЛИС Xilinx можно получить на сайте Yosys в разделе: Synt_xilinx (http://www.clifford.at/yosys/cmd_synth_xilinx.html).

По команде read_verilog example.v осуществляется построение абстрактного синтаксического дерева с последующим RTL-представлением (рис. 2.15). По команде techmap крупнозернистые RTL-ячейки (сумматоры, регистры) преоб-

разуются в мелкозернистые ячейки (вентили, триггеры) (рис. 2.16). Логика проекта с помощью команды abc -lut 6 отображается непосредственно в 6-входовые LUT (рис. 2.17). Команда techmap -map +/xilinx/cells_map.v отображает 6-входовые LUT и триггеры в технологический базис ПЛИС Xilinx Spartan-6 (рис. 2.18.). Yosys также позволяет осуществлять синтез в MAC-блоки (DSP48) ПЛИС Xilinx.

На рис. 2.19 показан проект, представленный edif-файлом и файлом временных ограничений, а также сведения по задействованным логическим ресурсам ПЛИС xc6slx9-2-tqg144. Рис. 2.20 показывает задействованные логические и трассировочные ресурсы ПЛИС. Рис. 2.19 и рис. 2.20 подтверждают, что edif-файл, сформированный синтезатором Yosys по Verilog-коду, успешно обрабатывается утилитами ngdbuild, map (упаковка элементарных логических элементов в секции КЛБ, но секции при этом еще не получают реальных мест в ПЛИС) и par (секции, сформированные на этапе map, получают свои места в матрице конфигурируемых логических блоков (Place) с последующей трассировкой связей между сигналами секций (Route)) САПР Xilinx ISE.

```
# read design, чтение проекта
read_verilog example.v
# high-level synthesis, построение иерархии
hierarchy -check -top top
proc; opt; fsm; opt; techmap; opt
# mapping logic to LUTs using Berkeley ABC
abc -lut 6; opt
# map internal cells to FPGA cells
techmap -map +/xilinx/cells_map.v; opt
# insert clock buffers
select -set clocks */t:FDRE %x:+FDRE[C] */t:FDRE %d
iopadmap -inpad BUFGP O:I @clocks
# insert i/o buffers
iopadmap -outpad OBUF I:O -inpad IBUF O:I @clocks %n
# write netlist
write_edif synth.edif
```

Пример 2. Скрипт для Yosys

```
NET "clk" TNM_NET = clk;
TIMESPEC TS_clk = PERIOD "clk" 50 MHz HIGH 50%;
NET "clk" LOC = P56;
NET "ctrl" LOC = P1;
NET "led_0" LOC = P134;
NET "led_1" LOC = P133;
NET "led_2" LOC = P132;
NET "led_3" LOC = P131;
NET "led_4" LOC = P127;
NET "led_5" LOC = P126;
NET "led_6" LOC = P124;
NET "led_7" LOC = P123;
```

Пример 3. ucf-файл для проекта

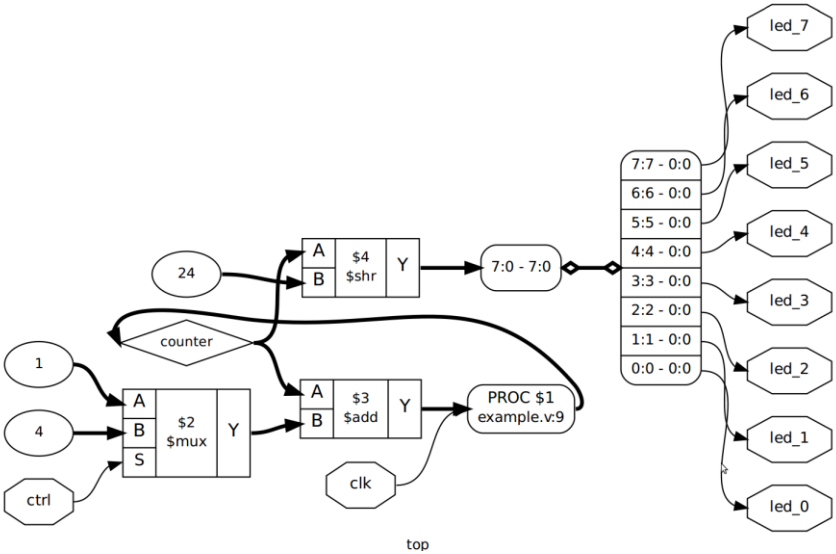


Рис. 2.15. RTL-представление проекта в Yosys

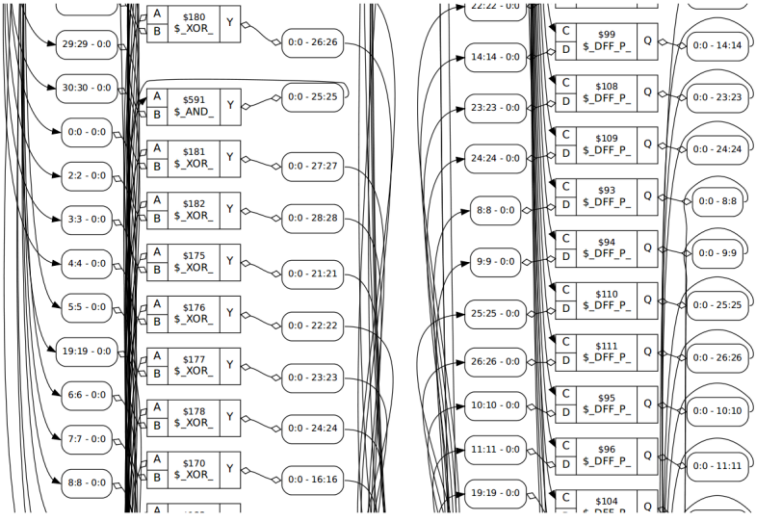


Рис. 2.16. Технологическое отображение (фрагмент)
в абстрактные вентили ($_AND_, $_XOR_)
и триггеры ($_DFF_P_)

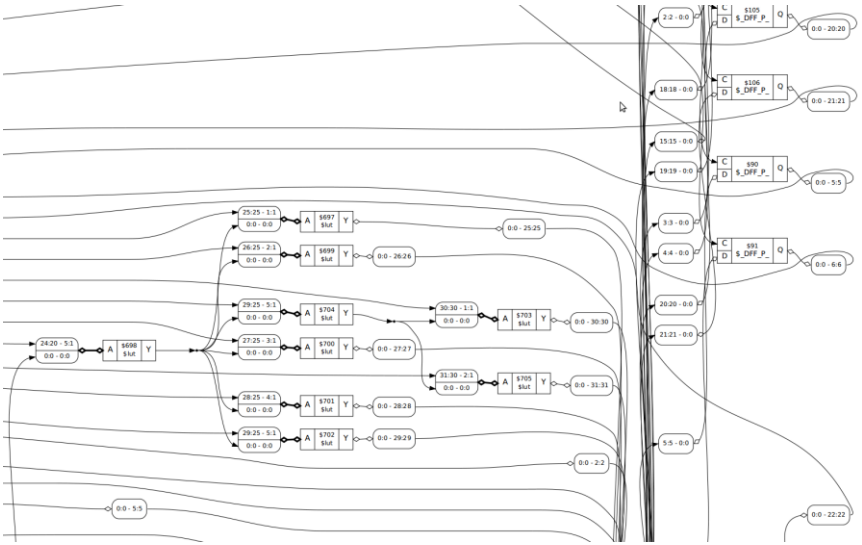


Рис. 2.17. Отображение логики (фрагмент) проекта
в базис 6-входовых LUT ($lut)

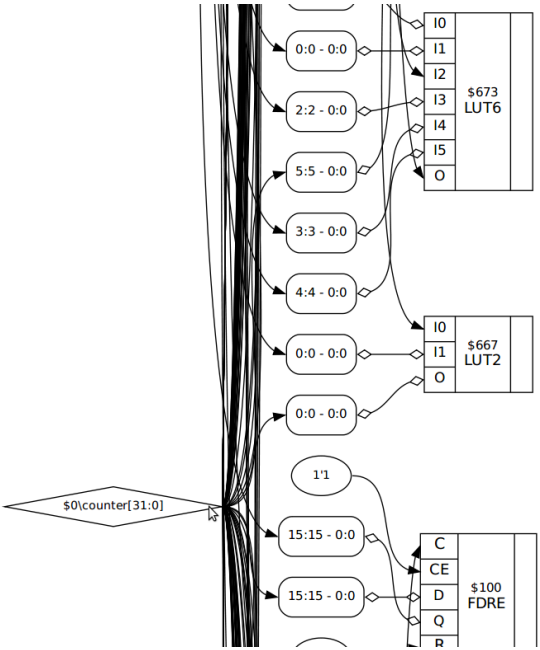


Рис. 2.18. Технологическое отображение (фрагмент)
в примитивы ПЛИС Xilinx (LUT2, LUT6, FDRE)

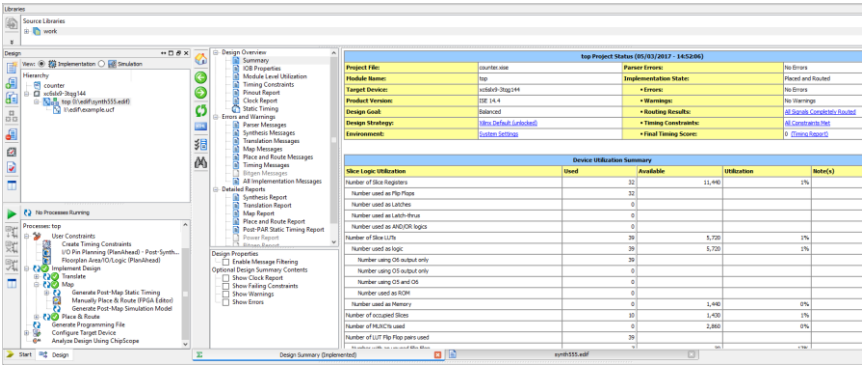


Рис. 2.19. Успешно реализованный проект,
представленный edif-файлом и файлом временных ограничений
в базис ПЛИС xc6slx9-2-tqg144

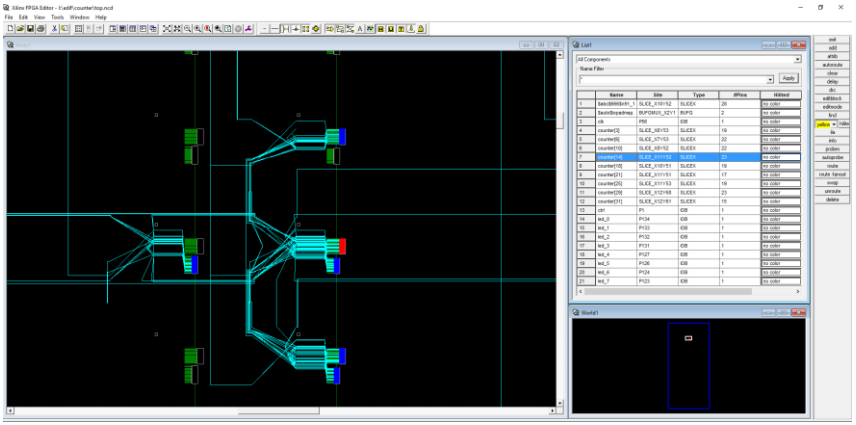


Рис. 2.20. Реализация проекта, представленного edif-файлом в базис ПЛИС xc6slx9-2-tqg144 (красным цветом подсвечена задействованная секция)

На сегодняшний момент времени программное средство для Verilog-синтеза с открытым программным кодом Yosys является наиболее популярным у зарубежных «радиолюбителей», так как позволяет реализовывать цифровые устройства в базис индустриальных ПЛИС компании Xilinx серий Spartan-6 и Virtex-7 с использованием САПР ISE и Vivado и в базис программируемых аналого-цифровых матриц GreenPAK4 компании Silego (маршрут Yosys Silego).

Также под это средство разработана открытая аппаратная платформа IcoBoard 1.0, представляющая собой отладочный набор на базе ПЛИС iCE40 фирмы Lattice Semiconductor с программным обеспечением icoTC (проект IceStorm). Yosys также позволяет отображать Verilog-проекты в базис библиотечных ячеек заказных БИС (в формате Liberty) и используется в маршруте проектирования заказных БИС Qflow.

Yosys успешно тестировался на проектах (с сайта Open Cores) с открытым исходным кодом OpenMSP430, Amber ARMv2 Clone, Navre AVR Clone, OpenRISC 1200, Rocket, PicoRV32.

2.3. Реализация Verilog-проектов в базис индустриальных ПЛИС Altera с применением синтезатора Yosys

Программный инструмент Yosys позволяет реализовывать Verilog-проекты в двух сериях ПЛИС: Altera Cyclone IV и MAX II посредством vqm-файла. Для отладки проектов можно использовать плату DE2i-150. Vqm-файл —это представление RTL-проекта на «атомном» уровне с использованием сетевых примитивов технологического нетлиста (списка межсоединений) и мегафункций конкретной серии ПЛИС Altera.

Рассмотрим пример реализации Verilog-проектов в базисе индустриальных ПЛИС Intel FPGA (Altera) с применением синтезатора Yosys. С сайта github.com/cliffordwolf/yosys обновляем Yosys до версии 0.7+194. Рассмотрим реализацию Verilog-проекта в САПР Quartus II версии 13.1 (32 и 64 разрядные) с использованием ПЛИС серии Cyclone IV GX. Протестирует vqm-файлы, формируемые Yosys на трех простых проектах: сумматор, умножитель и блок однопортового ОЗУ.

Verilog-код сумматора демонстрирует пример 1. На рис. 2.21 показано RTL-представление сумматора двух двухразрядных целых положительных чисел. Выходной сигнал сумматора f [2..0] специально расширен на один разряд. При RTL-представлении знаковый разряд входных сигналов a[1..0] и b[1..0] расширен логическим нулем (прямой код). Технологическое отображение в базис ПЛИС Cyclone IV GX EP4CGX15BF14A7 (14 400 логических элементов) по Verilog-коду показано на рис. 2.22.

```
module adder (a,b,f);
input [1:0] a;
input [1:0] b;
output [2:0] f;
assign f=a+b;
endmodule
```

Пример 1. Verilog-код сумматора

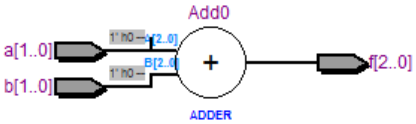


Рис. 2.21. RTL-представление сумматора целых положительных чисел на этапе анализа и синтеза по Verilog-коду

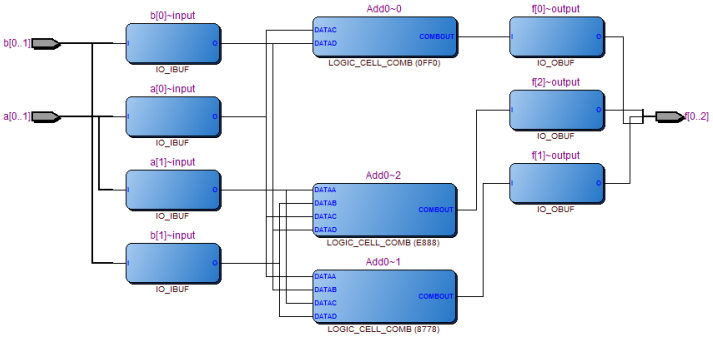


Рис. 2.22. Технологическое отображение в базис ПЛИС серии Cyclone IV GX по Verilog-коду

В папке ~/yosys/examples/intel/asicworld_lfsr запускаем скрипт run_cycloneiv (пример 2). При этом создается vqm-файл с именем adder.vqm. Пример 3 демонстрирует фрагмент vqm-файла сумматора. На рис. 2.23 показано RTL-представление Verilog-кода сумматора для ПЛИС серии Cyclone IV GX на этапе анализа и синтеза по vqm-файлу, сформированному Yosys, а на рис. 2.24 — после технологического отображения. Сравнение используемых ресурсов ПЛИС демонстрируют рис. 2.25 и рис. 2.26. Видим, что число логических элементов cycloneiv_lcell_comb в обоих случаях равно трем, а «маски» (содержимое lut) логических элементов (lcell_comb(0FF), lcell_comb(E888) и lcell_comb(E8778)) также одинаковы.

```
#!/bin/env bash
yosys -p "synth_intel -family cycloneiv -top adder -vout adder.vqm" adder.v
```

Пример 2. Содержимое скрипта run_cycloneiv для технологического отображения проекта adder.v в базис ПЛИС серии Cyclone IV GX с формированием VQM-файла

```
/* Generated by Yosys 0.7+194 (git sha1 0290b68, clang 3.4-1ubuntu3 -fPIC -Os) */
/* top = 1 */
/* src = "adder.v:1" */
module adder(a, b, f);
wire [1:0] yosys__00_;
wire [2:0] yosys__01_;
wire [1:0] yosys__02_;
/* src = "adder.v:2" */
input [1:0] a;
/* src = "adder.v:3" */
input [1:0] b;
/* src = "adder.v:4" */
output [2:0] f;
/* src = "/usr/local/bin/../share/yosys/altera_intel/cycloneiv/cells_map_cycloneiv.v:49" */
cycloneiv_lcell_comb yosys__03_ (
.combout(yosys__01_[0]),
.dataa(yosys__00_[0]),
.datab(yosys__02_[0]),
.datac(1'b1),
.datad(1'b1)
);
defparam yosys__03_.lut_mask = 16'b0110011001100110;
defparam yosys__03_.sum_lutc_input = "datac";
/* src = "/usr/local/bin/../share/yosys/altera_intel/cycloneiv/cells_map_cycloneiv.v:55" */
cycloneiv_lcell_comb yosys__04_ (
.combout(yosys__01_[1]),
.dataa(yosys__00_[0]),
.datab(yosys__02_[0]),
.datac(yosys__00_[1]),
.datad(yosys__02_[1])
);
defparam yosys__04_.lut_mask = 16'b1000011101111000;
defparam yosys__04_.sum_lutc_input = "datac";
/* src = "/usr/local/bin/../share/yosys/altera_intel/cycloneiv/cells_map_cycloneiv.v:55" */
cycloneiv_lcell_comb yosys__05_ (
.combout(yosys__01_[2]),
.dataa(yosys__00_[1]),
.datab(yosys__02_[1]),

.datac(yosys__00_[0]),
.datad(yosys__02_[0])
);
```

Пример 3. Фрагмент vqm-файла сумматора
для технологического отображения в базис ПЛИС
серии Cyclone IV GX, сформированный синтезатором Yosys

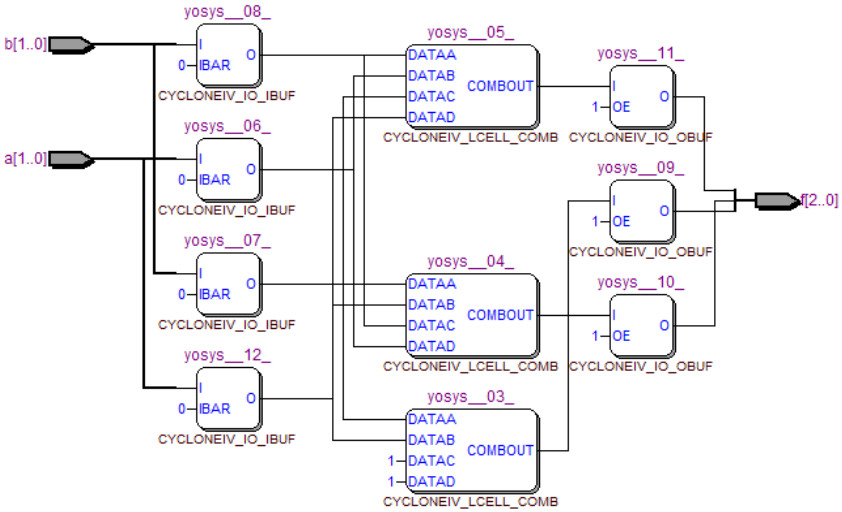


Рис. 2.23. RTL-представление Verilog-кода сумматора
для ПЛИС Cyclone IV GX EP4CGX15BF14A7 на этапе анализа
и синтеза по vqm-файлу, сформированному синтезатором Yosys

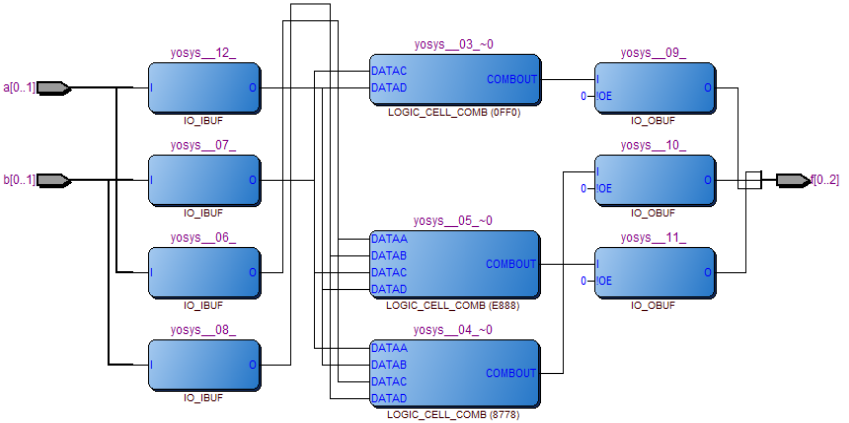


Рис. 2.24. RTL-представление Verilog-кода сумматора
после технологического отображения
в базис ПЛИС Cyclone IV GX EP4CGX15BF14A7

```
2.18. Printing statistics.

=== adder ===

   Number of wires:                 6
   Number of wire bits:            14
   Number of public wires:          3
   Number of public wire bits:      7
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                10
     cycloneiv_io_ibuf              4
     cycloneiv_io_obuf              3
     cycloneiv_lcell_comb           3

2.19. Executing CHECK pass (checking for obvious problems).
checking module adder..
found and reported 0 problems.

2.20. Executing Verilog backend.
Dumping module `\adder'.

End of script. Logfile hash: f9de992dcc
CPU: user 0.09s system 0.02s, MEM: 39.35 MB total, 9.77 MB resident
Yosys 0.7+194 (git sha1 0290b68, clang 3.4-1ubuntu3 -fPIC -Os)
Time spent: 26% 5x read_verilog (0 sec), 11% 12x opt_clean (0 sec), ...
```

Рис. 2.25. Используемые ресурсы проекта при технологическом отображении в базис ПЛИС серии Cyclone IV GX

Fitter Resource Usage Summary

	Resource	Usage
1	◢ Total logic elements	3 / 14,400 (< 1 %)
1	-- Combinational with no register	3
2	-- Register only	0
3	-- Combinational with a register	0
2		
3	◢ Logic element usage by number of LUT inputs	
1	-- 4 input functions	2
2	-- 3 input functions	0
3	-- <=2 input functions	1
4	-- Register only	0
4		
5	◢ Logic elements by mode	
1	-- normal mode	3
2	-- arithmetic mode	0
6		
7	◢ Total registers*	0 / 14,733 (0 %)
1	-- Dedicated logic registers	0 / 14,400 (0 %)
2	-- I/O registers	0 / 333 (0 %)
8		
9	Total LABs: partially or completely used	1 / 900 (< 1 %)
10	Virtual pins	0
11	◢ I/O pins	7 / 81 (9 %)
1	-- Clock pins	0 / 6 (0 %)
2	-- Dedicated input pins	0 / 12 (0 %)
12		
13	Global signals	0
14	M9Ks	0 / 60 (0 %)
15	Total block memory bits	0 / 552,960 (0 %)
16	Total block memory implementation bits	0 / 552,960 (0 %)
17	PLLs	0 / 3 (0 %)
18	Global clocks	0 / 20 (0 %)
19	JTAGs	0 / 1 (0 %)
20	CRC blocks	0 / 1 (0 %)
21	ASMI blocks	0 / 1 (0 %)

Рис. 2.26. Используемые ресурсы ПЛИС Cyclone IV GX EP4CGX15BF14A7 после полной компиляции

Для того чтобы полностью убедиться, что полученный VQM-файла проекта сумматора действительно функционирует, разработаем проект (рис. 2.27) и осуществим функциональное моделирование. Видим, что сумматор работает корректно (рис. 2.28).

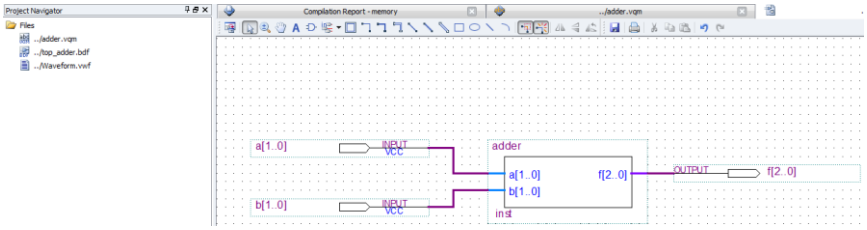


Рис. 2.27. Проект сумматора на основе vqm-файла, сформированного Yosys

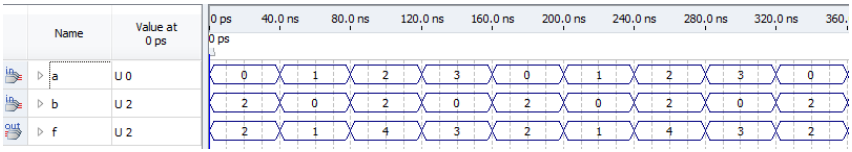


Рис. 2.28. Функциональное моделирование проекта на основе vqm-файла, сформированного программным инструментом Yosys

Verilog-код умножителя двух 16-разрядных сигналов показывает пример 4. Изучая технологическое отображение и размещение кластеров логических элементов на кристалле, видим, что по Verilog-коду реализовался умножитель на мегафункции lpm_mult только на логических ресурсах ПЛИС (встроенные умножители не используются) (рис. 2.29).

```
module mult (a,b,y);
input [15:0] a;
input [15:0] b;
output [32:0] y;
assign y = a*b;
endmodule
```

Пример 4. Verilog-код умножителя

На рис. 2.30 показано RTL-представление умножителя на логических ресурсах ПЛИС на стадии анализа и синтеза с использованием vqm-файла, сформированного Yosys. На рис. 2.31 и рис. 2.32 приведены ресурсы ПЛИС после полной компиляции. Рис. 2.33 показывает, что Yosys при синтезе не использует арифметический режим. Так общее число логических элементов по Verilog-коду, который обрабатывает синтезатор QIS САПР Quartus II, 183 логических элементов (lcell_comb) работают в нормальном режиме и 161 — в арифметическом режиме, при синтезе Yosys используются 651 логических элементов, которые работают только в нормальном режиме. Для ПЛИС, имеющую 21 280 логических элементов, увеличение числа логических ресурсов составляет 1%.

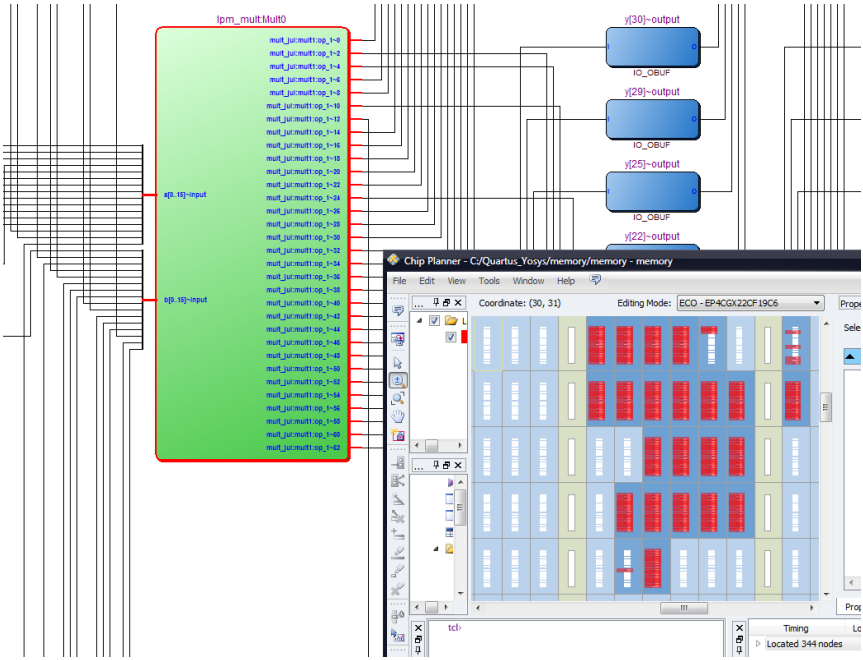


Рис. 2.29. Технологическое отображение
и расположение умножителя на кристалле
после стадии размещения и трассировки в базис ПЛИС

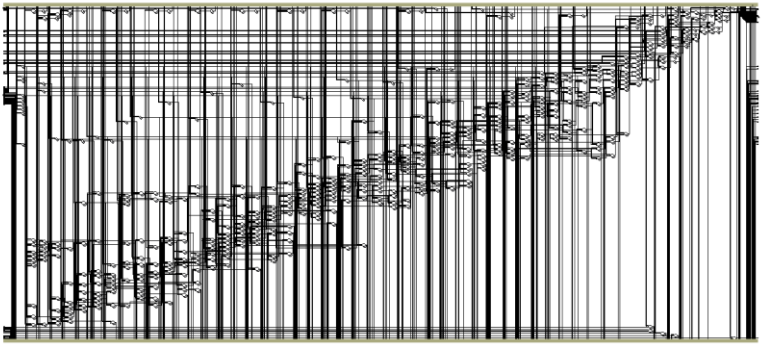


Рис. 2.30. RTL-представление умножителя
на логических ресурсах ПЛИС на стадии анализа и синтеза

Fitter Resource Usage Summary

	Resource	Usage
1	◢ Total logic elements	344 / 21,280 (2 %)
1	-- Combinational with no register	344
2	-- Register only	0
3	-- Combinational with a register	0
2		
3	◢ Logic element usage by number of LUT inputs	
1	-- 4 input functions	142
2	-- 3 input functions	158
3	-- <=2 input functions	44
4	-- Register only	0
4		
5	◢ Logic elements by mode	
1	-- normal mode	183
2	-- arithmetic mode	161
6		
7	◢ Total registers*	0 / 22,031 (0 %)
1	-- Dedicated logic registers	0 / 21,280 (0 %)
2	-- I/O registers	0 / 751 (0 %)
8		
9	Total LABs: partially or completely used	24 / 1,330 (2 %)
10	Virtual pins	0
11	◢ I/O pins	65 / 167 (39 %)
1	-- Clock pins	1 / 6 (17 %)
2	-- Dedicated input pins	0 / 16 (0 %)
12		
13	Global signals	0
14	M9Ks	0 / 84 (0 %)
15	Total block memory bits	0 / 774,144 (0 %)
16	Total block memory implementation bits	0 / 774,144 (0 %)
17	Embedded Multiplier 9-bit elements	0 / 80 (0 %)
18	PLLs	0 / 4 (0 %)
19	Global clocks	0 / 20 (0 %)
20	JTAGs	0 / 1 (0 %)
21	CRC blocks	0 / 1 (0 %)

Рис. 2.31. Используемые ресурсы ПЛИС EP4CGX22BF14C6 по Verilog-коду умножителя после полной компиляции

Fitter Resource Usage Summary

	Resource	Usage
1	◢ Total logic elements	651 / 21,280 (3 %)
1	-- Combinational with no register	651
2	-- Register only	0
3	-- Combinational with a register	0
2		
3	◢ Logic element usage by number of LUT inputs	
1	-- 4 input functions	366
2	-- 3 input functions	242
3	-- <=2 input functions	43
4	-- Register only	0
4		
5	◢ Logic elements by mode	
1	-- normal mode	651
2	-- arithmetic mode	0
6		
7	◢ Total registers*	0 / 22,031 (0 %)
1	-- Dedicated logic registers	0 / 21,280 (0 %)
2	-- I/O registers	0 / 751 (0 %)
8		
9	Total LABs: partially or completely used	42 / 1,330 (3 %)
10	Virtual pins	0
11	◢ I/O pins	65 / 167 (39 %)
1	-- Clock pins	1 / 6 (17 %)
2	-- Dedicated input pins	0 / 16 (0 %)
12		
13	Global signals	0
14	M9Ks	0 / 84 (0 %)
15	Total block memory bits	0 / 774,144 (0 %)
16	Total block memory implementation bits	0 / 774,144 (0 %)
17	Embedded Multiplier 9-bit elements	0 / 80 (0 %)
18	PLLs	0 / 4 (0 %)
19	Global clocks	0 / 20 (0 %)
20	JTAGs	0 / 1 (0 %)

Рис. 2.32. Используемые ресурсы ПЛИС EP4CGX22BF14C6 по vqm-файлу, сформированному синтезатором Yosys

В заключение отметим, что на сегодняшний момент времени программное средство для Verilog-синтеза с открытым программным кодом Yosys является наиболее популярным в академических университетских центрах, так как позволяет реализовывать цифровые устройства в базис индустриальных ПЛИС компании Xilinx серий Spartan-6 и Virtex-7 с использованием САПР ISE и Vivado, в базис программируемых аналого-цифровых матриц GreenPAK4 компании Silego (маршрут Yosys Silego) и в базис ПЛИС Intel FPGA (серии Cyclone IV и MAX II).

Yosys также позволяет отображать Verilog-проекты в базис библиотечных ячеек заказных БИС (в формате Liberty) и используется в маршруте проектирования заказных БИС Qflow с возможностью последующего изготовления на кремниевых фабриках по единым правилам проектирования масштабируемой КМОП-технологии (MOSIS Scalable CMOS design rules) от MOSIS (www.mosis.com), предоставляемые в открытом доступе.

В настоящее время создана открытая аппаратная платформа icoBoard 1.0. icoBoard представляет собой отладочный набор на базе ПЛИС iCE40 фирмы Lattice Semiconductor. Основной отличительной особенностью данной платформы является использование для программирования ПЛИС открытого и свободного программного обеспечения icoTC, основными компонентами которого являются: синтезатор YoSys; размещение и трассировка Arachne-pnr; программные инструменты для формирования битового потока и документация на ПЛИС iCE40 Lattice IceStorm; загрузчик битового потока (программатор) iceprog.

Yosys успешно тестировался на проектах (с сайта Open Cores) с открытым исходным кодом: OpenMSP430, Amber ARMv2 Clone, Navre AVR Clone, OpenRISC 1200, Rocket, PicoRV32.

3. РЕАЛИЗАЦИЯ VERILOG-ПРОЕКТОВ В БАЗИС ПЛИС СЕРИИ 5578

В рамках импортозамещения зарубежной электронной компонентной базы разработана новая серия ПЛИС 5578 по 180 нм КМОП-технологии с шестью слоями металлизации и напряжением питания 1,8 В. Серия представлена двумя типами ПЛИС: 5578ТС014 и 5578ТС024. ПЛИС содержат увеличенную емкость встроенной памяти и ЦОС-блоки (аппаратные умножители), ранее отсутствующие в ПЛИС серии 5576.

ПЛИС 5578ТС014 и 5578ТС024 не являются прямыми аналогами зарубежных ПЛИС серии Cyclone II (EP2C5 и EP2C8) фирмы Altera, а могут быть использованы для их замены. По архитектурным решениям новые отечественные ПЛИС занимают промежуточное положение между низкобюджетной серией Cyclone II и высокопроизводительными ПЛИС серии Stratix III с АЛМ. ПЛИС серии 5578 также близки по своим техническим характеристикам к ПЛИС серии Spartan-3 фирмы Xilinx.

3.1. Структура ПЛИС серии 5578

В основе структуры ПЛИС лежат следующие компоненты: матрица памяти, управляющая коммутацией и режимами работы элементов ядра ПЛИС; система локальных (СЛМ) и глобальных межсоединений (СГМ) с блоками коммутации (БК); глобальные сигналы управления; логические блоки (ЛБ), состоящие из 8 адаптивных логических модулей (АЛМ), которые используются для реализации комбинационной логики, счетчиков, сумматоров, конечных автоматов состояний; блоки пользовательской памяти (БПП), использующиеся для реализации разнообразных функций памяти или сложных логических функций; блоки цифровой обработки сигналов (БЦОС или ЦОС-блоки), предназначенные для реализации функций умножения, умножения-накопления и др., используемых при

построении цифровых фильтров; модули ввода-вывода (МВВ), осуществляющие передачу сигналов между ядром и периферией ПЛИС; программируемые периферийные блоки ввода-вывода (БВВ).

ПЛИС 5578ТС014 и 5578ТС024 в своей основе имеют одинаковую структурную схему, но ПЛИС 5578ТС014 обладает лучшей трассировочной способностью, за счет увеличения числа локальных и глобальных трассировочных ресурсов. На рис. 3.1 показана структурная схема ПЛИС серии 5578. Используются следующие сокращения: СЛМ — система локальных межсоединений; СГМ — система глобальных межсоединений; МПВВ — матрица памяти элементов ввода-вывода; МПЛБ — матрица памяти логического блока встроенной памяти; МПБЦОС — матрица памяти блока цифровой обработки сигналов; БВВ — буфер ввода-вывода; ВВ — элемент ввода-вывода; ЛБ — логический блок; БВП — блок встроенной памяти; БЦОС — блок цифровой обработки сигналов. В ПЛИС серии 5578 используется новая трассировочная структура с блоками памяти и цифровой обработкой сигналов, содержащая 7 уровней программируемых межсоединений с длиной сегментируемых межсоединений в 4 кластера и различной шириной трассировочного канала из разнонаправленных пар межсоединений.

На рис. 3.2 показан фрагмент структуры ПЛИС серии 5578, поясняющий принцип организации многоуровневых межсоединений. Фрагмент состоит из шести кластеров, расположенных в двух рядах соединительных блоков, коммутирующих сигналы с уровня треков (трек — короткое межсоединение в глобальном трассировочном канале) на уровень локальных межсоединений двух соседних кластеров. Между углами кластеров расположены два маршрутизатора. Кластер состоит из восьми АЛМ, имеет 16 выходов, выходящих на четыре внешние стороны кластера. Входы АЛМ расположены внутри кластера и выходят на локальные межсоединения.

Локальные межсоединения входят в кластер со всех сторон с соответствующих коммутаторов и расходятся ко всем входам АЛМ.

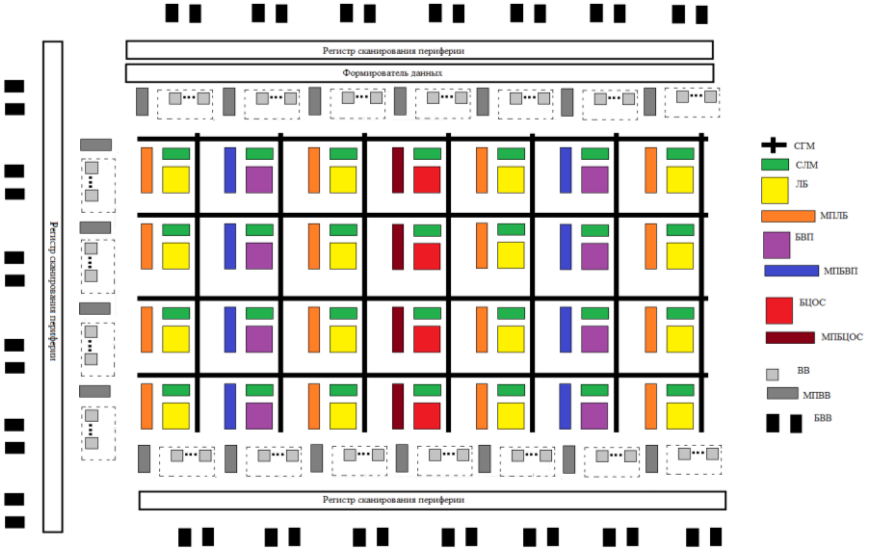


Рис. 3.1. Структура ПЛИС серии 5578

Логический блок ПЛИС серии 5578 отличается от ЛБ ПЛИС Altera серии Cyclone II. ЛБ состоит из 8 АЛМ, каждый из которых имеет 8 входов данных и может реализовывать одну произвольную 6-входовую или две произвольные 4-входовые функции.

В ЛБ входят 48 локальных линий, которые могут быть скоммутированы на входы данных АЛМ, а также быть использованы для управляющих сигналов. В центре ЛБ находятся блоки формирования и коммутации управляющих сигналов. Управляющие сигналы могут быть получены с 6 глобальных выделенных шин или с локальных линий. Кроме того, ЛБ имеет специальные линии register_chain и shared_arithmetic, а также 3 линии распространения переноса (соответственно схеме суммирования с выбором переноса).

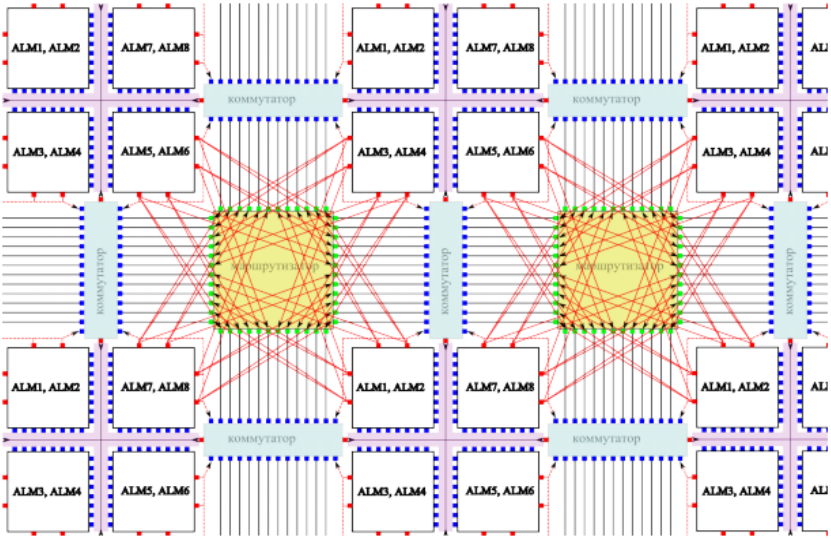


Рис. 3.2. Фрагмент структуры ПЛИС серии 5578 с локальными и глобальными трассировочными ресурсами

Разработанный АЛМ по своей структуре схож с АЛМ ПЛИС серии Stratix III, обеспечивающий повышение быстродействия ПЛИС и эффективность упаковки проектов пользователя за счет расширенных режимов работы LUT5, LUT6 и встроенного сумматора для логико-арифметических вычислений. В ЦОС-блоках ПЛИС серии Cyclone II нет встроенных аппаратных сумматоров, и все арифметические операции выполняются на LUT, работающих в арифметических режимах.

АЛМ состоит из восьми 3-входовых LUT, двух выделенных сумматоров и двух триггеров с возможностью синхронного и асинхронного сброса, синхронной загрузки из нескольких источников, использования обратной связи и обвода триггера.

Блок пользовательской памяти (БПП) предназначен для эффективного хранения пользовательских данных, передачи данных между доменами с помощью FIFO, хранения буферов и т. д. БПП поддерживают однопортовый (single-port), простой двухпортовый (simpe dual-port) и полнофункциональный

двухпортовый (true dual-port) режимы работы. По своей структуре БПП близок к блоку памяти типа M4K ПЛИС серии Cyclone II.

Основные характеристики блока: 4 096 бит памяти на блок (4 608 бит, включая биты четности); поддержка различных конфигураций портов; полнофункциональный двухпортовый режим (два независимых порта с возможностью чтения и записи по каждому); сигнал byte enable для маскирования входов при записи; возможность инициализации содержимого памяти при загрузке.

Блок цифровой обработки сигналов, ЦОС-блок используются для эффективной реализации функций, используемых при цифровой обработке сигналов. Структурная схема ЦОС-блока показана на рис. 3.3. По своей структуре ЦОС-блок ПЛИС серии 5578 близок к ЦОС-блоку ПЛИС Altera Cyclone II, но имеет некоторые архитектурные особенности, характерные для ЦОС-блоков серии Stratix III, например, содержит аппаратный сумматор на выходах умножителя (в настоящее время он не доступен, т. к. ПЛИС серии 5578 выполнены для замены ПЛИС серии Cyclone II). Состоит из входных и выходных регистров, умножителя с размерностью операндов 18×18.

По классификации фирмы Altera ЦОС-блок ПЛИС серии Cyclone II (DSP Block) называют MAC-Block, т. к. его основной функцией является умножение и накопление.

В каждом ЦОС-блоке имеется возможность скоммутировать входные сигналы на входные регистры или напрямую на умножитель. Каждый вход может быть скоммутирован через регистры независимо от других. Также возможно скоммутировать выходной сигнал на выходные регистры или напрямую на выход.

Для входных и выходных регистров используются следующие управляющие сигналы: тактовый сигнал; сигнал разрешения тактирования; асинхронный сброс. Все входные и выходные регистры в пределах одного ЦОС-блока управля-

ются одним тактовым сигналом, сигналом разрешения тактирования и асинхронным сбросом.

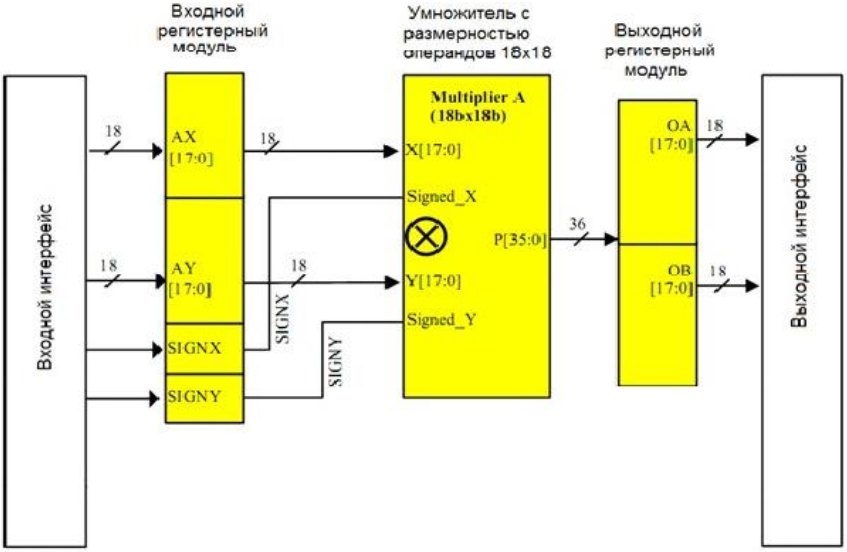


Рис. 3.3. Структура ЦОС-блока ПЛИС серии 5578 в режиме совместимости с MAC-блоком ПЛИС Altera Cyclone II

Умножитель поддерживает разрядности 18×18 бит и менее. Для определения знакового представления переменных используются сигналы: signa и signb. Если сигнал signa или signb находится в состоянии высокого уровня, то dataa или datab имеют знаковое представление. Если хотя бы одна из переменных знаковая, то результат умножения тоже знаковый.

3.2. Среда разработки конфигурационных данных ПЛИС серии 5578

Для программирования ПЛИС создана среда разработки конфигурационных данных (АО «КТЦ «ЭЛЕКТРОНИКА») и отладочная плата ОП5578 (АО «Воронежский завод полупроводниковых приборов — сборка»).

Процесс проектирования ПЛИС серии 5578 осуществляется в САПР Altera Quartus II версии 9.0 до 13.0 включительно с поддержкой ПЛИС серии Cyclone II и основывается на использовании VQM-файлов (Verilog Quartus Mapping). VQM-файл является ограниченным подмножеством формата Verilog и используется для технологического мэппирования (отображения) проекта с помощью сетевых примитивов в уникальный базис ПЛИС.

Составной частью процесса компиляции проекта в базис ПЛИС является этап анализа и синтеза. Под термином «синтез логики» следует понимать автоматическое преобразование проекта с уровня регистровых передач (RTL) на уровень вентилей технологического базиса ПЛИС.

VQM-файл можно извлечь как на этапе анализа и синтеза, так и после полной компиляции с учетом размещения и трассировки. На этапе синтеза логики в САПР Altera Quartus II с помощью опции Start VQM Writer меню Start необходимо получить синтезированный нетлист проекта. Среда разработки конфигурационных данных читать VQM-файлы, созданные синтезатором Synplify Pro, не может.

Среда постоянно модернизируется и обновляется. На сегодняшний момент доступна версия 1.1.5 (рис. 3.4). На этапе размещения и трассировки задается: параметр скорость/качество размещения проекта в базис ПЛИС (до 100); уровень кластеризации булевых функций (auto (автоматическая), off (выключена), lcell (кластеризация по LUT), alm (по АЛМ), alm2 (по двум АЛМ), alm4 (по четырем АЛМ), lab (по логическим блокам)); параметр скорость/качество трассировки (от 0 до 0.99); максимальное количество входов в LUT (четыре, пять и шесть); дополнительная оптимизация нетлиста при отображении логических функций в базис ПЛИС серии 5578. Опция «Временной анализ» позволяет выводить пути «регистр-регистр» с наименьшим запасом времени. Доступен выбор рабочих условий, для которых проводится анализ: «лучшие» и «худшие» (рис. 3.5 и рис. 3.6).

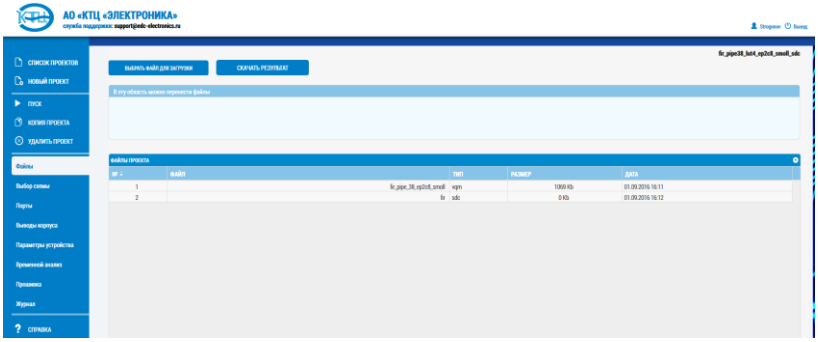


Рис. 3.4. Закладка «Файлы». Загружается vqm-файл проекта и файл временных ограничений в формате sdc

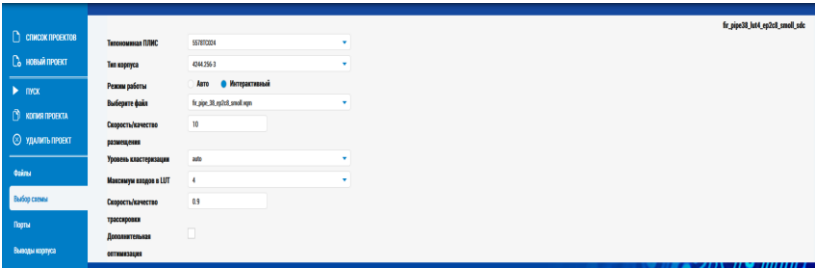


Рис. 3.5. Закладка «Выбор схемы» на этапе размещения и трассировки проекта в базис ПЛИС 5578ТС024

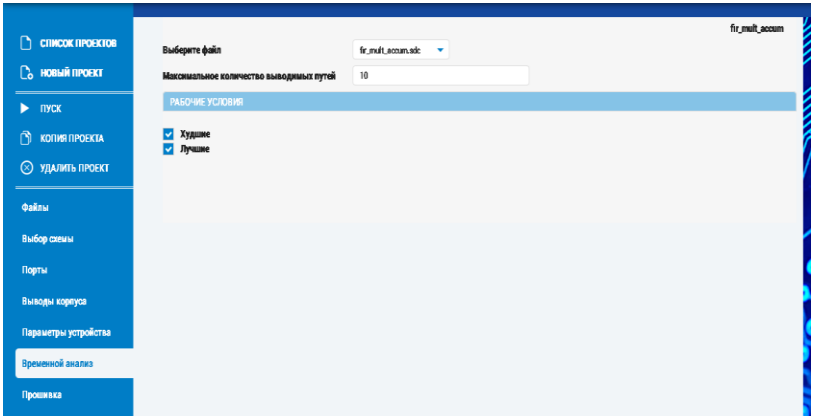


Рис. 3.6. Закладка «Временной анализ»

Для обеспечения гарантированного размещения пользовательских проектов в ПЛИС 5578ТС014, 5578ТС024 на этапе работы с САПР Quartus II (ф. Altera) необходимо учитывать следующие рекомендации по максимальному количеству задействованных в проекте логических элементов — 3 200 (для ПЛИС 5578ТС014) и 3 400 (для ПЛИС 5578ТС024).

Особенность представления конфигурационных файлов для ОЗУ и ПЗУ в проектах пользователя показана ниже (пример 1, пример 2).

```
WIDTH=8;
DEPTH=4;
ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;
CONTENT BEGIN
        0 :  00000000;
        1 :  00000000;
        2 :  00000000;
        3 :  00000000;
END;
```

Пример 1. Конфигурация ОЗУ

```
WIDTH=8;
DEPTH=4;
ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;
CONTENT BEGIN
        0 :  11111110;
        1 :  11111111;
        2 :  00000111;
        3 :  00000110;
END;
```

Пример 2. Конфигурация ПЗУ

3.3. Примеры реализации проектов цифровых устройств обработки сигналов в ПЛИС серии 5578 с использованием тестов производительности

Случайным образом отберем на интернет-ресурсе www.opencores.org семь проектов для реализации в базисе ПЛИС EP2C5F256C8 (табл. 3.1). Проекты oc_vga_lcd и oc_oc8051 не реализуются в ПЛИС EP2C5F256C8 по причине нехватки ресурсов ввода/вывода. Остальные пять проектов соответствуют требованиям по ресурсам и реализуются в ПЛИС EP2C5F256C8. А проект oc_minirisc не реализуется в ПЛИС 5578ТС014 по причине отсутствия поддержки режима работы блочной памяти «Old memory contents appear» (old_data). При выборе режима old_data на выходной шине q отображаются старые данные, хранящиеся в ОЗУ по конкретному адресу, прежде чем новые данные будут записаны по этому же адресу в память.

Таблица 3.1

Проекты, отобранные на интернет-ресурсе www.opencores.org для реализации в базисе ПЛИС EP2C5F256C8 и ПЛИС 5578ТС014

Проекты с Opencore.org	**Описание проекта и его ресурсы согласно тестам производительности Altera Quip (ввод/вывод/ логических элементов/ встроенная память, бит)**	**САПР Quartus II, реализация проекта в базис ПЛИС EP2C5F256C8, ресурсы на этапе анализа и синтеза (ввод/вывод/ логических элементов/ встроенная память, бит)**	**Среда разработки конфигурационных данных для ПЛИС 5578ТС014**
1	2	3	4
oc_vga_lcd (Verilog-код)	Ядро-видеоконтроллера VGA/LCD-дисплея: 585/2207/32600	Проект не реализуется. Ограничения по ресурсам ввода/вывода	–

Окончание табл. 3.1

1	2	3	4
oc_oc8051 (Verilog-код)	Ядро микроконтроллера 8051 (MCS-51 Intel): 189/3031/4608	Проект не реализуется. Ограничения по ресурсам ввода/вывода	–
oc_minirisc (Verilog-код)	Ядро Mini_Risc процессора/микроконтроллера, совместим с PIC16C57 от Microchip: 389/685/1024	Соответствует ресурсам ПЛИС. 99/632/1024. VQM-файл формируется	Проект не реализуется. Ошибка по причине отсутствия режима работы блочной памяти «Old memory contents appear»
oc_video_compression_systems_huffman_dec (Verilog-код)	Видеообработка изображения (JPEC энтропийное кодирование), декодер Хаффмана 23/639/0	Соответствует ресурсам ПЛИС. 23/689/0. VQM-файл формируется	Проект реализуется
oc_video_compression_systems_huffman_enc (Verilog-код)	кодер Хаффмана 23/613/0	Соответствует ресурсам ПЛИС. 23/605/0. VQM-файл формируется	Проект реализуется
oc_dct_slow (Verilog-код)	Дискретное косинусное преобразование (алгоритм сжатия изображения) 24/36/0	Соответствует ресурсам ПЛИС. Логических элементов 6%. Встроенная память не используется. VQM-файл формируется	Проект реализуется
oc_simple_fm_receiver (VHDL-код)	Ядро FM-приемника 22/826/8192	Соответствует ресурсам ПЛИС. 22/1298/0. VQM-файл формируется	Проект реализуется

На ПЛИС 5578ТС014 также реализуются все HDL-проекты barrel16 (устройство быстрого сдвига или комбинационный сдвигатель влево или вправо на указанное число позиций), barrel16a, barrel32, barrel64, mux32_16bit (мультиплексор 32 в 16), mux64_16bit (мультиплексор 64 в 16),

mux8_128bit (мультиплексор 8 в 128), mux8_64bit (мультиплексор 8 в 64), xbar_16x16 (коммутатор размерностью 16×16), ts_mike_fsm (конечный автомат), взятые из тестов производительности Quartus University Interface Program (QUIP, Version 1.0, June 6, 2005).

В заключение отметим, что ПЛИС 5578ТС014 и 5578ТС024 в своей основе имеют одинаковую структурную схему, но ПЛИС 5578ТС014 обладает лучшей трассировочной способностью за счет увеличения числа локальных и глобальных трассировочных ресурсов. ПЛИС могут быть рекомендованы для разработки устройств цифровой обработки сигналов. ПЛИС 5578ТС024 хорошо подходит для реализации проектов, насыщенных триггерами и умножителями, так как содержит повышенную функциональную емкость. Например, на ПЛИС 5578ТС024 можно разработать параллельный КИХ-фильтр на 38 отводов, который невозможно реализовать в ПЛИС 5578ТС014. По возможности необходимо предусмотреть замену режима работы блочной памяти ПЛИС «Old memory contents appear» на режим «new_data» или на режим «I do not care». При этом надо помнить, что линия задержки на основе двухпортовой памяти с использованием мегафункции altshift_taps (Shift Register (RAM-based)) работает только в режиме «Old». В этом случае линию задержки КИХ-фильтра можно реализовать на регистрах.

ЗАКЛЮЧЕНИЕ

В учебном пособии на обширном иллюстративном материале показана архитектура академических ПЛИС с адаптивными логическими элементами, встроенными умножителями и блоками памяти.

Для реализации Verilog-проектов в базис академических ПЛИС используется САПР VTR (Verilog to Routing) с открытым исходным кодом. Академический САПР VTR основан на следующих программных инструментах: логический синтез (ODIN-II); оптимизация и технологическое отображение (ABC); кластеризация, размещение и трассировка (VPR).

Также рассматривается применение синтезатора Yosys. Yosys, позволяющий реализовывать Verilog-проекты в базис индустриальных ПЛИС Xilinx серий Spartan-6 и Virtex-7. Для логической оптимизации и отображения в базисе ПЛИС в Yosys интегрирован программный инструмент ABC. Программный инструмент Yosys позволяет также реализовывать Verilog-проекты в двух сериях ПЛИС: Altera Cyclone IV и MAX II посредством vqm-файла. Yosys позволяет отображать Verilog-проекты в базис библиотечных ячеек заказных БИС (в формате Liberty) и используется в маршруте проектирования заказных БИС Qflow с возможностью последующего изготовления на кремниевых фабриках по единым правилам проектирования масштабируемой КМОП-технологии (MOSIS Scalable CMOS design rules), предоставляемые в открытом доступе.

Рассмотрены вопросы программирования ПЛИС серии 5578 с использованием среды разработки конфигурационных данных (АО «КТЦ «ЭЛЕКТРОНИКА») и отладочной платы ОП5578 (АО «Воронежский завод полупроводниковых приборов — сборка»). Процесс проектирования ПЛИС серии 5578 осуществляется в САПР Altera Quartus II версии 9.0 до 13.0 включительно с поддержкой ПЛИС серии Cyclone II и основывается на использовании VQM-файлов (Verilog Quartus Mapping).

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Армстронг, Дж. Р.* Моделирование цифровых систем на языке VHDL [Текст] : пер. с англ. / Р. Дж. Армстронг. — М. : Мир, 1992. — 348 с.

2. *Максфилд, К.* Проектирование на ПЛИС : курс молодого бойца [Текст] : пер. с англ. / К. Максфилд. — М. : Издательский дом Додэка XXI, 2007. — 408 с.

3. *Уэйкерли, Джон Ф.* Проектирование цифровых устройств : пер. с англ. / Ф. Джон Уэйкерли. — М. : Постмаркет, 2002. — 533 с.

4. *Угрюмов, Е.П.* Цифровая схемотехника [Текст] / Е.П. Угрюмов. — СПб. : БХВ, 2004. — 528 с.

5. *Стешенко, В.* ПЛИС фирмы ALTERA : проектирование устройств обработки сигналов [Текст] / В. Стешенко. — М. : Додэка, 2000. — 457 с.

6. *Строгонов, А.В.* САПР VTR7 для проектирования академических ПЛИС [Текст] / А.В. Строгонов, С.А. Цыбин, П.С. Городков // Компоненты и технологии. — 2016. — № 3. — С. 65–73.

7. *Строгонов, А.В.* Программные средства с открытым исходным кодом для проектирования цифровых устройств в базисах БИС и ПЛИС [Текст] / А.В. Строгонов, П.С. Городков // Компоненты и технологии. — 2017. — № 3. — С. 88–97.

8. *Строгонов, А.В.* Реализация Verilog-проектов в базисе академических ПЛИС с применением САПР VTR 7.0 [Текст] / А.В. Строгонов, П.С. Городков // Компоненты и технологии. — 2017. — № 5. — С. 12–17.

9. Реализация Verilog-проектов в базисе индустриальных ПЛИС Xilinx с применением синтезатора Yosys [Текст] / А.В. Строгонов, П.С. Городков // Компоненты и технологии. — 2017. — № 6. — С. 104–107.

10. *Строгонов, А.В.* Реализация Verilog-проектов в базисе ПЛИС Intel FPGA с использованием инструмента синтеза Yosys [Текст] / А.В. Строгонов, П.С. Городков // Электроника : наука, технология, бизнес. — 2017. — № 6 (166). — С. 1–6.

11. *Строгонов, А.В.* САПР VTR8 как инструмент исследования новых архитектур ПЛИС [Текст] / А.В. Строгонов, П.С. Городков // Компоненты и технологии. — 2017. — № 10. — С. 92–96.

12. *Строгонов, А.В.* Какие архитектуры ПЛИС можно разрабатывать с использованием САПР VTR 8.0. [Текст] / А.В. Строгонов, П.С. Городков // Компоненты и технологии. — 2018. — № 2. — С. 86–93.

13. *Строгонов, А.В.* Среда разработки конфигурационных данных для ПЛИС серии 5578 [Текст] / А.В. Строгонов, П.С. Городков // Компоненты и технологии. — 2018. — № 2. — С. 6–9.